aws

Entwicklerhandbuch

AWS IoT Core



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

AWS IoT Core: Entwicklerhandbuch

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

Die Marken und Handelsmarken von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, die geeignet ist, Kunden irrezuführen oder Amazon in irgendeiner Weise herabzusetzen oder zu diskreditieren. Alle anderen Handelsmarken, die nicht Eigentum von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise zu Amazon gehören oder nicht, mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

Was ist AWS IoT? 1
So greifen Ihre Geräte und Apps darauf zu AWS IoT 2
Was AWS IoT kann das 3
IoT 3
IoT in der Hausautomation 3
Wie AWS IoT funktioniert 4
Das IoT-Universum 4
AWS IoT Überblick über die Dienste 7
AWS IoT Core service 12
Erfahren Sie mehr über AWS IoT 16
Schulungsressourcen für AWS IoT 16
AWS IoT Ressourcen und Leitfäden 16
AWS IoT in den sozialen Medien 17
AWS Dienste, die verwendet werden von AWS IoT Core Regel-Engine 18
Kommunikationsprotokolle, unterstützt von AWS IoT Core 19
Was ist neu in der AWS IoT Konsole 20
Legende 23
Arbeiten mit AWS SDKs 24
Erste-Schritte-Tutorials 26
Connect dein erstes Gerät mit AWS IoT Core 26
Einrichten AWS-Konto 28
Melde dich an für ein AWS-Konto 28
Öffnen Sie AWS IoT Konsole 29
Interaktives Tutorial 29
IoT-Geräte verbinden 30
Der Status des Offline-Geräts wird gespeichert 31
Gerätedaten an Dienste weiterleiten 32
Anleitung zur Schnellverbindung 33
Schritt 1. Beginnen Sie mit dem Tutorial 34
Schritt 2. Dies erstellt ein Objekt 35
Schritt 3. Laden Sie Dateien auf Ihr Gerät herunter 39
Schritt 4. Ausführen des Beispiels 42
Schritt 5. Weiter erkunden 46
Testen Sie die Konnektivität 47

Connect-Tutorial für Fortgeschrittene 53
Welche Geräteoption ist für Sie am besten geeignet? 54
Create AWS IoT Ressourcen 55
Konfigurieren Ihres Geräts 60
MQTT-Nachrichten anzeigen mit dem AWS IoT MQTT-Client 101
Anzeigen von MQTT-Nachrichten im MQTT-Client 102
Veröffentlichen von MQTT-Nachrichten vom MQTT-Client 105
Testen von geteilten Abonnements im MQTT-Client 107
AWS IoT Tutorials 109
Erstellen von Demos mit dem AWS IoT Geräte-Client 109
Voraussetzungen für die Erstellung von Demos mit dem AWS IoT Geräte-Client 110
Vorbereitung der Verwendung des IoT Device Client 113
Installation und Konfiguration des IoT-Geräteclients 128
Kommunizieren Sie mit dem Geräteclient über MQTT 142
Führen Sie IoT-Jobs mit dem Device Client aus 163
Bereinigen 178
Erstellen von Lösungen mit dem AWS IoT Geräte-SDKs 189
Beginnen Sie mit der Entwicklung von Lösungen mit dem AWS IoT Geräte-SDKs 189
Ein Gerät AWS IoT Core mithilfe des AWS IoT Geräte-SDK verbinden 189
Erstellen von AWS IoT Regeln zur Weiterleitung von Gerätedaten an andere Dienste 215
Beibehalten des Gerätestatus mit Geräteschatten, während das Gerät offline ist 263
Erstellen eines benutzerdefinierten Autorisierers für AWS IoT Core 295
Überwachung der Bodenfeuchte mit AWS IoT und Raspberry Pi 314
Connect to (Verbinden mit) AWS IoT Core 329
AWS IoT Core - Endpunkte der Steuerungsebene 329
AWS IoT Geräte-Endpunkte 331
AWS IoT Core für LoRa WAN-Gateways und -Geräte 332
Connect to (Verbinden mit) AWS IoT Core Service-Endpunkte 333
AWS CLI for AWS IoT Core 334
AWS SDKs 335
AWS SDKs für Mobilgeräte 341
REST-APIs der AWS IoT Core service 342
Geräte verbinden mit AWS IoT 342
AWS IoT Gerätedaten und Dienstendpunkte 343
AWS IoT Geräte-SDKs 346
Gerätekommunikationsprotokolle 349

MQTT-Themen ... 423
Domain-Konfigurationen ... 451
Connect to (Verbinden mit) AWS IoT FIPS-Endpunkte ... 481
AWS IoT Core - Endpunkte der Steuerungsebene ... 482
AWS IoT Core - Endpunkte auf Datenebene ... 482
AWS IoT Core - Endpunkte des Anbieters von Anmeldeinformationen ... 483
AWS IoT Device Management - Endpunkte für Jobs und Daten ... 483
AWS IoT Device Management - Fleet Hub-Endpunkte ... 484
AWS IoT Device Management - sichere Tunneling-Endpunkte ... 484
AWS IoT Device Management - Endpunkte für verwaltete Integrationen ... 484
Verwalten von Geräten ... 485
Registrierung ... 486
Ein Objekt erstellen ... 486
Objekte auflisten ... 487
Objekte beschreiben ... 489
Ein Objekt aktualisieren ... 490
Ein Objekt löschen ... 491
Ein Prinzipal an ein Objekt anfügen ... 491
Listet Dinge auf, die mit einem Principal verknüpft sind ... 492
Listet Prinzipale auf, die einer Sache zugeordnet sind ... 492
Listet Dinge auf, die mit einem Principal V2 verknüpft sind ... 493
Listet Prinzipale auf, die einer Sache zugeordnet sind (V2) ... 494
Ein Prinzipal von einem Objekt trennen ... 494
Objekttypen ... 495
Objekttyp erstellen ... 496
Objekttypen auflisten ... 496
Einen Objekttyp beschreiben ... 497
Einen Objekttyp mit einem Objekt verknüpfen ... 497
Aktualisiere einen Dingtyp ... 498
Einen Objekttyp als veraltet einstufen ... 498
Einen Objekttyp löschen ... 500
Statische Objektgruppen ... 500
Erstellen einer statischen Objektgruppe ... 502
Beschreiben einer Objektgruppe ... 503
Hinzufügen eines Objekts zu einer statischen Objektgruppe ... 504
Entfernen eines Objekts aus einer statischen Objektgruppe ... 505

Auflisten von Objekten in einer Objektgruppe ... 505
Auflisten von Objektgruppen ... 506
Auflisten von Gruppen für ein Objekt ... 508
Aktualisieren einer statischen Objektgruppe ... 509
Löschen einer Objektgruppe ... 510
Anfügen einer Richtlinie an eine statische Objektgruppe ... 510
Trennen einer Richtlinie von einer statischen Objektgruppe ... 511
Auflisten der an eine statische Objektgruppe angefügten Richtlinien ... 511
Auflisten der Gruppen für eine Richtlinie ... 512
Abrufen gültiger Richtlinien für ein Objekt ... 512
Test-Autorisierung für MQTT-Aktionen ... 513
Dynamische Objektgruppen ... 515
Anwendungsfälle dynamischer Dinggruppen ... 516
Erstellen einer dynamischen Objektgruppe ... 518
Beschreiben einer dynamischen Objektgruppe ... 519
Aktualisieren einer dynamischen Objektgruppe ... 520
Löschen einer dynamischen Objektgruppe ... 521
Einschränkungen bei dynamischen und statischen Dinggruppen ... 521
Einschränkungen bei dynamischen Dinggruppen ... 521
Ordne etwas einer Verbindung zu ... 524
Anwendungsfälle ... 525
Wie verbinde ich eine Sache mit einer Verbindung ... 525
Fügen Sie Propagierungsattribute hinzu ... 529
AWS-Managementkonsole ... 530
AWS CLI ... 530
Markieren von Ressourcen ... 533
Grundlagen zu Tags (Markierungen) ... 533
Tag-Beschränkungen und -Einschränkungen ... 534
Mit IAM-Richtlinien taggen ... 535
Abrechnungsgruppen ... 538
Anzeigen von Kostenzuordnungs- und Nutzungsdaten ... 539
Sicherheit ... 541
Sicherheit in AWS IoT ... 542
Authentifizierung ... 543
X.509 Überblick über das Zertifikat ... 543
Serverauthentifizierung ... 543

Client-Authentifizierung 548
Benutzerspezifische Authentifizierung und Autorisierung 590
Autorisierung 621
AWS Schulung und Zertifizierung 624
AWS IoT Core Richtlinien 624
Autorisieren direkter Anrufe an AWS Dienste unter Verwendung AWS IoT Core Anbieter von Anmeldeinformationen 712
Kontenübergreifender Zugriff mit IAM 719
Datenschutz 721
Datenverschlüsselung in AWS IoT 723
Transportsicherheit in AWS IoT Core 723
Datenverschlüsselung 729
Identity and Access Management 747
Zielgruppe 748
Authentifizierung mit IAM-Identitäten 748
Verwalten des Zugriffs mit Richtlinien 749
Wie AWS IoT funktioniert mit IAM 752
Identity-based Beispiele für politische Maßnahmen 784
AWS verwaltete Richtlinien 789
Fehlerbehebung 804
Protokollieren und Überwachen 806
Überwachungstools 807
Compliance-Validierung 808
Ausfallsicherheit 809
Verwendung AWS IoT Core mit VPC-Endpunkten 810
VPC-Endpoints erstellen für AWS IoT Core Steuerebene 810
VPC-Endpoints erstellen für AWS IoT Core Datenebene 811
VPC-Endpoints erstellen für AWS IoT Core Anbieter von Anmeldeinformationen 812
Erstellen eines Amazon-VPC-Schnittstellenendpunkts 813
Konfigurieren Sie eine private gehostete Zone 815
Steuern des Zugriffs auf AWS IoT Core über VPC-Endpunkte 817
Einschränkungen 819
Skalierung von VPC-Endpunkten mit AWS IoT Core 820
Verwenden von benutzerdefinierten Domains mit VPC-Endpunkten 820
Verfügbarkeit von VPC-Endpunkten für AWS IoT Core 820
Verwendung AWS IoT Device Management mit VPC-Endpunkten 821

Sicherheit der Infrastruktur ... 826
Überwachung der Sicherheit ... 827
Bewährte Methoden für die Gewährleistung der Sicherheit ... 828
Schutz von MQTT-Verbindungen in AWS IoT ... 828
Synchronisieren der internen Uhr eines Geräts ... 831
Überprüfen des Serverzertifikats ... 831
Verwenden einer einzigen Identität pro Gerät ... 832
Verwenden Sie eine Sekunde AWS-Region als Backup ... 833
Just-in-Time-Bereitstellung nutzen ... 833
Berechtigungen zum Ausführen AWS IoT Device Advisor-Tests ... 833
Cross-service Confused Deputy Prevention für Device Advisor ... 835
AWS Schulung und Zertifizierung ... 836
Überwachen AWS IoT ... 837
Konfiguration AWS IoT Protokollierung ... 838
Konfiguration der V2-Anmeldung AWS IoT ... 839
Konfigurieren der Protokollierungsrolle und -richtlinie ... 839
Konfigurieren Sie die Protokollierung im AWS IoT (Konsole) ... 842
Konto konfigurieren und anmelden Event-level AWS IoT (CLI) ... 845
Konfigurieren Sie Resource-specific Überschreibungen in AWS IoT (CLI) ... 847
Protokollstufen ... 850
Überwachen AWS IoT Alarme und Metriken mithilfe von Amazon CloudWatch ... 851
Verwenden von AWS IoT Metriken ... 851
CloudWatch Alarme erstellen ... 852
Metriken und Dimensionen ... 857
Überwachen AWS IoT mithilfe von Protokollen CloudWatch ... 882
Anzeigen AWS IoT loggt sich in der CloudWatch Konsole ein ... 883
CloudWatch Protokolle AWS IoT Einträge protokollieren ... 884
Laden Sie geräteseitige Protokolle auf Amazon hoch CloudWatch ... 944
Funktionsweise ... 944
Laden Sie geräteseitige Protokolle hoch mit AWS IoT Regeln ... 946
AWS IoT API-Aufrufe protokollieren ... 956
AWS IoT Informationen in CloudTrail ... 956
Verstehen AWS IoT Protokolldateieinträge ... 957
Regeln ... 960
Zugriffskontrolle ... 961
Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt ... 961

Widerrufen des Zugriffs auf die Rule Engine ... 964
Beschränkung des HTTP-Aktionsverkehrs auf die IPv6-Bereiche der Rules Engine ... 964
Berechtigungen zum Übergeben einer Rolle ... 966
Erstellen einer Regel ... 968
Erstellen Sie eine Regel (Konsole) ... 969
Regel erstellen (CLI) ... 971
Eine Regel verwalten ... 974
Eine Regel taggen ... 975
Eine Regel anzeigen ... 976
Löschen einer Regel ... 976
AWS IoT Regelaktionen ... 977
Apache Kafka ... 980
CloudWatch Alarme ... 993
CloudWatch Logs ... 995
CloudWatch Metriken ... 998
DynamoDB ... 1000
DynamoDBv2 ... 1003
Elasticsearch ... 1006
HTTP ... 1008
InfluxDB ... 1026
AWS IoT SiteWise ... 1044
Firehose ... 1050
Kinesis Data Streams ... 1052
Lambda ... 1055
Speicherort ... 1058
OpenSearch ... 1062
Erneut veröffentlichen ... 1065
S3 ... 1068
SNS ... 1071
SQS ... 1073
Step Functions ... 1076
Timestream ... 1078
Fehlerbehebung bei einer Regel ... 1086
Greifen Sie auf kontoübergreifende Ressourcen zu ... 1086
Voraussetzungen ... 1087
Cross-account Einrichtung für Amazon SQS ... 1087

Cross-account Einrichtung für Amazon SNS ... 1089
Cross-account Einrichtung für Amazon S3 ... 1091
Cross-account einrichten für AWS Lambda ... 1093
Fehlerbehandlung (Fehleraktion) ... 1096
Nachrichtenformat für Fehleraktion ... 1096
Beispiel für Fehleraktion ... 1098
Basic Ingest ... 1099
Verwenden von Basic Ingest ... 1099
AWS IoT SQL-Referenz ... 1100
SELECT-Klausel ... 1102
FROM-Klausel ... 1104
SET-Klausel ... 1106
WHERE-Klausel ... 1109
Datentypen ... 1110
Operatoren ... 1116
Funktionen ... 1127
Literale ... 1204
Case-Anweisungen ... 1204
JSON-Erweiterungen ... 1206
Ersetzungsvorlagen ... 1208
Verschachtelte Objektabfragen ... 1214
Binäre Nutzlasten ... 1215
SQL-Versionen ... 1222
Reservierte Zeichenfolge in AWS IoT Regelaktionen ... 1224
Shadows ... 1225
Verwendung von Schatten ... 1225
Auswählen der Verwendung benannter oder unbenannter Schatten ... 1226
Zugreifen auf Schatten ... 1227
Verwenden von Schatten in Geräten, Apps und anderen Cloudservices ... 1228
Nachrichtenreihenfolge ... 1228
Kürzen von Shadow-Nachrichten ... 1230
Verwenden von Schatten in Geräten ... 1231
Initialisierung des Geräts bei der ersten Verbindung mit AWS IoT ... 1232
Nachrichten werden verarbeitet, während das Gerät angeschlossen ist AWS IoT ... 1235
Verarbeitung von Nachrichten, wenn das Gerät wieder eine Verbindung herstellt AWS IoT 1236
Verwenden von Schatten in Apps und Services ... 1236

Initialisierung der App oder des Dienstes beim Herstellen einer Verbindung zu AWS IoT ... 1238
Der Verarbeitungsstatus ändert sich, während die App oder der Dienst verbunden ist AWS IoT ... 1238
Erkennt, ob ein Gerät angeschlossen ist ... 1238
Simulieren der Device Shadow-Servicekommunikation ... 1241
Einrichten der Simulation ... 1241
Initialisieren des Geräts ... 1241
Senden einer Aktualisierung von der App ... 1246
Reaktion auf eine Aktualisierung im Gerät ... 1248
Beobachten Sie das Update in der App ... 1253
Über die Simulation hinaus ... 1255
Interaktion mit Schatten ... 1255
Protokollunterstützung ... 1256
Anforderungs- und Meldestatus ... 1256
Aktualisieren eines Shadows ... 1256
Abrufen eines Shadow-Dokuments ... 1261
Löschen von Schattendaten ... 1262
Geräteschatten-REST-API ... 1265
GetThingShadow ... 1267
UpdateThingShadow ... 1268
DeleteThingShadow ... 1269
ListNamedShadowsForThing ... 1270
MQTT-Themen für Geräteschatten ... 1272
/get ... 1273
/get/accepted ... 1274
/get/rejected ... 1275
/update ... 1276
/update/delta ... 1277
/update/accepted ... 1278
/update/documents ... 1279
/update/rejected ... 1280
/delete ... 1281
/delete/accepted ... 1282
/delete/rejected ... 1283
Dokumente des Device Shadow-Services ... 1284
Beispiele für Schatten-Dokumente ... 1284

Dokumenteigenschaften ... 1291
Delta-Status ... 1292
Versioning von Schattendokumenten ... 1294
Client-Tokens in Schattendokumenten ... 1295
Eigenschaften des leeren Schattendokuments ... 1295
Array-Werte in Schattendokumenten ... 1296
Device Shadow-Fehlermeldungen ... 1297
Softwarepaket-Katalog ... 1299
Vorbereitung der Verwendung des Softwarepaket-Katalogs ... 1300
Lebenszyklus der Paketversion ... 1300
Namenskonventionen für Paketversionen ... 1302
Standardversion ... 1302
Versionsattribute ... 1302
Software-Stückliste ... 1303
Aktivieren AWS IoT -Flottenindizierung ... 1307
Reservierter benannter Schatten ... 1307
Löschen eines Softwarepakets ... 1309
Vorbereitung der Sicherheit ... 1310
Resource-based authentication ... 1310
AWS IoT Arbeitsrechte zum Bereitstellen von Paketversionen ... 1312
AWS IoT Jobrechte zur Aktualisierung des reservierten benannten Shadows ... 1313
AWS IoT Jobs: Berechtigungen zum Herunterladen von Amazon S3 ... 1315
Berechtigungen zur Aktualisierung der Softwareliste für eine Paketversion ... 1315
Vorbereitung der Flottenindizierung ... 1318
Den `$package` Shadow als Datenquelle festlegen ... 1319
In der Konsole dargestellte Metriken ... 1320
Abfragemuster ... 1321
Erfassung der Paketversionsverteilung über Get `BucketsAggregation` ... 1323
Jobs vorbereiten AWS IoT ... 1324
Substitutionsparameter für AWS IoT jobs ... 1324
Vorbereitung des Auftragsdokuments und der Paketversion für die Bereitstellung ... 1328
Benennen der Pakete und Versionen bei der Bereitstellung ... 1332
Zielgruppenzuweisung für Jobs durch AWS IoT Dynamische Objektgruppen ... 1333
Reservierte benannte Schatten- und Paketversionen ... 1333
Deinstallation eines Softwarepakets ... 1334
Erste Schritte ... 1335

Ein Paket und eine Version erstellen ... 1336
Bereitstellen einer Paketversion ... 1338
Zuordnen einer Paketversion ... 1341
Jobs ... 1343
Zugriff AWS IoT jobs ... 1343
AWS IoT Jobs, Regionen und Endpunkte ... 1343
Was ist eine Fernsteuerung? ... 1344
Vorteile der Verwendung AWS IoT Device Management Jobs für Fernoperationen ... 1344
Was ist AWS IoT Jobs? ... 1347
Wichtige Konzepte von Jobs ... 1348
Aufträge und Status der Auftragsausführung ... 1352
Verwalten von Aufträgen ... 1358
Codesignatur für Aufgaben ... 1358
Auftragsdokument ... 1358
Vorsignierte URLs ... 1358
Vorsignierte URL für den Datei-Upload ... 1361
Vorsignierte URL unter Verwendung der Amazon S3-Versionierung ... 1362
Erstellen und Verwalten von Aufträgen mithilfe der Konsole ... 1364
Erstellen und Verwalten von Aufträgen mithilfe des CLI. ... 1367
Auftragsvorlagen ... 1379
Benutzerdefiniert und AWS verwaltete Vorlagen ... 1380
Verwenden Sie AWS verwaltete Vorlagen ... 1380
Erstellen von benutzerdefinierten Auftragsvorlagen ... 1401
Aufgabe configurations ... 1410
Wie funktionieren Auftragskonfigurationen ... 1411
Zusätzliche Konfigurationen angeben ... 1427
Geräte und Aufträge ... 1437
Programmieren von Geräten zur Arbeit mit Aufträgen ... 1440
Geräteworkflow ... 1441
Arbeitsablauf für Aufträge ... 1443
Auftragsbenachrichtigungen ... 1448
AWS IoT Jobs API-Operationen ... 1456
Auftragsverwaltung und -steuerung API und Datentypen ... 1459
Aufträge, Geräte, MQTT- und HTTPS-API-Operationen und Datentypen ... 1479
Schutz der Benutzer und Geräte für Aufträge ... 1494
Erforderlicher Richtlinientyp für AWS IoT Jobs ... 1495

Autorisieren von Benutzern für Aufträge und Cloud-Services ... 1496
Autorisieren von Geräten, Aufträge zu verwenden ... 1510
AWS IoT Grenzwerte für Aufträge ... 1514
Grenzwerte für die Ausführung von Aufträgen ... 1515
Limits für aktive und gleichzeitige Aufträge ... 1516
Befehle ... 1520
Schnellstart ... 1521
Befehle, Konzepte und Status ... 1522
Befehle und wichtige Konzepte ... 1522
Befehlsstatus ... 1527
Status der Befehlsausführung ... 1527
Arbeitsablauf für Befehle ... 1531
Erstellen und Verwalten von Befehlen ... 1532
Wählen Sie Ziele und abonnieren Sie Themen ... 1532
Starten und Überwachen von Befehlsausführungen ... 1534
(Optional) Aktivieren Sie Benachrichtigungen für Befehlsereignisse ... 1535
Erstellen und Verwalten von Befehlen ... 1537
Erstellen Sie eine Befehlsressource ... 1537
Ruft Informationen zu einem Befehl ab ... 1543
Listen Sie Befehle in Ihrem auf AWS-Konto ... 1545
Aktualisieren Sie eine Befehlsressource ... 1547
Verwerfen Sie eine Befehlsressource oder stellen Sie sie wieder her ... 1549
Löschen Sie eine Befehlsressource ... 1549
Starten und Überwachen von Befehlsausführungen ... 1551
Starten Sie eine Befehlsausführung ... 1552
Aktualisieren des Ergebnisses einer Befehlsausführung ... 1559
Ruft eine Befehlsausführung ab ... 1565
Befehlsaktualisierungen mit dem MQTT-Testclient anzeigen ... 1569
Listen Sie die Befehlsausführungen in Ihrem auf AWS-Konto ... 1571
Löschen Sie eine Befehlsausführung ... 1574
Kennzeichnen einer Befehlsressource als veraltet ... 1575
Wesentliche Überlegungen ... 1576
Verwerfen Sie eine Befehlsressource (Konsole) ... 1576
Verwerfen Sie eine Befehlsressource (CLI) ... 1576
Überprüfen Sie die Uhrzeit und den Status der veralteten Version ... 1577
Stellen Sie eine Befehlsressource wieder her ... 1577

Sicheres Tunneln 1579
Was ist Secure Tunneling? 1579
Secure Tunneling-Konzepte 1580
Wie funktioniert Secure Tunneling 1581
Sicherer Tunnellebenszyklus 1582
Secure Tunneling-Tutorials 1583
Tutorials in diesem Abschnitt 1584
Öffnen Sie einen Tunnel und starten Sie eine SSH-Sitzung zum Remote-Gerät 1585
Öffnen Sie einen Tunnel und verwenden Sie browserbasiertes SSH, um auf das Remote-Gerät zuzugreifen 1604
Lokaler Proxy 1608
Wie man den lokalen Proxy benutzt 1609
Konfigurieren Sie den lokalen Proxy für Geräte, die einen Web-Proxy verwenden 1616
Multiplexing und gleichzeitige TCP-Verbindungen 1624
Multiplexing mehrerer Datenströme 1625
Gleichzeitige TCP-Verbindungen verwenden 1629
Ein Remote-Gerät konfigurieren und IoT-Agent verwenden 1632
IoT-Agent-Snippet 1632
Steuern des Zugriffs auf Tunnel 1634
Voraussetzungen für den Tunnelzugriff 1634
Richtlinien für den Tunnelzugriff 1635
Lösung von Verbindungsproblemen beim Secure Tunneling 1642
Fehler beim ungültigen Client-Zugriffstoken 1643
Fehler bei Nichtübereinstimmung der Client-Tokens 1643
Verbindungsprobleme bei Remote-Geräten 1645
Gerätebereitstellung 1648
Bereitstellung von Geräten in AWS IoT 1649
Flottenbereitstellungs-APIs 1651
Bereitstellen von Geräten ohne Gerätezertifikate mithilfe der Flottenbereitstellung 1651
Bereitstellung durch Anspruch 1652
Bereitstellung durch vertrauenswürdigen Benutzer 1655
Verwenden von Pre-Provisioning-Hooks mit dem AWS CLI 1658
Bereitstellen von Geräten mit Gerätezertifikaten 1661
Bereitstellung eines einzelnen Objekts 1662
Just-in-time Bereitstellung 1662
Massenregistrierung 1669

Bereitstellungsvorlagen 1670
Bereich "Parameters" 1670
Bereich „Ressourcen“ 1671
Vorlagenbeispiel für eine Massenregistrierung 1677
Vorlagenbeispiel für eine Just-in-Time-Bereitstellung (JITP) 1678
Flottenbereitstellung 1680
Pre-provisioning Haken 1684
Pre-provision Hook-Eingabe 1685
Pre-provision Hook-Rückgabewert 1685
Pre-provisioning Hook-Lambda-Beispiel 1686
Self-managed Signieren von Zertifikaten mit AWS IoT Core Zertifikatsanbieter 1689
So funktioniert die selbstverwaltete Zertifikatsignierung bei der Flottenbereitstellung 1690
Eingabe der Lambda-Funktion des Zertifikatsanbieters 1692
Rückgabewert der Lambda-Funktion des Zertifikatsanbieters 1692
Beispiel-Lambda-Funktion 1692
Self-managed Unterzeichnung eines Zertifikats für die Bereitstellung der Flotte 1694
AWS CLI Befehle für den Zertifikatsanbieter 1696
Erstellen von IAM-Richtlinien und -Rollen für Benutzer, die ein Gerät installieren 1699
Erstellen einer IAM-Richtlinie für Benutzer, die ein Gerät installieren werden 1699
Erstellen einer IAM-Rolle für Benutzer, die ein Gerät installieren werden 1700
Aktualisieren einer vorhandenen Richtlinie, um eine neue Vorlage zu autorisieren 1701
MQTT-API für die Gerätebereitstellung 1702
CreateCertificateFromCsr 1703
CreateKeysAndCertificate 1706
RegisterThing 1708
-Flottenindizierung 1711
Verwalten von Indexaktualisierungen 1711
Der Konnektivitätsstatus für ein bestimmtes Gerät wird abgefragt 1711
Datenquellenübergreifende Suche 1711
Abfragen von Aggregatdaten 1712
Überwachung aggregierter Daten und Erstellung von Alarmen mithilfe von Flottenkennzahlen 1712
Verwalten der Flottenindizierung 1712
Objektindizierung 1712
Modus für die Objektgruppenindizierung. 1714
Verwaltete Felder 1714
Benutzerdefinierte Felder 1716

Verwalten der Objektindizierung ... 1717
Verwalten der Objektgruppenindizierung ... 1736
Status der Gerätekonnektivität ... 1738
Funktionsweise ... 1739
Features ... 1739
Vorteile ... 1739
Voraussetzungen ... 1740
Beispiel ... 1741
Abfragen von Aggregatdaten ... 1743
GetStatistics ... 1744
GetCardinality ... 1747
GetPercentiles ... 1748
GetBucketsAggregation ... 1750
Autorisierung ... 1751
Abfragesyntax ... 1752
Unterstützte Features ... 1752
Nicht unterstützte Funktionen ... 1752
Hinweise ... 1753
Beispiel für Objektabfragen ... 1753
Beispiel für Objektgruppenabfragen ... 1758
Indexierung von Standortdaten ... 1760
Unterstützte Datumsformate ... 1760
Wie indexiert man Standortdaten ... 1762
Konfiguration der Objektindizierung. ... 1762
Beispiele für Geoqueries ... 1765
Erste Schritte-Tutorial ... 1766
Flottenmetriken ... 1771
Erste Schritte-Tutorial ... 1772
Verwalten von Flottenmetriken ... 1779
MQTT-based Dateizustellung ... 1787
Was ist ein Stream? ... 1787
Einen Stream verwalten ... 1788
Erteilen von Berechtigungen für Ihre Geräte ... 1789
Verbinden Sie Ihre Geräte mit AWS IoT ... 1790
TagResource Nutzung ... 1791
Verwenden Sie AWS IoT MQTT-based die Dateizustellung auf Geräten ... 1792

Wird verwendet DescribeStream , um Stream-Daten abzurufen ... 1792
Abrufen von Datenblöcken aus einer Stream-Datei ... 1794
Behandlung von Fehlern von AWS IoT MQTT-based Lieferung von Dateien ... 1801
Ein Beispiel für einen Anwendungsfall in FreeRTOS OTA ... 1803
Device Advisor ... 1804
Einrichtung ... 1806
Erstellen eines IoT-Dings ... 1806
Erstellen einer IAM-Rolle, die Sie als Ihre Geräterolle verwenden möchten ... 1806
Erstellen einer individuell verwalteten Richtlinie für einen IAM-Benutzer zur Verwendung von Device Advisor ... 1810
Erstellen eines IAM-Benutzers für die Verwendung von Device Advisor ... 1810
Konfigurieren Ihres Geräts ... 1814
Erste Schritte mit Device Advisor in der Konsole ... 1815
Device-Advisor-Workflow ... 1825
Voraussetzungen ... 1825
Erstellen einer Testsuite-Definition ... 1825
Abrufen einer Testsuite-Definition ... 1828
Abrufen eines Testendpunkts ... 1828
Ausführen einer Testsuite ... 1829
Abrufen einer Testsuite-Ausführung ... 1829
Beenden einer Testsuite-Ausführung ... 1830
Abrufen eines Qualifizierungsberichts für eine erfolgreiche Ausführung der Qualifizierungstestsuite ... 1830
Ausführlicher Konsolen-Workflow von Device Advisor ... 1831
Voraussetzungen ... 1831
Erstellen einer Testsuite-Definition ... 1831
Ausführen einer Testsuite ... 1839
Beenden einer Testsuite-Ausführung (optional) ... 1841
Anzeigen von Details und Protokolle der Testsuite-Ausführung ... 1843
Laden Sie eine herunter AWS IoT Qualifizierungsbericht ... 1845
Konsolen-Workflow für Tests mit langer Dauer ... 1845
Device Advisor VPC-Endpunkte (AWS PrivateLink) ... 1854
Überlegungen für AWS IoT Core Device Advisor VPC-Endpunkte ... 1855
Erstellen Sie einen Schnittstellen-VPC-Endpunkt für AWS IoT Core Device Advisor ... 1856
Steuerung des Zugriffs auf AWS IoT Core Device Advisor über VPC-Endpunkte ... 1856
Device-Advisor-Testfälle ... 1858

Device Advisor-Testfälle, um sich für das zu qualifizieren AWS Programm zur Gerätequalifizierung. ... 1858
TLS ... 1859
MQTT ... 1866
Shadow ... 1881
Auftragsausführung ... 1883
Berechtigungen und Richtlinien ... 1885
Tests mit langer Dauer ... 1886
Standort des Geräts ... 1904
Messungstypen und Solver ... 1905
Wie AWS IoT Core Der Gerätestandort funktioniert ... 1906
Wie benutzt man AWS IoT Core Standort des Geräts ... 1908
Auflösen des Standorts von IoT-Geräten ... 1909
Auflösen des Gerätestandorts (Konsole) ... 1909
Auflösen des Gerätestandorts (API) ... 1913
Fehlerbehebung beim Auflösen des Standorts ... 1915
Auflösen des Gerätestandorts mithilfe von MQTT-Themen ... 1916
MQTT-Themen zum Format des Gerätestandorts ... 1917
Richtlinie für MQTT-Themen zum Gerätestandort ... 1918
Themen zum Gerätestandort und Nutzlast ... 1919
Location Solver und Geräte-Payload ... 1924
Wi-Fi basierter Solver ... 1925
Solver auf Mobilfunkbasis ... 1926
IP-Reverse-Lookup-Solver ... 1931
GNSS-Solver ... 1932
Multi-frame GNSS-Solver ... 1933
BLE-Solver für Amazon Sidewalk-fähige Geräte ... 1935
Erweiterte Konfiguration für AWS IoT Core Standort des Geräts ... 1936
WiFiCellular Konfiguration ... 1936
Wi-Fi Payload-Beispiel mit AdvancedConfiguration ... 1937
Beispiel für eine mobile Nutzlast mit AdvancedConfiguration ... 1938
Ereignismeldungen ... 1941
Generieren von Ereignisnachrichten ... 1941
Richtlinie für den Empfang von Ereignisnachrichten ... 1941
Aktivieren Sie Ereignisse für AWS IoT ... 1942
Registry-Ereignisse ... 1947

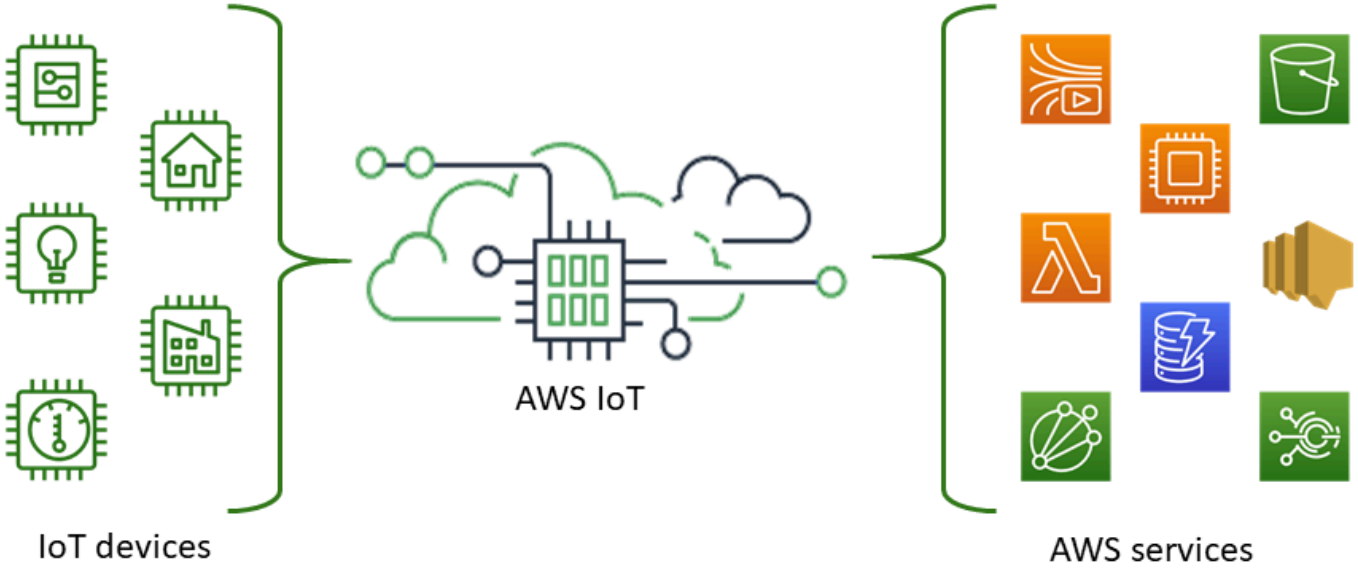
Objektereignisse ... 1947
Objekttypereignisse ... 1949
Objektgruppenereignisse ... 1953
Auftragsereignisse ... 1958
Ereignisse bei der Befehlsausführung ... 1963
Ereignisse im Lebenszyklus ... 1965
Connect/Disconnect Ereignisse ... 1965
Ereignis, dass der Verbindungsversuch fehlgeschlagen ist ... 1970
Subscribe/Unsubscribe Ereignisse ... 1972
Fehlerbehebung ... 1974
AWS IoT Core Anleitung zur Fehlerbehebung ... 1974
Diagnostizieren von Verbindungsproblemen ... 1975
Fehler bei der Regeldiagnose ... 1979
Diagnostizieren von Problemen mit Schatten ... 1981
Diagnostizieren von Stream-Limits ... 1983
Behebung von Verbindungsabbrüchen bei Geräteflotten ... 1984
AWS IoT Device Management Anleitung zur Fehlerbehebung ... 1985
AWS IoT Probleme bei Aufträgen ... 1985
AWS IoT Befehle zur Problembehandlung ... 1990
Fehlerbehebung bei der Flottenindizierung ... 1993
AWS IoT Fehlerbehebung im Softwarepaketkatalog für die Geräteverwaltung ... 1996
AWS IoT Anleitung zur Fehlerbehebung bei Device Advisor ... 2004
AWS IoT Fehler ... 2007
AWS IoT Geräte-SDKs, SDKs für Mobilgeräte und AWS IoT Geräte-Client ... 2009
AWS IoT Geräte-SDKs ... 2009
AWS IoT Geräte-SDK für Embedded C ... 2012
AWS SDKs für Mobilgeräte ... 2012
AWS IoT Geräte-Client ... 2013
Früher AWS IoT Versionen der Geräte-SDKs ... 2014
Codebeispiele ... 2015
Grundlagen ... 2016
Hallo AWS IoT ... 2017
Kennenlernen der Grundlagen ... 2025
Aktionen ... 2138
Szenarien ... 2246
Erste Schritte mit dem Geräteschutz im Internet der Dinge ... 2247

Erste Schritte mit Nachrichten im Internet der Dinge 2267
AWS IoT Kontingente 2278
AWS IoT Core Preise 2279
Häufig gestellte Fragen zu unterstützten Protokollen 2280
Was sind die AWS JSON- und Smithy RPC v2 CBOR-Protokolle, und wie unterscheiden sie sich? 2280
Wie fange ich mit dem Smithy RPC v2 CBOR-Protokoll an? 2280
Was sind die Risiken der Aktivierung des Smithy RPC v2 CBOR-Protokolls? 2281
Was ist, wenn meine Lösung Smithy RPC v2 CBOR nicht unterstützt? 2281
Was AWS Unterstützen Regionen das Smithy RPC v2 CBOR-Protokoll? 2281
Welche Latenzverbesserungen kann ich beim Upgrade auf SDK-Versionen erwarten, die Smithy RPC v2 CBOR unterstützen? 2282
Wird der AWS Ist das JSON-Protokoll veraltet? 2282
Wo finde ich weitere Informationen zu Smithy RPC v2 CBOR? 2282
..... mmcclxxxiii

Was ist AWS IoT?

AWS IoT stellt die Cloud-Dienste bereit, die Ihre IoT-Geräte mit anderen Geräten und AWS Cloud-Diensten verbinden. AWS IoT bietet Gerätesoftware, mit der Sie Ihre IoT-Geräte in AWS IoT basierte Lösungen integrieren können. Wenn Ihre Geräte eine Verbindung herstellen können AWS IoT, AWS IoT können Sie sie mit den bereitgestellten Cloud-Diensten verbinden. AWS

Eine praktische Einführung zu finden Sie AWS IoT unterErste-Schritte-Tutorials.



Mit AWS IoT können Sie die am besten geeigneten und aktuellsten Technologien für Ihre Lösung auswählen. Unterstützt die folgenden Protokolle, um Sie bei der Verwaltung und Unterstützung Ihrer IoT-Geräte vor Ort zu AWS IoT Core unterstützen:

- MQTT (Message Queuing und Telemetrietransport)
- MQTT über WSS (Websockets Secure)
- HTTPS (Hypertext Transfer Protocol – Secure)
- LoRaWAN (Weitbereichsnetzwerk mit großer Reichweite)

Der AWS IoT Core Message Broker unterstützt Geräte und Clients, die die Protokolle MQTT und MQTT über WSS verwenden, um Nachrichten zu veröffentlichen und zu abonnieren. Er unterstützt auch Geräte und Clients, die das HTTPS-Protokoll zum Veröffentlichen von Nachrichten verwenden.

AWS IoT Core for LoRa WAN hilft Ihnen dabei, WLAN-Geräte (Low-Power LoRa Long-range Wide Area Network) zu verbinden und zu verwalten. AWS IoT Core für LoRa WAN entfällt die Notwendigkeit, einen LoRa WAN-Netzwerkserver (LNS) zu entwickeln und zu betreiben.

Wenn Sie AWS IoT Funktionen wie Gerätekommunikation, Regeln oder Aufgaben nicht benötigen, finden Sie unter AWS Messaging Informationen zu anderen AWS IoT Messaging-Diensten, die Ihren Anforderungen möglicherweise besser entsprechen.

So greifen Ihre Geräte und Apps darauf zu AWS IoT

AWS IoT bietet die folgenden Schnittstellen fürAWS IoT Tutorials:

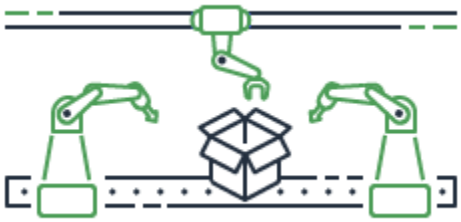
- AWS IoT Geräte-SDKs — Erstellen Sie Anwendungen auf Ihren Geräten, die Nachrichten an diese senden und von denen Nachrichten empfangen werden. AWS IoT Weitere Informationen finden Sie unter AWS IoT Geräte-SDKs, SDKs für Mobilgeräte und AWS IoT Geräte-Client.
- AWS IoT Core für LoRa WAN — Connect und verwalten Sie Ihre WAN-Geräte und -Gateways mit großer Reichweite, indem Sie AWS IoT Core for LoRa LoRa WAN verwenden.
- AWS Command Line Interface (AWS CLI) — Befehle für AWS IoT Windows, macOS und Linux ausführen. Diese Befehle ermöglichen Ihnen die Erstellung und Verwaltung von Dingobjekten, Zertifikaten, Regeln, Jobs und Richtlinien. Informationen zu den ersten Schritten finden Sie im AWS Command Line Interface -Benutzerhandbuch. Weitere Informationen zu den Befehlen für AWS IoT finden Sie unter iot in der AWS CLI Befehlsreferenz.
- AWS IoT API — Erstellen Sie Ihre IoT-Anwendungen mithilfe von HTTP- oder HTTPS-Anfragen. Diese API-Aktionen ermöglichen Ihnen die programmgesteuerte Erstellung und Verwaltung von Objekten, Zertifikaten, Regeln und Richtlinien. Weitere Informationen zu den API-Aktionen für AWS IoT finden Sie unter Aktionen in der AWS IoT API-Referenz.
- AWS SDKs — Erstellen Sie Ihre IoT-Anwendungen mithilfe sprachspezifischer APIs. Diese SDKs umschließen die HTTP/HTTPS API und ermöglichen es Ihnen, in jeder der unterstützten Sprachen zu programmieren. Weitere Informationen finden Sie unter AWS SDKs und Tools.

Sie können auch AWS IoT über die AWS IoT Konsole darauf zugreifen, die eine grafische Benutzeroberfläche (GUI) bietet, über die Sie die Ding-Objekte, Zertifikate, Regeln, Jobs, Richtlinien und andere Elemente Ihrer IoT-Lösungen konfigurieren und verwalten können.

Was AWS IoT kann das

In diesem Thema werden einige der von AWS IoT unterstützten Lösungen beschrieben, die Sie möglicherweise benötigen.

IoT



Dies sind einige Beispiele für AWS IoT Lösungen für industrielle Anwendungsfälle, bei denen IoT-Technologien eingesetzt werden, um die Leistung und Produktivität industrieller Prozesse zu verbessern.

Lösungen für industrielle Anwendungsfälle

- Verwenden Sie AWS IoT um prädiktive Qualitätsmodelle in Industriebetrieben zu erstellen

  Erfahren Sie, wie AWS IoT Sie Daten aus Industriebetrieben sammeln und analysieren können, um prädiktive Qualitätsmodelle zu erstellen. Weitere Informationen
- Verwenden Sie AWS IoT zur Unterstützung der vorausschauenden Wartung in Industriebetrieben

  Erfahren Sie, wie Sie bei der Planung präventiver Wartungsarbeiten helfen AWS IoT können, um ungeplante Ausfallzeiten zu reduzieren. Weitere Informationen

IoT in der Hausautomation



Dies sind einige Beispiele für AWS IoT Lösungen für Anwendungsfälle in der Hausautomation, bei denen IoT-Technologien eingesetzt werden, um skalierbare IoT-Anwendungen zu entwickeln, die Haushaltsaktivitäten mithilfe von vernetzten Heimgeräten automatisieren.

Lösungen für die Hausautomation

- Verwenden Sie AWS IoT in Ihrem vernetzten Zuhause

  Erfahren Sie, wie AWS IoT Sie integrierte Hausautomationslösungen anbieten können.
- Verwenden Sie AWS IoT um Sicherheit und Überwachung zu Hause zu gewährleisten

  Erfahren Sie, wie AWS IoT Sie maschinelles Lernen und Edge-Computing auf Ihre Hausautomationslösung anwenden können.

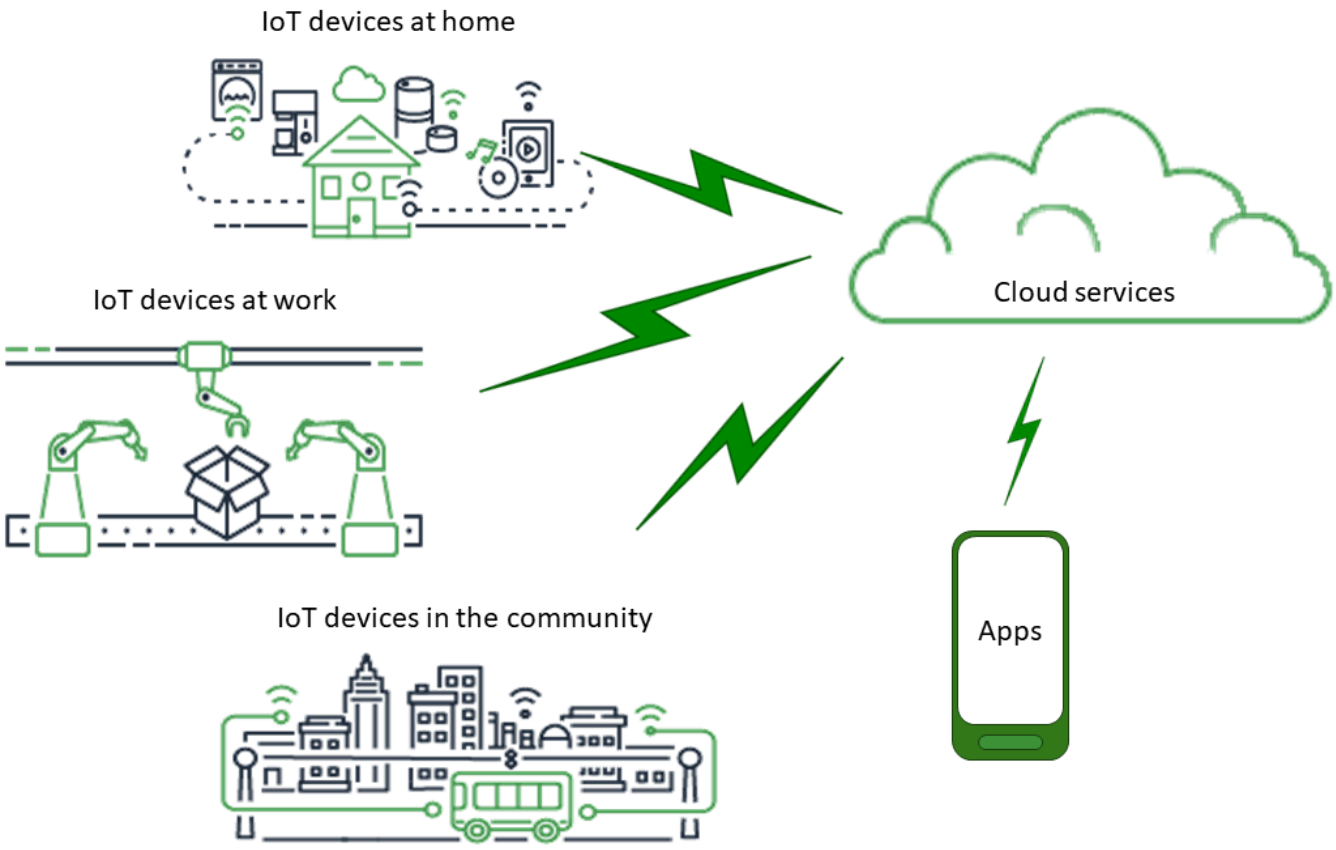
Eine Liste von Lösungen für industrielle, private und kommerzielle Anwendungsfälle finden Sie im AWS IoT Solution Repository.

Wie AWS IoT funktioniert

AWS IoT bietet Cloud-Dienste und Geräteunterstützung, mit denen Sie IoT-Lösungen implementieren können. AWS bietet viele Cloud-Dienste zur Unterstützung von IoT-based Anwendungen. Damit Sie verstehen, wo Sie anfangen sollen, bietet dieser Abschnitt ein Diagramm und eine Definition der wichtigsten Konzepte, um Sie in das IoT-Universum einzuführen.

Das IoT-Universum

Im Allgemeinen besteht das Internet der Dinge (IoT) aus den in diesem Diagramm gezeigten Schlüsselkomponenten.



Apps

Apps bieten Endbenutzern Zugriff auf IoT-Geräte und die Funktionen, die von den Cloud-Diensten bereitgestellt werden, mit denen diese Geräte verbunden sind.

Cloud-Dienste

Cloud-Dienste sind verteilte, groß angelegte Datenspeicher- und Verarbeitungsdienste, die mit dem Internet verbunden sind. Zu den Beispielen gehören:

- IoT-Verbindungs- und Verwaltungsdienste

  AWS IoT ist ein Beispiel für einen IoT-Verbindungs- und Verwaltungsdienst.
- Rechendienste wie Amazon Elastic Compute Cloud und AWS Lambda
- Datenbankdienste wie Amazon DynamoDB.

Kommunikation

Geräte kommunizieren mithilfe verschiedener Technologien und Protokolle mit Cloud-Diensten. Zu den Beispielen gehören:

- Wi-Fi/Broadband Internet
- Breitband-Mobilfunkdaten
- Narrow-band Mobilfunkdaten
- Long-range Weitverkehrsnetz (LoRaWAN)
- Proprietäre HF-Kommunikation

Geräte

Ein Gerät ist eine Art von Hardware, die Schnittstellen und Kommunikation verwaltet. Geräte befinden sich normalerweise in unmittelbarer Nähe der realen Schnittstellen, die sie überwachen und steuern. Geräte können Rechen- und Speicherressourcen wie Mikrocontroller, CPUs und Speicher enthalten. Zu den Beispielen gehören:

- Raspberry Pi
- Arduino
- Voice-interface Assistenten
- LoRaWAN und Geräte
- Amazon-Sidewalk-Geräte
- Maßgeschneiderte IoT-Geräte

Schnittstellen

Eine Schnittstelle ist eine Komponente, die ein Gerät mit der physischen Welt verbindet.

- Benutzeroberflächen

  Komponenten, mit denen Geräte und Benutzer miteinander kommunizieren können.

  - Eingangsschnittstellen

    Ermöglichen Sie einem Benutzer die Kommunikation mit einem Gerät

Beispiele: Tastatur, Knopf

- Eingangsschnittstellen

  Ermöglichen Sie einem Benutzer die Kommunikation mit einem Gerät

  Beispiele: Alpha-numeric Display, grafisches Display, Kontrollleuchte, Alarmglocke

- Sensoren

  Eingabekomponenten, die etwas in der Außenwelt so messen oder wahrnehmen, dass ein Gerät es versteht. Zu den Beispielen gehören:

  - Temperatursensor (wandelt die Temperatur in ein analoges oder digitales Signal um)
  - Feuchtigkeitssensor (wandelt die relative Luftfeuchtigkeit in ein analoges oder digitales Signal um)
  - Analog-Digital-Wandler (wandelt eine analoge Spannung in einen numerischen Wert um)
  - Ultraschall-Entfernungsmessgerät (wandelt eine Entfernung in einen numerischen Wert um)
  - Optischer Sensor (wandelt eine Lichtstärke in einen numerischen Wert um)
  - Kamera (wandelt Bilddaten in digitale Daten um)

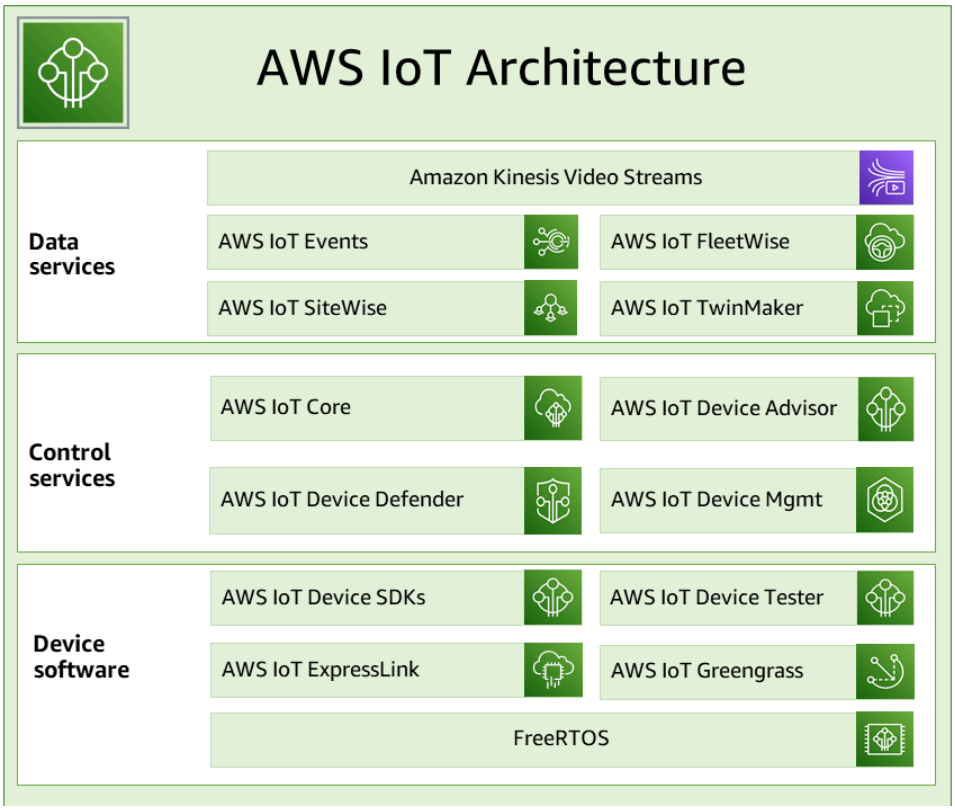
- Aktuator

  Ausgabekomponenten, mit denen das Gerät etwas in der Außenwelt steuern kann. Zu den Beispielen gehören:

  - Schrittmotoren (wandeln elektrische Signale in Bewegung um)
  - Relais (steuern hohe elektrische Spannungen und Ströme)

AWS IoT Überblick über die Dienste

AWS IoT Stellt im IoT-Universum die Dienste bereit, die die Geräte unterstützen, die mit der Welt interagieren, und die Daten, die zwischen ihnen ausgetauscht werden, und AWS IoT. AWS IoT besteht aus den in dieser Abbildung gezeigten Diensten zur Unterstützung Ihrer IoT-Lösung.



AWS IoT Gerätesoftware

AWS IoT stellt diese Software zur Unterstützung Ihrer IoT-Geräte bereit.

AWS IoT Geräte-SDKs

Die AWS IoT Geräte- und Mobil-SDKs helfen Ihnen dabei, Ihre Geräte effizient mit zu verbinden. AWS IoT Die AWS IoT Geräte- und Mobile-SDKs enthalten Open-Source-Bibliotheken, Entwicklerhandbücher mit Beispielen und Portierungsleitfäden, sodass Sie innovative IoT-Produkte oder -Lösungen auf den Hardwareplattformen Ihrer Wahl entwickeln können.

AWS IoT Device Tester

AWS IoT Device Testerfür FreeRTOS und AWS IoT Greengrass ist ein Testautomatisierungstool für Mikrocontroller. AWS IoT Device Tester testet Ihr Gerät, um festzustellen, ob es FreeRTOS oder oder Dienste ausführen kann AWS IoT Greengrass und ob es mit Diensten zusammenarbeitet. AWS IoT

AWS IoT ExpressLink

AWS IoT ExpressLink unterstützt eine Reihe von Hardwaremodulen, die von Partnern entwickelt und angeboten werden.AWS Zu den Konnektivitätsmodulen gehört AWS validierte Software,

mit der Sie Geräte schneller und einfacher mit der Cloud verbinden und problemlos in eine Reihe von AWS Diensten integrieren können. Weitere Informationen finden Sie auf der AWS IoT ExpressLinkÜbersichtsseite oder im AWS IoT ExpressLink Programmierhandbuch.

AWS IoT Greengrass

AWS IoT Greengrasserstreckt sich AWS IoT auf Edge-Geräte, sodass diese lokal auf die von ihnen generierten Daten reagieren, Vorhersagen auf der Grundlage von Modellen für maschinelles Lernen treffen und Gerätedaten filtern und aggregieren können. AWS IoT Greengrass ermöglicht es Ihren Geräten, Daten näher am Ort ihrer Entstehung zu sammeln und zu analysieren, selbstständig auf lokale Ereignisse zu reagieren und sicher mit anderen Geräten im lokalen Netzwerk zu kommunizieren. Sie können Edge-Anwendungen mithilfe von vorgefertigten Softwaremodulen, sogenannten Komponenten, erstellen, die Ihre Edge-Geräte mit AWS Diensten oder Diensten von Drittanbietern verbinden können. AWS IoT Greengrass

FreeRTOS

FreeRTOS ist ein Open-Source-Echtzeitbetriebssystem für Mikrocontroller, mit dem Sie kleine Edge-Geräte mit geringem Stromverbrauch in Ihre IoT-Lösung integrieren können. FreeRTOS umfasst einen Kernel und eine wachsende Anzahl von Softwarebibliotheken, die viele Anwendungen unterstützen. FreeRTOS-Systeme können kleine Geräte mit geringem Stromverbrauch sicher mit AWS IoT verbinden und leistungsstärkere Edge-Geräte unterstützen, die AWS IoT Greengrass ausführen.

AWS IoT Dienste steuern

Connect zu den folgenden AWS IoT Diensten her, um die Geräte in Ihrer IoT-Lösung zu verwalten.

AWS IoT Core

AWS IoT Coreist ein verwalteter Cloud-Dienst, der es verbundenen Geräten ermöglicht, sicher mit Cloud-Anwendungen und anderen Geräten zu interagieren. AWS IoT Core kann viele Geräte und Nachrichten unterstützen und diese Nachrichten verarbeiten und an AWS IoT Endpunkte und andere Geräte weiterleiten. Mit AWS IoT Core können Ihre Anwendungen mit all Ihren Geräten interagieren, auch wenn sie nicht verbunden sind.

AWS IoT Core Geräte-Ratgeber

AWS IoT Core Device Advisor ist eine cloudbasierte, vollständig verwaltete Testfunktion zur Validierung von IoT-Geräten während der Entwicklung von Gerätesoftware. Device Advisor

bietet vorgefertigte Tests, mit denen Sie IoT-Geräte auf zuverlässige und sichere Konnektivität überprüfen können AWS IoT Core, bevor Sie Geräte in der Produktion einsetzen.

AWS IoT Device Defender

AWS IoT Device Defender hilft Ihnen, Ihre IoT-Geräteflotte zu schützen. AWS IoT Device Defender überprüft Ihre IoT-Konfigurationen kontinuierlich, um sicherzustellen, dass sie nicht von den bewährten Sicherheitsmethoden abweichen. AWS IoT Device Defender sendet eine Warnung, wenn es Lücken in Ihrer IoT-Konfiguration erkennt, die ein Sicherheitsrisiko darstellen könnten, z. B. wenn Identitätszertifikate von mehreren Geräten gemeinsam genutzt werden oder wenn ein Gerät mit einem widerrufenen Identitätszertifikat versucht, eine Verbindung herzustellen. AWS IoT Core

AWS IoT Geräteverwaltung

AWS IoT Mithilfe der Geräteverwaltungsdienste können Sie die Vielzahl der verbundenen Geräte, aus denen sich Ihre Geräteflotten zusammensetzen, verfolgen, überwachen und verwalten. AWS IoT Mithilfe von Geräteverwaltungsdiensten können Sie sicherstellen, dass Ihre IoT-Geräte nach ihrer Bereitstellung ordnungsgemäß und sicher funktionieren. Sie bieten auch sicheres Tunneling für den Zugriff auf Ihre Geräte, die Überwachung ihres Zustands, die Erkennung und Behebung von Problemen aus der Ferne sowie Dienste zur Verwaltung von Gerätesoftware- und Firmware-Updates.

AWS IoT Datendienste

Analysieren Sie die Daten der Geräte in Ihrer IoT-Lösung und ergreifen Sie geeignete Maßnahmen, indem Sie die folgenden AWS IoT Dienste nutzen.

Amazon Kinesis Video Streams

Mit Amazon Kinesis Video Streams können Sie Live-Videos von Geräten in die AWS Cloud streamen, wo sie dauerhaft gespeichert, verschlüsselt und indexiert werden, sodass Sie über benutzerfreundliche APIs auf Ihre Daten zugreifen können. Sie können mit Amazon Kinesis Video Streams enorme Mengen an Live-Videodaten aus Millionen von Quellen erfassen, darunter Smartphones, Sicherheitskameras, Webcams sowie Kameras in Fahrzeugen, Drohnen und anderen Geräten. Amazon Kinesis Video Streams ermöglicht Ihnen die Wiedergabe von Videos zur Live- und On-Demand-Ansicht und die schnelle Erstellung von Anwendungen, die durch die Integration mit Amazon Rekognition Video und Bibliotheken für ML-Frameworks die Vorteile von Computer Vision und Videoanalysen nutzen. Sie können außer Videodaten auch andere zeitlich

serialisierte Daten senden, wie beispielsweise Audiodaten, Wärmebilder, Tiefendaten, RADAR-Daten usw.

Amazon Kinesis Video Streams mit WebRTC

Amazon Kinesis Video Streams mit WebRTC bietet eine standardkonforme WebRTC-Implementierung als vollständig verwaltete Funktion. Sie können Amazon Kinesis Video Streams mit WebRTC verwenden, um Medien sicher live zu streamen oder bidirektionale Audio- oder Videointeraktionen zwischen beliebigen Kamera-IoT-Geräten und WebRTC-compliant Mobil- oder Webplayern durchzuführen. Da es sich um eine vollständig verwaltete Funktion handelt, müssen Sie keine WebRTC-related Cloud-Infrastruktur wie Signal- oder Media-Relay-Server aufbauen, betreiben oder skalieren, um Medien sicher zwischen Anwendungen und Geräten zu streamen. Mithilfe von Amazon Kinesis Video Streams mit WebRTC können Sie ganz einfach Anwendungen für Live-Peer-to-Peer-Medienstreaming oder Audio- oder Videointeraktivität in Echtzeit zwischen Kamera-IoT-Geräten, Webbrowsern und Mobilgeräten für eine Vielzahl von Anwendungsfällen erstellen.

AWS IoT FleetWise

AWS IoT FleetWiseist ein verwalteter Dienst, mit dem Sie Fahrzeugdaten nahezu in Echtzeit sammeln und in die Cloud übertragen können. Damit AWS IoT FleetWise können Sie auf einfache Weise Daten von Fahrzeugen sammeln und organisieren, die unterschiedliche Protokolle und Datenformate verwenden. AWS IoT FleetWise hilft dabei, Nachrichten auf niedriger Ebene in für Menschen lesbare Werte umzuwandeln und das Datenformat in der Cloud für Datenanalysen zu standardisieren. Sie können auch Datenerfassungsschemata definieren, um zu kontrollieren, welche Daten in Fahrzeugen gesammelt und wann sie in die Cloud übertragen werden sollen.

AWS IoT SiteWise

AWS IoT SiteWisesammelt, speichert, organisiert und überwacht Daten, die von Industrieanlagen über MQTT-Nachrichten oder APIs übertragen werden, in großem Umfang, indem Software bereitgestellt wird, die auf einem Gateway in Ihren Einrichtungen ausgeführt wird. Das Gateway stellt eine sichere Verbindung zu Ihren lokalen Datenservern her und automatisiert den Prozess der Erfassung und Organisation der Daten sowie deren Übertragung an die Cloud. AWS

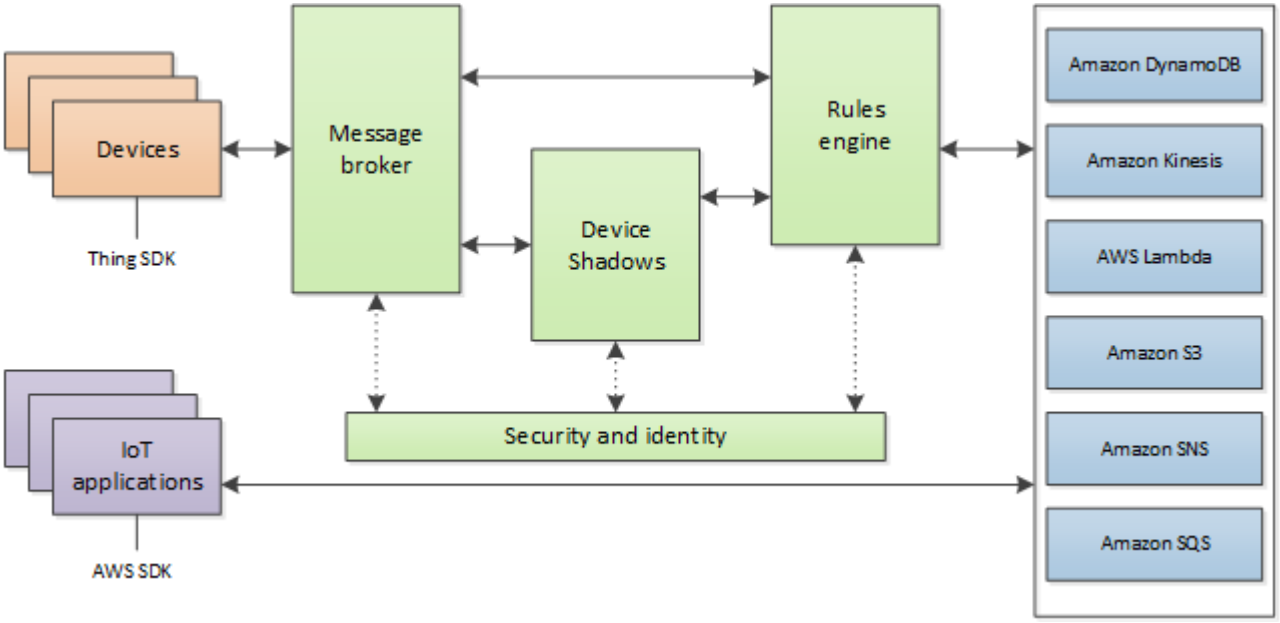
AWS IoT TwinMaker

AWS IoT TwinMakererstellt betriebsbereite digitale Zwillinge aus physischen und digitalen Systemen. AWS IoT TwinMaker erstellt digitale Visualisierungen mithilfe von Messungen und Analysen aus einer Vielzahl von realen Sensoren, Kameras und Unternehmensanwendungen, damit Sie den Überblick über Ihre physische Fabrik, Ihr Gebäude oder Ihre Industrieanlage

behalten. Sie können reale Daten verwenden, um den Betrieb zu überwachen, Fehler zu diagnostizieren und zu korrigieren und den Betrieb zu optimieren.

AWS IoT Core service

AWS IoT Core stellt die Dienste bereit, die Ihre IoT-Geräte mit der AWS Cloud verbinden, sodass andere Cloud-Dienste und -Anwendungen mit Ihren mit dem Internet verbundenen Geräten interagieren können.



Im nächsten Abschnitt werden die einzelnen in der Abbildung gezeigten AWS IoT Core Dienste beschrieben.

AWS IoT Core Messaging-Dienste

Die AWS IoT Core Konnektivitätsdienste bieten eine sichere Kommunikation mit den IoT-Geräten und verwalten die Nachrichten, die zwischen ihnen und übertragen AWS IoT werden.

Device Gateway

Ermöglicht Geräten die sichere und effiziente Kommunikation mit AWS IoT. Die Gerätekommunikation wird durch sichere Protokolle gesichert, die X.509 Zertifikate verwenden.

Message Broker

Bietet einen sicheren Mechanismus für Geräte und AWS IoT Anwendungen, um Nachrichten voneinander zu veröffentlichen und zu empfangen. Sie können entweder das MQTT-Protokoll

direkt oder MQTT-Over WebSocket zum Veröffentlichen und Abonnieren verwenden. Weitere Informationen zu von AWS IoT unterstützten Protokollen und Ports finden Sie unter the section called “Gerätekommunikationsprotokolle”. Geräte und Clients können auch die HTTP-REST-Schnittstelle verwenden, um Daten im Message Broker zu veröffentlichen.

Der Message Broker verteilt Gerätedaten an Geräte, die ihn abonniert haben, und an andere AWS IoT Core Dienste wie den Device Shadow-Dienst und die Rules Engine.

AWS IoT Core für WAN LoRa

AWS IoT Core for LoRa WAN ermöglicht die Einrichtung eines privaten LoRa WAN-Netzwerks, indem Sie Ihre LoRa WAN-Geräte und -Gateways miteinander verbinden, AWS ohne dass ein LoRa WAN-Netzwerkserver (LNS) entwickelt und betrieben werden muss. Von LoRa WAN-Geräten empfangene Nachrichten werden an die Rules Engine gesendet, wo sie formatiert und an andere Dienste gesendet werden können. AWS IoT

Regeln-Engine

Der Rules Engine verbindet Daten vom Message Broker mit anderen AWS IoT -Diensten zur Speicherung und weiteren Verarbeitung. Sie können beispielsweise eine DynamoDB-Tabelle einfügen, aktualisieren oder abfragen oder eine Lambda-Funktion aufrufen, die auf einem Ausdruck basiert, den Sie im Rules Engine definiert haben. Sie können eine SQL-based Sprache verwenden, um Daten aus Nachrichtennutzlasten auszuwählen und die Daten dann zu verarbeiten und an andere Dienste wie Amazon Simple Storage Service (Amazon S3), Amazon DynamoDB und zu senden. AWS Lambda Sie können den Message Broker auch verwenden, um Nachrichten erneut an weitere Abonnenten zu senden. Weitere Informationen finden Sie unter Regeln für AWS IoT.

AWS IoT Core Dienste steuern

Die AWS IoT Core Kontrolldienste bieten Funktionen zur Gerätesicherheit, Verwaltung und Registrierung.

Benutzerdefinierter Authentifizierungsservice

Sie können Genehmiger definieren, mit denen Sie Ihre eigene Authentifizierungs- und Autorisierungsstrategie unter Verwendung eines benutzerdefinierten Authentifizierungsservice und einer Lambda-Funktion verwalten können. Benutzerdefinierte Autorisierer ermöglichen die Authentifizierung Ihrer Geräte und AWS IoT die Autorisierung von Vorgängen mithilfe von Bearer-Token-Authentifizierungs- und Autorisierungsstrategien.

Sie können unterschiedliche Autorisierungsstrategien implementieren, z. B. JSON-Web-Verifizierung (JWT) und OAuth-Anbieter-Callout. Sie müssen Richtliniendokumente zurücksenden, die vom Gerätegateway zur Autorisierung von MQTT-Vorgängen verwendet werden. Weitere Informationen finden Sie unter Benutzerspezifische Authentifizierung und Autorisierung.

Service zur Gerätebereitstellung

Ermöglicht die Bereitstellung von Geräten mithilfe einer Vorlage, die die für Ihr Gerät erforderlichen Ressourcen beschreibt: ein Dingobjekt, ein Zertifikat und eine oder mehrere Richtlinien. Ein Dingobjekt ist ein Eintrag in der Registrierung, der Attribute zur Beschreibung eines Geräts enthält. Geräte verwenden Zertifikate zur Authentifizierung. AWS IoT Richtlinien bestimmen, welche Vorgänge ein Gerät in AWS IoT ausführen kann.

Die Vorlagen enthalten Variablen, die durch Werte in einem Wörterbuch (Map) ersetzt werden. Sie können mit derselben Vorlage mehrere Geräte bereitstellen, indem Sie einfach verschiedene Werte für die Vorlagenvariablen im Wörterbuch übergeben. Weitere Informationen finden Sie unter Gerätebereitstellung.

Gruppen-Registry

Mithilfe von Gruppen können Sie verschiedene Geräte gleichzeitig verwalten, indem Sie sie in Gruppen zusammenfassen. Gruppen können wiederum Gruppen enthalten – so können Sie eine Gruppenhierarchie aufbauen. Jede Aktion, die Sie an einer übergeordneten Gruppe ausführen, gilt auch für die untergeordneten Gruppen. Dieselbe Aktion gilt auch für alle Geräte in der übergeordneten Gruppe und für alle Geräte in den untergeordneten Gruppen. Berechtigungen für eine Gruppe gelten für alle Geräte in der Gruppe und in sämtlichen untergeordneten Gruppen. Weitere Informationen finden Sie unter Geräte verwalten mit AWS IoT.

Jobs-Service

Ermöglicht die Definition einer Reihe von Remote-Operationen, die an ein oder mehrere mit AWS IoT verbundene Geräte gesendet und dort ausgeführt werden. Sie können beispielsweise einen Auftrag definieren, der eine Reihe von Geräten anweist, Anwendungs- oder Firmware-Updates herunterzuladen und zu installieren, einen Neustart vorzunehmen, die Zertifikate zu rotieren oder Remote-Fehlerbehebungsvorgänge auszuführen.

Zum Erstellen eines Auftrags geben Sie eine Beschreibung der Remote-Operationen sowie eine Liste von Zielen an, die diese ausführen sollen. Bei den Zielen kann es sich um einzelne Geräte und/oder Gruppen handeln. Weitere Informationen finden Sie unter AWS IoT Jobs.

Registrierung

Organisiert die Ressourcen, die jedem Gerät in der AWS Cloud zugeordnet sind. Sie registrieren Ihre Geräte und weisen jedem bis zu drei benutzerdefinierte Attribute zu. Um Geräte besser verwalten und Fehler beheben zu können, können Sie jedem Gerät Zertifikate und MQTT-Client-IDs zuordnen. Weitere Informationen finden Sie unter Geräte verwalten mit AWS IoT.

Sicherheits- und Identitätsservice

Bietet gemeinsame Verantwortung für die Sicherheit in der AWS Cloud. Ihre Geräte müssen ihre Anmeldeinformationen sicher aufbewahren, damit Daten sicher an den Message Broker gesendet werden können. Message Broker und Rules Engine verwenden AWS-Sicherheitsfunktionen, um Daten sicher an Geräte oder sonstige AWS -Dienste zu senden. Weitere Informationen finden Sie unter Authentifizierung.

AWS IoT Core Datendienste

Die AWS IoT Core Datendienste helfen Ihren IoT-Lösungen dabei, auch mit Geräten, die nicht immer verbunden sind, ein zuverlässiges Anwendungserlebnis zu bieten.

Geräteschatten

Ein JSON-Dokument, das dazu verwendet wird, um aktuelle Statusinformationen für ein Gerät zu speichern und abzurufen.

Geräteschatten-Service

Der Geräteschatten-Dienst behält den Status eines Geräts bei, sodass Anwendungen mit einem Gerät kommunizieren können – unabhängig davon, ob das Gerät online ist oder nicht. Wenn ein Gerät offline ist, verwaltet der Geräteschatten-Dienst seine Daten für verbundene Anwendungen. Wenn das Gerät wieder eine Verbindung herstellt, synchronisiert es seinen Status mit dem seines Schattens im Geräteschatten-Dienst. Außerdem können Ihre Geräte ihren aktuellen Status zur Verwendung durch Anwendungen oder andere Geräte veröffentlichen. Weitere Informationen finden Sie unter AWS IoT Geräteschatten-Service.

AWS IoT Core Support-Service

Amazon Sidewalk-Integration für AWS IoT Core

Amazon Sidewalk ist ein gemeinsam genutztes Netzwerk, das die Konnektivitätsoptionen verbessert, damit Geräte besser zusammenarbeiten können. Amazon Sidewalk unterstützt eine

Vielzahl von Kundengeräten, z. B. Geräte zur Ortung von Haustieren oder Wertsachen, Geräte, die Smart-Home-Sicherheit und Lichtsteuerung bieten, und Geräte, die Ferndiagnosen für Geräte und Werkzeuge ermöglichen. Amazon Sidewalk Integration for AWS IoT Core ermöglicht es Geräteherstellern, ihre Sidewalk-Geräteflotte zur AWS IoT Cloud hinzuzufügen.

Weitere Informationen finden Sie unter AWS IoT Core für Amazon Sidewalk.

Erfahren Sie mehr über AWS IoT

Dieses Thema hilft Ihnen, sich mit der Welt von vertraut zu machen AWS IoT. Sie erhalten allgemeine Informationen darüber, wie IoT-Lösungen in verschiedenen Anwendungsfällen eingesetzt werden, Schulungsressourcen, Links zu sozialen Medien für AWS IoT und alle anderen AWS Dienste sowie eine Liste der AWS IoT verwendeten Dienste und Kommunikationsprotokolle.

Schulungsressourcen für AWS IoT

Wir bieten diese Schulungen an, damit Sie mehr darüber erfahren AWS IoT und erfahren, wie Sie sie auf Ihr Lösungsdesign anwenden können.

- Einführung in AWS IoT

  Ein Videoüberblick über AWS IoT und die wichtigsten Dienste.
- Tauchen Sie tief ein AWS IoT Authentifizierung und Autorisierung

  Ein Kurs für Fortgeschrittene, der sich mit den Konzepten der AWS IoT Authentifizierung und Autorisierung befasst. Sie lernen, wie Sie Clients authentifizieren und autorisieren, auf die APIs der AWS IoT Steuerungsebene und der Datenebene zuzugreifen.
- Reihe „Grundlagen zum Internet of Things (Internet der Dinge)“

  Ein Lernpfad mit IoT-eLearning-Modulen zu verschiedenen IoT-Technologien und -Funktionen.

AWS IoT Ressourcen und Leitfäden

Dies sind ausführliche technische Ressourcen zu bestimmten Aspekten von AWS IoT.

- IoT-Objektiv — AWS IoT Well-Architected Framework

  Ein Dokument, das die Best Practices für die Architektur Ihrer IoT-Anwendungen beschreibt. AWS

- Gestaltung von MQTT-Themen für AWS IoT Core

  Ein Whitepaper, das die bewährten Methoden für die Gestaltung von MQTT-Themen AWS IoT Core und die Nutzung AWS IoT Core von Funktionen mit MQTT beschreibt.

- Zusammenfassung und Einführung

  Ein PDF-Dokument, das die verschiedenen Möglichkeiten beschreibt, wie große Geräteflotten AWS IoT bereitgestellt werden können.

- AWS IoT Core Device Advisor

  AWS IoT Core Device Advisor bietet vorgefertigte Tests, mit denen Sie IoT-Geräte auf zuverlässige und sichere Konnektivität überprüfen können AWS IoT Core, bevor Sie Geräte in der Produktion einsetzen.

- AWS IoT Ressourcen

  IoT-specific Ressourcen wie technische Leitfäden, Referenzarchitekturen, E-Books und kuratierte Blogbeiträge, die in einem durchsuchbaren Index präsentiert werden.

- IoT-Atlas

  Übersichten zur Lösung gängiger IoT-Designprobleme. Der IoT-Atlas bietet detaillierte Einblicke in die Designherausforderungen, auf die Sie bei der Entwicklung Ihrer IoT-Lösung wahrscheinlich stoßen werden.

- AWS Whitepapers, Leitfäden &

  Unsere aktuelle Sammlung von Whitepapern und Leitfäden zu und anderen Technologien. AWS IoT AWS

AWS IoT in den sozialen Medien

Diese Social-Media-Kanäle informieren über AWS IoT und AWS verwandte Themen.

- Das Internet der Dinge auf AWS IoT — Offizieller Blog
- AWS IoT Videos im Amazon Web Services Services-Kanal auf YouTube

Diese Social-Media-Konten decken alle AWS Dienste ab, einschließlich AWS IoT

- Der Amazon Web Services Services-Kanal auf YouTube

- Amazon Web Services auf Twitter
- Amazon Web Services auf Facebook
- Amazon Web Services auf Instagram
- Amazon Web Services auf LinkedIn

AWS Dienste, die verwendet werden von AWS IoT Core Regel-Engine

Die AWS IoT Core Regel-Engine kann eine Verbindung zu diesen AWS Diensten herstellen.

- Amazon DynamoDB

  Amazon DynamoDB ist ein skalierbarer NoSQL-Datenbank-Service, der schnelle und vorhersehbare Datenbankleistung bietet.

- Amazon Kinesis

  Amazon Kinesis macht es einfach, Streaming-Daten in Echtzeit zu sammeln, zu verarbeiten und zu analysieren, sodass Sie zeitnahe Einblicke erhalten und schnell auf neue Informationen reagieren können. Amazon Kinesis kann Echtzeitdaten wie Video-, Audio-, Anwendungsprotokolle, Website-Clickstreams und IoT-Telemetriedaten für maschinelles Lernen, Analysen und andere Anwendungen aufnehmen.

- AWS Lambda

  AWS Lambda ermöglicht es Ihnen, Code auszuführen, ohne Server bereitzustellen oder zu verwalten. Sie können Ihren Code so einrichten, dass er automatisch bei AWS IoT Daten und Ereignissen ausgelöst wird, oder ihn direkt über eine Web- oder Mobil-App aufrufen.

- Amazon Simple Storage Service

  Amazon Simple Storage Service (Amazon S3) kann jede Datenmenge jederzeit und von überall im Internet speichern und abrufen. AWS IoT Regeln können Daten zur Speicherung an Amazon S3 senden.

- Amazon Simple Notification Service

  Amazon Simple Notification Service (Amazon SNS) ist ein Webservice, mit dem Anwendungen, Endanwender und Geräte sofort Benachrichtigungen senden und empfangen können.

- Amazon Simple Queue Service

Amazon Simple Queue Service (Amazon SQS) ist Nachrichten-Warteschlangen-Service, der Microservices, verteilte Systeme und Serverless Anwendungen entkoppelt und skaliert.

- OpenSearch Amazon-Dienst

  Amazon OpenSearch Service (OpenSearch Service) ist ein verwalteter Service, der die Bereitstellung, den Betrieb und die Skalierung OpenSearch vereinfacht. Dabei handelt es sich um eine beliebte Open-Source-Such- und Analyse-Engine.

- Amazon SageMaker KI

  Amazon SageMaker AI kann Modelle für maschinelles Lernen (ML) erstellen, indem es Muster in Ihren IoT-Daten findet. Der Service verwendet diese Modelle, um neue Daten zu verarbeiten und Prognosen für Ihre Anwendung zu generieren.

- Amazon CloudWatch

  Amazon CloudWatch bietet eine zuverlässige, skalierbare und flexible Überwachungslösung, mit der Sie Ihre eigenen Überwachungssysteme und -infrastruktur einrichten, verwalten und skalieren können.

Kommunikationsprotokolle, unterstützt von AWS IoT Core

Diese Themen enthalten weitere Informationen zu den Kommunikationsprotokollen, die von AWS IoT verwendet werden. Weitere Informationen zu den Protokollen, die von Geräten und Diensten verwendet werden AWS IoT und mit denen Geräte und Dienste verbunden werden AWS IoT, finden Sie unterConnect to (Verbinden mit) AWS IoT Core.

- MQTT (Message Queuing Telemetry Transport)

  Die Homepage der MQTT.org Website, auf der Sie die Spezifikationen des MQTT-Protokolls finden. Weitere Informationen darüber, wie MQTT AWS IoT unterstützt wird, finden Sie unter. MQTT

- HTTPS (Hypertext Transfer Protocol – Secure)

  Geräte und Apps können über HTTPS auf AWS IoT Dienste zugreifen.

- LoRaWAN (Weitverkehrsnetzwerk mit großer Reichweite)

  LoRaWAN-Geräte und Gateways können mithilfe AWS IoT Core von AWS IoT Core for LoRa WAN eine Verbindung herstellen.

- TLS (Transport Layer Security) v1.3

  Die Spezifikation von TLS v1.3 (RFC 5246). AWS IoT verwendet TLS v1.3, um sichere Verbindungen zwischen Geräten und herzustellen. AWS IoT

Was ist neu in der AWS IoT Konsole

Wir sind dabei, die Benutzeroberfläche der AWS IoT Konsole an ein neues Benutzererlebnis anzupassen. Wir aktualisieren die Benutzeroberfläche schrittweise, sodass einige Seiten der Konsole ein neues Benutzererlebnis bieten, andere sowohl das ursprüngliche als auch das neue Erlebnis bieten und wieder andere möglicherweise nur das ursprüngliche Erlebnis.

In dieser Tabelle wird der Stand einzelner Bereiche der AWS IoT Konsolen-Benutzeroberfläche am 27. Januar 2022 angezeigt.

AWS IoT Status der Konsolen-Benutzeroberfläche

Konsolenseite	Ursprüngliches Erlebnis	Neues Erlebnis	Kommentare
Monitor	Nicht verfügbar	Available (Verfügbar)	
Aktivität	Nicht verfügbar	Available (Verfügbar)	
Einstieg – Erste Schritte	Nicht verfügbar	Available (Verfügbar)	Nicht in allen CN-Regionen verfügbar
Einstieg – Vorlagen für die Flottenbe reitstellung	Available (Verfügbar)	Available (Verfügbar)	
Verwalten – Dingobjek te	Available (Verfügbar)	Available (Verfügbar)	
Verwalten – Typen	Available (Verfügbar)	Available (Verfügbar)	
Verwalten – Dinggruppen	Available (Verfügbar)	Available (Verfügbar)	

Konsolenseite	Ursprüngliches Erlebnis	Neues Erlebnis	Kommentare
Verwalten – Abrechnungsgruppen	Available (Verfügbar)	Available (Verfügbar)	
Verwalten – Jobs	Available (Verfügbar)	Available (Verfügbar)	
Verwalten – Jobvorlagen	Nicht verfügbar	Available (Verfügbar)	
Verwalten – Tunnel	Nicht verfügbar	Available (Verfügbar)	
Fleet Hub – Erste Schritte	Nicht verfügbar	Available (Verfügbar)	Nicht in allen verfügbar AWS-Regionen
Fleet Hub – Anwendungen	Nicht verfügbar	Available (Verfügbar)	Nicht in allen verfügbar AWS-Regionen
Greengrass – Erste Schritte	Nicht verfügbar	Available (Verfügbar)	Nicht in allen verfügbar AWS-Regionen
Greengrass – Core-Geräte	Nicht verfügbar	Available (Verfügbar)	Nicht in allen verfügbar AWS-Regionen
Greengrass – Komponenten	Nicht verfügbar	Available (Verfügbar)	Nicht in allen verfügbar AWS-Regionen
Greengrass – Bereitstellungen	Nicht verfügbar	Available (Verfügbar)	Nicht in allen verfügbar AWS-Regionen
Greengrass – Klassisch (V1)	Available (Verfügbar)	Available (Verfügbar)	

Konsolenseite	Ursprüngliches Erlebnis	Neues Erlebnis	Kommentare
Drahtlose Konnektivität – Einführung	Nicht verfügbar	Available (Verfügbar)	Nicht in allen verfügbar AWS-Regionen
Drahtlose Konnektivität – Gateways	Nicht verfügbar	Available (Verfügbar)	Nicht in allen verfügbar AWS-Regionen
Drahtlose Konnektivität – Geräte	Nicht verfügbar	Available (Verfügbar)	Nicht in allen verfügbar AWS-Regionen
Drahtlose Konnektivität – Profile	Nicht verfügbar	Available (Verfügbar)	Nicht in allen verfügbar AWS-Regionen
Drahtlose Konnektivität – Ziele	Nicht verfügbar	Available (Verfügbar)	Nicht in allen verfügbar AWS-Regionen
Secure – Zertifikate	Available (Verfügbar)	Available (Verfügbar)	
Secure – Richtlinien	Available (Verfügbar)	Available (Verfügbar)	
Secure – CAs	Available (Verfügbar)	Available (Verfügbar)	
Secure – Rollenaliase	Available (Verfügbar)	Available (Verfügbar)	
Secure – Autorisierer	Available (Verfügbar)	Available (Verfügbar)	
Defend – Einführung	Nicht verfügbar	Available (Verfügbar)	
Defend – Audit	Nicht verfügbar	Available (Verfügbar)	
Defend – Erkennen	Nicht verfügbar	Available (Verfügbar)	

Konsolenseite	Ursprüngliches Erlebnis	Neues Erlebnis	Kommentare
Defend – Maßnahmen zur Schadensbegrenzung	Nicht verfügbar	Available (Verfügbar)	
Defend – Einstellungen	Nicht verfügbar	Available (Verfügbar)	
Act – Regeln	Available (Verfügbar)	Available (Verfügbar)	
Act – Ziele	Available (Verfügbar)	Available (Verfügbar)	
Test – Geräteberater	Available (Verfügbar)	Available (Verfügbar)	Nicht in allen verfügbar AWS-Regionen
Test – MQTT-Testclient	Available (Verfügbar)	Available (Verfügbar)	
Software	Available (Verfügbar)	Available (Verfügbar)	
Einstellungen	Nicht verfügbar	Available (Verfügbar)	
Lernen	Available (Verfügbar)	Noch nicht verfügbar	

Legende

Statuswerte

- Available (Verfügbar)

  Diese Benutzerschnittstelle kann genutzt werden.
- Nicht verfügbar

  Diese Benutzerschnittstelle kann nicht genutzt werden.
- Noch nicht verfügbar

An der neuen Benutzeroberfläche wird gearbeitet, sie ist aber noch nicht fertig.

- In Bearbeitung

  Die neue Benutzeroberfläche wird gerade aktualisiert. Einige Seiten bieten jedoch möglicherweise immer noch die ursprüngliche Benutzererfahrung.

Verwenden AWS IoT mit einem AWS SDK

AWS Software Development Kits (SDKs) sind für viele beliebte Programmiersprachen verfügbar. Jedes SDK bietet eine API, Codebeispiele und Dokumentation, die es Entwicklern erleichtern, Anwendungen in ihrer bevorzugten Sprache zu erstellen.

SDK-Dokumentation	Codebeispiele
AWS SDK für C++	AWS SDK für C++ Code-Beispiele
AWS CLI	AWS CLI Codebeispiele
AWS SDK für Go	AWS SDK für Go Codebeispiele
AWS SDK für Java	AWS SDK für Java Codebeispiele
AWS SDK für JavaScript	AWS SDK für JavaScript Codebeispiele
AWS SDK für Kotlin	AWS SDK für Kotlin Codebeispiele
AWS SDK für .NET	AWS SDK für .NET Codebeispiele
AWS SDK für PHP	AWS SDK für PHP Codebeispiele
AWS -Tools für PowerShell	AWS -Tools für PowerShell Codebeispiele
AWS SDK für Python v4	AWS SDK für Python v4-Codebeispiele
AWS SDK für Python (Boto3)	AWS SDK für Python (Boto3) Codebeispiele
AWS SDK für Ruby	AWS SDK für Ruby Codebeispiele
AWS SDK für Rust	AWS SDK für Rust Codebeispiele

SDK-Dokumentation	Codebeispiele
AWS SDK für SAP ABAP	AWS SDK für SAP ABAP Codebeispiele
AWS SDK für Swift	AWS SDK für Swift Codebeispiele

Beispiel für die Verfügbarkeit

Sie können nicht finden, was Sie brauchen? Fordern Sie ein Codebeispiel an, indem Sie unten den Link Provide feedback (Feedback geben) auswählen.

Erste Schritte mit AWS IoT Core Tutorials

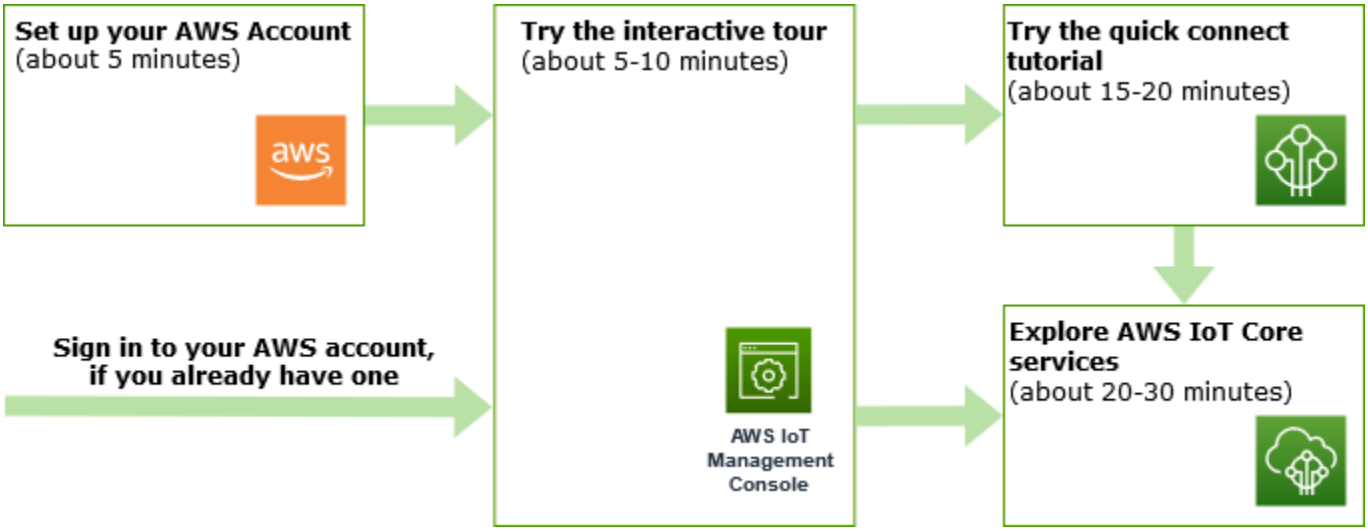
Ganz gleich, ob Sie mit IoT noch nicht vertraut sind oder über langjährige Erfahrung verfügen, in diesen Ressourcen finden Sie die AWS IoT Konzepte und Begriffe, die Ihnen beim Einstieg helfen werden AWS IoT.

- Schauen Sie hinein AWS IoT und sehen Sie sich die einzelnen Komponenten anWie AWS IoT funktioniert.
- Erfahren Sie mehr über AWS IoT in unserer Sammlung von Schulungsmaterialien und Videos. Dieses Thema enthält auch eine Liste von Diensten, mit denen AWS IoT eine Verbindung herstellen kann, Links zu sozialen Medien und Links zu Kommunikationsprotokollspezifikationen.
- the section called “Connect dein erstes Gerät mit AWS IoT Core”.
- Entwickeln Sie Ihre IoT-Lösungen mit Connect to (Verbinden mit) AWS IoT Core und erkunden Sie die AWS IoT Tutorials.
- Testen und validieren Sie Ihre IoT-Geräte für eine sichere und zuverlässige Kommunikation mit dem Device Advisor.
- Verwalten Sie Ihre Lösung mithilfe von AWS IoT Core Verwaltungsdiensten wie -Flottenindizierung, AWS IoT Jobs und AWS IoT Device Defender.
- Analysieren Sie die Daten von Ihren Geräten mithilfe der AWS IoT Datendienste.

Connect dein erstes Gerät mit AWS IoT Core

AWS IoT Core Dienste verbinden IoT-Geräte mit AWS IoT Diensten und anderen AWS Diensten. AWS IoT Core umfasst das Device Gateway und den Message Broker, die Nachrichten zwischen Ihren IoT-Geräten und der Cloud verbinden und verarbeiten.

So können Sie mit AWS IoT Core und beginnen AWS IoT.



In diesem Abschnitt werden die AWS IoT Core wichtigsten Dienste vorgestellt und es werden mehrere Beispiele dafür bereitgestellt, wie Sie ein Gerät mit diesen verbinden AWS IoT Core und Nachrichten zwischen ihnen weiterleiten können. Die Weitergabe von Nachrichten zwischen Geräten und der Cloud ist für jede IoT-Lösung von grundlegender Bedeutung. So können Ihre Geräte mit anderen AWS Diensten interagieren.

- Einrichten AWS-Konto

  Bevor Sie AWS IoT Dienste nutzen können, müssen Sie eine einrichten AWS-Konto. Wenn Sie bereits einen AWS-Konto und einen IAM-Benutzer für sich haben, können Sie diese verwenden und diesen Schritt überspringen.

- Probieren Sie das Quick Connect-Tutorial aus

  Dieses Tutorial eignet sich am besten, wenn Sie schnell damit beginnen AWS IoT und sehen möchten, wie es in einem begrenzten Szenario funktioniert. In diesem Tutorial benötigen Sie ein Gerät und installieren AWS IoT Software darauf. Wenn Sie kein IoT-Gerät haben, können Sie Ihren Windows-, Linux- oder macOS-PC als Gerät für dieses Tutorial verwenden. Wenn Sie es versuchen möchten AWS IoT, aber kein Gerät haben, versuchen Sie es mit der nächsten Option.

- Probieren Sie das interaktive Tutorial aus

  Diese Demo eignet sich am besten, wenn Sie sehen möchten, was eine AWS IoT Basislösung leisten kann, ohne ein Gerät anzuschließen oder Software herunterzuladen. Das interaktive Tutorial stellt eine simulierte Lösung vor, die auf AWS IoT Core Diensten basiert und veranschaulicht, wie sie interagieren.

- Erkunden AWS IoT Core Dienstleistungen mit einem praktischen Tutorial

Dieses Tutorial eignet sich am besten für Entwickler, die AWS IoT mit anderen AWS IoT Core Funktionen wie der Regel-Engine und Schatten beginnen möchten. Dieses Tutorial folgt einem ähnlichen Prozess wie das Schnellverbindungs-Tutorial, enthält jedoch mehr Details zu den einzelnen Schritten, um einen reibungsloseren Übergang zu den fortgeschritteneren Tutorials zu ermöglichen.

- Sehen Sie sich MQTT-Nachrichten an mit AWS IoT MQTT-Client

  Erfahren Sie, wie Sie den MQTT-Testclient verwenden, um zu beobachten, wie Ihr erstes Gerät MQTT-Nachrichten mit AWS IoT veröffentlicht. Der MQTT-Testclient ist ein nützliches Tool zur Überwachung und Fehlerbehebung von Geräteverbindungen.

> **Note**
>
> Wenn Sie mehr als eines dieser Tutorials für die ersten Schritte ausprobieren oder dasselbe Tutorial wiederholen möchten, sollten Sie das Objekt löschen, das Sie in einem früheren Tutorial erstellt haben, bevor Sie ein anderes beginnen. Wenn Sie das Objekt nicht aus einem früheren Tutorial löschen, müssen Sie für nachfolgende Tutorials einen anderen Objektnamen verwenden. Dies liegt daran, dass der Objektname in Ihrem Konto und AWS-Region eindeutig sein muss.

Weitere Informationen zu finden Sie AWS IoT Core unter Was ist AWS IoT Core?

Einrichten AWS-Konto

Führen Sie AWS IoT Core vor der ersten Verwendung die folgenden Aufgaben aus:

Topics

- Melde dich an für ein AWS-Konto
- Öffnen Sie AWS IoT Konsole

Melde dich an für ein AWS-Konto

Um loszulegen AWS, benötigen Sie eine AWS-Konto. Informationen zum Erstellen eines AWS-Konto finden Sie unter Erste Schritte mit einem AWS-Konto im AWS -Kontenverwaltung Referenzhandbuch.

- Öffnen Sie AWS IoT Konsole

Öffnen Sie AWS IoT Konsole

Die meisten konsolenorientierten Themen in diesem Abschnitt beginnen mit der Konsole. AWS IoT Wenn Sie noch nicht bei Ihrem angemeldet sind AWS-Konto, melden Sie sich an, öffnen Sie dann die AWS IoT Konsole und fahren Sie mit dem nächsten Abschnitt fort, um mit den ersten Schritten fortzufahren. AWS IoT

Interaktives Tutorial

Das interaktive Tutorial zeigt die Komponenten einer einfachen IoT-Lösung, die auf AWS IoT basiert. Das Tutorial zeigt, wie IoT-Geräte mit AWS IoT Core Diensten interagieren. Dieses Thema bietet eine Vorschau auf das AWS IoT Core interaktive Tutorial.

> **Note**
>
> Die Bilder in der Konsole enthalten Animationen, die in den Bildern dieses Tutorials nicht vorkommen.

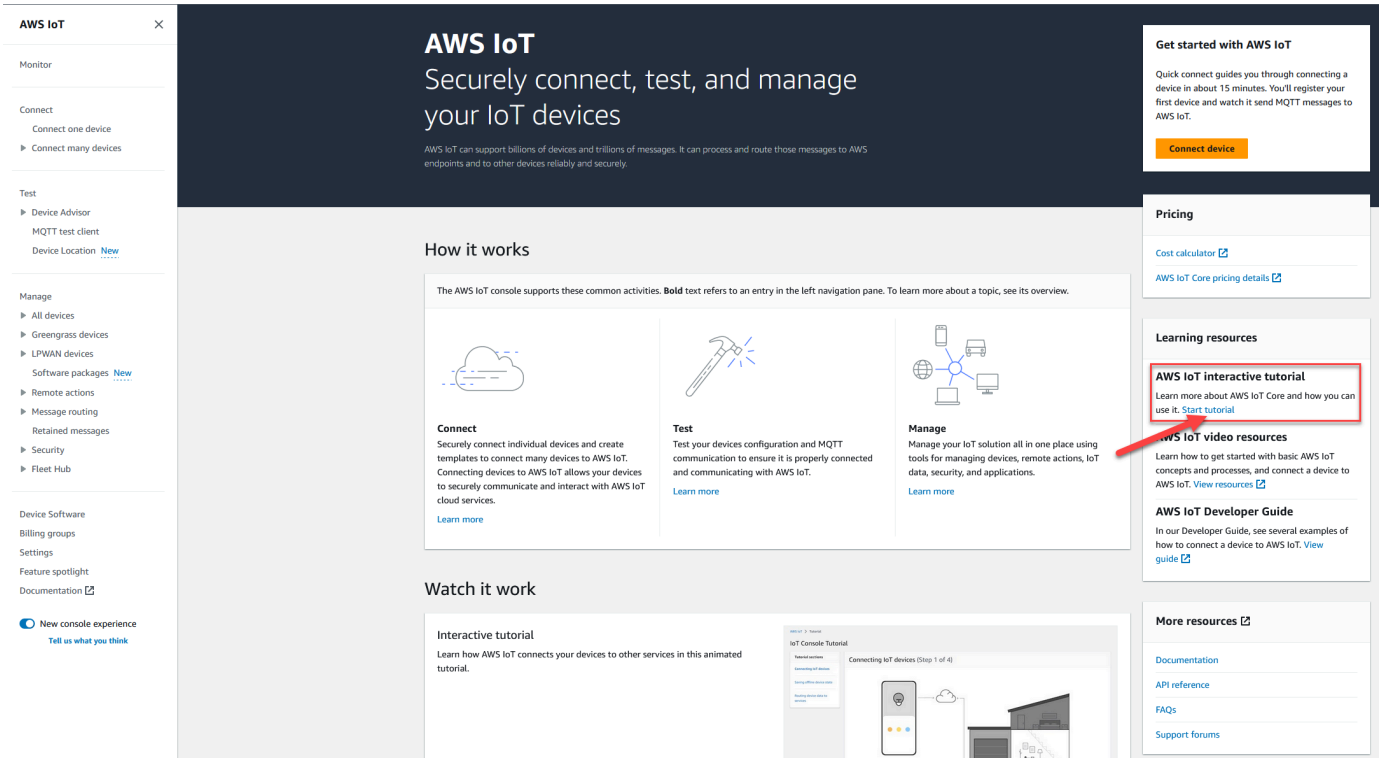
Um die Demo auszuführen, müssen Sie zuerst the section called “Einrichten AWS-Konto”. Für das Tutorial sind jedoch keine AWS IoT Ressourcen, zusätzliche Software oder Programmierung erforderlich.

Rechnen Sie damit, etwa 5-10 Minuten mit dieser Demo zu verbringen. Wenn Sie sich 10 Minuten Zeit nehmen, haben Sie mehr Zeit, um die einzelnen Schritte zu verstehen.

Um das auszuführen AWS IoT Core interaktives Tutorial

1. Öffnen Sie die AWS IoT Startseite in der AWS IoT Konsole.

   Wählen Sie auf der AWS IoT -Startseite im Fensterbereich Lernressourcen die Option Tutorial starten aus.



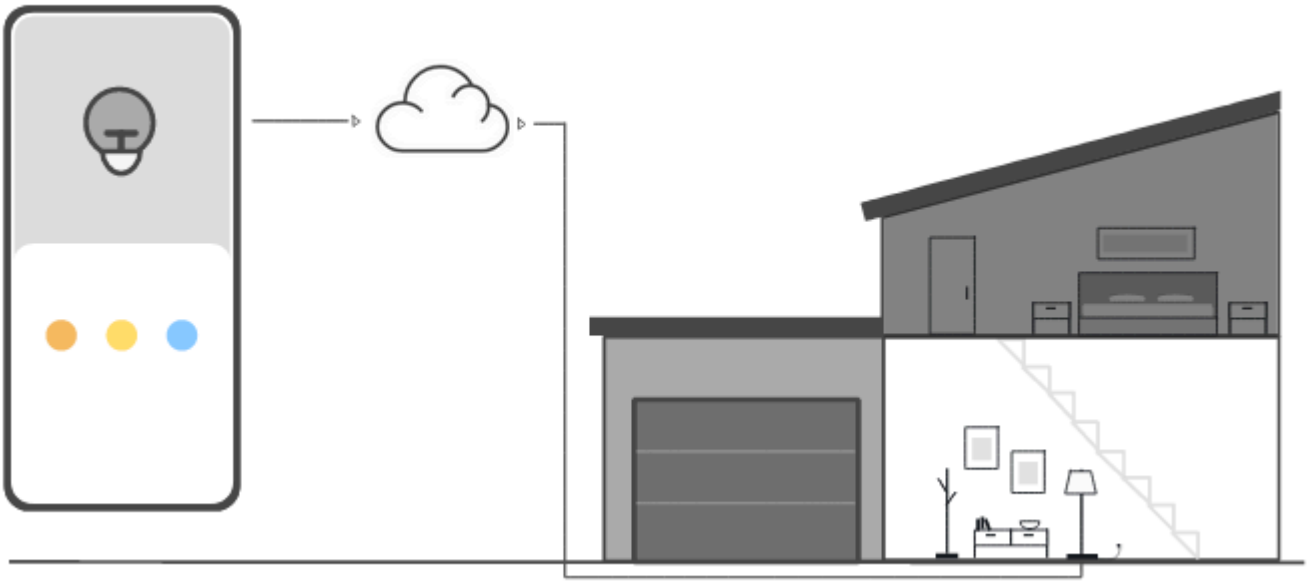
2. Sehen Sie sich auf der Seite mit dem AWS IoT Konsolen-Tutorial die Abschnitte mit den Tutorials an und wählen Sie den Abschnitt Start, wenn Sie bereit sind, fortzufahren.

In den folgenden Abschnitten wird beschrieben, wie das AWS IoT Konsolen-Tutorial diese AWS IoT Core Funktionen präsentiert:

- IoT-Geräte verbinden
- Der Status des Offline-Geräts wird gespeichert
- Gerätedaten an Dienste weiterleiten

IoT-Geräte verbinden

Erfahren Sie, wie IoT-Geräte mit kommunizieren AWS IoT Core.

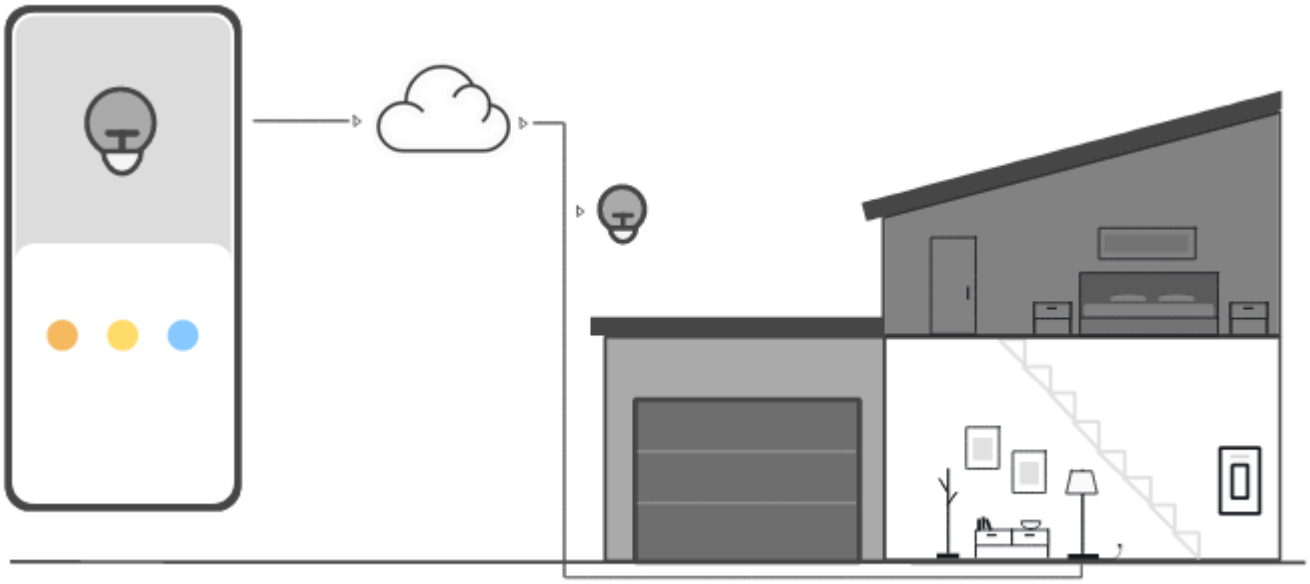


Die Animation in diesem Schritt zeigt, wie sich zwei Geräte, das Steuergerät auf der linken Seite und eine intelligente Lampe im Haus auf der rechten Seite, mit AWS IoT Core in der Cloud verbinden und kommunizieren. Die Animation zeigt, wie die Geräte mit den empfangenen Nachrichten kommunizieren AWS IoT Core und darauf reagieren.

Weitere Informationen zum Anschließen von Geräten an finden Sie AWS IoT Core unterConnect to (Verbinden mit) AWS IoT Core.

Der Status des Offline-Geräts wird gespeichert

Erfahren Sie, wie der Gerätestatus AWS IoT Core gespeichert wird, solange ein Gerät oder eine App offline ist.



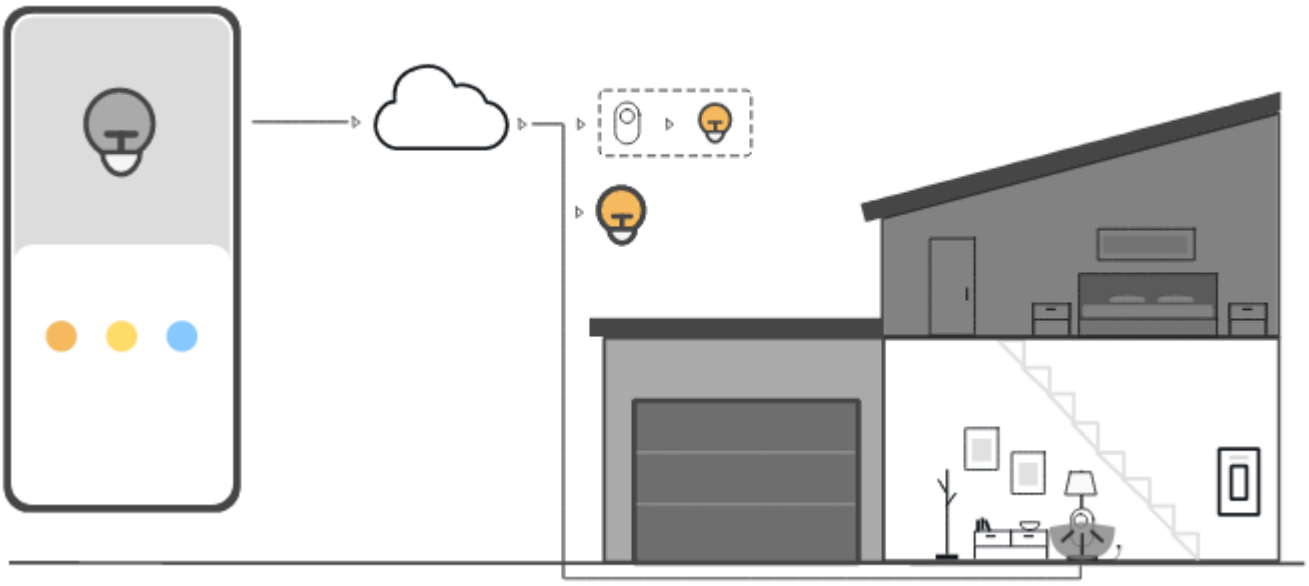
Die Animation in diesem Schritt zeigt, wie der Device Shadow-Dienst Gerätestatusinformationen für das Steuergerät und die intelligente Lampe AWS IoT Core speichert. Während die intelligente Lampe offline ist, speichert der Geräteschatten Befehle vom Steuergerät.

Wenn die Smart-Lampe wieder eine Verbindung mit herstellt AWS IoT Core, ruft sie diese Befehle ab. Wenn das Steuergerät offline ist, speichert der Geräteschatten Statusinformationen von der intelligenten Lampe. Wenn das Steuergerät wieder eine Verbindung herstellt, ruft es den aktuellen Status der intelligenten Lampe ab, um deren Anzeige zu aktualisieren.

Weitere Informationen finden Sie unter Geräteschatten, siehe AWS IoT Geräteschatten-Service.

Gerätedaten an Dienste weiterleiten

Erfahren Sie, wie der Gerätestatus an andere AWS Dienste AWS IoT Core gesendet wird.



Die Animation in diesem Schritt zeigt, wie Daten mithilfe von AWS IoT Regeln von den Geräten an andere AWS Dienste AWS IoT Core gesendet werden. AWS IoT Regeln abonnieren bestimmte Nachrichten von den Geräten, interpretieren die Daten in diesen Nachrichten und leiten die interpretierten Daten an andere Dienste weiter. In diesem Beispiel interpretiert eine AWS IoT Regel Daten von einem Bewegungssensor und sendet Befehle an einen Device Shadow, der sie dann an die Smart-Bulb sendet. Wie im vorherigen Beispiel speichert der Geräteschatten die Gerätestatusinformationen für das Steuergerät.

Weitere Informationen zu AWS IoT Regeln finden Sie unterRegeln für AWS IoT.

Versuchen Sie es mit AWS IoT Core Schnellverbindungs-Tutorial

In diesem Tutorial erstellen Sie Ihr erstes Objekt, verbinden ein Gerät damit und sehen, wie es MQTT-Nachrichten sendet.

Sie können damit rechnen, 15 bis 20 Minuten für dieses Tutorial aufzuwenden.

Dieses Tutorial eignet sich am besten für Personen, die schnell damit beginnen möchten AWS IoT , zu erfahren, wie es in einem begrenzten Szenario funktioniert. Wenn Sie nach einem Beispiel suchen, das Ihnen den Einstieg erleichtert, damit Sie weitere Funktionen und Dienste erkunden können, versuchen Sie Erkunden AWS IoT Core in praktischen Tutorials.

In diesem Tutorial laden Sie Software herunter und führen sie auf einem Gerät aus, das im AWS IoT Core Rahmen einer sehr kleinen IoT-Lösung eine Verbindung zu einer Ding-Ressource herstellt. Das

Gerät kann ein IoT-Gerät sein, z. B. ein Raspberry Pi, oder es kann sich auch um einen Computer handeln, auf dem Linux, OS und OSX oder Windows ausgeführt werden. Wenn Sie ein WAN-Gerät (Long Range WAN) mit einem LoRa WAN verbinden möchten AWS IoT, finden Sie weitere Informationen im Tutorial >Geräte und Gateways mit AWS IoT Core für LoRa WAN verbinden.

Wenn Ihr Gerät einen Browser unterstützt, der die AWS IoT -Konsole ausführen kann, empfehlen wir Ihnen, dieses Tutorial auf diesem Gerät abzuschließen.

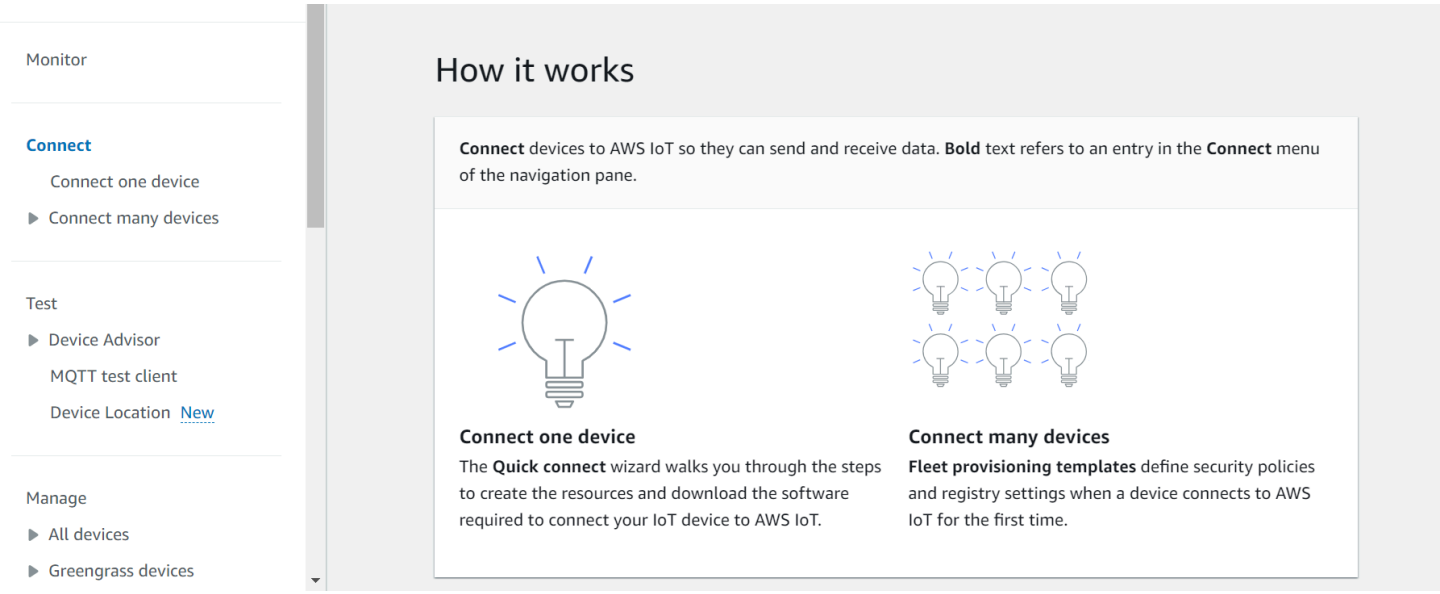
> **Note**
>
> Wenn Ihr Gerät keinen kompatiblen Browser hat, folgen Sie diesem Tutorial auf einem Computer. Wenn Sie im Rahmen des Verfahrens aufgefordert werden, die Datei herunterzuladen, laden Sie sie auf Ihren Computer herunter und übertragen Sie dann die heruntergeladene Datei mithilfe von Secure Copy (SCP) oder einem ähnlichen Verfahren auf Ihr Gerät.

Für das Tutorial muss Ihr IoT-Gerät mit Port 8443 an dem Gerätedatenendpunkt Ihres AWS-Konto kommunizieren. Um zu testen, ob es auf diesen Port zugreifen kann, probieren Sie die Verfahren unter Testen Sie die Konnektivität mit Ihrem Gerätedatenendpunkt aus.

Schritt 1. Beginnen Sie mit dem Tutorial

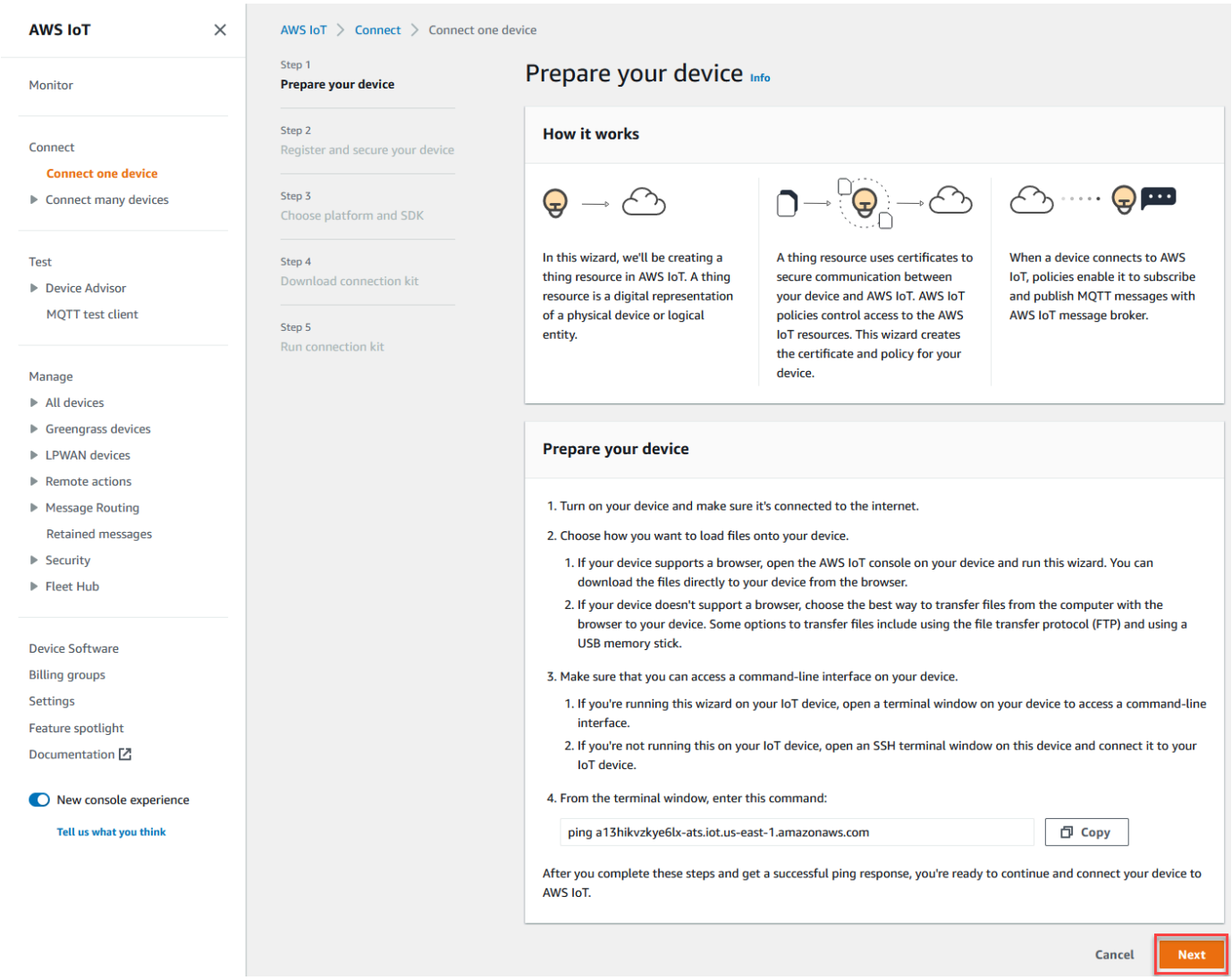
Wenn möglich, führen Sie dieses Verfahren auf Ihrem Gerät aus. Andernfalls sollten Sie bereit sein, später in diesem Verfahren eine Datei auf Ihr Gerät zu übertragen.

Melden Sie sich an der AWS IoT -Konsole an, um das Tutorial zu starten. Wählen Sie auf der Startseite der AWS IoT Konsole auf der linken Seite Connect und dann Connect one device (Ein Gerät verbinden).



Schritt 2. Dies erstellt ein Objekt

1. Folgen Sie im Abschnitt Gerät vorbereiten den Anweisungen auf dem Bildschirm, um Ihr Gerät für die Verbindung mit AWS IoT vorzubereiten.

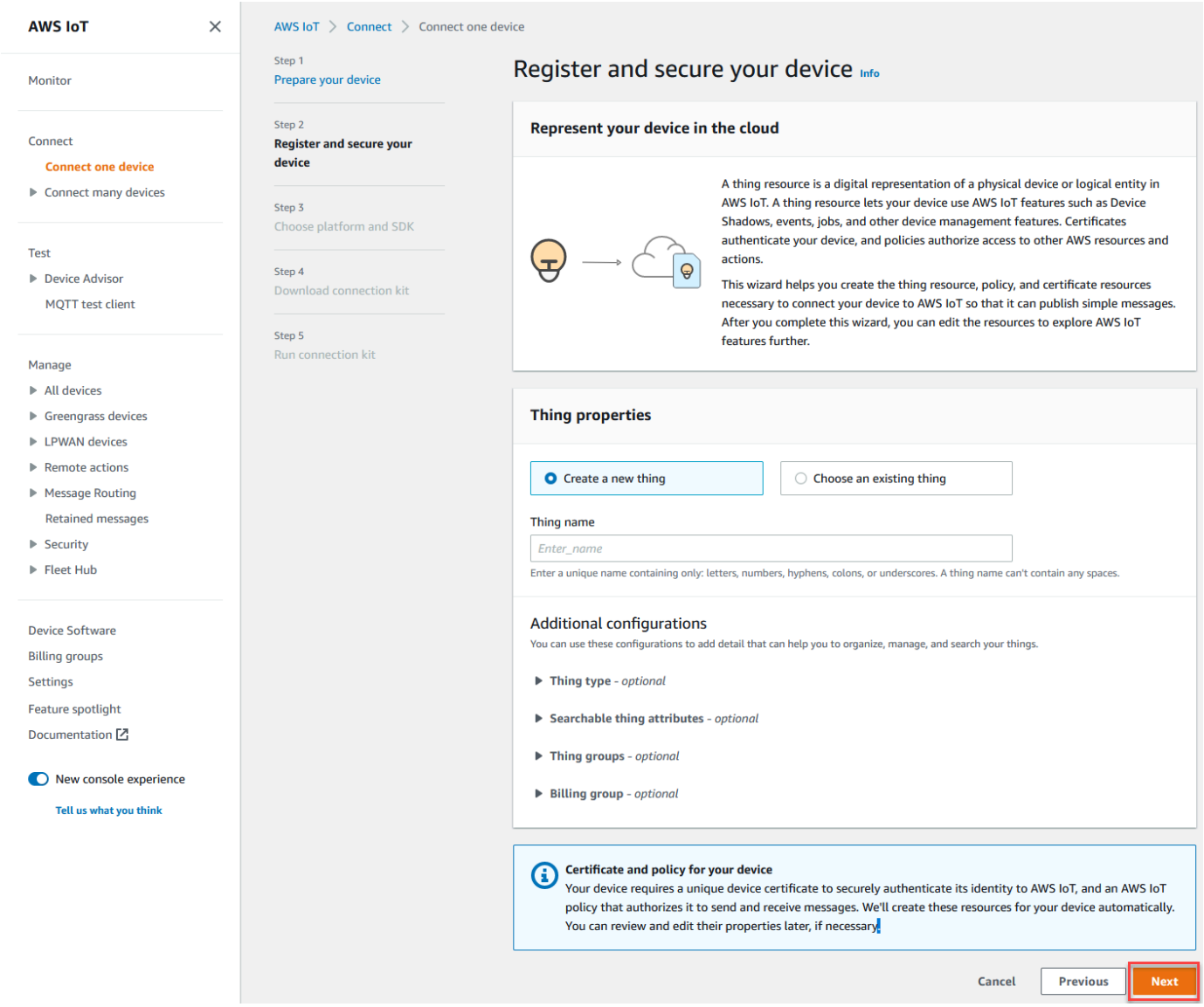


2. Wählen Sie im Bereich Gerät registrieren und sichern die Option Neues Objekt erstellen oder Bestehendes Objekt auswählen aus. Geben Sie im Feld Objektname den Namen für Ihr Objekt ein. Der in diesem Beispiel verwendete Objektname lautet **`TutorialTestThing`**

> **Important**
>
> Double-check Name deines Dings, bevor du weitermachst.
> Ein Objektname kann nicht mehr geändert werden, nachdem das Objekt erstellt wurde. Wenn Sie den Namen eines Objekts ändern möchten, müssen Sie ein neues Objekt mit dem richtigen Namen erstellen und dann das Objekt mit dem falschen Namen löschen.

Passen Sie im Abschnitt Zusätzliche Konfigurationen Ihre Objekt-Ressource mithilfe der aufgelisteten optionalen Konfigurationen weiter an.

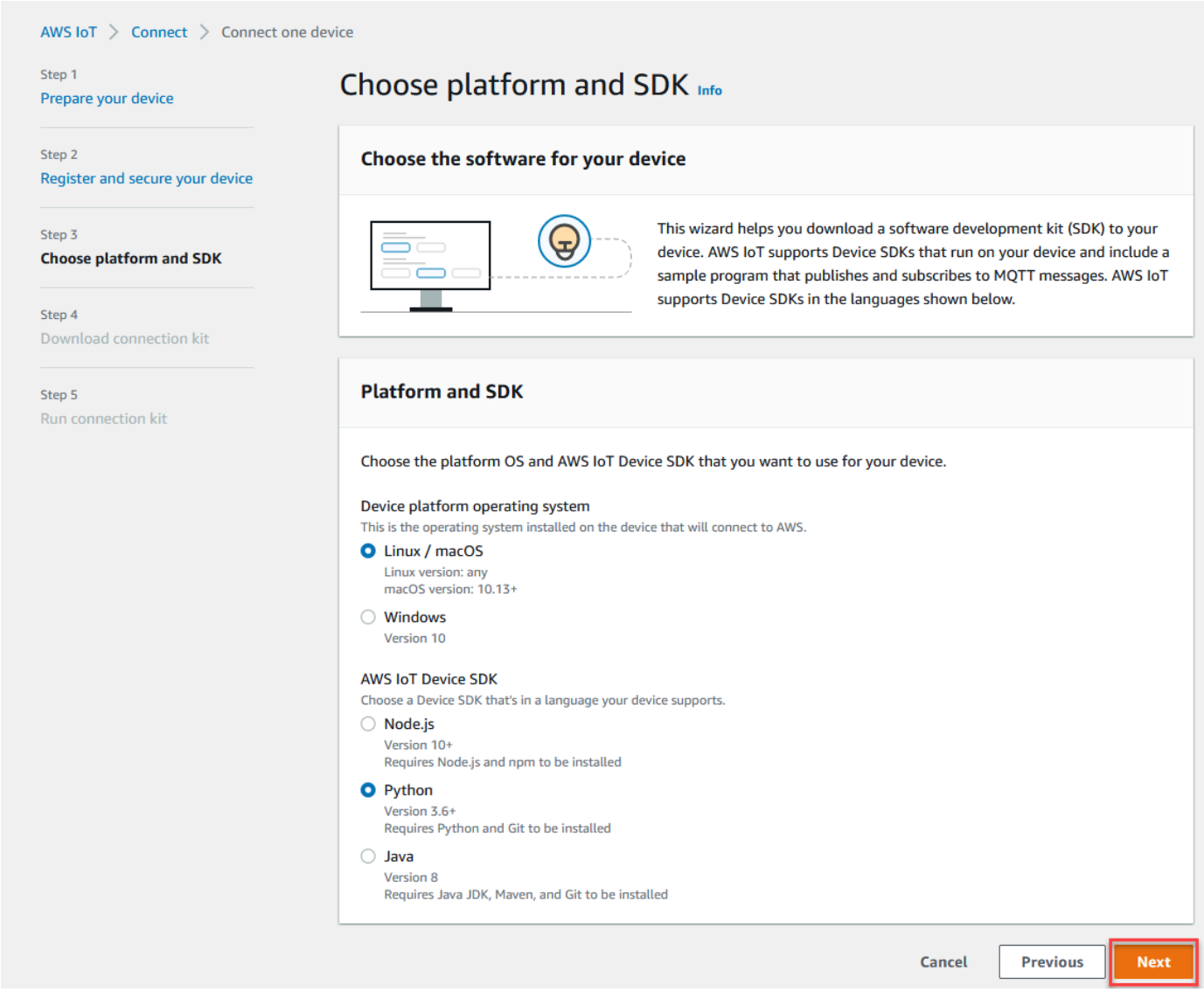
Nachdem Sie Ihrem Objekt einen Namen gegeben und weitere Konfigurationen ausgewählt haben, wählen Sie Weiter.



3. Wählen Sie im Abschnitt Plattform und SDK auswählen die Plattform und die Sprache des AWS IoT Geräte-SDK aus, das Sie verwenden möchten. In diesem Beispiel werden die Linux/OSX Plattform und das Python-SDK verwendet. Stellen Sie sicher, dass Sie Python3 und Pip3 auf Ihrem Zielgerät installiert haben, bevor Sie mit dem nächsten Schritt fortfahren.

> **Note**
>
> Schauen Sie sich am Ende der Konsolenseite unbedingt die Liste der erforderlichen Software an, die für das von Ihnen gewählte SDK erforderlich ist.
> Sie müssen die erforderliche Software auf Ihrem Zielcomputer installiert haben, bevor Sie mit dem nächsten Schritt fortfahren können.

Nachdem Sie die Plattform und die SDK-Sprache für das Gerät ausgewählt haben, wählen Sie Weiter.

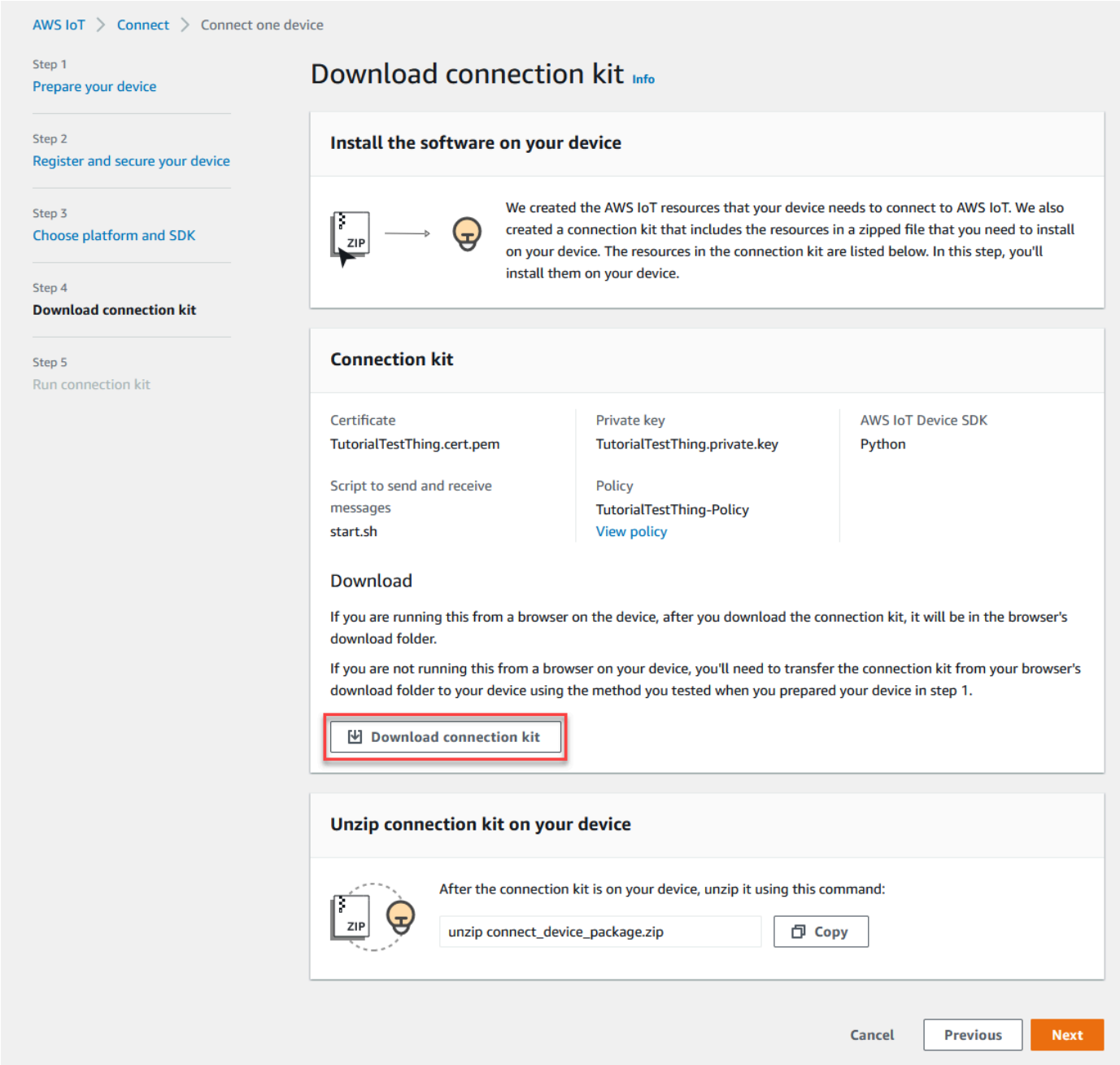


Schritt 3. Laden Sie Dateien auf Ihr Gerät herunter

Diese Seite wird angezeigt, nachdem das Verbindungskit erstellt AWS IoT wurde, das die folgenden Dateien und Ressourcen enthält, die Ihr Gerät benötigt:

- Die Zertifikatsdateien des Objekts, die zur Authentifizierung des Geräts verwendet werden
- Eine Richtlinienressource, mit der Ihr Objekt zur Interaktion mit AWS IoT autorisiert wird
- Das Skript zum Herunterladen des AWS Geräte-SDK und zum Ausführen des Beispielprogramms auf Ihrem Gerät

1. Wenn Sie bereit sind, fortzufahren, klicken Sie auf die Schaltfläche Verbindungskit herunterladen für, um das Verbindungskit für die zuvor gewählte Plattform herunterzuladen.



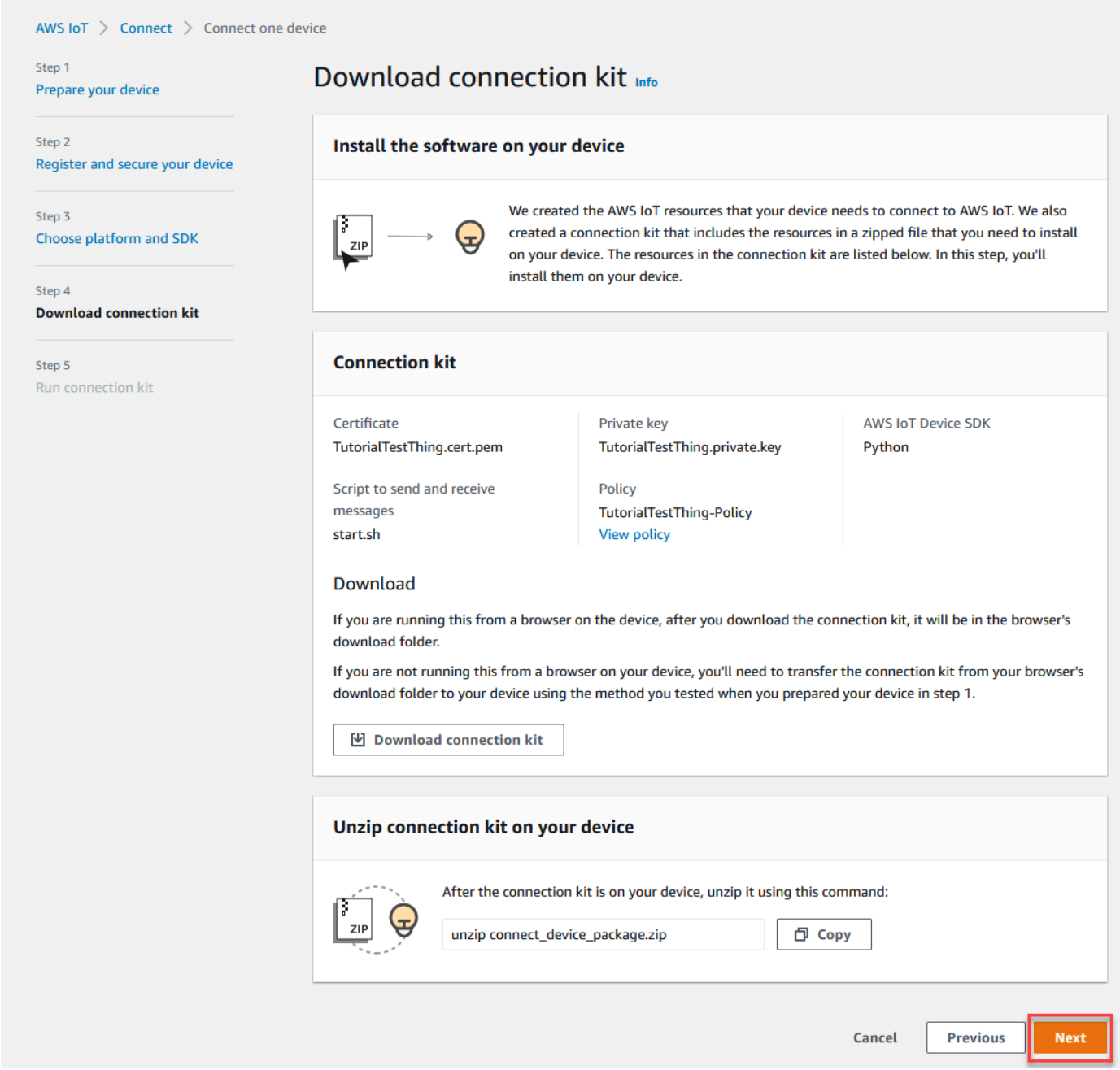
2. Wenn Sie dieses Verfahren auf Ihrem Gerät ausführen, speichern Sie die Verbindungskit-Datei in einem Verzeichnis, von dem aus Sie Befehlszeilen in der Kommandozeile ausführen können.

   Wenn Sie dieses Verfahren nicht auf Ihrem Gerät ausführen, speichern Sie die Verbindungskit-Datei in einem lokalen Verzeichnis und übertragen Sie die Datei dann auf Ihr Gerät.

3. Geben Sie im Abschnitt Verbindungskit auf Ihrem Gerät entpacken unzip connect_device_package.zip in das Verzeichnis ein, in dem sich die Verbindungskit-Dateien befinden.

Wenn Sie ein PowerShell Windows-Befehlsfenster verwenden und der unzip Befehl nicht funktioniert, ersetzen Sie ihn durch und versuchen Sie es erneut unzip mit expand-archive der Befehlszeile.

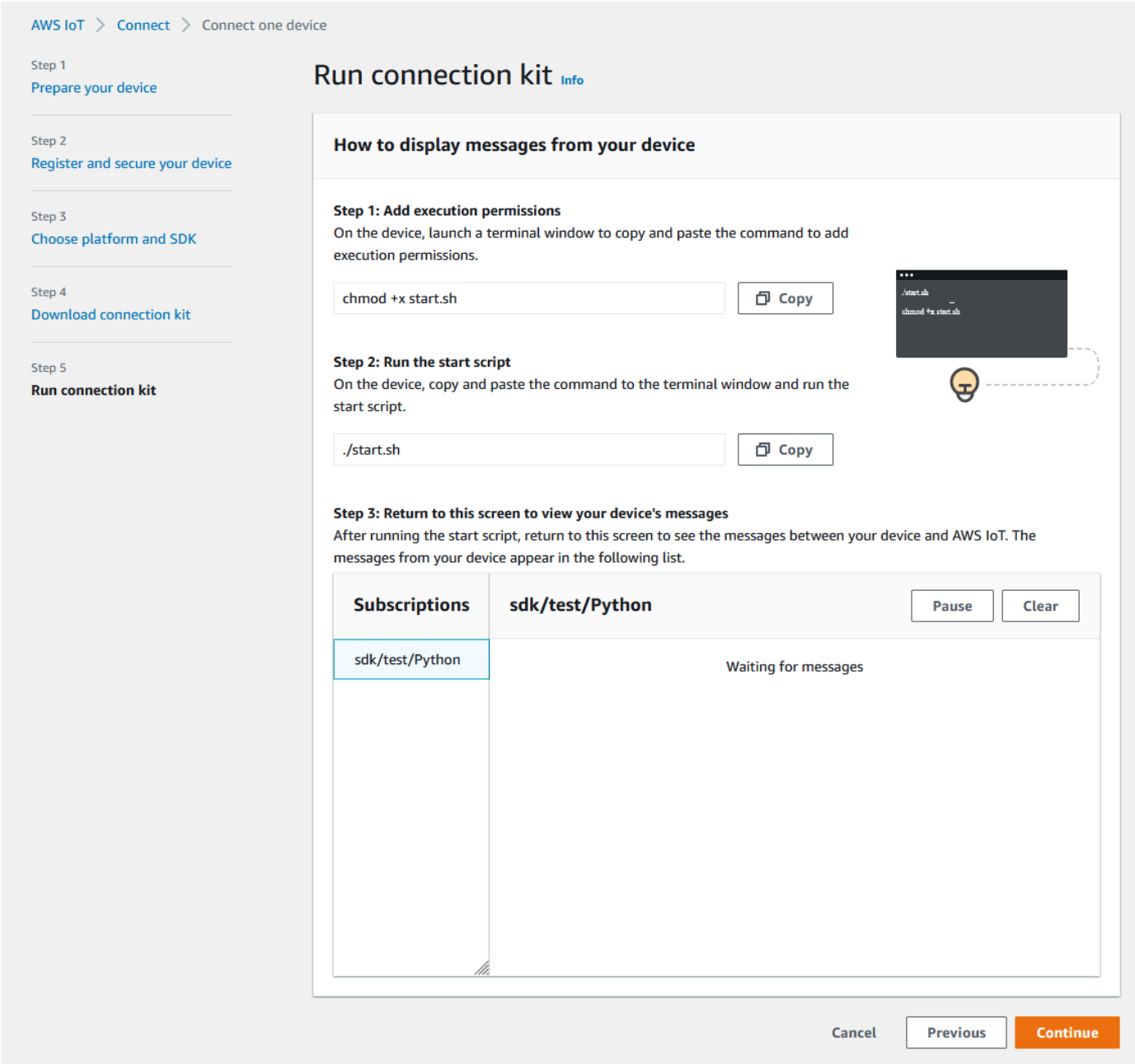
4. Nachdem Sie die Verbindungskit-Datei auf dem Gerät gespeichert haben, setzen Sie das Tutorial fort, indem Sie Weiter wählen.



Schritt 4. Ausführen des Beispiels

Sie führen dieses Verfahren in einem Terminal oder Befehlsfenster auf Ihrem Gerät durch und folgen dabei den Anweisungen in der Konsole. Die Befehle, die Sie in der Konsole sehen, beziehen sich auf das Betriebssystem, für das Sie sich in the section called “Schritt 2. Dies erstellt ein Objekt” entschieden haben. Die hier gezeigten sind für die Linux/OSX Betriebssysteme.

1. Führen Sie in einem Terminal- oder Befehlsfenster auf Ihrem Gerät im Verzeichnis mit der Verbindungskit-Datei die in der AWS IoT Konsole angezeigten Schritte aus.



2. Nachdem Sie den Befehl aus Schritt 2 in der Konsole eingegeben haben, sollte im Terminal oder Befehlsfenster des Geräts eine Ausgabe angezeigt werden, die der folgenden ähnelt. Diese Ausgabe stammt aus den Nachrichten, an die das Programm sendet und denen, die es dann von AWS IoT Core zurückempfängt.

```
Running pub/sub sample application...
Connecting to a13hikvzkye6lx-ats.iot.us-east-1.amazonaws.com with client ID 'basicPubSub'...
Connected!
Subscribing to topic 'sdk/test/Python'...
Subscribed with QoS.AT_LEAST_ONCE
Sending messages until program killed
Publishing message to topic 'sdk/test/Python': Hello World! [1]
Received message from topic 'sdk/test/Python': b'"Hello World! [1]"'
Publishing message to topic 'sdk/test/Python': Hello World! [2]
Received message from topic 'sdk/test/Python': b'"Hello World! [2]"'
Publishing message to topic 'sdk/test/Python': Hello World! [3]
Received message from topic 'sdk/test/Python': b'"Hello World! [3]"'
```

   Während das Beispielprogramm läuft, wird auch die Testnachricht `Hello World!` angezeigt. Die Testnachricht wird im Terminal oder Befehlsfenster auf Ihrem Gerät angezeigt.

   > **Note**
   >
   > Weitere Informationen zum Abonnieren und Veröffentlichen von Themen finden Sie im Beispielcode des von Ihnen ausgewählten SDK.

3. Um das Beispielprogramm erneut auszuführen, können Sie die Befehle aus Schritt 2 in der Konsole dieses Verfahrens wiederholen.

4. (Optional) Wenn Sie die Nachrichten von Ihrem IoT-Client in der AWS IoT Konsole sehen möchten, öffnen Sie den MQTT-Testclient auf der Testseite der AWS IoT Konsole. Wenn Sie Python SDK ausgewählt haben, geben Sie im MQTT-Testclient unter Themenfilter das Thema ein, beispielsweise **`sdk/test/python`**, um die Nachrichten von Ihrem Gerät zu abonnieren. Bei den Themenfiltern wird zwischen Groß- und Kleinschreibung unterschieden und sie hängen von der Programmiersprache des SDK ab, welches Sie in Schritt 1 ausgewählt haben. Weitere Informationen zum Abonnieren und Veröffentlichen von Themen finden Sie im Codebeispiel des von Ihnen ausgewählten SDK.

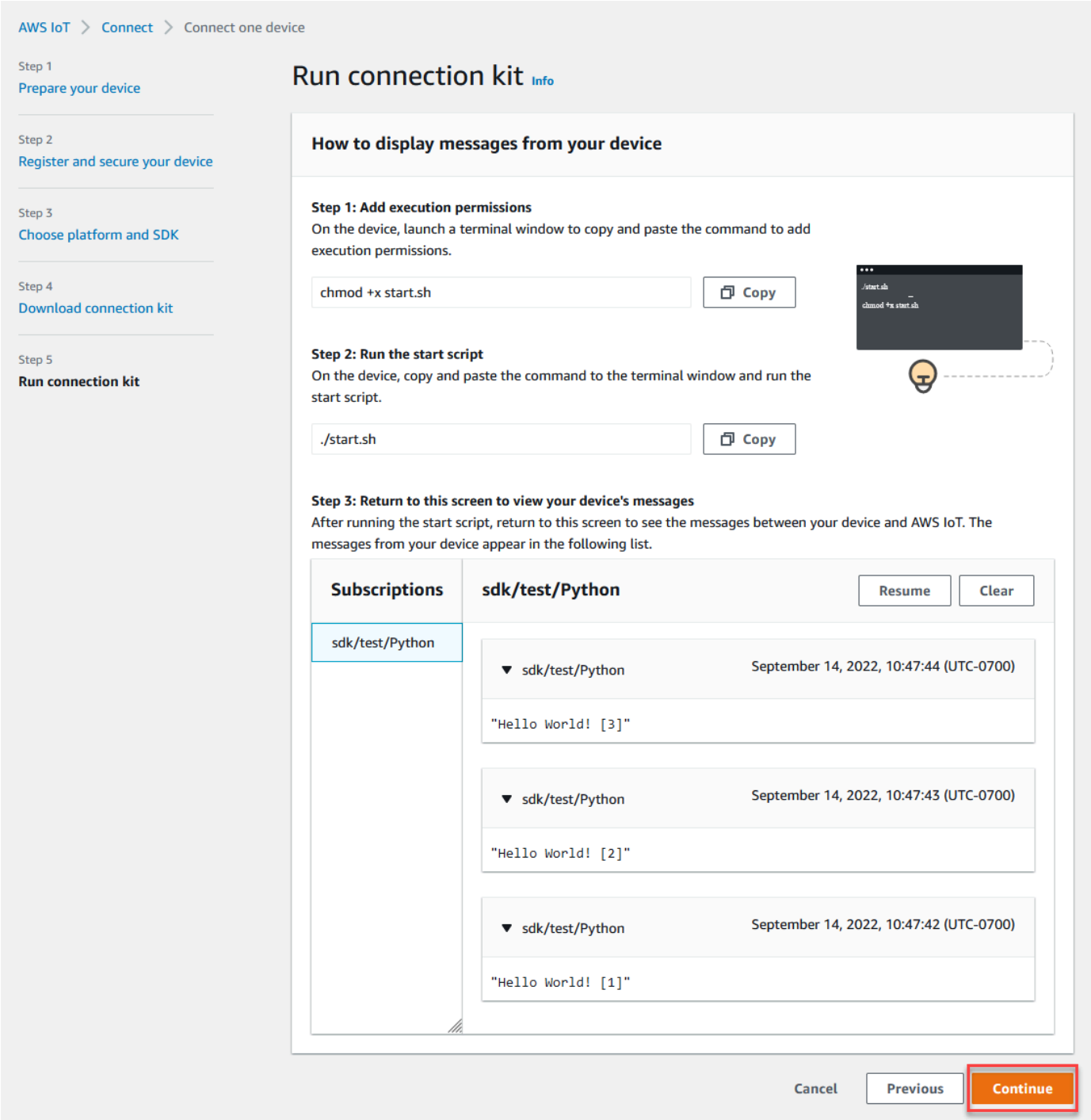
5. Nachdem Sie das Testthema abonniert haben, führen Sie ./start.sh auf Ihrem Gerät aus. Weitere Informationen finden Sie unter the section called “MQTT-Nachrichten anzeigen mit dem AWS IoT MQTT-Client”.

   Nach der Ausführung von ./start.sh werden im MQTT-Client Meldungen angezeigt, die den folgenden ähneln:

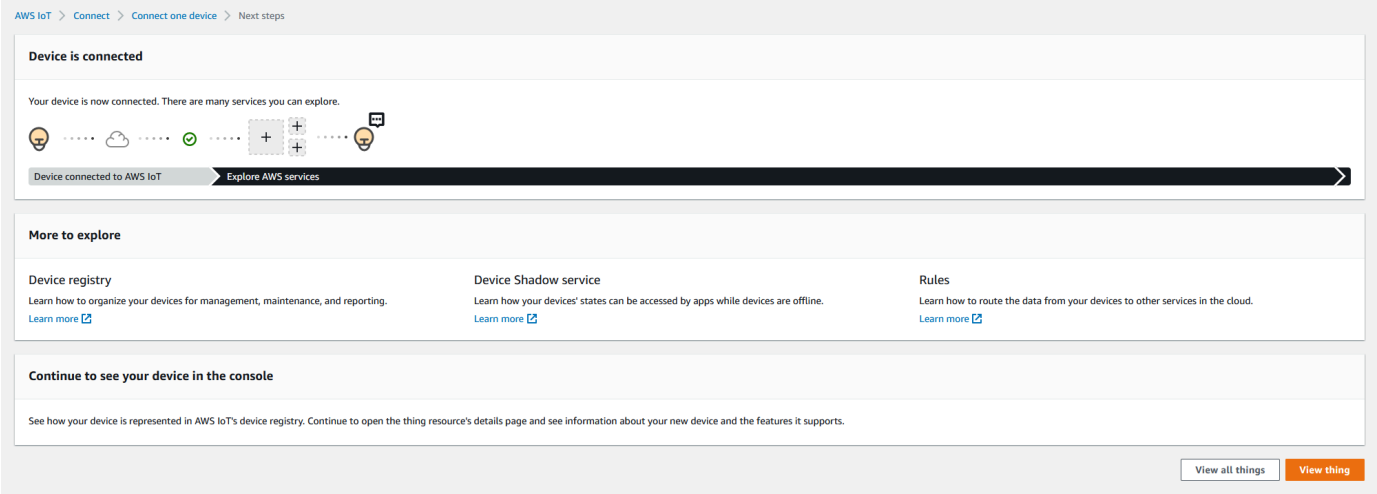
```
{
  "message": "Hello World!" [1]
}
```

Die `sequence` Zahl wird bei jedem Empfang einer neuen `Hello World!` Nachricht um `[]` erhöht und endet, wenn Sie das Programm beenden.

6. Um das Tutorial zu beenden und eine Zusammenfassung zu sehen, wählen Sie in der AWS IoT Konsole Weiter.



7. Eine Zusammenfassung Ihres AWS IoT Quick Connect-Tutorials wird nun angezeigt.



Schritt 5. Weiter erkunden

Im Folgenden finden Sie einige Ideen, die Sie nach Abschluss des Schnellstarts AWS IoT weiter untersuchen sollten.

- So zeigen Sie MQTT-Nachrichten im MQTT-Client an

  Von der AWS IoT Konsole aus können Sie den MQTT-Client auf der Testseite der AWS IoT Konsole öffnen. Abonnieren Sie **#** im MQTT-Testclient und führen Sie es dann auf Ihrem Gerät aus, ./start.sh wie im vorherigen Schritt beschrieben. Weitere Informationen finden Sie unter the section called “MQTT-Nachrichten anzeigen mit dem AWS IoT MQTT-Client”.
- Führen Sie Tests auf Ihren Geräten durch mit Device Advisor

  Testen Sie mit Device Advisor, ob Ihre Geräte sicher und zuverlässig eine Verbindung herstellen und mit ihnen interagieren können AWS IoT.
- the section called “Interaktives Tutorial”

  Um das interaktive Tutorial zu starten, wählen Sie auf der Lernseite der AWS IoT Konsole in der Kachel „So AWS IoT funktioniert“ die Option Tutorial starten aus.
- Machen Sie sich bereit, weitere Tutorials zu entdecken

  Dieser Schnellstart gibt Ihnen nur ein Beispiel für AWS IoT. Wenn Sie mehr darüber erfahren und mehr über die Funktionen erfahren möchten, die sie zu einer leistungsstarken IoT-Lösungsplattform machen, beginnen Sie mit der Vorbereitung Ihrer Entwicklungsplattform durchErkunden AWS IoT Core in praktischen Tutorials. AWS IoT

Testen Sie die Konnektivität mit Ihrem Gerätedatenendpunkt

In diesem Thema wird beschrieben, wie Sie die Verbindung eines Geräts mit dem Gerätedatenendpunkt Ihres Kontos testen, dem Endpunkt, mit dem Ihre IoT-Geräte eine Verbindung zu AWS IoT herstellen.

Führen Sie diese Verfahren auf dem Gerät aus, das Sie testen möchten, oder verwenden Sie eine SSH-Terminalsitzung, die mit dem Gerät verbunden ist, das Sie testen möchten.

Um die Konnektivität eines Geräts mit Ihrem Gerätedatenendpunkt zu testen.

- Finden Sie den Datenendpunkt Ihres Geräts
- Testen Sie die Verbindung schnell
- Laden Sie die App herunter, um die Verbindung zu Ihrem Gerätedatenendpunkt und Port zu testen
- Testen Sie die Verbindung zu Ihrem Gerätedatenendpunkt und Port

Finden Sie den Datenendpunkt Ihres Geräts

In diesem Verfahren wird erklärt, wie Sie Ihren Gerätedatenendpunkt in der AWS IoT Konsole finden, um die Verbindung zu Ihrem IoT-Gerät zu testen.

So finden Sie Ihren Gerätedatenendpunkt

1. Gehen Sie in der AWS IoT Konsole im Bereich Connect zu Domain-Konfigurationen.
2. Gehen Sie auf der Seite Domain-Konfigurationen zum Container Domain-Konfigurationen und kopieren Sie den Domainnamen. Ihr Endpunktwert ist einzigartig für Sie AWS-Konto und ähnelt diesem Beispiel:`a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com`.
3. Gehen Sie wie folgt vor, um Ihren Datenbank-Endpunkt für die Verwendung zu speichern.

Testen Sie die Verbindung schnell

Dieses Verfahren testet die allgemeine Konnektivität mit Ihrem Gerätedatenendpunkt, nicht jedoch den spezifischen Port, den Ihre Geräte verwenden werden. Dieser Test verwendet ein gängiges Programm und reicht normalerweise aus, um herauszufinden, ob Ihre Geräte eine Verbindung mit AWS IoT herstellen können.

Wenn Sie die Konnektivität mit dem spezifischen Port testen möchten, den Ihre Geräte verwenden werden, überspringen Sie dieses Verfahren und fahren Sie mit Laden Sie die App herunter, um die Verbindung zu Ihrem Gerätedatenendpunkt und Port zu testen fort.

Um den Gerätedatenendpunkt schnell zu testen

1. Ersetzen Sie in einem Terminal- oder Befehlszeilenfenster auf Ihrem Gerät den Beispielendpunkt für Gerätedaten (*a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com*) durch den Gerätedatenendpunkt für Ihr Konto und geben Sie dann diesen Befehl ein.

   Linux

   ```
   ping -c 5 a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com
   ```

   Windows

   ```
   ping -n 5 a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com
   ```

2. Wenn von `ping` eine Ausgabe ähnlich der folgenden angezeigt wird, wurde erfolgreich eine Verbindung zu Ihrem Gerätedatenendpunkt hergestellt. Es hat zwar nicht AWS IoT direkt mit dem Server kommuniziert, aber er hat den Server gefunden, und es AWS IoT ist wahrscheinlich, dass er über diesen Endpunkt erreichbar ist.

   ```
   PING a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com (xx.xx.xxx.xxx) 56(84) bytes of
    data.
   64 bytes from ec2-EXAMPLE-218.eu-west-1.compute.amazonaws.com (xx.xx.xxx.xxx):
    icmp_seq=1 ttl=231 time=127 ms
   64 bytes from ec2-EXAMPLE-218.eu-west-1.compute.amazonaws.com (xx.xx.xxx.xxx):
    icmp_seq=2 ttl=231 time=127 ms
   64 bytes from ec2-EXAMPLE-218.eu-west-1.compute.amazonaws.com (xx.xx.xxx.xxx):
    icmp_seq=3 ttl=231 time=127 ms
   64 bytes from ec2-EXAMPLE-218.eu-west-1.compute.amazonaws.com (xx.xx.xxx.xxx):
    icmp_seq=4 ttl=231 time=127 ms
   64 bytes from ec2-EXAMPLE-218.eu-west-1.compute.amazonaws.com (xx.xx.xxx.xxx):
    icmp_seq=5 ttl=231 time=127 ms
   ```

   Wenn Sie mit diesem Ergebnis zufrieden sind, können Sie den Test hier beenden.

Wenn Sie die Konnektivität mit dem spezifischen Port testen möchten, der von AWS IoT verwendet wird, fahren Sie fort mit Laden Sie die App herunter, um die Verbindung zu Ihrem Gerätedatenendpunkt und Port zu testen.

3. Wenn von `ping` keine erfolgreiche Ausgabe zurückgegeben wurde, überprüfen Sie den Endpunktwert, um sicherzustellen, dass Sie den richtigen Endpunkt haben, und überprüfen Sie die Verbindung des Geräts mit dem Internet.

Laden Sie die App herunter, um die Verbindung zu Ihrem Gerätedatenendpunkt und Port zu testen

Ein gründlicherer Konnektivitätstest kann mit `nmap` durchgeführt werden. Mit diesem Verfahren wird getestet, ob `nmap` auf Ihrem Gerät installiert ist.

Um nach **`Nmap auf dem Gerät`** zu suchen

1. Geben Sie in einem Terminal- oder Befehlszeilenfenster auf dem Gerät, das Sie testen möchten, diesen Befehl ein, um zu überprüfen, ob `nmap` installiert ist.

```
nmap --version
```

2. Wenn Sie eine Ausgabe ähnlich der folgenden sehen, ist `nmap` installiert und Sie können mit the section called “Testen Sie die Verbindung zu Ihrem Gerätedatenendpunkt und Port” fortfahren.

```
Nmap version 6.40 ( http://nmap.org )
Platform: x86_64-koji-linux-gnu
Compiled with: nmap-liblua-5.2.2 openssl-1.0.2k libpcre-8.32 libpcap-1.5.3 nmap-
libdnet-1.12 ipv6
Compiled without:
Available nsock engines: epoll poll select
```

3. Wird Ihnen keine ähnliche Antwort angezeigt wie im vorherigen Schritt, müssen Sie die Installation von `nmap` auf dem Gerät ausführen. Wählen Sie das Verfahren für das Betriebssystem Ihres Geräts aus.

Linux

Dieser Vorgang erfordert, dass Sie über die Berechtigung verfügen, Software auf dem Computer zu installieren.

Um nmap auf Ihrem Linux-Computer zu installieren

1. Geben Sie in einem Terminal- oder Befehlszeilenfenster auf Ihrem Gerät den Befehl ein, der der Linux-Version entspricht, die auf dem Gerät ausgeführt wird.

   a. Debian oder Ubuntu:

   ```
   sudo apt install nmap
   ```

   b. Cent OS oder RHEL:

   ```
   sudo yum install nmap
   ```

2. Testen Sie die Installation mit diesem Befehl:

   ```
   nmap --version
   ```

3. Wenn Sie eine Ausgabe ähnlich der folgenden sehen, ist nmap installiert und Sie können mit the section called “Testen Sie die Verbindung zu Ihrem Gerätedatenendpunkt und Port” fortfahren.

   ```
   Nmap version 6.40 ( http://nmap.org )
   Platform: x86_64-koji-linux-gnu
   Compiled with: nmap-liblua-5.2.2 openssl-1.0.2k libpcre-8.32 libpcap-1.5.3 nmap-
   libdnet-1.12 ipv6
   Compiled without:
   Available nsock engines: epoll poll select
   ```

macOS

Dieser Vorgang erfordert, dass Sie über die Berechtigung verfügen, Software auf dem Computer zu installieren.

nmap-Installation auf Ihrem macOS-Computer

1. Öffnen Sie in einem Browser https://nmap.org/download#macosx und laden Sie das neueste stabile Installationsprogramm herunter.

   Wenn Sie dazu aufgefordert werden, wählen Sie Öffnen mit DiskImageInstaller.

2. Verschieben Sie das Paket im Installationsfenster in den Ordner Applications.

3. Suchen Sie im Finder das `nmap-xxxx-mpkg` Paket im Ordner Anwendungen. Ctrl-click Sie auf das Paket und wählen Sie Öffnen, um das Paket zu öffnen.
4. Überprüfen Sie das Sicherheitsdialogfeld. Wenn Sie bereit für die Installation von nmap sind, wählen Sie Öffnen, um nmap zu installieren.
5. Testen Sie in Terminal die Installation mit diesem Befehl.

   ```
   nmap --version
   ```

6. Wenn Sie eine Ausgabe ähnlich der folgenden sehen, ist `nmap` installiert und Sie können mit the section called “Testen Sie die Verbindung zu Ihrem Gerätedatenendpunkt und Port” fortfahren.

   ```
   Nmap version 7.92 ( https://nmap.org )
   Platform: x86_64-apple-darwin17.7.0
   Compiled with: nmap-liblua-5.3.5 openssl-1.1.1k nmap-libssh2-1.9.0 libz-1.2.11
    nmap-libpcre-7.6 nmap-libpcap-1.9.1 nmap-libdnet-1.12 ipv6 Compiled without:
   Available nsock engines: kqueue poll select
   ```

Windows

Dieser Vorgang erfordert, dass Sie über die Berechtigung verfügen, Software auf dem Computer zu installieren.

Um nmap auf Ihrem Windows-Computer zu installieren

1. Öffnen Sie in einem Browser https://nmap.org/download#windows und laden Sie die neueste stabile Version des Setup-Programms herunter.

   Wenn Sie dazu aufgefordert werden, wählen Sie Datei speichern. Nachdem die Datei heruntergeladen wurde, öffnen Sie sie im Download-Ordner.
2. Nachdem der Download der Setup-Datei abgeschlossen ist, öffnen Sie den Download nmap-xxxx-setup.exe um die App zu installieren.
3. Übernehmen Sie bei der Installation des Programms die Standardeinstellungen.

   Sie benötigen die Npcap-App für diesen Test nicht. Sie können diese Option deaktivieren, wenn Sie sie nicht installieren möchten.
4. Testen Sie in Command die Installation mit diesem Befehl.

```
nmap --version
```

5. Wenn Sie eine Ausgabe ähnlich der folgenden sehen, ist nmap installiert und Sie können mit the section called “Testen Sie die Verbindung zu Ihrem Gerätedatenendpunkt und Port” fortfahren.

```
Nmap version 7.92 ( https://nmap.org )
Platform: i686-pc-windows-windows
Compiled with: nmap-liblua-5.3.5 openssl-1.1.1k nmap-libssh2-1.9.0 nmap-
libz-1.2.11 nmap-libpcre-7.6 Npcap-1.50 nmap-libdnet-1.12 ipv6
Compiled without:
Available nsock engines: iocp poll select
```

Testen Sie die Verbindung zu Ihrem Gerätedatenendpunkt und Port

Dieses Verfahren testet die Verbindung Ihres IoT-Geräts mit Ihrem Gerätedatenendpunkt über den ausgewählten Port.

Um den Endpunkt und den Port Ihrer Gerätedaten zu testen

1. Ersetzen Sie in einem Terminal- oder Befehlszeilenfenster auf Ihrem Gerät den Beispielendpunkt für Gerätedaten (*a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com*) durch den Gerätedatenendpunkt für Ihr Konto und geben Sie dann diesen Befehl ein.

```
nmap -p 8443 a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com
```

2. Wenn von nmap eine Ausgabe ähnlich der folgenden angezeigt wird, konnte am ausgewählten Port von nmap erfolgreich eine Verbindung zu Ihrem Gerätedatenendpunkt hergestellt werden.

```
Starting Nmap 7.92 ( https://nmap.org ) at 2022-02-18 16:23 Pacific Standard Time
Nmap scan report for a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com
 (xx.xxx.147.160)
Host is up (0.036s latency).
Other addresses for a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com (not scanned):
 xx.xxx.134.144 xx.xxx.55.139 xx.xxx.110.235 xx.xxx.174.233 xx.xxx.74.65
 xx.xxx.122.179 xx.xxx.127.126
rDNS record for xx.xxx.147.160: ec2-EXAMPLE-160.eu-west-1.compute.amazonaws.com

PORT     STATE SERVICE
```

```
8443/tcp open  https-alt
MAC Address: 00:11:22:33:44:55 (Cimsys)

Nmap done: 1 IP address (1 host up) scanned in 0.91 seconds
```

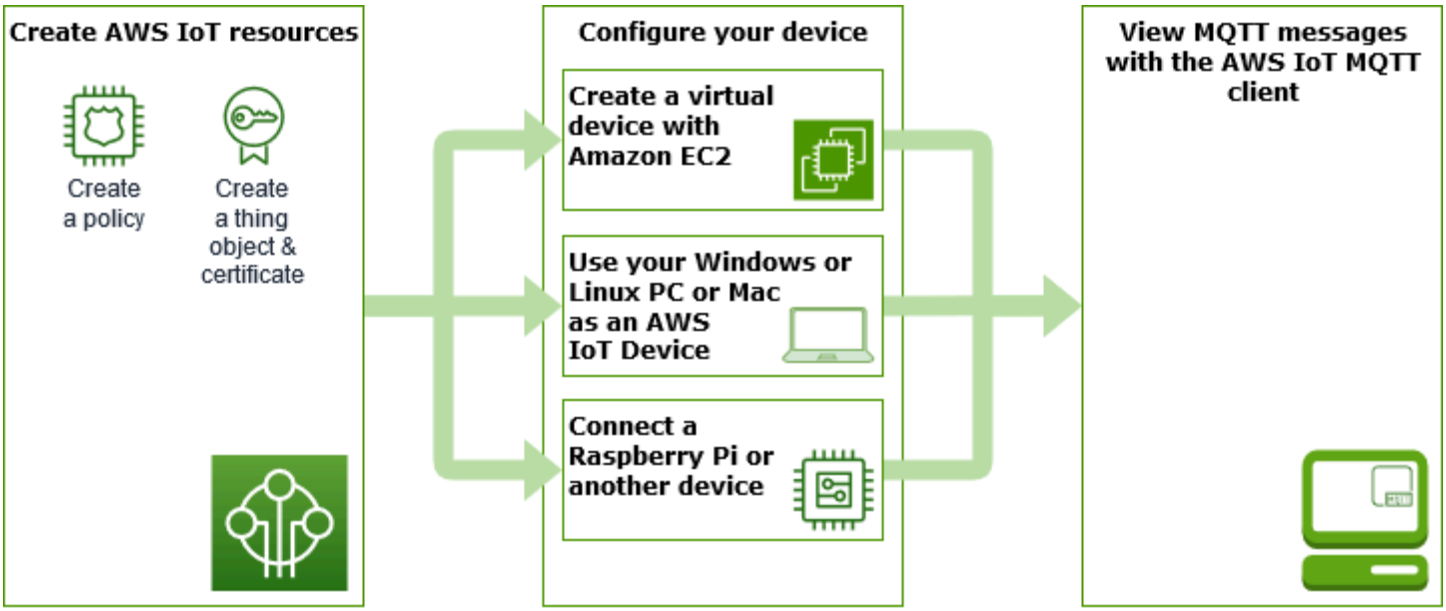
3. Wenn von nmap keine erfolgreiche Ausgabe zurückgegeben wurde, überprüfen Sie den Endpunktwert, um sicherzustellen, dass Sie den richtigen Endpunkt haben, und überprüfen Sie die Verbindung Ihres Geräts mit dem Internet.

Sie können andere Anschlüsse an Ihrem Gerätedatenendpunkt testen, z. B. Port 443, den primären HTTPS-Port, indem Sie den in Schritt 1 verwendeten Port durch den Port **8443** ersetzen, den Sie testen möchten.

Erkunden AWS IoT Core in praktischen Tutorials

In diesem Tutorial installieren Sie die Software und erstellen die AWS IoT Ressourcen, die erforderlich sind, um ein Gerät anzuschließen, mit AWS IoT Core dem es MQTT-Nachrichten senden und empfangen kann. AWS IoT Core Sie werden die Nachrichten im MQTT-Client in der AWS IoT Konsole sehen.

Sie können damit rechnen, 20 bis 30 Minuten für dieses Tutorial aufzuwenden. Wenn Sie ein IoT-Gerät oder einen Raspberry Pi verwenden, kann dieses Tutorial länger dauern, wenn Sie beispielsweise das Betriebssystem installieren und das Gerät konfigurieren müssen.



Dieses Tutorial eignet sich am besten für Entwickler, die sich AWS IoT Core zunächst mit fortgeschritteneren Funktionen wie der Regel-Engine und Schatten vertraut machen möchten. Dieses Tutorial bereitet Sie darauf vor, mehr über andere Dienste zu erfahren AWS IoT Core und zu erfahren, wie es mit anderen AWS Diensten interagiert, indem es die Schritte detaillierter erklärt als das Schnellstart-Tutorial. Wenn Sie nur ein schnelles Hello World-Erlebnis suchen, probieren Sie Versuchen Sie es mit AWS IoT Core Schnellverbindungs-Tutorial aus.

Nachdem Sie Ihre AWS-Konto AWS IoT Konsole eingerichtet haben, folgen Sie diesen Schritten, um zu erfahren, wie Sie ein Gerät anschließen und Nachrichten senden AWS IoT Core können.

Nächste Schritte

- Wählen Sie, welche Geräteoption für Sie am besten geeignet ist
- the section called “Create AWS IoT Ressourcen”, wenn Sie kein virtuelles Gerät mit Amazon EC2 erstellen möchten
- the section called “Konfigurieren Ihres Geräts”
- the section called “MQTT-Nachrichten anzeigen mit dem AWS IoT MQTT-Client”

Weitere Informationen zu AWS IoT Core finden Sie unter Was ist AWS IoT Core?

Welche Geräteoption ist für Sie am besten geeignet?

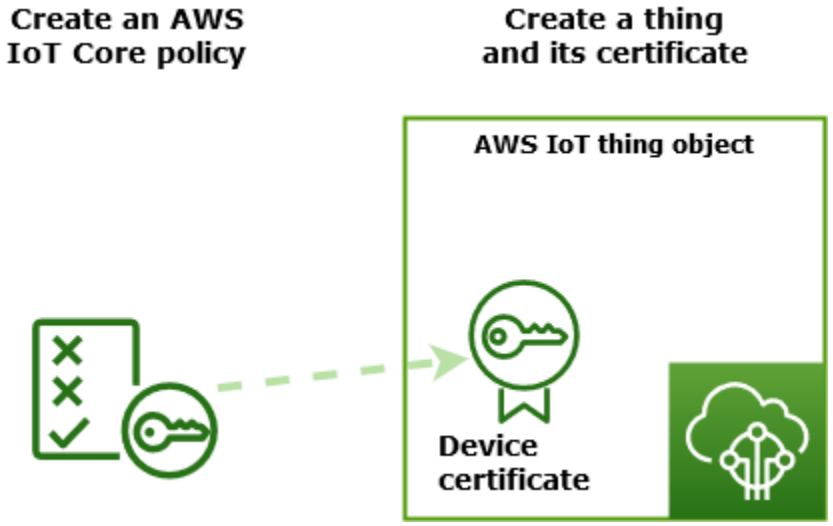
Wenn Sie sich nicht sicher sind, welche Option Sie wählen sollen, können Sie anhand der folgenden Liste der Vor- und Nachteile der einzelnen Optionen entscheiden, welche für Sie am besten geeignet ist.

Option	Dies könnte eine gute Option sein, wenn:	Dies ist möglicherweise keine gute Option, wenn:
the section called “Erstelle n Sie ein virtuelles Gerät mit Amazon EC2”	• Sie kein eigenes Gerät zum Testen haben. • Sie keine Software auf Ihrem eigenen System installieren möchten. • Sie auf einem Linux-Betriebssystem testen möchten.	• Sie mit einer Befehlsze ilenschnittstelle vertraut sind. • Sie nicht möchten, dass zusätzliche AWS Gebühren anfallen.

Option	Dies könnte eine gute Option sein, wenn:	Dies ist möglicherweise keine gute Option, wenn:
		• Sie nicht auf einem Linux-Betriebssystem testen möchten.
the section called “Verwenden Sie Ihren Windows- oder Linux-PC oder Mac als AWS IoT Gerät”	• Sie nicht möchten, dass zusätzliche AWS Gebühren anfallen. • Sie keine zusätzlichen Geräte konfigurieren möchten.	• Sie keine Software auf Ihrem PC installieren möchten. • Sie eine repräsentativere Testplattform möchten.
the section called “Verbinden eines Raspberry Pi oder eines anderes Gerätes”	• Sie möchten AWS IoT mit einem echten Gerät testen. • Sie bereits ein Gerät haben, mit dem Sie testen können. • Sie mit der Integration von Hardware in Systeme Erfahrung haben.	• Sie kein Gerät kaufen oder konfigurieren möchten, nur um es auszuprobieren. • Sie möchten vorerst AWS IoT so einfach wie möglich testen.

Create AWS IoT Ressourcen

In diesem Tutorial erstellen Sie die AWS IoT Ressourcen, die ein Gerät benötigt, um eine Verbindung zu AWS IoT Core und den Austausch von Nachrichten herzustellen.



1. Erstellen Sie ein AWS IoT Richtliniendokument, das Ihr Gerät zur Interaktion mit AWS IoT Diensten autorisiert.
2. Erstellen Sie ein Ding-Objekt AWS IoT und das dazugehörige X.509 Gerätezertifikat und hängen Sie dann das Richtliniendokument an. Das Ding-Objekt ist die virtuelle Darstellung Ihres Geräts in der AWS IoT Registrierung. Das Zertifikat authentifiziert Ihr Gerät gegenüber AWS IoT Core, und das Richtliniendokument autorisiert Ihr Gerät zur Interaktion mit. AWS IoT

> **Note**
>
> Wenn Sie the section called “Erstellen Sie ein virtuelles Gerät mit Amazon EC2” planen, können Sie diese Seite überspringen und mit the section called “Konfigurieren Ihres Geräts”weitermachen. Sie werden diese Ressourcen erstellen, wenn Sie Ihr virtuelles Objekt erstellen.

In diesem Tutorial werden die Ressourcen mithilfe der AWS IoT Konsole erstellt. AWS IoT Wenn Ihr Gerät einen Webbrowser unterstützt, ist es möglicherweise einfacher, dieses Verfahren im Webbrowser des Geräts auszuführen, da Sie die Zertifikatsdateien direkt auf Ihr Gerät herunterladen können. Wenn Sie dieses Verfahren auf einem anderen Computer ausführen, müssen Sie die Zertifikatsdateien auf Ihr Gerät kopieren, bevor sie von der Beispiel-App verwendet werden können.

Erstellen Sie ein AWS IoT policy

Geräte verwenden ein X.509 Zertifikat zur Authentifizierung. AWS IoT Core Mit dem Zertifikat AWS IoT sind Richtlinien verknüpft. Diese Richtlinien legen fest, welche AWS IoT Operationen, wie das

Abonnieren oder Veröffentlichen von MQTT-Themen, das Gerät ausführen darf. Ihr Gerät legt sein Zertifikat vor, wenn es eine Verbindung herstellt und Nachrichten an sendet AWS IoT Core.

Folgen Sie den Schritten, um eine Richtlinie zu erstellen, die es Ihrem Gerät ermöglicht, die zur Ausführung des Beispielprogramms erforderlichen AWS IoT -Vorgänge auszuführen. Sie müssen die AWS IoT -Richtlinie erstellen, bevor Sie sie an das Gerätezertifikat anhängen können, das Sie später erstellen werden.

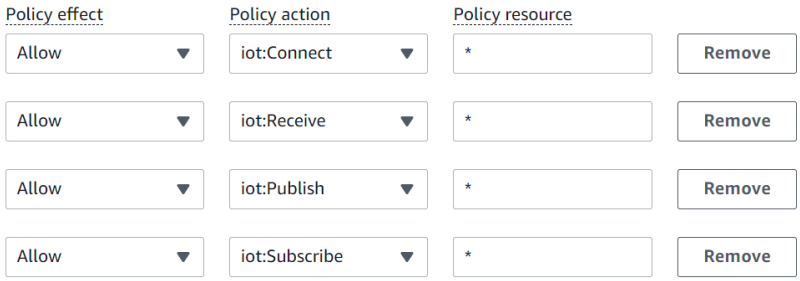
Um ein zu erstellen AWS IoT policy

1. Wählen Sie in der AWS IoT -Konsole im linken Menü Sicherheit und dann Richtlinien aus.
2. Wählen Sie auf der Seite Sie haben noch keine Richtlinie die Option Richtlinie erstellen.

   Wenn Ihr Konto bereits Richtlinien hat, wählen Sie Richtlinie erstellen aus.
3. Auf der Seite Richtlinie erstellen:

   1. Geben Sie im Abschnitt Richtlinieneigenschaften im Feld Richtlinienname einen Namen für die Richtlinie ein (z. B. **`My_Iot_Policy`**). Verwenden Sie keine personenbezogenen Informationen in Ihren Richtliniennamen.
   2. Erstellen Sie im Abschnitt Richtliniendokument die Richtlinienerklärungen, die Ressourcen den Zugriff auf AWS IoT Core -Operationen gewähren oder verweigern. Gehen Sie wie folgt vor, um eine Richtlinienerklärung zu erstellen, die allen Clients die Möglichkeit einräumt, **`iot:Connect`** auszuführen:
      - Wählen Sie im Feld Richtlinieneffekt die Option Zulassen aus. Dadurch können alle Clients, deren Zertifikat diese Richtlinie an ihr Zertifikat angehängt hat, die Aktion ausführen, die im Feld Richtlinienaktion aufgeführt ist.
      - Wählen Sie im Feld Richtlinienaktion eine Richtlinienaktion wie **`iot:Connect`**. Richtlinienaktionen sind die Aktionen, für deren Ausführung Ihr Gerät eine Genehmigung benötigt, wenn es das Beispielprogramm aus dem Geräte-SDK ausführt.
      - Geben Sie im Feld Richtlinienressource einen Amazon-Ressourcennamen (ARN) oder **`*`** ein. Ein **`*`**, um einen beliebigen Client (Gerät) auszuwählen.

      Um die Richtlinienerklärungen für **`iot:Receive`**, **`iot:Publish`** und **`iot:Subscribe`** zu erstellen, wählen Sie Neue Erklärung hinzufügen und wiederholen Sie die Schritte.



> **Note**
>
> In diesem Schnellstart wird der Einfachheit halber der Platzhalter (*) verwendet. Um die Sicherheit zu erhöhen, sollten Sie einschränken, welche Clients (Geräte) Verbindungen herstellen und Nachrichten veröffentlichen können, indem Sie einen Client-ARN anstelle des Platzhalterzeichens als Ressource angeben. Die Client-ARNs weisen das folgende Format auf: `arn:aws:iot:`*`your-region`*`:`*`your-aws-account`*`:client/`*`my-client-id`*.
> Sie müssen jedoch zuerst die Ressource (z. B. ein Client-Gerät oder einen Objektschatten) erstellen, bevor Sie ihren ARN einer Richtlinie zuweisen können. Weitere Informationen finden Sie unter AWS IoT Core -Aktionsressourcen.

4. Nachdem Sie die Informationen für Ihre Richtlinie eingegeben haben, wählen Sie Erstellen.

Weitere Informationen finden Sie unter Wie AWS IoT funktioniert mit IAM.

Dies erstellt ein Objekt

Geräte, mit AWS IoT Core denen eine Verbindung besteht, werden durch Ding-Objekte in der AWS IoT Registrierung dargestellt. Ein Objekt stellt ein bestimmtes Gerät oder eine logische Entität dar. Es kann ein physisches Gerät oder ein Sensor sein (beispielsweise eine Glühbirne oder ein Wandschalter). Es kann sich auch um eine logische Einheit handeln, z. B. eine Instanz einer Anwendung oder eine physische Entität AWS IoT, die keine Verbindung zu anderen Geräten herstellt, die dies tun (z. B. ein Auto mit Motorsensoren oder einem Bedienfeld).

Um ein Ding in der AWS IoT Konsole

1. Wählen Sie in der AWS IoT -Konsole im linken Menü Alle Geräte und dann Objekte aus.
2. Wählen Sie auf der Seite Objekte die Option Objekte erstellen aus.

3. Wählen Sie auf der Seite Objekte erstellen den Eintrag Einzelnes Objekt erstellen und dann Weiter aus.
4. Geben Sie auf der Seite Objekteigenschaften angeben unter Objektname einen Namen für Ihr Objekt ein, z. B. **`MyIotThing`**.

   Wählen Sie die Namen der Objekte sorgfältig aus, da Sie den Namen eines Objekts später nicht mehr ändern können.

   Um den Namen eines Objekts zu ändern, müssen Sie ein neues Objekt erstellen, diesem den neuen Namen geben und dann das alte Objekt löschen.

   > **Note**
   >
   > Verwenden Sie keine personenbezogenen Informationen in Ihren Objektnamen. Der Name der Sache kann in unverschlüsselten Mitteilungen und Berichten vorkommen.

5. Lassen Sie die restlichen Felder auf dieser Seite leer. Wählen Sie Weiter aus.
6. Wählen Sie auf der Seite Gerätezertifikat konfigurieren — optional Auto-generate ein neues Zertifikat aus (empfohlen). Wählen Sie Weiter aus.
7. Wählen Sie auf der Seite Richtlinien an Zertifikat anhängen — optional die Richtlinie aus, die Sie im vorherigen Abschnitt erstellt haben. In diesem Abschnitt erhielt die Richtlinie den Namen **`My_Iot_Policy`**. Wählen Sie Objekt erstellen aus.
8. Gehen Sie auf der Seite Zertifikate und Schlüssel herunterladen wie folgt vor:

   1. Laden Sie die einzelnen Zertifikat- und Schlüsseldateien herunter und speichern Sie sie für später. Sie müssen diese Dateien auf Ihrem Gerät installieren.

      Wenn Sie Ihre Zertifikatsdateien speichern, geben Sie ihnen die Namen in der folgenden Tabelle. Dies sind die Dateinamen, die in späteren Beispielen verwendet wurden.

      Namen der Zertifikatsdateien

Datei	Dateipfad
Privater Schlüssel	`private.pem.key`
Öffentlicher Schlüssel	(in diesen Beispielen nicht verwendet)
Gerätezertifikat	`device.pem.crt`

Datei	Dateipfad
CA-Stammzertifikat	`Amazon-root-CA-1.pem`

2. Um die Root-CA-Datei für diese Dateien herunterzuladen, wählen Sie den Download-Link der Root-CA-Zertifikatsdatei, die dem Typ des Datenendpunkts und der Cipher Suite entspricht, die Sie verwenden. Wählen Sie in diesem Tutorial rechts neben RSA 2048 bit key: Amazon Root CA 1 die Option Herunterladen und laden Sie die Zertifikatsdatei RSA 2048 bit key: Amazon Root CA 1 herunter.

   > **Important**
   >
   > Sie müssen die Zertifikatsdateien speichern, bevor Sie diese Seite verlassen. Nachdem Sie diese Seite in der Konsole verlassen haben, haben Sie keinen Zugriff mehr auf die Zertifikatsdateien.
   > Wenn Sie vergessen haben, die in diesem Schritt erstellten Zertifikatsdateien herunterzuladen, müssen Sie diesen Konsolenbildschirm verlassen, zur Liste der Objekte in der Konsole wechseln, das von Ihnen erstellte Objekt löschen und diesen Vorgang dann von vorne beginnen.

3. Wählen Sie Erledigt aus.

Nachdem Sie dieses Verfahren abgeschlossen haben, sollte das neue Objekt in Ihrer Liste der Objekte angezeigt werden.

Konfigurieren Ihres Geräts

In diesem Abschnitt wird beschrieben, wie Sie Ihr Gerät für die Verbindung mit AWS IoT Core konfigurieren. Wenn Sie damit beginnen möchten, AWS IoT Core aber noch kein Gerät haben, können Sie mithilfe von Amazon EC2 ein virtuelles Gerät erstellen oder Ihren Windows-PC oder Mac als IoT-Gerät verwenden.

Wählen Sie die beste Geräteoption aus, die Sie ausprobieren AWS IoT Core möchten. Natürlich können Sie alle ausprobieren, aber versuchen Sie es jeweils nur mit einer. Wenn Sie sich nicht sicher sind, welche Geräteoption für Sie am besten geeignet ist, lesen Sie, wie Sie die beste Geräteoption auswählen können, und kehren Sie dann zu dieser Seite zurück.

Geräteoptionen

- Erstellen Sie ein virtuelles Gerät mit Amazon EC2
- Verwenden Sie Ihren Windows- oder Linux-PC oder Mac als AWS IoT Gerät
- Verbinden eines Raspberry Pi oder eines anderes Gerätes

Erstellen Sie ein virtuelles Gerät mit Amazon EC2

In diesem Tutorial erstellen Sie eine Amazon EC2-Instance, die als Ihr virtuelles Gerät in der Cloud dient.

Um dieses Tutorial abzuschließen, benötigen Sie eine AWS-Konto. Wenn dies nicht der Fall ist, führen Sie die unter Einrichten AWS-Konto beschriebenen Schritte aus, bevor Sie fortfahren.

In diesem Tutorial führen Sie die folgenden Aktivitäten durch:

- Eine Amazon-EC2-Instance konfigurieren
- Installiere Git Node.js und konfiguriere das AWS CLI
- Create AWS IoT Ressourcen für Ihr virtuelles Gerät
- Installieren Sie das AWS IoT Geräte-SDK für JavaScript
- Ausführen der Beispielanwendungen
- Nachrichten aus der Beispiel-App anzeigen im AWS IoT Konsole

Eine Amazon-EC2-Instance konfigurieren

Die folgenden Schritte zeigen Ihnen, wie Sie eine Amazon EC2-Instance erstellen, die anstelle eines physischen Geräts als Ihr virtuelles Gerät fungiert.

Wenn Sie zum ersten Mal eine Amazon EC2-Instance erstellen, sind die Anweisungen unter Erste Schritte mit Amazon EC2Linux-Instances möglicherweise hilfreicher.

So starten Sie eine Instance

1. Öffnen Sie die Amazon-EC2-Konsole unter https://console.aws.amazon.com/ec2/.
2. Erweitern Sie im Konsolenmenü auf der linken Seite den Abschnitt Instances und wählen Sie Instances aus. Wählen Sie im Instances-Dashboard rechts die Option Instances starten aus, um eine Liste der Basiskonfigurationen anzuzeigen.

3. Geben Sie im Abschnitt Name und Tags einen Namen für die Instance ein und fügen Sie optional Tags hinzu.
4. Wählen Sie im Abschnitt Anwendungs- und Betriebssystem-Images (Amazon Machine Image) eine AMI-Vorlage für Ihre Instance aus, z. B. Amazon Linux 2 AMI (HVM). Diese AMIs sind als „Zur kostenlosen Nutzung berechtigt" gekennzeichnet.
5. Auf der Seite Instance Type können Sie die Hardware-Konfiguration Ihrer Instance auswählen. Wählen Sie den Typ `t2.micro` aus (Standardeinstellung). Beachten Sie, dass dieser Instance-Typ über die Berechtigung für das kostenlose Kontingent verfügt.
6. Wählen Sie im Abschnitt Schlüsselpaar (Anmeldung) einen Schlüsselpaar-Namen aus der Dropdown-Liste oder wählen Sie Neues Schlüsselpaar erstellen, um ein neues Schlüsselpaar zu erstellen. Wenn Sie ein neues Schlüsselpaar erstellen, stellen Sie sicher, dass Sie die private Schlüsseldatei herunterladen und an einem sicheren Ort speichern, da dies Ihre einzige Möglichkeit ist, sie herunterzuladen und zu speichern. Sie müssen den Namen für Ihr Schlüsselpaar beim Starten einer Instance angeben. Der entsprechende private Schlüssel muss jedes Mal angegeben werden, wenn Sie eine Verbindung mit der Instance herstellen.

   > ⚠ Warning
   >
   > Wählen Sie nicht die Option Ohne Schlüsselpaar fortfahren aus. Wenn Sie Ihre Instance ohne Schlüsselpaar starten, können Sie keine Verbindung zu ihr herstellen.

7. In den Abschnitten Netzwerkeinstellungen und Speicher konfigurieren können Sie die Standardeinstellungen beibehalten. Sobald Sie bereit sind, wählen Sie Instance starten aus.
8. Auf einer Bestätigungsseite wird Ihnen mitgeteilt, dass die Instance gestartet wird. Wählen Sie View Instances aus, um die Bestätigungsseite zu schließen und zur Konsole zurückzukehren.
9. Auf dem Bildschirm Instances können Sie den Status des Starts anzeigen. Es dauert einige Zeit, bis die Instance startet. Wenn Sie eine Instance starten, lautet ihr anfänglicher Status `pending`. Nachdem die Instance gestartet wurde, ändert sich der Status in `running`. Sie erhält dann einen öffentlichen DNS-Namen. (Wenn die Spalte Public DNS (IPv4) ausgeblendet ist, wählen Sie Show/Hide Spalten (das zahnradförmige Symbol) in der oberen rechten Ecke der Seite und dann Public DNS (IPv4) aus.)
10. Es kann einige Minuten dauern, bis die Instance für die Verbindungsherstellung bereit ist. Prüfen Sie, ob die Instance die Statusprüfungen bestanden hat. Sie finden diese Information in der Spalte Status Checks.

Nachdem Ihre neue Instance die Statusprüfungen bestanden hat, fahren Sie mit dem nächsten Verfahren fort und stellen Sie eine Verbindung zu ihr her.

So stellen Sie eine Verbindung zu Ihrer Instance her

Sie können über die Amazon EC2-Konsole (browserbasierter Client) Verbindungen mit einer Instance herstellen, indem Sie die Instance in der Konsole auswählen und die Verbindung mittels Amazon EC2 Instance Connect festlegen. Instance Connect verarbeitet die Berechtigungen und stellt eine erfolgreiche Verbindung bereit.

1. Öffnen Sie die Amazon-EC2-Konsole unter https://console.aws.amazon.com/ec2/.
2. Wählen Sie im linken Menü Instances aus.
3. Wählen Sie die Instance und Connect (Verbinden) aus.
4. Wählen Sie Amazon EC2 Instance Connect, Connect.

Sie sollten jetzt ein Amazon EC2 Instance Connect-Fenster haben, das bei Ihrer neuen Amazon EC2 Instance angemeldet ist.

Installiere Git Node.js und konfiguriere das AWS CLI

In diesem Abschnitt installieren Sie Git und Node.js auf Ihrer Linux-Instance.

So installieren Sie Git

1. Aktualisieren Sie Ihre Instance in Ihrem Amazon EC2 Instance Connect-Fenster mit dem folgenden Befehl.

   ```
   sudo yum update -y
   ```

2. Installieren Sie Git in Ihrem Amazon EC2 Instance Connect-Fenster mit dem folgenden Befehl.

   ```
   sudo yum install git -y
   ```

3. Führen Sie den folgenden Befehl aus, um zu überprüfen, ob Git installiert wurde und ob es sich um die aktuelle Version von Git handelt:

   ```
   git --version
   ```

Um zu installieren Node.js

1. Installieren Sie in Ihrem Amazon EC2 Instance Connect-Fenster den Node Version Manager (nvm) mit dem folgenden Befehl.

   ```
   curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash
   ```

   Wir werden nvm für die Installation verwenden Node.js , da nvm mehrere Versionen von installieren kann Node.js und Sie zwischen ihnen wechseln können.

2. Aktivieren Sie nvm in Ihrem Amazon EC2 Instance Connect-Fenster mit diesem Befehl.

   ```
   . ~/.nvm/nvm.sh
   ```

3. Verwenden Sie in Ihrem Amazon EC2 Instance Connect-Fenster nvm, um die neueste Version Node.js von mit diesem Befehl zu installieren.

   ```
   nvm install 16
   ```

   > **Note**
   >
   > Dadurch wird die neueste LTS-Version von installiert. Node.js

   Bei der Installation wird Node.js auch der Node Package Manager (npm) installiert, sodass Sie bei Bedarf zusätzliche Module installieren können.

4. Testen Sie in Ihrem Amazon EC2 Instance Connect-Fenster, ob Node.js es installiert ist und ordnungsgemäß ausgeführt wird, indem Sie diesen Befehl verwenden.

   ```
   node -e "console.log('Running Node.js ' + process.version)"
   ```

   Dieses Tutorial erfordert Node v10.0 oder neuer. Weitere Informationen finden Sie unter Tutorial: Einrichtung Node.js auf einer Amazon EC2 EC2-Instance.

Um zu konfigurieren AWS CLI

Auf Ihrer Amazon EC2-Instance ist AWS CLI vorinstalliert. Sie müssen jedoch Ihr AWS CLI Profil vervollständigen. Weitere Informationen zum Konfigurieren Ihrer CLI finden Sie unter Konfiguration von AWS CLI.

1. Das folgende Beispiel zeigt Beispielwerte. Ersetzen Sie sie mit Ihren eigenen Werten. Sie finden diese Werte in Ihrer AWS -Konsole in Ihren Kontoinformationen unter Sicherheitsanmeldedaten.

   Geben Sie in Ihrem Amazon EC2 Instance Connect-Fenster diesen Befehl ein:

   ```
   aws configure
   ```

   Geben Sie dann an den angezeigten Eingabeaufforderungen die Werte aus Ihrem Konto ein.

   ```
   AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
   AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   Default region name [None]: us-west-2
   Default output format [None]: json
   ```

2. Sie können Ihre AWS CLI Konfiguration mit diesem Befehl testen:

   ```
   aws iot describe-endpoint --endpoint-type iot:Data-ATS
   ```

   Wenn Ihr korrekt konfiguriert AWS CLI ist, sollte der Befehl eine Endpunktadresse von Ihrem zurückgeben AWS-Konto.

Create AWS IoT Ressourcen für Ihr virtuelles Gerät

In diesem Abschnitt wird beschrieben, wie Sie AWS CLI das Ding-Objekt und seine Zertifikatsdateien direkt auf dem virtuellen Gerät erstellen können. Dies erfolgt direkt auf dem Gerät, um mögliche Komplikationen zu vermeiden, die entstehen könnten, wenn sie von einem anderen Computer auf das Gerät kopiert werden. In diesem Abschnitt erstellen Sie die folgenden Ressourcen für Ihr virtuelles Gerät:

- Ein Ding-Objekt, in dem Ihr virtuelles Gerät dargestellt AWS IoT werden soll.
- Ein Zertifikat zur Authentifizierung Ihres virtuellen Geräts.
- Ein Richtliniendokument, mit dem Sie Ihr virtuelles Gerät autorisieren, eine Verbindung zu AWS IoT herzustellen und Nachrichten zu veröffentlichen, zu empfangen und zu abonnieren.

Um ein zu erstellen AWS IoT Ding-Objekt in Ihrer Linux-Instanz

Geräte, mit AWS IoT denen verbunden ist, werden durch Ding-Objekte in der AWS IoT Registrierung dargestellt. Ein Objekt stellt ein bestimmtes Gerät oder eine logische Entität dar. In diesem Fall repräsentiert Ihr Objekt Ihr virtuelles Gerät, diese Amazon EC2-Instance.

1. Führen Sie in Ihrem Amazon EC2 Instance Connect-Fenster den folgenden Befehl aus, um Ihr Objekt zu erstellen.

   ```
   aws iot create-thing --thing-name "MyIotThing"
   ```

2. Die JSON-Antwort sollte wie folgt aussehen:

   ```
   {
       "thingArn": "arn:aws:iot:your-region:your-aws-account:thing/MyIotThing",
       "thingName": "MyIotThing",
       "thingId": "6cf922a8-d8ea-4136-f3401EXAMPLE"
   }
   ```

Zum Erstellen und Anhängen AWS IoT Schlüssel und Zertifikate in Ihrer Linux-Instanz

Der Befehl create-keys-and-certificate erstellt Clientzertifikate, die von der Amazon Root-Zertifizierungsstelle signiert wurden. Dieses Zertifikat wird verwendet, um die Identität Ihres virtuellen Geräts zu authentifizieren.

1. Erstellen Sie in Ihrem Amazon EC2 Instance Connect-Fenster ein Verzeichnis zum Speichern Ihres Zertifikats und Ihrer Schlüsseldateien.

   ```
   mkdir ~/certs
   ```

2. Laden Sie in Ihrem Amazon EC2 Instance Connect-Fenster mit diesem Befehl eine Kopie des Zertifikats der Amazon Certificate Authority (CA) herunter.

   ```
   curl -o ~/certs/Amazon-root-CA-1.pem \
       https://www.amazontrust.com/repository/AmazonRootCA1.pem
   ```

3. Führen Sie in Ihrem Amazon EC2 Instance Connect-Fenster den folgenden Befehl aus, um Ihre privaten Schlüssel-, öffentlichen Schlüssel- und X.509 Zertifikatsdateien zu erstellen. Dieser Befehl registriert und aktiviert auch das Zertifikat mit AWS IoT.

```
aws iot create-keys-and-certificate \
    --set-as-active \
    --certificate-pem-outfile "~/certs/device.pem.crt" \
    --public-key-outfile "~/certs/public.pem.key" \
    --private-key-outfile "~/certs/private.pem.key"
```

Die Antwort sieht wie folgt aus. Speichern Sie das `certificateArn`, damit Sie es in nachfolgenden Befehlen verwenden können. Sie benötigen es, um Ihr Zertifikat an Ihr Objekt anzuhängen und die Richtlinie in späteren Schritten an das Zertifikat anzuhängen.

```
{
    "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/9894ba17925e663f1d29c23af4582b8e3b7619c31f3fbd93adcb51ae54b83dc2",
    "certificateId":
 "9894ba17925e663f1d29c23af4582b8e3b7619c31f3fbd93adcb51ae54b83dc2",
    "certificatePem": "
-----BEGIN CERTIFICATE-----
MIICiTCCEXAMPLE6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMC
VVMxCzAJBgNVBAgEXAMPLEAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6
b24xFDASBgNVBAsTC0lBTSEXAMPLE2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAd
BgkqhkiG9w0BCQEWEG5vb25lQGFtYEXAMPLEb20wHhcNMTEwNDI1MjA0NTIxWhcN
MTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCEXAMPLEJBgNVBAgTAldBMRAwDgYD
VQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDAEXAMPLEsTC0lBTSBDb25z
b2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEXAMPLE25lQGFt
YXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMaK0dn+aEXAMPLE
EXAMPLEfEvySWtC2XADZ4nB+BLYgVIk60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9T
rDHudUZEXAMPLELG5M43q7Wgc/MbQITxOUSQv7c7ugFFDzQGBzZswY6786m86gpE
Ibb3OhjZnzcvQAEXAMPLEWIMm2nrAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4
nUhVVxYUntneD9+h8Mg9qEXAMPLEyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0Fkb
FFBjvSfpJIlJ00zbhNYS5f6GuoEDEXAMPLEBHjJnyp378OD8uTs7fLvjx79LjSTb
NYiytVbZPQUQ5Yaxu2jXnimvw3rrszlaEXAMPLE=
-----END CERTIFICATE-----\n",
    "keyPair": {
        "PublicKey": "-----BEGIN PUBLIC
 KEY-----\nMIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h
\nMMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/
gHr99VEEXAMPLE5VF13\n59VK7cEXAMPLE67GK+y+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQWE
\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa
\nhJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\nFQIDAQAB\n-----
END PUBLIC KEY-----\n",
```

```
        "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\nkey omitted for security
 reasons\n-----END RSA PRIVATE KEY-----\n"
    }
}
```

4. Hängen Sie in Ihrem Amazon EC2 Instance Connect-Fenster Ihr Ding-Objekt mit dem Zertifikat an, das Sie gerade erstellt haben, indem Sie den *certificateArn* folgenden Befehl und die Antwort des vorherigen Befehls verwenden.

   ```
   aws iot attach-thing-principal \
       --thing-name "MyIotThing" \
       --principal "certificateArn"
   ```

   Dieser Befehl gibt keine Ausgabe zurück, wenn er erfolgreich ist.

Erstellen und Anfügen einer Richtlinie

1. Erstellen Sie in Ihrem Amazon EC2 Instance Connect-Fenster die Richtliniendatei, indem Sie dieses Richtliniendokument kopieren und in eine Datei mit dem Namen **~/policy.json** einfügen.

   Wenn Sie keinen bevorzugten Linux-Editor haben, können Sie nano mit diesem Befehl öffnen.

   ```
   nano ~/policy.json
   ```

   Fügen Sie das Richtliniendokument für `policy.json` ein. Speichern Sie die Datei und beenden Sie den nano-Editor (Strg+X).

   Kopieren Sie den Inhalt des Richtliniendokuments für `policy.json`.

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "iot:Publish",
                   "iot:Subscribe",
                   "iot:Receive",
   ```

```
                "iot:Connect"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

2. Erstellen Sie in Ihrem Amazon EC2 Instance Connect-Fenster Ihre Richtlinie mithilfe des folgenden Befehls.

```
aws iot create-policy \
    --policy-name "MyIotThingPolicy" \
    --policy-document "file://~/policy.json"
```

Ausgabe:

```
{
    "policyName": "MyIotThingPolicy",
    "policyArn": "arn:aws:iot:us-east-1:your-aws-account:policy/
MyIotThingPolicy",
    "policyDocument": "{
        \"Version\": \"2012-10-17\",
        \"Statement\": [
            {
                \"Effect\": \"Allow\",
                \"Action\": [
                    \"iot:Publish\",
                    \"iot:Receive\",
                    \"iot:Subscribe\",
                    \"iot:Connect\"
                ],
                \"Resource\": [
                    \"*\"
                ]
        }
        ]
    }",
    "policyVersionId": "1"
}
```

3. Fügen Sie in Ihrem Amazon EC2 Instance Connect-Fenster die Richtlinie mithilfe des folgenden Befehls dem Zertifikat Ihres virtuellen Geräts hinzu.

```
aws iot attach-policy \
    --policy-name "MyIotThingPolicy" \
    --target "certificateArn"
```

   Dieser Befehl gibt keine Ausgabe zurück, wenn er erfolgreich ist.

Installieren Sie das AWS IoT Geräte-SDK für JavaScript

In diesem Abschnitt installieren Sie das AWS IoT Geräte-SDK für JavaScript. Es enthält den Code, mit dem Anwendungen kommunizieren können, AWS IoT sowie die Beispielprogramme. Weitere Informationen finden Sie im AWS IoT Geräte-SDK für das JavaScript GitHub Repository.

Um das zu installieren AWS IoT Geräte-SDK für Ihre JavaScript Linux-Instance

1. Klonen Sie in Ihrem Amazon EC2 Instance Connect-Fenster mit diesem Befehl das AWS IoT Geräte-SDK für das JavaScript Repository in das `aws-iot-device-sdk-js-v2` Verzeichnis Ihres Home-Verzeichnisses.

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-js-v2.git
```

2. Navigieren Sie zum `aws-iot-device-sdk-js-v2`-Verzeichnis, das Sie im vorherigen Schritt erstellt haben.

```
cd aws-iot-device-sdk-js-v2
```

3. Verwenden Sie zum Installieren der SDK npm.

```
npm install
```

Ausführen der Beispielanwendungen

Bei den Befehlen im nächsten Abschnitt wird davon ausgegangen, dass Ihr Schlüssel und Ihr Zertifikatdateien wie in dieser Tabelle gezeigt auf Ihrem virtuellen Gerät gespeichert sind.

Namen der Zertifikatsdateien

Datei	Dateipfad
Privater Schlüssel	`~/certs/private.pem.key`
Gerätezertifikat	`~/certs/device.pem.crt`
CA-Stammzertifikat	`~/certs/Amazon-root-CA-1.pem`

In diesem Abschnitt installieren und führen Sie die `pub-sub.js` Beispiel-App aus, die Sie im `aws-iot-device-sdk-js-v2/samples/node` Verzeichnis des AWS IoT Geräte-SDK für JavaScript finden. Diese App zeigt, wie ein Gerät, Ihre Amazon EC2-Instance, die MQTT-Bibliothek verwendet, um MQTT-Nachrichten zu veröffentlichen und zu abonnieren. Die `pub-sub.js`-Beispiel-App abonniert ein Thema, `topic_1`, veröffentlicht 10 Nachrichten zu diesem Thema und zeigt die Nachrichten so an, wie sie vom Message Broker empfangen wurden.

Um die Beispiel-App zu installieren und auszuführen

1. Navigieren Sie in Ihrem Amazon EC2 Instance Connect-Fenster zu dem Verzeichnis `aws-iot-device-sdk-js-v2/samples/node/pub_sub`, das das SDK erstellt hat, und installieren Sie die Beispiel-App mithilfe dieser Befehle.

   ```
   cd ~/aws-iot-device-sdk-js-v2/samples/node/pub_sub
   npm install
   ```

2. Rufen Sie in Ihrem Amazon EC2 Instance Connect-Fenster mit diesem Befehl AWS IoT von *`your-iot-endpoint`* ab.

   ```
   aws iot describe-endpoint --endpoint-type iot:Data-ATS
   ```

3. Fügen Sie in Ihrem Amazon EC2 Instance Connect-Fenster *`your-iot-endpoint`* wie angegeben den Befehl ein und führen Sie ihn aus.

   ```
   node dist/index.js --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/
   certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint
   ```

Die Beispiel-App:

1. Stellt eine Verbindung AWS IoT Core zu Ihrem Konto her.
2. Das Nachrichtenthema topic_1 abonniert und die Nachrichten anzeigt, die es zu diesem Thema erhält.
3. 10 Nachrichten zum Thema topic_1 veröffentlicht.
4. Ihre Ausgabe sieht ähnlich aus wie:

```
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":1}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":2}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":3}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":4}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":5}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":6}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":7}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":8}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":9}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":10}
```

Wenn Probleme bei der Ausführung der Beispiel-App auftreten, überprüfen Sie the section called “Beheben Sie Probleme mit der Beispielanwendung”.

Sie können den Parameter `--verbosity debug` auch zur Befehlszeile hinzufügen, sodass die Beispiel-App detaillierte Meldungen darüber anzeigt, was sie tut. Diese Informationen bieten Ihnen möglicherweise die Hilfe, die Sie zur Behebung des Problems benötigen.

Nachrichten aus der Beispiel-App anzeigen im AWS IoT Konsole

Mithilfe des MQTT-Testclients in der AWS IoT -Konsole können Sie die Nachrichten der Beispiel-App sehen, während sie den Message Broker durchlaufen.

Um die von der Beispiel-App veröffentlichten MQTT-Nachrichten anzuzeigen

1. Sehen Sie sich MQTT-Nachrichten anzeigen mit dem AWS IoT MQTT-Client an. Auf diese Weise lernen Sie, wie Sie den MQTT-Testclient in der AWS IoT -Konsole verwenden, um MQTT-Nachrichten anzuzeigen, während sie den Message Broker passieren.
2. Öffnen Sie den MQTT-Testclient in der AWS IoT -Konsole.
3. Unter Thema abonnieren, Thema abonnieren, topic_1.
4. Führen Sie in Ihrem Amazon EC2 Instance Connect-Fenster die Beispiel-App erneut aus und sehen Sie sich die Nachrichten im MQTT-Testclient in der AWS IoT -Konsole an.

```
cd ~/aws-iot-device-sdk-js-v2/samples/node/pub_sub
node dist/index.js --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/
certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint
```

Weitere Informationen zu MQTT und zur AWS IoT Core Unterstützung des Protokolls finden Sie unter MQTT.

Verwenden Sie Ihren Windows- oder Linux-PC oder Mac als AWS IoT Gerät

In diesem Tutorial konfigurieren Sie einen PC für die Verwendung mit AWS IoT. Diese Anweisungen unterstützen Windows- und Linux-PCs und Macs. Um dies zu erreichen, müssen Sie einige Software auf Ihrem Computer installieren. Wenn Sie keine Software auf Ihrem Computer installieren möchten, können Sie mit Erstellen Sie ein virtuelles Gerät mit Amazon EC2 versuchen, die gesamte Software auf einer virtuellen Maschine zu installieren.

In diesem Tutorial führen Sie die folgenden Aktivitäten durch:

- Einrichten Ihres PCs
- Installiere Git, Python und die AWS IoT Geräte-SDK für Python
- Konfigurieren Sie die Richtlinie und führen Sie die Beispielanwendung aus
- Nachrichten aus der Beispiel-App anzeigen im AWS IoT Konsole
- Führen Sie das Beispiel „Shared Subscription“ in Python aus

Einrichten Ihres PCs

Um dieses Tutorial abzuschließen, benötigen Sie einen Windows- oder Linux-PC oder einen Mac mit Internetverbindung.

Bevor Sie mit dem nächsten Schritt fortfahren, stellen Sie sicher, dass Sie ein Befehlszeilenfenster auf Ihrem Computer öffnen können. cmd.exe auf einem Windows-PC verwenden. Verwenden Sie auf einem Linux-PC oder Mac Terminal.

Installiere Git, Python und die AWS IoT Geräte-SDK für Python

In diesem Abschnitt installieren Sie Python und das AWS IoT Device SDK für Python auf Ihrem Computer.

Installieren Sie die neueste Version von Git und Python

In diesem Verfahren wird erklärt, wie Sie die neueste Version von Git und Python auf Ihrem PC installieren.

Um Git und Python herunterzuladen und auf Ihrem Computer zu installieren

1. Überprüfen Sie, ob Git auf Ihrem Computer installiert ist. Geben Sie an der Befehlszeile den folgenden Befehl ein.

   ```
   git --version
   ```

   Wird mit dem Befehl die Git-Version angezeigt, ist Git installiert und Sie können mit dem nächsten Schritt fortfahren.

   Wenn der Befehl einen Fehler anzeigt, öffnen Sie https://git-scm.com/download und installieren Sie Git für Ihren Computer.
2. Überprüfen Sie, ob Python bereits installiert ist. Geben Sie in der Kommandozeile den Befehl ein.

   ```
   python -V
   ```

   > **Note**
   >
   > Wenn dieser Befehl einen Fehler: `Python was not found` ausgibt, liegt das möglicherweise daran, dass Ihr Betriebssystem die ausführbare Python v3.x-Datei als `Python3` aufruft. Ersetzen Sie in diesem Fall alle Instanzen von `python` durch `python3` und fahren Sie mit dem Rest dieses Tutorials fort.

Wenn der Befehl die Python-Version anzeigt, ist Python bereits installiert. Dieses Skript erfordert Python 3.7 oder höher.

3. Wenn Python installiert ist, können Sie die restlichen Schritte in diesem Abschnitt überspringen. Wenn nicht, fahren Sie fort.
4. Öffnen Sie das Installationsprogramm für Ihren Computer https://www.python.org/downloads/und laden Sie es herunter.
5. Wenn der Download nicht automatisch mit der Installation gestartet wurde, führen Sie das heruntergeladene Programm aus, um Python zu installieren.
6. Überprüfen Sie die Installation von Python.

```
python -V
```

Vergewissern Sie sich, dass der Befehl die Python-Version anzeigt. Wenn die Python-Version nicht angezeigt wird, versuchen Sie erneut, Python herunterzuladen und zu installieren.

Installieren Sie das AWS IoT Geräte-SDK für Python

Um das zu installieren AWS IoT Geräte-SDK für Python auf Ihrem Computer

1. Installieren Sie Version 2 des AWS IoT Geräte-SDK für Python.

```
python3 -m pip install awsiotsdk
```

2. Klonen Sie das AWS IoT Device SDK for Python-Repository in das Verzeichnis aws-iot-device-sdk-python-v2 Ihres Home-Verzeichnisses. Dieses Verfahren bezieht sich auf das Basisverzeichnis für die Dateien, unter denen Sie die Installation durchführen. *home*

   Der tatsächliche Speicherort des *home* Verzeichnisses hängt von Ihrem Betriebssystem ab.

   Linux/macOS

   In macOS und Linux ist das *home* Verzeichnis~.

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
```

Windows

In Windows können Sie den *home* Verzeichnispfad finden, indem Sie diesen Befehl im `cmd` Fenster ausführen.

```
echo %USERPROFILE%
cd %USERPROFILE%
git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
```

> **Note**
>
> Wenn Sie Windows PowerShell anstelle von verwendencmd.exe, verwenden Sie den folgenden Befehl.
>
> ```
> echo $home
> ```

Weitere Informationen finden Sie im GitHub Repository AWS IoT Device SDK for Python.

Vorbereiten der Ausführung von Beispielanwendungen

Vorbereiten der Ausführung von Beispielanwendungen

- Erstellen des `certs` Verzeichnisses. Kopieren Sie die Dateien für den privaten Schlüssel, das Gerätezertifikat und das Stammzertifikat der Zertifizierungsstelle, die Sie bei der Erstellung und Registrierung des Objekts gespeichert haben, in the section called “Create AWS IoT Ressourcen” des `certs` Verzeichnisses. Die Dateinamen der einzelnen Dateien im Zielverzeichnis sollten mit denen in der Tabelle übereinstimmen.

  Bei den Befehlen im nächsten Abschnitt wird davon ausgegangen, dass Ihr Schlüssel und Ihre Zertifikatdateien wie in dieser Tabelle gezeigt auf dem Gerät gespeichert sind.

  Linux/macOS

  Führen Sie diesen Befehl aus, um das Unterverzeichnis `certs` zu erstellen, das Sie beim Ausführen der Beispielanwendungen verwenden werden.

```
mkdir ~/certs
```

Kopieren Sie die Dateien in das neue Unterverzeichnis in die Zieldateipfade, die in der folgenden Tabelle aufgeführt sind.

Namen der Zertifikatsdateien

Datei	Dateipfad
Privater Schlüssel	`~/certs/private.pem.key`
Gerätezertifikat	`~/certs/device.pem.crt`
CA-Stammzertifikat	`~/certs/Amazon-root-CA-1.pem`

Führen Sie diesen Befehl aus, um die Dateien im Verzeichnis `certs` aufzulisten und sie mit den in der Tabelle aufgeführten Dateien zu vergleichen.

```
ls -l ~/certs
```

Windows

Führen Sie diesen Befehl aus, um das Unterverzeichnis `certs` zu erstellen, das Sie beim Ausführen der Beispielanwendungen verwenden werden.

```
mkdir %USERPROFILE%\certs
```

Kopieren Sie die Dateien in das neue Unterverzeichnis in die Zieldateipfade, die in der folgenden Tabelle aufgeführt sind.

Namen der Zertifikatsdateien

Datei	Dateipfad
Privater Schlüssel	`%USERPROFILE%\certs\private.pem.key`

Datei	Dateipfad
Gerätezertifikat	`%USERPROFILE%\certs\device.pem.crt`
CA-Stammzertifikat	`%USERPROFILE%\certs\Amazon-root-CA-1.pem`

Führen Sie diesen Befehl aus, um die Dateien im Verzeichnis `certs` aufzulisten und sie mit den in der Tabelle aufgeführten Dateien zu vergleichen.

```
dir %USERPROFILE%\certs
```

Konfigurieren Sie die Richtlinie und führen Sie die Beispielanwendung aus

In diesem Abschnitt richten Sie Ihre Richtlinie ein und führen das `pubsub.py` Beispielskript aus, das sich im `aws-iot-device-sdk-python-v2/samples` Verzeichnis von befindet AWS IoT-Geräte-SDK for Python. Dieses Skript zeigt, wie Ihr Gerät die MQTT-Bibliothek verwendet, um MQTT-Nachrichten zu veröffentlichen und zu abonnieren.

Die `pubsub.py` Beispiel-App abonniert ein Thema, `test/topic`, veröffentlicht 10 Nachrichten zu diesem Thema und zeigt die Nachrichten so an, wie sie vom Message Broker empfangen wurden.

Zur Ausführung der Beispielanwendung `pubsub.py` benötigen Sie die folgenden Informationen:

Anwendungsparameterwerte

Parameter	Wo der Wert gefunden werden kann
your-iot-endpoint	1. Wählen Sie in der AWS IoT -Konsole im linken Menü Einstellungen. 2. Auf der Seite Einstellungen wird Ihr Endpunkt im Abschnitt Gerätedatenendpunkt angezeigt.

Der *your-iot-endpoint* Wert hat das Format:*endpoint_id*-ats.iot.*region*.amazonaws.com, zum Beispiel*a3qj468EXAMPLE*-ats.iot.us-west-2.amazonaws.com.

Bevor Sie das Skript ausführen, stellen Sie sicher, dass die Richtlinie für Ihr Objekt die Berechtigungen für das Beispielskript zum Herstellen einer Verbindung, zum Abonnieren, Veröffentlichen und Empfangen vorsieht.

Um das Richtliniendokument für eine Objektressource zu finden und zu überprüfen

1. Suchen Sie in der AWS IoT -Konsole in der Liste Objekte nach der Objektressource, die Ihrem Gerät entspricht.
2. Wählen Sie den Link Name der Objektressource, die für Ihr Gerät steht, um die Seite mit den Objektdetails zu öffnen.
3. Wählen Sie auf der Seite mit den Objektdetails auf der Registerkarte Zertifikate das Zertifikat aus, das an die Objektressource angehängt ist. Die Liste sollte nur ein Zertifikat enthalten. Wenn es mehrere gibt, wählen Sie das Zertifikat aus, dessen Dateien auf Ihrem Gerät installiert sind und mit dem eine Verbindung zu AWS IoT Core hergestellt werden soll.

   Wählen Sie auf der Seite mit den Zertifikatsdetails auf der Registerkarte Richtlinien die Richtlinie aus, die mit dem Zertifikat verknüpft ist. Es sollte nur eine geben. Wenn es mehrere gibt, wiederholen Sie den nächsten Schritt für alle, um sicherzustellen, dass mindestens eine Richtlinie den erforderlichen Zugriff gewährt.
4. Suchen Sie auf der Seite mit der Richtlinienübersicht den JSON-Editor und wählen Sie Richtliniendokument bearbeiten aus, um das Richtliniendokument nach Bedarf zu überprüfen und zu bearbeiten.
5. Das Richtlinien-JSON wird im folgenden Beispiel angezeigt. Ersetzen Sie das `"Resource"` Element *region:account* durch Ihr AWS-Region und AWS-Konto in jedem der `Resource` Werte.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
```

```
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topic/test/topic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topicfilter/test/topic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:client/test-*"
            ]
        }
    ]
}
```

Linux/macOS

Um das Beispielskript auszuführen Linux/macOS

1. Navigieren Sie in Ihrem Befehlszeilenfenster zu dem `~/aws-iot-device-sdk-python-v2/samples/node/pub_sub`-Verzeichnis, das das SDK mithilfe dieser Befehle erstellt hat.

   ```
   cd ~/aws-iot-device-sdk-python-v2/samples
   ```

2. Ersetzen Sie in Ihrem Befehlszeilenfenster *`your-iot-endpoint`* wie angegeben und führen Sie diesen Befehl aus.

   ```
   python3 pubsub.py --endpoint your-iot-endpoint --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/certs/device.pem.crt --key ~/certs/private.pem.key
   ```

Windows

Um die Beispiel-App auf einem Windows-PC auszuführen

1. Navigieren Sie in Ihrem Befehlszeilenfenster zu dem `%USERPROFILE%\aws-iot-device-sdk-python-v2\samples`-Verzeichnis, das das SDK erstellt hat, und installieren Sie die Beispiel-App mithilfe dieser Befehle.

   ```
   cd %USERPROFILE%\aws-iot-device-sdk-python-v2\samples
   ```

2. Ersetzen Sie in Ihrem Befehlszeilenfenster *`your-iot-endpoint`* wie angegeben und führen Sie diesen Befehl aus.

   ```
   python3 pubsub.py --endpoint your-iot-endpoint --ca_file %USERPROFILE%
   \certs\Amazon-root-CA-1.pem --cert %USERPROFILE%\certs\device.pem.crt --key
    %USERPROFILE%\certs\private.pem.key
   ```

Das Beispielskript:

1. Stellt eine Verbindung mit dem AWS IoT Core für Ihr Konto her.
2. Abonniert das Nachrichtenthema und zeigt die Nachrichten an test/topic, die es zu diesem Thema erhält.
3. Veröffentlicht 10 Nachrichten zum Thema, test/topic.
4. Die Ausgabe sieht ähnlich aus wie:

```
Connected!
Subscribing to topic 'test/topic'...
Subscribed with QoS.AT_LEAST_ONCE
Sending 10 message(s)
Publishing message to topic 'test/topic': Hello World! [1]
Received message from topic 'test/topic': b'"Hello World! [1]"'
Publishing message to topic 'test/topic': Hello World! [2]
Received message from topic 'test/topic': b'"Hello World! [2]"'
Publishing message to topic 'test/topic': Hello World! [3]
Received message from topic 'test/topic': b'"Hello World! [3]"'
Publishing message to topic 'test/topic': Hello World! [4]
Received message from topic 'test/topic': b'"Hello World! [4]"'
Publishing message to topic 'test/topic': Hello World! [5]
Received message from topic 'test/topic': b'"Hello World! [5]"'
```

```
Publishing message to topic 'test/topic': Hello World! [6]
Received message from topic 'test/topic': b'"Hello World! [6]"'
Publishing message to topic 'test/topic': Hello World! [7]
Received message from topic 'test/topic': b'"Hello World! [7]"'
Publishing message to topic 'test/topic': Hello World! [8]
Received message from topic 'test/topic': b'"Hello World! [8]"'
Publishing message to topic 'test/topic': Hello World! [9]
Received message from topic 'test/topic': b'"Hello World! [9]"'
Publishing message to topic 'test/topic': Hello World! [10]
Received message from topic 'test/topic': b'"Hello World! [10]"'
10 message(s) received.
Disconnecting...
Disconnected!
```

Wenn Probleme bei der Ausführung der Beispiel-App auftreten, überprüfen Sie the section called “Beheben Sie Probleme mit der Beispielanwendung”.

Sie können den Parameter `--verbosity Debug` auch zur Befehlszeile hinzufügen, sodass die Beispiel-App detaillierte Meldungen darüber anzeigt, was sie tut. Diese Informationen können Ihnen bei der Behebung des Problems helfen.

Nachrichten aus der Beispiel-App anzeigen im AWS IoT Konsole

Mithilfe des MQTT-Testclients in der AWS IoT -Konsole können Sie die Nachrichten der Beispiel-App sehen, während sie den Message Broker durchlaufen.

Um die von der Beispiel-App veröffentlichten MQTT-Nachrichten anzuzeigen

1. Sehen Sie sich MQTT-Nachrichten anzeigen mit dem AWS IoT MQTT-Client an. Auf diese Weise lernen Sie, wie Sie den MQTT-Testclient in der AWS IoT -Konsole verwenden, um MQTT-Nachrichten anzuzeigen, während sie den Message Broker passieren.
2. Öffnen Sie den MQTT-Testclient in der AWS IoT -Konsole.
3. Abonnieren Sie unter Thema abonnieren das Thema, test/topic.
4. Führen Sie in Ihrem Befehlszeilenfenster die Beispiel-App erneut aus und sehen Sie sich die Nachrichten im MQTT-Client in der AWS IoT -Konsole an.

   Linux/macOS

   ```
   cd ~/aws-iot-device-sdk-python-v2/samples
   ```

```
python3 pubsub.py --topic test/topic --ca_file ~/certs/Amazon-root-CA-1.pem --
cert ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-
endpoint
```

Windows

```
cd %USERPROFILE%\aws-iot-device-sdk-python-v2\samples
python3 pubsub.py --topic test/topic --ca_file %USERPROFILE%\certs\Amazon-root-
CA-1.pem --cert %USERPROFILE%\certs\device.pem.crt --key %USERPROFILE%\certs
\private.pem.key --endpoint your-iot-endpoint
```

Weitere Informationen zu MQTT und zur AWS IoT Core Unterstützung des Protokolls finden Sie unter MQTT.

Führen Sie das Beispiel „Shared Subscription“ in Python aus

AWS IoT Core unterstützt Shared Subscriptions sowohl für MQTT 3 als auch für MQTT 5. Gemeinsame Abonnements ermöglichen es mehreren Clients, ein Abonnement für ein Thema gemeinsam zu nutzen, und nur ein Client erhält Nachrichten, die zu diesem Thema veröffentlicht wurden, nach dem Zufallsprinzip. Um gemeinsame Abonnements zu verwenden, abonnieren Clients den Themenfilter eines gemeinsamen Abonnements: `$share/{ShareName}/{TopicFilter}`.

So konfigurieren Sie die Richtlinie und führen das Beispiel für ein geteiltes Abonnement aus

1. Um das Beispiel Gemeinsame Abonnements auszuführen, müssen Sie die Richtlinie Ihres Objekts so einrichten, wie es in MQTT 5 Gemeinsame Abonnements dokumentiert ist.
2. Führen Sie die folgenden Befehle aus, um das Gemeinsame Abonnements-Beispiel auszuführen.

   Linux/macOS

   Um das Beispielskript auszuführen Linux/macOS

   1. Navigieren Sie in Ihrem Befehlszeilenfenster zu dem `~/aws-iot-device-sdk-python-v2/samples`-Verzeichnis, das das SDK mithilfe dieser Befehle erstellt hat.

      ```
      cd ~/aws-iot-device-sdk-python-v2/samples
      ```

   2. Ersetzen Sie in Ihrem Befehlszeilenfenster *your-iot-endpoint* wie angegeben und führen Sie diesen Befehl aus.

```
python3 mqtt5_shared_subscription.py --endpoint your-iot-endpoint --ca_file
 ~/certs/Amazon-root-CA-1.pem --cert ~/certs/device.pem.crt --key ~/certs/
private.pem.key --group_identifier consumer
```

Windows

Um die Beispiel-App auf einem Windows-PC auszuführen

1. Navigieren Sie in Ihrem Befehlszeilenfenster zu dem `%USERPROFILE%\aws-iot-device-sdk-python-v2\samples`-Verzeichnis, das das SDK erstellt hat, und installieren Sie die Beispiel-App mithilfe dieser Befehle.

   ```
   cd %USERPROFILE%\aws-iot-device-sdk-python-v2\samples
   ```

2. Ersetzen Sie in Ihrem Befehlszeilenfenster *`your-iot-endpoint`* wie angegeben und führen Sie diesen Befehl aus.

   ```
   python3 mqtt5_shared_subscription.py --endpoint your-iot-endpoint --ca_file
    %USERPROFILE%\certs\Amazon-root-CA-1.pem --cert %USERPROFILE%\certs
   \device.pem.crt --key %USERPROFILE%\certs\private.pem.key --group_identifier
    consumer
   ```

> **Note**
>
> Sie können optional eine Gruppen-ID angeben, die Ihren Anforderungen entspricht, wenn Sie das Beispiel ausführen (z. B. `--group_identifier consumer`). Wenn Sie kein angeben, ist `python-sample` der Standardgruppen-Identifier.

3. Die Ausgabe in Ihrer Befehlszeile kann wie folgt aussehen:

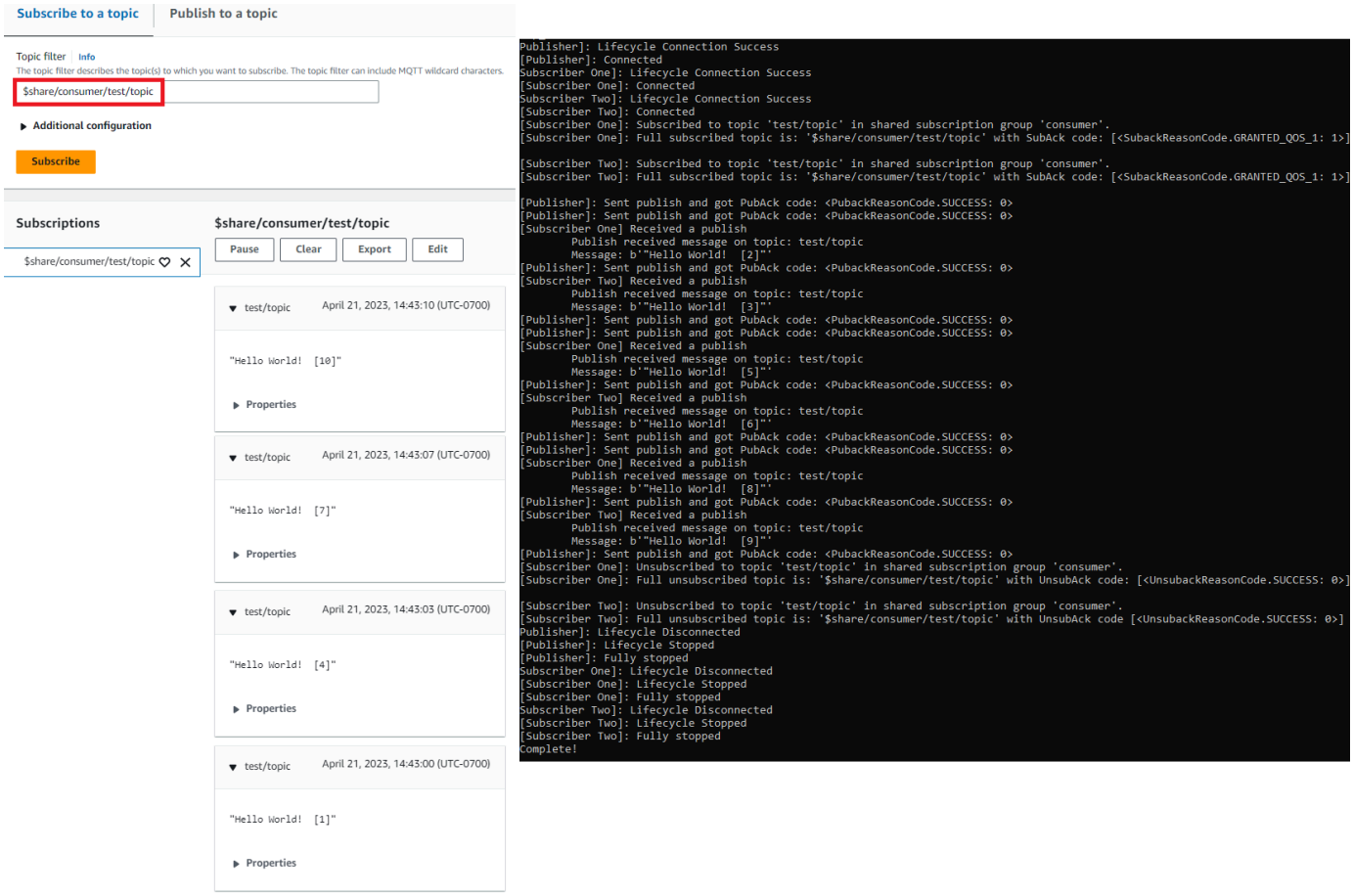
```
Publisher]: Lifecycle Connection Success
[Publisher]: Connected
Subscriber One]: Lifecycle Connection Success
[Subscriber One]: Connected
Subscriber Two]: Lifecycle Connection Success
[Subscriber Two]: Connected
[Subscriber One]: Subscribed to topic 'test/topic' in shared subscription group
 'consumer'.
```

```
[Subscriber One]: Full subscribed topic is: '$share/consumer/test/topic' with
 SubAck code: [<SubackReasonCode.GRANTED_QOS_1: 1>]
[Subscriber Two]: Subscribed to topic 'test/topic' in shared subscription group
 'consumer'.
[Subscriber Two]: Full subscribed topic is: '$share/consumer/test/topic' with
 SubAck code: [<SubackReasonCode.GRANTED_QOS_1: 1>]
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber Two] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [1]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber One] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [2]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber Two] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [3]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber One] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [4]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber Two] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [5]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber One] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [6]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber Two] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [7]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber One] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [8]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber Two] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [9]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber One] Received a publish
```

```
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [10]"'
[Subscriber One]: Unsubscribed to topic 'test/topic' in shared subscription group
 'consumer'.
[Subscriber One]: Full unsubscribed topic is: '$share/consumer/test/topic' with
 UnsubAck code: [<UnsubackReasonCode.SUCCESS: 0>]
[Subscriber Two]: Unsubscribed to topic 'test/topic' in shared subscription group
 'consumer'.
[Subscriber Two]: Full unsubscribed topic is: '$share/consumer/test/topic' with
 UnsubAck code [<UnsubackReasonCode.SUCCESS: 0>]
Publisher]: Lifecycle Disconnected
[Publisher]: Lifecycle Stopped
[Publisher]: Fully stopped
Subscriber One]: Lifecycle Disconnected
[Subscriber One]: Lifecycle Stopped
[Subscriber One]: Fully stopped
Subscriber Two]: Lifecycle Disconnected
[Subscriber Two]: Lifecycle Stopped
[Subscriber Two]: Fully stopped
Complete!
```

4. Öffnen Sie den MQTT-Testclient in der AWS IoT -Konsole. Abonnieren Sie unter Thema abonnieren das Thema des gemeinsamen Abonnements, z. B.: `$share/consumer/test/topic`. Sie können bei der Ausführung des Beispiels eine Gruppen-ID angeben, die Ihren Anforderungen entspricht (z. B. `--group_identifier consumer`). Wenn Sie keine Gruppen-ID angeben, ist der Standardwert `python-sample`. Weitere Informationen finden Sie im Python-Beispiel für MQTT 5 Shared Subscription und im AWS IoT Core Developer Guide unter Shared Subscriptions.

   Führen Sie in Ihrem Befehlszeilenfenster die Beispiel-App erneut aus und beobachten Sie die Verteilung der Nachrichten in Ihrem MQTT-Testclient der AWS IoT -Konsole und der Befehlszeile.



Verbinden eines Raspberry Pi oder eines anderes Gerätes

In diesem Abschnitt konfigurieren wir einen Raspberry Pi für die Verwendung mit AWS IoT. Wenn Sie ein anderes Gerät haben, das Sie anschließen möchten, finden Sie in der Anleitung für den Raspberry Pi Hinweise, die Ihnen helfen können, diese Anweisungen an Ihr Gerät anzupassen.

Dies dauert normalerweise etwa 20 Minuten, kann aber länger dauern, wenn Sie viele Systemsoftware-Upgrades installieren müssen.

In diesem Tutorial führen Sie die folgenden Aktivitäten durch:

- Einrichten Ihres Geräts
- Installieren Sie die erforderlichen Tools und Bibliotheken für AWS IoT Geräte-SDK
- Installieren AWS IoT Geräte-SDK
- Installieren und Ausführen der Beispiel-App
- Nachrichten aus der Beispiel-App anzeigen im AWS IoT Konsole

> **Important**
>
> Die Anpassung dieser Anweisungen an andere Geräte und Betriebssysteme kann schwierig sein. Sie müssen Ihr Gerät gut genug verstehen, um diese Anweisungen interpretieren und auf Ihr Gerät anwenden zu können.
> Wenn Sie bei der Konfiguration Ihres Geräts für auf Schwierigkeiten stoßen AWS IoT, können Sie alternativ eine der anderen Geräteoptionen ausprobieren, z. B. Erstellen Sie ein virtuelles Gerät mit Amazon EC2 oderVerwenden Sie Ihren Windows- oder Linux-PC oder Mac als AWS IoT Gerät.

Einrichten Ihres Geräts

Ziel dieses Schritts ist es, alles zu sammeln, was Sie benötigen, um Ihr Gerät so zu konfigurieren, dass es das Betriebssystem (OS) starten, eine Verbindung zum Internet herstellen und Ihnen die Interaktion mit dem Gerät über eine Befehlszeilenschnittstelle ermöglichen kann.

Zum Durcharbeiten dieses Tutorials ist Folgendes erforderlich:

- Ein AWS-Konto. Wenn dies nicht der Fall ist, führen Sie die unter Einrichten AWS-Konto beschriebenen Schritte aus, bevor Sie fortfahren.
- Ein Raspberry Pi 3 Modell B oder ein neueres Modell. Dies funktioniert möglicherweise auf früheren Versionen des Raspberry Pi, diese wurden jedoch nicht getestet.
- Raspberry Pi OS (32-Bit) oder höher. Wir empfehlen die neueste Version des Raspberry Pi OS zu verwenden. Frühere Versionen des Betriebssystems funktionieren möglicherweise, wurden jedoch nicht getestet.

  Um dieses Beispiel auszuführen, müssen Sie den Desktop nicht mit der grafischen Benutzeroberfläche (GUI) installieren. Wenn Sie jedoch mit dem Raspberry Pi noch nicht vertraut sind und Ihre Raspberry Pi-Hardware ihn unterstützt, ist es möglicherweise einfacher, den Desktop mit der GUI zu verwenden.
- Ein Ethernet oder eine WiFi Verbindung.
- Tastatur, Maus, Monitor, Kabel, Netzteile und andere Hardware, die für Ihr Gerät erforderlich ist.

> **Important**
>
> Bevor Sie mit dem nächsten Schritt fortfahren, muss das Betriebssystem Ihres Geräts installiert, konfiguriert und ausgeführt werden. Das Gerät muss mit dem Internet verbunden sein und Sie müssen über die Befehlszeilenschnittstelle auf das Gerät zugreifen können. Der Befehlszeilenzugriff kann über eine direkt verbundene Tastatur, Maus und einen Monitor oder über eine SSH-Terminal-Fernschnittstelle erfolgen.

Wenn Sie auf Ihrem Raspberry Pi ein Betriebssystem mit grafischer Benutzeroberfläche (GUI) ausführen, öffnen Sie ein Terminalfenster auf dem Gerät und führen Sie in diesem Fenster die folgenden Anweisungen aus. Andernfalls, wenn Sie über ein Remote-Terminal wie PuTTY eine Verbindung zu Ihrem Gerät herstellen, öffnen Sie ein Remote-Terminal für Ihr Gerät und verwenden Sie dieses.

Installieren Sie die erforderlichen Tools und Bibliotheken für AWS IoT Geräte-SDK

Bevor Sie das AWS IoT Geräte-SDK und den Beispielcode installieren, stellen Sie sicher, dass Ihr System auf dem neuesten Stand ist und über die erforderlichen Tools und Bibliotheken für die Installation der SDKs verfügt.

1. Aktualisieren Sie das Betriebssystem und installieren Sie die erforderlichen Bibliotheken

   Bevor Sie ein AWS IoT Geräte-SDK installieren, führen Sie diese Befehle in einem Terminalfenster auf Ihrem Gerät aus, um das Betriebssystem zu aktualisieren und die erforderlichen Bibliotheken zu installieren.

   ```
   sudo apt-get update
   ```

   ```
   sudo apt-get upgrade
   ```

   ```
   sudo apt-get install cmake
   ```

   ```
   sudo apt-get install libssl-dev
   ```

2. Installieren Sie Git

Wenn auf dem Betriebssystem Ihres Geräts Git nicht installiert ist, müssen Sie es installieren, um das AWS IoT Geräte-SDK für zu installieren JavaScript.

a. Testen Sie, ob Git bereits installiert ist, indem Sie diesen Befehl ausführen.

```
git --version
```

b. Wenn der vorherige Befehl die Git-Version anzeigt, ist Git bereits installiert und Sie können mit Schritt 3 fortfahren.

c. Wenn bei der Ausführung des Befehls git ein Fehler angezeigt wird, installieren Sie Git, indem Sie diesen Befehl ausführen.

```
sudo apt-get install git
```

d. Testen Sie erneut, ob Git installiert ist, indem Sie diesen Befehl ausführen.

```
git --version
```

e. Wenn Git installiert ist, fahren Sie mit dem nächsten Abschnitt fort. Wenn nicht, beheben Sie den Fehler und korrigieren Sie ihn, bevor Sie fortfahren. Sie benötigen Git, um das AWS IoT Device SDK für zu installieren JavaScript.

Installieren AWS IoT Geräte-SDK

Installieren Sie das AWS IoT Geräte-SDK.

Python

In diesem Abschnitt installieren Sie Python, seine Entwicklungstools und das AWS IoT Device SDK für Python auf Ihrem Gerät. Diese Anweisungen gelten für einen Raspberry Pi, auf dem das neueste Raspberry Pi-Betriebssystem ausgeführt wird. Wenn Sie mit einem anderen Gerät arbeiten oder ein anderes Betriebssystem verwenden, müssen Sie diese Anweisungen für Ihr Gerät anpassen.

1. Installieren Sie Python und seine Entwicklungstools

   Für das AWS IoT Device SDK für Python muss Python v3.5 oder höher auf Ihrem Raspberry Pi installiert sein.

Führen Sie diese Befehle in einem Terminalfenster auf Ihrem Gerät aus.

1. Führen Sie diesen Befehl aus, um die auf Ihrem Gerät installierte Python-Version zu bestimmen.

   ```
   python3 --version
   ```

   Wenn Python installiert ist, wird seine Version angezeigt.
2. Wenn die angezeigte Version `Python 3.5` oder höher ist, können Sie mit Schritt 2 fortfahren.
3. Wenn die angezeigte Version niedriger als `Python 3.5` ist, können Sie die richtige Version installieren, indem Sie diesen Befehl ausführen.

   ```
   sudo apt install python3
   ```
4. Führen Sie diesen Befehl aus, um zu bestätigen, dass die richtige Version von Python jetzt installiert ist.

   ```
   python3 --version
   ```

2. Testen Sie auf pip3

Führen Sie diese Befehle in einem Terminalfenster Ihres Geräts aus.

1. Führen Sie diesen Befehl aus, um zu sehen, ob pip3 installiert ist.

   ```
   pip3 --version
   ```
2. Wenn der Befehl eine Versionsnummer zurückgibt, ist pip3 installiert und Sie können mit Schritt 3 fortfahren.
3. Wenn der vorherige Befehl einen Fehler zurückgibt, führen Sie diesen Befehl zur Installation von pip3 aus.

   ```
   sudo apt install python3-pip
   ```
4. Führen Sie diesen Befehl aus, um zu sehen, ob pip3 installiert ist.

   ```
   pip3 --version
   ```

3. Installieren Sie das aktuelle AWS IoT Geräte-SDK für Python

   Installieren Sie das AWS IoT Device SDK für Python und laden Sie die Beispiel-Apps auf Ihr Gerät herunter.

   Führen Sie diese Befehle auf Ihrem Gerät aus.

   ```
   cd ~
   python3 -m pip install awsiotsdk
   ```

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
   ```

JavaScript

In diesem Abschnitt installieren Node.js Sie den npm-Paketmanager und das AWS IoT Geräte-SDK für JavaScript auf Ihrem Gerät. Diese Anweisungen gelten für einen Raspberry Pi, auf dem das Raspberry Pi OS ausgeführt wird. Wenn Sie mit einem anderen Gerät arbeiten oder ein anderes Betriebssystem verwenden, müssen Sie diese Anweisungen für Ihr Gerät anpassen.

1. Installieren Sie die neueste Version von Node.js

   Das AWS IoT Geräte-SDK für JavaScript erfordert, dass Node.js und der NPM-Paketmanager auf Ihrem Raspberry Pi installiert sind.

   a. Laden Sie die aktuelle Version des Knoten-Repositorys herunter, indem Sie einen der folgenden Befehle eingeben.

      ```
      cd ~
      curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -
      ```

   b. Installieren Sie Node und npm.

      ```
      sudo apt-get install -y nodejs
      ```

   c. Überprüfen Sie die Installation von Node.

      ```
      node -v
      ```

Vergewissern Sie sich, dass der Befehl die Node-Version anzeigt. Dieses Tutorial erfordert Node v10.0 oder neuer. Wenn die Node-Version nicht angezeigt wird, versuchen Sie erneut, das Node-Repository herunterzuladen.

d. Überprüfen Sie die Installation von npm.

```
npm -v
```

Vergewissern Sie sich, dass der Befehl die npm-Version anzeigt. Wenn die npm-Version nicht angezeigt wird, versuchen Sie erneut, Node und npm zu installieren.

e. Starten Sie das Gerät neu.

```
sudo shutdown -r 0
```

Fahren Sie nach dem Neustart des Geräts fort.

2. Installieren Sie das AWS IoT Geräte-SDK für JavaScript

Installieren Sie das AWS IoT Geräte-SDK für JavaScript auf Ihrem Raspberry Pi.

a. Klonen Sie das AWS IoT Geräte-SDK für das JavaScript Repository in das `aws-iot-device-sdk-js-v2` Verzeichnis Ihres *home* Verzeichnisses. Auf dem Raspberry Pi ist das *home* Verzeichnis`~/`, das in den folgenden Befehlen als *home* Verzeichnis verwendet wird. Wenn Ihr Gerät einen anderen Pfad für das *home* Verzeichnis verwendet, müssen Sie es in `~/` den folgenden Befehlen durch den richtigen Pfad für Ihr Gerät ersetzen.

Diese Befehle erstellen das Verzeichnis `~/aws-iot-device-sdk-js-v2` und kopieren den SDK-Code hinein.

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-js-v2.git
```

b. Wechseln Sie zu dem Verzeichnis `aws-iot-device-sdk-js-v2`, das Sie im vorherigen Schritt erstellt haben, und führen Sie `npm install` aus, um das SDK zu installieren. Der Befehl `npm install` ruft den Build der Bibliothek `aws-crt` auf, der einige Minuten in Anspruch nehmen kann.

```
cd ~/aws-iot-device-sdk-js-v2
```

```
npm install
```

Installieren und Ausführen der Beispiel-App

In diesem Abschnitt installieren und führen Sie die `pubsub` Beispiel-App aus dem AWS IoT Geräte-SDK aus. Diese App zeigt, wie Ihr Gerät die MQTT-Bibliothek verwendet, um MQTT-Nachrichten zu veröffentlichen und zu abonnieren. Die Beispiel-App abonniert ein Thema, `topic_1`, veröffentlicht 10 Nachrichten zu diesem Thema und zeigt die Nachrichten so an, wie sie vom Message Broker empfangen wurden.

Installieren Sie die Zertifikatdateien

Für die Beispiel-App müssen die Zertifikatsdateien, die das Gerät authentifizieren, auf dem Gerät installiert sein.

Um die Gerätezertifikatsdateien für die Beispiel-App zu installieren

1. Erstellen Sie ein `certs` Unterverzeichnis in Ihrem *home* Verzeichnis, indem Sie diese Befehle ausführen.

   ```
   cd ~
   mkdir certs
   ```

2. Kopieren Sie das Zertifikat, den privaten Schlüssel und das Stammzertifikat, das Sie in `~/certs` erstellt haben, in das Verzeichnis the section called “Create AWS IoT Ressourcen”.

   Wie Sie die Zertifikatsdateien auf Ihr Gerät kopieren, hängt vom Gerät und Betriebssystem ab und wird hier nicht beschrieben. Wenn Ihr Gerät jedoch eine grafische Benutzeroberfläche (GUI) unterstützt und über einen Webbrowser verfügt, können Sie das unter the section called “Create AWS IoT Ressourcen” beschriebene Verfahren vom Webbrowser Ihres Geräts aus ausführen, um die resultierenden Dateien direkt auf Ihr Gerät herunterzuladen.

   Bei den Befehlen im nächsten Abschnitt wird davon ausgegangen, dass Ihr Schlüssel und Ihr Zertifikatdateien wie in dieser Tabelle gezeigt auf dem Gerät gespeichert sind.

   Namen der Zertifikatsdateien

Datei	Dateipfad
CA-Stammzertifikat	`~/certs/Amazon-root-CA-1.pem`

Datei	Dateipfad
Gerätezertifikat	`~/certs/device.pem.crt`
Privater Schlüssel	`~/certs/private.pem.key`

Zur Ausführung der Beispielanwendung benötigen Sie die folgenden Informationen:

Anwendungsparameterwerte

Parameter	Wo der Wert gefunden werden kann
`your-iot-endpoint`	Wählen Sie in der AWS IoT -Konsole Alle Geräte und dann Objekte aus. Auf der Einstellungsseite im AWS IoT Menü. Ihr Endpunkt wird im Abschnitt Gerätedaten-Endpunkt angezeigt.

Der *`your-iot-endpoint`* Wert hat ein Format von:*`endpoint_id`*`-ats.iot.`*`region`*`.amazonaws.com`, zum Beispiel`a3qj468EXAMPLE-ats.iot.us-west-2.amazonaws.com`.

Python

Um die Beispiel-App zu installieren und auszuführen

1. Navigieren Sie zum Beispielverzeichnis.

   ```
   cd ~/aws-iot-device-sdk-python-v2/samples
   ```

2. Ersetzen Sie im Befehlszeilenfenster *`your-iot-endpoint`* wie angegeben und führen Sie diesen Befehl aus.

   ```
   python3 pubsub.py --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --
   cert ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-
   endpoint
   ```

3. Beachten Sie, dass die Beispiel-App:

1. Stellt eine Verbindung mit dem AWS IoT Dienst für Ihr Konto her.
2. Das Nachrichtenthema topic_1 abonniert und die Nachrichten anzeigt, die es zu diesem Thema erhält.
3. 10 Nachrichten zum Thema topic_1 veröffentlicht.
4. Ihre Ausgabe sieht ähnlich aus wie:

```
Connecting to a3qEXAMPLEffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-0c8ae2ff-cc87-49d2-a82a-ae7ba1d0ca5a'...
Connected!
Subscribing to topic 'topic_1'...
Subscribed with QoS.AT_LEAST_ONCE
Sending 10 message(s)
Publishing message to topic 'topic_1': Hello World! [1]
Received message from topic 'topic_1': b'Hello World! [1]'
Publishing message to topic 'topic_1': Hello World! [2]
Received message from topic 'topic_1': b'Hello World! [2]'
Publishing message to topic 'topic_1': Hello World! [3]
Received message from topic 'topic_1': b'Hello World! [3]'
Publishing message to topic 'topic_1': Hello World! [4]
Received message from topic 'topic_1': b'Hello World! [4]'
Publishing message to topic 'topic_1': Hello World! [5]
Received message from topic 'topic_1': b'Hello World! [5]'
Publishing message to topic 'topic_1': Hello World! [6]
Received message from topic 'topic_1': b'Hello World! [6]'
Publishing message to topic 'topic_1': Hello World! [7]
Received message from topic 'topic_1': b'Hello World! [7]'
Publishing message to topic 'topic_1': Hello World! [8]
Received message from topic 'topic_1': b'Hello World! [8]'
Publishing message to topic 'topic_1': Hello World! [9]
Received message from topic 'topic_1': b'Hello World! [9]'
Publishing message to topic 'topic_1': Hello World! [10]
Received message from topic 'topic_1': b'Hello World! [10]'
10 message(s) received.
Disconnecting...
Disconnected!
```

Wenn Probleme bei der Ausführung der Beispiel-App auftreten, überprüfen Sie the section called “Beheben Sie Probleme mit der Beispielanwendung”.

Sie können den Parameter `--verbosity Debug` auch zur Befehlszeile hinzufügen, sodass die Beispiel-App detaillierte Meldungen darüber anzeigt, was sie tut. Diese Informationen bieten Ihnen möglicherweise die Hilfe, die Sie zur Behebung des Problems benötigen.

JavaScript

Um die Beispiel-App zu installieren und auszuführen

1. Navigieren Sie in Ihrem Befehlszeilenfenster zu dem `~/aws-iot-device-sdk-js-v2/samples/node/pub_sub`-Verzeichnis, das das SDK erstellt hat, und installieren Sie die Beispiel-App mithilfe dieser Befehle. Der Befehl `npm install` ruft den Build der `aws-crt` Bibliothek auf, der einige Minuten in Anspruch nehmen kann.

   ```
   cd ~/aws-iot-device-sdk-js-v2/samples/node/pub_sub
   npm install
   ```

2. Ersetzen Sie im Befehlszeilenfenster *`your-iot-endpoint`* wie angegeben und führen Sie diesen Befehl aus.

   ```
   node dist/index.js --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --
   cert ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-
   endpoint
   ```

3. Beachten Sie, dass die Beispiel-App:

   1. Stellt eine Verbindung mit dem AWS IoT Dienst für Ihr Konto her.
   2. Das Nachrichtenthema topic_1 abonniert und die Nachrichten anzeigt, die es zu diesem Thema erhält.
   3. 10 Nachrichten zum Thema topic_1 veröffentlicht.
   4. Ihre Ausgabe sieht ähnlich aus wie:

   ```
   Publish received on topic topic_1
   {"message":"Hello world!","sequence":1}
   Publish received on topic topic_1
   {"message":"Hello world!","sequence":2}
   Publish received on topic topic_1
   {"message":"Hello world!","sequence":3}
   Publish received on topic topic_1
   ```

```
{"message":"Hello world!","sequence":4}
Publish received on topic topic_1
{"message":"Hello world!","sequence":5}
Publish received on topic topic_1
{"message":"Hello world!","sequence":6}
Publish received on topic topic_1
{"message":"Hello world!","sequence":7}
Publish received on topic topic_1
{"message":"Hello world!","sequence":8}
Publish received on topic topic_1
{"message":"Hello world!","sequence":9}
Publish received on topic topic_1
{"message":"Hello world!","sequence":10}
```

Wenn Probleme bei der Ausführung der Beispiel-App auftreten, überprüfen Sie the section called “Beheben Sie Probleme mit der Beispielanwendung”.

Sie können den Parameter `--verbosity Debug` auch zur Befehlszeile hinzufügen, sodass die Beispiel-App detaillierte Meldungen darüber anzeigt, was sie tut. Diese Informationen bieten Ihnen möglicherweise die Hilfe, die Sie zur Behebung des Problems benötigen.

Nachrichten aus der Beispiel-App anzeigen im AWS IoT Konsole

Mithilfe des MQTT-Testclients in der AWS IoT -Konsole können Sie die Nachrichten der Beispiel-App sehen, während sie den Message Broker durchlaufen.

Um die von der Beispiel-App veröffentlichten MQTT-Nachrichten anzuzeigen

1. Sehen Sie sich MQTT-Nachrichten anzeigen mit dem AWS IoT MQTT-Client an. Auf diese Weise lernen Sie, wie Sie den MQTT-Testclient in der AWS IoT -Konsole verwenden, um MQTT-Nachrichten anzuzeigen, während sie den Message Broker passieren.
2. Öffnen Sie den MQTT-Testclient in der AWS IoT -Konsole.
3. Abonnieren Sie das Thema topic_1.
4. Führen Sie in Ihrem Befehlszeilenfenster die Beispiel-App erneut aus und sehen Sie sich die Nachrichten im MQTT-Client in der AWS IoT -Konsole an.

   Python

   ```
   cd ~/aws-iot-device-sdk-python-v2/samples
   ```

```
python3 pubsub.py --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --
cert ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-
endpoint
```

JavaScript

```
cd ~/aws-iot-device-sdk-js-v2/samples/node/pub_sub
node dist/index.js --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --
cert ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-
endpoint
```

Beheben Sie Probleme mit der Beispielanwendung

Wenn beim Versuch, die Beispiel-App auszuführen, ein Fehler auftritt, sollten Sie Folgendes überprüfen.

Überprüfen Sie das Zertifikat

Wenn das Zertifikat nicht aktiv ist, AWS IoT werden keine Verbindungsversuche akzeptiert, bei denen es für die Autorisierung verwendet wird. Bei der Erstellung Ihres Zertifikats ist es leicht, die Schaltfläche Aktivieren zu übersehen. Zum Glück können Sie Ihr Zertifikat von der AWS IoT -Konsole aus aktivieren.

Um die Aktivierung Ihres Zertifikats zu überprüfen

1. Wählen Sie in der AWS IoT -Konsole im linken Menü die Option Sicher und dann Zertifikate aus.
2. Suchen Sie in der Liste der Zertifikate nach dem Zertifikat, das Sie für die Übung erstellt haben, und überprüfen Sie seinen Status in der Spalte Status.

   Wenn Sie sich nicht an den Namen des Zertifikats erinnern, suchen Sie nach den Zertifikaten, die inaktiv sind, um festzustellen, ob es sich dabei möglicherweise um das Zertifikat handelt, das Sie verwenden.

   Wählen Sie das Zertifikat in der Liste aus, um die Detailseite zu öffnen. Auf der Detailseite können Sie das Erstellungsdatum sehen, damit Sie das Zertifikat leichter identifizieren können.
3. Um ein inaktives Zertifikat zu aktivieren, wählen Sie auf der Detailseite des Zertifikats Aktionen und dann Aktivieren aus.

Wenn Sie das richtige Zertifikat gefunden haben und es aktiv ist, Sie aber immer noch Probleme beim Ausführen der Beispiel-App haben, überprüfen Sie die Richtlinien, wie im nächsten Schritt beschrieben.

Sie können auch versuchen, eine neue Sache und ein neues Zertifikat zu erstellen, indem Sie die Schritte unter the section called “Dies erstellt ein Objekt” befolgen. Wenn Sie eine neue Sache erstellen, müssen Sie ihr einen neuen Namen geben und die neuen Zertifikatsdateien auf Ihr Gerät herunterladen.

Prüfen Sie die dem Zertifikat angefügte Richtlinie

Richtlinien autorisieren Aktionen in AWS IoT. Wenn das Zertifikat, mit dem eine Verbindung zu AWS IoT hergestellt wird, keine Richtlinie hat oder nicht über eine Richtlinie verfügt, die das Herstellen einer Verbindung ermöglicht, wird die Verbindung verweigert, auch wenn das Zertifikat aktiv ist.

Um die einem Zertifikat beigefügten Richtlinien zu überprüfen

1. Suchen Sie das Zertifikat, wie im vorherigen Artikel beschrieben, und öffnen Sie die zugehörige Detailseite.
2. Wählen Sie im linken Menü der Detailseite des Zertifikats die Option Richtlinien aus, um die mit dem Zertifikat verknüpften Richtlinien anzuzeigen.
3. Wenn dem Zertifikat keine Richtlinien zugeordnet sind, fügen Sie eine hinzu, indem Sie das Menü Aktionen und dann Richtlinie anhängen wählen.

   Wählen Sie die Regel aus, die Sie zuvor in the section called “Create AWS IoT Ressourcen” erstellt haben.
4. Wenn eine Richtlinie angehängt ist, wählen Sie die Richtlinienkachel aus, um die Detailseite zu öffnen.

   Überprüfen Sie auf der Detailseite das Richtliniendokument, um sicherzustellen, dass es dieselben Informationen enthält wie das, in dem Sie es in the section called “Erstellen Sie ein AWS IoT policy” erstellt haben.

Überprüfen Sie die Befehlszeile

Stellen Sie sicher, dass Sie die richtige Befehlszeile für Ihr System verwendet haben. Die auf Linux- und MacOS-Systemen verwendeten Befehle unterscheiden sich häufig von denen, die auf Windows-Systemen verwendet werden.

Überprüfen Sie die Endpunktadresse

Überprüfen Sie den Befehl, den Sie eingegeben haben, und überprüfen Sie die Endpunktadresse in Ihrem Befehl noch einmal mit der Adresse in Ihrer AWS IoT -Konsole.

Überprüfen Sie die Dateinamen der Zertifikatsdateien

Vergleichen Sie die Dateinamen in dem Befehl, den Sie eingegeben haben, mit den Dateinamen der Zertifikatsdateien im Verzeichnis `certs`.

Bei einigen Systemen müssen die Dateinamen möglicherweise in Anführungszeichen gesetzt werden, damit sie korrekt funktionieren.

Überprüfen Sie die SDK-Installation

Stellen Sie sicher, dass Ihre SDK-Installation vollständig und korrekt ist.

Installieren Sie im Zweifelsfall das SDK erneut auf Ihrem Gerät. In den meisten Fällen müssen Sie den Abschnitt des Tutorials mit dem Titel Installieren des AWS IoT Geräte-SDK für finden *`SDK language`* und das Verfahren erneut ausführen.

Wenn Sie das AWS IoT Geräte-SDK für verwenden JavaScript, denken Sie daran, die Beispiel-Apps zu installieren, bevor Sie versuchen, sie auszuführen. Durch die Installation des SDK werden die Beispiel-Apps nicht automatisch installiert. Die Beispiel-Apps müssen nach der Installation des SDK manuell installiert werden.

MQTT-Nachrichten anzeigen mit dem AWS IoT MQTT-Client

In diesem Abschnitt wird beschrieben, wie Sie den AWS IoT MQTT-Testclient in der AWS IoT Konsole verwenden, um die gesendeten und empfangenen MQTT-Nachrichten von zu beobachten. AWS IoT Das in diesem Abschnitt verwendete Beispiel bezieht sich auf die in verwendeten Beispiele. Sie können das in den Beispielen *`topicName`* verwendete Beispiel jedoch durch einen beliebigen Themennamen oder Themenfilter ersetzenErste Schritte mit AWS IoT Core Tutorials, der von Ihrer IoT-Lösung verwendet wird.

Geräte veröffentlichen MQTT-Nachrichten, die durch Themen gekennzeichnet sind, um ihnen ihren Status mitzuteilen AWS IoT, und veröffentlichen AWS IoT MQTT-Nachrichten, um die Geräte und Apps über Änderungen und Ereignisse zu informieren. Sie können den MQTT-Client verwenden, um diese Themen zu abonnieren und die Nachrichten zu beobachten, sobald sie auftreten. Sie können

den MQTT-Testclient auch verwenden, um MQTT-Nachrichten auf abonnierten Geräten und Diensten in Ihrem zu veröffentlichen. AWS-Konto

Inhalt

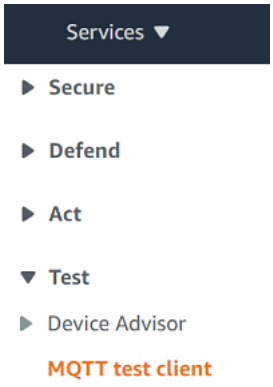
- Anzeigen von MQTT-Nachrichten im MQTT-Client
- Veröffentlichen von MQTT-Nachrichten vom MQTT-Client
- Testen von geteilten Abonnements im MQTT-Client

Anzeigen von MQTT-Nachrichten im MQTT-Client

Das folgende Verfahren erklärt, wie Sie ein bestimmtes MQTT-Thema abonnieren, für das Ihr Gerät Nachrichten veröffentlicht, und wie Sie diese Nachrichten in der Konsole anzeigen können.AWS IoT

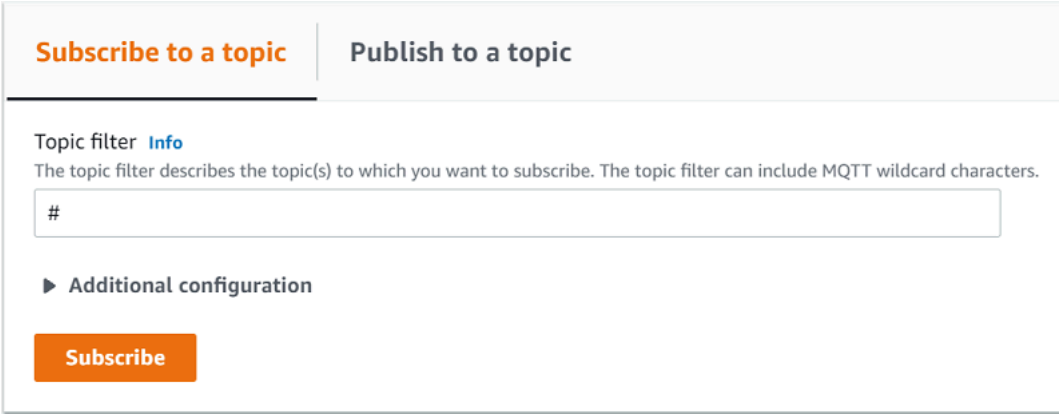
So zeigen Sie MQTT-Nachrichten im MQTT-Testclient an

1. Öffnen Sie die AWS IoT -Konsole und wählen Sie im linken Menü Test, um den MQTT-Client zu öffnen.

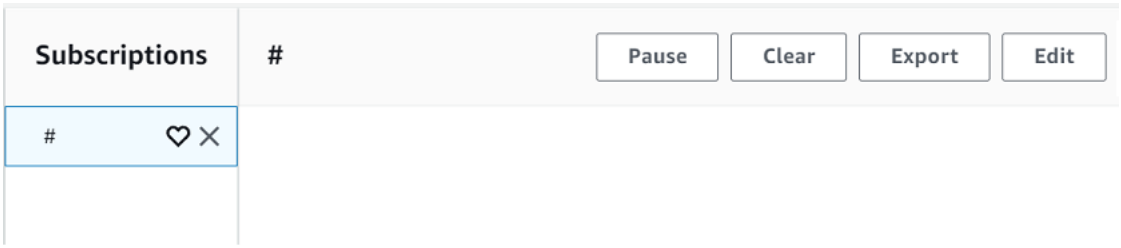


2. Geben Sie auf der Registerkarte Thema abonnieren die Option ein, *topicName* um das Thema zu abonnieren, zu dem Ihr Gerät veröffentlicht. Abonnieren Sie für die Beispiel-App „Erste Schritte“ **#**, womit alle Nachrichtenthemen abonniert werden.

   Um mit dem Beispiel Erste Schritte fortzufahren, geben Sie auf der Registerkarte Thema abonnieren im Feld Themenfilter **#** ein, und wählen Sie dann Abonnieren aus.



Die Protokollseite der Themennachricht, #, wird geöffnet und **#** wird in der Abonnementliste angezeigt. Wenn das Gerät, auf dem Sie konfiguriert haben, das Beispielprogramm ausführt, sollten Sie die Nachrichten, an die es sendet, AWS IoT im # Message-Log sehen. the section called “Konfigurieren Ihres Geräts” Die Nachrichtenprotokolleinträge werden unter dem Abschnitt Veröffentlichen angezeigt, wenn Nachrichten mit dem abonnierten Thema bei eingegangen AWS IoT sind.



3. Auf der Seite # Nachrichtenprotokoll können Sie auch Nachrichten zu einem Thema veröffentlichen, dafür müssen Sie jedoch den Namen des Themas angeben. Veröffentlichen Sie eine Benachrichtigung für das Thema #.

   Nachrichten, die zu abonnierten Themen veröffentlicht wurden, werden im Nachrichtenprotokoll angezeigt, sobald sie empfangen wurden, wobei die neueste Nachricht an erster Stelle steht.

Fehlerbehebung für MQTT-Nachrichten

Verwenden Sie den Themenfilter mit Platzhaltern

Wenn Ihre Nachrichten nicht wie erwartet im Nachrichtenprotokoll angezeigt werden, versuchen Sie, wie unter Filter für Themennamen beschrieben, ein Platzhalter-Thema zu abonnieren. Der mehrstufige MQTT-Wildcard-Themenfilter ist das Hash- oder Rautenzeichen (#) und kann als Themenfilter im Themenfeld Abonnementsthemen verwendet werden.

Wenn Sie den # Themenfilter abonnieren, abonnieren Sie jedes Thema, das der Message Broker empfängt. Sie können den Filter eingrenzen, indem Sie Elemente des Themenfilterpfads durch ein # Platzhalterzeichen mit mehreren Ebenen oder das einstufige Platzhalterzeichen '+' ersetzen.

Bei der Verwendung von Platzhaltern in einem Themenfilter

- Das Platzhalterzeichen mit mehreren Ebenen muss das letzte Zeichen im Themenfilter sein.
- Der Themenfilterpfad kann nur ein Platzhalterzeichen mit einer Ebene pro Themenebene enthalten.

Beispielsweise:

Themenfilter	Zeigt Nachrichten an mit
#	Beliebiger Themenname
`topic_1/#`	Ein Themenname, der mit `topic_1/` beginnt
`topic_1/level_2/#`	Ein Themenname, der mit `topic_1/level_2/` beginnt
`topic_1/+/level_3`	Ein Themenname, der mit `topic_1/` beginnt, mit `/level_3` endet und dazwischen ein Element beliebigen Wertes enthält.

Weitere Informationen zu Filtern finden Sie unter Filter für Themennamen.

Suchen Sie nach Fehlern beim Themennamen

MQTT-Themennamen und Themenfilter berücksichtigen Groß- und Kleinschreibung. Wenn Ihr Gerät beispielsweise Nachrichten an `Topic_1` (mit einem großen T) statt an `topic_1`, das Thema, das Sie abonniert haben, veröffentlicht, werden seine Nachrichten nicht im MQTT-Testclient angezeigt. Wenn Sie jedoch den Themenfilter mit Platzhaltern abonnieren, wird angezeigt, dass das Gerät gerade Nachrichten veröffentlicht, und Sie könnten sehen, dass es einen Themennamen verwendet, der nicht dem entspricht, den Sie erwartet haben.

Veröffentlichen von MQTT-Nachrichten vom MQTT-Client

So veröffentlichen Sie eine Nachricht in einem MQTT-Thema

1. Geben Sie auf der Seite des MQTT-Testclients auf der Registerkarte In einem Thema veröffentlichen im Feld Themenname den Text *topicName* Ihrer Nachricht ein. Verwenden Sie **my/topic** in diesem Beispiel.

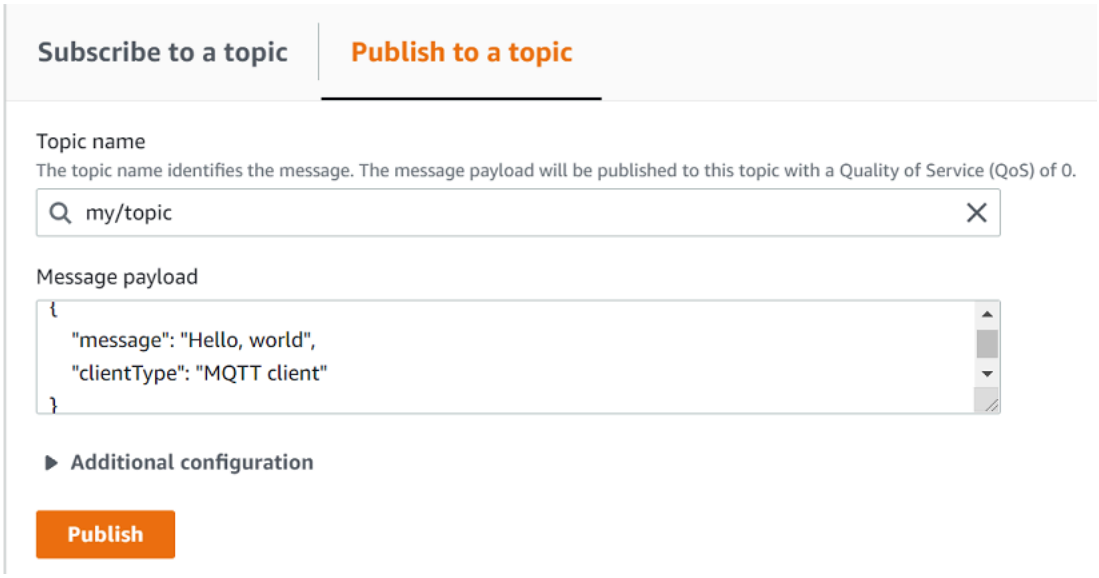
   > **Note**
   >
   > Verwenden Sie keine persönlich identifizierbaren Informationen in Themennamen, unabhängig davon, ob Sie sie im MQTT-Testclient oder in Ihrer Systemimplementierung verwenden. Themennamen können in unverschlüsselten Mitteilungen und Berichten vorkommen.

2. Geben Sie im Abschnitt für die Nachrichten-Nutzlast den folgenden JSON-Code ein:

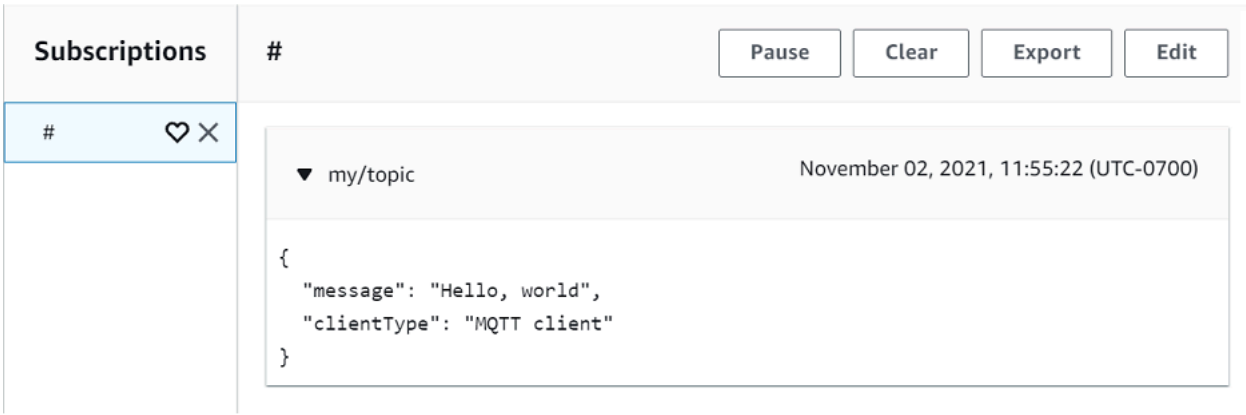
   ```
   {
       "message": "Hello, world",
       "clientType": "MQTT test client"
   }
   ```

3. Wählen Sie Veröffentlichen aus, um Ihre Nachricht in AWS IoT zu veröffentlichen.

   > **Note**
   >
   > Stellen Sie sicher, dass Sie das my/topicThema abonniert haben, bevor Sie Ihre Nachricht veröffentlichen.



4. Wählen Sie in der Abonnementliste aus, ob my/topicdie Nachricht angezeigt werden soll. Sie sollten sehen, dass die Nachricht im MQTT-Testclient unter dem Payload-Fenster für die Nachrichtenveröffentlichung erscheint.



Sie können MQTT-Nachrichten zu anderen Themen veröffentlichen, indem Sie das *topicName* im Feld Themenname ändern und die Schaltfläche Veröffentlichen wählen.

> **Important**
>
> Wenn Sie mehrere Abonnements mit sich überschneidenden Themen erstellen (z. B. probe1/temperature und probe1/#), besteht die Möglichkeit, dass eine einzelne Nachricht, die zu einem Thema veröffentlicht wurde, das zu beiden Abonnements passt, mehrfach zugestellt wird, einmal für jedes überlappende Abonnement.

Testen von geteilten Abonnements im MQTT-Client

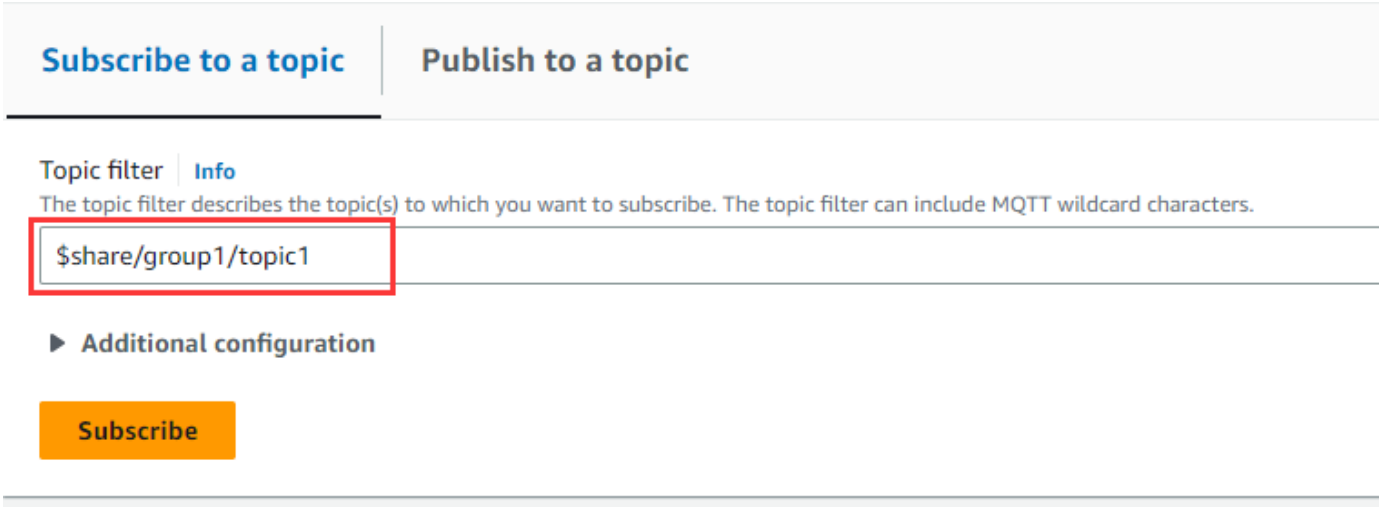
In diesem Abschnitt wird beschrieben, wie Sie den AWS IoT MQTT-Client in der AWS IoT Konsole verwenden, um die mit Shared Subscriptions gesendeten und empfangenen MQTT-Nachrichten anzusehen. AWS IoT ???ermöglicht es mehreren Clients, ein Abonnement für ein Thema gemeinsam zu nutzen, wobei nur ein Client Nachrichten erhält, die zu diesem Thema veröffentlicht wurden, und zwar nach dem Zufallsprinzip. Um zu simulieren, dass mehrere MQTT-Clients (in diesem Beispiel zwei MQTT-Clients) dasselbe Abonnement nutzen, öffnen Sie den AWS IoT MQTT-Client in der AWS IoT Konsole von mehreren Webbrowsern aus. Das in diesem Abschnitt verwendete Beispiel bezieht sich nicht auf die in Erste Schritte mit AWS IoT Core Tutorials verwendeten Beispiele. Weitere Informationen finden Sie unter Geteilte Abonnements.

Um ein Abonnement für ein MQTT-Thema zu teilen

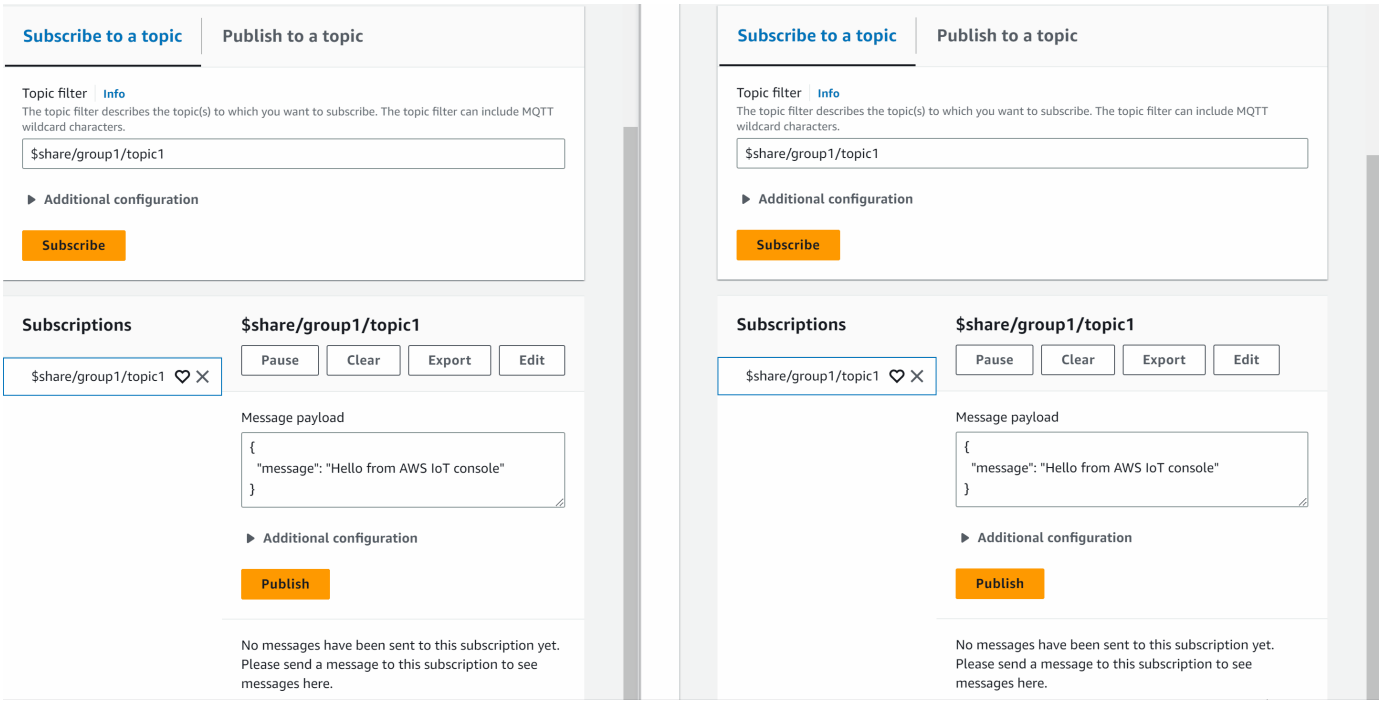
1. Wählen Sie in der AWS IoT -Konsole im Navigationsbereich Test und dann MQTT-Testclient aus.
2. Geben Sie auf der Registerkarte Thema abonnieren den ein, *topicName* um das Thema zu abonnieren, zu dem Ihr Gerät veröffentlicht. Um geteilte Abonnements zu verwenden, abonnieren Sie den Themenfilter eines geteilten Abonnements wie folgt:

   ```
   $share/{ShareName}/{TopicFilter}
   ```

   Ein Beispiel für einen Themenfilter kann **$share/group1/topic1** sein, der das Nachrichtenthema **topic1** abonniert.



3. Öffnen Sie einen anderen Webbrowser und wiederholen Sie Schritt 1 und Schritt 2. Auf diese Weise simulieren Sie zwei verschiedene MQTT-Clients, die dasselbe Abonnement **`$share/group1/topic1`** verwenden.
4. Wählen Sie einen MQTT-Client aus und geben Sie auf der Registerkarte In einem Thema veröffentlichen im Feld Themenname den Text *`topicName`* Ihrer Nachricht ein. Verwenden Sie **`topic1`** in diesem Beispiel. Versuchen Sie, die Nachricht ein paar Mal zu veröffentlichen. Aus der Abonnementliste der beiden MQTT-Clients sollten Sie erkennen können, dass die Clients die Nachricht mit einer zufälligen Verteilung empfangen. In diesem Beispiel veröffentlichen wir dreimal dieselbe Nachricht „Hallo von der AWS IoT Konsole“. Der MQTT-Client auf der linken Seite hat die Nachricht zweimal empfangen und der MQTT-Client auf der rechten Seite hat die Nachricht einmal erhalten.



AWS IoT Tutorials

Die AWS IoT Tutorials sind in zwei Lernpfade unterteilt, um zwei verschiedene Ziele zu unterstützen. Wählen Sie den Lernpfad, der am besten zu Ihrem Ziel passt.

- Sie möchten einen Machbarkeitsnachweis erstellen, um Folgendes zu testen oder zu demonstrieren AWS IoT Lösungsidee

  Folgen Sie dem the section called “Erstellen von Demos mit dem AWS IoT Geräte-Client” Lernpfad, um gängige IoT-Aufgaben und -Anwendungen mit dem AWS IoT Device Client auf Ihren Geräten zu demonstrieren. Der AWS IoT Device Client bietet Gerätesoftware, mit der Sie Ihre eigenen Cloud-Ressourcen einsetzen können, um eine Komplettlösung mit minimalem Entwicklungsaufwand zu demonstrieren.

  Informationen zum AWS IoT Device Client finden Sie im AWS IoT Device Client.
- Sie möchten lernen, wie Sie Produktionssoftware für die Implementierung Ihrer Lösung erstellen

  Folgen Sie dem the section called “Erstellen von Lösungen mit dem AWS IoT Geräte-SDKs” Lernpfad, um mithilfe eines AWS IoT Geräte-SDK Ihre eigene Lösungssoftware zu erstellen, die Ihren spezifischen Anforderungen entspricht.

  Informationen zu den verfügbaren AWS IoT Geräte-SDKs finden Sie unter???. Informationen zu den AWS SDKs finden Sie unter Tools, auf denen Sie aufbauen können. AWS

AWS IoT Optionen für den Lernpfad im Tutorial

- Erstellen von Demos mit dem AWS IoT Geräte-Client
- Erstellen von Lösungen mit dem AWS IoT Geräte-SDKs

Erstellen von Demos mit dem AWS IoT Geräte-Client

Die Tutorials in diesem Lernpfad führen Sie Schritt für Schritt durch die Entwicklung von Demonstrationssoftware mithilfe des AWS IoT Device Clients. Der AWS IoT Device Client bietet Software, die auf Ihrem IoT-Gerät ausgeführt wird, um Aspekte einer darauf aufbauenden IoT-Lösung zu testen und zu demonstrieren AWS IoT.

Das Ziel dieser Tutorials ist es, das Erkunden und Experimentieren zu erleichtern, damit Sie sicher sein können, dass Ihre Lösung AWS IoT unterstützt wird, bevor Sie Ihre Gerätesoftware entwickeln.

Was Sie in diesen Tutorials lernen werden:

- Wie bereite ich einen Raspberry Pi für die Verwendung als IoT-Gerät vor mit AWS IoT
- So demonstrieren Sie AWS IoT Funktionen mithilfe des AWS IoT Device Clients auf Ihrem Gerät

In diesem Lernpfad installieren Sie den AWS IoT Device Client auf Ihrem eigenen Raspberry Pi und erstellen die AWS IoT Ressourcen in der Cloud, um Ideen für IoT-Lösungen zu demonstrieren. In den Tutorials in diesem Lernpfad werden zwar Features anhand eines Raspberry Pi demonstriert, aber sie erläutern die Ziele und Verfahren, um Sie bei der Anpassung an andere Geräte zu unterstützen.

Voraussetzungen für die Erstellung von Demos mit dem AWS IoT Geräte-Client

In diesem Abschnitt wird beschrieben, was Sie benötigen, bevor Sie mit den Tutorials in diesem Lernpfad beginnen.

Um die Tutorials in diesem Lernpfad abzuschließen, benötigen Sie:

- Importieren in &S3; AWS-Konto

  Sie können Ihren vorhandenen verwenden AWS-Konto, falls Sie über eine verfügen. Möglicherweise müssen Sie jedoch zusätzliche Rollen oder Berechtigungen hinzufügen, um die in diesen Tutorials verwendeten AWS IoT Funktionen nutzen zu können.

  Wenn Sie ein neues erstellen müssen AWS-Konto, finden Sie weitere Informationen unterthe section called “Einrichten AWS-Konto”.
- Ein Raspberry Pi oder ein kompatibles IoT-Gerät

  In den Tutorials wird ein Raspberry Pi verwendet, da er in verschiedenen Formfaktoren erhältlich ist, allgegenwärtig ist und es sich um ein relativ kostengünstiges Demonstrationsgerät handelt. Die Tutorials wurden auf dem Raspberry Pi 3 Model B+, dem Raspberry Pi 4 Model B und auf einer Amazon EC2-Instance getestet, auf der Ubuntu Server 20.04 LTS (HVM) läuft. Um die Befehle zu verwenden AWS CLI und auszuführen, empfehlen wir, die neueste Version des Raspberry Pi OS (Raspberry Pi OS (64 Bit) oder OS Lite) zu verwenden. Frühere Versionen des Betriebssystems funktionieren möglicherweise, aber wir haben sie nicht getestet.

> **Note**
>
> In den Tutorials werden die Ziele der einzelnen Schritte erklärt, um Ihnen zu helfen, sie an IoT-Hardware anzupassen, auf der wir sie noch nicht getestet haben. Sie beschreiben jedoch nicht speziell, wie Sie sie an andere Geräte anpassen können.

- Vertrautheit mit dem Betriebssystem des IoT-Geräts

  Bei den Schritten in diesen Tutorials wird davon ausgegangen, dass Sie mit der Verwendung grundlegender Linux-Befehle und -Operationen über die von einem Raspberry Pi unterstützte Befehlszeilenschnittstelle vertraut sind. Wenn Sie mit diesen Vorgängen nicht vertraut sind, sollten Sie sich vielleicht mehr Zeit nehmen, um die Tutorials abzuschließen.

  Um diese Tutorials abzuschließen, sollten Sie bereits wissen, wie Sie:

  - grundlegende Geräteoperationen wie das Zusammenbauen und Anschließen von Komponenten, das Anschließen des Geräts an die erforderlichen Stromquellen und das Installieren und Entfernen von Speicherkarten sicher durchführen.
  - Systemsoftware und Dateien auf das Gerät hoch- und herunterladen. Wenn Ihr Gerät kein Wechselspeichergerät wie eine microSD-Karte verwendet, müssen Sie wissen, wie Sie eine Verbindung zu Ihrem Gerät herstellen und Systemsoftware und Dateien auf das Gerät hoch- und herunterladen.
  - Verbinden Sie Ihr Gerät mit den Netzwerken, in denen Sie es verwenden möchten.
  - Stellen Sie über ein SSH-Terminal oder ein ähnliches Programm von einem anderen Computer aus eine Verbindung zu Ihrem Gerät her.
  - Verwenden Sie eine Befehlszeilenschnittstelle, um Dateien und Verzeichnisse auf dem Gerät zu erstellen, zu kopieren, zu verschieben, umzubenennen und deren Berechtigungen festzulegen.
  - Installieren Sie neue Programme auf dem Gerät.
  - Übertragen Sie Dateien mit Tools wie FTP oder SCP zu und von Ihrem Gerät.
- Eine Entwicklungs- und Testumgebung für Ihre IoT-Lösung

  In den Tutorials wird die erforderliche Software und Hardware beschrieben. In den Tutorials wird jedoch davon ausgegangen, dass Sie Operationen ausführen können, die möglicherweise nicht explizit beschrieben werden. Zu den Beispielen für solche Hardware und Operationen gehören:

  - Ein lokaler Host-Computer zum Herunterladen und Speichern von Dateien

Für den Raspberry Pi ist dies normalerweise ein PC oder Laptop, der microSD-Speicherkarten lesen und auf sie schreiben kann. Der lokale Host-Computer muss:

- mit dem Internet verbunden sein.
- AWS CLI installiert und konfiguriert haben.
- Haben Sie einen Webbrowser, der die AWS Konsole unterstützt.

- Eine Möglichkeit, Ihren lokalen Host-Computer mit Ihrem Gerät zu verbinden, um mit ihm zu kommunizieren, Befehle einzugeben und Dateien zu übertragen

  Auf dem Raspberry Pi erfolgt dies häufig mithilfe von SSH und SCP vom lokalen Host-Computer aus.

- Ein Monitor und eine Tastatur zum Anschluss an Ihr IoT-Gerät

  Diese können hilfreich sein, sind aber nicht erforderlich, um die Tutorials abzuschließen.

- Eine Möglichkeit für Ihren lokalen Host-Computer und Ihre IoT-Geräte, sich mit dem Internet zu verbinden

  Dies kann eine verkabelte oder eine drahtlose Netzwerkverbindung zu einem Router oder Gateway sein, der mit dem Internet verbunden ist. Der lokale Host muss eine Verbindung zum Raspberry Pi herstellen können. Dazu müssen sie sich möglicherweise im selben lokalen Netzwerk befinden. Die Tutorials können Ihnen nicht zeigen, wie Sie dies für Ihr spezielles Gerät oder Ihre Gerätekonfiguration einrichten, aber sie zeigen, wie Sie diese Konnektivität testen können.

- Zugriff auf den Router Ihres lokalen Netzwerks, um die angeschlossenen Geräte anzuzeigen

  Um die Tutorials in diesem Lernpfad abschließen zu können, müssen Sie in der Lage sein, die IP-Adresse Ihres IoT-Geräts zu ermitteln.

  In einem lokalen Netzwerk können Sie dazu auf die Admin-Oberfläche des Netzwerk-Routers zugreifen, mit dem Ihre Geräte verbunden sind. Wenn Sie Ihrem Gerät im Router eine feste IP-Adresse zuweisen können, können Sie die Wiederverbindung nach jedem Neustart des Geräts vereinfachen.

  Wenn Sie eine Tastatur und einen Monitor an das Gerät angeschlossen haben, kann ifconfig die IP-Adresse des Geräts angezeigt werden.

  Wenn keine dieser Optionen in Frage kommt, müssen Sie einen Weg finden, die IP-Adresse des Geräts nach jedem Neustart zu ermitteln.

Wenn Sie alle Ihre Materialien haben, fahren Sie mit the section called “Vorbereitung der Verwendung des IoT Device Client” fort.

Tutorials in diesem Lernpfad

- Tutorial: Vorbereiten Ihrer Geräte für den AWS IoT Geräte-Client
- Tutorial: Installation und Konfiguration des AWS IoT Geräte-Client
- Tutorial: Demonstrieren Sie die MQTT-Nachrichtenkommunikation mit dem AWS IoT Geräte-Client
- Tutorial: Demonstrieren Sie Remote-Aktionen (Jobs) mit dem AWS IoT Geräte-Client
- Tutorial: Aufräumen nach dem Ausführen des AWS IoT Tutorials zum Device Client

Tutorial: Vorbereiten Ihrer Geräte für den AWS IoT Geräte-Client

Dieses Tutorial führt Sie durch die Initialisierung Ihres Raspberry Pi, um ihn auf die nachfolgenden Tutorials in diesem Lernpfad vorzubereiten.

Ziel dieses Tutorials ist es, die aktuelle Version des Betriebssystems des Geräts zu installieren und sicherzustellen, dass Sie im Kontext Ihrer Entwicklungsumgebung mit Ihrem Gerät kommunizieren können.

Voraussetzungen

Bevor Sie mit diesem Tutorial beginnen, stellen Sie sicher, dass die unter aufgeführten Elemente the section called “Voraussetzungen für die Erstellung von Demos mit dem AWS IoT Geräte-Client” verfügbar und einsatzbereit sind.

Für dieses Tutorial brauchen Sie ungefähr 90 Minuten.

In diesem Tutorial führen Sie die folgenden Aktivitäten durch:

- Installieren und aktualisieren Sie das Betriebssystem Ihres Geräts.
- Installieren und überprüfen Sie jegliche zusätzliche Software, die für die Ausführung der Tutorials erforderlich ist.
- Testen Sie die Konnektivität Ihres Geräts und installieren Sie die erforderlichen Zertifikate.

Nachdem Sie dieses Tutorial abgeschlossen haben, bereitet das nächste Tutorial Ihr Gerät auf die Demos vor, bei denen der AWS IoT Device Client verwendet wird.

Verfahren in diesem Tutorial

- Installieren und aktualisieren Sie das Betriebssystem des Geräts
- Installieren und verifizieren Sie die erforderliche Software auf Ihrem Gerät
- Testen Sie Ihr Gerät und speichern Sie das Amazon CA-Zertifikat

Installieren und aktualisieren Sie das Betriebssystem des Geräts

Die Verfahren in diesem Abschnitt beschreiben, wie Sie die microSD-Karte initialisieren, die der Raspberry Pi für sein Systemlaufwerk verwendet. Die microSD-Karte des Raspberry Pi enthält die Betriebssystemsoftware (OS) sowie Speicherplatz für die Anwendungsdateien. Wenn Sie keinen Raspberry Pi verwenden, folgen Sie den Anweisungen des Geräts, um die Betriebssystemsoftware des Geräts zu installieren und zu aktualisieren.

Nachdem Sie diesen Abschnitt abgeschlossen haben, sollten Sie in der Lage sein, Ihr IoT-Gerät zu starten und über das Terminalprogramm auf Ihrem lokalen Host-Computer eine Verbindung herzustellen.

Erforderliche Ausstattung:

- Ihre lokale Entwicklungs- und Testumgebung
- Ein Raspberry Pi oder Ihr IoT-Gerät, das eine Verbindung zum Internet herstellen kann
- Eine microSD-Speicherkarte mit einer Kapazität von mindestens 8 GB oder ausreichend Speicherplatz für das Betriebssystem und die erforderliche Software.

> **Note**
>
> Wählen Sie für diese Übungen eine microSD-Karte, die so groß wie nötig, aber so klein wie möglich ist.
> Eine kleine SD-Karte lässt sich schneller sichern und aktualisieren. Auf dem Raspberry Pi benötigen Sie für diese Tutorials nicht mehr als eine microSD-Karte mit 8 GB. Wenn Sie mehr Speicherplatz für Ihre spezifische Anwendung benötigen, können Sie mit den kleineren Image-Dateien, die Sie in diesen Tutorials speichern, die Größe des Dateisystems auf einer größeren Karte ändern, sodass der gesamte unterstützte Speicherplatz der von Ihnen ausgewählten Karte genutzt wird.

Optionale Ausstattung:

- Eine an den Raspberry Pi angeschlossene USB-Tastatur

- Ein HDMI-Monitor und ein Kabel zum Anschließen des Monitors an den Raspberry Pi

Verfahren in diesem Abschnitt:

- Herunterladen des Betriebssystems des Geräts auf die microSD-Karte
- Starten Ihres IoT-Geräts mit dem neuen Betriebssystem
- Verbinden Ihres Geräts mit Ihrem lokalen Host-Computer

Herunterladen des Betriebssystems des Geräts auf die microSD-Karte

Bei diesem Verfahren wird der lokale Host-Computer verwendet, um das Betriebssystem des Geräts auf eine microSD-Karte zu laden.

> **Note**
>
> Wenn Ihr Gerät kein Wechselspeichermedium für sein Betriebssystem verwendet, installieren Sie das Betriebssystem anhand des Verfahrens für dieses Gerät und fahren Sie fort mit the section called “Starten Sie Ihr IoT-Gerät”.

So installieren Sie das Betriebssystem auf Ihrem Raspberry Pi:

1. Laden Sie das Raspberry-Pi-Betriebssystem-Image, das Sie verwenden möchten, auf Ihren lokalen Host-Computer herunter und entpacken Sie es. Die neuesten Versionen sind erhältlich unter https://www.raspberrypi.com/software/operating-systems/

   Auswahl einer Version des Raspberry Pi OS

   In diesem Tutorial wird die Lite-Version des Raspberry Pi OS verwendet, da dies die kleinste Version ist, die die Tutorials in diesem Lernpfad unterstützt. Diese Version des Raspberry Pi OS hat nur eine Befehlszeilenschnittstelle und keine grafische Benutzeroberfläche. Eine Version des neuesten Raspberry-Pi-Betriebssystems mit grafischer Benutzeroberfläche kann im Rahmen dieser Tutorials auch verwendet werden. Die in diesem Lernpfad beschriebenen Verfahren verwenden jedoch nur die Befehlszeilenschnittstelle zur Durchführung von Vorgängen auf dem Raspberry Pi.
2. Legen Sie die microSD-Karte in den lokalen Host-Computer ein.
3. Schreiben Sie die entpackte OS-Image-Datei mit einem SD-Karten-Imaging-Tool auf die microSD-Karte.

4. Nach dem Schreiben des Raspberry-Pi-OS-Images auf die microSD-Karte:

    a. Öffnen Sie die BOOT-Partition auf der microSD-Karte in einem Befehlszeilenfenster oder einem Datei-Explorer-Fenster.

    b. Erstellen Sie in der BOOT-Partition der microSD-Karte im Stammverzeichnis eine leere Datei mit dem Namen `ssh` ohne Dateierweiterung und ohne Inhalt. Dadurch wird der Raspberry Pi angewiesen, die SSH-Kommunikation beim ersten Start zu aktivieren.

5. Werfen Sie die microSD-Karte aus und entfernen Sie sie sicher vom lokalen Host-Computer.

Ihre microSD-Karte ist nun einsatzbereit für the section called “Starten Sie Ihr IoT-Gerät”.

Starten Ihres IoT-Geräts mit dem neuen Betriebssystem

Dieses Verfahren installiert die microSD-Karte und startet Ihren Raspberry Pi zum ersten Mal mit dem heruntergeladenen Betriebssystem.

So starten Sie Ihr IoT-Gerät mit dem neuen Betriebssystem:

1. Trennen Sie das Gerät von der Stromversorgung und legen Sie die microSD-Karte aus dem vorherigen Schritt, the section called “Laden Sie das Betriebssystem”, in den Raspberry Pi ein.
2. Verbinden Sie das Gerät mit einem kabelgebundenen Netzwerk.
3. Diese Tutorials interagieren mit Ihrem Raspberry Pi von Ihrem lokalen Host-Computer aus über ein SSH-Terminal.

    Wenn Sie auch direkt mit dem Gerät interagieren möchten, können Sie:

    a. Einen HDMI-Monitor anschließen, um die Konsolenmeldungen des Raspberry Pi anzusehen, bevor Sie das Terminalfenster auf Ihrem lokalen Host-Computer mit Ihrem Raspberry Pi verbinden können

    b. Eine USB-Tastatur anschließen, wenn Sie direkt mit dem Raspberry Pi interagieren möchten

4. Einen Raspberry Pi an die Stromversorgung anschließen und etwa eine Minute warten, bis er initialisiert ist

    Wenn Sie einen Monitor an Ihren Raspberry Pi angeschlossen haben, können Sie den Startvorgang darauf verfolgen.

5. Finden Sie die IP-Adresse Ihres Geräts heraus:

- Wenn Sie einen HDMI-Monitor an den Raspberry Pi angeschlossen haben, erscheint die IP-Adresse in den auf dem Monitor angezeigten Meldungen
- Wenn Sie Zugriff auf den Router haben, mit dem Ihr Raspberry Pi verbunden ist, können Sie dessen Adresse in der Admin-Oberfläche des Routers sehen.

Wenn Sie die IP-Adresse Ihres Raspberry Pi haben, sind Sie bereit für the section called “Verbinden Sie Ihren Host-Computer”.

Verbinden Ihres Geräts mit Ihrem lokalen Host-Computer

Bei diesem Verfahren wird das Terminalprogramm auf Ihrem lokalen Host-Computer verwendet, um eine Verbindung zu Ihrem Raspberry Pi herzustellen und das Standardpasswort zu ändern.

So verbinden Sie Ihr Gerät mit Ihrem lokalen Host-Computer:

1. Öffnen Sie auf Ihrem lokalen Host-Computer das SSH-Terminalprogramm:

   - Windows: `PuTTY`
   - Linux/macOS: `Terminal`

   > **Note**
   >
   > PuTTY wird unter Windows nicht automatisch installiert. Wenn es sich nicht auf Ihrem Computer befindet, müssen Sie es möglicherweise herunterladen und installieren.

2. Verbinden Sie das Terminalprogramm mit der IP-Adresse Ihres Raspberry Pi und melden Sie sich mit den Standardanmeldeinformationen an.

   ```
   username: pi
   password: raspberry
   ```

3. Nachdem Sie sich bei Ihrem Raspberry Pi angemeldet haben, ändern Sie das Passwort für den Benutzer `pi`.

   ```
   passwd
   ```

   Folgen Sie den Eingabeaufforderungen, um das Passwort zu ändern.

```
Changing password for pi.
Current password: raspberry
New password: YourNewPassword
Retype new password: YourNewPassword
passwd: password updated successfully
```

Nachdem Sie die Befehlszeilenaufforderung des Raspberry Pi im Terminalfenster angezeigt und das Passwort geändert haben, können Sie mit the section called “Installieren und verifizieren Sie die erforderliche Software” fortfahren.

Installieren und verifizieren Sie die erforderliche Software auf Ihrem Gerät

Die Verfahren in diesem Abschnitt setzen die Verfahren aus dem vorherigen Abschnitt fort, um das Betriebssystem Ihres Raspberry Pi auf den neuesten Stand zu bringen und die Software auf dem Raspberry Pi zu installieren, die im nächsten Abschnitt zum Erstellen und Installieren des AWS IoT Device Clients verwendet wird.

Nachdem Sie diesen Abschnitt abgeschlossen haben, verfügt Ihr Raspberry Pi über ein aktuelles Betriebssystem, die Software, die für die Tutorials in diesem Lernpfad erforderlich ist, und er ist für Ihren Standort konfiguriert.

Erforderliche Ausstattung:

- Ihre lokale Entwicklungs- und Testumgebung aus dem vorherigen Abschnitt
- Der Raspberry Pi, den Sie im vorherigen Abschnitt verwendet haben
- Die microSD-Speicherkarte aus dem vorherigen Abschnitt

> **Note**
>
> Raspberry Pi Model 3+ und das Raspberry Pi Model 4 können alle in diesem Lernpfad beschriebenen Befehle ausführen. Wenn Ihr IoT-Gerät keine Software kompilieren oder ausführen kann AWS Command Line Interface, müssen Sie möglicherweise die erforderlichen Compiler auf Ihrem lokalen Host-Computer installieren, um die Software zu erstellen und sie dann auf Ihr IoT-Gerät zu übertragen. Weitere Informationen zum Installieren und Erstellen der Software für Ihr Gerät finden Sie in der Dokumentation für Ihre Gerätesoftware.

Verfahren in diesem Abschnitt:

- Aktualisieren Sie die Betriebssystemsoftware
- Installieren der erforderlichen Anwendungen und Bibliotheken
- (Optional) Speichern des microSD-Karten-Images

Aktualisieren Sie die Betriebssystemsoftware

Mit diesem Verfahren wird die Betriebssystemsoftware aktualisiert.

So aktualisieren Sie die Betriebssystemsoftware auf dem Raspberry Pi:

Führen Sie diese Schritte im Terminalfenster Ihres lokalen Host-Computers aus.

1. Geben Sie diese Befehle ein, um die Systemsoftware auf Ihrem Raspberry Pi zu aktualisieren.

   ```
   sudo apt-get -y update
   sudo apt-get -y upgrade
   sudo apt-get -y autoremove
   ```

2. Aktualisieren Sie die Gebietsschema- und Zeitzoneneinstellungen des Raspberry Pi (optional).

   Geben Sie diesen Befehl ein, um die Gebietsschema- und Zeitzoneneinstellungen des Geräts zu aktualisieren.

   ```
   sudo raspi-config
   ```

   a. So legen Sie das Gebietsschema des Geräts fest:

      i. Wählen Sie im Bildschirm Raspberry Pi Software Configuration Tool (raspi-config) Option 5.

      5 Localisation Options Configure language and regional settings

      Verwenden Sie die Tab-Taste, um <Select> zu bewegen, und drücken Sie dann die space bar.

      ii. Wählen Sie im Menü mit den Lokalisierungsoptionen die Option L1 aus.

      L1 Locale Configure language and regional settings

Verwenden Sie die Tab-Taste, um <Select> zu bewegen, und drücken Sie dann die space bar.

iii. Wählen Sie in der Liste der Gebietsschemaoptionen die Gebietsschemas aus, die Sie auf Ihrem Raspberry Pi installieren möchten, indem Sie mit den Pfeiltasten blättern und dann die gewünschten Gebietsschemas mit der space bar markieren.

In den Vereinigten Staaten ist **`en_US.UTF-8`** eine gute Wahl.

iv. Nachdem Sie die Gebietsschemas für Ihr Gerät ausgewählt haben, wählen Sie mit der Tab-Taste <OK>, und drücken Sie dann die space bar, um die Bestätigungsseite für die Konfiguration der Gebietsschemas aufzurufen.

b. So legen Sie die Zeitzone des Geräts fest:

i. Wählen Sie im Bildschirm raspi-config Option 5.

```
5 Localisation Options Configure language and regional settings
```

Verwenden Sie die Tab-Taste, um <Select> zu bewegen, und drücken Sie dann die space bar.

ii. Wählen Sie im Menü mit den Lokalisierungsoptionen mit der Pfeiltaste die Option L2 aus:

```
L2 time zone Configure time zone
```

Verwenden Sie die Tab-Taste, um <Select> zu bewegen, und drücken Sie dann die space bar.

iii. Wählen Sie im Menü Konfiguration von tzdata Ihr geografisches Gebiet aus der Liste aus.

Verwenden Sie die Tab-Taste, um zu <OK> zu gehen, und drücken Sie dann die space bar.

iv. Wählen Sie in der Liste der Städte mit den Pfeiltasten eine Stadt in Ihrer Zeitzone aus.

Um die Zeitzone festzulegen, verwenden Sie die Tab-Taste, um zu <OK> zu gehen, und drücken Sie dann die space bar.

c. Wenn Sie mit der Aktualisierung der Einstellungen fertig sind, wechseln Sie mit der Tab-Taste zu <Finish>, und drücken Sie dann auf space bar, um die App raspi-config zu schließen.

3. Geben Sie diesen Befehl ein, um Ihren Raspberry Pi neu zu starten.

```
sudo shutdown -r 0
```

4. Warten Sie, bis Ihr Raspberry Pi neu gestartet wird.
5. Nachdem Ihr Raspberry Pi neu gestartet wurde, verbinden Sie das Terminalfenster auf Ihrem lokalen Host-Computer erneut mit Ihrem Raspberry Pi.

Ihre Raspberry-Pi-Systemsoftware ist jetzt konfiguriert und Sie können mit the section called “Installieren Sie Anwendungen und Bibliotheken” fortfahren.

Installieren der erforderlichen Anwendungen und Bibliotheken

Mit diesem Verfahren werden die Anwendungssoftware und die Bibliotheken installiert, die in den nachfolgenden Tutorials verwendet werden.

Wenn Sie einen Raspberry Pi verwenden oder die erforderliche Software auf Ihrem IoT-Gerät kompilieren können, führen Sie diese Schritte im Terminalfenster auf Ihrem lokalen Host-Computer aus. Wenn Sie Software für Ihr IoT-Gerät auf Ihrem lokalen Host-Computer kompilieren müssen, lesen Sie in der Softwaredokumentation für Ihr IoT-Gerät nach, wie Sie diese Schritte auf Ihrem Gerät ausführen.

So installieren Sie die Anwendungssoftware und die Bibliotheken auf Ihrem Raspberry Pi:

1. Geben Sie diesen Befehl ein, um die Anwendungssoftware und die Bibliotheken zu installieren.

```
sudo apt-get -y install build-essential libssl-dev cmake unzip git python3-pip
```

2. Geben Sie diese Befehle ein, um zu bestätigen, dass die richtige Version der Software installiert wurde.

```
gcc --version
cmake --version
openssl version
git --version
```

3. Vergewissern Sie sich, dass diese Versionen der Anwendungssoftware installiert sind:

   - `gcc`: 9.3.0 oder höher
   - `cmake`: 3.10.x oder höher

- `OpenSSL`: 1.1.1 oder höher
- `git`: 2.20.1 oder höher

Wenn Ihr Raspberry Pi über akzeptable Versionen der erforderlichen Anwendungssoftware verfügt, können Sie mit the section called “(Optional) Speichern Sie das microSD-Image” fortfahren.

(Optional) Speichern des microSD-Karten-Images

In den Tutorials in diesem Lernpfad werden Sie auf diese Verfahren stoßen, um eine Kopie des microSD-Karten-Images des Raspberry Pi in einer Datei auf Ihrem lokalen Host-Computer zu speichern. Sie werden zwar empfohlen, sind aber keine Pflichtaufgaben. Wenn Sie das microSD-Karten-Image an der empfohlenen Stelle speichern, können Sie die Verfahren überspringen, die dem Speichern in diesem Lernpfad vorausgehen. Dies kann Zeit sparen, wenn Sie etwas erneut versuchen müssen. Wenn Sie das microSD-Karten-Image nicht regelmäßig speichern, müssen Sie die Tutorials im Lernpfad möglicherweise von Anfang an neu beginnen, wenn Ihre microSD-Karte beschädigt ist oder wenn Sie versehentlich eine App oder deren Einstellungen falsch konfigurieren.

Die microSD-Karte Ihres Raspberry Pi verfügt jetzt über ein aktualisiertes Betriebssystem und die grundlegende Anwendungssoftware ist geladen. Sie können die Zeit sparen, die Sie für die Ausführung der vorherigen Schritte benötigen, indem Sie den Inhalt der microSD-Karte jetzt in einer Datei speichern. Wenn Sie das aktuelle Image des microSD-Karten-Images Ihres Geräts haben, können Sie von diesem Punkt aus beginnen, um ein Tutorial oder einen Vorgang fortzusetzen oder erneut zu versuchen, ohne die Software von Grund auf neu installieren und aktualisieren zu müssen.

So speichern Sie das Image der microSD-Karte in einer Datei:

1. Geben Sie diesen Befehl ein, um den Raspberry Pi herunterzufahren.

   ```
   sudo shutdown -h 0
   ```

2. Nachdem der Raspberry Pi vollständig heruntergefahren ist, trennen Sie ihn von der Stromversorgung.
3. Entfernen Sie die microSD-Karte aus dem Raspberry Pi.
4. Auf Ihrem lokalen Host-Computer:

   a. Legen Sie die microSD-Karte ein.
   b. Speichern Sie das Image der microSD-Karte mithilfe des Imaging-Tools für die SD-Karte in eine Datei.

c. Nachdem das Image der microSD-Karte gespeichert wurde, werfen Sie die Karte aus dem lokalen Host-Computer aus.

5. Trennen Sie den Raspberry Pi von der Stromversorgung und legen Sie die microSD-Karte in den Raspberry Pi ein.
6. Schließen Sie den Raspberry Pi an die Stromversorgung an.
7. Nachdem Sie etwa eine Minute gewartet haben, verbinden Sie erneut das Terminalfenster auf dem lokalen Host-Computer, der mit Ihrem Raspberry Pi verbunden war, und melden Sie sich dann beim Raspberry Pi an.

Testen Sie Ihr Gerät und speichern Sie das Amazon CA-Zertifikat

Die Verfahren in diesem Abschnitt sind die gleichen wie im vorherigen Abschnitt zur Installation des Zertifikats der Zertifizierungsstelle AWS Command Line Interface und des Zertifikats der Zertifizierungsstelle, mit dem AWS IoT Core Ihre Verbindungen authentifiziert werden.

Nachdem Sie diesen Abschnitt abgeschlossen haben, wissen Sie, dass Ihr Raspberry Pi über die erforderliche Systemsoftware zur Installation des AWS IoT Geräteclients verfügt und dass er über eine funktionierende Verbindung zum Internet verfügt.

Erforderliche Ausstattung:

- Ihre lokale Entwicklungs- und Testumgebung aus dem vorherigen Abschnitt
- Der Raspberry Pi, den Sie im vorherigen Abschnitt verwendet haben
- Die microSD-Speicherkarte aus dem vorherigen Abschnitt

Verfahren in diesem Abschnitt:

- Installieren Sie den AWS Command Line Interface
- Konfiguriere deine AWS-Konto Anmeldedaten
- Herunterladen des Amazon-Root-CA-Zertifikats
- (Optional) Speichern des microSD-Karten-Images

Installieren Sie den AWS Command Line Interface

Dieses Verfahren installiert das AWS CLI auf Ihrem Raspberry Pi.

Wenn Sie einen Raspberry Pi verwenden oder Software auf Ihrem IoT-Gerät kompilieren können, führen Sie diese Schritte im Terminalfenster auf Ihrem lokalen Host-Computer aus. Wenn Sie Software für Ihr IoT-Gerät auf Ihrem lokalen Host-Computer kompilieren müssen, lesen Sie in der Softwaredokumentation für Ihr IoT-Gerät nach, welche Bibliotheken dafür erforderlich sind.

Um das zu installieren AWS CLI auf deinem Raspberry Pi

1. Führen Sie den folgenden Befehl aus, um die AWS CLI herunterzuladen und zu installieren.

```
export PATH=$PATH:~/.local/bin # configures the path to include the directory with
 the AWS CLI
git clone https://github.com/aws/aws-cli.git # download the AWS CLI code from
 GitHub
cd aws-cli && git checkout v2 # go to the directory with the repo and checkout
 version 2
pip3 install -r requirements.txt # install the prerequisite software
```

2. Führen Sie diesen Befehl aus, um das zu installieren AWS CLI. Die Ausführung dieses Befehls kann bis zu 15 Minuten dauern.

```
pip3 install . # install the AWS CLI
```

3. Führen Sie diesen Befehl aus, um zu bestätigen, dass die richtige Version von installiert AWS CLI wurde.

```
aws --version
```

   Die Version von AWS CLI sollte 2.2 oder höher sein.

Wenn die aktuelle Version AWS CLI angezeigt wurde, können Sie fortfahrenthe section called “Kontoanmeldedaten konfigurieren”.

Konfiguriere deine AWS-Konto Anmeldedaten

In diesem Verfahren erhalten Sie AWS-Konto Anmeldeinformationen und fügen sie zur Verwendung auf Ihrem Raspberry Pi hinzu.

Um deine hinzuzufügen AWS-Konto Anmeldeinformationen für Ihr Gerät

1. Besorgen Sie sich von Ihrem eine Zugangsschlüssel-ID und einen geheimen Zugriffsschlüssel AWS-Konto , um diese AWS CLI auf Ihrem Gerät zu authentifizieren.

   Wenn Sie AWS IAM noch nicht kennen, wird hier der Prozess https://aws.amazon.com/premiumsupport/knowledge-center/create-access-key/ beschrieben, der in der AWS Konsole ausgeführt wird, um AWS IAM-Anmeldeinformationen zu erstellen, die Sie auf Ihrem Gerät verwenden können.

2. Gehen Sie im Terminalfenster auf Ihrem lokalen Host-Computer, der mit Ihrem Raspberry Pi verbunden ist, mit den Anmeldeinformationen Zugriffsschlüssel-ID und Geheimer Zugriffsschlüssel für Ihr Gerät folgendermaßen vor:

   a. Führen Sie die AWS Configure-App mit diesem Befehl aus:

   ```
   aws configure
   ```

   b. Geben Sie Ihre Anmeldeinformationen und Konfigurationsinformationen ein, wenn Sie dazu aufgefordert werden:

   ```
   AWS Access Key ID: your Access Key ID
   AWS Secret Access Key: your Secret Access Key
   Default region name: your AWS-Region code
   Default output format: json
   ```

3. Führen Sie diesen Befehl aus, um den Zugriff Ihres Geräts auf Ihren AWS IoT Core Endpunkt AWS-Konto und zu testen.

   ```
   aws iot describe-endpoint --endpoint-type iot:Data-ATS
   ```

   Es sollte Ihren AWS-Konto spezifischen AWS IoT Datenendpunkt zurückgeben, wie zum Beispiel in diesem Beispiel:

   ```
   {
       "endpointAddress": "a3EXAMPLEffp-ats.iot.us-west-2.amazonaws.com"
   }
   ```

Wenn Sie Ihren AWS-Konto-spezifischen AWS IoT Datenendpunkt sehen, verfügt Ihr Raspberry Pi über die Konnektivität und die Berechtigungen, um fortzufahren. the section called “Herunterladen des Amazon-Root-CA-Zertifikats”

> **Important**
>
> Ihre AWS-Konto Anmeldeinformationen sind jetzt auf der microSD-Karte in Ihrem Raspberry Pi gespeichert. Dies erleichtert zwar zukünftige Interaktionen für Sie und die Software, die Sie in diesen Tutorials erstellen, aber sie werden standardmäßig auch in allen microSD-Kartenbildern gespeichert und dupliziert, die Sie nach diesem Schritt erstellen. AWS
> Um die Sicherheit Ihrer AWS-Konto Anmeldeinformationen zu schützen, sollten Sie erwägen, die Anmeldeinformationen zu löschen, bevor Sie weitere microSD-Karten-Images speichern, indem Sie den Vorgang `aws configure` erneut ausführen und zufällige Zeichen für die Zugriffsschlüssel-ID und den Secret Access Key eingeben, um zu verhindern, dass Ihre AWS-Konto Anmeldeinformationen kompromittiert werden.
> Wenn Sie feststellen, dass Sie Ihre AWS-Konto Anmeldeinformationen versehentlich gespeichert haben, können Sie sie in der IAM-Konsole deaktivieren. AWS

Herunterladen des Amazon-Root-CA-Zertifikats

Bei diesem Verfahren wird eine Kopie eines Zertifikats der Amazon Root Certificate Authority (CA) heruntergeladen und gespeichert. Durch das Herunterladen dieses Zertifikats wird es zur Verwendung in den nachfolgenden Tutorials gespeichert. Außerdem wird die Konnektivität Ihres Geräts mit AWS -Diensten getestet.

So laden Sie das Amazon-Root-CA-Zertifikat herunter und speichern es:

1. Führen Sie den folgenden Befehl aus, um ein Verzeichnis für das Zertifikat zu erstellen.

   ```
   mkdir ~/certs
   ```

2. Führen Sie diesen Befehl aus, um das Amazon-Root-CA-Zertifikat herunterzuladen.

   ```
   curl -o ~/certs/AmazonRootCA1.pem https://www.amazontrust.com/repository/
   AmazonRootCA1.pem
   ```

3. Führen Sie diese Befehle aus, um den Zugriff auf das Zertifikatsverzeichnis und die zugehörige Datei festzulegen.

```
chmod 745 ~
chmod 700 ~/certs
chmod 644 ~/certs/AmazonRootCA1.pem
```

4. Führen Sie diesen Befehl aus, um die CA-Zertifikatsdatei im neuen Verzeichnis anzuzeigen.

   ```
   ls -l ~/certs
   ```

   Sie sollten einen Eintrag wie diesen sehen. Das Datum und die Uhrzeit werden unterschiedlich sein. Die Dateigröße und alle anderen Informationen sollten jedoch mit den hier abgebildeten übereinstimmen.

   ```
   -rw-r--r-- 1 pi pi 1188 Oct 28 13:02 AmazonRootCA1.pem
   ```

   Wenn die Dateigröße nicht 1188 beträgt, überprüfen Sie die curl-Befehlsparameter. Möglicherweise haben Sie eine falsche Datei heruntergeladen.

(Optional) Speichern des microSD-Karten-Images

Die microSD-Karte Ihres Raspberry Pi verfügt jetzt über ein aktualisiertes Betriebssystem und die grundlegende Anwendungssoftware ist geladen.

So speichern Sie das Image der microSD-Karte in einer Datei:

1. Löschen Sie im Terminalfenster auf Ihrem lokalen Host-Computer Ihre AWS -Anmeldeinformationen.

   a. Führen Sie die AWS Configure-App mit diesem Befehl aus:

      ```
      aws configure
      ```

   b. Ersetzen Sie Ihre Anmeldeinformationen, wenn Sie dazu aufgefordert werden. Sie können den Standardregionsnamen und das Standardausgabeformat unverändert lassen, indem Sie die Eingabetaste drücken.

      ```
      AWS Access Key ID [****************YT2H]: XYXYXYXYX
      AWS Secret Access Key [****************9plH]: XYXYXYXYX
      Default region name [us-west-2]:
      ```

```
Default output format [json]:
```

2. Geben Sie diesen Befehl ein, um den Raspberry Pi herunterzufahren.

```
sudo shutdown -h 0
```

3. Nachdem der Raspberry Pi vollständig heruntergefahren ist, trennen Sie ihn von der Stromversorgung.
4. Entfernen Sie die microSD-Karte aus Ihrem Gerät.
5. Auf Ihrem lokalen Host-Computer:
   a. Legen Sie die microSD-Karte ein.
   b. Speichern Sie das Image der microSD-Karte mithilfe des Imaging-Tools für die SD-Karte in eine Datei.
   c. Nachdem das Image der microSD-Karte gespeichert wurde, werfen Sie die Karte aus dem lokalen Host-Computer aus.
6. Trennen Sie den Raspberry Pi von der Stromversorgung und legen Sie die microSD-Karte in den Raspberry Pi ein.
7. Schließen Sie das Gerät an die Stromversorgung an.
8. Starten Sie nach etwa einer Minute auf dem lokalen Host-Computer die Terminalfenstersitzung neu und melden Sie sich beim Gerät an.

   Geben Sie Ihre AWS-Konto Anmeldeinformationen noch nicht erneut ein.

Nachdem Sie Ihren Raspberry Pi neu gestartet und sich angemeldet haben, können Sie mit the section called “Installation und Konfiguration des IoT-Geräteclients” fortfahren.

Tutorial: Installation und Konfiguration des AWS IoT Geräte-Client

Dieses Tutorial führt Sie durch die Installation und Konfiguration des AWS IoT Device Clients und die Erstellung von AWS IoT Ressourcen, die Sie in dieser und anderen Demos verwenden werden.

So beginnen Sie dieses Tutorial:

- Halten Sie Ihren lokalen Host-Computer und den Raspberry Pi aus dem vorherigen Tutorial bereit.

Dieses Tutorial kann bis zu 90 Minuten dauern.

Wenn Sie mit diesem Thema fertig sind:

- Ihr IoT-Gerät kann dann in anderen AWS IoT Device Client-Demos verwendet werden.
- Sie werden Ihr IoT-Gerät in bereitgestellt haben. AWS IoT Core
- Sie haben den AWS IoT Device Client heruntergeladen und auf Ihrem Gerät installiert.
- Sie haben ein Image der microSD-Karte Ihres Geräts gespeichert, das in nachfolgenden Tutorials verwendet werden kann.

Erforderliche Ausstattung:

- Ihre lokale Entwicklungs- und Testumgebung aus dem vorherigen Abschnitt
- Der Raspberry Pi, den Sie im vorherigen Abschnitt verwendet haben
- Die microSD-Speicherkarte des Raspberry Pi, die Sie im vorherigen Abschnitt verwendet haben

Verfahren in diesem Tutorial

- Laden Sie das herunter und speichern Sie es AWS IoT Geräte-Client
- Stellen Sie Ihren Raspberry Pi bereit AWS IoT
- Konfigurieren Sie das AWS IoT Device Client zum Testen der Konnektivität

Laden Sie das herunter und speichern Sie es AWS IoT Geräte-Client

Mit den Verfahren in diesem Abschnitt wird der AWS IoT Device Client heruntergeladen, kompiliert und auf Ihrem Raspberry Pi installiert. Nachdem Sie die Installation getestet haben, können Sie das Image der microSD-Karte des Raspberry Pi speichern, um es später zu verwenden, wenn Sie die Tutorials erneut ausprobieren möchten.

Verfahren in diesem Abschnitt:

- Laden Sie das herunter und erstellen Sie es AWS IoT Geräte-Client
- Erstellen der in den Tutorials verwendeten Verzeichnisse
- (Optional) Speichern des microSD-Karten-Images

Laden Sie das herunter und erstellen Sie es AWS IoT Geräte-Client

Dieses Verfahren installiert den AWS IoT Device Client auf Ihrem Raspberry Pi.

Führen Sie diese Befehle im Terminalfenster auf Ihrem lokalen Host-Computer aus, der mit Ihrem Raspberry Pi verbunden ist.

Um den zu installieren AWS IoT Device Client auf Ihrem Raspberry Pi

1. Geben Sie diese Befehle ein, um den AWS IoT Device Client herunterzuladen und auf Ihrem Raspberry Pi zu erstellen.

   ```
   cd ~
   git clone https://github.com/awslabs/aws-iot-device-client aws-iot-device-client
   mkdir ~/aws-iot-device-client/build && cd ~/aws-iot-device-client/build
   cmake ../
   ```

2. Führen Sie diesen Befehl aus, um den AWS IoT Device Client zu erstellen. Die Ausführung dieses Befehls kann bis zu 15 Minuten dauern.

   ```
   cmake --build . --target aws-iot-device-client
   ```

   Die Warnmeldungen, die beim Kompilieren des AWS IoT Geräteclients angezeigt werden, können ignoriert werden.

   Diese Tutorials wurden mit dem AWS IoT Device Client getestetgcc, der auf Version (Raspbian 10.2.1-6+rpi1) 10.2.1 20210110 auf der Version vom 30. Oktober 2021 von Raspberry Pi OS (Bullseye) basiert, Version (Raspbian 8.3.0-6+rpi1) 8.3.0 auf gcc der Version vom 7. Mai 2021 des Raspberry Pi OS (Buster).

3. Nachdem AWS IoT der Device Client mit der Erstellung fertig ist, testen Sie ihn, indem Sie diesen Befehl ausführen.

   ```
   ./aws-iot-device-client --help
   ```

Wenn Sie die Befehlszeilenhilfe für den AWS IoT Device Client sehen, wurde der AWS IoT Device Client erfolgreich erstellt und kann von Ihnen verwendet werden.

Erstellen der in den Tutorials verwendeten Verzeichnisse

Mit diesem Verfahren werden die Verzeichnisse auf dem Raspberry Pi erstellt, in denen die in den Tutorials in diesem Lernpfad verwendeten Dateien gespeichert werden.

So erstellen Sie die in den Tutorials in diesem Lernpfad verwendeten Verzeichnisse:

1. Führen Sie diese Befehle aus, um die erforderlichen Verzeichnisse zu erstellen.

```
mkdir ~/dc-configs
mkdir ~/policies
mkdir ~/messages
mkdir ~/certs/testconn
mkdir ~/certs/pubsub
mkdir ~/certs/jobs
```

2. Führen Sie diese Befehle aus, um die Berechtigungen für die neuen Verzeichnisse festzulegen.

```
chmod 745 ~
chmod 700 ~/certs/testconn
chmod 700 ~/certs/pubsub
chmod 700 ~/certs/jobs
```

Nachdem Sie diese Verzeichnisse erstellt und ihre Berechtigungen festgelegt haben, fahren Sie fort mit the section called “(Optional) Speichern des microSD-Karten-Images”.

(Optional) Speichern des microSD-Karten-Images

Zu diesem Zeitpunkt verfügt die microSD-Karte Ihres Raspberry Pi über ein aktualisiertes Betriebssystem, die grundlegende Anwendungssoftware und den AWS IoT Device Client.

Wenn Sie diese Übungen und Tutorials erneut ausprobieren möchten, können Sie die vorherigen Verfahren überspringen, indem Sie das mit diesem Verfahren gespeicherte microSD-Karten-Image auf eine neue microSD-Karte schreiben und mit den Tutorials ab the section called “Stellen Sie Ihren Raspberry Pi bereit” fortfahren.

So speichern Sie das Image der microSD-Karte in einer Datei:

Gehen Sie im Terminalfenster auf Ihrem lokalen Host-Computer, der mit Ihrem Raspberry Pi verbunden ist, folgendermaßen vor:

1. Bestätigen Sie, dass Ihre AWS-Konto Anmeldeinformationen nicht gespeichert wurden.

   a. Führen AWS Sie die Configure-App mit diesem Befehl aus:

```
aws configure
```

b. Wenn Ihre Anmeldeinformationen gespeichert wurden (also in der Eingabeaufforderung angezeigt werden), geben Sie die Zeichenfolge **XYXYXYXYX** ein, wenn Sie dazu aufgefordert werden, wie hier gezeigt. Lassen Sie den Namen der Standardregion und das Standardausgabeformat leer.

```
AWS Access Key ID [****************YXYX]: XYXYXYXYX
AWS Secret Access Key [****************YXYX]: XYXYXYXYX
Default region name:
Default output format:
```

2. Geben Sie diesen Befehl ein, um den Raspberry Pi herunterzufahren.

```
sudo shutdown -h 0
```

3. Nachdem der Raspberry Pi vollständig heruntergefahren ist, trennen Sie ihn von der Stromversorgung.
4. Entfernen Sie die microSD-Karte aus Ihrem Gerät.
5. Auf Ihrem lokalen Host-Computer:

   a. Legen Sie die microSD-Karte ein.
   b. Speichern Sie das Image der microSD-Karte mithilfe des Imaging-Tools für die SD-Karte in eine Datei.
   c. Nachdem das Image der microSD-Karte gespeichert wurde, werfen Sie die Karte aus dem lokalen Host-Computer aus.

Sie können mit dieser microSD-Karte in the section called “Stellen Sie Ihren Raspberry Pi bereit” fortfahren.

Stellen Sie Ihren Raspberry Pi bereit AWS IoT

Die Verfahren in diesem Abschnitt beginnen mit dem gespeicherten microSD-Image, auf dem der AWS CLI und AWS IoT Device Client installiert ist, und erstellen die AWS IoT Ressourcen und Gerätezertifikate, auf denen Ihr Raspberry Pi bereitgestellt wird. AWS IoT

Installieren der microSD-Karte in Ihrem Raspberry Pi

Dieses Verfahren installiert die microSD-Karte mit der erforderlichen Software, die auf dem Raspberry Pi geladen und konfiguriert ist, und konfiguriert Ihre, AWS-Konto sodass Sie mit den Tutorials in diesem Lernpfad fortfahren können.

Verwenden Sie eine microSD-Karte aus the section called “(Optional) Speichern des microSD-Karten-Images” mit der erforderlichen Software für die Übungen und Tutorials in diesem Lernpfad.

So installieren Sie die microSD-Karte in Ihrem Raspberry Pi:

1. Trennen Sie den Raspberry Pi von der Stromversorgung und legen Sie die microSD-Karte in den Raspberry Pi ein.
2. Schließen Sie den Raspberry Pi an die Stromversorgung an.
3. Starten Sie nach etwa einer Minute auf dem lokalen Host-Computer die Terminalfenstersitzung neu und melden Sie sich beim Raspberry Pi an.
4. Auf Ihrem lokalen Host-Computer, im Terminalfenster und mit den Anmeldeinformationen Zugriffsschlüssel-ID und Geheimer Zugriffsschlüssel für Ihren Raspberry Pi:

   a. Führen Sie die AWS Konfigurations-App mit diesem Befehl aus:

   ```
   aws configure
   ```

   b. Geben Sie Ihre AWS-Konto Anmeldeinformationen und Konfigurationsinformationen ein, wenn Sie dazu aufgefordert werden:

   ```
   AWS Access Key ID [****************YXYX]: your Access Key ID
   AWS Secret Access Key [****************YXYX]: your Secret Access Key
   Default region name [us-west-2]: your AWS-Region code
   Default output format [json]: json
   ```

Nachdem Sie Ihre AWS-Konto Anmeldeinformationen wiederhergestellt haben, können Sie fortfahrenthe section called “Stellen Sie Ihr Gerät bereit in AWS IoT Core”.

Stellen Sie Ihr Gerät bereit in AWS IoT Core

Die Verfahren in diesem Abschnitt erstellen die AWS IoT Ressourcen, die Ihren Raspberry Pi bereitstellen AWS IoT. Bei der Erstellung dieser Ressourcen werden Sie aufgefordert, verschiedene Informationen aufzuzeichnen. Diese Informationen werden im nächsten Verfahren von der Konfiguration des AWS IoT Geräteclients verwendet.

Damit Ihr Raspberry Pi funktioniert AWS IoT, muss er bereitgestellt werden. Bei der Bereitstellung werden die AWS IoT Ressourcen erstellt und konfiguriert, die zur Unterstützung Ihres Raspberry Pi als IoT-Gerät erforderlich sind.

Wenn Ihr Raspberry Pi eingeschaltet ist und neu gestartet wird, verbinden Sie das Terminalfenster auf Ihrem lokalen Host-Computer mit dem Raspberry Pi und führen Sie diese Verfahren durch.

Verfahren in diesem Abschnitt:

- Erstellen und Herunterladen der Gerätezertifikatsdateien
- Create AWS IoT Ressourcen

Erstellen und Herunterladen der Gerätezertifikatsdateien

Mit diesem Verfahren werden die Gerätezertifikatsdateien für diese Demo erstellt.

So erstellen Sie die Gerätezertifikatsdateien für Ihren Raspberry Pi und laden sie herunter:

1. Geben Sie im Terminalfenster auf Ihrem lokalen Host-Computer diese Befehle ein, um die Gerätezertifikatsdateien für Ihr Gerät zu erstellen.

   ```
   mkdir ~/certs/testconn
   aws iot create-keys-and-certificate \
   --set-as-active \
   --certificate-pem-outfile "~/certs/testconn/device.pem.crt" \
   --public-key-outfile "~/certs/testconn/public.pem.key" \
   --private-key-outfile "~/certs/testconn/private.pem.key"
   ```

   Die Ausgabe des Befehls ähnelt der folgenden: Notieren Sie sich den *certificateArn*-Wert zur späteren Verwendung.

   ```
   {
       "certificateArn": "arn:aws:iot:us-
   west-2:57EXAMPLE833:cert/76e7e4edb3e52f52334be2f387a06145b2aa4c7fcd810f3aea2d92abc227d269",
       "certificateId":
    "76e7e4edb3e52f5233EXAMPLE7a06145b2aa4c7fcd810f3aea2d92abc227d269",
       "certificatePem": "-----BEGIN CERTIFICATE-----
   \nMIIDWTCCAkGgAwIBAgI_SHORTENED_FOR_EXAMPLE_Lgn4jfgtS\n-----END CERTIFICATE-----
   \n",
       "keyPair": {
           "PublicKey": "-----BEGIN PUBLIC KEY-----
   \nMIIBIjANBgkqhkiG9w0BA_SHORTENED_FOR_EXAMPLE_ImwIDAQAB\n-----END PUBLIC KEY-----
   \n",
           "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----
   \nMIIEowIBAAKCAQE_SHORTENED_FOR_EXAMPLE_T9RoDiukY\n-----END RSA PRIVATE KEY-----\n"
       }
   ```

```
}
```

2. Geben Sie die folgenden Befehle ein, um die Berechtigungen für das Zertifikatsverzeichnis und seine Dateien festzulegen.

```
chmod 745 ~
chmod 700 ~/certs/testconn
chmod 644 ~/certs/testconn/*
chmod 600 ~/certs/testconn/private.pem.key
```

3. Führen Sie diesen Befehl aus, um die Berechtigungen für Ihre Zertifikatsverzeichnisse und -dateien zu überprüfen.

```
ls -l ~/certs/testconn
```

   Der Befehl sollte ähnliche Werte wie die hier angezeigten ausgeben, mit der Ausnahme, dass Datum und Uhrzeit unterschiedlich sein werden.

```
-rw-r--r-- 1 pi pi 1220 Oct 28 13:02 device.pem.crt
-rw------- 1 pi pi 1675 Oct 28 13:02 private.pem.key
-rw-r--r-- 1 pi pi  451 Oct 28 13:02 public.pem.key
```

Sie haben die Gerätezertifikatsdateien bereits auf Ihrem Raspberry Pi installiert und können mit the section called “Create AWS IoT Ressourcen” fortfahren.

Create AWS IoT Ressourcen

Bei diesem Verfahren wird Ihr Gerät bereitgestellt, AWS IoT indem die Ressourcen erstellt werden, die Ihr Gerät für den Zugriff auf AWS IoT Funktionen und Dienste benötigt.

Um Ihr Gerät bereitzustellen in AWS IoT

1. Geben Sie im Terminalfenster auf Ihrem lokalen Host-Computer den folgenden Befehl ein, um die Adresse des Gerätedatenendpunkts für Ihr AWS-Konto abzurufen.

```
aws iot describe-endpoint --endpoint-type IoT:Data-ATS
```

   Der Befehl aus den vorhergehenden Schritten gibt eine Antwort aus, die der folgenden ähnelt: Notieren Sie sich den *endpointAddress*-Wert zur späteren Verwendung.

```
{
    "endpointAddress": "a3qjEXAMPLEffp-ats.iot.us-west-2.amazonaws.com"
}
```

2. Geben Sie diesen Befehl ein, um eine AWS IoT Ding-Ressource für Ihren Raspberry Pi zu erstellen.

```
aws iot create-thing --thing-name "DevCliTestThing"
```

Wenn Ihre AWS IoT Ding-Ressource erstellt wurde, gibt der Befehl eine Antwort wie diese zurück.

```
{
    "thingName": "DevCliTestThing",
    "thingArn": "arn:aws:iot:us-west-2:57EXAMPLE833:thing/DevCliTestThing",
    "thingId": "8ea78707-32c3-4f8a-9232-14bEXAMPLEfd"
}
```

3. Im Terminalfenster:

   a. Öffnen Sie einen Texteditor, z. B. `nano`.

   b. Kopieren Sie dieses JSON-Richtliniendokument und fügen Sie es in Ihren geöffneten Texteditor ein.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Subscribe",
                "iot:Receive",
                "iot:Connect"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
```

```
}
```

> **Note**
>
> Dieses Richtliniendokument gewährt allen Ressourcen großzügig die Erlaubnis, Verbindungen herzustellen, zu empfangen, zu veröffentlichen und zu abonnieren. Normalerweise gewähren Richtlinien nur bestimmten Ressourcen die Erlaubnis, bestimmte Aktionen auszuführen. Beim ersten Test der Gerätekonnektivität wird diese zu allgemeine und großzügige Richtlinie jedoch verwendet, um die Wahrscheinlichkeit eines Zugriffsproblems während dieses Tests zu minimieren. In den nachfolgenden Tutorials werden enger abgegrenzte Richtliniendokumente verwendet, um bessere Verfahren für die Richtliniengestaltung zu demonstrieren.

c. Speichern Sie die Datei als **~/policies/dev_cli_test_thing_policy.json** in Ihrem Texteditor.

4. Führen Sie diesen Befehl aus, um das Richtliniendokument aus den vorherigen Schritten zum Erstellen einer AWS IoT Richtlinie zu verwenden.

```
aws iot create-policy \
--policy-name "DevCliTestThingPolicy" \
--policy-document "file://~/policies/dev_cli_test_thing_policy.json"
```

Wenn die Richtlinie erstellt wurde, gibt der Befehl eine Antwort wie die folgende zurück:

```
{
    "policyName": "DevCliTestThingPolicy",
    "policyArn": "arn:aws:iot:us-west-2:57EXAMPLE833:policy/DevCliTestThingPolicy",
    "policyDocument": "{\n    \"Version\": \"2012-10-17\",       \n    \"Statement
\": [\n        {\n            \"Effect\": \"Allow\",\n            \"Action\": [\n
                \"iot:Publish\",\n                \"iot:Subscribe\",\n
    \"iot:Receive\",\n                \"iot:Connect\"\n            ],\n
 \"Resource\": [\n                \"*\"\n            ]\n        }\n    ]\n}\n",
    "policyVersionId": "1"
}
```

5. Führen Sie diesen Befehl aus, um die Richtlinie an das Gerätezertifikat anzufügen. Ersetzen Sie *certificateArn* durch den `certificateArn`-Wert, den Sie zuvor gespeichert haben.

```
aws iot attach-policy \
--policy-name "DevCliTestThingPolicy" \
--target "certificateArn"
```

Bei erfolgreicher Ausführung gibt dieser Befehl nichts zurück.

6. Führen Sie diesen Befehl aus, um das Gerätezertifikat an die AWS IoT Ding-Ressource anzuhängen. Ersetzen Sie *certificateArn* durch den `certificateArn`-Wert, den Sie zuvor gespeichert haben.

```
aws iot attach-thing-principal \
--thing-name "DevCliTestThing" \
--principal "certificateArn"
```

Bei erfolgreicher Ausführung gibt dieser Befehl nichts zurück.

Nachdem Sie Ihr Gerät erfolgreich bereitgestellt haben AWS IoT, können Sie fortfahrenthe section called “Konfigurieren Sie den Geräteclient und testen Sie die Konnektivität”.

Konfigurieren Sie das AWS IoT Device Client zum Testen der Konnektivität

Die Verfahren in diesem Abschnitt konfigurieren den AWS IoT Device Client so, dass er eine MQTT-Nachricht von Ihrem Raspberry Pi veröffentlicht.

Verfahren in diesem Abschnitt:

- Erstellen der Konfigurationsdatei
- Öffnen des MQTT-Testclients
- Ausführen AWS IoT Geräte-Client

Erstellen der Konfigurationsdatei

Dieses Verfahren erstellt die Konfigurationsdatei zum Testen des AWS IoT Geräteclients.

Um die Konfigurationsdatei zum Testen von zu erstellen AWS IoT Geräte-Client

- Gehen Sie im Terminalfenster auf Ihrem lokalen Host-Computer, der mit Ihrem Raspberry Pi verbunden ist, folgendermaßen vor:

a. Geben Sie diese Befehle ein, um ein Verzeichnis für die Konfigurationsdateien zu erstellen und die Berechtigungen für das Verzeichnis festzulegen:

```
mkdir ~/dc-configs
chmod 745 ~/dc-configs
```

b. Öffnen Sie einen Texteditor, z. B. `nano`.

c. Kopieren Sie dieses JSON-Dokument und fügen Sie es in Ihren geöffneten Texteditor ein.

```
{
  "endpoint": "a3qEXAMPLEaffp-ats.iot.us-west-2.amazonaws.com",
  "cert": "~/certs/testconn/device.pem.crt",
  "key": "~/certs/testconn/private.pem.key",
  "root-ca": "~/certs/AmazonRootCA1.pem",
  "thing-name": "DevCliTestThing",
  "logging": {
    "enable-sdk-logging": true,
    "level": "DEBUG",
    "type": "STDOUT",
    "file": ""
  },
  "jobs": {
    "enabled": false,
    "handler-directory": ""
  },
  "tunneling": {
    "enabled": false
  },
  "device-defender": {
    "enabled": false,
    "interval": 300
  },
  "fleet-provisioning": {
    "enabled": false,
    "template-name": "",
    "template-parameters": "",
    "csr-file": "",
    "device-key": ""
  },
  "samples": {
    "pub-sub": {
      "enabled": true,
```

```
      "publish-topic": "test/dc/pubtopic",
      "publish-file": "",
      "subscribe-topic": "test/dc/subtopic",
      "subscribe-file": ""
    }
  },
  "config-shadow": {
    "enabled": false
  },
  "sample-shadow": {
    "enabled": false,
    "shadow-name": "",
    "shadow-input-file": "",
    "shadow-output-file": ""
  }
}
```

d. Ersetzen Sie den *endpoint* Wert durch den Gerätedatenendpunkt für Ihren AWS-Konto , den Sie gefunden habenthe section called “Stellen Sie Ihr Gerät bereit in AWS IoT Core”.

e. Speichern Sie die Datei als **~/dc-configs/dc-testconn-config.json** in Ihrem Texteditor.

f. Führen Sie diesen Befehl aus, um die Berechtigungen für die neue Konfigurationsdatei festzulegen.

```
chmod 644 ~/dc-configs/dc-testconn-config.json
```

Nachdem Sie die Datei gespeichert haben, können Sie mit the section called “Öffnen des MQTT-Testclients” fortfahren.

Öffnen des MQTT-Testclients

Dieses Verfahren bereitet den MQTT-Testclient in der AWS IoT Konsole darauf vor, die MQTT-Nachricht zu abonnieren, die der AWS IoT Device Client veröffentlicht, wenn er ausgeführt wird.

Zur Vorbereitung der MQTT-Testclient um alle MQTT-Nachrichten zu abonnieren

1. Wählen Sie auf Ihrem lokalen Host-Computer in der AWS IoT -Konsole MQTT-Testclient aus.
2. Geben Sie auf der Registerkarte Thema abonnieren unter Themenfilter **#** (ein einzelnes Rautenzeichen) ein und wählen Sie Abonnieren aus, um alle MQTT-Themen zu abonnieren.

3. Vergewissern Sie sich, dass Sie unter der Bezeichnung Abonnements **#** (ein einzelnes Pfundzeichen) sehen.

Lassen Sie das Fenster mit dem MQTT-Testclient geöffnet, während Sie mit the section called “Ausführen AWS IoT Geräte-Client” fortfahren.

Ausführen AWS IoT Geräte-Client

Dieses Verfahren führt den AWS IoT Geräteclient so aus, dass er eine einzelne MQTT-Nachricht veröffentlicht, die der MQTT-Testclient empfängt und anzeigt.

Um eine MQTT-Nachricht von der zu senden AWS IoT Geräte-Client

1. Stellen Sie sicher, dass sowohl das Terminalfenster, das mit Ihrem Raspberry Pi verbunden ist, als auch das Fenster mit dem MQTT-Testclient sichtbar sind, während Sie dieses Verfahren ausführen.
2. Geben Sie im Terminalfenster diese Befehle ein, um den AWS IoT Device Client mithilfe der in erstellten Konfigurationsdatei auszuführenthe section called “Erstellen der Konfigurationsdatei”.

   ```
   cd ~/aws-iot-device-client/build
   ./aws-iot-device-client --config-file ~/dc-configs/dc-testconn-config.json
   ```

   Im Terminalfenster zeigt der AWS IoT Device Client Informationsmeldungen und alle Fehler an, die bei der Ausführung auftreten.

   Wenn im Terminalfenster keine Fehler angezeigt werden, überprüfen Sie den MQTT-Testclient.
3. Im MQTT-Testclient finden Sie im Abonnementfenster die Nachricht Hallo Welt!, die an das `test/dc/pubtopic`-Nachrichtenthema gesendet wurde.
4. Wenn der AWS IoT Geräteclient keine Fehler anzeigt und Sie Hello World! sehen an die `test/dc/pubtopic` Nachricht im MQTT-Testclient gesendet, haben Sie eine erfolgreiche Verbindung nachgewiesen.
5. Geben Sie im Terminalfenster **^C** (Ctrl-C) ein, um den AWS IoT Device Client zu beenden.

Nachdem Sie nachgewiesen haben, dass der AWS IoT Device Client auf Ihrem Raspberry Pi korrekt läuft und mit ihm kommunizieren kann AWS IoT, können Sie mit dem fortfahrenthe section called “Kommunizieren Sie mit dem Geräteclient über MQTT”.

Tutorial: Demonstrieren Sie die MQTT-Nachrichtenkommunikation mit dem AWS IoT Geräte-Client

Dieses Tutorial zeigt, wie der AWS IoT Device Client MQTT-Nachrichten abonnieren und veröffentlichen kann, die häufig in IoT-Lösungen verwendet werden.

So beginnen Sie dieses Tutorial:

- Sie haben Ihren lokalen Host-Computer und den Raspberry Pi so konfiguriert, wie im vorherigen Abschnitt zu sehen.

  Wenn Sie das microSD-Karten-Image nach der Installation des AWS IoT Device Clients gespeichert haben, können Sie eine microSD-Karte mit diesem Image mit Ihrem Raspberry Pi verwenden.
- Wenn Sie diese Demo schon einmal ausgeführt haben, überprüfen ??? Sie, ob alle AWS IoT Ressourcen gelöscht werden, die Sie in früheren Durchläufen erstellt haben, um doppelte Ressourcenfehler zu vermeiden.

Für dieses Tutorial brauchen Sie ungefähr 45 Minuten.

Wenn Sie mit diesem Thema fertig sind:

- Sie haben verschiedene Möglichkeiten demonstriert, wie Ihr IoT-Gerät MQTT-Nachrichten von abonnieren AWS IoT und MQTT-Nachrichten veröffentlichen kann. AWS IoT

Erforderliche Ausstattung:

- Ihre lokale Entwicklungs- und Testumgebung aus dem vorherigen Abschnitt
- Der Raspberry Pi, den Sie im vorherigen Abschnitt verwendet haben
- Die microSD-Speicherkarte des Raspberry Pi, die Sie im vorherigen Abschnitt verwendet haben

Verfahren in diesem Tutorial

- Bereiten Sie den Raspberry Pi vor, um die MQTT-Nachrichtenkommunikation zu demonstrieren
- Demonstrieren Sie das Veröffentlichen von Nachrichten mit dem AWS IoT Geräte-Client
- Demonstrieren Sie das Abonnieren von Nachrichten mit dem AWS IoT Geräte-Client

Bereiten Sie den Raspberry Pi vor, um die MQTT-Nachrichtenkommunikation zu demonstrieren

Dieses Verfahren erstellt die Ressourcen im AWS IoT und im Raspberry Pi, um die MQTT-Nachrichtenkommunikation mit dem AWS IoT Device Client zu demonstrieren.

Verfahren in diesem Abschnitt:

- Erstellen der Zertifikatsdateien für die Demonstration der MQTT-Kommunikation
- Bereitstellen Ihres Geräts für die Demonstration der MQTT-Kommunikation
- Konfigurieren Sie das AWS IoT Device Client-Konfigurationsdatei und MQTT-Testclient zur Demonstration der MQTT-Kommunikation

Erstellen der Zertifikatsdateien für die Demonstration der MQTT-Kommunikation

Mit diesem Verfahren werden die Gerätezertifikatsdateien für diese Demo erstellt.

So erstellen Sie die Gerätezertifikatsdateien für Ihren Raspberry Pi und laden sie herunter:

1. Geben Sie im Terminalfenster auf Ihrem lokalen Host-Computer den folgenden Befehl ein, um die Gerätezertifikatsdateien für Ihr Gerät zu erstellen.

   ```
   mkdir ~/certs/pubsub
   aws iot create-keys-and-certificate \
   --set-as-active \
   --certificate-pem-outfile "~/certs/pubsub/device.pem.crt" \
   --public-key-outfile "~/certs/pubsub/public.pem.key" \
   --private-key-outfile "~/certs/pubsub/private.pem.key"
   ```

   Die Ausgabe des Befehls ähnelt der folgenden: Speichern Sie den *certificateArn*-Wert zur späteren Verwendung.

   ```
   {
   "certificateArn": "arn:aws:iot:us-
   west-2:57EXAMPLE833:cert/76e7e4edb3e52f52334be2f387a06145b2aa4c7fcd810f3aea2d92abc227d269",
   "certificateId":
    "76e7e4edb3e52f5233EXAMPLE7a06145b2aa4c7fcd810f3aea2d92abc227d269",
   "certificatePem": "-----BEGIN CERTIFICATE-----
   \nMIIDWTCCAkGgAwIBAgI_SHORTENED_FOR_EXAMPLE_Lgn4jfgtS\n-----END CERTIFICATE-----
   \n",
   ```

```
"keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BA_SHORTENED_FOR_EXAMPLE_ImwIDAQAB\n-----END PUBLIC KEY-----
\n",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----
\nMIIEowIBAAKCAQE_SHORTENED_FOR_EXAMPLE_T9RoDiukY\n-----END RSA PRIVATE KEY-----\n"
}
}
```

2. Geben Sie die folgenden Befehle ein, um die Berechtigungen für das Zertifikatsverzeichnis und seine Dateien festzulegen.

```
chmod 700 ~/certs/pubsub
chmod 644 ~/certs/pubsub/*
chmod 600 ~/certs/pubsub/private.pem.key
```

3. Führen Sie diesen Befehl aus, um die Berechtigungen für Ihre Zertifikatsverzeichnisse und -dateien zu überprüfen.

```
ls -l ~/certs/pubsub
```

   Der Befehl sollte ähnliche Werte wie die hier angezeigten ausgeben, mit der Ausnahme, dass Datum und Uhrzeit unterschiedlich sein werden.

```
-rw-r--r-- 1 pi pi 1220 Oct 28 13:02 device.pem.crt
-rw------- 1 pi pi 1675 Oct 28 13:02 private.pem.key
-rw-r--r-- 1 pi pi  451 Oct 28 13:02 public.pem.key
```

4. Geben Sie diese Befehle ein, um die Verzeichnisse für die Protokolldateien zu erstellen.

```
mkdir ~/.aws-iot-device-client
mkdir ~/.aws-iot-device-client/log
chmod 745 ~/.aws-iot-device-client/log
echo " " > ~/.aws-iot-device-client/log/aws-iot-device-client.log
echo " " > ~/.aws-iot-device-client/log/pubsub_rx_msgs.log
chmod 600 ~/.aws-iot-device-client/log/*
```

Bereitstellen Ihres Geräts für die Demonstration der MQTT-Kommunikation

In diesem Abschnitt werden die AWS IoT Ressourcen erstellt, die Ihren Raspberry Pi bereitstellen. AWS IoT

Um Ihr Gerät bereitzustellen in AWS IoT:

1. Geben Sie im Terminalfenster auf Ihrem lokalen Host-Computer den folgenden Befehl ein, um die Adresse des Gerätedatenendpunkts für Ihr AWS-Konto abzurufen.

   ```
   aws iot describe-endpoint --endpoint-type IoT:Data-ATS
   ```

   Der Endpunktwert hat sich seit der letzten Ausführung dieses Befehls im vorhergehenden Tutorial nicht geändert. Wenn Sie den Befehl hier erneut ausführen, können Sie den Datenendpunktwert leicht finden und in die in diesem Tutorial verwendete Konfigurationsdatei einfügen.

   Der Befehl aus den vorhergehenden Schritten gibt eine Antwort aus, die der folgenden ähnelt: Notieren Sie sich den *endpointAddress*-Wert zur späteren Verwendung.

   ```
   {
   "endpointAddress": "a3qjEXAMPLEffp-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. Geben Sie diesen Befehl ein, um eine neue Ressource AWS IoT für Ihren Raspberry Pi zu erstellen.

   ```
   aws iot create-thing --thing-name "PubSubTestThing"
   ```

   Da eine AWS IoT Ding-Ressource eine virtuelle Repräsentation Ihres Geräts in der Cloud ist, können wir mehrere Ressourcen erstellen, AWS IoT um sie für verschiedene Zwecke zu verwenden. Sie können alle von demselben physischen IoT-Gerät verwendet werden, um verschiedene Aspekte des Geräts darzustellen.

   In diesen Tutorials wird jeweils nur eine Objektressource verwendet, um den Raspberry Pi darzustellen. Auf diese Weise stellen sie in diesen Tutorials die verschiedenen Demos dar, sodass Sie, nachdem Sie die AWS IoT Ressourcen für eine Demo erstellt haben, die Demo mit den Ressourcen wiederholen können, die Sie speziell für die jeweilige Demo erstellt haben.

Wenn Ihre AWS IoT Ding-Ressource erstellt wurde, gibt der Befehl eine Antwort wie diese zurück.

```
{
"thingName": "PubSubTestThing",
"thingArn": "arn:aws:iot:us-west-2:57EXAMPLE833:thing/PubSubTestThing",
"thingId": "8ea78707-32c3-4f8a-9232-14bEXAMPLEfd"
}
```

3. Im Terminalfenster:

   a. Öffnen Sie einen Texteditor, z. B. nano.

   b. Kopieren Sie dieses JSON-Dokument und fügen Sie es in Ihren geöffneten Texteditor ein.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:client/PubSubTestThing"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/test/dc/pubtopic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
```

```
                "arn:aws:iot:us-west-2:123456789012:topicfilter/test/dc/
subtopic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/test/dc/subtopic"
            ]
        }
    ]
}
```

c. Ersetzen Sie im Editor in jedem `Resource` Abschnitt des Richtliniendokuments *us-west-2:57EXAMPLE833* durch Ihren AWS-Region, einen Doppelpunkt (:) und Ihre 12-stellige AWS-Konto Zahl.

d. Speichern Sie die Datei als **~/policies/pubsub_test_thing_policy.json** in Ihrem Texteditor.

4. Führen Sie diesen Befehl aus, um das Richtliniendokument aus den vorherigen Schritten zum Erstellen einer AWS IoT Richtlinie zu verwenden.

```
aws iot create-policy \
--policy-name "PubSubTestThingPolicy" \
--policy-document "file://~/policies/pubsub_test_thing_policy.json"
```

Wenn die Richtlinie erstellt wurde, gibt der Befehl eine Antwort wie die folgende zurück:

```
{
                                    "policyName": "PubSubTestThingPolicy",
                                    "policyArn": "arn:aws:iot:us-
west-2:57EXAMPLE833:policy/PubSubTestThingPolicy",
                                    "policyDocument": "{\n\"Version\":
 \"2012-10-17\",       \n\"Statement\": [\n{\n\"Effect\": \"Allow\",\n
\"Action\": [\n\"iot:Connect\"\n],\n\"Resource\": [\n\"arn:aws:iot:us-
west-2:57EXAMPLE833:client/PubSubTestThing\"\n]\n},\n{\n\"Effect\": \"Allow
\",\n\"Action\": [\n\"iot:Publish\"\n],\n\"Resource\": [\n\"arn:aws:iot:us-
west-2:57EXAMPLE833:topic/test/dc/pubtopic\"\n]\n},\n{\n\"Effect\": \"Allow
\",\n\"Action\": [\n\"iot:Subscribe\"\n],\n\"Resource\": [\n\"arn:aws:iot:us-
```

```
west-2:57EXAMPLE833:topicfilter/test/dc/subtopic\"\n]\n},\n{\n\"Effect\": \"Allow
\",\n\"Action\": [\n\"iot:Receive\"\n],\n\"Resource\": [\n\"arn:aws:iot:us-
west-2:57EXAMPLE833:topic/test/dc/*\"\n]\n}\n]\n}\n",
                                    "policyVersionId": "1"
                                    }
```

5. Führen Sie diesen Befehl aus, um die Richtlinie an das Gerätezertifikat anzufügen. Ersetzen Sie *certificateArn* durch den `certificateArn`-Wert, den Sie in diesem Abschnitt bereits gespeichert haben.

   ```
   aws iot attach-policy \
   --policy-name "PubSubTestThingPolicy" \
   --target "certificateArn"
   ```

   Bei erfolgreicher Ausführung gibt dieser Befehl nichts zurück.

6. Führen Sie diesen Befehl aus, um das Gerätezertifikat an die AWS IoT -Objektressource anzuhängen. Ersetzen Sie *certificateArn* durch den `certificateArn`-Wert, den Sie in diesem Abschnitt bereits gespeichert haben.

   ```
   aws iot attach-thing-principal \
   --thing-name "PubSubTestThing" \
   --principal "certificateArn"
   ```

   Bei erfolgreicher Ausführung gibt dieser Befehl nichts zurück.

Nachdem Sie Ihr Gerät erfolgreich bereitgestellt haben AWS IoT, können Sie fortfahrenthe section called “Konfigurieren Sie die Device Client-Konfigurationsdatei und den MQTT-Client”.

Konfigurieren Sie das AWS IoT Device Client-Konfigurationsdatei und MQTT-Testclient zur Demonstration der MQTT-Kommunikation

Dieses Verfahren erstellt eine Konfigurationsdatei, um den AWS IoT Device Client zu testen.

Um die Konfigurationsdatei zu erstellen, um das zu testen AWS IoT Geräte-Client

1. Gehen Sie im Terminalfenster auf Ihrem lokalen Host-Computer, der mit Ihrem Raspberry Pi verbunden ist, folgendermaßen vor:

   a. Öffnen Sie einen Texteditor, z. B. `nano`.

   b. Kopieren Sie dieses JSON-Dokument und fügen Sie es in Ihren geöffneten Texteditor ein.

```
{
  "endpoint": "a3qEXAMPLEaffp-ats.iot.us-west-2.amazonaws.com",
  "cert": "~/certs/pubsub/device.pem.crt",
  "key": "~/certs/pubsub/private.pem.key",
  "root-ca": "~/certs/AmazonRootCA1.pem",
  "thing-name": "PubSubTestThing",
  "logging": {
    "enable-sdk-logging": true,
    "level": "DEBUG",
    "type": "STDOUT",
    "file": ""
  },
  "jobs": {
    "enabled": false,
    "handler-directory": ""
  },
  "tunneling": {
    "enabled": false
  },
  "device-defender": {
    "enabled": false,
    "interval": 300
  },
  "fleet-provisioning": {
    "enabled": false,
    "template-name": "",
    "template-parameters": "",
    "csr-file": "",
    "device-key": ""
  },
  "samples": {
    "pub-sub": {
      "enabled": true,
      "publish-topic": "test/dc/pubtopic",
      "publish-file": "",
      "subscribe-topic": "test/dc/subtopic",
      "subscribe-file": "~/.aws-iot-device-client/log/pubsub_rx_msgs.log"
    }
  },
  "config-shadow": {
    "enabled": false
  },
  "sample-shadow": {
```

```
    "enabled": false,
    "shadow-name": "",
    "shadow-input-file": "",
    "shadow-output-file": ""
  }
}
```

c. Ersetzen Sie den *endpoint* Wert durch den Gerätedatenendpunkt für Ihren AWS-Konto , den Sie gefunden habenthe section called “Stellen Sie Ihr Gerät bereit in AWS IoT Core”.

d. Speichern Sie die Datei als **~/dc-configs/dc-pubsub-config.json** in Ihrem Texteditor.

e. Führen Sie diesen Befehl aus, um die Berechtigungen für die neue Konfigurationsdatei festzulegen.

```
chmod 644 ~/dc-configs/dc-pubsub-config.json
```

2. So bereiten Sie den MQTT-Testclient darauf vor, alle MQTT-Nachrichten zu abonnieren:

   a. Wählen Sie auf Ihrem lokalen Host-Computer in der AWS IoT -Konsole MQTT-Testclient aus.

   b. Geben Sie auf der Registerkarte Thema abonnieren unter Themenfilter **#** (ein einzelnes Rautenzeichen) ein und wählen Sie Abonnieren aus.

   c. Vergewissern Sie sich, dass Sie unter der Bezeichnung Abonnements **#** (ein einzelnes Pfundzeichen) sehen.

   Lassen Sie das Fenster mit dem MQTT-Testclient geöffnet, während Sie mit diesem Tutorial fortfahren.

Nachdem Sie die Konfigurationsdatei gespeichert und den MQTT-Testclient konfiguriert haben, können Sie mit the section called “Veröffentlichen Sie Nachrichten mit dem IoT Device Client” fortfahren.

Demonstrieren Sie das Veröffentlichen von Nachrichten mit dem AWS IoT Geräte-Client

Die Verfahren in diesem Abschnitt zeigen, wie der AWS IoT Device Client Standard- und benutzerdefinierte MQTT-Nachrichten senden kann.

Die Richtlinienanweisungen in der Richtlinie, die Sie im vorherigen Schritt für diese Übungen erstellt haben, geben dem Raspberry Pi die Erlaubnis, die folgenden Aktionen auszuführen:

- **`iot:Connect`**

  Weist den Client mit dem Namen Ihres Raspberry Pi`PubSubTestThing`, auf dem der AWS IoT Device Client ausgeführt wird, an, eine Verbindung herzustellen.

  ```
  {
    "Effect": "Allow",
    "Action": [
      "iot:Connect"
    ],
    "Resource": [
      "arn:aws:iot:us-west-2:57EXAMPLE833:client/PubSubTestThing"
    ]
  }
  ```

- **`iot:Publish`**

  Erteilt dem Raspberry Pi die Erlaubnis, Nachrichten mit dem MQTT-Thema von `test/dc/pubtopic` zu veröffentlichen.

  ```
  {
    "Effect": "Allow",
    "Action": [
      "iot:Publish"
    ],
    "Resource": [
      "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/pubtopic"
    ]
  }
  ```

  Die `iot:Publish`-Aktion erteilt die Erlaubnis zu Veröffentlichungen zu den MQTT-Themen, die im Resourcen-Array aufgeführt sind. Der Inhalt dieser Nachrichten wird nicht durch die Richtlinienanweisung geregelt.

Veröffentlichen Sie die Standardnachricht mit dem AWS IoT Geräte-Client

Dieses Verfahren führt den AWS IoT Geräteclient so aus, dass er eine einzelne Standard-MQTT-Nachricht veröffentlicht, die der MQTT-Testclient empfängt und anzeigt.

Um die Standard-MQTT-Nachricht von der AWS IoT Geräte-Client

1. Stellen Sie sicher, dass sowohl das Terminalfenster auf Ihrem lokalen Host-Computer, das mit dem Raspberry Pi verbunden ist, als auch das Fenster mit dem MQTT-Testclient sichtbar sind, während Sie dieses Verfahren ausführen.
2. Geben Sie im Terminalfenster diese Befehle ein, um den AWS IoT Device Client mithilfe der in erstellten Konfigurationsdatei auszuführenthe section called “Erstellen der Konfigurationsdatei”.

   ```
   cd ~/aws-iot-device-client/build
   ./aws-iot-device-client --config-file ~/dc-configs/dc-pubsub-config.json
   ```

   Im Terminalfenster zeigt der AWS IoT Device Client Informationsmeldungen und alle Fehler an, die bei der Ausführung auftreten.

   Wenn im Terminalfenster keine Fehler angezeigt werden, überprüfen Sie den MQTT-Testclient.
3. Im MQTT-Testclient finden Sie im Fenster Abonnements die Nachricht Hallo Welt!, die an das `test/dc/pubtopic`-Nachrichtenthema gesendet wurde.
4. Wenn der AWS IoT Geräteclient keine Fehler anzeigt und Sie Hello World! sehen an die `test/dc/pubtopic` Nachricht im MQTT-Testclient gesendet, haben Sie eine erfolgreiche Verbindung nachgewiesen.
5. Geben Sie im Terminalfenster **^C** (Ctrl-C) ein, um den AWS IoT Device Client zu beenden.

Nachdem Sie nachgewiesen haben, dass der AWS IoT Device Client die Standard-MQTT-Nachricht veröffentlicht hat, können Sie mit dem the section called “Veröffentlichen Sie eine benutzerdefinierte MQTT-Nachricht” fortfahren.

Veröffentlichen Sie eine benutzerdefinierte Nachricht mit dem AWS IoT Geräte-Client

Die Verfahren in diesem Abschnitt erstellen eine benutzerdefinierte MQTT-Nachricht und führen dann den AWS IoT Device Client aus, sodass er die benutzerdefinierte MQTT-Nachricht einmal veröffentlicht, damit der MQTT-Testclient sie empfangen und anzeigen kann.

Erstellen Sie eine benutzerdefinierte MQTT-Nachricht für AWS IoT Geräte-Client

Führen Sie diese Schritte im Terminalfenster auf Ihrem lokalen Host-Computer aus, der mit Ihrem Raspberry Pi verbunden ist.

Um eine benutzerdefinierte Nachricht für den zu erstellen AWS IoT Zu veröffentlichender Geräte-Client

1. Öffnen Sie im Terminalfenster einen Texteditor Ihrer Wahl, z. B. `nano`.
2. Kopieren Sie im Texteditor das folgende JSON-Dokument und fügen Sie es ein: Dies ist die Nutzlast der MQTT-Nachricht, die der AWS IoT Geräteclient veröffentlicht.

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

3. Speichern Sie den Inhalt des Texteditors als **`~/messages/sample-ws-message.json`**.
4. Geben Sie den folgenden Befehl ein, um die Berechtigungen der Nachrichtendatei festzulegen, die Sie gerade erstellt haben.

```
chmod 600 ~/messages/*
```

Um eine Konfigurationsdatei für den zu erstellen AWS IoT Geräte-Client, der zum Senden der benutzerdefinierten Nachricht verwendet werden soll

1. Öffnen Sie im Terminalfenster in einem Texteditor wie `nano` z. B. die vorhandene AWS IoT Device Client-Konfigurationsdatei:**`~/dc-configs/dc-pubsub-config.json`**.
2. Bearbeiten Sie das `samples`-Objekt so, dass es folgendermaßen aussieht. Kein anderer Teil dieser Datei muss geändert werden.

```
  "samples": {
    "pub-sub": {
      "enabled": true,
      "publish-topic": "test/dc/pubtopic",
      "publish-file": "~/messages/sample-ws-message.json",
      "subscribe-topic": "test/dc/subtopic",
      "subscribe-file": "~/.aws-iot-device-client/log/pubsub_rx_msgs.log"
```

3. Speichern Sie den Inhalt des Texteditors als **`~/dc-configs/dc-pubsub-custom-config.json`**.
4. Führen Sie diesen Befehl aus, um die Berechtigungen für die neue Konfigurationsdatei festzulegen.

```
chmod 644 ~/dc-configs/dc-pubsub-custom-config.json
```

Veröffentlichen Sie die benutzerdefinierte MQTT-Nachricht mit dem AWS IoT Geräte-Client

Diese Änderung wirkt sich nur auf den Inhalt der MQTT-Nachrichtennutzlast aus, sodass die aktuelle Richtlinie weiterhin funktioniert. Wenn jedoch das MQTT-Thema (wie durch den `publish-topic`-Wert in `~/dc-configs/dc-pubsub-custom-config.json` definiert) geändert wurde, muss auch die `iot::Publish`-Richtlinienanweisung geändert werden, damit der Raspberry Pi unter dem neuen MQTT-Thema veröffentlichen kann.

Um die MQTT-Nachricht von AWS IoT Geräte-Client

1. Stellen Sie sicher, dass sowohl das Terminalfenster als auch das Fenster mit dem MQTT-Testclient sichtbar sind, während Sie dieses Verfahren ausführen. Stellen Sie außerdem sicher, dass Ihr MQTT-Testclient weiterhin den #-Themenfilter abonniert hat. Falls nicht, abonnieren Sie den #-Themenfilter erneut.
2. Geben Sie im Terminalfenster diese Befehle ein, um den AWS IoT Device Client mithilfe der in the section called “Erstellen der Konfigurationsdatei” erstellten Konfigurationsdatei auszuführen.

```
cd ~/aws-iot-device-client/build
./aws-iot-device-client --config-file ~/dc-configs/dc-pubsub-custom-config.json
```

   Im Terminalfenster zeigt der AWS IoT Device Client Informationsmeldungen und alle Fehler an, die bei der Ausführung auftreten.

   Wenn im Terminalfenster keine Fehler angezeigt werden, überprüfen Sie den MQTT-Testclient.
3. Im MQTT-Testclient finden Sie im Fenster Abonnements die benutzerdefinierte Nachrichtennutzlast „Hallo Welt!“, die an das `test/dc/pubtopic`-Nachrichtenthema gesendet wurde.
4. Wenn der AWS IoT Device Client keine Fehler anzeigt und Sie die benutzerdefinierte Nachrichtennutzlast sehen, die Sie für die `test/dc/pubtopic` Nachricht im MQTT-Testclient veröffentlicht haben, haben Sie eine benutzerdefinierte Nachricht erfolgreich veröffentlicht.

5. Geben Sie im Terminalfenster **^C** (Ctrl-C) ein, um den AWS IoT Device Client zu beenden.

Nachdem Sie nachgewiesen haben, dass der AWS IoT Device Client eine benutzerdefinierte Nachrichtennutzlast veröffentlicht hat, können Sie fortfahrenthe section called “Abonnieren Sie Nachrichten mit dem IoT Device Client”.

Demonstrieren Sie das Abonnieren von Nachrichten mit dem AWS IoT Geräte-Client

In diesem Abschnitt werden Sie zwei Arten von Nachrichtenabonnements demonstrieren:

- Abonnement für ein einzelnes Thema
- Wild-card Thema-Abonnement

Die Richtlinienanweisungen in der Richtlinie, die Sie für diese Übungen erstellt haben, geben dem Raspberry Pi die Erlaubnis, die folgenden Aktionen auszuführen:

- **`iot:Receive`**

  Erteilt dem AWS IoT Device Client die Erlaubnis, MQTT-Themen zu empfangen, die den im `Resource` Objekt genannten Themen entsprechen.

  ```
  {
    "Effect": "Allow",
    "Action": [
      "iot:Receive"
    ],
    "Resource": [
      "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/subtopic"
    ]
  }
  ```

- **`iot:Subscribe`**

  Erteilt dem AWS IoT Geräteclient die Berechtigung, MQTT-Themenfilter zu abonnieren, die den `Resource` im Objekt genannten Filtern entsprechen.

  ```
  {
    "Effect": "Allow",
    "Action": [
      "iot:Subscribe"
  ```

```
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/subtopic"
      ]
    }
```

Abonnieren eines einzelnen MQTT-Nachrichtenthemas

Dieses Verfahren zeigt, wie der AWS IoT Device Client MQTT-Nachrichten abonnieren und protokollieren kann.

Listen Sie im Terminalfenster auf Ihrem lokalen Host-Computer, der mit Ihrem Raspberry Pi verbunden ist, den Inhalt der Datei von **`~/dc-configs/dc-pubsub-custom-config.json`** auf oder öffnen Sie die Datei in einem Texteditor, um den Inhalt zu überprüfen. Suchen Sie das `samples`-Objekt, das wie folgt aussehen sollte.

```
"samples": {
  "pub-sub": {
    "enabled": true,
    "publish-topic": "test/dc/pubtopic",
    "publish-file": "~/messages/sample-ws-message.json",
    "subscribe-topic": "test/dc/subtopic",
    "subscribe-file": "~/.aws-iot-device-client/log/pubsub_rx_msgs.log"
```

Beachten Sie, dass der `subscribe-topic`-Wert das MQTT-Thema ist, das der AWS IoT Device Client bei der Ausführung abonniert. Der AWS IoT Device Client schreibt die Nachrichten-Payloads, die er von diesem Abonnement erhält, in die `subscribe-file` im Wert angegebene Datei.

Um ein MQTT-Nachrichtenthema von der AWS IoT Geräte-Client

1. Stellen Sie sicher, dass sowohl das Terminalfenster als auch das Fenster mit dem MQTT-Testclient sichtbar sind, während Sie dieses Verfahren ausführen. Stellen Sie außerdem sicher, dass Ihr MQTT-Testclient weiterhin den #-Themenfilter abonniert hat. Falls nicht, abonnieren Sie den #-Themenfilter erneut.
2. Geben Sie im Terminalfenster diese Befehle ein, um den AWS IoT Device Client mithilfe der in erstellten Konfigurationsdatei auszuführenthe section called “Erstellen der Konfigurationsdatei”.

   ```
   cd ~/aws-iot-device-client/build
   ./aws-iot-device-client --config-file ~/dc-configs/dc-pubsub-custom-config.json
   ```

Im Terminalfenster zeigt der AWS IoT Device Client Informationsmeldungen und alle Fehler an, die bei der Ausführung auftreten.

Wenn im Terminalfenster keine Fehler angezeigt werden, fahren Sie in der AWS IoT -Konsole fort.

3. Wählen Sie in der AWS IoT Konsole im MQTT-Testclient die Registerkarte In einem Thema veröffentlichen.
4. Geben Sie in das Feld Name des Themas **`test/dc/subtopic`** ein.
5. Überprüfen Sie unter Nachrichtennutzlast den Nachrichteninhalt.
6. Wählen Sie Veröffentlichen, um die MQTT-Nachricht zu veröffentlichen.
7. Achten Sie im Terminalfenster auf den Eintrag „Nachricht erhalten“ vom AWS IoT Device Client, der wie folgt aussieht.

```
2021-11-10T16:02:20.890Z [DEBUG] {samples/PubSubFeature.cpp}: Message received on
 subscribe topic, size: 45 bytes
```

8. Wenn Sie den Eintrag „Nachricht erhalten“ sehen, aus dem hervorgeht, dass die Nachricht empfangen wurde, geben Sie **`^C`** (Ctrl-C) ein, um den AWS IoT Device Client zu beenden.
9. Geben Sie diesen Befehl ein, um das Ende der Nachrichtenprotokolldatei und die Nachricht zu sehen, die Sie vom MQTT-Testclient aus veröffentlicht haben.

```
tail ~/.aws-iot-device-client/log/pubsub_rx_msgs.log
```

Wenn Sie die Nachricht in der Protokolldatei angesehen haben, haben Sie nachgewiesen, dass der AWS IoT Device Client die von Ihnen über den MQTT-Testclient veröffentlichte Nachricht erhalten hat.

Abonnieren mehrerer MQTT-Nachrichtenthemen mit Platzhalterzeichen

Diese Verfahren zeigen, wie der AWS IoT Device Client MQTT-Nachrichten mithilfe von Platzhalterzeichen abonnieren und protokollieren kann. Gehen Sie dazu wie folgt vor:

1. Aktualisieren Sie den Themenfilter, den der AWS IoT Geräteclient zum Abonnieren von MQTT-Themen verwendet.
2. Aktualisieren Sie die vom Gerät verwendete Richtlinie, um die neuen Abonnements zuzulassen.

3. Führen Sie den AWS IoT Device Client aus und veröffentlichen Sie Nachrichten von der MQTT-Testkonsole aus.

So erstellen Sie eine Konfigurationsdatei, um mehrere MQTT-Nachrichtenthemen mithilfe eines MQTT-Themenfilters mit Platzhaltern zu abonnieren:

1. Öffnen Sie im Terminalfenster auf Ihrem lokalen Host-Computer, der mit Ihrem Raspberry Pi verbunden ist, **`~/dc-configs/dc-pubsub-custom-config.json`**, um das `samples`-Objekt zu finden und zu bearbeiten.
2. Suchen Sie im Texteditor das `samples`-Objekt und aktualisieren Sie den `subscribe-topic`-Wert so, dass er folgendermaßen aussieht:

   ```
   "samples": {
     "pub-sub": {
       "enabled": true,
       "publish-topic": "test/dc/pubtopic",
       "publish-file": "~/messages/sample-ws-message.json",
       "subscribe-topic": "test/dc/#",
       "subscribe-file": "~/.aws-iot-device-client/log/pubsub_rx_msgs.log"
   ```

   Der neue `subscribe-topic`-Wert ist ein MQTT-Themenfilter mit einem MQTT-Platzhalterzeichen am Ende. Dies beschreibt ein Abonnement für alle MQTT-Themen, die mit `test/dc/` beginnen. Der AWS IoT Device Client schreibt die Nachrichten-Payloads, die er von diesem Abonnement erhält, in die angegebene Datei. `subscribe-file`
3. Speichern Sie die geänderte Konfigurationsdatei als **`~/dc-configs/dc-pubsub-wild-config.json`** und schließen Sie den Editor.

So ändern Sie die von Ihrem Raspberry Pi verwendete Richtlinie, um das Abonnieren und Empfangen mehrerer MQTT-Nachrichtenthemen zu ermöglichen:

1. Öffnen Sie im Terminalfenster auf Ihrem lokalen Host-Computer, der mit Ihrem Raspberry Pi verbunden ist, in Ihrem bevorzugten Texteditor **`~/policies/pubsub_test_thing_policy.json`** zur Bearbeitung und suchen Sie dann die `iot::Subscribe`- und `iot::Receive`-Richtlinienanweisungen in der Datei.
2. Aktualisieren Sie in der `iot::Subscribe`-Richtlinienanweisung die Zeichenfolge im Ressourcenobjekt, indem Sie `subtopic` durch `*` ersetzten, sodass sie wie folgt aussieht:

```
{
  "Effect": "Allow",
  "Action": [
    "iot:Subscribe"
  ],
  "Resource": [
    "arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/*"
  ]
}
```

> **Note**
>
> Die Platzhalterzeichen für den MQTT-Themenfilter sind das + (Pluszeichen) und das # (Pfundzeichen). Eine Abonnementanfrage mit # am Ende abonniert alle Themen, die mit der Zeichenfolge beginnen, die dem #-Zeichen vorausgeht (zum Beispiel `test/dc/` in diesem Fall).
> Der Ressourcenwert in der Richtlinienanweisung, die dieses Abonnement autorisiert, muss jedoch * (ein Sternchen) anstelle von # (ein Pfundzeichen) im Themenfilter ARN verwenden. Das liegt daran, dass der Richtlinienprozessor ein anderes Platzhalterzeichen verwendet als MQTT.
> Weitere Informationen zur Verwendung von Platzhalterzeichen für Themen und Themenfilter in Richtlinien finden Sie unter Verwendung von Platzhalterzeichen in MQTT und AWS IoT Core Richtlinien.

3. Aktualisieren Sie in der `iot::Receive`-Richtlinienanweisung die Zeichenfolge im Ressourcenobjekt, indem Sie `subtopic` durch * ersetzten, sodass sie wie folgt aussieht:

```
{
  "Effect": "Allow",
  "Action": [
    "iot:Receive"
  ],
  "Resource": [
    "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/*"
  ]
}
```

4. Speichern Sie das aktualisierte Richtliniendokument unter **`~/policies/pubsub_wild_test_thing_policy.json`** und schließen Sie den Editor.

5. Geben Sie diesen Befehl ein, um die Richtlinie für dieses Tutorial zu aktualisieren und die neuen Ressourcendefinitionen zu verwenden.

```
aws iot create-policy-version \
--set-as-default \
--policy-name "PubSubTestThingPolicy" \
--policy-document "file://~/policies/pubsub_wild_test_thing_policy.json"
```

Bei Erfolg gibt der Befehl ein Ergebnis wie dieses zurück: Beachten Sie, dass `policyVersionId` jetzt 2 ist, was darauf hindeutet, dass es sich um die zweite Version dieser Richtlinie handelt.

Wenn Sie die Richtlinie erfolgreich aktualisiert haben, können Sie mit dem nächsten Verfahren fortfahren.

```
{
    "policyArn": "arn:aws:iot:us-west-2:57EXAMPLE833:policy/PubSubTestThingPolicy",
    "policyDocument": "{\n  \"Version\": \"2012-10-17\",       \n  \"Statement
\": [\n    {\n      \"Effect\": \"Allow\",\n      \"Action\": [\n
 \"iot:Connect\"\n      ],\n      \"Resource\": [\n        \"arn:aws:iot:us-
west-2:57EXAMPLE833:client/PubSubTestThing\"\n      ]\n    },\n    {\n
 \"Effect\": \"Allow\",\n      \"Action\": [\n        \"iot:Publish\"\n      ],
\n      \"Resource\": [\n        \"arn:aws:iot:us-west-2:57EXAMPLE833:topic/
test/dc/pubtopic\"\n      ]\n    },\n    {\n      \"Effect\": \"Allow\",\n
 \"Action\": [\n        \"iot:Subscribe\"\n      ],\n      \"Resource\": [\n
  \"arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/*\"\n      ]\n    },\n
   {\n      \"Effect\": \"Allow\",\n      \"Action\": [\n        \"iot:Receive\"\n
     ],\n      \"Resource\": [\n        \"arn:aws:iot:us-west-2:57EXAMPLE833:topic/
test/dc/*\"\n      ]\n    }\n  ]\n}\n",
    "policyVersionId": "2",
    "isDefaultVersion": true
}
```

Wenn Sie die Fehlermeldung erhalten, dass zu viele Richtlinienversionen vorhanden sind, um eine neue zu speichern, geben Sie diesen Befehl ein, um die aktuellen Versionen der Richtlinie aufzulisten. Suchen Sie in der Liste, die dieser Befehl zurückgibt, nach einer Richtlinienversion, die Sie löschen können.

```
aws iot list-policy-versions --policy-name "PubSubTestThingPolicy"
```

Geben Sie diesen Befehl ein, um eine nicht mehr benötigte Version zu löschen. Beachten Sie, dass Sie die Standardversion der Richtlinie nicht löschen können. Die Standardversion der Richtlinie hat einen `isDefaultVersion`-Wert von `true`.

```
aws iot delete-policy-version \
--policy-name "PubSubTestThingPolicy" \
--policy-version-id policyId
```

Versuchen Sie diesen Schritt erneut, nachdem Sie eine Richtlinienversion gelöscht haben.

Mit der aktualisierten Konfigurationsdatei und Richtlinie sind Sie bereit, Wildcard-Abonnements mit dem AWS IoT Device Client zu demonstrieren.

Um zu demonstrieren, wie AWS IoT Device Client abonniert und empfängt mehrere MQTT-Nachrichtenthemen

1. Überprüfen Sie im MQTT-Testclient die Abonnements. Wenn der MQTT-Testclient den **#**-Themenfilter abonniert hat, fahren Sie mit dem nächsten Schritt fort. Falls nicht, geben Sie im MQTT-Testclient auf der Registerkarte Thema abonnieren im Themenfilter **#** (ein Pfundzeichen) ein und wählen Sie dann Abonnieren aus, um es zu abonnieren.
2. Gehen Sie im Terminalfenster auf Ihrem lokalen Host-Computer, der mit Ihrem Raspberry Pi verbunden ist, diese Befehle ein, um den AWS IoT Device Client zu starten.

   ```
   cd ~/aws-iot-device-client/build
   ./aws-iot-device-client --config-file ~/dc-configs/dc-pubsub-wild-config.json
   ```

3. Kehren Sie zum MQTT-Testclient zurück, während Sie die AWS IoT Device Client-Ausgabe im Terminalfenster auf dem lokalen Host-Computer beobachten. Geben Sie auf der Registerkarte In einem Thema veröffentlichen unter Themenname **`test/dc/subtopic`** ein, und wählen Sie dann Veröffentlichen aus.
4. Vergewissern Sie sich im Terminalfenster, dass die Nachricht empfangen wurde, indem Sie nach einer Nachricht suchen, wie z. B.:

   ```
   2021-11-10T16:34:20.101Z [DEBUG] {samples/PubSubFeature.cpp}: Message received on
    subscribe topic, size: 76 bytes
   ```

5. Kehren Sie zum MQTT-Testclient zurück, während Sie sich die Ausgabe des AWS IoT Device Clients im Terminalfenster des lokalen Host-Computers ansehen. Geben Sie auf der Registerkarte In einem Thema veröffentlichen unter Themenname **`test/dc/subtopic2`** ein, und wählen Sie dann Veröffentlichen aus.
6. Vergewissern Sie sich im Terminalfenster, dass die Nachricht empfangen wurde, indem Sie nach einer Nachricht suchen, wie z. B.:

   ```
   2021-11-10T16:34:32.078Z [DEBUG] {samples/PubSubFeature.cpp}: Message received on
    subscribe topic, size: 77 bytes
   ```
7. Nachdem Sie die Meldungen sehen, die bestätigen, dass beide Nachrichten empfangen wurden, geben Sie **`^C`** (Ctrl-C) ein, um den AWS IoT Device Client zu beenden.
8. Geben Sie diesen Befehl ein, um das Ende der Nachrichtenprotokolldatei und die Nachricht zu sehen, die Sie vom MQTT-Testclient aus veröffentlicht haben.

   ```
   tail -n 20 ~/.aws-iot-device-client/log/pubsub_rx_msgs.log
   ```

   > **Note**
   >
   > Die Protokolldatei enthält nur Nachrichtennutzlasten. Die Nachrichtenthemen werden nicht in der Protokolldatei für empfangene Nachrichten aufgezeichnet. Möglicherweise sehen Sie auch die vom AWS IoT Device Client veröffentlichte Nachricht im Empfangsprotokoll. Das liegt daran, dass der Themenfilter mit Platzhaltern dieses Nachrichtenthema enthält und die Abonnementanforderung manchmal vom Message Broker verarbeitet werden kann, bevor die veröffentlichte Nachricht an Subscriber gesendet wird.

Die Einträge in der Protokolldatei belegen, dass die Nachrichten empfangen wurden. Sie können dieses Verfahren mit anderen Themennamen wiederholen. Alle Nachrichten, deren Themenname mit `test/dc/` beginnt, sollten empfangen und protokolliert werden. Nachrichten mit Themennamen, die mit einem anderen Text beginnen, werden ignoriert.

Nachdem Sie demonstriert haben, wie der AWS IoT Device Client MQTT-Nachrichten veröffentlichen und abonnieren kann, fahren Sie mit Tutorial: Demonstrieren Sie Remote-Aktionen (Jobs) mit dem AWS IoT Geräte-Client fort.

Tutorial: Demonstrieren Sie Remote-Aktionen (Jobs) mit dem AWS IoT Geräte-Client

In diesen Tutorials konfigurieren und implementieren Sie Jobs auf Ihrem Raspberry Pi, um nachzuweisen, dass Sie Fernoperationen an Ihre IoT-Geräte senden können.

So beginnen Sie dieses Tutorial:

- Lassen Sie Ihren lokalen Host-Computer einen Raspberry Pi konfigurieren, wie im vorherigen Abschnitt zu sehen.
- Wenn Sie das Tutorial im vorherigen Abschnitt nicht abgeschlossen haben, können Sie dieses Tutorial ausprobieren, indem Sie den Raspberry Pi mit einer microSD-Karte verwenden, auf der sich das Image befindet, das Sie nach der Installation des AWS IoT Device Clients gespeichert haben. (Optional) Speichern des microSD-Karten-Images
- Wenn Sie diese Demo schon einmal ausgeführt haben, überprüfen ??? Sie, ob alle AWS IoT Ressourcen gelöscht werden, die Sie in früheren Durchläufen erstellt haben, um doppelte Ressourcenfehler zu vermeiden.

Für dieses Tutorial brauchen Sie ungefähr 45 Minuten.

Wenn Sie mit diesem Thema fertig sind:

- Sie haben verschiedene Möglichkeiten demonstriert, wie Ihr IoT-Gerät das verwenden kann AWS IoT Core , um Fernoperationen auszuführen, die von verwaltet werden AWS IoT .

Erforderliche Ausstattung:

- Ihre lokale Entwicklungs- und Testumgebung, die Sie in einem vorherigen Abschnitt getestet haben
- Der Raspberry Pi, den Sie im vorherigen Abschnitt getestet haben
- Die microSD-Speicherkarte des Raspberry Pi, die Sie in einem vorherigen Abschnitt getestet haben

Verfahren in diesem Tutorial

- Bereiten Sie den Raspberry Pi auf die Ausführung von Jobs vor
- Erstellen Sie den Job und führen Sie ihn aus in AWS IoT mit AWS IoT Geräte-Client

Bereiten Sie den Raspberry Pi auf die Ausführung von Jobs vor

Die Verfahren in diesem Abschnitt beschreiben, wie Sie Ihren Raspberry Pi für die Ausführung von Jobs mithilfe des AWS IoT Geräteclients vorbereiten.

> **Note**
>
> Diese Verfahren sind gerätespezifisch. Wenn Sie die Verfahren in diesem Abschnitt mit mehr als einem Gerät gleichzeitig durchführen möchten, benötigt jedes Gerät eine eigene Richtlinie und ein eindeutiges, gerätespezifisches Zertifikat sowie einen ebensolchen Objektnamen. Um jedem Gerät seine eigenen Ressourcen zuzuweisen, führen Sie dieses Verfahren einmal für jedes Gerät durch und ändern Sie dabei die gerätespezifischen Elemente, wie in den Verfahren beschrieben.

Verfahren in diesem Tutorial

- Bereitstellen Ihres Raspberry Pi, um Jobs zu demonstrieren
- Konfigurieren Sie das AWS IoT Device Client zum Ausführen des Jobs Agents

Bereitstellen Ihres Raspberry Pi, um Jobs zu demonstrieren

Die Verfahren in diesem Abschnitt stellen Ihren Raspberry Pi bereit, AWS IoT indem Sie AWS IoT Ressourcen und Gerätezertifikate dafür erstellen.

Themen

- Erstellen Sie Gerätezertifikatsdateien und laden Sie sie herunter, um dies zu demonstrieren AWS IoT jobs
- Create AWS IoT Ressourcen zum Demonstrieren AWS IoT jobs

Erstellen Sie Gerätezertifikatsdateien und laden Sie sie herunter, um dies zu demonstrieren AWS IoT jobs

Mit diesem Verfahren werden die Gerätezertifikatsdateien für diese Demo erstellt.

Wenn Sie mehr als ein Gerät vorbereiten, muss dieses Verfahren auf jedem Gerät durchgeführt werden.

So erstellen Sie die Gerätezertifikatsdateien für Ihren Raspberry Pi und laden sie herunter:

Gehen Sie im Terminalfenster auf Ihrem lokalen Host-Computer, der mit Ihrem Raspberry Pi verbunden ist, folgendermaßen vor:

1. Geben Sie den folgenden Befehl ein, um die Gerätezertifikatsdateien für Ihr Gerät zu erstellen.

   ```
   aws iot create-keys-and-certificate \
   --set-as-active \
   --certificate-pem-outfile "~/certs/jobs/device.pem.crt" \
   --public-key-outfile "~/certs/jobs/public.pem.key" \
   --private-key-outfile "~/certs/jobs/private.pem.key"
   ```

   Die Ausgabe des Befehls ähnelt der Folgenden: Speichern Sie den *certificateArn*-Wert zur späteren Verwendung.

   ```
   {
   "certificateArn": "arn:aws:iot:us-
   west-2:57EXAMPLE833:cert/76e7e4edb3e52f52334be2f387a06145b2aa4c7fcd810f3aea2d92abc227d269",
   "certificateId":
    "76e7e4edb3e52f5233EXAMPLE7a06145b2aa4c7fcd810f3aea2d92abc227d269",
   "certificatePem": "-----BEGIN CERTIFICATE-----
   \nMIIDWTCCAkGgAwIBAgI_SHORTENED_FOR_EXAMPLE_Lgn4jfgtS\n-----END CERTIFICATE-----
   \n",
   "keyPair": {
       "PublicKey": "-----BEGIN PUBLIC KEY-----
   \nMIIBIjANBgkqhkiG9w0BA_SHORTENED_FOR_EXAMPLE_ImwIDAQAB\n-----END PUBLIC KEY-----
   \n",
       "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----
   \nMIIEowIBAAKCAQE_SHORTENED_FOR_EXAMPLE_T9RoDiukY\n-----END RSA PRIVATE KEY-----\n"
   }
   }
   ```

2. Geben Sie die folgenden Befehle ein, um die Berechtigungen für das Zertifikatsverzeichnis und seine Dateien festzulegen.

   ```
   chmod 700 ~/certs/jobs
   chmod 644 ~/certs/jobs/*
   chmod 600 ~/certs/jobs/private.pem.key
   ```

3. Führen Sie diesen Befehl aus, um die Berechtigungen für Ihre Zertifikatsverzeichnisse und -dateien zu überprüfen.

```
ls -l ~/certs/jobs
```

Der Befehl sollte ähnliche Werte wie die hier angezeigten ausgeben, mit der Ausnahme, dass Datum und Uhrzeit unterschiedlich sein werden.

```
-rw-r--r-- 1 pi pi 1220 Oct 28 13:02 device.pem.crt
-rw------- 1 pi pi 1675 Oct 28 13:02 private.pem.key
-rw-r--r-- 1 pi pi  451 Oct 28 13:02 public.pem.key
```

Nachdem Sie die Gerätezertifikatsdateien auf Ihren Raspberry Pi heruntergeladen haben, können Sie mit the section called “Stellen Sie Raspberry Pi für Jobs bereit” fortfahren.

Create AWS IoT Ressourcen zum Demonstrieren AWS IoT jobs

Erstellen Sie die AWS IoT Ressourcen für dieses Gerät.

Wenn Sie mehr als ein Gerät vorbereiten, muss dieses Verfahren auf jedem Gerät durchgeführt werden.

Um Ihr Gerät bereitzustellen in AWS IoT:

Gehen Sie im Terminalfenster auf Ihrem lokalen Host-Computer, der mit Ihrem Raspberry Pi verbunden ist, folgendermaßen vor:

1. Geben Sie den folgenden Befehl ein, um die Adresse des Gerätedatenendpunkts für Ihr AWS-Konto abzurufen.

   ```
   aws iot describe-endpoint --endpoint-type IoT:Data-ATS
   ```

   Der Endpunktwert hat sich seit der letzten Ausführung dieses Befehls nicht geändert. Wenn Sie den Befehl hier erneut ausführen, können Sie den Datenendpunktwert leicht finden und in die in diesem Tutorial verwendete Konfigurationsdatei einfügen.

   Die Ausgabe des describe-endpoint-Befehls ähnelt der folgenden: Notieren Sie sich den *endpointAddress*-Wert zur späteren Verwendung.

   ```
   {
   "endpointAddress": "a3qjEXAMPLEffp-ats.iot.us-west-2.amazonaws.com"
   ```

```
}
```

2. *uniqueThingName*Ersetzen Sie es durch einen eindeutigen Namen für Ihr Gerät. Wenn Sie dieses Tutorial mit mehreren Geräten durchführen möchten, geben Sie jedem Gerät einen eigenen Namen. Beispiele: **TestDevice01**, **TestDevice02** usw.

   Geben Sie diesen Befehl ein, um eine neue Ressource AWS IoT für Ihren Raspberry Pi zu erstellen.

   ```
   aws iot create-thing --thing-name "uniqueThingName"
   ```

   Da eine AWS IoT Ding-Ressource eine virtuelle Repräsentation Ihres Geräts in der Cloud ist, können wir mehrere Ressourcen erstellen, AWS IoT um sie für verschiedene Zwecke zu verwenden. Sie können alle von demselben physischen IoT-Gerät verwendet werden, um verschiedene Aspekte des Geräts darzustellen.

   > **Note**
   >
   > Wenn Sie die Richtlinie für mehrere Geräte sichern möchten, können Sie `${iot:Thing.`*ThingName*`}` anstelle des statischen Objektnamens verwenden, *uniqueThingName*.

   In diesen Tutorials wird jeweils nur eine Objektressource pro Gerät verwendet. Auf diese Weise stellen sie in diesen Tutorials die verschiedenen Demos dar, sodass Sie, nachdem Sie die AWS IoT Ressourcen für eine Demo erstellt haben, die Demos mit den Ressourcen wiederholen können, die Sie speziell für die jeweilige Demo erstellt haben.

   Wenn Ihre AWS IoT Ding-Ressource erstellt wurde, gibt der Befehl eine Antwort wie diese zurück. Notieren Sie sich den *thingArn*-Wert, damit er später verwendet werden kann, wenn Sie den Job für die Ausführung auf diesem Gerät erstellen.

   ```
   {
   "thingName": "uniqueThingName",
   "thingArn": "arn:aws:iot:us-west-2:57EXAMPLE833:thing/uniqueThingName",
   "thingId": "8ea78707-32c3-4f8a-9232-14bEXAMPLEfd"
   }
   ```

3. Im Terminalfenster:

a. Öffnen Sie einen Texteditor, z. B. nano.

b. Kopieren Sie dieses JSON-Dokument und fügen Sie es in Ihren geöffneten Texteditor ein.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:client/uniqueThingName"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/test/dc/pubtopic",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/events/job/
*",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/events/
jobExecution/*",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/
things/uniqueThingName/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topicfilter/test/dc/
subtopic",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/events/
jobExecution/*",
```

```
                "arn:aws:iot:us-west-2:123456789012:topicfilter/$aws/
things/uniqueThingName/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/test/dc/subtopic",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/
things/uniqueThingName/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:DescribeJobExecution",
                "iot:GetPendingJobExecutions",
                "iot:StartNextPendingJobExecution",
                "iot:UpdateJobExecution"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/
things/uniqueThingName"
            ]
        }
    ]
}
```

c. Ersetzen Sie im Editor im `Resource` Abschnitt jeder Richtlinienerklärung *us-west-2:57EXAMPLE833* durch Ihren AWS-Region, einen Doppelpunkt (:) und Ihre 12-stellige AWS-Konto Zahl.

d. Ersetzen Sie im Editor in jeder Grundsatzerklärung durch den Namen des *uniqueThingName* Dings, den Sie dieser Ding-Ressource gegeben haben.

e. Speichern Sie die Datei als **~/policies/jobs_test_thing_policy.json** in Ihrem Texteditor.

   Wenn Sie dieses Verfahren für mehrere Geräte ausführen, speichern Sie die Datei auf jedem Gerät unter diesem Dateinamen.

4. *uniqueThingName*Ersetzen Sie es durch den Dingnamen für das Gerät, und führen Sie dann diesen Befehl aus, um eine auf das Gerät zugeschnittene AWS IoT Richtlinie zu erstellen.

```
aws iot create-policy \
--policy-name "JobTestPolicyFor<uniqueThingName>" \
--policy-document "file://~/policies/jobs_test_thing_policy.json"
```

Wenn die Richtlinie erstellt wurde, gibt der Befehl eine Antwort wie die folgende zurück:

```
{
    "policyName": "JobTestPolicyFor<uniqueThingName>",
    "policyArn": "arn:aws:iot:us-west-2:57EXAMPLE833:policy/
JobTestPolicyFor<uniqueThingName>",
    "policyDocument": "{\n\"Version\": \"2012-10-17\",\n\"Statement\": [\n{\n
\"Effect\": \"Allow\",\n\"Action\": [\n\"iot:Connect\"\n],\n\"Resource\":
 [\n\"arn:aws:iot:us-west-2:57EXAMPLE833:client/PubSubTestThing\"\n]\n},\n{\n
\"Effect\": \"Allow\",\n\"Action\": [\n\"iot:Publish\"\n],\n\"Resource\":
 [\n\"arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/pubtopic\"\n]\n},\n{\n
\"Effect\": \"Allow\",\n\"Action\": [\n\"iot:Subscribe\"\n],\n\"Resource\": [\n
\"arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/subtopic\"\n]\n},\n{\n
\"Effect\": \"Allow\",\n\"Action\": [\n\"iot:Receive\"\n],\n\"Resource\": [\n
\"arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/*\"\n]\n}\n]\n}\n",
    "policyVersionId": "1"
}
```

5. *uniqueThingName*Ersetzen Sie es durch den Dingnamen für das Gerät und *certificateArn* durch den `certificateArn` Wert, den Sie zuvor in diesem Abschnitt für dieses Gerät gespeichert haben, und führen Sie dann diesen Befehl aus, um die Richtlinie an das Gerätezertifikat anzuhängen.

```
aws iot attach-policy \
--policy-name "JobTestPolicyFor<uniqueThingName>" \
--target "<certificateArn>"
```

Bei erfolgreicher Ausführung gibt dieser Befehl nichts zurück.

6. *uniqueThingName*Ersetzen Sie es durch den Dingnamen für das Gerät, *certificateArn* ersetzen Sie es durch den `certificateArn` Wert, den Sie zuvor in diesem Abschnitt gespeichert haben, und führen Sie dann diesen Befehl aus, um das Gerätezertifikat an die AWS IoT Dingressource anzuhängen.

```
aws iot attach-thing-principal \
--thing-name "uniqueThingName" \
--principal "certificateArn"
```

Bei erfolgreicher Ausführung gibt dieser Befehl nichts zurück.

Nachdem Sie Ihren Raspberry Pi erfolgreich bereitgestellt haben, können Sie diesen Abschnitt für einen anderen Raspberry Pi in Ihrem Test wiederholen oder, falls alle Geräte bereitgestellt wurden, mit the section called “Konfigurieren Sie den Device Client für die Ausführung von Jobs” fortfahren.

Konfigurieren Sie das AWS IoT Device Client zum Ausführen des Jobs Agents

Dieses Verfahren erstellt eine Konfigurationsdatei für den AWS IoT Device Client, um den Job-Agenten auszuführen:.

Hinweis: Wenn Sie mehr als ein Gerät vorbereiten, muss dieses Verfahren auf jedem Gerät durchgeführt werden.

Um die Konfigurationsdatei zum Testen der zu erstellen AWS IoT Geräte-Client:

1. Gehen Sie im Terminalfenster auf Ihrem lokalen Host-Computer, der mit Ihrem Raspberry Pi verbunden ist, folgendermaßen vor:

   a. Öffnen Sie einen Texteditor, z. B. `nano`.

   b. Kopieren Sie dieses JSON-Dokument und fügen Sie es in Ihren geöffneten Texteditor ein.

```
{
  "endpoint": "a3qEXAMPLEaffp-ats.iot.us-west-2.amazonaws.com",
  "cert": "~/certs/jobs/device.pem.crt",
  "key": "~/certs/jobs/private.pem.key",
  "root-ca": "~/certs/AmazonRootCA1.pem",
  "thing-name": "uniqueThingName",
  "logging": {
    "enable-sdk-logging": true,
    "level": "DEBUG",
    "type": "STDOUT",
    "file": ""
  },
  "jobs": {
    "enabled": true,
```

```
    "handler-directory": ""
  },
  "tunneling": {
    "enabled": false
  },
  "device-defender": {
    "enabled": false,
    "interval": 300
  },
  "fleet-provisioning": {
    "enabled": false,
    "template-name": "",
    "template-parameters": "",
    "csr-file": "",
    "device-key": ""
  },
  "samples": {
    "pub-sub": {
      "enabled": false,
      "publish-topic": "",
      "publish-file": "",
      "subscribe-topic": "",
      "subscribe-file": ""
    }
  },
  "config-shadow": {
    "enabled": false
  },
  "sample-shadow": {
    "enabled": false,
    "shadow-name": "",
    "shadow-input-file": "",
    "shadow-output-file": ""
  }
}
```

c. Ersetzen Sie den *endpoint* Wert durch den Gerätedaten-Endpunktwert für Ihr AWS-Konto , den Sie in gefunden habenthe section called “Stellen Sie Ihr Gerät bereit in AWS IoT Core”.

d. *uniqueThingName*Ersetzen Sie ihn durch den Namen der Sache, den Sie für dieses Gerät verwendet haben.

e. Speichern Sie die Datei als **~/dc-configs/dc-jobs-config.json** in Ihrem Texteditor.

2. Führen Sie diesen Befehl aus, um die Dateiberechtigungen der neuen Konfigurationsdatei festzulegen.

```
chmod 644 ~/dc-configs/dc-jobs-config.json
```

Sie werden den MQTT-Testclient für diesen Test nicht verwenden. Das Gerät tauscht zwar auftragsbezogene MQTT-Nachrichten mit ihnen aus AWS IoT, Meldungen zum Auftragsfortschritt werden jedoch nur mit dem Gerät ausgetauscht, auf dem der Job ausgeführt wird. Da Meldungen zum Auftragsfortschritt nur mit dem Gerät ausgetauscht werden, auf dem der Job ausgeführt wird, können Sie sie nicht von einem anderen Gerät aus abonnieren, z. B. von der Konsole aus. AWS IoT

Nachdem Sie die Konfigurationsdatei gespeichert haben, können Sie mit the section called “IoT-Job erstellen und ausführen” fortfahren.

Erstellen Sie den Job und führen Sie ihn aus in AWS IoT mit AWS IoT Geräte-Client

Mit den Verfahren in diesem Abschnitt werden ein Jobdokument und eine AWS IoT Jobressource erstellt. Nachdem Sie die Jobressource erstellt haben, AWS IoT sendet das Jobdokument an die angegebenen Jobziele, auf denen ein Job-Agent das Jobdokument auf das Gerät oder den Client anwendet.

Verfahren in diesem Abschnitt

- Erstellen und speichern Sie das Jobdokument für den IoT-Job
- Führen Sie einen Job in aus AWS IoT für ein IoT-Gerät

Erstellen und speichern Sie das Jobdokument für den IoT-Job

Dieses Verfahren erstellt ein einfaches Auftragsdokument, das in eine AWS IoT Jobressource aufgenommen werden soll. In diesem Jobdokument wird „Hallo Welt!“ auf dem Jobziel angezeigt.

So erstellen und speichern Sie ein Jobdokument:

1. Wählen Sie den Amazon-S3-Bucket aus, in dem Sie Ihr Jobdokument speichern möchten. Wenn Sie nicht über einen vorhandenen Amazon-S3-Bucket verfügen, den Sie verwenden können, müssen Sie einen erstellen. Weitere Informationen zum Erstellen von Amazon-S3-Buckets finden Sie unterErste Schritte mit Amazon S3.
2. Erstellen und Speichern des Jobdokuments für diesen Job

a. Öffnen Sie auf Ihrem lokalen Host-Computer einen Texteditor.

b. Kopieren Sie den folgenden Text und fügen Sie ihn in den Editor ein.

```
{
    "operation": "echo",
    "args": ["Hello world!"]
}
```

c. Speichern Sie auf dem lokalen Host-Computer den Inhalt des Editors in einer Datei mit dem Namen **`hello-world-job.json`**.

d. Vergewissern Sie sich, dass die Datei korrekt gespeichert wurde. Einige Texteditoren fügen `.txt` beim Speichern einer Textdatei automatisch an den Dateinamen an. Wenn Ihr Editor `.txt` an den Dateinamen angehängt hat, korrigieren Sie den Dateinamen, bevor Sie fortfahren.

3. Ersetzen Sie den *`path_to_file`* durch den Pfad zu**`hello-world-job.json`**, falls er sich nicht in Ihrem aktuellen Verzeichnis befindet, *`s3_bucket_name`* ersetzen Sie ihn durch den Amazon S3-Bucket-Pfad zu dem von Ihnen ausgewählten Bucket, und führen Sie dann diesen Befehl aus, um Ihr Job-Dokument in den Amazon S3-Bucket einzufügen.

```
aws s3api put-object \
--key hello-world-job.json \
--body path_to_file/hello-world-job.json --bucket s3_bucket_name
```

Die URL des Jobdokuments, die das Auftragsdokument identifiziert, das Sie in Amazon S3 gespeichert haben, wird bestimmt, indem Sie das *`s3_bucket_name`* und *`AWS_region`* in der folgenden URL ersetzen. Notieren Sie sich die resultierende URL, um sie später als zu verwenden *`job_document_path`*

```
https://s3_bucket_name.s3.AWS_Region.amazonaws.com/hello-world-job.json
```

> **Note**
>
> AWS Die Sicherheit verhindert, dass Sie diese URL außerhalb von Ihnen öffnen können AWS-Konto, z. B. mithilfe eines Browsers. Die URL wird standardmäßig von der AWS IoT Job-Engine verwendet, die Zugriff auf die Datei hat. In einer Produktionsumgebung

müssen Sie sicherstellen, dass Ihre AWS IoT -Services berechtigt sind, auf die in Amazon S3 gespeicherten Jobdokumente zuzugreifen.

Nachdem Sie die URL des Jobdokuments gespeichert haben, fahren Sie fort mit the section called “Job für ein einzelnes Gerät ausführen”.

Führen Sie einen Job in aus AWS IoT für ein IoT-Gerät

Die Verfahren in diesem Abschnitt starten den AWS IoT Device Client auf Ihrem Raspberry Pi, um den Job-Agenten auf dem Gerät auszuführen und auf die Ausführung von Jobs zu warten. Außerdem wird eine Jobressource in erstellt AWS IoT, die den Job an Ihr IoT-Gerät sendet und auf diesem ausführt.

> **Note**
>
> Dieses Verfahren führt einen Job nur auf einem einzigen Gerät aus.

So starten Sie den Job-Agenten auf Ihrem Raspberry Pi:

1. Führen Sie im Terminalfenster auf Ihrem lokalen Host-Computer, der mit Ihrem Raspberry Pi verbunden ist, diesen Befehl aus, um den AWS IoT Device Client zu starten.

   ```
   cd ~/aws-iot-device-client/build
   ./aws-iot-device-client --config-file ~/dc-configs/dc-jobs-config.json
   ```

2. Bestätigen Sie im Terminalfenster, dass der AWS IoT Device Client und diese Meldungen anzeigt

   ```
   2021-11-15T18:45:56.708Z [INFO]  {Main.cpp}: Jobs is enabled
                          .
                          .
                          .
   2021-11-15T18:45:56.708Z [INFO]  {Main.cpp}: Client base has been notified that
    Jobs has started
   2021-11-15T18:45:56.708Z [INFO]  {JobsFeature.cpp}: Running Jobs!
   2021-11-15T18:45:56.708Z [DEBUG] {JobsFeature.cpp}: Attempting to subscribe to
    startNextPendingJobExecution accepted and rejected
   2021-11-15T18:45:56.708Z [DEBUG] {JobsFeature.cpp}: Attempting to subscribe to
    nextJobChanged events
   ```

```
2021-11-15T18:45:56.708Z [DEBUG] {JobsFeature.cpp}: Attempting to subscribe to
 updateJobExecutionStatusAccepted for jobId +
2021-11-15T18:45:56.738Z [DEBUG] {JobsFeature.cpp}: Ack received for
 SubscribeToUpdateJobExecutionAccepted with code {0}
2021-11-15T18:45:56.739Z [DEBUG] {JobsFeature.cpp}: Attempting to subscribe to
 updateJobExecutionStatusRejected for jobId +
2021-11-15T18:45:56.753Z [DEBUG] {JobsFeature.cpp}: Ack received for
 SubscribeToNextJobChanged with code {0}
2021-11-15T18:45:56.760Z [DEBUG] {JobsFeature.cpp}: Ack received for
 SubscribeToStartNextJobRejected with code {0}
2021-11-15T18:45:56.776Z [DEBUG] {JobsFeature.cpp}: Ack received for
 SubscribeToStartNextJobAccepted with code {0}
2021-11-15T18:45:56.776Z [DEBUG] {JobsFeature.cpp}: Ack received for
 SubscribeToUpdateJobExecutionRejected with code {0}
2021-11-15T18:45:56.777Z [DEBUG] {JobsFeature.cpp}: Publishing
 startNextPendingJobExecutionRequest
2021-11-15T18:45:56.785Z [DEBUG] {JobsFeature.cpp}: Ack received for
 StartNextPendingJobPub with code {0}
2021-11-15T18:45:56.785Z [INFO]  {JobsFeature.cpp}: No pending jobs are scheduled,
 waiting for the next incoming job
```

3. Wenn Sie im Terminalfenster diese Meldung sehen, fahren Sie mit dem nächsten Verfahren fort und erstellen Sie die Jobressource. Beachten Sie, dass dies möglicherweise nicht der letzte Eintrag in der Liste ist.

```
2021-11-15T18:45:56.785Z [INFO]  {JobsFeature.cpp}: No pending jobs are scheduled,
 waiting for the next incoming job
```

Um eine zu erstellen AWS IoT Job-Ressource

1. Auf Ihrem lokalen Host-Computer:

   a. *job_document_url* Ersetzen Sie durch die URL des Jobdokuments vonthe section called “Jobdokument erstellen und speichern”.

   b. *thing_arn* Ersetzen Sie es durch den ARN der Ding-Ressource, die Sie für Ihr Gerät erstellt haben, und führen Sie dann diesen Befehl aus.

```
aws iot create-job \
--job-id hello-world-job-1 \
--document-source "job_document_url" \
--targets "thing_arn" \
```

```
--target-selection SNAPSHOT
```

Bei Erfolg gibt der Befehl ein Ergebnis wie dieses zurück:

```
{
  "jobArn": "arn:aws:iot:us-west-2:57EXAMPLE833:job/hello-world-job-1",
  "jobId": "hello-world-job-1"
}
```

2. Im Terminalfenster sollten Sie die folgende Ausgabe vom AWS IoT Device Client sehen.

```
2021-11-15T18:02:26.688Z [INFO]  {JobsFeature.cpp}: No pending jobs are scheduled,
 waiting for the next incoming job
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Job ids differ
2021-11-15T18:10:24.890Z [INFO]  {JobsFeature.cpp}: Executing job: hello-world-
job-1
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Attempting to update job
 execution status!
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Not including stdout with the
 status details
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Not including stderr with the
 status details
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Assuming executable is in PATH
2021-11-15T18:10:24.890Z [INFO]  {JobsFeature.cpp}: About to execute: echo Hello
 world!
2021-11-15T18:10:24.890Z [DEBUG] {Retry.cpp}: Retryable function starting, it will
 retry until success
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Created EphermalPromise for
 ClientToken 3TEWba9Xj6 in the updateJobExecution promises map
2021-11-15T18:10:24.890Z [DEBUG] {JobEngine.cpp}: Child process now running
2021-11-15T18:10:24.890Z [DEBUG] {JobEngine.cpp}: Child process about to call
 execvp
2021-11-15T18:10:24.890Z [DEBUG] {JobEngine.cpp}: Parent process now running, child
 PID is 16737
2021-11-15T18:10:24.891Z [DEBUG] {16737}: Hello world!
2021-11-15T18:10:24.891Z [DEBUG] {JobEngine.cpp}: JobEngine finished waiting for
 child process, returning 0
2021-11-15T18:10:24.891Z [INFO]  {JobsFeature.cpp}: Job exited with status: 0
2021-11-15T18:10:24.891Z [INFO]  {JobsFeature.cpp}: Job executed successfully!
2021-11-15T18:10:24.891Z [DEBUG] {JobsFeature.cpp}: Attempting to update job
 execution status!
2021-11-15T18:10:24.891Z [DEBUG] {JobsFeature.cpp}: Not including stdout with the
 status details
```

```
2021-11-15T18:10:24.891Z [DEBUG] {JobsFeature.cpp}: Not including stderr with the
 status details
2021-11-15T18:10:24.892Z [DEBUG] {Retry.cpp}: Retryable function starting, it will
 retry until success
2021-11-15T18:10:24.892Z [DEBUG] {JobsFeature.cpp}: Created EphermalPromise for
 ClientToken GmQ0HTzWGg in the updateJobExecution promises map
2021-11-15T18:10:24.905Z [DEBUG] {JobsFeature.cpp}: Ack received for
 PublishUpdateJobExecutionStatus with code {0}
2021-11-15T18:10:24.905Z [DEBUG] {JobsFeature.cpp}: Removing ClientToken 3TEWba9Xj6
 from the updateJobExecution promises map
2021-11-15T18:10:24.905Z [DEBUG] {JobsFeature.cpp}: Success response after
 UpdateJobExecution for job hello-world-job-1
2021-11-15T18:10:24.917Z [DEBUG] {JobsFeature.cpp}: Ack received for
 PublishUpdateJobExecutionStatus with code {0}
2021-11-15T18:10:24.918Z [DEBUG] {JobsFeature.cpp}: Removing ClientToken GmQ0HTzWGg
 from the updateJobExecution promises map
2021-11-15T18:10:24.918Z [DEBUG] {JobsFeature.cpp}: Success response after
 UpdateJobExecution for job hello-world-job-1
2021-11-15T18:10:25.861Z [INFO]  {JobsFeature.cpp}: No pending jobs are scheduled,
 waiting for the next incoming job
```

3. Während der AWS IoT Device Client läuft und auf einen Job wartet, können Sie einen weiteren Job einreichen, indem Sie den `job-id` Wert ändern und den Befehl create-job aus Schritt 1 erneut ausführen.

Wenn Sie mit der Ausführung der Jobs fertig sind, geben Sie im Terminalfenster ^C (Strg-C) ein, um den AWS IoT Device Client zu beenden.

Tutorial: Aufräumen nach dem Ausführen des AWS IoT Tutorials zum Device Client

Die Verfahren in diesem Tutorial führen Sie Schritt für Schritt durch das Entfernen der Dateien und Ressourcen, die Sie während der Durchführung der Tutorials in diesem Lernpfad erstellt haben.

Verfahren in diesem Tutorial

- Schritt 1: Bereinigen Sie Ihre Geräte nach der Erstellung von Demos mit dem AWS IoT Geräte-Client
- Schritt 2: Bereinigen Sie Ihre AWS-Konto nach dem Erstellen von Demos mit dem AWS IoT Geräte-Client

Schritt 1: Bereinigen Sie Ihre Geräte nach der Erstellung von Demos mit dem AWS IoT Geräte-Client

In diesem Tutorial werden zwei Optionen beschrieben, wie Sie die microSD-Karte bereinigen können, nachdem Sie die Demos in diesem Lernpfad erstellt haben. Wählen Sie die Option aus, die das von Ihnen benötigte Sicherheitsniveau bietet.

Beachten Sie, dass durch das Reinigen der microSD-Karte des Geräts keine AWS IoT Ressourcen entfernt werden, die Sie erstellt haben. Um die AWS IoT Ressourcen nach dem Reinigen der microSD-Karte des Geräts zu bereinigen, sollten Sie das Tutorial unter lesen. the section called “Aufräumen nach dem Erstellen von Demos mit dem AWS IoT Device Client”

Option 1: Bereinigen durch Umschreiben der microSD-Karte

Die einfachste und gründlichste Methode, die microSD-Karte nach Abschluss der Tutorials in diesem Lernpfad zu bereinigen, besteht darin, die microSD-Karte mit einer gespeicherten Image-Datei zu überschreiben, die Sie bei der ersten Vorbereitung Ihres Geräts erstellt haben.

Bei diesem Verfahren wird der lokale Hostcomputer verwendet, um ein gespeichertes microSD-Karten-Image auf eine microSD-Karte zu schreiben.

> **Note**
>
> Wenn Ihr Gerät kein Wechselspeichermedium für sein Betriebssystem verwendet, lesen Sie das Verfahren für dieses Gerät.

So schreiben Sie ein neues Image auf die microSD-Karte

1. Suchen Sie auf Ihrem lokalen Hostcomputer das gespeicherte microSD-Karten-Image, das Sie auf Ihre microSD-Karte schreiben möchten.
2. Legen Sie die microSD-Karte in Ihren Hostcomputer ein.
3. Schreiben Sie die ausgewählte Image-Datei mit einem SD-Karten-Imaging-Tool auf die microSD-Karte.
4. Nachdem Sie das Raspberry Pi OS-Image auf die microSD-Karte geschrieben haben, werfen Sie die microSD-Karte aus und entfernen Sie sie sicher vom lokalen Hostcomputer.

Ihre microSD-Karte ist nun einsatzbereit.

Option 2: Bereinigen durch Löschen von Benutzerverzeichnissen

Um die microSD-Karte nach Abschluss der Tutorials zu bereinigen, ohne das microSD-Karten-Image neu zu schreiben, können Sie die Benutzerverzeichnisse einzeln löschen. Dies ist nicht so gründlich wie das Neuschreiben der microSD-Karte aus einem gespeicherten Image, da dabei keine Systemdateien entfernt werden, die möglicherweise installiert wurden.

Wenn das Entfernen der Benutzerverzeichnisse für Ihre Bedürfnisse ausreichend ist, können Sie wie folgt vorgehen.

So löschen Sie die Benutzerverzeichnisse dieses Lernpfads von Ihrem Gerät

1. Führen Sie diese Befehle aus, um die Benutzerverzeichnisse, Unterverzeichnisse und alle zugehörigen Dateien, die in diesem Lernpfad erstellt wurden, im mit Ihrem Gerät verbundenen Terminalfenster zu löschen.

   > **Note**
   >
   > Nachdem Sie diese Verzeichnisse und Dateien gelöscht haben, können Sie die Demos nicht mehr ausführen, ohne die Tutorials erneut abzuschließen.

   ```
   rm -Rf ~/dc-configs
   rm -Rf ~/policies
   rm -Rf ~/messages
   rm -Rf ~/certs
   rm -Rf ~/.aws-iot-device-client
   ```

2. Führen Sie diese Befehle aus, um die Quellverzeichnisse und Dateien der Anwendung im Terminalfenster zu löschen, das mit Ihrem Gerät verbunden ist.

   > **Note**
   >
   > Mit diesen Befehlen werden keine Programme deinstalliert. Sie entfernen nur die Quelldateien, die zu ihrer Erstellung und Installation verwendet wurden. Nachdem Sie diese Dateien gelöscht haben, funktionieren der AWS CLI und der AWS IoT Device Client möglicherweise nicht.

```
rm -Rf ~/aws-cli
rm -Rf ~/aws
rm -Rf ~/aws-iot-device-client
```

Schritt 2: Bereinigen Sie Ihre AWS-Konto nach dem Erstellen von Demos mit dem AWS IoT Geräte-Client

Diese Verfahren helfen Ihnen dabei, die AWS Ressourcen zu identifizieren und zu entfernen, die Sie beim Abschließen der Tutorials in diesem Lernpfad erstellt haben.

Bereinigen AWS IoT Ressourcen

Dieses Verfahren hilft Ihnen dabei, die AWS IoT Ressourcen zu identifizieren und zu entfernen, die Sie beim Abschließen der Tutorials in diesem Lernpfad erstellt haben.

AWS IoT Ressourcen, die in diesem Lernpfad erstellt wurden

Tutorial	Objektressource	Richtlinienressource
the section called “Installation und Konfiguration des IoT-Geräteclients”	DevCliTestThing	DevCliTestThingPolicy
the section called “Kommunizieren Sie mit dem Geräteclient über MQTT”	PubSubTestThing	PubSubTestThingPolicy
the section called “Führen Sie IoT-Jobs mit dem Device Client aus”	benutzerdefiniert (es können mehrere vorhanden sein)	benutzerdefiniert (es können mehrere vorhanden sein)

Um das zu löschen AWS IoT Ressourcen, folgen Sie diesem Verfahren für jede Ressource, die Sie erstellt haben

1. Ersetzen Sie *thing_name* durch den Namen der Objektressource, die Sie löschen möchten, und führen Sie dann diesen Befehl aus, um die Zertifikate aufzulisten, die an die Objektressource vom lokalen Hostcomputer aus angehängt sind.

```
aws iot list-thing-principals --thing-name thing_name
```

Dieser Befehl gibt eine Antwort wie diese zurück, in der die Zertifikate aufgeführt sind, die an *thing_name* angehängt sind. In den meisten Fällen wird die Liste nur ein Zertifikat enthalten.

```
{
    "principals": [
        "arn:aws:iot:us-
west-2:57EXAMPLE833:cert/23853eea3cf0edc7f8a69c74abeafa27b2b52823cab5b3e156295e94b26ae8ac"
    ]
}
```

2. Gehen Sie wie folgt für jedes Zertifikat vor, das im vorherigen Befehl aufgeführt wurde:

   a. Ersetzen Sie *certificate_ID* durch die Zertifikat-ID aus dem vorherigen Befehl. Die Zertifikat-ID besteht aus den alphanumerischen Zeichen, die auf `cert/` in dem vom vorherigen Befehl zurückgegebenen ARN folgen. Führen Sie dann diesen Befehl aus, um das Zertifikat zu deaktivieren.

   ```
   aws iot update-certificate --new-status INACTIVE --certificate-
   id certificate_ID
   ```

   Bei Erfolg gibt dieser Befehl nichts zurück.

   b. Ersetzen Sie *certificate_ARN* durch den Zertifikat-ARN aus der zuvor zurückgegebenen Zertifikatsliste, und führen Sie dann diesen Befehl aus, um die mit diesem Zertifikat verknüpften Richtlinien aufzulisten.

   ```
   aws iot list-attached-policies --target certificate_ARN
   ```

   Dieser Befehl gibt eine Antwort wie diese zurück, in der die Richtlinien aufgeführt sind, die an das Zertifikat angehängt sind. In den meisten Fällen wird die Liste nur eine Richtlinie enthalten.

   ```
   {
       "policies": [
           {
               "policyName": "DevCliTestThingPolicy",
   ```

```
            "policyArn": "arn:aws:iot:us-west-2:57EXAMPLE833:policy/
DevCliTestThingPolicy"
        }
    ]
}
```

c. Gehen Sie für jede dem Zertifikat angefügte Richtlinie wie folgt vor:

i. Ersetzen Sie *policy_name* durch den `policyName`-Wert aus dem vorherigen Befehl, ersetzen Sie *certificate_ARN* durch den ARN des Zertifikats, und führen Sie dann diesen Befehl aus, um die Richtlinie vom Zertifikat zu trennen.

```
aws iot detach-policy --policy-name policy_name --target certificate_ARN
```

Bei Erfolg gibt dieser Befehl nichts zurück.

ii. Ersetzen Sie *policy_name* durch den `policyName`-Wert, und führen Sie dann diesen Befehl aus, um zu überprüfen, ob die Richtlinie an weitere Zertifikate angehängt ist.

```
aws iot list-targets-for-policy --policy-name policy_name
```

Wenn der Befehl eine leere Liste wie diese zurückgibt, ist die Richtlinie an keine Zertifikate angehängt, und Sie können die Richtlinienversionen weiterhin auflisten. Wenn der Richtlinie immer noch Zertifikate angehängt sind, fahren Sie mit dem Schritt detach-thing-principal fort.

```
{
    "targets": []
}
```

iii. Ersetzen Sie *policy_name* durch den `policyName`-Wert, und führen Sie dann diesen Befehl aus, um nach Richtlinienversionen zu suchen. Um die Richtlinie zu löschen, darf sie nur eine Version haben.

```
aws iot list-policy-versions --policy-name policy_name
```

Wenn die Richtlinie nur eine Version hat, wie in diesem Beispiel, können Sie mit dem Schritt delete-policy fortfahren und die Richtlinie jetzt löschen.

```
{
```

```
    "policyVersions": [
        {
            "versionId": "1",
            "isDefaultVersion": true,
            "createDate": "2021-11-18T01:02:46.778000+00:00"
        }
    ]
}
```

Wenn die Richtlinie mehr als eine Version hat, wie in diesem Beispiel, müssen die Richtlinienversionen mit einem `isDefaultVersion`-Wert von `false` gelöscht werden, bevor die Richtlinie gelöscht werden kann.

```
{
    "policyVersions": [
        {
            "versionId": "2",
            "isDefaultVersion": true,
            "createDate": "2021-11-18T01:52:04.423000+00:00"
        },
        {
            "versionId": "1",
            "isDefaultVersion": false,
            "createDate": "2021-11-18T01:30:18.083000+00:00"
        }
    ]
}
```

Wenn Sie eine Richtlinienversion löschen müssen, ersetzen Sie *`policy_name`* durch den `policyName`-Wert, ersetzen Sie *`version_ID`* durch den `versionId`-Wert aus dem vorherigen Befehl, und führen Sie dann diesen Befehl aus, um eine Richtlinienversion zu löschen.

```
aws iot delete-policy-version --policy-name policy_name --policy-version-
id version_ID
```

Bei Erfolg gibt dieser Befehl nichts zurück.

Nachdem Sie eine Richtlinienversion gelöscht haben, wiederholen Sie diesen Schritt, bis die Richtlinie nur noch eine Richtlinienversion hat.

iv. Ersetzen Sie *policy_name* durch den `policyName`-Wert, und führen Sie dann diesen Befehl aus, um die Richtlinie zu löschen.

```
aws iot delete-policy --policy-name policy_name
```

d. Ersetzen Sie *thing_name* durch den Objektnamen, ersetzen Sie *certificate_ARN* durch den ARN des Zertifikats, und führen Sie dann diesen Befehl aus, um das Zertifikat von der Objektressource zu trennen.

```
aws iot detach-thing-principal --thing-name thing_name --
principal certificate_ARN
```

Bei Erfolg gibt dieser Befehl nichts zurück.

e. Ersetzen Sie *certificate_ID* durch die Zertifikat-ID aus dem vorherigen Befehl. Die Zertifikat-ID besteht aus den alphanumerischen Zeichen, die auf `cert/` in dem vom vorherigen Befehl zurückgegebenen ARN folgen. Führen Sie dann diesen Befehl aus, um die Zertifikatressource zu löschen.

```
aws iot delete-certificate --certificate-id certificate_ID
```

Bei Erfolg gibt dieser Befehl nichts zurück.

3. Ersetzen Sie *thing_name* durch den Namen des Objekts, und führen Sie dann diesen Befehl aus, um das Objekt zu löschen.

```
aws iot delete-thing --thing-name thing_name
```

Bei Erfolg gibt dieser Befehl nichts zurück.

Bereinigen AWS Ressourcen

Dieses Verfahren hilft Ihnen dabei, andere AWS Ressourcen zu identifizieren und zu entfernen, die Sie beim Durcharbeiten der Tutorials in diesem Lernpfad erstellt haben.

Sonstige AWS Ressourcen, die in diesem Lernpfad erstellt wurden

Tutorial	Ressourcentyp	Name oder ID der Ressource
the section called “Führen Sie IoT-Jobs mit dem Device Client aus”	Amazon S3-Objekt	hello-world-job.json
the section called “Führen Sie IoT-Jobs mit dem Device Client aus”	AWS IoT Ressourcen für die Arbeit	benutzerdefiniert

Um die zu löschen AWS Ressourcen, die auf diesem Lernpfad erstellt wurden

1. So löschen Sie die Aufträge, die in diesem Lernpfad erstellt wurden

   a. Führen Sie diesen Befehl aus, um die Jobs in Ihrem aufzulisten AWS-Konto.

   ```
   aws iot list-jobs
   ```

   Der Befehl gibt eine Liste der AWS IoT Jobs in Ihrem zurück AWS-Konto und AWS-Region das sieht so aus.

   ```
   {
       "jobs": [
           {
               "jobArn": "arn:aws:iot:us-west-2:57EXAMPLE833:job/hello-world-
   job-2",
               "jobId": "hello-world-job-2",
               "targetSelection": "SNAPSHOT",
               "status": "COMPLETED",
               "createdAt": "2021-11-16T23:40:36.825000+00:00",
               "lastUpdatedAt": "2021-11-16T23:40:41.375000+00:00",
               "completedAt": "2021-11-16T23:40:41.375000+00:00"
           },
           {
               "jobArn": "arn:aws:iot:us-west-2:57EXAMPLE833:job/hello-world-
   job-1",
               "jobId": "hello-world-job-1",
               "targetSelection": "SNAPSHOT",
   ```

```
            "status": "COMPLETED",
            "createdAt": "2021-11-16T23:35:26.381000+00:00",
            "lastUpdatedAt": "2021-11-16T23:35:29.239000+00:00",
            "completedAt": "2021-11-16T23:35:29.239000+00:00"
        }
    ]
}
```

b. Ersetzen Sie für jeden Job, den Sie in der Liste als einen Job erkennen, den Sie in diesem Lernpfad erstellt haben, *jobId* durch den `jobId` Wert des Jobs, der gelöscht werden soll, und führen Sie dann diesen Befehl aus, um einen AWS IoT Job zu löschen.

```
aws iot delete-job --job-id jobId
```

Wenn der Befehl erfolgreich ist, wird nichts zurückgegeben.

2. So löschen Sie die Auftragsdokumente, die Sie in diesem Lernpfad in einem Amazon S3-Bucket gespeichert haben.

a. Ersetzen Sie *bucket* durch den Namen des Buckets, den Sie verwendet haben, und führen Sie dann diesen Befehl aus, um die Objekte im Amazon S3-Bucket aufzulisten, den Sie verwendet haben.

```
aws s3api list-objects --bucket bucket
```

Der Befehl gibt eine Liste der Amazon S3-Objekte in Ihrem Bucket zurück, die so aussieht:

```
{
    "Contents": [
        {
            "Key": "hello-world-job.json",
            "LastModified": "2021-11-18T03:02:12+00:00",
            "ETag": "\"868c8bc3f56b5787964764d4b18ed5ef\"",
            "Size": 54,
            "StorageClass": "STANDARD",
            "Owner": {
                "DisplayName": "EXAMPLE",
                "ID":
 "e9e3d6ec1EXAMPLEf5bfb5e6bd0a2b6ed03884d1ed392a82ad011c144736a4ee"
            }
        },
        {
```

```
            "Key": "iot_job_firmware_update.json",
            "LastModified": "2021-04-13T21:57:07+00:00",
            "ETag": "\"7c68c591949391791ecf625253658c61\"",
            "Size": 66,
            "StorageClass": "STANDARD",
            "Owner": {
                "DisplayName": "EXAMPLE",
                "ID":
 "e9e3d6ec1EXAMPLEf5bfb5e6bd0a2b6ed03884d1ed392a82ad011c144736a4ee"
            }
        },
        {
            "Key": "order66.json",
            "LastModified": "2021-04-13T21:57:07+00:00",
            "ETag": "\"bca60d5380b88e1a70cc27d321caba72\"",
            "Size": 29,
            "StorageClass": "STANDARD",
            "Owner": {
                "DisplayName": "EXAMPLE",
                "ID":
 "e9e3d6ec1EXAMPLEf5bfb5e6bd0a2b6ed03884d1ed392a82ad011c144736a4ee"
            }
        }
    ]
}
```

b. Ersetzen Sie für jedes Objekt, das Sie in der Liste als ein Objekt erkennen, das Sie in diesem Lernpfad erstellt haben, *bucket* durch den Bucketnamen und *key* mit dem Schlüsselwert des zu löschenden Objekts, und führen Sie dann diesen Befehl aus, um ein Amazon S3-Objekt zu löschen.

```
aws s3api delete-object --bucket bucket --key key
```

Wenn der Befehl erfolgreich ist, wird nichts zurückgegeben.

Nachdem Sie alle AWS Ressourcen und Objekte gelöscht haben, die Sie beim Abschluss dieses Lernpfads erstellt haben, können Sie von vorne beginnen und die Tutorials wiederholen.

Erstellen von Lösungen mit dem AWS IoT Geräte-SDKs

Die Tutorials in diesem Abschnitt führen Sie Schritt für Schritt durch die Entwicklung einer IoT-Lösung, die mithilfe AWS IoT von... in einer Produktionsumgebung bereitgestellt werden kann.

Die Fertigstellung dieser Tutorials kann mehr Zeit in Anspruch nehmen als die Tutorials im Abschnitt über, the section called “Erstellen von Demos mit dem AWS IoT Geräte-Client” da sie die AWS IoT Geräte-SDKs verwenden und die angewandten Konzepte detaillierter erläutern, um Ihnen bei der Erstellung sicherer und zuverlässiger Lösungen zu helfen.

Beginnen Sie mit der Entwicklung von Lösungen mit dem AWS IoT Geräte-SDKs

Diese Tutorials führen Sie durch verschiedene AWS IoT Szenarien. Gegebenenfalls verwenden die Tutorials die AWS IoT Geräte-SDKs.

Themen

- Tutorial: Ein Gerät anschließen an AWS IoT Core mit dem AWS IoT Geräte-SDK
- Erstellen AWS IoT Regeln zum Weiterleiten von Gerätedaten an andere Dienste
- Beibehalten des Gerätestatus mit Geräteschatten, während das Gerät offline ist
- Tutorial: Erstellen eines benutzerdefinierten Autorisierers für AWS IoT Core
- Tutorial: Überwachung der Bodenfeuchte mit AWS IoT und Raspberry Pi

Tutorial: Ein Gerät anschließen an AWS IoT Core mit dem AWS IoT Geräte-SDK

Dieses Tutorial zeigt, wie Sie ein Gerät verbinden, AWS IoT Core damit es Daten an und von dort senden und empfangen kann AWS IoT. Nachdem Sie dieses Tutorial abgeschlossen haben, wird Ihr Gerät so konfiguriert, dass es eine Verbindung herstellen kann, AWS IoT Core und Sie werden verstehen, wie Geräte miteinander kommunizieren AWS IoT.

Themen

- Pre-requisites
- Bereiten Sie Ihr Gerät vor für AWS IoT
- Ansehen des MQTT-Protokolls
- Ansehen der Beispiel-App „pubsub.py“ des Device SDK

- Verbinde dein Gerät und kommuniziere mit AWS IoT Core
- Überprüfen Sie die Ergebnisse.
- Tutorial: Verwenden des AWS IoT Device SDK for Embedded C

Pre-requisites

Stellen Sie vor Beginn dieses Tutorials sicher, dass Sie über Folgendes verfügen:

- Completed Erste Schritte mit AWS IoT Core Tutorials

  Wählen Sie im Abschnitt des besagten Tutorials, in dem Sie the section called “Konfigurieren Ihres Geräts” müssen, die Option the section called “Verbinden eines Raspberry Pi oder eines anderes Gerätes” für Ihr Gerät und verwenden Sie die Python-Sprachoptionen, um Ihr Gerät zu konfigurieren.

  > **Note**
  >
  > Lassen Sie das im besagten Tutorial verwendete Terminalfenster geöffnet, da Sie es auch in diesem Tutorial verwenden.

- Ein Gerät, auf dem das ausgeführt werden kann AWS IoT Geräte-SDK v2 für Python.

  Dieses Tutorial zeigt anhand AWS IoT Core von Python-Codebeispielen, für die ein relativ leistungsfähiges Gerät erforderlich ist, eine Verbindung zu einem Gerät herstellen. Wenn Sie mit ressourcenbeschränkten Geräten arbeiten, funktionieren diese Codebeispiele möglicherweise nicht. In diesem Fall haben Sie möglicherweise mehr Erfolg mit dem the section called “Mit dem AWS IoT Device SDK for Embedded C” Tutorial.

- Sie haben die erforderlichen Informationen erhalten, um eine Verbindung zum Gerät herzustellen

  Um Ihr Gerät mit zu verbinden AWS IoT, benötigen Sie Informationen über den Namen des Dings, den Hostnamen und die Portnummer.

  > **Note**
  >
  > Sie können auch eine benutzerdefinierte Authentifizierung verwenden, um Geräte zu verbinden AWS IoT Core. Die Verbindungsdaten, die Sie an Ihre Autorisierungs-Lambda-Funktion übergeben, hängen vom verwendeten Protokoll ab.

- Name der Sache: Der Name der AWS IoT Sache, zu der Sie eine Verbindung herstellen möchten. Sie müssen sich als AWS IoT Ding als Ihr Gerät registriert haben. Weitere Informationen finden Sie unter Geräte verwalten mit AWS IoT.
- Hostname: Der Hostname für den kontospezifischen IoT-Endpunkt.
- Portnummer: Die Portnummer, zu der eine Verbindung hergestellt werden soll.

Sie können die `configureEndpoint` Methode im AWS IoT Python-SDK verwenden, um den Hostnamen und die Portnummer zu konfigurieren.

```
myAWSIoTMQTTClient.configureEndpoint("random.iot.region.amazonaws.com", 8883)
```

Bereiten Sie Ihr Gerät vor für AWS IoT

In Erste Schritte mit AWS IoT Core Tutorials haben Sie Ihr Gerät und Ihr AWS -Konto vorbereitet, sodass sie kommunizieren können. In diesem Abschnitt werden die Aspekte dieser Vorbereitung beschrieben, die für alle Geräteverbindungen mit gelten AWS IoT Core.

Damit ein Gerät sich mit AWS IoT Core verbinden kann:

1. Benötigen Sie ein AWS-Konto.

   Das Verfahren unter Einrichten AWS-Konto beschreibt, wie Sie eine erstellen, AWS-Konto falls Sie noch keine haben.
2. In diesem Konto müssen die folgenden AWS IoT Ressourcen für das Gerät in Ihrer AWS-Konto und Region definiert sein.

   Das Verfahren unter Create AWS IoT Ressourcen beschreibt, wie Sie diese Ressourcen für das Gerät in Ihrem AWS-Konto und Ihrer Region erstellen.

   - Ein Gerätezertifikat, das bei AWS IoT registriert und zur Authentifizierung des Geräts aktiviert wurde.

     Das Zertifikat wird häufig mit einem AWS IoT -Objekt erstellt und an dieses angehängt. Ein Ding-Objekt ist zwar nicht erforderlich, damit ein Gerät eine Verbindung herstellen kann AWS IoT, es stellt dem Gerät jedoch zusätzliche AWS IoT Funktionen zur Verfügung.
   - Eine an das Gerätezertifikat angehängte Richtlinie, die es autorisiert, eine Verbindung mit dem Gerätezertifikat herzustellen AWS IoT Core und alle gewünschten Aktionen auszuführen.

3. Eine Internetverbindung, die auf die Endpunkte des Geräts Ihres AWS-Konto s zugreifen kann.

   Die Geräteendpunkte werden auf der Einstellungsseite der Konsole beschrieben AWS IoT Gerätedaten und Dienstendpunkte und können dort eingesehen werden. AWS IoT

4. Kommunikationssoftware, wie sie die AWS IoT Device SDKs bieten. In diesem Tutorial wird das AWS IoT Device SDK v2 für Python verwendet.

Ansehen des MQTT-Protokolls

Bevor wir über die Beispiel-App sprechen, ist ein Verständnis des MQTT-Protokolls hilfreich. Das MQTT-Protokoll bietet einige Vorteile gegenüber anderen Netzwerk-Kommunikationsprotokollen wie HTTP, was es zu einer beliebten Wahl für IoT-Geräte macht. In diesem Abschnitt werden die wichtigsten Aspekte von MQTT beschrieben, die für dieses Tutorial gelten. Weitere Informationen über den Vergleich zwischen MQTT und HTTP finden Sie unter Wählen Sie ein Anwendungsprotokoll für Ihre Gerätekommunikation.

MQTT verwendet ein Kommunikationsmodell publish/subscribe

Das MQTT-Protokoll verwendet ein publish/subscribe Kommunikationsmodell mit seinem Host. Dieses Modell unterscheidet sich von dem request/response Modell, das HTTP verwendet. Bei MQTT richten Geräte beim Host eine Sitzung ein, die durch eine eindeutige Client-ID identifiziert ist. Zum Senden von Daten veröffentlichen die Geräte durch Themen identifizierte Nachrichten an einen Message Broker auf dem Host. Zum Empfangen von Nachrichten vom Message Broker abonnieren Geräte Themen, indem sie Themenfilter in Subscriber-Anfragen an den Message Broker senden.

MQTT unterstützt persistente Sitzungen

Der Message Broker empfängt Nachrichten von Geräten und veröffentlicht Nachrichten an Geräte, die diese abonniert haben. Mit persistenten Sitzungen – Sitzungen, die auch dann aktiv bleiben, wenn das initiierende Gerät getrennt ist – können Geräte Nachrichten abrufen, die während der Unterbrechung der Verbindung veröffentlicht wurden. Auf der Geräteseite unterstützt MQTT Quality-of-Service-Levels (QoS), die sicherstellen, dass der Host vom Gerät gesendete Nachrichten empfängt.

Ansehen der Beispiel-App „pubsub.py“ des Device SDK

In diesem Abschnitt wird die Beispiel-App `pubsub.py` aus dem AWS IoT Device SDK v2 für Python beschrieben, die in diesem Tutorial verwendet wird. Hier werden wir überprüfen, wie es eine

Verbindung herstellt, AWS IoT Core um MQTT-Nachrichten zu veröffentlichen und zu abonnieren. Im nächsten Abschnitt finden Sie einige Übungen, anhand derer Sie herausfinden können, wie ein Gerät eine Verbindung herstellt und mit dem Gerät kommuniziert. AWS IoT Core

Die pubsub.py Beispiel-App demonstriert diese Aspekte einer MQTT-Verbindung mit: AWS IoT Core

- Kommunikationsprotokolle
- Persistente Sitzungen
- Quality of Service (Servicequalität)
- Nachrichten veröffentlichen
- Nachrichten abonnieren
- Trennen und erneutes Anschließen des Geräts

Kommunikationsprotokolle

Das `pubsub.py`-Beispiel zeigt eine MQTT-Verbindung mit den Protokollen MQTT und MQTT über WSS. Die AWS Common Runtime (AWS CRT) -Bibliothek bietet die Unterstützung des Low-Level-Kommunikationsprotokolls und ist im AWS IoT Device SDK v2 für Python enthalten.

MQTT

Das `pubsub.py` Beispiel ruft `mtls_from_path` (hier gezeigt) auf, `mqtt_connection_builder` um AWS IoT Core mithilfe des MQTT-Protokolls eine Verbindung herzustellen. `mtls_from_path`verwendet X.509 Zertifikate und TLS v1.2, um das Gerät zu authentifizieren. Die AWS CRT-Bibliothek verarbeitet die untergeordneten Details dieser Verbindung.

```
mqtt_connection = mqtt_connection_builder.mtls_from_path(
    endpoint=args.endpoint,
    cert_filepath=args.cert,
    pri_key_filepath=args.key,
    ca_filepath=args.ca_file,
    client_bootstrap=client_bootstrap,
    on_connection_interrupted=on_connection_interrupted,
    on_connection_resumed=on_connection_resumed,
    client_id=args.client_id,
    clean_session=False,
    keep_alive_secs=6
)
```

`endpoint`

Der Endpunkt AWS-Konto Ihres IoT-Geräts

In der Beispiel-App wird dieser Wert von der Befehlszeile aus übergeben.

`cert_filepath`

Der Pfad zur Zertifikatsdatei des Geräts

In der Beispiel-App wird dieser Wert von der Befehlszeile aus übergeben.

`pri_key_filepath`

Der Pfad zur privaten Schlüsseldatei des Geräts, die mit der zugehörigen Zertifikatsdatei erstellt wurde

In der Beispiel-App wird dieser Wert von der Befehlszeile aus übergeben.

`ca_filepath`

Der Pfad zur Root-CA-Datei Nur erforderlich, wenn der MQTT-Server ein Zertifikat verwendet, das sich noch nicht in Ihrem Trust Store befindet.

In der Beispiel-App wird dieser Wert von der Befehlszeile aus übergeben.

`client_bootstrap`

Das Common-Runtime-Objekt, das Socket-Kommunikationsaktivitäten verarbeitet

In der Beispiel-App wird dieses Objekt vor dem Aufruf zum `mqtt_connection_builder.mtls_from_path` instanziiert.

`on_connection_interrupted`, `on_connection_resumed`

Die Callback-Funktionen, die aufgerufen werden, wenn die Verbindung des Geräts unterbrochen und wieder aufgenommen wird

`client_id`

Die ID, die dieses Gerät eindeutig in der AWS-Region identifiziert

In der Beispiel-App wird dieser Wert von der Befehlszeile aus übergeben.

`clean_session`

Wahl der Option, eine neue persistente Sitzung zu starten oder, falls vorhanden, eine Verbindung zu einer bestehenden Sitzung wiederherzustellen

`keep_alive_secs`

Der Keep-Alive-Wert in Sekunden, der in der `CONNECT`-Anfrage gesendet werden soll. In diesem Intervall wird automatisch ein Ping gesendet. Wenn der Server nach dem 1,5-fachen dieses Werts keinen Ping empfängt, wird davon ausgegangen, dass die Verbindung unterbrochen wurde.

MQTT über WSS

Das `pubsub.py`-Beispiel ruft `websockets_with_default_aws_signing` (hier gezeigt) in `mqtt_connection_builder` auf, zur Herstellung einer Verbindung mit AWS IoT Core mithilfe des MQTT-Protokolls über WSS. `websockets_with_default_aws_signing` erstellt mithilfe von Signature V4 eine MQTT-Verbindung über WSS zur Authentifizierung des Geräts her.

```
mqtt_connection = mqtt_connection_builder.websockets_with_default_aws_signing(
    endpoint=args.endpoint,
    client_bootstrap=client_bootstrap,
    region=args.signing_region,
    credentials_provider=credentials_provider,
    websocket_proxy_options=proxy_options,
    ca_filepath=args.ca_file,
    on_connection_interrupted=on_connection_interrupted,
    on_connection_resumed=on_connection_resumed,
    client_id=args.client_id,
    clean_session=False,
    keep_alive_secs=6
)
```

`endpoint`

Der Endpunkt AWS-Konto Ihres IoT-Geräts

In der Beispiel-App wird dieser Wert von der Befehlszeile aus übergeben.

`client_bootstrap`

Das Common-Runtime-Objekt, das Socket-Kommunikationsaktivitäten verarbeitet

In der Beispiel-App wird dieses Objekt vor dem Aufruf zum `mqtt_connection_builder.websockets_with_default_aws_signing` instanziiert.

`region`

Die AWS Signaturregion, die für die Signature V4-Authentifizierung verwendet wird. In `pubsub.py` übergibt sie den in der Befehlszeile eingegebenen Parameter.

In der Beispiel-App wird dieser Wert von der Befehlszeile aus übergeben.

`credentials_provider`

Die für die Authentifizierung bereitgestellten AWS Anmeldeinformationen

In der Beispiel-App wird dieses Objekt vor dem Aufruf zu `mqtt_connection_builder.websockets_with_default_aws_signing` instanziiert.

`websocket_proxy_options`

HTTP-Proxyoptionen, wenn Sie einen Proxyhost verwenden

In der Beispiel-App wird dieser Wert vor dem Aufruf zu `mqtt_connection_builder.websockets_with_default_aws_signing` initialisiert.

`ca_filepath`

Der Pfad zur Root-CA-Datei Nur erforderlich, wenn der MQTT-Server ein Zertifikat verwendet, das sich noch nicht in Ihrem Trust Store befindet.

In der Beispiel-App wird dieser Wert von der Befehlszeile aus übergeben.

`on_connection_interrupted`, `on_connection_resumed`

Die Callback-Funktionen, die aufgerufen werden, wenn die Verbindung des Geräts unterbrochen und wieder aufgenommen wird

`client_id`

Die ID, die dieses Gerät eindeutig in der AWS-Region identifiziert.

In der Beispiel-App wird dieser Wert von der Befehlszeile aus übergeben.

`clean_session`

Wahl der Option, eine neue persistente Sitzung zu starten oder, falls vorhanden, eine Verbindung zu einer bestehenden Sitzung wiederherzustellen

`keep_alive_secs`

Der Keep-Alive-Wert in Sekunden, der in der `CONNECT`-Anfrage gesendet werden soll. In diesem Intervall wird automatisch ein Ping gesendet. Wenn der Server nach dem 1,5-fachen dieses Werts keinen Ping empfängt, wird davon ausgegangen, dass die Verbindung unterbrochen wurde.

HTTPS

Was ist mit HTTPS? AWS IoT Core unterstützt Geräte, die HTTPS-Anfragen veröffentlichen. Aus Sicht der Programmierung senden Geräte HTTPS-Anfragen an AWS IoT Core , so wie jede andere Anwendung es tun würde. Ein Beispiel für ein Python-Programm, das eine HTTP-Nachricht von einem Gerät sendet, finden Sie im HTTPS-Codebeispiel mit der `requests` Python-Bibliothek. In diesem Beispiel wird AWS IoT Core mithilfe von HTTPS eine Nachricht gesendet, sodass diese als MQTT-Nachricht AWS IoT Core interpretiert wird.

AWS IoT Core Unterstützt zwar HTTPS-Anfragen von Geräten, aber überprüfen Sie unbedingt die Informationen dazu, Wählen Sie ein Anwendungsprotokoll für Ihre Gerätekommunikation damit Sie eine fundierte Entscheidung darüber treffen können, welches Protokoll Sie für Ihre Gerätekommunikation verwenden möchten.

Persistente Sitzungen

In der Beispiel-App bedeutet die Einstellung des `clean_session`-Parameters auf `False`, dass die Verbindung dauerhaft sein soll. In der Praxis bedeutet dies, dass die durch diesen Aufruf geöffnete Verbindung erneut eine Verbindung zu einer bestehenden persistenten Sitzung herstellt, sofern eine besteht. Andernfalls wird eine neue persistente Sitzung erstellt und eine Verbindung zu ihr hergestellt.

Bei einer persistenten Sitzung werden Nachrichten, die an das Gerät gesendet werden, vom Message Broker gespeichert, solange das Gerät nicht verbunden ist. Wenn ein Gerät die Verbindung zu einer persistenten Sitzung wieder aufnimmt, sendet der Message Broker alle von dem Gerät abonnierten gespeicherten Nachrichten an das Gerät.

Ohne eine persistente Sitzung empfängt das Gerät keine Nachrichten, die gesendet werden, während das Gerät nicht verbunden ist. Welche Option Sie verwenden, hängt von Ihrer App ab und davon, ob Nachrichten, die auftreten, während ein Gerät nicht angeschlossen ist, übermittelt werden müssen. Weitere Informationen finden Sie unter Persistente MQTT-Sitzungen.

Quality of Service (Servicequalität)

Wenn das Gerät Nachrichten veröffentlicht und abonniert, kann die bevorzugte Servicequalität (QoS) festgelegt werden. AWS IoT unterstützt die QoS-Stufen 0 und 1 für Veröffentlichungs- und Abonnementvorgänge. Weitere Hinweise zu den QoS-Stufen in finden Sie AWS IoT unter. MQTT QoS-(Quality of Service)-Optionen

Die AWS CRT-Laufzeit für Python definiert diese Konstanten für die QoS-Stufen, die sie unterstützt:

Servicequalitätsstufen von Python

QoS-Stufe von MQTT	Vom SDK verwendeter symbolischer Python-Wert	Description
QoS Stufe 0	`mqtt.QoS.AT_MOST_ONCE`	Es wird nur ein Versuch zum Senden der Nachricht unternommen, unabhängig davon, ob sie empfangen wird oder nicht. Die Nachricht wird möglicherweise überhaupt nicht gesendet, z. B. wenn das Gerät nicht angeschlossen ist oder ein Netzwerkfehler vorliegt.
QoS-Stufe 1	`mqtt.QoS.AT_LEAST_ONCE`	Die Nachricht wird wiederholt gesendet, bis eine PUBACK-Bestätigung eingeht.

In der Beispiel-App werden die Veröffentlichungs- und Abonnementanfragen mit einer QoS-Stufe von 1 (`mqtt.QoS.AT_LEAST_ONCE`) gestellt.

- QoS beim Veröffentlichen

  Wenn ein Gerät eine Nachricht mit QoS Stufe 1 veröffentlicht, sendet es die Nachricht wiederholt, bis es eine PUBACK-Antwort vom Message Broker erhält. Wenn das Gerät nicht verbunden ist, wird die Nachricht in die Warteschlange gestellt, damit sie nach dem erneuten Herstellen der Verbindung gesendet werden kann.
- QoS beim Abonnieren

Wenn ein Gerät eine Nachricht mit QoS Stufe 1 abonniert, speichert der Message Broker die Nachrichten, die das Gerät abonniert hat, bis sie an das Gerät gesendet werden können. Der Message Broker sendet die Nachrichten so oft, bis er eine PUBACK-Antwort vom Gerät erhält.

Nachrichten veröffentlichen

Nach erfolgreichem Herstellen einer Verbindung zu AWS IoT Core können Geräte Nachrichten veröffentlichen. Das `pubsub.py`-Beispiel tut dies, indem es die `publish`-Operation des `mqtt_connection`-Objekts aufruft.

```
mqtt_connection.publish(
    topic=args.topic,
    payload=message,
    qos=mqtt.QoS.AT_LEAST_ONCE
)
```

`topic`

Der Themenname der Nachricht, der die Nachricht identifiziert

In der Beispiel-App wird dieser Wert von der Befehlszeile aus übergeben.

`payload`

Die als Zeichenfolge formatierte Nachrichten-Payload (z. B. ein JSON-Dokument)

In der Beispiel-App wird dieser Wert von der Befehlszeile aus übergeben.

Ein JSON-Dokument ist ein gängiges Nutzdatenformat und eines, das von anderen AWS IoT Diensten erkannt wird. Das Datenformat der Nachrichtennutzlast kann jedoch ein beliebiges Format sein, auf das sich Herausgeber und Abonnenten einigen. Andere AWS IoT -Dienste erkennen jedoch für die meisten Operationen nur JSON und in einigen Fällen auch CBOR an.

`qos`

Die QoS-Stufe für diese Nachricht

Nachrichten abonnieren

Um Nachrichten von AWS IoT und anderen Diensten und Geräten zu empfangen, abonnieren Geräte diese Nachrichten anhand ihres Themennamens. Geräte können einzelne Nachrichten abonnieren,

indem sie einen Themennamen angeben, und eine Gruppe von Nachrichten, indem sie einen Themenfilter angeben, der Platzhalterzeichen enthalten kann. Das `pubsub.py`-Beispiel verwendet den hier gezeigten Code, um Nachrichten zu abonnieren und die Callback-Funktionen zu registrieren, um die Nachricht nach dem Empfang zu verarbeiten.

```
subscribe_future, packet_id = mqtt_connection.subscribe(
    topic=args.topic,
    qos=mqtt.QoS.AT_LEAST_ONCE,
    callback=on_message_received
)
subscribe_result = subscribe_future.result()
```

`topic`

Das zu abonnierende Thema. Dies kann ein Themenname oder ein Themenfilter sein.

In der Beispiel-App wird dieser Wert von der Befehlszeile aus übergeben.

`qos`

Ob der Message Broker diese Nachrichten speichern soll, während das Gerät nicht angeschlossen ist.

Ein Wert von `mqtt.QoS.AT_LEAST_ONCE` (QoS Stufe 1) erfordert die Angabe einer persistenten Sitzung (`clean_session=False`), wenn die Verbindung hergestellt wird.

`callback`

Die Funktion, die aufgerufen werden soll, um die abonnierte Nachricht zu verarbeiten.

Die `mqtt_connection.subscribe`-Funktion gibt eine Future- und eine Paket-ID zurück. Wenn die Anfrage nach einem Abonnement erfolgreich initiiert wurde, ist die zurückgegebene Paket-ID größer als 0. Um sicherzustellen, dass das Abonnement vom Message Broker empfangen und registriert wurde, müssen Sie warten, bis das Ergebnis des asynchronen Vorgangs zurückgegeben wird, wie im Codebeispiel gezeigt.

Die Callback-Funktion

Der Callback im `pubsub.py`-Beispiel verarbeitet die abonnierten Nachrichten so, wie das Gerät sie empfängt.

```
def on_message_received(topic, payload, **kwargs):
```

```
    print("Received message from topic '{}': {}".format(topic, payload))
    global received_count
    received_count += 1
    if received_count == args.count:
        received_all_event.set()
```

`topic`

Das Thema der Nachricht

Dies ist der spezifische Themenname der empfangenen Nachricht, auch wenn Sie einen Themenfilter abonniert haben.

`payload`

Die Nutzdaten der Nachricht

Das Format dafür ist anwendungsspezifisch.

`kwargs`

Mögliche zusätzliche Argumente, wie unter `mqtt.Connection.subscribe` beschrieben.

Im `pubsub.py`-Beispiel zeigt nur `on_message_received` das Thema und seine Payload an. Außerdem werden die eingegangenen Nachrichten gezählt, um das Programm zu beenden, nachdem das Limit erreicht wurde.

Ihre App würde das Thema und die Payload auswerten, um zu bestimmen, welche Aktionen ausgeführt werden sollen.

Trennen und erneutes Anschließen des Geräts

Das `pubsub.py`-Beispiel enthält Callback-Funktionen, die aufgerufen werden, wenn das Gerät getrennt wird und wenn die Verbindung wiederhergestellt wird. Welche Aktionen Ihr Gerät bei diesen Ereignissen ausführt, ist anwendungsspezifisch.

Wenn ein Gerät zum ersten Mal eine Verbindung herstellt, muss es Themen abonnieren, um empfangen zu können. Wenn die Sitzung eines Geräts beim erneuten Herstellen der Verbindung vorhanden ist, werden seine Abonnements wiederhergestellt, und alle gespeicherten Nachrichten aus diesen Abonnements werden nach dem erneuten Herstellen der Verbindung an das Gerät gesendet.

Wenn die Sitzung eines Geräts beim erneuten Herstellen der Verbindung nicht mehr besteht, muss das Gerät seine Abonnements erneut abonnieren. Persistente Sitzungen haben eine begrenzte Lebensdauer und können ablaufen, wenn das Gerät zu lange keine Verbindung hat.

Verbinde dein Gerät und kommuniziere mit AWS IoT Core

In diesem Abschnitt finden Sie einige Übungen, mit denen Sie die verschiedenen Aspekte der Verbindung Ihres Geräts mit AWS IoT Core kennenlernen können. Für diese Übungen verwenden Sie den MQTT-Testclient in der AWS IoT -Konsole, um zu sehen, was Ihr Gerät veröffentlicht, und um Nachrichten auf Ihrem Gerät zu veröffentlichen. Diese Übungen verwenden das `pubsub.py`-Beispiel aus dem AWS IoT -Device SDK v2 für Python und bauen auf Ihren Erfahrungen mit Erste-Schritte-Tutorials-Tutorials auf.

In diesem Abschnitt führen Sie folgende Schritte aus:

- Abonnieren von Themenfiltern mit Platzhaltern
- Abonnements mit Themenfiltern verarbeiten
- Veröffentlichen von Nachrichten vom Gerät.

Für diese Übungen beginnen Sie mit dem `pubsub.py`-Beispielprogramm.

> **Note**
>
> Bei diesen Übungen wird davon ausgegangen, dass Sie die Erste-Schritte-Tutorials-Tutorials abgeschlossen haben und das Terminalfenster für das Gerät aus diesem Tutorial verwenden.

Abonnieren von Themenfiltern mit Platzhaltern

In dieser Übung ändern Sie die Befehlszeile, die verwendet wird, um `pubsub.py` zum Abonnieren eines Themenfilters mit Platzhaltern aufzurufen, und verarbeiten die empfangenen Nachrichten auf der Grundlage des Themas der Nachricht.

Ablauf der Übung

Stellen Sie sich für diese Übung vor, dass das Gerät über eine Temperatursteuerung und eine Lichtsteuerung verfügt. Es verwendet diese Themennamen, um die Nachrichten zu diesen Themen zu identifizieren.

1. Bevor Sie mit der Übung beginnen, versuchen Sie, diesen Befehl aus den Erste-Schritte-Tutorials-Tutorials auf dem Gerät auszuführen, um sicherzustellen, dass alles für die Übung bereit ist.

```
cd ~/aws-iot-device-sdk-python-v2/samples
python3 pubsub.py --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/
certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint
```

   Sie sollten dieselbe Ausgabe wie im Tutorial Erste Schritte sehen.

2. Ändern Sie für diese Übung diese Befehlszeilen-Parameter.

Action	Befehlszeilen-Parameter	Auswirkung
hinzufügen	--message ""	Konfigurieren Sie `pubsub.py` so, dass nur zugehört wird.
hinzufügen	--count 2	Beenden Sie das Programm, nachdem Sie zwei Nachricht en erhalten haben
ändern	--topic `device/+/details`	Definieren Sie den Themenfil ter, den Sie abonnieren möchten

   Wenn Sie diese Änderungen an der ersten Befehlszeile vornehmen, wird diese Befehlszeile angezeigt. Geben Sie diesen Befehl in das Terminalfenster Ihres Geräts ein.

```
python3 pubsub.py --message "" --count 2 --topic device/+/details --ca_file
 ~/certs/Amazon-root-CA-1.pem --cert ~/certs/device.pem.crt --key ~/certs/
private.pem.key --endpoint your-iot-endpoint
```

   Das Programm sollte wie folgt aussehen:

```
Connecting to a3qexamplesffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-24d7cdcc-cc01-458c-8488-2d05849691e1'...
Connected!
```

```
Subscribing to topic 'device/+/details'...
Subscribed with QoS.AT_LEAST_ONCE
Waiting for all messages to be received...
```

Wenn Sie so etwas auf Ihrem Terminal sehen, ist Ihr Gerät bereit und wartet auf Nachrichten, bei denen die Themennamen mit `device` beginnen und mit `/detail` enden. Lassen Sie uns das also testen.

3. Hier sind ein paar Nachrichten, die Ihr Gerät möglicherweise empfängt.

Themenname	Nachrichten-Payload
`device/temp/details`	`{ "desiredTemp": 20, "currentTemp": 15 }`
`device/light/details`	`{ "desiredLight": 100, "currentLight": 50 }`

4. Senden Sie mit dem MQTT-Testclient in der AWS IoT Konsole die im vorherigen Schritt beschriebenen Nachrichten an Ihr Gerät.

   a. Öffnen Sie den MQTT-Testclient in der AWS IoT Konsole.

   b. Geben Sie unter Abonnieren eines Themas im Feld Abonnementthema den Themenfilter **`device/+/details`** ein und wählen Sie anschließend Thema Abonnieren aus.

   c. Wählen Sie in der Spalte Abonnements des MQTT-Testclients Gerät/+/Details aus.

   d. Gehen Sie für jedes der Themen in der vorherigen Tabelle im MQTT-Testclient wie folgt vor:

      1. Geben Sie unter Veröffentlichen den Wert aus der Spalte Themenname in der Tabelle ein.
      2. Geben Sie in das Feld Nachrichten-Payload unter dem Themennamen den Wert aus der Spalte Nachrichten-Payload in der Tabelle ein.
      3. Sehen Sie sich das Terminalfenster an, in dem `pubsub.py` läuft, und wählen Sie im MQTT-Testclient die Option Für Thema veröffentlichen.

      Sie sollten im Terminalfenster sehen, dass die Nachricht von `pubsub.py` empfangen wurde.

Ergebnis der Übung

Damit hat `pubsub.py` die Nachrichten mithilfe eines Themenfilters mit Platzhaltern abonniert, sie empfangen und im Terminalfenster angezeigt. Beachten Sie, dass Sie einen einzelnen Themenfilter abonniert haben und die Callback-Funktion aufgerufen wurde, um Nachrichten mit zwei unterschiedlichen Themen zu verarbeiten.

Abonnements mit Themenfiltern verarbeiten

Aufbauend auf der vorherigen Übung ändern Sie die `pubsub.py` Beispiel-App, um die Nachrichtenthemen auszuwerten und die abonnierten Nachrichten auf der Grundlage des Themas zu verarbeiten.

Ablauf der Übung

Um das Thema der Nachricht zu bewerten,

1. Kopieren Sie `pubsub.py` in `pubsub2.py`.
2. Öffnen Sie `pubsub2.py` in Ihrem bevorzugten Texteditor oder IDE.
3. Suchen Sie in `pubsub2.py` nach der `on_message_received`-Funktion.
4. Fügen Sie in `on_message_received` den folgenden Code nach der Zeile ein, die mit `print("Received message` beginnt, und vor der Zeile, die mit `global received_count` beginnt.

```
    topic_parsed = False
    if "/" in topic:
        parsed_topic = topic.split("/")
        if len(parsed_topic) == 3:
            # this topic has the correct format
            if (parsed_topic[0] == 'device') and (parsed_topic[2] == 'details'):
                # this is a topic we care about, so check the 2nd element
                if (parsed_topic[1] == 'temp'):
                    print("Received temperature request: {}".format(payload))
                    topic_parsed = True
                if (parsed_topic[1] == 'light'):
                    print("Received light request: {}".format(payload))
                    topic_parsed = True
    if not topic_parsed:
        print("Unrecognized message topic.")
```

5. Speichern Sie Ihre Änderungen und führen Sie das geänderte Programm mithilfe dieser Befehlszeile aus.

```
python3 pubsub2.py --message "" --count 2 --topic device/+/details --ca_file
 ~/certs/Amazon-root-CA-1.pem --cert ~/certs/device.pem.crt --key ~/certs/
private.pem.key --endpoint your-iot-endpoint
```

6. Öffnen Sie in der AWS IoT Konsole den MQTT-Testclient.
7. Geben Sie unter Abonnieren eines Themas im Feld Abonnementthema den Themenfilter **`device/+/details`** ein und wählen Sie anschließend Thema Abonnieren aus.
8. Wählen Sie in der Spalte Abonnements des MQTT-Testclients die Option Gerät/+/Details aus.
9. Gehen Sie für jedes der Themen in dieser Tabelle im MQTT-Testclient wie folgt vor:

Themenname	Nachrichten-Payload
`device/temp/details`	`{ "desiredTemp": 20, "currentTemp": 15 }`
`device/light/details`	`{ "desiredLight": 100, "currentLight": 50 }`

   1. Geben Sie unter Veröffentlichen den Wert aus der Spalte Themenname in der Tabelle ein.
   2. Geben Sie in das Feld Nachrichten-Payload unter dem Themennamen den Wert aus der Spalte Nachrichten-Payload in der Tabelle ein.
   3. Sehen Sie sich das Terminalfenster an, in dem `pubsub.py` läuft, und wählen Sie im MQTT-Testclient die Option Für Thema veröffentlichen.

   Sie sollten im Terminalfenster sehen, dass die Nachricht von `pubsub.py` empfangen wurde.

Sie sollten in Ihrem Terminalfenster etwas Ähnliches sehen.

```
Connecting to a3qexamplesffp-ats.iot.us-west-2.amazonaws.com with client ID 'test-
af794be0-7542-45a0-b0af-0b0ea7474517'...
Connected!
Subscribing to topic 'device/+/details'...
Subscribed with QoS.AT_LEAST_ONCE
```

```
Waiting for all messages to be received...
Received message from topic 'device/light/details': b'{ "desiredLight": 100,
 "currentLight": 50 }'
Received light request: b'{ "desiredLight": 100, "currentLight": 50 }'
Received message from topic 'device/temp/details': b'{ "desiredTemp": 20,
 "currentTemp": 15 }'
Received temperature request: b'{ "desiredTemp": 20, "currentTemp": 15 }'
2 message(s) received.
Disconnecting...
Disconnected!
```

Ergebnis der Übung

In dieser Übung haben Sie Code hinzugefügt, damit die Beispiel-App mehrere Nachrichten in der Callback-Funktion erkennt und verarbeitet. Damit könnte Ihr Gerät Nachrichten empfangen und darauf reagieren.

Ihr Gerät kann auch mehrere Nachrichten empfangen und verarbeiten, indem es verschiedene Nachrichten separat abonniert und jedem Abonnement eine eigene Callback-Funktion zuweist.

Veröffentlichen von Nachrichten vom Gerät.

Sie können die Beispiel-App pubsub.py verwenden, um Nachrichten von Ihrem Gerät aus zu veröffentlichen. Nachrichten werden zwar so veröffentlicht, wie sie sind, aber die Nachrichten können nicht als JSON-Dokumente gelesen werden. In dieser Übung wird die Beispiel-App so geändert, dass sie JSON-Dokumente in der Nachrichtennutzlast veröffentlichen kann, von der gelesen werden kann. AWS IoT Core

Ablauf der Übung

In dieser Übung wird die folgende Nachricht zusammen mit dem `device/data`-Thema gesendet.

```
{
    "timestamp": 1601048303,
    "sensorId": 28,
    "sensorData": [
        {
        "sensorName": "Wind speed",
        "sensorValue": 34.2211224
        }
    ]
}
```

So bereiten Sie Ihren MQTT-Testclient für die Überwachung der Nachrichten aus dieser Übung vor

1. Geben Sie unter Abonnieren eines Themas im Feld Abonnementthema den Themenfilter **`device/data`** ein und wählen Sie anschließend Thema Abonnieren aus.
2. Wählen Sie in der Spalte Abonnements des MQTT-Testclients die Option. device/data
3. Lassen Sie das Fenster des MQTT-Testclients geöffnet, um auf Nachrichten von Ihrem Gerät zu warten.

Um JSON-Dokumente mit der Beispiel-App „pubsub.py“ zu senden,

1. Kopieren Sie auf dem Gerät `pubsub.py` nach `pubsub3.py`.
2. Bearbeiten Sie `pubsub3.py`, um die Formatierung der zu veröffentlichenden Nachrichten zu ändern.
   a. Öffnen Sie `pubsub3.py` in einem Texteditor.
   b. Suchen Sie diese Codezeile:

      ```
      message = "{} [{}]".format(message_string, publish_count)
      ```

   c. Ändern Sie sie in:

      ```
      message = "{}".format(message_string)
      ```

   d. Suchen Sie diese Codezeile:

      ```
      message_json = json.dumps(message)
      ```

   e. Ändern Sie sie in:

      ```
      message = "{}".json.dumps(json.loads(message))
      ```

   f. Speichern Sie Ihre Änderungen.
3. Führen Sie auf dem Gerät diesen Befehl aus, um die Nachricht zweimal zu senden.

   ```
   python3 pubsub3.py  --ca_file ~/certs/Amazon-root-CA-1.pem  --cert ~/certs/
   device.pem.crt  --key ~/certs/private.pem.key  --topic device/data  --count 2 --
   message '{"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
    speed","sensorValue":34.2211224}]}'  --endpoint your-iot-endpoint
   ```

4. Prüfen Sie im MQTT-Testclient, ob er das JSON-Dokument in der Nachrichten-Payload wie folgt interpretiert und formatiert hat:



`pubsub3.py` abonniert standardmäßig auch die Nachrichten, die es sendet. Sie sollten sehen können, dass es die Nachrichten im Ausgabefenster der App empfangen hat. Das Terminalfenster sollte etwa folgendermaßen aussehen.

```
Connecting to a3qEXAMPLEsffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-5cff18ae-1e92-4c38-a9d4-7b9771afc52f'...
Connected!
Subscribing to topic 'device/data'...
Subscribed with QoS.AT_LEAST_ONCE
Sending 2 message(s)
Publishing message to topic 'device/data':
 {"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}
Received message from topic 'device/data':
 b'{"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}'
Publishing message to topic 'device/data':
 {"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}
Received message from topic 'device/data':
 b'{"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}'
2 message(s) received.
Disconnecting...
Disconnected!
```

Ergebnis der Übung

Auf diese Weise kann Ihr Gerät Nachrichten generieren, an die Sie senden können, AWS IoT Core um die grundlegende Konnektivität zu testen und Gerätenachrichten AWS IoT Core zur Verarbeitung bereitzustellen. Sie könnten diese App beispielsweise verwenden, um Testdaten von Ihrem Gerät zu senden, um AWS IoT Regelaktionen zu testen.

Überprüfen Sie die Ergebnisse.

Die Beispiele in diesem Tutorial haben Ihnen praktische Erfahrungen mit den Grundlagen vermittelt, wie Geräte mit einem grundlegenden AWS IoT Core Teil Ihrer AWS IoT Lösung kommunizieren können. Wenn Ihre Geräte kommunizieren können AWS IoT Core, können sie Nachrichten an AWS Dienste und andere Geräte weitergeben, auf denen sie agieren können. Ebenso können AWS Dienste und andere Geräte Informationen verarbeiten, die dazu führen, dass Nachrichten an Ihre Geräte zurückgesendet werden.

Wenn Sie bereit sind, AWS IoT Core weitere Informationen zu erhalten, probieren Sie diese Tutorials aus:

- the section called “Senden einer Amazon SNS-Benachrichtigung”
- the section called “Speichern von Gerätedaten in einer DynamoDB-Tabelle”
- the section called “Formatieren einer Benachrichtigung mithilfe einer AWS Lambda Funktion”

Tutorial: Verwenden des AWS IoT Device SDK for Embedded C

In diesem Abschnitt wird beschrieben, wie Sie das ausführen AWS IoT Device SDK for Embedded C.

Verfahren in diesem Abschnitt

- Schritt 1: Installieren Sie den AWS IoT Device SDK for Embedded C
- Schritt 2: Konfigurieren der Beispielanwendung
- Schritt 3: So laden Sie die Beispielanwendung herunter und führen sie aus

Schritt 1: Installieren Sie den AWS IoT Device SDK for Embedded C

Das AWS IoT Device SDK for Embedded C richtet sich in der Regel an Geräte mit eingeschränkten Ressourcen, die eine optimierte C-Laufzeitumgebung benötigen. Sie können das SDK auf jedem Betriebssystem verwenden und es auf jedem Prozessortyp hosten (z. B. MCUs und MPUs). Wenn

Ihnen mehr Speicher- und Verarbeitungsressourcen zur Verfügung stehen, empfehlen wir Ihnen, eines der übergeordneten AWS IoT Device- und Mobile-SDKs zu verwenden (z. B. C++ JavaScript, Java und Python).

Im Allgemeinen AWS IoT Device SDK for Embedded C ist das für Systeme vorgesehen, die MCUs oder Low-End-MPUs verwenden, auf denen eingebettete Betriebssysteme ausgeführt werden. Für das Programmierbeispiel in diesem Abschnitt gehen wir davon aus, dass Ihr Gerät Linux verwendet.

Example

1. Laden Sie das von AWS IoT Device SDK for Embedded C auf Ihr Gerät herunter. GitHub

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-embedded-c.git --recurse-
   submodules
   ```

   Dadurch wird ein Verzeichnis mit dem Namen `aws-iot-device-sdk-embedded-c` im aktuellen Verzeichnis erstellt.
2. Navigieren Sie zu diesem Verzeichnis und machen Sie sich mit der neuesten Version vertraut. Bitte sehen Sie sich Github an. com/awsC/tags/aws-iot-device-sdk-embedded- für das neueste Release-Tag.

   ```
   cd aws-iot-device-sdk-embedded-c
   git checkout latest-release-tag
   ```
3. Installieren Sie die OpenSSL-Version 1.1.0 oder höher. Die OpenSSL-Entwicklungsbibliotheken werden in der Regel „libssl-dev“ oder „openssl-devel“ genannt, wenn sie über einen Paketmanager installiert werden.

   ```
   sudo apt-get install libssl-dev
   ```

Schritt 2: Konfigurieren der Beispielanwendung

Das beinhaltet Beispielanwendungen, die Sie AWS IoT Device SDK for Embedded C ausprobieren können. Der Einfachheit halber wird in diesem Tutorial die `mqtt_demo_mutual_auth` Anwendung verwendet, die veranschaulicht, wie Sie eine Verbindung zum AWS IoT Core Message Broker herstellen und MQTT-Themen abonnieren und veröffentlichen.

1. Kopieren Sie das Zertifikat und den privaten Schlüssel, den Sie in Erste Schritte mit AWS IoT Core Tutorials erstellt haben, in das Verzeichnis `build/bin/certificates`.

> **Note**
>
> Geräte- und CA-Stammzertifikate laufen ab oder können widerrufen werden. Wenn diese Zertifikate ablaufen oder widerrufen werden, müssen Sie ein neues CA-Zertifikat oder einen privaten Schlüssel und ein Gerätezertifikat auf Ihr Gerät kopieren.

2. Sie müssen das Beispiel mit Ihrem persönlichen AWS IoT Core Endpunkt, Ihrem privaten Schlüssel, Ihrem Zertifikat und Ihrem Root-CA-Zertifikat konfigurieren. Navigieren Sie zum `aws-iot-device-sdk-embedded-c/demos/mqtt/mqtt_demo_mutual_auth` Verzeichnis .

   Wenn Sie das AWS CLI installiert haben, können Sie diesen Befehl verwenden, um die Endpunkt-URL Ihres Kontos zu ermitteln.

   ```
   aws iot describe-endpoint --endpoint-type iot:Data-ATS
   ```

   Wenn Sie das nicht AWS CLI installiert haben, öffnen Sie Ihre AWS IoT Konsole. Wählen Sie im Navigationsbereich Verwalten und dann Objekte aus. Wählen Sie das IoT-Objekt für Ihr Gerät und dann Interagieren aus. Ihr Endpunkt wird im Abschnitt HTTPS auf der Seite mit den Objektdetails angezeigt.

3. Öffnen Sie die Datei `demo_config.h` und aktualisieren Sie die Werte für Folgendes:

   AWS_IOT_ENDPOINT

   Ihr persönlicher Endpunkt.

   CLIENT_CERT_PATH

   Der Dateipfad Ihres Zertifikats, zum Beispiel `certificates/device.pem.crt"`.

   CLIENT_PRIVATE_KEY_PATH

   Der Dateiname Ihres privaten Schlüssels, zum Beispiel `certificates/private.pem.key`.

   Beispiel:

   ```
   // Get from demo_config.h
   // ================================================
   ```

```
#define AWS_IOT_ENDPOINT                "my-endpoint-ats.iot.us-
east-1.amazonaws.com"
#define AWS_MQTT_PORT                   8883
#define CLIENT_IDENTIFIER               "testclient"
#define ROOT_CA_CERT_PATH               "certificates/AmazonRootCA1.crt"
#define CLIENT_CERT_PATH                "certificates/my-device-cert.pem.crt"
#define CLIENT_PRIVATE_KEY_PATH         "certificates/my-device-private-key.pem.key"
// ================================================
```

4. Überprüfen Sie mithilfe dieses Befehls, ob CMake auf Ihrem Gerät installiert ist.

   ```
   cmake --version
   ```

   Wenn die Versionsinformationen für den Compiler angezeigt werden, können Sie mit dem nächsten Abschnitt fortfahren.

   Wenn Sie eine Fehlermeldung erhalten oder keine Informationen angezeigt werden, müssen Sie das CMake-Paket mit diesem Befehl installieren.

   ```
   sudo apt-get install cmake
   ```

   Führen Sie den Befehl cmake --version erneut aus und vergewissern Sie sich, dass CMake installiert wurde und Sie fortfahren können.

5. Überprüfen Sie mithilfe dieses Befehls, ob die Entwicklungstools auf Ihrem Gerät installiert sind.

   ```
   gcc --version
   ```

   Wenn die Versionsinformationen für den Compiler angezeigt werden, können Sie mit dem nächsten Abschnitt fortfahren.

   Wenn Sie eine Fehlermeldung erhalten oder keine Compilerinformationen angezeigt werden, müssen Sie das Paket `build-essential` mit diesem Befehl installieren.

   ```
   sudo apt-get install build-essential
   ```

   Führen Sie den Befehl gcc --version erneut aus und vergewissern Sie sich, dass die Build-Tools installiert wurden und Sie fortfahren können.

Schritt 3: So laden Sie die Beispielanwendung herunter und führen sie aus

In diesem Verfahren wird erklärt, wie Sie die `mqtt_demo_mutual_auth` Anwendung auf Ihrem Gerät generieren und sie mithilfe von mit der AWS IoT Konsole verbinden. AWS IoT Device SDK for Embedded C

Um das auszuführen AWS IoT Device SDK for Embedded C Beispielanwendungen

1. Navigieren Sie zu `aws-iot-device-sdk-embedded-c` und erstellen Sie ein Build-Verzeichnis.

   ```
   mkdir build && cd build
   ```

2. Geben Sie den folgenden Befehl ein, um die zum Erstellen benötigten Makefiles zu generieren:

   ```
   cmake ..
   ```

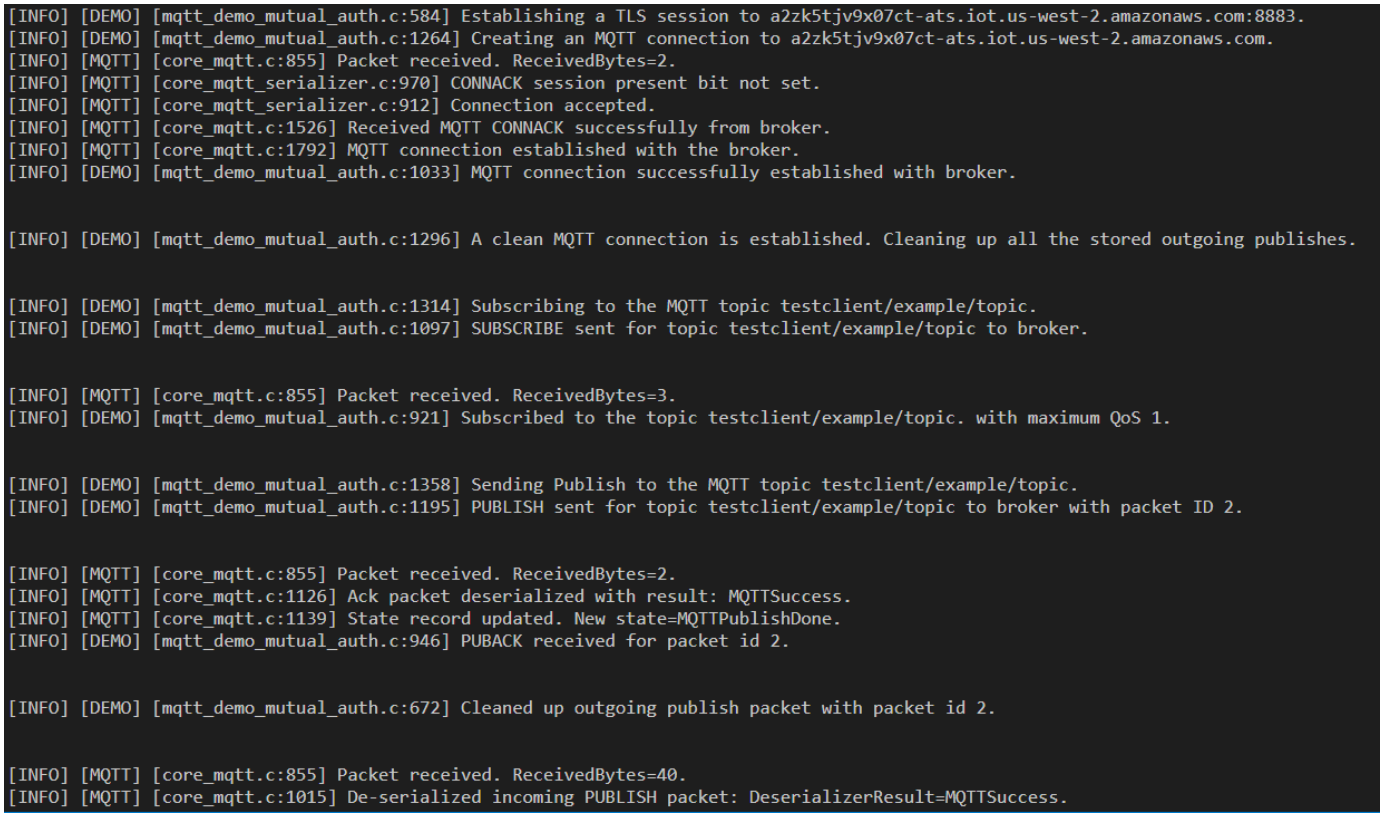
3. Geben Sie den folgenden Befehl ein, um die ausführbare App-Datei zu erstellen:

   ```
   make
   ```

4. Führen Sie die App `mqtt_demo_mutual_auth` mit diesem Befehl aus.

   ```
   cd bin
   ./mqtt_demo_mutual_auth
   ```

   Die Ausgabe sollte folgendermaßen oder ähnlich aussehen:



Ihr Gerät ist jetzt mit der AWS IoT Verwendung von verbunden AWS IoT Device SDK for Embedded C.

Sie können die AWS IoT Konsole auch verwenden, um die MQTT-Nachrichten anzuzeigen, die die Beispiel-App veröffentlicht. Weitere Informationen zur Verwendung des MQTT-Clients in der AWS IoT -Konsole finden Sie unter the section called “MQTT-Nachrichten anzeigen mit dem AWS IoT MQTT-Client”.

Erstellen AWS IoT Regeln zum Weiterleiten von Gerätedaten an andere Dienste

In diesen Tutorials erfahren Sie, wie Sie mithilfe einiger der gängigsten Regelaktionen AWS IoT Regeln erstellen und testen.

AWS IoT Regeln senden Daten von Ihren Geräten an andere AWS Dienste. Sie warten auf bestimmte MQTT-Nachrichten, formatieren die Daten in den Nachrichtennutzlasten und senden das Ergebnis an andere AWS Dienste.

Wir empfehlen Ihnen, diese in der Reihenfolge auszuprobieren, in der sie hier angezeigt werden, auch wenn Sie eine Regel erstellen möchten, die eine Lambda-Funktion oder etwas komplexeres verwendet. Die Tutorials werden in der Reihenfolge von einfach bis komplex präsentiert. Sie stellen schrittweise neue Konzepte vor, damit Sie sich mit den Konzepten vertraut machen können, die Sie zum Erstellen von Regelaktionen verwenden können, für die es kein spezielles Tutorial gibt.

> **Note**
>
> AWS IoT Regeln helfen Ihnen dabei, die Daten von Ihren IoT-Geräten an andere AWS Dienste zu senden. Um dies erfolgreich zu tun, benötigen Sie jedoch Grundkenntnisse der anderen Dienste, an die Sie Daten senden möchten. Diese Tutorials enthalten zwar die notwendigen Informationen, um die Aufgaben zu erledigen, aber es könnte hilfreich sein, mehr über die Dienste zu erfahren, an die Sie Daten senden möchten, bevor Sie sie in Ihrer Lösung verwenden. Eine ausführliche Erläuterung der anderen AWS Dienste würde den Rahmen dieser Tutorials sprengen.

Überblick über Tutorial-Szenarien

Das Szenario für diese Tutorials ist das eines Wettersensorgeräts, das seine Daten regelmäßig veröffentlicht. In diesem imaginären System gibt es viele solcher Sensorgeräte. Die Tutorials in diesem Abschnitt konzentrieren sich jedoch auf ein einzelnes Gerät und zeigen, wie Sie mehrere Sensoren unterbringen können.

Die Tutorials in diesem Abschnitt zeigen Ihnen, wie Sie mithilfe von AWS IoT Regeln die folgenden Aufgaben mit diesem imaginären System von Wettersensorgeräten ausführen können.

- Tutorial: Eine MQTT-Nachricht erneut veröffentlichen

  Dieses Tutorial zeigt, wie Sie eine von den Wettersensoren empfangene MQTT-Nachricht als Nachricht, die nur die Sensor-ID und den Temperaturwert enthält, erneut veröffentlichen. Es verwendet nur AWS IoT Core Dienste und demonstriert eine einfache SQL-Abfrage und wie Sie den MQTT-Client verwenden, um Ihre Regel zu testen.
- Tutorial: Senden einer Amazon SNS-Benachrichtigung

  Dieses Tutorial zeigt, wie Sie eine SNS-Nachricht senden, wenn ein Wert von einem Wettersensorgerät einen bestimmten Wert überschreitet. Es baut auf den im vorherigen Tutorial vorgestellten Konzepten auf und erklärt, wie Sie mit einem anderen AWS Dienst, dem Amazon Simple Notification Service (Amazon SNS), arbeiten.

Wenn Sie noch keine Erfahrung mit Amazon SNS haben, lesen Sie die Übungen Erste Schritte, bevor Sie mit dieser Anleitung beginnen.

- Tutorial: Gerätedaten in einer DynamoDB-Tabelle speichern

  Dieses Tutorial zeigt, wie Sie die Daten der Wettersensorgeräte in einer Datenbanktabelle speichern. Es verwendet die Regelabfrageanweisung und die Ersatzvorlagen, um die Nachrichtendaten für den Zieldienst Amazon DynamoDB zu formatieren.

  Wenn Sie mit DynamoDB noch nicht vertraut sind, lesen Sie sich die Übungen Erste Schritte durch, bevor Sie mit diesem Tutorial beginnen.

- Tutorial: Formatieren einer Benachrichtigung mithilfe einer AWS Lambda function

  Dieses Tutorial zeigt, wie Sie eine Lambda-Funktion aufrufen, um die Gerätedaten neu zu formatieren und sie dann als Textnachricht zu senden. Es fügt ein Python-Skript und AWS SDK-Funktionen in einer AWS Lambda Funktion hinzu, um mit den Nachrichtennutzdaten von den Wettersensorgeräten zu formatieren und eine Textnachricht zu senden.

  Wenn Sie Lambda noch nicht kennen, lesen Sie sich die Übungen Erste Schritte durch, bevor Sie mit diesem Tutorial beginnen.

AWS IoT Überblick über die Regeln

In all diesen Tutorials werden AWS IoT Regeln erstellt.

Eine AWS IoT Regel zum Senden der Daten von einem Gerät an einen anderen AWS Dienst verwendet:

- Eine Anweisung zur Regelabfrage, die aus folgenden Elementen besteht:
  - Eine SQL SELECT-Klausel, die die Daten aus der Nachrichtennutzlast auswählt und formatiert
  - Ein Themenfilter (das FROM-Objekt in der Regelabfrageanweisung), der die zu verwendenden Nachrichten identifiziert
  - Eine optionale bedingte Anweisung (eine SQL WHERE-Klausel), die bestimmte Bedingungen festlegt, auf die reagiert werden soll
- Mindestens eine Regelaktion

Geräte veröffentlichen Nachrichten zu MQTT-Themen. Der Themenfilter in der SQL SELECT-Anweisung identifiziert die MQTT-Themen, auf die die Regel angewendet werden soll. Die in der SQL SELECT-Anweisung angegebenen Felder formatieren die Daten aus der Nutzlast der eingehenden MQTT-Nachricht zur Verwendung durch die Aktionen der Regel. Eine vollständige Liste der Regelaktionen finden Sie unter AWS IoT Regelaktionen.

Tutorials in diesem Abschnitt

- Tutorial: Eine MQTT-Nachricht erneut veröffentlichen
- Tutorial: Senden einer Amazon SNS-Benachrichtigung
- Tutorial: Gerätedaten in einer DynamoDB-Tabelle speichern
- Tutorial: Formatieren einer Benachrichtigung mithilfe einer AWS Lambda function

Tutorial: Eine MQTT-Nachricht erneut veröffentlichen

Dieses Tutorial zeigt, wie Sie eine AWS IoT Regel erstellen, die eine MQTT-Nachricht veröffentlicht, wenn eine bestimmte MQTT-Nachricht empfangen wird. Die Nutzlast für eingehende Nachrichten kann durch die Regel geändert werden, bevor sie veröffentlicht wird. Auf diese Weise können Sie Nachrichten erstellen, die auf bestimmte Anwendungen zugeschnitten sind, ohne dass Sie Ihr Gerät oder dessen Firmware ändern müssen. Sie können auch den Filteraspekt einer Regel verwenden, um Nachrichten nur zu veröffentlichen, wenn eine bestimmte Bedingung erfüllt ist.

Die durch eine Regel erneut veröffentlichten Nachrichten verhalten sich wie Nachrichten, die von einem anderen AWS IoT Gerät oder Client gesendet werden. Geräte können die erneut veröffentlichten Nachrichten genauso abonnieren, wie sie jedes andere MQTT-Nachrichtenthema abonnieren können.

Was Sie in diesem Tutorial lernen werden:

- Wie man einfache SQL-Abfragen und Funktionen in einer Regelabfrageanweisung verwendet
- Wie verwende ich den MQTT-Client, um eine Regel zu testen AWS IoT

Für dieses Tutorial brauchen Sie ungefähr 30 Minuten.

In diesem Tutorial führen Sie die folgenden Aktivitäten durch:

- Überprüfen Sie die MQTT-Themen und AWS IoT Regeln
- Schritt 1: Erstellen Sie eine AWS IoT Regel zum erneuten Veröffentlichen einer MQTT-Nachricht

- Schritt 2: Testen Ihrer neuen Regel
- Schritt 3: Überprüfen Sie die Ergebnisse und die nächsten Schritte

Stellen Sie vor Beginn dieses Tutorials sicher, dass Sie über Folgendes verfügen:

- Einrichten AWS-Konto

  Sie benötigen Ihre eigene AWS-Konto AWS IoT Konsole, um dieses Tutorial abzuschließen.
- Überprüft MQTT-Nachrichten anzeigen mit dem AWS IoT MQTT-Client

  Stellen Sie sicher, dass Sie den MQTT-Client verwenden können, um ein Thema zu abonnieren und zu veröffentlichen. In diesem Verfahren werden Sie den MQTT-Client verwenden, um Ihre neue Regel zu testen.

Überprüfen Sie die MQTT-Themen und AWS IoT Regeln

Bevor wir über AWS IoT Regeln sprechen, hilft es, das MQTT-Protokoll zu verstehen.
In IoT-Lösungen bietet das MQTT-Protokoll einige Vorteile gegenüber anderen Netzwerkkommunikationsprotokollen wie HTTP, was es zu einer beliebten Wahl für die Verwendung durch IoT-Geräte macht. In diesem Abschnitt werden die wichtigsten Aspekte von MQTT in Bezug auf dieses Tutorial beschrieben. Informationen darüber, wie MQTT im Vergleich zu HTTP abschneidet, finden Sie unter Wählen Sie ein Anwendungsprotokoll für Ihre Gerätekommunikation.

MQTT-Protokoll

Das MQTT-Protokoll verwendet ein publish/subscribe Kommunikationsmodell mit seinem Host.
Um Daten zu senden, veröffentlichen Geräte Nachrichten, die anhand von Themen identifiziert sind, an den AWS IoT Message Broker. Um Nachrichten vom Message Broker zu empfangen, abonnieren Geräte die Themen, die sie erhalten, indem sie Themenfilter in Abonnementanfragen an den Message Broker senden. Die AWS IoT Regel-Engine empfängt MQTT-Nachrichten vom Message Broker.

AWS IoT Regeln

AWS IoT Regeln bestehen aus einer Regelabfrageanweisung und einer oder mehreren Regelaktionen. Wenn die AWS IoT Regel-Engine eine MQTT-Nachricht empfängt, reagieren diese Elemente wie folgt auf die Nachricht.

- Anweisung zur Regelabfrage

Die Abfrageanweisung der Regel beschreibt die zu verwendenden MQTT-Themen, interpretiert die Daten aus der Nachrichtennutzlast und formatiert die Daten wie in einer SQL-Anweisung beschrieben, die den Anweisungen ähnelt, die in gängigen SQL-Datenbanken verwendet werden. Das Ergebnis der Abfrageanweisung sind die Daten, die an die Aktionen der Regel gesendet werden.

- Regelaktion

  Jede Regelaktion in einer Regel wirkt sich auf die Daten aus, die sich aus der Abfrageanweisung der Regel ergeben. AWS IoT unterstützt viele Regelaktionen. In diesem Tutorial konzentrieren Sie sich jedoch auf die Erneut veröffentlichen Regelaktion, bei der das Ergebnis der Abfrageanweisung als MQTT-Nachricht mit einem bestimmten Thema veröffentlicht wird.

Schritt 1: Erstellen Sie eine AWS IoT Regel zum erneuten Veröffentlichen einer MQTT-Nachricht

Die AWS IoT Regel, die Sie in diesem Tutorial erstellen, abonniert die `device/`*`device_id`*`/data` MQTT-Themen, in denen die ID des Geräts *`device_id`* steht, das die Nachricht gesendet hat. Diese Themen werden durch einen Themenfilter als `device/+/data` beschrieben, wobei es sich bei + um ein Platzhalterzeichen handelt, das einer beliebigen Zeichenfolge zwischen den beiden Schrägstrichen entspricht.

Wenn die Regel eine Nachricht von einem passenden Thema empfängt, veröffentlicht sie die `device_id` und `temperature` Werte erneut als neue MQTT-Nachricht mit dem `device/data/temp` Thema.

Die Nutzlast einer MQTT-Nachricht mit dem `device/22/data` Thema sieht beispielsweise so aus:

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

Die Regel verwendet den `temperature` Wert aus der Nachrichtennutzlast und den `device_id` aus dem Thema und veröffentlicht sie erneut als MQTT-Nachricht mit dem `device/data/temp` Thema und einer Nachrichtennutzlast, die wie folgt aussieht:

```
{
  "device_id": "22",
  "temperature": 28
}
```

Mit dieser Regel abonnieren Geräte, die nur die Geräte-ID und die Temperaturdaten benötigen, das `device/data/temp` Thema, um nur diese Informationen zu erhalten.

Erstellen einer Regel, die eine MQTT-Nachricht erneut veröffentlicht

1. Öffnen Sie den Regel-Hub der AWS IoT Konsole.
2. Wählen Sie unter Regeln die Option Erstellen aus und beginnen Sie mit der Erstellung Ihrer neuen Regel.
3. Gehen Sie im oberen Teil von Regel erstellen wie folgt vor:

   a. Geben Sie im Feld Name den Namen der Regel ein. Für dieses Tutorial nennen Sie es **`republish_temp`**.

      Denken Sie daran, dass ein Regelname innerhalb Ihres Kontos und Ihrer Region eindeutig sein muss und keine Leerzeichen enthalten darf. Wir haben in diesem Namen einen Unterstrich verwendet, um die beiden Wörter im Namen der Regel voneinander zu trennen.

   b. Beschreiben Sie die Regel im Feld Beschreibung.

      Eine aussagekräftige Beschreibung hilft Ihnen dabei, sich daran zu erinnern, was diese Regel bewirkt und warum Sie sie erstellt haben. Die Beschreibung kann so lang wie nötig sein, also seien Sie so detailliert wie möglich.
4. In der Regelabfrageanweisung von Regel erstellen:

   a. Wählen Sie unter SQL-Version verwenden die Option **`2016-03-23`** aus.

   b. Geben Sie im Bearbeitungsfeld Regelabfrageanweisung die folgende Anweisung ein:

      ```
      SELECT topic(2) as device_id, temperature FROM 'device/+/data'
      ```

      Diese Aussage:

      - Hört auf MQTT-Nachrichten mit einem Thema, das dem `device/+/data` Themenfilter entspricht.

- Wählt das zweite Element aus der Themenzeichenfolge aus und weist es dem `device_id` Feld zu.
- Wählt das Wert `temperature` Feld aus der Nachrichtennutzlast aus und weist es dem `temperature` Feld zu.

5. Gehen Sie im Feld Eine oder mehrere Aktionen festlegen wie folgt vor:

   a. Um die Liste der Regelaktionen für diese Regel zu öffnen, wählen Sie Aktion hinzufügen.

   b. Wählen Sie unter Aktion auswählen die Option Nachricht zu einem AWS IoT Thema erneut veröffentlichen aus.

   c. Wählen Sie unten in der Aktionsliste die Option Aktion konfigurieren aus, um die Konfigurationsseite der ausgewählten Aktion zu öffnen.

6. Gehen Sie in Aktion konfigurieren wie folgt vor:

   a. Geben Sie im Feld Thema **`device/data/temp`** ein. Dies ist das MQTT-Thema der Nachricht, die diese Regel veröffentlichen wird.

   b. Wählen Sie unter Qualität der Dienstleistung die Option 0 aus. Die Nachricht wird null oder mehrere Mal zugestellt.

   c. Gehen Sie unter Rolle auswählen oder erstellen, der Sie AWS IoT Zugriff auf die Durchführung dieser Aktion gewähren möchten, wie folgt vor:

      i. Wählen Sie Rolle erstellen aus. Das Dialogfeld Eine neue Rolle erstellen wird geöffnet.

      ii. Geben Sie einen Namen ein, der die neue Rolle beschreibt. Verwenden Sie in diesem Tutorial **`republish_role`**.

      Wenn Sie eine neue Rolle erstellen, werden die richtigen Richtlinien für die Ausführung der Regelaktion erstellt und der neuen Rolle zugeordnet. Wenn Sie das Thema dieser Regelaktion ändern oder diese Rolle in einer anderen Regelaktion verwenden, müssen Sie die Richtlinie für diese Rolle aktualisieren, um das neue Thema oder die neue Aktion zu autorisieren. Um eine bestehende Rolle zu aktualisieren, wählen Sie in diesem Abschnitt die Option Rolle aktualisieren aus.

      iii. Wählen Sie Rolle erstellen aus, um die Rolle zu erstellen und das Dialogfeld zu schließen.

   d. Wählen Sie Aktion hinzufügen, um die Aktion zur Regel hinzuzufügen und zur Seite Regel erstellen zurückzukehren.

7. Die Aktion Nachricht erneut zu einem AWS IoT Thema veröffentlichen ist jetzt unter Eine oder mehrere Aktionen festlegen aufgeführt.

   In der Kachel der neuen Aktion unter Nachricht zu einem AWS IoT Thema erneut veröffentlichen können Sie das Thema sehen, in dem die neue Aktion veröffentlicht wird.

   Dies ist die einzige Regelaktion, die Sie zu dieser Regel hinzufügen werden.

8. Scrollen Sie in Regel erstellen ganz nach unten und wählen Sie Regel erstellen aus, um die Regel zu erstellen und diesen Schritt abzuschließen.

Schritt 2: Testen Ihrer neuen Regel

Um Ihre neue Regel zu testen, verwenden Sie den MQTT-Client, um die von dieser Regel verwendeten MQTT-Nachrichten zu veröffentlichen und zu abonnieren.

Öffnen Sie den MQTT-Client in der AWS IoT Konsole in einem neuen Fenster. Auf diese Weise können Sie die Regel bearbeiten, ohne die Konfiguration Ihres MQTT-Clients zu verlieren. Der MQTT-Client speichert keine Abonnements oder Nachrichtenprotokolle, wenn Sie ihn verlassen, um zu einer anderen Seite in der Konsole zu wechseln.

Um den MQTT-Client zum Testen Ihrer Regel verwenden.

1. Abonnieren Sie im MQTT-Client in der AWS IoT Konsole die Eingabethemen, in diesem Fall `device/+/data`.

   a. Wählen Sie im MQTT-Client unter Abonnements die Option Thema abonnieren aus.

   b. Geben Sie im Abonnementthema das Thema des Eingabethemenfilters **`device/+/data`** ein.

   c. Belassen Sie die übrigen Felder auf ihren Standardeinstellungen.

   d. Wählen Sie Thema abonnieren aus.

      In der Spalte Abonnements wird **`device/+/data`** unter In einem Thema veröffentlichen angezeigt.

2. Abonnieren Sie das Thema, das Ihre Regel veröffentlichen wird: `device/data/temp`.

   a. Wählen Sie unter Abonnements erneut die Option Thema abonnieren aus, und geben Sie unter Abonnementthema das Thema der erneut veröffentlichten Nachricht **`device/data/temp`** ein.

b. Belassen Sie die übrigen Felder auf ihren Standardeinstellungen.

c. Wählen Sie Thema abonnieren aus.

In der Spalte Abonnements wird **`device/data/temp`** unter Geräte/+/Daten angezeigt.

3. Veröffentlichen Sie eine Nachricht zum Eingabethema mit einer bestimmten Geräte-ID, **`device/22/data`**. Sie können keine Beiträge in MQTT-Themen veröffentlichen, die Platzhalterzeichen enthalten.

a. Wählen Sie im MQTT-Client unter Abonnements die Option In einem Thema veröffentlichen.

b. Geben Sie im Feld Veröffentlichen den Namen des Eingabethemas **`device/22/data`** ein.

c. Kopieren Sie die hier gezeigten Beispieldaten und fügen Sie die Beispieldaten in das Bearbeitungsfeld unter dem Themennamen ein.

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

d. Um Ihre MQTT-Nachricht zu senden, wählen Sie Im Thema veröffentlichen.

4. Überprüfen Sie die gesendeten Nachrichten.

a. Im MQTT-Client befindet sich unter Abonnements ein grüner Punkt neben den beiden Themen, die Sie zuvor abonniert haben.

Die grünen Punkte zeigen an, dass eine oder mehrere neue Nachrichten eingegangen sind, seit Sie sie das letzte Mal angesehen haben.

b. Wählen Sie unter Abonnements die Option Gerät/+/Daten aus, um zu überprüfen, ob die Nutzlast der Nachricht mit dem übereinstimmt, was Sie gerade veröffentlicht haben, und wie folgt aussieht:

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
```

```
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

c. Wählen Sie unter Abonnements die Option device/data /temp aus, um zu überprüfen, ob Ihre Payload für erneut veröffentlichte Nachrichten wie folgt aussieht:

```
{
  "device_id": "22",
  "temperature": 28
}
```

Beachten Sie, dass es sich bei dem `device_id` Wert um eine Zeichenfolge in Anführungszeichen handelt und der Wert `temperature` numerisch ist. Das liegt daran, dass die `topic()` Funktion die Zeichenfolge aus dem Themennamen der Eingabenachricht extrahiert hat, während der `temperature` Wert den numerischen Wert aus der Nutzlast der Eingabenachricht verwendet.

Wenn Sie den Wert zu einem numerischen `device_id` Wert machen möchten, ersetzen Sie `topic(2)` in der Anweisung zur Regelabfrage durch:

```
cast(topic(2) AS DECIMAL)
```

Beachten Sie, dass die Umwandlung des `topic(2)` Werts in einen numerischen Wert nur funktioniert, wenn dieser Teil des Themas nur numerische Zeichen enthält.

5. Wenn Sie sehen, dass die richtige Nachricht im device/data /temp-Thema veröffentlicht wurde, hat Ihre Regel funktioniert. Weitere Informationen zur Aktion „Regel erneut veröffentlichen“ finden Sie im nächsten Abschnitt.

   Wenn Sie nicht sehen, dass die richtige Nachricht in den Themen device/+/data oder device/data /temp veröffentlicht wurde, lesen Sie die Tipps zur Problembehandlung.

Problembehebung bei der Regel „Nachricht erneut veröffentlichen“

Hier sind einige Dinge, die Sie überprüfen sollten, falls Sie nicht die erwarteten Ergebnisse sehen.

- Sie haben ein Fehlerbanner

Wenn bei der Veröffentlichung der Eingabemeldung ein Fehler aufgetreten ist, korrigieren Sie diesen Fehler zuerst. Die folgenden Schritte können Ihnen helfen, diesen Fehler zu korrigieren.

- Sie sehen die Eingabenachricht nicht im MQTT-Client

  Jedes Mal, wenn Sie Ihre Eingabenachricht zum `device/22/data` Thema veröffentlichen, sollte diese Nachricht im MQTT-Client erscheinen, wenn Sie den `device/+/data` Themenfilter wie im Verfahren beschrieben abonniert haben.

  Zu überprüfende Dinge

  - Überprüfen Sie den Themenfilter, den Sie abonniert haben

    Wenn Sie das Thema der Eingabenachricht wie im Verfahren beschrieben abonniert haben, sollte Ihnen bei jeder Veröffentlichung eine Kopie der Eingabenachricht angezeigt werden.

    Wenn Sie die Nachricht nicht sehen, überprüfen Sie den Themennamen, den Sie abonniert haben, und vergleichen Sie ihn mit dem Thema, zu dem Sie sie veröffentlicht haben. Bei Themennamen wird Groß- und Kleinschreibung beachtet, und das Thema, das Sie abonniert haben, muss mit dem Thema identisch sein, zu dem Sie die Nachrichtennutzlast veröffentlicht haben.

  - Überprüfen Sie die Funktion zum Veröffentlichen von Nachrichten

    Wählen Sie im MQTT-Client unter Abonnements die Option Gerät/+/Daten aus, überprüfen Sie das Thema der Veröffentlichungsnachricht und wählen Sie dann Im Thema veröffentlichen aus. Sie sollten sehen, dass die Nutzlast der Nachricht aus dem Bearbeitungsfeld unter dem Thema in der Nachrichtenliste erscheinen.

- Sie sehen Ihre erneut veröffentlichte Nachricht nicht im MQTT-Client

  Damit Ihre Regel funktioniert, muss sie über die richtige Richtlinie verfügen, die sie autorisiert, eine Nachricht zu empfangen und erneut zu veröffentlichen, und sie muss die Nachricht empfangen.

  Zu überprüfende Dinge

  - Prüfen Sie die AWS-Region Ihres MQTT-Clients und der Regel, die Sie erstellt haben

    Die Konsole, in der Sie den MQTT-Client ausführen, muss sich in derselben AWS Region befinden wie die von Ihnen erstellte Regel.

  - Überprüfen Sie das Thema der Eingabemeldung in der Regelabfrageanweisung

Damit die Regel funktioniert, muss sie eine Nachricht mit dem Themennamen erhalten, der dem Themenfilter in der FROM-Klausel der Regelabfrageanweisung entspricht.

Überprüfen Sie die Schreibweise des Themenfilters in der Regelabfrageanweisung mit der des Themas im MQTT-Client. Bei Themennamen wird Groß- und Kleinschreibung beachtet, und das Thema der Nachricht muss mit dem Themenfilter in der Regelabfrageanweisung übereinstimmen.

- Überprüfen Sie den Inhalt der Nutzlast der Input-Nachricht

  Damit die Regel funktioniert, muss sie das Datenfeld in der Nachrichtennutzlast finden, das in der SELECT-Anweisung deklariert ist.

  Überprüfen Sie die Schreibweise des `temperature` Felds in der Regelabfrageanweisung mit der Schreibweise der Nachrichtennutzlast im MQTT-Client. Bei Feldnamen wird zwischen Groß- und Kleinschreibung unterschieden, und das `temperature` Feld in der Regelabfrageanweisung muss mit dem `temperature` Feld in der Nachrichtennutzlast identisch sein.

  Stellen Sie sicher, dass das JSON-Dokument in der Nachrichtennutzlast korrekt formatiert ist. Wenn das JSON Fehler enthält, z. B. ein fehlendes Komma, kann die Regel es nicht lesen.

- Überprüfen Sie das Thema der erneut veröffentlichten Nachricht in der Regelaktion

  Das Thema, zu dem die Regelaktion „Erneut veröffentlichen“ die neue Nachricht veröffentlicht, muss mit dem Thema übereinstimmen, das Sie im MQTT-Client abonniert haben.

  Öffnen Sie die Regel, die Sie in der Konsole erstellt haben, und überprüfen Sie das Thema, zu dem die Regelaktion die Nachricht erneut veröffentlicht.

- Überprüfen Sie die Rolle, die von der Regel verwendet wird

  Die Regelaktion muss berechtigt sein, das ursprüngliche Thema zu empfangen und das neue Thema zu veröffentlichen.

  Die Richtlinien, mit denen die Regel autorisiert wird, Nachrichtendaten zu empfangen und erneut zu veröffentlichen, sind spezifisch für die verwendeten Themen. Wenn Sie das Thema ändern, das für die erneute Veröffentlichung der Nachrichtendaten verwendet wird, müssen Sie die Rolle der Regelaktion aktualisieren, damit ihre Richtlinie dem aktuellen Thema entspricht.

Wenn Sie vermuten, dass dies das Problem ist, bearbeiten Sie die Aktion Regel erneut veröffentlichen und erstellen Sie eine neue Rolle. Neue Rollen, die durch die Regelaktion erstellt wurden, erhalten die erforderlichen Autorisierungen, um diese Aktionen auszuführen.

Schritt 3: Überprüfen Sie die Ergebnisse und die nächsten Schritte

In diesem Tutorial

- Sie haben eine einfache SQL-Abfrage und einige Funktionen in einer Regelabfrageanweisung verwendet, um eine neue MQTT-Nachricht zu erzeugen.
- Sie haben eine Regel erstellt, die diese neue Nachricht erneut veröffentlicht hat.
- Sie haben den MQTT-Client verwendet, um Ihre AWS IoT Regel zu testen.

Nächste Schritte

Nachdem Sie einige Nachrichten mit dieser Regel erneut veröffentlicht haben, probieren Sie sie aus, um zu sehen, wie sich Änderungen einiger Aspekte des Tutorials auf die erneut veröffentlichte Nachricht auswirken. Hier sind einige Ideen, die Ihnen den Einstieg erleichtern sollen.

- Ändern Sie das Thema *device_id* der Eingabenachricht und beobachten Sie den Effekt in der erneut veröffentlichten Nachrichtennutzlast.
- Ändern Sie die in der Regelabfrageanweisung ausgewählten Felder und beobachten Sie die Auswirkungen auf die Nutzlast der erneut veröffentlichten Nachricht.
- Probieren Sie das nächste Tutorial in dieser Serie und lernen Sie, wie man Tutorial: Senden einer Amazon SNS-Benachrichtigung.

Die in diesem Tutorial verwendete Aktion Regel erneut veröffentlichen kann Ihnen auch beim Debuggen von Regelabfrageanweisungen helfen. Sie können diese Aktion beispielsweise einer Regel hinzufügen, um zu sehen, wie ihre Regelabfrageanweisung die von ihren Regelaktionen verwendeten Daten formatiert.

Tutorial: Senden einer Amazon SNS-Benachrichtigung

Dieses Tutorial zeigt, wie Sie eine AWS IoT Regel erstellen, die MQTT-Nachrichtendaten an ein Amazon SNS-Thema sendet, sodass sie als SMS-Textnachricht gesendet werden können.

In diesem Tutorial erstellen Sie eine Regel, die Nachrichtendaten von einem Wettersensor an alle Abonnenten eines Amazon SNS-Themas sendet, wenn die Temperatur den in der Regel festgelegten Wert überschreitet. Die Regel erkennt, wenn die gemeldete Temperatur den in der Regel festgelegten Wert überschreitet, und erstellt dann eine neue Nachrichtennutzlast, die nur die Geräte-ID, die gemeldete Temperatur und den Temperaturgrenzwert, der überschritten wurde, enthält. Die Regel sendet die neue Nachrichtennutzlast als JSON-Dokument an ein SNS-Thema, wodurch alle Abonnenten des SNS-Themas benachrichtigt werden.

Was Sie in diesem Tutorial lernen werden:

- So erstellen und testen Sie eine Amazon SNS-Benachrichtigung
- Wie rufe ich eine Amazon SNS-Benachrichtigung von einer Regel aus auf AWS IoT
- Wie man einfache SQL-Abfragen und Funktionen in einer Regelabfrageanweisung verwendet
- Wie verwende ich den MQTT-Client, um eine Regel zu testen AWS IoT

Für dieses Tutorial brauchen Sie ungefähr 30 Minuten.

In diesem Tutorial führen Sie die folgenden Aktivitäten durch:

- Schritt 1: Erstellen Sie ein Amazon SNS-Thema, das eine SMS-Textnachricht sendet
- Schritt 2: Erstellen Sie eine AWS IoT Regel zum Senden der Textnachricht
- Schritt 3: Testen Sie den AWS IoT Regel und Amazon SNS-Benachrichtigung
- Schritt 4: Überprüfen Sie die Ergebnisse und die nächsten Schritte

Stellen Sie vor Beginn dieses Tutorials sicher, dass Sie über Folgendes verfügen:

- Einrichten AWS-Konto

  Sie benötigen Ihre eigene AWS-Konto AWS IoT Konsole, um dieses Tutorial abzuschließen.
- Überprüft MQTT-Nachrichten anzeigen mit dem AWS IoT MQTT-Client

  Stellen Sie sicher, dass Sie den MQTT-Client verwenden können, um ein Thema zu abonnieren und zu veröffentlichen. In diesem Verfahren werden Sie den MQTT-Client verwenden, um Ihre neue Regel zu testen.
- Überprüfte das Amazon Simple Notification Service

Wenn Sie Amazon SNS noch nicht verwendet haben, lesen Sie den Artikel Zugriff für Amazon SNS einrichten. Wenn Sie bereits andere AWS IoT Tutorials abgeschlossen haben, AWS-Konto sollte Ihr Tutorial bereits korrekt konfiguriert sein.

Schritt 1: Erstellen Sie ein Amazon SNS-Thema, das eine SMS-Textnachricht sendet

In diesem Verfahren wird erklärt, wie Sie das Amazon SNS-Thema erstellen, an das Ihr Wettersensor Nachrichtendaten senden kann. Das Amazon SNS-Thema benachrichtigt dann alle Abonnenten per SMS-Textnachricht über die Überschreitung der Temperaturgrenze.

Ein Amazon-SNS-Thema erstellen, das eine SMS-Textnachricht sendet

1. Erstellen Sie ein Amazon-SNS-Thema.
   a. Melden Sie sich bei der Amazon-SNS-Konsole an.
   b. Wählen Sie im linken Navigationsbereich Topics (Themen).
   c. Wählen Sie auf der Seite Topics (Themen) Create New Topic (Neues Thema erstellen) aus.
   d. Wählen Sie unter Details den Standardtyp aus. Standardmäßig erstellt die Konsole ein FIFO-Thema.
   e. Geben Sie im Feld Name den Namen des SNS-Themas ein. Geben Sie für dieses Tutorial **`high_temp_notice`** ein.
   f. Scrollen Sie zum Ende der Seite und wählen Sie Create topic (Erstellen eines Themas) aus.

      In der Konsole wird die Seite Details geöffnet.
2. Erstellen Sie ein Amazon-SNS-Abonnement.

   > **Note**
   >
   > Für die Telefonnummer, die Sie in diesem Abonnement verwenden, können Gebühren für Textnachrichten aufgrund der Nachrichten anfallen, die Sie in diesem Tutorial versenden.

   a. Wählen Sie auf der Detailseite des Themas high_temp_notice die Option Abonnement erstellen.

b. Wählen Sie unter Abonnement erstellen im Abschnitt Details in der Protokoll-Liste die Option SMS aus.

c. Geben Sie im Feld Endpunkt die Nummer eines Telefons ein, das Textnachrichten empfangen kann. Achten Sie darauf, dass Sie es so eingeben, dass es mit einem + beginnt, die Landes- und Ortsvorwahl enthält und keine anderen Satzzeichen enthält.

d. Wählen Sie Create subscription (Abonnement erstellen) aus.

3. Testen Sie die Amazon SNS-Benachrichtigung.

a. Wählen Sie in der Amazon SNS-Konsole im linken Navigationsbereich Themen.

b. Um die Detailseite des Themas zu öffnen, wählen Sie unter Themen in der Liste der Themen den Eintrag high_temp_notice aus.

c. Um die Seite Nachricht im Thema veröffentlichen zu öffnen, wählen Sie auf der Detailseite von high_temp_notice die Option Nachricht veröffentlichen aus.

d. Geben Sie unter Nachricht zum Thema veröffentlichen im Abschnitt Nachrichtentext im Feld Nachrichtentext, der an den Endpunkt gesendet werden soll, eine Kurznachricht ein.

e. Scrollen Sie auf der Seite nach unten und wählen Sie Nachricht veröffentlichen.

f. Vergewissern Sie sich auf dem Telefon mit der Nummer, die Sie zuvor bei der Erstellung des Abonnements verwendet haben, dass die Nachricht empfangen wurde.

Wenn Sie die Testnachricht nicht erhalten haben, überprüfen Sie die Telefonnummer und die Einstellungen Ihres Telefons.

Stellen Sie sicher, dass Sie Testnachrichten von der Amazon SNS-Konsole aus veröffentlichen können, bevor Sie mit dem Tutorial fortfahren.

Schritt 2: Erstellen Sie eine AWS IoT Regel zum Senden der Textnachricht

Die AWS IoT Regel, die Sie in diesem Tutorial erstellen, abonniert die `device/`*`device_id`*`/data` MQTT-Themen, in denen die ID des Geräts *`device_id`* steht, das die Nachricht gesendet hat. Diese Themen werden in einem Themenfilter als `device/+/data` beschrieben, wobei es sich bei + um ein Platzhalterzeichen handelt, das einer beliebigen Zeichenfolge zwischen den beiden Schrägstrichen entspricht. Diese Regel testet auch den Wert des `temperature` Felds in der Nachrichtennutzlast.

Wenn die Regel eine Nachricht von einem passenden Thema empfängt, verwendet sie den *`device_id`* Themennamen, den `temperature` Wert aus der Nachrichtennutzlast, fügt einen

konstanten Wert für das zu testende Limit hinzu und sendet diese Werte als JSON-Dokument an ein Amazon SNS-Benachrichtigungsthema.

Zum Beispiel verwendet eine MQTT-Nachricht vom Wettersensorgerät Nummer 32 das `device/32/data` Thema und hat eine Nachrichtennutzlast, die wie folgt aussieht:

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

Die Regelabfrageanweisung der Regel nimmt den `temperature` Wert aus der Nachrichtennutzlast, den *`device_id`* aus dem Themennamen und fügt den konstanten `max_temperature` Wert hinzu, um eine Nachrichtennutzlast zu senden, die wie folgt aussieht, zum Amazonc SNS-Thema:

```
{
  "device_id": "32",
  "reported_temperature": 38,
  "max_temperature": 30
}
```

Um eine zu erstellen AWS IoT Regel zur Erkennung einer Temperaturüberschreitung und Erstellung der Daten, die an das Amazon SNS-Thema gesendet werden sollen

1. Öffnen Sie den Regel-Hub der AWS IoT Konsole.
2. Wenn dies Ihre erste Regel ist, wählen Sie Erstellen oder Regel erstellen.
3. In Erstellen einer Regel:

   a. Geben Sie unter Name **`temp_limit_notify`** ein.

      Denken Sie daran, dass ein Regelname innerhalb Ihrer Region AWS-Konto und Ihrer Region eindeutig sein muss und keine Leerzeichen enthalten darf. Wir haben in diesem Namen einen Unterstrich verwendet, um die Wörter im Namen der Regel voneinander zu trennen.

   b. Beschreiben Sie die Regel im Feld Beschreibung.

Eine aussagekräftige Beschreibung macht es einfacher, sich daran zu erinnern, was diese Regel bewirkt und warum Sie sie erstellt haben. Die Beschreibung kann so lang wie nötig sein, also seien Sie so detailliert wie möglich.

4. In der Regelabfrageanweisung von Regel erstellen:

   a. Wählen Sie unter SQL-Version verwenden die Option 2016-03-23 aus.

   b. Geben Sie im Bearbeitungsfeld Regelabfrageanweisung die folgende Anweisung ein:

```
SELECT topic(2) as device_id,
    temperature as reported_temperature,
    30 as max_temperature
  FROM 'device/+/data'
  WHERE temperature > 30
```

   Diese Aussage:

   - Lauscht auf MQTT-Nachrichten mit einem Thema, das dem `device/+/data` Themenfilter entspricht und deren `temperature` Wert größer als 30 ist.
   - Wählt das zweite Element aus der Themenzeichenfolge aus und weist es dem `device_id` Feld zu.
   - Wählt das Wert `temperature` Feld aus der Nachrichtennutzlast aus und weist es dem `reported_temperature` Feld zu.
   - Erstellt einen konstanten Wert `30`, der den Grenzwert darstellt, und weist ihn dem `max_temperature` Feld zu.

5. Um die Liste der Regelaktionen für diese Regel zu öffnen, wählen Sie unter Eine oder mehrere Aktionen festlegen die Option Aktion hinzufügen aus.

6. Wählen Sie unter Aktion auswählen die Option Eine Nachricht als SNS-Push-Benachrichtigung senden aus.

7. Um die Konfigurationsseite der ausgewählten Aktion zu öffnen, wählen Sie unten in der Aktionsliste die Option Aktion konfigurieren aus.

8. Gehen Sie in Aktion konfigurieren wie folgt vor:

   a. Wählen Sie unter SNS-Ziel die Option Auswählen aus, suchen Sie Ihr SNS-Thema mit dem Namen high_temp_notice und wählen Sie Auswählen aus.

   b. Wählen Sie unter Nachrichtenformat das Format RAW.

c. Wählen Sie unter Rolle auswählen oder erstellen, um AWS IoT Zugriff auf diese Aktion zu gewähren, die Option Rolle erstellen aus.

d. Geben Sie unter Neue Rolle erstellen im Feld Name einen eindeutigen Namen für die neue Rolle ein. Verwenden Sie für dieses Tutorial **`sns_rule_role`**.

e. Wählen Sie Rolle erstellen aus.

Wenn Sie dieses Tutorial wiederholen oder eine bestehende Rolle wiederverwenden, wählen Sie Rolle aktualisieren, bevor Sie fortfahren. Dadurch wird das Richtliniendokument der Rolle aktualisiert, sodass es mit dem SNS-Ziel funktioniert.

9. Wählen Sie Aktion hinzufügen und kehren Sie zur Seite Regel erstellen zurück.

   In der Kachel der neuen Aktion unter Eine Nachricht als SNS-Push-Benachrichtigung senden finden Sie das SNS-Thema, das Ihre Regel aufrufen wird.

   Dies ist die einzige Regelaktion, die Sie zu dieser Regel hinzufügen werden.

10. Um die Regel zu erstellen und diesen Schritt abzuschließen, scrollen Sie unter Regel erstellen nach unten und wählen Sie Regel erstellen aus.

Schritt 3: Testen Sie den AWS IoT Regel und Amazon SNS-Benachrichtigung

Um Ihre neue Regel zu testen, verwenden Sie den MQTT-Client, um die von dieser Regel verwendeten MQTT-Nachrichten zu veröffentlichen und zu abonnieren.

Öffnen Sie den MQTT-Client in der AWS IoT Konsole in einem neuen Fenster. Auf diese Weise können Sie die Regel bearbeiten, ohne die Konfiguration Ihres MQTT-Clients zu verlieren. Wenn Sie den MQTT-Client verlassen, um zu einer anderen Seite in der Konsole zu wechseln, werden keine Abonnements oder Nachrichtenprotokolle gespeichert.

Um den MQTT-Client zum Testen Ihrer Regel verwenden.

1. Abonnieren Sie im MQTT-Client in der AWS IoT Konsole die Eingabethemen, in diesem Fall `device/+/data`.

   a. Wählen Sie im MQTT-Client unter Abonnements die Option Thema abonnieren aus.

   b. Geben Sie im Abonnementthema das Thema des Eingabethemenfilters **`device/+/data`** ein.

   c. Belassen Sie die übrigen Felder auf ihren Standardeinstellungen.

d. Wählen Sie Thema abonnieren aus.

   In der Spalte Abonnements wird **`device/+/data`** unter In einem Thema veröffentlichen angezeigt.

2. Veröffentlichen Sie eine Nachricht zum Eingabethema mit einer bestimmten Geräte-ID, **`device/32/data`**. Sie können keine Beiträge in MQTT-Themen veröffentlichen, die Platzhalterzeichen enthalten.

   a. Wählen Sie im MQTT-Client unter Abonnements die Option In einem Thema veröffentlichen.

   b. Geben Sie im Feld Veröffentlichen den Namen des Eingabethemas **`device/32/data`** ein.

   c. Kopieren Sie die hier gezeigten Beispieldaten und fügen Sie die Beispieldaten in das Bearbeitungsfeld unter dem Themennamen ein.

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

   d. Wählen Sie In Thema veröffentlichen aus, um Ihre MQTT-Nachricht zu veröffentlichen.

3. Bestätigen Sie, dass die Textnachricht gesendet wurde.

   a. Im MQTT-Client befindet sich unter Abonnements ein grüner Punkt neben dem Thema, das Sie zuvor abonniert haben.

      Der grüne Punkt zeigt an, dass eine oder mehrere neue Nachrichten eingegangen sind, seit Sie sie das letzte Mal angesehen haben.

   b. Wählen Sie unter Abonnements die Option Gerät/+/Daten aus, um zu überprüfen, ob die Nutzlast der Nachricht mit dem übereinstimmt, was Sie gerade veröffentlicht haben, und wie folgt aussieht:

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
```

```
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

c. Überprüfen Sie das Telefon, mit dem Sie das SNS-Thema abonniert haben, und vergewissern Sie sich, dass der Inhalt der Nutzlast der Nachricht so aussieht:

```
{"device_id":"32","reported_temperature":38,"max_temperature":30}
```

Beachten Sie, dass es sich bei dem `device_id` Wert um eine Zeichenfolge in Anführungszeichen handelt und der Wert `temperature` numerisch ist. Das liegt daran, dass die `topic()` Funktion die Zeichenfolge aus dem Themennamen der Eingabenachricht extrahiert hat, während der `temperature` Wert den numerischen Wert aus der Nutzlast der Eingabenachricht verwendet.

Wenn Sie den Wert zu einem numerischen `device_id` Wert machen möchten, ersetzen Sie `topic(2)` in der Anweisung zur Regelabfrage durch:

```
cast(topic(2) AS DECIMAL)
```

Beachten Sie, dass die Umwandlung des `topic(2)` Werts in einen numerischen `DECIMAL` Wert nur funktioniert, wenn dieser Teil des Themas nur numerische Zeichen enthält.

4. Versuchen Sie, eine MQTT-Nachricht zu senden, in der die Temperatur den Grenzwert nicht überschreitet.

   a. Wählen Sie im MQTT-Client unter Abonnements die Option In einem Thema veröffentlichen.

   b. Geben Sie im Feld Veröffentlichen den Namen des Eingabethemas **`device/33/data`** ein.

   c. Kopieren Sie die hier gezeigten Beispieldaten und fügen Sie die Beispieldaten in das Bearbeitungsfeld unter dem Themennamen ein.

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
```

```
  }
}
```

d. Um Ihre MQTT-Nachricht zu senden, wählen Sie Im Thema veröffentlichen.

Sie sollten die Nachricht sehen, die Sie im **`device/+/data`** Abonnement gesendet haben. Da der Temperaturwert jedoch unter der Höchsttemperatur in der Regelabfrageanweisung liegt, sollten Sie keine Textnachricht erhalten.

Wenn Sie nicht das richtige Verhalten feststellen, lesen Sie die Tipps zur Fehlerbehebung.

Fehlerbehebung bei Ihrer SNS-Nachrichtenregel

Hier sind einige Dinge, die Sie überprüfen sollten, falls Sie nicht die erwarteten Ergebnisse sehen.

- Sie haben ein Fehlerbanner

  Wenn bei der Veröffentlichung der Eingabemeldung ein Fehler aufgetreten ist, korrigieren Sie diesen Fehler zuerst. Die folgenden Schritte können Ihnen helfen, diesen Fehler zu korrigieren.
- Sie sehen die Eingabenachricht nicht im MQTT-Client

  Jedes Mal, wenn Sie Ihre Eingabenachricht zum `device/22/data` Thema veröffentlichen, sollte diese Nachricht im MQTT-Client erscheinen, wenn Sie den `device/+/data` Themenfilter wie im Verfahren beschrieben abonniert haben.

  Zu überprüfende Dinge

  - Überprüfen Sie den Themenfilter, den Sie abonniert haben

    Wenn Sie das Thema der Eingabenachricht wie im Verfahren beschrieben abonniert haben, sollte Ihnen bei jeder Veröffentlichung eine Kopie der Eingabenachricht angezeigt werden.

    Wenn Sie die Nachricht nicht sehen, überprüfen Sie den Themennamen, den Sie abonniert haben, und vergleichen Sie ihn mit dem Thema, zu dem Sie sie veröffentlicht haben. Bei Themennamen wird Groß- und Kleinschreibung beachtet, und das Thema, das Sie abonniert haben, muss mit dem Thema identisch sein, zu dem Sie die Nachrichtennutzlast veröffentlicht haben.
  - Überprüfen Sie die Funktion zum Veröffentlichen von Nachrichten

Wählen Sie im MQTT-Client unter Abonnements die Option Gerät/+/Daten aus, überprüfen Sie das Thema der Veröffentlichungsnachricht und wählen Sie dann Im Thema veröffentlichen aus. Sie sollten sehen, dass die Nutzlast der Nachricht aus dem Bearbeitungsfeld unter dem Thema in der Nachrichtenliste erscheinen.

- Sie erhalten keine SMS-Nachricht

  Damit Ihre Regel funktioniert, muss sie über die richtige Richtlinie verfügen, die sie autorisiert, eine Nachricht zu empfangen und eine SNS-Benachrichtigung zu senden, und sie muss die Nachricht empfangen.

  Zu überprüfende Dinge

  - Prüfen Sie das AWS-Region Ihres MQTT-Clients und der Regel, die Sie erstellt haben

    Die Konsole, in der Sie den MQTT-Client ausführen, muss sich in derselben AWS Region befinden wie die von Ihnen erstellte Regel.
  - Überprüfen Sie, ob der Temperaturwert in der Nachrichtennutzlast den Testschwellenwert überschreitet

    Wenn der Temperaturwert kleiner oder gleich 30 ist, wie in der Regelabfrageanweisung definiert, führt die Regel keine ihrer Aktionen aus.
  - Überprüfen Sie das Thema der Eingabemeldung in der Regelabfrageanweisung

    Damit die Regel funktioniert, muss sie eine Nachricht mit dem Themennamen erhalten, der dem Themenfilter in der FROM-Klausel der Regelabfrageanweisung entspricht.

    Überprüfen Sie die Schreibweise des Themenfilters in der Regelabfrageanweisung mit der des Themas im MQTT-Client. Bei Themennamen wird Groß- und Kleinschreibung beachtet, und das Thema der Nachricht muss mit dem Themenfilter in der Regelabfrageanweisung übereinstimmen.
  - Überprüfen Sie den Inhalt der Nutzlast der Input-Nachricht

    Damit die Regel funktioniert, muss sie das Datenfeld in der Nachrichtennutzlast finden, das in der SELECT-Anweisung deklariert ist.

    Überprüfen Sie die Schreibweise des `temperature` Felds in der Regelabfrageanweisung mit der Schreibweise der Nachrichtennutzlast im MQTT-Client. Bei Feldnamen wird zwischen Groß-

und Kleinschreibung unterschieden, und das `temperature` Feld in der Regelabfrageanweisung muss mit dem `temperature` Feld in der Nachrichtennutzlast identisch sein.

Stellen Sie sicher, dass das JSON-Dokument in der Nachrichtennutzlast korrekt formatiert ist. Wenn das JSON Fehler enthält, z. B. ein fehlendes Komma, kann die Regel es nicht lesen.

- Überprüfen Sie das Thema der erneut veröffentlichten Nachricht in der Regelaktion

  Das Thema, zu dem die Regelaktion „Erneut veröffentlichen“ die neue Nachricht veröffentlicht, muss mit dem Thema übereinstimmen, das Sie im MQTT-Client abonniert haben.

  Öffnen Sie die Regel, die Sie in der Konsole erstellt haben, und überprüfen Sie das Thema, zu dem die Regelaktion die Nachricht erneut veröffentlicht.
- Überprüfen Sie die Rolle, die von der Regel verwendet wird

  Die Regelaktion muss berechtigt sein, das ursprüngliche Thema zu empfangen und das neue Thema zu veröffentlichen.

  Die Richtlinien, mit denen die Regel autorisiert wird, Nachrichtendaten zu empfangen und erneut zu veröffentlichen, sind spezifisch für die verwendeten Themen. Wenn Sie das Thema ändern, das für die erneute Veröffentlichung der Nachrichtendaten verwendet wird, müssen Sie die Rolle der Regelaktion aktualisieren, damit ihre Richtlinie dem aktuellen Thema entspricht.

  Wenn Sie vermuten, dass dies das Problem ist, bearbeiten Sie die Aktion Regel erneut veröffentlichen und erstellen Sie eine neue Rolle. Neue Rollen, die durch die Regelaktion erstellt wurden, erhalten die erforderlichen Autorisierungen, um diese Aktionen auszuführen.

Schritt 4: Überprüfen Sie die Ergebnisse und die nächsten Schritte

In diesem Tutorial:

- Sie haben ein Amazon-SNS-Benachrichtigungsthema und ein Abonnement erstellt und getestet.
- Sie haben eine einfache SQL-Abfrage und Funktionen in einer Regelabfrageanweisung verwendet, um eine neue Nachricht für Ihre Benachrichtigung zu erstellen.
- Sie haben eine AWS IoT Regel zum Senden einer Amazon SNS-Benachrichtigung erstellt, die Ihre benutzerdefinierte Nachrichtennutzlast verwendet hat.
- Sie haben den MQTT-Client verwendet, um Ihre Regel zu testen. AWS IoT

Nächste Schritte

Nachdem Sie einige Textnachrichten mit dieser Regel gesendet haben, versuchen Sie, damit zu experimentieren, um zu sehen, wie sich Änderungen einiger Aspekte des Tutorials auf die Nachricht auswirken und wann sie gesendet wird. Hier sind einige Ideen, die Ihnen den Einstieg erleichtern sollen.

- Ändern Sie das Thema *device_id* der Eingabenachricht und beobachten Sie den Effekt im Inhalt der Textnachricht.
- Ändern Sie die in der Regelabfrageanweisung ausgewählten Felder und beobachten Sie die Auswirkungen auf den Inhalt der Textnachricht.
- Ändern Sie den Test in der Regelabfrageanweisung so, dass er auf eine Mindesttemperatur statt auf eine Höchsttemperatur testet. Denken Sie daran, den Namen von `max_temperature` zu ändern!
- Fügen Sie eine Regelaktion zum erneuten Veröffentlichen hinzu, um eine MQTT-Nachricht zu senden, wenn eine SNS-Benachrichtigung gesendet wird.
- Probieren Sie das nächste Tutorial in dieser Serie und lernen Sie, wie man Tutorial: Gerätedaten in einer DynamoDB-Tabelle speichern.

Tutorial: Gerätedaten in einer DynamoDB-Tabelle speichern

Dieses Tutorial zeigt, wie Sie eine AWS IoT Regel erstellen, die Nachrichtendaten an eine DynamoDB-Tabelle sendet.

In diesem Tutorial erstellen Sie eine Regel, die Nachrichtendaten von einem imaginären Wettersensorgerät an eine DynamoDB-Tabelle sendet. Die Regel formatiert die Daten vieler Wettersensoren so, dass sie zu einer einzigen Datenbanktabelle hinzugefügt werden können.

Was Sie in diesem Tutorial lernen werden

- So erstellen Sie eine DynamoDB-Tabelle
- Wie sendet man Nachrichtendaten aus einer Regel an eine DynamoDB-Tabelle AWS IoT
- Wie verwendet man Substitutionsvorlagen in einer Regel AWS IoT
- Wie man einfache SQL-Abfragen und Funktionen in einer Regelabfrageanweisung verwendet
- Wie verwende ich den MQTT-Client, um eine Regel zu testen AWS IoT

Für dieses Tutorial brauchen Sie ungefähr 30 Minuten.

In diesem Tutorial führen Sie die folgenden Aktivitäten durch:

- Schritt 1: Erstellen Sie die DynamoDB-Tabelle für dieses Tutorial
- Schritt 2: Erstellen Sie eine AWS IoT Regel zum Senden von Daten an die DynamoDB-Tabelle
- Schritt 3: Testen Sie AWS IoT Regel und DynamoDB-Tabelle
- Schritt 4: Überprüfen Sie die Ergebnisse und die nächsten Schritte

Stellen Sie vor Beginn dieses Tutorials sicher, dass Sie über Folgendes verfügen:

- Einrichten AWS-Konto

  Sie benötigen Ihre eigene AWS-Konto AWS IoT Konsole, um dieses Tutorial abzuschließen.
- Überprüft MQTT-Nachrichten anzeigen mit dem AWS IoT MQTT-Client

  Stellen Sie sicher, dass Sie den MQTT-Client verwenden können, um ein Thema zu abonnieren und zu veröffentlichen. In diesem Verfahren werden Sie den MQTT-Client verwenden, um Ihre neue Regel zu testen.
- Überprüfte das Amazon DynamoDB -Übersicht

  Wenn Sie DynamoDB noch nicht verwendet haben, lesen Sie Erste Schritte mit DynamoDB, um sich mit den grundlegenden Konzepten und Funktionen von DynamoDB vertraut zu machen.

Schritt 1: Erstellen Sie die DynamoDB-Tabelle für dieses Tutorial

In diesem Tutorial erstellen Sie eine DynamoDB-Tabelle mit diesen Attributen, um die Daten der imaginären Wettersensorgeräte aufzuzeichnen:

- `sample_time` ist ein Primärschlüssel und beschreibt den Zeitpunkt, zu dem die Probe aufgenommen wurde.
- `device_id` ist ein Sortierschlüssel und beschreibt das Gerät, das die Probe bereitgestellt hat
- `device_data` sind die vom Gerät empfangenen und durch die Regelabfrageanweisung formatierten Daten

Erstellen einer DynamoDB-Tabelle für dieses Tutorial

1. Öffnen Sie die DynamoDB-Konsole und wählen Sie Tabelle erstellen aus.
2. Gehen Sie in Tabelle erstellen wie folgt vor:

a. Geben Sie unter Tabellenname den Tabellennamen **`wx_data`** ein.

b. Geben Sie im Feld Partitionsschlüssel **`sample_time`** ein, und wählen Sie **`Number`** in der Optionsliste neben dem Feld aus.

c. Geben Sie im Feld Sortierschlüssel **`device_id`** ein und wählen Sie **`Number`** in der Optionsliste neben dem Feld aus.

d. Wählen Sie unten auf der Seite Erstellen aus.

Sie definieren `device_data` später, wenn Sie die DynamoDB-Regelaktion konfigurieren.

Schritt 2: Erstellen Sie eine AWS IoT Regel zum Senden von Daten an die DynamoDB-Tabelle

In diesem Schritt verwenden Sie die Regelabfrageanweisung, um die Daten der imaginären Wettersensorgeräte so zu formatieren, dass sie in die Datenbanktabelle geschrieben werden.

Ein Beispiel für eine Nachrichtennutzlast, die von einem Wettersensorgerät empfangen wurde, sieht wie folgt aus:

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

Für den Datenbankeintrag verwenden Sie die Regelabfrageanweisung, um die Struktur der Nutzlast der Nachricht so zu reduzieren, dass sie wie folgt aussieht:

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind_velocity": 22,
  "wind_bearing": 255
}
```

In dieser Regel verwenden Sie auch ein paar Ersetzungsvorlagen. Ersatzvorlagen sind Ausdrücke, mit denen Sie dynamische Werte aus Funktionen und Nachrichtendaten einfügen können.

Um das zu erstellen AWS IoT Regel zum Senden von Daten an die DynamoDB-Tabelle

1. Öffnen Sie den Regel-Hub der AWS IoT Konsole. Sie können auch die AWS IoT Startseite innerhalb von öffnen AWS-Managementkonsole und zu Message Routing>Rules navigieren.
2. Um mit der Erstellung Ihrer neuen Regel unter Regeln zu beginnen, wählen Sie Regel erstellen aus.
3. In den Regeleigenschaften:

   a. Geben Sie in Role Name (Rollenname) **`wx_data_ddb`** ein.

      Denken Sie daran, dass ein Regelname innerhalb Ihrer Region AWS-Konto und Ihrer Region eindeutig sein muss und keine Leerzeichen enthalten darf. Wir haben in diesem Namen einen Unterstrich verwendet, um die beiden Wörter im Namen der Regel voneinander zu trennen.

   b. Beschreiben Sie die Regel in der Regelbeschreibung.

      Eine aussagekräftige Beschreibung macht es einfacher, sich daran zu erinnern, was diese Regel bewirkt und warum Sie sie erstellt haben. Die Beschreibung kann so lang wie nötig sein, also seien Sie so detailliert wie möglich.
4. Wählen Sie Next (Weiter), um fortzufahren.
5. In SQL-Anweisung:

   a. Wählen Sie in der SQL-Version **`2016-03-23`** aus.

   b. Geben Sie im Bearbeitungsfeld für die SQL-Anweisung die folgende Anweisung ein:

```
SELECT temperature, humidity, barometer,
  wind.velocity as wind_velocity,
  wind.bearing as wind_bearing,
FROM 'device/+/data'
```

      Diese Aussage:

      - Hört auf MQTT-Nachrichten mit einem Thema, das dem `device/+/data` Themenfilter entspricht.
      - Formatiert die Elemente des `wind` Attributs als einzelne Attribute.

- Übergibt die `temperature`, `humidity`, und `barometer` Attribute unverändert.

6. Wählen Sie Next (Weiter), um fortzufahren.
7. In Regelaktionen:

   a. Um die Liste der Regelaktionen für diese Regel zu öffnen, wählen Sie in Aktion 1 die Option **`DynamoDB`**.

   > **Note**
   >
   > Stellen Sie sicher, dass Sie DynamoDB und nicht DynamoDBv2 als Regelaktion ausgewählt haben.

   b. Wählen Sie unter Tabellenname den Namen der DynamoDB-Tabelle aus, die Sie in einem vorherigen Schritt erstellt haben: **`wx_data`**

   Die Felder Partitionsschlüsseltyp und Sortierschlüsseltyp werden mit den Werten aus Ihrer DynamoDB-Tabelle gefüllt.

   c. Geben Sie im Feld Partitionsschlüssel den Wert **`sample_time`** ein.

   d. Geben Sie unter Partition key value (Partitions-Schlüsselwert) **`${timestamp()}`** ein.

   Dies ist die erste der Ersetzungsvorlagen, die Sie in dieser Regel verwenden werden. Anstatt einen Wert aus der Nachrichtennutzlast zu verwenden, wird der von der Zeitstempelfunktion zurückgegebene Wert verwendet. Weitere Informationen finden Sie unter Zeitstempel im AWS IoT Core Entwicklerhandbuch.

   e. Geben Sie unter Sort key (Sortierschlüssel) **`device_id`** ein.

   f. Geben Sie unter Sort key value (Schlüsselwert sortieren) **`${cast(topic(2) AS DECIMAL)}`** ein.

   Dies ist der zweite der Ersetzungsvorlagen den Sie in dieser Regel verwenden werden. Es fügt den Wert des zweiten Elements in den Themennamen ein, bei dem es sich um die ID des Geräts handelt, nachdem es ihn in einen DEZIMALWERT umgewandelt hat, der dem numerischen Format des Schlüssels entspricht. Weitere Informationen zu Themen finden Sie unter Thema im AWS IoT Core Entwicklerhandbuch. Weitere Informationen zum Casting finden Sie unter Besetzung im AWS IoT Core Entwicklerhandbuch.

   g. Geben Sie in Write message data to this column (Nachrichtendaten in diese Spalte schreiben) **`device_data`** ein.

Dadurch wird die `device_data` Spalte in der DynamoDB-Tabelle erstellt.

h. Lassen Sie Operation leer.

i. Wählen Sie unter IAM Role (IAM-Rolle) die Option Create a New Role (Neue Rolle erstellen) aus.

j. Geben Sie im Dialogfeld Rolle erstellen als Rollenname wx_ddb_role ein. Diese neue Rolle enthält automatisch eine Richtlinie mit dem Präfix „aws-iot-rule“, die es der **`wx_data_ddb`** Regel ermöglicht, Daten an die von Ihnen erstellte **`wx_data`** DynamoDB-Tabelle zu senden.

k. Wählen Sie unter IAM Role (IAM-Rolle) **`wx_ddb_role`** aus.

l. Wählen Sie unten auf der Seite Next (Weiter) aus.

8. Wählen Sie unten auf der Seite Überprüfen und erstellen die Option Erstellen aus, um die Regel zu erstellen.

Schritt 3: Testen Sie AWS IoT Regel und DynamoDB-Tabelle

Um die neue Regel zu testen, verwenden Sie den MQTT-Client, um die in diesem Test verwendeten MQTT-Nachrichten zu veröffentlichen und zu abonnieren.

Öffnen Sie den MQTT-Client in der AWS IoT Konsole in einem neuen Fenster. Auf diese Weise können Sie die Regel bearbeiten, ohne die Konfiguration Ihres MQTT-Clients zu verlieren. Der MQTT-Client speichert keine Abonnements oder Nachrichtenprotokolle, wenn Sie ihn verlassen, um zu einer anderen Seite in der Konsole zu wechseln. Außerdem sollten Sie in der Konsole ein separates Konsolenfenster für den AWS IoT DynamoDB-Tabellen-Hub öffnen, um die neuen Einträge anzuzeigen, die Ihre Regel sendet.

Um den MQTT-Client zum Testen Ihrer Regel verwenden.

1. Abonnieren Sie im MQTT-Client in der AWS IoT Konsole das Eingabethema, `device/+/data`.

   a. Wählen Sie im MQTT-Client die Option Publish to a topic (In einem Thema veröffentlichen) aus.

   b. Geben Sie für Themenfilter das Thema des Eingabethemenfilters **`device/+/data`** ein.

   c. Wählen Sie Subscribe (Abonnieren) aus.

2. Veröffentlichen Sie jetzt eine Nachricht zum Eingabethema mit einer bestimmten Geräte-ID, **`device/22/data`**. Sie können keine Beiträge in MQTT-Themen veröffentlichen, die Platzhalterzeichen enthalten.

a. Wählen Sie im MQTT-Client die Option Publish to a topic (In einem Thema veröffentlichen) aus.

b. Geben Sie bei Themenname den Namen des eingegebenen Themas ein, **`device/22/data`**.

c. Geben Sie für Nachrichtennutzlast die folgenden Beispieldaten ein.

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

d. Um die MQTT-Nachricht zu veröffentlichen, wählen Sie Veröffentlichen.

e. Wählen Sie nun im MQTT-Client Ein Thema abonnieren aus. Wählen Sie in der Spalte Abonnieren das **`device/+/data`** Abonnement aus. Vergewissern Sie sich, dass die Beispieldaten aus dem vorherigen Schritt dort angezeigt werden.

3. Prüfen Sie, ob die Zeile in der DynamoDB-Tabelle angezeigt wird, die Ihre Regel erstellt hat.

a. Wählen Sie im DynamoDB-Tabellen-Hub in der AWS IoT Konsole wx_data und dann die Registerkarte Elemente aus.

Wenn Sie sich bereits auf der Registerkarte Elemente befinden, müssen Sie möglicherweise die Anzeige aktualisieren, indem Sie das Aktualisierungssymbol in der oberen rechten Ecke der Kopfzeile der Tabelle auswählen.

b. Beachten Sie, dass es sich bei den sample_time-Werten in der Tabelle um Links handelt, und öffnen Sie einen. Wenn Sie gerade Ihre erste Nachricht gesendet haben, ist dies die einzige in der Liste.

Dieser Link zeigt alle Daten in dieser Zeile der Tabelle an.

c. Erweitern Sie den Eintrag device_data, um die Daten anzuzeigen, die sich aus der Regelabfrageanweisung ergeben haben.

d. Erkunden Sie die verschiedenen Darstellungen der Daten, die in dieser Anzeige verfügbar sind. Sie können die Daten in dieser Anzeige auch bearbeiten.

e. Wenn Sie die Überprüfung dieser Datenzeile abgeschlossen haben, klicken Sie auf Speichern, um alle von Ihnen vorgenommenen Änderungen zu speichern, oder klicken Sie auf Abbrechen, um den Vorgang zu beenden, ohne die Änderungen zu speichern.

Wenn Sie nicht das richtige Verhalten feststellen, lesen Sie die Tipps zur Fehlerbehebung.

Problembehandlung bei Ihrer DynamoDB-Regel

Hier sind einige Dinge, die Sie überprüfen sollten, falls Sie nicht die erwarteten Ergebnisse sehen.

- Sie haben ein Fehlerbanner

  Wenn bei der Veröffentlichung der Eingabemeldung ein Fehler aufgetreten ist, korrigieren Sie diesen Fehler zuerst. Die folgenden Schritte können Ihnen helfen, diesen Fehler zu korrigieren.
- Sie sehen die Eingabenachricht nicht im MQTT-Client

  Jedes Mal, wenn Sie Ihre Eingabenachricht zum `device/22/data` Thema veröffentlichen, sollte diese Nachricht im MQTT-Client erscheinen, wenn Sie den `device/+/data` Themenfilter wie im Verfahren beschrieben abonniert haben.

  Zu überprüfende Dinge

  - Überprüfen Sie den Themenfilter, den Sie abonniert haben

    Wenn Sie das Thema der Eingabenachricht wie im Verfahren beschrieben abonniert haben, sollte Ihnen bei jeder Veröffentlichung eine Kopie der Eingabenachricht angezeigt werden.

    Wenn Sie die Nachricht nicht sehen, überprüfen Sie den Themennamen, den Sie abonniert haben, und vergleichen Sie ihn mit dem Thema, zu dem Sie sie veröffentlicht haben. Bei Themennamen wird Groß- und Kleinschreibung beachtet, und das Thema, das Sie abonniert haben, muss mit dem Thema identisch sein, zu dem Sie die Nachrichtennutzlast veröffentlicht haben.
  - Überprüfen Sie die Funktion zum Veröffentlichen von Nachrichten

    Wählen Sie im MQTT-Client unter Abonnements die Option Gerät/+/Daten aus, überprüfen Sie das Thema der Veröffentlichungsnachricht und wählen Sie dann Im Thema veröffentlichen aus. Sie sollten sehen, dass die Nutzlast der Nachricht aus dem Bearbeitungsfeld unter dem Thema in der Nachrichtenliste erscheinen.
- Sie finden Ihre Daten nicht in der DynamoDB-Tabelle

Als Erstes müssen Sie die Anzeige aktualisieren, indem Sie oben in der Kopfzeile der Tabelle das Aktualisierungssymbol auswählen. Wenn die Daten, nach denen Sie suchen, nicht angezeigt werden, überprüfen Sie Folgendes.

Zu überprüfende Dinge

- Prüfen Sie das AWS-Region Ihres MQTT-Clients und der Regel, die Sie erstellt haben

  Die Konsole, in der Sie den MQTT-Client ausführen, muss sich in derselben AWS Region befinden wie die von Ihnen erstellte Regel.
- Überprüfen Sie das Thema der Eingabemeldung in der Regelabfrageanweisung

  Damit die Regel funktioniert, muss sie eine Nachricht mit dem Themennamen erhalten, der dem Themenfilter in der FROM-Klausel der Regelabfrageanweisung entspricht.

  Überprüfen Sie die Schreibweise des Themenfilters in der Regelabfrageanweisung mit der des Themas im MQTT-Client. Bei Themennamen wird Groß- und Kleinschreibung beachtet, und das Thema der Nachricht muss mit dem Themenfilter in der Regelabfrageanweisung übereinstimmen.
- Überprüfen Sie den Inhalt der Nutzlast der Input-Nachricht

  Damit die Regel funktioniert, muss sie das Datenfeld in der Nachrichtennutzlast finden, das in der SELECT-Anweisung deklariert ist.

  Überprüfen Sie die Schreibweise des `temperature` Felds in der Regelabfrageanweisung mit der Schreibweise der Nachrichtennutzlast im MQTT-Client. Bei Feldnamen wird zwischen Groß- und Kleinschreibung unterschieden, und das `temperature` Feld in der Regelabfrageanweisung muss mit dem `temperature` Feld in der Nachrichtennutzlast identisch sein.

  Stellen Sie sicher, dass das JSON-Dokument in der Nachrichtennutzlast korrekt formatiert ist. Wenn das JSON Fehler enthält, z. B. ein fehlendes Komma, kann die Regel es nicht lesen.
- Überprüfen Sie die Schlüssel- und Feldnamen, die in der Regelaktion verwendet wurden

  Die in der Themenregel verwendeten Feldnamen müssen mit denen in der JSON-Nachrichtennutzlast der veröffentlichten Nachricht übereinstimmen.

  Öffnen Sie die Regel, die Sie in der Konsole erstellt haben, und überprüfen Sie die Feldnamen in der Regelaktionskonfiguration mit denen, die im MQTT-Client verwendet werden.
- Überprüfen Sie die Rolle, die von der Regel verwendet wird

Die Regelaktion muss berechtigt sein, das ursprüngliche Thema zu empfangen und das neue Thema zu veröffentlichen.

Die Richtlinien, die die Regel autorisieren, Nachrichtendaten zu empfangen und die DynamoDB-Tabelle zu aktualisieren, sind spezifisch für die verwendeten Themen. Wenn Sie das von der Regel verwendete Thema oder den Namen der DynamoDB-Tabelle ändern, müssen Sie die Rolle der Regelaktion aktualisieren, damit ihre Richtlinie entsprechend aktualisiert wird.

Wenn Sie vermuten, dass dies das Problem ist, bearbeiten Sie die Regelaktion und erstellen Sie eine neue Rolle. Neue Rollen, die durch die Regelaktion erstellt wurden, erhalten die erforderlichen Autorisierungen, um diese Aktionen auszuführen.

Schritt 4: Überprüfen Sie die Ergebnisse und die nächsten Schritte

Nachdem Sie mit dieser Regel einige Nachrichten an die DynamoDB-Tabelle gesendet haben, versuchen Sie, damit zu experimentieren, um zu sehen, wie sich Änderungen einiger Aspekte aus dem Tutorial auf die in die Tabelle geschriebenen Daten auswirken. Hier sind einige Ideen, die Ihnen den Einstieg erleichtern sollen.

- Ändern Sie das Thema `device_id` der Eingabenachricht und beobachten Sie die Auswirkung auf die Daten. Sie könnten dies verwenden, um den Empfang von Daten von mehreren Wettersensoren zu simulieren.
- Ändern Sie die in der Regelabfrageanweisung ausgewählten Felder und beobachten Sie die Auswirkungen auf die Daten. Sie könnten dies verwenden, um die in der Tabelle gespeicherten Daten zu filtern.
- Fügen Sie eine Regelaktion zum erneuten Veröffentlichen hinzu, um für jede Zeile, die der Tabelle hinzugefügt wird, eine MQTT-Nachricht zu senden. Sie könnten dies zum Debuggen verwenden.

Nachdem Sie dieses Tutorial abgeschlossen haben, besuchen Sie the section called “Formatieren einer Benachrichtigung mithilfe einer AWS Lambda Funktion”.

Tutorial: Formatieren einer Benachrichtigung mithilfe einer AWS Lambda function

Dieses Tutorial zeigt, wie Sie MQTT-Nachrichtendaten an eine AWS Lambda Aktion senden, um sie zu formatieren und an einen anderen AWS Dienst zu senden. In diesem Tutorial verwendet die AWS Lambda Aktion das AWS SDK, um die formatierte Nachricht an das Amazon SNS-Thema zu senden,

das Sie im Tutorial zur Vorgehensweise erstellt haben. the section called “Senden einer Amazon SNS-Benachrichtigung”

Im Tutorial zum Thema wie man the section called “Senden einer Amazon SNS-Benachrichtigung” wurde das JSON-Dokument, das sich aus der Abfrageanweisung der Regel ergab, als Hauptteil der Textnachricht gesendet. Das Ergebnis war eine Textnachricht, die in etwa so aussah wie in diesem Beispiel:

```
{"device_id":"32","reported_temperature":38,"max_temperature":30}
```

In diesem Tutorial verwenden Sie eine AWS Lambda Regelaktion, um eine AWS Lambda Funktion aufzurufen, die die Daten aus der Regelabfrageanweisung in ein benutzerfreundlicheres Format formatiert, wie in diesem Beispiel:

```
Device 32 reports a temperature of 38, which exceeds the limit of 30.
```

Die AWS Lambda Funktion, die Sie in diesem Tutorial erstellen, formatiert die Nachrichtenzeichenfolge mithilfe der Daten aus der Regelabfrageanweisung und ruft die https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sns.html#SNS.Client.publish SNS-Veröffentlichungsfunktion des AWS SDK auf, um die Benachrichtigung zu erstellen.

Was Sie in diesem Tutorial lernen werden

- Wie erstelle und teste ich eine Funktion AWS Lambda
- Wie verwende ich das AWS SDK in einer AWS Lambda Funktion, um eine Amazon SNS-Benachrichtigung zu veröffentlichen
- Wie man einfache SQL-Abfragen und Funktionen in einer Regelabfrageanweisung verwendet
- Wie verwende ich den MQTT-Client, um eine Regel zu testen AWS IoT

Für dieses Tutorial brauchen Sie ungefähr 45 Minuten.

In diesem Tutorial führen Sie die folgenden Aktivitäten durch:

- Schritt 1: Erstellen Sie eine AWS Lambda Funktion, die eine Textnachricht sendet
- Schritt 2: Erstellen Sie eine AWS IoT Regel mit einem AWS Lambda -Regelaktion
- Schritt 3: Testen Sie den AWS IoT Regel und AWS Lambda -Regelaktion
- Schritt 4: Überprüfen Sie die Ergebnisse und die nächsten Schritte

Stellen Sie vor Beginn dieses Tutorials sicher, dass Sie über Folgendes verfügen:

- Einrichten AWS-Konto

  Sie benötigen Ihre eigene AWS-Konto AWS IoT Konsole, um dieses Tutorial abzuschließen.

- Überprüft MQTT-Nachrichten anzeigen mit dem AWS IoT MQTT-Client

  Stellen Sie sicher, dass Sie den MQTT-Client verwenden können, um ein Thema zu abonnieren und zu veröffentlichen. In diesem Verfahren werden Sie den MQTT-Client verwenden, um Ihre neue Regel zu testen.

- Sie haben die anderen Regel-Tutorials in diesem Abschnitt abgeschlossen

  Für dieses Tutorial ist das Thema SNS-Benachrichtigung erforderlich, das Sie im Tutorial zur the section called “Senden einer Amazon SNS-Benachrichtigung” Vorgehensweise erstellt haben. Außerdem wird davon ausgegangen, dass Sie die anderen regelbezogenen Tutorials in diesem Abschnitt abgeschlossen haben.

- Überprüfte das AWS Lambda -Übersicht

  Wenn Sie Lambda noch nicht verwendet AWS Lambda haben, lesen AWS Lambda Sie das Kapitel Erste Schritte mit Lambda, um sich mit den Begriffen und Konzepten vertraut zu machen.

Schritt 1: Erstellen Sie eine AWS Lambda Funktion, die eine Textnachricht sendet

Die AWS Lambda Funktion in diesem Tutorial empfängt das Ergebnis der Regelabfrageanweisung, fügt die Elemente in eine Textzeichenfolge ein und sendet die resultierende Zeichenfolge als Nachricht in einer Benachrichtigung an Amazon SNS.

Im Gegensatz zum Tutorial zur Vorgehensweisethe section called “Senden einer Amazon SNS-Benachrichtigung”, bei dem eine AWS IoT Regelaktion zum Senden der Benachrichtigung verwendet wurde, sendet dieses Tutorial die Benachrichtigung von der Lambda-Funktion aus mithilfe einer Funktion des AWS SDK. Das eigentliche Amazon SNS-Benachrichtigungsthema, das in diesem Tutorial verwendet wird, ist jedoch dasselbe, das Sie im Tutorial zum Thema wie man the section called “Senden einer Amazon SNS-Benachrichtigung” verwendet haben.

Um eine zu erstellen AWS Lambda Funktion, die eine Textnachricht sendet

1. Erstellen Sie eine neue AWS Lambda Funktion.

a. Wählen Sie in der AWS Lambda Konsole die Option Create function (Funktion erstellen) aus.

b. Wählen Sie unter Funktion erstellen die Option Blueprint verwenden aus.

   Suchen Sie nach dem **`hello-world-python`** Blueprint, wählen Sie ihn aus und wählen Sie dann Konfigurieren aus.

c. Unter Grundlegende Informationen:

   i. Geben Sie unter Funktionsname den Namen dieser Funktion ein, **`format-high-temp-notification`**.

   ii. Wählen Sie unter Ausführungsrolle die Option Neue Rolle aus AWS Richtlinienvorlagen erstellen aus.

   iii. Geben Sie im Feld Rollenname den Namen der neuen Rolle ein, **`format-high-temp-notification-role`**.

   iv. Suchen Sie unter Richtlinienvorlagen — optional nach der Amazon SNS-Veröffentlichungsrichtlinie und wählen Sie diese aus.

   v. Wählen Sie Funktion erstellen.

2. Ändern Sie den Blueprint-Code, um eine Amazon SNS-Benachrichtigung zu formatieren und zu senden.

   a. Nachdem Sie Ihre Funktion erstellt haben, sollten Sie die Detailseite format-high-temp-notification sehen. Wenn Sie dies nicht tun, öffnen Sie es auf der Seite Lambda Funktionen.

   b. Wählen Sie auf der Detailseite von format-high-temp-notification die Registerkarte Konfiguration und blättern Sie zum Feld Funktionscode.

   c. Wählen Sie im Fenster Funktionscode im Bereich Umgebung die Python-Datei `lambda_function.py` aus.

   d. Löschen Sie im Fenster Funktionscode den gesamten ursprünglichen Programmcode aus dem Blueprint und ersetzen Sie ihn durch diesen Code.

```
import boto3
#
#   expects event parameter to contain:
#   {
#       "device_id": "32",
#       "reported_temperature": 38,
#       "max_temperature": 30,
```

```
#       "notify_topic_arn": "arn:aws:sns:us-
east-1:57EXAMPLE833:high_temp_notice"
#   }
#
#   sends a plain text string to be used in a text message
#
#      "Device {0} reports a temperature of {1}, which exceeds the limit of
 {2}."
#
#   where:
#       {0} is the device_id value
#       {1} is the reported_temperature value
#       {2} is the max_temperature value
#
def lambda_handler(event, context):

    # Create an SNS client to send notification
    sns = boto3.client('sns')

    # Format text message from data
    message_text = "Device {0} reports a temperature of {1}, which exceeds the
 limit of {2}.".format(
            str(event['device_id']),
            str(event['reported_temperature']),
            str(event['max_temperature'])
        )

    # Publish the formatted message
    response = sns.publish(
            TopicArn = event['notify_topic_arn'],
            Message = message_text
        )

    return response
```

e. Wählen Sie Bereitstellen.

3. Suchen Sie in einem neuen Fenster den Amazon-Ressourcennamen (ARN) Ihres Amazon-SNS-Themas aus dem Tutorial zum Thema wie man the section called “Senden einer Amazon SNS-Benachrichtigung”.

   a. Öffnen Sie in einem neuen Fenster die Themenseite der Amazon SNS-Konsole.

b. Suchen Sie auf der Seite Themen das Benachrichtigungsthema high_temp_notice in der Liste der Amazon SNS-Themen.

c. Suchen Sie den ARN des Benachrichtigungsthemas high_temp_notice, das Sie im nächsten Schritt verwenden möchten.

4. Erstellen Sie einen Testfall für Ihre Lambda-Funktion.

a. Wählen Sie auf der Seite Lambda Funktionen der Konsole auf der Detailseite format-high-temp-notification die Option Testereignis auswählen in der oberen rechten Ecke der Seite aus (auch wenn es deaktiviert aussieht) und wählen Sie dann Testereignisse konfigurieren aus.

b. Wählen Sie unter Testereignis konfigurieren die Option Neues Testereignis erstellen.

c. Geben Sie unter Event name (Ereignisname) **`SampleRuleOutput`** ein.

d. Fügen Sie im JSON-Editor unter dem Ereignisnamen dieses JSON-Beispieldokument ein. Dies ist ein Beispiel dafür, was Ihre AWS IoT Regel an die Lambda-Funktion sendet.

```
{
  "device_id": "32",
  "reported_temperature": 38,
  "max_temperature": 30,
  "notify_topic_arn": "arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice"
}
```

e. Sehen Sie sich das Fenster an, das den ARN des Benachrichtigungsthemas high_temp_notice enthält, und kopieren Sie den ARN-Wert.

f. Ersetzen Sie den `notify_topic_arn` Wert im JSON-Editor durch den ARN aus Ihrem Benachrichtigungsthema.

Lassen Sie dieses Fenster geöffnet, damit Sie diesen ARN-Wert erneut verwenden können, wenn Sie die AWS IoT Regel erstellen.

g. Wählen Sie Erstellen aus.

5. Testen Sie die Funktion mit Beispieldaten.

a. Bestätigen Sie auf der Seite mit den Details zur Format-High-Temp-Benachrichtigung in der oberen rechten Ecke der Seite, dass neben der Schaltfläche Testen angezeigt wird. SampleRuleOutput Wenn nicht, wählen Sie es aus der Liste der verfügbaren Testereignisse aus.

b. Um die Ausgabenachricht der Beispielregel an Ihre Funktion zu senden, wählen Sie Test.

Wenn sowohl die Funktion als auch die Benachrichtigung funktioniert haben, erhalten Sie eine Textnachricht auf dem Telefon, das die Benachrichtigung abonniert hat.

Wenn Sie am Telefon keine Textnachricht erhalten haben, überprüfen Sie das Ergebnis des Vorgangs. Überprüfen Sie im Bereich Funktionscode auf der Registerkarte Ausführungsergebnisse die Antwort auf aufgetretene Fehler. Fahren Sie erst mit dem nächsten Schritt fort, wenn Ihre Funktion die Benachrichtigung an Ihr Telefon senden kann.

Schritt 2: Erstellen Sie eine AWS IoT Regel mit einem AWS Lambda -Regelaktion

In diesem Schritt verwenden Sie die Regelabfrageanweisung, um die Daten vom imaginären Wettersensorgerät zu formatieren, um sie an eine Lambda-Funktion zu senden, die eine Textnachricht formatiert und sendet.

Ein Beispiel für eine von den Wettergeräten empfangene Nachrichtennutzlast sieht wie folgt aus:

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

In dieser Regel verwenden Sie die Regelabfrageanweisung, um eine Nachrichtennutzlast für die Lambda-Funktion zu erstellen, die wie folgt aussieht:

```
{
  "device_id": "32",
  "reported_temperature": 38,
  "max_temperature": 30,
  "notify_topic_arn": "arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice"
}
```

Dies enthält alle Informationen, die die Lambda-Funktion benötigt, um die richtige Textnachricht zu formatieren und zu senden.

Um das zu erstellen AWS IoT Regel zum Aufrufen einer Lambda-Funktion

1. Öffnen Sie den Regel-Hub der AWS IoT Konsole.

2. Um mit der Erstellung Ihrer neuen Regel unter Regeln zu beginnen, wählen Sie Erstellen.
3. Gehen Sie im oberen Teil von Regel erstellen wie folgt vor:

   a. Geben Sie im Feld Name den Namen der Regel ein, **`wx_friendly_text`**.

      Denken Sie daran, dass ein Regelname innerhalb Ihrer Region AWS-Konto und Ihrer Region eindeutig sein muss und keine Leerzeichen enthalten darf. Wir haben in diesem Namen einen Unterstrich verwendet, um die beiden Wörter im Namen der Regel voneinander zu trennen.

   b. Beschreiben Sie die Regel im Feld Beschreibung.

      Eine aussagekräftige Beschreibung macht es einfacher, sich daran zu erinnern, was diese Regel bewirkt und warum Sie sie erstellt haben. Die Beschreibung kann so lang wie nötig sein, also seien Sie so detailliert wie möglich.

4. In der Regelabfrageanweisung von Regel erstellen:

   a. Wählen Sie unter SQL-Version verwenden die Option **`2016-03-23`** aus.

   b. Geben Sie im Bearbeitungsfeld Regelabfrageanweisung die folgende Anweisung ein:

```
SELECT
  cast(topic(2) AS DECIMAL) as device_id,
  temperature as reported_temperature,
  30 as max_temperature,
  'arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice' as notify_topic_arn
FROM 'device/+/data' WHERE temperature > 30
```

      Diese Aussage:

      - Lauscht auf MQTT-Nachrichten mit einem Thema, das dem `device/+/data` Themenfilter entspricht und deren `temperature` Wert größer als 30 ist.
      - Wählt das zweite Element aus der Themenzeichenfolge aus, konvertiert es in eine Dezimalzahl und weist es dann dem `device_id` Feld zu.
      - Wählt den Wert des `temperature` Felds aus der Nachrichtennutzlast aus und weist ihn dem `reported_temperature` Feld zu.
      - Erstellt einen konstanten Wert, `30`, um den Grenzwert darzustellen, und weist ihn dem `max_temperature` Feld zu.
      - Erstellt einen konstanten Wert für das `notify_topic_arn` Feld.

c. Sehen Sie sich das Fenster an, das den ARN des Benachrichtigungsthemas high_temp_notice enthält, und kopieren Sie den ARN-Wert.

d. Ersetzen Sie den ARN-Wert (*arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice*) im Editor für Regelabfragen durch den ARN Ihres Benachrichtigungsthemas.

5. Gehen Sie im Feld Eine oder mehrere Aktionen festlegen wie folgt vor:

a. Um die Liste der Regelaktionen für diese Regel zu öffnen, wählen Sie Aktion hinzufügen.

b. Wählen Sie unter Aktion auswählen die Option Nachricht an eine Lambda-Funktion senden aus.

c. Um die Konfigurationsseite der ausgewählten Aktion zu öffnen, wählen Sie unten in der Aktionsliste die Option Aktion konfigurieren aus.

6. Gehen Sie in Aktion konfigurieren wie folgt vor:

a. Wählen Sie unter Funktionsname die Option Auswählen aus.

b. Wählen Sie format-high-temp-notification.

c. Wählen Sie unten im Bereich Aktion konfigurieren die Option Aktion hinzufügen aus.

d. Um die Regel zu erstellen, wählen Sie unten im Bereich Eine Regel erstellen die Option Regel erstellen aus.

Schritt 3: Testen Sie den AWS IoT Regel und AWS Lambda -Regelaktion

Um Ihre neue Regel zu testen, verwenden Sie den MQTT-Client, um die von dieser Regel verwendeten MQTT-Nachrichten zu veröffentlichen und zu abonnieren.

Öffnen Sie den MQTT-Client in der AWS IoT Konsole in einem neuen Fenster. Jetzt können Sie die Regel bearbeiten, ohne die Konfiguration Ihres MQTT-Clients zu verlieren. Wenn Sie den MQTT-Client verlassen, um zu einer anderen Seite in der Konsole zu wechseln, verlieren Sie Ihre Abonnements oder Nachrichtenprotokolle.

Um den MQTT-Client zum Testen Ihrer Regel verwenden.

1. Abonnieren Sie im MQTT-Client in der AWS IoT Konsole die Eingabethemen, in diesem Fall `device/+/data`.

a. Wählen Sie im MQTT-Client unter Abonnements die Option Thema abonnieren aus.

b. Geben Sie im Abonnementthema das Thema des Eingabethemenfilters **`device/+/data`** ein.

c. Belassen Sie die übrigen Felder auf ihren Standardeinstellungen.

d. Wählen Sie Thema abonnieren aus.

In der Spalte Abonnements wird **`device/+/data`** unter In einem Thema veröffentlichen angezeigt.

2. Veröffentlichen Sie eine Nachricht zum Eingabethema mit einer bestimmten Geräte-ID, **`device/32/data`**. Sie können keine Beiträge in MQTT-Themen veröffentlichen, die Platzhalterzeichen enthalten.

a. Wählen Sie im MQTT-Client unter Abonnements die Option In einem Thema veröffentlichen.

b. Geben Sie im Feld Veröffentlichen den Namen des Eingabethemas **`device/32/data`** ein.

c. Kopieren Sie die hier gezeigten Beispieldaten und fügen Sie die Beispieldaten in das Bearbeitungsfeld unter dem Themennamen ein.

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

d. Wählen Sie In Thema veröffentlichen aus, um Ihre MQTT-Nachricht in zu veröffentlichen.

3. Bestätigen Sie, dass die Textnachricht gesendet wurde.

a. Im MQTT-Client befindet sich unter Abonnements ein grüner Punkt neben dem Thema, das Sie zuvor abonniert haben.

Der grüne Punkt zeigt an, dass eine oder mehrere neue Nachrichten eingegangen sind, seit Sie sie das letzte Mal angesehen haben.

b. Wählen Sie unter Abonnements die Option Gerät/+/Daten aus, um zu überprüfen, ob die Nutzlast der Nachricht mit dem übereinstimmt, was Sie gerade veröffentlicht haben, und wie folgt aussieht:

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

c. Überprüfen Sie das Telefon, mit dem Sie das SNS-Thema abonniert haben, und vergewissern Sie sich, dass der Inhalt der Nutzlast der Nachricht so aussieht:

```
Device 32 reports a temperature of 38, which exceeds the limit of 30.
```

Wenn Sie das Thema-ID-Element im Nachrichtenthema ändern, denken Sie daran, dass die Umwandlung des `topic(2)` Werts in einen numerischen Wert nur funktioniert, wenn dieses Element im Nachrichtenthema nur numerische Zeichen enthält.

4. Versuchen Sie, eine MQTT-Nachricht zu senden, in der die Temperatur den Grenzwert nicht überschreitet.

a. Wählen Sie im MQTT-Client unter Abonnements die Option In einem Thema veröffentlichen.

b. Geben Sie im Feld Veröffentlichen den Namen des Eingabethemas **`device/33/data`** ein.

c. Kopieren Sie die hier gezeigten Beispieldaten und fügen Sie die Beispieldaten in das Bearbeitungsfeld unter dem Themennamen ein.

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

d. Um Ihre MQTT-Nachricht zu senden, wählen Sie Im Thema veröffentlichen.

Sie sollten die Nachricht sehen, die Sie im **`device/+/data`** Abonnement gesendet haben. Da der Temperaturwert jedoch unter der Höchsttemperatur in der Regelabfrageanweisung liegt, sollten Sie keine Textnachricht erhalten.

Wenn Sie nicht das richtige Verhalten feststellen, lesen Sie die Tipps zur Fehlerbehebung.

Fehlerbehebung bei Ihrem AWS Lambda Regel und Benachrichtigung

Hier sind einige Dinge, die Sie überprüfen sollten, falls Sie nicht die erwarteten Ergebnisse sehen.

- Sie haben ein Fehlerbanner

  Wenn bei der Veröffentlichung der Eingabemeldung ein Fehler aufgetreten ist, korrigieren Sie diesen Fehler zuerst. Die folgenden Schritte können Ihnen helfen, diesen Fehler zu korrigieren.
- Sie sehen die Eingabenachricht nicht im MQTT-Client

  Jedes Mal, wenn Sie Ihre Eingabenachricht zum `device/32/data` Thema veröffentlichen, sollte diese Nachricht im MQTT-Client erscheinen, wenn Sie den `device/+/data` Themenfilter wie im Verfahren beschrieben abonniert haben.

  Zu überprüfende Dinge

  - Überprüfen Sie den Themenfilter, den Sie abonniert haben

    Wenn Sie das Thema der Eingabenachricht wie im Verfahren beschrieben abonniert haben, sollte Ihnen bei jeder Veröffentlichung eine Kopie der Eingabenachricht angezeigt werden.

    Wenn Sie die Nachricht nicht sehen, überprüfen Sie den Themennamen, den Sie abonniert haben, und vergleichen Sie ihn mit dem Thema, zu dem Sie sie veröffentlicht haben. Bei Themennamen wird Groß- und Kleinschreibung beachtet, und das Thema, das Sie abonniert haben, muss mit dem Thema identisch sein, zu dem Sie die Nachrichtennutzlast veröffentlicht haben.
  - Überprüfen Sie die Funktion zum Veröffentlichen von Nachrichten

    Wählen Sie im MQTT-Client unter Abonnements die Option Gerät/+/Daten aus, überprüfen Sie das Thema der Veröffentlichungsnachricht und wählen Sie dann Im Thema veröffentlichen aus. Sie sollten sehen, dass die Nutzlast der Nachricht aus dem Bearbeitungsfeld unter dem Thema in der Nachrichtenliste erscheinen.

- Sie erhalten keine SMS-Nachricht

  Damit Ihre Regel funktioniert, muss sie über die richtige Richtlinie verfügen, die sie autorisiert, eine Nachricht zu empfangen und eine SNS-Benachrichtigung zu senden, und sie muss die Nachricht empfangen.

  Zu überprüfende Dinge

  - Überprüfe das AWS-Region Ihres MQTT-Clients und der Regel, die Sie erstellt haben

    Die Konsole, in der Sie den MQTT-Client ausführen, muss sich in derselben AWS Region befinden wie die von Ihnen erstellte Regel.
  - Überprüfen Sie, ob der Temperaturwert in der Nachrichtennutzlast den Testschwellenwert überschreitet

    Wenn der Temperaturwert kleiner oder gleich 30 ist, wie in der Regelabfrageanweisung definiert, führt die Regel keine ihrer Aktionen aus.
  - Überprüfen Sie das Thema der Eingabemeldung in der Regelabfrageanweisung

    Damit die Regel funktioniert, muss sie eine Nachricht mit dem Themennamen erhalten, der dem Themenfilter in der FROM-Klausel der Regelabfrageanweisung entspricht.

    Überprüfen Sie die Schreibweise des Themenfilters in der Regelabfrageanweisung mit der des Themas im MQTT-Client. Bei Themennamen wird Groß- und Kleinschreibung beachtet, und das Thema der Nachricht muss mit dem Themenfilter in der Regelabfrageanweisung übereinstimmen.
  - Überprüfen Sie den Inhalt der Nutzlast der Input-Nachricht

    Damit die Regel funktioniert, muss sie das Datenfeld in der Nachrichtennutzlast finden, das in der SELECT-Anweisung deklariert ist.

    Überprüfen Sie die Schreibweise des `temperature` Felds in der Regelabfrageanweisung mit der Schreibweise der Nachrichtennutzlast im MQTT-Client. Bei Feldnamen wird zwischen Groß- und Kleinschreibung unterschieden, und das `temperature` Feld in der Regelabfrageanweisung muss mit dem `temperature` Feld in der Nachrichtennutzlast identisch sein.

    Stellen Sie sicher, dass das JSON-Dokument in der Nachrichtennutzlast korrekt formatiert ist. Wenn das JSON Fehler enthält, z. B. ein fehlendes Komma, kann die Regel es nicht lesen.
  - Überprüfen Sie die Amazon-SNS-Benachrichtigung

Unter Schritt 1: Erstellen Sie ein Amazon SNS-Thema, das eine SMS-Textnachricht sendet, lesen Sie Schritt 3, in dem beschrieben wird, wie Sie die Amazon SNS-Benachrichtigung testen und die Benachrichtigung testen, um sicherzustellen, dass die Benachrichtigung funktioniert.

- Überprüfen Sie die Lambda-Funktion

  Unter Schritt 1: Erstellen Sie eine AWS Lambda Funktion, die eine Textnachricht sendet, lesen Sie Schritt 5, in dem beschrieben wird, wie Sie die Lambda-Funktion anhand von Testdaten testen und die Lambda-Funktion testen.

- Überprüfen Sie die Rolle, die von der Regel verwendet wird

  Die Regelaktion muss berechtigt sein, das ursprüngliche Thema zu empfangen und das neue Thema zu veröffentlichen.

  Die Richtlinien, mit denen die Regel autorisiert wird, Nachrichtendaten zu empfangen und erneut zu veröffentlichen, sind spezifisch für die verwendeten Themen. Wenn Sie das Thema ändern, das für die erneute Veröffentlichung der Nachrichtendaten verwendet wird, müssen Sie die Rolle der Regelaktion aktualisieren, damit ihre Richtlinie dem aktuellen Thema entspricht.

  Wenn Sie vermuten, dass dies das Problem ist, bearbeiten Sie die Aktion Regel erneut veröffentlichen und erstellen Sie eine neue Rolle. Neue Rollen, die durch die Regelaktion erstellt wurden, erhalten die erforderlichen Autorisierungen, um diese Aktionen auszuführen.

Schritt 4: Überprüfen Sie die Ergebnisse und die nächsten Schritte

In diesem Tutorial:

- Sie haben eine AWS IoT Regel zum Aufrufen einer Lambda-Funktion erstellt, die eine Amazon SNS-Benachrichtigung gesendet hat, die Ihre benutzerdefinierte Nachrichtennutzlast verwendet hat.
- Sie haben eine einfache SQL-Abfrage und Funktionen in einer Regelabfrageanweisung verwendet, um eine neue Nachrichtennutzlast für Ihre Lambda-Funktion zu erstellen.
- Sie haben den MQTT-Client verwendet, um Ihre Regel zu testen. AWS IoT

Nächste Schritte

Nachdem Sie einige Textnachrichten mit dieser Regel gesendet haben, versuchen Sie, damit zu experimentieren, um zu sehen, wie sich Änderungen einiger Aspekte des Tutorials auf die Nachricht

auswirken und wann sie gesendet wird. Hier sind einige Ideen, die Ihnen den Einstieg erleichtern sollen.

- Ändern Sie das Thema *`device_id`* der Eingabenachricht und beobachten Sie den Effekt im Inhalt der Textnachricht.
- Ändern Sie die in der Regelabfrageanweisung ausgewählten Felder, aktualisieren Sie die Lambda-Funktion, um sie in einer neuen Nachricht zu verwenden, und beobachten Sie die Auswirkungen im Inhalt der Textnachricht.
- Ändern Sie den Test in der Regelabfrageanweisung so, dass er auf eine Mindesttemperatur statt auf eine Höchsttemperatur testet. Aktualisieren Sie die Lambda-Funktion, um eine neue Nachricht zu formatieren, und denken Sie daran, den Namen von `max_temperature` zu ändern.
- Weitere Informationen zum Auffinden von Fehlern, die beim Entwickeln und Verwenden von AWS IoT Regeln auftreten können, finden Sie unterÜberwachen AWS IoT.

Beibehalten des Gerätestatus mit Geräteschatten, während das Gerät offline ist

Diese Tutorials zeigen Ihnen, wie Sie den AWS IoT Device Shadow-Dienst verwenden, um die Statusinformationen eines Geräts zu speichern und zu aktualisieren. Das Schattendokument, bei dem es sich um ein JSON-Dokument handelt, zeigt die Änderung des Gerätestatus auf der Grundlage der von einem Gerät, einer lokalen App oder einem Dienst veröffentlichten Nachrichten. In diesem Tutorial zeigt das Schattendokument die Änderung der Farbe einer Glühbirne. In diesen Tutorials wird auch gezeigt, wie der Schatten diese Informationen auch dann speichert, wenn das Gerät nicht mit dem Internet verbunden ist, und die neuesten Statusinformationen an das Gerät zurückgibt, wenn es wieder online ist und diese Informationen anfordert.

Wir empfehlen Ihnen, diese Tutorials in der Reihenfolge anzusehen, in der sie hier gezeigt werden. Beginnen Sie dabei mit den AWS IoT -Ressourcen, die Sie erstellen müssen, und der erforderlichen Hardwarekonfiguration, damit Sie die Konzepte schrittweise erlernen können. Diese Tutorials zeigen, wie Sie ein Raspberry Pi-Gerät zur Verwendung mit konfigurieren und anschließen AWS IoT. Wenn Sie nicht über die erforderliche Hardware verfügen, können Sie diesen Tutorials folgen, indem Sie sie an ein Gerät Ihrer Wahl anpassen oder ein virtuelles Gerät mit Amazon EC2 erstellen.

Übersicht über Tutorial-Szenario

Das Szenario für diese Tutorials ist eine lokale App oder ein Dienst, der die Farbe einer Glühbirne ändert und deren Daten in reservierten Schattenthemen veröffentlicht. Diese Tutorials ähneln der

Geräteschatten-Funktionalität, die im interaktiven Tutorial „Erste Schritte“ beschrieben wurde, und sind auf einem Raspberry Pi-Gerät implementiert. Die Tutorials in diesem Abschnitt konzentrieren sich auf einen einzelnen, klassischen Schatten und zeigen, wie Sie benannte Schatten oder mehrere Geräte unterbringen können.

In den folgenden Tutorials erfahren Sie, wie Sie den AWS IoT Device Shadow-Dienst verwenden.

- Tutorial: Ihren Raspberry Pi für die Ausführung der Schattenanwendung vorbereiten

  Dieses Tutorial zeigt, wie Sie ein Raspberry Pi-Gerät für die Verbindung mit einrichten AWS IoT. Sie erstellen auch ein AWS IoT Richtliniendokument und eine Ding-Ressource, laden die Zertifikate herunter und hängen dann die Richtlinie an diese Ding-Ressource an. Für dieses Tutorial brauchen Sie ungefähr 30 Minuten.
- Tutorial: Installation des Geräte-SDK und Ausführen der Beispielanwendung für Geräteschatten

  Dieses Tutorial zeigt, wie Sie die erforderlichen Tools, Software und das AWS IoT Geräte-SDK für Python installieren und dann die Beispiel-Shadow-Anwendung ausführen. Dieses Tutorial baut auf den unter Verbinden eines Raspberry Pi oder eines anderes Gerätes vorgestellten Konzepten auf und nimmt 20 Minuten in Anspruch.
- Tutorial: Interaktion mit Geräteschatten mithilfe der Beispiel-App und des MQTT-Testclients

  Dieses Tutorial zeigt, wie Sie die `shadow.py` Beispiel-App und die AWS IoT Konsole verwenden, um die Interaktion zwischen AWS IoT Device Shadows und den Statusänderungen der Glühbirne zu beobachten. Das Tutorial zeigt auch, wie Sie MQTT-Nachrichten an die reservierten Geräteschattenthemen senden. Für dieses Tutorial brauchen Sie ungefähr 45 Minuten.

AWS IoT Device Shadow im Überblick

Ein Device Shadow ist eine persistente, virtuelle Repräsentation eines Geräts, das von einer Ding-Ressource verwaltet wird, die Sie in der AWS IoT Registrierung erstellen. Das Shadow-Dokument ist ein JSON- oder JavaScript Notationsdokument, das zum Speichern und Abrufen der aktuellen Statusinformationen für ein Gerät verwendet wird. Sie können den Schatten nutzen, um über MQTT-Themen oder HTTP REST APIs den Status eines Geräts abzurufen oder festzulegen, unabhängig davon, ob das Gerät mit dem Internet verbunden ist oder nicht.

Das Schattendokument enthält eine `state`-Eigenschaft, die die folgenden Aspekte des Gerätezustands beschreibt.

- `desired`: Apps geben die gewünschten Status der Geräteeigenschaften an, indem sie das `desired`-Objekt aktualisieren.
- `reported`: Geräte melden ihren aktuellen Status im `reported`-Objekt.
- `delta`: AWS IoT meldet Unterschiede zwischen dem gewünschten und dem gemeldeten Status im `delta` Objekt.

Hier sehen Sie ein Beispiel für ein Schattenstatusdokument.

```
{
  "state": {
    "desired": {
      "color": "green"
      },
    "reported": {
      "color": "blue"
      },
    "delta": {
      "color": "green"
      }
   }
}
```

Um das Schattendokument eines Geräts zu aktualisieren, können Sie die reservierten MQTT-Themen, die Geräteschatten-REST-APIs, die die `GET`-, `UPDATE`- und `DELETE`-Operationen mit HTTP unterstützen, und die AWS IoT CLI verwenden.

Nehmen wir im vorherigen Beispiel an, Sie möchten die `desired` Farbe in `yellow` ändern. Senden Sie dazu eine Anfrage an die UpdateThingShadow API oder veröffentlichen Sie eine Nachricht zum Thema Aktualisieren`$aws/things/THING_NAME/shadow/update`.

```
{
  "state": {
    "desired": {
      "color": yellow
    }
  }
}
```

Aktualisierungen betreffen lediglich die in der Anfrage angegebenen Felder. Nach der erfolgreichen Aktualisierung von Device Shadow wird der neue `desired` Status unter dem `delta` Thema AWS

IoT veröffentlicht`$aws/things/THING_NAME/shadow/delta`. Das Schattendokument sieht in diesem Fall so aus:

```
{
  "state": {
    "desired": {
      "color": yellow
    },
    "reported": {
      "color": green
    },
    "delta": {
      "color": yellow
      }
  }
}
```

Der neue Status wird dann unter Verwendung des `Update` Themas `$aws/things/THING_NAME/shadow/update` mit der folgenden JSON-Meldung an den AWS IoT Device Shadow gemeldet:

```
{
  "state": {
    "reported": {
      "color": yellow
    }
  }
}
```

Wenn Sie die aktuellen Statusinformationen abrufen möchten, senden Sie eine Anfrage an die GetThingShadow API oder veröffentlichen Sie eine MQTT-Nachricht im Thema Get`$aws/things/THING_NAME/shadow/get`.

Weitere Informationen zur Verwendung des Geräteschattendienstes finden Sie in AWS IoT Geräteschatten-Service.

Weitere Informationen zur Verwendung von Geräteschatten in Geräten, Apps und Diensten finden Sie unter Verwenden von Schatten in Geräten und Verwenden von Schatten in Apps und Services.

Informationen zur Interaktion mit AWS IoT Schatten finden Sie unterInteraktion mit Schatten.

Informationen zu den reservierten MQTT-Themen und HTTP-REST-APIs finden Sie unter MQTT-Themen für Geräteschatten und Geräteschatten-REST-API.

Tutorial: Ihren Raspberry Pi für die Ausführung der Schattenanwendung vorbereiten

Dieses Tutorial zeigt, wie Sie ein Raspberry Pi-Gerät einrichten und konfigurieren und die AWS IoT Ressourcen erstellen, die ein Gerät für die Verbindung und den Austausch von MQTT-Nachrichten benötigt.

> **Note**
>
> Wenn Sie the section called “Erstellen Sie ein virtuelles Gerät mit Amazon EC2” planen, können Sie diese Seite überspringen und mit the section called “Konfigurieren Ihres Geräts” weitermachen. Sie werden diese Ressourcen erstellen, wenn Sie Ihr virtuelles Objekt erstellen. Wenn Sie anstelle des Raspberry Pi ein anderes Gerät verwenden möchten, können Sie versuchen, diese Anleitungen zu befolgen, indem Sie sie an ein Gerät Ihrer Wahl anpassen.

In diesem Kurs lernen Sie Folgendes:

- Richten Sie ein Raspberry Pi-Gerät ein und konfigurieren Sie es für die Verwendung mit AWS IoT.
- Erstellen Sie ein AWS IoT Richtliniendokument, das Ihr Gerät zur Interaktion mit AWS IoT Diensten autorisiert.
- Erstellen Sie eine Ding-Ressource in AWS IoT den X.509 Gerätezertifikaten und hängen Sie dann das Richtliniendokument an.

  Das Objekt ist die virtuelle Darstellung Ihres Geräts in der AWS IoT Registrierung. Das Zertifikat authentifiziert Ihr Gerät gegenüber AWS IoT Core, und das Richtliniendokument autorisiert Ihr Gerät zur Interaktion mit Core. AWS IoT

So führen Sie dieses Tutorial aus

Um die `shadow.py` Beispielanwendung für Geräteschatten auszuführen, benötigen Sie ein Raspberry Pi-Gerät, das eine Verbindung zu AWS IoT herstellt. Wir empfehlen, dass Sie dieses Tutorial in der Reihenfolge folgen, in der es hier vorgestellt wird. Beginnen Sie mit der Einrichtung des Raspberry Pi und seines Zubehörs, erstellen Sie dann eine Richtlinie und fügen Sie die Richtlinie einer von Ihnen erstellten Objekt-Ressource hinzu. Anschließend können Sie dieser Anleitung folgen, indem Sie die vom Raspberry Pi unterstützte grafische Benutzeroberfläche (GUI) verwenden, um die AWS IoT Konsole im Webbrowser des Geräts zu öffnen. Dadurch ist es auch einfacher, die Zertifikate direkt auf Ihren Raspberry Pi herunterzuladen, um eine Verbindung herzustellen. AWS IoT

Stellen Sie vor Beginn dieses Tutorials sicher, dass Sie über Folgendes verfügen:

- Und AWS-Konto. Wenn dies nicht der Fall ist, führen Sie die unter Einrichten AWS-Konto beschriebenen Schritte aus, bevor Sie fortfahren. Sie benötigen Ihre eigene AWS-Konto AWS IoT Konsole, um dieses Tutorial abzuschließen.
- Der Raspberry Pi und das notwendige Zubehör. Sie benötigen:
  - Ein Raspberry Pi 3 Modell B oder ein neueres Modell. Dieses Tutorial funktioniert möglicherweise auf früheren Versionen des Raspberry Pi, aber wir haben es nicht getestet.
  - Raspberry Pi OS (32-Bit) oder höher. Wir empfehlen die neueste Version des Raspberry Pi OS zu verwenden. Frühere Versionen des Betriebssystems funktionieren möglicherweise, aber wir haben es nicht getestet.
  - Ein Ethernet oder eine Wi-Fi Verbindung.
  - Tastatur, Maus, Monitor, Kabel und Netzteile.

Für dieses Tutorial brauchen Sie ungefähr 30 Minuten.

Schritt 1: Einrichten und Konfigurieren des Raspberry Pi-Geräts

In diesem Abschnitt konfigurieren wir ein Raspberry Pi-Gerät zur Verwendung mit AWS IoT.

> **Important**
>
> Die Anpassung dieser Anweisungen an andere Geräte und Betriebssysteme kann eine Herausforderung sein. Sie müssen Ihr Gerät gut genug verstehen, um diese Anweisungen interpretieren und auf Ihr Gerät anwenden zu können. Wenn Sie auf Schwierigkeiten stoßen, können Sie alternativ eine der anderen Geräteoptionen ausprobieren, z. B. Erstellen Sie ein virtuelles Gerät mit Amazon EC2 oderVerwenden Sie Ihren Windows- oder Linux-PC oder Mac als AWS IoT Gerät.

Sie müssen Ihren Raspberry Pi so konfigurieren, dass er das Betriebssystem (OS) starten, eine Verbindung zum Internet herstellen und Ihnen die Interaktion mit ihm über eine Befehlszeilenschnittstelle ermöglicht. Sie können auch die vom Raspberry Pi unterstützte grafische Benutzeroberfläche (GUI) verwenden, um die AWS IoT Konsole zu öffnen und den Rest dieses Tutorials auszuführen.

So richten Sie den Raspberry Pi ein

1. Setzen Sie die SD-Karte in den MicroSD-Kartensteckplatz im Raspberry Pi ein. Auf einigen SD-Karten ist ein Installationsmanager vorinstalliert, der Sie nach dem Booten der Karte mit einem Menü zur Installation des Betriebssystems auffordert. Sie können auch den Raspberry Pi-Imager verwenden, um das Betriebssystem auf Ihrer Karte zu installieren.
2. Schließen Sie ein HDMI TV oder einen HDMI-Monitor an das HDMI-Kabel an, das an den HDMI-Anschluss des Raspberry Pi angeschlossen ist.
3. Schließe Sie Tastatur und Maus an die USB-Anschlüsse des Raspberry Pi an und schließen Sie dann den Netzadapter an, um die Platine zu starten.

Wenn nach dem Start des Raspberry Pi der Installationsmanager auf der SD-Karte vorinstalliert war, erscheint ein Menü zur Installation des Betriebssystems. Wenn Sie Probleme bei der Installation des Betriebssystems haben, können Sie die folgenden Schritte ausführen. Weitere Informationen zum Einrichten von Raspberry Pi finden Sie unter Einrichten Ihres Raspberry Pi.

Wenn Sie Probleme beim Einrichten des Raspberry Pi haben:

- Überprüfen Sie, ob Sie die SD-Karte eingelegt haben, bevor Sie das Board starten. Wenn Sie die SD-Karte nach dem Booten des Boards einstecken, wird das Installationsmenü möglicherweise nicht angezeigt.
- Vergewissern Sie sich, dass das TV oder der Monitor eingeschaltet und der richtige Eingang ausgewählt ist.
- Stellen Sie sicher, dass Sie Raspberry Pi-kompatible Software verwenden.

Nachdem Sie das Raspberry Pi OS installiert und konfiguriert haben, öffnen Sie den Webbrowser des Raspberry Pi und navigieren Sie zur AWS IoT Core Konsole, um mit den restlichen Schritten in diesem Tutorial fortzufahren.

Wenn Sie die AWS IoT Core Konsole öffnen können, ist Ihr Raspberry Pi bereit und Sie können fortfahrenthe section called “Bereitstellen Ihres Geräts in AWS IoT ”.

Wenn Sie Probleme haben oder zusätzliche Hilfe benötigen, finden Sie weitere Informationen unter Hilfe für Ihren Raspberry Pi erhalten.

Tutorial: Bereitstellen Ihres Geräts in AWS IoT

In diesem Abschnitt werden die AWS IoT Core Ressourcen erstellt, die Ihr Tutorial verwenden wird.

Schritte zur Bereitstellung Ihres Geräts in AWS IoT

- Schritt 1: Erstellen Sie eine AWS IoT Richtlinie für den Device Shadow
- Schritt 2: Erstellen Sie eine Objekt-Ressource und fügen Sie die Richtlinie dem Objekt hinzu
- Schritt 3: Überprüfen Sie die Ergebnisse und die nächsten Schritte

Schritt 1: Erstellen Sie eine AWS IoT Richtlinie für den Device Shadow

X.509 Zertifikate authentifizieren Ihr Gerät mit AWS IoT Core. AWS IoT Dem Zertifikat sind Richtlinien beigefügt, die es dem Gerät ermöglichen, AWS IoT Vorgänge wie das Abonnieren oder Veröffentlichen von MQTT-Themen, die vom Device Shadow-Dienst verwendet werden, durchzuführen. Ihr Gerät legt sein Zertifikat vor, wenn es eine Verbindung herstellt und Nachrichten an das Gerät sendet. AWS IoT Core

In diesem Verfahren erstellen Sie eine Richtlinie, die es Ihrem Gerät ermöglicht, die zur Ausführung des Beispielprogramms erforderlichen AWS IoT -Vorgänge auszuführen. Wir empfehlen, dass Sie eine Richtlinie erstellen, die nur Berechtigungen gewährt, die zum Ausführen einer Aufgabe erforderlich sind. Sie erstellen zuerst die AWS IoT Richtlinie und hängen sie dann an das Gerätezertifikat an, das Sie später erstellen werden.

Um eine zu erstellen AWS IoT policy

1. Wählen Sie im linken Menü Sicher und dann Richtlinien aus. Wenn in Ihrem Konto bereits Richtlinien vorhanden sind, wählen Sie Erstellen aus. Andernfalls wählen Sie auf der Seite Sie haben noch keine Richtlinie erstellt die Option Richtlinie erstellen aus.
2. Auf der Seite Create a policy (Richtlinie erstellen):

    a. Geben Sie im Feld Name einen Namen für die Richtlinie ein (z. B. **`My_Device_Shadow_policy`**). Verwenden Sie keine personenbezogenen Informationen in Ihren Richtliniennamen.

    b. In dem Richtliniendokument beschreiben Sie die Aktionen „Verbinden“, „Abonnieren“, „Empfangen“ und „Veröffentlichen“, die dem Gerät die Erlaubnis geben, die reservierten MQTT-Themen zu veröffentlichen und zu abonnieren.

    Kopieren Sie die folgende Beispielrichtlinie und fügen Sie sie in Ihr Richtliniendokument ein. `thingname`Ersetzen Sie es durch den Namen des Objekts, das Sie erstellen werden (z. B.`My_light_bulb`), `region` durch die AWS IoT Region, in der Sie die Dienste nutzen, und

`account` durch Ihre AWS-Konto Nummer. Weitere Informationen zu AWS IoT Richtlinien finden Sie unterAWS IoT Core Richtlinien.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/get",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/update"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/get/accepted",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/get/rejected",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/update/accepted",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/update/rejected",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/
things/thingname/shadow/update/delta"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
```

```
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingname/shadow/get/accepted",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingname/shadow/get/rejected",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingname/shadow/update/accepted",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingname/shadow/update/rejected",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingname/shadow/update/delta"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/test-*"
        }
    ]
}
```

Schritt 2: Erstellen Sie eine Objekt-Ressource und fügen Sie die Richtlinie dem Objekt hinzu

Geräte, mit denen eine Verbindung hergestellt wird, AWS IoT können durch Ding-Ressourcen in der AWS IoT Registrierung dargestellt werden. Eine Objekt-Ressource steht für ein bestimmtes Gerät oder eine logische Einheit, z. B. die Glühbirne in diesem Tutorial.

Um zu erfahren, wie Sie ein Objekt in erstellen AWS IoT, folgen Sie den unter beschriebenen SchrittenDies erstellt ein Objekt. Hier sind einige wichtige Dinge, die Sie beachten sollten, wenn Sie die Schritte in diesem Tutorial befolgen:

1. Wählen Sie Ein einzelnes Objekt erstellen und geben Sie im Feld Name einen Namen für das Objekt ein, der `thingname` entspricht (z. B. `My_light_bulb`), den Sie bei der Erstellung der Richtlinie zuvor angegeben haben.

   Es ist nicht möglich, einen Objektnamen nach dem Erstellen umzubenennen. Wenn Sie dem Objekt einen anderen Namen als `thingname` gegeben haben, erstellen Sie ein neues Objekt mit dem Namen `thingname` und löschen Sie das alte Objekt.

> **Note**
>
> Verwenden Sie keine personenbezogenen Informationen in Ihren Objektnamen. Der Name des Objekts kann in unverschlüsselten Mitteilungen und Berichten vorkommen.

2. Wir empfehlen, dass Sie alle Zertifikatsdateien auf der Seite Zertifikat erstellt! in ein Verzeichnis herunterladen, in dem Sie sie leicht finden können. Sie müssen diese Dateien installieren, um die Beispielanwendung ausführen zu können.

   Wir empfehlen Ihnen, die Dateien in ein `certs` Unterverzeichnis in Ihrem `home` Verzeichnis auf dem Raspberry Pi herunterzuladen und sie jeweils mit einem einfacheren Namen zu benennen, wie in der folgenden Tabelle vorgeschlagen.

   Namen der Zertifikatsdateien

Datei	Dateipfad
CA-Stammzertifikat	`~/certs/Amazon-root-CA-1.pem`
Gerätezertifikat	`~/certs/device.pem.crt`
Privater Schlüssel	`~/certs/private.pem.key`

3. Nachdem Sie das Zertifikat aktiviert haben, mit dem Verbindungen hergestellt werden sollen AWS IoT, wählen Sie Richtlinie anhängen und stellen Sie sicher, dass Sie die zuvor erstellte Richtlinie (z. B.**`My_Device_Shadow_policy`**) an das Ding anhängen.

   Nachdem Sie ein Ding erstellt haben, können Sie sehen, dass Ihre Ding-Ressource in der Liste der Dinge in der AWS IoT Konsole angezeigt wird.

Schritt 3: Überprüfen Sie die Ergebnisse und die nächsten Schritte

In diesem Kurs haben Sie Folgendes gelernt:

- Einrichten und Konfigurieren des Raspberry Pi-Geräts.
- Erstellen Sie ein AWS IoT Richtliniendokument, das Ihr Gerät zur Interaktion mit AWS IoT Diensten autorisiert.
- Erstellen Sie eine Ding-Ressource und X.509 das zugehörige Gerätezertifikat und hängen Sie das Richtliniendokument an.

Nächste Schritte

Sie können jetzt das AWS IoT Geräte-SDK für Python installieren, die `shadow.py` Beispielanwendung ausführen und Device Shadows verwenden, um den Status zu steuern. Weitere Informationen darüber, wie Sie dieses Tutorial ausführen, finden Sie unter Tutorial: Installation des Geräte-SDK und Ausführen der Beispielanwendung für Geräteschatten.

Tutorial: Installation des Geräte-SDK und Ausführen der Beispielanwendung für Geräteschatten

In diesem Abschnitt wird gezeigt, wie Sie die erforderliche Software und das AWS IoT Geräte-SDK für Python installieren und die `shadow.py` Beispielanwendung ausführen können, um das Shadow-Dokument zu bearbeiten und den Status des Shadows zu steuern.

In diesem Kurs lernen Sie Folgendes:

- Verwenden Sie die installierte Software und AWS IoT das Geräte-SDK für Python, um die Beispiel-App auszuführen.
- Lernen, wie die Eingabe eines Werts mithilfe der Beispiel-App den gewünschten Wert in der AWS IoT Konsole veröffentlicht.
- Prüfen der `shadow.py` Beispiel-App an und wie sie das MQTT-Protokoll verwendet, um den Status des Schattens zu aktualisieren.

Bevor Sie dieses Tutorial ausführen:

Sie müssen Ihr Raspberry Pi-Gerät eingerichtet AWS-Konto, konfiguriert und eine AWS IoT Sache und eine Richtlinie erstellt haben, die dem Gerät die Berechtigungen zum Veröffentlichen und Abonnieren der für MQTT reservierten Themen des Device Shadow-Dienstes erteilt. Weitere Informationen finden Sie unter Tutorial: Ihren Raspberry Pi für die Ausführung der Schattenanwendung vorbereiten.

Sie müssen auch Git, Python und das AWS IoT Device SDK für Python installiert haben. Dieses Tutorial baut auf den im Tutorial Verbinden eines Raspberry Pi oder eines anderes Gerätes vorgestellten Konzepten auf. Wenn Sie dieses Tutorial noch nicht ausprobiert haben, empfehlen wir Ihnen, die in diesem Tutorial beschriebenen Schritte zu befolgen, um die Zertifikatsdateien und das Geräte-SDK zu installieren, und dann zu diesem Tutorial zurückzukehren, um die `shadow.py` Beispiel-App auszuführen.

In diesem Tutorial führen Sie die folgenden Aktivitäten durch:

- Schritt 1: shadow.py-Beispiel-App ausführen
- Schritt 2: Geräte-SDK-Beispiel-App shadow.py prüfen
- Schritt 3: Beheben Sie Probleme mit der Beispiel-App shadow.py
- Schritt 4: Ergebnisse überprüfen und nächste Schritte

Für dieses Tutorial brauchen Sie ungefähr 20 Minuten.

Schritt 1: shadow.py-Beispiel-App ausführen

Bevor Sie die `shadow.py` Beispiel-App ausführen, benötigen Sie zusätzlich zu den Namen und dem Speicherort der installierten Zertifikatsdateien die folgenden Informationen.

Werte für Anwendungsparameter

Parameter	Wo der Wert gefunden werden kann
your-iot-thing-name	Name der AWS IoT Sache, die Sie zuvor erstellt habenthe section called “Schritt 2: Erstellen Sie eine Objekt-Ressource und fügen Sie die Richtlinie dem Objekt hinzu”. Um diesen Wert zu finden, wählen Sie in der AWS IoT Konsole Verwalten und dann Objekte aus.
your-iot-endpoint	Der *your-iot-endpoint* Wert hat das Format:*endpoint_id* `-ats.iot.` *region*`.amazonaws.com` , zum Beispiel`a3qj468EXAMPLE-ats.iot.us-west-2.amazonaws.com` . So finden Sie diesen Wert: 1. Wählen Sie in der AWS IoT -Konsole Verwalten und danach Objekte aus. 2. Wählen Sie das IoT-Ding aus My_light_ bulb, das Sie für Ihr Gerät erstellt haben und das Sie zuvor verwendet haben, und wählen Sie dann Interact. Ihr Endpunkt wird

Parameter	Wo der Wert gefunden werden kann
	im Abschnitt HTTPS auf der Seite mit den Objekt-Details angezeigt.

Installieren und Ausführen der Beispiel-App

1. Navigieren Sie zum Verzeichnis der Beispiel-App.

```
cd ~/aws-iot-device-sdk-python-v2/samples/service-clients
```

2. Ersetzen Sie im Befehlszeilenfenster *your-iot-endpoint* und *your-iot-thing-name* wie angegeben und führen Sie diesen Befehl aus.

```
python3 shadow.py --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/certs/
device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint --
thing_name your-iot-thing-name
```

3. Beachten Sie, dass die Beispiel-App:

   1. Stellt eine Verbindung zum AWS IoT-Dienst für Ihr Konto her.
   2. `Delta`-Ereignisse und `Update` und `Get`-Antworten abonniert.
   3. Sie dazu auffordert, einen gewünschten Wert in das Terminal einzugeben.
   4. Die Ausgabe sieht ähnlich aus wie:

```
Connecting to a3qEXAMPLEffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-0c8ae2ff-cc87-49d2-a82a-ae7ba1d0ca5a'...
Connected!
Subscribing to Delta events...
Subscribing to Update responses...
Subscribing to Get responses...
Requesting current shadow state...
Launching thread to read user input...
Finished getting initial shadow state.
Shadow contains reported value 'off'.
Enter desired value:
```

> **Note**
>
> Wenn Sie Probleme beim Ausführen der `shadow.py`-Beispiel-App haben, überprüfen Sie the section called “Schritt 3: Beheben Sie Probleme mit der `Beispiel-App shadow.py`”. Um zusätzliche Informationen zu erhalten, die Ihnen bei der Behebung des Problems helfen könnten, fügen Sie den `--verbosity debug` Parameter zur Befehlszeile hinzu, sodass die Beispiel-App detaillierte Meldungen darüber anzeigt, was sie tut.

Geben Sie Werte ein und beobachten Sie die Aktualisierungen im Schattendokument

Sie können Werte in das Terminal eingeben, um den `desired`-Wert anzugeben, wodurch auch der `reported`-Wert aktualisiert wird. Angenommen, Sie geben die Farbe `yellow` im Terminal ein. Der `reported`-Wert wird ebenfalls auf die Farbe `yellow` aktualisiert. Im Folgenden werden die im Terminal angezeigten Meldungen angezeigt:

```
Enter desired value:
yellow
Changed local shadow value to 'yellow'.
Updating reported shadow value to 'yellow'...
Update request published.
Finished updating reported shadow value to 'yellow'.
```

Wenn Sie diese Aktualisierungsanforderung veröffentlichen, AWS IoT wird ein standardmäßiger, klassischer Schatten für die Ding-Ressource erstellt. Sie können die Aktualisierungsanforderung, die Sie für die `desired` Werte `reported` und veröffentlicht haben, in der AWS IoT Konsole beobachten, indem Sie sich das Shadow-Dokument für die von Ihnen erstellte Ding-Ressource ansehen (z. B.`My_light_bulb`). Um das Update im Schattendokument zu sehen:

1. Wählen Sie in der AWS IoT Konsole „Verwalten“ und dann „Dinge“ aus.
2. Wählen Sie in der angezeigten Liste das Objekt aus, das Sie erstellt haben, wählen Sie Schatten und anschließend Klassischer Schatten.

Das Schattendokument sollte wie folgt aussehen und für die `reported`- und `desired`-Werte die Farbe `yellow` anzeigen. Sie sehen diese Werte im Abschnitt Schattenstatus des Dokuments.

```
{
"desired": {
```

```
  "welcome": "aws-iot",
  "color": "yellow"
},
"reported": {
  "welcome": "aws-iot",
  "color": "yellow"
}
}
```

Sie sehen auch einen Abschnitt mit den Metadaten, der die Zeitstempelinformationen und die Versionsnummer der Anfrage enthält.

Sie können die Version des Statusdokuments verwenden, um sicherzustellen, dass Sie die neueste Version eines Geräteschattendokuments aktualisieren. Wenn Sie eine weitere Aktualisierungsanfrage senden, wird die Versionsnummer um 1 erhöht. Geben Sie bei einer Aktualisierungsanfrage eine Version an, lehnt der Service die Anfrage mit einem Konflikt-Antwortcode HTTP 409 ab, wenn die aktuelle Version des Statusdokuments nicht der angegebenen Version entspricht.

```
{
"metadata": {
  "desired": {
    "welcome": {
      "timestamp": 1620156892
    },
    "color": {
      "timestamp": 1620156893
    }
  },
  "reported": {
    "welcome": {
      "timestamp": 1620156892
    },
    "color": {
      "timestamp": 1620156893
    }
  }
},
"version": 10
}
```

Um mehr über das Schattendokument zu erfahren und Änderungen an den Statusinformationen einzusehen, fahren Sie mit dem nächsten Tutorial Tutorial: Interaktion mit Geräteschatten mithilfe der

Beispiel-App und des MQTT-Testclients fort, wie im Schritt 4: Ergebnisse überprüfen und nächste Schritte Abschnitt dieses Tutorials beschrieben. Optional können Sie im folgenden Abschnitt auch mehr über den `shadow.py`-Beispielcode und dessen Verwendung des MQTT-Protokolls erfahren.

Schritt 2: Geräte-SDK-Beispiel-App shadow.py prüfen

In diesem Abschnitt wird die `shadow.py`-Beispiel-App aus dem AWS IoT -Geräte-SDK v2 für Python beschrieben, die in diesem Tutorial verwendet wird. Hier werden wir überprüfen, wie AWS IoT Core mithilfe des MQTT- und MQTT-über-WSS-Protokolls eine Verbindung hergestellt wird. Die AWS Common Runtime (AWS-CRT) -Bibliothek bietet die Unterstützung von Kommunikationsprotokollen auf niedriger Ebene und ist im AWS IoT Device SDK v2 für Python enthalten.

Dieses Tutorial verwendet zwar MQTT und MQTT über WSS, AWS IoT unterstützt aber Geräte, die HTTPS-Anfragen veröffentlichen. Ein Beispiel für ein Python-Programm, das eine HTTP-Nachricht von einem Gerät sendet, finden Sie im HTTPS-Codebeispiel unter Verwendung der Python `requests`-Bibliothek.

Informationen darüber, wie Sie eine fundierte Entscheidung darüber treffen können, welches Protokoll Sie für die Kommunikation mit Ihrem Gerät verwenden möchten, finden Sie unter Wählen Sie ein Anwendungsprotokoll für Ihre Gerätekommunikation.

MQTT

Das `shadow.py` Beispiel ruft `mtls_from_path` (hier gezeigt) auf, `mqtt_connection_builder` um AWS IoT Core mithilfe des MQTT-Protokolls eine Verbindung herzustellen. `mtls_from_path`verwendet X.509 Zertifikate und TLS v1.2, um das Gerät zu authentifizieren. Die AWS-CRT-Bibliothek verarbeitet die untergeordneten Details dieser Verbindung.

```
mqtt_connection = mqtt_connection_builder.mtls_from_path(
  endpoint=args.endpoint,
  cert_filepath=args.cert,
  pri_key_filepath=args.key,
  ca_filepath=args.ca_file,
  client_bootstrap=client_bootstrap,
  on_connection_interrupted=on_connection_interrupted,
  on_connection_resumed=on_connection_resumed,
  client_id=args.client_id,
  clean_session=False,
  keep_alive_secs=6
)
```

- `endpoint`ist Ihr AWS IoT Endpunkt, den Sie über die Befehlszeile eingegeben haben, und `client_id` ist die ID, die dieses Gerät in der eindeutig identifiziert. AWS-Region
- `cert_filepath`, `pri_key_filepath` und `ca_filepath` sind die Pfade zu den Zertifikats- und privaten Schlüsseldateien des Geräts sowie zur Root-CA-Datei.
- `client_bootstrap` ist das allgemeine Runtime-Objekt, das Socket-Kommunikationsaktivitäten verarbeitet und vor dem Aufruf von `mqtt_connection_builder.mtls_from_path` instanziiert wird.
- `on_connection_interrupted` und `on_connection_resumed` sind Callback-Funktionen, die aufgerufen werden, wenn die Verbindung des Geräts unterbrochen und wieder aufgenommen wird.
- `clean_session` gibt an, ob eine neue, persistente Sitzung gestartet werden soll oder ob, falls eine vorhanden ist, die Verbindung zu einer bestehenden wiederhergestellt werden soll. `keep_alive_secs` ist der Keep-Alive-Wert in Sekunden, der in der `CONNECT`-Anfrage gesendet werden soll. In diesem Intervall wird automatisch ein Ping gesendet. Der Server geht davon aus, dass die Verbindung unterbrochen ist, wenn er nach dem 1,5-fachen dieses Werts keinen Ping erhält.

Das `shadow.py` Beispiel ruft auch `websockets_with_default_aws_signing` in `mqtt_connection_builder` auf, um mithilfe des MQTT-Protokolls über WSS eine Verbindung zu AWS IoT Core herzustellen. MQTT über WSS verwendet ebenfalls dieselben Parameter wie MQTT und verwendet diese zusätzlichen Parameter:

- `region`ist die AWS Signaturregion, die für die Signature V4-Authentifizierung verwendet `credentials_provider` wird, und sind die AWS Anmeldeinformationen, die für die Authentifizierung bereitgestellt werden. Die Region wird von der Befehlszeile aus übergeben, und das `credentials_provider`-Objekt wird unmittelbar vor dem Aufruf von `mqtt_connection_builder.websockets_with_default_aws_signing` instanziiert.
- `websocket_proxy_options` sind die HTTP-Proxyoptionen, wenn ein Proxyhost verwendet wird. In der `shadow.py` Beispiel-App wird dieser Wert unmittelbar vor dem Aufruf von `mqtt_connection_builder.websockets_with_default_aws_signing` instanziiert.

Abonnieren von Shadow-Themen und -Ereignissen

Das `shadow.py`-Beispiel versucht, eine Verbindung herzustellen, und wartet, bis die Verbindung vollständig hergestellt ist. Wenn es nicht verbunden ist, werden Befehle in die Warteschlange gestellt. Sobald die Verbindung hergestellt ist, abonniert das Beispiel Delta-Ereignisse und Aktualisierungs-

und Abrufnachrichten und veröffentlicht Nachrichten mit einer QoS-Stufe (Quality of Service) von 1 (`mqtt.QoS.AT_LEAST_ONCE`).

Wenn ein Gerät eine Nachricht mit QoS Level 1 abonniert, speichert der Message Broker die Nachrichten, die das Gerät abonniert hat, bis sie an das Gerät gesendet werden können. Der Message Broker sendet die Nachrichten erneut, bis er eine PUBACK-Antwort vom Gerät erhält.

Weitere Informationen zum MQTT-Protokoll finden Sie unter Ansehen des MQTT-Protokolls und MQTT.

Weitere Informationen zur Verwendung von MQTT, MQTT über WSS, persistente Sitzungen und QoS-Stufen, die in diesem Tutorial verwendet werden, finden Sie unter Ansehen der Beispiel-App „pubsub.py“ des Device SDK.

Schritt 3: Beheben Sie Probleme mit der **`Beispiel-App`** shadow.py

Wenn Sie die `shadow.py`-Beispiel-App ausführen, sollten im Terminal einige Meldungen und eine Aufforderung zur Eingabe eines `desired`-Werts angezeigt werden. Wenn das Programm einen Fehler ausgibt, können Sie zum Debuggen des Fehlers zunächst überprüfen, ob Sie den richtigen Befehl für Ihr System ausgeführt haben.

In einigen Fällen weist die Fehlermeldung möglicherweise auf Verbindungsprobleme hin und sieht ähnlich aus wie: `Host name was invalid for dns resolution` oder `Connection was closed unexpectedly`. In solchen Fällen können Sie Folgendes überprüfen:

- Die Endpunktadresse im Befehl

  Das `endpoint`-Argument in dem Befehl, den Sie zum Ausführen der Beispiel-App eingegeben haben (z. B. `a3qEXAMPLEffp-ats.iot.us-west-2.amazonaws.com`). Diesen Wert können Sie auch in der AWS IoT Konsole prüfen.

  Um zu überprüfen, ob Sie den richtigen Wert verwendet haben:

  1. Wählen Sie in der AWS IoT -Konsole Verwalten und danach Objekte.
  2. Wählen Sie das Ding aus, das Sie für Ihre Beispiel-App erstellt haben (z. B. My_light_bulb), und wählen Sie dann Interact.

  Ihr Endpunkt wird im Abschnitt HTTPS auf der Seite mit den Objekt-Details angezeigt. Sie sollten auch eine Meldung sehen, die besagt: `This thing already appears to be connected.`
- Aktivierung des Zertifikats überprüfen

Zertifikate authentifizieren Ihr Gerät mit AWS IoT Core.

Um zu überprüfen, ob Ihr Zertifikat aktiv ist:

1. Wählen Sie in der AWS IoT -Konsole Verwalten und danach Objekte.
2. Wählen Sie das Ding aus, das Sie für Ihre Beispiel-App erstellt haben (z. B. My_light_bulb), und wählen Sie dann Sicherheit.
3. Wählen Sie das Zertifikat aus und wählen Sie dann auf der Detailseite des Zertifikats die Option „Zertifikat auswählen“ und dann auf der Detailseite des Zertifikats die Option Aktionen aus.

Wenn in der Dropdownliste Aktivieren nicht verfügbar ist und Sie nur Deaktivieren auswählen können, ist Ihr Zertifikat aktiv. Wenn nicht, wählen Sie Aktivieren und führen Sie das Beispielprogramm erneut aus.

Wenn das Programm immer noch nicht ausgeführt wird, überprüfen Sie die Namen der Zertifikatsdateien im `certs`-Ordner.

- Überprüfen der mit der Objekt-Ressource verknüpften Richtlinie

  Während Zertifikate Ihr Gerät authentifizieren, ermöglichen AWS IoT Richtlinien dem Gerät, bestimmte AWS IoT Vorgänge auszuführen, z. B. das Abonnieren oder Veröffentlichen von für MQTT reservierten Themen.

  Um zu überprüfen, ob die richtige Richtlinie angehängt ist:

  1. Suchen Sie das Zertifikat wie zuvor beschrieben und wählen Sie dann Richtlinien aus.
  2. Wählen Sie die angezeigte Richtlinie aus und überprüfen Sie, ob sie die Aktionen `connect`, `subscribe`, `receive` und `publish` beschreibt, die dem Gerät die Erlaubnis geben, die reservierten MQTT-Themen zu veröffentlichen und zu abonnieren.

     Eine Beispielrichtlinie finden Sie unter Schritt 1: Erstellen Sie eine AWS IoT Richtlinie für den Device Shadow.

  Wenn Sie Fehlermeldungen sehen, die auf Probleme beim Herstellen einer Verbindung hinweisen AWS IoT, kann das an den Berechtigungen liegen, die Sie für die Richtlinie verwenden. In diesem Fall empfehlen wir, mit einer Richtlinie zu beginnen, die vollen Zugriff auf AWS IoT Ressourcen gewährt, und dann das Beispielprogramm erneut auszuführen. Sie können entweder die aktuelle Richtlinie bearbeiten oder die aktuelle Richtlinie auswählen, Trennen auswählen und dann eine weitere Richtlinie erstellen, die vollen Zugriff gewährt, und diese an Ihre Objekt-Ressource

anhängen. Sie können die Richtlinie später nur auf die Aktionen und Richtlinien beschränken, die Sie zur Ausführung des Programms benötigen.

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

- Überprüfen der Installation Ihres Geräte-SDKs

  Wenn das Programm immer noch nicht ausgeführt wird, können Sie das Geräte-SDK erneut installieren, um sicherzustellen, dass Ihre SDK-Installation vollständig und korrekt ist.

Schritt 4: Ergebnisse überprüfen und nächste Schritte

In diesem Kurs haben Sie gelernt, wie Sie:

- Installieren Sie die erforderliche Software, Tools und das AWS IoT Geräte-SDK für Python.
- Die Beispiel-App `shadow.py` das MQTT-Protokoll zum Abrufen und Aktualisieren des aktuellen Status des Schattens verwendet.
- Führen Sie die Beispiel-App für Device Shadows aus und beobachten Sie die Aktualisierung des Shadow-Dokuments in der AWS IoT Konsole. Sie haben darüber hinaus gelernt, Probleme und Fehler bei der Ausführung des Programms zu beheben.

Nächste Schritte

Sie können jetzt die `shadow.py`-Beispielanwendung ausführen und Geräteschatten verwenden, um den Status zu kontrollieren. Sie können die Aktualisierungen des Schattendokuments in der AWS IoT -Konsole verfolgen und Delta-Ereignisse beobachten, auf die die Beispiel-App reagiert. Mit dem MQTT-Testclient können Sie die reservierten Schattenthemen abonnieren und die Nachrichten beobachten, die die Themen bei der Ausführung des Beispielprogramms erhalten. Weitere

Informationen darüber, wie Sie dieses Tutorial ausführen, finden Sie unter Tutorial: Interaktion mit Geräteschatten mithilfe der Beispiel-App und des MQTT-Testclients.

Tutorial: Interaktion mit Geräteschatten mithilfe der Beispiel-App und des MQTT-Testclients

Um mit der `shadow.py`-Beispiel-App zu interagieren, geben Sie im Terminal einen Wert für den `desired`-Wert ein. Sie können beispielsweise Farben angeben, die den Ampeln ähneln und AWS IoT auf die Anfrage reagieren und die gemeldeten Werte aktualisieren.

In diesem Kurs lernen Sie Folgendes:

- Verwenden der `shadow.py`-Beispiel-App, um die gewünschten Status anzugeben und den aktuellen Status des Schattens zu aktualisieren.
- Bearbeiten Sie das Schattendokument, um Delta-Ereignisse und die Reaktion der `shadow.py`-Beispiel-App auf diese Ereignisse zu beobachten.
- Verwenden Sie den MQTT-Testclient, um Schattenthemen zu abonnieren und Updates zu beobachten, wenn Sie das Beispielprogramm ausführen.

Für dieses Tutorial benötigen Sie Folgendes:

Richten Sie Ihr Raspberry Pi-Gerät ein AWS-Konto, konfigurieren Sie Ihr Raspberry Pi-Gerät und erstellen Sie eine AWS IoT Sache und eine Richtlinie. Sie müssen auch die erforderliche Software, das Geräte-SDK und die Zertifikatsdateien installiert haben und das Beispielprogramm im Terminal ausführen. Weitere Informationen finden Sie in den vorherigen Tutorials Tutorial: Ihren Raspberry Pi für die Ausführung der Schattenanwendung vorbereiten und Schritt 1: shadow.py-Beispiel-App ausführen. Sie müssen diese Tutorials abschließen, wenn das noch nicht geschehen ist.

In diesem Tutorial führen Sie die folgenden Aktivitäten durch:

- Schritt 1: Aktualisieren Sie die gewünschten und gemeldeten Werte mithilfe der Beispiel-App shadow.py
- Schritt 2: Nachrichten aus der Beispiel-App shadow.py im MQTT-Testclient anzeigen
- Schritt 3: Fehlerbehebung bei Interaktionen mit Geräteschatten
- Schritt 4: Überprüfen der Ergebnisse und nächste Schritte

Für dieses Tutorial brauchen Sie ungefähr 45 Minuten.

Schritt 1: Aktualisieren Sie die gewünschten und gemeldeten Werte mithilfe der **Beispiel-App** shadow.py

Im vorherigen Tutorial haben Sie gelerntSchritt 1: shadow.py-Beispiel-App ausführen, wie Sie eine Meldung beobachten können, die im Shadow-Dokument in der AWS IoT Konsole veröffentlicht wird, wenn Sie einen gewünschten Wert eingeben, wie im Abschnitt beschriebenTutorial: Installation des Geräte-SDK und Ausführen der Beispielanwendung für Geräteschatten.

Im vorherigen Beispiel haben wir die gewünschte Farbe auf `yellow` eingestellt. Nachdem Sie jeden Wert eingegeben haben, fordert Sie das Terminal auf, einen anderen `desired`-Wert einzugeben. Wenn Sie erneut denselben Wert eingeben (`yellow`), erkennt die App dies und fordert Sie auf, einen neuen `desired`-Wert einzugeben.

```
Enter desired value:
yellow
Local value is already 'yellow'.
Enter desired value:
```

Nehmen wir nun an, Sie geben die Farbe ein`green`. AWS IoT reagiert auf die Anfrage und aktualisiert den `reported` Wert auf`green`. Auf diese Weise erfolgt die Aktualisierung, wenn sich der `desired`-Status vom `reported`-Status unterscheidet, was zu einem Delta führt.

So simuliert die **Beispiel-App** shadow.py Device Shadow-Interaktionen:

1. Geben Sie einen `desired`-Wert (sagen wir `yellow`) in das Terminal ein, um den gewünschten Status zu veröffentlichen.
2. Da sich der `desired`-Status vom `reported`-Status unterscheidet (sagen wir die Farbe `green`), tritt ein Delta auf, und die App, die das Delta abonniert hat, empfängt diese Nachricht.
3. Die App reagiert auf die Nachricht und aktualisiert ihren Status auf den `desired`-Wert `yellow`.
4. Die App veröffentlicht dann eine Aktualisierungsnachricht mit dem neuen gemeldeten Wert des Gerätestatus, `yellow`.

Im Folgenden werden die im Terminal angezeigten Meldungen angezeigt, aus denen hervorgeht, wie die Aktualisierungsanforderung veröffentlicht wird.

```
Enter desired value:
green
Changed local shadow value to 'green'.
```

```
Updating reported shadow value to 'green'...
Update request published.
Finished updating reported shadow value to 'green'.
```

In der AWS IoT Konsole gibt das Shadow-Dokument den aktualisierten Wert `green` für die `desired` Felder `reported` und wieder, und die Versionsnummer wird um 1 erhöht. Wenn die vorherige Versionsnummer beispielsweise als 10 angezeigt wurde, wird die aktuelle Versionsnummer als 11 angezeigt.

> **Note**
>
> Durch das Löschen eines Schattens wird seine Versionsnummer nicht sofort auf null zurückgesetzt. Sie werden feststellen, dass die Schattenversion um 1 erhöht wird, wenn Sie eine Aktualisierungsanforderung veröffentlichen oder einen anderen Schatten mit demselben Namen erstellen.

Das Schattendokument berbeiten, um Delta-Ereignisse zu beobachten

Die `shadow.py`-Beispiel-App hat auch `delta`-Ereignisse abonniert und reagiert, wenn sich der `desired`-Wert ändert. Sie können den `desired`-Wert beispielsweise in die Farbe `red` ändern. Bearbeiten Sie dazu in der AWS IoT Konsole das Shadow-Dokument, indem Sie auf Bearbeiten klicken und dann den `desired` Wert `red` in der JSON-Datei auf festlegen, wobei der `reported` Wert auf `green` beibehalten wird. Bevor Sie die Änderungen speichern, lassen Sie das Terminal auf dem Raspberry Pi geöffnet, da im Terminal Meldungen angezeigt werden, wenn die Änderung erfolgt.

```
{
"desired": {
  "welcome": "aws-iot",
  "color": "red"
},
"reported": {
  "welcome": "aws-iot",
  "color": "green"
}
}
```

Nachdem Sie den neuen Wert gespeichert haben, reagiert die `shadow.py`-Beispiel-App auf diese Änderung und zeigt im Terminal Meldungen an, die auf das Delta hinweisen. Sie sollten dann die folgenden Meldungen unter der Aufforderung zur Eingabe des `desired`-Werts sehen.

```
Enter desired value:
Received shadow delta event.
Delta reports that desired value is 'red'. Changing local value...
Changed local shadow value to 'red'.
Updating reported shadow value to 'red'...
Finished updating reported shadow value to 'red'.
Enter desired value:
Update request published.
Finished updating reported shadow value to 'red'.
```

Schritt 2: Nachrichten aus der **`Beispiel-App`** shadow.py im MQTT-Testclient anzeigen

Sie können den MQTT-Testclient in der AWS IoT Konsole verwenden, um MQTT-Nachrichten zu überwachen, die zu Ihrem AWS-Kontoübermittelt werden. Wenn Sie reservierte MQTT-Themen abonnieren, die vom Geräteschattendienst verwendet werden, können Sie die Nachrichten beobachten, die von den Themen beim Ausführen der Beispiel-App empfangen wurden.

Wenn Sie den MQTT-Testclient noch nicht verwendet haben, können Sie sich MQTT-Nachrichten anzeigen mit dem AWS IoT MQTT-Client ansehen. Auf diese Weise lernen Sie, wie Sie den MQTT-Testclient in der AWS IoT Konsole verwenden, um MQTT-Nachrichten anzuzeigen, während sie durch den Message Broker laufen.

1. Öffnen des MQTT-Testclients

   Öffnen Sie den MQTT-Testclient in der AWS IoT -Konsole in einem neuen Fenster, damit Sie die von den MQTT-Themen empfangenen Nachrichten beobachten können, ohne die Konfiguration Ihres MQTT-Testclients zu verlieren. Der MQTT-Testclient speichert keine Abonnements oder Nachrichtenprotokolle, wenn Sie ihn verlassen, um zu einer anderen Seite in der Konsole zu wechseln. Für diesen Abschnitt des Tutorials können Sie das Shadow-Dokument Ihres AWS IoT Dings und den MQTT-Testclient in separaten Fenstern öffnen lassen, um die Interaktion mit Device Shadows leichter beobachten zu können.

2. Abonnieren der reservierten MQTT-Schatten-Themen

   Sie können den MQTT-Testclient verwenden, um die Namen der reservierten MQTT-Themen von Geräteschatten einzugeben und sie zu abonnieren, um Updates zu erhalten, wenn Sie die `shadow.py`-Beispiel-App ausführen. So abonnieren Sie MQTT-Themen:

   a. Wählen Sie im MQTT-Testclient in der AWS IoT Konsole die Option Thema abonnieren aus.

b. Geben Sie im Abschnitt Themenfilter Folgendes ein: $aws/things/***thingname***/shadow/update/#. Hier ist `thingname` der Name der Objekt-Ressource, die Sie zuvor erstellt haben (z. B. `My_light_bulb`).

c. Behalten Sie die Standardwerte für die zusätzlichen Konfigurationseinstellungen bei und wählen Sie dann Abonnieren.

Durch die Verwendung des Platzhalters # im Themenabonnement können Sie mehrere MQTT-Themen gleichzeitig abonnieren und alle Nachrichten, die zwischen dem Gerät und seinem Schatten ausgetauscht werden, in einem einzigen Fenster beobachten. Weitere Informationen über Platzhalterzeichen und ihre Verwendung finden Sie unter MQTT-Themen.

3. Führen Sie das **`Beispielprogramm`** shadow.py aus und beobachten Sie die Meldungen

Wenn Sie in Ihrem Befehlszeilenfenster des Raspberry Pi das Programm getrennt haben, führen Sie die Beispiel-App erneut aus und sehen Sie sich die Meldungen im MQTT-Testclient in der AWS IoT Konsole an.

a. Geben Sie den folgenden Befehl ein, um das Beispielprogramm neu zu starten. Ersetzen Sie *`your-iot-thing-name`* und *`your-iot-endpoint`* durch die Namen des Objekts AWS IoT , das Sie zuvor erstellt haben (z. B.`My_light_bulb`), und durch den Endpunkt, der mit dem Gerät interagieren soll.

```
cd ~/aws-iot-device-sdk-python-v2/samples/service-clients
python3 shadow.py --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/certs/
device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint --
thing_name your-iot-thing-name
```

Die `shadow.py`-Beispiel-App wird dann ausgeführt und ruft den aktuellen Schattenstatus ab. Wenn Sie den Schatten oder den aktuellen Status gelöscht haben, setzt das Programm den aktuellen Wert auf `off` und fordert Sie dann auf, einen `desired`-Wert einzugeben.

```
Connecting to a3qEXAMPLEffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-0c8ae2ff-cc87-49d2-a82a-ae7ba1d0ca5a'...
Connected!
Subscribing to Delta events...
Subscribing to Update responses...
Subscribing to Get responses...
Requesting current shadow state...
Launching thread to read user input...
```

```
Finished getting initial shadow state.
Shadow document lacks 'color' property. Setting defaults...
Changed local shadow value to 'off'.
Updating reported shadow value to 'off'...
Update request published.
Finished updating reported shadow value to 'off'...
Enter desired value:
```

Wenn das Programm jedoch ausgeführt wurde und Sie es neu gestartet haben, wird der letzte Farbwert im Terminal gemeldet. Im MQTT-Testclient werden Sie ein Update für die Themen $aws/things//shadow/getund $***thingname***/aws/things***thingname***/shadow/get/accepted sehen.

Nehmen wir an, die zuletzt gemeldete Farbe war `green`. Im Folgenden wird der Inhalt der $aws/things/***thingname***/shadow/get/accepted JSON-Datei gezeigt.

```
{
"state": {
  "desired": {
    "welcome": "aws-iot",
    "color": "green"
  },
  "reported": {
    "welcome": "aws-iot",
    "color": "green"
  }
},
"metadata": {
  "desired": {
    "welcome": {
      "timestamp": 1620156892
    },
    "color": {
      "timestamp": 1620161643
    }
  },
  "reported": {
    "welcome": {
      "timestamp": 1620156892
    },
    "color": {
      "timestamp": 1620161643
```

```
    }
  }
},
"version": 10,
"timestamp": 1620173908
}
```

b. Geben Sie einen `desired`-Wert in das Terminal ein, z. B. `yellow`. Die `shadow.py`-Beispiel-App reagiert und zeigt im Terminal die folgenden Meldungen an, die die Änderung des `reported`-Werts auf `yellow` zeigen.

```
Enter desired value:
yellow
Changed local shadow value to 'yellow'.
Updating reported shadow value to 'yellow'...
Update request published.
Finished updating reported shadow value to 'yellow'.
```

Im MQTT-Testclient in der AWS IoT Konsole sehen Sie unter Abonnements, dass die folgenden Themen eine Nachricht erhalten haben:

- $aws/things/***thingname***/shadow/update: zeigt, dass sich `desired` sowohl die `updated` N-Werte als auch die Farbe `yellow` ändern.
- $aws/things/***thingname***/shadow/update/accepted: zeigt die aktuellen Werte der `reported` UND-Staaten `desired` sowie ihre Metadaten und Versionsinformationen an.
- $aws/things/***thingname***/shadow/update/documents: zeigt die vorherigen und aktuellen Werte der `desired` und `reported` Staaten sowie ihre Metadaten und Versionsinformationen an.

Da das Dokument $aws/things/***thingname***/shadow/update/documents auch Informationen enthält, die in den anderen beiden Themen enthalten sind, können wir es überprüfen, um die Statusinformationen zu sehen. Der vorherige Status zeigt den auf `green` festgelegten Wert, seine Metadaten und Versionsinformationen sowie den aktuellen Status `yellow`, auf den der gemeldete Wert aktualisiert wurde.

```
{
"previous": {
  "state": {
```

```
    "desired": {
      "welcome": "aws-iot",
      "color": "green"
    },
    "reported": {
      "welcome": "aws-iot",
      "color": "green"
    }
  },
  "metadata": {
    "desired": {
      "welcome": {
        "timestamp": 1617297888
      },
      "color": {
        "timestamp": 1617297898
      }
    },
    "reported": {
      "welcome": {
        "timestamp": 1617297888
      },
      "color": {
        "timestamp": 1617297898
      }
    }
  },
  "version": 10
},
"current": {
  "state": {
    "desired": {
      "welcome": "aws-iot",
      "color": "yellow"
    },
    "reported": {
      "welcome": "aws-iot",
      "color": "yellow"
    }
  },
  "metadata": {
    "desired": {
      "welcome": {
        "timestamp": 1617297888
```

```
      },
      "color": {
        "timestamp": 1617297904
      }
    },
    "reported": {
      "welcome": {
        "timestamp": 1617297888
      },
      "color": {
        "timestamp": 1617297904
      }
    }
  },
  "version": 11
},
"timestamp": 1617297904
}
```

c. Wenn Sie nun einen anderen `desired`-Wert eingeben, sehen Sie weitere Änderungen an den `reported`-Werten und Nachrichtenaktualisierungen, die in diesen Themen eingegangen sind. Die Versionsnummer wird ebenfalls um 1 erhöht. Wenn Sie beispielsweise den Wert `green` eingeben, meldet der vorherige Status den Wert `yellow` und der aktuelle Status meldet den Wert `green`.

4. Bearbeiten des Schattendokuments, um Delta-Ereignisse zu beobachten

   Um Änderungen am Delta-Thema zu beobachten, bearbeiten Sie das Schattendokument in der AWS IoT -Konsole. Sie können den `desired`-Wert beispielsweise in die Farbe `red` ändern. Wählen Sie dazu in der AWS IoT Konsole „Bearbeiten“ und setzen Sie dann den `desired` Wert in der JSON-Datei auf Rot, während der `reported` Wert auf `green` beibehalten wird. Lassen Sie das Terminal geöffnet, bevor Sie die Änderung speichern, da die Deltameldung im Terminal angezeigt wird.

```
{
"desired": {
  "welcome": "aws-iot",
  "color": "red"
},
"reported": {
  "welcome": "aws-iot",
  "color": "green"
```

```
}
}
```

Die `shadow.py`-Beispiel-App reagiert auf diese Änderung und zeigt im Terminal Meldungen an, die auf das Delta hinweisen. Im MQTT-Testclient haben die `update`-Themen eine Nachricht erhalten, die Änderungen an den `desired`- und `reported`-Werten anzeigt.

Sie sehen auch, dass das Thema $aws/things/***thingname***/shadow/update/delta eine Nachricht erhalten hat. Um die Nachricht zu sehen, wählen Sie dieses Thema aus, das unter Abonnements aufgeführt ist.

```
{
"version": 13,
"timestamp": 1617318480,
"state": {
  "color": "red"
},
"metadata": {
  "color": {
    "timestamp": 1617318480
  }
}
}
```

Schritt 3: Fehlerbehebung bei Interaktionen mit Geräteschatten

Wenn Sie die Schatten-Beispiel-App ausführen, können Probleme bei der Beobachtung von Interaktionen mit dem Geräteschattendienst auftreten.

Wenn das Programm erfolgreich ausgeführt wird und Sie zur Eingabe eines `desired`-Werts auffordert werden, sollten Sie in der Lage sein, die Geräteschatten-Interaktionen mithilfe des Schattendokuments und des MQTT-Testclients wie zuvor beschrieben zu beobachten. Wenn Sie die Interaktionen jedoch nicht sehen können, können Sie Folgendes überprüfen:

- Überprüfe den Namen des Dings und seinen Schatten im AWS IoT Konsole

  Wenn Sie die Meldungen im Schattendokument nicht sehen, überprüfen Sie den Befehl und stellen Sie sicher, dass er mit dem Namen des Objekts in der AWS IoT Konsole übereinstimmt. Sie können auch überprüfen, ob Sie einen klassischen Schatten haben, indem Sie Ihre Objekt-

Ressource und dann Schatten auswählen. Dieses Tutorial konzentriert sich hauptsächlich auf Interaktionen mit dem klassischen Schatten.

Sie können auch überprüfen, ob das von Ihnen verwendete Gerät mit dem Internet verbunden ist. Wählen Sie in der AWS IoT Konsole das Objekt aus, das Sie zuvor erstellt haben, und wählen Sie dann Interagieren aus. Auf der Seite mit den Objekt-Details solltest Sie hier eine Meldung sehen, die besagt: `This thing already appears to be connected.`

- Überprüfen der reservierten abonnierten MQTT-Themen

  Wenn Sie die Nachrichten nicht im MQTT-Testclient sehen, überprüfen Sie, ob die Themen, die Sie abonniert haben, richtig formatiert sind. MQTT Device Shadow-Themen haben das Format $aws/things/***thingname***/shadow/ und können dieses Format haben oder `delete` folgen `updateget`, je nachdem, welche Aktionen Sie für den Schatten ausführen möchten. In diesem Tutorial wird das Thema $aws/things/***thingname***/shadow/ # verwendet. Vergewissern Sie sich also, dass Sie es korrekt eingegeben haben, wenn Sie das Thema im Themenfilterbereich des Testclients abonnieren.

  Achten Sie bei der Eingabe des Themennamens darauf, dass dieser mit dem Namen des Objekts übereinstimmt, das Sie zuvor erstellt AWS IoT haben. *`thingname`* Sie können auch zusätzliche MQTT-Themen abonnieren, um zu sehen, ob ein Update erfolgreich durchgeführt wurde. Sie können beispielsweise das Thema $aws/things/***thingname***/shadow/update/rejected abonnieren, um eine Meldung zu erhalten, wenn eine Aktualisierungsanforderung fehlgeschlagen ist, sodass Sie Verbindungsprobleme debuggen können. Weitere Informationen zu den reservierten Themen finden Sie unter the section called “Schatten-Themen” und MQTT-Themen für Geräteschatten.

Schritt 4: Überprüfen der Ergebnisse und nächste Schritte

In diesem Kurs haben Sie gelernt, wie Sie:

- Die `shadow.py`-Beispiel-App verwenden, um die gewünschten Status anzugeben und den aktuellen Status des Schattens zu aktualisieren.
- Bearbeiten Sie das Schattendokument, um Delta-Ereignisse und die Reaktion der `shadow.py`-Beispiel-App auf diese Ereignisse zu beobachten.
- Verwenden Sie den MQTT-Testclient, um Schattenthemen zu abonnieren und Updates zu beobachten, wenn Sie das Beispielprogramm ausführen.

Nächste Schritte

Sie können zusätzliche reservierte MQTT-Themen abonnieren, um Aktualisierungen der Schattenanwendung zu beobachten. Wenn Sie beispielsweise nur das Thema $aws/things/***thingname***/shadow/update/accepted abonnieren, werden Ihnen nur die aktuellen Statusinformationen angezeigt, wenn ein Update erfolgreich durchgeführt wurde.

Sie können auch zusätzliche Schattenthemen abonnieren, um Probleme zu beheben oder mehr über die Geräteschatten-Interaktionen zu erfahren sowie mögliche Probleme mit den Geräteschatten-Interaktionen zu debuggen. Weitere Informationen erhalten Sie unter the section called “Schatten-Themen” und MQTT-Themen für Geräteschatten.

Sie können Ihre Anwendung auch erweitern, indem Sie benannte Schatten verwenden oder zusätzliche Hardware verwenden, die mit dem Raspberry Pi für die LEDs verbunden ist, und Statusänderungen anhand von Nachrichten beobachten, die vom Terminal gesendet werden.

Weitere Informationen zum Geräteschattendienst und zur Verwendung des Dienstes in Geräten, Apps und Diensten finden Sie unter AWS IoT Geräteschatten-Service, Verwenden von Schatten in Geräten und Verwenden von Schatten in Apps und Services.

Tutorial: Erstellen eines benutzerdefinierten Autorisierers für AWS IoT Core

Dieses Tutorial zeigt die Schritte zum Erstellen, Validieren und Verwenden einer benutzerdefinierten Authentifizierung mithilfe von AWS CLI. Optional können Sie mithilfe dieses Tutorials Postman verwenden, um mithilfe der HTTP Publish API Daten an AWS IoT Core zu senden.

In diesem Tutorial erfahren Sie, wie Sie eine exemplarische Lambda-Funktion erstellen, die die Autorisierungs- und Authentifizierungslogik implementiert, und einen benutzerdefinierten Autorisierer mithilfe des create-authorizer-Aufrufs mit aktivierter Tokensignatur. Der Authorizer wird dann mit dem validierttest-invoke-authorizer, und schließlich können Sie AWS IoT Core mithilfe der HTTP Publish API Daten an ein MQTT-Testthema senden. Die Beispielanfrage spezifiziert den Autorisierer, der aufgerufen werden soll, mithilfe des Headers `x-amz-customauthorizer-name` und übergibt token-key-name und `x-amz-customauthorizer-signature` in Anfrageheadern.

Was Sie in diesem Tutorial lernen werden:

- So erstellen Sie eine Lambda-Funktion als benutzerdefinierten Autorisierer-Handler
- Wie erstelle ich einen benutzerdefinierten Autorisierer mit aktivierter AWS CLI Tokensignierung
- So testen Sie Ihren benutzerdefinierten Autorisierer mit dem Befehl test-invoke-authorizer
- So veröffentlichen Sie ein MQTT-Thema mit Postman und validieren die Anfrage mit Ihrem benutzerdefinierten Autorisierer

Für dieses Tutorial brauchen Sie ungefähr 60 Minuten.

In diesem Tutorial führen Sie die folgenden Aktivitäten durch:

- Schritt 1: Erstellen einer Lambda-Funktion für Ihren benutzerdefinierten Autorisierer
- Schritt 2: Erstellen eines öffentlichen und eines privaten Schlüsselpaars für Ihren benutzerdefinierten Autorisierer
- Schritt 3: Erstellen Sie eine benutzerdefinierte Autorisierungsressource und deren Autorisierung
- Schritt 4: Testen des Autorisierers durch Aufrufen von test-invoke-authorizer
- Schritt 5: Testen der Veröffentlichung der MQTT-Nachricht mithilfe von Postman
- Schritt 6: Anzeigen von Nachrichten im MQTT-Testclient
- Schritt 7: Überprüfen der Ergebnisse und die nächsten Schritte
- Schritt 8: Bereinigen

Stellen Sie vor Beginn dieses Tutorials sicher, dass Sie über Folgendes verfügen:

- Einrichten AWS-Konto

  Sie benötigen Ihre eigene AWS-Konto AWS IoT Konsole, um dieses Tutorial abzuschließen.

  Das Konto, das Sie für dieses Tutorial verwenden, funktioniert am besten, wenn es mindestens die folgenden AWS -verwalteten Richtlinien umfasst:

  - `IAMFullAccess`
  - `AWSIoTFullAccess`
  - `AWSLambda_FullAccess`

  > **Important**
  >
  > Die in diesem Tutorial verwendeten IAM-Richtlinien sind permissiver, als Sie sie bei einer Produktionsimplementierung befolgen sollten. Stellen Sie in einer Produktionsumgebung sicher, dass Ihre Konto- und Ressourcenrichtlinien nur die erforderlichen Berechtigungen gewähren.
  > Wenn Sie IAM-Richtlinien für die Produktion erstellen, bestimmen Sie, welchen Zugriff Benutzer und Rollen benötigen. Generieren Sie dann die Richtlinien, die es ihnen ermöglichen, nur diese Aufgaben auszuführen.
  > Weitere Informationen finden Sie unter Bewährte IAM-Methoden.

- Installierte den AWS CLI

  Informationen zur Installation von finden Sie AWS CLI unter AWS CLI installieren. Für dieses Tutorial ist eine AWS CLI Version `aws-cli/2.1.3 Python/3.7.4 Darwin/18.7.0 exe/x86_64` oder höher erforderlich.

- OpenSSL-Tools

  Die Beispiele in diesem Tutorial verwenden LibreSSL 2.6.5. Sie können für dieses Tutorial auch die OpenSSL v1.1.1i-Tools verwenden.

- Überprüfte das AWS Lambda -Übersicht

  Wenn Sie Lambda noch nicht verwendet AWS Lambda haben, lesen AWS Lambda Sie das Kapitel Erste Schritte mit Lambda, um sich mit den Begriffen und Konzepten vertraut zu machen.

- Es wurde überprüft, wie Anfragen in Postman erstellt werden.

  Weitere Informationen finden Sie unter Erstellen von Anfragen.

- Benutzerdefinierte Autorisierer wurden aus dem vorherigen Tutorial entfernt.

  Sie AWS-Konto können nur eine begrenzte Anzahl von benutzerdefinierten Autorisierern gleichzeitig konfigurieren. Weitere Informationen zum Erstellen eines benutzerdefinierten Autorisierers finden Sie unter the section called “Schritt 8: Bereinigen”.

Schritt 1: Erstellen einer Lambda-Funktion für Ihren benutzerdefinierten Autorisierer

Bei der benutzerdefinierten Authentifizierung werden Autorisierungsressourcen AWS IoT Core verwendet, die Sie zur Authentifizierung und Autorisierung von Clients erstellen. Die Funktion, die Sie in diesem Abschnitt erstellen, authentifiziert und autorisiert Clients, wenn sie eine Verbindung zu Ressourcen herstellen und darauf zugreifen. AWS IoT Core AWS IoT

Die Lambda-Funktion bewirkt Folgendes:

- Wenn eine Anfrage von test-invoke-authorizer kommt, gibt sie eine IAM-Richtlinie mit einer `Deny`-Aktion zurück.
- Wenn eine Anfrage von Postman über HTTP kommt und der `actionToken` Parameter den Wert von hat`allow`, wird eine IAM-Richtlinie mit einer Aktion zurückgegeben. `Allow` Andernfalls wird eine IAM-Richtlinie mit einer Aktion `Deny` zurückgegeben.

So erstellen Sie die Lambda-Funktion für Ihren benutzerdefinierten Autorisierer

1. Öffnen Sie auf der Lambda-Konsole die Option Funktionen.
2. Wählen Sie Funktion erstellen.
3. Bestätigen Sie, dass Ohne Vorgabe erstellen ausgewählt ist.
4. Unter Grundlegende Informationen:
   a. Geben Sie unter Funktionsname **`custom-auth-function`** ein.
   b. Bestätigen Sie in Runtime 18.x Node.js
5. Wählen Sie Funktion erstellen.

   Lambda erstellt eine Node.js Funktion und eine Ausführungsrolle, die der Funktion die Berechtigung zum Hochladen von Protokollen gewähren. Die Lambda-Funktion übernimmt die Ausführungsrolle, wenn Sie Ihre Funktion aufrufen, und verwendet die Ausführungsrolle, um Anmeldeinformationen für das AWS SDK zu erstellen und Daten aus Ereignisquellen zu lesen.
6. Um den Code und die Konfiguration der Funktion im AWS Cloud9-Editor zu sehen, wählen Sie im Designer-Fenster custom-auth-function und dann im Navigationsbereich des Editors index.js.

   Für Skriptsprachen wie Lambda enthält eine Basisfunktion Node.js, die eine Erfolgsmeldung zurückgibt. Sie können den AWS Cloud9-Editor verwenden, um Ihre Funktion zu bearbeiten, solange Ihr Quellcode nicht größer als 3 MB ist.
7. Ersetzen Sie den Code index.js im Editor durch den folgenden Code:

```
// A simple Lambda function for an authorizer. It demonstrates
// How to parse a CLI and Http password to generate a response.

export const handler = async (event, context, callback) => {

    //Http parameter to initiate allow/deny request
    const HTTP_PARAM_NAME='actionToken';
    const ALLOW_ACTION = 'Allow';
    const DENY_ACTION = 'Deny';

    //Event data passed to Lambda function
    var event_str = JSON.stringify(event);
    console.log('Complete event :'+ event_str);

    //Read protocolData from the event json passed to Lambda function
    var protocolData = event.protocolData;
```

```
    console.log('protocolData value---> ' + protocolData);

    //Get the dynamic account ID from function's ARN to be used
    // as full resource for IAM policy
    var ACCOUNT_ID = context.invokedFunctionArn.split(":")[4];
    console.log("ACCOUNT_ID---"+ACCOUNT_ID);

    //Get the dynamic region from function's ARN to be used
    // as full resource for IAM policy
    var REGION = context.invokedFunctionArn.split(":")[3];
    console.log("REGION---"+REGION);

    //protocolData data will be undefined if testing is done via CLI.
    // This will help to test the set up.
    if (protocolData === undefined) {

        //If CLI testing, pass deny action as this is for testing purpose only.
        console.log('Using the test-invoke-authorizer cli for testing only');
        callback(null, generateAuthResponse(DENY_ACTION,ACCOUNT_ID,REGION));

    } else{

        //Http Testing from Postman
        //Get the query string from the request
        var queryString = event.protocolData.http.queryString;
        console.log('queryString values -- ' + queryString);
        /*         global URLSearchParams       */
        const params = new URLSearchParams(queryString);
        var action = params.get(HTTP_PARAM_NAME);

        if(action!=null && action.toLowerCase() === 'allow'){

            callback(null, generateAuthResponse(ALLOW_ACTION,ACCOUNT_ID,REGION));

        }else{

            callback(null, generateAuthResponse(DENY_ACTION,ACCOUNT_ID,REGION));

        }

    }

};
```

```
// Helper function to generate the authorization IAM response.
var generateAuthResponse = function(effect,ACCOUNT_ID,REGION) {

    var full_resource = "arn:aws:iot:"+ REGION + ":" + ACCOUNT_ID + ":*";
    console.log("full_resource---"+full_resource);

    var authResponse = {};
    authResponse.isAuthenticated = true;
    authResponse.principalId = 'principalId';

    var policyDocument = {};
    policyDocument.Version = '2012-10-17';
    policyDocument.Statement = [];
    var statement = {};
    statement.Action = 'iot:*';
    statement.Effect = effect;
    statement.Resource = full_resource;
    policyDocument.Statement[0] = statement;
    authResponse.policyDocuments = [policyDocument];
    authResponse.disconnectAfterInSeconds = 3600;
    authResponse.refreshAfterInSeconds = 600;

    console.log('custom auth policy function called from http');
    console.log('authResponse --> ' + JSON.stringify(authResponse));
    console.log(authResponse.policyDocuments[0]);

    return authResponse;
}
```

8. Wählen Sie Bereitstellen.
9. Nachdem Änderungen bereitgestellt über dem Editor angezeigt wird:
    a. Scrollen Sie zum Abschnitt Funktionsübersicht über dem Editor.
    b. Kopieren Sie die Funktion ARN und speichern Sie sie, um sie später in diesem Tutorial zu verwenden.
10. Testen Sie Ihre Funktion.
    a. Wählen Sie die Registerkarte Test.
    b. Wählen Sie unter Verwendung der Standard-Testeinstellungen Aufrufen.
    c. Wenn der Test erfolgreich war, öffnen Sie in den Ausführungsergebnissen die Ansicht Details. Sie sollten das Richtliniendokument sehen, das die Funktion zurückgegeben hat.

Wenn der Test fehlgeschlagen ist oder Sie kein Richtliniendokument finden, überprüfen Sie den Code, um die Fehler zu finden und zu korrigieren.

Schritt 2: Erstellen eines öffentlichen und eines privaten Schlüsselpaars für Ihren benutzerdefinierten Autorisierer

Ihr benutzerdefinierter Autorisierer benötigt für seine Authentifizierung einen öffentlichen und einen privaten Schlüssel. Die Befehle in diesem Abschnitt verwenden OpenSSL-Tools zum Erstellen dieses Schlüsselpaars.

So erstellen Sie das Paar aus öffentlichem und privatem Schlüssel für Ihren benutzerdefinierten Autorisierer

1. Erstellen Sie die Datei mit dem privaten Schlüssel.

   ```
   openssl genrsa -out private-key.pem 4096
   ```

2. Überprüfen Sie den Speicherort der Datei mit dem privaten Schlüssel, die Sie gerade erstellt haben.

   ```
   openssl rsa -check -in private-key.pem -noout
   ```

   Wenn der Befehl keine Fehler anzeigt, ist die Datei mit dem privaten Schlüssel gültig.

3. Erstellen Sie die Datei mit dem öffentlichen Schlüssel.

   ```
   openssl rsa -in private-key.pem -pubout -out public-key.pem
   ```

4. Überprüfen Sie die Datei mit dem öffentlichen Schlüssel.

   ```
   openssl pkey -inform PEM -pubin -in public-key.pem -noout
   ```

   Wenn der Befehl keine Fehler anzeigt, ist die Datei mit dem öffentlichen Schlüssel gültig.

Schritt 3: Erstellen Sie eine benutzerdefinierte Autorisierungsressource und deren Autorisierung

Der AWS IoT benutzerdefinierte Autorisierer ist die Ressource, die alle in den vorherigen Schritten erstellten Elemente miteinander verknüpft. In diesem Abschnitt erstellen Sie eine benutzerdefinierte Autorisierer-Ressource und erteilen ihr die Erlaubnis, die zuvor erstellte Lambda-Funktion auszuführen. Sie können eine benutzerdefinierte Autorisierungsressource mithilfe der AWS IoT Konsole, der oder der AWS CLI AWS API erstellen.

Für dieses Tutorial müssen Sie nur einen benutzerdefinierten Autorisierer erstellen. In diesem Abschnitt wird beschrieben, wie Sie mithilfe der AWS IoT Konsole und der erstellen AWS CLI, sodass Sie die für Sie am besten geeignete Methode verwenden können. Es gibt keinen Unterschied zwischen den mithilfe dieser beiden Methoden erstellen benutzerdefinierten Autorisierer-Ressourcen.

Erstellen Sie eine benutzerdefinierte Autorisierungsressource

Wählen Sie eine dieser Optionen, um Ihre benutzerdefinierte Autorisierer-Ressource zu erstellen.

- Erstellen Sie mithilfe der Konsole einen benutzerdefinierten Autorisierer AWS IoT
- So erstellen Sie einen benutzerdefinierten Autorisierer mithilfe der AWS CLI

So erstellen Sie einen benutzerdefinierten Autorisierer (Konsole)

1. Öffnen Sie die Seite „Benutzerdefinierter Autorisierer“ der AWS IoT Konsole und wählen Sie „Autorisierer erstellen“.
2. In Autorisierer erstellen:
    a. Geben Sie in das Feld Autorisierer-Name **`my-new-authorizer`** ein.
    b. Markieren Sie unter Autorisierer-Status die Option Aktiv.
    c. Wählen Sie in der Autorisierer-Funktion die Lambda-Funktion aus, die Sie zuvor erstellt haben.
    d. Unter Token-Validierung – optional:
        i. Aktivieren Sie die Token-Validierung.
        ii. Geben Sie **`tokenKeyName`** in das Feld Token-Schlüsselname ein.
        iii. Wählen Sie Schlüssel hinzufügen.
        iv. Geben Sie unter Schlüsselname **`FirstKey`** ein.

v. Geben Sie in das Feld Öffentlicher Schlüssel den Inhalt der Datei `public-key.pem` ein. Achten Sie darauf, die Zeilen aus der Datei mit `-----BEGIN PUBLIC KEY-----` und `-----END PUBLIC KEY-----` einzuschließen und dem Inhalt der Datei keine Zeilenvorschübe, Zeilenumbrüche oder andere Zeichen hinzuzufügen oder aus ihm zu entfernen. Die Zeichenfolge, die Sie eingeben, sollte in etwa wie im folgenden Beispiel aussehen.

```
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAvEBzOk4vhN+3LgslvEWt
sLCqNmt5Damas3bmiTRvq2gjRJ6KXGTGQChqArAJwL1a9dkS9+maaXC3vc6xzx9z
QPu/vQOe5tyzz1MsKdmtFGxMqQ3qjEXAMPLEOmqyUKPP5mff58k6ePSfXAnzBH0q
lg2HioefrpU5OSAnpuRAjYKofKjbc2Vrn6N2G7hV+IfTBvCElf0csalS/Rk4phD5
oa4Y0GHISRnevypg5C8n9Rrz91PWGqP6M/q5DNJJXjMyleG92hQgu1N696bn5Dw8
FhedszFa6b2x6xrItZFzewNQkPMLMFhNrQIIyvshtT/F1LVCS5+v8AQ8UGGDfZmv
QeqAMAF7WgagDMXcfgKSVU8yid2sIm56qsCLMvD2Sq8Lgzpey9N5ON1o1Cvldwvc
KrJJtgwW6hVqRGuShnownLpgG86M6neZ5sRMbVNZO8OzcobLngJ0Ibw9KkcUdklW
gvZ6HEJqBY2XE70iEXAMPLETPHzhqvK6Ei1HGxpHsXx6BNft582J1VpgYjXha8oa
/NN7l7Zbj/euAb41IVtmX8JrD9z613d1iM5L8HluJlUzn62Q+VeNV2tdA7MfPfMC
8btGYladFAnitThaz6+F0VSBJPu7pZQoLnqyEp5zLMtF+kFl2yOBmGAP0RBivRd9
JWBUCG0bqcLQPeQyjbXSOfUCAwEAAQ==
-----END PUBLIC KEY-----
```

3. Wählen Sie Autorisierer erstellen.
4. Wenn die benutzerdefinierte Autorisierer-Ressource erstellt wurde, wird die Liste der benutzerdefinierten Autorisierer angezeigt und Ihr neuer benutzerdefinierter Autorisierer sollte in der Liste angezeigt werden. Sie können dann mit dem nächsten Abschnitt fortfahren, um ihn zu testen.

   Wenn Sie einen Fehler sehen, überprüfen Sie den Fehler und versuchen Sie erneut, Ihren benutzerdefinierten Autorisierer zu erstellen, und überprüfen Sie die Einträge erneut. Beachten Sie, dass jeder Benutzer einen eindeutigen Namen haben muss.

Um einen benutzerdefinierten Autorisierer zu erstellen (AWS CLI)

1. Ersetzen Sie Ihre Werte durch `authorizer-function-arn` und `token-signing-public-keys`, und führen Sie dann den folgenden Befehl aus:

```
aws iot create-authorizer \
--authorizer-name "my-new-authorizer" \
--token-key-name "tokenKeyName" \
```

```
--status ACTIVE \
--no-signing-disabled \
--authorizer-function-arn "arn:aws:lambda:Region:57EXAMPLE833:function:custom-auth-function" \
--token-signing-public-keys FirstKey="-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAvEBzOk4vhN+3LgslvEWt
sLCqNmt5Damas3bmiTRvq2gjRJ6KXGTGQChqArAJwL1a9dkS9+maaXC3vc6xzx9z
QPu/vQOe5tyzz1MsKdmtFGxMqQ3qjEXAMPLEOmqyUKPP5mff58k6ePSfXAnzBH0q
lg2HioefrpU5OSAnpuRAjYKofKjbc2Vrn6N2G7hV+IfTBvCElf0csalS/Rk4phD5
oa4Y0GHISRnevypg5C8n9Rrz91PWGqP6M/q5DNJJXjMyleG92hQgu1N696bn5Dw8
FhedszFa6b2x6xrItZFzewNQkPMLMFhNrQIIyvshtT/F1LVCS5+v8AQ8UGGDfZmv
QeqAMAF7WgagDMXcfgKSVU8yid2sIm56qsCLMvD2Sq8Lgzpey9N5ON1o1Cvldwvc
KrJJtgwW6hVqRGuShnownLpgG86M6neZ5sRMbVNZO8OzcobLngJ0Ibw9KkcUdklW
gvZ6HEJqBY2XE70iEXAMPLETPHzhqvK6Ei1HGxpHsXx6BNft582J1VpgYjXha8oa
/NN7l7Zbj/euAb41IVtmX8JrD9z613d1iM5L8HluJlUzn62Q+VeNV2tdA7MfPfMC
8btGYladFAnitThaz6+F0VSBJPu7pZQoLnqyEp5zLMtF+kFl2yOBmGAP0RBivRd9
JWBUCG0bqcLQPeQyjbXSOfUCAwEAAQ==
-----END PUBLIC KEY-----"
```

Wobei Folgendes gilt:

- Der Wert `authorizer-function-arn` ist der Amazon-Ressourcenname (ARN) der Lambda-Funktion, die Sie für Ihren benutzerdefinierten Autorisierer erstellt haben.
- Der Wert `token-signing-public-keys` umfasst den Namen des Schlüssels, **`FirstKey`**, und den Inhalt der Datei `public-key.pem`. Achten Sie darauf, die Zeilen aus der Datei mit `-----BEGIN PUBLIC KEY-----` und `-----END PUBLIC KEY-----` einzuschließen und dem Inhalt der Datei keine Zeilenvorschübe, Zeilenumbrüche oder andere Zeichen hinzuzufügen oder aus ihm zu entfernen.

  Hinweis: Seien Sie vorsichtig bei der Eingabe des öffentlichen Schlüssels, da jede Wertänderung des öffentlichen Schlüssels diesen unbrauchbar macht.

2. Wenn der benutzerdefinierte Autorisierer erstellt wird, gibt der Befehl den Namen und den ARN der neuen Ressource zurück, z. B. folgendermaßen:

```
{
    "authorizerName": "my-new-authorizer",
    "authorizerArn": "arn:aws:iot:Region:57EXAMPLE833:authorizer/my-new-authorizer"
}
```

Speichern Sie den Wert `authorizerArn` zur Verwendung im nächsten Schritt.

Denken Sie daran, dass jeder Benutzer einen eindeutigen Namen haben muss.

Autorisieren der benutzerdefinierten Autorisierer-Ressource

In diesem Abschnitt erteilen Sie der benutzerdefinierten Autorisierer-Ressource, die Sie gerade erstellt haben, die Berechtigung zum Ausführen der Lambda-Funktion. Um die Berechtigung zu erteilen, können Sie den CLI-Befehl add-permission verwenden.

Erteilen Sie Ihrer Lambda-Funktion die Erlaubnis mit dem AWS CLI

1. Geben Sie nach der Eingabe Ihrer Werte den folgenden Befehl ein. Beachten Sie, dass der Wert `statement-id` eindeutig sein muss. Ersetzen Sie *`Id-1234`* durch einen anderen Wert, wenn Sie dieses Tutorial schon einmal ausgeführt haben oder wenn Sie einen Fehler `ResourceConflictException` angezeigt bekommen.

   ```
   aws lambda add-permission  \
   --function-name "custom-auth-function" \
   --principal "iot.amazonaws.com" \
   --action "lambda:InvokeFunction" \
   --statement-id "Id-1234" \
   --source-arn authorizerArn
   ```

2. Wenn der Befehl erfolgreich ist, gibt er eine Berechtigungsanweisung zurück, wie in diesem Beispiel. Sie können mit dem nächsten Abschnitt fortfahren, um den benutzerdefinierten Autorisierer zu testen.

   ```
   {
       "Statement": "{\"Sid\":\"Id-1234\",\"Effect\":\"Allow\",\"Principal
   \":{\"Service\":\"iot.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction
   \",\"Resource\":\"arn:aws:lambda:Region:57EXAMPLE833:function:custom-
   auth-function\",\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":
   \"arn:aws:lambda:Region:57EXAMPLE833:function:custom-auth-function\"}}}"
   }
   ```

   Wenn der Befehl nicht erfolgreich ist, wird ein Fehler zurückgegeben, wie in diesem Beispiel. Sie müssen den Fehler überprüfen und korrigieren, bevor Sie fortfahren können.

```
An error occurred (AccessDeniedException) when calling the AddPermission operation:
 User: arn:aws:iam::57EXAMPLE833:user/EXAMPLE-1 is not authorized to perform:
 lambda:AddPer
mission on resource: arn:aws:lambda:Region:57EXAMPLE833:function:custom-auth-
function
```

Schritt 4: Testen des Autorisierers durch Aufrufen von test-invoke-authorizer

Wenn alle Ressourcen definiert sind, rufen Sie in diesem Abschnitt test-invoke-authorizer aus der Befehlszeile auf, um den Autorisierungsdurchgang zu testen.

Beachten Sie, dass `protocolData` nicht definiert ist, wenn der Autorisierer aus der Befehlszeile aufgerufen wird, sodass der Autorisierer immer ein DENY-Dokument zurückgibt. Dieser Test bestätigt jedoch, dass Ihr benutzerdefinierter Autorisierer und Ihre Lambda-Funktion korrekt konfiguriert sind – auch wenn die Lambda-Funktion nicht vollständig getestet wird.

Um Ihren benutzerdefinierten Autorisierer und seine Lambda-Funktion zu testen, verwenden Sie den AWS CLI

1. Führen Sie in dem Verzeichnis, welches die `private-key.pem`-Datei, die Sie in einem vorherigen Schritt erstellt haben, enthält, den folgenden Befehl aus.

   ```
   echo -n "tokenKeyValue" | openssl dgst -sha256 -sign private-key.pem | openssl
    base64 -A
   ```

   Mit diesem Befehl wird eine Signaturzeichenfolge zur Verwendung im nächsten Schritt generiert. Die Signaturzeichenfolge sollte wie folgt aussehen:

   ```
   dBwykzlb+fo+JmSGdwoGr8dyC2qB/IyLefJJr+rbCvmu9Jl4KHAA9DG+V
   +MMWu09YSA86+64Y3Gt4tOykpZqn9mn
   VB1wyxp+0bDZh8hmqUAUH3fwi3fPjBvCa4cwNuLQNqBZzbCvsluv7i2IMjEg
   +CPY0zrWt1jr9BikgGPDxWkjaeeh
   bQHHTo357TegKs9pP30Uf4TrxypNmFswA5k7QIc01n4bIyRTm90OyZ94R4bdJsHNig1JePgnuOBvMGCEFE09jGjj
   szEHfgAUAQIWXiVGQj16BU1xKpTGSiTAwheLKUjITOEXAMPLECK3aHKYKY
   +d1vTvdthKtYHBq8MjhzJ0kggbt29V
   QJCb8RilN/P5+vcVniSXWPplyB5jkYs9UvG08REoy64AtizfUhvSul/r/F3VV8ITtQp3aXiUtcspACi6ca
   +tsDuX
   f3LzCwQQF/YSUy02u5XkWn
   +sto6KCkpNlkD0wU8gl3+kOzxrthnQ8gEajd5Iylx230iqcXo3osjPha7JDyWM5o+K
   ```

```
EWckTe91I1mokDr5sJ4JXixvnJTVSx1li49IalW4en1DAkc1a0s2U2UNm236EXAMPLELotyh7h
+flFeloZlAWQFH
xRlXsPqiVKS1ZIUClaZWprh/orDJplpiWfBgBIOgokJIDGP9gwhXIIk7zWrGmWpMK9o=
```

Kopieren Sie diese Signaturzeichenfolge zur Verwendung im nächsten Schritt. Achten Sie darauf, keine zusätzlichen Zeichen einzufügen oder Zeichen wegzulassen.

2. Ersetzen Sie in diesem Befehl den Wert `token-signature` durch die Signaturzeichenfolge aus dem vorherigen Schritt und führen Sie diesen Befehl aus, um Ihren Autorisierer zu testen.

```
aws iot test-invoke-authorizer \
--authorizer-name my-new-authorizer \
--token tokenKeyValue \
--token-signature dBwykzlb+fo+JmSGdwoGr8dyC2qB/IyLefJJr
+rbCvmu9Jl4KHAA9DG+V+MMWu09YSA86+64Y3Gt4tOykpZqn9mnVB1wyxp
+0bDZh8hmqUAUH3fwi3fPjBvCa4cwNuLQNqBZzbCvsluv7i2IMjEg
+CPY0zrWt1jr9BikgGPDxWkjaeehbQHHTo357TegKs9pP30Uf4TrxypNmFswA5k7QIc01n4bIyRTm90OyZ94R4bdJsh
+d1vTvdthKtYHBq8MjhzJ0kggbt29VQJCb8RilN/
P5+vcVniSXWPplyB5jkYs9UvG08REoy64AtizfUhvSul/r/F3VV8ITtQp3aXiUtcspACi6ca
+tsDuXf3LzCwQQF/YSUy02u5XkWn
+sto6KCkpNlkD0wU8gl3+kOzxrthnQ8gEajd5Iylx230iqcXo3osjPha7JDyWM5o
+KEWckTe91I1mokDr5sJ4JXixvnJTVSx1li49IalW4en1DAkc1a0s2U2UNm236EXAMPLELotyh7h
+flFeloZlAWQFHxRlXsPqiVKS1ZIUClaZWprh/orDJplpiWfBgBIOgokJIDGP9gwhXIIk7zWrGmWpMK9o=
```

Wenn der Befehl erfolgreich ist, gibt er die von Ihrer benutzerdefinierten Autorisierungsfunktion generierten Informationen zurück, wie in diesem Beispiel.

```
{
    "isAuthenticated": true,
    "principalId": "principalId",
    "policyDocuments": [
        "{\"Version\":\"2012-10-17\",       \"Statement\":[{\"Action\":\"iot:*\",
\"Effect\":\"Deny\",\"Resource\":\"arn:aws:iot:Region:57EXAMPLE833:*\"}]}"
    ],
    "refreshAfterInSeconds": 600,
    "disconnectAfterInSeconds": 3600
}
```

Wenn bei diesem Befehl ein Fehler zurückgegeben wird, überprüfen Sie den Fehler und überprüfen Sie erneut die Befehle, die Sie in diesem Abschnitt verwendet haben.

Schritt 5: Testen der Veröffentlichung der MQTT-Nachricht mithilfe von Postman

1. Um Ihren Gerätedaten-Endpunkt über die Befehlszeile abzurufen, rufen Sie describe-endpoint auf, wie hier dargestellt

   ```
   aws iot describe-endpoint --output text --endpoint-type iot:Data-ATS
   ```

   Speichern Sie diese Adresse, um sie *device_data_endpoint_address* in einem späteren Schritt als zu verwenden.

2. Öffnen Sie ein neues Postman-Fenster, und erstellen Sie eine neue HTTP-POST-Anfrage.

   a. Öffnen Sie auf Ihrem Computer die Postman-App.

   b. Wählen Sie in Postman im Menü Datei die Option Neu….

   c. Wählen Sie im Dialogfeld Neu die Option Anfrage.

   d. Unter Anfrage speichern:

      i. Geben Sie unter Name der Anfrage **`Custom authorizer test request`** ein.

      ii. Wählen Sie unter Sammlung oder Ordner zum Speichern auswählen: eine Sammlung aus bzw. erstellen Sie eine, in der diese Anfrage gespeichert werden soll.

      iii. Wählen Sie Speichern unter ***collection_name***.

3. Erstellen Sie die POST-Anfrage, um Ihren benutzerdefinierten Autorisierer zu testen.

   a. Wählen Sie in der Auswahl der Anfragemethode neben dem URL-Feld die Option POST.

   b. Erstellen Sie im URL-Feld die URL für Ihre Anfrage, indem Sie die folgende URL mit dem Befehl *device_data_endpoint_address* from the describe-endpoint in einem vorherigen Schritt verwenden.

      ```
      https://device_data_endpoint_address:443/topics/test/cust-auth/topic?
      qos=0&actionToken=allow
      ```

      Beachten Sie, dass diese URL den Abfrageparameter `actionToken=allow` enthält, der Ihre Lambda-Funktion anweist, ein Richtliniendokument zurückzugeben, das den Zugriff auf das AWS IoT ermöglicht. Nachdem Sie die URL eingegeben haben, werden die Abfrageparameter auch auf der Registerkarte Params von Postman angezeigt.

   c. Wählen Sie auf der Registerkarte Auth im Feld Typ die Option Keine Auth.

   d. Auf der Registerkarte Header:

i. Wenn ein Host-Schlüssel aktiviert ist, deaktivieren Sie diesen.

ii. Fügen Sie diese neuen Header am Ende der Header-Liste hinzu, und bestätigen Sie, dass sie aktiviert sind. Ersetzen Sie den **`Host`** Wert durch Ihren *`device_data_endpoint_address`* und den **`x-amz-customauthorizer-signature`** Wert durch die Signaturzeichenfolge, die Sie mit dem test-invoke-authorize Befehl im vorherigen Abschnitt verwendet haben.

Key (Schlüssel)	Value (Wert)
`x-amz-customauthorizer-name`	**`my-new-authorizer`**
`Host`	*`device_data_endpoint_address`*
`tokenKeyName`	**`tokenKeyValue`**

Key (Schlüssel)	Value (Wert)
x-amz-customauthorizer-signature	*dBwykzlb+fo+JmSGdwoGr8dyC2qB/IyLefJJr+rbCvmu9Jl4KHAA9DG+V+MMWu09YSA86+64Y3Gt4tOykpZqn9mnVB1wyxp+0bDZh8hmqUAUH3fwi3fPjBvCa4cwNuLQNqBZzbCvsluv7i2IMjEg+CPY0zrWt1jr9BikgGPDxWkjaeehbQHHTo357TegKs9pP30Uf4TrxypNmFswA5k7QIc01n4bIyRTm90OyZ94R4bdJsHNig1JePgnuOBvMGCEFE09jGjjszEHfgAUAQIWXiVGQj16BU1xKpTGSiTAwheLKUjITOEXAMPLECK3aHKYKY+d1vTvdthKtYHBq8MjhzJ0kggbt29VQJCb8RilN/P5+vcVniSXWPplyB5jkYs9UvG08REoy64AtizfUhvSul/r/F3VV8ITtQp3aXiUtcspACi6ca+tsDuXf3LzCwQQF/YSUy02u5XkWn+sto6KCkpNlkD0wU8gl3+kOzxrthnQ8gEajd5Iylx230iqcXo3osjPha7JDyWM5o+KEWckTe91I1mokDr5sJ4JXixvnJTVSx1li49IalW4en1DAkc1a0s2U2UNm236EXAMPLELotyh7h+flFeloZlAWQFHxRlXsPqiVKS1ZIUClaZWprh/orDJplpiWfBgBIOgokJIDGP9gwhXIIk7zWrGmWpMK9o=*

e. Auf der Registerkarte „Text“:

i. Wählen Sie im Optionsfeld für das Datenformat die Option Raw.

ii. Wählen Sie in der Datentypliste JavaScript.

iii. Geben Sie in das Textfeld diese JSON-Nachrichtennutzlast für Ihre Testnachricht ein:

```
{
    "data_mode": "test",
```

```
    "vibration": 200,
    "temperature": 40
}
```

4. Wählen Sie Senden aus, um die Anfrage zu senden.

   Wenn die Anfrage erfolgreich war, wird Folgendes zurückgegeben:

   ```
   {
       "message": "OK",
       "traceId": "ff35c33f-409a-ea90-b06f-fbEXAMPLE25c"
   }
   ```

   Die erfolgreiche Antwort zeigt an, dass Ihr benutzerdefinierter Autorisierer die Verbindung zu Broker in zugelassen hat AWS IoT und dass die Testnachricht an Broker In AWS IoT Core zugestellt wurde.

   Wenn ein Fehler zurückgegeben wird, überprüfen Sie die Fehlermeldung, die*`device_data_endpoint_address`*, die Signaturzeichenfolge und die anderen Header-Werte.

Bewahren Sie diese Anfrage in Postman auf, um Sie im nächsten Abschnitt zu verwenden.

Schritt 6: Anzeigen von Nachrichten im MQTT-Testclient

Im vorherigen Schritt haben Sie mithilfe AWS IoT von Postman simulierte Gerätenachrichten an Sie gesendet. Die erfolgreiche Antwort wies darauf hin, dass Ihr benutzerdefinierter Autorisierer die Verbindung zum AWS IoT gestattet hat und dass die Testnachricht an den Broker in AWS IoT Core zugestellt wurde. In diesem Abschnitt verwenden Sie den MQTT-Testclient in der AWS IoT Konsole, um den Nachrichteninhalt dieser Nachricht so anzuzeigen, wie dies bei anderen Geräten und Diensten der Fall sein könnte.

So zeigen Sie die von Ihrem benutzerdefinierten Autorisierer autorisierten Testnachrichten an

1. Öffnen Sie in der AWS IoT Konsole den MQTT-Testclient.
2. Geben Sie auf der Registerkarte Thema abonnieren im Themenfilter **`test/cust-auth/topic`** ein – das Nachrichtenthema aus dem vorherigen Abschnitt, das im Postman-Beispiel verwendet wurde.
3. Wählen Sie Abonnieren.

Lassen Sie dieses Fenster für den nächsten Schritt geöffnet.

4. Wählen Sie in Postman in der Anfrage, die Sie für den vorherigen Abschnitt erstellt haben, die Option Senden.

   Überprüfen Sie die Antwort, um sicherzustellen, dass sie erfolgreich war. Falls nicht, beheben Sie den Fehler, wie im vorherigen Abschnitt beschrieben.

5. Im MQTT-Testclient sollten Sie einen neuen Eintrag sehen, der das Nachrichtenthema und, falls erweitert, die Nachrichten-Nutzlast aus der Anfrage anzeigt, die Sie von Postman gesendet haben.

   Wenn Sie Ihre Nachrichten nicht im MQTT-Testclient sehen, sollten Sie Folgendes überprüfen:

   - Stellen Sie sicher, dass Ihre Postman-Anfrage erfolgreich zurückgegeben wurde. Wenn die Verbindung AWS IoT abgelehnt wird und ein Fehler zurückgegeben wird, wird die Nachricht in der Anfrage nicht an den Message Broker weitergeleitet.
   - Stellen Sie sicher, dass die AWS-Konto und, die zum Öffnen der AWS IoT Konsole AWS-Region verwendet werden, mit der URL übereinstimmen, die Sie in der Postman-URL verwenden.
   - Stellen Sie sicher, dass Sie den entsprechenden Endpunkt für den benutzerdefinierten Autorisierer verwenden. Der Standard-IoT-Endpunkt unterstützt möglicherweise nicht die Verwendung benutzerdefinierter Autorisierer mit Lambda-Funktionen. Stattdessen können Sie Domänenkonfigurationen verwenden, um einen neuen Endpunkt zu definieren und diesen Endpunkt dann für den benutzerdefinierten Autorisierer anzugeben.
   - Stellen Sie sicher, dass Sie das Thema korrekt in den MQTT-Testclient eingegeben haben. Beim Themenfilter wird die Groß-/Kleinschreibung berücksichtigt. Im Zweifelsfall können Sie auch das **#** Thema abonnieren, das alle MQTT-Nachrichten abonniert, die den Message Broker passieren AWS-Konto und zum Öffnen der Konsole AWS-Region verwendet werden. AWS IoT

Schritt 7: Überprüfen der Ergebnisse und die nächsten Schritte

In diesem Tutorial:

- Haben Sie eine Lambda-Funktion als benutzerdefinierten Autorisierer-Handler erstellt.
- Haben Sie einen benutzerdefinierten Autorisierer mit aktivierter Token-Signatur erstellt.
- Haben Sie Ihren benutzerdefinierten Autorisierer mit dem Befehl test-invoke-authorizer getestet.

- Haben Sie ein MQTT-Thema mit Postman veröffentlicht und die Anfrage mit Ihrem benutzerdefinierten Autorisierer validiert.
- Haben Sie den MQTT-Testclient verwendet, um die von Ihrem Postman-Test gesendeten Nachrichten anzusehen.

Nächste Schritte

Nachdem Sie einige Nachrichten von Postman gesendet haben, um zu überprüfen, ob der benutzerdefinierte Autorisierer funktioniert, versuchen Sie zu experimentieren, um festzustellen, wie sich Änderungen an verschiedenen Aspekten dieses Tutorials auf die Ergebnisse auswirken. Hier sind einige Beispiele, die Ihnen den Einstieg erleichtern sollen.

- Ändern Sie die Signaturzeichenfolge so, dass sie nicht mehr gültig ist, um zu sehen, wie unbefugte Verbindungsversuche behandelt werden. Sie sollten eine Fehlerantwort wie diese erhalten und die Meldung sollte nicht auf dem MQTT-Testclient erscheinen.

```
{
    "message": "Forbidden",
    "traceId": "15969756-a4a4-917c-b47a-5433e25b1356"
}
```

- Weitere Informationen zum Auffinden von Fehlern, die beim Entwickeln und Verwenden von AWS IoT Regeln auftreten können, finden Sie unter. Überwachen AWS IoT

Schritt 8: Bereinigen

Wenn Sie dieses Tutorial wiederholen möchten, müssen Sie möglicherweise einige Ihrer benutzerdefinierten Autorisierer entfernen. Sie AWS-Konto können nur eine begrenzte Anzahl von benutzerdefinierten Autorisierern gleichzeitig konfigurieren. Sie können einen erhalten, `LimitExceededException` wenn Sie versuchen, einen neuen hinzuzufügen, ohne einen vorhandenen benutzerdefinierten Autorisierer zu entfernen.

So entfernen Sie einen benutzerdefinierten Autorisierer (Konsole)

1. Öffnen Sie die Seite Benutzerdefinierter Autorisierer der AWS IoT Konsole und suchen Sie in der Liste der benutzerdefinierten Autorisierer nach dem benutzerdefinierten Autorisierer, den Sie entfernen möchten.

2. Öffnen Sie die Seite mit den Details des benutzerdefinierten Autorisierers, und wählen Sie aus dem Menü Aktionen die Option Bearbeiten.
3. Heben Sie die Auswahl von Autorisierer aktivieren auf, und wählen Sie dann Aktualisieren.

   Während ein benutzerdefinierter Autorisierer aktiv ist, können Sie ihn nicht löschen.
4. Öffnen Sie auf der Seite mit den Details des benutzerdefinierten Autorisierers das Menü Aktionen, und wählen Sie Löschen.

Um einen benutzerdefinierten Autorisierer zu entfernen (AWS CLI)

1. Listen Sie die benutzerdefinierten Autorisierer auf, die Sie installiert haben, und suchen Sie nach dem Namen des benutzerdefinierten Autorisierers, den Sie löschen möchten.

   ```
   aws iot list-authorizers
   ```
2. Legen Sie den benutzerdefinierten Autorisierer auf `inactive` fest, indem Sie diesen Befehl ausführen, nachdem Sie *`Custom_Auth_Name`* durch `authorizerName` des benutzerdefinierten Autorisierers, der gelöscht werden soll, ersetzt haben.

   ```
   aws iot update-authorizer --status INACTIVE --authorizer-name Custom_Auth_Name
   ```
3. Löschen Sie den benutzerdefinierten Autorisierer, indem Sie diesen Befehl ausführen, nachdem Sie *`Custom_Auth_Name`* durch `authorizerName` des benutzerdefinierten Autorisierers, der gelöscht werden soll, ersetzt haben.

   ```
   aws iot delete-authorizer --authorizer-name Custom_Auth_Name
   ```

Tutorial: Überwachung der Bodenfeuchte mit AWS IoT und Raspberry Pi

Dieses Tutorial zeigt Ihnen, wie Sie einen Raspberry Pi, einen Feuchtigkeitssensor, verwenden und AWS IoT den Bodenfeuchtigkeitsgehalt einer Zimmerpflanze oder eines Gartens überwachen. Der Raspberry Pi führt einen Code aus, der den Feuchtigkeitsgehalt und die Temperatur vom Sensor liest und die Daten dann an den Sensor sendet AWS IoT. Sie erstellen eine Regel AWS IoT , die eine E-Mail an eine Adresse sendet, die ein Amazon SNS-Thema abonniert hat, wenn der Feuchtigkeitsgehalt unter einen Schwellenwert fällt.

> **Note**
>
> Dieses Tutorial ist möglicherweise nicht aktuell. Einige Verweise wurden möglicherweise seit der ursprünglichen Veröffentlichung dieses Themas ersetzt.

Inhalt

- Voraussetzungen
- Einrichtung AWS IoT
  - Schritt 1: Erstellen Sie den AWS IoT policy
  - Schritt 2: Erstellen Sie den AWS IoT Ding, Zertifikat und privater Schlüssel
  - Schritt 3: Erstellen eines Amazon-SNS-Themas und -Abonnements
  - Schritt 4: Erstellen Sie eine AWS IoT Regel zum Senden einer E-Mail
- Einrichten des Raspberry Pi und des Feuchtesensors

Voraussetzungen

Zum Durchführen dieses Tutorials benötigen Sie Folgendes:

- Ein. AWS-Konto
- Ein IAM-Benutzer mit Administratorberechtigungen.
- Ein Entwicklungscomputer mit Windows, macOS, Linux oder Unix für den Zugriff auf die AWS IoT -Konsole.
- Ein Raspberry Pi 3B oder 4B mit dem neuesten Raspberry Pi OS. Anweisungen zur Installation finden Sie unter Installieren eines Betriebssystems auf der Raspberry Pi-Website.
- Ein Monitor, eine Tastatur, eine Maus und eine Wi-Fi Netzwerk- oder Ethernet-Verbindung für Ihren Raspberry Pi.
- Ein Pi-compatible Himbeer-Feuchtigkeitssensor. Bei dem in diesem Tutorial verwendeten Sensor handelt es sich um einen Adafruit STEMMA I2C Capacitive Moisture Sensor mit einer JST 4-adrigen Kabelbuchsenleiste.

Einrichtung AWS IoT

Um dieses Tutorial abzuschließen, müssen Sie die folgenden Ressourcen erstellen. Um ein Gerät mit dem zu verbinden AWS IoT, erstellen Sie ein IoT-Ding, ein Gerätezertifikat und eine AWS IoT Richtlinie.

- AWS IoT Irgendein Ding.

  Ein Objekt stellt ein physisches Gerät (in diesem Fall Ihr Rasberry Pi) dar und enthält statische Metadaten über das Gerät.

- Ein Gerätezertifikat.

  Alle Geräte müssen über ein Gerätezertifikat verfügen, um eine Verbindung herzustellen und sich mit AWS IoT zu authentifizieren.

- Eine AWS IoT Richtlinie.

  Jedem Gerätezertifikat sind eine oder mehrere AWS IoT Richtlinien zugeordnet. Diese Richtlinien legen fest, auf welche AWS IoT Ressourcen das Gerät zugreifen kann.

- Ein AWS IoT Root-CA-Zertifikat.

  Geräte und andere Clients verwenden ein AWS IoT Root-CA-Zertifikat, um den AWS IoT Server zu authentifizieren, mit dem sie kommunizieren. Weitere Informationen finden Sie unter Serverauthentifizierung.

- Eine AWS IoT Regel.

  Eine Regel enthält eine Abfrage und mindestens eine Regelaktion. Die Abfrage extrahiert Daten aus Gerätenachrichten, um zu bestimmen, ob die Nachrichtendaten verarbeitet werden sollen. Die Regelaktion gibt an, was zu tun ist, wenn die Daten mit der Abfrage übereinstimmen.

- Ein Amazon SNS-Thema und ein Themenabonnement.

  Die Regel überwacht die Feuchtigkeitsdaten über Ihren Raspberry Pi. Wenn der Wert unter einem Schwellenwert liegt, wird eine Nachricht an das Amazon-SNS-Thema gesendet. Amazon SNS sendet diese Nachricht an alle E-Mail-Adressen, die das Thema abonniert haben.

Schritt 1: Erstellen Sie den AWS IoT policy

Erstellen Sie eine AWS IoT Richtlinie, die es Ihrem Raspberry Pi ermöglicht, eine Verbindung herzustellen und Nachrichten an sie zu senden AWS IoT.

1. Wenn in der AWS IoT -Konsole die Schaltfläche Erste Schritte erscheint, klicken Sie darauf. Erweitern Sie andernfalls im Navigationsbereich Sicherheit und wählen Sie dann Richtlinien aus.
2. Wenn das Dialogfeld You don‘t have any policies yet (Sie haben noch keine Richtlinien) angezeigt wird, wählen Sie Create a policy (Richtlinie erstellen) aus. Wählen Sie andernfalls Erstellen.
3. Geben Sie einen Namen für die AWS IoT Richtlinie ein (z. B.**`MoistureSensorPolicy`**).
4. Ersetzen Sie im Abschnitt Add statements (Anweisungen hinzufügen) die vorhandene Richtlinie durch das folgende JSON-Objekt. Ersetzen Sie *`region`* und *`account`* durch Ihre AWS-Region AWS-Konto UND-Nummer.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/RaspberryPi"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Publish",
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/update",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/delete",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/get"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": [
```

```
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/update/accepted",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/delete/accepted",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/get/accepted",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/update/rejected",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
RaspberryPi/shadow/delete/rejected"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
RaspberryPi/shadow/update/accepted",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
RaspberryPi/shadow/delete/accepted",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
RaspberryPi/shadow/get/accepted",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
RaspberryPi/shadow/update/rejected",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
RaspberryPi/shadow/delete/rejected"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
                "iot:UpdateThingShadow",
                "iot:DeleteThingShadow"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:thing/RaspberryPi"
        }
    ]
}
```

5. Wählen Sie Erstellen aus.

Schritt 2: Erstellen Sie den AWS IoT Ding, Zertifikat und privater Schlüssel

Erstellen Sie in der AWS IoT Registrierung ein Ding, das Ihren Raspberry Pi repräsentiert.

1. Wählen Sie in der AWS IoT -Konsole im Navigationsbereich Manage (Verwalten) und dann Things (Objekte).
2. Wenn das Dialogfeld You don‘t have any things yet (Sie haben noch keine Objekte) angezeigt wird, wählen Sie Register a thing (Objekt registrieren) aus. Wählen Sie andernfalls Erstellen.
3. Wählen Sie auf der Seite AWS IoT Dinge erstellen die Option Eine einzelne Sache erstellen aus.
4. Geben Sie auf der Seite Add your device to the device registry (Ihr Gerät zur Geräteregistrierung hinzufügen) einen Namen für Ihr IoT-Objekt ein (z. B. **`RaspberryPi`**) und wählen Sie dann Next (Weiter). Sie können den Namen eines Objekts nicht mehr ändern, nachdem Sie es erstellt haben. Um den Namen eines Objekts zu ändern, müssen Sie ein neues Objekt erstellen, diesem den neuen Namen geben und dann das alte Objekt löschen.
5. Wählen Sie auf der Seite Add a certificate for your thing (Fügen Sie ein Zertifikat für Ihr Objekt hinzu.) die Option Create certificate (Zertifikat erstellen).
6. Klicken Sie auf die Download-Links, um das Zertifikat, den privaten Schlüssel und das CA-Stammzertifikat herunterzuladen.

   > **Important**
   >
   > Dies ist das einzige Mal, dass Sie Ihr Zertifikat und Ihren privaten Schlüssel herunterladen können.

7. Wählen Sie Aktivieren aus, um Ihr Zertifikat zu aktivieren. Das Zertifikat muss aktiv sein, damit ein Gerät eine Verbindung mit AWS IoT herstellen kann.
8. Wählen Sie Attach a policy (Richtlinie anfügen) aus.
9. Wählen MoistureSensorPolicy Sie unter Richtlinie für Ihr Ding hinzufügen die Option und dann Ding registrieren aus.

Schritt 3: Erstellen eines Amazon-SNS-Themas und -Abonnements

Erstellen eines Amazon-SNS-Themas und -Abonnements.

1. Klicken Sie in der AWS SNS-Konsole im Navigationsbereich auf Themen und wählen Sie dann Thema erstellen aus.

2. Wählen Sie als Typ „Standard“ und geben Sie einen Namen für das Thema ein (z. B.**`MoistureSensorTopic`**).
3. Geben Sie einen Anzeigenamen für das Thema ein (z. B. **`Moisture Sensor Topic`**). Dies ist der Name, der für Ihr Thema in der Amazon SNS -Konsole angezeigt wird.
4. Wählen Sie Thema erstellen aus.
5. Wählen Sie auf der Seite mit den Details des Amazon SNS-Themas die Option Create subscription (Abonnement erstellen) aus.
6. Wählen Sie unter Protocol (Protokoll) die Option Email (E-Mail) aus.
7. Geben Sie unter Endpunkt Ihre E-Mail-Adresse ein.
8. Wählen Sie Create subscription (Abonnement erstellen) aus.
9. Öffnen Sie Ihren E-Mail-Client und suchen Sie nach einer Nachricht mit dem Betreff **`MoistureSensorTopic`**. Öffnen Sie die E-Mail und klicken Sie auf den Link Confirm subscription (Abonnement bestätigen).

> **Important**
>
> Sie erhalten keine E-Mail-Benachrichtigungen von diesem Amazon SNS-Thema, bis Sie das Abonnement bestätigen.

Sie sollten eine E-Mail-Nachricht mit dem von Ihnen eingegebenen Text erhalten.

Schritt 4: Erstellen Sie eine AWS IoT Regel zum Senden einer E-Mail

Eine AWS IoT Regel definiert eine Abfrage und eine oder mehrere Aktionen, die ausgeführt werden, wenn eine Nachricht von einem Gerät empfangen wird. Die AWS IoT Regel-Engine wartet auf Nachrichten, die von Geräten gesendet werden, und bestimmt anhand der Daten in den Nachrichten, ob Maßnahmen ergriffen werden sollten. Weitere Informationen finden Sie unter Regeln für AWS IoT.

In diesem Tutorial veröffentlicht Ihr Raspberry Pi Nachrichten auf `aws/things/RaspberryPi/shadow/update`. Dies ist ein internes MQTT-Thema, das von Geräten und dem Thing Shadow-Service verwendet wird. Der Raspberry Pi veröffentlicht Nachrichten in der folgenden Form:

```
{
    "reported": {
        "moisture" : moisture-reading,
        "temp" : temperature-reading
```

```
    }
}
```

Sie erstellen eine Abfrage, die die Feuchtigkeits- und Temperaturdaten aus der eingehenden Nachricht extrahiert. Sie erstellen auch eine Amazon SNS-Aktion, die die Daten übernimmt und an Abonnenten des Amazon SNS-Themas sendet, wenn der Feuchtigkeitswert unter einem Schwellenwert liegt.

Erstellen Sie eine Amazon SNS-Regel

1. Wählen Sie in der AWS IoT Konsole Nachrichtenrouting und dann Regeln aus. Wenn das Dialogfeld You don't have any rules yet (Sie haben noch keine Regeln) angezeigt wird, wählen Sie Create a rule (Regel erstellen) aus. Wählen Sie andernfalls Regel erstellen.
2. Geben Sie auf der Seite mit den Regeleigenschaften einen Regelnamen wie **`MoistureSensorRule`** ein und geben Sie eine kurze Regelbeschreibung ein, z.B. **`Sends an alert when soil moisture level readings are too low`**.
3. Wählen Sie Weiter und konfigurieren Sie Ihre SQL-Anweisung. Wählen Sie die SQL-Version als 23.03.2016 und geben Sie die folgende AWS IoT SQL-Abfrageanweisung ein:

   ```
   SELECT * FROM '$aws/things/RaspberryPi/shadow/update/accepted' WHERE
    state.reported.moisture < 400
   ```

   Diese Anweisung löst die Regelaktion aus, wenn der `moisture`-Lesevorgang kleiner als `400` ist.

   > **Note**
   >
   > Möglicherweise müssen Sie einen anderen Wert verwenden. Nachdem Sie den Code auf Ihrem Raspberry Pi ausgeführt haben, können Sie die Werte sehen, die Sie von Ihrem Sensor erhalten, indem Sie den Sensor berühren, ihn in Wasser platzieren oder ihn in einem Übertopf platzieren.

4. Wählen Sie Weiter und hängen Sie Regelaktionen an. Wählen Sie für Aktion 1 Einfacher Benachrichtigungsservice aus. Die Beschreibung für diese Regelaktion lautet Eine Nachricht als SNS-Push-Benachrichtigung senden.
5. Wählen Sie für das SNS-Thema das Thema aus, in dem Sie erstellt haben Schritt 3: Erstellen eines Amazon-SNS-Themas und -Abonnements MoistureSensorTopic, und belassen Sie das

Nachrichtenformat bei RAW. Wählen Sie für IAM Role (IAM-Rolle) die Option Create a New Role (Neue Rolle erstellen) aus. Geben Sie einen Namen für die Rolle ein, beispielsweise **`LowMoistureTopicRole`**, und wählen Sie dann Rolle erstellen aus.

6. Wählen Sie Weiter aus, um die Regel zu überprüfen, und klicken Sie dann auf Erstellen, um die Regel zu erstellen.

Einrichten des Raspberry Pi und des Feuchtesensors

Stecken Sie Ihre microSD-Karte in den Raspberry Pi, schließen Sie Monitor, Tastatur, Maus und, falls Sie es nicht verwenden Wi-Fi, das Ethernet-Kabel an. Schließen Sie das Stromkabel noch nicht an.

Schließen Sie das JST-Überbrückungskabel an den Feuchtigkeitssensor an. Die andere Seite des Kabels hat vier Drähte:

- Grün: I2C SCL
- Weiß: I2C SDA
- Rot: Stromversorgung (3,5 V)
- Schwarz: Erdung

Halten Sie den Raspberry Pi mit der Ethernet-Buchse auf der rechten Seite. In dieser Ausrichtung befinden sich oben zwei Reihen von GPIO-Stiften. Verbinden Sie die Drähte vom Feuchtigkeitssensor in der folgenden Reihenfolge mit der unteren Reihe der Stifte. Beginnen Sie mit dem Stift ganz links. Schließen Sie rot (Strom), weiß (SDA) und grün (SCL) an. Überspringen Sie einen Stift und schließen Sie dann den schwarzen Draht (Erdung) an. Weitere Informationen finden Sie unter Python Computer Wiring.

Schließen Sie das Stromkabel an den Raspberry Pi und das andere Ende an eine Steckdose an, um ihn einzuschalten.

Konfigurieren Ihres Raspberry Pi

1. Wählen Sie unter Welcome to Raspberry Pi (Willkommen bei Raspberry Pi), Next (Weiter).
2. Wählen Sie Ihr Land, Ihre Sprache, die Zeitzone und das Tastaturlayout. Wählen Sie Weiter aus.
3. Geben Sie ein Passwort für Ihren Raspberry Pi ein und wählen Sie dann Next (Weiter).
4. Wählen Sie Ihr Wi-Fi Netzwerk und dann Weiter. Wenn Sie kein Wi-Fi Netzwerk verwenden, wählen Sie Überspringen.

5. Wählen Sie Next (Weiter), um nach Software-Updates zu suchen. Wenn die Updates abgeschlossen sind, wählen Sie Restart (Neu starten), um Ihren Raspberry Pi neu zu starten.

Nach dem Start Ihres Raspberry Pi aktivieren Sie die I2C-Schnittstelle.

1. Klicken Sie in der oberen linken Ecke des Raspbian-Desktops auf das Raspberry-Symbol, wählen Sie Preferences (Einstellungen) und dann Raspberry Pi Configuration (Raspberry Pi-Konfiguration).
2. Wählen Sie auf der Registerkarte Interfaces (Schnittstellen) für I2C die Option Enable (Aktivieren).
3. Wählen Sie OK aus.

Die Bibliotheken für den Adafruit STEMMA-Feuchtigkeitssensor sind dafür geschrieben. CircuitPython Um sie auf einem Raspberry Pi auszuführen, müssen Sie die neueste Version von Python 3 installieren.

1. Führen Sie die folgenden Befehle über eine Eingabeaufforderung aus, um Ihre Raspberry Pi-Software zu aktualisieren:

   ```
   sudo apt-get update
   ```

   ```
   sudo apt-get upgrade
   ```
2. Führen Sie den folgenden Befehl aus, um Ihre Python 3-Installation zu aktualisieren:

   ```
   sudo pip3 install --upgrade setuptools
   ```
3. Führen Sie den folgenden Befehl aus, um die Raspberry Pi GPIO-Bibliotheken zu installieren:

   ```
   pip3 install RPI.GPIO
   ```
4. Führen Sie den folgenden Befehl aus, um die Adafruit Blinka-Bibliotheken zu installieren:

   ```
   pip3 install adafruit-blinka
   ```

   Weitere Informationen finden Sie unter CircuitPython Bibliotheken auf dem Raspberry Pi installieren.
5. Führen Sie den folgenden Befehl aus, um die Adafruit Seesaw-Bibliotheken zu installieren:

   ```
   sudo pip3 install adafruit-circuitpython-seesaw
   ```

6. Führen Sie den folgenden Befehl aus, um das AWS IoT Geräte-SDK für Python zu installieren:

```
pip3 install AWSIoTPythonSDK
```

Ihr Raspberry Pi verfügt jetzt über alle erforderlichen Bibliotheken. Erstellen Sie eine Datei mit dem Namen **`moistureSensor.py`** und kopieren Sie den folgenden Python-Code in die Datei:

```
from adafruit_seesaw.seesaw import Seesaw
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTShadowClient
from board import SCL, SDA

import logging
import time
import json
import argparse
import busio

# Shadow JSON schema:
#
# {
#   "state": {
#       "desired":{
#           "moisture":<INT VALUE>,
#           "temp":<INT VALUE>
#       }
#   }
# }

# Function called when a shadow is updated
def customShadowCallback_Update(payload, responseStatus, token):

    # Display status and data from update request
    if responseStatus == "timeout":
        print("Update request " + token + " time out!")

    if responseStatus == "accepted":
        payloadDict = json.loads(payload)
        print("~~~~~~~~~~~~~~~~~~~~~~~")
        print("Update request with token: " + token + " accepted!")
        print("moisture: " + str(payloadDict["state"]["reported"]["moisture"]))
        print("temperature: " + str(payloadDict["state"]["reported"]["temp"]))
        print("~~~~~~~~~~~~~~~~~~~~~~~\n\n")
```

```
    if responseStatus == "rejected":
        print("Update request " + token + " rejected!")

# Function called when a shadow is deleted
def customShadowCallback_Delete(payload, responseStatus, token):

     # Display status and data from delete request
    if responseStatus == "timeout":
        print("Delete request " + token + " time out!")

    if responseStatus == "accepted":
        print("~~~~~~~~~~~~~~~~~~~~~~~")
        print("Delete request with token: " + token + " accepted!")
        print("~~~~~~~~~~~~~~~~~~~~~~~\n\n")

    if responseStatus == "rejected":
        print("Delete request " + token + " rejected!")


# Read in command-line parameters
def parseArgs():

    parser = argparse.ArgumentParser()
    parser.add_argument("-e", "--endpoint", action="store", required=True, dest="host",
 help="Your device data endpoint")
    parser.add_argument("-r", "--rootCA", action="store", required=True,
 dest="rootCAPath", help="Root CA file path")
    parser.add_argument("-c", "--cert", action="store", dest="certificatePath",
 help="Certificate file path")
    parser.add_argument("-k", "--key", action="store", dest="privateKeyPath",
 help="Private key file path")
    parser.add_argument("-p", "--port", action="store", dest="port", type=int,
 help="Port number override")
    parser.add_argument("-n", "--thingName", action="store", dest="thingName",
 default="Bot", help="Targeted thing name")
    parser.add_argument("-id", "--clientId", action="store", dest="clientId",
 default="basicShadowUpdater", help="Targeted client id")

    args = parser.parse_args()
    return args


# Configure logging
```

```
# AWSIoTMQTTShadowClient writes data to the log
def configureLogging():

    logger = logging.getLogger("AWSIoTPythonSDK.core")
    logger.setLevel(logging.DEBUG)
    streamHandler = logging.StreamHandler()
    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s -
 %(message)s')
    streamHandler.setFormatter(formatter)
    logger.addHandler(streamHandler)


# Parse command line arguments
args = parseArgs()

if not args.certificatePath or not args.privateKeyPath:
    parser.error("Missing credentials for authentication.")
    exit(2)

# If no --port argument is passed, default to 8883
if not args.port:
    args.port = 8883


# Init AWSIoTMQTTShadowClient
myAWSIoTMQTTShadowClient = None
myAWSIoTMQTTShadowClient = AWSIoTMQTTShadowClient(args.clientId)
myAWSIoTMQTTShadowClient.configureEndpoint(args.host, args.port)
myAWSIoTMQTTShadowClient.configureCredentials(args.rootCAPath, args.privateKeyPath,
 args.certificatePath)

# AWSIoTMQTTShadowClient connection configuration
myAWSIoTMQTTShadowClient.configureAutoReconnectBackoffTime(1, 32, 20)
myAWSIoTMQTTShadowClient.configureConnectDisconnectTimeout(10) # 10 sec
myAWSIoTMQTTShadowClient.configureMQTTOperationTimeout(5) # 5 sec

# Initialize Raspberry Pi's I2C interface
i2c_bus = busio.I2C(SCL, SDA)

# Intialize SeeSaw, Adafruit's Circuit Python library
ss = Seesaw(i2c_bus, addr=0x36)

# Connect to AWS IoT
myAWSIoTMQTTShadowClient.connect()
```

```
# Create a device shadow handler, use this to update and delete shadow document
deviceShadowHandler =
 myAWSIoTMQTTShadowClient.createShadowHandlerWithName(args.thingName, True)

# Delete current shadow JSON doc
deviceShadowHandler.shadowDelete(customShadowCallback_Delete, 5)

# Read data from moisture sensor and update shadow
while True:

    # read moisture level through capacitive touch pad
    moistureLevel = ss.moisture_read()

    # read temperature from the temperature sensor
    temp = ss.get_temp()

    # Display moisture and temp readings
    print("Moisture Level: {}".format(moistureLevel))
    print("Temperature: {}".format(temp))

    # Create message payload
    payload = {"state":{"reported":{"moisture":str(moistureLevel),"temp":str(temp)}}}

    # Update shadow
    deviceShadowHandler.shadowUpdate(json.dumps(payload), customShadowCallback_Update,
 5)
    time.sleep(1)
```

Speichern Sie die Datei an einem Speicherort, an dem Sie sie finden können. Führen Sie `moistureSensor.py` über die Befehlszeile mit den folgenden Parametern aus:

Endpunkt

Ihr benutzerdefinierter AWS IoT Endpunkt. Weitere Informationen finden Sie unter Geräteschatten-REST-API.

rootCA

Der vollständige Pfad zu Ihrem AWS IoT Root-CA-Zertifikat.

cert

Der vollständige Pfad zu Ihrem AWS IoT Gerätezertifikat.

Schlüssel

Der vollständige Pfad zum privaten Schlüssel Ihres AWS IoT Gerätezertifikats.

thingName

Ihr Objektname (in diesem Fall `RaspberryPi`).

clientId

Die MQTT-Client-ID. Verwenden Sie `RaspberryPi`.

Die Befehlszeile sollte wie folgt aussehen:

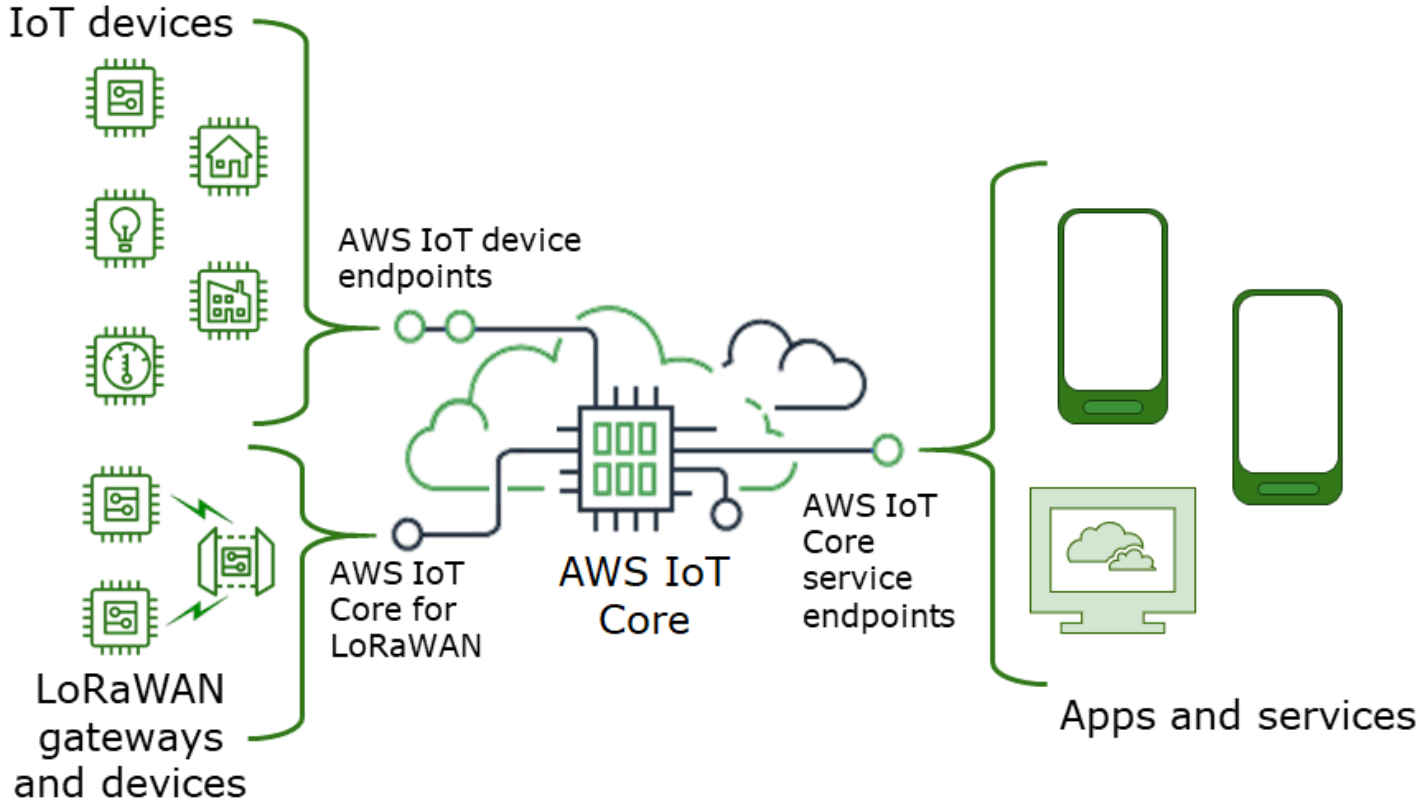
```
python3 moistureSensor.py --endpoint your-endpoint --rootCA ~/certs/
AmazonRootCA1.pem --cert ~/certs/raspberrypi-certificate.pem.crt --key
~/certs/raspberrypi-private.pem.key --thingName RaspberryPi --clientId
RaspberryPi
```

Versuchen Sie, den Sensor zu berühren, ihn in einen Übertopf zu legen oder ihn in ein Glas Wasser zu legen, um zu sehen, wie der Sensor auf verschiedene Feuchtigkeitsstufen reagiert. Bei Bedarf können Sie den Schwellenwert in `MoistureSensorRule` ändern. Wenn der Wert des Feuchtigkeitssensors unter den Wert fällt, der in der SQL-Abfrageanweisung Ihrer Regel angegeben ist, AWS IoT veröffentlicht eine Nachricht zum Amazon SNS-Thema. Sie sollten eine E-Mail-Nachricht erhalten, die die Feuchtigkeits- und Temperaturdaten enthält.

Nachdem Sie den Empfang von E-Mail-Nachrichten von Amazon SNS überprüft haben, drücken Sie CTRL+C (STRG+C), um das Python-Programm zu beenden. Es ist unwahrscheinlich, dass das Python-Programm so viele Nachrichten sendet, dass Kosten anfallen, aber es hat sich bewährt, das Programm zu beenden, wenn Sie fertig sind.

Connect to (Verbinden mit) AWS IoT Core

AWS IoT Core unterstützt Verbindungen mit IoT-Geräten, drahtlosen Gateways, Diensten und Apps. Geräte stellen eine Verbindung zu her, AWS IoT Core sodass sie Daten an AWS IoT Dienste und andere Geräte senden und von diesen empfangen können. Apps und andere Dienste stellen ebenfalls eine Verbindung AWS IoT Core zu den IoT-Geräten her, um sie zu steuern und zu verwalten und die Daten aus Ihrer IoT-Lösung zu verarbeiten. In diesem Abschnitt wird beschrieben, wie Sie AWS IoT Core für jeden Aspekt Ihrer IoT-Lösung die beste Art der Verbindung und Kommunikation auswählen.



Es gibt verschiedene Möglichkeiten, damit zu interagieren. AWS IoT Apps und Dienste können das verwenden AWS IoT Core - Endpunkte der Steuerungsebene und Geräte können AWS IoT Core mithilfe des AWS IoT Geräte-Endpunkte oder AWS IoT Core für LoRa WAN-Regionen und Endpunkte eine Verbindung herstellen.

AWS IoT Core - Endpunkte der Steuerungsebene

Die AWS IoT Core Endpunkte der Steuerungsebene bieten Zugriff auf Funktionen, mit denen Sie Ihre AWS IoT Lösung steuern und verwalten können.

- Endpunkte

  Die Endpunkte von AWS IoT Core – Steuerebene und AWS IoT Core Device Advisor – Steuerebene sind regionsspezifisch und unter AWS IoT Core -Endpunkte und -Kontingente aufgeführt. Die Endpunkte haben folgende Formate:

Zweck des Endpunkts	Endpunkt-Format	Dient
AWS IoT Core – Steuerebene	AWS IoT Endpunkte der Steuerungsebene	AWS IoT Control Plane API
AWS IoT Core Device Advisor — Steuerungsebene	`api.iotdeviceadvisor.` *`aws-region`* `.amazonaws.com`	AWS IoT Core Device Advisor — API für die Steuerungsebene

  - IPv4-Endpunkte — IPv4-Endpunkte unterstützen nur IPv4-Verkehr und sind für alle Regionen verfügbar.

    IPv4-Endpunkte verwenden die folgende Namenskonvention:

    ```
    iot.aws-region.amazonaws.com
    ```

    Der IPv4-Endpunktname für die Region us-east-1 lautet beispielsweise. `iot.us-east-1.amazonaws.com`

  - Dual-stack Endpunkte (IPv4 und IPv6) — Dual-stack Endpunkte unterstützen sowohl IPv4- als auch IPv6-Verkehr. Wenn eine Anfrage an einen Dual-Stack-Endpunkt gestellt wird, wird die Endpunkt-URL je nach dem vom Netzwerk und vom Client verwendeten Protokoll in eine IPv6- oder IPv4-Adresse aufgelöst.

    Dual-stack Endpunkte verwenden die folgende Namenskonvention:

    ```
    iot.aws-region.api.aws
    ```

    Der Dual-Stack-Endpunktname für die Region us-east-1 lautet beispielsweise. `iot.us-east-1.api.aws`

- SDKs und Tools

Die AWS SDKs bieten sprachspezifische Unterstützung für die AWS IoT Core APIs und die APIs anderer Dienste. AWS Die AWS Mobile SDKs bieten App-Entwicklern plattformspezifische Unterstützung für die AWS IoT Core API und andere Dienste auf Mobilgeräten. AWS

Das AWS CLI bietet Befehlszeilenzugriff auf die Funktionen, die von den Dienstendpunkten bereitgestellt werden. AWS IoT AWS Tools für PowerShell stellt Tools zur Verwaltung von AWS Diensten und Ressourcen in der PowerShell Skriptumgebung bereit.

- Authentifizierung

  Die Dienstendpunkte verwenden IAM-Benutzer und AWS Anmeldeinformationen, um Benutzer zu authentifizieren.

- Weitere Informationen

  Weitere Informationen und Links zu SDK-Referenzen finden Sie unter the section called “Connect to (Verbinden mit) AWS IoT Core Service-Endpunkte”.

AWS IoT Geräte-Endpunkte

Die AWS IoT Geräteendpunkte unterstützen die Kommunikation zwischen Ihren IoT-Geräten und. AWS IoT

- Endpunkte

  Die Geräteendpunkte unterstützen AWS IoT Core und AWS IoT Device Management funktionieren. Sie sind spezifisch für Sie AWS-Konto und Sie können sehen, um welche es sich handelt, indem Sie den describe-endpoint Befehl verwenden.

Zweck des Endpunkts	Endpunkt-Format	Dient
AWS IoT Core – Datenebene	Siehe ???.	AWS IoT Datenebene-API
AWS IoT Device Management – Jobdaten	Siehe ???.	AWS IoT API für Jobs auf der Datenebene
AWS IoT Device Advisor — Datenebene	Siehe ???.	Nicht zutreffend

Zweck des Endpunkts	Endpunkt-Format	Dient
AWS IoT Device Management – Secure Tunneling	`api.tunneling.iot.` *`aws-region`* `.amazonaws.com`	AWS IoT Sichere Tunneling-API

Wenn Sie Dual-Stack-Endpunkte (IPv4 und IPv6) für Operationen auf Datenebene verwenden, verwenden Sie den Endpunkttyp. `iot:Data-ATS` `iot:Jobs`kann nur für IPv4 verwendet werden. Weitere Informationen zu diesen Endpunkten und den Funktionen, die sie unterstützen, finden Sie unter the section called “AWS IoT Gerätedaten und Dienstendpunkte”.

- SDKs

  Die AWS IoT Geräte-SDKs bieten sprachspezifische Unterstützung für die Protokolle Message Queueing Telemetry Transport (MQTT) und WebSocket Secure (WSS), mit denen Geräte kommunizieren. AWS IoTAWS SDKs für Mobilgerätebieten auch Unterstützung für MQTT-Gerätekommunikation, AWS IoT APIs und die APIs anderer Dienste auf Mobilgeräten. AWS

- Authentifizierung

  Die Geräteendpunkte verwenden X.509 Zertifikate oder AWS IAM-Benutzer mit Anmeldeinformationen, um Benutzer zu authentifizieren.

- Weitere Informationen

  Weitere Informationen und Links zu SDK-Referenzen finden Sie unter the section called “AWS IoT Geräte-SDKs”.

AWS IoT Core für LoRa WAN-Gateways und -Geräte

AWS IoT Core für LoRa WAN verbindet drahtlose Gateways und Geräte mit. AWS IoT Core

- Endpunkte

  AWS IoT Core for LoRa WAN verwaltet die Gateway-Verbindungen zu Konten und Region-specific AWS IoT Core Endpunkten. Gateways können eine Verbindung zum CUPS (Configuration and Update Server) -Endpunkt Ihres Kontos herstellen, der AWS IoT Core für LoRa WAN bereitgestellt wird.

Zweck des Endpunkts	Endpunkt-Format	Dient
Konfigurations- und Aktualisierungsserver (CUPS)	*account-specific-prefix* .cups.lorawan. *aws-region* .amazonaws.com:443	Gateway-Kommunikation mit dem von AWS IoT Core for LoRa WAN bereitgestellten Konfigurations- und Aktualisierungsserver
LoRaWAN-Netzwerkserver (LNS)	*account-specific-prefix* .gateway.lorawan. *aws-region* .amazonaws.com:443	Gateway-Kommunikation mit dem LoRa WAN-Netzwerkserver, der von AWS IoT Core for LoRa WAN bereitgestellt wird

- SDKs

  Die AWS IoT Wireless-API, auf der AWS IoT Core für LoRa WAN aufgebaut ist, wird vom AWS SDK unterstützt. Weitere Informationen finden Sie unter AWS SDKs und Toolkits.

- Authentifizierung

  AWS IoT Core Verwenden Sie für die Kommunikation mit LoRa WAN-Geräten X.509 Zertifikate, um die Kommunikation mit zu sichern AWS IoT.

- Weitere Informationen

  Weitere Informationen zur Konfiguration und Verbindung drahtloser Geräte finden Sie unter AWS IoT Core Für LoRa WAN-Regionen und Endpunkte.

Connect to (Verbinden mit) AWS IoT Core Service-Endpunkte

Sie können auf die Funktionen der AWS IoT Core Steuerungsebene zugreifen, indem Sie das AWS CLI, das AWS SDK für Ihre bevorzugte Sprache, verwenden oder die REST-API direkt aufrufen. Wir empfehlen, das AWS CLI oder ein AWS SDK für die Interaktion zu verwenden AWS IoT Core , da diese die bewährten Methoden für das Aufrufen von AWS Diensten beinhalten. Das direkte Aufrufen der REST APIs ist eine Option, Sie müssen jedoch die erforderlichen Sicherheitsanmeldedaten angeben, die den Zugriff auf die API ermöglichen.

> **Note**
>
> IoT-Geräte sollten AWS IoT Geräte-SDKs verwenden. Die Geräte-SDKs sind für die Verwendung auf Geräten optimiert, unterstützen die MQTT-Kommunikation mit Geräten und unterstützen die AWS IoT APIs AWS IoT, die am häufigsten von Geräten verwendet werden. Weitere Informationen zu den Geräte SDKs und den Funktionen, die diese bereitstellen, erhalten Sie unter AWS IoT Geräte-SDKs.
> Mobilgeräte sollten AWS SDKs für Mobilgeräte verwenden. Die Mobile SDKs bieten Unterstützung für AWS IoT APIs, MQTT-Gerätekommunikation und die APIs anderer AWS Dienste auf Mobilgeräten. Weitere Informationen zu den Mobilgeräte-SDKs und den Funktionen, die diese bereitstellen, erhalten Sie unter AWS SDKs für Mobilgeräte.

Sie können AWS Amplify Tools und Ressourcen in Web- und Mobilanwendungen verwenden, um einfacher eine Verbindung herzustellen. AWS IoT Core Weitere Informationen zum Herstellen einer Verbindung mithilfe AWS IoT Core von Amplify finden Sie PubSub in der Amplify-Dokumentation.

In den folgenden Abschnitten werden die Tools und SDKs beschrieben, die Sie für die Entwicklung und Interaktion mit AWS IoT anderen Diensten verwenden können. AWS Eine vollständige Liste der AWS Tools und Entwicklungskits, mit denen Apps erstellt und verwaltet werden können, finden Sie unter Tools AWS, auf denen Sie aufbauen können. AWS

AWS CLI for AWS IoT Core

Das AWS CLI bietet Befehlszeilenzugriff auf AWS APIs.

- Installation

  Informationen zur Installation von finden Sie unter Installation von. AWS CLIAWS CLI
- Authentifizierung

  Das AWS CLI verwendet Anmeldeinformationen von Ihrem AWS-Konto.
- Referenz

  Informationen zu den AWS CLI Befehlen für diese AWS IoT Core Dienste finden Sie unter:
  - AWS CLI Befehlsreferenz für IoT
  - AWS CLI Befehlsreferenz für IoT-Daten
  - AWS CLI Befehlsreferenz für IoT-Jobdaten

- AWS CLI Befehlsreferenz für sicheres IoT-Tunneling

Tools zur Verwaltung von AWS Diensten und Ressourcen in der PowerShell Skriptumgebung finden Sie unter AWS Tools für. PowerShell

AWS SDKs

Mit AWS SDKs können Ihre Apps und kompatiblen Geräte AWS IoT APIs und die APIs anderer AWS Dienste aufrufen. Dieser Abschnitt enthält Links zu den AWS SDKs und zur API-Referenzdokumentation für die APIs der AWS IoT Core Dienste.

Das Tool AWS SDKs unterstützen diese AWS IoT Core APIs

- AWS IoT
- AWS IoT Datenebene
- AWS IoT Jobs (Datenebene)
- AWS IoT Sicheres Tunneln
- AWS IoT Kabellos

C++

Um das zu installieren AWS SDK für C++ und benutze es, um eine Verbindung herzustellen AWS IoT:

1. Folgen Sie den Anweisungen unter Erste Schritte mit dem AWS SDK für C++

   In diesen Anweisungen wird beschrieben, wie Sie:

   - Das SDK aus Quelldateien installieren und erstellen
   - Anmeldeinformationen bereitstellen, um das SDK mit Ihrem AWS-Konto zu verwenden
   - Das SDK in Ihrer App oder Ihrem Dienst initialisieren und beenden
   - Ein CMake-Projekt anlegen, um Ihre App oder Ihren Dienst zu erstellen
2. Eine Beispiel-App erstellen und ausführen Beispiel-Apps, die das AWS SDK für C++ verwenden, finden Sie unter AWS SDK für C++ -Codebeispiele.

Dokumentation für den AWS IoT Core Dienstleistungen, die AWS SDK für C++ unterstützt

- Referenzdokumentation zu AWS::IoTClient"
- Aws:IoTDataPlane:: IoTDataPlaneClient Referenzdokumentation
- AwsIoTJobsDataPlane:: IoTJobsDataPlaneClient Referenzdokumentation
- AwsIoTSecureTunneling:: IoTSecureTunnelingClient Referenzdokumentation

Go

Um das zu installieren AWS SDK für Go und benutze es, um eine Verbindung herzustellen AWS IoT:

1. Folgen Sie den Anweisungen unter Erste Schritte mit der AWS SDK für Go

   In diesen Anweisungen wird beschrieben, wie Sie:

   - Installieren Sie das AWS SDK für Go
   - Den Zugriffsschlüssel für das SDK beziehen, um auf Ihr AWS-Konto zuzugreifen
   - Pakete in den Quellcode unserer Apps oder Dienste importieren
2. Eine Beispiel-App erstellen und ausführen. Informationen zu Beispiel-Apps, die AWS SDK für Go nutzen, finden Sie unter AWS SDK für Go -Codebeispiele.

Dokumentation für den AWS IoT Core Dienstleistungen, die AWS SDK für Go unterstützt

- Referenzdokumentation zu IoT
- IoTDataPlane Referenzdokumentation
- IoTJobsDataPlane Referenzdokumentation
- IoTSecureTunneling Referenzdokumentation

Java

Um das zu installieren AWS SDK für Java und benutze es, um eine Verbindung herzustellen AWS IoT:

1. Folgen Sie den Anweisungen unter Erste Schritte mit AWS SDK for Java 2.x

   In diesen Anweisungen wird beschrieben, wie Sie:

- Melden Sie sich für einen IAM-Benutzer an AWS und erstellen Sie ihn
- Das SDK herunterladen
- Richten Sie AWS Anmeldeinformationen und Region ein
- Das SDK mit Apache Maven verwenden
- Das SDK mit Gradle verwenden

2. Mit einem der AWS SDK for Java 2.x Codebeispiele eine Beispiel-App erstellen und ausführen.
3. Die SDK-API-Referenzdokumentation prüfen

Dokumentation für den AWS IoT Core Dienstleistungen, die AWS SDK für Java unterstützt

- IotClient Referenzdokumentation
- IotDataPlaneClient Referenzdokumentation
- IotJobsDataPlaneClient Referenzdokumentation
- IoTSecureTunnelingClient Referenzdokumentation

JavaScript

Um das zu installieren AWS SDK für JavaScript und benutze es, um eine Verbindung herzustellen AWS IoT:

1. Folgen Sie den Anweisungen in Einrichten von AWS SDK für JavaScript. Diese Anweisungen gelten für die Verwendung AWS SDK für JavaScript im Browser und mit Node.JS. Stellen Sie sicher, dass Sie die für Ihre Installation geltenden Anweisungen befolgen.

   In diesen Anweisungen wird beschrieben, wie Sie:

   - Auf Voraussetzungen prüfen
   - Installieren Sie das SDK für JavaScript
   - Laden Sie das SDK für JavaScript

2. Eine Beispiel-App erstellen und ausführen, um mit dem SDK zu beginnen, wie in der Option Erste Schritte für Ihre Umgebung beschrieben wird.
   - Beginnen Sie mit dem AWS SDK für JavaScript im Browser, oder
   - Beginnen Sie mit dem AWS SDK für JavaScript in Node.js

Dokumentation für den AWS IoT Core Dienstleistungen, die der AWS SDK für JavaScript unterstützt

- AWS.Iot reference documentation
- AWS.IotData reference documentation
- AWS.IotJobsDataPlane reference documentation
- AWS.IotSecureTunneling reference documentation

.NET

Um das zu installieren AWS SDK für .NET und benutze es, um eine Verbindung herzustellen AWS IoT:

1. Folgen Sie den Anweisungen unter Einrichtung Ihrer AWS SDK für .NET Umgebung
2. Folgen Sie den Anweisungen unter AWS SDK für .NET Projekt einrichten

   In diesen Anweisungen wird beschrieben, wie Sie:

   - Ein neues Projekt starten
   - AWS Zugangsdaten abrufen und konfigurieren
   - Installieren Sie AWS SDK-Pakete
3. Erstellen Sie eines der Beispielprogramme unter Arbeiten mit AWS Diensten im AWS SDK für .NET und führen Sie es aus
4. Die SDK-API-Referenzdokumentation prüfen

Dokumentation für AWS IoT Core Dienstleistungen, die der AWS SDK für .NET unterstützt

- Amazon.IoT.Model Referenzdokumentation
- Amazon.IotData.Model Referenzdokumentation
- Amazon.IoTJobsDataPlane.Model Referenzdokumentation
- Amazon.IoTSecureTunneling.Model Referenzdokumentation

PHP

Um das zu installieren AWS SDK für PHP und benutze es, um eine Verbindung herzustellen AWS IoT:

1. Folgen Sie den Anweisungen unter Erste Schritte mit der AWS SDK für PHP Version 3

   In diesen Anweisungen wird beschrieben, wie Sie:
   - Auf Voraussetzungen prüfen
   - Das SDK installieren
   - Das SDK auf ein PHP-Skript anwenden
2. Eine Beispiel-App mit einem der AWS SDK für PHP Version 3-Codebeispiele erstellen und ausführen

Dokumentation für die AWS IoT Core Dienstleistungen, die der AWS SDK für PHP unterstützt

- Referenzdokumentation zu IoTClient
- IoTDataPlaneClient Referenzdokumentation
- IoTJobsDataPlaneClient Referenzdokumentation
- IoTSecureTunnelingClient Referenzdokumentation

Python

Um das zu installieren AWS SDK für Python (Boto3) und benutze es, um eine Verbindung herzustellen AWS IoT:

1. Folgen Sie den Anweisungen unter AWS SDK für Python (Boto3) Schnellstart

   In diesen Anweisungen wird beschrieben, wie Sie:
   - Das SDK installieren
   - Das SDKs konfigurieren
   - Das SDK in Ihrem Code verwenden
2. Ein Beispielprogramm, das AWS SDK für Python (Boto3) nutzt, erstellen und ausführen

   Dieses Programm zeigt die derzeit konfigurierten Protokollierungsoptionen des Kontos an. Nachdem Sie das SDK installiert und für Ihr Konto konfiguriert haben, sollten Sie dieses Programm ausführen können.

```
import boto3
import json

# initialize client
iot = boto3.client('iot')

# get current logging levels, format them as JSON, and write them to stdout
response = iot.get_v2_logging_options()
print(json.dumps(response, indent=4))
```

Weitere Informationen zu der in diesem Beispiel verwendeten Funktion finden Sie unter the section called “Konfiguration AWS IoT Protokollierung”.

Dokumentation für den AWS IoT Core Dienstleistungen, die der AWS SDK für Python (Boto3) unterstützt

- Referenzdokumentation zu IoT
- IoTDataPlane Referenzdokumentation
- IoTJobsDataPlane Referenzdokumentation
- IoTSecureTunneling Referenzdokumentation

Ruby

Um das zu installieren AWS SDK für Ruby und benutze es, um eine Verbindung herzustellen AWS IoT:

- Folgen Sie den Anweisungen unter Erste Schritte mit der AWS SDK für Ruby

  In diesen Anweisungen wird beschrieben, wie Sie:

  - Das SDK installieren
  - Das SDKs konfigurieren
- Das Hello World Tutorial erstellen und ausführen

Dokumentation für den AWS IoT Core Dienstleistungen, die der AWS SDK für Ruby unterstützt

- Referenzdokumentation zu Aws::IoT::Client

- Referenzdokumentation zu AwsIoTDataPlane:: :Client
- Referenzdokumentation zu AwsIoTJobsDataPlane:: :Client
- Referenzdokumentation zu AwsIoTSecureTunneling:: :Client

AWS SDKs für Mobilgeräte

Die AWS mobilen SDKs bieten Entwicklern mobiler Apps plattformspezifische Unterstützung für die APIs der AWS IoT Core Dienste, die IoT-Gerätekommunikation mithilfe von MQTT und die APIs anderer Dienste. AWS

Android

AWS Mobile SDK für Android

Das AWS Mobile SDK für Android enthält eine Bibliothek, Beispiele und Dokumentation, mit der Entwickler verbundene mobile Anwendungen erstellen können. AWS Dieses SDK unterstützt auch die Kommunikation mit MQTT-Geräten und das Aufrufen der APIs der AWS IoT Core Dienste. Weitere Informationen finden Sie hier:

- AWS Mobiles SDK für Android auf GitHub
- AWS Readme zum mobilen SDK für Android
- AWS Beispiele für das mobile SDK für Android
- AWS API-Referenz für SDK für Android
- AWSIoTClient Dokumentation zur Klassenreferenz

iOS

AWS Mobile SDK for iOS

Das AWS Mobile SDK for iOS ist ein Open-Source-Software-Entwicklungskit, das unter einer Apache Open Source-Lizenz vertrieben wird. Das SDK für iOS bietet eine Bibliothek, Codebeispiele und Dokumentation, mit denen Entwickler verbundene mobile Anwendungen erstellen AWS können. Dieses SDK unterstützt auch die Kommunikation mit MQTT-Geräten und das Aufrufen der APIs der AWS IoT Core Dienste. Weitere Informationen finden Sie hier:

- AWS Mobile SDK for iOS auf GitHub
- AWS SDK für iOS Readme

- AWS Beispiele für SDK für iOS
- AWS IoT Klassenreferenzdokumente im AWS SDK für iOS

REST-APIs der AWS IoT Core service

Die REST-APIs der AWS IoT Core Dienste können direkt mithilfe von HTTP-Anfragen aufgerufen werden.

- Endpunkt-URL

  Die Dienstendpunkte, die die REST-APIs der AWS IoT Core -Dienste verfügbar machen, variieren je nach Region und sind unter AWS IoT Core -Endpunkte und -Kontingente aufgeführt. Sie müssen den Endpunkt für die Region verwenden, die die AWS IoT Ressourcen enthält, auf die Sie zugreifen möchten, da die AWS IoT Ressourcen regionsspezifisch sind.
- Authentifizierung

  Die REST-APIs der AWS IoT Core Dienste verwenden AWS IAM-Anmeldeinformationen für die Authentifizierung. Weitere Informationen finden Sie unter Signieren von AWS API-Anfragen in der AWS Allgemeinen Referenz.
- API-Referenz

  Informationen zu den spezifischen Funktionen, die von den REST-APIs der AWS IoT Core Dienste bereitgestellt werden, finden Sie unter:
  - API-Referenz für das IoT.
  - API-Referenz für IoT-Daten.
  - API-Referenz für IoT–Jobdaten.
  - API-Referenz für IoT Secure Tunneling

Geräte verbinden mit AWS IoT

Geräte stellen eine Verbindung zu AWS IoT und anderen Diensten über AWS IoT Core. Über AWS IoT Core diese Funktion senden und empfangen Geräte Nachrichten über Geräteendpunkte, die für Ihr Konto spezifisch sind. Die the section called “AWS IoT Geräte-SDKs” unterstützen die Gerätekommunikation mithilfe der MQTT- und WSS-Protokolle. Weitere Informationen zu von Geräten unterstützten Protokollen finden Sie unter the section called “Gerätekommunikationsprotokolle”.

Der Message Broker

AWS IoT verwaltet die Gerätekommunikation über einen Message Broker. Geräte und Kunden veröffentlichen Nachrichten im Message Broker und abonnieren auch Nachrichten, die der Message Broker veröffentlicht. Nachrichten werden durch ein anwendungsdefiniertes Thema identifiziert. Wenn der Message Broker eine Nachricht empfängt, die von einem Gerät oder Kunden veröffentlicht wurde, veröffentlicht er diese Nachricht an die Geräte und Kunden, die das Nachrichtenthema abonniert haben. Der Message Broker leitet Nachrichten auch an die AWS IoT Rules Engine weiter, die auf den Inhalt der Nachricht reagieren kann.

AWS IoT Nachrichtensicherheit

Zu AWS IoT verwendende Geräteverbindungen the section called “X.509 Client-Zertifikate” und AWS Signatur V4 für die Authentifizierung. Die Gerätekommunikation wird durch TLS Version 1.3 gesichert und AWS IoT erfordert, dass Geräte die Server Name Indication (SNI) - Erweiterung senden, wenn sie eine Verbindung herstellen. Weitere Informationen finden Sie unter Transportsicherheit unter AWS IoT.

AWS IoT Gerätedaten und Dienstendpunkte

> **Important**
>
> Sie können die Endpunkte auf Ihrem Gerät zwischenspeichern oder speichern. Das bedeutet, dass Sie die `DescribeEndpoint` API nicht jedes Mal abfragen müssen, wenn ein neues Gerät angeschlossen wird. Die Endpunkte ändern sich nicht, nachdem sie für Ihr Konto AWS IoT Core erstellt wurden.

Jedes Konto hat mehrere Geräteendpunkte, die für das Konto einzigartig sind und bestimmte IoT-Funktionen unterstützen. Die AWS IoT Gerätedatenendpunkte unterstützen ein publish/subscribe Protokoll, das für die Kommunikationsanforderungen von IoT-Geräten entwickelt wurde. Andere Clients, wie Apps und Dienste, können diese Schnittstelle jedoch ebenfalls verwenden, wenn ihre Anwendung die speziellen Funktionen erfordert, die diese Endpunkte bieten. Die AWS IoT Geräte-Serviceendpunkte unterstützen den geräteorientierten Zugriff auf Sicherheits- und Verwaltungsdienste.

Den Endpunkt der Gerätedaten Ihres Kontos finden Sie auf der Seite „Einstellungen“ Ihrer Konsole. AWS IoT Core

Um den Geräteendpunkt Ihres Kontos für einen bestimmten Zweck zu ermitteln, einschließlich des Gerätedatenendpunkts, verwenden Sie den hier gezeigten CLI-Befehl describe-endpoint oder die REST-API `DescribeEndpoint` und geben Sie den Parameterwert *`endpointType`* aus der folgenden Tabelle an.

```
aws iot describe-endpoint --endpoint-type endpointType
```

Dieser Befehl gibt eine *`iot-endpoint`* im folgenden Format zurück:*`account-specific-prefix`*`.iot.`*`aws-region`*`.amazonaws.com`.

Jeder Kunde verfügt über einen `iot:Data-ATS` und einen `iot:Data`-Endpunkt. Jeder Endpunkt verwendet ein X.509 Zertifikat, um den Client zu authentifizieren. Wir empfehlen dringend, dass Kunden den neueren `iot:Data-ATS`-Endpunkttyp verwenden, um Probleme im Zusammenhang mit dem weit verbreiteten Misstrauen gegenüber Symantec-Zertifizierungsstellen zu vermeiden. Wir bieten den `iot:Data` Endpunkt, an dem Geräte Daten von alten Endpunkten abrufen können, die VeriSign Zertifikate aus Gründen der Abwärtskompatibilität verwenden. Weitere Informationen finden Sie unter Server-Authentifizierung.

AWS IoT Endpunkte für Geräte

Zweck des Endpunkts	***`endpointType`*** Wert	Description
AWS IoT Core – Operationen auf Datenebene	`iot:Data-ATS`	Wird zum Senden und Empfangen von Daten an und von den Message Broker-, Geräteschatten- und Regel-Engine-Komponenten von AWS IoT verwendet. `iot:Data-ATS` gibt einen ATS-signierten Datenendpunkt zurück
AWS IoT Core – Operationen auf Datenebene (veraltet)	`iot:Data`	`iot:Data`gibt einen VeriSign signierten Datenendpunkt zurück, der aus Gründen der Abwärtskompatibili tät bereitgestellt wird. MQTT 5 wird auf Symantec

Zweck des Endpunkts	***endpointType*** Wert	Description
		(`iot:Data`)-Endpunkten nicht unterstützt.
AWS IoT Core Zugriff auf Anmeldeinformationen	`iot:CredentialProvider`	Wird verwendet, um das integrierte X.509 Zertifikat eines Geräts gegen temporäre Anmeldeinformationen auszutauschen, um eine direkte Verbindung mit anderen AWS Diensten herzustellen. Weitere Informationen zum Herstellen einer Verbindung zu anderen AWS Diensten finden Sie unter Autorisieren von Direktanrufen an AWS Dienste.
AWS IoT Device Management – Jobdaten-Operationen	`iot:Jobs`	Wird verwendet, um Geräten die Interaktion mit dem Jobs-Dienst mithilfe der AWS IoT Jobs-Geräte-HTTPS-APIs zu ermöglichen. `iot:Jobs`kann nur für IPv4 verwendet werden. Wenn Sie Dual-Stack-Endpunkte (IPv4 und IPv6) verwenden, verwenden Sie den Endpunkttyp. `iot:Data-ATS`
AWS IoT Funktionen des Device Advisors	`iot:DeviceAdvisor`	Ein Testendpunkttyp, der zum Testen von Geräten mit Device Advisor verwendet wird. Weitere Informationen finden Sie unter ???.

Zweck des Endpunkts	*endpointType* Wert	Description
AWS IoT Core Daten-Beta (Vorschau)	`iot:Data-Beta`	Ein Endpunkttyp, der Betaversionen vorbehalten ist. Informationen zu seiner aktuellen Verwendung finden Sie unter ???.

Sie können auch Ihren eigenen vollqualifizierten Domänennamen (FQDN) verwenden, z. B. und das zugehörige Serverzertifikat*example.com*, mit dem Sie Geräte verbinden können AWS IoT . the section called “Domain-Konfigurationen”

AWS IoT Geräte-SDKs

Die AWS IoT Geräte-SDKs helfen Ihnen dabei, Ihre IoT-Geräte mit den Protokollen MQTT AWS IoT Core und MQTT über WSS zu verbinden.

Die AWS IoT Geräte-SDKs unterscheiden sich von den AWS SDKs darin, dass die AWS IoT Geräte-SDKs die speziellen Kommunikationsanforderungen von IoT-Geräten unterstützen, aber nicht alle von den SDKs unterstützten Dienste unterstützen. AWS Die AWS IoT Geräte-SDKs sind mit den AWS SDKs kompatibel, die alle AWS Dienste unterstützen. Sie verwenden jedoch unterschiedliche Authentifizierungsmethoden und stellen Verbindungen zu verschiedenen Endpunkten her, was die Verwendung der SDKs auf einem IoT-Gerät unpraktisch machen könnte. AWS

Mobilgeräte

Sie the section called “AWS SDKs für Mobilgeräte” unterstützen sowohl die MQTT-Gerätekommunikation, einige der AWS IoT Service-APIs als auch die APIs anderer AWS Dienste. Wenn Sie auf einem unterstützten Mobilgerät entwickeln, überprüfen Sie dieses SDK, um festzustellen, ob es die beste Option für die Entwicklung Ihrer IoT-Lösung ist.

C++

AWS IoT C++-Geräte-SDK

Das AWS IoT C++ Device SDK ermöglicht es Entwicklern, verbundene Anwendungen mithilfe AWS der APIs der AWS IoT Core Dienste zu erstellen. Dieses SDK wurde speziell für Geräte entwickelt, die nicht ressourcenbeschränkt sind und die erweiterte Funktionen benötigen,

wie beispielsweise Nachrichtenwarteschlangen, Multithreading-Support und die aktuellen Sprachfunktionen. Weitere Informationen finden Sie hier:

- AWS IoT Geräte-SDK C++ v2 ein GitHub
- AWS IoT Readme-Datei zum Geräte-SDK C++ v2
- AWS IoT Beispiele für Geräte-SDK C++ v2
- AWS IoT API-Dokumentation zum Geräte-SDK C++ v2

Python

AWS IoT Geräte-SDK für Python

Das AWS IoT Geräte-SDK für Python ermöglicht es Entwicklern, Python-Skripte zu schreiben, um ihre Geräte für den Zugriff auf die AWS IoT Plattform über MQTT oder MQTT über das WebSocket Secure (WSS) -Protokoll zu verwenden. Indem sie ihre Geräte mit den APIs der AWS IoT Core Dienste verbinden, können Benutzer sicher mit dem Message Broker, den Regeln und dem Device Shadow-Service arbeiten, der sie AWS IoT Core bereitstellt AWS Lambda, sowie mit anderen AWS Diensten wie Amazon Kinesis und Amazon S3 und mehr.

- AWS IoT Das Geräte-SDK für Python v2 ist aktiviert GitHub
- AWS IoT Readme-Datei zum Geräte-SDK für Python v2
- AWS IoT Beispiele für das Geräte-SDK für Python v2
- AWS IoT API-Dokumentation zum Geräte-SDK für Python v2

JavaScript

AWS IoT Geräte-SDK für JavaScript

Das AWS IoT Geräte-SDK für JavaScript ermöglicht es Entwicklern, JavaScript Anwendungen zu schreiben, die über das Protokoll MQTT oder MQTT auf APIs zugreifen. AWS IoT Core WebSocket Es kann in Node.js Umgebungen und Browseranwendungen verwendet werden. Weitere Informationen finden Sie hier:

- AWS IoT Geräte-SDK für JavaScript Version 2 an GitHub
- AWS IoT Readme-Datei zum Geräte-SDK JavaScript für Version 2
- AWS IoT Geräte-SDK für JavaScript v2-Beispiele

- AWS IoT Geräte-SDK für die JavaScript v2-API-Dokumentation

Java

AWS IoT Geräte-SDK für Java

Das AWS IoT Geräte-SDK für Java ermöglicht es Java-Entwicklern, AWS IoT Core über MQTT oder MQTT über das Protokoll auf die WebSocket APIs von zuzugreifen. Das SDK unterstützt den Geräteschatten-Dienst. Sie können über die HTTP-Methoden wie ABRUFEN, AKTUALISIEREN und LÖSCHEN auf Schattengeräte zugreifen. Das SDK unterstützt auch ein vereinfachtes Zugangsmodell für Schattengeräte, sodass Entwickler mithilfe der Methoden „Getter” und „Setter” Daten mit den Schattengeräten austauschen können, ohne JSON-Dokumente serialisieren oder deserialisieren zu müssen. Weitere Informationen finden Sie hier:

- AWS IoT Geräte-SDK für Java v2 an GitHub
- AWS IoT Readme-Datei zum Geräte-SDK für Java v2
- AWS IoT Beispiele für Geräte-SDK für Java v2
- AWS IoT API-Dokumentation zum Geräte-SDK für Java v2

Embedded C

AWS IoT Geräte-SDK für Embedded C

> **Important**
>
> Dieses SDK ist für die Verwendung durch erfahrene Entwickler eingebetteter Software vorgesehen.

AWS IoT Device SDK for Embedded C (C-SDK) ist eine Sammlung von C-Quelldateien unter der MIT-Open-Source-Lizenz, die in eingebetteten Anwendungen verwendet werden können, um IoT-Geräte sicher mit AWS IoT Core zu verbinden. Es umfasst MQTT-, JSON Parser- und AWS IoT Device Shadow-Bibliotheken und andere. Es wird als Quellcode verteilt und soll zusammen mit einem Anwendungscode, anderen Bibliotheken und optional einem RTOS (Real Time Operating System) in die Kunden-Firmware integriert werden.

Das AWS IoT Device SDK for Embedded C richtet sich im Allgemeinen an Geräte mit eingeschränkten Ressourcen, die eine optimierte C-Sprachlaufzeit benötigen. Sie können das

SDK auf jedem Betriebssystem verwenden und es auf jedem Prozessortyp hosten (z. B. MCUs und MPUs). Wenn Ihr Gerät über ausreichend Speicher- und Verarbeitungsressourcen verfügt, empfehlen wir Ihnen, eines der anderen AWS IoT Geräte- und Mobile-SDKs zu verwenden, z. B. das AWS IoT Geräte-SDK für C++ JavaScript, Java oder Python.

Weitere Informationen finden Sie hier:

- AWS IoT Geräte-SDK für Embedded C an GitHub
- AWS IoT Readme-Datei zum Geräte-SDK für Embedded C
- AWS IoT Geräte-SDK für Embedded C-Beispiele

Gerätekommunikationsprotokolle

AWS IoT Core unterstützt Geräte und Clients, die die Protokolle MQTT und MQTT over WebSocket Secure (WSS) verwenden, um Nachrichten zu veröffentlichen und zu abonnieren, Geräte und Clients, die das HTTPS-Protokoll zum Veröffentlichen von Nachrichten verwenden, und die Direct Messaging-API, um Nachrichten an bestimmte verbundene Clients zu senden. Alle Protokolle unterstützen IPv4 und IPv6. In diesem Abschnitt werden die verschiedenen Verbindungsoptionen für Geräte und Kunden beschrieben.

Versionen des TLS-Protokolls

AWS IoT Core verwendet TLS Version 1.2 und TLS Version 1.3, um die gesamte Kommunikation zu verschlüsseln. Sie können zusätzliche TLS-Richtlinienversionen für Ihren Endpunkt konfigurieren, indem Sie die TLS-Einstellungen in den Domänenkonfigurationen https://docs.aws.amazon.com/iot/latest/developerguide/iot-endpoints-tls-config.html konfigurieren. Wenn Kunden Geräte mit verbinden AWS IoT Core, können sie die Server Name Indication (SNI) -Erweiterung senden, die für Funktionen wie die Registrierung mehrerer Konten, konfigurierbare Endpunkte, benutzerdefinierte Domänen und VPC-Endpunkte erforderlich ist. Weitere Informationen finden Sie unter Transportsicherheit in AWS IoT.

AWS IoT Geräte-SDKs unterstützen MQTT und MQTT over WSS und unterstützen die Sicherheitsanforderungen von Client-Verbindungen. Wir empfehlen die Verwendung von AWS IoT Geräte-SDKs, um Clients mit dem AWS IoT zu verbinden.

Protokolle, Port-Zuweisungen und Authentifizierung

Wie ein Gerät oder ein Client eine Verbindung zum Message Broker herstellt, kann mithilfe eines Authentifizierungstyps konfiguriert werden. ??? Standardmäßig oder wenn keine SNI-Erweiterung

gesendet wird, basiert die Authentifizierungsmethode auf dem Anwendungsprotokoll, dem Port und der ALPN (Application Layer Protocol Negotiation) TLS-Erweiterung, die die Geräte verwenden. In der folgenden Tabelle ist die erwartete Authentifizierung auf der Grundlage von Port, Port und ALPN aufgeführt.

Protokolle, Authentifizierung und Port-Zuweisungen

Protocol (Protokoll)	Unterstützte Operationen	Authentifizierung	Port	ALPN-Protokollname
MQTT über WebSocket	Veröffentlichen, Abonnieren	Signaturversion 4	443	N/A
MQTT vorbei WebSocket	Veröffentlichen, Abonnieren	Benutzerdefinierte Authentifizierung	443	N/A
MQTT	Veröffentlichen, Abonnieren	X.509 Client-Zertifikat	443†	`x-amzn-mqtt-ca`
MQTT	Veröffentlichen, Abonnieren	X.509 Client-Zertifikat	8883	N/A
MQTT	Veröffentlichen, Abonnieren	Benutzerdefinierte Authentifizierung	443†	`mqtt`
HTTPS	Nur veröffentlichen	Signaturversion 4	443	N/A
HTTPS	Nur veröffentlichen	X.509 Client-Zertifikat	443†	`x-amzn-http-ca`
HTTPS	Nur veröffentlichen	X.509 Client-Zertifikat	8443	N/A
HTTPS	Nur veröffentlichen	Benutzerdefinierte Authentifizierung	443	N/A

ALPN (Application Layer Protocol Negotiation)

† Bei Verwendung von Standard-Endpunktkonfigurationen müssen Clients, die eine Verbindung über Port 443 mit X.509 Client-Zertifikatsauthentifizierung herstellen, die TLS-Erweiterung Application Layer Protocol Negotiation (ALPN) implementieren und den ALPN-Protokollnamen verwenden, der in dem vom Client als Teil der ProtocolNameList Nachricht gesendeten ALPN aufgeführt ist. `ClientHello`
Auf Port 443 unterstützt der IoT:Data-ATS Endpunkt ALPN x-amzn-http-ca HTTP, der Endpunkt jedoch nicht. IoT:Jobs
An Port 8443 HTTPS und Port 443 MQTT mit ALPN x-amzn-mqtt-ca kann die benutzerdefinierte Authentifizierung nicht verwendet werden.

Clients stellen eine Verbindung zu ihren Geräteendpunkten her. AWS-Konto Informationen darüber, wie Sie die Geräteendpunkte Ihres Kontos finden, finden Sie unter the section called “AWS IoT Gerätedaten und Dienstendpunkte”.

Note

AWS SDKs benötigen nicht die gesamte URL. Sie benötigen nur den Endpunkt-Hostnamen, wie im `pubsub.py` Beispiel für AWS IoT Device SDK for Python on. GitHub Wenn Sie die gesamte URL wie in der folgenden Tabelle aufgeführt übergeben, kann dies zu einem Fehler wie einem ungültigen Hostnamen führen.

Verbindung herstellen mit AWS IoT Core

Protocol (Protokoll)	Endpunkt oder URL
MQTT	*iot-endpoint*
MQTT over WSS	wss://*iot-endpoint* /mqtt
HTTPS	https://*iot-endpoint* /topics

Wählen Sie ein Anwendungsprotokoll für Ihre Gerätekommunikation

Für die meisten IoT-Gerätekommunikationen über die Geräteendpunkte sollten Sie die Protokolle Secure MQTT oder MQTT over WebSocket Secure (WSS) verwenden. Die Geräteendpunkte unterstützen jedoch auch HTTPS.

In der folgenden Tabelle wird verglichen, wie die beiden High-Level-Protokolle (MQTT und HTTPS) für die Gerätekommunikation AWS IoT Core verwendet werden.

AWS IoT Geräteprotokolle (MQTT und HTTPS) Seite an Seite

Funktion	MQTT	HTTPS
Publish/Subscribe Unterstützung	Veröffentlichen und Abonnieren	Nur veröffentlichen
SDK-Unterstützung	AWS Geräte-SDKs unterstützen MQTT- und WSS-Protokolle	Keine SDK-Unterstützung, aber Sie können sprachspezifische Methoden verwenden, um HTTPS-Anfragen zu stellen
Qualität der Service-Unterstützung	MQTT QoS Stufen 0 und 1	QoS wird durch die Übergabe eines Abfragezeichenfolge-Parameter `?qos=qos` unterstütz, dessen Wert 0 oder 1 sein kann. Sie können diese Abfragezeichenfolge hinzufügen, um eine Nachricht mit dem gewünschten QoS-Wert zu veröffentlichen.
Können empfangene Nachrichten verpasst werden, während das Gerät offline war	Ja	Nein
Unterstützung von `clientId`-Feldern	Ja	Nein

Funktion	MQTT	HTTPS
Erkennung von Geräteunterbrechungen	Ja	Nein
Sichere Kommunikationen	Ja. Siehe ???	Ja. Siehe ???
Themendefinitionen	Anwendung definiert	Anwendung definiert
Format der Nachrichtendaten	Anwendung definiert	Anwendung definiert
Protokoll-Overhead	Niedriger	Höher
Stromverbrauch	Niedriger	Höher

Wählen Sie einen Authentifizierungstyp für Ihre Gerätekommunikation

Sie können den Authentifizierungstyp für Ihren IoT-Endpunkt mithilfe konfigurierbarer Endpunkte konfigurieren. Verwenden Sie alternativ die Standardkonfiguration und legen Sie fest, wie sich Ihre Geräte mit der Kombination aus Anwendungsprotokoll, Port und ALPN-TLS-Erweiterung authentifizieren. Der von Ihnen gewählte Authentifizierungstyp bestimmt, wie sich Ihre Geräte authentifizieren, wenn Sie eine Verbindung herstellen. AWS IoT Core Es gibt fünf Authentifizierungstypen:

X.509 Zertifikat

Authentifizieren Sie Geräte mithilfe von X.509 Client-Zertifikaten, die AWS IoT Core zur Authentifizierung des Geräts dienen. Dieser Authentifizierungstyp funktioniert mit den Protokollen Secure MQTT (MQTT over TLS) und HTTPS.

X.509 Zertifikat mit benutzerdefiniertem Autorisierer

Authentifizieren Sie Geräte mithilfe von X.509 Client-Zertifikaten und führen Sie zusätzliche Authentifizierungsaktionen mithilfe eines benutzerdefinierten Autorisierers durch, der die X.509 Client-Zertifikatsinformationen erhält. Dieser Authentifizierungstyp funktioniert mit den Protokollen Secure MQTT (MQTT over TLS) und HTTPS. Dieser Authentifizierungstyp ist nur mit konfigurierbaren Endpunkten mit X.509 benutzerdefinierter Authentifizierung möglich. Es gibt keine ALPN-Option.

AWS Signatur Version 4 (SigV4)

Authentifizieren Sie Geräte mithilfe von Cognito oder Ihrem Backend-Service und unterstützen Sie dabei soziale Netzwerke und Unternehmensverbünde. Dieser Authentifizierungstyp funktioniert mit den Protokollen MQTT over WebSocket Secure (WSS) und HTTPS.

Benutzerdefinierter Autorisierer

Authentifizieren Sie Geräte, indem Sie eine Lambda-Funktion so konfigurieren, dass benutzerdefinierte Authentifizierungsinformationen verarbeitet werden, die an gesendet werden. AWS IoT Core Dieser Authentifizierungstyp funktioniert mit den Protokollen Secure MQTT (MQTT over TLS), HTTPS und MQTT over WebSocket Secure (WSS).

Standard

Authentifizieren Sie Geräte auf der Grundlage der ALPN-Erweiterung (Port and/or Application Layer Protocol Negotiation), die Geräte verwenden. Einige zusätzliche Authentifizierungsoptionen werden nicht unterstützt. Weitere Informationen finden Sie unter ???.

Die folgende Tabelle zeigt alle unterstützten Kombinationen von Authentifizierungstypen und Anwendungsprotokollen.

Unterstützte Kombinationen von Authentifizierungstypen und Anwendungsprotokollen

Authentifizierungstyp	Sicheres MQTT (MQTT über TLS)	MQTT über WebSocket Secure (WSS)	HTTPS	Standard
X.509 Zertifikat	✓		✓	
X.509 Zertifikat mit benutzerdefiniertem Autorisierer	✓		✓	
AWS Signatur Version 4 (SigV4)		✓	✓	
Benutzerdefinierter Autorisierer	✓	✓	✓	

Authentifizierungstyp	Sicheres MQTT (MQTT über TLS)	MQTT über WebSocket Secure (WSS)	HTTPS	Standard
Standard				✓

Einschränkungen der Verbindungsdauer

Es kann nicht garantiert werden, dass HTTPS-Verbindungen länger dauern als die Zeit, die für den Empfang und die Beantwortung von Anfragen benötigt wird.

Die Dauer der MQTT-Verbindung ist von der Authentifizierungsfunktion abhängig, die Sie verwenden. In der folgenden Tabelle ist die maximale Verbindungsdauer unter idealen Bedingungen für jede Funktion aufgeführt.

MQTT-Verbindungsdauer nach Authentifizierungsfunktion

Feature	Maximale Dauer *
X.509 Client-Zertifikat	1 bis 2 Wochen
Benutzerdefinierte Authentifizierung	1 bis 2 Wochen
Signaturversion 4	Bis zu 24 Stunden

* Nicht garantiert

Bei X.509 Zertifikaten und benutzerdefinierter Authentifizierung gibt es keine feste Begrenzung für die Verbindungsdauer, sie kann jedoch nur wenige Minuten betragen. Verbindungsunterbrechungen können aus verschiedenen Gründen auftreten. Die folgende Liste enthält einige der gängigsten Gründe.

- Wi-Fi Unterbrechungen der Verfügbarkeit
- Verbindungsunterbrechungen des Internetdienstanbieters (ISP)
- Service-Patches
- Dienstbereitstellungen
- Service Auto Scaling

- Nicht verfügbarer Dienst-Host
- Load Balancer-Probleme und -Aktualisierungen
- Client-side Fehler

Ihre Geräte müssen Strategien zur Erkennung von Verbindungsabbrüchen und zur Wiederherstellung der Verbindung implementieren. Informationen zu Trennungsereignissen und Anleitungen, wie Sie damit umgehen können, finden Sie in ??? unter ???.

MQTT

MQTT (Message Queuing Telemetry Transport) ist ein einfaches, weit verbreitetes Messaging-Protokoll, das für eingeschränkte Geräte konzipiert ist. AWS IoT Core Unterstützung für MQTT basiert auf der MQTT-Spezifikation v3.1.1 und v5.0, mit einigen Unterschieden, wie in the section called “AWS IoT Unterschiede zu den MQTT-Spezifikationen” dokumentiert. Als neueste Version des Standards führt MQTT 5 mehrere wichtige Funktionen ein, die ein MQTT-based System robuster machen, darunter neue Verbesserungen der Skalierbarkeit, verbesserte Fehlerberichte mit Antworten auf Ursachencodes, Timer für den Ablauf von Nachrichten und Sitzungen sowie benutzerdefinierte Nachrichtenüberschriften. Weitere Informationen zu den von MQTT 5 AWS IoT Core unterstützten Funktionen finden Sie unter Von MQTT 5 unterstützte Funktionen. AWS IoT Core unterstützt auch die Kommunikation zwischen den MQTT-Versionen (MQTT 3 und MQTT 5). Ein MQTT-3-Publisher kann eine MQTT-3-Meldung an einen MQTT-5-Subscriber senden, der eine MQTT-5-Publish-Meldung erhält, und umgekehrt.

AWS IoT Core unterstützt Geräteverbindungen, die das MQTT-Protokoll und das MQTT-über-WSS-Protokoll verwenden und die durch eine Client-ID identifiziert werden. AWS IoT Geräte-SDKs unterstützen beide Protokolle und sind die empfohlenen Verbindungsmethoden für Geräte zu AWS IoT Core Die AWS IoT Geräte-SDKs unterstützen die Funktionen, die Geräte und Clients benötigen, um eine Verbindung zu Diensten herzustellen und darauf zuzugreifen. AWS IoT Die Geräte-SDKs unterstützen die Authentifizierungsprotokolle, die die AWS IoT Dienste benötigen, und die Verbindungs-ID-Anforderungen, die das MQTT-Protokoll und die MQTT-über-WSS-Protokolle erfordern. Informationen zum Herstellen einer Verbindung AWS IoT mit den AWS Geräte-SDKs sowie Links zu Beispielen AWS IoT in den unterstützten Sprachen finden Sie unter. the section called “Herstellen einer Verbindung mit MQTT mithilfe der AWS IoT Geräte-SDKs” Weitere Informationen zu Authentifizierung und Portzuordnungen für MQTT-Meldungen unter ???.

Wir empfehlen zwar, die AWS IoT Geräte-SDKs für die Verbindung zu verwenden AWS IoT, diese sind jedoch nicht erforderlich. Wenn Sie die AWS IoT Geräte-SDKs jedoch nicht verwenden, müssen

Sie für die erforderliche Verbindungs- und Kommunikationssicherheit sorgen. Clients müssen auch die Server Name Indication (SNI)-TLS-Erweiterung in der Verbindungsanforderung senden. Verbindungsversuche, die das SNI nicht enthalten, werden abgelehnt. Weitere Informationen finden Sie unter Transportsicherheit unter AWS IoT. Clients, die IAM-Benutzer und AWS Anmeldeinformationen zur Authentifizierung von Clients verwenden, müssen die korrekte Signature Version 4-Authentifizierung bereitstellen.

Nachdem Ihre Kunden verbunden sind, können Sie ihre MQTT-Clientverbindungen mithilfe der Konsole oder der APIs überwachen und verwalten. Weitere Informationen finden Sie unter the section called “MQTT-Verbindungsverwaltung”.

In diesem Thema:

- Herstellen einer Verbindung mit MQTT mithilfe der AWS IoT Geräte-SDKs
- MQTT QoS-(Quality of Service)-Optionen
- Persistente MQTT-Sitzungen
- Beibehaltene MQTT-Meldungen
- MQTT-Meldungen des letzten Willens und Testaments (LWT)
- ConnectAttributes verwenden
- Von MQTT 5 unterstützte Funktionen
- MQTT 5 Eigenschaften
- MQTT-Ursachencodes
- AWS IoT Unterschiede zu den MQTT-Spezifikationen
- MQTT-Verbindungsverwaltung

Herstellen einer Verbindung mit MQTT mithilfe der AWS IoT Geräte-SDKs

Dieser Abschnitt enthält Links zu den AWS IoT Geräte-SDKs und zum Quellcode von Beispielprogrammen, die veranschaulichen, wie ein Gerät angeschlossen wird. AWS IoT Die hier verlinkten Beispiel-Apps zeigen, wie Sie AWS IoT mithilfe des MQTT-Protokolls und MQTT über WSS eine Verbindung herstellen.

> **Note**
>
> Die AWS IoT Device SDKs haben einen MQTT 5-Client veröffentlicht.

C++

Verwenden des AWS IoT C++ Device SDK zum Verbinden von Geräten

- Quellcode einer App, als Beispiel für eine MQTT-Verbindung in C++
- AWS IoT Geräte-SDK für C++ v2 an GitHub

Python

Verwenden des AWS IoT Geräte-SDK für Python zum Verbinden von Geräten

- Quellcode einer App, als Beispiel für eine MQTT-Verbindung in Python
- AWS IoT Geräte-SDK v2 für Python an GitHub

JavaScript

Verwenden Sie das AWS IoT Geräte-SDK für JavaScript , um Geräte zu verbinden

- Quellcode einer Beispiel-App, die ein Beispiel für eine MQTT-Verbindung zeigt in JavaScript
- AWS IoT Geräte-SDK für JavaScript Version 2 an GitHub

Java

Verwenden des AWS IoT Geräte-SDK für Java zum Verbinden von Geräten

> **Note**
>
> Das AWS IoT Geräte-SDK für Java v2 unterstützt jetzt die Android-Entwicklung. Weitere Informationen finden Sie unter AWS IoT Geräte-SDK für Android.

- Quellcode einer App, die ein Beispiel für eine MQTT-Verbindung in Java zeigt
- AWS IoT Geräte-SDK für Java v2 auf GitHub

Embedded C

Verwenden des AWS IoT Geräte-SDK für Embedded C zum Verbinden von Geräten

> **Important**
>
> Dieses SDK ist für die Verwendung durch erfahrene Entwickler eingebetteter Software vorgesehen.

- Quellcode einer App, die ein Beispiel für eine MQTT-Verbindung in Embedded C zeigt
- AWS IoT Geräte-SDK für Embedded C an GitHub

MQTT QoS-(Quality of Service)-Optionen

AWS IoT und die AWS IoT Geräte-SDKs unterstützen die MQTT-Qualitätsstufen (QoS) und. `01` Das MQTT-Protokoll definiert eine dritte QoS-Ebene, Stufe2, unterstützt sie jedoch nicht AWS IoT . Nur das MQTT-Protokoll unterstützt die QoS-Funktion. HTTPS unterstützt QoS, indem es einen Abfragezeichenfolge-Parameter `?qos=qos` übergibt, sein Wert kann 0 oder 1 sein.

Die Tabelle beschreibt, wie sich jede QoS-Stufe auf Meldungen auswirkt, die an und von Message Broker veröffentlicht werden.

Mit einem QoS-Level von ...	Die Meldung ist …	Kommentare
QoS Stufe 0	Null oder mehr Mal gesendet	Diese Stufe sollte für Meldungen verwendet werden, die über zuverlässige Kommunikationsverbindungen gesendet werden oder die problemlos übersehen werden können.
QoS Stufe 1	Mindestens einmal gesendet und dann wiederholt, bis eine PUBACK Antwort eingeht.	Die Meldung gilt erst dann als vollständig, wenn der Absender eine PUBACK Antwort erhält, die auf eine erfolgreiche Zustellung hinweist.

Persistente MQTT-Sitzungen

Persistente Sitzungen speichern Abonnements und Meldungen eines Kunden mit einer Quality of Service (QoS) von 1, sofern diese nicht vom Kunden bestätigt wurden. Wenn das Gerät wieder eine Verbindung zu einer dauerhaften Sitzung herstellt, wird die Sitzung wieder aufgenommen, Abonnements werden wieder hergestellt und unbestätigte abonnierte Meldungen, die vor der Wiederverbindung empfangen und gespeichert wurden, werden an den Client gesendet.

Die Verarbeitung der gespeicherten Nachrichten wird in CloudWatch Metriken und Protokollen aufgezeichnet. CloudWatch Hinweise zu den Eingaben, in die geschrieben wurde, CloudWatch und zu den CloudWatch Protokollen finden Sie unter Message Broker-Metriken undProtokolleintrag in der Warteschlange.

Erstellen einer persistenten Sitzung

Erstellen Sie in MQTT 3 eine persistente MQTT-Sitzung, indem Sie eine `CONNECT`Meldung versenden und das `cleanSession` Flag auf `0` festlegen. Wenn für den Client, der die `CONNECT`-Meldung versendet, keine Sitzung vorhanden ist, wird eine neue persistente Sitzung erstellt. Wenn für den Client bereits eine Sitzung existiert, nimmt der Client die bestehende Sitzung wieder auf. Um eine saubere Sitzung zu erstellen, senden Sie eine `CONNECT` Meldung und setzen das `cleanSession` Flag auf 1. Der Broker speichert dann keinen Sitzungsstatus, wenn der Client die Verbindung trennt.

In MQTT 5 behandeln Sie persistente Sitzungen, indem Sie das `Clean Start` Flag und `Session Expiry Interval` festlegen. Clean Start steuert den Beginn der Verbindungssitzung und das Ende der vorherigen Sitzung. Wenn Sie `Clean Start` = 1 setzen, wird eine neue Sitzung erstellt und eine vorherige Sitzung wird beendet, falls sie existiert. Wenn Sie `Clean Start` = `0` setzen, nimmt die Verbindungssitzung eine vorherige Sitzung wieder auf, falls sie existiert. Das Sitzungsablaufintervall bestimmt das Ende der Verbindungssitzung. Das Sitzungsablaufintervall gibt die Zeit in Sekunden (4-Byte-Ganzzahl) an, für die eine Sitzung nach der Trennung bestehen bleibt. Einstellung `Session Expiry interval` = `0` bewirkt, dass die Sitzung sofort nach der Trennung beendet wird. Wenn das Sitzungsablaufintervall nicht in der CONNECT-Meldung angegeben ist, ist der Standardwert 0.

MQTT 5 Clean Start und Ablauf der Sitzung

Eigenschaftenwert	Description
`Clean Start`= 1	Erzeugt eine neue Sitzung und beendet eine vorherige Sitzung, falls eine existiert.

Eigenschaftenwert	Description
`Clean Start= 0`	Nimmt eine Sitzung wieder auf, falls eine vorherige Sitzung existiert .
`Session Expiry Interval> 0`	Hält eine Sitzung aufrecht.
`Session Expiry interval= 0`	Behält eine Sitzung nicht bei.

Wenn Sie in MQTT 5 `Clean Start` = `1` und `Session Expiry Interval` = `0` setzen, entspricht dies einer sauberen MQTT-3-Sitzung. Wenn Sie `Clean Start` = `0` und `Session Expiry Interval` >`0` setzen, entspricht dies einer persistenten MQTT-3-Sitzung.

> **Note**
>
> MQTT-übergreifende (MQTT 3 und MQTT 5) persistente Sitzungen werden nicht unterstützt. Eine persistente MQTT 3-Sitzung kann nicht als MQTT 5-Sitzung wieder aufgenommen werden und umgekehrt.

Operationen während einer persistenten Sitzung

Clients müssen das `sessionPresent`-Attribut in der Connection Acknowledged (`CONNACK`)-Meldung nutzen, um festzustellen, ob eine persistente Sitzung vorhanden ist. Wenn `sessionPresent` auf 1 gesetzt ist, liegt eine persistente Sitzung vor und alle gespeicherten Meldungen für den Client werden an den Client zugestellt, nachdem der Client `CONNACK` empfangen hat, wie unter Meldungsverkehr nach Wiederverbindung mit einer persistenten Sitzung beschrieben. Wenn `sessionPresent` auf 1 gesetzt ist, muss der Client kein erneutes Abonnement abschließen. Wenn `sessionPresent` auf `0` gesetzt ist, ist keine persistente Sitzung vorhanden, und der Client muss die Themenfilter erneut abonnieren.

Nachdem der Client der persistenten Sitzung beigetreten ist, kann er weiterhin Meldungen veröffentlichen und Themenfilter ohne zusätzliche Flags für jede Maßnahme abonnieren.

Meldungsverkehr nach Wiederverbindung zu einer persistenten Sitzung

Eine persistente Sitzung stellt eine fortlaufende Verbindung zwischen einem Client und einem MQTT-Message Broker dar. Wenn ein Client eine Verbindung mit dem Message Broker über eine persistente Sitzung herstellt, speichert der Message Broker alle Abonnements, die der Client während der Verbindung erstellt. Wenn der Client getrennt wird, speichert der Message Broker unbestätigte QoS 1-Meldungen und neue QoS 1-Meldungen, die zu Themen veröffentlicht wurden, die der Client abonniert hat. Meldungen werden gemäß dem Kontolimit gespeichert. Meldungen, die das Limit überschreiten werden gelöscht. Weitere Informationen zu persistenten Nachrichtenlimits finden Sie unter AWS IoT Core Endpunkte und Kontingente. Wenn der Client die Verbindung zur persistenten Sitzung wiederherstellt, werden alle Abonnements reaktiviert und alle gespeicherten Meldungen werden bei einer maximalen Rate von 10 Meldungen pro Sekunde an den Client gesendet. Wenn in MQTT 5 eine ausgehende QoS 1 mit dem Nachrichtenablaufintervall abläuft, während ein Client offline ist, erhält der Client nach Wiederaufnahme der Verbindung die abgelaufene Nachricht nicht.

Nach der Wiederverbindung werden die gespeicherten Meldungen an den Client gesendet, und zwar mit einer Geschwindigkeit, die auf 10 gespeicherte Meldungen pro Sekunde begrenzt ist, zusammen mit dem aktuellen Meldungsverkehr, bis das `Publish requests per second per connection`-Limit erreicht ist. Da die Zustellungsrate der gespeicherten Meldungen begrenzt ist, dauert es mehrere Sekunden, bis alle gespeicherten Meldungen zugestellt werden, wenn in einer Sitzung nach der Wiederverbindung mehr als 10 gespeicherte Meldungen zugestellt werden müssen.

Bei gemeinsam genutzten Abonnenten werden Nachrichten in die Warteschlange gestellt, wenn mindestens ein Abonnent einer Gruppe eine persistente Sitzung verwendet und kein Abonnent online ist, um die QoS 1-Nachricht zu empfangen. Das Entfernen von Nachrichten aus der Warteschlange erfolgt mit einer maximalen Geschwindigkeit von 20 Nachrichten pro Sekunde pro aktivem Abonnenten in einer Gruppe. Weitere Informationen finden Sie unter Nachrichtenwarteschlangen für gemeinsame Abonnements.

Eine persistente Sitzung wird beendet.

Dauerhafte Sitzungen können wie folgt enden:

- Die Ablaufzeit der persistenten Sitzung ist abgelaufen. Der Timer für den Ablauf einer persistenten Sitzung wird gestartet, wenn der Message Broker feststellt, dass ein Client die Verbindung getrennt hat, entweder weil der Client die Verbindung getrennt hat oder weil die Verbindung das Timeout überschritten hat.
- Der Client sendet eine `CONNECT` Nachricht, die das `cleanSession` Flag auf 1 setzt.

- Sie trennen den Client manuell und löschen die Sitzung mithilfe der Konsole oder `DeleteConnection` API. Weitere Informationen finden Sie unter the section called “MQTT-Verbindungsverwaltung”.

In MQTT 3 ist der Standardwert für die Ablaufzeit persistenter Sitzungen eine Stunde, und dies gilt für alle Sitzungen im Konto.

In MQTT 5 können Sie das Sitzungsablaufintervall für jede Sitzung in den Paketen CONNECT und DISCONNECT festlegen.

Für das Sitzungsablaufintervall für das DISCONNECT-Paket:

- Wenn die aktuelle Sitzung ein Sitzungsablaufintervall von 0 hat, können Sie das Sitzungsablaufintervall für das DISCONNECT-Paket nicht auf einen Wert über 0 setzen.
- Wenn die aktuelle Sitzung ein Sitzungsablaufintervall von mehr als 0 hat und Sie das Sitzungsablaufintervall im DISCONNECT-Paket auf 0 setzen, wird die Sitzung auf DISCONNECT beendet.
- Andernfalls aktualisiert das Sitzungsablaufintervall für das DISCONNECT-Paket das Sitzungsablaufintervall der aktuellen Sitzung.

> **Note**
>
> Die gespeicherten Meldungen, die darauf warten, am Ende einer Sitzung an den Client gesendet zu werden, werden verworfen. Sie werden jedoch weiterhin zum Standardnachrichtentarif in Rechnung gestellt, obwohl sie nicht gesendet werden konnten. Weitere Informationen zu Preisen erhalten Sie unter AWS IoT Core Preise. Sie können das Ablaufzeitintervall konfigurieren.

Wiederverbindung nach Ablauf einer persistenten Sitzung

Wenn ein Client vor Ablauf nicht wieder eine Verbindung zu seiner persistenten Sitzung herstellt, wird die Sitzung beendet und die gespeicherten Meldungen werden verworfen. Wenn ein Client nach Ablauf der Sitzung wieder eine Verbindung herstellt und das `cleanSession`-Flag auf `0` setzt, erstellt der Dienst eine neue persistente Sitzung. Alle Abonnements oder Meldungen aus der vorherigen Sitzung sind für diese Sitzung nicht verfügbar, da sie beim Ablauf der vorherigen Sitzung verworfen wurden.

Gebühren für Meldungen während einer dauerhaften Sitzung

Nachrichten werden Ihnen in Rechnung gestellt AWS-Konto , wenn der Message Broker eine Nachricht an einen Client oder eine persistente Offline-Sitzung sendet. Wenn ein Offline-Gerät mit einer dauerhaften Sitzung erneut eine Verbindung herstellt und seine Sitzung wieder aufnimmt, werden die gespeicherten Meldungen an das Gerät übermittelt und Ihrem Konto erneut belastet. Weitere Informationen zu den Preisen für Meldungen finden Sie unter AWS IoT Core Preise – Meldungsübermittlung.

Die standardmäßige Ablaufzeit einer persistenten Sitzung von einer Stunde kann erhöht werden, indem das standardmäßige Verfahren zum Erhöhen von Limits verwendet wird. Beachten Sie, dass durch eine Verlängerung der Sitzungsablaufzeit Ihre Meldungsgebühren steigen können, da durch die zusätzliche Zeit mehr Meldungen für das Offline-Gerät gespeichert werden können und diese zusätzlichen Meldungen Ihrem Konto zum Standardnachrichtentarif in Rechnung gestellt würden. Bei der Ablaufzeit der Sitzung handelt es sich um einen ungefähren Wert, und eine Sitzung kann bis zu 30 Minuten länger als das Kontolimit andauern. Eine Sitzung darf das Kontolimit jedoch nicht unterschreiten. Weitere Informationen zu Sitzungslimits finden Sie unter AWS -Service Quotas.

Beibehaltene MQTT-Meldungen

AWS IoT Core unterstützt das im MQTT-Protokoll beschriebene `RETAIN` Flag. Wenn ein Client einer MQTT-Nachricht, die er veröffentlicht, das `RETAIN` Kennzeichen setzt, AWS IoT Core speichert er die Nachricht. Sie kann dann an neue Abonnenten gesendet, durch Aufrufen des `GetRetainedMessage`Vorgangs abgerufen und in der AWS IoT Konsole angezeigt werden.

Beispiele für die Verwendung von gespeicherten MQTT-Meldungen

- Als erste Konfigurationsnachricht

  In MQTT gespeicherte Meldungen werden an einen Client gesendet, nachdem der Client ein Thema abonniert hat. Wenn Sie möchten, dass alle Clients, die ein Thema abonnieren, die in MQTT gespeicherte Nachricht direkt nach ihrem Abonnement erhalten, können Sie eine Konfigurationsnachricht mit gesetztem `RETAIN` Flag veröffentlichen. Abonnierende Clients erhalten außerdem Updates für diese Konfiguration, wenn eine neue Konfigurationsnachricht veröffentlicht wird.
- Als letzte bekannte Zustandsmeldung

  Geräte können Nachrichten mit dem aktuellen Status `RETAIN` kennzeichnen, sodass sie gespeichert AWS IoT Core werden. Wenn Anwendungen eine Verbindung herstellen oder erneut

eine Verbindung herstellen, können sie dieses Thema abonnieren und erhalten den zuletzt gemeldeten Status direkt nach dem Abonnieren des Themas der gespeicherten Nachricht. Auf diese Weise können sie vermeiden, dass sie bis zur nächsten Meldung vom Gerät warten müssen, um den aktuellen Status zu sehen.

In diesem Abschnitt:

- Allgemeine Aufgaben mit gespeicherten MQTT-Nachrichten in AWS IoT Core
- Abrechnung und gespeicherte Meldungen
- Vergleich von beibehaltenen MQTT-Meldungen und persistenten MQTT-Sitzungen
- MQTT hat Nachrichten aufbewahrt und AWS IoT Geräteschatten

Allgemeine Aufgaben mit gespeicherten MQTT-Nachrichten in AWS IoT Core

AWS IoT Core speichert MQTT-Nachrichten mit gesetztem `RETAIN` Flag. Diese gespeicherten Meldungen werden als normale MQTT-Meldung an alle Clients gesendet, die das Thema abonniert haben, und sie werden auch gespeichert, um an neue Abonnenten des Themas gesendet zu werden.

Archivierte MQTT-Meldungen erfordern spezifische Richtlinienaktionen, um Clients den Zugriff auf sie zu autorisieren. Beispiele für die Verwendung von Richtlinien für gespeicherte Meldungen finden Sie unter Beispielrichtlinien für beibehaltene Nachrichten.

In diesem Abschnitt werden allgemeine Vorgänge beschrieben, bei denen gespeicherte Meldungen verwendet werden.

- Erstellen einer beibehaltenen Nachricht

  Der Client bestimmt, ob eine Meldung beibehalten wird, wenn er eine MQTT-Meldung veröffentlicht. Clients können das `RETAIN` Kennzeichen setzen, wenn sie eine Nachricht mithilfe eines Geräte-SDK veröffentlichen. Anwendungen und Dienste können das `RETAIN` Kennzeichen setzen, wenn sie die `Publish` Aktion verwenden, um eine MQTT-Nachricht zu veröffentlichen.

  Pro Themenname wird nur eine Meldung beibehalten. Eine neue Meldung mit gesetztem RETAIN-Flag, die zu einem Thema veröffentlicht wurde, ersetzt alle vorhandenen beibehaltenen Meldungen, die zuvor an das Thema gesendet wurden.

> **Note**
>
> Sie können nicht in einem reservierten Thema veröffentlichen, wenn das `RETAIN` Flag gesetzt ist.

- Abonnieren eines beibehaltenen Meldungsthemas

  Kunden abonnieren Themen für beibehaltene Meldungen wie jedes andere MQTT-Meldungsthema. Bei einbehaltenen Nachrichten, die Sie erhalten, wenn Sie ein Thema für eine beibehaltene Nachricht abonniert haben, ist das `RETAIN` Kennzeichen gesetzt.

  Zurückbehaltene Nachrichten werden gelöscht AWS IoT Core , wenn ein Client eine beibehaltene Nachricht mit einer Nachrichtennutzlast von 0 Byte im Thema der beibehaltenen Nachricht veröffentlicht. Clients, die das beibehaltene Meldungsthema abonniert haben, erhalten auch die 0-Byte-Nachricht.

  Wenn Sie einen Platzhalter-Themenfilter abonnieren, der ein beibehaltenes Nachrichtenthema enthält, kann der Client nachfolgende Nachrichten zu diesem Thema erhalten. Die beibehaltene Nachricht wird jedoch nicht zum Zeitpunkt des Abonnements zugestellt.

  > **Note**
  >
  > Um nach dem Abonnement eine beibehaltene Nachricht zu erhalten, muss der Themenfilter in der Abonnementanforderung exakt dem Thema der beibehaltenen Nachricht entsprechen.

  Bei einbehaltenen Nachrichten, die nach dem Abonnieren eines Themas einer beibehaltenen Nachricht eingehen, ist das `RETAIN` Kennzeichen gesetzt. Bei gespeicherten Meldungen, die nach dem Abonnement von einem abonnierten Client empfangen werden, ist dies nicht der Fall.

- Eine beibehaltene Meldung wird abgerufen.

  Aufbewahrte Meldungen werden automatisch an Clients zugestellt, wenn sie das Thema mit der gespeicherten Meldung abonnieren. Damit ein Kunde die beibehaltene Meldung nach dem Abonnement erhalten kann, muss er den genauen Themennamen der gespeicherten Meldung abonnieren. Wenn Sie einen Wildcard-Themenfilter abonnieren, der ein aufbewahrtes Meldungsthema enthält, kann der Client nachfolgende Meldungen empfangen, die zum Thema

der beibehaltenen Meldung veröffentlicht wurden, aber die beibehaltene Meldung wird beim Abonnement nicht zugestellt.

Dienste und Apps können gespeicherte Meldungen auflisten und abrufen, indem sie `ListRetainedMessages`und `GetRetainedMessage`aufrufen.

Ein Client wird nicht daran gehindert, Nachrichten zu einem beibehaltenen Nachrichtenthema zu veröffentlichen, ohne das *`RETAIN`* Kennzeichen zu setzen. Dies kann zu unerwarteten Ergebnissen führen, z. B. dass die gespeicherte Meldung nicht mit der Meldung übereinstimmt, die durch das Abonnieren des Themas empfangen wurde.

Wenn bei MQTT 5 für eine beibehaltene Meldung das Meldungsablaufintervall festgelegt ist und die beibehaltene Meldung abläuft, erhält ein neuer Abonnent, der dieses Thema abonniert, die beibehaltene Meldung bei erfolgreichem Abonnement nicht.

- Themen der beibehaltenen Meldungen auflisten

  Sie können die gespeicherten Meldungen auflisten, indem Sie `ListRetainedMessages` telefonisch abrufen, und die gespeicherten Meldungen können in der AWS IoT Konsole eingesehen werden.

- Details zu gespeicherten Meldungen abrufen

  Sie können die gespeicherten Meldungsdetails telefonisch unter `GetRetainedMessage` abrufen, und sie können in der AWS IoT Konsole angezeigt werden.

- Beibehaltung einer Willensnachricht

  MQTT-Will-Meldungen, die erstellt werden, wenn ein Gerät eine Verbindung herstellt, können beibehalten werden, indem das `Will Retain` Flag im `Connect Flag bits` Feld gesetzt wird.

- Löschen einer beibehaltenen Nachricht

  Geräte, Anwendungen und Dienste können eine beibehaltene Nachricht löschen, indem sie eine Nachricht mit gesetztem `RETAIN` Flag und einer leeren Nachrichtennutzlast (0 Byte) unter dem Themennamen der zu löschenden beibehaltenen Nachricht veröffentlichen. Solche Nachrichten löschen die beibehaltene Nachricht von und werden an Kunden gesendet AWS IoT Core, die das Thema abonniert haben, aber sie werden nicht gespeichert. AWS IoT Core

  Beibehaltene Meldungen können auch interaktiv gelöscht werden, indem Sie in der AWS IoT Konsole auf die gespeicherte Meldung zugreifen. Archivierte Meldungen, die mithilfe der AWS

IoT Konsole gelöscht werden, senden auch eine 0-Byte-Meldung an Clients, die das Thema der gespeicherten Meldung abonniert haben.

Aufbewahrte Meldungen können nicht wiederhergestellt werden, nachdem sie gelöscht wurden. Ein Client müsste eine neue beibehaltene Meldung veröffentlichen, um die gelöschte Meldung zu ersetzen.

- Debuggen und Problembehandlung bei gespeicherten Meldungen

  Die AWS IoT Konsole bietet mehrere Tools, mit denen Sie Fehler bei gespeicherten Meldungen beheben können:

  - Das Tool Beibehaltene Meldungen angezeigten

    Die Seite „Zurückbehaltene Meldungen“ in der AWS IoT Konsole enthält eine paginierte Liste der gespeicherten Meldungen, die von Ihrem Konto in der aktuellen Region gespeichert wurden. Auf dieser Seite können Sie:

    - Sehen Sie sich die Details jeder gespeicherten Meldung an, z. B. die Meldungsnutzlast, QoS und den Zeitpunkt, zu dem sie empfangen wurde.
    - Aktualisieren Sie den Inhalt einer gespeicherten Nachricht.
    - Löscht eine beibehaltene Nachricht.
  - Das Tool MQTT-Testclient

    Auf der MQTT-Testclient-Seite in der AWS IoT Konsole können MQTT-Themen abonniert und veröffentlicht werden. Mit der Veröffentlichungsoption können Sie das RETAIN-Flag für die von Ihnen veröffentlichten Meldungen setzen, um zu simulieren, wie sich Ihre Geräte verhalten könnten. Sie können den MQTT-Testclient auch verwenden, um Nachrichten von verbundenen Clients zu überwachen, die Sie über die Client-Verbindungsschnittstelle verwalten. Weitere Hinweise zur Verwaltung von Clientverbindungen finden Sie unterthe section called “MQTT-Verbindungsverwaltung”.

Einige unerwartete Ergebnisse könnten das Ergebnis dieser Aspekte der Implementierung von aufbewahrten Nachrichten sein AWS IoT Core.

- Beibehaltene Meldungslimits

  Wenn ein Konto die maximale Anzahl an aufbewahrten Nachrichten gespeichert hat, wird eine gedrosselte Antwort auf Nachrichten AWS IoT Core zurückgegeben, bei denen RETAIN gesetzt ist und Nutzlasten größer als 0 Byte sind, bis einige der aufbewahrten Nachrichten gelöscht werden und die Anzahl der aufbewahrten Nachrichten unter den Grenzwert fällt.

- Beibehaltene Reihenfolge der Meldungszustellung

  Die Reihenfolge zwischen beibehaltener und abonnierter Meldungszustellung kann nicht garantiert werden.

Abrechnung und gespeicherte Meldungen

Beim Veröffentlichen von Nachrichten mit `RETAIN` gesetztem Kennzeichen von einem Client aus, über die AWS IoT Konsole oder per Anruf `Publish` fallen zusätzliche Messaging-Gebühren an, die unter Preisgestaltung — Messaging beschrieben werden. AWS IoT Core

Für das Abrufen von zurückgehaltenen Nachrichten durch einen Client, über die AWS IoT Konsole oder durch einen Anruf `GetRetainedMessage` fallen zusätzlich zu den normalen API-Nutzungsgebühren Messaging-Gebühren an. Die zusätzlichen Gebühren sind unter AWS IoT Core Preise – Meldungsübermittlung beschrieben.

Für Will-Meldungen, die veröffentlicht werden, wenn die Verbindung zu einem Gerät unerwartet unterbrochen wird, fallen Gebühren für die Meldungsübermittlung an, die unter AWS IoT Core Preise – Meldungsübermittlung beschrieben sind.

Weitere Informationen zu den Messaging-Kosten finden Sie unter AWS IoT Core Preise – Meldungsübermittlung.

Vergleich von beibehaltenen MQTT-Meldungen und persistenten MQTT-Sitzungen

Aufbewahrte Meldungen und persistente Sitzungen sind Standardfunktionen von MQTT, die es Geräten ermöglichen, Meldungen zu empfangen, die veröffentlicht wurden, während sie offline waren. Aufbewahrte Meldungen können aus persistenten Sitzungen heraus veröffentlicht werden. In diesem Abschnitt werden die wichtigsten Aspekte dieser Funktionen und ihr Zusammenspiel beschrieben.

	Beibehaltene Meldungen	Persistente MQTT-Sitzungen
Schlüssel-Features	Aufbewahrte Meldungen können verwendet werden, um große Gruppen von Geräten zu konfigurieren oder zu benachrichtigen, nachdem sie eine Verbindung hergestellt haben.	Dauerhafte Sitzungen sind nützlich für Geräte, deren Konnektivität unterbrochen ist und bei denen möglicherweise mehrere wichtige Meldungen übersehen werden.

	Beibehaltene Meldungen	Persistente MQTT-Sitzungen
	Aufbewahrte Meldungen können auch verwendet werden, wenn Geräte nach einer erneuten Verbindung nur die zuletzt zu einem Thema veröffentlichte Meldung empfangen sollen.	Geräte können sich mit einer dauerhaften Sitzung verbinden, um Meldungen zu empfangen, die gesendet werden, während sie offline sind.
Beispiele	Gespeicherte Meldungen können Geräten Konfigurationsinformationen über ihre Umgebung geben, wenn sie online gehen. Die Erstkonfiguration könnte eine Liste anderer Meldungsthemen enthalten, die das Gerät abonnieren sollte, oder Informationen darüber, wie es seine lokale Zeitzone konfigurieren sollte.	Geräte, die über ein Mobilfunknetz mit intermittierender Konnektivität eine Verbindung herstellen, könnten persistente Sitzungen verwenden, um zu verhindern, dass wichtige Meldungen verpasst werden, die gesendet werden, wenn ein Gerät nicht mit Netzempfang versorgt ist oder sein Mobilfunknetz ausgeschaltet werden muss.
Meldungen, die beim ersten Abonnement eines Themas eingegangen sind	Nach dem Abonnieren eines Themas mit einer beibehaltenen Meldung wird die neueste beibehaltene Meldung empfangen.	Nach dem Abonnieren eines Themas ohne beibehaltene Meldung wird keine Meldung empfangen, bis eine Meldung zu dem Thema veröffentlicht wird.
Abonnierte Themen nach dem erneuten Verbindungsaufbau	Ohne eine dauerhafte Sitzung muss der Client nach der Wiederverbindung Themen abonnieren.	Abonnierte Themen werden nach der Wiederverbindung wiederhergestellt.

	Beibehaltene Meldungen	Persistente MQTT-Sitzungen
Nach der Wiederverbindung empfangene Meldungen	Nach dem Abonnieren eines Themas mit einer beibehalt enen Meldung wird die neueste beibehaltene Meldung empfangen.	Alle Meldungen, die mit QOS = 1 veröffentlicht und mit QOS =1 abonniert wurden, während das Gerät getrennt wurde, werden gesendet, nachdem das Gerät wieder eine Verbindung hergestellt hat.
Data/session Ablauf	In MQTT 3 laufen gespeiche rte Meldungen nicht ab. Sie werden gespeichert, bis sie ersetzt oder gelöscht werden. In MQTT 5 laufen aufbewahr te Nachrichten nach dem von Ihnen festgelegten Nachricht enablaufintervall ab. Weitere Informationen finden Sie unter Message templates (Meldungs vorlagen).	Persistente Sitzungen laufen ab, wenn der Client innerhalb des Timeout-Zeitraums keine erneute Verbindun g herstellt. Nach Ablauf einer dauerhaften Sitzung werden die Abonnements und gespeicherten Meldungen des Clients gelöscht, die mit einem QOS = 1 veröffentlicht und mit einem QOS =1 abonniert wurden, während das Gerät getrennt wurde. Abgelaufe ne Meldungen werden nicht zugestellt. Weitere Hinweise zum Ablauf von Sitzungen mit persistenten Sitzungen finden Sie unter the section called “Persistente MQTT-Sitz ungen”.

Informationen zu persistentem Speicher siehe the section called “Persistente MQTT-Sitzungen”.

Bei Retained Messages bestimmt der Publishing-Client, ob eine Meldung aufbewahrt und an ein Gerät gesendet werden soll, nachdem das Gerät eine Verbindung hergestellt hat, unabhängig davon, ob es eine vorherige Sitzung hatte oder nicht. Die Entscheidung, ob eine Nachricht

gespeichert werden soll, wird vom Herausgeber getroffen, und die gespeicherte Nachricht wird an alle aktuellen und zukünftigen Clients zugestellt, die QoS 0- oder QoS 1-Abonnements abonnieren. Bei gespeicherten Meldungen wird jeweils nur eine Meldung zu einem bestimmten Thema gespeichert.

Wenn ein Konto die maximale Anzahl an gespeicherten Meldungen gespeichert hat, wird AWS IoT Core eine gedrosselte Antwort auf Meldungen zurückgeben, bei denen RETAIN aktiviert war, und Nutzlasten von mehr als 0 Byte, bis einige beibehaltene Meldungen gelöscht wurden und die Anzahl der gespeicherten Meldungen unter den Grenzwert fällt.

MQTT hat Nachrichten aufbewahrt und AWS IoT Geräteschatten

Sowohl bei gespeicherten Meldungen als auch bei Geräteschatten werden Daten von einem Gerät gespeichert, sie verhalten sich jedoch unterschiedlich und dienen unterschiedlichen Zwecken. In diesem Abschnitt werden ihre Ähnlichkeiten und Unterschiede beschrieben.

	Beibehaltene Meldungen	Geräteschatten
Die Meldungsnutzlast hat eine vordefinierte Struktur oder ein vordefiniertes Schema.	Wie in der Implementierung definiert. MQTT spezifizi ert keine Struktur oder kein Schema für seine Meldungsn utzlast.	AWS IoT unterstützt eine bestimmte Datenstruktur.
Durch die Aktualisierung der Meldungsnutzlast werden Ereignismeldungen generiert.	Durch das Veröffentlichen einer gespeicherten Meldung wird die Meldung an abonniert e Clients gesendet, es werden jedoch keine zusätzlichen Aktualisierungsnachrichten generiert.	Beim Aktualisieren eines Geräteschattens werden Aktualisierungsmeldungen angezeigt, die die Änderung beschreiben.
Meldungsaktualisierungen sind nummeriert.	Aufbewahrte Meldungen werden nicht automatisch nummeriert.	Geräteschatten-Dok umente haben automatis che Versionsnummern und Zeitstempel.
Die Meldungen-Payload ist an eine Ding-Ressource angehängt.	Beibehaltene Meldungen sind nicht an eine Ding-Ressource angehängt.	Geräteschatten sind an eine Ding-Ressource angehängt.

	Beibehaltene Meldungen	Geräteschatten
Einzelne Elemente der Meldungs-Payload werden aktualisiert.	Einzelne Elemente der Meldung können nicht geändert werden, ohne die gesamte Meldungsnutzlast zu aktualisieren.	Einzelne Elemente eines Geräteschatten-Dokuments können aktualisiert werden, ohne dass das gesamte Geräteschatten-Dokument aktualisiert werden muss.
Der Kunde erhält Meldungsd aten bei Abschluss des Abonnements.	Der Client erhält automatisch eine beibehaltene Nachricht , nachdem er ein Thema mit einer beibehaltenen Nachricht abonniert hat.	Clients können Gerätesch atten-Updates abonnieren, müssen den aktuellen Status jedoch bewusst anfordern.
Indizierung und Durchsuch barkeit	Zurückbehaltene Meldungen werden nicht für die Suche indexiert.	Flottenindizierung indexiert Geräteschatten-Daten für die Suche und Aggregation.

MQTT-Meldungen des letzten Willens und Testaments (LWT)

Last Will and Testament (LWT) ist ein Feature in MQTT. Mit LWT können Kunden eine Meldung angeben, die der Broker zu einem vom Kunden definierten Thema veröffentlicht und an alle Clients sendet, die das Thema abonniert haben, wenn eine uninitiierte Verbindungsunterbrechung auftritt. Die von den Clients angegebene Meldung wird als LWT-Meldung oder Will-Meldung bezeichnet, und das von den Clients definierte Thema wird Will-Thema genannt. Sie können eine LWT-Meldung angeben, wenn ein Gerät eine Verbindung zum Broker herstellt. Diese Meldungen können beibehalten werden, indem während der Verbindung die `Will Retain` Markierung im `Connect Flag bits` Feld gesetzt wird. Wenn die `Will Retain` Markierung beispielsweise auf `1` gesetzt ist, wird eine Will-Meldung im Broker im zugehörigen Testament-Thema gespeichert. Weitere Informationen finden Sie unter Meldungsvorlagen.

Bei der Verwaltung von Client-Verbindungen können Sie steuern, ob LWT-Nachrichten veröffentlicht werden, wenn Sie die Verbindung zu einem Client trennen. Weitere Informationen finden Sie unter the section called “MQTT-Verbindungsverwaltung”.

Der Broker speichert die Will-Meldungen, bis es zu einer uninitiierten Verbindungsunterbrechung kommt. In diesem Fall veröffentlicht der Broker die Meldungen an alle Clients, die das Will-Thema

abonniert haben, um den Verbindungsabbruch zu melden. Wenn der Client die Verbindung zum Broker mit einer vom Client initiierten Verbindungsunterbrechung mithilfe der MQTT DISCONNECT-Meldung trennt, veröffentlicht der Broker die gespeicherten LWT-Meldungen nicht. In allen anderen Fällen werden die LWT-Meldungen versendet. Aufgrund der asynchronen Verarbeitung von Verbindungsabbrüchen kann nicht garantiert werden, dass LWT-Nachrichten bei der Wiederverbindung in der richtigen Reihenfolge versendet werden. Wir empfehlen, Lebenszyklusereignisse zu verwenden, um die Genauigkeit der Erkennung des Verbindungsstatus zu verbessern, da diese Ereignisse Attribute wie Zeitstempel und Versionsnummern bereitstellen, um Ereignisse zu verwalten, die nicht in der richtigen Reihenfolge liegen. Eine vollständige Liste der Trennungsszenarien, in denen der Broker die LWT-Nachrichten sendet, finden Sie unter Ereignisse. Connect/Disconnect

ConnectAttributes verwenden

`ConnectAttributes` ermöglichen es Ihnen, in Ihren IAM-Richtlinien anzugeben, welche Attribute Sie in Ihrer Connect-Meldung verwenden möchten, z. B. `PersistentConnect` und `LastWill` Mit `ConnectAttributes` können Sie Richtlinien erstellen, die Geräten standardmäßig keinen Zugriff auf neue Funktionen gewähren. Dies kann hilfreich sein, wenn ein Gerät kompromittiert wurde.

`connectAttributes` unterstützt die folgenden Funktionen:

`PersistentConnect`

Verwenden Sie die `PersistentConnect` Funktion, um alle Abonnements zu speichern, die der Client während der Verbindung abschließt, wenn die Verbindung zwischen dem Client und dem Broker unterbrochen wird.

`LastWill`

Verwenden Sie die `LastWill` Funktion, um eine Meldung an `LastWillTopic` zu veröffentlichen, wenn ein Client unerwartet die Verbindung trennt.

Standardmäßig hat Ihre Richtlinie eine nicht persistente Verbindung, und für diese Verbindung werden keine Attribute übergeben. Sie müssen in Ihrer IAM-Richtlinie eine persistente Verbindung angeben, wenn Sie eine haben möchten.

Wenn Sie Client-Verbindungen über die Konsole oder APIs verwalten, können Sie die Verbindungsattribute und die Sitzungskonfiguration für verbundene Clients einsehen. Weitere Informationen finden Sie unter the section called “MQTT-Verbindungsverwaltung”.

`ConnectAttributes`Beispiele finden Sie unter Beispiele für Connect-Richtlinien.

Von MQTT 5 unterstützte Funktionen

AWS IoT Core Die Unterstützung für MQTT 5 basiert auf der MQTT v5.0-Spezifikation mit einigen Unterschieden, wie unter beschrieben. the section called “AWS IoT Unterschiede zu den MQTT-Spezifikationen”

AWS IoT Core unterstützt die folgenden MQTT 5-Funktionen:

- Geteilte Abonnements
- Clean Start und Ablauf der Sitzung
- Ursachencode auf allen ACKs
- Aliasnamen zum Thema
- Nachrichtablauf
- Weitere Funktionen von MQTT 5

Geteilte Abonnements

AWS IoT Core unterstützt gemeinsame Abonnements sowohl für MQTT 3.1.1 als auch für MQTT 5. Gemeinsame Abonnements ermöglichen es mehreren Kunden, ein Abonnement für ein Thema gemeinsam zu nutzen, und nur ein Kunde erhält Nachrichten, die zu diesem Thema veröffentlicht werden, wobei eine zufällige Verteilung erfolgt. Durch gemeinsame Abonnements können die MQTT-Nachrichten effektiv auf mehrere Abonnenten verteilt werden. Nehmen wir zum Beispiel an, Sie haben 1.000 Geräte, die zum gleichen Thema veröffentlichen, und 10 Backend-Anwendungen, die diese Meldungen verarbeiten. In diesem Fall können die Backend-Anwendungen dasselbe Thema abonnieren, und jede von ihnen würde nach dem Zufallsprinzip Nachrichten empfangen, die von den Geräten zu dem gemeinsamen Thema veröffentlicht wurden. Auf diese Weise wird die Menge dieser Meldungen quasi „geteilt“. Gemeinsame Abonnements ermöglichen auch eine bessere Ausfallsicherheit. Wenn eine Back-End-Anwendung die Verbindung trennt, verteilt der Broker die Last auf die verbleibenden Abonnenten in der Gruppe. Wenn alle Abonnenten die Verbindung trennen, werden die Nachrichten in die Warteschlange gestellt.

Nachrichtenwarteschlangenfunktionen sind für gemeinsame Abonnements sowohl auf MQTT 3.1.1- als auch auf MQTT 5-Verbindungen verfügbar, um die Zuverlässigkeit der Nachrichtenübermittlung zu erhöhen.

Um gemeinsame Abonnements zu verwenden, abonnieren Kunden den Themenfilter eines gemeinsamen Abonnements wie folgt:

```
$share/{ShareName}/{TopicFilter}
```

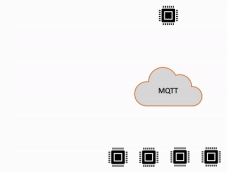
- `$share` ist eine wörtliche Zeichenfolge, die den Themenfilter eines geteilten Abonnements angibt, der mit `$share` beginnen muss.
- `{ShareName}` ist eine Zeichenfolge zur Angabe des gemeinsamen Namens, der von einer Gruppe von Abonnenten verwendet wird. Der Themenfilter eines gemeinsamen Abonnements muss das Zeichen „a“ `ShareName` und danach das `/` Zeichen enthalten. `{ShareName}` darf die folgenden Zeichen nicht enthalten: `/`,`+` oder`#`. Die maximale Größe für `{ShareName}` beträgt 128 UTF-8 Zeichen.
- `{TopicFilter}`folgt derselben https://docs.aws.amazon.com/iot/latest/developerguide/topics.html#topicfilters Themenfiltersyntax wie ein Abonnement, das nicht gemeinsam genutzt wird. Die maximale Größe für `{TopicFilter}` beträgt 256 UTF-8 Zeichen.
- Die beiden erforderlichen Schrägstriche (`/`) für `$share/{ShareName}/{TopicFilter}` sind nicht in den Grenzwerten „Maximale Anzahl von Schrägstrichen im Thema“ und „Themenfilter“ enthalten.

Abonnements, die dasselbe haben, `{ShareName}/{TopicFilter}` gehören derselben gemeinsamen Abonnementgruppe an. Sie können mehrere gemeinsame Abonnementgruppen erstellen und das Limit für gemeinsame Abonnements pro Gruppe nicht überschreiten. Weitere Informationen finden Sie unter AWS IoT Core -Endpunkte und -Kontingente in der AWS Allgemeinen Referenz.

In den folgenden Tabellen werden nicht geteilte Abonnements und geteilte Abonnements verglichen:

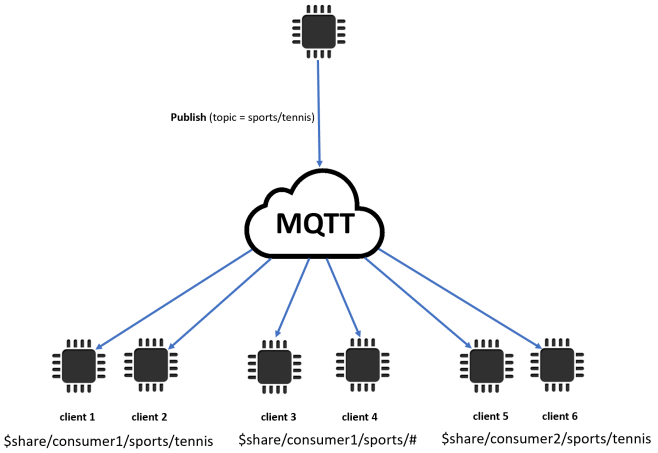
Abonnement	Description	Beispiele für Metrikfilter
Non-shared Abonnements	Jeder Client erstellt ein separates Abonnement für den Empfang der veröffentlichten Meldungen . Wenn eine Meldung zu einem Thema veröffentlicht wird, erhalten alle Abonnenten dieses Themas eine Kopie der Nachricht.	`sports/tennis` `sports/#`
Geteilte Abonnements	Mehrere Kunden können sich ein Abonnement für ein Thema teilen, und nur ein Kunde erhält	`$share/consumer/sports/tennis`

Abonnement	Description	Beispiele für Metrikfilter
	Meldungen, die zu diesem Thema veröffentlicht wurden, nach dem Zufallsprinzip.	`$share/consumer/sports/#`

Non-shared Abonnements fließen	Ablauf geteilter Abonnements		
			

Wichtige Hinweise zur Verwendung geteilter Abonnements

- Wenn die gemeinsame Abonnentengruppe aus Abonnenten einer dauerhaften Sitzung besteht, wenn alle Abonnenten in der gemeinsamen Gruppe getrennt werden oder wenn ein Abonnent das Limit für Veröffentlichungsanfragen pro Sekunde pro Verbindung überschreitet, werden alle unbestätigten QoS 1-Nachrichten und nicht zugestellten QoS 1-Nachrichten, die in einer gemeinsamen Abonnementgruppe veröffentlicht wurden, in die Warteschlange gestellt. Weitere Informationen finden Sie unter Nachrichtenwarteschlangen für gemeinsame Abonnements. ???
- QoS 0-Nachrichten, die in einer gemeinsamen Abonnementgruppe veröffentlicht wurden, werden bei einem Ausfall gelöscht.
- Bei gemeinsamen Abonnements werden keine Nachrichten gespeichert, wenn sie Themenmuster als Teil einer gemeinsamen Abonnentengruppe abonnieren. Nachrichten, die zu Themen veröffentlicht werden, die gemeinsame Abonnenten haben und für die das `RETAIN` Kennzeichen gesetzt ist, werden wie jede andere Veröffentlichungsnachricht an gemeinsame Abonnenten zugestellt.
- Wenn gemeinsame Abonnements Platzhalterzeichen (# oder +) enthalten, gibt es möglicherweise mehrere übereinstimmende gemeinsame Abonnements zu einem Thema. In diesem Fall kopiert der Message Broker die Veröffentlichungsnachricht und sendet sie in jedem passenden gemeinsamen Abonnement an einen zufälligen Client. Das Platzhalterverhalten von gemeinsamen Abonnements kann in der folgenden Abbildung erklärt werden.



In diesem Beispiel gibt es drei passende gemeinsame Abonnements für das MQTT-Thema zur Veröffentlichung. `sports/tennis` Der Message Broker kopiert die veröffentlichte Meldung und sendet die Meldung an einen zufälligen Client in jeder passenden Gruppe.

Kunde 1 und Kunde 2 teilen sich das Abonnement: `$share/consumer1/sports/tennis`

Kunde 3 und Kunde 4 teilen sich das Abonnement: `$share/consumer1/sports/#`

Kunde 5 und Kunde 6 teilen sich das Abonnement: `$share/consumer2/sports/tennis`

Weitere Informationen zu den Beschränkungen für gemeinsame Abonnements finden Sie in der AWS Allgemeinen AWS IoT Core Referenz unter Endpunkte und Kontingente. Informationen zum Testen gemeinsamer Abonnements mit dem AWS IoT MQTT-Client in der AWS IoT Konsole finden Sie unter. ??? Mithilfe der Funktionen zur Verwaltung der Clientverbindungen können Sie auch sehen, welche Themen verbundene Clients abonniert haben, einschließlich gemeinsamer Abonnements. Weitere Informationen finden Sie unter the section called “MQTT-Verbindungsverwaltung”. Weitere Informationen zu gemeinsamen Abonnements finden Sie in der MQTTv5.0 Spezifikation unter Gemeinsame Abonnements.

Nachrichtenwarteschlangen für gemeinsame Abonnements

Um die Zuverlässigkeit der Nachrichtenübermittlung zu erhöhen, enthalten gemeinsame Abonnements Nachrichtenwarteschlangenfunktionen, mit denen Nachrichten gespeichert werden, wenn keine Online-Abonnenten verfügbar sind. Wenn eine gemeinsame Abonnementgruppe mindestens ein Mitglied mit einer dauerhaften Sitzung enthält, ist die Warteschlangenfunktion für die Gruppe aktiviert. Beim Verteilen von Nachrichten werden Online-Mitglieder als Empfänger ausgewählt. QoS 1-Nachrichten werden in die Warteschlange gestellt, wenn keine Mitglieder

online gefunden werden oder wenn Abonnenten das `Publish requests per second per connection` Limit überschreiten. Nachrichten in der Warteschlange werden zugestellt, wenn entweder bestehende Mitglieder ihre ständigen Sitzungen wieder aufnehmen oder neue Mitglieder der Gruppe beitreten. Nachrichten in der Warteschlange werden mit bis zu 20 Nachrichten pro Sekunde pro aktivem Gruppenabonnenten zugestellt, zusammen mit allen anderen Nachrichten, die den Abonnenten gemäß den Abonnements zugestellt werden.

Standardmäßig folgt die Aufbewahrung von Nachrichten in der Warteschlange dem Kontingent. `Persistent Session expiry period` Wenn in der eingehenden Veröffentlichungsnachricht jedoch ein Nachrichtenablaufintervall (MEI) festgelegt ist, hat das MEI Vorrang. Wenn MEI vorhanden ist, bestimmt es den Aufbewahrungszeitraum für Nachrichten, unabhängig vom Ablaufzeitraum der persistenten Sitzung.

Die Anzahl der Nachrichtenwarteschlangen ist entsprechend dem `Queued messages per second per account` Kontingent begrenzt, und die Anzahl der Nachrichten ist durch das `Maximum number of queued messages per shared subscription group` Kontingent begrenzt. Um Ihre Kontingente einzusehen und zu verwalten, greifen Sie auf die Service Quota-Konsole zu.

Sie können die Warteschlange überwachen, CloudWatch indem Sie `ApproximateQueueDepth` im `AWS/Usage` Namespace danach suchen, oder Sie können den folgenden CLI-Befehl verwenden, um die Metriken aufzulisten, die der Warteschlangentiefe jeder gemeinsamen Abonnementgruppe zugeordnet sind.

```
aws cloudwatch list-metrics --namespace "AWS/Usage" --dimensions
 Name=Resource,Value='ApproximateQueueDepth'
```

Wenn diese Grenzwerte überschritten werden, werden nur die Nachrichten aufbewahrt, die bereits vor Erreichen des Limits in die Warteschlange gestellt wurden. Neue eingehende Nachrichten, die die Grenzwerte überschreiten würden, werden verworfen. Das System ersetzt ältere Nachrichten in der Warteschlange nicht durch neuere.

Das Einreihen von Nachrichten in die Warteschlange wird in CloudWatch Metriken und CloudWatch Protokollen aufgezeichnet. Hinweise zu den Einträgen, in die geschrieben wurde, CloudWatch und zu den CloudWatch Protokollen finden Sie unter Message Broker-Metriken undProtokolleintrag in der Warteschlange. Nachrichten in der Warteschlange werden weiterhin zum Standardtarif für Nachrichten in Rechnung gestellt. Weitere Informationen zu Preisen erhalten Sie unter AWS IoT Core Preise.

Sitzungslebenszyklus in einer gemeinsamen Abonnementgruppe

Wenn eine Sitzung ohne Inhalt eine Gruppe abonniert, wird sie zu einem Online-Mitglied der Gruppe. Wenn sie sich abmeldet oder die Verbindung trennt, verlässt die saubere Sitzung die Gruppe.

Wenn eine permanente Sitzung eine Gruppe abonniert, wird sie ein Online-Mitglied der Gruppe. Wenn die Verbindung unterbrochen wird, verbleibt sie immer noch in der Gruppe, wird aber Offline-Mitglied der Gruppe. Wenn es wieder eine Verbindung herstellt, wird es wieder ein Online-Mitglied. Die persistente Sitzung verlässt die Gruppe, wenn sie sich explizit abmeldet oder wenn sie nach der Trennung abläuft.

Clean Start und Ablauf der Sitzung

Sie können Clean Start und Session Expiry verwenden, um Ihre persistenten Sitzungen flexibler zu handhaben. Ein Clean Start-Flag gibt an, ob die Sitzung gestartet werden soll, ohne eine bestehende Sitzung zu verwenden. Ein Sitzungsablaufintervall gibt an, wie lange die Sitzung nach einer Unterbrechung beibehalten werden soll. Das Ablaufintervall für die Sitzung kann bei der Trennung geändert werden. Weitere Informationen finden Sie unter the section called “Persistente MQTT-Sitzungen”.

Ursachencode auf allen ACKs

Mithilfe der Ursachencodes können Sie Fehlermeldungen einfacher debuggen oder verarbeiten. Ursachencodes werden vom Message Broker auf der Grundlage der Art der Interaktion mit dem Broker zurückgegeben (Abonnieren, Veröffentlichen, Bestätigen). Weitere Informationen finden Sie unter MQTT. Eine vollständige Liste der MQTT-Ursachencodes finden Sie in der MQTT v5-Spezifikation.

Aliasnamen zum Thema

Sie können einen Themennamen durch einen Themenalias ersetzen, bei dem es sich um eine Zwei-Byte-Ganzzahl handelt. Durch die Verwendung von Themenaliasen kann die Übertragung von Themennamen optimiert werden, sodass die Datenkosten für gebührenpflichtige Datendienste potenziell gesenkt werden können. AWS IoT Core hat eine Standardbeschränkung von 8 Themenaliasen. Weitere Informationen finden Sie unter AWS IoT Core -Endpunkte und -Kontingente in der AWS Allgemeinen Referenz.

Nachrichtablauf

Sie können veröffentlichten Meldungen Werte für den Ablauf von Meldungen hinzufügen. Diese Werte stellen das Nachrichtenablaufintervall (MEI) in Sekunden dar. Wenn die Meldungen innerhalb

dieses Intervalls nicht an die Abonnenten gesendet wurden, läuft die Meldung ab und wird entfernt. Wenn Sie den Wert für das Verfallsdatum der Meldung nicht festlegen, läuft die Meldung nicht ab.

Beim ausgehenden Versand erhält der Abonnent eine Meldung mit der verbleibenden Restzeit des Ablaufintervalls. Wenn beispielsweise eine eingehende Veröffentlichungsnachricht einen Meldungsablauf von 30 Sekunden hat und sie nach 20 Sekunden an den Abonnenten weitergeleitet wird, wird das Feld für den Ablauf der Meldung auf 10 aktualisiert. Es ist möglich, dass die vom Abonnenten empfangene Meldung einen aktualisierten MEI von 0 hat. Dies liegt daran, dass sie auf 0 aktualisiert wird, sobald die verbleibende Zeit 999 ms oder weniger beträgt.

In AWS IoT Core ist der minimale MEI 1. Wenn das Intervall auf der Clientseite auf 0 gesetzt ist, wird es auf 1 eingestellt. Das maximale Verfallsintervall für Meldungen beträgt 604800 (7 Tage). Alle Werte, die über diesem Wert liegen, werden an den Maximalwert angepasst.

Bei der versionsübergreifenden Kommunikation wird das Verhalten des Meldungsablaufs durch die MQTT-Version der eingehenden Veröffentlichungsnachricht bestimmt. Beispielsweise kann eine Meldung mit Ablauf der Nachricht, die von einer über MQTT5 verbundenen Sitzung gesendet wurde, für Geräte, die mit MQTT3-Sitzungen abonniert wurden, ablaufen. In der folgenden Tabelle ist aufgeführt, wie der Ablauf von Meldungen die folgenden Arten von Veröffentlichungsnachrichten unterstützt:

Veröffentlichen eines Meldungstyps	Ablaufintervall für Meldungen
Reguläres Veröffentlichen	Wenn ein Server die Meldung nicht innerhalb der angegebenen Zeit zustellt, wird die abgelaufene Meldung entfernt und der Abonnent erhält sie nicht. Dies schließt Situationen ein, z. B. wenn ein Gerät seine QoS 1-Meldungen nicht veröffentlicht.
Beibehalten	Wenn eine gespeicherte Meldung abläuft und ein neuer Kunde das Thema abonniert, erhält der Client die Meldung nach Abschluss des Abonnements nicht.
Letzter Wille	Das Intervall für Meldungen des letzten Willens beginnt, nachdem der Client die Verbindung getrennt hat und der Server versucht, seinen Abonnenten die letzte Willensnachricht zuzustellen.

Veröffentlichen eines Meldungstyps	Ablaufintervall für Meldungen
In die Warteschlange eingestellte Meldungen	Wenn ein ausgehender QoS 1 mit Nachrichtenablaufintervall abläuft, wenn ein Client offline ist, erhält der Client nach Wiederaufnahme der persistenten Sitzung die abgelaufene Nachricht nicht.

Weitere Funktionen von MQTT 5

Verbindung zum Server trennen

Wenn eine Verbindung unterbrochen wird, kann der Server dem Client proaktiv eine DISCONNECT-Meldung senden, um den Verbindungsabbruch mit einem Ursachencode für die Trennung zu benachrichtigen.

Request/Response

Verlage können verlangen, dass der Empfänger nach Erhalt eine Antwort auf ein vom Verlag spezifiziertes Thema sendet.

Maximale Teilegröße

Client und Server können unabhängig voneinander die maximale Paketgröße angeben, die sie unterstützen.

Payload-Format und Inhaltstyp

Sie können das Payload-Format (Binär, Text) und den Inhaltstyp angeben, wenn eine Meldung veröffentlicht wird. Diese werden an den Empfänger der Meldung weitergeleitet.

MQTT 5 Eigenschaften

MQTT 5-Eigenschaften sind wichtige Ergänzungen zum MQTT-Standard zur Unterstützung neuer MQTT 5-Funktionen wie Sitzungsablauf und Muster. Request/Response In können Sie Regeln erstellen AWS IoT Core, die die Eigenschaften in ausgehenden Nachrichten weiterleiten können, oder HTTP Publish verwenden, um MQTT-Nachrichten mit einigen der neuen Eigenschaften zu veröffentlichen.

In der folgenden Tabelle sind alle MQTT 5-Eigenschaften aufgeführt, die unterstützt werden. AWS IoT Core

Property (Eigenschaft)	Description (Beschreibung)	Eingab p	packets
Nutzlastformatindikator Nutzlastnutzlast	Ein boolescher Wert, der angibt, ob die Nutzlast als formatiert ist. UTF-8	Byte	VERÖFFENTLICHEN, VERBINDEN
Inhaltstyp	Eine UTF-8 Zeichenfolge, die den Inhalt der Nutzlast beschreibt.	UTF-8 Zeichelge	VERÖFFENTLICHEN, VERBINDEN
Thema der Antwort	Eine UTF-8 Zeichenfolge, die das Thema beschreibt, zu dem der Empfänger im Rahmen des Anfrage-Antwort-Ablaufs veröffentlichen soll. Das Thema darf keine Platzhalterzeichen enthalten.	UTF-8 Zeichelge	VERÖFFENTLICHEN, VERBINDEN
Korrelationsdaten	Binärdaten, die vom Absender der Anforderungsnachricht verwendet werden, um zu identifizieren, für welche Anfrage die Antwortnachricht bestimmt ist.	Binär	VERÖFFENTLICHEN, VERBINDEN
Benutzereigenschaften	Ein UTF-8 Zeichenkettenpaar. Diese Eigenschaft kann in einem Paket mehrfach vorkommen. Die Empfänger empfangen die Schlüssel-Wert-Paare in derselben Reihenfolge, in der sie gesendet wurden.	UTF-8 Zeichettenpa	VERBINDEN, VERÖFFENTLICHEN, Will-Eigenschaften, ABONNIEREN, TRENNEN, ABBESTELLEN
Ablaufintervall für Meldungen	Eine 4-Byte-Ganzzahl, die das Nachrichtenablaufintervall in Sekunden darstellt. Wenn nicht vorhanden, läuft die Meldung nicht ab.	4-Byte-Ganzzahl	VERÖFFENTLICHEN, VERBINDEN

Property (Eigenschaft)	Description (Beschreibung)	Eingab p	packets
Ablaufintervall der Sitzung	Eine 4-Byte-Ganzzahl, die das Ablaufintervall der Sitzung in Sekunden darstellt. AWS IoT Core unterstützt ein Maximum von 7 Tagen, mit einem Standardmaximum von einer Stunde. Wenn der von Ihnen eingestellte Wert den Höchstwert Ihres Kontos überschreitet, AWS IoT Core wird der angepasste Wert im CONNACK zurückgegeben.	4-Byte-Ganzzahl	VERBINDEN, VERBINDEN, TRENNEN
Zugewiesene Client-ID	Eine zufällige Client-ID, die generiert wird AWS IoT Core , wenn eine Client-ID nicht von Geräten angegeben wird. Bei der zufälligen Client-ID muss es sich um eine neue Client-ID handeln, die von keiner anderen Sitzung verwendet wird, die derzeit vom Broker verwaltet wird.	UTF-8 Zeichenlge	CONNACK
Server Keep Alive	Eine 2-Byte-Ganzzahl, die die vom Server zugewiesene Keep-Alive-Zeit darstellt. Der Server trennt die Verbindung zum Client, wenn der Client länger als die Keep-Alive-Zeit inaktiv ist.	2-Byte-Ganzzahl	CONNACK
Problemin formationen anfordern	Ein boolescher Wert, der angibt, ob die Reason-Zeichenfolge oder die Benutzereigenschaften bei Fehlern gesendet werden.	Byte	CONNECT
Maximal empfangen	Eine 2-Byte-Ganzzahl, die die maximale Anzahl von PUBLISH QOS > 0-Paketen darstellt, die gesendet werden können, ohne dass ein PUBACK empfangen wird.	2-Byte-Ganzzahl	VERBINDEN, CONNACK

Property (Eigenschaft)	Description (Beschreibung)	Eingab p	packets
Thema Alias Maximum	Dieser Wert gibt den höchsten Wert an, der als Themen-Alias akzeptiert wird. Standard = 0.	2-Byte-Ganzzahl	VERBINDEN, CONNACK
Maximale QoS	Der maximale QoS-Wert, der unterstützt wird. AWS IoT Core Die Standardeinstellung ist 1. AWS IoT Core unterstützt QoS 2 nicht.	Byte	CONNACK
Beibehaltung verfügbar	Ein boolescher Wert, der angibt, ob der AWS IoT Core Message Broker beibehaltene Nachrichten unterstützt. Der Standardwert ist 1.	Byte	CONNACK
Maximale Teilegröße	Die maximale Paketgröße, die AWS IoT Core akzeptiert und gesendet wird. 128 KB darf nicht überschritten werden.	4-Byte-Ganzzahl	VERBINDEN, CONNACK
Wildcard-Abonnement verfügbar	Ein boolescher Wert, der angibt, ob der AWS IoT Core Message Broker Wildcard Subscription Available unterstützt. Der Standardwert ist 1.	Byte	CONNACK
Abonnement-ID verfügbar	Ein boolescher Wert, der angibt, ob der AWS IoT Core Message Broker Subscription Identifier Available unterstützt. Der Standardwert ist 0.	Byte	CONNACK

MQTT-Ursachencodes

MQTT 5 führt eine verbesserte Fehlerberichterstattung mit Antworten auf Ursachencodes ein. AWS IoT Core kann Ursachencodes zurückgeben, einschließlich, aber nicht beschränkt auf die folgenden, nach Paketen gruppierten Codes. Eine vollständige Liste der von MQTT 5 unterstützten Ursachencodes finden Sie in den MQTT 5-Spezifikationen.

CONNACK-Ursachencodes

Wert	HEX()	Name des Grundcodes	Description
0	0x00	Herzlichen Glückwunsch	Die Verbindung wurde akzeptiert.
128	0x80	Unbekannter Fehler	Der Server möchte den Grund für den Fehler nicht preisgeben, oder keiner der anderen Ursachencodes trifft zu.
133	0x85	Die Client-ID ist nicht gültig.	Die Client-ID ist eine gültige Zeichenfolge, die jedoch vom Server nicht zugelassen wird.
134	0x86	Falscher Benutzername oder falsches Passwort	Der Server akzeptiert den vom Client angegebenen Benutzernamen oder das Passwort nicht.
135	0x87	Nicht autorisiert.	Der Client ist nicht autorisiert, eine Verbindung herzustellen.
144	0x90	Themenname ungültig	Der Name des Will-Themas ist korrekt formatiert, wird aber vom Server nicht akzeptiert.
151	0 x 97	Kontingent überschritten	Ein von der Implementierung oder vom Administrator auferlegtes Limit wurde überschritten.
155	0 x 9 B	QoS nicht unterstützt	Der Server unterstützt die in Will QoS eingestellte QoS nicht.

PUBACK-Ursachencodes

Wert	HEX()	Name des Grundcodes	Description
0	0x00	Herzlichen Glückwunsch	Die Meldung wurde akzeptiert. Die Veröffentlichung der QoS 1-Meldung wird fortgesetzt.

Wert	HEX()	Name des Grundcodes	Description
128	0x80	Unbekannter Fehler	Der Empfänger akzeptiert die Veröffentlichung nicht, möchte aber entweder den Grund nicht preisgeben, oder er entspricht keinem der anderen Werte.
135	0x87	Nicht autorisiert.	Die Veröffentlichung ist nicht autorisiert.
144	0x90	Themenname ungültig	Der Themenname ist nicht falsch formatiert, wird aber vom Client oder Server nicht akzeptiert.
145	0x91	Paket-ID wird verwendet.	Die Paket-ID wird bereits verwendet. Dies könnte auf eine Nichtübereinstimmung des Sitzungsstatus zwischen Client und Server hinweisen.
151	0 x 97	Kontingent überschritten	Ein von der Implementierung oder vom Administrator auferlegtes Limit wurde überschritten.

Disconnect – Ursachencodes

Wert	HEX()	Name des Grundcodes	Description
129	0x81	Fehlerhaftes Paket	Das empfangene Paket entspricht nicht dieser Spezifikation.
130	0x82	Protokollfehler	Ein unerwartetes Paket oder ein Paket, das nicht in der richtigen Reihenfolge ist, wurde empfangen.
135	0x87	Nicht autorisiert.	Die Anfrage ist nicht autorisiert.
139	0x8B	Server wird heruntergefahren.	Der Server wird heruntergefahren.

Wert	HEX()	Name des Grundcodes	Description
141	0x8D	Keep Alive Timeout	Die Verbindung wurde geschlossen, weil seit dem 1,5-fachen der Keep-Alive-Zeit kein Paket mehr empfangen wurde.
142	0x8E	Sitzung übernommen	Eine andere Verbindung, die dieselbe Client-ID verwendet, hat eine Verbindung hergestellt, wodurch diese Verbindung geschlossen wurde.
143	0x8F	Ungültiger Themenfilter	Der Themenfilter ist korrekt formatiert, wird aber vom Server nicht akzeptiert.
144	0x90	Themenname ungültig	Der Themenname ist korrekt formatiert, wird aber von diesem Client oder Server nicht akzeptiert.
147	0 x 93	Empfangsm aximum überschritten	Der Client oder Server hat mehr als die Empfangsm aximum-Publikation erhalten, für die er weder PUBACK noch PUBCOMP gesendet hat.
148	0 x 94	Ungültiges Thema-Alias	Der Client oder Server hat ein PUBLISH-Paket erhalten, das einen Themen-Alias enthält, der größer ist als die maximale Anzahl an Themen-Alias, die er im CONNECT- oder CONNACK-Paket gesendet hat.
151	0 x 97	Kontingent überschritten	Ein von der Implementierung oder vom Administrator auferlegtes Limit wurde überschritten.
152	0x98	Administrative Maßnahmen	Die Verbindung wurde aufgrund einer Verwaltun gsmaßnahme geschlossen.
155	0 x 9 B	QoS nicht unterstützt	Der Client hat eine QoS angegeben, die höher ist als die QoS, die in einem Maximum QoS im CONNACK angegeben ist.

Wert	HEX()	Name des Grundcodes	Description
161	0 x A1	Abonnement-Identifikatoren werden nicht unterstützt.	Der Server unterstützt keine Abonnement-IDs. Das Abonnement wird nicht akzeptiert.

SUBACK-Ursachencodes

Wert	HEX()	Name des Grundcodes	Description
0	0x00	QoS gewährt 0	Das Abonnement wurde akzeptiert, und die maximale gesendete QoS beträgt QoS 0. Dies könnte eine niedrigere QoS sein als gewünscht.
1	0x01	QoS 1 gewährt 1	Das Abonnement wurde akzeptiert, und die maximale gesendete QoS beträgt QoS 1. Dies könnte eine niedrigere QoS sein als gewünscht.
128	0x80	Unbekannter Fehler	Das Abonnement wurde nicht akzeptiert, und der Server möchte den Grund entweder nicht preisgeben oder keiner der anderen Ursachencodes trifft zu.
135	0x87	Nicht autorisiert.	Der Kunde ist nicht berechtigt, dieses Abonnement abzuschli eßen.
143	0x8F	Ungültiger Themenfilter	Der Themenfilter ist korrekt formatiert, ist aber für diesen Client nicht zulässig.
145	0x91	Paket-ID wird verwendet.	Die angegebene Paket-ID wird bereits verwendet.
151	0 x 97	Kontingent überschritten	Ein von der Implementierung oder vom Administrator auferlegtes Limit wurde überschritten.

UNSUBACK-Ursachencodes

Wert	HEX()	Name des Grundcodes	Description
0	0x00	Herzlichen Glückwunsch	Das Abonnement wird gelöscht.
128	0x80	Unbekannter Fehler	Die Abmeldung konnte nicht abgeschlossen werden, und der Server möchte den Grund entweder nicht preisgeben oder keiner der anderen Ursachencodes trifft zu.
143	0x8F	Ungültiger Themenfilter	Der Themenfilter ist korrekt formatiert, ist aber für diesen Client nicht zulässig.
145	0x91	Paket-ID wird verwendet.	Die angegebene Paket-ID wird bereits verwendet.

AWS IoT Unterschiede zu den MQTT-Spezifikationen

Die Message-Broker-Implementierung basiert auf der MQTT-Spezifikation v3.1.1 und der MQTT-Spezifikation v5.0, unterscheidet sich jedoch in folgenden Punkten von den Spezifikationen:

- AWS IoT unterstützt die folgenden Pakete für MQTT 3 nicht: PUBREC, PUBREL und PUBCOMP.
- AWS IoT unterstützt die folgenden Pakete für MQTT 5 nicht: PUBREC, PUBREL, PUBCOMP und AUTH.
- AWS IoT unterstützt keine MQTT 5-Serverumleitung.
- AWS IoT unterstützt nur die MQTT Quality of Service (QoS) -Stufen 0 und 1. AWS IoT unterstützt das Veröffentlichen oder Abonnieren mit QoS Level 2 nicht. Bei Abfragen über QoS-Ebene 2 sendet der Message Broker kein PUBACK oder SUBACK.
- In AWS IoT bedeutet das Abonnieren eines Themas mit QoS-Stufe 0, dass eine Nachricht null oder mehrmals zugestellt wird. Meldungen können mehr als einmal übertragen werden. Meldungen, die mehrmals übertragen werden, können mit unterschiedlicher Paket-ID gesendet werden. In diesem Fall wird das DUP-Flag nicht festgelegt.
- Der Message Broker sendet als Antwort auf eine Verbindungsanfrage eine CONNACK-Nachricht. Diese enthält ein Flag, das angibt, ob eine frühere Sitzung wieder aufgenommen wird.

- Bevor der Client zusätzliche Steuerpakete oder eine Anfrage zum Trennen der Verbindung sendet, muss er warten, bis die CONNACK-Meldung vom Message Broker auf seinem Gerät empfangen wird. AWS IoT
- Wenn ein Client ein Thema abonniert, kann es zwischen dem Zeitpunkt, an dem der Message Broker eine SUBACK-Meldung sendet, und den ersten eingehenden passenden Meldungen eine Verzögerung geben.
- Wenn ein Client das Platzhalterzeichen # im Themenfilter verwendet, um ein Thema zu abonnieren, werden alle Zeichenfolgen auf und unter seiner Ebene in der Themenhierarchie abgeglichen. Das übergeordnete Thema stimmt jedoch nicht überein. Wenn Sie beispielsweise das Thema `sensor/#` abonnieren, erhalten Sie Meldungen, die für `sensor/`, `sensor/temperature` oder `sensor/temperature/room1` veröffentlicht werden, nicht jedoch Meldungen, die für `sensor` veröffentlicht werden. Weitere Informationen zu Platzhaltern unter Filter für Themennamen.
- Der Message Broker verwendet die Client-ID, um die einzelnen Clients zu identifizieren. Die Client-ID wird vom Client im Rahmen der MQTT-Nutzlast an den Message Broker übermittelt. Zwei Clients mit identischen Client-IDs können nicht gleichzeitig mit dem Message Broker verbunden sein. Stellt ein Client eine Verbindung zum Message Broker über eine Client-ID her, die bereits von einem anderen Client verwendet wird, wird die neue Client-Verbindung akzeptiert und die Verbindung des früher verbundenen Clients getrennt. Sie können Clients auch manuell über die Konsole oder APIs trennen. Weitere Informationen finden Sie unter the section called “MQTT-Verbindungsverwaltung”.
- In seltenen Fällen kann der Message Broker die gleiche logische PUBLISH Meldung mit einer anderen Paket-ID erneut senden.
- Wenn Sie Themenfilter abonnieren, die ein Platzhalterzeichen enthalten, können keine beibehaltenen Meldungen empfangen werden. Um eine gespeicherte Meldung zu erhalten, muss die Anforderung einen Themenfilter enthalten, der genau dem Thema der gespeicherten Meldung entspricht.
- Die Reihenfolge, in der Meldungen und ACK empfangen werden, kann vom Message Broker nicht garantiert werden.
- AWS IoT kann Grenzwerte haben, die von den Spezifikationen abweichen. Weitere Informationen zu den Grenzwerten von AWS IoT Core Message Broker finden Sie im AWS IoT Referenzhandbuch.
- Das MQTT DUP-Flag wird nicht unterstützt.

MQTT-Verbindungsverwaltung

AWS IoT Core bietet APIs und Konsolenfunktionen, die Sie bei der Verwaltung von MQTT-Verbindungen unterstützen, einschließlich der Möglichkeit, Clients zu trennen, Verbindungsdetails anzuzeigen und deren Sitzungen zu verwalten. Diese Funktionen geben Ihnen mehr Kontrolle über Ihre AWS IoT Kundenflotte und helfen Ihnen bei der Behebung von Verbindungsproblemen.

In diesem Abschnitt

- the section called “MQTT-Clients trennen”
- the section called “Anzeigen von Verbindungsdetails”
- the section called “Verwaltung von Kundenabonnements”
- the section called “CloudWatch Loggt AWS IoT Einträge protokollieren”

MQTT-Clients trennen

Verwenden Sie die `DeleteConnection` API, um MQTT-Geräte zu trennen, AWS IoT Core indem Sie deren Client-IDs angeben. Wenn Sie die AWS IoT Core Verbindung zu einem Client trennen, wird der Client vom AWS IoT Core Message Broker getrennt und Sie können optional den Sitzungsstatus bereinigen und die LWT-Nachricht (Last Will and Testament) unterdrücken.

AWS IoT Core Führt beim Aufrufen der `DeleteConnection` API mehrere Aktionen aus, um eine saubere Trennung der Verbindung sicherzustellen. AWS IoT Core sendet zunächst eine MQTT-Trennmeldung an den Client, um die MQTT-Sitzung zu beenden. Der Dienst schließt dann den zugrunde TCP/TLS liegenden Socket.

Der Message Broker sendet ein `DISCONNECT` Paket an das Gerät und veröffentlicht ein Lebenszyklusereignis mit dem Grund für die Trennung`API_INITIATED_DISCONNECT`. Auf diese Weise können Sie feststellen, wann eine Verbindungsunterbrechung über die API und nicht durch den Client oder aufgrund von Netzwerkproblemen initiiert wurde. Sie können diese Ereignisse aus Gründen der Sichtbarkeit, Problembehandlung und Prüfung überwachen. Beispielsweise können Sie AWS IoT Regeln verwenden, um diese Ereignisse zu verarbeiten, um nachzuverfolgen, wann und warum die Client-Verbindung unterbrochen wurde.

Wenn Sie den `cleanSession` Parameter auf setzen`true`, AWS IoT Core wird der Sitzungsstatus des Clients entfernt, einschließlich aller Abonnements und Nachrichten in der Warteschlange. Wenn Sie eine Sitzung bereinigen, wird die persistente Sitzung beendet. Wenn der Client eine persistente Sitzung war und der `preventWillMessage` Parameter auf gesetzt ist`false`, sendet der Dienst die LWT-Meldung, falls verfügbar, was bei geplanten Wartungsarbeiten nützlich ist.

Wenn Sie die `DeleteConnection` API aufrufen, beginnt der Verbindungsabbruch sofort. Der genaue Zeitpunkt, zu dem der Client die Trennung erkennt, kann jedoch je nach Netzwerkbedingungen und Client-Implementierung variieren. Die meisten Clients erkennen die Trennung innerhalb weniger Sekunden, aber in einigen Fällen kann es bei schlechter Netzwerkkonnektivität länger dauern, bis der Client erkennt, dass die Verbindung getrennt wurde.

> **Note**
>
> Wenn ein Client eine Verbindungsunterbrechung erzwingt, muss er sich erneut authentifizieren und mit einem neuen Sitzungsstatus erneut autorisieren. Der API-Aufruf selbst verhindert nicht die Wiederverbindung von Clients. Wenn dies gewünscht wird, müssen zusätzlich die Anmeldeinformationen oder Richtlinien eines Kunden geändert werden, bevor der `DeleteConnection` API-Aufruf ausgeführt wird.

Informationen zu Preisen finden Sie unter AWS IoT Core -Preise.

Anwendungsfälle

Die `DeleteConnection` API ist nützlich für die Verwaltung von Kunden, die sich schlecht benehmen und problematisches Verhalten zeigen oder übermäßig viele Ressourcen verbrauchen. Indem Sie eine Unterbrechung erzwingen, können Sie sicherstellen, dass die Clients ihre Verbindungen mit der richtigen Authentifizierung und Autorisierung wieder herstellen. Dies kann helfen, Probleme mit dem Ressourcenverbrauch zu lösen.

Szenarien zur Client-Umleitung profitieren ebenfalls von dieser API. Wenn Sie Clients zu verschiedenen Endpunkten umleiten müssen AWS-Regionen, oder Sie können sie programmgesteuert trennen und sie durch Ändern der DNS-Einstellungen erneut eine Verbindung zu einem anderen AWS IoT Core Endpunkt herstellen lassen. Diese API kann helfen, festgefahrene Verbindungen zu lösen oder problematische Sitzungszustände zu löschen, die den normalen Betrieb verhindern könnten.

API-Parameter

Die `DeleteConnection` API akzeptiert die folgenden Parameter:

clientId (erforderlich)

Die eindeutige Kennung des MQTT-Clients, dessen Verbindung getrennt werden soll. Dies ist im URL-Pfad angegeben. Die Client-ID darf nicht mit einem Dollarzeichen ($) beginnen.

> **Note**
>
> MQTT-Client-IDs können Zeichen enthalten, die in HTTP-Anfragen nicht gültig sind. Wenn Sie die `DeleteConnection` API verwenden, müssen Sie alle Zeichen in der Client-ID, die in MQTT, aber nicht in HTTP gültig sind, per URL codieren (prozentual codieren). Dazu gehören Sonderzeichen wie Leerzeichen, Schrägstriche (/) und Zeichen. UTF-8 Beispielsweise wird aus einem Leerzeichen %20, aus einem Schrägstrich %2F und aus dem UTF-8 Zeichen ü wird %C 3% BC. Durch die richtige Codierung wird sichergestellt, dass die MQTT-Client-IDs im API-Aufruf korrekt übertragen werden. HTTP-based

CleanSession (optional)

Gibt an, ob der Sitzungsstatus des Clients beim Trennen der Verbindung entfernt werden soll. Legen Sie diese Option `true` auf fest, um alle Sitzungsinformationen zu löschen, einschließlich Abonnements und Nachrichten in der Warteschlange. Auf setzen, `false` um den Sitzungsstatus beizubehalten. In der Standardeinstellung ist dies auf `false` (behält den Sitzungsstatus bei) gesetzt. Bei sauberen Sitzungen wird dieser Parameter ignoriert.

verhindern WillMessage (optional)

Steuert, ob AWS IoT Core die LWT-Nachricht (Last Will and Testament) ausgesendet wird, falls diese bei Verbindungsunterbrechung verfügbar ist. Stellen Sie diese Option auf ein, `true` um den Versand der LWT-Nachricht zu verhindern. Auf setzen, um den `false` Versand zu ermöglichen. Standardmäßig ist dies auf `false` (versendet LWT, falls verfügbar) gesetzt.

API-Syntax

Die `DeleteConnection` API verwendet das folgende HTTP-Anforderungsformat:

```
DELETE /connections/<clientId>?
cleanSession=<cleanSession>&preventWillMessage=<preventWillMessage> HTTP/1.1
```

Beispielanfragen:

```
// Basic disconnect (preserves session, allows LWT message)
DELETE /connections/myDevice123 HTTP/1.1

// Disconnect and clear session
```

```
DELETE /connections/myDevice123?cleanSession=TRUE HTTP/1.1

// Disconnect, clear session, and prevent LWT message
DELETE /connections/myDevice123?cleanSession=TRUE&preventWillMessage=TRUE HTTP/1.1
```

Erfolgreiche Anfragen geben HTTP 200 OK ohne Antworttext zurück.

> **Note**
>
> Der von AWS Signature Version 4 zum Signieren von Anfragen verwendete Dienstname lautet: iotdevicegateway. Sie können Ihren Endpunkt finden, indem Sie den Befehl verwenden. `aws iot describe-endpoint --endpoint-type iot:Data-ATS`

Erforderliche Berechtigungen

Um die `DeleteConnection` API verwenden zu können, benötigen Sie die folgende IAM-Berechtigung:

```
iot:DeleteConnection
```

Sie können diese Berechtigung mithilfe ressourcenbasierter Richtlinien auf bestimmte Client-IDs beschränken. Beispiel:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:DeleteConnection",
            "Resource": "arn:aws:iot:us-east-1:987654321:client/myDevice*"
        }
    ]
}
```

Wichtige Überlegungen

Getrennte Clients können sofort versuchen, die Verbindung wiederherzustellen, sofern Sie keine zusätzliche Logik implementiert haben, um eine erneute Verbindung zu verhindern. Der Trennvorgang beendet nur die aktuelle Verbindung und verhindert nicht, dass eine Verbindung

wieder hergestellt wird. Wenn Sie eine erneute Verbindung verhindern müssen, sollten Sie erwägen, eine clientseitige Logik zu implementieren oder die Anmeldeinformationen eines Geräts zu deaktivieren.

Für die API gelten im Rahmen der standardmäßigen API-Ratenbegrenzung Ratenbegrenzungen. AWS IoT Core Stellen Sie bei der Planung von Massenunterbrechungen sicher, dass Sie diese Grenzwerte berücksichtigen und eine angemessene Wiederholungslogik und Batch-Strategien implementieren, um Drosselungen zu vermeiden. Weitere Informationen finden Sie unter AWS IoT Core -Endpunkte und -Kontingente.

Fehlermeldungen

Die `DeleteConnection` API kann die folgenden Fehlerantworten zurückgeben:

InvalidRequestException

Die Anforderung ist ungültig. Dies kann auftreten, wenn das Client-ID-Format ungültig ist, ein Dollarzeichen ($) -Präfix enthält oder wenn erforderliche Parameter fehlen.

ResourceNotFoundException

Die angegebene Client-ID existiert nicht oder ist derzeit nicht verbunden und es besteht keine persistente Sitzung.

UnauthorizedException

Sie sind zur Ausführung dieser Operation nicht autorisiert. Stellen Sie sicher, dass Sie über die erforderliche `iot:DeleteConnection` Berechtigung verfügen.

ForbiddenException

Der Anrufer ist nicht berechtigt, die Anfrage zu stellen. Dies kann auf unzureichende IAM-Berechtigungen oder ressourcenbasierte Richtlinieneinschränkungen zurückzuführen sein.

ThrottlingException

Die Rate überschreitet den Grenzwert. Reduzieren Sie die Häufigkeit Ihrer API-Aufrufe und implementieren Sie eine entsprechende Wiederholungslogik mit exponentiellem Backoff.

InternalFailureException

Ein unerwarteter Fehler ist aufgetreten. Versuchen Sie die Anfrage nach einer kurzen Verzögerung erneut.

ServiceUnavailableException

Der Service ist vorübergehend nicht verfügbar. Versuchen Sie die Anfrage nach einer kurzen Verzögerung erneut.

Anzeigen von Verbindungsdetails

Verwenden Sie die `GetConnection` API, um detaillierte Informationen zu aktiven MQTT-Clientverbindungen abzurufen, einschließlich Verbindungsstatus, Sitzungsdetails und Netzwerkinformationen.

Die `GetConnection` API gibt umfassende Verbindungsmetadaten zurück, die bei der Behebung von Verbindungsproblemen, der Überwachung des Clientverhaltens und der Überprüfung von Verbindungsmustern helfen. Sie können diese Informationen verwenden, um die Lebenszyklen von Client-Verbindungen zu verstehen und Probleme mit MQTT-Sitzungen zu diagnostizieren. Verbindungsinformationen sind nach dem Trennen der Verbindung 30 Minuten lang verfügbar, unabhängig davon, ob der Client eine saubere oder eine persistente Sitzung verwendet hat. ??? Verwenden Sie für Metadaten, die länger als 30 Minuten getrennt sind, die GetThingConnectivityData API, für die die Flottenindizierung aktiviert sein muss.

Anwendungsfälle

Die `GetConnection` API ist nützlich, um Verbindungsprobleme zu beheben und das Verhalten der Kunden zu überwachen. Sie können den Client-Konnektivitätsstatus überprüfen, die Einstellungen für persistente Sitzungen überprüfen und die Verbindungszeitstempel überprüfen, um die Verbindungsmuster der Clients zu verstehen.

Netzwerkdiagnosen profitieren ebenfalls von dieser API. Zu den Verbindungsdetails gehören Quell- und Ziel-IP-Adressen und -Ports, anhand derer Probleme mit dem Netzwerkrouting oder Verbindungsendpunkten identifiziert werden können. Diese Informationen sind beim Debuggen von Verbindungsproblemen oder bei der Optimierung von Netzwerkkonfigurationen hilfreich.

API-Parameter

Die `GetConnection` API akzeptiert die folgenden Parameter:

clientId (erforderlich)

Die eindeutige Kennung des MQTT-Clients, für den Verbindungsdetails abgerufen werden sollen. Dies ist im URL-Pfad angegeben. Die Client-ID darf nicht mit einem Dollarzeichen ($) beginnen.

> **Note**
>
> MQTT-Client-IDs können Zeichen enthalten, die in HTTP-Anfragen nicht gültig sind. Wenn Sie die `GetConnection` API verwenden, müssen Sie alle Zeichen in der Client-ID, die in MQTT, aber nicht in HTTP gültig sind, per URL codieren (prozentual codieren). Dazu gehören Sonderzeichen wie Leerzeichen, Schrägstriche (/) und Zeichen. UTF-8 Beispielsweise wird aus einem Leerzeichen %20, aus einem Schrägstrich %2F und aus dem UTF-8 Zeichen ü wird %C 3% BC. Durch die richtige Codierung wird sichergestellt, dass die MQTT-Client-IDs im API-Aufruf korrekt übertragen werden. HTTP-based Überlegungen zum Routing: Die URL-Codierung ist erforderlich, wenn Client-IDs sowohl Schrägstriche als auch die Zeichenfolge „Abonnements“ enthalten. Ohne eine korrekte Codierung kann die Pfadstruktur falsch interpretiert werden und Ihre Anfrage wird stattdessen an die `ListSubscriptions` API weitergeleitet. `GetConnection` Beispielsweise client/subscriptions muss eine Client-ID als client%2FSubscriptions codiert werden, um sicherzustellen, dass die Anfrage korrekt weitergeleitet wird und `GetConnection` nicht als Anruf interpretiert wird. `ListSubscriptions`

einschließen (optional) SocketInformation

Typ: Boolescher Wert

Standard: false

Der `includeSocketInformation` Parameter steuert, ob Netzwerkinformationen auf Socket-Ebene in der API-Antwort enthalten sind. Wenn auf gesetzt`true`, enthält die Antwort die folgenden Felder:`sourceIp`,,`sourcePort`,`targetIp`. `targetPort` `vpcEndpointId` Wenn nicht angegeben oder auf gesetzt `includeSocketInformation` ist`false`, werden diese Socket-Felder von der Antwort ausgeschlossen. Wenn der `includeSocketInformation` Parameter auf gesetzt ist, der Aufrufer `true` jedoch nicht berechtigt ist, Socket-Informationen einzusehen, schlägt der gesamte API-Aufruf mit 403 Forbidden fehl.

API-Syntax

Die `GetConnection` API verwendet das folgende HTTP-Anforderungsformat:

```
GET /connections/<client-id>?includeSocketInformation=false HTTP/1.1
```

Beispielanfragen:

```
GET /connections/myDevice123?includeSocketInformation=false HTTP/1.1

// Get connection details for a client with special characters (URL encoded)
// Original client ID: "my device/123"
GET /connections/my%20device%2F123?includeSocketInformation=true HTTP/1.1
```

Beispielantwort für einen verbundenen Client:

```
{
    "clientId": "abcd12345fvgbh",
    "connected": true,
    "cleanSession": false,
    "connectedSince": 1573002340451,
    "thingName": "deviceABC",
    "sourceIp": "203.0.113.1",
    "sourcePort": 8883,
    "targetIp": "198.51.100.1",
    "targetPort": 8883,
    "keepAliveDuration": 60,
    "disconnectReason": null,
    "sessionExpiry": 3600
}
```

Antwortfelder:

clientId

Die Client-ID des MQTT-Clients.

verbunden

Der Verbindungsstatus des Clients. Gibt zurück AWS IoT Core, `true` ob der Client gerade mit dem Client verbunden ist oder `false` ob der Client nicht verbunden ist.

CleanSession

Gibt an, ob der Client eine saubere Sitzung verwendet. Gibt `true` für saubere Sitzungen oder `false` für persistente Sitzungen zurück. Saubere Sitzungen behalten ihren Status nach dem Trennen der Verbindung nicht bei, während persistente Sitzungen Abonnements und Nachrichten in der Warteschlange beibehalten.

Verbunden seit

Unix-Zeitstempel (in Millisekunden), der angibt, wann der Client eine Verbindung hergestellt hat. Nur vorhanden, wenn es ist. `connected` `true`

Die Verbindung wurde unterbrochen seit

Unix-Zeitstempel (in Millisekunden), der angibt, wann der Client die Verbindung getrennt hat. Nur vorhanden, wenn es ist. `connected` `false`

thingName

Dingname des Clients, der nur vorhanden ist, wenn der ClientID etwas zugeordnet ist.

disconnectReason

Der Grund für die letzte Trennung, falls der Client gerade getrennt ist. Dieses Feld hilft bei der Identifizierung, ob die Trennung vom Client oder vom Server initiiert wurde oder auf Netzwerkprobleme zurückzuführen ist. Nur vorhanden, wenn es ist. `connected` `false` Die Ursachencodes für die Unterbrechung der Verbindung finden Sie unter LifeCycleEvents.

behalten AliveDuration

Das Keep-Alive-Intervall in Sekunden, das der Client beim Verbindungsaufbau angegeben hat. Dies bestimmt, wie oft der Client Keep-Alive-Nachrichten sendet, um die Verbindung aufrechtzuerhalten.

sourceIp

Die IP-Adresse des Clients, der die Verbindung initiiert hat. Wird nur zurückgegeben, wenn auf gesetzt `includeSocketInformation` ist `true` und Sie berechtigt sind, diese Informationen abzurufen.

sourcePort

Die Portnummer, die vom Client für die Verbindung verwendet wird. Wird nur zurückgegeben, wenn auf gesetzt `includeSocketInformation` ist `true` und Sie berechtigt sind, diese Informationen abzurufen.

Ziel-IP

Die IP-Adresse, an die die Verbindungsanfrage gestellt wurde. Wird nur zurückgegeben, wenn auf gesetzt `includeSocketInformation` ist `true` und Sie berechtigt sind, diese Informationen abzurufen.

Zielport

Die Portnummer des AWS IoT Core Endpunkts, mit dem der Client eine Verbindung hergestellt hat. Wird nur zurückgegeben, wenn auf gesetzt `includeSocketInformation` ist `true` und Sie berechtigt sind, diese Informationen abzurufen.

vpc EndpointId

Die ID des VPC-Endpunktes. Vorhanden für Clients, mit denen AWS IoT Core über einen VPC-Endpunkt eine Verbindung hergestellt wurde. Wird nur zurückgegeben, wenn auf gesetzt `includeSocketInformation` ist `true` und Sie berechtigt sind, diese Informationen abzurufen.

Ablauf der Sitzung

Die Konfiguration für den dauerhaften Sitzungsablauf, die der Client beim Verbindungsaufbau angegeben oder als Standard verwendet hat. Dies bestimmt, wie lange eine Sitzung aktiv bleibt, nachdem der Client die Verbindung getrennt hat.

> **Note**
>
> Der von AWS Signature Version 4 zum Signieren von Anfragen verwendete Dienstname lautet: iotdevicegateway. Sie können Ihren Endpunkt mithilfe des Befehls finden. `aws iot describe-endpoint --endpoint-type iot:Data-ATS`

Erforderliche Berechtigungen

Um die `GetConnection` API verwenden zu können, benötigen Sie die folgende IAM-Berechtigung:

```
iot:GetConnection
```

Sie können diese Berechtigung mithilfe ressourcenbasierter Richtlinien auf bestimmte Client-IDs beschränken.

> **Tip**
>
> Verwenden Sie die `iot:IncludeSocketInformation` Bedingung, um eine granulare Zugriffskontrolle zu implementieren, die auf den Datenschutzanforderungen Ihres Unternehmens basiert.

Beispiele für Autorisierungsrichtlinien

Beispiel 1: GetConnection Mit Socket-Informationen für alle Clients zulassen

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:GetConnection",
            "Resource": "arn:aws:iot:us-east-1:987654321:client/*"
        }
    ]
}
```

Mit dieser Richtlinie gibt der folgende API-Aufruf Verbindungsdetails einschließlich Socket-Informationen zurück:

```
GET /connections/myDevice123?includeSocketInformation=true HTTP/1.1
```

Ergebnis: Die API gibt alle Verbindungsfelder zurück`sourceIp`, einschließlich `sourcePort``targetIp`, und`targetPort`.

Beispiel 2: Socket-Informationen für bestimmte Clients zulassen, GetConnection aber verweigern

Sie können entweder einen bedingungsbasierten Ansatz oder einen expliziten Deny-Ansatz verwenden:

Condition-based Ansatz:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:GetConnection",
            "Resource": "arn:aws:iot:us-east-1:987654321:client/myDevice*",
            "Condition": {
                "Bool": {
                    "iot:IncludeSocketInformation": "false"
                }
            }
```

```
        }
    ]
}
```

Expliziter Deny-Ansatz:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": "iot:GetConnection",
            "Resource": "arn:aws:iot:us-east-1:987654321:client/myDevice*",
            "Condition": {
                "Bool": {
                    "iot:IncludeSocketInformation": "true"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": ["iot:GetConnection"],
            "Resource": "arn:aws:iot:us-east-1:987654321:client/myDevice*"
        }
    ]
}
```

Bei beiden Richtlinien wird der folgende API-Aufruf verweigert:

```
GET /connections/myDevice123?includeSocketInformation=true HTTP/1.1
```

Ergebnis: Die API gibt den Fehler 403 Forbidden zurück, da der Zugriff auf Socket-Informationen für diese Clients eingeschränkt ist.

Fehlermeldungen

Die `GetConnection` API kann die folgenden Fehlerantworten zurückgeben:

InvalidRequestException

Die Anforderung ist ungültig. Dies kann auftreten, wenn das Client-ID-Format ungültig ist, ein Dollarzeichen ($) -Präfix enthält oder wenn erforderliche Parameter fehlen. HTTP-Statuscode: 400

ResourceNotFoundException

Die angegebene Client-ID existiert nicht oder ist derzeit nicht verbunden und es besteht keine persistente Sitzung. HTTP-Statuscode: 404

ForbiddenException

Der Anrufer ist nicht berechtigt, die Anfrage zu stellen. Dies kann auf unzureichende IAM-Berechtigungen oder ressourcenbasierte Richtlinieneinschränkungen zurückzuführen sein. HTTP-Statuscode: 403

ThrottlingException

Die Rate überschreitet den Grenzwert. Reduzieren Sie die Häufigkeit Ihrer API-Aufrufe und implementieren Sie eine entsprechende Wiederholungslogik mit exponentiellem Backoff. HTTP-Statuscode: 429

InternalFailureException

Ein unerwarteter Fehler ist aufgetreten. Versuchen Sie die Anfrage nach einer kurzen Verzögerung erneut. HTTP Status Code: 500

Verwaltung von Kundenabonnements

Verwenden Sie die `ListSubscriptions` API, um alle Themenabonnements für eine MQTT-Clientsitzung abzurufen. Diese API gibt Abonnements sowohl für verbundene Clients als auch für Offline-Clients mit dauerhaften Sitzungen zurück. Dadurch erhalten Sie Einblick in die Abonnementmuster der Kunden und helfen bei der Behebung von Abonnementproblemen.

Die `ListSubscriptions` API bietet umfassende Abonnementinformationen, die Ihnen helfen, das Kundenverhalten zu überprüfen, Probleme bei der Nachrichtenübermittlung zu beheben und Abonnementmuster in Ihrer gesamten Geräteflotte zu verstehen. Anhand dieser Informationen können Sie überprüfen, ob die Kunden die erwarteten Themen abonniert haben, und potenzielle Abonnementkonflikte identifizieren.

Anwendungsfälle

Die `ListSubscriptions` API hilft Ihnen bei der Behebung von Problemen bei der Nachrichtenübermittlung und bei der Überprüfung des Abonnementverhaltens Ihrer Kunden. Verwenden Sie diese API, um zu überprüfen, welche Themen ein Kunde abonniert hat, um die QoS-Stufen zu überprüfen und um sich überschneidende Abonnements oder übermäßig freizügige Whildcards zu identifizieren.

Szenarien zur Abonnementverwaltung profitieren ebenfalls von dieser API. Wenn Kunden fehlende Nachrichten oder eine unerwartete Nachrichtenzustellung melden, können Sie diese API verwenden, um ihre aktuellen Abonnements zu überprüfen und sicherzustellen, dass sie der erwarteten Konfiguration entsprechen. Dies ist besonders nützlich für das Debuggen komplexer Abonnementhierarchien und gemeinsam genutzter Abonnementkonfigurationen.

API-Parameter

Die `ListSubscriptions` API akzeptiert die folgenden Parameter:

clientId (erforderlich)

Die eindeutige Kennung des MQTT-Clients, für den Abonnements abgerufen werden sollen. Dies ist im URL-Pfad angegeben. Die Client-ID darf nicht mit einem Dollarzeichen ($) beginnen.

> **Note**
>
> MQTT-Client-IDs können Zeichen enthalten, die in HTTP-Anfragen nicht gültig sind. Wenn Sie die `ListSubscriptions` API verwenden, müssen Sie alle Zeichen in der Client-ID, die in MQTT, aber nicht in HTTP gültig sind, per URL codieren (prozentual codieren). Dazu gehören Sonderzeichen wie Leerzeichen, Schrägstriche (/) und Zeichen. UTF-8 Beispielsweise wird aus einem Leerzeichen %20, aus einem Schrägstrich %2F und aus dem UTF-8 Zeichen ü wird %C 3% BC. Durch die richtige Codierung wird sichergestellt, dass die MQTT-Client-IDs im API-Aufruf korrekt übertragen werden. HTTP-based Überlegungen zum Routing: Die URL-Codierung ist erforderlich, wenn Client-IDs sowohl Schrägstriche als auch die Zeichenfolge „Abonnements“ enthalten. Ohne eine korrekte Codierung kann die Pfadstruktur falsch interpretiert werden und Ihre Anfrage wird stattdessen an die `ListSubscriptions` API weitergeleitet. `GetConnection` Beispielsweise client/subscriptions muss eine Client-ID als client%2FSubscriptions codiert werden, um sicherzustellen, dass die Anfrage korrekt weitergeleitet wird und `GetConnection` nicht als Anruf interpretiert wird. `ListSubscriptions`

nextToken (optional)

Das Paginierungstoken zum Abrufen der nächsten Gruppe von Ergebnissen. Verwenden Sie dieses Token aus einer früheren Antwort, um mit der Paginierung großer Abonnementlisten fortzufahren.

MaxResults (optional)

Die maximale Anzahl von Abonnements, die in einer einzigen Antwort zurückgegeben werden sollen. Der gültige Bereich liegt zwischen 1 und 200. Der Standardwert ist 20.

API-Syntax

Die `ListSubscriptions` API verwendet das folgende HTTP-Anforderungsformat:

```
GET /connections/<client-id>/subscriptions?nextToken=token&maxResults=max HTTP/1.1
```

Beispielanfragen:

```
// Get subscriptions for a specific client
GET /connections/myDevice123/subscriptions HTTP/1.1

// Get subscriptions with custom page size
GET /connections/myDevice123/subscriptions?maxResults=50 HTTP/1.1

// Get next page of results using pagination token
GET /connections/myDevice123/subscriptions?nextToken=eyJhbGciOiJIUzI1NiJ9&maxResults=50
 HTTP/1.1
```

Beispielantwort:

```
{
    "subscriptions": [
        {"topicFilter": "device/myDevice123/commands", "qos": 1},
        {"topicFilter": "device/myDevice123/shadow", "qos": 0}
    ],
    "nextToken": "eyJhbGciOiJIUzI1NiJ9"
}
```

Antwortfelder:

Abonnements

Ein Array von Abonnementobjekten, die jeweils einen Themenfilter und eine QoS-Stufe enthalten.

Thema-Filter

Der MQTT-Themenfilter, den der Client abonniert hat. Dies kann Platzhalter (+, #) für den Mustervergleich beinhalten.

qos

Die Servicequalitätsstufe für das Abonnement (0 oder 1).

nextToken

Das Paginierungstoken zum Abrufen zusätzlicher Ergebnisse. Nur vorhanden, wenn mehr Abonnements abgerufen werden müssen.

> **Note**
>
> Der von AWS Signature Version 4 zum Signieren von Anfragen verwendete Dienstname lautet: iotdevicegateway. Sie können Ihren Endpunkt mithilfe des Befehls finden. `aws iot describe-endpoint --endpoint-type iot:Data-ATS`

Erforderliche Berechtigungen

Um die `ListSubscriptions` API verwenden zu können, benötigen Sie die folgende IAM-Berechtigung:

```
iot:ListSubscriptions
```

Sie können diese Berechtigung mithilfe ressourcenbasierter Richtlinien auf bestimmte Client-IDs beschränken. Beispiel:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:ListSubscriptions",
            "Resource": "arn:aws:iot:us-east-1:987654321:client/myDevice*"
        }
    ]
```

```
}
```

Fehlermeldungen

Die `ListSubscriptions` API kann die folgenden Fehlerantworten zurückgeben:

InvalidRequestException

Die Anforderung ist ungültig. HTTP-Statuscode: 400

ResourceNotFoundException

Die angegebene Ressource ist nicht vorhanden. HTTP-Statuscode: 404

ForbiddenException

Der Anrufer ist nicht berechtigt, die Anfrage zu stellen. HTTP-Statuscode: 403

ThrottlingException

Die Rate überschreitet den Grenzwert. HTTP-Statuscode: 429

InternalFailureException

Ein unerwarteter Fehler ist aufgetreten. HTTP Status Code: 500

MQTT-Verbindungsverwaltung mithilfe der Konsole

Sie können die AWS IoT Konsole verwenden, um MQTT-Geräte zu trennen und ihre Verbindungen zu verwalten. Die Konsole bietet eine intuitive Oberfläche zum Anzeigen verbundener Geräte und zum Durchführen von Trennvorgängen.

Verbundene MQTT-Clients anzeigen

Dieses Verfahren zeigt Ihnen, wie Sie aktuell verbundene MQTT-Geräte in der AWS IoT Konsole anzeigen.

Um verbundene MQTT-Geräte anzuzeigen

1. Öffnen Sie die AWS IoT -Konsole.
2. Wählen Sie im linken Navigationsbereich Verwalten und dann Alle Geräte aus.
3. Wählen Sie Client-Verbindungen, um die Liste der aktuell verbundenen MQTT-Geräte anzuzeigen.

4. Auf der Seite „Client-Verbindungen“ werden umfassende Informationen zu jedem verbundenen Gerät angezeigt. Sie können die Client-ID, die als eindeutige Kennung für das MQTT-Gerät dient, und den Verbindungsstatus einsehen, der angibt, ob das Gerät gerade verbunden ist. Auf der Seite werden auch der Zeitstempel „Verbunden seit“ angezeigt, der angibt, wann das Gerät seine Verbindung hergestellt hat, die Quell-IP-Adresse, von der aus das Gerät eine Verbindung herstellt, und das für die Geräteverbindung konfigurierte Keep-Alive-Intervall.

MQTT-Client-Abonnements anzeigen

Dieses Verfahren zeigt Ihnen, wie Sie die Themenabonnements für einen bestimmten MQTT-Client mithilfe der AWS IoT Konsole anzeigen.

Um Kundenabonnements einzusehen

1. Öffnen Sie die AWS IoT -Konsole.
2. Wählen Sie im linken Navigationsbereich Verwalten und dann Alle Geräte aus.
3. Wählen Sie Client-Verbindungen, um die Liste der MQTT-Geräte anzuzeigen.
4. Suchen Sie mithilfe der Suchfunktion nach der spezifischen Client-ID, für die Sie Abonnements anzeigen möchten.
5. Wählen Sie die Client-ID aus, um die zugehörige Detailseite anzuzeigen.
6. Suchen Sie auf der Seite mit den Kundendetails nach der Tabelle Abonnierte Themen. In dieser Tabelle werden alle Themenfilter angezeigt, die der Kunde derzeit abonniert hat, zusammen mit den entsprechenden Quality of Service (QoS) -Stufen.
7. Die Tabelle „Abonnierte Themen“ zeigt:

   - Themenfilter: Das MQTT-Themenmuster, das der Client abonniert hat. Es kann Platzhalter (+, #) enthalten
   - QoS: Die Servicequalitätsstufe für das Abonnement (0 oder 1)

> **Note**
>
> Abonnementinformationen sind sowohl für verbundene Clients als auch für Offline-Clients mit dauerhaften Sitzungen verfügbar. Wenn ein Client keine aktiven Abonnements hat, ist die Tabelle leer.

Trennen Sie ein MQTT-Gerät

Dieses Verfahren zeigt Ihnen, wie Sie ein bestimmtes MQTT-Gerät mithilfe der AWS IoT Konsole trennen.

So trennen Sie die Verbindung zu einem MQTT-Gerät

1. Suchen Sie auf der Seite mit den Client-Verbindungen das Gerät, das Sie trennen möchten.
2. Wählen Sie das Gerät aus, indem Sie auf das Optionsfeld neben der Client-ID klicken.
3. Wählen Sie „Aktionen“ und dann „Client trennen“.
4. Konfigurieren Sie im Dialogfeld Client trennen die Optionen zum Trennen der Verbindung:

    a. Bei Sitzung reinigen können Sie wählen, ob der Sitzungsstatus beibehalten oder gelöscht werden soll. Wenn Sie Sitzungsstatus beibehalten auswählen, werden die Sitzungsinformationen des Geräts, einschließlich Abonnements und Nachrichten in der Warteschlange, beibehalten, sodass das Gerät die vorherige Sitzung nach der Wiederherstellung der Verbindung fortsetzen kann. Wenn Sie alternativ Sitzungsstatus löschen wählen, werden alle Sitzungsinformationen entfernt, wenn das Gerät getrennt wird, sodass das Gerät gezwungen wird, eine komplett neue Sitzung aufzubauen, wenn es erneut verbunden wird.

    b. Bei der Meldung „Letzter Wille und Testament“ können Sie steuern, ob die LWT-Nachricht des Geräts beim Trennen der Verbindung veröffentlicht wird. Wenn Sie LWT-Nachricht zulassen wählen, wird beim Trennen der Verbindung die Meldung „Letzter Wille und Testament“ des Geräts veröffentlicht, wodurch andere Geräte über die unerwartete Trennung der Verbindung informiert werden. Wenn Sie die Option „LWT-Nachricht verhindern“ auswählen, wird die Meldung „Testament des letzten Testaments“ nicht veröffentlicht. Dies ist bei geplanten Wartungsarbeiten oder kontrollierten Verbindungsabbrüchen hilfreich.

5. Wählen Sie Trennen, um die Aktion zu bestätigen.
6. Die Konsole zeigt eine Bestätigungsmeldung an, wenn das Gerät erfolgreich getrennt wurde. Das Gerät wird nicht mehr in der Liste der verbundenen Geräte angezeigt.

> **Note**
>
> Getrennte Geräte können sofort versuchen, die Verbindung wiederherzustellen, sofern Sie keine zusätzliche Logik implementiert haben, um eine erneute Verbindung zu verhindern. Durch den Trennvorgang wird nur die aktuelle Verbindung beendet.

AWS IoT Unterschiede zu den MQTT-Spezifikationen

Die Message-Broker-Implementierung basiert auf der MQTT-Spezifikation v3.1.1 und der MQTT-Spezifikation v5.0, unterscheidet sich jedoch in folgenden Punkten von den Spezifikationen:

- AWS IoT unterstützt die folgenden Pakete für MQTT 3 nicht: PUBREC, PUBREL und PUBCOMP.
- AWS IoT unterstützt die folgenden Pakete für MQTT 5 nicht: PUBREC, PUBREL, PUBCOMP und AUTH.
- AWS IoT unterstützt keine MQTT 5-Serverumleitung.
- AWS IoT unterstützt nur die MQTT Quality of Service (QoS) -Stufen 0 und 1. AWS IoT unterstützt das Veröffentlichen oder Abonnieren mit QoS Level 2 nicht. Bei Abfragen über QoS-Ebene 2 sendet der Message Broker kein PUBACK oder SUBACK.
- In AWS IoT bedeutet das Abonnieren eines Themas mit QoS-Stufe 0, dass eine Nachricht null oder mehrmals zugestellt wird. Meldungen können mehr als einmal übertragen werden. Meldungen, die mehrmals übertragen werden, können mit unterschiedlicher Paket-ID gesendet werden. In diesem Fall wird das DUP-Flag nicht festgelegt.
- Der Message Broker sendet als Antwort auf eine Verbindungsanfrage eine CONNACK-Nachricht. Diese enthält ein Flag, das angibt, ob eine frühere Sitzung wieder aufgenommen wird.

CloudWatch Loggt AWS IoT Einträge protokollieren

Der AWS IoT Message Broker generiert einen Logeintrag mit einem `eventType` von, `Connect` wenn ein MQTT-Client eine Verbindung herstellt.

Beispiel für einen Connect-Logeintrag

```
{
    "timestamp": "2017-08-10 15:37:23.476",
    "logLevel": "INFO",
    "traceId": "20b23f3f-d7f1-feae-169f-82263394fbdb",
    "accountId": "123456789012",
```

```
    "status": "Success",
    "eventType": "Connect",
    "protocol": "MQTT",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "thingName": "myDevice",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490,
    "keepAliveDuration": 60,
    "sessionExpiry": 3600,
    "cleanSession": boolean
}
```

Zusätzlich zu den allgemeinen CloudWatch Log-Attributen enthalten Connect-Logeinträge die folgenden Attribute:

clientId

Die ID des Clients, der die Anforderung stellt.

principalId

Die ID des Prinzipals, der die Anforderung stellt.

thingName

Der Dingname des Clients. Gibt einen Wert zurück, wenn ein Ding mit der Client-ID verknüpft ist.

Protokoll

Das für die Anforderung verwendete Protokoll. Gültige Werte sind MQTT oder HTTP.

sourceIp

Die IP-Adresse, von der die Anforderung stammt.

sourcePort

Der Port, von dem die Anforderung stammt.

behalten AliveDuration

Das Keep-Alive-Intervall in Sekunden, das der Client beim Verbindungsaufbau angegeben hat.

Ablauf der Sitzung

Die Konfiguration für den dauerhaften Sitzungsablauf, die der Client beim Verbindungsaufbau angegeben hat.

CleanSession

Gibt an, ob der Client eine saubere Sitzung verwendet.

HTTPS veröffentlichen

Clients können Nachrichten veröffentlichen, indem sie Anforderungen an die REST-API mit den Protokollen HTTP 1.0 oder 1.1 stellen. Informationen zu den Authentifizierungs- und Portzuordnungen, die von HTTP-Anforderungen verwendet werden, finden Sie unter ???.

> **Note**
>
> HTTPS unterstützt keinen `clientId`-Wert wie MQTT. `clientId` ist verfügbar, wenn MQTT verwendet wird, aber es ist nicht verfügbar, wenn HTTPS verwendet wird.

HTTPS-Nachrichten-URL

Geräte und CLients veröffentlichen ihre Nachrichten, indem sie POST-Anfragen an einen clientspezifischen Endpunkt und eine themenspezifische URL stellen:

```
https://IoT_data_endpoint/topics/url_encoded_topic_name?qos=1
```

- *`IoT_data_endpoint`*ist der Endpunkt der AWS IoT Gerätedaten. Sie finden den Endpunkt in der AWS IoT Konsole auf der Detailseite der Sache oder auf dem Client, indem Sie den AWS CLI folgenden Befehl verwenden:

  ```
  aws iot describe-endpoint --endpoint-type iot:Data-ATS
  ```

  Der Endpunkt sollte etwa so aussehen: `a3qjEXAMPLEffp-ats.iot.us-west-2.amazonaws.com`
- *`url_encoded_topic_name`*ist der vollständige Themenname der gesendeten Nachricht.

Beispiele für HTTPS-Nachrichtencodes

Dies sind einige Beispiele dafür, wie Sie eine HTTPS-Nachricht an AWS IoT senden können.

Python (port 8443)

```
import requests
import argparse

# define command-line parameters
parser = argparse.ArgumentParser(description="Send messages through an HTTPS
 connection.")
parser.add_argument('--endpoint', required=True, help="Your AWS IoT data custom
 endpoint, not including a port. " +
                                                       "Ex: \"abcdEXAMPLExyz-
ats.iot.us-east-1.amazonaws.com\"")
parser.add_argument('--cert', required=True, help="File path to your client
 certificate, in PEM format.")
parser.add_argument('--key', required=True, help="File path to your private key, in
 PEM format.")
parser.add_argument('--topic', required=True, default="test/topic", help="Topic to
 publish messages to.")
parser.add_argument('--message', default="Hello World!", help="Message to publish. "
 +
                                                       "Specify empty string to
 publish nothing.")

# parse and load command-line parameter values
args = parser.parse_args()

# create and format values for HTTPS request
publish_url = 'https://' + args.endpoint + ':8443/topics/' + args.topic + '?qos=1'
publish_msg = args.message.encode('utf-8')

# make request
publish = requests.request('POST',
            publish_url,
            data=publish_msg,
            cert=[args.cert, args.key])

# print results
print("Response status: ", str(publish.status_code))
if publish.status_code == 200:
        print("Response body:", publish.text)
```

Python (port 443)

```
import requests
import http.client
import json
import ssl

ssl_context = ssl.SSLContext(protocol=ssl.PROTOCOL_TLS_CLIENT)
ssl_context.minimum_version = ssl.TLSVersion.TLSv1_2

# note the use of ALPN
ssl_context.set_alpn_protocols(["x-amzn-http-ca"])
ssl_context.load_verify_locations(cafile="./<root_certificate>")

# update the certificate and the AWS endpoint
ssl_context.load_cert_chain("./<certificate_in_PEM_Format>",
 "<private_key_in_PEM_format>")
connection = http.client.HTTPSConnection('<the ats IoT endpoint>', 443,
 context=ssl_context)
message = {'data': 'Hello, I'm using TLS Client authentication!'}
json_data = json.dumps(message)
connection.request('POST', '/topics/device%2Fmessage?qos=1', json_data)

# make request
response = connection.getresponse()

# print results
print(response.read().decode())
```

CURL

So verwenden Sie curl zum Senden einer Nachricht an das AWS IoT.

Um curl zu verwenden, um eine Nachricht von einem zu senden AWS IoT Client-Gerät

1. Überprüfen Sie die curl-Version.

   a. Führen Sie diesen Befehl auf Ihrem Client an einer Eingabeaufforderung aus.

      curl --help

      Suchen Sie im Hilfetext nach den TLS-Optionen. Sie sollten die `--tlsv1.2`-Option sehen.

b. Wenn die `--tlsv1.2`-Option angezeigt wird, fahren Sie fort.

c. Wenn die `--tlsv1.2`-Option nicht angezeigt wird oder Sie einen `command not found`-Fehler angezeigt bekommen, müssen Sie vor dem Fortfahren gegebenenfalls curl auf Ihrem Client aktualisieren oder `openssl` installieren.

2. Installieren Sie die Zertifikate auf Ihrem Client.

   Kopieren Sie die Zertifikatsdateien, die Sie bei der Registrierung Ihres Clients (Things) in der AWS IoT Konsole erstellt haben. Stellen Sie sicher, dass sich diese drei Zertifikatdateien auf Ihrem Client befinden, bevor Sie fortfahren.

   - Die CA-Zertifikatdatei (*Amazon-root-CA-1.pem* in diesem Beispiel).
   - Die Zertifikatdatei des Clients (*device.pem.crt* in diesem Beispiel).
   - Die private Schlüsseldatei des Clients (*private.pem.key* in diesem Beispiel).

3. Erstellen Sie die curl-Befehlszeile und ersetzen Sie die ersetzbaren Werte gegen die Werte Ihres Kontos und Systems.

```
curl --tlsv1.2 \
    --cacert Amazon-root-CA-1.pem \
    --cert device.pem.crt \
    --key private.pem.key \
    --request POST \
    --data "{ \"message\": \"Hello, world\" }" \
    "https://IoT_data_endpoint:8443/topics/topic?qos=1"
```

--tlsv1.2

Verwenden Sie TLS 1.2 (SSL).

--cacert *Amazon-root-CA-1.pem*

Der Dateiname und Pfad, falls erforderlich, des CA-Zertifikats für die Verifizierung des Peers.

--Zertifikat *device.pem.crt*

Der Dateiname und Pfad der Clientzertifikatdatei, falls erforderlich.

--Schlüssel *private.pem.key*

Der Dateiname und Pfad des privaten Schlüssels des Clients, falls erforderlich.

--request POST

Die Art der HTTP-Anfrage (in diesem Fall POST).

--Daten "*{ \"message\": \"Hello, world\" }*

Die HTTP POST-Daten, die Sie veröffentlichen möchten. In diesem Fall handelt es sich um eine JSON-Zeichenfolge, wobei die internen Anführungszeichen mit dem umgekehrten Schrägstrich (\) als Escape-Zeichen markiert sind.

„https://*IoT_data_endpoint*:8443/topics/? *topic* qos=1"

Die URL des AWS IoT Gerätedatenendpunkts Ihres Clients, gefolgt vom HTTPS-Port`:8443`, gefolgt vom Schlüsselwort `/topics/` und in diesem Fall dem Themennamen. `topic` Geben Sie die Servicequalität als Abfrageparameter an, `?qos=1`, an.

4. Öffnen Sie den MQTT-Testclient in der AWS IoT Konsole.

   Folgen Sie den Anweisungen in MQTT-Nachrichten anzeigen mit dem AWS IoT MQTT-Client und konfigurieren Sie die Konsole so, dass sie Nachrichten abonniert, deren Themenname in Ihrem curl Befehl *topic* verwendet wird, oder verwenden Sie den Platzhalter-Themenfilter von. #

5. Testen Sie den Befehl.

   Während Sie das Thema im Testclient der AWS IoT -Konsole überwachen, rufen Sie im Client die in Schritt 3 erstellte curl-Befehlszeile auf. Sie sollten die Nachrichten Ihres Clients in der Konsole sehen.

Direktnachrichten

AWS IoT Core unterstützt jetzt Direct Messaging. Sie können eine Nachricht über die MQTT-Client-ID an ein einzelnes verbundenes Gerät senden, ohne dass das Gerät ein Thema abonnieren muss.

Bisher erforderte das Senden einer Nachricht an ein bestimmtes Gerät die Veröffentlichung eines Themas, das das Gerät abonniert hatte, ohne dass eine integrierte Methode zur Bestätigung der Zustellung vorhanden war. Der Absender ruft die SendDirectMessage HTTP-API auf und gibt dabei die Client-ID des Empfängers und ein Zielthema an. Wenn`confirmation=true`, AWS IoT Core liefert mit QoS 1 und wartet auf den PUBACK-Wert des Empfängers, bevor eine erfolgreiche Antwort zurückgegeben wird. Dadurch erhalten Sie eine durchgängige Zustellungsbestätigung. API-Antworten und Amazon CloudWatch Logs bieten einen vollständigen Überblick über den Lieferstatus und die Fehlerursachen.

Direktnachrichten werden nicht nach AWS IoT Regeln für die Regelausführung verarbeitet, für Geräte, die offline sind, nicht in die Warteschlange gestellt und unterstützen keine gespeicherten Nachrichten.

In diesem Thema:

- Voraussetzungen
- SendDirectMessage API
- Verhalten des Receiver-Clients

Voraussetzungen

Sowohl der Absender als auch der Empfänger benötigen bestimmte Richtlinien, um Direktnachrichten verwenden zu können. Der Absender muss die `iot:SendDirectMessage` Erlaubnis haben. Die Ziel-Client-ID wird als Ressource angegeben und der `iot:Topic` Bedingungsschlüssel (optional) schränkt ein, an welche Themen ein Absender Direktnachrichten senden kann. Der Empfänger muss über die `iot:Receive` Genehmigung für das Zielthema verfügen. Der Empfänger benötigt keine `iot:Subscribe` Genehmigung — AWS IoT Core übermittelt Direktnachrichten, ohne dass ein Themenabonnement erforderlich ist. Weitere Informationen und Beispielrichtlinien finden Sie unterBeispiele für Richtlinien für Direktnachrichten.

Informationen zu den Authentifizierungs- und Portzuordnungen, die von HTTP-Anforderungen verwendet werden, finden Sie unter ???.

SendDirectMessage API

Absender können Direktnachrichten senden, indem sie HTTP-POST-Anfragen an eine clientspezifische URL stellen:

```
https://IoT_data_endpoint/connections/client_id/messages?
topic=topic_name&confirmation=true&timeout=10
```

- *IoT_data_endpoint*ist der AWS IoT Endpunkt der Gerätedaten. Suchen Sie unterthe section called “AWS IoT Gerätedaten und Dienstendpunkte”, um Ihren Endpunkt zu finden.
- *client_id*ist die eindeutige Kennung des MQTT-Clients, an den die Nachricht gesendet werden soll. Client-IDs dürfen 128 Zeichen nicht überschreiten und dürfen nicht mit einem Dollarzeichen ($) beginnen. MQTT-Client-IDs müssen URL-codiert (prozentcodiert) sein, wenn sie Zeichen enthalten, die in HTTP-Anfragen nicht gültig sind, wie Leerzeichen, Schrägstriche (/) und Zeichen. UTF-8

Weitere Informationen finden Sie unter Grenzwerte und Kontingente für AWS IoT Core Message-Broker und Protokolle.

- *topic_name*ist das Thema, zu dem der Empfänger die Nachricht empfängt, URL-encoded. Darf nicht mit $ beginnen. Darf kein AWS IoT Core reserviertes Thema sein. Informationen zu Längen- und Tiefenbeschränkungen für Themen finden Sie auf der Seite mit den AWS IoT Core Servicekontingenten. Weitere Informationen finden Sie unter Grenzwerte und Kontingente für AWS IoT Core Message Broker und Protokolle.
- *confirmation*ist ein boolescher Wert. Bei Einstellung auf übermittelt die API die Nachricht mit QoS 1 und wartet darauf`true`, dass der MQTT-Client eine Übermittlungsbestätigung (PUBACK) sendet, bevor eine erfolgreiche Antwort zurückgegeben wird. Wenn die Lieferbestätigung nicht innerhalb des angegebenen Timeout-Zeitraums eingeht, gibt die API HTTP 504 zurück.
- *timeout*ist eine Ganzzahl, die die maximale Wartezeit in Sekunden auf eine Zustellungsbestätigung (PUBACK) vom empfangenden Client nach Zustellung der Nachricht angibt. Dieser Parameter wird nur verwendet, wenn er auf `true` gesetzt `confirmation` ist. Wenn `confirmation` ja`false`, wird dieser Parameter ignoriert. Die gesamte API-Antwortzeit kann aufgrund interner Verarbeitung höher als dieser Wert sein. Stellen Sie Ihr HTTP-Client-Timeout auf einen Wert ein, der größer als dieser Parameter ist.

Statuscodes der API-Antwort

In der folgenden Tabelle sind die von der SendDirectMessage API zurückgegebenen HTTP-Statuscodes und die jeweils empfohlenen Aktionen aufgeführt. Aktivieren Sie AWS IoT Core CloudWatch Protokolle, um detaillierte SendDirectMessage Ereignisprotokolle einschließlich des Felds „Grund“ für die programmatische Fehlerbehandlung anzuzeigen.

SendDirectMessage Statuscodes für API-Antworten

HTTP-Code	Empfohlene Aktion
200 OK	Wenn die Zustellungsbestätigung mit angeforde rt wurde`confirmation=true` , bedeutet dies, dass der Empfänger den Nachricht enempfang bestätigt hat. Andernfalls bedeutet dies, dass die Nachricht erfolgreich versendet wurde.

HTTP-Code	Empfohlene Aktion
400 Bad Request (400 Ungültige Anfrage)	Das bedeutet, dass einer der Parameter ungültig ist. Überprüfen Sie die HTTP-Antwortnachricht oder die CloudWatch Protokolle, um bestimmte Fehler zu identifizieren und zu beheben. Stellen Sie sicher, dass der Themenname und Client-id die Themen gültig und URL-encoded korrekt sind.
403 Forbidden	Dies bedeutet, dass die Richtlinie des Absenders für den Zielclient und das Zielthema keine Gewährung gewährt`iot:SendDirectMessage`, oder dass die Richtlinie des Empfängers keine Gewährung für das `iot:Receive` Thema gewährt. Überprüfen Sie die HTTP-Antwortnachricht oder die CloudWatch Protokolle, um bestimmte Fehler zu identifizieren, und aktualisieren Sie die entsprechende Richtlinie. Siehe Beispiele für Richtlinien für Direktnachrichten.
404 Not Found (404 Nicht gefunden)	Das bedeutet, dass die Zielclient-ID nicht mit verbunden ist AWS IoT Core. Überprüfen Sie die HTTP-Antwortnachricht oder die CloudWatch Protokolle auf den spezifischen Grund, stellen Sie sicher, dass der Empfänger verbunden ist, und versuchen Sie es erneut. Wenn in der Antwortnachricht „Die Zielclient-ID ist nicht verbunden, aber sie hat eine aktive persistente Sitzung“ lautet, hat der Zielclient eine noch nicht abgelaufene persistente Sitzung, ist aber derzeit offline.

HTTP-Code	Empfohlene Aktion
4.1.3 Payload: Zu groß	Die Nutzlast überschreitet die maximal zulässige Größe. Reduzieren Sie die Nutzlastgröße und versuchen Sie es erneut. Siehe AWS IoT Core -Servicekontingente.
429 Zu viele Anfragen	Dies bedeutet, dass das Konto das Limit für SendDirectMessage Anfragen pro Sekunde überschritten hat oder dass die Empfängerverbindung das Limit für ausgehende Veröffentlichungen überschritten hat. Prüfen Sie die HTTP-Antwortnachricht oder die CloudWatch Protokolle auf den spezifischen Grund, reduzieren Sie die Anforderungsrate und implementieren Sie exponentielles Backoff. Siehe AWS IoT Core -Servicekontingente.
500 Internal Server Error	Dies deutet auf einen unerwarteten serverseitigen Fehler hin. Versuchen Sie die Anfrage mit exponentiellem Backoff erneut. Wenn das Problem weiterhin besteht, wenden Sie sich mit der TraceID aus der Antwort an den AWS Support.
504 Gateway-Timeout	Dies bedeutet, dass der Empfänger PUBACK nicht innerhalb des angegebenen Timeout-Zeitraums gesendet hat. Erhöhen Sie den Timeout-Wert, überprüfen Sie, ob der MQTT-Client des Empfängers PUBACK für QoS 1-Nachrichten sendet, oder überprüfen Sie, ob der Empfänger Nachrichten langsam verarbeitet.

Beispiele

AWS CLI

```
aws iot-data send-direct-message \
    --client-id myDevice \
    --topic commands/reboot \
    --confirmation \
    --timeout 10 \
    --payload '{"action": "reboot"}' \
    --cli-binary-format raw-in-base64-out \
    --region us-west-2 \
    --endpoint-url https://IoT_data_endpoint
```

Die `--cli-binary-format` Option ist erforderlich, wenn Sie Version 2 verwenden. AWS Command Line Interface Um dies zur Standardeinstellung zu machen, führen Sie `aws configure set cli-binary-format raw-in-base64-out` aus. Weitere Informationen finden Sie unter Von AWS CLI unterstützte globale Befehlszeilenoptionen im AWS Command Line Interface -Benutzerhandbuch für Version 2.

curl (X.509 client certificate, port 8443)

```
curl --tlsv1.2 \
    --cacert Amazon-root-CA-1.pem \
    --cert device.pem.crt \
    --key private.pem.key \
    --request POST \
    --data '{"action": "reboot"}' \
    "https://IoT_data_endpoint:8443/connections/myDevice/messages?topic=commands
%2Freboot&confirmation=true&timeout=10"
```

Verhalten des Receiver-Clients

Direct Messaging übermittelt Nachrichten an MQTT-Clients (Empfänger), ohne dass ein Themenabonnement erforderlich ist. Um alle Vorteile von Direct Messaging nutzen zu können, muss der Empfänger die folgenden Verhaltensweisen unterstützen:

- Nachrichten zu Themen empfangen, die nicht explizit abonniert wurden — Das Direct Messaging des Empfängers kann Nachrichten zu Themen senden, die der Empfänger nicht explizit abonniert hat. Einige MQTT-Client-Implementierungen filtern jedoch Nachrichten zu abgemeldeten Themen oder verwerfen sie. Wenn Ihr Client diese Nachrichten verwirft, funktionieren Direktnachrichten nur

zu Themen, die der Empfänger ebenfalls abonniert hat. Um Direktnachrichten zu einem beliebigen Thema zu erhalten, stellen Sie sicher, dass der Nachrichtenhandler Ihres Kunden Nachrichten unabhängig vom Abonnementstatus verarbeitet.

- Von der API festgelegte QoS verarbeiten — Die QoS-Stufe der übermittelten Nachricht wird durch den `confirmation` Parameter in der API-Anforderung des Absenders festgelegt, nicht durch das Abonnement des Empfängers. Wenn die Nachricht QoS 1 erreicht`confirmation=true`, muss der Client des Empfängers ein PUBACK senden, um die Zustellung zu bestätigen. Wenn`confirmation=false`, erreicht die Nachricht QoS 0, ohne dass eine Bestätigung erforderlich ist. Stellen Sie sicher, dass die MQTT-Implementierung Ihres Clients sowohl eingehende QoS 0- als auch QoS 1-Nachrichten korrekt verarbeitet.

MQTT-Themen

MQTT-Themen identifizieren AWS IoT Nachrichten. AWS IoT Clients identifizieren die Nachrichten, die sie veröffentlichen, indem sie den Nachrichten Themennamen geben. Clients identifizieren die Nachrichten, die sie abonnieren (empfangen) möchten, indem sie einen Themenfilter bei AWS IoT Core registrieren. Der Message Broker verwendet Themennamen und Themenfilter, um Nachrichten von veröffentlichenden Clients an abonnierende Clients zu senden.

Der Message Broker verwendet Topics, um Nachrichten zu identifizieren, die über MQTT und über HTTP an die HTTPS-Nachrichten-URL gesendet wurden.

AWS IoT Unterstützt zwar einige reservierte Systemthemen, die meisten MQTT-Themen werden jedoch von Ihnen, dem Systemdesigner, erstellt und verwaltet. AWS IoT verwendet Themen, um Nachrichten zu identifizieren, die von Veröffentlichungsclients empfangen wurden, und wählt Nachrichten aus, die an abonnierende Clients gesendet werden sollen, wie in den folgenden Abschnitten beschrieben. Bevor Sie einen Themen-Namespace für Ihr System erstellen, überprüfen Sie die Merkmale von MQTT-Themen, um die Hierarchie der Themennamen zu erstellen, die für Ihr IoT-System am besten geeignet ist.

Themennamen

Themennamen und Themenfilter sind UTF-8 kodierte Zeichenfolgen. Sie können eine Hierarchie von Informationen darstellen, indem Sie den Schrägstrich (/) verwenden, um die Ebenen der Hierarchie zu trennen. Dieser Themenname könnte sich beispielsweise auf einen Temperatursensor in Raum 1 beziehen:

- `sensor/temperature/room1`

In diesem Beispiel kann es auch andere Arten von Sensoren in anderen Räumen mit Themennamen geben, z. B.:

- `sensor/temperature/room2`
- `sensor/humidity/room1`
- `sensor/humidity/room2`

> **Note**
>
> Beachten Sie bei der Betrachtung von Themennamen für die Nachrichten in Ihrem System Folgendes:
>
> - Themennamen und Themenfilter berücksichtigen Groß- und Kleinschreibung.
> - Themennamen dürfen keine personenbezogenen Informationen enthalten.
> - Themennamen, die mit einem "$" beginnen, sind reservierte Themen, die nur von AWS IoT Core verwendet werden können.
> - AWS IoT Core Ich kann keine Nachrichten zwischen AWS-Konto s oder Regionen senden oder empfangen.

Weitere Informationen zum Entwurf von Themennamen und Ihres Namespaces finden Sie in unserem Whitepaper Entwerfen von MQTT-Themen für AWS IoT Core.

Beispiele dafür, wie Apps Nachrichten veröffentlichen und abonnieren können, finden Sie am Anfang mit Erste Schritte mit AWS IoT Core Tutorials und AWS IoT Geräte-SDKs, SDKs für Mobilgeräte und AWS IoT Geräte-Client.

> **Important**
>
> Der Themen-Namespace ist auf eine AWS-Konto UND-Region beschränkt. Beispielsweise unterscheidet sich das `sensor/temp/room1` Thema, das von einem AWS-Konto in einer Region verwendet wird, von dem `sensor/temp/room1` Thema, das von demselben AWS Konto in einer anderen Region verwendet wird oder von einem anderen AWS-Konto in einer beliebigen Region verwendet wird.

Thema-ARN

Alle Themen-ARNs (Amazon Resource Names) haben das folgende Format.

```
arn:aws:iot:aws-region:AWS-account-ID:topic/Topic
```

Zum Beispiel ist `arn:aws:iot:us-west-2:123EXAMPLE456:topic/application/topic/device/sensor` ein ARN für das Thema `application/topic/device/sensor`.

Filter für Themennamen

Abonnierende Kunden registrieren Filter für Themennamen beim Message Broker, um die Nachrichtenthemen anzugeben, die der Message Broker an sie senden soll. Ein Themennamenfilter kann ein einzelner Themenname sein, um einen einzelnen Themennamen zu abonnieren, oder er kann Platzhalterzeichen enthalten, um mehrere Themennamen gleichzeitig zu abonnieren.

Veröffentlichende Clients können keine Platzhalterzeichen in den von ihnen veröffentlichten Themennamen verwenden.

In der folgenden Tabelle sind die Platzhalterzeichen aufgeführt, die in einem Themenfilter verwendet werden können.

Themenplatzhalter

Platzhalterzeichen	Entspricht	Hinweise
#	Alle Zeichenfolgen auf und unter seiner Ebene in der Themenhierarchie.	Muss das letzte Zeichen im Themenfilter sein. Muss das einzige Zeichen auf seiner Ebene der Themenhierarchie sein. Kann in einem Themenfilter verwendet werden, der auch das Platzhalterzeichen "+" enthält.
+	Jede Zeichenfolge in der Ebene, die das Zeichen enthält.	Muss das einzige Zeichen auf seiner Ebene der Themenhierarchie sein.

Platzhalterzeichen	Entspricht	Hinweise
		Kann in mehreren Ebenen eines Themenfilters verwendet werden.

Verwenden von Platzhaltern mit den vorherigen Beispielen der Sensor-Themennamen:

- Ein Abonnement für `sensor/#` empfängt Nachrichten, die für `sensor/`, `sensor/temperature` oder `sensor/temperature/room1` veröffentlicht werden, nicht jedoch Nachrichten, die für `sensor` veröffentlicht werden.
- Ein Abonnement für `sensor/+/room1` empfängt Nachrichten, die für `sensor/temperature/room1` und `sensor/humidity/room1` veröffentlicht wurden, aber keine Nachrichten, die an `sensor/temperature/room2` oder `sensor/humidity/room2` gesendet wurden.

Themenfilter-ARN

Alle Themenfilter-ARNs (Amazon Resource Names) haben das folgende Format:

```
arn:aws:iot:aws-region:AWS-account-ID:topicfilter/TopicFilter
```

`arn:aws:iot:us-west-2:123EXAMPLE456:topicfilter/application/topic/+/sensor` ist beispielsweise ein ARN für den Themenfilter `application/topic/+/sensor`.

Nutzlast von MQTT-Nachrichten

Die Nachrichtennutzlast, die in Ihren MQTT-Nachrichten gesendet wird, wird nicht von angegeben, es sei denn AWS IoT, sie bezieht sich auf eine der. the section called “Reservierte Themen” Um den Anforderungen Ihrer Anwendung gerecht zu werden, empfehlen wir Ihnen, die Nachrichtennutzlast für Ihre Themen innerhalb der Einschränkungen der AWS IoT Core Service Quotas für Protokolle zu definieren.

Die Verwendung eines JSON-Formats für Ihre Nachrichtennutzlast ermöglicht es der AWS IoT Regel-Engine, Ihre Nachrichten zu analysieren und SQL-Abfragen darauf anzuwenden. Wenn Ihre Anwendung die Regel-Engine nicht benötigt, um SQL-Abfragen auf Ihre Nachrichtennutzlasten anzuwenden, können Sie jedes Datenformat verwenden, das Ihre Anwendung benötigt. Hinweise zu Einschränkungen und reservierten Zeichen in einem JSON-Dokument, das in SQL-Abfragen verwendet wird, finden Sie unter JSON-Erweiterungen.

Weitere Informationen zum Entwerfen Ihrer MQTT-Themen und der entsprechenden Nachrichtennutzlasten finden Sie unter Entwurf von MQTT-Themen für AWS IoT Core.

Überschreitet eine Nachricht das Größenlimit des Dienstes, führt dies zu einem `CLIENT_ERROR` mit dem Grund `PAYLOAD_LIMIT_EXCEEDED` und "Die Nutzlast der Nachricht überschreitet das Größenlimit für den Nachrichtentyp". Weitere Informationen zur Größenbeschränkung für Nachrichten finden Sie unter AWS IoT Core Grenzwerte und Kontingente für Message Broker.

Reservierte Themen

Themen, die mit einem Dollarzeichen ($) beginnen, sind für die Verwendung durch reserviert AWS IoT. Sie können diese reservierten Themen abonnieren und veröffentlichen, wenn sie dies zulassen. Sie können jedoch keine neuen Themen erstellen, die mit einem Dollarzeichen beginnen. Nicht unterstützte Veröffentlichungs- oder Abonnementvorgänge für reservierte Themen können zu einer abgebrochenen Verbindung führen.

Komponentenmodell-Themen

Thema	Zulässige Client-Operationen	Description
$ aws/sitewise /asset-models/ /assets/ *assetModelId* /properties/ *assetId* *propertyId*	Abonnieren	AWS IoT SiteWise veröffentlicht Benachrichtigungen zu Asset-Eigenschaften zu diesem Thema. Weitere Informationen finden Sie im AWS IoT SiteWise Benutzerhandbuch unter Interagieren mit anderen AWS Diensten.

AWS IoT Device Defender topics

Diese Nachrichten unterstützen Antwortpuffer im Format Concise Binary Object Representation (CBOR) und JavaScript Object Notation (JSON), je nach Thema. *payload-format* AWS IoT Device Defender Themen unterstützen nur MQTT-Veröffentlichungen.

payload-format	Datentyp des Antwortformats
cbor	Concise Binary Object Representation (CBOR)
json	JavaScript Objektnotation (JSON)

Weitere Informationen finden Sie unter Metriken von Geräten senden.

Thema	Zulässige Operationen	Description
$aws/things/*thingName* /defender/metrics/*payload-format*	Veröffentlichen	AWS IoT Device Defender Agenten veröffentlichen Metriken zu diesem Thema. Weitere Informationen finden Sie unter Messwerte von Geräten senden.
$aws/things//*thingName* defender/metrics/*payload-format* /akzeptiert	Abonnieren	AWS IoT veröffentlicht zu diesem Thema, nachdem ein AWS IoT Device Defender Agent eine erfolgreiche Nachricht an $aws/things//*thingName* defender/metrics/*payload-format* veröffentlicht hat. Weitere Informationen finden Sie unter Messwerte von Geräten senden.
$aws/things//*thingName* defender/metrics/*payload-format* /abgelehnt	Abonnieren	AWS IoT veröffentlicht zu diesem Thema, nachdem ein AWS IoT Device Defender Agent eine erfolglose Nachricht an $aws/things//*thingName* defender/metrics/*payload-format* veröffentlicht hat. Weitere Informationen finden Sie unter Messwerte von Geräten senden.

AWS IoT Core Themen zum Standort des Geräts

AWS IoT Core Der Gerätestandort kann die Messdaten Ihres Geräts auflösen und einen geschätzten Standort Ihrer IoT-Geräte angeben. Die Messdaten des Geräts können GNSS- Wi-Fi, Mobilfunk-

und IP-Adressen enthalten. AWS IoT Core Der Gerätestandort wählt dann den Messtyp aus, der die beste Genauigkeit bietet, und berechnet die Standortinformationen des Geräts. Weitere Informationen erhalten Sie unter AWS IoT Core Standort des Geräts und Auflösen des Gerätestandorts mit AWS IoT Core MQTT-Themen zum Gerätestandort.

Thema	Zulässige Operationen	Description
$aws/device_location//get_position_estimate*customer_device_id*	Veröffentlichen	Ein Gerät veröffentlicht zu diesem Thema, um die gescannten Rohmessdaten abzurufen, die nach Gerätestandort aufgelöst werden sollen. AWS IoT Core
$aws/device_location/*customer_device_id*/get_position_estimate/accepted	Abonnieren	AWS IoT Core Gerätestandort veröffentlicht zu diesem Thema, nachdem der Gerätestandort erfolgreich ermittelt wurde.
$aws/device_location/*customer_device_id*/get_position_estimate/rejected	Abonnieren	AWS IoT Core Der Gerätestandort veröffentlicht zu diesem Thema, wenn der Gerätestandort aufgrund von 4xx-Fehlern nicht erfolgreich ermittelt werden kann.

Ereignisthemen

Die Ereignismeldungen werden veröffentlicht, wenn bestimmte Ereignisse eintreten. Beispielsweise werden Ereignisse von der Registry generiert, wenn Geräte hinzugefügt, aktualisiert oder gelöscht werden. Die Tabelle zeigt die verschiedenen AWS IoT Ereignisse und ihre reservierten Themen.

Thema	Zulässige Client-Operationen	Description
$aws/events/certificates/registered/*caCertificateId*	Abonnieren	AWS IoT veröffentlicht diese Meldung, wenn ein Zertifikat AWS IoT automatisch registriert wird und wenn ein Client ein Zertifikat mit dem `PENDING_ACTIVATION` Status vorlegt. Weitere

Thema	Zulässige Client-Operationen	Description
		Informationen finden Sie unter the section called “Konfigurieren der ersten Verbindung durch einen Client für die automatische Registrierung”.
$ aws/events /job/ /storniert *jobID*	Abonnieren	AWS IoT veröffentlicht diese Nachricht, wenn ein Job storniert wird. Weitere Informationen finden Sie unter Auftragsereignisse.
$ aws/events /job/ /cancellation_in_progress *jobID*	Abonnieren	AWS IoT veröffentlicht diese Nachricht, wenn eine Jobstornierung im Gange ist. Weitere Informationen finden Sie unter Auftragsereignisse.
$ aws/events /job/ /completed *jobID*	Abonnieren	AWS IoT veröffentlicht diese Nachricht, wenn ein Job abgeschlossen ist. Weitere Informationen finden Sie unter Auftragsereignisse.
$ aws/events /job/ /deleted *jobID*	Abonnieren	AWS IoT veröffentlicht diese Nachricht, wenn ein Job gelöscht wird. Weitere Informationen finden Sie unter Auftragsereignisse.
$ aws/events /job/ /deletion_in_progress *jobID*	Abonnieren	AWS IoT veröffentlicht diese Meldung, wenn ein Job gerade gelöscht wird. Weitere Informationen finden Sie unter Auftragsereignisse.
$ aws/events /JobExecution/ /storniert *jobID*	Abonnieren	AWS IoT veröffentlicht diese Nachricht, wenn die Ausführung eines Jobs abgebrochen wird. Weitere Informationen finden Sie unter Auftragsereignisse.

Thema	Zulässige Client-Operationen	Description
$ aws/events /JobExecution/ /deleted *jobID*	Abonnieren	AWS IoT veröffentlicht diese Nachricht, wenn eine Jobausführung gelöscht wird. Weitere Informationen finden Sie unter Auftragsereignisse.
$ aws/events /JobExecution/ /failed *jobID*	Abonnieren	AWS IoT veröffentlicht diese Meldung, wenn die Ausführung eines Jobs fehlgeschlagen ist. Weitere Informationen finden Sie unter Auftragsereignisse.
$ aws/events /JobExecution/ /rejected *jobID*	Abonnieren	AWS IoT veröffentlicht diese Nachricht, wenn eine Jobausführung abgelehnt wurde. Weitere Informationen finden Sie unter Auftragsereignisse.
$ aws/events /JobExecution/ /removed *jobID*	Abonnieren	AWS IoT veröffentlicht diese Nachricht, wenn eine Jobausführung entfernt wurde. Weitere Informationen finden Sie unter Auftragsereignisse.
$ aws/events /JobExecution/ /successfully *jobID*	Abonnieren	AWS IoT veröffentlicht diese Nachricht, wenn eine Jobausführung erfolgreich war. Weitere Informationen finden Sie unter Auftragsereignisse.
$ aws/events /JobExecution/ /timed_out *jobID*	Abonnieren	AWS IoT veröffentlicht diese Meldung, wenn bei einer Jobausführung ein Timeout aufgetreten ist. Weitere Informationen finden Sie unter Auftragsereignisse.
$aws/events/presence/connected/*clientId*	Abonnieren	AWS IoT veröffentlicht zu diesem Thema, wenn ein MQTT-Client mit der angegebenen Client-ID eine Verbindung herstellt. AWS IoT Weitere Informationen finden Sie unter Connect/Disconnect Ereignisse.

Thema	Zulässige Client-Operationen	Description
$aws/events/presence/disconnected/*clientId*	Abonnieren	AWS IoT veröffentlicht zu diesem Thema, wenn ein MQTT-Client mit der angegebenen Client-ID die Verbindung zu trennt. AWS IoT Weitere Informationen finden Sie unter Connect/Disconnect Ereignisse.
$aws/events/subscriptions/subscribed/*clientId*	Abonnieren	AWS IoT veröffentlicht zu diesem Thema, wenn ein MQTT-Client mit der angegebenen Client-ID ein MQTT-Thema abonniert. Weitere Informationen finden Sie unter Subscribe/Unsubscribe Ereignisse.
$aws/events/subscriptions/unsubscribed/*clientId*	Abonnieren	AWS IoT veröffentlicht zu diesem Thema, wenn sich ein MQTT-Client mit der angegebenen Client-ID von einem MQTT-Thema abmeldet. Weitere Informationen finden Sie unter Subscribe/Unsubscribe Ereignisse.
$ /thing/ /created aws/events *thingName*	Abonnieren	AWS IoT veröffentlicht zu diesem Thema, wenn das *thingName* Ding erstellt wird. Weitere Informationen finden Sie unter the section called “Registry-Ereignisse”.
$ aws/events /thing/ / updated *thingName*	Abonnieren	AWS IoT veröffentlicht zu diesem Thema, wenn das *thingName* Ding aktualisiert wird. Weitere Informationen finden Sie unter the section called “Registry-Ereignisse”.
$ aws/events /thing/ / deleted *thingName*	Abonnieren	AWS IoT veröffentlicht zu diesem Thema, wenn das *thingName* Ding gelöscht wird. Weitere Informationen finden Sie unter the section called “Registry-Ereignisse”.

Thema	Zulässige Client-Operationen	Description
$ aws/events /ThingGroup/ /created *thingGroupName*	Abonnieren	AWS IoT veröffentlicht zu diesem Thema, wenn eine Dinggruppe erstellt wird. *thingGroupName* Weitere Informationen finden Sie unter the section called “Registry-Ereignisse”.
$ aws/events /thingGroup/ /updated *thingGroupName*	Abonnieren	AWS IoT veröffentlicht zu diesem Thema, wenn die Dinggruppe aktualisiert wird. *thingGroupName* Weitere Informationen finden Sie unter the section called “Registry-Ereignisse”.
$ aws/events /ThingGroup/ /deleted *thingGroupName*	Abonnieren	AWS IoT veröffentlicht zu diesem Thema, wenn eine Dinggruppe gelöscht wird. *thingGroupName* Weitere Informationen finden Sie unter the section called “Registry-Ereignisse”.
$ aws/events /ThingType/ /created *thingTypeName*	Abonnieren	AWS IoT veröffentlicht zu diesem Thema, wenn der Dingtyp erstellt wird*thingTypeName* . Weitere Informationen finden Sie unter the section called “Registry-Ereignisse”.
$ aws/events /thingType/ /updated *thingTypeName*	Abonnieren	AWS IoT veröffentlicht zu diesem Thema, wenn der Dingtyp aktualisiert wird*thingTypeName* . Weitere Informationen finden Sie unter the section called “Registry-Ereignisse”.
$ aws/events /ThingType/ /deleted *thingTypeName*	Abonnieren	AWS IoT veröffentlicht zu diesem Thema, wenn der Dingtyp gelöscht wird*thingTypeName* . Weitere Informationen finden Sie unter the section called “Registry-Ereignisse”.

Thema	Zulässige Client-Operationen	Description
$aws/events/thingTypeAssociation/thing/*thingName*/*thingTypeName*	Abonnieren	AWS IoT veröffentlicht zu diesem Thema, wenn eine Sache mit einem Dingtyp *thingTypeName* verknüpft *thingName* ist oder die Verknüpfung mit diesem aufgehoben wird. Weitere Informationen finden Sie unter the section called “Registry-Ereignisse”.
$aws/events//thingGroupMembership/thingGroup/thing/*thingGroupName*/added*thingName*	Abonnieren	AWS IoT veröffentlicht zu diesem Thema, wenn ein Ding zur *thingName* Dinggruppe hinzugefügt wird. *thingGroupName* Weitere Informationen finden Sie unter the section called “Registry-Ereignisse”.
$aws/events//thingGroupMembership/thingGroup*thingGroupName*/thing/ /removed*thingName*	Abonnieren	AWS IoT veröffentlicht zu diesem Thema, wenn ein Ding aus der *thingName* Dinggruppe entfernt wird. *thingGroupName* Weitere Informationen finden Sie unter the section called “Registry-Ereignisse”.
$aws/events/thingGroupHierarchy/thingGroup/*parentThingGroupName*/childThingGroup/*childThingGroupName*/added	Abonnieren	AWS IoT veröffentlicht zu diesem Thema, wenn eine Dinggruppe zur Dinggruppe hinzugefügt *childThingGroupName* wird. *parentThingGroupName* Weitere Informationen finden Sie unter the section called “Registry-Ereignisse”.

Thema	Zulässige Client-Operationen	Description
$aws/events/thingGroupHierarchy/thingGroup/*parentThingGroupName* /childThingGroup/*childThingGroupName* /removed	Abonnieren	AWS IoT veröffentlicht zu diesem Thema, wenn eine Dinggruppe aus der Dinggruppe entfernt *childThingGroupName* wird. *parentThingGroupName* Weitere Informationen finden Sie unter the section called “Registry-Ereignisse”.

Flottenbereitstellungsthemen

> **Note**
>
> Die Client-Operationen, die in dieser Tabelle als Empfangen vermerkt sind, geben Themen an, die direkt an den Client AWS IoT veröffentlicht werden, der sie angefordert hat, unabhängig davon, ob der Client das Thema abonniert hat oder nicht. Kunden sollten damit rechnen, diese Antwortnachrichten zu erhalten, auch wenn sie sie nicht abonniert haben. Diese Antwortnachrichten werden nicht über den Message Broker weitergeleitet und können auch nicht von anderen Clients oder Regeln abonniert werden.

Diese Nachrichten unterstützen Antwortpuffer im Format Concise Binary Object Representation (CBOR) und in JavaScript Object Notation (JSON), je nach Thema. *payload-format*

payload-format	Datentyp des Antwortformats
cbor	Concise Binary Object Representation (CBOR)
json	JavaScript Objektnotation (JSON)

Weitere Informationen finden Sie unter MQTT-API für die Gerätebereitstellung.

Thema	Zulässige Client-Operationen	Description
$ aws/certificates /create/ *payload-format*	Veröffentlichen	Veröffentlichen Sie in diesem Thema, um ein Zertifikat über eine Zertifikatssignierungsanforderung (Certificate Signing Request, CSR) zu erstellen.
$ /create/ /accepted aws/certificates *payload-format*	Abonnieren, Empfangen	AWS IoT veröffentlicht nach einem erfolgreichen Aufruf von $ /create/ zu diesem Thema. aws/certificates *payload-format*
$ aws/certificates /create/ / rejected *payload-format*	Abonnieren, Empfangen	AWS IoT veröffentlicht zu diesem Thema nach einem erfolglosen Aufruf von $ / create/. aws/certificates *payload-format*
$ /create-from-csr/ aws/certificates *payload-format*	Veröffentlichen	Veröffentlicht in diesem Thema, um ein Zertifikat aus einer CSR zu erstellen.
$ aws/certificates *payload-format* / create-from-csr/ /accepted	Abonnieren, Empfangen	AWS IoT veröffentlicht zu diesem Thema einen erfolgreichen Aufruf von $ /create-from-csr/. aws/certificates *payload-format*
$ /create-from-csr/ /rejected aws/certificates *payload-format*	Abonnieren, Empfangen	AWS IoT veröffentlicht zu diesem Thema einen erfolglosen Aufruf von $ /create-from-csr/. aws/certificates *payload-format*
$ -templates/ /provision/ aws/provisioning *templateName payload-format*	Veröffentlichen	Veröffentlichen Sie in diesem Thema, um ein Objekt zu registrieren.
$ aws/provisioning -templates/ *templateN*	Abonnieren, Empfangen	AWS IoT veröffentlicht zu diesem Thema nach einem erfolgreichen Aufruf von $ -

Thema	Zulässige Client-Operationen	Description
ame /provision/ *payload-format* /accepted		templates/ /provision/. aws/provisioning *templateName* *payload-format*
$ -templates/ /provisio n/ /rejected aws/provi sioning *templateName* *payload-format*	Abonnieren, Empfangen	AWS IoT veröffentlicht zu diesem Thema nach einem erfolglosen Aufruf von $ -templates/ /provision/. aws/provisioning *templateName* *payload-format*

Auftragsthemen

> **Note**
>
> Die Client-Operationen, die in dieser Tabelle als Receive aufgeführt sind, geben Themen an, die direkt an den Client AWS IoT veröffentlicht werden, der sie angefordert hat, unabhängig davon, ob der Client das Thema abonniert hat oder nicht. Kunden sollten damit rechnen, diese Antwortnachrichten zu erhalten, auch wenn sie sie nicht abonniert haben.
> Diese Antwortnachrichten werden nicht über den Message Broker weitergeleitet und können auch nicht von anderen Clients oder Regeln abonniert werden. Verwenden Sie die `notify` und `notify-next` Themen und, um Nachrichten zu Auftragsaktivitäten zu abonnieren.
> Wenn Sie die Auftrags- und `jobExecution` Veranstaltungsthemen für Ihre Flottenüberwachungslösung abonnieren, müssen Sie zunächst die Auftrags- und Auftragsausführungsereignisse aktivieren, um alle Ereignisse auf der Cloud-Seite empfangen zu können.
> Weitere Informationen finden Sie unter Führt MQTT-API-Operationen auf dem Gerät aus.

Thema	Zulässige Client-Operationen	Description
$aws/things/*thingName* / jobs/get	Veröffentlichen	Geräte veröffentlichen eine Nachricht in diesem Topic, um eine `GetPendingJobExecutions` -Anforderung auszugeben. Weitere Informationen

Thema	Zulässige Client-Operationen	Description
		finden Sie unter Führt MQTT-API-Operationen auf dem Gerät aus.
$aws/things/*thingName* // akzeptiert jobs/get	Abonnieren, empfangen	Geräte abonnieren dieses Thema, um Antworten von einer `GetPendingJobExecutions` -Anforderung zu empfangen. Weitere Informationen finden Sie unter Führt MQTT-API-Operationen auf dem Gerät aus.
$aws/things/*thingName* // abgelehnt jobs/get	Abonnieren, Empfangen	Geräte abonnieren dieses Thema, um eine Antwort zu erhalten, wenn eine `GetPendingJobExecutions` Anforderung abgelehnt wird. Weitere Informationen finden Sie unter Führt MQTT-API-Operationen auf dem Gerät aus.
$aws/things/*thingName* /-weiter jobs/start	Veröffentlichen	Geräte veröffentlichen eine Nachricht in diesem Topic, um eine `StartNextPendingJobExecution` -Anforderung auszugeben. Weitere Informationen finden Sie unter Führt MQTT-API-Operationen auf dem Gerät aus.
$aws/things/*thingName* / jobs/start-next/accepted	Abonnieren, Empfangen	Geräte abonnieren dieses Topic, um Antworten an eine `StartNextPendingJobExecution` -Anforderung zu empfangen. Weitere Informationen finden Sie unter Führt MQTT-API-Operationen auf dem Gerät aus.

Thema	Zulässige Client-Operationen	Description
$aws/things/*thingName* /jobs/start-next/rejected	Abonnieren, Empfangen	Geräte abonnieren dieses Thema, um eine Antwort zu erhalten, wenn eine `StartNextPendingJobExecution` Anforderung abgelehnt wird. Weitere Informationen finden Sie unter Führt MQTT-API-Operationen auf dem Gerät aus.
$aws/things/*thingName* /jobs/ /get *jobId*	Veröffentlichen	Geräte veröffentlichen eine Nachricht in diesem Topic, um eine `DescribeJobExecution` -Anforderung auszugeben. Weitere Informationen finden Sie unter Führt MQTT-API-Operationen auf dem Gerät aus.
$//jobs//aws/things*thingName* *jobId*get/accepted	Abonnieren, Empfangen	Geräte abonnieren dieses Topic, um Antworten an eine `DescribeJobExecution` -Anforderung zu empfangen. Weitere Informationen finden Sie unter Führt MQTT-API-Operationen auf dem Gerät aus.
$//jobs//aws/things*thingName* *jobId*get/rejected	Abonnieren, Empfangen	Geräte abonnieren dieses Thema, um eine Antwort zu erhalten, wenn eine `DescribeJobExecution` Anforderung abgelehnt wird. Weitere Informationen finden Sie unter Führt MQTT-API-Operationen auf dem Gerät aus.

Thema	Zulässige Client-Operationen	Description
$//jobs/ /update aws/things *thingName* *jobId*	Veröffentlichen	Geräte veröffentlichen eine Nachricht in diesem Thema, um eine `UpdateJobExecution` -Anforderung auszugeben. Weitere Informationen finden Sie unter Führt MQTT-API-Operationen auf dem Gerät aus.
$//jobs//aws/things*thingName* *jobId*update/accepted	Abonnieren, Empfangen	Geräte abonnieren dieses Thema, um Erfolgsantworten auf eine `UpdateJobExecution` -Anforderung zu empfangen. Weitere Informationen finden Sie unter Führt MQTT-API-Operationen auf dem Gerät aus. **Hinweis** Nur das Gerät, das auf $aws/things/*thingName* /jobs/ *jobId* /update veröffentlicht, empfängt Nachrichten zu diesem Thema.

Thema	Zulässige Client-Operationen	Description
$//jobs//aws/things*thingName* *jobId*update/rejected	Abonnieren, Empfangen	Geräte abonnieren dieses Thema, um eine Antwort zu erhalten, wenn eine `UpdateJobExecution` Anforderung abgelehnt wird. Weitere Informationen finden Sie unter Führt MQTT-API-Operationen auf dem Gerät aus. **Hinweis** Nur das Gerät, das auf $aws/things/*thingName* /jobs/ *jobId* /update veröffentlicht, empfängt Nachrichten zu diesem Thema.
$aws/things/*thingName* /jobs/notify	Abonnieren, Empfangen	Geräte abonnieren dieses Thema, um Benachrichtigungen zu empfangen, wenn eine Auftragsausführung der Liste der ausstehenden Ausführungen für ein Objekt hinzugefügt oder aus dieser entfernt wird. Weitere Informationen finden Sie unter Führt MQTT-API-Operationen auf dem Gerät aus.
$//-next aws/things *thingName* jobs/notify	Abonnieren, Empfangen	Geräte abonnieren dieses Thema, um Benachrichtigungen zu empfangen, wenn die nächste ausstehende Auftragsausführung für das Objekt geändert wird. Weitere Informationen finden Sie unter Führt MQTT-API-Operationen auf dem Gerät aus.

Thema	Zulässige Client-Operationen	Description
$ aws/events /job/ / abgeschlossen *jobId*	Abonnieren	Der Jobs-Service veröffentlicht ein Ereignis in diesem Thema, wenn ein Auftrag abgeschlossen wird. Weitere Informationen finden Sie unter Auftragsereignisse.
$ /job/ /storniert aws/events *jobId*	Abonnieren	Der Jobs-Service veröffentlicht ein Ereignis in diesem Thema, wenn ein Auftrag abgebrochen wird. Weitere Informationen finden Sie unter Auftragsereignisse.
$ /job/ /gelöscht aws/events *jobId*	Abonnieren	Der Jobs-Service veröffentlicht ein Ereignis in diesem Thema, wenn ein Auftrag gelöscht wird. Weitere Informationen finden Sie unter Auftragsereignisse.
$ /job/ /stornierung_in_in Bearbeitung aws/events *jobId*	Abonnieren	Der Jobs-Service veröffentlicht ein Ereignis in diesem Thema, wenn eine Auftragsstornierung beginnt. Weitere Informationen finden Sie unter Auftragsereignisse.
$ aws/events /job/ *jobId* / deletion_in_progress	Abonnieren	Der Jobs-Service veröffentlicht ein Ereignis in diesem Thema, wenn eine Auftragslöschung beginnt. Weitere Informationen finden Sie unter Auftragsereignisse.
$ aws/events *jobId* / JobExecution/ /erfolgreich	Abonnieren	Der Jobs-Service veröffentlicht ein Ereignis in diesem Thema, wenn der Auftrag erfolgreich ausgeführt wurde. Weitere Informationen finden Sie unter Auftragsereignisse.

Thema	Zulässige Client-Operationen	Description
$ aws/events *jobId* / JobExecution/ /fehlgeschlagen	Abonnieren	Der Jobs-Service veröffentlicht ein Ereignis in diesem Thema, wenn eine Auftragsausführung fehlschlägt. Weitere Informationen finden Sie unter Auftragsereignisse.
$ aws/events *jobId* / JobExecution/ /rejected	Abonnieren	Der Jobs-Service veröffentlicht ein Ereignis in diesem Thema, wenn eine Auftragsausführung abgewiesen wird. Weitere Informationen finden Sie unter Auftragsereignisse.
$ aws/events *jobId* / JobExecution/ /storniert	Abonnieren	Der Jobs-Service veröffentlicht ein Ereignis in diesem Thema, wenn eine Auftragsausführung storniert wird. Weitere Informationen finden Sie unter Auftragsereignisse.
$ aws/events *jobId* / JobExecution/ /timed_out	Abonnieren	Der Jobs-Service veröffentlicht ein Ereignis in diesem Thema, wenn eine Zeitüberschreitung der Auftragsausführung auftritt. Weitere Informationen finden Sie unter Auftragsereignisse.
$ aws/events *jobId* / JobExecution/ /entfernt	Abonnieren	Der Jobs-Service veröffentlicht ein Ereignis in diesem Thema, wenn eine Auftragsausführung entfernt wird. Weitere Informationen finden Sie unter Auftragsereignisse.
$ aws/events *jobId* / JobExecution/ /gelöscht	Abonnieren	Der Jobs-Service veröffentlicht ein Ereignis in diesem Thema, wenn eine Auftragsausführung gelöscht wird. Weitere Informationen finden Sie unter Auftragsereignisse.

Befehle, Themen

> **Note**
>
> Die Client-Operationen, die in dieser Tabelle als Receive aufgeführt sind, geben Themen an, die direkt an den Client AWS IoT veröffentlicht werden, der sie angefordert hat, unabhängig davon, ob der Client das Thema abonniert hat oder nicht. Kunden sollten damit rechnen, diese Antwortnachrichten zu erhalten, auch wenn sie sie nicht abonniert haben.
> Diese Antwortnachrichten werden nicht über den Message Broker weitergeleitet und können auch nicht von anderen Clients oder Regeln abonniert werden.

Thema	Zulässige Client-Operationen	Description
$aws/commands//*<devices>* / executions/ *<DeviceID>* *<ExecutionId>* /request/ *<PayloadFormat>* $///ausführungen/ / request aws/commands *<devices>* *<DeviceID>* *<ExecutionId>*	Abonnieren, Empfangen	Geräte erhalten eine Meldung zu diesem Thema, wenn eine Anforderung gestellt wird, eine Befehlsausführung von der Konsole aus oder über die API zu starten. `StartCommandExecution` In diesem Fall *<devices>* kann es sich entweder um IoT-Dinge oder MQTT-Clients handeln, und es *<DeviceID>* kann sich entweder um den IoT-Dingnamen oder die MQTT-Client-ID handeln.
$aws/commands///executions/ *<devices>* /response/ *<DeviceID>* *<ExecutionId>* *<PayloadFormat>*	Veröffentlichen	Geräte verwenden die `UpdateCommandExecution` MQTT-API, um eine Nachricht zu diesem Thema über die Befehlsausführung zu veröffentlichen. Die Nachricht wird als Antwort auf die Anfrage veröffentlicht, eine Befehlsausführung von der Konsole aus oder mithilfe der `StartCommandExecution` API zu starten. Die veröffentlichte Nachricht verwendet

Thema	Zulässige Client-Operationen	Description
		JSON oder CBOR als. *<PayloadFormat>*
$aws/commands//*<devices>* *<DeviceID>* *<ExecutionId>* / executions///response/accepted*<PayloadFormat>* $aws/commands///ausführungen//*<devices>* *<DeviceID>* *<ExecutionId>* response/accepted	Abonnieren, Empfangen	Wenn der Cloud-Dienst das Ergebnis der Befehlsausführung erfolgreich verarbeitet hat, AWS IoT Device Management veröffentlicht er eine Antwort auf das Thema /accepted.
$aws/commands//*<devices>* *<DeviceID>* *<ExecutionId>* / executions///response/rejected*<PayloadFormat>* $aws/commands///ausführungen//*<devices>* *<DeviceID>* *<ExecutionId>* response/rejected	Veröffentlichen	Wenn der Cloud-Dienst das Ergebnis der Befehlsausführung nicht verarbeiten konnte, AWS IoT Device Management veröffentlicht er eine Antwort auf das Thema /rejected.

Regelthemen

Thema	Zulässige Client-Operationen	Description
$aws/rules/*ruleName*	Veröffentlichen	Ein Gerät oder eine Anwendung veröffentlicht in diesem Thema, um Regeln direkt auszulösen. Weitere Informationen finden Sie unter Senken der Messaging-Kosten mit Basic Ingest.

Themen zu Secure Tunneling

Thema	Zulässige Client-Operationen	Description
$aws/things/*thing-name* /tunnels/notify	Abonnieren	AWS IoT veröffentlicht diese Nachricht, damit ein IoT-Agent einen lokalen Proxy auf dem Remote-Gerät starten kann. Weitere Informationen finden Sie unter the section called “IoT-Agent-Snippet”.

Schatten-Themen

Die Themen in diesem Abschnitt werden von benannten und unbenannten Schatten verwendet. Die jeweils verwendeten Themen unterscheiden sich nur durch das Themenpräfix. In dieser Tabelle wird das Themenpräfix angezeigt, das von jedem Schattentyp verwendet wird.

ShadowTopicPrefix Wert	Schattentyp
$aws/things/*thingName* /shadow	Unbenannter (klassischer) Schatten
$aws/things/*thingName* /shadow/name/*shadowName*	Benannter Schatten

Um ein vollständiges Thema zu erstellen, wählen Sie *ShadowTopicPrefix* für den Shadow-Typ, auf den Sie sich beziehen möchten, den Shadow aus, ersetzen Sie ihn *thingName* und, falls zutreffend*shadowName*, durch die entsprechenden Werte, und hängen Sie ihn dann mit dem Topic-Stub an, wie in der folgenden Tabelle gezeigt. Denken Sie daran, dass bei Themen zwischen Groß- und Kleinschreibung unterschieden wird.

Thema	Zulässige Client-Operationen	Description
ShadowTopicPrefix / löschen	Publish/Subscribe	Ein Gerät oder eine Anwendung veröffentlicht in diesem Thema, um ein Schattengerät zu löschen. Weitere Informationen finden Sie unter /delete.
ShadowTopicPrefix / delete/accepted	Abonnieren	Der Device Shadow-Service sendet Nachrichten an dieses Thema, wenn ein Schattengerät gelöscht wird. Weitere Informationen finden Sie unter/delete/accepted.
ShadowTopicPrefix / delete/rejected	Abonnieren	Der Device Shadow-Service sendet Nachrichten an dieses Thema, wenn ein Anfrage zum Löschen eines Schattengeräts abgelehnt wird. Weitere Informationen finden Sie unter/delete/rejected.
ShadowTopicPrefix /get	Publish/Subscribe	Eine Anwendung oder ein Gerät veröffentlicht eine leere Nachricht für dieses Thema, um ein Schattengerät zugewiesen zu bekommen. Weitere Informationen finden Sie unter MQTT-Themen für Geräteschatten.
ShadowTopicPrefix / get/accepted	Abonnieren	Der Device Shadow-Service sendet Nachrichten an dieses Thema, wenn ein Anfrage für ein Schattengerät erfolgrei

Thema	Zulässige Client-Operationen	Description
		ch war. Weitere Informationen finden Sie unter/get/accepted.
ShadowTopicPrefix / get/rejected	Abonnieren	Der Device Shadow-Service sendet Nachrichten an dieses Thema, wenn ein Anfrage für ein Schattengerät abgelehnt wird. Weitere Informationen finden Sie unter/get/rejected.
ShadowTopicPrefix / aktualisieren	Publish/Subscribe	Ein Gerät oder eine Anwendung veröffentlicht eine Nachricht in diesem Thema, um ein Schattengerät zu aktualisieren. Weitere Informationen finden Sie unter /update.
ShadowTopicPrefix / update/accepted	Abonnieren	Der Device Shadow-Service sendet Nachrichten an dieses Thema, wenn ein Schattengerät erfolgreich aktualisiert wurde. Weitere Informationen finden Sie unter/update/accepted.
ShadowTopicPrefix / update/rejected	Abonnieren	Der Device Shadow-Service sendet Nachrichten an dieses Thema, wenn die Aktualisierung eines Schattengeräts abgelehnt wurde. Weitere Informationen finden Sie unter/update/rejected.
ShadowTopicPrefix / update/delta	Abonnieren	Der Device Shadow-Service sendet Nachrichten an dieses Thema, wenn zwischen den gemeldeten und gewünschten Abschnitten eines Schattengeräts Abweichungen auftreten . Weitere Informationen finden Sie unter/update/delta.

Thema	Zulässige Client-Operationen	Description
ShadowTopicPrefix / update/documents	Abonnieren	AWS IoT veröffentlicht ein Statusdokument zu diesem Thema, wenn eine Aktualisierung des Shadows erfolgreich durchgeführt wurde. Weitere Informationen finden Sie unter/update/documents.

MQTT-based Themen zur Dateibereitstellung

> **Note**
>
> Die Client-Operationen, die in dieser Tabelle als Empfangen vermerkt sind, geben Themen an, die direkt an den Client AWS IoT veröffentlicht werden, der sie angefordert hat, unabhängig davon, ob der Client das Thema abonniert hat oder nicht. Kunden sollten damit rechnen, diese Antwortnachrichten zu erhalten, auch wenn sie sie nicht abonniert haben. Diese Antwortnachrichten werden nicht über den Message Broker weitergeleitet und können auch nicht von anderen Clients oder Regeln abonniert werden.

Diese Nachrichten unterstützen Antwortpuffer im Format Concise Binary Object Representation (CBOR) und in JavaScript Object Notation (JSON), je nach Thema. *payload-format*

payload-format	Datentyp des Antwortformats
cbor	Concise Binary Object Representation (CBOR)
json	JavaScript Objektnotation (JSON)

Thema	Zulässige Client-Operationen	Description
$aws/things/*ThingName* /streams/ *StreamId* /data/ *payload-format*	Abonnieren, Empfangen	AWS MQTT-based Die Dateibereitstellung veröffentlicht zu diesem Thema, wenn die "GetStream" -Anfrage von einem Gerät akzeptiert wird. Die Nutzlast enthält die Stream-Daten. Weitere Informationen finden Sie unter Verwenden AWS IoT MQTT-based Dateizustellung in Geräten.
$aws/things/*ThingName* /streams/ *StreamId* /get/ *payload-format*	Veröffentlichen	Ein Gerät veröffentlicht zu diesem Thema, um eine "" -Anfrage auszuführen. GetStream Weitere Informationen finden Sie unter Verwenden AWS IoT MQTT-based Dateizustellung in Geräten.
$aws/things/*ThingName* /streams/ *StreamId* /description/ *payload-format*	Abonnieren, Empfangen	AWS MQTT-based Die Dateibereitstellung veröffentlicht zu diesem Thema, wenn die "DescribeStream" -Anfrage von einem Gerät akzeptiert wird. Die Nutzlast enthält die Stream-Beschreibung. Weitere Informationen finden Sie unter Verwenden AWS IoT MQTT-based Dateizustellung in Geräten.
$aws/things/*ThingName* /streams/ *StreamId* /describe/ *payload-format*	Veröffentlichen	Ein Gerät veröffentlicht zu diesem Thema, um eine "" -Anfrage auszuführen. DescribeStream Weitere Informationen finden Sie unter Verwenden AWS IoT MQTT-based Dateizustellung in Geräten.

Thema	Zulässige Client-Operationen	Description
$aws/things/*ThingName*/streams/*StreamId*/rejected/*payload-format*	Abonnieren, Empfangen	AWS MQTT-based Die Dateibereitstellung veröffentlicht zu diesem Thema, wenn eine "" - oder "Describe Stream" -Anfrage von einem Gerät GetStream abgelehnt wird. Weitere Informationen finden Sie unter Verwenden AWS IoT MQTT-based Dateizustellung in Geräten.

Reservierte Themen-ARN

Alle ARNs (Amazon Resource Names) für reservierte Themen haben das folgende Format:

```
arn:aws:iot:aws-region:AWS-account-ID:topic/Topic
```

`arn:aws:iot:us-west-2:123EXAMPLE456:topic/$aws/things/thingName/jobs/get/accepted` ist beispielsweise ein ARN für das reservierte Thema `$aws/things/thingName/jobs/get/accepted`.

Domain-Konfigurationen

In können Sie Domänenkonfigurationen verwenden AWS IoT Core, um das Verhalten Ihrer Datenendpunkte zu konfigurieren und zu verwalten. Mit Domänenkonfigurationen können Sie mehrere AWS IoT Core Datenendpunkte generieren, sie mit Ihren eigenen vollqualifizierten Domainnamen (FQDN) und zugehörigen Serverzertifikaten anpassen und auch einen benutzerdefinierten Authorizer zuordnen. Weitere Informationen finden Sie unter Benutzerspezifische Authentifizierung und Autorisierung.

In diesem Kapitel:

- Was ist eine Domain-Konfiguration?
- Erstellen und Konfigurieren AWS verwaltete Domänen
- Vom Kunden verwaltete Domains erstellen und konfigurieren
- Verwalten von Domänenkonfigurationen

- Konfiguration von TLS-Einstellungen in Domänenkonfigurationen
- Konfiguration des Serverzertifikats für OCSP-Stapling

Was ist eine Domain-Konfiguration?

In bezieht AWS IoT Core sich eine Domänenkonfiguration auf die Einrichtung und Konfiguration einer Domain (entweder AWS verwaltete Domain oder vom Kunden verwaltete Domain) für Ihre AWS IoT Core Datenendpunkte. AWS IoT Core bietet auch einen Standardendpunkt für Ihr AWS Konto (`iot:Data-ATS`), mit AWS IoT Core dem Geräte kommunizieren können.

In diesem Thema:

- Anwendungsfälle
- Die wichtigsten Konzepte
- Wichtige Hinweise

Anwendungsfälle

Sie können Domänenkonfigurationen verwenden, um Aufgaben wie die folgenden zu vereinfachen.

- Migrieren Sie Geräte zu AWS IoT Core.
- Unterstützung heterogener Geräteflotten durch Beibehaltung separater Domänenkonfigurationen für separate Gerätetypen.
- Behalten Sie die Markenidentität bei (z. B. durch den Domainnamen) bei, während Sie die Anwendungsinfrastruktur zu AWS IoT Core migrieren.

Die wichtigsten Konzepte

Die folgenden Konzepte enthalten Einzelheiten zu Domänenkonfigurationen und verwandten Konzepten.

- Domain-Konfiguration

  Die Einrichtung und Konfiguration einer Domain für Ihre AWS IoT Core Endpunkte.

- Standard-Endpunktdomäne

Die Domäne, die den Standardendpunkt AWS IoT bereitstellt, z. `iot:Data-ATS` B. Um den Standardendpunkt zu finden, führen Sie den CLI-Befehl describe-endpoint oder https://docs.aws.amazon.com/cli/latest/reference/iot/describe-domain-configuration.html describe-domain-configuration aus. Gehen Sie alternativ zur AWS IoT Core Konsole und wählen Sie im linken Navigationsbereich unter Connect die Option Domänenkonfigurationen aus. Der Standard-Endpunkt wird mit dem Namen aufgeführt`iot:Data-ATS`.

- AWS verwaltete Domäne

  Die Domain, die verwaltet AWS wird. Wenn Sie die AWS verwaltete Domain wählen, stellen Ihre Geräte eine Verbindung über einen Datenendpunkt her, der von bereitgestellt wird AWS. AWS verwaltet die Domain und die Zertifikate.

- Vom Kunden verwaltete Domain

  Die Domain, die Sie verwalten werden. Auch als benutzerdefinierte Domain bekannt. Wenn Sie eine vom Kunden verwaltete Domain wählen, stellen Ihre Geräte eine Verbindung über einen benutzerdefinierten Domain-Datenendpunkt her. Sie verwalten die Domain und die Zertifikate. Mit der vom Kunden verwalteten Domain können Sie die Endpunkt-URLs an Ihre Bedürfnisse anpassen. Sie können beispielsweise einen benutzerdefinierten Domainnamen (`your-domain-name.com`) verwenden oder bestimmte Zugriffsrichtlinien anwenden.

- Authentifizierungstyp

  Der Authentifizierungstyp, mit dem Sie Ihre Geräte authentifizieren, wenn Sie eine Verbindung herstellen AWS IoT Core. Wenn Sie eine Domänenkonfiguration erstellen, müssen Sie einen Authentifizierungstyp angeben. Weitere Informationen finden Sie unter ???.

- Protokoll der Anwendung

  Die Protokolle der Anwendungsebene, mit denen Ihre Geräte eine Verbindung herstellen AWS IoT Core. Wenn Sie eine Domänenkonfiguration erstellen, müssen Sie ein Anwendungsprotokoll angeben. Weitere Informationen finden Sie unter ???.

Wichtige Hinweise

AWS IoT Core verwendet die TLS-Erweiterung „Server Name Indication“ (SNI), um Domänenkonfigurationen anzuwenden. Beim Verbinden von Geräten mit können Clients die SNI-Erweiterung (Server Name Indication) senden, die für Funktionen wie die Registrierung mehrerer Konten, konfigurierbare Endpunkte, benutzerdefinierte Domänen und VPC-Endpunkte erforderlich ist. AWS IoT Corehttps://docs.aws.amazon.com/iot/latest/developerguide/IoTCore-VPC.html Sie müssen des Weiteren einen Servernamen übergeben, der mit dem Domänennamen identisch ist, den Sie in der Domänenkonfiguration angeben. Um diesen Dienst zu testen, verwenden Sie die Version 2 der Geräte-SDKs in. AWS IoT GitHub

Wenn Sie mehrere Datenendpunkte in Ihrem Gerät erstellen AWS-Konto, teilen sich diese AWS IoT Core Ressourcen wie MQTT-Themen, Geräteschatten und Regeln.

Wenn Sie die Serverzertifikate für die AWS IoT Core benutzerdefinierte Domänenkonfiguration angeben, haben die Zertifikate maximal vier Domainnamen. Weitere Informationen finden Sie unter AWS IoT Core -Endpunkte und -Kontingente.

Erstellen und Konfigurieren AWS verwaltete Domänen

Mithilfe der CreateDomainConfiguration API erstellen Sie einen konfigurierbaren Endpunkt auf einer AWS verwalteten Domain. Eine Domänenkonfiguration für eine AWS verwaltete Domain besteht aus den folgenden Komponenten:

- `domainConfigurationName`

  Ein benutzerdefinierter Name, der die Domänenkonfiguration identifiziert, und der Wert muss für Sie AWS-Region eindeutig sein. Sie können keine Domänenkonfigurationsnamen verwenden, die mit `IoT:` beginnen, da diese Standardendpunkten vorbehalten sind.

- `defaultAuthorizerName` (optional)

  Der Name des benutzerdefinierten Autorisierers, der am Endpunkt verwendet werden soll.

- `allowAuthorizerOverride` (optional)

  Ein boolescher Wert, der angibt, ob Geräte den Standard-Autorisierer überschreiben können, indem sie im HTTP-Header der Anfrage einen anderen Autorisierer angeben. Dieser Wert ist erforderlich, wenn ein Wert für `defaultAuthorizerName` angegeben wird.

- `serviceType` (optional)

Der Diensttyp, den der Endpunkt bereitstellt. AWS IoT Core unterstützt nur den `DATA` Servicetyp. Wenn Sie `DATA` angeben, gibt AWS IoT Core einen Endpunkt mit dem Endpunkttyp `iot:Data-ATS` zurück. Sie können keinen konfigurierbaren `iot:Data` (VeriSign) -Endpunkt erstellen.

- `TlsConfig` (optional)

  Ein Objekt, das die TLS-Konfiguration für eine Domäne spezifiziert. Weitere Informationen finden Sie unter ???.

Der folgende AWS CLI Beispielbefehl erstellt eine Domänenkonfiguration für einen `Data` Endpunkt.

```
aws iot create-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" --service-type "DATA"
```

Die Ausgabe des Befehls kann wie folgt aussehen.

```
{
    "domainConfigurationName": "myDomainConfigurationName",
    "domainConfigurationArn": "arn:aws:iot:us-east-1:123456789012:domainconfiguration/
myDomainConfigurationName/itihw"
}
```

Vom Kunden verwaltete Domains erstellen und konfigurieren

Mit Domänenkonfigurationen können Sie einen benutzerdefinierten vollqualifizierten Domänennamen (Fully Qualified Domain Name, FQDN) angeben, mit dem eine Verbindung zu AWS IoT Core hergestellt werden soll. Die Verwendung von kundenverwalteten Domains (auch bekannt als benutzerdefinierte Domains) hat viele Vorteile: Sie können Kunden Ihre eigene Domain oder die Domain Ihres Unternehmens zu Branding-Zwecken zur Verfügung stellen; Sie können Ihre eigene Domain einfach so ändern, dass sie auf einen neuen Broker verweist; Sie können Mehrmandantenfähigkeit unterstützen, um Kunden mit unterschiedlichen Domains innerhalb derselben Domain zu bedienen AWS-Konto; und Sie können die Details Ihrer eigenen Serverzertifikate verwalten, z. B. die Root-Zertifizierungsstelle (CA), mit der das Zertifikat signiert wurde, den Signaturalgorithmus, Tiefe der Zertifikatskette und der Lebenszyklus des Zertifikat.

Der Workflow zum Einrichten einer Domänenkonfiguration mit einer benutzerdefinierten Domäne besteht aus den folgenden drei Phasen.

1. Registrierung von Serverzertifikaten in AWS Certificate Manager

2. Erstellen einer Domänenkonfiguration
3. Erstellen von DNS-Datensätzen

Registrierung von Serverzertifikaten in AWS Certificate Manager

Bevor Sie eine Domänenkonfiguration mit einer benutzerdefinierten Domäne erstellen, müssen Sie die Serverzertifikatskette in AWS Certificate Manager (ACM) registrieren. Sie können die folgenden drei Serverzertifikattypen verwenden.

- In ACM generierte öffentliche Zertifikate
- Von einer öffentlichen Zertifizierungsstelle signierte externe Zertifikate
- Von einer privaten Zertifizierungsstelle signierte externe Zertifikate

> **Note**
>
> AWS IoT Core betrachtet ein Zertifikat als von einer öffentlichen Zertifizierungsstelle signiert, wenn es in Mozillas vertrauenswürdigem CA-Bundle enthalten ist.

Zertifikatanforderungen

Die Anforderungen für den Import von Zertifikaten in ACM finden Sie unter Anforderungen an den Import von Zertifikaten. Zusätzlich zu diesen Anforderungen fügt AWS IoT Core folgende Anforderungen hinzu.

- Das Leaf-Zertifikat muss die Erweiterung Extended Key Usage x509 v3 mit dem Wert ServerAuth (TLS Web Server Authentication) enthalten. Wenn Sie das Zertifikat von ACM anfordern, wird diese Erweiterung automatisch hinzugefügt.
- Die maximale Tiefe der Zertifikatskette beträgt 5 Zertifikate.
- Die maximale Größe der Zertifikatskette beträgt 16 KB.
- Zu den unterstützten kryptografischen Algorithmen und Schlüsselgrößen gehören RSA 2048-Bit (RSA_2048) und ECDSA 256-Bit (EC_Prime256v1).

Verwenden eines Zertifikats für mehrere Domänen

Wenn Sie planen, ein Zertifikat für mehrere Unterdomänen zu verwenden, verwenden Sie eine Platzhalterdomäne im Feld „Common Name (CN)“ oder „Subject Alternative Names (SAN)“. Verwenden Sie zum Beispiel **`*.iot.example.com`** dev.iot.example.com, qa.iot.example.com und prod.iot.example.com. Jeder FQDN erfordert eine eigene Domänenkonfiguration, aber mehr als eine Domänenkonfiguration kann denselben Platzhalterwert verwenden. Der CN oder das SAN muss den FQDN abdecken, den Sie als benutzerdefinierte Domäne verwenden möchten. Wenn SANs vorhanden sind, wird die CN ignoriert und ein SAN muss den FQDN abdecken, den Sie als benutzerdefinierte Domäne verwenden möchten. Hierbei kann es sich um eine exakte Übereinstimmung oder um eine Platzhalterübereinstimmung handeln. Nachdem ein Wildcard-Zertifikat validiert und für ein Konto registriert wurde, werden andere Konten in der Region daran gehindert, benutzerdefinierte Domänen zu erstellen, die sich mit dem Zertifikat überschneiden.

In den folgenden Abschnitten wird beschrieben, wie Sie die einzelnen Zertifikatstypen erhalten. Für jede Zertifikatressource ist ein bei ACM registrierter Amazon-Ressourcenname (ARN) erforderlich, den Sie beim Erstellen Ihrer Domänenkonfiguration verwenden.

ACM-generated öffentliche Zertifikate

Mithilfe der RequestCertificate API können Sie ein öffentliches Zertifikat für Ihre benutzerdefinierte Domain generieren. Wenn Sie auf diese Weise ein Zertifikat generieren, überprüft ACM Ihr Eigentum an der benutzerdefinierten Domäne. Weitere Informationen finden Sie unter Anfordern eines öffentlichen Zertifikats im AWS Certificate Manager -Benutzerhandbuch.

Von einer öffentlichen Zertifizierungsstelle signierte externe Zertifikate

Wenn Sie bereits ein Serverzertifikat haben, das von einer öffentlichen CA signiert ist (eine CA, die in Mozillas vertrauenswürdigem CA-Bundle enthalten ist), können Sie die Zertifikatskette mithilfe der ImportCertificate API direkt in ACM importieren. Weitere Informationen zu dieser Aufgabe und den Voraussetzungen und Anforderungen an das Zertifikatformat finden Sie unter Importieren von Zertifikaten.

Von einer privaten Zertifizierungsstelle signierte externe Zertifikate

Wenn Sie bereits über ein Serverzertifikat verfügen, das von einer privaten Zertifizierungsstelle signiert oder selbstsigniert ist, können Sie das Zertifikat zum Erstellen Ihrer Domänenkonfiguration verwenden. Sie müssen jedoch auch ein zusätzliches öffentliches Zertifikat in ACM erstellen, um den Besitz Ihrer Domäne zu überprüfen. Registrieren Sie dazu Ihre Serverzertifikatskette mithilfe der API

in ACM. ImportCertificate Weitere Informationen zu dieser Aufgabe und den Voraussetzungen und Anforderungen an das Zertifikatformat finden Sie unter Importieren von Zertifikaten.

Erstellen eines Validierungszertifikats

Nachdem Sie Ihr Zertifikat in ACM importiert haben, generieren Sie mithilfe der API ein öffentliches Zertifikat für Ihre benutzerdefinierte Domain. RequestCertificate Wenn Sie auf diese Weise ein Zertifikat generieren, überprüft ACM Ihr Eigentum an der benutzerdefinierten Domäne. Weitere Informationen finden Sie unter Anfordern eines öffentlichen Zertifikats. Wenn Sie Ihre Domänenkonfiguration erstellen, verwenden Sie dieses öffentliche Zertifikat als Validierungszertifikat.

Erstellen einer Domänenkonfiguration

Mithilfe der CreateDomainConfiguration API erstellen Sie einen konfigurierbaren Endpunkt auf einer benutzerdefinierten Domain. Eine Domänenkonfiguration für eine benutzerdefinierte Domäne besteht aus folgenden Elementen:

- `domainConfigurationName`

  Ein benutzerdefinierter Name, der die Domänenkonfiguration identifiziert Domänenkonfigurationsnamen, die mit `IoT:` beginnen, sind für Standardendpunkte reserviert und können nicht verwendet werden. Außerdem muss dieser Wert für Sie eindeutig sein AWS-Region.
- `domainName`

  Der FQDN, mit dem Ihre Geräte eine Verbindung herstellen AWS IoT Core. AWS IoT Core nutzt die TLS-Erweiterung „Server Name Indication“ (SNI), um Domänenkonfigurationen anzuwenden. Geräte müssen diese Erweiterung verwenden, wenn sie eine Verbindung herstellen und einen Servernamen übergeben, der mit dem Domänennamen identisch ist, der in der Domänenkonfiguration angegeben ist.
- `serverCertificateArns`

  Der ARN der Serverzertifikatskette, die Sie bei ACM registriert haben. AWS IoT Core unterstützt derzeit nur ein Serverzertifikat.
- `validationCertificateArn`

  Der ARN des öffentlichen Zertifikats, das von ACM ausgestellt wurde, um das Eigentum an Ihrer benutzerdefinierten Domäne zu validieren. Dieses Argument ist nicht erforderlich, wenn Sie ein öffentlich signiertes Zertifikat oder ein ACM-generated Serverzertifikat verwenden.
- `defaultAuthorizerName (optional)`

Der Name des benutzerdefinierten Autorisierers, der am Endpunkt verwendet werden soll.

- `allowAuthorizerOverride`

  Ein boolescher Wert, der angibt, ob Geräte den Standard-Autorisierer überschreiben können, indem sie im HTTP-Header der Anfrage einen anderen Autorisierer angeben. Dieser Wert ist erforderlich, wenn ein Wert für `defaultAuthorizerName` angegeben wird.

- `serviceType`

  AWS IoT Core unterstützt derzeit nur den `DATA` Diensttyp. Wenn Sie angeben`DATA`, wird ein Endpunkt mit dem Endpunkttyp von AWS IoT zurückgegeben`iot:Data-ATS`.

- `TlsConfig` (optional)

  Ein Objekt, das die TLS-Konfiguration für eine Domäne spezifiziert. Weitere Informationen finden Sie unter ???.

- `serverCertificateConfig` (optional)

  Ein Objekt, das die Konfiguration des Serverzertifikats für eine Domäne angibt. Weitere Informationen finden Sie unter ???.

Der folgende AWS CLI Befehl erstellt eine Domänenkonfiguration für iot.example.com.

```
aws iot create-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" --service-type "DATA"
--domain-name "iot.example.com" --server-certificate-arns serverCertARN --validation-
certificate-arn validationCertArn
```

> **Note**
>
> Nachdem Sie Ihre Domainkonfiguration erstellt haben, kann es bis zu 60 Minuten dauern, bis Ihre benutzerdefinierten Serverzertifikate AWS IoT Core bereitgestellt werden.

Weitere Informationen finden Sie unter ???.

Erstellen von DNS-Datensätzen

Nachdem Sie die Serverzertifikatkette registriert und die Domänenkonfiguration erstellt haben, erstellen Sie einen DNS-Datensatz, damit Ihre benutzerdefinierte Domäne auf eine

AWS IoT -Domäne verweist. Dieser Datensatz muss auf einen AWS IoT Endpunkt des Typs verweisen`iot:Data-ATS`. Sie können Ihren Endpunkt mithilfe der DescribeEndpoint API abrufen.

Der folgende AWS CLI Befehl zeigt, wie Sie Ihren Endpunkt abrufen.

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

Nachdem Sie Ihren `iot:Data-ATS` Endpunkt erhalten haben, erstellen Sie einen `CNAME` Datensatz von Ihrer benutzerdefinierten Domain zu diesem AWS IoT Endpunkt. Wenn du mehrere benutzerdefinierte Domains in derselben Domain erstellst AWS-Konto, verwende einen Alias für denselben `iot:Data-ATS` Endpunkt.

Fehlerbehebung

Wenn Sie Probleme haben, Geräte mit einer benutzerdefinierten Domain zu verbinden, vergewissern Sie sich, dass Ihr Serverzertifikat akzeptiert und angewendet AWS IoT Core wurde. Sie können überprüfen, ob Ihr Zertifikat akzeptiert AWS IoT Core wurde, indem Sie entweder die AWS IoT Core Konsole oder die verwenden AWS CLI.

Um die AWS IoT Core Konsole zu verwenden, navigieren Sie zur Seite mit den Domänenkonfigurationen und wählen Sie den Namen der Domänenkonfiguration aus. Überprüfen Sie im Abschnitt Serverzertifikatsdetails den Status und die Statusdetails. Wenn das Zertifikat ungültig ist, ersetzen Sie es in ACM durch ein Zertifikat, das die im vorherigen Abschnitt aufgeführten Zertifikatsanforderungen erfüllt. Wenn das Zertifikat dieselbe ARN hat, AWS IoT Core wird es abgeholt und automatisch angewendet.

Um den Zertifikatsstatus mithilfe von zu überprüfen AWS CLI, rufen Sie die DescribeDomainConfiguration API auf und geben Sie Ihren Domain-Konfigurationsnamen an.

> **Note**
>
> Wenn Ihr Zertifikat ungültig ist, AWS IoT Core wird weiterhin das letzte gültige Zertifikat bereitgestellt.

Mit dem folgenden openssl-Befehl können Sie überprüfen, welches Zertifikat auf Ihrem Endpunkt zugestellt wird.

```
openssl s_client -connect custom-domain-name:8883 -showcerts -servername
custom-domain-name
```

Verwalten von Domänenkonfigurationen

In diesem Thema werden die wichtigsten Vorgänge zur Verwaltung Ihrer Domänenkonfigurationsressourcen behandelt. Sie können die Lebenszyklen vorhandener Konfigurationen auch mithilfe der folgenden APIs verwalten: ListDomainConfigurations, DescribeDomainConfiguration UpdateDomainConfiguration, und. DeleteDomainConfiguration

In diesem Thema:

- Anzeigen von Domänenkonfigurationen
- Aktualisieren von Domänenkonfigurationen
- Löschen von Domänenkonfigurationen
- Rotierende Zertifikate in benutzerdefinierten Domänen

Anzeigen von Domänenkonfigurationen

Verwenden Sie die API, um eine paginierte Liste aller Domänenkonfigurationen in Ihrem AWS-Konto zurückzugeben. ListDomainConfigurations Sie können die Details einer bestimmten Domainkonfiguration mithilfe der DescribeDomainConfiguration API einsehen. Diese API nimmt einen einzelnen `domainConfigurationName`-Parameter und gibt die Details der angegebenen Konfiguration zurück.

Beispiel

Aktualisieren von Domänenkonfigurationen

Verwenden Sie die UpdateDomainConfiguration API, um den Status oder den benutzerdefinierten Authorizer Ihrer Domain-Konfiguration zu aktualisieren. Sie können den Status auf `ENABLED` oder `DISABLED` setzen. Wenn Sie die Domänenkonfiguration deaktivieren, erhalten Geräte, die mit dieser Domäne verbunden sind, einen Authentifizierungsfehler. Derzeit können Sie das Serverzertifikat in Ihrer Domänenkonfiguration nicht aktualisieren. Um das Zertifikat einer Domänenkonfiguration zu ändern, müssen Sie es löschen und neu erstellen.

Beispiel

Löschen von Domänenkonfigurationen

Bevor Sie eine Domainkonfiguration löschen, verwenden Sie die UpdateDomainConfiguration API, um den Status auf `DISABLED` festzulegen. Auf diese Weise vermeiden Sie ein versehentliches

Löschen des Endpunkts. Nachdem Sie die Domainkonfiguration deaktiviert haben, löschen Sie sie mithilfe der DeleteDomainConfiguration API. Sie müssen AWS-verwaltete Domains 7 Tage lang in `DISABLED` den Status versetzen, bevor Sie sie löschen können. Sie können benutzerdefinierte Domains in `DISABLED` den Status versetzen und sie dann sofort löschen.

Beispiel

Nachdem Sie eine Domänenkonfiguration gelöscht haben, wird das Serverzertifikat, das dieser benutzerdefinierten Domain zugeordnet ist, nicht AWS IoT Core mehr bereitgestellt.

Rotierende Zertifikate in benutzerdefinierten Domänen

Möglicherweise müssen Sie Ihr Serverzertifikat regelmäßig durch ein aktualisiertes Zertifikat ersetzen. Die Häufigkeit, mit der Sie dies tun müssen, ist von der Gültigkeitsdauer Ihres Zertifikats abhängig. Wenn Sie Ihr Serverzertifikat mithilfe von AWS Certificate Manager (ACM) generiert haben, können Sie festlegen, dass das Zertifikat automatisch verlängert wird. Wenn ACM Ihr Zertifikat erneuert, holt es sich AWS IoT Core automatisch das neue Zertifikat. Sie müssen keine weiteren Aktionen ausführen. Wenn Sie Ihr Serverzertifikat aus einer anderen Quelle importiert haben, können Sie es rotieren, indem Sie es erneut in ACM importieren. Informationen zum erneuten Importieren von Zertifikaten finden Sie unter Zertifikat erneut importieren.

> **Note**
>
> AWS IoT Core nimmt Zertifikatsaktualisierungen nur unter den folgenden Bedingungen auf.
>
> - Das neue Zertifikat hat denselben ARN wie das alte.
> - Das neue Zertifikat hat denselben Signaturalgorithmus, denselben generischen Namen oder denselben alternativen Betreffnamen wie das alte.

Konfiguration von TLS-Einstellungen in Domänenkonfigurationen

AWS IoT Core stellt vordefinierte Sicherheitsrichtlinien bereit, mit denen Sie Ihre Transport Layer Security (TLS) -Einstellungen für TLS 1.2 und TLS 1.3 in Domänenkonfigurationen anpassen können. Eine Sicherheitsrichtlinie ist eine Kombination aus TLS-Protokollen und ihren Verschlüsselungen, die die unterstützten Protokolle und Verschlüsselungen bei TLS-Verhandlungen zwischen einem Client und einem Server festlegen. Mit den unterstützten Sicherheitsrichtlinien können Sie die TLS-Einstellungen Ihrer Geräte flexibler verwalten, beim Anschließen neuer

Geräte die aktuellsten Sicherheitsmaßnahmen anwenden und konsistente TLS-Konfigurationen für vorhandene Geräte beibehalten.

Die folgende Tabelle beschreibt die Sicherheitsrichtlinien, ihre TLS-Versionen sowie die unterstützten Regionen:

Name der Sicherheitsrichtlinie	Unterstützt AWS-Regionen
IoTSecurityPolicy_TLS13_1_3_2022_10	Alle, außer AWS-Regionen AWS GovCloud (US)
IoTSecurityPolicy_TLS13_1_2_2022_10	Alle, außer AWS-Regionen AWS GovCloud (US)
IoTSecurityPolicy_TLS12_1_2_2022_10	Alles AWS-Regionen
IoTSecurityPolicy_TLS12_1_0_2016_01	AP-Ost-1, AP-Nordost-2, AP-Süd-1, AP-Southeast-2, CA-Zentral-1, CN-Nord-1, CN-Nordwest-1, EU-Nord-1, EU-West-2, EU-West-3, Me-Süd-1, SA-Ost-1, US-Ost-2, US-West-1
IoTSecurityPolicy_TLS12_1_0_2015_01	ap-northeast-1, ap-southeast-1, eu-central-1, eu-west-1, us-east-1, us-west-2
IoTSecurityPolicy_TLS13_1_2_2022_01	AWS GovCloud (US) nur (FIPS-Richtlinie)

Die Namen der Sicherheitsrichtlinien AWS IoT Core enthalten Versionsinformationen, die auf dem Jahr und Monat basieren, in dem sie veröffentlicht wurden. Wenn Sie eine neue Domänenkonfiguration erstellen, wird die Sicherheitsrichtlinie standardmäßig `IoTSecurityPolicy_TLS13_1_2_2022_10` für alle Regionen außer verwendet AWS GovCloud (US). Für AWS GovCloud (US) Regionen wird die Sicherheitsrichtlinie standardmäßig auf `IoTSecurityPolicy_TLS13_1_2_2022_01` die FIPS-Richtlinie gesetzt. Für Nicht-FIPS-Endpunkte in AWS GovCloud (US) Regionen können Sie die verwenden. `IoTSecurityPolicy_TLS12_1_2_2022_10` Eine vollständige Tabelle der Sicherheitsrichtlinien mit Einzelheiten zu Protokollen, TCP-Ports und Chiffren finden Sie unter Sicherheitsrichtlinien. ???

AWS IoT Core unterstützt keine benutzerdefinierten Sicherheitsrichtlinien. Weitere Informationen finden Sie unter ???.

Um TLS-Einstellungen in Domänenkonfigurationen zu konfigurieren, können Sie die AWS IoT Konsole oder die verwenden AWS CLI.

Inhalt

- Konfigurieren von TLS-Einstellungen in den Domänenkonfigurationen (Konsole)
- Konfigurieren von TLS-Einstellungen in Domänenkonfigurationen (CLI)

Konfigurieren von TLS-Einstellungen in den Domänenkonfigurationen (Konsole)

Um TLS-Einstellungen mit dem zu konfigurieren AWS IoT Konsole

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die AWS IoT Konsole.
2. Gehen Sie beim Erstellen einer neuen Domänenkonfiguration folgendermaßen vor, um die TLS-Einstellungen zu konfigurieren.
    1. Wählen Sie im linken Navigationsbereich Domänenkonfigurationen und dann Domänenkonfiguration erstellen aus.
    2. Wählen Sie auf der Seite Domänenkonfiguration erstellen im Abschnitt Benutzerdefinierte Domäneneinstellungen – optional unter Sicherheitsrichtlinie auswählen eine Sicherheitsrichtlinie.
    3. Folgen Sie dem Widget, und führen Sie die restlichen Schritte durch. Wählen Sie Domänenkonfiguration erstellen.
3. Gehen Sie wie folgt vor, um die TLS-Einstellungen in einer vorhandenen Domänenkonfiguration zu aktualisieren.
    1. Wählen Sie im linken Navigationsbereich Domänenkonfigurationen und anschließend eine Domänenkonfiguration aus.
    2. Wählen Sie auf der Seite mit den Details zur Domänenkonfiguration die Option Bearbeiten. Wählen Sie dann im Abschnitt Benutzerdefinierte Domäneneinstellungen – optional unter Sicherheitsrichtlinie auswählen eine Sicherheitsrichtlinie.
    3. Wählen Sie Konfiguration aktualisieren.

Weitere Informationen finden Sie unter Eine Domänenkonfiguration erstellen und Domänenkonfigurationen verwalten.

Konfigurieren von TLS-Einstellungen in Domänenkonfigurationen (CLI)

Sie können die CLI-Befehle create-domain-configuration und update-domain-configuration verwenden, um Ihre TLS-Einstellungen in Domänenkonfigurationen zu konfigurieren.

1. So legen Sie die TLS-Einstellungen mit dem create-domain-configuration-CLI-Befehl fest:

```
aws iot create-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --tls-config securityPolicy=IoTSecurityPolicy_TLS13_1_2_2022_10
```

   Die Ausgabe dieses Befehls kann folgendermaßen aussehen:

```
{
"domainConfigurationName": "test",
"domainConfigurationArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
test/34ga9"
}
```

   Wenn Sie eine neue Domänenkonfiguration erstellen, ohne die Sicherheitsrichtlinie anzugeben, lautet der Standardwert: `IoTSecurityPolicy_TLS13_1_2_2022_10`.

2. So beschreiben Sie die TLS-Einstellungen mit dem describe-domain-configuration-CLI-Befehl:

```
aws iot describe-domain-configuration \
    --domain-configuration-name domainConfigurationName
```

   Dieser Befehl kann die Domänenkonfigurationsdetails, die die TLS-Einstellungen enthalten, folgendermaßen zurückgeben:

```
{
 "tlsConfig": {
 "securityPolicy": "IoTSecurityPolicy_TLS13_1_2_2022_10"
 },
 "domainConfigurationStatus": "ENABLED",
 "serviceType": "DATA",
 "domainType": "AWS_MANAGED",
 "domainName": "d1234567890abcdefghij-ats.iot.us-west-2.amazonaws.com",
```

```
 "serverCertificates": [],
 "lastStatusChangeDate": 1678750928.997,
 "domainConfigurationName": "test",
 "domainConfigurationArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
test/34ga9"
}
```

3. So aktualisieren Sie die TLS-Einstellungen mit dem update-domain-configuration-CLI-Befehl:

```
aws iot update-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --tls-config securityPolicy=IoTSecurityPolicy_TLS13_1_2_2022_10
```

   Die Ausgabe dieses Befehls kann folgendermaßen aussehen:

```
{
"domainConfigurationName": "test",
"domainConfigurationArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
test/34ga9"
}
```

4. Führen Sie den update-domain-configuration-CLI-Befehl aus, um die TLS-Einstellungen für Ihren ATS-Endpunkt zu aktualisieren. Der Domänenkonfigurationsname für Ihren ATS-Endpunkt lautet `iot:Data-ATS`.

```
aws iot update-domain-configuration \
    --domain-configuration-name "iot:Data-ATS" \
    --tls-config securityPolicy=IoTSecurityPolicy_TLS13_1_2_2022_10
```

   Die Ausgabe dieses Befehls kann folgendermaßen aussehen:

```
{
"domainConfigurationName": "iot:Data-ATS",
"domainConfigurationArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
iot:Data-ATS"
}
```

Weitere Informationen finden Sie unter CreateDomainConfiguration und UpdateDomainConfiguration in der AWS -API-Referenz.

Konfiguration des Serverzertifikats für OCSP-Stapling

AWS IoT Core unterstützt das Online Certificate Status Protocol (OCSP) -Stapling für Serverzertifikate, auch bekannt als Serverzertifikat-OCSP-Stapling oder OCSP-Stapling. Es handelt sich um einen Sicherheitsmechanismus, der verwendet wird, um den Sperrstatus des Serverzertifikats in einem TLS-Handshake (Transport Layer Security) zu überprüfen. Mithilfe von OCSP-Stapling AWS IoT Core können Sie die Gültigkeit des Serverzertifikats Ihrer benutzerdefinierten Domain um eine zusätzliche Überprüfungsebene erweitern.

Sie können das OCSP-Stapling in für Serverzertifikate aktivieren, um die Gültigkeit des Zertifikats AWS IoT Core zu überprüfen, indem Sie den OCSP-Responder regelmäßig abfragen. Die OCSP-Stapling-Einstellung ist Teil des Prozesses zum Erstellen oder Aktualisieren einer Domänenkonfiguration mit einer benutzerdefinierten Domäne. OCSP-Stapling überprüft kontinuierlich den Sperrstatus des Serverzertifikats. Auf diese Weise können Sie überprüfen, ob die Clients, die eine Verbindung zu Ihren benutzerdefinierten Domänen herstellen, allen Zertifikaten, die von der CA gesperrt wurden, nicht mehr vertrauen. Weitere Informationen finden Sie unter ???.

Das OCSP-Stapling von Serverzertifikaten ermöglicht die Überprüfung des Sperrstatus in Echtzeit, reduziert die mit der Überprüfung des Sperrstatus verbundene Latenz und verbessert den Datenschutz und die Zuverlässigkeit sicherer Verbindungen. Weitere Informationen zu den Vorteilen der Verwendung von OCSP-Stapling finden Sie unter. ???

> **Note**
>
> Diese Funktion ist in nicht verfügbar. AWS GovCloud (US) Regions

In diesem Thema:

- Was ist OCSP?
- So funktioniert OCSP-Stapling
- Das Serverzertifikat OCSP wird aktiviert in AWS IoT Core
- Konfiguration des Serverzertifikats OCSP für private Endpunkte in AWS IoT Core
- Wichtige Hinweise zur Verwendung von OCSP Stapling für Serverzertifikate in AWS IoT Core
- Problembehandlung beim Einheften von Serverzertifikaten durch OCSP AWS IoT Core

Was ist OCSP?

Das Online Certificate Status Protocol (OCSP) hilft dabei, den Sperrstatus eines Serverzertifikats für einen Transport Layer Security (TLS) -Handshake bereitzustellen.

Die wichtigsten Konzepte

Die folgenden Schlüsselkonzepte enthalten Einzelheiten zum Online Certificate Status Protocol (OCSP).

OCSP

OCSP wird verwendet, um den Zertifikatssperrstatus während des Transport Layer Security (TLS) -Handshakes zu überprüfen. OCSP ermöglicht die Validierung von Zertifikaten in Echtzeit. Dies bestätigt, dass das Zertifikat seit seiner Ausstellung nicht gesperrt wurde oder abgelaufen ist. OCSP ist im Vergleich zu herkömmlichen Zertifikatssperrlisten (CRLs) auch besser skalierbar. OCSP-Antworten sind kleiner und können effizient generiert werden, weshalb sie sich besser für umfangreiche Private-Key-Infrastrukturen (PKIs) eignen.

OCSP-Responder

Ein OCSP-Responder (auch als OCSP-Server bekannt) empfängt OCSP-Anfragen von Clients, die den Sperrstatus von Zertifikaten überprüfen möchten, und beantwortet diese.

Client-side OCSP

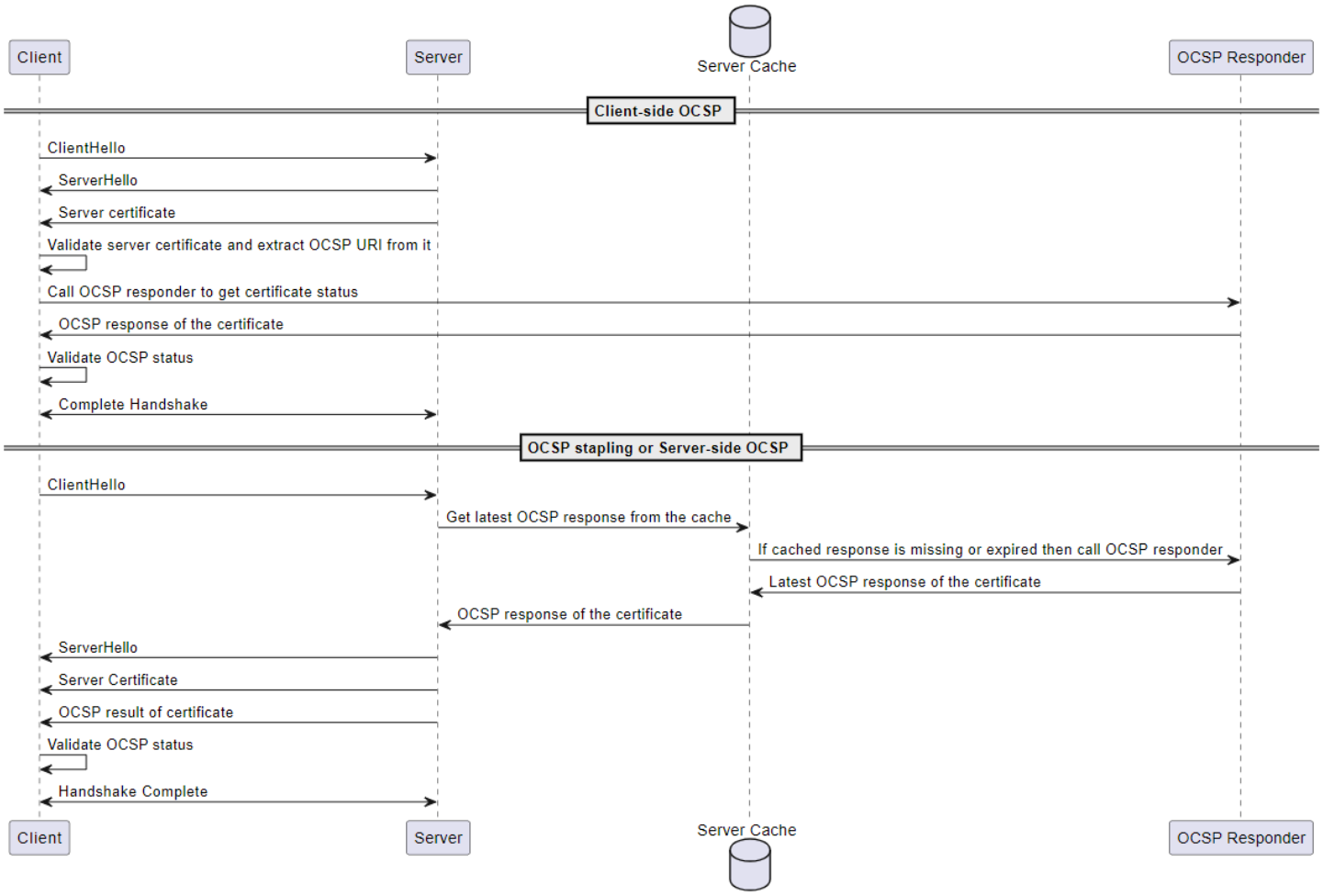
Bei clientseitigem OCSP verwendet der Client OCSP, um einen OCSP-Responder zu kontaktieren, um den Sperrstatus des Zertifikats während des TLS-Handshakes zu überprüfen.

Server-side OCSP

Bei serverseitigem OCSP (auch bekannt als OCSP-Stapling) ist der Server (und nicht der Client) in der Lage, die Anfrage an den OCSP-Responder zu stellen. Der Server heftet die OCSP-Antwort auf das Zertifikat und gibt sie während des TLS-Handshakes an den Client zurück.

OCSP-Diagramme

Das folgende Diagramm veranschaulicht, wie clientseitiges OCSP und serverseitiges OCSP funktionieren.



Client-side OCSP

1. Der Client sendet eine `ClientHello` Nachricht, um den TLS-Handshake mit dem Server zu initiieren.
2. Der Server empfängt die Nachricht und antwortet mit einer `ServerHello` Nachricht. Der Server sendet auch das Serverzertifikat an den Client.
3. Der Client validiert das Serverzertifikat und extrahiert daraus eine OCSP-URI.
4. Der Client sendet eine Anfrage zur Überprüfung des Zertifikats an den OCSP-Responder.
5. Der OCSP-Responder sendet eine OCSP-Antwort.
6. Der Client überprüft den Zertifikatsstatus anhand der OCSP-Antwort.
7. Der TLS-Handshake ist abgeschlossen.

Server-side OCSP

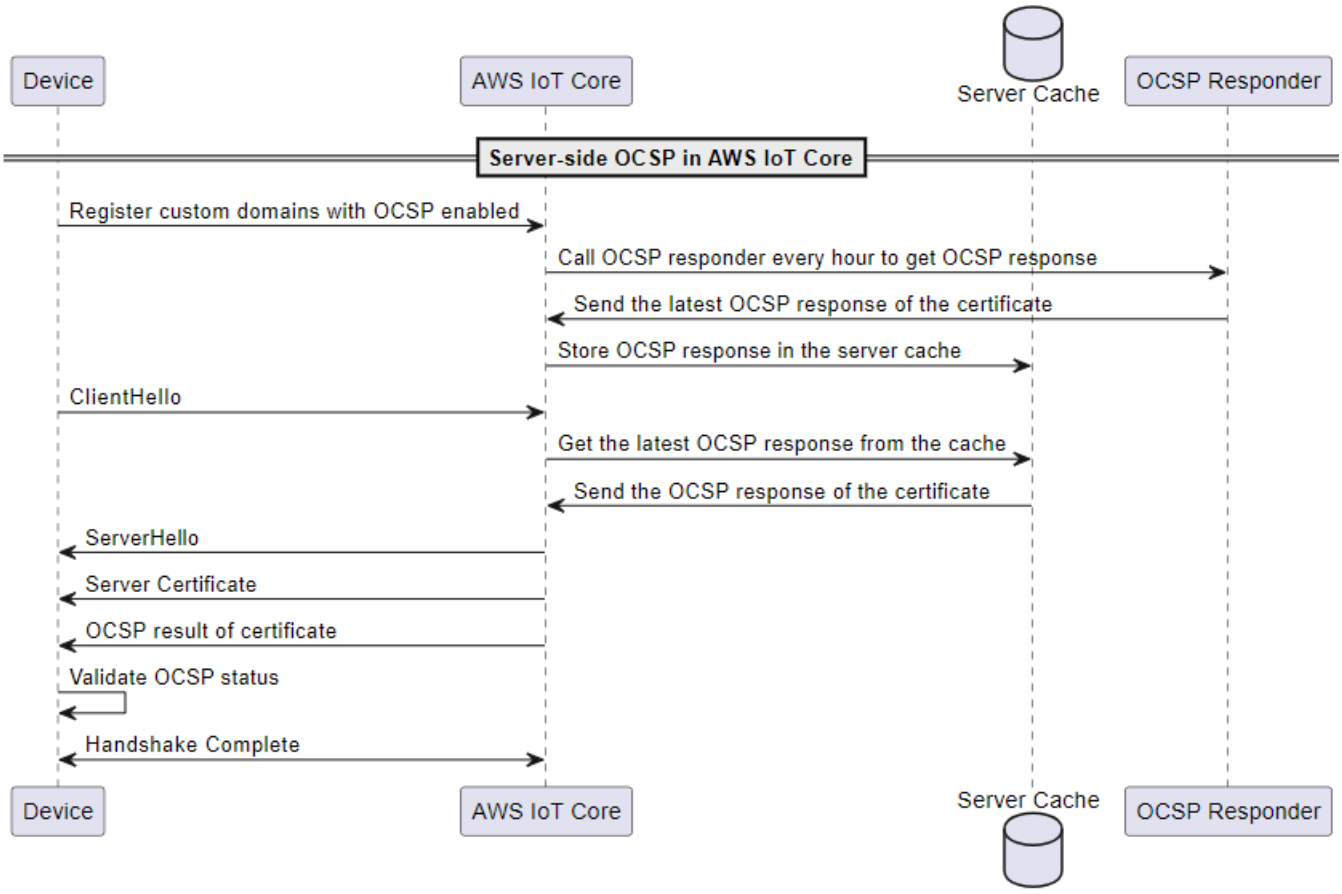
1. Der Client sendet eine `ClientHello` Nachricht, um den TLS-Handshake mit dem Server zu initiieren.
2. Der Server empfängt die Nachricht und erhält die neueste zwischengespeicherte OCSP-Antwort. Wenn die zwischengespeicherte Antwort fehlt oder abgelaufen ist, ruft der Server den OCSP-Responder auf, um den Zertifikatsstatus abzufragen.
3. Der OCSP-Responder sendet eine OCSP-Antwort an den Server.
4. Der Server sendet eine Nachricht. `ServerHello` Der Server sendet auch das Serverzertifikat und den Zertifikatsstatus an den Client.
5. Der Client validiert den OCSP-Zertifikatsstatus.
6. Der TLS-Handshake ist abgeschlossen.

So funktioniert OCSP-Stapling

OCSP-Stapling wird während des TLS-Handshakes zwischen dem Client und dem Server verwendet, um den Sperrstatus des Serverzertifikats zu überprüfen. Der Server stellt die OCSP-Anfrage an den OCSP-Responder und heftet die OCSP-Antworten auf die an den Client zurückgegebenen Zertifikate. Wenn der Server die Anfrage an den OCSP-Responder stellt, können die Antworten zwischengespeichert und dann für viele Clients mehrfach verwendet werden.

So funktioniert OCSP-Stapling in AWS IoT Core

Das folgende Diagramm zeigt, wie serverseitiges OCSP-Stapling in funktioniert. AWS IoT Core



1. Das Gerät muss bei benutzerdefinierten Domänen mit aktiviertem OCSP-Stapling registriert sein.
2. AWS IoT Core ruft jede Stunde den OCSP-Responder auf, um den Zertifikatsstatus abzurufen.
3. Der OCSP-Responder empfängt die Anfrage, sendet die neueste OCSP-Antwort und speichert die zwischengespeicherte OCSP-Antwort.
4. Das Gerät sendet eine `ClientHello` Nachricht, mit der der TLS-Handshake initiiert werden soll. AWS IoT Core
5. AWS IoT Core ruft die neueste OCSP-Antwort aus dem Server-Cache ab, der mit einer OCSP-Antwort des Zertifikats antwortet.
6. Der Server sendet eine `ServerHello` Nachricht an das Gerät. Der Server sendet auch das Serverzertifikat und den Zertifikatsstatus an den Client.
7. Das Gerät validiert den OCSP-Zertifikatsstatus.
8. Der TLS-Handshake ist abgeschlossen.

Vorteile der Verwendung von OCSP-Stapling im Vergleich zu clientseitigen OCSP-Prüfungen

Die Verwendung von OCSP-Stapling für Serverzertifikate bietet unter anderem die folgenden Vorteile:

Verbesserter Datenschutz

Ohne OCSP-Stapling kann das Gerät des Kunden Informationen an OCSP-Responder von Drittanbietern weitergeben, wodurch möglicherweise die Privatsphäre der Benutzer gefährdet wird. OCSP-Stapling mindert dieses Problem, indem der Server die OCSP-Antwort erhält und sie direkt an den Client sendet.

Verbesserte Zuverlässigkeit

OCSP-Stapling kann die Zuverlässigkeit sicherer Verbindungen verbessern, da es das Risiko von OCSP-Serverausfällen reduziert. Wenn OCSP-Antworten geheftet werden, fügt der Server dem Zertifikat die neueste Antwort bei. Auf diese Weise können Clients auch dann auf den Sperrstatus zugreifen, wenn der OCSP-Responder vorübergehend nicht verfügbar ist. OCSP-Stapling hilft, diese Probleme zu mindern, da der Server regelmäßig OCSP-Antworten abruft und die zwischengespeicherten Antworten in den TLS-Handshake einbezieht. Dadurch wird die Abhängigkeit von der Echtzeitverfügbarkeit der OCSP-Responder verringert.

Reduzierte Serverlast

OCSP-Stapling entlastet die OCSP-Responder bei der Beantwortung von OCSP-Anfragen auf den Server. Dies kann dazu beitragen, die Last gleichmäßiger zu verteilen und den Zertifikatsvalidierungsprozess effizienter und skalierbarer zu machen.

Reduzierte Latenz

OCSP-Stapling reduziert die Latenz, die mit der Überprüfung des Sperrstatus eines Zertifikats während des TLS-Handshakes verbunden ist. Anstatt dass der Client einen OCSP-Server separat abfragen muss, sendet der Server die Anfrage und hängt die OCSP-Antwort während des Handshakes an das Serverzertifikat an.

Das Serverzertifikat OCSP wird aktiviert in AWS IoT Core

Um das OCSP Stapling in das Serverzertifikat zu aktivieren AWS IoT Core, erstellen Sie eine Domänenkonfiguration für eine benutzerdefinierte Domäne oder aktualisieren Sie eine vorhandene benutzerdefinierte Domänenkonfiguration. Allgemeine Informationen zum Erstellen einer Domänenkonfiguration mit einer benutzerdefinierten Domäne finden Sie unter. ???

Verwenden Sie die folgenden Anweisungen, um das OCSP-Server-Stapling mithilfe von AWS-Managementkonsole oder zu aktivieren. AWS CLI

Konsole

Um das OCSP-Stapling von Serverzertifikaten zu aktivieren, verwenden Sie das AWS IoT Konsole:

1. Wählen Sie im Navigationsmenü Einstellungen und dann Domänenkonfiguration erstellen aus, oder wählen Sie eine vorhandene Domänenkonfiguration für eine benutzerdefinierte Domain aus.
2. Wenn Sie sich im vorherigen Schritt dafür entschieden haben, eine neue Domain-Konfiguration zu erstellen, wird die Seite Domain-Konfiguration erstellen angezeigt. Wählen Sie im Abschnitt Eigenschaften der Domänenkonfiguration die Option Benutzerdefinierte Domäne aus. Geben Sie die Informationen ein, um eine Domänenkonfiguration zu erstellen.

   Wenn Sie sich dafür entscheiden, eine vorhandene Domänenkonfiguration für eine benutzerdefinierte Domain zu aktualisieren, wird die Seite mit den Details zur Domänenkonfiguration angezeigt. Wählen Sie Bearbeiten aus.
3. Um das OCSP-Server-Stapling zu aktivieren, wählen Sie im Unterabschnitt Serverzertifikatkonfigurationen die Option OCSP-Stapling für Serverzertifikate aktivieren aus.
4. Wählen Sie „Domänenkonfiguration erstellen“ oder „Domänenkonfiguration aktualisieren“.

AWS CLI

Um das OCSP-Stapling von Serverzertifikaten zu aktivieren, verwenden Sie AWS CLI:

1. Wenn Sie eine neue Domänenkonfiguration für eine benutzerdefinierte Domain erstellen, kann der Befehl zum Aktivieren des OCSP-Server-Staplings wie folgt aussehen:

```
aws iot create-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" \
        --server-certificate-arns arn:aws:iot:us-
east-1:123456789012:cert/
f8c1e5480266caef0fdb1bf97dc1c82d7ba2d3e2642c5f25f5ba364fc6b79ba3 \
        --server-certificate-config "enableOCSPCheck=true|false"
```

2. Wenn Sie eine bestehende Domänenkonfiguration für eine benutzerdefinierte Domain aktualisieren, kann der Befehl zum Aktivieren des OCSP-Server-Staplings wie folgt aussehen:

```
aws iot update-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" \
        --server-certificate-arns arn:aws:iot:us-
east-1:123456789012:cert/
f8c1e5480266caef0fdb1bf97dc1c82d7ba2d3e2642c5f25f5ba364fc6b79ba3 \
        --server-certificate-config "enableOCSPCheck=true|false"
```

Weitere Informationen finden Sie unter CreateDomainConfiguration und UpdateDomainConfiguration aus der AWS IoT API-Referenz.

Konfiguration des Serverzertifikats OCSP für private Endpunkte in AWS IoT Core

Mit OCSP für private Endpunkte können Sie Ihre privaten OCSP-Ressourcen in Ihrer Amazon Virtual Private Cloud (Amazon VPC) für den Betrieb verwenden. AWS IoT Core Der Prozess beinhaltet die Einrichtung einer Lambda-Funktion, die als OCSP-Responder fungiert. Die Lambda-Funktion verwendet möglicherweise Ihre privaten OCSP-Ressourcen, um OCSP-Antworten zu erstellen, die verwendet werden. AWS IoT Core

Lambda-Funktion

Bevor Sie Server-OCSP für einen privaten Endpunkt konfigurieren, erstellen Sie eine Lambda-Funktion, die als Request for Comments (RFC) 6960-kompatibler OCSP-Responder (Online Certificate Status Protocol) fungiert und grundlegende OCSP-Antworten unterstützt. Die Lambda-Funktion akzeptiert eine Base64-Kodierung der OCSP-Anforderung im DER-Format (Distinguished Encoding Rules). Die Antwort der Lambda-Funktion ist ebenfalls eine Base64-kodierte OCSP-Antwort im DER-Format. Die Größe der Antwort darf 4 Kilobyte (KiB) nicht überschreiten. Die Lambda-Funktion muss sich in derselben Konfiguration AWS-Region wie die AWS-Konto Domänenkonfiguration befinden. Im Folgenden finden Sie Beispiele für Lambda-Funktionen.

Beispiele für Lambda-Funktionen

JavaScript

```
import * as pkijs from 'pkijs';
console.log('Loading function');

export const handler = async (event, context) => {
    const requestBytes = decodeBase64(event);
    const ocspRequest = pkijs.OCSPRequest.fromBER(requestBytes);
```

```
    console.log("Here is a better look at the OCSP request");
    console.log(ocspRequest.toJSON());

    const ocspResponse = getOcspResponse();

    console.log("Here is a better look at the OCSP response");
    console.log(ocspResponse.toJSON());

   const responseBytes = ocspResponse.toSchema().toBER();
   return encodeBase64(responseBytes);
};

function getOcspResponse() {
    const responseString = "MIIC/
woBAKCCAvgwggL0BgkrBgEFBQcwAQEEggLlMIIC4TCByqFkMGIxJzAlBgNVBAoMHlJpY2hhcmQncyBEaXNjb3VudCBMY
p5w7WOtPjp3otNtVgIBAYAAGA8yMDI0MDQyMzE4NTMyNVowDQYJKoZIhvcNAQELBQADggIBAJFRyjDAHfazNejo704Ra
+s82R1spDarr3k7Pzkod9jJhwsZ2YgushlS4Npfe4lHCdwFyZR75WXrW55aXFddy03KLz01ZLNYyxkleW3f5dgrUcRU3
DEBiyS7ZsyhKo6igWU/SY7YMSKgwBvFsqSDcOa/hRYQkxWKWJ19gcz8CIkWN7NvfIxCs6VrAdzEJwmE7y3v
+jdfhxW9JmI4xStE4K0tAR9vVOOfKs7NvxXj7oc9pCSG60xl96kaEE6PaY1YsfNTsKQ7pyCJ0s7/2q
+ieZ4AtNyzw1XBadPzPJNv6E0LvI24yQZqN5wACvtut5prMMRxAHbOy
+abLZR58wloFSELtGJ7UD96LFv1GgtC5s
+2QlzPc4bEEof7Lo1EISt3j2ibNch8LxhqTQ4ufrbhsMkpSOTFYEJVMJF6aKj/OGXBUUqgc0Jx6jjJXNQd
+l5KCY9pQFeb/wVUYC6mYqZOkNNMMJxPbHHbFnqb68yO+g5BE9011N44YXoPVJYoXxBLFX+OpRu9cqPkT9/
vlkKd+SYXQknwZ81agKzhf1HsBKabtJwNVMlBKaI8g5UGa7Bxi6ewH3ezdWiERRUK7F56OM53wto/";
    const responseBytes = decodeBase64(responseString);
    return pkijs.OCSPResponse.fromBER(responseBytes);
}

function decodeBase64(input) {
    const binaryString = atob(input);

    const byteArray = new Uint8Array(binaryString.length);
    for (var i = 0; i < binaryString.length; i++) {
        byteArray[i] = binaryString.charCodeAt(i);
    }

    return byteArray.buffer;
}

function encodeBase64(buffer) {
    var binary = '';
    const bytes = new Uint8Array( buffer );
    const len = bytes.byteLength;
```

```
    for (var i = 0; i < len; i++) {
        binary += String.fromCharCode( bytes[ i ] );
    }

    return btoa(binary);
}
```

Java

```
package com.example.ocsp.responder;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import org.bouncycastle.cert.ocsp.OCSPReq;
import org.bouncycastle.cert.ocsp.OCSPResp;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Base64;

public class LambdaResponderApplication implements RequestHandler<String, String> {
    @Override
    public String handleRequest(final String input, final Context context) {
        LambdaLogger logger = context.getLogger();

        byte[] decodedInput = Base64.getDecoder().decode(input);

        OCSPReq req;
        try {
            req = new OCSPReq(decodedInput);
        } catch (IOException e) {
            logger.log("Got an IOException creating the OCSP request: " +
 e.getMessage());
            throw new RuntimeException(e);
        }

        try {
            OCSPResp response = businessLogic.getMyResponse();
            String toReturn =
 Base64.getEncoder().encodeToString(response.getEncoded());
            return toReturn;
        } catch (Exception e) {
```

```
            logger.log("Got an exception creating the response: " + e.getMessage());
            return "";
        }
    }
}
```

Autorisieren AWS IoT um Ihre Lambda-Funktion aufzurufen

Bei der Erstellung der Domänenkonfiguration mit einem Lambda-OCSP-Responder müssen Sie die AWS IoT Erlaubnis zum Aufrufen der Lambda-Funktion erteilen, nachdem die Funktion erstellt wurde. Um die Berechtigung zu erteilen, können Sie den CLI-Befehl add-permission verwenden.

Erteilen Sie die Erlaubnis für Ihre Lambda-Funktion mithilfe der AWS CLI

1. Geben Sie nach der Eingabe Ihrer Werte den folgenden Befehl ein. Beachten Sie, dass der Wert `statement-id` eindeutig sein muss. *`Id-1234`*Ersetzen Sie durch den exakten Wert, den Sie haben, andernfalls erhalten Sie möglicherweise eine `ResourceConflictException` Fehlermeldung.

   ```
   aws lambda add-permission  \
   --function-name "ocsp-function" \
   --principal "iot.amazonaws.com" \
   --action "lambda:InvokeFunction" \
   --statement-id "Id-1234" \
   --source-arn arn:aws:iot:us-east-1:123456789012:domainconfiguration/<domain-config-name>/*
   --source-account 123456789012
   ```

   Die ARNs der IoT-Domänenkonfiguration folgen dem folgenden Muster. Das vom Dienst generierte Suffix wird vor der Erstellung nicht bekannt sein, daher müssen Sie das Suffix durch ein ersetzen. * Sie können die Berechtigung aktualisieren, sobald die Domänenkonfiguration erstellt wurde und der genaue ARN bekannt ist.

   `arn:`*`aws`*`:iot:`*`use-east-1:123456789012`*`:domainconfiguration/`*`domain-config-name/service-generated-suffix`*

2. Wenn der Befehl erfolgreich ist, gibt er eine Berechtigungsanweisung zurück, wie in diesem Beispiel. Sie können mit dem nächsten Abschnitt fortfahren, um OCSP-Stapling für private Endpunkte zu konfigurieren.

```
{
    "Statement": "{\"Sid\":\"Id-1234\",\"Effect\":\"Allow\",\"Principal
\":{\"Service\":\"iot.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction
\",\"Resource\":\"arn:aws:lambda:us-east-1:123456789012:function:ocsp-
function\",\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:iot:us-
east-1:123456789012:domainconfiguration/domain-config-name/*\"}}}"
}
```

Wenn der Befehl nicht erfolgreich ist, wird ein Fehler zurückgegeben, wie in diesem Beispiel. Sie müssen den Fehler überprüfen und korrigieren, bevor Sie fortfahren können.

```
An error occurred (AccessDeniedException) when calling the AddPermission operation:
 User: arn:aws:iam::57EXAMPLE833:user/EXAMPLE-1 is not authorized to perform:
 lambda:AddPer
mission on resource: arn:aws:lambda:us-east-1:123456789012:function:ocsp-function
```

Konfiguration des Server-OCSP-Staplings für private Endpunkte

Konsole

So konfigurieren Sie das OCSP-Stapling von Serverzertifikaten mit dem AWS IoT Konsole:

1. Wählen Sie im Navigationsmenü Einstellungen und dann Domänenkonfiguration erstellen aus, oder wählen Sie eine vorhandene Domänenkonfiguration für eine benutzerdefinierte Domain aus.
2. Wenn Sie sich im vorherigen Schritt dafür entschieden haben, eine neue Domain-Konfiguration zu erstellen, wird die Seite Domain-Konfiguration erstellen angezeigt. Wählen Sie im Abschnitt Eigenschaften der Domänenkonfiguration die Option Benutzerdefinierte Domäne aus. Geben Sie die Informationen ein, um eine Domänenkonfiguration zu erstellen.

   Wenn Sie sich dafür entscheiden, eine vorhandene Domänenkonfiguration für eine benutzerdefinierte Domain zu aktualisieren, wird die Seite mit den Details zur Domänenkonfiguration angezeigt. Wählen Sie Bearbeiten aus.
3. Um das OCSP-Server-Stapling zu aktivieren, wählen Sie im Unterabschnitt Serverzertifikatkonfigurationen die Option OCSP-Stapling für Serverzertifikate aktivieren aus.
4. Wählen Sie „Domänenkonfiguration erstellen“ oder „Domänenkonfiguration aktualisieren“.

AWS CLI

Um das OCSP-Stapling von Serverzertifikaten zu konfigurieren, verwenden Sie AWS CLI:

1. Wenn Sie eine neue Domänenkonfiguration für eine benutzerdefinierte Domain erstellen, kann der Befehl zur Konfiguration des Serverzertifikats OCSP für private Endpunkte wie folgt aussehen:

```
aws iot create-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" \
        --server-certificate-arns arn:aws:iot:us-
east-1:123456789012:cert/
f8c1e5480266caef0fdb1bf97dc1c82d7ba2d3e2642c5f25f5ba364fc6b79ba3 \
        --server-certificate-config "enableOCSPCheck=true,
 ocspAuthorizedResponderArn=arn:aws:acm:us-
east-1:123456789012:certificate/certificate_ID, ocspLambdaArn=arn:aws:lambda:us-
east-1:123456789012:function:my-function"
```

2. Wenn Sie eine vorhandene Domänenkonfiguration für eine benutzerdefinierte Domain aktualisieren, kann der Befehl zur Konfiguration des Serverzertifikats OCSP für private Endpunkte wie folgt aussehen:

```
aws iot update-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" \
        --server-certificate-arns arn:aws:iot:us-
east-1:123456789012:cert/
f8c1e5480266caef0fdb1bf97dc1c82d7ba2d3e2642c5f25f5ba364fc6b79ba3 \
        --server-certificate-config "enableOCSPCheck=true,
 ocspAuthorizedResponderArn=arn:aws:acm:us-
east-1:123456789012:certificate/certificate_ID, ocspLambdaArn=arn:aws:lambda:us-
east-1:123456789012:function:my-function"
```

Aktivieren Sie OCSP CHECK

Dies ist ein boolescher Wert, der angibt, ob die Server-OCSP-Stapling-Prüfung aktiviert ist oder nicht. Um das OCSP-Stapling von Serverzertifikaten zu aktivieren, muss dieser Wert wahr sein.

ocsp AuthorizedResponderArn

Dies ist ein Zeichenkettenwert des Amazon-Ressourcennamens (ARN) für ein in AWS Certificate Manager (ACM) gespeichertes X.509 Zertifikat. Falls angegeben, AWS IoT Core wird dieses

Zertifikat verwendet, um die Signatur der empfangenen OCSP-Antwort zu validieren. Falls nicht angegeben, AWS IoT Core wird das ausstellende Zertifikat verwendet, um die Antworten zu validieren. Das Zertifikat muss sich in derselben AWS-Konto Konfiguration AWS-Region wie die Domain befinden. Weitere Informationen zur Registrierung Ihres autorisierten Responder-Zertifikats finden Sie unter Zertifikate importieren in AWS Certificate Manager.

ocsp LambdaArn

Dies ist ein Zeichenkettenwert des Amazon-Ressourcennamens (ARN) für eine Lambda-Funktion, die als Request for Comments (RFC) 6960-konformer (OCSP) -Responder fungiert und grundlegende OCSP-Antworten unterstützt. Die Lambda-Funktion akzeptiert eine Base64-Kodierung der OCSP-Anforderung, die im DER-Format codiert ist. Die Antwort der Lambda-Funktion ist ebenfalls eine Base64-kodierte OCSP-Antwort im DER-Format. Die Größe der Antwort darf 4 Kilobyte (KiB) nicht überschreiten. Die Lambda-Funktion muss sich in derselben Konfiguration AWS-Region wie die AWS-Konto Domänenkonfiguration befinden.

Weitere Informationen finden Sie unter CreateDomainConfiguration und UpdateDomainConfiguration aus der AWS IoT API-Referenz.

Wichtige Hinweise zur Verwendung von OCSP Stapling für Serverzertifikate in AWS IoT Core

Beachten Sie Folgendes, wenn Sie das Serverzertifikat OCSP in AWS IoT Core verwenden:

1. AWS IoT Core unterstützt nur die OCSP-Responder, die über öffentliche IPv4-Adressen erreichbar sind.
2. Die OCSP-Stapling-Funktion in AWS IoT Core unterstützt autorisierte Responder nicht. Alle OCSP-Antworten müssen von der CA signiert werden, die das Zertifikat signiert hat, und die CA muss Teil der Zertifikatskette der benutzerdefinierten Domain sein.
3. Die OCSP-Stapling-Funktion in unterstützt AWS IoT Core keine benutzerdefinierten Domänen, die mit selbstsignierten Zertifikaten erstellt wurden.
4. AWS IoT Core ruft jede Stunde einen OCSP-Responder auf und speichert die Antwort im Cache. Wenn der Aufruf an den Responder fehlschlägt, AWS IoT Core wird die letzte gültige Antwort geheftet.
5. Wenn sie nicht mehr gültig `nextUpdateTime` ist, AWS IoT Core wird die Antwort aus dem Cache entfernt, und der TLS-Handshake enthält die OCSP-Antwortdaten erst beim nächsten erfolgreichen Aufruf an den OCSP-Responder. Dies kann passieren, wenn die zwischengespeicherte Antwort abgelaufen ist, bevor der Server eine gültige Antwort vom OCSP-

Responder erhält. Der Wert von `nextUpdateTime` legt nahe, dass die OCSP-Antwort bis zu diesem Zeitpunkt gültig ist. Mehr über `nextUpdateTime` erfahren Sie unter ???.

6. Manchmal kann die OCSP-Antwort AWS IoT Core nicht empfangen werden oder die vorhandene OCSP-Antwort wird entfernt, weil sie abgelaufen ist. In solchen Situationen AWS IoT Core wird weiterhin das von der benutzerdefinierten Domain bereitgestellte Serverzertifikat ohne die OCSP-Antwort verwendet.
7. Die Größe der OCSP-Antwort darf 4 KiB nicht überschreiten.

Problembehandlung beim Einheften von Serverzertifikaten durch OCSP AWS IoT Core

AWS IoT Core gibt das `RetrieveOCSPStapleData.Success` Messobjekt und die `RetrieveOCSPStapleData` Logeinträge an aus. CloudWatch Die Kennzahl und die Protokolleinträge können dabei helfen, Probleme im Zusammenhang mit dem Abrufen von OCSP-Antworten zu erkennen. Weitere Informationen erhalten Sie unter ??? und ???.

Connect to (Verbinden mit) AWS IoT FIPS-Endpunkte

AWS IoT stellt Endpunkte bereit, die den Federal Information Processing Standard (FIPS) 140-2 unterstützen. FIPS-konforme Endpunkte unterscheiden sich von Standardendpunkten. AWS Um auf eine AWS IoT FIPS-compliant Weise mit Ihnen zu interagieren, müssen Sie die unten beschriebenen Endpunkte mit Ihrem FIPS-kompatiblen Client verwenden. Die AWS IoT Konsole ist nicht FIPS-konform.

In den folgenden Abschnitten wird beschrieben, wie Sie mithilfe der REST-API, eines SDK oder des auf die FIPS-kompatiblen AWS IoT Endpunkte zugreifen. AWS CLI

Themen

- AWS IoT Core - Endpunkte der Steuerungsebene
- AWS IoT Core - Endpunkte auf Datenebene
- AWS IoT Core - Endpunkte des Anbieters von Anmeldeinformationen
- AWS IoT Device Management - Endpunkte für Jobs und Daten
- AWS IoT Device Management - Fleet Hub-Endpunkte
- AWS IoT Device Management - sichere Tunneling-Endpunkte
- AWS IoT Device Management - Endpunkte für verwaltete Integrationen

AWS IoT Core - Endpunkte der Steuerungsebene

Die FIPS-konformen AWS IoT Core – Steuerebene-Endpunkte, die die AWS IoT-Operationen unterstützen und ihre zugehörigen CLI-Befehle sind unter FIPS-Endpunkte nach Dienst aufgeführt. Suchen Sie unter FIPS-Endpunkte nach Dienst nach dem Dienst AWS IoT Core –Steuerebene und suchen Sie nach dem Endpunkt für Ihre AWS-Region.

Um den FIPS-kompatiblen Endpunkt beim Zugriff auf die AWS IoT Operationen zu verwenden, verwenden Sie das AWS SDK oder die REST-API mit dem für Sie geeigneten Endpunkt. AWS-Region

Um den FIPS-konformen Endpunkt bei der Ausführung von --endpoint CLI-Befehlen zu verwenden, fügen Sie dem Befehl den Parameter aws iot mit dem entsprechenden Endpunkt für Ihre AWS-Region hinzu.

AWS IoT Core - Endpunkte auf Datenebene

Die FIPS-konformen Endpunkte von AWS IoT Core – Datenebene sind unter FIPS-Endpunkte nach Dienst aufgeführt. Suchen Sie unter FIPS-Endpunkte nach Dienst nach dem Dienst AWS IoT Core – Datenebene und suchen Sie nach dem Endpunkt für Ihre AWS-Region.

Sie können den FIPS-kompatiblen Endpunkt für Sie AWS-Region mit einem FIPS-kompatiblen Client verwenden, indem Sie das AWS IoT Geräte-SDK verwenden und den Endpunkt für die Verbindungsfunktion des SDK anstelle des Standardendpunkts Ihres Kontos — der Datenebene AWS IoT Core— angeben. Die Verbindungsfunktion ist spezifisch für das AWS IoT Geräte-SDK. Ein Beispiel für eine Verbindungsfunktion finden Sie unter der Connection-Funktion im AWS IoT Geräte-SDK für Python.

> **Note**
>
> AWS IoT unterstützt keine AWS-Konto-spezifischen AWS IoT Core- Datenebenen-Endpunkte, die es sind. FIPS-compliant Servicefunktionen, die einen AWS-Konto-spezifischen Endpunkt in der Server Name Indication (SNI) erfordern, können nicht verwendet werden. FIPS-compliant AWS IoT Core- Endpunkte auf Datenebene können Multi-Account Registrierungszertifikate, benutzerdefinierte Domänen, benutzerdefinierte Autorisierer und konfigurierbare Endpunkte (einschließlich unterstützter TLS-Richtlinien) nicht unterstützen. ???

AWS IoT Core - Endpunkte des Anbieters von Anmeldeinformationen

Die FIPS-kompatiblen Endpunkte AWS IoT Core des Anbieters von Anmeldeinformationen sind unter FIPS-Endpunkte nach Dienst aufgeführt. Suchen Sie unter FIPS Endpoints by Service nach dem Service AWS IoT Core— Credential Provider Service und suchen Sie nach dem Endpunkt für Ihren. AWS-Region

> **Note**
>
> AWS IoT unterstützt keine AWS-Konto spezifischen AWS IoT Core Anmeldeinformationsanbieter-Endpunkte, die FIPS-compliant Dienstfunktionen, die einen AWS-Konto-spezifischen Endpunkt in der Server Name Indication (SNI) erfordern, können nicht verwendet werden. FIPS-compliant AWS IoT Core- Endpunkte von Anmeldeinformationsanbietern können Multi-Account Registrierungszertifikate, benutzerdefinierte Domänen, benutzerdefinierte Autorisierer und konfigurierbare Endpunkte (einschließlich unterstützter TLS-Richtlinien) nicht unterstützen. ???

AWS IoT Device Management - Endpunkte für Jobs und Daten

Die FIPS-konformen Endpunkte von AWS IoT Device Management – Auftragsdaten sind unter FIPS-Endpunkte nach Dienst aufgeführt. Suchen Sie unter FIPS-Endpunkte nach Dienst nach dem Dienst AWS IoT Device Management –Auftragsdaten und suchen Sie nach dem Endpunkt für Ihre AWS-Region.

Um den FIPS-konformen AWS IoT Device Management – Auftragsdaten-Endpunkt zu verwenden, wenn Sie aws iot-jobs-data CLI-Befehle ausführen, fügen Sie dem Befehl den Parameter --endpoint mit dem entsprechenden Endpunkt für Ihre AWS-Region hinzu. Sie können mit diesem Endpunkt auch die REST-API verwenden.

Wir empfehlen die Verwendung von `Data-ATS` anstelle von. `iot:Jobs` `iot:Data-ATS`unterstützt Dual-Stack-Endpunkte (IPv4 und IPv6), unterstützt aber `iot:Jobs` nur IPv4.

Sie können den FIPS-kompatiblen Endpunkt für Sie AWS-Region mit einem FIPS-kompatiblen Client verwenden, indem Sie das AWS IoT Geräte-SDK verwenden und den Endpunkt für die Verbindungsfunktion des SDK anstelle des Standardendpunkts für Auftragsdaten Ihres Kontos angeben. AWS IoT Device Management Die Verbindungsfunktion ist spezifisch für das AWS IoT Geräte-SDK. Ein Beispiel für eine Verbindungsfunktion finden Sie unter der Verbindungsfunktion im AWS IoT Geräte-SDK für Python.

AWS IoT Device Management - Fleet Hub-Endpunkte

Die FIPS-kompatiblen AWS IoT Device Management Fleet Hub-Endpunkte, die mit den CLI-Befehlen von Fleet Hub for AWS IoT Device Management verwendet werden können, sind unter FIPS-Endpunkte nach Service aufgeführt. Suchen Sie unter FIPS-Endpunkte nach Dienst nach dem AWS IoT Device Management – Fleet Hub-Dienst und suchen Sie nach dem Endpunkt für Ihre AWS-Region.

Um den FIPS-kompatiblen AWS IoT Device Management Fleet Hub-Endpunkt bei der Ausführung von aws iotfleethub CLI-Befehlen zu verwenden, fügen Sie dem Befehl den --endpoint Parameter mit dem entsprechenden Endpunkt für Sie hinzu. AWS-Region Sie können mit diesem Endpunkt auch die REST-API verwenden.

AWS IoT Device Management - sichere Tunneling-Endpunkte

Die FIPS-konformen AWS IoT Device Management – Secure Tunneling-Endpunkte für die AWS IoT Secure Tunneling API und die entsprechenden CLI-Befehle sind unter FIPS-Endpunkte nach Dienst aufgeführt. Suchen Sie unter FIPS-Endpunkte nach Dienst nach dem Dienst AWS IoT Device Management –Secure Tunneling und suchen Sie nach dem Endpunkt für Ihre AWS-Region.

Um den FIPS-konformen AWS IoT Device Management – Secure Tunneling-Endpunkt zu verwenden, wenn Sie aws iotsecuretunneling CLI-Befehle ausführen, fügen Sie dem Befehl den Parameter --endpoint mit dem entsprechenden Endpunkt für Ihre AWS-Region hinzu. Sie können mit diesem Endpunkt auch die REST-API verwenden.

AWS IoT Device Management - Endpunkte für verwaltete Integrationen

Die FIPS-kompatiblen Steuerungsebenen-Endpunkte, die die verwalteten Integrationsvorgänge und die zugehörigen AWS CLI Befehle unterstützen, sind unter FIPS Endpoints by Service aufgeführt. https://aws.amazon.com/compliance/fips/#FIPS_Endpoints_by_Service Suchen Sie unter FIPS Endpoints by Service nach dem Service AWS IoT Device Management — Managed Integrations Service und suchen Sie nach dem Endpunkt für Ihren. AWS-Region

Um den FIPS-kompatiblen Endpunkt zu verwenden, wenn Sie auf die verwalteten Integrationsvorgänge zugreifen, verwenden Sie das AWS SDK oder die REST-API mit dem für Sie geeigneten Endpunkt. AWS-Region

Um den FIPS-kompatiblen Endpunkt zu verwenden, wenn Sie CLI-Befehle für verwaltete Integrationen ausführen, fügen Sie dem Befehl den --endpoint Parameter mit dem entsprechenden Endpunkt für Sie hinzu AWS-Region .

Geräte verwalten mit AWS IoT

AWS IoT bietet eine Registrierung, die Ihnen bei der Verwaltung von Dingen hilft. Ein Objekt ist eine Darstellung eines bestimmten Geräts oder einer logischen Entität. Es kann ein physisches Gerät oder ein Sensor sein (beispielsweise eine Glühbirne oder ein Wandschalter). Es kann sich auch um eine logische Entität wie eine Instanz einer Anwendung oder eine physische Entität handeln, die keine Verbindung zu anderen Geräten herstellt, die eine Verbindung herstellen, AWS IoT aber mit diesen verknüpft sind (z. B. ein Auto mit Motorsensoren oder einem Bedienfeld).

Informationen zu einem Objekt werden in der Registry als JSON-Daten gespeichert. Hier sehen Sie ein Beispiel für ein Objekt:

```
{
     "version": 3,
    "thingName": "MyLightBulb",
    "defaultClientId": "MyLightBulb",
    "thingTypeName": "LightBulb",
    "attributes": {
        "model": "123",
        "wattage": "75"
    }
}
```

Objekte werden anhand eines Namens identifiziert. Objekte können auch Attribute in Form von Name-Wert-Paaren haben, die Sie verwenden können, um Informationen zu einem Objekt, z. B. Seriennummer oder Hersteller, zu speichern.

Ein typischer Anwendungsfall für ein Gerät involviert die Verwendung des Objektnamens als Standard-MQTT-Client-ID. Obwohl wir keine Zuordnung zwischen dem Registrierungsnamen einer Sache und ihrer Verwendung von MQTT-Client-IDs, Zertifikaten oder Shadow-Status erzwingen, empfehlen wir Ihnen, einen Dingnamen zu wählen und ihn als MQTT-Client-ID sowohl für die Registrierung als auch für den Device Shadow-Dienst zu verwenden. Dies sorgt für Ordnung und bequeme Verwaltung Ihrer IoT-Flotte, ohne dass Sie auf die Flexibilität des zugrunde liegenden Gerätezertifikatmodells oder Shadows verzichten müssen.

Sie müssen kein Objekt in der Registry erstellen, um ein Gerät mit AWS IoT zu verbinden. Durch das Hinzufügen Ihrer Objekte zur Registry können Sie Geräte einfacher verwalten und finden.

Ding-Registrierungsdaten (einschließlich Attributen, Dingtypen und Gruppenmitgliedschaften) können auch dynamisch in der Regel-Engine abgerufen werden, um sie bei der Nachrichtenverarbeitung und beim Routing zu verwenden. Weitere Informationen finden Sie unter `get_registry_data`.

Dinge mit der Registrierung verwalten

Sie verwenden die AWS IoT Konsole, die AWS IoT API oder die, AWS CLI um mit der Registrierung zu interagieren. Das folgenden Abschnitte zeigen, wie Sie die CLI zur Arbeit mit dem Registry verwenden.

Wenn du deine Objekt-Objekte benennst:

- Verwende keine persönlich identifizierbaren Informationen in deinem Dingnamen. Der Name des Objekts kann in unverschlüsselten Mitteilungen und Berichten vorkommen.

Topics

- Ein Objekt erstellen
- Objekte auflisten
- Objekte beschreiben
- Ein Objekt aktualisieren
- Ein Objekt löschen
- Ein Prinzipal an ein Objekt anfügen
- Listet Dinge auf, die mit einem Principal verknüpft sind
- Listet Prinzipale auf, die einer Sache zugeordnet sind
- Listet Dinge auf, die mit einem Principal V2 verknüpft sind
- Listet Prinzipale auf, die einer Sache zugeordnet sind (V2)
- Ein Prinzipal von einem Objekt trennen

Ein Objekt erstellen

Der folgende Befehl zeigt, wie Sie den AWS IoT CreateThing Befehl aus der CLI verwenden, um ein Ding zu erstellen. Sie können den Namen eines Objekts nicht ändern, nachdem Sie es erstellt haben. Um den Namen einer Sache zu ändern, erstellen Sie eine neue Sache, geben Sie ihr den neuen Namen und löschen Sie dann die alte Sache.

```
$ aws iot create-thing \
    --thing-type-name "MyLightBulb" \
    --attribute-payload "{\"attributes\": {\"wattage\":\"75\", \"model\":\"123\"}}"
```

Der Befehl CreateThing zeigt den Namen und den Amazon-Ressourcennamen (ARN) des neuen Objekts an:

```
{
    "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/MyLightBulb",
    "thingName": "MyLightBulb",
    "thingId": "12345678abcdefgh12345678ijklmnop12345678"
}
```

> **Note**
>
> Es wird nicht empfohlen, für Objektnamen personenbezogene Informationen zu verwenden.

Weitere Informationen finden Sie unter create-thing in der AWS CLI -Befehlsreferenz.

Objekte auflisten

Mit dem Befehl ListThings können Sie alle Objekte in Ihrem Konto auflisten:

```
$ aws iot list-things
```

```
{
    "things": [
       {
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MyLightBulb"
        },
        {
            "attributes": {
                "numOfStates":"3"
             },
            "version": 11,
```

```
            "thingName": "MyWallSwitch"
        }
    ]
}
```

Sie können den Befehl ListThings verwenden, um nach allen Objekten eines bestimmten Objekttyps zu suchen:

```
$  aws iot list-things --thing-type-name "LightBulb"
```

```
{
    "things": [
        {
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MyRGBLight"
        },
        {
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MySecondLightBulb"
        }
    ]
}
```

Mit dem Befehl ListThings können Sie nach allen Objekten suchen, die über ein Attribut mit einem bestimmten Wert verfügen. Mit diesem Befehl werden bis zu drei Attribute durchsucht.

```
$  aws iot list-things --attribute-name "wattage" --attribute-value "75"
```

```
{
    "things": [
```

```
        {
            "thingTypeName": "StopLight",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 3,
            "thingName": "MyLightBulb"
        },
        {
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MyRGBLight"
        },
        {
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MySecondLightBulb"
        }
    ]
}
```

Weitere Informationen finden Sie unter list-things in der AWS CLI Befehlsreferenz.

Objekte beschreiben

Mit dem Befehl DescribeThing können Sie detailliertere Informationen zu einem Objekt anzeigen:

```
$ aws iot describe-thing --thing-name "MyLightBulb"
{
    "version": 3,
    "thingName": "MyLightBulb",
    "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/MyLightBulb",
    "thingId": "12345678abcdefgh12345678ijklmnop12345678",
    "defaultClientId": "MyLightBulb",
```

```
    "thingTypeName": "StopLight",
    "attributes": {
        "model": "123",
        "wattage": "75"
    }
}
```

Sie können auch innerhalb der Regel-Engine mithilfe der Inline-Funktion auf diese API zugreifen`get_registry_data()`. Sie können diese Funktion verwenden, um dynamisch auf Dingregistrierungsinformationen (einschließlich Attribute, Dingtypen und Gruppenmitgliedschaften) zuzugreifen und diese zu verwenden, indem Sie `ListThingGroupsForThing` APIs direkt innerhalb von AWS IoT Regeln aufrufen `DescribeThing` und so die Nachrichtenverarbeitung und das Routing in Echtzeit auf der Grundlage Ihrer Geräteregistrierungsdaten ermöglichen. Weitere Informationen finden Sie unter `get_registry_data`.

Weitere Informationen finden Sie in der Befehlsreferenz unter describe-thing. AWS CLI

Ein Objekt aktualisieren

Mit dem Befehl UpdateThing können Sie ein Objekt aktualisieren. Dieser Befehl aktualisiert nur die Attribute des Dings. Sie können den Namen eines Objekts nicht ändern. Um den Namen einer Sache zu ändern, erstellen Sie eine neue Sache, geben Sie ihr den neuen Namen und löschen Sie dann die alte Sache.

```
$ aws iot update-thing --thing-name "MyLightBulb" --attribute-payload "{\"attributes\":
 {\"wattage\":\"150\", \"model\":\"456\"}}"
```

Der Befehl UpdateThing erzeugt keine Ausgabe. Mit dem Befehl DescribeThing können Sie das Ergebnis anzeigen:

```
$ aws iot describe-thing --thing-name "MyLightBulb"
{
    "attributes": {
        "model": "456",
        "wattage": "150"
    },
    "version": 2,
    "thingName": "MyLightBulb"
}
```

Weitere Informationen finden Sie unter update-thing in der AWS CLI Befehlsreferenz.

Ein Objekt löschen

Mit dem Befehl DeleteThing können Sie ein Objekt löschen:

```
$ aws iot delete-thing --thing-name "MyThing"
```

Dieser Befehl wird erfolgreich und ohne Fehler zurückgegeben, wenn der Löschvorgang erfolgreich ist oder Sie ein Objekt angeben, das nicht vorhanden ist.

Weitere Informationen finden Sie unter delete-thing in der AWS CLI Befehlsreferenz.

Ein Prinzipal an ein Objekt anfügen

Ein physisches Gerät kann einen Principal verwenden, um mit ihm zu kommunizieren AWS IoT. Ein Principal kann ein X.509 Zertifikat oder eine Amazon Cognito-ID sein. Sie können dem Objekt in der Registrierung, das Ihr Gerät repräsentiert, ein Zertifikat oder eine Amazon Cognito-ID zuordnen, indem Sie den Befehl https://docs.aws.amazon.com/cli/latest/reference/iot/attach-thing-principal.html attach-thing-principal ausführen.

Um ein Zertifikat oder eine Amazon Cognito-ID an Ihr Ding anzuhängen, verwenden Sie den Befehl attach-thing-principal: https://docs.aws.amazon.com/cli/latest/reference/iot/attach-thing-principal.html

```
$ aws iot attach-thing-principal \
    --thing-name "MyLightBulb1" \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847"
```

Um Ihrem Ding ein Zertifikat mit einem Anhangstyp (exklusiver Anhang oder nicht exklusiver Anhang) anzuhängen, verwenden Sie den Befehl _ attach-thing-principal und geben Sie einen Typ in das Feld ein. `--thing-principal-type` Ein exklusiver Anhang bedeutet, dass Ihr IoT-Ding das einzige Objekt ist, das an das Zertifikat angehängt ist, und dieses Zertifikat kann mit keinen anderen Dingen verknüpft werden. Ein nicht exklusiver Anhang bedeutet, dass Ihr IoT-Ding an das Zertifikat angehängt ist und dieses Zertifikat mit anderen Dingen verknüpft werden kann. Weitere Informationen finden Sie unter ???.

> **Note**
>
> Für ??? diese Funktion können Sie das X.509 Zertifikat nur als Prinzipal verwenden.

```
$ aws iot attach-thing-principal \
    --thing-name "MyLightBulb2" \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847" \
    --thing-principal-type "EXCLUSIVE_THING"
```

Wenn die Anlage erfolgreich ist, erzeugt der AttachThingPrincipal Befehl keine Ausgabe. Verwenden Sie den CLI-Befehl list-thing-principals-v2, um den Anhang zu beschreiben.

Weitere Informationen finden Sie in der API-Referenz. AttachThingPrincipal AWS IoT Core

Listet Dinge auf, die mit einem Principal verknüpft sind

Führen Sie den `list-principal-things` Befehl aus, um die Dinge aufzulisten, die dem angegebenen Prinzipal zugeordnet sind. Beachten Sie, dass dieser Befehl den Anhangstyp zwischen dem Ding und dem Zertifikat nicht auflistet. Verwenden Sie den `list-principal-things-v2` Befehl, um den Anhangstyp aufzulisten. Weitere Informationen finden Sie unter ???.

```
$ aws iot list-principal-things \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
```

Die Ausgabe kann wie folgt aussehen.

```
{
    "things": [
        "MyLightBulb1",
        "MyLightBulb2"
    ]
}
```

Weitere Informationen finden Sie in ListPrincipalThings der AWS IoT Core API-Referenz.

Listet Prinzipale auf, die einer Sache zugeordnet sind

Führen Sie den Befehl aus, um die Prinzipale aufzulisten, die dem angegebenen Ding zugeordnet sind. `list-thing-principals` Beachten Sie, dass dieser Befehl nicht den Anhangstyp zwischen

dem Ding und dem Zertifikat auflistet. Verwenden Sie den `list-thing-principals-v2` Befehl, um den Anhangstyp aufzulisten. Weitere Informationen finden Sie unter ???.

```
$ aws iot list-thing-principals \
    --thing-name "MyLightBulb1"
```

Die Ausgabe kann wie folgt aussehen.

```
{
    "principals": [
        "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8",
        "arn:aws:iot:us-
east-1:123456789012:cert/
1a234b39b4b68278f2e9d84bf97eac2cbf4a1c28b23ea29a44559b9bcf8d395b"
    ]
}
```

Weitere Informationen finden Sie in ListThingPrincipals der AWS IoT Core API-Referenz.

Listet Dinge auf, die mit einem Principal V2 verknüpft sind

Führen Sie den `list-principal-things-v2` Befehl aus, um die mit dem angegebenen Zertifikat verknüpften Elemente zusammen mit dem Anhangstyp aufzulisten. Der Anhangstyp bezieht sich darauf, wie das Zertifikat an das Ding angehängt wird.

```
$ aws iot list-principal-things-v2 \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
```

Die Ausgabe kann wie folgt aussehen.

```
{
    "PrincipalThingObjects": [
        {
            "thingPrincipalType": "NON_EXCLUSIVE_THING",
            "thing": "arn:aws:iot:us-east-1:123456789012:thing/thing_1"
        },
        {
```

```
            "thingPrincipalType": "NON_EXCLUSIVE_THING",
            "thing": "arn:aws:iot:us-east-1:123456789012:thing/thing_2"
        }

    ]
}
```

Weitere Informationen finden Sie in ListPrincipalThingsV2 der AWS IoT Core API-Referenz.

Listet Prinzipale auf, die einer Sache zugeordnet sind (V2)

Führen Sie den `list-thing-principals-V2` Befehl aus, um die mit der angegebenen Sache verknüpften Zertifikate zusammen mit dem Anhangstyp aufzulisten. Der Anhangstyp bezieht sich darauf, wie das Zertifikat an das Ding angehängt wird.

```
$ aws iot list-thing-principals-v2 \
    --thing-name "thing_1"
```

Die Ausgabe kann wie folgt aussehen.

```
{
    "ThingPrincipalObjects": [
        {
            "thingPrincipalType": "NON_EXCLUSIVE_THING",
            "principal": "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
        },
        {
            "thingPrincipalType": "NON_EXCLUSIVE_THING",
            "principal": "arn:aws:iot:us-
east-1:123456789012:cert/
1a234b39b4b68278f2e9d84bf97eac2cbf4a1c28b23ea29a44559b9bcf8d395b"
        }
    ]
}
```

Weitere Informationen finden Sie in ListThingsPrincipalV2 der AWS IoT Core API-Referenz.

Ein Prinzipal von einem Objekt trennen

Mit dem Befehl `DetachThingPrincipal` können Sie ein Zertifikat von einem Objekt trennen:

```
$ aws iot detach-thing-principal \
    --thing-name "MyLightBulb" \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
```

Der Befehl DetachThingPrincipal erzeugt keine Ausgabe.

Weitere Informationen finden Sie unter detach-thing-principal aus der API-Referenz. AWS IoT Core

Objekttypen

Mit Objekttypen können Sie Informationen zur Beschreibung und Konfiguration speichern, die auf alle Objekte desselben Objekttyps zutreffen. Dadurch wird die Verwaltung der Objekte im Registry vereinfacht. Sie können beispielsweise einen Dingtyp definieren. LightBulb Alle Dinge, die mit dem LightBulb Dingtyp verknüpft sind, haben eine Reihe von Attributen gemeinsam: Seriennummer, Hersteller und Wattleistung. Wenn Sie ein Objekt vom Typ erstellen LightBulb (oder den Typ eines vorhandenen Objekts ändern LightBulb), können Sie Werte für jedes der LightBulb im Dingtyp definierten Attribute angeben.

Auch wenn Objekttypen optional sind, sind sie hilfreich, um Objekte leichter zu erkennen.

- Objekte mit Objekttyp können bis zu 50 Attribute haben.
- Objekte ohne Objekttyp können bis zu drei Attribute haben.
- Ein Objekt kann nur mit einem Objekttyp verknüpft werden.
- Für die Anzahl der Objekttypen in Ihrem Konto ist keine Einschränkung vorhanden.

Es ist nicht möglich, einen Objekttyp nach dem Erstellen umzubenennen. Sie können einen Objekttyp jederzeit als veraltet einstufen, um zu verhindern, dass neue Objekte mit diesem verknüpft werden. Außerdem können Sie Objekttypen, mit denen keine Objekte verknüpft sind, löschen.

Themen:

- Objekttyp erstellen
- Objekttypen auflisten
- Einen Objekttyp beschreiben
- Einen Objekttyp mit einem Objekt verknüpfen

- Aktualisiere einen Dingtyp
- Einen Objekttyp als veraltet einstufen
- Einen Objekttyp löschen

Objekttyp erstellen

Mit dem Befehl CreateThingType können Sie einen Objekttyp erstellen:

```
$ aws iot create-thing-type

                --thing-type-name "LightBulb" --thing-type-properties
 "thingTypeDescription=light bulb type, searchableAttributes=wattage,model"
```

Der Befehl CreateThingType gibt eine Reaktion zurück, die Objekttyp und ARN enthält:

```
{
    "thingTypeName": "LightBulb",
    "thingTypeId": "df9c2d8c-894d-46a9-8192-9068d01b2886",
    "thingTypeArn": "arn:aws:iot:us-west-2:123456789012:thingtype/LightBulb"
}
```

Objekttypen auflisten

Mit dem Befehl ListThingTypes können Sie Objekttypen auflisten:

```
$ aws iot list-thing-types
```

Der ListThingTypes Befehl gibt eine Liste der Dingtypen zurück, die in Ihrem AWS-Konto: definiert sind:

```
{
    "thingTypes": [
        {
            "thingTypeName": "LightBulb",
            "thingTypeProperties": {
                "searchableAttributes": [
                    "wattage",
                    "model"
                ],
```

```
                "thingTypeDescription": "light bulb type"
            },
            "thingTypeMetadata": {
                "deprecated": false,
                "creationDate": 1468423800950
            }
        }
    ]
}
```

Einen Objekttyp beschreiben

Mit dem Befehl DescribeThingType können Sie Informationen zu einem Objekttyp abrufen:

```
$ aws iot describe-thing-type --thing-type-name "LightBulb"
```

Der Befehl DescribeThingType gibt Informationen zum angegebenen Typ zurück:

```
{
    "thingTypeProperties": {
        "searchableAttributes": [
            "model",
            "wattage"
        ],
        "thingTypeDescription": "light bulb type"
    },
    "thingTypeId": "df9c2d8c-894d-46a9-8192-9068d01b2886",
    "thingTypeArn": "arn:aws:iot:us-west-2:123456789012:thingtype/LightBulb",
    "thingTypeName": "LightBulb",
    "thingTypeMetadata": {
        "deprecated": false,
        "creationDate": 1544466338.399
    }
}
```

Einen Objekttyp mit einem Objekt verknüpfen

Mit dem Befehl CreateThing können Sie beim Erstellen eines Objekts einen Objekttyp festlegen:

```
$ aws iot create-thing --thing-name "MyLightBulb" --thing-type-name "LightBulb" --
attribute-payload "{\"attributes\": {\"wattage\":\"75\", \"model\":\"123\"}}"
```

Mit dem Befehl UpdateThing können Sie den mit einem Objekt verknüpften Objekttyp jederzeit ändern:

```
$ aws iot update-thing --thing-name "MyLightBulb"
                --thing-type-name "LightBulb" --attribute-payload  "{\"attributes\":
 {\"wattage\":\"75\", \"model\":\"123\"}}"
```

Mit dem Befehl UpdateThing können Sie außerdem die Verknüpfung eines Objekts mit einem Objekttyp aufheben.

Aktualisiere einen Dingtyp

Sie können den UpdateThingType Befehl verwenden, um einen Dingtyp zu aktualisieren, wenn Sie ein Ding erstellen:

```
$ aws iot create-thing --thing-name "MyLightBulb" --thing-type-name "LightBulb" --
attribute-payload "{\"attributes\": {\"wattage\":\"75\", \"model\":\"123\"}}"
```

Mit dem Befehl UpdateThing können Sie den mit einem Objekt verknüpften Objekttyp jederzeit ändern:

```
$ aws iot update-thing --thing-name "MyLightBulb"
                --thing-type-name "LightBulb" --attribute-payload  "{\"attributes\":
 {\"wattage\":\"75\", \"model\":\"123\"}}"
```

Mit dem Befehl UpdateThing können Sie außerdem die Verknüpfung eines Objekts mit einem Objekttyp aufheben.

Einen Objekttyp als veraltet einstufen

Objekttypen sind unveränderlich. Nach ihrer Definition können sie nicht geändert werden. Sie können einen Objekttyp jedoch als veraltet einstufen, um zu verhindern, dass Benutzer neue Objekte mit diesem verknüpfen. Alle bestehenden Objekte, die mit diesem Objekttyp verknüpft sind, bleiben unverändert.

Mit dem Befehl DeprecateThingType können Sie einen Objekttyp als veraltet einstufen:

```
$ aws iot deprecate-thing-type --thing-type-name "myThingType"
```

Mit dem Befehl DescribeThingType können Sie das Ergebnis anzeigen:

```
$ aws iot describe-thing-type --thing-type-name "StopLight":
```

```
{
    "thingTypeName": "StopLight",
    "thingTypeProperties": {
        "searchableAttributes": [
            "wattage",
            "numOfLights",
            "model"
        ],
        "thingTypeDescription": "traffic light type",
    },
    "thingTypeMetadata": {
        "deprecated": true,
        "creationDate": 1468425854308,
        "deprecationDate": 1468446026349
    }
}
```

Einen Objekttyp als veraltet einzustufen kann rückgängig gemacht werden. Mit der Markierung `--undo-deprecate` mit dem CLI-Befehl DeprecateThingType können Sie die Einstufung als veraltet rückgängig machen:

```
$ aws iot deprecate-thing-type --thing-type-name "myThingType" --undo-deprecate
```

Mit dem CLI-Befehl DescribeThingType können Sie das Ergebnis anzeigen:

```
$ aws iot describe-thing-type --thing-type-name "StopLight":
```

```
{
    "thingTypeName": "StopLight",
    "thingTypeArn": "arn:aws:iot:us-east-1:123456789012:thingtype/StopLight",
    "thingTypeId": "12345678abcdefgh12345678ijklmnop12345678"
    "thingTypeProperties": {
        "searchableAttributes": [
            "wattage",
            "numOfLights",
            "model"
        ],
```

```
        "thingTypeDescription": "traffic light type"
    },
    "thingTypeMetadata": {
        "deprecated": false,
        "creationDate": 1468425854308,
    }
}
```

Einen Objekttyp löschen

Sie können Objekttypen erst löschen, nachdem sie als veraltet eingestuft wurden. Mit dem Befehl DeleteThingType können Sie einen Objekttyp löschen:

```
$ aws iot delete-thing-type --thing-type-name "StopLight"
```

> **Note**
>
> Bevor Sie einen Dingtyp löschen können, warten Sie fünf Minuten, nachdem Sie ihn als veraltet markiert haben.

Statische Objektgruppen

Mithilfe von statischen Objektgruppen können Sie verschiedene Objekte gleichzeitig verwalten, indem Sie sie in Gruppen zusammenfassen. Statische Objektgruppen enthalten eine Gruppe von Objekten, die über die Konsole, die CLI oder die API verwaltet werden. Dynamische Objektgruppen dagegen enthalten Objekte, die mit einer angegebenen Abfrage übereinstimmen. Statische Objektgruppen können auch andere statische Objektgruppen enthalten - Sie können eine Gruppenhierarchie aufbauen. Sie können eine Richtlinie mit einer übergeordneten Gruppe verbinden, die von den dieser untergeordneten Gruppen und von allen Objekten in der Gruppe sowie allen untergeordneten Gruppen übernommen wird. Dies vereinfacht die Steuerung von Berechtigungen für große Zahlen von Objekten.

> **Note**
>
> Ding-Gruppenrichtlinien erlauben keinen Zugriff auf Operationen auf AWS IoT Greengrass Datenebene. Um einem Ding den Zugriff auf einen Vorgang auf AWS IoT Greengrass Datenebene zu ermöglichen, fügen Sie die Berechtigung einer AWS IoT Richtlinie

hinzu, die Sie dem Zertifikat des Dings beifügen. Weitere Informationen finden Sie unter Geräteauthentifizierung und -autorisierung im AWS IoT Greengrass Entwicklerhandbuch.

Folgende Aktionen sind mit statischen Objektgruppen möglich:

- Eine Gruppe erstellen, beschreiben oder löschen.
- Ein Objekt zu einer Gruppe oder zu mehreren Gruppen hinzufügen.
- Ein Objekt aus einer Gruppe entfernen.
- Die erstellten Gruppen auflisten.
- Alle (direkt oder indirekt) untergeordneten Gruppen einer Gruppe auflisten.
- Die Objekte in einer Gruppe, einschließlich aller Objekte in dieser untergeordneten Gruppen, auflisten.
- Alle (direkt oder indirekt) übergeordneten Gruppen einer Gruppe auflisten.
- Die Attribute einer Gruppe hinzufügen, löschen oder aktualisieren. (Attribute sind Name/Wert-Paare, mit denen Sie Informationen zu einer Gruppe speichern können.)
- Eine Richtlinie mit einer Gruppe verbinden oder von dieser entfernen.
- Die mit einer Gruppe verbundenen Richtlinien auflisten.
- Die von einem Objekt (aufgrund der mit seiner Gruppe oder einer der dieser übergeordneten Gruppen verbundenen Richtlinie) übernommenen Richtlinien auflisten.
- Konfiguration von Protokollierungsoptionen für Objekte in einer Gruppe. Siehe Konfiguration AWS IoT Protokollierung.
- Erstellen von Aufträgen, die an jedes Objekt in einer Gruppe und in dieser untergeordneten Gruppen gesendet und dort ausgeführt werden. Siehe AWS IoT Jobs.

> **Note**
>
> Wenn ein Ding an eine statische Dinggruppe angehängt ist, an die eine AWS IoT Core Richtlinie angehängt ist, muss der Name des Dings mit der Client-ID übereinstimmen.

Hier sind Sie einige Einschränkungen für statische Objektgruppen:

- Eine Gruppe kann höchstens eine direkt übergeordnete Gruppe haben.

- Wenn eine Gruppe einer anderen Gruppe untergeordnet ist, geben Sie dies bei der Erstellung an.
- Sie können das übergeordnete Element einer Gruppe nicht später ändern. Planen Sie daher die Gruppenhierarchie, und erstellen Sie eine übergeordnete Gruppe, bevor Sie untergeordnete Gruppen erstellen.
- Die Anzahl der Gruppen, zu denen ein Objekt gehören kann, ist begrenzt.
- Ein Objekt kann nicht mehr als einer Gruppe in derselben Hierarchie hinzugefügt werden. (Anders ausgedrückt: Sie können ein Objekt nicht zwei Gruppen mit derselben übergeordneten Gruppe hinzufügen).
- Gruppen können nicht umbenannt werden.
- Namen der Objektgruppen dürfen keine internationale Zeichen enthalten, z. B. û, é und ñ.
- Verwende keine persönlich identifizierbaren Informationen in deinem Ding-Gruppennamen. Der Name der Objektgruppe kann in unverschlüsselten Mitteilungen und Berichten vorkommen.

Das Anfügen bzw. Entfernen von Richtlinien zu/von Gruppen kann die Sicherheit Ihrer AWS IoT -Operationen unter verschiedenen Aspekten verbessern. Das Anfügen einer Richtlinie pro Gerät zu einem Zertifikat, das wiederum mit einem Objekt verbunden wird, ist zeitaufwändig und erschwert die schnelle Aktualisierung oder Änderung von Richtlinien für eine ganze Geräteflotte. Das Anhängen einer Richtlinie an die Gruppe des Objekts erspart Schritte, wenn die Zertifikate auf einem Objekt rotiert werden müssen. Dazu werden Richtlinien dynamisch auf Objekte angewendet, wenn sich deren Gruppenmitgliedschaft ändert, Sie müssen daher nicht jedes Mal, wenn ein Gerät Mitglied einer anderen Gruppe wird, einen komplexen Satz von Berechtigungen neu erstellen.

Erstellen einer statischen Objektgruppe

Mit dem Befehl CreateThingGroup können Sie eine statische Objektgruppe erstellen:

```
$ aws iot create-thing-group --thing-group-name LightBulbs
```

Der Befehl CreateThingGroup gibt eine Antwort zurück, die den Namen, die ID und den ARN der statischen Objektgruppe enthält:

```
{
    "thingGroupName": "LightBulbs",
    "thingGroupId": "abcdefgh12345678ijklmnop12345678qrstuvwx",
    "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs"
}
```

> **Note**
>
> Es wird nicht empfohlen, für Objektgruppennamen personenbezogene Informationen zu verwenden.

Hier ist ein Beispiel, bei dem bei der Erstellung eine der statischen Objektgruppe übergeordnete Gruppe angegeben wird:

```
$ aws iot create-thing-group --thing-group-name RedLights --parent-group-name
 LightBulbs
```

Wie zuvor gibt der Befehl CreateThingGroup eine Antwort zurück, die den Namen, die ID und den ARN der statischen Objektgruppe enthält:

```
{
    "thingGroupName": "RedLights",
    "thingGroupId": "abcdefgh12345678ijklmnop12345678qrstuvwx",
    "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights",
}
```

> **Important**
>
> Beachten Sie beim Erstellen von Objektgruppenhierarchien die folgenden Einschränkungen:
>
> - Zu einer Objektgruppe kann es nur eine direkte übergeordnete Gruppe geben.
> - Die Anzahl der direkten untergeordneten Gruppen, die eine Objektgruppe haben kann, ist begrenzt.
> - Die maximale Tiefe einer Gruppenhierarchie ist begrenzt.
> - Die Anzahl der Attribute, die eine Objektgruppe haben kann, ist begrenzt. (Attribute sind Name/Wert-Paare, mit denen Sie Informationen zu einer Gruppe speichern können.) Die Längen der einzelnen Attributnamen und Werte sind ebenfalls begrenzt.

Beschreiben einer Objektgruppe

Mit dem Befehl DescribeThingGroup können Sie Informationen zu einer Objektgruppe abrufen:

```
$ aws iot describe-thing-group --thing-group-name RedLights
```

Der Befehl DescribeThingGroup gibt Informationen zur angegebenen Gruppe zurück:

```
{
    "thingGroupName": "RedLights",
    "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights",
    "thingGroupId": "12345678abcdefgh12345678ijklmnop12345678",
    "version": 1,
    "thingGroupMetadata": {
        "creationDate": 1478299948.882
        "parentGroupName": "Lights",
        "rootToParentThingGroups": [
            {
                "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
ShinyObjects",
                "groupName": "ShinyObjects"
            },
            {
                "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs",
                "groupName": "LightBulbs"
            }
        ]
    },
    "thingGroupProperties": {
        "attributePayload": {
            "attributes": {
                "brightness": "3400_lumens"
            },
        },
        "thingGroupDescription": "string"
    },
}
```

Hinzufügen eines Objekts zu einer statischen Objektgruppe

Mit dem Befehl AddThingToThingGroup können Sie einer statischen Objektgruppe ein Objekt hinzufügen:

```
$ aws iot add-thing-to-thing-group --thing-name MyLightBulb --thing-group-name
 RedLights
```

Der Befehl AddThingToThingGroup erzeugt keine Ausgabe.

> **Important**
>
> Sie können ein Objekt maximal 10 Gruppen hinzufügen. Ein Objekt kann jedoch nicht mehr als einer Gruppe in derselben Hierarchie hinzugefügt werden. (Anders ausgedrückt: Sie können ein Objekt nicht zwei Gruppen mit derselben übergeordneten Gruppe hinzufügen). Wenn ein Objekt zu der maximalen Anzahl an Objektgruppen gehört und es sich bei einer oder mehreren dieser Gruppen um (eine) dynamische Objektgruppe(n) handelt, können Sie mit der Markierung overrideDynamicGroups statischen Gruppen den Vorrang vor dynamischen Gruppen geben.

Entfernen eines Objekts aus einer statischen Objektgruppe

Mit dem Befehl RemoveThingFromThingGroup können Sie ein Objekt aus einer Objektgruppe entfernen:

```
$ aws iot remove-thing-from-thing-group --thing-name MyLightBulb --thing-group-name
 RedLights
```

Der Befehl RemoveThingFromThingGroup erzeugt keine Ausgabe.

Auflisten von Objekten in einer Objektgruppe

Mit dem Befehl ListThingsInThingGroup können Sie die Objekte einer Gruppe auflisten:

```
$ aws iot list-things-in-thing-group --thing-group-name LightBulbs
```

Der Befehl ListThingsInThingGroup gibt eine Liste der Objekte in der betreffenden Gruppe aus:

```
{
    "things":[
        "TestThingA"
    ]
}
```

Mit dem Parameter `--recursive` können Sie die Objekte einer Gruppe und in allen dieser untergeordneten Gruppen auflisten:

```
$ aws iot list-things-in-thing-group --thing-group-name LightBulbs --recursive
```

```
{
    "things":[
        "TestThingA",
        "MyLightBulb"
    ]
}
```

> **Note**
>
> Diese Operation ist letztlich konsistent. Mit anderen Worten, Änderungen an der Dinggruppe werden möglicherweise nicht sofort übernommen.

Auflisten von Objektgruppen

Mit dem Befehl ListThingGroups können Sie die Objektgruppen Ihres Kontos auflisten:

```
$ aws iot list-thing-groups
```

Der ListThingGroups Befehl gibt eine Liste der Dinggruppen in Ihrem zurück AWS-Konto:

```
{
    "thingGroups": [
        {
            "groupName": "LightBulbs",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs"
        },
        {
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        },
        {
            "groupName": "RedLEDLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLEDLights"
        },
        {
            "groupName": "RedIncandescentLights",
```

```
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
RedIncandescentLights"
        }
        {
            "groupName": "ReplaceableObjects",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
ReplaceableObjects"
        }
    ]
}
```

Verwenden Sie die optionalen Filter, um die Gruppen aufzulisten, die eine bestimmte Gruppe als übergeordnete Gruppe haben (`--parent-group`), oder deren Namen mit einem bestimmten Präfix beginnen (`--name-prefix-filter`). Mit dem Parameter `--recursive` können Sie alle untergeordneten Gruppen auflisten, nicht nur die direkt untergeordneten Gruppen einer Objektgruppe:

```
$ aws iot list-thing-groups --parent-group LightBulbs
```

In diesem Fall gibt der ListThingGroups Befehl eine Liste der direkten untergeordneten Gruppen der Dinggruppe zurück, die in Ihrem definiert sind AWS-Konto:

```
{
    "childGroups":[
        {
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        }
    ]
}
```

Verwenden Sie den Parameter `--recursive` mit dem Befehl ListThingGroups, um alle (und nicht nur die direkt) untergeordneten Gruppen einer Objektgruppe aufzulisten:

```
$ aws iot list-thing-groups --parent-group LightBulbs --recursive
```

Der Befehl ListThingGroups gibt eine Liste aller untergeordneten Gruppen einer Objektgruppe zurück:

```
{
```

```
    "childGroups":[
        {
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        },
        {
            "groupName": "RedLEDLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLEDLights"
        },
        {
            "groupName": "RedIncandescentLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
RedIncandescentLights"
        }
    ]
}
```

> **Note**
>
> Diese Operation ist letztlich konsistent. Mit anderen Worten, Änderungen an der Dinggruppe werden möglicherweise nicht sofort übernommen.

Auflisten von Gruppen für ein Objekt

Mit dem Befehl ListThingGroupsForThing können Sie die Objekte einer Gruppe auflisten, zu denen ein Objekt gehört:

```
$ aws iot list-thing-groups-for-thing --thing-name MyLightBulb
```

Der Befehl ListThingGroupsForThing gibt eine Liste aller Objektgruppen aus, zu denen dieses Objekt gehört:

```
{
    "thingGroups":[
        {
            "groupName": "LightBulbs",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs"
        },
        {
```

```
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        },
        {
            "groupName": "ReplaceableObjects",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
ReplaceableObjects"
        }
    ]
}
```

Sie können auch innerhalb der Regel-Engine mithilfe der Inline-Funktion auf diese API zugreifen`get_registry_data()`. Sie können diese Funktion verwenden, um dynamisch auf Dingregistrierungsinformationen (einschließlich Attribute, Dingtypen und Gruppenmitgliedschaften) zuzugreifen und diese zu verwenden, indem Sie `ListThingGroupsForThing` APIs direkt innerhalb von AWS IoT Regeln aufrufen `DescribeThing` und so die Nachrichtenverarbeitung und das Routing in Echtzeit auf der Grundlage Ihrer Geräteregistrierungsdaten ermöglichen. Weitere Informationen finden Sie unter `get_registry_data`.

Aktualisieren einer statischen Objektgruppe

Mit dem Befehl UpdateThingGroup können Sie die Attribute einer statischen Objektgruppe aktualisieren:

```
$ aws iot update-thing-group --thing-group-name "LightBulbs" --thing-group-properties
 "thingGroupDescription=\"this is a test group\", attributePayload=\"{\"attributes
\"={\"Owner\"=\"150\",\"modelNames\"=\"456\"}}"
```

Der UpdateThingGroup Befehl gibt eine Antwort zurück, die die Versionsnummer der Gruppe nach der Aktualisierung enthält:

```
{
    "version": 4
}
```

> **Note**
>
> Die Anzahl der Attribute, die ein Objekt haben kann, ist begrenzt.

Löschen einer Objektgruppe

Mit dem Befehl DeleteThingGroup können Sie eine Objektgruppe löschen:

```
$ aws iot delete-thing-group --thing-group-name "RedLights"
```

Der Befehl DeleteThingGroup erzeugt keine Ausgabe.

> **Important**
>
> Wenn Sie versuchen, eine Objektgruppe zu löschen, zu der untergeordnete Gruppen gehören, führt dies zu einem Fehler:
>
> ```
> A client error (InvalidRequestException) occurred when calling the
>  DeleteThingGroup
> operation: Cannot delete thing group : RedLights when there are still child
>  groups attached to it.
> ```
>
> Bevor Sie die Gruppe löschen, löschen Sie zuerst alle untergeordneten Gruppen.

Sie können eine Gruppe löschen, die untergeordnete Objekte hat, Berechtigungen, die aufgrund der Mitgliedschaft des Objekts in dieser Gruppe bestehen, gelten dann jedoch nicht mehr. Prüfen Sie vor dem Löschen einer Gruppe, der eine Richtlinie zugeordnet ist, sorgfältig, dass das Entfernen dieser Berechtigungen nicht dazu führt, dass die Objekte in der Gruppe ihre Funktionen nicht mehr korrekt ausführen können. Außerdem können Befehle, die anzeigen, zu welchen Gruppen ein Objekt gehört (z. B.ListGroupsForThing), die Gruppe weiterhin anzeigen, während Datensätze in der Cloud aktualisiert werden.

Anfügen einer Richtlinie an eine statische Objektgruppe

Mit dem Befehl AttachPolicy können Sie eine Richtlinie an eine statische Objektgruppe und damit auch an alle Objekte in dieser Gruppe und in allen ihren untergeordneten Gruppen anfügen:

```
$ aws iot attach-policy \
  --target "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs" \
  --policy-name "myLightBulbPolicy"
```

Der Befehl AttachPolicy erzeugt keine Ausgabe

> **Important**
>
> Sie können maximal zwei Richtlinien an eine Gruppe anhängen.

> **Note**
>
> Es wird nicht empfohlen, für Richtliniennamen personenbezogene Informationen zu verwenden.

Der Parameter `--target` kann ein Objektsgruppen-ARN (wie oben), ein Zertifikat-ARN oder eine Amazon Cognito-Identität sein. Weitere Informationen zu Richtlinien, Zertifikaten und Authentifizierung finden Sie unter Authentifizierung.

Weitere Informationen finden Sie unter AWS IoT Core Richtlinien.

Trennen einer Richtlinie von einer statischen Objektgruppe

Mit dem Befehl DetachPolicy können Sie eine Richtlinie von einer Gruppe und damit von alle Objekten in dieser Gruppe und in allen ihren untergeordneten Gruppen trennen:

```
$ aws iot detach-policy --target "arn:aws:iot:us-west-2:123456789012:thinggroup/
LightBulbs" --policy-name "myLightBulbPolicy"
```

Der Befehl DetachPolicy erzeugt keine Ausgabe.

Auflisten der an eine statische Objektgruppe angefügten Richtlinien

Mit dem Befehl ListAttachedPolicies können Sie die an eine statische Objektgruppe angefügten Richtlinien auflisten:

```
$ aws iot list-attached-policies --target "arn:aws:iot:us-
west-2:123456789012:thinggroup/RedLights"
```

Der Parameter `--target` kann ein Objektsgruppen-ARN (wie oben), ein Zertifikat-ARN oder eine Amazon Cognito -Identität sein.

Fügen Sie den optionalen Parameter `--recursive` hinzu, um alle den übergeordneten Gruppen der Gruppe angehängten Richtlinien einzuschließen.

Der Befehl ListAttachedPolicies gibt eine Liste von Richtlinien zurück:

```
{
    "policies": [
        "MyLightBulbPolicy"
        ...
    ]
}
```

Auflisten der Gruppen für eine Richtlinie

Mit dem Befehl ListTargetsForPolicy können Sie die Ziele auflisten, einschließlich aller Gruppen, an die eine Richtlinie angehängt ist:

```
$ aws iot list-targets-for-policy --policy-name "MyLightBulbPolicy"
```

Fügen Sie den optionalen Parameter `--page-size` *`number`* hinzu, um die maximale Anzahl der für jede Abfrage anzuzeigenden Ergebnisse anzugeben, sowie den Parameter `--marker` *`string`* auf allen folgenden Aufrufen, um den eventuell vorhandenen nächsten Ergebnissatz abzurufen.

Der Befehl ListTargetsForPolicy gibt eine Liste von Zielen und das Token für den Abruf weiterer Ergebnisse zurück:

```
{
    "nextMarker": "string",
    "targets": [ "string" ... ]
}
```

Abrufen gültiger Richtlinien für ein Objekt

Mit dem Befehl GetEffectivePolicies können Sie die für ein Objekt geltenden Richtlinien auflisten, einschließlich der Richtlinien, die an Gruppen angehängt sind, zu denen das Objekt gehört (unabhängig davon, ob es sich um direkt oder indirekt übergeordnete Gruppen handelt):

```
$ aws iot get-effective-policies \
  --thing-name "MyLightBulb" \
  --principal "arn:aws:iot:us-east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847"
```

Mit dem Parameter `--principal` geben Sie den ARN des an das Objekt angehängten Zertifikats an. Wenn Sie die Amazon Cognito-Identitätsauthentifizierung verwenden, verwenden Sie den Parameter `--cognito-identity-pool-id`, und fügen Sie optional den Parameter `--principal` hinzu, um eine Amazon Cognito-Identität anzugeben. Wenn Sie nur die `--cognito-identity-pool-id` angeben, werden die mit der Rolle des Identitätspools verbundenen Richtlinien für nicht authentifizierte Benutzer zurückgegeben. Wenn Sie beide verwenden, werden die mit der Rolle des Identitätspools verbundenen Richtlinien für authentifizierte Benutzer zurückgegeben.

Der Parameter `--thing-name` ist optional und kann anstelle des Parameters `--principal` verwendet werden. Wenn er verwendet wird, werden die mit einer Gruppe, zu der das Objekt gehört, verbundenen Richtlinien, sowie die Richtlinien, die eventuellen dieser Gruppe übergeordneten Gruppen angefügt sind (bis zur Root-Gruppe in dieser Hierarchie) zurückgegeben.

Der Befehl GetEffectivePolicies gibt eine Liste von Richtlinien zurück:

```
{
    "effectivePolicies": [
        {
            "policyArn": "string",
            "policyDocument": "string",
            "policyName": "string"
        }
        ...
    ]
}
```

Test-Autorisierung für MQTT-Aktionen

Mit dem Befehl TestAuthorization können Sie testen, ob eine MQTT-Aktion (`Publish`, `Subscribe`) für ein Objekt zulässig ist:

```
aws iot test-authorization \
    --principal "arn:aws:iot:us-east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847" \
    --auth-infos "{\"actionType\": \"PUBLISH\", \"resources\": [ \"arn:aws:iot:us-
east-1:123456789012:topic/my/topic\"]}"
```

Mit dem Parameter `--principal` geben Sie den ARN des an das Objekt angehängten Zertifikats an. Wenn Sie die Amazon Cognito-Identitätsauthentifizierung verwenden, geben Sie eine Cognito-

Identität als `--principal` an, oder verwenden Sie den Parameter `--cognito-identity-pool-id` oder beides. (Wenn Sie nur die `--cognito-identity-pool-id` angeben, werden die mit der Rolle des Identitätspools verbundenen Richtlinien für nicht authentifizierte Benutzer berücksichtigt. Wenn Sie beide verwenden, werden die mit der Rolle des Identitätspools verbundenen Richtlinien für authentifizierte Benutzer berücksichtigt.

Geben Sie eine oder mehrere MQTT-Aktionen an, die Sie testen möchten, indem Sie Sätze von Ressourcen und Aktionstypen nach dem Parameter `--auth-infos` auflisten. Das Feld `actionType` sollte „PUBLISH“, „SUBSCRIBE“, „RECEIVE” oder „CONNECT“ enthalten. Das Feld `resources` sollte eine Liste von Ressourcen-ARNs enthalten. Weitere Informationen finden Sie unter AWS IoT Core Richtlinien.

Sie können die Auswirkungen des Hinzufügens von Richtlinien testen, indem Sie sie mit dem Parameter `--policy-names-to-add` angeben. Sie können auch die Auswirkungen des Entfernens von Richtlinien testen, indem Sie den Parameter `--policy-names-to-skip` verwenden.

Mit dem optionalen Parameter `--client-id` können Sie Ihre Ergebnisse weiter verfeinern.

Der Befehl TestAuthorization gibt Details zu Aktionen aus, die für jeden angegebenen Satz von `--auth-infos`-Abfragen zulässig sind oder abgelehnt wurden:

```
{
    "authResults": [
        {
            "allowed": {
                "policies": [
                    {
                        "policyArn": "string",
                        "policyName": "string"
                    }
                ]
            },
            "authDecision": "string",
            "authInfo": {
                "actionType": "string",
                "resources": [ "string" ]
            },
            "denied": {
                "explicitDeny": {
                    "policies": [
                        {
```

```
                        "policyArn": "string",
                        "policyName": "string"
                    }
                ]
            },
            "implicitDeny": {
                "policies": [
                    {
                        "policyArn": "string",
                        "policyName": "string"
                    }
                ]
            }
        },
        "missingContextValues": [ "string" ]
    }
  ]
}
```

Dynamische Objektgruppen

Dynamische Dinggruppen werden aus bestimmten Suchabfragen in der Registrierung erstellt. Suchabfrageparameter wie Gerätekonnektivität, Erstellung von Geräteschatten und Daten zu AWS IoT Device Defender Verstößen unterstützen dies. Für dynamische Dinggruppen muss die Flottenindizierung aktiviert sein, um die Daten Ihrer Geräte zu indizieren, zu durchsuchen und zu aggregieren. Sie können eine Vorschau der Dinge in einer dynamischen Dinggruppe mithilfe einer Suchabfrage zur Flottenindizierung anzeigen, bevor Sie sie erstellen. Weitere Informationen erhalten Sie unter -Flottenindizierung und Abfragesyntax.

> **Note**
>
> Dynamische Objektgruppenvorgänge werden im Rahmen von Registrierungsvorgängen gemessen. Weitere Informationen finden Sie unter AWS IoT Core Zusätzliche Details zur Messung.

Dynamische Objektgruppen unterscheiden sich von statischen Objektgruppen in folgender Hinsicht:

- Die Objektmitgliedschaft ist nicht explizit definiert. Um eine dynamische Dinggruppe zu erstellen, definieren Sie eine Suchabfragezeichenfolge, um die Gruppenmitgliedschaft zu ermitteln.

- Dynamische Objektgruppen können nicht Teil einer Hierarchie sein.
- Auf dynamische Objektgruppen können keine Richtlinien angewendet werden.
- Sie verwenden eine andere Gruppe von Befehlen zum Erstellen, Aktualisieren und Löschen von dynamischen Objektgruppen. Für alle anderen Operationen verwenden Sie dieselben Befehle für beide Typen von Dinggruppen.
- Die Anzahl der dynamischen Gruppen pro AWS-Konto ist begrenzt.

- Verwende keine persönlich identifizierbaren Informationen in deinem Dinggruppennamen. Der Name der Objektgruppe kann in unverschlüsselten Mitteilungen und Berichten vorkommen.

Weitere Informationen über statische Objektgruppen finden Sie unter Statische Objektgruppen.

Anwendungsfälle dynamischer Dinggruppen

Sie können dynamische Dinggruppen für die folgenden Anwendungsfälle verwenden:

Geben Sie eine dynamische Dinggruppe als Ziel für einen Job an

Wenn Sie einen kontinuierlichen Job mit einer dynamischen Dinggruppe als Ziel erstellen, können Sie Geräte automatisch als Ziel auswählen, wenn sie die gewünschten Kriterien erfüllen. Bei den Kriterien kann es sich um den Konnektivitätsstatus oder um Kriterien handeln, die in der Registrierung oder im Shadow gespeichert sind, wie z. B. Softwareversion oder Modell. Wenn ein Objekt nicht in der dynamischen Dinggruppe erscheint, erhält es das Jobdokument aus dem Job nicht.

Zum Beispiel, wenn Ihre Geräteflotte ein Firmware-Update benötigt, um das Risiko einer Unterbrechung während des Aktualisierungsvorgangs zu minimieren, und Sie die Firmware nur auf Geräten mit einer Akkulaufzeit von mehr als 80% aktualisieren möchten. Sie können eine dynamische Dinggruppe namens 80 erstellenPercentBatteryLife , die nur Geräte mit einer Akkulaufzeit von über 80% enthält, und sie als Ziel für Ihre Arbeit verwenden. Nur Geräte, die Ihre Kriterien für die Akkulaufzeit erfüllen, erhalten das Firmware-Update. Wenn Geräte die Kriterien für eine Akkulaufzeit von 80% erreichen, werden sie automatisch zur dynamischen Dinggruppe hinzugefügt und erhalten das Firmware-Update.

Möglicherweise haben Sie auch mehrere Gerätemodelle mit unterschiedlicher Firmware oder Betriebssystem, sodass unterschiedliche Versionen neuer Softwareupdates erforderlich sind. Dies ist der häufigste Anwendungsfall für dynamische Gruppen mit fortlaufenden Aufträgen, bei

denen Sie für jede Kombination aus Gerätemodell, Firmware und Betriebssystem eine dynamische Gruppe erstellen können. Anschließend können Sie fortlaufende Jobs für jede dieser dynamischen Gruppen einrichten, um Softwareupdates zu übertragen, da Geräte anhand der definierten Kriterien automatisch Mitglieder dieser Gruppen werden.

Weitere Informationen zum Angeben von Objektgruppen als Auftragszielen finden Sie unter CreateJob.

Verwenden Sie dynamische Änderungen der Gruppenmitgliedschaft, um die gewünschten Aktionen auszuführen

Jedes Mal, wenn ein Gerät zu einer dynamischen Dinggruppe hinzugefügt oder daraus entfernt wird, wird im Rahmen von https://docs.aws.amazon.com/iot/latest/developerguide/registry-events.html Registrierungsereignisaktualisierungen eine Benachrichtigung an ein MQTT-Thema gesendet. Sie können AWS IoT Core Regeln für die Interaktion mit AWS Diensten auf der Grundlage der dynamischen Aktualisierungen der Gruppenmitgliedschaft konfigurieren und die gewünschten Aktionen ausführen. Zu den Beispielaktionen gehören das Schreiben an Amazon SNS Amazon DynamoDB, das Aufrufen einer Lambda-Funktion oder das Senden einer Benachrichtigung an Amazon SNS.

Fügen Sie Geräte zur automatischen Erkennung von Verstößen zu einer dynamischen Dinggruppe hinzu

AWS IoT Device Defender Kunden von Detect können ein Sicherheitsprofil für eine dynamische Dinggruppe definieren. Geräte der dynamischen Dinggruppe werden automatisch auf Verstöße gegen das für die Gruppe definierte Sicherheitsprofil hin erkannt.

Legen Sie Protokollebenen für dynamische Dinggruppen fest, um Geräte mit einer detaillierten Protokollierung zu beobachten

Sie können eine Protokollebene für eine dynamische Dinggruppe angeben. Dies ist nützlich, wenn Sie den Protokollierungsgrad und die Details nur für Geräte anpassen möchten, die bestimmte Kriterien erfüllen. Wenn Sie beispielsweise vermuten, dass Geräte mit einer bestimmten Firmware-Version Fehler im Zusammenhang mit dem veröffentlichten Thema einer bestimmten Regel verursachen, sollten Sie möglicherweise eine detaillierte Protokollierung einrichten, um diese Probleme zu beheben. In diesem Fall können Sie eine dynamische Gruppe für alle Geräte erstellen, die über diese Firmware-Version verfügen. Wir gehen davon aus, dass sie als Registrierungsattribut oder in einem Geräteschatten gespeichert ist. Sie können dann eine Debug-Stufe festlegen,

wobei das Logging-Ziel als diese dynamische Dinggruppe definiert ist. Weitere Informationen zur detaillierten Protokollierung finden Sie unter Überwachen AWS IoT mithilfe von CloudWatch Protokollen. Weitere Informationen zum Angeben einer Protokollierungsstufe für eine bestimmte Dinggruppe finden Sie unter Konfigurieren der ressourcenspezifischen Anmeldung. AWS IoT

Erstellen einer dynamischen Objektgruppe

Mit dem Befehl CreateDynamicThingGroup können Sie eine dynamische Objektgruppe erstellen. Verwenden Sie den CLI-Befehl, um eine dynamische Dinggruppe für das PercentBatteryLife 80-Szenario zu erstellen: create-dynamic-thing-group

```
$ aws iot create-dynamic-thing-group --thing-group-name "80PercentBatteryLife" --query-
string "attributes.batterylife80"
```

> **Note**
>
> Verwenden Sie in Ihren dynamischen Dinggruppennamen keine personenbezogenen Daten.

Der CreateDynamicThingGroup Befehl gibt eine Antwort zurück. Die Antwort enthält den Indexnamen, die Abfragezeichenfolge, die Abfrageversion, den Dinggruppennamen, die Dinggruppen-ID und den Amazon-Ressourcennamen (ARN) Ihrer Dinggruppe:

```
{
    "indexName": "AWS_Things",
    "queryVersion": "2017-09-30",
    "thingGroupName": "80PercentBatteryLife",
    "thingGroupArn": "arn:aws:iot:us-
west-2:123456789012:thinggroup/80PercentBatteryLife",
    "queryString": "attributes.batterylife80\n",
    "thingGroupId": "abcdefgh12345678ijklmnop12345678qrstuvwx"
}
```

Die Erstellung dynamischer Dinggruppen erfolgt nicht gleichzeitig. Der Backfillvorgang für die dynamische Objektgruppe nimmt einige Zeit in Anspruch. Wenn Sie eine dynamische Dinggruppe erstellen, wird der Status der Gruppe auf gesetzt`BUILDING`. Wenn der Backfillvorgang abgeschlossen ist, wechselt der Status zu `ACTIVE`. Verwenden Sie den DescribeThingGroup Befehl, um den Status Ihrer dynamischen Dinggruppe zu überprüfen.

Beschreiben einer dynamischen Objektgruppe

Mit dem Befehl DescribeThingGroup können Sie Informationen zu einer dynamischen Objektgruppe abrufen:

```
$ aws iot describe-thing-group --thing-group-name "80PercentBatteryLife"
```

Der Befehl DescribeThingGroup gibt Informationen zur angegebenen Gruppe zurück:

```
{
    "status": "ACTIVE",
    "indexName": "AWS_Things",
    "thingGroupName": "80PercentBatteryLife",
    "thingGroupArn": "arn:aws:iot:us-
west-2:123456789012:thinggroup/80PercentBatteryLife",
    "queryString": "attributes.batterylife80\n",
    "version": 1,
    "thingGroupMetadata": {
        "creationDate": 1548716921.289
    },
    "thingGroupProperties": {},
    "queryVersion": "2017-09-30",
    "thingGroupId": "84dd9b5b-2b98-4c65-84e4-be0e1ecf4fd8"
}
```

Wenn DescribeThingGroup Sie auf einer dynamischen Dinggruppe laufen, werden Attribute zurückgegeben, die für die dynamischen Dinggruppen spezifisch sind. Beispiele für Rückgabeattribute sind der QueryString und der Status.

Der Status einer dynamischen Objektgruppe kann die folgenden Werte haben:

`ACTIVE`

Die dynamische Objektgruppe ist einsatzbereit.

`BUILDING`

Die dynamische Objektgruppe wird erstellt, und die Mitgliedschaft wird bearbeitet.

`REBUILDING`

Die Mitgliedschaft der dynamischen Objektgruppe wird aktualisiert, nach der Anpassung Suchabfrage der Gruppe.

> **Note**
>
> Nachdem Sie eine dynamische Dinggruppe erstellt haben, verwenden Sie sie unabhängig von ihrem Status. Nur dynamische Objektgruppen mit dem Status `ACTIVE` enthalten alle Objekte, die der Suchabfrage für diese dynamische Objektgruppe entsprechen. Dynamische Objektgruppen mit den Status `BUILDING` und `REBUILDING` enthalten möglicherweise nicht alle Objekte, die der Suchabfrage entsprechen.

Aktualisieren einer dynamischen Objektgruppe

Verwenden Sie den Befehl UpdateDynamicThingGroup, um die Attribute einer dynamischen Objektgruppe zu aktualisieren, einschließlich der Suchabfrage der Gruppe: Mit dem folgenden Befehl werden zwei Attribute aktualisiert. Eines ist die Beschreibung der Dinggruppe und das andere ist die Abfragezeichenfolge, die die Mitgliedschaftskriterien auf eine Akkulaufzeit > 85 ändert:

```
$ aws iot update-dynamic-thing-group --thing-group-name "80PercentBatteryLife" --thing-
group-properties "thingGroupDescription=\"This thing group contains devices with a
 battery life greater than 85 percent.\"" --query-string "attributes.batterylife85"
```

Der UpdateDynamicThingGroup Befehl gibt eine Antwort zurück, die die Versionsnummer der Gruppe nach der Aktualisierung enthält:

```
{
    "version": 2
}
```

Die Aktualisierung einer dynamischen Dinggruppe erfolgt nicht sofort. Der Backfillvorgang für die dynamische Objektgruppe nimmt einige Zeit in Anspruch. Wenn Sie eine dynamische Dinggruppe aktualisieren, ändert sich der Status der Gruppe zu, `REBUILDING` während die Gruppe ihre Mitgliedschaft aktualisiert. Wenn der Backfillvorgang abgeschlossen ist, wechselt der Status zu `ACTIVE`. Verwenden Sie den DescribeThingGroup Befehl, um den Status Ihrer dynamischen Dinggruppe zu überprüfen.

Löschen einer dynamischen Objektgruppe

Mit dem Befehl DeleteDynamicThingGroup können Sie eine dynamische Objektgruppe löschen:

```
$ aws iot delete-dynamic-thing-group --thing-group-name "80PercentBatteryLife"
```

Der Befehl DeleteDynamicThingGroup erzeugt keine Ausgabe.

Befehle, die zeigen, zu welchen Gruppen ein Objekt gehört (z. B. ListGroupsForThing) zeigen möglicherweise weiterhin die Gruppe an, während Datensätze in der Cloud aktualisiert werden.

Einschränkungen bei dynamischen und statischen Dinggruppen

Dynamische und statische Dinggruppen haben die folgenden Einschränkungen gemeinsam:

- Die Anzahl der Attribute, die eine Dinggruppe haben kann, ist begrenzt.
- Die Anzahl der Gruppen, zu denen ein Objekt gehören kann, ist begrenzt.
- Sie können Dinggruppen nicht umbenennen.
- Namen der Objektgruppen dürfen keine internationale Zeichen enthalten, z. B. û, é und ñ.

Einschränkungen bei dynamischen Dinggruppen

Dynamische Dinggruppen haben die folgenden Einschränkungen:

-Flottenindizierung

Wenn der Flottenindexierungsdienst aktiviert ist, können Sie Suchabfragen auf Ihrer Geräteflotte durchführen. Sie können dynamische Objektgruppen erstellen und verwalten, nachdem das Backfill für die Flottenindizierung abgeschlossen ist. Die Abschlusszeit für den Backfill-Vorgang hängt direkt von der Größe Ihrer Geräteflotte ab, die bei der registriert ist. AWS Cloud Nachdem Sie den Flottenindizierungsservice für dynamische Objektgruppen aktiviert haben, können Sie ihn erst wieder deaktivieren, wenn Sie alle Ihre dynamischen Objektgruppen gelöscht haben.

> **Note**
>
> Wenn Sie über die Berechtigungen verfügen, den Flottenindex abzufragen, können Sie auf Daten zu Objekten über die gesamte Flotte hinweg zugreifen.

Die Anzahl der dynamischen Objektgruppen ist begrenzt

Die Anzahl der dynamischen Dinggruppen ist begrenzt.

Erfolgreiche Befehle können Fehler protokollieren

Wenn Sie eine dynamische Dinggruppe erstellen oder aktualisieren, ist es möglich, dass einige Dinge in eine dynamische Dinggruppe aufgenommen werden können, dieser aber nicht hinzugefügt werden. Dieses Szenario führt dazu, dass der Befehl zum Erstellen oder Aktualisieren erfolgreich ausgeführt wird, während ein Fehler protokolliert und eine Metrik generiert wird. `AddThingToDynamicThingGroupsFailed` Eine einzelne Metrik kann mehrere Logeinträge repräsentieren.

Ein Fehlerprotokolleintrag im CloudWatch Protokoll wird erstellt, wenn Folgendes eintritt:

- Eine geeignete Sache kann nicht zu einer dynamischen Dinggruppe hinzugefügt werden.
- Ein Ding wird aus einer dynamischen Dinggruppe entfernt, um es einer anderen Gruppe hinzuzufügen.

Wenn ein Objekt dazu in Frage kommt, einer dynamischen Dinggruppe hinzugefügt zu werden, sollten Sie Folgendes beachten:

- Ist das Objekt schon in der maximalen Anzahl von Gruppen enthalten? (Weitere Informationen finden Sie unter Limits).
  - NEIN: Das Objekt wird der dynamischen Objektgruppe hinzugefügt.
  - JA: Gehört das Objekt dynamischen Objektgruppen an?
    - NEIN: Das Objekt kann nicht zu der dynamischen Objektgruppe hinzugefügt werden, ein Fehler wird protokolliert und eine `AddThingToDynamicThingGroupsFailed` Metrik wird generiert.
    - JA: Ist die dynamische Objektgruppe, in die das Objekt aufgenommen werden soll, älter als jede dynamische Objektgruppe, der das Objekt bereits angehört?
      - NEIN: Das Objekt kann nicht zu der dynamischen Objektgruppe hinzugefügt werden, ein Fehler wird protokolliert und eine `AddThingToDynamicThingGroupsFailed` Metrik wird generiert.
      - JA: Entferne das Ding aus der neuesten dynamischen Dinggruppe, protokolliere einen Fehler und füge das Ding der dynamischen Dinggruppe hinzu. Dies erzeugt einen Fehler

und es wird eine AddThingToDynamicThingGroupsFailed Metrik für die dynamische Objektgruppe generiert, aus der das Objekt entfernt wurde.

Wenn ein Ding in einer dynamischen Dinggruppe die Suchabfrage nicht mehr erfüllt, wird das Ding aus der dynamischen Dinggruppe entfernt. Ebenso wird ein Ding, wenn es aktualisiert wird, um die Suchabfrage einer dynamischen Dinggruppe zu erfüllen, der Gruppe hinzugefügt, wie zuvor beschrieben. Diese Hinzufügungen und Entfernungen sind normal und verursachen keine Fehlerprotokolleinträge.

DynamicGroups**Wenn** Override aktiviert ist, haben statische Gruppen Vorrang vor dynamischen Gruppen

Die Anzahl der Gruppen, zu denen ein Objekt gehören kann, ist begrenzt. Wenn Sie die UpdateThingGroupsForThing Befehle AddThingToThingGroup oder verwenden, um die Dingmitgliedschaft zu aktualisieren, gibt das Hinzufügen des `--overrideDynamicGroups` Parameters statischen Dinggruppen Vorrang vor dynamischen Dinggruppen.

Wenn Sie einer statischen Dinggruppe ein Ding hinzufügen, beachten Sie Folgendes:

- Gehört das Objekt bereits der maximalen Anzahl von Gruppen an?
  - NEIN: Das Objekt wird der statischen Objektgruppe hinzugefügt.
  - JA: Ist das Objekt in dynamischen Gruppen enthalten?
    - NEIN: Das Objekt kann nicht zu der Objektgruppe hinzugefügt werden. Der Befehl löst eine Ausnahme aus.
    - JA: Wurde --overrideDynamicGroups aktiviert?
      - NEIN: Das Objekt kann nicht zu der Objektgruppe hinzugefügt werden. Der Befehl löst eine Ausnahme aus.
      - JA: Das Objekt wird aus der zuletzt erstellten dynamischen Objektgruppe entfernt, ein Fehler wird protokolliert und es wird eine AddThingToDynamicThingGroupsFailed Metrik für die dynamische Objektgruppe generiert, aus der das Objekt entfernt wurde. Anschließend wird das Objekt der statischen Objektgruppe hinzugefügt.

Ältere dynamische Objektgruppen haben Vorrang vor neueren.

Die Anzahl der Gruppen, zu denen ein Objekt gehören kann, ist begrenzt. Wenn ein Erstellungs- oder Aktualisierungsvorgang zusätzliche Gruppenberechtigungen für ein Ding schafft und das

Ding sein Gruppenlimit erreicht hat, kann es aus einer anderen dynamischen Dinggruppe entfernt werden, um dieses Hinzufügen zu ermöglichen. Wenn Sie mehr Informationen wünschen, wie dies geschieht, beachten Sie die Beispiele unter Erfolgreiche Befehle können Fehler protokollieren und DynamicGroupsWenn Override aktiviert ist, haben statische Gruppen Vorrang vor dynamischen Gruppen.

Wenn ein Ding aus einer dynamischen Dinggruppe entfernt wird, wird ein Fehler protokolliert und ein Ereignis ausgelöst.

Sie können keine Richtlinien auf dynamische Objektgruppen anwenden.

Wenn Sie versuchen, eine Richtlinie auf eine dynamische Objektgruppe anzuwenden, wird eine Ausnahme generiert.

Die Mitgliedschaft in dynamischen Objektgruppen ist letztlich konsistent.

Nur der letzte Status eines Geräts wird für die Registrierung evaluiert. Zwischenzustände können bei schneller Aktualisierung der Zustände übersprungen werden. Vermeiden Sie es, eine Regel oder einen Job einer dynamischen Dinggruppe zuzuordnen, deren Mitgliedschaft von einem zwischengeschalteten Staat abhängt.

Zuordnen einer AWS IoT Ding zu einer MQTT-Client-Verbindung

Eine exklusive Dingzuordnung liegt vor, wenn Sie ein X.509 Zertifikat an eine einzelne AWS IoT Sache anhängen. In diesem Fall kann das Zertifikat nicht zusammen mit anderen Dingen verwendet werden. Indem sichergestellt wird, dass ein Zertifikat nur von einem einzigen IoT-Gerät verwendet wird, werden Sicherheitslücken vermieden.

In AWS IoT ist die Client-ID eine eindeutige Kennung für ein Ding oder ein Gerät, wenn es eine Verbindung zum AWS IoT Core MQTT-Broker herstellt. Wenn Sie eine nicht exklusive Verknüpfung verwenden, können mehrere Dinge an dasselbe Zertifikat angehängt werden. Wenn eine nicht ausschließliche Zuordnung besteht, müssen Sie Ihre Client-ID mit dem Namen des Dings abgleichen, um eine klare Zuordnung aufrechtzuerhalten und potenzielle Konflikte zu vermeiden.

In diesem Thema

- Anwendungsfälle
- Wie verbinde ich eine Sache mit einer Verbindung

Anwendungsfälle

Das Zuordnen eines Dings zu einer Verbindung bietet die folgenden Funktionen.

> **Note**
>
> Beachten Sie, dass Sie, wenn Ihr IoT-Ding und Ihre Client-Verbindung eine nicht ausschließliche Verknüpfung haben, alle folgenden Funktionen mit Ausnahme der Funktion für Lebenszyklusereignisse verwenden können. Um Ihren Ding-Namen in die Lebenszyklus-Ereignismeldungen aufzunehmen, müssen Ihr IoT-Ding und Ihre Client-Verbindung eine exklusive Verknüpfung haben.

Ding-Richtlinienvariablen — Sie können Ding-Richtlinienvariablen verwenden, um den Gerätezugriff auf AWS IoT API-Operationen zu autorisieren. Mit diesen Variablen können Sie AWS IoT Core Richtlinien schreiben, die Berechtigungen auf der Grundlage von Dingeigenschaften wie Namen, Typen und Attributwerten gewähren oder verweigern. Mithilfe von Dingrichtlinienvariablen können Sie dieselbe Richtlinie anwenden, um mehrere AWS IoT Core Geräte zu steuern. Auf diese Weise können Sie die Richtlinienverwaltung vereinfachen und die doppelte Nutzung von Ressourcen reduzieren. Weitere Informationen finden Sie unter Ding-Richtlinienvariablen.

Lebenszyklusereignisse — Sie können den Namen des Dings in Lebenszyklusereignissen erhalten (z. B. Verbinden, Trennen und Abonnieren und Abmelden). Dies ermöglicht die Verarbeitung des in den Nachrichten enthaltenen Dingnamens, z. B. in Regeln. Weitere Informationen finden Sie unter Lifecycle-Ereignisse.

Resource-specific Protokollierung — Sie können die ressourcenspezifische Protokollierung für Dinggruppen konfigurieren und auf einfache Weise die gewünschte Protokollierungskonfiguration für alle Dinge innerhalb der definierten Dinggruppe anwenden. Weitere Informationen finden Sie unter ???.

Kostenzuweisung — Sie können Abrechnungsgruppen mit benutzerdefinierten Stichwörtern für die Kostenzuweisung erstellen und die Dinge zu diesen Gruppen hinzufügen. Weitere Informationen finden Sie unter Abrechnungsgruppen.

Wie verbinde ich eine Sache mit einer Verbindung

Wenn Ihre Client-ID mit dem Namen Ihres Dings in der Registrierung übereinstimmt, AWS IoT Core wird die Client-Verbindung dem Ding zugeordnet, nachdem Sie ein X.509 Zertifikat an dieses IoT-

Ding angehängt haben. Wenn Ihre Client-ID nicht mit dem Namen des Dings in der Registrierung übereinstimmt, können Sie dem Ding ausschließlich ein X.509 Zertifikat beifügen, um diese Zuordnung herzustellen. Das Ding, das diesen exklusiven Anhang hat, wird als exklusives Ding bezeichnet. Andernfalls wird es als nicht ausschließliches Ding bezeichnet. Wenn ein Zertifikat mit einer exklusiven Sache verknüpft ist, kann dieses Zertifikat nur dann mit anderen Dingen verknüpft werden, wenn Sie es von der exklusiven Sache trennen. Wählen Sie in diesem Abschnitt entweder AWS-Managementkonsole oder aus, AWS CLI um einer Verbindung ein Ding zuzuordnen.

AWS-Managementkonsole

Um ein Zertifikat an ein Objekt anzuhängen, verwenden Sie ausschließlich die AWS-Managementkonsole.

1. Öffnen Sie die AWS IoT Startseite in der AWS IoT Konsole. Wählen Sie im linken Navigationsbereich unter Sicherheit die Option Zertifikate aus.
2. Wählen Sie auf der Seite „Zertifikate“ ein Zertifikat aus, an das Sie etwas anhängen möchten. Wählen Sie dann in der oberen rechten Ecke der Seite unter Aktionen die Option An Dinge anhängen aus.

   Wählen Sie alternativ ein Zertifikat aus und navigieren Sie zur Seite mit den Zertifikatsdetails. Wählen Sie den Tab „Dinge“ und dann „An Dinge anhängen“.
3. Markieren Sie auf der Seite Zertifikat an Sache (en) anhängen das Kontrollkästchen Ding mit Verbindung verknüpfen. Wählen Sie dann aus der Dropdownliste Dinge ein Objekt aus, an das dieses Zertifikat angehängt werden soll.
4. Wählen Sie Ding (e) anhängen aus. Wenn die Aktion erfolgreich ist, wird ein Banner mit der Aufschrift „Ihrem Zertifikat wurde erfolgreich eine Sache angehängt“ angezeigt, und das Ding wird dem Tab „Dinge“ hinzugefügt.

Um ein Zertifikat von einem exklusiven Objekt zu trennen, verwenden Sie den AWS-Managementkonsole

1. Öffnen Sie die AWS IoT Startseite in der AWS IoT Konsole. Wählen Sie im linken Navigationsbereich unter Sicherheit die Option Zertifikate aus.
2. Wählen Sie auf der Seite Zertifikate ein Zertifikat aus und navigieren Sie zur Seite mit den Zertifikatsdetails.

3. Wählen Sie auf der Seite mit den Zertifikatsdetails die Registerkarte Dinge aus. Wählen Sie dann eine Sache aus, von der Sie das Zertifikat trennen möchten. Wählen Sie Dinge trennen aus.
4. Bestätigen Sie im Fenster Dinge trennen Ihre Aktion. Wählen Sie Trennen aus. Wenn die Aktion erfolgreich ist, wird ein Banner mit der Aufschrift „Eine Sache wurde erfolgreich von Ihrem Zertifikat getrennt“ angezeigt, und das Ding wird nicht mehr auf der Registerkarte „Dinge“ angezeigt.

AWS CLI

1. Um ein Zertifikat mit einem Objekt anzuhängen AWS CLI, führen Sie den Befehl attach-thing-principal aus. Um den exklusiven Anhang des Zertifikats an das Ding anzugeben, müssen Sie dies `EXCLUSIVE_THING` in dem Feld angeben. `--thing-principal-type` Ein Beispielbefehl kann der folgende sein.

```
aws iot attach-thing-principal \
    --thing-name "thing_1" \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8" \
    --thing-principal-type "EXCLUSIVE_THING"
```

   Dieser Befehl liefert keine Ausgabe. Weitere Informationen finden Sie unter ???.

2. Führen Sie den `list-principal-things-v2` Befehl aus, um die mit dem angegebenen Zertifikat verknüpften Elemente zusammen mit dem Anhangstyp aufzulisten. Der Anhangstyp bezieht sich darauf, wie das Zertifikat an das Ding angehängt wird. Ein Beispielbefehl kann der folgende sein.

```
$ aws iot list-principal-things-v2 \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
```

   Die Ausgabe kann wie folgt aussehen.

```
{
    "PrincipalThingObjects": [
        {
```

```
            "thingPrincipalType": "EXCLUSIVE_THING",
            "thing": "arn:aws:iot:us-east-1:123456789012:thing/thing_1"
        }
    ]
}
```

Weitere Informationen finden Sie unter ???.

3. Führen Sie den `list-thing-principals-v2` Befehl aus, um die Prinzipale, die dem angegebenen Objekt zugeordnet sind, zusammen mit dem Anhangstyp aufzulisten. Der Anhangstyp bezieht sich darauf, wie das Zertifikat an das Ding angehängt wird. Ein Beispielbefehl kann der folgende sein.

```
$ aws iot list-thing-principals-v2 \
    --thing-name "thing_1"
```

Die Ausgabe kann wie folgt aussehen.

```
{
    "ThingPrincipalObjects": [
        {
            "thingPrincipalType": "EXCLUSIVE_THING",
            "principal": "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8"
        }
    ]
}
```

Weitere Informationen finden Sie unter ???.

4. Um ein Zertifikat von einem Ding zu trennen, führen Sie den Befehl https://docs.aws.amazon.com/cli/latest/reference/iot/detach-thing-principal.html detach-thing-principal aus.

```
aws iot detach-thing-principal \
    --principal "arn:aws:iot:us-
east-1:123456789012:cert/
2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8" \
    --thing-name "thing_1"
```

Dieser Befehl liefert keine Ausgabe. Weitere Informationen finden Sie unter ???.

Hinzufügen von Propagierungsattributen für die Nachrichtenanreicherung

In können Sie MQTT-Nachrichten von Geräten anreichern AWS IoT Core, indem Sie Propagierungsattribute hinzufügen, bei denen es sich um kontextbezogene Metadaten aus Dingattributen oder Verbindungsdetails handelt. Dieser als Nachrichtenanreicherung bekannte Prozess kann in verschiedenen Szenarien hilfreich sein. Beispielsweise können Sie Nachrichten für jeden eingehenden Veröffentlichungsvorgang anreichern, ohne dass Sie dabei geräteseitige Änderungen vornehmen oder Regeln verwenden müssen. Durch die Nutzung von Propagierungsattributen können Sie Ihre IoT-Daten effizienter und kostengünstiger anreichern, ohne dass Sie Regeln konfigurieren oder Konfigurationen erneut veröffentlichen müssen.

Die Funktion zur Nachrichtenanreicherung steht AWS IoT Core Kunden zur Verfügung, die Basic Ingest und Message Broker verwenden. https://docs.aws.amazon.com/iot/latest/developerguide/mqtt.html Es ist wichtig zu beachten, dass Veröffentlichungsgeräte zwar jede MQTT-Version verwenden können, Abonnenten (Anwendungen oder Dienste, die Nachrichten konsumieren) jedoch MQTT 5 unterstützen müssen, um die angereicherten Nachrichten mit Propagierungsattributen zu empfangen. Die angereicherten Nachrichten werden jeder Nachricht, die von Geräten veröffentlicht wird, als MQTT 5-Benutzereigenschaften hinzugefügt. Wenn Sie Regeln verwenden, können Sie die Funktion get_user_properties nutzen, um die angereicherten Daten für die Nachrichtenweiterleitung oder -verarbeitung auf der Grundlage der Daten abzurufen.

In können Sie Propagierungsattribute hinzufügen AWS IoT Core, wenn Sie einen Dingtyp erstellen oder aktualisieren, indem Sie die oder verwenden. AWS-Managementkonsole AWS CLI

> **Important**
>
> Beim Hinzufügen von Propagierungsattributen müssen Sie sicherstellen, dass der Client, der die Nachricht veröffentlicht, mit einem Zertifikat authentifiziert wurde. Weitere Informationen finden Sie unter Client-Authentifizierung.

> **Note**
>
> Wenn Sie versuchen, diese Funktion mit dem MQTT-Testclient in der Konsole zu testen, funktioniert sie möglicherweise nicht, da für diese Funktion MQTT-Clients erforderlich sind, die mit einem zugehörigen Zertifikat authentifiziert sind.

AWS-Managementkonsole

Um Propagierungsattribute für die Nachrichtenanreicherung hinzuzufügen, verwenden Sie den AWS-Managementkonsole

1. Öffnen Sie die AWS IoT Startseite in der AWS IoT Konsole. Wählen Sie im linken Navigationsbereich unter Verwalten die Option Alle Geräte aus. Wählen Sie dann Dingtypen aus.
2. Wählen Sie auf der Seite „Dingtypen“ die Option „Dingtyp erstellen“ aus.

   Um die Nachrichtenanreicherung durch Aktualisierung eines Dingtyps zu konfigurieren, wählen Sie einen Dingtyp aus. Wählen Sie dann auf der Seite mit den Dingtyp-Details die Option Aktualisieren aus.
3. Wählen Sie auf der Seite „Dingtyp erstellen“ die Informationen zum Dingtyp in den Eigenschaften des Dingtyps aus, oder geben Sie sie ein.

   Wenn Sie einen Dingtyp aktualisieren möchten, werden die Eigenschaften des Dingtyps angezeigt, nachdem Sie im vorherigen Schritt auf Aktualisieren geklickt haben.
4. Erweitern Sie unter Zusätzliche Konfiguration die Option Attribute weitergeben. Wählen Sie dann Dingattribut und geben Sie das Dingattribut ein, das Sie in die veröffentlichten MQTT5-Nachrichten eintragen möchten. Mithilfe der Konsole können Sie bis zu drei Dingattribute hinzufügen.

   Wählen Sie im Abschnitt Attribute weitergeben die Option Verbindungsattribut aus und geben Sie den Attributtyp und optional den Attributnamen ein.
5. Fügen Sie optional Tags hinzu. Wählen Sie dann Dingtyp erstellen.

   Wenn Sie einen Dingtyp aktualisieren möchten, wählen Sie Dingtyp aktualisieren.

AWS CLI

1. Um Propagierungsattribute für die Nachrichtenanreicherung hinzuzufügen, indem Sie mithilfe von einen neuen Dingtyp erstellen AWS CLI, führen Sie den Befehl create-thing-type aus. Ein Beispielbefehl kann der folgende sein.

```
aws iot create-thing-type \
    --thing-type-name "LightBulb" \
```

```
    --thing-type-properties "{\"mqtt5Configuration\":{\"propagatingAttributes\":
[{\"userPropertyKey\":\"iot:ClientId\", \"connectionAttribute\":\"iot:ClientId\"},
 {\"userPropertyKey\":\"test\", \"thingAttribute\":\"A\"}]}}" \
```

Die Ausgabe des Befehls kann wie folgt aussehen.

```
{
 "thingTypeName": "LightBulb",
 "thingTypeArn": "arn:aws:iot:us-west-2:123456789012:thingtype/LightBulb",
 "thingTypeId": "ce3573b0-0a3c-45a7-ac93-4e0ce14cd190"
}
```

2. Führen Sie den Befehl update-thing-type aus, um die Nachrichtenanreicherung zu konfigurieren AWS CLI, indem Sie einen Dingtyp mit aktualisieren . Beachten Sie, dass Sie nur aktualisieren können, `mqtt5Configuration` wenn Sie diesen Befehl ausführen. Ein Beispielbefehl kann der folgende sein.

```
aws iot update-thing-type \
    --thing-type-name "MyThingType" \
    --thing-type-properties "{\"mqtt5Configuration\":{\"propagatingAttributes\":
[{\"userPropertyKey\":\"iot:ClientId\", \"connectionAttribute\":\"iot:ClientId\"},
 {\"userPropertyKey\":\"test\", \"thingAttribute\":\"A\"}]}}" \
```

Dieser Befehl liefert keine Ausgabe.

3. Führen Sie den `describe-thing-type` Befehl aus, um einen Dingtyp zu beschreiben. Dieser Befehl erzeugt eine Ausgabe mit Konfigurationsinformationen zur Nachrichtenanreicherung im `thing-type-properties` Feld. Ein Beispielbefehl kann der folgende sein.

```
aws iot describe-thing-type \
    --thing-type-name "LightBulb"
```

Die Ausgabe kann wie folgt aussehen.

```
{
 "thingTypeName": "LightBulb",
 "thingTypeId": "bdf72512-0116-4392-8d79-bf39b17ef73d",
 "thingTypeArn": "arn:aws:iot:us-east-1:123456789012:thingtype/LightBulb",
 "thingTypeProperties": {
  "mqtt5Configuration": {
   "propagatingAttributes": [
```

```
    {
     "userPropertyKey": "iot:ClientId",
     "connectionAttribute": "iot:ClientId"
    },
    {
     "userPropertyKey": "test",
     "thingAttribute": "attribute"
    }
   ]
  }
 },
 "thingTypeMetadata": {
  "deprecated": false,
  "creationDate": "2024-10-18T17:37:46.656000+00:00"
 }
}
```

Weitere Informationen finden Sie unter ???.

Taggen Sie Ihre AWS IoT Ressourcen

Zur einfacheren Verwaltung und Organisation Ihrer Objektgruppen, Objekttypen, Themenregeln, Aufträge, geplanten Audits und Sicherheitsprofile können Sie diesen Ressourcen optional Ihre eigenen Metadaten in Form von Tags zuweisen. In diesem Abschnitt werden Tags und deren Erstellung beschrieben.

Zur besseren Verwaltung der mit Objekten verbundenen Kosten können Sie Abrechnungsgruppen erstellen, die Objekte enthalten. Anschließend können Sie allen diesen Abrechnungsgruppen Tags mit Ihren Metadaten zuweisen. Dieser Abschnitt erläutert auch die Abrechnungsgruppen und die Befehle für ihre Erstellung und Verwaltung.

Grundlagen zu Tags (Markierungen)

Sie können Stichwörter verwenden, um Ihre AWS IoT Ressourcen auf unterschiedliche Weise zu kategorisieren (z. B. nach Zweck, Besitzer oder Umgebung). Dies ist nützlich, wenn Sie viele Ressourcen desselben Typs haben. In diesem Fall können Sie schnell bestimmte Ressourcen basierend auf den zugewiesenen Tags bestimmen. Jedes Tag besteht aus einem Schlüssel und einem optionalen Wert, die Sie beide selbst definieren können. Sie können zum Beispiel eine Reihe von Tags für Ihre Objekttypen definieren, die Ihnen helfen, Ihre Geräte nach Typ nachzuverfolgen. Wir empfehlen die Erstellung von Tag-Schlüsseln, die die Anforderungen der jeweiligen Ressourcenart erfüllen. Die Verwendung einheitlicher Tag-Schlüssel vereinfacht das Verwalten der -Ressourcen.

Sie können die Ressourcen auf Grundlage der hinzugefügten oder angewendeten Tags durchsuchen und filtern. Sie können Abrechnungsgruppentags auch verwenden, um Ihre Kosten zu kategorisieren und zu verfolgen. Sie können auch Tags verwenden, um den Zugriff auf Ihre Ressourcen zu steuern (in Verwenden von Tags mit IAM-Richtlinien beschrieben).

Zur Vereinfachung der Verwendung bietet der Tag-Editor in der AWS Management Console eine zentrale, einheitliche Methode zum Erstellen und Verwalten Ihrer Tags. Weitere Informationen finden Sie unter Arbeiten mit dem Tag Editor in Arbeiten mit der AWS Management Console.

Sie können auch mithilfe der AWS CLI und der AWS IoT API mit Tags arbeiten. Sie können Tags bei der Erstellung mit Objektgruppen, Objekttypen, Themenregeln, Aufträgen, Sicherheitsprofilen und Abrechnungsgruppen verbinden, indem Sie in den folgenden Befehlen das Feld `Tags` verwenden:

- CreateBillingGroup

- CreateDestination
- CreateDeviceProfile
- CreateDynamicThingGroup
- CreateJob
- CreateOTAUpdate
- CreatePolicy
- CreateScheduledAudit
- CreateSecurityProfile
- CreateServiceProfile
- CreateStream
- CreateThingGroup
- CreateThingType
- CreateTopicRule
- CreateWirelessGateway
- CreateWirelessDevice

Sie können Tags für vorhandene Ressourcen, die das Markieren unterstützen, hinzufügen, ändern oder löschen. Verwenden Sie dazu die folgenden Befehle:

- TagResource
- ListTagsForResource
- UntagResource

Sie können Tag (Markierung)-Schlüssel und -Werte bearbeiten und Tags (Markierungen) jederzeit von einer Ressource entfernen. Sie können den Wert eines Tags (Markierung) zwar auf eine leere Zeichenfolge, jedoch nicht null festlegen. Wenn Sie ein Tag (Markierung) mit demselben Schlüssel wie ein vorhandener Tag (Markierung) für die Ressource hinzufügen, wird der alte Wert mit dem neuen überschrieben. Wenn Sie eine Ressource löschen, werden alle der Ressource zugeordneten Tags ebenfalls gelöscht.

Tag-Beschränkungen und -Einschränkungen

Die folgenden grundlegenden Einschränkungen gelten für Tags (Markierungen):

- Maximale Anzahl von Tags pro Ressource: 50
- Maximale Schlüssellänge — 127 Unicode-Zeichen in UTF-8
- Maximale Wertelänge — 255 Unicode-Zeichen in UTF-8
- Bei Tag-Schlüsseln und -Werten muss die Groß- und Kleinschreibung beachtet werden.
- Verwenden Sie nicht das Präfix `aws:` in Ihren Tag-Namen oder -Werten. Es ist für den AWS Gebrauch reserviert. Sie können keine Tag-Namen oder Werte mit diesem Präfix bearbeiten oder löschen. Tags mit diesem Präfix werden nicht zum Limit für Tags pro Ressource gezählt.
- Wenn Ihr Markierungsschema für mehrere Services und Ressourcen verwendet wird, denken Sie daran, dass die zulässigen Zeichen bei anderen Services möglicherweise eingeschränkt sind. Zu den erlaubten Zeichen gehören Buchstaben, Leerzeichen und Zahlen, die in dargestellt werden können UTF-8, sowie die folgenden Sonderzeichen: + - =. _:/@.

Verwenden von Tags mit IAM-Richtlinien

Sie können Tag-basierte Berechtigungen auf Ressourcenebene in den IAM-Richtlinien anwenden, die Sie für AWS IoT -API-Aktionen verwenden. Dies ermöglicht Ihnen eine bessere Kontrolle darüber, welche Ressourcen ein Benutzer erstellen, ändern oder verwenden kann. Sie können das `Condition`-Element (auch als `Condition`-Block bezeichnet) mit den folgenden Bedingungskontextschlüsseln und Werten in einer IAM-Richtlinie zum Steuern des Benutzerzugriffs (Berechtigungen) basierend auf den Tags einer Ressource verwenden:

- Verwenden Sie `aws:ResourceTag/`*`tag-key`*`: `*`tag-value`*, um Benutzeraktionen für Ressourcen mit bestimmten Tags zuzulassen oder zu verweigern.
- Verwenden Sie `aws:RequestTag/`*`tag-key`*`: `*`tag-value`*, um festzulegen, dass ein bestimmtes Tag verwendet (oder nicht verwendet) wird, wenn Sie eine API-Anfrage stellen, um eine Ressource zu erstellen oder zu ändern, die Tags zulässt.
- Verwenden Sie `aws:TagKeys: [`*`tag-key`*`, ...]`, um zu verlangen, dass ein bestimmter Satz von Tag-Schlüsseln verwendet wird (oder nicht), wenn eine API-Anforderung zum Erstellen einer Ressource durchgeführt wird, die Tags zulässt.

> **Note**
>
> Die Bedingungskontextschlüssel und -werte in einer IAM-Richtlinie gelten nur für AWS IoT Aktionen, bei denen ein Bezeichner für eine Ressource, die markiert werden kann, ein erforderlicher Parameter ist. Beispielsweise DescribeEndpoint ist die Verwendung von

nicht zulässig oder verweigert auf der Grundlage von Bedingungskontextschlüsseln und -werten, da in dieser Anforderung keine markierbare Ressource (Dinggruppen, Dingtypen, Themenregeln, Jobs oder Sicherheitsprofile) referenziert wird. Weitere Informationen über markierbare AWS IoT Ressourcen und die von ihnen unterstützten Bedingungsschlüssel finden Sie unter Aktionen, Ressourcen und Bedingungsschlüssel für. AWS IoT

Weitere Informationen finden Sie unter Zugriffssteuerung mit Tags im AWS Identity and Access Management Benutzerhandbuch. Der Abschnitt IAM-JSON-Richtlinienreferenz dieses Handbuchs enthält die detaillierte Syntax sowie Beschreibungen und Beispiele für Elemente, Variablen und die Auswertungslogik von JSON-Richtlinien in IAM.

Die folgende Beispielrichtlinie wendet zwei auf Tags basierende Einschränkungen für die `ThingGroup`-Aktionen an. Ein von dieser Richtlinie eingeschränkter IAM-Benutzer:

- Es kann keine Objektgruppe mit dem Tag „env=prod“ erstellt werden (im Beispiel siehe Zeile `"aws:RequestTag/env" : "prod"`).
- Kann keine Objektgruppe modifizieren oder darauf zugreifen, die den Tag „env=prod“ aufweist (im Beispiel vgl. die Zeile `"aws:ResourceTag/env" : "prod"`).

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "iot:CreateThingGroup",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/env": "prod"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
```

```
        "iot:UpdateThingGroup"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/env": "prod"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:UpdateThingGroup"
      ],
      "Resource": "*"
    }
  ]
}
```

Sie können auch mehrere Tag-Werte für einen bestimmten Tag-Schlüssel angeben, indem Sie sie wie folgt in einer Liste angeben:

```
        "StringEquals" : {
          "aws:ResourceTag/env" : ["dev", "test"]
        }
```

> **Note**
>
> Wenn Sie Benutzern den Zugriff zu Ressourcen auf der Grundlage von Tags (Markierungen) gewähren oder verweigern, müssen Sie daran denken, Benutzern explizit das Hinzufügen und Entfernen dieser Tags (Markierungen) von den jeweiligen Ressourcen unmöglich zu machen. Andernfalls können Benutzer möglicherweise Ihre Einschränkungen umgehen und sich Zugriff auf eine Ressource verschaffen, indem sie ihre Tags (Markierungen) modifizieren.

Abrechnungsgruppen

AWS IoT ermöglicht es Ihnen nicht, Stichwörter direkt auf einzelne Dinge anzuwenden, aber es ermöglicht Ihnen, Dinge in Abrechnungsgruppen zu platzieren und diesen Stichwörter zuzuweisen. Denn die AWS IoT Zuordnung von Kosten- und Nutzungsdaten auf der Grundlage von Stichwörtern ist auf Abrechnungsgruppen beschränkt.

AWS IoT Core denn LoRa WAN-Ressourcen wie WLAN-Geräte und Gateways können nicht zu Abrechnungsgruppen hinzugefügt werden. Sie können jedoch AWS IoT Dingen zugeordnet werden, die zu Abrechnungsgruppen hinzugefügt werden können.

Die folgenden Befehle sind verfügbar:

- AddThingToBillingGroup fügt ein Objekt zu einer Abrechnungsgruppe hinzu.
- CreateBillingGroup erstellt eine Abrechnungsgruppe.
- DeleteBillingGroup löscht die Abrechnungsgruppe.
- DescribeBillingGroup gibt Informationen über eine Abrechnungsgruppe zurück.
- ListBillingGroups listet die von Ihnen erstellen Abrechnungsgruppen auf.
- ListThingsInBillingGroup listet die Objekte auf, die Sie der jeweiligen Abrechnungsgruppe hinzugefügt haben.
- RemoveThingFromBillingGroup entfernt das angegebene Objekt aus der Abrechnungsgruppe.
- UpdateBillingGroup aktualisiert die Informationen zu der Abrechnungsgruppe.
- CreateThing ermöglicht es Ihnen, eine Abrechnungsgruppe für das Objekt anzugeben, wenn Sie es erstellen.
- DescribeThing gibt die Beschreibung eines Objekts zusammen mit der Abrechnungsgruppe, zu der das Objekt gehört (falls vorhanden) zurück.

Die AWS IoT Wireless-API bietet diese Aktionen, um drahtlose Geräte und Gateways mit AWS IoT Dingen zu verknüpfen.

- AssociateWirelessDeviceWithThing
- AssociateWirelessGatewayWithThing

Anzeigen von Kostenzuordnungs- und Nutzungsdaten

Sie können Abrechnungsgruppentags verwenden, um Ihre Kosten zu kategorisieren und zu verfolgen. Wenn Sie Stichwörter auf Abrechnungsgruppen (und damit auf die darin enthaltenen Elemente) anwenden, wird ein Kostenzuordnungsbericht als CSV-Datei (Comma-Separated Value) AWS generiert, in dem Ihre Nutzung und die Kosten nach Ihren Stichwörtern zusammengefasst werden. Sie können Tags anwenden, die geschäftliche Kategorien (wie Kostenstellen, Anwendungsnamen oder Eigentümer) darstellen, um die Kosten für mehrere Services zu organisieren. Weitere Informationen zur Verwendung von Tags für die Kostenzuordnung finden Sie unter Verwenden von Kostenzuordnungs-Tags im Benutzerhandbuch AWS -Fakturierungs- und Kostenverwaltung.

> **Note**
>
> Um Nutzungs- und Kostendaten korrekt den Objekten zuzuordnen, die Sie in Abrechnungsgruppen gesetzt haben, muss jedes Gerät bzw. jede Anwendung die folgenden Bedingungen erfüllen:
>
> - Registriert sein als Ding in. AWS IoT Weitere Informationen finden Sie unter Geräte verwalten mit AWS IoT.
> - Stellen Sie über MQTT eine Verbindung zum AWS IoT Message Broker her und verwenden Sie nur den Namen des Dings als Client-ID. Weitere Informationen finden Sie unter the section called “Gerätekommunikationsprotokolle”. Wenn Ihre Client-ID nicht mit dem Namen des Dings übereinstimmt, können Sie den exklusiven Ding-Anhang aktivieren, um die Zuordnung herzustellen. Weitere Informationen finden Sie unter ???.
> - Authentifizierung mit einem mit dem Objekt verbundenen Client-Zertifikat.

Für Abrechnungsgruppen sind die folgenden Preisdimensionen verfügbar (basierend auf der Aktivität der mit der Abrechnungsgruppe verbundenen Objekte):

- Verbindung (basierend auf dem Objektnamen, der als Client-ID verwendet wird)
- Messaging (basierend auf eingehenden und ausgehenden Nachrichten eines Objekts, nur MQTT)
- Schattenoperationen (basierend auf dem Objekt, dessen Nachricht eine Schattenaktualisierung ausgelöst hat)
- Ausgelöste Regeln (basierend auf dem Objekt, dessen eingehende Nachricht die Regel ausgelöst hat; gilt nicht für die Regeln, die von MQTT-Lebenszyklusereignissen ausgelöst wurden)

- Objektindex-Aktualisierungen (basierend auf dem Objekt, das dem Index hinzugefügt wurde)
- Remote-Aktionen (basierend auf dem aktualisierten Objekt)
- AWS IoT Device Defender erkennt Berichte (basierend auf dem Ding, dessen Aktivität gemeldet wird).

Kosten- und Nutzungsdaten, die auf Tags basieren (und für eine Abrechnungsgruppe gemeldet werden) reflektieren nicht die folgenden Aktivitäten:

- Geräte-Registrierungsoperationen (einschließlich Aktualisierungen von Objekten, Objektgruppen und Objekttypen). Weitere Informationen finden Sie unter Geräte verwalten mit AWS IoT).
- Aktualisierungen des Objektgruppenindex (beim Hinzufügen einer Objektgruppe)
- Index-Suchabfragen
- Gerätebereitstellung.
- AWS IoT Device DefenderAuditberichte.

Sicherheit bei AWS IoT

Cloud-Sicherheit AWS hat höchste Priorität. Als AWS Kunde profitieren Sie von einer Rechenzentrums- und Netzwerkarchitektur, die darauf ausgelegt sind, die Anforderungen der sicherheitssensibelsten Unternehmen zu erfüllen.

Sicherheit ist eine gemeinsame Verantwortung von Ihnen AWS und Ihnen. Das Modell der geteilten Verantwortung beschreibt dies als Sicherheit der Cloud und Sicherheit in der Cloud:

- Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, die AWS Dienste in der AWS Cloud ausführt. AWS bietet Ihnen auch Dienste, die Sie sicher nutzen können. Third-party Auditoren testen und verifizieren regelmäßig die Wirksamkeit unserer Sicherheitsmaßnahmen im Rahmen der AWS Compliance-Programme. Weitere Informationen zu den Compliance-Programmen, die für gelten AWS IoT, finden Sie unter AWS Services im Umfang der einzelnen Compliance-Programme.
- Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem AWS Dienst, den Sie nutzen. Sie sind auch für andere Faktoren verantwortlich, einschließlich der Vertraulichkeit Ihrer Daten, für die Anforderungen Ihres Unternehmens und für die geltenden Gesetze und Vorschriften.

Diese Dokumentation hilft Ihnen zu verstehen, wie Sie das Modell der gemeinsamen Verantwortung bei der Nutzung anwenden können AWS IoT. In den folgenden Themen erfahren Sie, wie Sie die Konfiguration vornehmen AWS IoT , um Ihre Sicherheits- und Compliance-Ziele zu erreichen. Sie erfahren auch, wie Sie andere AWS Dienste nutzen können, die Sie bei der Überwachung und Sicherung Ihrer AWS IoT Ressourcen unterstützen.

Topics

- AWS IoT Sicherheit
- Authentifizierung
- Autorisierung
- Datenschutz in AWS IoT Core
- Identitäts- und Zugriffsmanagement für AWS IoT
- Protokollieren und Überwachen
- Überprüfung der Einhaltung von Vorschriften für AWS IoT Core
- Resilienz in AWS IoT-Kern
- Verwenden AWS IoT Core mit VPC-Endpunkten mit Schnittstelle

- Sicherheit der Infrastruktur in AWS IoT
- Sicherheitsüberwachung von Produktionsflotten oder Geräten mit AWS IoT Core
- Bewährte Sicherheitsmethoden in AWS IoT Core
- AWS Schulung und Zertifizierung

AWS IoT Sicherheit

Jedes verbundene Gerät oder jeder verbundene Client muss über eine Berechtigung verfügen, um mit AWS IoT zu interagieren. Der gesamte Verkehr zu und von AWS IoT dort wird sicher über Transport Layer Security (TLS) gesendet. AWS Cloud-Sicherheitsmechanismen schützen Daten, wenn sie zwischen AWS IoT und anderen AWS Diensten übertragen werden.



- Sie sind verantwortlich für die Verwaltung von Geräte-Anmeldeinformationen (X.509 Zertifikate, AWS Anmeldeinformationen, Amazon Cognito Cognito-Identitäten, Verbundidentitäten oder benutzerdefinierte Authentifizierungstoken) und Richtlinien in. AWS IoT Sie sind auch dafür verantwortlich, jedem Gerät eindeutige Identitäten zuzuweisen und die Berechtigungen für jedes Gerät oder jede Gerätegruppe zu verwalten.
- Ihre Geräte stellen AWS IoT mithilfe von X.509 Zertifikaten oder Amazon Cognito Cognito-Identitäten über eine sichere TLS-Verbindung eine Verbindung her. Während der Forschung und Entwicklung sowie für einige Anwendungen, die API-Aufrufe tätigen oder verwenden WebSockets, können Sie sich auch mithilfe von IAM-Benutzern und -Gruppen oder benutzerdefinierten Authentifizierungstoken authentifizieren. Weitere Informationen finden Sie unter IAM-Benutzer, -Gruppen und -Rollen.

- Wenn Sie die AWS IoT Authentifizierung verwenden, ist der Message Broker dafür verantwortlich, Ihre Geräte zu authentifizieren, Gerätedaten sicher aufzunehmen und Zugriffsberechtigungen zu gewähren oder zu verweigern, die Sie mithilfe von Richtlinien für Ihre Geräte angeben. AWS IoT
- Wenn Sie die benutzerdefinierte Authentifizierung verwenden, ist ein benutzerdefinierter Autorisierer dafür verantwortlich, Ihre Geräte zu authentifizieren und Zugriffsberechtigungen zu gewähren oder zu verweigern, die Sie für Ihre Geräte mithilfe unserer IAM-Richtlinien angeben. AWS IoT
- Die AWS IoT Regel-Engine leitet Gerätedaten gemäß den von Ihnen definierten Regeln an andere Geräte oder AWS Dienste weiter. Es dient AWS Identity and Access Management dazu, Daten sicher an ihren Bestimmungsort zu übertragen. Weitere Informationen finden Sie unter Identitäts- und Zugriffsmanagement für AWS IoT.

Authentifizierung

Die Authentifizierung ist ein Mechanismus, mit dem Sie die Identität eines Clients oder Servers überprüfen können. Bei der Serverauthentifizierung stellen Geräte oder andere Clients sicher, dass sie mit einem tatsächlichen AWS IoT Endpunkt kommunizieren. Die Client-Authentifizierung ist der Prozess, bei dem sich Geräte oder andere Clients selbst authentifizieren. AWS IoT

X.509 Überblick über das Zertifikat

X.509 Zertifikate sind digitale Zertifikate, die den X.509 Public-Key-Infrastrukturstandard verwenden, um einen öffentlichen Schlüssel mit einer in einem Zertifikat enthaltenen Identität zu verknüpfen. X.509 Zertifikate werden von einer vertrauenswürdigen Stelle, einer so genannten Zertifizierungsstelle (CA), ausgestellt. Die Zertifizierungsstelle verwaltet ein oder mehrere spezielle Zertifikate, sogenannte CA-Zertifikate, die sie zur Ausstellung von X.509 Zertifikaten verwendet. Nur die Zertifizierungsstelle hat Zugriff auf CA-Zertifikate. X.509 Zertifikatsketten werden sowohl für die Serverauthentifizierung durch Clients als auch für die Clientauthentifizierung durch den Server verwendet.

Serverauthentifizierung

Wenn Ihr Gerät oder ein anderer Client versucht, eine Verbindung herzustellen AWS IoT Core, sendet der AWS IoT Core Server ein X.509 Zertifikat, das Ihr Gerät zur Authentifizierung des Servers verwendet. Die Authentifizierung erfolgt auf der TLS-Ebene durch Validierung der X.509 Zertifikatskette. Dies ist dieselbe Methode, die Ihr Browser verwendet, wenn Sie eine HTTPS-URL

besuchen. Informationen zum Verwenden von Zertifikaten von Ihrer eigenen Zertifizierungsstelle finden Sie unter Verwalten eigener CA-Zertifikate.

Wenn Ihre Geräte oder andere Clients eine TLS-Verbindung zu einem AWS IoT Core Endpunkt aufbauen, wird eine Zertifikatskette AWS IoT Core angezeigt, anhand derer die Geräte überprüfen, ob sie mit einem anderen Server kommunizieren AWS IoT Core und nicht mit einem anderen Server, der sich als solcher ausgibt. AWS IoT Core Welche Kette angezeigt wird, hängt von einer Kombination aus dem Endpunkttyp, zu dem das Gerät eine Verbindung herstellt, und der Cipher Suite ab, die der Client und die während des TLS-Handshakes AWS IoT Core ausgehandelt haben.

Endpunkttypen

AWS IoT Core unterstützt. `iot:Data-ATS` `iot:Data-ATS`Endgeräte legen ein Serverzertifikat vor, das von einer Amazon Trust Services-Zertifizierungsstelle signiert wurde.

Zertifikate, die von ATS-Endpunkten vorgewiesen werden, sind von Starfield gegensigniert. Einige TLS-Client-Implementierungen erfordern die Überprüfung der Vertrauenswürdigkeit und die Installation der Starfield CA-Zertifikate in den Vertrauensspeichern des Clients.

> **Warning**
>
> Die Verwendung einer Zertifikat-Pinning-Methode, die das gesamte Zertifikat (einschließlich des Ausstellernamens usw.) hasht, wird nicht empfohlen, da dies zum Scheitern der Zertifikatsprüfung führt, da die von uns bereitgestellten ATS-Zertifikate von Starfield gegensigniert sind und einen anderen Ausstellernamen haben.

> **Important**
>
> Verwenden Sie `iot:Data-ATS` Endpunkte. Zertifikate von Symantec und Verisign sind veraltet und werden von nicht mehr unterstützt. AWS IoT Core

Sie können den Befehl `describe-endpoint` verwenden, um Ihren ATS-Endpunkt zu erstellen.

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

Der Befehl `describe-endpoint` gibt einen Endpunkt im folgenden Format zurück.

```
account-specific-prefix.iot.your-region.amazonaws.com
```

> **Note**
>
> Beim ersten Aufruf von `describe-endpoint` wird ein Endpunkt erstellt. Alle nachfolgenden Aufrufe von `describe-endpoint` geben den gleichen Endpunkt zurück.

> **Note**
>
> Um Ihren **`iot:Data-ATS`** Endpunkt in der Konsole zu sehen, wählen Sie Einstellungen AWS IoT Core . Die Konsole zeigt nur den `iot:Data-ATS`-Endpunkt an.

Erstellen eines **`IotDataPlaneClient`**mit dem AWS SDK für Java

Um eine zu erstellen`IotDataPlaneClient`, die einen `iot:Data-ATS` Endpunkt verwendet, müssen Sie wie folgt vorgehen.

- Erstellen Sie mithilfe der DescribeEndpointAPI einen `iot:Data-ATS` Endpunkt.
- Geben Sie diesen Endpunkt an, wenn Sie den `IotDataPlaneClient` erstellen.

Im folgenden Beispiel werden beide Operationen ausgeführt.

```
public void setup() throws Exception {
        IotClient client =
 IotClient.builder().credentialsProvider(CREDENTIALS_PROVIDER_CHAIN).region(Region.US_EAST_1).b
        String endpoint = client.describeEndpoint(r -> r.endpointType("iot:Data-
ATS")).endpointAddress();
        iot = IotDataPlaneClient.builder()
                                .credentialsProvider(CREDENTIALS_PROVIDER_CHAIN)
                                .endpointOverride(URI.create("https://" + endpoint))
                                .region(Region.US_EAST_1)
                                .build();
}
```

CA-Zertifikate für die Serverauthentifizierung

Je nachdem, welchen Typ von Datenendpunkt Sie verwenden und welche Verschlüsselungssuite Sie ausgehandelt haben, werden AWS IoT Core Serverauthentifizierungszertifikate mit einem der folgenden Root-CA-Zertifikate signiert:

Amazon Trust Services-Endpunkte (bevorzugt)

> **Note**
>
> Möglicherweise müssen Sie mit der rechten Maustaste auf diese Links klicken und Link speichern unter... auswählen, um diese Zertifikate als Dateien zu speichern.

- RSA 2048-Bit-Schlüssel: Amazon Root CA 1.
- RSA 4096-Bit-Schlüssel: Amazon Root CA 2. Für die spätere Verwendung reserviert.
- ECC-256-Bit-Schlüssel: Amazon Root CA 3.
- ECC-384-Bit-Schlüssel: Amazon Root CA 4. Für die spätere Verwendung reserviert.

Diese Zertifikate werden alle durch das Starfield Root CA-Zertifikat signiert. Alle neuen AWS IoT Core Regionen, beginnend mit der Markteinführung von AWS IoT Core in der Region Asien-Pazifik (Mumbai) am 9. Mai 2018, bieten ausschließlich ATS-Zertifikate an.

VeriSign Endgeräte (veraltet)

- RSA 2048-Bit-Schlüssel: Öffentliches primäres G5-Root-CA-Zertifikat der VeriSign Klasse 3

Richtlinien für die Serverauthentifizierung

Es gibt viele Variablen, die die Fähigkeit eines Geräts zur Überprüfung des AWS IoT Core -Server-Authentifizierungszertifikats beeinflussen können. So können beispielsweise Geräte zu speicherbeschränkt sein, um alle möglichen Root-CA-Zertifikate aufzunehmen, oder Geräte können eine nicht standardmäßige Methode der Zertifikatsvalidierung implementieren. Aus diesen Gründen empfehlen wir, diese Richtlinien zu befolgen:

- Wir empfehlen, dass Sie Ihren ATS-Endpunkt verwenden und alle unterstützten Amazon Root CA-Zertifikate installieren.

- Wenn Sie nicht alle diese Zertifikate auf Ihrem Gerät speichern können und Ihre Geräte keine ECC-based Validierung verwenden, können Sie die Zertifikate und ECC-Zertifikate weglassen. Amazon Root CA 3Amazon Root CA 4 Wenn Ihre Geräte die RSA-based Zertifikatsvalidierung nicht implementieren, können Sie die Zertifikate Amazon Root CA 1und Amazon Root CA 2RSA weglassen. Möglicherweise müssen Sie mit der rechten Maustaste auf diese Links klicken und Link speichern unter... auswählen, um diese Zertifikate als Dateien zu speichern.
- Wenn bei der Verbindung mit Ihrem ATS-Endpunkt Probleme bei der Validierung von Serverzertifikaten auftreten, versuchen Sie, das entsprechende signierte Amazon Root CA-Zertifikat in Ihrem Vertrauensspeicher hinzuzufügen. Möglicherweise müssen Sie mit der rechten Maustaste auf diese Links klicken und Link speichern unter... auswählen, um diese Zertifikate als Dateien zu speichern.
  - Cross-signed Amazon Root CA 1
  - Cross-signed Amazon Root CA 2- Reserviert für future Verwendung.
  - Cross-signed Amazon Root CA 3
  - Cross-signed Amazon Root CA 4- Reserviert für future Verwendung.
- Wenn bei der Validierung von Serverzertifikaten Probleme auftreten, muss Ihr Gerät möglicherweise explizit der Stammzertifizierungsstelle vertrauen. Versuchen Sie, das Starfield Root CA Certificate zu Ihrem Trust Store hinzuzufügen.
- Wenn nach der Ausführung der obigen Schritte immer noch Probleme auftreten, wenden Sie sich bitte an den AWS -Developer Support.

> **Note**
>
> CA-Zertifikate haben ein Ablaufdatum, nach dem sie nicht mehr zur Validierung eines Serverzertifikats verwendet werden können. Es kann sein, dass CA-Zertifikate vor ihrem Ablaufdatum ersetzt werden müssen. Stellen Sie sicher, dass Sie die CA-Stammzertifikate auf all Ihren Geräten oder Clients aktualisieren können, um eine ununterbrochene Verbindung sicherzustellen und bei bewährten Sicherheitsmethoden stets auf dem aktuellen Stand zu sein.

> **Note**
>
> Wenn Sie AWS IoT Core in Ihrem Gerätecode eine Verbindung herstellen, übergeben Sie das Zertifikat an die API, die Sie für die Verbindung verwenden. Die von Ihnen verwendete

API hängt vom SDK ab. Weitere Informationen finden Sie im Abschnitt zu AWS IoT Core - Geräte-SDKs.

Client-Authentifizierung

AWS IoT unterstützt drei Arten von Identitätsprinzipalen für die Geräte- oder Client-Authentifizierung:

- X.509 Client-Zertifikate
- IAM-Benutzer, -Gruppen und -Rollen
- Amazon-Cognito-Identitäten

Diese Identitäten können mit Geräten, mobilen, Web- oder Desktop-Anwendungen verwendet werden. Sie können sogar von Benutzern verwendet werden, die Befehle der AWS IoT Befehlszeilenschnittstelle (CLI) eingeben. In der Regel verwenden AWS IoT Geräte X.509 Zertifikate, während mobile Anwendungen Amazon Cognito Cognito-Identitäten verwenden. Web- und Desktop-Anwendungen verwenden IAM- oder Verbundidentitäten. AWS CLI -Befehle verwenden IAM. Weitere Informationen zu IAM-Identitäten finden Sie unter Identitäts- und Zugriffsmanagement für AWS IoT.

X.509 Client-Zertifikate

X.509 Zertifikate AWS IoT bieten die Möglichkeit, Client- und Geräteverbindungen zu authentifizieren. Client-Zertifikate müssen registriert werden, AWS IoT bevor ein Client mit AWS IoT ihnen kommunizieren kann. Ein Client-Zertifikat kann für mehrere AWS-Konto s in derselben Region registriert werden AWS-Region , um das Verschieben von Geräten zwischen Ihren AWS-Konto s in derselben Region zu erleichtern. Weitere Informationen finden Sie unter Verwendung von X.509 Client-Zertifikaten in mehreren AWS-Konto s bei der Registrierung mehrerer Konten.

Wir empfehlen, jedem Gerät oder Client ein eindeutiges Zertifikat zuzuordnen, damit feingranulare Client-Verwaltungsaktionen einschließlich Zertifikatswiderruf möglich sind. Geräte und Clients müssen darüber hinaus das Rotieren und Ersetzen von Zertifikaten unterstützen, um eine reibungslose Ausführung sicherzustellen, wenn Zertifikate ablaufen.

Informationen zur Verwendung von X.509 Zertifikaten zur Unterstützung mehrerer Geräte finden Sie unter Gerätebereitstellung So überprüfen Sie die verschiedenen Zertifikatsverwaltungs- und Bereitstellungsoptionen, die AWS IoT unterstützt werden.

AWS IoT unterstützt die folgenden Typen von X.509 Client-Zertifikaten:

- X.509 Zertifikate, generiert von AWS IoT
- X.509 Zertifikate, die von einer Zertifizierungsstelle signiert wurden, die bei registriert ist AWS IoT.
- X.509 Zertifikate, die von einer Zertifizierungsstelle signiert wurden, bei der nicht registriert ist AWS IoT.

In diesem Abschnitt wird beschrieben, wie X.509 Zertifikate in verwaltet AWS IoT werden. Sie können die AWS IoT Konsole verwenden oder AWS CLI die folgenden Zertifikatsvorgänge ausführen:

- Create AWS IoT Client-Zertifikate
- Erstellen eigener Clientzertifikate
- Registrieren eines Clientzertifikats
- Aktivieren oder Deaktivieren eines Clientzertifikats
- Widerrufen eines Clientzertifikats

Weitere Informationen zu den AWS CLI Befehlen, die diese Operationen ausführen, finden Sie unter AWS IoT CLI-Referenz.

Verwenden von X.509 Client-Zertifikaten

X.509 Zertifikate authentifizieren Client- und Geräteverbindungen zu AWS IoT. X.509Zertifikate bieten mehrere Vorteile gegenüber anderen Identifizierungs- und Authentifizierungsmechanismen. X.509 Zertifikate ermöglichen die Verwendung asymmetrischer Schlüssel mit Geräten. Sie könnten beispielsweise private Schlüssel in einen sicheren Speicher auf einem Gerät brennen, sodass vertrauliches kryptografisches Material das Gerät niemals verlässt. X.509Zertifikate bieten eine stärkere Client-Authentifizierung als andere Schemata wie Benutzername und Passwort oder Inhaber-Token, da der private Schlüssel das Gerät niemals verlässt.

AWS IoT authentifiziert Client-Zertifikate mithilfe des Client-Authentifizierungsmodus des TLS-Protokolls. TLS-Unterstützung ist in vielen Programmiersprachen und Betriebssystemen verfügbar und stellt die gängige Verschlüsselungsmethode für Daten dar. AWS IoT Fordert bei der TLS-Client-Authentifizierung ein X.509 Client-Zertifikat an und validiert den Status des Zertifikats AWS-Konto anhand einer Zertifikatsregistrierung. Anschließend fordert es den Client auf, den Besitz des privaten Schlüssels nachzuweisen, der dem im Zertifikat enthaltenen öffentlichen Schlüssel entspricht. AWS IoT verlangt von den Clients, die Server Name Indication (SNI) -Erweiterung an das Transport Layer

Security (TLS) -Protokoll zu senden. Weitere Informationen zum Konfigurieren der SNI-Erweiterung finden Sie unter Transportsicherheit in AWS IoT Core.

Um eine sichere und konsistente Client-Verbindung zum AWS IoT Core zu ermöglichen, muss ein X.509 Client-Zertifikat über Folgendes verfügen:

- In AWS IoT Core registriert. Weitere Informationen finden Sie unter Registrieren eines Clientzertifikats.
- Über den Status „`ACTIVE`“ verfügen. Weitere Informationen finden Sie unter Aktivieren oder Deaktivieren eines Clientzertifikats.
- Das Ablaufdatum des Zertifikats ist noch nicht erreicht.

Sie können Clientzertifikate erstellen, die die Amazon Root CA verwenden, und Sie können Ihre eigenen von einer anderen Zertifizierungsstelle (Certificate Authority, CA) signierten Clientzertifikate verwenden. Weitere Informationen zur Verwendung der AWS IoT Konsole zum Erstellen von Zertifikaten, die die Amazon Root-CA verwenden, finden Sie unterCreate AWS IoT Client-Zertifikate. Weitere Informationen zur Verwendung Ihrer eigenen X.509 Zertifikate finden Sie unterErstellen eigener Clientzertifikate.

Datum und Uhrzeit, zu denen von einem CA-Zertifikat signierte Zertifikate ablaufen, werden bei der Erstellung des Zertifikats festgelegt. X.509 Zertifikate, die von generiert wurden, AWS IoT laufen am 31. Dezember 2049 um Mitternacht UTC ab (31.12.2049T23:59:59Z).

AWS IoT Device Defender kann Audits an Ihren Geräten durchführen AWS-Konto und dabei die gängigen Best Practices für die IoT-Sicherheit unterstützen. Dazu gehört die Verwaltung der Ablaufdaten von X.509 Zertifikaten, die von Ihrer CA oder der Amazon Root CA signiert wurden. Weitere Informationen zur Verwaltung des Ablaufdatums eines Zertifikats finden Sie unter Ablaufen von Gerätezertifikaten und Ablaufende CA-Zertifikate.

Im offiziellen AWS IoT Blog finden Sie einen tieferen Einblick in die Verwaltung der Rotation von Gerätezertifikaten und bewährte Sicherheitsmethoden unter So verwalten Sie die Rotation von IoT-Gerätezertifikaten mithilfe von AWS IoT.

Verwendung von X.509 Client-Zertifikaten in mehreren AWS-Konto s bei der Registrierung mehrerer Konten

Multi-account Die Registrierung ermöglicht es, Geräte zwischen Ihren AWS-Konto Geräten in derselben Region oder in verschiedenen Regionen zu verschieben. Sie können ein Gerät in

einem Testkonto registrieren, testen und konfigurieren und dasselbe Gerät samt Gerätezertifikat anschließend in einem Produktionskonto registrieren und verwenden. Sie können auch das Client-Zertifikat auf dem Gerät oder die Gerätezertifikate ohne registrierte Zertifizierungsstelle registrieren AWS IoT. Weitere Informationen finden Sie unter Registrieren eines Clientzertifikats, das von einer nicht registrierten CA (CLI) signiert wurde.

> **Note**
>
> Zertifikate, die für die Registrierung mehrerer Konten verwendet werden, werden auf den Endpunkttypen `iot:Data-ATS`, `iot:Data` (Legacy), `iot:Jobs` und `iot:CredentialProvider` unterstützt. Weitere Informationen zu AWS IoT Geräteendpunkten finden Sie unterAWS IoT Gerätedaten und Dienstendpunkte.

Geräte, die die Registrierung mehrerer Konten verwenden, müssen die Server Name Indication (SNI) -Erweiterung an das Transport Layer Security (TLS) -Protokoll senden und die vollständige Endpunktadresse im `host_name` Feld angeben, wenn sie eine Verbindung herstellen. AWS IoT AWS IoT verwendet die Endpunktadresse`host_name`, um die Verbindung an das richtige AWS IoT Konto weiterzuleiten. Vorhandene Geräte, die keine gültige Endpunktadresse in `host_name` senden, funktionieren weiterhin, können aber die Funktionen nicht nutzen, für die diese Informationen benötigt werden. Weitere Informationen zur SNI-Erweiterung und zum Identifizieren der Endpunktadresse für das Feld `host_name` finden Sie unter Transportsicherheit in AWS IoT Core.

So verwenden Sie die Registrierung für mehrere Konten

1. Sie können die Gerätezertifikate ohne Zertifizierungsstelle registrieren. Sie können die signierende Zertifizierungsstelle in mehreren Konten im `SNI_ONLY`-Modus registrieren und diese Zertifizierungsstelle verwenden, um dasselbe Clientzertifikat für mehrere Konten zu registrieren. Weitere Informationen finden Sie unter Registrieren Sie ein CA-Zertifikat im SNI_ONLY-Modus (CLI) – Empfohlen.
2. Sie können die Gerätezertifikate ohne Zertifizierungsstelle registrieren. Siehe Registrieren eines von einer nicht registrierten CA signierten Clientzertifikats (CLI). Die Registrierung einer Zertifizierungsstelle ist optional. Sie müssen die Zertifizierungsstelle, mit der die Gerätezertifikate signiert wurden, nicht registrieren AWS IoT.

Algorithmen zum Signieren von Zertifikaten werden unterstützt von AWS IoT

AWS IoT unterstützt die folgenden Algorithmen zum Signieren von Zertifikaten:

- SHA256WITHRSA
- SHA384WITHRSA
- SHA512WITHRSA
- SHA256 MIT RSA UND MGF1 () RSASSA-PSS
- SHA384 MIT RSA UND MGF1 () RSASSA-PSS
- SHA512 MIT RSA UND MGF1 () RSASSA-PSS
- DSA_WITH_SHA256
- ECDSA-WITH-SHA256
- ECDSA-WITH-SHA384
- ECDSA-WITH-SHA512

Weitere Informationen zur Authentifizierung und Sicherheit von Zertifikaten finden Sie unter Qualität der Gerätezertifikatschlüssel.

> **Note**
>
> Die Zertifikatssignierungsanforderung (Certificate Signing Request, CSR) muss einen öffentlichen Schlüssel enthalten. Der Schlüssel kann entweder ein RSA-Schlüssel mit einer Länge von mindestens 2.048 Bit oder ein ECC-Schlüssel aus NIST- P-256, NIST- oder NIST-Kurven sein. P-384 P-521 Weitere Informationen finden Sie unter CreateCertificateFromCsr in der AWS IoT -API-Referenz.

Schlüsselalgorithmen, unterstützt von AWS IoT

Die folgende Tabelle zeigt, wie Schlüsselalgorithmen unterstützt werden:

Schlüsselalgorithmus	Algorithmus zum Signieren von Zertifikaten	TLS-Version	Unterstützt? Sie können zwischen Yes und No wählen
RSA mit einer Schlüssel größe von mindestens 2048 Bit	Alle	TLS 1.2 TLS 1.3	Ja

Schlüsselalgorithmus	Algorithmus zum Signieren von Zertifikaten	TLS-Version	Unterstützt? Sie können zwischen Yes und No wählen
ECC NIST P-256/P-384/ P-521	Alle	TLS 1.2 TLS 1.3	Ja
RSA-PSS mit einer Schlüsselgröße von mindestens 2048 Bit	Alle	TLS 1.2	Nein
RSA-PSS mit einer Schlüsselgröße von mindestens 2048 Bit	Alle	TLS 1.3	Ja

Um ein Zertifikat zu erstellen CreateCertificateFromCSR, können Sie einen unterstützten Schlüsselalgorithmus verwenden, um einen öffentlichen Schlüssel für Ihre CSR zu generieren. Um Ihr eigenes Zertifikat mit RegisterCertificateoder zu registrieren RegisterCertificateWithoutCA, können Sie einen unterstützten Schlüsselalgorithmus verwenden, um einen öffentlichen Schlüssel für das Zertifikat zu generieren.

Weitere Informationen finden Sie unter Sicherheitsrichtlinien.

Create AWS IoT Client-Zertifikate

AWS IoT stellt Client-Zertifikate bereit, die von der Amazon Root Certificate Authority (CA) signiert wurden.

In diesem Thema wird beschrieben, wie Sie ein von der Amazon Root-Zertifizierungsstelle signiertes Clientzertifikat erstellen und die Zertifikatsdateien herunterladen. Nachdem Sie die Clientzertifikatdateien erstellt haben, müssen Sie sie auf dem Client installieren.

> **Note**
>
> Jedes von bereitgestellte X.509 Client-Zertifikat AWS IoT enthält die Attribute des Ausstellers und des Antragstellers, die Sie bei der Erstellung des Zertifikats festgelegt haben. Die Zertifikatsattribute können erst ab der Erstellung des Zertifikats nicht mehr modifiziert werden.

Sie können die AWS IoT Konsole oder die verwenden AWS CLI , um ein von der Amazon Root-Zertifizierungsstelle signiertes AWS IoT Zertifikat zu erstellen.

Erstellen Sie ein AWS IoT Zertifikat (Konsole)

Um ein zu erstellen AWS IoT Zertifikat mit dem AWS IoT Konsole

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die AWS IoT Konsole.
2. Wählen Sie im Navigationsbereich Sichern, Zertifikate und anschließend Erstellen.
3. Wählen Sie One-click Zertifikatserstellung (empfohlen) — Zertifikat erstellen.
4. Laden Sie die Clientzertifikatdateien für das Objekt, den öffentlichen Schlüssel und den privaten Schlüssel auf der Seite Zertifikat erstellt! an einen sicheren Speicherort herunter. Diese von generierten Zertifikate AWS IoT sind nur für die Verwendung mit AWS IoT Diensten verfügbar.

   Wenn Sie auch die Amazon-Root-CA-Zertifikatdatei benötigen, finden Sie auf dieser Seite auch den Link zu der Seite, auf der Sie sie herunterladen können.
5. Es wurde nun ein Clientzertifikat erstellt und bei AWS IoT registriert. Sie müssen das Zertifikat aktivieren, bevor Sie es in einem Client verwenden.

   Wählen Sie Aktivieren, um das Clientzertifikat jetzt zu aktivieren. Wenn Sie das Zertifikat jetzt nicht aktivieren möchten, können Sie unter Aktivieren eines Clientzertifikats (Konsole) erfahren, wie Sie das Zertifikat zu einem späteren Zeitpunkt aktivieren können.
6. Wenn Sie eine Richtlinie an das Zertifikat anfügen möchten, wählen Sie Anfügen einer Richtlinie.

   Wenn Sie jetzt keine Richtlinie anfügen möchten, wählen Sie Fertig , um den Vorgang abzuschließen. Sie können zu einem späteren Zeitpunkt eine Richtlinie anfügen.

Wenn Sie den Vorgang abgeschlossen haben, installieren Sie die Zertifikatdateien auf dem Client.

Erstellen Sie ein AWS IoT Zertifikat (CLI)

Das AWS CLI bietet den create-keys-and-certificate Befehl zum Erstellen von Client-Zertifikaten, die von der Amazon Root-Zertifizierungsstelle signiert wurden. Mit diesem Befehl wird die Amazon Root-CA-Zertifikatdatei jedoch nicht heruntergeladen. Sie können die Amazon Root-CA-Zertifikatdatei von CA-Zertifikate für die Serverauthentifizierung herunterladen.

Dieser Befehl erstellt private Schlüssel-, öffentliche Schlüssel- und X.509 Zertifikatsdateien und registriert und aktiviert das Zertifikat mit AWS IoT.

```
aws iot create-keys-and-certificate \
    --set-as-active \
    --certificate-pem-outfile certificate_filename.pem \
    --public-key-outfile public_filename.key \
    --private-key-outfile private_filename.key
```

Wenn Sie das Zertifikat bei der Erstellung und Registrierung nicht aktivieren möchten, erstellt dieser Befehl Dateien mit privatem Schlüssel, öffentlichem Schlüssel und X.509 Zertifikat und registriert das Zertifikat, aktiviert es jedoch nicht. Aktivieren eines Clientzertifikats (CLI)beschreibt, wie das Zertifikat später aktiviert wird.

```
aws iot create-keys-and-certificate \
    --no-set-as-active \
    --certificate-pem-outfile certificate_filename.pem \
    --public-key-outfile public_filename.key \
    --private-key-outfile private_filename.key
```

Installieren Sie die Zertifikatdateien auf dem Client.

Erstellen eigener Clientzertifikate

AWS IoT unterstützt Client-Zertifikate, die von einer beliebigen Stamm- oder Zwischenzertifizierungsstelle (CA) signiert wurden. AWS IoT verwendet CA-Zertifikate, um den Besitz von Zertifikaten zu überprüfen. Um Gerätezertifikate verwenden zu können, die von einer Zertifizierungsstelle signiert wurden, die nicht die Zertifizierungsstelle von Amazon ist, muss das Zertifikat der Zertifizierungsstelle registriert sein, AWS IoT damit wir den Besitz des Gerätezertifikats überprüfen können.

AWS IoT unterstützt mehrere Möglichkeiten, eigene Zertifikate mitzubringen (BYOC):

- Registrieren Sie zunächst die Zertifizierungsstelle, die zum Signieren der Clientzertifikate verwendet wird, und registrieren Sie dann einzelne Clientzertifikate. Wenn Sie das Gerät oder den Client bei der ersten Verbindung mit seinem Client-Zertifikat registrieren möchten AWS IoT (auch bekannt als Just-in-TimeProvisioning), müssen Sie die signierende Zertifizierungsstelle bei registrieren AWS IoT und die automatische Registrierung aktivieren.

- Wenn Sie die signierende Zertifizierungsstelle nicht registrieren können, können Sie Clientzertifikate auch ohne CA registrieren. Bei Geräten, die ohne CA registriert sind, müssen Sie Server Name Indication (SNI) angeben, wenn Sie eine Verbindung zu AWS IoT herstellen.

> **Note**
>
> Um Client-Zertifikate mithilfe einer Zertifizierungsstelle zu registrieren, müssen Sie die signierende Zertifizierungsstelle bei AWS IoT keiner anderen Zertifizierungsstelle in der Hierarchie registrieren.

> **Note**
>
> Ein CA-Zertifikat kann nur im `DEFAULT`-Modus von einem Konto in einer Region registriert werden. Ein CA-Zertifikat kann nur im `SNI_ONLY`-Modus von mehreren Konten in einer Region registriert werden.

Weitere Informationen zur Verwendung von X.509 Zertifikaten zur Unterstützung mehrerer Geräte finden Sie unter Gerätebereitstellung So überprüfen Sie die verschiedenen Zertifikatsverwaltungs- und Bereitstellungsoptionen, die AWS IoT unterstützt werden.

Themen

- Verwalten eigener CA-Zertifikate
- Erstellen eines Clientzertifikats mit Ihrem CA-Zertifikat

Verwalten eigener CA-Zertifikate

In diesem Abschnitt werden allgemeine Aufgaben für die Verwaltung eigener Zertifizierungsstellenzertifikate (Certificate Authority, CA) beschrieben.

Sie können Ihre Zertifizierungsstelle (CA) registrieren, AWS IoT wenn Sie Clientzertifikate verwenden, die von einer Zertifizierungsstelle signiert wurden, die diese AWS IoT nicht erkennt.

Wenn Sie möchten, dass Clients ihre Client-Zertifikate AWS IoT bei der ersten Verbindung automatisch registrieren, muss die Zertifizierungsstelle, die die Client-Zertifikate signiert hat, bei der

registriert sein AWS IoT. Andernfalls müssen Sie das CA-Zertifikat, das die Clientzertifikate signiert hat, nicht registrieren.

> **Note**
>
> Ein CA-Zertifikat kann nur im `DEFAULT`-Modus von einem Konto in einer Region registriert werden. Ein CA-Zertifikat kann nur im `SNI_ONLY`-Modus von mehreren Konten in einer Region registriert werden.

Themen:

- So erstellen Sie ein CA-Zertifikat
- Registrieren eines CA-Zertifikats
- Deaktivieren eines CA-Zertifikats

So erstellen Sie ein CA-Zertifikat

Wenn Sie kein CA-Zertifikat besitzen, können Sie OpenSSL-v1.1.1i-Tools verwenden, um ein Zertifikat zu erstellen.

> **Note**
>
> Sie können dieses Verfahren nicht in der AWS IoT Konsole ausführen.

Um ein CA-Zertifikat zu erstellen, verwenden Sie OpenSSL v1.1.1i Tools

1. Erzeugen Sie ein Schlüsselpaar.

```
openssl genrsa -out root_CA_key_filename.key 2048
```

2. Verwenden Sie den privaten Schlüssel des Schlüsselpaars zur Erzeugung eines CA-Zertifikats.

```
openssl req -x509 -new -nodes \
    -key root_CA_key_filename.key \
    -sha256 -days 1024 \
    -out root_CA_cert_filename.pem
```

Registrieren eines CA-Zertifikats

Diese Verfahren beschreiben, wie Sie ein Zertifikat von einer Zertifizierungsstelle (CA) registrieren, die nicht die Zertifizierungsstelle von Amazon ist. AWS IoT Core verwendet CA-Zertifikate, um den Besitz von Zertifikaten zu überprüfen. Um Gerätezertifikate zu verwenden, die von einer Zertifizierungsstelle signiert wurden, bei der es sich nicht um die Zertifizierungsstelle von Amazon handelt, müssen Sie das CA-Zertifikat registrieren, AWS IoT Core damit die Inhaberschaft des Gerätezertifikats verifiziert werden kann.

Registrieren eines CA-Zertifikats (Konsole)

> **Note**
>
> Starten Sie die Konsole unter CA-Zertifikat registrieren, um ein CA-Zertifikat in der Konsole zu registrieren. Sie können Ihre CA im Multi-account Modus registrieren, ohne dass Sie ein Bestätigungszertifikat vorlegen oder auf den privaten Schlüssel zugreifen müssen. Eine CA kann im Multi-account Modus von mehreren AWS-Konten gleichzeitig registriert werden AWS-Region. Sie können Ihre CA im Single-account Modus registrieren, indem Sie ein Bestätigungszertifikat und einen Eigentumsnachweis für den privaten Schlüssel der CA vorlegen.

Registrieren eines CA-Zertifikats (CLI)

Sie können ein CA-Zertifikat im `DEFAULT`-Modus oder im `SNI_ONLY`-Modus registrieren. Eine Zertifizierungsstelle kann einzeln im `DEFAULT` Modus registriert werden AWS-Region. AWS-Konto Eine CA kann im `SNI_ONLY` Modus von mehreren AWS-Konten gleichzeitig registriert werden AWS-Region. Weitere Informationen zu CA-Zertifikaten finden Sie unter certificateMode.

> **Note**
>
> Es wird empfohlen, dass Sie eine CA im `SNI_ONLY`-Modus registrieren. Sie müssen weder ein Bestätigungszertifikat noch Zugriff auf den privaten Schlüssel vorlegen, und Sie können die Zertifizierungsstelle von mehreren AWS-Konten gleichzeitig registrieren AWS-Region.

Registrieren Sie ein CA-Zertifikat im SNI_ONLY-Modus (CLI) – Empfohlen

Voraussetzungen

Stellen Sie sicher, dass Sie auf Ihrem Computer über Folgendes verfügen, bevor Sie fortfahren:

- Die Zertifikatsdatei der Root-CA (im folgenden Beispiel referenziert als *root_CA_cert_filename.pem*)
- OpenSSL v1.1.1i oder höher

Um ein CA-Zertifikat im **SNI_ONLY-Modus zu registrieren, verwenden** Sie AWS CLI

1. Registrieren Sie das CA-Zertifikat bei. AWS IoT Geben Sie mit dem Befehl register-ca-certificate den Namen der CA-Zertifikatsdatei ein. Weitere Informationen finden Sie unter register-ca-certificate in der AWS CLI -Befehlsreferenz.

   ```
   aws iot register-ca-certificate \
       --ca-certificate file://root_CA_cert_filename.pem \
       --certificate-mode SNI_ONLY
   ```

   Bei Erfolg gibt dieser Befehl den zurück*certificateId*.

2. Zu diesem Zeitpunkt wurde das CA-Zertifikat registriert, ist AWS IoT aber inaktiv. Das CA-Zertifikat muss aktiv sein, damit Sie Clientzertifikate registrieren können, die von diesem Zertifikat signiert sind.

   In diesem Schritt wird das CA-Zertifikat aktiviert.

   Verwenden Sie Befehl update-certificate wie folgt, um das CA-Zertifikat zu aktivieren. Weitere Informationen finden Sie unter update-certificate in der AWS CLI -Befehlsreferenz.

   ```
   aws iot update-ca-certificate \
       --certificate-id certificateId \
       --new-status ACTIVE
   ```

Verwenden Sie den Befehl describe-ca-certificate, um den Status des CA-Zertifikats anzuzeigen. Weitere Informationen finden Sie unter describe-ca-certificate in der AWS CLI -Befehlsreferenz.

Registrieren Sie ein CA-Zertifikat im **DEFAULT-Modus** (CLI)

Voraussetzungen

Stellen Sie sicher, dass Sie auf Ihrem Computer über Folgendes verfügen, bevor Sie fortfahren:

- Die Zertifikatsdatei der Root-CA (im folgenden Beispiel referenziert als *root_CA_cert_filename.pem*)
- Die private Schlüsseldatei des Root-CA-Zertifikats (im folgenden Beispiel referenziert als *root_CA_key_filename.key*)
- OpenSSL v1.1.1i oder höher

Um ein CA-Zertifikat im **DEFAULT-Modus** zu registrieren, verwenden Sie den AWS CLI

1. Um einen Registrierungscode von zu erhalten AWS IoT, verwenden Sieget-registration-code. Speichern Sie den zurückgegebenen `registrationCode`, um ihn als `Common Name` des Verifizierungszertifikats für private Schlüssel zu verwenden. Weitere Informationen finden sie unter get-registration-code in der AWS CLI -Befehlsreferenz.

   ```
   aws iot get-registration-code
   ```

2. Generieren Sie ein Schlüsselpaar für das Verifizierungszertifikat für private Schlüssel:

   ```
   openssl genrsa -out verification_cert_key_filename.key 2048
   ```

3. Erstellen Sie eine Zertifikatssignierungsanforderung (Certificate Signing Request, CSR) für das Verifizierungszertifikat für private Schlüssel. Geben Sie im Feld `Common Name` des Zertifikats den von get-registration-code zurückgegebenen Wert für `registrationCode` ein.

   ```
   openssl req -new \
       -key verification_cert_key_filename.key \
       -out verification_cert_csr_filename.csr
   ```

   Sie werden aufgefordert, einige Informationen zum Zertifikat einzugeben, z. B. den `Common Name`.

   ```
   You are about to be asked to enter information that will be incorporated
   into your certificate request.
   What you are about to enter is what is called a Distinguished Name or a DN.
   There are quite a few fields but you can leave some blank
   For some fields there will be a default value,
   If you enter '.', the field will be left blank.
   -----
   Country Name (2 letter code) [AU]:
       State or Province Name (full name) []:
   ```

```
    Locality Name (for example, city) []:
    Organization Name (for example, company) []:
    Organizational Unit Name (for example, section) []:
    Common Name (e.g. server FQDN or YOUR name) []:your_registration_code
    Email Address []:

    Please enter the following 'extra' attributes
    to be sent with your certificate request
    A challenge password []:
    An optional company name []:
```

4. Verwenden Sie die CSR, um ein Verifizierungszertifikat für private Schlüssel zu erstellen:

```
openssl x509 -req \
    -in verification_cert_csr_filename.csr \
    -CA root_CA_cert_filename.pem \
    -CAkey root_CA_key_filename.key \
    -CAcreateserial \
    -out verification_cert_filename.pem \
    -days 500 -sha256
```

5. Registrieren Sie das CA-Zertifikat mit AWS IoT. Geben Sie den Dateinamen des CA-Zertifikats und den Dateinamen des Verifizierungszertifikat für private Schlüssel wie folgt an den Befehl register-ca-certificate weiter. Weitere Informationen finden Sie unter register-ca-certificate in der AWS CLI -Befehlsreferenz.

```
aws iot register-ca-certificate \
    --ca-certificate file://root_CA_cert_filename.pem \
    --verification-cert file://verification_cert_filename.pem
```

   Dieser Befehl gibt bei Erfolg den *certificateId* zurück.

6. Zu diesem Zeitpunkt wurde das CA-Zertifikat registriert, ist AWS IoT aber nicht aktiv. Das CA-Zertifikat muss aktiv sein, damit Sie Clientzertifikate registrieren können, die von diesem Zertifikat signiert sind.

   In diesem Schritt wird das CA-Zertifikat aktiviert.

   Verwenden Sie Befehl update-certificate wie folgt, um das CA-Zertifikat zu aktivieren. Weitere Informationen finden Sie unter update-certificate in der AWS CLI -Befehlsreferenz.

```
aws iot update-ca-certificate \
```

```
    --certificate-id certificateId \
    --new-status ACTIVE
```

Verwenden Sie den Befehl describe-ca-certificate, um den Status des CA-Zertifikats anzuzeigen. Weitere Informationen finden Sie unter describe-ca-certificate in der AWS CLI -Befehlsreferenz.

Erstellen Sie ein CA-Verifizierungszertifikat, um das CA-Zertifikat in der Konsole zu registrieren.

> **Note**
>
> Dieses Verfahren wird nur verwendet, wenn Sie ein CA-Zertifikat von der AWS IoT Konsole aus registrieren.
> Wenn Sie dieses Verfahren nicht von der AWS IoT Konsole aus aufgerufen haben, starten Sie den Registrierungsprozess für das CA-Zertifikat in der Konsole unter CA-Zertifikat registrieren.

Stellen Sie sicher, dass Sie auf demselben Computer über Folgendes verfügen, bevor Sie fortfahren:

- Die Zertifikatsdatei der Root-CA (im folgenden Beispiel referenziert als *root_CA_cert_filename.pem*)
- Die private Schlüsseldatei des Root-CA-Zertifikats (im folgenden Beispiel referenziert als *root_CA_key_filename.key*)
- OpenSSL v1.1.1i oder höher

Erstellen Sie ein CA-Verifizierungszertifikat, um Ihr CA-Zertifikat in der Konsole zu registrieren und die Befehlszeilenschnittstelle zu verwenden.

1. Ersetzen Sie *verification_cert_key_filename.key* durch den Namen der Schlüsseldatei für das Verifizierungszertifikat, die Sie erstellen möchten (z. B. **verification_cert.key**). Führen Sie anschließend diesen Befehl aus, um ein Schlüsselpaar für das Verifizierungszertifikat für private Schlüssel zu generieren:

   ```
   openssl genrsa -out verification_cert_key_filename.key 2048
   ```

2. Ersetzen Sie *verification_cert_key_filename.key* durch den Namen der Schlüsseldatei, die Sie in Schritt 1 erstellt haben.

Ersetzen Sie *verification_cert_csr_filename.csr* durch den Namen der Zertifikatsignierungsanforderungsdatei (Certificate Signing Request, CSR), die Sie erstellen möchten. Beispiel, **verification_cert.csr**.

Führen Sie den Befehl aus, um die CSR-Datei zu erstellen.

```
openssl req -new \
    -key verification_cert_key_filename.key \
    -out verification_cert_csr_filename.csr
```

Der Befehl fordert Sie zur Eingabe zusätzlicher Informationen auf, die an späterer Stelle erläutert werden.

3. Kopieren Sie in der AWS IoT Konsole im Container für das Bestätigungszertifikat den Registrierungscode.
4. Im folgenden Beispiel wird gezeigt, welche Informationen der Befehl openssl fordert. Mit Ausnahme des Felds `Common Name` können Sie Ihre eigenen Werte eingeben oder sie leer lassen.

   Fügen Sie in das Feld `Common Name` den Registrierungscode ein, den Sie im vorherigen Schritt kopiert haben.

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
    State or Province Name (full name) []:
    Locality Name (for example, city) []:
    Organization Name (for example, company) []:
    Organizational Unit Name (for example, section) []:
    Common Name (e.g. server FQDN or YOUR name) []:your_registration_code
    Email Address []:

    Please enter the following 'extra' attributes
    to be sent with your certificate request
    A challenge password []:
```

```
    An optional company name []:
```

Danach erstellt der Befehl die CSR-Datei.

5. Ersetzen Sie die *verification_cert_csr_filename.csr* durch die *verification_cert_csr_filename.csr*, die Sie im vorherigen Schritt verwendet haben.

   Ersetzen Sie *root_CA_cert_filename.pem* durch den Namen der CA-Zertifikatsdatei, die Sie registrieren möchten.

   Ersetzen Sie *root_CA_key_filename.key* durch den Namen der privaten Schlüsseldatei des CA-Zertifikats.

   Ersetzen Sie *verification_cert_filename.pem* durch den Namen der Verifizierungszertifikatdatei, die Sie erstellen möchten. Beispiel, **verification_cert.pem**.

   ```
   openssl x509 -req \
       -in verification_cert_csr_filename.csr \
       -CA root_CA_cert_filename.pem \
       -CAkey root_CA_key_filename.key \
       -CAcreateserial \
       -out verification_cert_filename.pem \
       -days 500 -sha256
   ```

6. Nachdem der OpenSSL-Befehl abgeschlossen ist, sollten Sie diese Dateien bereit haben, wenn Sie zur Konsole zurückkehren.

   - Ihre CA-Zertifikatsdatei (wurde im vorherigen Befehl *root_CA_cert_filename.pem* verwendet)
   - Das Bestätigungszertifikat, das Sie im vorherigen Schritt erstellt haben (das im vorherigen Befehl *verification_cert_filename.pem* verwendet wurde)

Deaktivieren eines CA-Zertifikats

Wenn ein Zertifizierungsstellenzertifikat (CA) für die automatische Registrierung von Client-Zertifikaten aktiviert ist, wird das CA-Zertifikat AWS IoT überprüft, um sicherzustellen, dass es sich um die Zertifizierungsstelle handeltACTIVE. Wenn das CA-Zertifikat aktiviert istINACTIVE, kann das Client-Zertifikat AWS IoT nicht registriert werden.

Wenn Sie den Status des CA-Zertifikats auf `INACTIVE` setzen, verhindern Sie, dass neue Clientzertifikate, die von der CA ausgestellt werden, automatisch registriert werden.

> **Note**
>
> Alle registrierten Clientzertifikate, die vom kompromittierten CA-Zertifikat signiert wurden, sind so lange weiterhin aktiv, bis Sie sie jeweils explizit widerrufen.

Deaktivieren eines CA-Zertifikats (Konsole)

Um ein CA-Zertifikat mit dem zu deaktivieren AWS IoT Konsole

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die AWS IoT Konsole.
2. Wählen Sie im linken Navigationsbereich Sichern und dann CAs.
3. Suchen Sie in der Liste der Zertifizierungsstellen diejenige, die Sie deaktivieren möchten, und öffnen Sie das Optionsmenü über das Ellipsensymbol.
4. Wählen Sie im Optionsmenü die Option Deaktivieren.

Die Zertifizierungsstelle sollte in der Liste als Inaktiv angezeigt werden.

> **Note**
>
> Die AWS IoT Konsole bietet keine Möglichkeit, die Zertifikate aufzulisten, die von der Zertifizierungsstelle signiert wurden, die Sie deaktiviert haben. Eine Option zum Auflisten dieser Zertifikate mithilfe der AWS CLI finden Sie unter Deaktivieren eines CA-Zertifikats (CLI).

Deaktivieren eines CA-Zertifikats (CLI)

Das AWS CLI stellt den update-ca-certificateBefehl zum Deaktivieren eines CA-Zertifikats bereit.

```
aws iot update-ca-certificate \
    --certificate-id certificateId \
    --new-status INACTIVE
```

Verwenden Sie den Befehl list-certificates-by-ca, um eine Liste aller registrierten Clientzertifikate zu erhalten, die von der angegebenen CA signiert wurden. Verwenden Sie für jedes Clientzertifikat, das mit dem angegebenen CA-Zertifikat signiert ist, den Befehl update-certificate, um das Clientzertifikat zu widerrufen und dadurch zu verhindern, dass es verwendet wird.

Verwenden Sie den Befehl describe-ca-certificate, um den Status des CA-Zertifikats anzuzeigen.

Erstellen eines Clientzertifikats mit Ihrem CA-Zertifikat

Sie können Ihre eigene Zertifizierungsstelle (Certificate Authority, CA) zum Erstellen von Clientzertifikaten verwenden. Das Client-Zertifikat muss AWS IoT vor der Verwendung registriert werden. Weitere Informationen zu den Registrierungsoptionen für Ihre Clientzertifikate finden Sie unter Registrieren eines Clientzertifikats.

Erstellen eines Clientzertifikats (CLI)

> **Note**
>
> Sie können dieses Verfahren nicht in der AWS IoT Konsole ausführen.

Um ein Client-Zertifikat mit dem zu erstellen AWS CLI

1. Erzeugen Sie ein Schlüsselpaar.

   ```
   openssl genrsa -out device_cert_key_filename.key 2048
   ```

2. Erstellen Sie eine CSR für das Clientzertifikat.

   ```
   openssl req -new \
       -key device_cert_key_filename.key \
       -out device_cert_csr_filename.csr
   ```

   Sie werden zur Eingabe einiger Informationen aufgefordert, wie hier gezeigt:

   ```
   You are about to be asked to enter information that will be incorporated
   into your certificate request.
   What you are about to enter is what is called a Distinguished Name or a DN.
   There are quite a few fields but you can leave some blank
   For some fields there will be a default value,
   ```

```
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
    State or Province Name (full name) []:
    Locality Name (for example, city) []:
    Organization Name (for example, company) []:
    Organizational Unit Name (for example, section) []:
    Common Name (e.g. server FQDN or YOUR name) []:
    Email Address []:

    Please enter the following 'extra' attributes
    to be sent with your certificate request
    A challenge password []:
    An optional company name []:
```

3. Erstellen Sie ein Clientzertifikat aus der CSR.

```
openssl x509 -req \
    -in device_cert_csr_filename.csr \
    -CA root_CA_cert_filename.pem \
    -CAkey root_CA_key_filename.key \
    -CAcreateserial \
    -out device_cert_filename.pem \
    -days 500 -sha256
```

Zu diesem Zeitpunkt wurde das Client-Zertifikat erstellt, aber es wurde noch nicht registriert AWS IoT. Weitere Informationen dazu, wie und wann das Clientzertifikat registriert werden soll, finden Sie unter Registrieren eines Clientzertifikats.

Registrieren eines Clientzertifikats

Client-Zertifikate müssen registriert sein AWS IoT , um die Kommunikation zwischen dem Client und zu ermöglichen AWS IoT. Sie können jedes Client-Zertifikat manuell registrieren, oder Sie können die Client-Zertifikate so konfigurieren, dass sie automatisch registriert werden, wenn der Client AWS IoT zum ersten Mal eine Verbindung herstellt.

Wenn Sie möchten, dass Ihre Clients und Geräte ihre Clientzertifikate registrieren, wenn sie sich zum ersten Mal verbinden, müssen Sie Registrieren eines CA-Zertifikats das Clientzertifikat in den Regionen bei AWS IoT signieren, in denen Sie es verwenden möchten. Die Amazon Root-CA wird automatisch bei registriert AWS IoT.

Kundenzertifikate können nach AWS-Konten Regionen gemeinsam genutzt werden. Die Verfahren in diesen Themen müssen in jedem Konto und jeder Region ausgeführt werden, in der Sie das Clientzertifikat verwenden möchten. Die Registrierung eines Clientzertifikats in einem Konto oder einer Region wird von einem anderen Konto/einer anderen Region nicht automatisch erkannt.

> **Note**
>
> Clients, die das TLS-Protokoll (Transport Layer Security) verwenden, um eine Verbindung mit AWS IoT herzustellen, müssen die SNI-Erweiterung (Server Name Indication) für TLS unterstützen. Weitere Informationen finden Sie unter Transportsicherheit in AWS IoT Core.

Themen

- Manuelles Registrieren eines Clientzertifikats
- Registrieren Sie ein Client-Zertifikat, wenn der Client eine Verbindung herstellt AWS IoT Just-in-Time-Registrierung (JITR)

Manuelles Registrieren eines Clientzertifikats

Sie können ein Client-Zertifikat manuell registrieren, indem Sie die AWS IoT Konsole und verwenden AWS CLI.

Das anzuwendende Registrierungsverfahren hängt davon ab, ob das Zertifikat von AWS-Konto s und Regionen gemeinsam genutzt wird. Die Registrierung eines Clientzertifikats in einem Konto oder einer Region wird von einem anderen Konto/einer anderen Region nicht automatisch erkannt.

Die Verfahren in diesem Thema müssen in jedem Konto und jeder Region ausgeführt werden, in der Sie das Clientzertifikat verwenden möchten. Client-Zertifikate können von AWS-Konto s und Regionen gemeinsam genutzt werden.

Registrieren eines von einer registrierten CA signierten Clientzertifikats (Konsole)

> **Note**
>
> Bevor Sie dieses Verfahren ausführen, stellen Sie sicher, dass Sie über die PEM-Datei des Client-Zertifikats verfügen und dass das Client-Zertifikat von einer Zertifizierungsstelle signiert wurde, bei der Sie sich registriert haben. AWS IoT

Um ein vorhandenes Zertifikat zu registrieren mit AWS IoT mit der Konsole

1. Melden Sie sich bei der AWS Management Console an und öffnen Sie die AWS IoT Konsole.
2. Wählen Sie im Navigationsbereich im Abschnitt Verwalten die Option Sicherheit und anschließend Zertifikate.
3. Wählen Sie auf der Seite Zertifikate im Dialogfeld Zertifikate die Option Zertifikat hinzufügen und dann Zertifikate registrieren.
4. Gehen Sie auf der Seite Zertifikat registrieren im Dialogfeld Hochzuladende Zertifikate wie folgt vor:

   - Wählen Sie Zertifizierungsstelle ist bei AWS IoT registriert.
   - Wählen Sie unter CA-Zertifikat auswählen Ihre Zertifizierungsstelle aus.
     - Wählen Sie Neue Zertifizierungsstelle registrieren aus, um eine neue Zertifizierungsstelle zu registrieren, die nicht bei AWS IoT registriert ist.
     - Lassen Sie CA-Zertifikat auswählen leer, wenn die Amazon-Root-Zertifizierungsstelle Ihre Zertifizierungsstelle ist.
   - Wählen Sie bis zu 10 Zertifikate aus, die Sie hochladen und mit denen Sie sich registrieren möchten AWS IoT.
     - Verwenden Sie die Zertifikatsdateien, die Sie in Create AWS IoT Client-Zertifikate und Erstellen eines Clientzertifikats mit Ihrem CA-Zertifikat erstellt haben.
   - Wählen Sie Aktivieren oder Deaktivieren. Wenn Sie Deaktivieren wählen, erklärt Aktivieren oder Deaktivieren eines Clientzertifikats, wie Sie Ihr Zertifikat nach der Zertifikatsregistrierung aktivieren.
   - Wählen Sie Registrieren aus.

Auf der Seite Zertifikate im Dialogfeld Zertifikate werden nun Ihre registrierten Zertifikate angezeigt.

Registrieren eines von einer nicht registrierten CA signierten Clientzertifikats (Konsole)

> **Note**
>
> Stellen Sie sicher, dass Sie über die PEM-Datei des Clientzertifikats verfügen, bevor Sie dieses Verfahren ausführen.

Um ein vorhandenes Zertifikat zu registrieren bei AWS IoT mit der Konsole

1. Melden Sie sich bei der AWS Management Console an und öffnen Sie die AWS IoT Konsole.
2. Wählen Sie im linken Navigationsbereich Sichern, Zertifikate und anschließend Erstellen aus.
3. Suchen Sie unter Zertifikat erstellen den Eintrag Eigenes Zertifikat verwenden und wählen Sie Erste Schritte.
4. Wählen Sie unter CA auswählen die Option Weiter.
5. Wählen Sie unter Vorhandene Gerätezertifikate registrieren die Option Zertifikate auswählen und wählen Sie bis zu zehn Zertifikatdateien zum Registrieren aus.
6. Schließen Sie das Dateidialogfeld und wählen Sie aus, ob Sie die Clientzertifikate bei der Registrierung aktivieren oder widerrufen möchten.

   Wenn Sie ein Zertifikat bei der Registrierung nicht aktivieren, können Sie unter Aktivieren eines Clientzertifikats (Konsole) nachlesen, wie Sie es zu einem späteren Zeitpunkt aktivieren können.

   Wenn ein Zertifikat bei der Registrierung widerrufen wird, kann es später nicht aktiviert werden.

   Nachdem Sie die zu registrierenden Zertifikatdateien und die nach der Registrierung auszuführenden Aktionen ausgewählt haben, klicken Sie auf Zertifikate registrieren.

Die erfolgreich registrierten Clientzertifikate werden in der Liste der Zertifikate angezeigt.

Registrieren eines von einer registrierten CA signierten Clientzertifikats (CLI)

> **Note**
>
> Stellen Sie sicher, dass Sie über die PEM-Datei der CA und über die PEM-Datei des Clientzertifikats verfügen, bevor Sie dieses Verfahren ausführen. Das Client-Zertifikat muss von einer Zertifizierungsstelle (CA) signiert sein, bei der Sie sich registriert haben AWS IoT.

Verwenden Sie den Befehl register-certificate, um ein Clientzertifikat zu registrieren, ohne es zu aktivieren.

```
aws iot register-certificate \
    --certificate-pem file://device_cert_filename.pem \
    --ca-certificate-pem file://ca_cert_filename.pem
```

Das Client-Zertifikat ist bei registriert AWS IoT, aber es ist noch nicht aktiv. Informationen dazu, wie Sie es zu einem späteren Zeitpunkt aktivieren können, finden Sie unter Aktivieren eines Clientzertifikats (CLI).

Sie können das Clientzertifikat auch aktivieren, wenn Sie es mit dem folgenden Befehl registrieren.

```
aws iot register-certificate \
    --set-as-active \
    --certificate-pem file://device_cert_filename.pem \
    --ca-certificate-pem file://ca_cert_filename.pem
```

Weitere Informationen zur Aktivierung des Zertifikats, sodass es für die Herstellung einer Verbindung verwendet werden kann AWS IoT, finden Sie unter Aktivieren oder Deaktivieren eines Clientzertifikats

Registrieren eines von einer nicht registrierten CA signierten Clientzertifikats (CLI)

> **Note**
>
> Stellen Sie sicher, dass Sie über die PEM-Datei des Zertifikats verfügen, bevor Sie dieses Verfahren ausführen.

Verwenden Sie den Befehl register-certificate-without-ca, um ein Clientzertifikat zu registrieren, ohne es zu aktivieren.

```
aws iot register-certificate-without-ca \
    --certificate-pem file://device_cert_filename.pem
```

Das Client-Zertifikat ist registriert AWS IoT, aber es ist noch nicht aktiv. Informationen dazu, wie Sie es zu einem späteren Zeitpunkt aktivieren können, finden Sie unter Aktivieren eines Clientzertifikats (CLI).

Sie können das Clientzertifikat auch aktivieren, wenn Sie es mit dem folgenden Befehl registrieren.

```
aws iot register-certificate-without-ca \
    --status ACTIVE \
    --certificate-pem file://device_cert_filename.pem
```

Weitere Hinweise zur Aktivierung des Zertifikats, sodass es für die Herstellung einer Verbindung verwendet werden kann AWS IoT, finden Sie unterAktivieren oder Deaktivieren eines Clientzertifikats.

Registrieren Sie ein Client-Zertifikat, wenn der Client eine Verbindung herstellt AWS IoT Just-in-Time-Registrierung (JITR)

Sie können ein CA-Zertifikat so konfigurieren, dass Client-Zertifikate, die es signiert hat, AWS IoT automatisch registriert werden, wenn der Client zum ersten Mal eine Verbindung herstellt. AWS IoT

Um Client-Zertifikate zu registrieren, wenn ein Client AWS IoT zum ersten Mal eine Verbindung herstellt, müssen Sie das CA-Zertifikat für die automatische Registrierung aktivieren und die erste Verbindung des Clients so konfigurieren, dass die erforderlichen Zertifikate bereitgestellt werden.

Konfigurieren eines CA-Zertifikats zur Unterstützung der automatischen Registrierung (Konsole)

Um ein CA-Zertifikat so zu konfigurieren, dass es die automatische Registrierung von Client-Zertifikaten unterstützt, verwenden Sie den AWS IoT Konsole

1. Melden Sie sich bei der AWS Management Console an und öffnen Sie die AWS IoT Konsole.
2. Wählen Sie im linken Navigationsbereich Sichern und dann CAs.
3. Suchen Sie in der Liste der Zertifizierungsstellen diejenige, für die Sie die automatische Registrierung aktivieren möchten, und öffnen Sie das Optionsmenü über das Ellipsensymbol.
4. Wählen Sie im Optionsmenü Automatische Registrierung aktivieren.

> **Note**
>
> Der Status der automatischen Registrierung wird in der Liste der Zertifizierungsstellen nicht angezeigt. Um den Status der automatischen Registrierung einer Zertifizierungsstelle anzuzeigen, müssen Sie die Seite Details der Zertifizierungsstelle öffnen.

Konfigurieren eines CA-Zertifikats zur Unterstützung der automatischen Registrierung (CLI)

Wenn Sie Ihr CA-Zertifikat bereits registriert haben AWS IoT, verwenden Sie den update-ca-certificateBefehl, um das CA-Zertifikat auf festzulegen `autoRegistrationStatus`ENABLE.

```
aws iot update-ca-certificate \
--certificate-id caCertificateId \
--new-auto-registration-status ENABLE
```

Wenn Sie `autoRegistrationStatus` bei der Registrierung des CA-Zertifikats aktivieren möchten, verwenden Sie den Befehl register-ca-certificate.

```
aws iot register-ca-certificate \
--allow-auto-registration  \
--ca-certificate file://root_CA_cert_filename.pem \
--verification-cert file://verification_cert_filename.pem
```

Verwenden Sie den Befehl describe-ca-certificate, um den Status des CA-Zertifikats anzuzeigen.

Konfigurieren der ersten Verbindung durch einen Client für die automatische Registrierung

Wenn ein Client zum ersten Mal versucht, eine Verbindung herzustellen, muss das mit Ihrem CA-Zertifikat signierte Client-Zertifikat während des Transport Layer Security (TLS) -Handshakes auf dem Client vorhanden sein. AWS IoT

Wenn der Client eine Verbindung herstellt AWS IoT, verwenden Sie das Client-Zertifikat, das Sie unter AWS IoT Client-Zertifikate erstellen oder Eigene Client-Zertifikate erstellen erstellt haben. AWS IoT erkennt das CA-Zertifikat als registriertes CA-Zertifikat, registriert das Client-Zertifikat und setzt seinen Status auf`PENDING_ACTIVATION`. Das bedeutet, dass das Clientzertifikat automatisch registriert wurde und auf die Aktivierung wartet. Das Clientzertifikat muss den Status `ACTIVE` aufweisen, damit es zur Verbindung mit AWS IoT verwendet werden kann. Informationen zur Aktivierung eines Clientzertifikats finden Sie unter Aktivieren oder Deaktivieren eines Clientzertifikats.

> **Note**
>
> Sie können Geräte mithilfe der AWS IoT Core Just-in-Time-Registrierung (JITR) bereitstellen, ohne die gesamte Vertrauenskette bei der ersten Verbindung der Geräte an senden zu müssen. AWS IoT Core Die Vorlage des CA-Zertifikats ist optional, aber das Gerät muss die Server Name Indication (SNI) senden, wenn es eine Verbindung herstellt.

Wenn ein Zertifikat AWS IoT automatisch registriert wird oder wenn ein Client ein Zertifikat mit dem `PENDING_ACTIVATION` Status vorlegt, wird eine Nachricht zum folgenden AWS IoT MQTT-Thema veröffentlicht:

`$aws/events/certificates/registered/`*`caCertificateId`*

Dabei ist *`caCertificateId`* die ID des CA-Zertifikats, das das Gerätezertifikat ausgestellt hat.

Die im Topic veröffentlichte Nachricht weist die folgende Struktur auf:

```
{
```

```
        "certificateId": "certificateId",
        "caCertificateId": "caCertificateId",
        "timestamp": timestamp,
        "certificateStatus": "PENDING_ACTIVATION",
        "awsAccountId": "awsAccountId",
        "certificateRegistrationTimestamp": "certificateRegistrationTimestamp"
}
```

Sie können eine Regel zum Beobachten dieses Topics und Ausführen bestimmter Aktionen erstellen. Wir empfehlen, eine Lambda-Regel zu erstellen, mit der sichergestellt wird, dass sich das Clientzertifikat nicht auf einer Zertifikataufhebungsliste (Certificate Revocation List, CRL) befindet, und mit der das Zertifikat aktiviert und eine Richtlinie erstellt und an das Zertifikat angefügt wird. Die Richtlinie bestimmt, auf welche Ressourcen der Client zugreifen kann. Wenn die Richtlinie, die Sie erstellen, die Client-ID von den Verbindungsgeräten benötigt, können Sie die Client-ID () -Funktion der Regel verwenden, um die Client-ID abzurufen. Ein Beispiel für eine Regeldefinition kann wie folgt aussehen:

```
SELECT *,
   clientid() as clientid
from $aws/events/certificates/registered/caCertificateId
```

In diesem Beispiel abonniert die Regel das JITR-Thema `$aws/events/certificates/registered/caCertificateID` und verwendet die Funktion clientid (), um die Client-ID abzurufen. Die Regel hängt dann die Client-ID an die JITR-Nutzlast an. Weitere Hinweise zur clientid () -Funktion der Regel finden Sie unter clientid ().

Weitere Informationen zum Erstellen einer Lambda-Regel, die das Thema `$aws/events/certificates/registered/caCertificateID` überwacht und diese Aktionen ausführt, finden Sie unter Just-in-Time-Registrierung von Clientzertifikaten in AWS IoT

Wenn bei der automatischen Registrierung der Client-Zertifikate ein Fehler oder eine Ausnahme auftritt, AWS IoT sendet es Ereignisse oder Meldungen an Ihre Logs in CloudWatch Logs. Weitere Informationen zur Einrichtung der Logs für Ihr Konto finden Sie in der CloudWatch Amazon-Dokumentation.

Kundenzertifikate verwalten

AWS IoT bietet Funktionen zur Verwaltung von Client-Zertifikaten.

In diesem Thema:

- Aktivieren oder Deaktivieren eines Clientzertifikats
- Anfügen eines Objekts oder einer Richtlinie an ein Clientzertifikat
- Widerrufen eines Clientzertifikats
- Übertragen eines Zertifikats in ein anderes Konto

Aktivieren oder Deaktivieren eines Clientzertifikats

AWS IoT überprüft, ob ein Client-Zertifikat aktiv ist, wenn es eine Verbindung authentifiziert.

Sie können Clientzertifikate erstellen und registrieren, ohne sie zu aktivieren, sodass sie erst verwendet werden können, wenn Sie dies möchten. Sie können auch aktive Clientzertifikate deaktivieren, um sie vorübergehend zu deaktivieren. Und Sie können Clientzertifikate widerrufen, um deren zukünftige Verwendung zu verhindern.

Aktivieren eines Clientzertifikats (Konsole)

Um ein Client-Zertifikat mit dem zu aktivieren AWS IoT Konsole

1. Melden Sie sich bei der AWS Management Console an und öffnen Sie die AWS IoT Konsole.
2. Wählen Sie im linken Navigationsbereich Sichern und dann Zertifikate.
3. Suchen Sie in der Liste der Zertifikate dasjenige, die Sie aktivieren möchten, und öffnen Sie das Optionsmenü über das Ellipsensymbol.
4. Wählen Sie im Optionsmenü die Option Aktivieren.

Das Zertifikat sollte in der Liste der Zertifikate als Aktiv angezeigt werden.

Deaktivieren eines Clientzertifikats (Konsole)

Um ein Client-Zertifikat zu deaktivieren, verwenden Sie den AWS IoT Konsole

1. Melden Sie sich bei der AWS Management Console an und öffnen Sie die AWS IoT Konsole.
2. Wählen Sie im linken Navigationsbereich Sichern und dann Zertifikate.
3. Suchen Sie in der Liste der Zertifikate dasjenige, die Sie deaktivieren möchten, und öffnen Sie das Optionsmenü über das Ellipsensymbol.
4. Wählen Sie im Optionsmenü die Option Deaktivieren.

Das Zertifikat sollte in der Liste der Zertifikate als Inaktiv angezeigt werden.

Aktivieren eines Clientzertifikats (CLI)

Das AWS CLI stellt den update-certificateBefehl zur Aktivierung eines Zertifikats bereit.

```
aws iot update-certificate \
    --certificate-id certificateId \
    --new-status ACTIVE
```

Wenn der Befehl erfolgreich ist, wird der Status des Zertifikats auf `ACTIVE` gesetzt. Führen Sie describe-certificate aus, um den Status des Zertifikats anzuzeigen.

```
aws iot describe-certificate \
    --certificate-id certificateId
```

Deaktivieren eines Clientzertifikats (CLI)

Der AWS CLI stellt den update-certificateBefehl zum Deaktivieren eines Zertifikats bereit.

```
aws iot update-certificate \
    --certificate-id certificateId \
    --new-status INACTIVE
```

Wenn der Befehl erfolgreich ist, wird der Status des Zertifikats auf `INACTIVE` gesetzt. Führen Sie describe-certificate aus, um den Status des Zertifikats anzuzeigen.

```
aws iot describe-certificate \
    --certificate-id certificateId
```

Anfügen eines Objekts oder einer Richtlinie an ein Clientzertifikat

Wenn Sie ein Zertifikat unabhängig von einer AWS IoT Sache erstellen und registrieren, wird es weder über Richtlinien verfügen, die AWS IoT Operationen autorisieren, noch wird es mit einem Objekt verknüpft. AWS IoT In diesem Abschnitt wird beschrieben, wie Sie diese Beziehungen zu einem registrierten Zertifikat hinzufügen.

> **Important**
>
> Um dieses Verfahren ausführen zu können, müssen Sie das Objekt oder die Richtlinie, das/die Sie an das Zertifikat anfügen möchten, bereits erstellt haben.

Das Zertifikat authentifiziert ein Gerät mit, AWS IoT sodass es eine Verbindung herstellen kann. Durch das Anhängen des Zertifikats an eine Objektressource wird die Beziehung zwischen dem Gerät (über das Zertifikat) und der Objektressource hergestellt. Um das Gerät zur Ausführung von AWS IoT Aktionen zu autorisieren, z. B. damit das Gerät eine Verbindung herstellen und Nachrichten veröffentlichen kann, muss dem Zertifikat des Geräts eine entsprechende Richtlinie beigefügt werden.

Anfügen eines Objekts an ein Clientzertifikat (Konsole)

Sie benötigen für dieses Verfahren den Namen des Objekts.

So fügen Sie ein Objekt an ein registriertes Zertifikat an

1. Melden Sie sich bei der AWS Management Console an und öffnen Sie die AWS IoT Konsole.
2. Wählen Sie im linken Navigationsbereich Sicher und dann Zertifikate.
3. Suchen Sie in der Zertifikatsliste das Zertifikat, an das Sie eine Richtlinie anfügen möchten, öffnen Sie das Optionsmenü des Zertifikats über das Ellipsensymbol und wählen Sie Objekt anfügen.
4. Suchen Sie im Pop-up-Fenster den Namen des Objekts, das Sie an das Zertifikat anfügen möchten, markieren Sie das entsprechende Kontrollkästchen und wählen Sie Anfügen.

Das Objekt sollte nun in der Objektliste auf der Detailseite des Zertifikats angezeigt werden.

Anfügen einer Richtlinie an ein Clientzertifikat (Konsole)

Sie benötigen für dieses Verfahren den Namen des Richtlinienobjekts.

So fügen Sie ein Richtlinienobjekt an ein registriertes Zertifikat an

1. Melden Sie sich bei der AWS Management Console an und öffnen Sie die AWS IoT Konsole.
2. Wählen Sie im linken Navigationsbereich Sichern und dann Zertifikate.
3. Suchen Sie in der Zertifikatsliste das Zertifikat, an das Sie eine Richtlinie anfügen möchten, öffnen Sie das Optionsmenü des Zertifikats über das Ellipsensymbol und wählen Sie Richtlinie anfügen.
4. Suchen Sie im Pop-up-Fenster den Namen der Richtlinie, die Sie an das Zertifikat anfügen möchten, markieren Sie das entsprechende Kontrollkästchen und wählen Sie Anfügen.

Das Richtlinienobjekt sollte nun in der Richtlinienliste auf der Detailseite des Zertifikats angezeigt werden.

Anfügen eines Objekts an ein Clientzertifikat (CLI)

Das AWS CLI stellt den attach-thing-principalBefehl zum Anhängen eines Dingobjekts an ein Zertifikat bereit.

```
aws iot attach-thing-principal \
    --principal certificateArn \
    --thing-name thingName
```

Anfügen einer Richtlinie an ein Clientzertifikat (CLI)

Der AWS CLI stellt den attach-policyBefehl zum Anhängen eines Richtlinienobjekts an ein Zertifikat bereit.

```
aws iot attach-policy \
    --target certificateArn \
    --policy-name policyName
```

Widerrufen eines Clientzertifikats

Wenn Sie verdächtige Aktivitäten in einem registrierten Clientzertifikat erkennen, können Sie es widerrufen, damit es nicht noch einmal verwendet werden kann.

> **Note**
>
> Sobald ein Zertifikat gesperrt wurde, kann sein Status nicht mehr geändert werden. Das heißt, dass der Status des Zertifikats nicht in `Active` oder einen anderen Status geändert werden kann.

Widerrufen eines Clientzertifikats (Konsole)

Um ein Client-Zertifikat zu widerrufen, verwenden Sie AWS IoT Konsole

1. Melden Sie sich bei der AWS Management Console an und öffnen Sie die AWS IoT Konsole.
2. Wählen Sie im linken Navigationsbereich Sichern und dann Zertifikate.
3. Suchen Sie in der Liste der Zertifikate dasjenige, die Sie widerrufen möchten, und öffnen Sie das Optionsmenü über das Ellipsensymbol.

4. Wählen Sie im Optionsmenü die Option Widerrufen.

Wenn das Zertifikat erfolgreich widerrufen wurde, wird es in der Liste der Zertifikate als Widerrufen angezeigt.

Widerrufen eines Clientzertifikats (CLI)

Das AWS CLI stellt den update-certificateBefehl zum Widerrufen eines Zertifikats bereit.

```
aws iot update-certificate \
    --certificate-id certificateId \
    --new-status REVOKED
```

Wenn der Befehl erfolgreich ist, wird der Status des Zertifikats auf `REVOKED` gesetzt. Führen Sie describe-certificate aus, um den Status des Zertifikats anzuzeigen.

```
aws iot describe-certificate \
    --certificate-id certificateId
```

Übertragen eines Zertifikats in ein anderes Konto

X.509 Zertifikate, die zu einem gehören, AWS-Konto können auf ein anderes übertragen werden AWS-Konto.

Um ein X.509 Zertifikat von einem zu übertragen AWS-Konto zu einem anderen

1. the section called “Starten einer Zertifikatsübertragung”

   Das Zertifikat muss deaktiviert und von allen Richtlinien und Objekten getrennt werden, bevor die Übertragung gestartet wird.
2. the section called “Annehmen oder Ablehnen einer Zertifikatsübertragung”

   Das empfangende Konto muss das übertragene Zertifikat ausdrücklich akzeptieren oder ablehnen. Nachdem das Empfängerkonto das Zertifikat akzeptiert hat, muss das Zertifikat aktiviert werden, bevor es verwendet werden kann.
3. the section called “Abbrechen einer Zertifikatsübertragung”

   Das ursprüngliche Konto kann eine Übertragung abbrechen, wenn das Zertifikat nicht akzeptiert wurde.

Starten einer Zertifikatsübertragung

Sie können mit der Übertragung eines Zertifikats auf ein anderes beginnen, AWS-Konto indem Sie die AWS IoT Konsole oder das verwenden AWS CLI.

Starten einer Zertifikatsübertragung (Konsole)

Sie benötigen die ID des Zertifikats, das Sie übertragen möchten. um dieses Verfahren abzuschließen.

Führen Sie dieses Verfahren von dem Konto aus, das das zu übertragende Zertifikat enthält.

Um mit der Übertragung eines Zertifikats auf ein anderes zu beginnen AWS-Konto

1. Melden Sie sich bei der AWS Management Console an und öffnen Sie die AWS IoT Konsole.
2. Wählen Sie im linken Navigationsbereich Sichern und dann Zertifikate.

   Wählen Sie das Zertifikat mit dem Status Aktiv oder Inaktiv aus, das Sie übertragen möchten, und öffnen Sie die zugehörige Detailseite.
3. Wenn auf der Seite Details des Zertifikats im Menü Aktionen die Option Deaktivieren verfügbar ist, wählen Sie die Option Deaktivieren, um das Zertifikat zu deaktivieren.
4. Wählen Sie auf der Seite Details des Zertifikats im linken Menü die Option Richtlinien.
5. Wenn auf der Seite Richtlinien des Zertifikats Richtlinien mit dem Zertifikat verknüpft sind, trennen Sie die einzelnen Richtlinien, indem Sie das Optionsmenü der Richtlinie öffnen und Trennen wählen.

   Bevor Sie fortfahren, dürfen mit dem Zertifikat keine Richtlinien verbunden sein.
6. Wählen Sie auf der Seite Richtlinien des Zertifikats im linken Menü die Option Objekte.
7. Wenn auf der Seite Objekte des Zertifikats Objekte mit dem Zertifikat verknüpft sind, trennen Sie die einzelnen Objekte, indem Sie das Optionsmenü des Objekts öffnen und Trennen wählen.

   Bevor Sie fortfahren, dürfen mit dem Zertifikat keine Objekte verbunden sein.
8. Wählen Sie auf der Seite Details des Zertifikats im linken Menü die Option Details.
9. Wählen Sie auf der Seite Details des Zertifikats im Menü Aktionen die Option Übertragung starten, um das Dialogfeld Übertragung starten zu öffnen.
10. Geben Sie im Dialogfeld Übertragung starten die AWS-Konto Nummer des Kontos ein, das das Zertifikat erhalten soll, und optional eine Kurznachricht.
11. Wählen Sie Übertragung starten, um das Zertifikat zu übertragen.

Die Konsole sollte eine Meldung anzeigen, die den Erfolg oder Misserfolg der Übertragung angibt. Wenn die Übertragung gestartet wurde, wird der Status des Zertifikats auf Übertragen aktualisiert.

Starten einer Zertifikatsübertragung (CLI)

Um dieses Verfahren abzuschließen, benötigen Sie das *certificateId* und das *certificateArn* Zertifikat, das Sie übertragen möchten.

Führen Sie dieses Verfahren von dem Konto aus, das das zu übertragende Zertifikat enthält.

Um mit der Übertragung eines Zertifikats auf ein anderes zu beginnen AWS Konto

1. Verwenden Sie den Befehl update-certificate zum Deaktivieren des Zertifikats.

   ```
   aws iot update-certificate --certificate-id certificateId --new-status INACTIVE
   ```

2. Trennen Sie alle Richtlinien.

   1. Verwenden Sie den Befehl list-attached-policies, um die an das Zertifikat angehängten Richtlinien aufzulisten.

      ```
      aws iot list-attached-policies --target certificateArn
      ```

   2. Verwenden Sie für jede angehängte Richtlinie den Befehl detach-policy, um die Richtlinie zu trennen.

      ```
      aws iot detach-policy --target certificateArn --policy-name policy-name
      ```

3. Trennen Sie alle Objekte.

   1. Verwenden Sie den Befehl list-principal-things, um die an das Zertifikat angehängten Objekte aufzulisten.

      ```
      aws iot list-principal-things --principal certificateArn
      ```

   2. Verwenden Sie für jedes angehängte Objekt den Befehl detach-thing-principal, um das Objekt zu trennen.

      ```
      aws iot detach-thing-principal --principal certificateArn --thing-name thing-name
      ```

4. Verwenden Sie den Befehl transfer-certificate, um die Zertifikatsübertragung zu starten.

```
aws iot transfer-certificate --certificate-id certificateId --target-aws-
account account-id
```

Annehmen oder Ablehnen einer Zertifikatsübertragung

Sie können ein Zertifikat, das Ihnen AWS-Konto von einem anderen übertragen wurde, akzeptieren oder ablehnen, AWS-Konto indem Sie die AWS IoT Konsole oder die verwenden AWS CLI.

Annehmen oder Ablehnen einer Zertifikatsübertragung (Konsole)

Sie benötigen die ID des Zertifikats, das in Ihr Konto übertragen wurde, um dieses Verfahren abzuschließen.

Führen Sie dieses Verfahren von dem Konto aus, das das übertragende Zertifikat empfängt.

Um ein Zertifikat zu akzeptieren oder abzulehnen, das auf Ihr AWS-Konto

1. Melden Sie sich bei der AWS Management Console an und öffnen Sie die AWS IoT Konsole.
2. Wählen Sie im linken Navigationsbereich Sichern und dann Zertifikate.

   Wählen Sie das Zertifikat mit dem Status Ausstehende Übertragung aus, das Sie akzeptieren oder ablehnen möchten, und öffnen Sie die zugehörige Detailseite.
3. Auf der Detailseite des Zertifikats im Menü Aktionen,

   - Wählen Sie Übertragung akzeptieren, um das Zertifikat zu akzeptieren.
   - Wählen Sie Übertragung ablehnen, um das Zertifikat abzulehnen.

Annehmen oder Ablehnen einer Zertifikatsübertragung (CLI)

Um dieses Verfahren abzuschließen, benötigen Sie den Namen *certificateId* der Zertifikatsübertragung, die Sie akzeptieren oder ablehnen möchten.

Führen Sie dieses Verfahren von dem Konto aus, das das übertragende Zertifikat empfängt.

Um ein Zertifikat zu akzeptieren oder abzulehnen, das auf Ihr AWS-Konto

1. Verwenden Sie den Befehl accept-certificate-transfer, um das Zertifikat zu akzeptieren.

```
aws iot accept-certificate-transfer --certificate-id certificateId
```

2. Verwenden Sie den Befehl reject-certificate-transfer, um das Zertifikat abzulehnen.

```
aws iot reject-certificate-transfer --certificate-id certificateId
```

Abbrechen einer Zertifikatsübertragung

Sie können eine Zertifikatsübertragung abbrechen, bevor sie akzeptiert wurde, indem Sie die AWS IoT -Konsole oder die AWS CLI verwenden.

Abbrechen einer Zertifikatsübertragung (Konsole)

Sie benötigen die ID der Zertifikatsübertragung, die Sie abbrechen möchten, um dieses Verfahren abzuschließen.

Führen Sie dieses Verfahren von dem Konto aus, das die Zertifikatsübertragung gestartet hat.

So brechen Sie eine Zertifikatsübertragung ab

1. Melden Sie sich bei der AWS Management Console an und öffnen Sie die AWS IoT Konsole.
2. Wählen Sie im linken Navigationsbereich Sichern und dann Zertifikate.

   Wählen Sie das Zertifikat mit dem Status Übertragen aus, dessen Übertragung Sie stornieren möchten, und öffnen Sie das zugehörige Optionsmenü.
3. Wählen Sie im Optionsmenü des Zertifikats die Option Übertragung widerrufen, um die Zertifikatsübertragung abzubrechen.

> **Important**
>
> Achten Sie darauf, die Option Übertragung widerrufen nicht mit der Option Widerrufen zu verwechseln.
> Mit der Option Übertragung widerrufen wird die Zertifikatsübertragung abgebrochen, während die Option Widerrufen das Zertifikat für das AWS IoT unwiderruflich unbrauchbar macht.

Abbrechen einer Zertifikatsübertragung (CLI)

Um dieses Verfahren abzuschließen, benötigen Sie den Namen *certificateId* der Zertifikatsübertragung, die Sie stornieren möchten.

Führen Sie dieses Verfahren von dem Konto aus, das die Zertifikatsübertragung gestartet hat.

Verwenden Sie den Befehl cancel-certificate-transfer, um die Zertifikatsübertragung abzubrechen.

```
aws iot cancel-certificate-transfer --certificate-id certificateId
```

Benutzerdefinierte Validierung von Client-Zertifikaten

AWS IoT Core unterstützt die benutzerdefinierte Überprüfung von Client-Zertifikaten für X.509 Client-Zertifikate, wodurch die Verwaltung der Client-Authentifizierung verbessert wird. Diese Methode zur Zertifikatsvalidierung wird auch als Zertifikatsprüfung vor der Authentifizierung bezeichnet. Dabei bewerten Sie Client-Zertifikate anhand Ihrer eigenen Kriterien (definiert in einer Lambda-Funktion) und widerrufen Client-Zertifikate oder das Zertifikat der Signing Certificate Authority (CA) der Zertifikate, um zu verhindern, dass Clients eine Verbindung herstellen können. AWS IoT Core Sie können beispielsweise Ihre eigenen Zertifikatssperrprüfungen erstellen, die den Status der Zertifikate anhand von Validierungsstellen überprüfen, die Online Certificate Status Protocol (OCSP) oder Certificate Revocation Lists (CRL) Endpunkte unterstützen, und Verbindungen für Clients mit gesperrten Zertifikaten verhindern. Die Kriterien, die zur Bewertung von Client-Zertifikaten verwendet werden, werden in einer Lambda-Funktion (auch bekannt als Lambda vor der Authentifizierung) definiert. Sie müssen die in den Domänenkonfigurationen festgelegten Endpunkte verwenden und der Authentifizierungstyp muss Zertifikat sein. X.509 Darüber hinaus müssen Clients die Erweiterung Server Name Indication (SNI) angeben, wenn sie eine Verbindung herstellen. AWS IoT Core

> **Note**
>
> Diese Funktion wird in den AWS GovCloud (US) Regionen nicht unterstützt.

Die Überprüfung benutzerdefinierter Client-Zertifikate umfasst die folgenden Schritte.

- Schritt 1: Registrieren Sie Ihre X.509 Kundenzertifikate bei AWS IoT Core
- Schritt 2: Erstellen einer Lambda-Funktion
- Schritt 3: Autorisieren AWS IoT um Ihre Lambda-Funktion aufzurufen
- Schritt 4: Legen Sie die Authentifizierungskonfiguration für eine Domain fest

Schritt 1: Registrieren Sie Ihre X.509 Kundenzertifikate bei AWS IoT Core

Falls Sie dies noch nicht getan haben, registrieren und aktivieren Sie Ihre X.509 Client-Zertifikate mit AWS IoT Core. Andernfalls überspringen Sie diesen Schritt und gehen Sie direkt zum nächsten.

Gehen Sie wie folgt vor AWS IoT Core, um Ihre Kundenzertifikate bei zu registrieren und zu aktivieren:

1. Wenn Sie Kundenzertifikate direkt mit erstellen AWS IoT. Diese Client-Zertifikate werden automatisch bei registriert AWS IoT Core.
2. Wenn Sie Ihre eigenen Client-Zertifikate erstellen, folgen Sie diesen Anweisungen, um sie zu registrieren AWS IoT Core.
3. Folgen Sie diesen Anweisungen, um Ihre Client-Zertifikate zu aktivieren.

Schritt 2: Erstellen einer Lambda-Funktion

Sie müssen eine Lambda-Funktion erstellen, die die Zertifikatsverifizierung durchführt und bei jedem Verbindungsversuch eines Clients für den konfigurierten Endpunkt aufgerufen wird. Folgen Sie beim Erstellen dieser Lambda-Funktion den allgemeinen Anweisungen unter Erstellen Sie Ihre erste Lambda-Funktion. Stellen Sie außerdem sicher, dass die Lambda-Funktion die erwarteten Anfrage- und Antwortformate wie folgt einhält:

Beispiel für ein Lambda-Funktionsereignis

```
{
 "connectionMetadata": {
  "id": "string"
 },
 "principalId": "string",
 "serverName": "string",
 "clientCertificateChain": [
  "string",
  "string"
 ]
}
```

`connectionMetadata`

Metadaten oder zusätzliche Informationen im Zusammenhang mit der Verbindung des Clients zu AWS IoT Core.

`principalId`

Die Haupt-ID, die dem Client in der TLS-Verbindung zugeordnet ist.

`serverName`

Die Hostnamenzeichenfolge (Server Name Indication, SNI). AWS IoT Core erfordert, dass Geräte die SNI-Erweiterung an das Transport Layer Security (TLS) -Protokoll senden und die vollständige Endpunktadresse im Feld angeben. `host_name`

`clientCertificateChain`

Das Zeichenkettenarray, das die X.509 Zertifikatskette des Clients darstellt.

Beispiel für eine Antwort auf eine Lambda-Funktion

```
{
 "isAuthenticated": "boolean"
}
```

`isAuthenticated`

Ein boolescher Wert, der angibt, ob die Anfrage authentifiziert ist.

> **Note**
>
> In der Lambda-Antwort `isAuthenticated` muss `true` mit der weiteren Authentifizierung und Autorisierung fortgefahren werden. Andernfalls kann das IoT-Client-Zertifikat deaktiviert und die benutzerdefinierte Authentifizierung mit X.509 Client-Zertifikaten für die weitere Authentifizierung und Autorisierung blockiert werden.

Schritt 3: Autorisieren AWS IoT um Ihre Lambda-Funktion aufzurufen

Nachdem Sie die Lambda-Funktion erstellt haben, müssen Sie mithilfe des AWS IoT CLI-Befehls add-permission die Erlaubnis erteilen, sie aufzurufen. Beachten Sie, dass diese Lambda-Funktion bei jedem Verbindungsversuch zu Ihrem konfigurierten Endpunkt aufgerufen wird. Weitere Informationen finden Sie unter Autorisieren des AWS IoT Aufrufs Ihrer Lambda-Funktion.

Schritt 4: Legen Sie die Authentifizierungskonfiguration für eine Domain fest

Im folgenden Abschnitt wird beschrieben, wie Sie die Authentifizierungskonfiguration für eine benutzerdefinierte Domain mithilfe von festlegen AWS CLI.

Konfiguration des Client-Zertifikats für eine Domain festlegen (CLI)

Wenn Sie keine Domänenkonfiguration haben, verwenden Sie den create-domain-configurationCLI-Befehl, um eine zu erstellen. Wenn Sie bereits über eine Domänenkonfiguration verfügen, verwenden Sie den update-domain-configurationCLI-Befehl, um die Konfiguration des Client-Zertifikats für eine Domain zu aktualisieren. Sie müssen den ARN der Lambda-Funktion hinzufügen, die Sie im vorherigen Schritt erstellt haben.

```
aws iot create-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --authentication-type AWS_X509|CUSTOM_AUTH_X509 \
    --application-protocol SECURE_MQTT|HTTPS \
    --client-certificate-config 'clientCertificateCallbackArn":"arn:aws:lambda:us-
east-2:123456789012:function:my-function:1"}'
```

```
aws iot update-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --authentication-type AWS_X509|CUSTOM_AUTH_X509 \
    --application-protocol SECURE_MQTT|HTTPS \
    --client-certificate-config '{"clientCertificateCallbackArn":"arn:aws:lambda:us-
east-2:123456789012:function:my-function:1"}'
```

`domain-configuration-name`

Der Name der Domänenkonfiguration.

`authentication-type`

Der Authentifizierungstyp der Domänenkonfiguration. Weitere Informationen finden Sie unter Auswahl eines Authentifizierungstyps.

`application-protocol`

Das Anwendungsprotokoll, mit dem Geräte kommunizieren AWS IoT Core. Weitere Informationen finden Sie unter Auswahl eines Anwendungsprotokolls.

`client-certificate-config`

Ein Objekt, das die Konfiguration der Client-Authentifizierung für eine Domäne angibt.

`clientCertificateCallbackArn`

Der Amazon-Ressourcenname (ARN) der Lambda-Funktion, die in der AWS IoT TLS-Schicht aufgerufen wird, wenn eine neue Verbindung hergestellt wird. Um die Client-Authentifizierung für

die Durchführung einer benutzerdefinierten Client-Zertifikatsvalidierung anzupassen, müssen Sie den ARN der Lambda-Funktion hinzufügen, die Sie im vorherigen Schritt erstellt haben.

Weitere Informationen finden Sie in CreateDomainConfigurationund in UpdateDomainConfigurationder AWS IoT API-Referenz. Weitere Informationen zu Domänenkonfigurationen finden Sie unter Domänenkonfigurationen.

IAM-Benutzer, -Gruppen und -Rollen

IAM-Benutzer, -Gruppen und -Rollen sind die Standardmechanismen zum Verwalten von Identitäts- und Authentifizierungsmethoden in AWS. Sie können sie verwenden, um mithilfe des AWS SDK und eine Verbindung zu AWS IoT HTTP-Schnittstellen herzustellen AWS CLI.

Mit IAM-Rollen können AWS IoT Sie auch in Ihrem Namen auf andere AWS Ressourcen in Ihrem Konto zugreifen. Wenn Sie beispielsweise möchten, dass ein Gerät seinen Status in einer DynamoDB-Tabelle veröffentlicht, ermöglichen AWS IoT IAM-Rollen die Interaktion mit Amazon DynamoDB. Weitere Informationen finden Sie unter IAM-Rollen.

AWS IoT Authentifiziert bei Message-Broker-Verbindungen über HTTP Benutzer, Gruppen und Rollen mithilfe des Signaturprozesses von Signature Version 4. Weitere Informationen finden Sie unter AWS API-Anfragen signieren.

Wenn Sie Anfragen mithilfe von Abfrageparametern mit temporären Sicherheitsanmeldeinformationen authentifizieren, die vom AWS Security Token Service (AWS STS) bereitgestellt werden, sollten Sie bei der Berechnung der Signatur nichts `X-Amz-Security-Token` in die kanonische Abfragezeichenfolge einbeziehen. Hängen Sie stattdessen `X-Amz-Security-Token` als Abfrageparameter an, nachdem die Signatur berechnet wurde. Dies unterscheidet sich von einigen anderen AWS Diensten, bei denen das Sicherheitstoken Teil der kanonischen Anfrage sein muss. Weitere Informationen finden Sie unter Signieren von Anfragen mit temporären Sicherheitsanmeldedaten.

> **Note**
>
> Die AWS IoT Device SDKs behandeln dieses Signaturverhalten automatisch. Wenn Sie benutzerdefinierten Signaturcode implementieren, finden Sie weitere Informationen in der SDK-Quelle:

- AWS IoT-Geräte-SDK für Python v2 — siehe `websockets_with_default_aws_signing()`, welche Sets `omit_session_token=True`
- AWS IoT-Geräte-SDK SDK for Java v2 — siehe `setOmitSessionToken(true)`

Wenn Sie AWS Signature Version 4 mit verwenden AWS IoT, müssen Kunden in ihrer TLS-Implementierung Folgendes unterstützen:

- TLS 1.2
- SHA-256 Überprüfung der Signatur des RSA-Zertifikats
- Eine der Cipher-Suiten aus dem Support-Abschnitt der TLS Cipher Suite.

Weitere Informationen finden Sie unter Identitäts- und Zugriffsmanagement für AWS IoT.

Amazon-Cognito-Identitäten

Amazon Cognito Identity ermöglicht es Ihnen, temporäre AWS Anmeldeinformationen mit eingeschränkten Rechten für die Verwendung in Mobil- und Webanwendungen zu erstellen. Wenn Sie Amazon Cognito Identity verwenden, erstellen Sie Identitätspools, die eindeutige Identitäten für Ihre Benutzer erstellen, und authentifizieren Sie sie bei Identitätsanbietern wie Login with Amazon, Facebook und Google. Sie können auch Amazon-Cognito-Identitäten mit Ihren eigenen vom Entwickler authentifizierten Identitäten verwenden. Weitere Informationen finden Sie unter Amazon Cognito Identity.

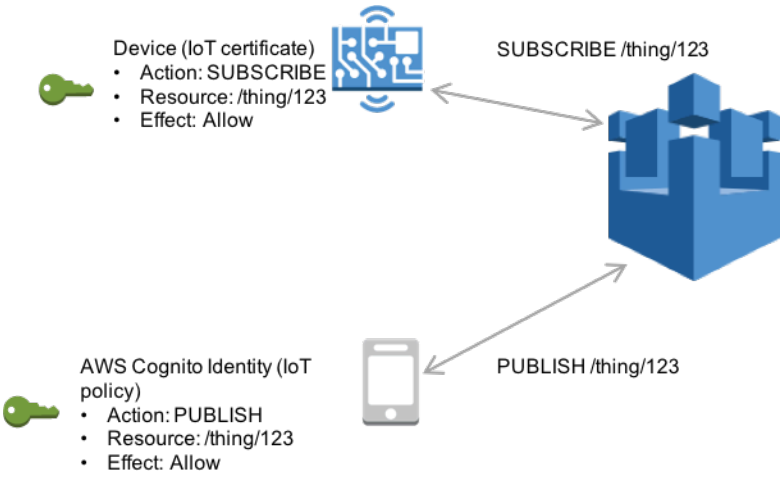
Um Amazon Cognito Identity zu verwenden, definieren Sie einen Amazon Cognito Cognito-Identitätspool, der einer IAM-Rolle zugeordnet ist. Die IAM-Rolle ist mit einer IAM-Richtlinie verknüpft, die Identitäten aus Ihrem Identitätspool den Zugriff auf Ressourcen wie Anrufdienste gewährt. AWS AWS

Amazon Cognito Identity erstellt nicht authentifizierte und authentifizierte Identitäten. Nicht authentifizierte Identitäten werden für Gastbenutzer in einer mobilen oder Webanwendung verwendet, die die App ohne Anmeldung nutzen möchten. Nicht authentifizierten Benutzern werden nur die Berechtigungen gewährt, die in der dem Identitätspool zugeordneten IAM-Richtlinie angegeben sind.

Wenn Sie authentifizierte Identitäten verwenden, müssen Sie zusätzlich zu der dem Identitätspool angehängten IAM-Richtlinie eine Richtlinie an eine Amazon Cognito AWS IoT Cognito-Identität

anhängen. Um eine AWS IoT Richtlinie anzuhängen, verwenden Sie die AttachPolicyAPI und erteilen Sie einem einzelnen Benutzer Ihrer Anwendung Berechtigungen. AWS IoT Sie können die AWS IoT Richtlinie verwenden, um spezifische Berechtigungen für bestimmte Kunden und deren Geräte zuzuweisen.

Authentifizierte und nicht authentifizierte Benutzer sind unterschiedliche Identitätstypen. Wenn Sie der Amazon Cognito Identity keine AWS IoT Richtlinie beifügen, schlägt ein authentifizierter Benutzer die Autorisierung fehl AWS IoT und hat keinen Zugriff auf AWS IoT Ressourcen und Aktionen. Weitere Informationen zum Erstellen von Richtlinien für Amazon-Cognito-Identitäten finden Sie unter Publish/Subscribe Beispiele für Richtlinien und Autorisierung mit Amazon-Cognito-Identitäten.



Benutzerspezifische Authentifizierung und Autorisierung

AWS IoT Core ermöglicht es Ihnen, benutzerdefinierte Autorisierer zu definieren, sodass Sie Ihre eigene Client-Authentifizierung und -Autorisierung verwalten können. Dies ist nützlich, wenn Sie andere Authentifizierungsmechanismen als die verwenden müssen, die von Haus aus unterstützt werden. AWS IoT Core (Weitere Informationen zu den nativ unterstützten Mechanismen finden Sie unter the section called “Client-Authentifizierung”.)

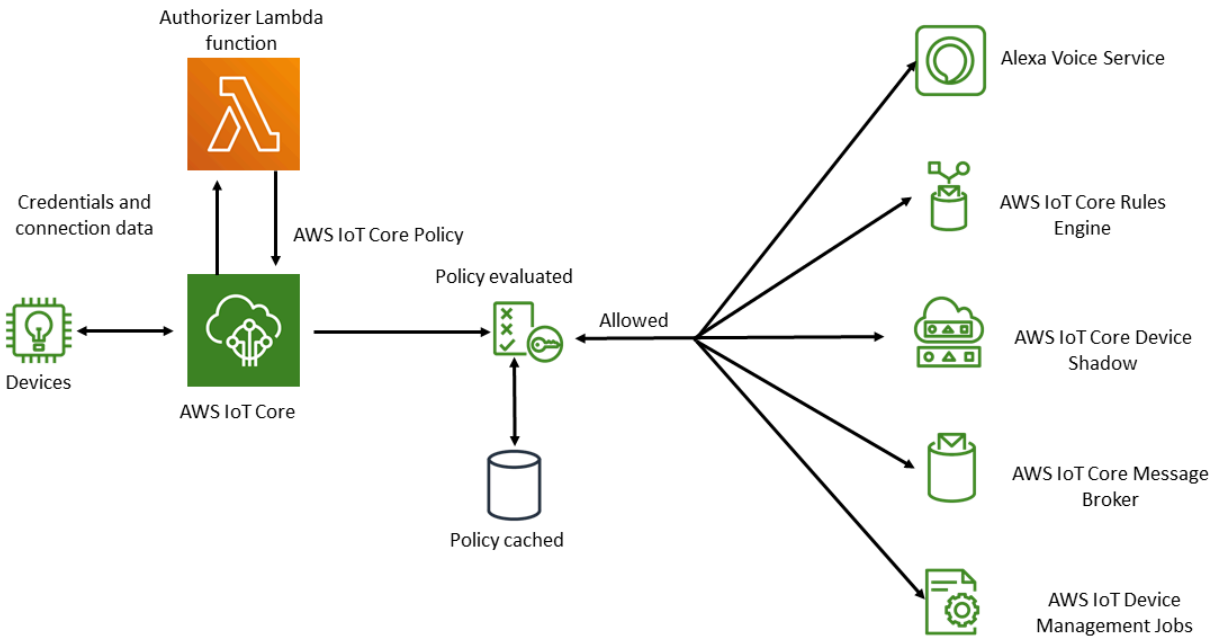
Wenn Sie beispielsweise vorhandene Geräte vor Ort zu migrieren AWS IoT Core und diese Geräte ein benutzerdefiniertes Bearer-Token oder einen MQTT-Benutzernamen und ein Passwort zur Authentifizierung verwenden, können Sie sie zu migrieren, AWS IoT Core ohne ihnen neue Identitäten bereitstellen zu müssen. Sie können die benutzerdefinierte Authentifizierung mit jedem der unterstützten Kommunikationsprotokolle verwenden. AWS IoT Core Weitere Informationen zu von AWS IoT Core unterstützten Protokollen finden Sie unter the section called “Gerätekommunikationsprotokolle”.

Themen

- Grundlegendes zum Workflow für die benutzerdefinierte Authentifizierung
- Benutzerdefinierte Autorisierer (CLI) erstellen und verwalten
- Benutzerdefinierte Authentifizierung mit Client-Zertifikaten X.509
- Verbindung herstellen zu AWS IoT Core mithilfe der benutzerdefinierten Authentifizierung
- Fehlerbehebung für Ihre Genehmiger

Grundlegendes zum Workflow für die benutzerdefinierte Authentifizierung

Mit der benutzerdefinierten Authentifizierung können Sie festlegen, wie Cliente mithilfe von Genehmigerressourcen authentifiziert und autorisiert werden. Jeder Autorisierer enthält einen Verweis auf eine vom Kunden verwaltete Lambda-Funktion, einen optionalen öffentlichen Schlüssel zur Überprüfung der Geräteanmeldedaten und zusätzliche Konfigurationsinformationen. Das folgende Diagramm veranschaulicht den Autorisierungsablauf für die benutzerdefinierte Authentifizierung in. AWS IoT Core



AWS IoT Core benutzerdefinierter Authentifizierungs- und Autorisierungs-Workflow

In der folgenden Liste werden die einzelnen Schritte des benutzerdefinierten Authentifizierungs- und Autorisierungsworkflows erläutert.

1. Ein Gerät stellt über einen der unterstützten Geräte eine Verbindung zum AWS IoT Core Datenendpunkt eines Kunden herthe section called “Gerätekommunikationsprotokolle”. Das Gerät übergibt Anmeldeinformationen entweder in den Header-Feldern oder Abfrageparametern der Anfrage (für die WebSockets Protokolle HTTP Publish oder MQTT over) oder in das Feld für den Benutzernamen und das Passwort der MQTT CONNECT-Nachricht (für die Protokolle MQTT und MQTT over). WebSockets
2. AWS IoT Core prüft auf eine von zwei Bedingungen:
   - Die eingehende Anforderung gibt einen Genehmiger an.
   - Für den AWS IoT Core Datenendpunkt, der die Anfrage empfängt, ist ein Standardautorisierer dafür konfiguriert.

   Wenn auf AWS IoT Core eine dieser Arten ein Autorisierer gefunden wird, wird die dem Autorisierer zugeordnete Lambda-Funktion AWS IoT Core ausgelöst.
3. (Optional) Wenn Sie die Tokensignatur aktiviert haben, AWS IoT Core validiert die Anforderungssignatur mithilfe des im Autorisierer gespeicherten öffentlichen Schlüssels, bevor die Lambda-Funktion ausgelöst wird. Wenn die Validierung fehlschlägt, stoppt AWS IoT Core die Anforderung, ohne die Lambda-Funktion aufzurufen.
4. Die Lambda-Funktion empfängt die Anmeldeinformationen und Verbindungsmetadaten in der Anforderung und trifft eine Authentifizierungsentscheidung.
5. Die Lambda-Funktion gibt die Ergebnisse der Authentifizierungsentscheidung und ein AWS IoT Core Richtliniendokument zurück, das festlegt, welche Aktionen in der Verbindung zulässig sind. Die Lambda-Funktion gibt auch Informationen zurück, die angeben, wie oft die Anmeldeinformationen in der Anfrage AWS IoT Core erneut validiert werden, indem die Lambda-Funktion aufgerufen wird.
6. AWS IoT Core bewertet die Aktivität auf der Verbindung anhand der Richtlinie, die sie von der Lambda-Funktion erhalten hat.
7. Nachdem die Verbindung hergestellt wurde und Ihr benutzerdefinierter Authorizer Lambda zum ersten Mal aufgerufen wurde, kann der nächste Aufruf bei inaktiven Verbindungen ohne MQTT-Operationen um bis zu 5 Minuten verzögert werden. Danach folgen nachfolgende Aufrufe dem Aktualisierungsintervall in Ihrem benutzerdefinierten Authorizer Lambda. Dieser Ansatz kann übermäßige Aufrufe verhindern, die Ihr Lambda-Parallelitätslimit überschreiten könnten. AWS-Konto

Überlegungen zur Skalierung

Da eine Lambda-Funktion die Authentifizierung und Autorisierung für Ihren Genehmiger abwickelt, unterliegt die Funktion den Lambda-Preis- und Servicebeschränkungen, z. B. der Rate der gleichzeitigen Ausführung. Weitere Informationen zu den Preisen für Lambda finden Sie unter Lambda-Preise. Sie können die Belastung Ihrer Lambda-Funktion verwalten, indem Sie die `refreshAfterInSeconds`- und `disconnectAfterInSeconds`-Parameter in Ihrer Lambda-Funktionsantwort anpassen. Weitere Informationen über den Inhalt Ihrer Lambda-Funktionsantwort finden Sie unter the section called “Definieren Ihrer Lambda-Funktion”.

> **Note**
>
> Wenn Sie die Signierung aktiviert lassen, können Sie verhindern, dass Ihr Lambda übermäßig oft durch unbekannte Clients ausgelöst wird. Bedenken Sie dies, bevor Sie die Signierung in Ihrem Genehmiger deaktivieren.

> **Note**
>
> Das Timeout-Limit der Lambda-Funktion für den benutzerdefinierten Genehmiger beträgt 5 Sekunden.

Benutzerdefinierte Autorisierer (CLI) erstellen und verwalten

AWS IoT Core implementiert benutzerdefinierte Authentifizierungs- und Autorisierungsschemata mithilfe benutzerdefinierter Autorisierer. Ein benutzerdefinierter Autorisierer ist eine AWS IoT Core Ressource, die Ihnen die Flexibilität bietet, Regeln und Richtlinien auf der Grundlage Ihrer spezifischen Anforderungen zu definieren und zu implementieren. Informationen zum Erstellen eines benutzerdefinierten Autorisierers mit schrittweisen Anweisungen finden Sie unter Tutorial: Erstellen eines benutzerdefinierten Autorisierers für. AWS IoT Core

Jeder Genehmiger umfasst folgende Komponenten:

- Name: Eine eindeutige benutzerdefinierte Zeichenfolge, in der der Genehmiger identifiziert wird.
- ARN der Lambda-Funktion: Der Amazon-Ressourcenname (ARN) der Lambda-Funktion, die die Autorisierungs- und Authentifizierungslogik implementiert.

- Token-Schlüsselname: Der Schlüsselname, der verwendet wird, um das Token aus den HTTP-Headern, Abfrageparametern oder dem MQTT-CONNECT-Benutzernamen zu extrahieren, um die Signaturvalidierung durchzuführen. Dieser Wert ist erforderlich, wenn in Ihrem Genehmiger die Signierung aktiviert ist.
- Markierung „Signieren deaktiviert“ (optional): Ein boolescher Wert, der angibt, ob die Signaturanforderung für Anmeldeinformationen deaktiviert werden soll. Dies ist nützlich für Szenarien, in denen das Signieren der Anmeldeinformationen keinen Sinn macht, z. B. bei Authentifizierungsschemata, die MQTT-Benutzernamen und -Passwörter verwenden. Der Standardwert ist `false`, also ist das Signieren standardmäßig aktiviert.
- Öffentlicher Schlüssel zur Tokensignierung: Der öffentliche Schlüssel, den AWS IoT Core zur Validierung der Tokensignatur verwendet. Die Mindestlänge beträgt 2.048 Bit. Dieser Wert ist erforderlich, wenn in Ihrem Genehmiger die Signierung aktiviert ist.

Lambda berechnet Ihnen die Anzahl der Ausführungen Ihrer Lambda-Funktion und die Dauer der Ausführung des Codes in Ihrer Funktion. Weitere Informationen zu den Preisen für Lambda finden Sie unter Lambda-Preise. Weitere Informationen zum Erstellen von Lambda-Funktionen finden Sie im Lambda-Entwicklerhandbuch.

> **Note**
>
> Wenn Sie die Signierung aktiviert lassen, können Sie verhindern, dass Ihr Lambda übermäßig oft durch unbekannte Clients ausgelöst wird. Bedenken Sie dies, bevor Sie die Signierung in Ihrem Genehmiger deaktivieren.

> **Note**
>
> Das Timeout-Limit der Lambda-Funktion für den benutzerdefinierten Genehmiger beträgt 5 Sekunden.

In diesem Kapitel:

- Definieren Ihrer Lambda-Funktion
- Erstellen eines Genehmigers
- Autorisieren AWS IoT um Ihre Lambda-Funktion aufzurufen
- Testen Ihrer Genehmiger

- Verwalten von benutzerdefinierten Genehmigern

Definieren Ihrer Lambda-Funktion

Wenn AWS IoT Core Sie Ihren Autorisierer aufrufen, löst er das dem Autorisierer zugeordnete Lambda mit einem Ereignis aus, das das folgende JSON-Objekt enthält. Das JSON-Beispielobjekt enthält alle möglichen Felder. Felder, die für die Verbindungsanforderung nicht relevant sind, sind nicht enthalten.

```
{
    "token" :"aToken",
    "signatureVerified": Boolean, // Indicates whether the device gateway has validated
 the signature.
    "protocols": ["tls", "http", "mqtt"], // Indicates which protocols to expect for
 the request.
    "protocolData": {
        "tls" : {
            "serverName": "serverName" // The server name indication (SNI) host_name
 string.
        },
        "http": {
            "headers": {
                "#{name}": "#{value}"
            },
            "queryString": "?#{name}=#{value}"
        },
        "mqtt": {
            "username": "myUserName",
            "password": "myPassword", // A base64-encoded string.
            "clientId": "myClientId" // Included in the event only when the device
 sends the value.
        }
    },
    "connectionMetadata": {
        "id": UUID // The connection ID. You can use this for logging.
    },
}
```

Die Lambda-Funktion sollte diese Informationen verwenden, um die eingehende Verbindung zu authentifizieren und zu entscheiden, welche Aktionen in der Verbindung zulässig sind. Die Funktion sollte eine Antwort senden, die die folgenden Werte enthält.

- `isAuthenticated`: Ein boolescher Wert, der angibt, ob die Anforderung authentifiziert wurde.
- `principalId`: Eine alphanumerische Zeichenfolge, die als Kennung für das Token dient, das von der benutzerdefinierten Autorisierungsanforderung gesendet wurde. Der Wert muss eine alphanumerische Zeichenfolge mit mindestens einem und nicht mehr als 128 Zeichen sein und dem folgenden regulären Ausdrucksmuster (Regex) entsprechen: `([a-zA-Z0-9]){1,128}`. Sonderzeichen, die nicht alphanumerisch sind, dürfen nicht zusammen mit dem in verwendet werden. `principalId` AWS IoT Core Informationen darüber, ob nicht-alphanumerische Sonderzeichen für die zulässig sind, finden Sie in der Dokumentation zu anderen AWS Diensten. `principalId`
- `policyDocuments`: Eine Liste von JSON-formatted AWS IoT Core Richtliniendokumenten Weitere Informationen zum Erstellen von AWS IoT Core Richtlinien finden Sie unter. the section called “AWS IoT Core Richtlinien” Die maximale Anzahl von Richtliniendokumenten ist 10. Jedes Richtliniendokument darf maximal 2048 Zeichen enthalten.
- `disconnectAfterInSeconds`: Eine Ganzzahl, die die maximale Dauer der Verbindung zum AWS IoT Core -Gateway angibt (in Sekunden). Der Mindestwert ist 300 Sekunden und der Höchstwert 86 400 Sekunden. Der Standardwert ist 86.400.

  > **Note**
  >
  > Der Wert von `disconnectAfterInSeconds` (von der Lambda-Funktion zurückgegeben) wird festgelegt, wenn die Verbindung hergestellt wird. Dieser Wert kann bei nachfolgenden Lambda-Aufrufen zur Richtlinienaktualisierung nicht geändert werden.

- `refreshAfterInSeconds`: Eine Ganzzahl, die das Intervall zwischen Richtlinienaktualisierungen angibt. Wenn dieses Intervall abläuft, ruft AWS IoT Core die Lambda-Funktion auf, um Richtlinienaktualisierungen zu ermöglichen. Der Mindestwert ist 300 Sekunden und der Höchstwert 86 400 Sekunden.

Das folgende JSON-Objekt enthält ein Beispiel für eine Antwort, die Ihre Lambda-Funktion senden kann.

```
{ "isAuthenticated":true, //A Boolean that determines whether
client can connect. "principalId": "xxxxxxxx",  //A string that
identifies the connection in logs. "disconnectAfterInSeconds": 86400,
  "refreshAfterInSeconds": 300,   "policyDocuments": [       {
 "Version": "2012-10-17",         "Statement": [           {
```

```
    "Action": "iot:Publish",             "Effect": "Allow",
      "Resource": "arn:aws:iot:us-east-1:<your_aws_account_id>:topic/
customauthtesting"           }         ]       }     ] }
```

Der `policyDocument` Wert muss ein AWS IoT Core gültiges Richtliniendokument enthalten. Weitere Informationen zu AWS IoT Core Richtlinien finden Sie unterthe section called “AWS IoT Core Richtlinien”. In MQTT über TLS und MQTT über WebSockets Verbindungen wird diese Richtlinie für das im Feldwert angegebene Intervall AWS IoT Core zwischengespeichert. `refreshAfterInSeconds` Bei HTTP-Verbindungen wird die Lambda-Funktion für jede Autorisierungsanforderung aufgerufen, es sei denn, Ihr Gerät verwendet persistente HTTP-Verbindungen (auch HTTP-Keep-Alive oder HTTP-Verbindungswiederverwendung genannt). Sie können bei der Konfiguration des Genehmigers wählen, ob Sie das Caching aktivieren möchten. AWS IoT Core Autorisiert während dieses Intervalls Aktionen in einer bestehenden Verbindung gegen diese zwischengespeicherte Richtlinie, ohne dass Ihre Lambda-Funktion erneut ausgelöst wird. Wenn bei der benutzerdefinierten Authentifizierung Fehler auftreten, AWS IoT Core wird die Verbindung beendet. AWS IoT Core beendet die Verbindung auch, wenn sie länger als der im Parameter angegebene Wert geöffnet war. `disconnectAfterInSeconds`

Im Folgenden finden JavaScript Sie eine Node.js Lambda-Beispielfunktion, die in der MQTT Connect-Nachricht nach einem Passwort mit dem Wert von sucht `test` und eine Richtlinie zurückgibt, die die Erlaubnis erteilt, eine Verbindung AWS IoT Core mit einem Client herzustellen `myClientName` und zu einem Thema zu veröffentlichen, das denselben Clientnamen enthält. Wenn es das erwartete Passwort nicht findet, gibt es eine Richtlinie zurück, die diese beiden Aktionen verweigert.

```
// A simple Lambda function for an authorizer. It demonstrates
// how to parse an MQTT password and generate a response.

exports.handler = function(event, context, callback) {
    var uname = event.protocolData.mqtt.username;
    var pwd = event.protocolData.mqtt.password;
    var buff = new Buffer(pwd, 'base64');
    var passwd = buff.toString('ascii');
    switch (passwd) {
        case 'test':
            callback(null, generateAuthResponse(passwd, 'Allow'));
            break;
        default:
            callback(null, generateAuthResponse(passwd, 'Deny'));
    }
};
```

```
// Helper function to generate the authorization response.
var generateAuthResponse = function(token, effect) {
    var authResponse = {};
    authResponse.isAuthenticated = true;
    authResponse.principalId = 'TEST123';

    var policyDocument = {};
    policyDocument.Version = '2012-10-17';
    policyDocument.Statement = [];
    var publishStatement = {};
    var connectStatement = {};
    connectStatement.Action = ["iot:Connect"];
    connectStatement.Effect = effect;
    connectStatement.Resource = ["arn:aws:iot:us-east-1:123456789012:client/
myClientName"];
    publishStatement.Action = ["iot:Publish"];
    publishStatement.Effect = effect;
    publishStatement.Resource = ["arn:aws:iot:us-east-1:123456789012:topic/telemetry/
myClientName"];
    policyDocument.Statement[0] = connectStatement;
    policyDocument.Statement[1] = publishStatement;
    authResponse.policyDocuments = [policyDocument];
    authResponse.disconnectAfterInSeconds = 3600;
    authResponse.refreshAfterInSeconds = 300;

    return authResponse;
}
```

Die vorhergehende Lambda-Funktion gibt das folgende JSON zurück, wenn sie das erwartete Passwort von `test` in der MQTT-Connect-Nachricht empfängt. Die Werte der `password`- und `principalId`-Eigenschaften sind die Werte aus der MQTT-Connect-Nachricht.

```
{
  "password": "password",
  "isAuthenticated": true,
  "principalId": "principalId",
  "policyDocuments": [
    {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Action": "iot:Connect",
```

```
          "Effect": "Allow",
          "Resource": "*"
        },
        {
          "Action": "iot:Publish",
          "Effect": "Allow",
          "Resource": "arn:aws:iot:region:accountId:topic/telemetry/${iot:ClientId}"
        },
        {
          "Action": "iot:Subscribe",
          "Effect": "Allow",
          "Resource": "arn:aws:iot:region:accountId:topicfilter/telemetry/
${iot:ClientId}"
        },
        {
          "Action": "iot:Receive",
          "Effect": "Allow",
          "Resource": "arn:aws:iot:region:accountId:topic/telemetry/${iot:ClientId}"
        }
      ]
    }
  ],
  "disconnectAfterInSeconds": 3600,
  "refreshAfterInSeconds": 300
}
```

Erstellen eines Genehmigers

Sie können mithilfe der API einen Autorisierer erstellen. CreateAuthorizer Das folgende Beispiel beschreibt den Befehl.

```
aws iot create-authorizer
--authorizer-name MyAuthorizer
--authorizer-function-arn arn:aws:lambda:us-
west-2:<account_id>:function:MyAuthorizerFunction  //The ARN of the Lambda function.
[--token-key-name MyAuthorizerToken //The key used to extract the token from headers.
[--token-signing-public-keys FirstKey=
 "-----BEGIN PUBLIC KEY-----
  [...insert your public key here...]
  -----END PUBLIC KEY-----"
[--status ACTIVE]
[--tags <value>]
[--signing-disabled | --no-signing-disabled]
```

Sie können den `signing-disabled`-Parameter verwenden, um die Signaturvalidierung für jeden Aufruf Ihres Genehmigers zu deaktivieren. Es wird ausdrücklich empfohlen, dass die Signierung nur wenn unbedingt notwendig zu deaktivieren. Die Signaturvalidierung schützt Sie vor übermäßigen Aufrufen Ihrer Lambda-Funktion von unbekannten Geräten. Sie können den `signing-disabled`-Status eines Genehmigers nicht mehr ändern, nachdem Sie ihn erstellt haben. Zum Ändern dieses Verhaltens müssen Sie einen anderen benutzerdefinierten Genehmiger mit einem anderen Wert für den `signing-disabled`-Parameter erstellen.

Die Werte für die `tokenKeyName`- und `tokenSigningPublicKeys`-Parameter sind optional, wenn Sie das Signieren deaktiviert haben. Sie sind jedoch erforderlich, wenn das Signieren aktiviert ist.

Nachdem Sie Ihre Lambda-Funktion und den benutzerdefinierten Autorisierer erstellt haben, müssen Sie dem AWS IoT Core Dienst ausdrücklich die Berechtigung erteilen, die Funktion in Ihrem Namen aufzurufen. Sie können dies mit dem folgenden Befehl tun.

> **Note**
>
> Der Standard-IoT-Endpunkt unterstützt möglicherweise nicht die Verwendung benutzerdefinierter Autorisierer mit Lambda-Funktionen. Stattdessen können Sie Domänenkonfigurationen verwenden, um einen neuen Endpunkt zu definieren und diesen Endpunkt dann für den benutzerdefinierten Autorisierer anzugeben.

```
aws lambda add-permission --function-name <lambda_function_name>
--principal iot.amazonaws.com --source-arn <authorizer_arn>
--statement-id Id-123 --action "lambda:InvokeFunction"
```

Autorisieren AWS IoT um Ihre Lambda-Funktion aufzurufen

In diesem Abschnitt erteilen Sie der benutzerdefinierten Autorisierungsressource, die Sie gerade erstellt haben, die Erlaubnis, die Lambda-Funktion auszuführen. Um die Berechtigung zu erteilen, können Sie den CLI-Befehl add-permission verwenden.

Erteilen Sie Ihrer Lambda-Funktion die Erlaubnis mit dem AWS CLI

1. Geben Sie nach der Eingabe Ihrer Werte den folgenden Befehl ein. Beachten Sie, dass der Wert `statement-id` eindeutig sein muss. *`Id-1234`*Ersetzen Sie es durch den exakten Wert, den Sie haben, andernfalls wird möglicherweise eine `ResourceConflictException` Fehlermeldung angezeigt.

```
aws lambda add-permission  \
--function-name "custom-auth-function" \
--principal "iot.amazonaws.com" \
--action "lambda:InvokeFunction" \
--statement-id "Id-1234" \
--source-arn authorizerArn
```

2. Wenn der Befehl erfolgreich ist, gibt er eine Berechtigungsanweisung zurück, wie in diesem Beispiel. Sie können mit dem nächsten Abschnitt fortfahren, um den benutzerdefinierten Autorisierer zu testen.

```
{
    "Statement": "{\"Sid\":\"Id-1234\",\"Effect\":\"Allow\",\"Principal
\":{\"Service\":\"iot.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction
\",\"Resource\":\"arn:aws:lambda:Region:57EXAMPLE833:function:custom-
auth-function\",\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":
\"arn:aws:lambda:Region:57EXAMPLE833:function:custom-auth-function\"}}}"
}
```

   Wenn der Befehl nicht erfolgreich ist, wird ein Fehler zurückgegeben, wie in diesem Beispiel. Sie müssen den Fehler überprüfen und korrigieren, bevor Sie fortfahren können.

```
An error occurred (AccessDeniedException) when calling the AddPermission operation:
 User: arn:aws:iam::57EXAMPLE833:user/EXAMPLE-1 is not authorized to perform:
 lambda:AddPer
mission on resource: arn:aws:lambda:Region:57EXAMPLE833:function:custom-auth-
function
```

Testen Ihrer Genehmiger

Sie können die TestInvokeAuthorizerAPI verwenden, um den Aufruf und die Rückgabewerte Ihres Autorisierers zu testen. Mit dieser API können Sie Protokollmetadaten angeben und die Signaturvalidierung in Ihrem Authorizer testen.

Die folgenden Tabs zeigen, wie Sie den verwenden können, AWS CLI um Ihren Authorizer zu testen.

Unix-like

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER \
```

```
--token TOKEN_VALUE --token-signature TOKEN_SIGNATURE
```

Windows CMD

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER ^
--token TOKEN_VALUE --token-signature TOKEN_SIGNATURE
```

Windows PowerShell

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER `
--token TOKEN_VALUE --token-signature TOKEN_SIGNATURE
```

Der Wert des `token-signature`-Parameters ist das signierte Token. Weitere Informationen zum Abrufen dieses Werts finden Sie unter the section called “Signieren des Tokens”.

Wenn Ihr Genehmiger einen Benutzernamen und ein Passwort verwendet, können Sie diese Informationen mithilfe des `--mqtt-context`-Parameters weitergeben. Die folgenden Registerkarten zeigen, wie Sie mithilfe der `TestInvokeAuthorizer`-API ein JSON-Objekt, das einen Benutzernamen, ein Passwort und einen Clientnamen enthält, an Ihren benutzerdefinierten Genehmiger senden.

Unix-like

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER  \
--mqtt-context '{"username": "USER_NAME", "password": "dGVzdA==",
 "clientId":"CLIENT_NAME"}'
```

Windows CMD

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER  ^
--mqtt-context '{"username": "USER_NAME", "password": "dGVzdA==",
 "clientId":"CLIENT_NAME"}'
```

Windows PowerShell

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER  `
--mqtt-context '{"username": "USER_NAME", "password": "dGVzdA==",
 "clientId":"CLIENT_NAME"}'
```

Das Passwort muss base64-kodiert sein. Das folgende Beispiel zeigt, wie Sie ein Passwort in einer Unix-like Umgebung codieren.

```
echo -n PASSWORD | base64
```

Verwalten von benutzerdefinierten Genehmigern

Sie können Ihre Genehmiger mithilfe der folgenden APIs verwalten.

- ListAuthorizers: Zeigt alle Autorisierer in Ihrem Konto an.
- DescribeAuthorizer: Zeigt die Eigenschaften des angegebenen Autorisierers an. Zu diesen Werten gehören das Erstellungsdatum, das Datum der letzten Änderung und andere Attribute.
- SetDefaultAuthorizer: Gibt den Standardautorisierer für Ihre AWS IoT Core Datenendpunkte an. AWS IoT Core verwendet diesen Autorisierer, wenn ein Gerät keine AWS IoT Core Anmeldeinformationen weitergibt und keinen Autorisierer angibt. Weitere Informationen zur Verwendung von AWS IoT Core Anmeldeinformationen finden Sie unter. the section called “Client-Authentifizierung”
- UpdateAuthorizer: Ändert den Status, den Namen des Token-Schlüssels oder die öffentlichen Schlüssel für den angegebenen Autorisierer.
- DeleteAuthorizer: Löscht den angegebenen Autorisierer.

> **Note**
>
> Sie können die Signaturanforderungen eines Genehmigers nicht aktualisieren. Das bedeutet, dass Sie das Signieren in einem vorhandenen Genehmiger, der dies fordert, nicht deaktivieren können. Sie können auch nicht die Anmeldung bei einem vorhandenen Genehmiger verlangen, der dies nicht fordert.

Benutzerdefinierte Authentifizierung mit Client-Zertifikaten X.509

Wenn Sie Geräte mit verbinden AWS IoT Core, stehen Ihnen mehrere Authentifizierungstypen zur Verfügung. Sie können X.509 Client-Zertifikate verwenden, mit denen Sie Client- und Geräteverbindungen authentifizieren können, oder benutzerdefinierte Autorisierer definieren, um Ihre eigene Client-Authentifizierungs- und Autorisierungslogik zu verwalten. In diesem Thema wird beschrieben, wie Sie die benutzerdefinierte Authentifizierung mit X.509 Client-Zertifikaten verwenden.

Die benutzerdefinierte Authentifizierung mit X.509 Zertifikaten kann hilfreich sein, wenn Sie Ihre Geräte bereits mithilfe von X.509 Zertifikaten authentifiziert haben und zusätzliche Überprüfungen und benutzerdefinierte Autorisierungen durchführen möchten. Wenn Sie beispielsweise die Daten Ihrer Geräte wie deren Seriennummern im X.509 Client-Zertifikat speichern, können Sie nach der AWS IoT Core Authentifizierung des X.509 Client-Zertifikats einen benutzerdefinierten Autorisierer verwenden, um bestimmte Geräte anhand der im Zertifikatsfeld gespeicherten Informationen zu identifizieren. CommonName Die Verwendung einer benutzerdefinierten Authentifizierung mit X.509 Zertifikaten kann Ihr Gerätesicherheitsmanagement beim Anschließen von Geräten verbessern AWS IoT Core und bietet mehr Flexibilität bei der Verwaltung der Authentifizierungs- und Autorisierungslogik. AWS IoT Core unterstützt die benutzerdefinierte Authentifizierung mit X.509 Zertifikaten, die den Authentifizierungstyp „ X.509Zertifikat“ und „Benutzerdefinierter Authorizer“ verwenden, der sowohl mit dem MQTT-Protokoll als auch mit dem HTTPS-Protokoll funktioniert. Weitere Informationen zu den Authentifizierungstypen und Anwendungsprotokollen, die von AWS IoT Core Geräteendpunkten unterstützt werden, finden Sie unter Gerätekommunikationsprotokolle.

> **Note**
>
> Die benutzerdefinierte Authentifizierung mit X.509 Client-Zertifikaten wird in den AWS GovCloud (US) Regionen nicht unterstützt.

> **Important**
>
> Sie müssen einen Endpunkt verwenden, der mithilfe von Domänenkonfigurationen erstellt wurde. Darüber hinaus müssen Clients die Erweiterung Server Name Indication (SNI) angeben, wenn sie eine Verbindung herstellen AWS IoT Core.

Der Prozess zur Authentifizierung von Geräten mithilfe einer benutzerdefinierten Authentifizierung mit X.509 Client-Zertifikaten besteht aus den folgenden Schritten.

- Schritt 1: Registrieren Sie Ihre X.509 Client-Zertifikate bei AWS IoT Core
- Schritt 2: Erstellen einer Lambda-Funktion
- Schritt 3: Erstellen Sie einen benutzerdefinierten Autorisierer
- Schritt 4: Legen Sie den Authentifizierungstyp und das Anwendungsprotokoll in einer Domänenkonfiguration fest

Schritt 1: Registrieren Sie Ihre X.509 Client-Zertifikate bei AWS IoT Core

Falls Sie dies noch nicht getan haben, registrieren und aktivieren Sie Ihre X.509 Client-Zertifikate mit AWS IoT Core. Andernfalls überspringen Sie diesen Schritt und gehen Sie direkt zum nächsten.

Gehen Sie wie folgt vor AWS IoT Core, um Ihre Kundenzertifikate bei zu registrieren und zu aktivieren:

1. Wenn Sie Kundenzertifikate direkt mit erstellen AWS IoT. Diese Client-Zertifikate werden automatisch bei registriert AWS IoT Core.
2. Wenn Sie Ihre eigenen Client-Zertifikate erstellen, folgen Sie diesen Anweisungen, um sie zu registrieren AWS IoT Core.
3. Folgen Sie diesen Anweisungen, um Ihre Client-Zertifikate zu aktivieren.

Schritt 2: Erstellen einer Lambda-Funktion

AWS IoT Core verwendet benutzerdefinierte Autorisierer, um benutzerdefinierte Authentifizierungs- und Autorisierungsschemata zu implementieren. Ein benutzerdefinierter Autorisierer ist mit einer Lambda-Funktion verknüpft, die bestimmt, ob ein Gerät authentifiziert ist und welche Operationen das Gerät ausführen darf. Wenn ein Gerät eine Verbindung herstellt AWS IoT Core, AWS IoT Core ruft es die Autorisierungsdetails ab, einschließlich des Autorisierernamens und der zugehörigen Lambda-Funktion, und ruft die Lambda-Funktion auf. Die Lambda-Funktion empfängt ein Ereignis, das ein JSON-Objekt mit den X.509 Client-Zertifikatsdaten des Geräts enthält. Ihre Lambda-Funktion verwendet dieses Event-JSON-Objekt, um die Authentifizierungsanfrage auszuwerten, die zu ergreifenden Aktionen zu entscheiden und eine Antwort zurückzusenden.

Beispiel für ein Lambda-Funktionsereignis

Das folgende JSON-Beispielobjekt enthält alle möglichen Felder, die eingeschlossen werden können. Das eigentliche JSON-Objekt wird nur Felder enthalten, die für die spezifische Verbindungsanforderung relevant sind.

```
{
 "token": "aToken",
 "signatureVerified": true,
 "protocols": [
  "tls",
  "mqtt"
 ],
```

```
 "protocolData": {
  "tls": {
   "serverName": "serverName",
   "x509CertificatePem": "x509CertificatePem",
   "principalId": "principalId"
  },
  "mqtt": {
   "clientId": "myClientId",
                    "username": "myUserName",
                    "password": "myPassword"
  }
 },
 "connectionMetadata": {
  "id": "UUID"
 }
}
```

`signatureVerified`

Ein boolescher Wert, der angibt, ob die im Authorizer konfigurierte Token-Signatur verifiziert wurde oder nicht, bevor die Lambda-Funktion des Autorisierers aufgerufen wird. Wenn der Autorisierer so konfiguriert ist, dass er die Tokensignatur deaktiviert, ist dieses Feld falsch.

`protocols`

Ein Array, das die Protokolle enthält, die für die Anfrage zu erwarten sind.

`protocolData`

Ein Objekt, das Informationen zu den in der Verbindung verwendeten Protokollen enthält. Es enthält protokollspezifische Details, die für die Authentifizierung, Autorisierung und mehr nützlich sein können.

`tls`- Dieses Objekt enthält Informationen zum TLS-Protokoll (Transport Layer Security).

- `serverName`- Die Hostnamenzeichenfolge (Server Name Indication, SNI). AWS IoT Core erfordert, dass Geräte die SNI-Erweiterung an das Transport Layer Security (TLS) -Protokoll senden und die vollständige Endpunktadresse vor Ort angeben. `host_name`
- `x509CertificatePem`- Das X.509 Zertifikat im PEM-Format, das für die Client-Authentifizierung in der TLS-Verbindung verwendet wird.
- `principalId`- Die dem Client in der TLS-Verbindung zugeordnete Prinzipal-ID.

`mqtt`- Dieses Objekt enthält Informationen zum MQTT-Protokoll.

- `clientId`- Eine Zeichenfolge muss nur für den Fall angegeben werden, dass das Gerät diesen Wert sendet.
- `username`- Der im MQTT Connect-Paket angegebene Benutzername.
- `password`- Das im MQTT Connect-Paket angegebene Passwort.

`connectionMetadata`

Metadaten der Verbindung.

`id`- Die Verbindungs-ID, die Sie für die Protokollierung und Fehlerbehebung verwenden können.

> **Note**
>
> In diesem Fall `principalId` handelt es sich um das JSON-Objekt `x509CertificatePem` und um zwei neue Felder in der Anfrage. Der Wert von `principalId` ist derselbe wie der Wert von`certificateId`. Weitere Informationen finden Sie unter Zertifikat.

Beispiel für eine Antwort auf eine Lambda-Funktion

Die Lambda-Funktion sollte Informationen aus dem JSON-Ereignisobjekt verwenden, um die eingehende Verbindung zu authentifizieren und zu entscheiden, welche Aktionen in der Verbindung zulässig sind.

Das folgende JSON-Objekt enthält eine Beispielantwort, die Ihre Lambda-Funktion senden kann.

```
{
 "isAuthenticated": true,
 "principalId": "xxxxxxxx",
 "disconnectAfterInSeconds": 86400,
 "refreshAfterInSeconds": 300,
 "policyDocuments": [
  {
   "Version": "2012-10-17",
   "Statement": [
    {
     "Effect": "Allow",
     "Action": "iot:Publish",
     "Resource": "arn:aws:iot:us-east-1:123456789012:topic/customauthtesting"
    }
```

```
    ]
   }
  ]
}
```

In diesem Beispiel sollte diese Funktion eine Antwort senden, die die folgenden Werte enthält.

`isAuthenticated`

Ein boolescher Wert, der angibt, ob die Anfrage authentifiziert ist.

`principalId`

Eine alphanumerische Zeichenfolge, die als Kennung für das Token dient, das von der benutzerdefinierten Autorisierungsanfrage gesendet wurde. Der Wert muss eine alphanumerische Zeichenfolge mit mindestens einem und nicht mehr als 128 Zeichen sein. Es identifiziert die Verbindung in Protokollen. Der Wert von `principalId` muss mit dem Wert von `principalId` im Event-JSON-Objekt identisch sein (d. h. CertificateID certificateId). X.509

`policyDocuments`

Eine Liste von JSON-formatted AWS IoT Core Richtliniendokumenten. Der Wert ist optional und unterstützt Dingrichtlinienvariablen und Zertifikatsrichtlinienvariablen. Die maximale Anzahl von Richtliniendokumenten ist 10. Jedes Richtliniendokument darf maximal 2048 Zeichen enthalten. Wenn Sie mehrere Richtlinien an Ihr Client-Zertifikat und die Lambda-Funktion angehängt haben, ist die Berechtigung eine Sammlung aller Richtlinien. Weitere Informationen zum Erstellen von AWS IoT Core Richtlinien finden Sie unter Richtlinien.

`disconnectAfterInSeconds`

Eine Ganzzahl, die die maximale Dauer (in Sekunden) der Verbindung zum AWS IoT Core Gateway angibt. Der Mindestwert ist 300 Sekunden und der Höchstwert ist 86.400 Sekunden. `disconnectAfterInSeconds`gilt für die gesamte Lebensdauer einer Verbindung und wird bei aufeinanderfolgenden Richtlinienaktualisierungen nicht aktualisiert.

`refreshAfterInSeconds`

Eine Ganzzahl, die das Intervall zwischen Richtlinienaktualisierungen angibt. Wenn dieses Intervall abgelaufen ist, AWS IoT Core ruft die Lambda-Funktion auf, um Richtlinienaktualisierungen zu ermöglichen. Der Mindestwert ist 300 Sekunden und der Höchstwert 86 400 Sekunden.

Beispiel-Lambda-Funktion

Im Folgenden finden Sie ein Beispiel für eine Node.js Lambda-Funktion. Die Funktion untersucht das X.509 Zertifikat des Kunden und extrahiert relevante Informationen wie die Seriennummer, den Fingerabdruck und den Namen des Antragstellers. Wenn die extrahierten Informationen den erwarteten Werten entsprechen, wird dem Client der Zugriff auf die Verbindung gewährt. Dieser Mechanismus stellt sicher, dass nur autorisierte Clients mit gültigen Zertifikaten eine Verbindung herstellen können.

```
const crypto = require('crypto');

exports.handler = async (event) => {

    // Extract the certificate PEM from the event
    const certPem = event.protocolData.tls.x509CertificatePem;

    // Parse the certificate using Node's crypto module
    const cert = new crypto.X509Certificate(certPem);

    var effect = "Deny";
    // Allow permissions only for a particular certificate serial, fingerprint, and
 subject
    if (cert.serialNumber === "7F8D2E4B9C1A5036DE8F7C4B2A91E5D80463BC9A1257" // This is
 a random serial
       && cert.fingerprint ===
 "F2:9A:C4:1D:B5:E7:08:3F:6B:D0:4E:92:A7:C1:5B:8D:16:0F:E3:7A" // This is a random
 fingerprint
       && cert.subject === "allow.example.com") {
      effect = "Allow";
    }

    return generateAuthResponse(event.protocolData.tls.principalId, effect);
};


// Helper function to generate the authorization response.
function generateAuthResponse(principalId, effect) {
    const authResponse = {
        isAuthenticated: true,
        principalId,
        disconnectAfterInSeconds: 3600,
        refreshAfterInSeconds: 300,
        policyDocuments: [
```

```
          {
            Version: "2012-10-17",
            Statement: [
              {
                Action: ["iot:Connect"],
                Effect: effect,
                Resource: [
                  "arn:aws:iot:us-east-1:123456789012:client/myClientName"
                ]
              },
              {
                Action: ["iot:Publish"],
                Effect: effect,
                Resource: [
                  "arn:aws:iot:us-east-1:123456789012:topic/telemetry/myClientName"
                ]
              },
              {
                Action: ["iot:Subscribe"],
                Effect: effect,
                Resource: [
                   "arn:aws:iot:us-east-1:123456789012:topicfilter/telemetry/
myClientName"
                ]
              },
              {
                Action: ["iot:Receive"],
                Effect: effect,
                Resource: [
                   "arn:aws:iot:us-east-1:123456789012:topic/telemetry/myClientName"
                ]
              }
            ]
          }
        ]
      };

  return authResponse;
}
```

Die vorhergehende Lambda-Funktion gibt das folgende JSON zurück, wenn sie ein Zertifikat mit der erwarteten Seriennummer, dem Fingerabdruck und dem Betreff empfängt. Der Wert

von ist `x509CertificatePem` das im TLS-Handshake bereitgestellte Client-Zertifikat. Weitere Informationen finden Sie unter Definieren Ihrer Lambda-Funktion.

```
{
 "isAuthenticated": true,
 "principalId": "principalId in the event JSON object",
 "policyDocuments": [
  {
   "Version": "2012-10-17",
   "Statement": [
    {
     "Action": "iot:Connect",
     "Effect": "Allow",
     "Resource": "arn:aws:iot:us-east-1:123456789012:client/myClientName"
    },
    {
     "Action": "iot:Publish",
     "Effect": "Allow",
     "Resource": "arn:aws:iot:us-east-1:123456789012:topic/telemetry/myClientName"
    },
    {
     "Action": "iot:Subscribe",
     "Effect": "Allow",
     "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/telemetry/
myClientName"
    },
    {
     "Action": "iot:Receive",
     "Effect": "Allow",
     "Resource": "arn:aws:iot:us-east-1:123456789012:topic/telemetry/myClientName"
    }
   ]
  }
 ],
 "disconnectAfterInSeconds": 3600,
 "refreshAfterInSeconds": 300
}
```

Schritt 3: Erstellen Sie einen benutzerdefinierten Autorisierer

Nachdem Sie die Lambda-Funktion definiert haben, erstellen Sie einen benutzerdefinierten Authorizer, um Ihre eigene Client-Authentifizierungs- und Autorisierungslogik zu verwalten. Sie

können den detaillierten Anweisungen in Schritt 3: Erstellen einer Kundenautorisierungsressource und deren Autorisierung folgen. Weitere Informationen finden Sie unter Einen Autorisierer erstellen.

Bei der Erstellung des benutzerdefinierten Autorisierers müssen Sie die AWS IoT Erlaubnis erteilen, die Lambda-Funktion aufzurufen, nachdem sie erstellt wurde. Eine ausführliche Anleitung finden Sie unter Autorisieren AWS IoT zum Aufrufen Ihrer Lambda-Funktion.

Schritt 4: Legen Sie den Authentifizierungstyp und das Anwendungsprotokoll in einer Domänenkonfiguration fest

Um Geräte mithilfe einer benutzerdefinierten Authentifizierung mit X.509 Client-Zertifikaten zu authentifizieren, müssen Sie den Authentifizierungstyp und das Anwendungsprotokoll in einer Domänenkonfiguration festlegen und die SNI-Erweiterung senden. Der Wert von `authenticationType` muss sein`CUSTOM_AUTH_X509`, und der Wert von `applicationProtocol` kann entweder oder sein`SECURE_MQTT`. `HTTPS`

Legen Sie den Authentifizierungstyp und das Anwendungsprotokoll in der Domänenkonfiguration (CLI) fest

Wenn Sie keine Domänenkonfiguration haben, verwenden Sie den create-domain-configurationBefehl, um eine zu erstellen. Der Wert von `authenticationType` muss sein`CUSTOM_AUTH_X509`, und der Wert von `applicationProtocol` kann entweder `SECURE_MQTT` oder sein`HTTPS`.

```
aws iot create-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --authentication-type CUSTOM_AUTH_X509 \
    --application-protocol SECURE_MQTT \
    --authorizer-config '{
        "defaultAuthorizerName": my-custom-authorizer
    }'
```

Wenn Sie bereits über eine Domänenkonfiguration verfügen, verwenden Sie bei `applicationProtocol` Bedarf den update-domain-configurationBefehl update `authenticationType` und. Beachten Sie, dass Sie den Authentifizierungstyp oder das Protokoll auf dem Standardendpunkt (`iot:Data-ATS`) nicht ändern können.

```
aws iot update-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --authentication-type CUSTOM_AUTH_X509 \
```

```
    --application-protocol SECURE_MQTT \
    --authorizer-config '{
        "defaultAuthorizerName": my-custom-authorizer
    }'
```

`domain-configuration-name`

Der Name der Domänenkonfiguration.

`authentication-type`

Der Authentifizierungstyp der Domänenkonfiguration. Weitere Informationen finden Sie unter Auswahl eines Authentifizierungstyps.

`application-protocol`

Das Anwendungsprotokoll, mit dem Geräte kommunizieren AWS IoT Core. Weitere Informationen finden Sie unter Auswahl eines Anwendungsprotokolls.

`--authorizer-config`

Ein Objekt, das die Autorisierungskonfiguration in einer Domänenkonfiguration angibt.

`defaultAuthorizerName`

Der Name des Autorisierers für eine Domänenkonfiguration.

Weitere Informationen finden Sie unter CreateDomainConfigurationund in UpdateDomainConfigurationder AWS IoT API-Referenz. Weitere Informationen zur Domänenkonfiguration finden Sie unter Domänenkonfigurationen.

Verbindung herstellen zu AWS IoT Core mithilfe der benutzerdefinierten Authentifizierung

Geräte können AWS IoT Core mithilfe der benutzerdefinierten Authentifizierung mit jedem Protokoll, das Geräte-Messaging AWS IoT Core unterstützt, eine Verbindung herstellen. Weitere Informationen zu unterstützten Kommunikationsprotokollen finden Sie unter the section called “Gerätekommunikationsprotokolle”. Die Verbindungsdaten, die Sie an die Lambda-Funktion Ihres Genehmigers übergeben, hängen vom verwendeten Protokoll ab. Weitere Informationen zum Erstellen der Lambda-Funktion Ihres Genehmigers finden Sie unter the section called “Definieren Ihrer Lambda-Funktion”. In den folgenden Abschnitten wird erläutert, wie Sie eine Verbindung zur Authentifizierung mit jedem unterstützten Protokoll herstellen.

HTTPS

Geräte, an die Daten AWS IoT Core mithilfe der HTTP Publish API gesendet werden, können Anmeldeinformationen entweder über Anforderungsheader oder Abfrageparameter in ihren HTTP-POST-Anfragen übergeben. Geräte können mithilfe des Header- oder Abfrageparameters `x-amz-customauthorizer-name` einen aufzurufenden Genehmiger angeben. Wenn Sie die Tokensignatur in Ihrem Genehmiger aktiviert haben, müssen Sie den *`token-key-name`* und die `x-amz-customauthorizer-signature` entweder in den Anforderungsheadern oder Abfrageparametern übergeben. Beachten Sie, dass der *`token-signature`* Wert angegeben werden muss URL-encoded , wenn Sie ihn JavaScript aus dem Browser heraus verwenden.

> **Note**
>
> Der Kundengenehmiger für das HTTPS-Protokoll unterstützt nur Veröffentlichungsvorgänge. Weitere Informationen über das HTTP-Protokoll finden Sie unter the section called “Gerätekommunikationsprotokolle”.

Die folgenden Beispielanforderungen zeigen, wie Sie diese Parameter sowohl in Anforderungsheadern als auch in Abfrageparametern übergeben.

```
//Passing credentials via headers
POST /topics/topic?qos=qos HTTP/1.1
Host: your-endpoint
x-amz-customauthorizer-signature: token-signature
token-key-name: token-value
x-amz-customauthorizer-name: authorizer-name

//Passing credentials via query parameters
POST /topics/topic?qos=qos&x-amz-customauthorizer-signature=token-signature&token-key-name=token-value HTTP/1.1
```

MQTT

Geräte, die AWS IoT Core über eine MQTT-Verbindung eine Verbindung herstellen, können Anmeldeinformationen über die `password` Felder `username` und von MQTT-Nachrichten weiterleiten. Der Wert `username` kann optional auch eine Abfragezeichenfolge enthalten, die zusätzliche Werte (einschließlich eines Tokens, einer Signatur und eines Genehmigernamens) an Ihren Genehmiger übergibt. Sie können diese Abfragezeichenfolge verwenden, wenn Sie anstelle der Werte `username` und `password` ein tokenbasiertes Authentifizierungsschema verwenden möchten.

> **Note**
>
> Die Daten im Passwortfeld sind Base64-codiert von. AWS IoT Core Ihre Lambda-Funktion muss sie dekodieren.

Das folgende Beispiel enthält eine `username`-Zeichenfolge mit zusätzlichen Parametern, die ein Token und eine Signatur angeben.

```
username?x-amz-customauthorizer-name=authorizer-name&x-amz-customauthorizer-
signature=token-signature&token-key-name=token-value
```

Um einen Authorizer aufzurufen, müssen Geräte, die über MQTT und benutzerdefinierte Authentifizierung eine Verbindung herstellen, eine Verbindung über Port 443 herstellen. AWS IoT Core Sie müssen außerdem die TLS-Erweiterung Application Layer Protocol Negotiation (ALPN) mit dem Wert `mqtt` und die Erweiterung Server Name Indication (SNI) mit dem Hostnamen ihres Datenendpunkts übergeben. AWS IoT Core Der Wert `x-amz-customauthorizer-signature` sollte URL-kodiert sein, um Fehler zu vermeiden. Wir empfehlen außerdem dringend, dass die Werte `x-amz-customauthorizer-name` und `token-key-name` URL-kodiert sind. Weitere Informationen zu diesen Werten finden Sie unter the section called “Gerätekommunikationsprotokolle”. Die V2 AWS IoT Geräte-SDKs, SDKs für Mobilgeräte und AWS IoT Geräte-Client kann diese beiden Erweiterungen konfigurieren.

MQTT über WebSockets

Geräte, die AWS IoT Core über MQTT-Over eine Verbindung herstellen, WebSockets können Anmeldeinformationen auf eine der beiden folgenden Arten weitergeben.

- Über Anforderungsheader oder Abfrageparameter in der HTTP-UPGRADE-Anfrage, um die WebSockets Verbindung herzustellen.
- Über die Felder `username` und `password` in der MQTT-CONNECT-Nachricht.

Wenn Sie Anmeldeinformationen über die MQTT-Connect-Nachricht weitergeben, sind die TLS-Erweiterungen ALPN und SNI erforderlich. Weitere Informationen zu diesen Erweiterungen finden Sie unter the section called “MQTT”. Das folgende Beispiel zeigt, wie Sie Anmeldeinformationen über die HTTP-Upgrade-Anforderung übergeben.

```
GET /mqtt HTTP/1.1
```

```
Host: your-endpoint
Upgrade: WebSocket
Connection: Upgrade
x-amz-customauthorizer-signature: token-signature
token-key-name: token-value
sec-WebSocket-Key: any random base64 value
sec-websocket-protocol: mqtt
sec-WebSocket-Version: websocket version
```

Signieren des Tokens

Sie müssen das Token mit dem privaten Schlüssel des Paars aus öffentlichem und privatem Schlüssel signieren, das Sie im `create-authorizer`-Aufruf verwendet haben. Die folgenden Beispiele zeigen, wie die Tokensignatur mithilfe eines UNIX-like Befehls und JavaScript erstellt wird. Sie verwenden den SHA-256 Hash-Algorithmus, um die Signatur zu codieren.

Command line

```
echo -n TOKEN_VALUE | openssl dgst -sha256 -sign PEM encoded RSA private key |
 openssl base64
```

JavaScript

```
const crypto = require('crypto')

const key = "PEM encoded RSA private key"

const k = crypto.createPrivateKey(key)
let sign = crypto.createSign('SHA256')
sign.write(t)
sign.end()
const s = sign.sign(k, 'base64')
```

Fehlerbehebung für Ihre Genehmiger

In diesem Thema werden häufig auftretende Probleme, die zu Konflikten bei benutzerdefinierten Authentifizierungsworkflows führen können, sowie Schritte zu deren Behebung beschrieben. Um

Probleme am effektivsten zu beheben, aktivieren Sie CloudWatch Protokolle für AWS IoT Core und setzen Sie die Protokollebene auf DEBUG. Sie können CloudWatch Protokolle in der AWS IoT Core Konsole aktivieren (https://console.aws.amazon.com/iot/). Weitere Informationen zum Aktivieren und Konfigurieren von Protokollen für AWS IoT Core finden Sie unter the section called “Konfiguration AWS IoT Protokollierung”.

> **Note**
>
> Wenn Sie die Protokollebene für längere Zeit auf DEBUG belassen, CloudWatch können große Mengen an Protokolldaten gespeichert werden. Dies kann Ihre CloudWatch Gebühren erhöhen. Erwägen Sie, die ressourcenbasierte Protokollierung zu verwenden, um die Ausführlichkeit nur für Geräte in einer bestimmten Objektgruppe zu erhöhen. Weitere Informationen zur ressourcenbasierten Protokollierung finden Sie unter the section called “Konfiguration AWS IoT Protokollierung”. Wenn Sie mit der Fehlerbehebung fertig sind, reduzieren Sie außerdem die Protokollebene auf eine weniger ausführliche Ebene.

Bevor Sie mit der Problembehandlung beginnen, überprüfen Sie the section called “Grundlegendes zum Workflow für die benutzerdefinierte Authentifizierung” für eine allgemeine Übersicht über den benutzerdefinierten Authentifizierungsprozess. Dadurch können Sie leichter die mögliche Ursache eines Problems nachvollziehen und danach suchen.

In diesem Thema werden zwei Bereiche behandelt, die Sie untersuchen sollten.

- Probleme im Zusammenhang mit der Lambda-Funktion Ihres Genehmigers.
- Probleme im Zusammenhang mit Ihrem Gerät.

Nach Problemen mit der Lambda-Funktion Ihres Genehmigers suchen

Gehen Sie wie folgt vor, um sicherzustellen, dass die Verbindungsversuche Ihrer Geräte Ihre Lambda-Funktion aufrufen.

1. Überprüfen Sie, welche Lambda-Funktion Ihrem Genehmiger zugeordnet ist.

   Sie können dies tun, indem Sie die DescribeAuthorizerAPI aufrufen oder im Bereich Secure der AWS IoT Core Konsole auf den gewünschten Autorisierer klicken.
2. Überprüfen Sie die Aufrufmetriken der Lambda-Funktion. Führen Sie dazu die folgenden Schritte aus.

a. Öffnen Sie die AWS Lambda Konsole (https://console.aws.amazon.com/lambda/) und wählen Sie die Funktion aus, die Ihrem Autorisierer zugeordnet ist.

b. Wählen Sie die Registerkarte Überwachen und überprüfen Sie die Metriken des für Ihr Problem relevanten Zeitraums.

3. Wenn Sie keine Aufrufe sehen, überprüfen Sie, ob der Benutzer berechtigt AWS IoT Core ist, Ihre Lambda-Funktion aufzurufen. Wenn Sie Aufrufe sehen, fahren Sie mit dem nächsten Schritt fort. Führen Sie die folgenden Schritte aus, um sicherzustellen, dass Ihre Lambda-Funktion über die erforderlichen Berechtigungen verfügt.

   a. Wählen Sie in der Konsole den Tab „Berechtigungen“ für Ihre Funktion aus. AWS Lambda

   b. Suchen Sie unten auf der Seite nach dem Abschnitt „Resource-basedRichtlinien“. Wenn Ihre Lambda-Funktion über die erforderlichen Berechtigungen verfügt, sieht die Richtlinie wie im folgenden Beispiel aus.

```
{
  "Version":"2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "Id123",
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-
east-1:111111111111:function:FunctionName",
      "Condition": {
        "ArnLike": {
          "AWS:SourceArn": "arn:aws:iot:us-east-1:111111111111:authorizer/
AuthorizerName"
        },
        "StringEquals": {
          "AWS:SourceAccount": "111111111111"
        }
      }
    }
  ]
}
```

c. Diese Richtlinie erteilt dem AWS IoT Core Schulleiter die `InvokeFunction` Genehmigung für Ihre Funktion. Wenn Sie sie nicht sehen, müssen Sie sie mithilfe der AddPermissionAPI hinzufügen. Im folgenden Beispiel wird gezeigt, wie Sie sie über AWS CLI hinzufügen können.

```
aws lambda add-permission --function-name FunctionName --principal
 iot.amazonaws.com --source-arn AuthorizerARn --statement-id Id-123 --action
 "lambda:InvokeFunction"
```

4. Wenn Sie Aufrufe sehen, stellen Sie sicher, dass keine Fehler vorliegen. Ein Fehler könnte darauf hinweisen, dass die Lambda-Funktion das Verbindungsereignis, das an sie AWS IoT Core gesendet wird, nicht richtig verarbeitet.

   Hinweise zur Behandlung des Ereignisses in Ihrer Lambda-Funktion finden Sie unter the section called “Definieren Ihrer Lambda-Funktion”. Sie können die Testfunktion in der AWS Lambda Konsole (https://console.aws.amazon.com/lambda/) verwenden, um Testwerte in der Funktion fest zu codieren, um sicherzustellen, dass die Funktion Ereignisse korrekt verarbeitet.

5. Wenn Sie Aufrufe ohne Fehler sehen, Ihre Geräte jedoch keine Verbindung herstellen (oder Nachrichten veröffentlichen, abonnieren und empfangen) können, liegt das Problem möglicherweise daran, dass die Richtlinie, die Ihre Lambda-Funktion zurückgibt, keine Berechtigungen für die Aktionen gewährt, die Ihre Geräte ausführen möchten. Führen Sie die folgenden Schritte aus, um festzustellen, ob ein Fehler bei der Richtlinie vorliegt, die die Funktion zurückgibt.

   a. Verwenden Sie eine Amazon CloudWatch Logs Insights-Abfrage, um Protokolle über einen kurzen Zeitraum auf Fehler zu überprüfen. Die folgende Beispielabfrage sortiert Ereignisse nach Zeitstempel und sucht nach Fehlern.

```
display clientId, eventType, status, @timestamp | sort @timestamp desc | filter
 status = "Failure"
```

   b. Aktualisieren Sie Ihre Lambda-Funktion, um die Daten zu protokollieren, zu denen sie zurückkehrt, AWS IoT Core und das Ereignis, das die Funktion auslöst. Sie können diese Protokolle verwenden, um die Richtlinie zu überprüfen, die die Funktion erstellt.

6. Wenn Sie Aufrufe ohne Fehler sehen, Ihre Geräte jedoch keine Verbindung herstellen (oder Nachrichten veröffentlichen, abonnieren und empfangen) können, könnte eine weitere Ursache sein, dass Ihre Lambda-Funktion das Timeout-Limit überschritten hat. Das Timeout-Limit der

Lambda-Funktion für den benutzerdefinierten Genehmiger beträgt 5 Sekunden. Sie können die Funktionsdauer in CloudWatch Protokollen oder Metriken überprüfen.

Untersuchen von Geräteproblemen

Wenn Sie keine Probleme mit dem Aufrufen Ihrer Lambda-Funktion oder mit der Richtlinie, die die Funktion zurückgibt, feststellen, suchen Sie nach Problemen bei den Verbindungsversuchen Ihrer Geräte. Fehlerhaft formatierte Verbindungsanfragen können dazu führen, dass Ihr Autorisierer AWS IoT Core nicht ausgelöst wird. Verbindungsprobleme können sowohl auf TLS- als auch auf Anwendungsebene auftreten.

Mögliche Probleme mit der TLS-Ebene:

- Kunden müssen bei allen benutzerdefinierten Authentifizierungsanfragen entweder einen Hostnamen-Header (HTTP, MQTT over WebSockets) oder die Server Name Indication TLS-Erweiterung (HTTP, MQTT over WebSockets, MQTT) übergeben. In beiden Fällen muss der übergebene Wert mit einem der Datenendpunkte Ihres Kontos übereinstimmen. AWS IoT Core Dies sind die Endpunkte, die zurückgegeben werden, wenn Sie die folgenden CLI-Befehle ausführen.
  - `aws iot describe-endpoint --endpoint-type iot:Data-ATS`
  - `aws iot describe-endpoint --endpoint-type iot:Data`(für ältere VeriSign Endpunkte)
- Geräte, die eine benutzerdefinierte Authentifizierung in MQTT-Verbindungen verwenden, müssen auch die TLS-Erweiterung „Application Layer Protocol Negotiation“ (ALPN) mit dem Wert `mqtt` übergeben.
- Die benutzerdefinierte Authentifizierung ist derzeit nur auf Port 443 verfügbar.

Mögliche Probleme auf Anwendungsebene:

- Wenn das Signieren aktiviert ist (das Feld `signingDisabled` in Ihrem Genehmiger lautet „false“), suchen Sie nach den folgenden Signaturproblemen.
  - Stellen Sie sicher, dass Sie die Tokensignatur entweder im `x-amz-customauthorizer-signature`-Header oder in einem Abfragezeichenfolgenparameter übergeben.
  - Stellen Sie sicher, dass der Service keinen anderen Wert als das Token signiert.
  - Stellen Sie sicher, dass Sie das Token im Header- oder Abfrageparameter übergeben, den Sie im Feld `token-key-name` in Ihrem Genehmiger angegeben haben.

- Stellen Sie sicher, dass der Genehmigername, den Sie im `x-amz-customauthorizer-name`-Header- oder Abfragezeichenfolgenparameter übergeben, gültig ist oder dass Sie einen Standardgenehmiger für Ihr Konto angegeben haben.

Autorisierung

Autorisierung ist der Prozess der Erteilung von Berechtigungen an eine authentifizierte Identität. Sie gewähren Nutzungsberechtigungen AWS IoT Core und IAM-Richtlinien. AWS IoT Core Dieses Thema behandelt AWS IoT Core -Richtlinien. Weitere Informationen zu IAM-Richtlinien finden Sie unter Identitäts- und Zugriffsmanagement für AWS IoT und Wie AWS IoT funktioniert mit IAM.

AWS IoT Core Richtlinien bestimmen, was eine authentifizierte Identität bewirken kann. Eine authentifizierte Identität wird von Geräten, mobilen Anwendungen, Webanwendungen und Desktop-Anwendungen verwendet. Eine authentifizierte Identität kann sogar ein Benutzer sein, der AWS IoT Core CLI-Befehle eingibt. Eine Identität kann nur dann AWS IoT Core Operationen ausführen, wenn sie über eine Richtlinie verfügt, die ihr die Erlaubnis für diese Operationen erteilt.

Sowohl AWS IoT Core Richtlinien als auch IAM-Richtlinien werden verwendet, um die Operationen AWS IoT Core zu steuern, die eine Identität (auch Principal genannt) ausführen kann. Welchen Richtlinientyp Sie verwenden, hängt von der Art der Identität ab, mit der Sie sich authentifizieren. AWS IoT Core

AWS IoT Core Operationen sind in zwei Gruppen unterteilt:

- Mit der API der Steuerebene können Sie administrative Aufgaben wie Erstellen oder Aktualisieren von Zertifikaten, Things, Regeln usw. ausführen.
- Die Datenebene-API ermöglicht das Senden von Daten an und das Empfangen von Daten von AWS IoT Core.

Der von Ihnen verwendete Richtlinientyp hängt davon ab, ob Sie die API der Steuerebene oder der Datenebene nutzen.

In der folgenden Tabelle sind die Identitätstypen, die von ihnen verwendeten Protokolle und die Richtlinientypen für die Autorisierung aufgelistet.

AWS IoT Core Datenebenen-API und Richtlinientypen

Protokoll- und Authentifizierungsmechanismus	SDK	Identitätstyp	Richtlinientyp		
MQTT-Over TLS/TCP, gegenseitige TLS-Authentifizierung (Port 8883 oder 443) †)	AWS IoT Geräte-SDK	X.509 zertifikate	AWS IoT Core Politik		
MQTT-Over HTTPS/WebSocket, AWS SigV4-Authentifizierung (Port 443)	AWS SDK für Mobilgeräte	Eine authentifizierte Amazon-Cognito-Identität	IAM und Richtlinien AWS IoT Core		
		Eine nicht authentifizierte Amazon-Cognito-Identität	IAM-Richtlinie		
		IAM oder Verbundidentität	IAM-Richtlinie		
HTTPS, Authentifizierung mit AWS Signaturversion 4 (Port 443)	AWS CLI	Amazon-Cognito, IAM oder Verbundidentität	IAM-Richtlinie		

Protokoll- und Authentifizierungsmechanismus	SDK	Identitätstyp	Richtlinientyp		
HTTPS, gegenseitige TLS-Authentifizierung (Port 8443)	Keine SDK-Unterstützung	X.509 zertifikate	AWS IoT Core Politik		
HTTPS über benutzerdefinierte Authentifizierung (Port 443)	AWS IoT Geräte-SDK	Benutzerdefinierter Autorisierer	Genehmigerrichtlinie		

AWS IoT Core API- und Richtlinientypen auf der Kontrollebene

Protokoll- und Authentifizierungsmechanismus	SDK	Identitätstyp	Richtlinientyp		
AWS HTTPS-Signaturauthentifizierung, Version 4 (Port 443)	AWS CLI	Amazon-Cognito-Identität	IAM-Richtlinie		
		IAM oder Verbundidentität	IAM-Richtlinie		

AWS IoT Core Richtlinien sind X.509 Zertifikaten, Amazon Cognito Cognito-Identitäten oder Dinggruppen zugeordnet. IAM-Richtlinien sind einem IAM-Benutzer, einer IAM-Gruppe oder einer IAM-Rolle angefügt. Wenn Sie die AWS IoT Konsole oder die AWS IoT Core CLI verwenden, um die Richtlinie anzuhängen (an ein Zertifikat, Amazon Cognito Identity oder eine Dinggruppe), verwenden

Sie eine AWS IoT Core Richtlinie. Andernfalls verwenden Sie eine IAM-Richtlinie. AWS IoT Core Richtlinien, die einer Dinggruppe zugeordnet sind, gelten für alle Dinge innerhalb dieser Dinggruppe. Damit die AWS IoT Core Richtlinie wirksam wird, müssen der Name `clientId` und der Name der Sache übereinstimmen.

Policy-based Die Autorisierung ist ein mächtiges Werkzeug. Sie gibt Ihnen die komplette Kontrolle darüber, welche Berechtigungen ein Gerät, ein Benutzer oder eine Anwendung in AWS IoT Core hat. Stellen Sie sich zum Beispiel ein Gerät vor, zu dem über ein Zertifikat eine AWS IoT Core Verbindung hergestellt wird. Sie können dem Gerät den Zugriff auf alle MQTT-Topics gewähren oder seinen Zugriff auf ein einzelnes Topic einschränken. Ein anderes Beispiel ist ein Benutzer, der CLI-Befehle in der Befehlszeile eingibt. Mithilfe einer Richtlinie können Sie dem Benutzer den Zugriff auf Befehle oder AWS IoT Core Ressourcen gewähren oder verweigern. Sie können auch den Zugriff einer Anwendung auf AWS IoT Core -Ressourcen steuern.

Aufgrund der Art und Weise, in der AWS IoT Richtliniendokumente zwischenspeichert, kann es einige Minuten dauern, bis Änderungen an einer Richtlinie wirksam werden. Das heißt, dass es einige Minuten dauern kann, bis der kürzlich gewährte Zugriff auf eine Ressource tatsächlich hergestellt ist. Außerdem kann nach dem Widerruf des Zugriffs noch mehrere Minuten lang auf eine Ressource zugegriffen werden.

AWS Schulung und Zertifizierung

Weitere Informationen zur Autorisierung finden Sie im AWS IoT Core Kurs Deep Dive into AWS IoT Core Authentication and Authorization auf der Website für AWS Schulungen und Zertifizierungen.

AWS IoT Core Richtlinien

AWS IoT Core Richtlinien sind JSON-Dokumente. Sie folgen denselben Konventionen wie IAM-Richtlinien. AWS IoT Core unterstützt benannte Richtlinien, sodass viele Identitäten auf dasselbe Richtliniendokument verweisen können. Bei benannten Richtlinien wird Versionierung verwendet, um Rollbacks zu vereinfachen.

AWS IoT Core Mithilfe von Richtlinien können Sie den Zugriff auf die AWS IoT Core Datenebene steuern. Die AWS IoT Core -Datenebene besteht aus Operationen, mit denen Sie eine Verbindung zum AWS IoT Core Message Broker herstellen, MQTT-Nachrichten senden und empfangen sowie einen Geräteschatten abrufen oder aktualisieren können.

Eine AWS IoT Core Richtlinie ist ein JSON-Dokument, das eine oder mehrere Richtlinienerklärungen enthält. Jede Anweisung enthält:

- `Effect`, das angibt, ob die Aktion zugelassen oder verweigert wird.
- `Action`, das die Aktion angibt, die die Richtlinie zulässt oder verweigert.
- `Resource`, das die Ressource oder die Ressourcen angibt, für die die Aktion zugelassen oder verweigert wird.

Es kann zwischen 6 und 8 Minuten dauern, bis Änderungen an einer Richtlinie wirksam werden, da die Richtliniendokumente AWS IoT zwischengespeichert werden. Das heißt, dass es einige Minuten dauern kann, bis der kürzlich gewährte Zugriff auf eine Ressource tatsächlich hergestellt ist. Außerdem kann nach dem Widerruf des Zugriffs noch mehrere Minuten lang auf eine Ressource zugegriffen werden.

AWS IoT Core Richtlinien können an X.509 Zertifikate, Amazon Cognito Cognito-Identitäten und Dinggruppen angehängt werden. Die einer Objektgruppe angefügten Richtlinien gelten für alle Objekte innerhalb dieser Gruppe. Damit die Richtlinie wirksam wird, müssen die `clientId` und der Objektname übereinstimmen. AWS IoT Core -Richtlinien folgen derselben Richtlinienbewertungslogik wie IAM-Richtlinien. Standardmäßig werden alle Richtlinien implizit verweigert. Eine explizite Zugriffserlaubnis in einer identitätsbasierten oder ressourcenbasierten Richtlinie setzt das Standardverhalten außer Kraft. Eine explizite Zugriffsverweigerung überschreibt jede Zugriffserlaubnis in einer Richtlinie. Weitere Informationen finden Sie unter Auswertungslogik für Richtlinien im AWS Identity and Access Management -Benutzerhandbuch.

Themen

- AWS IoT Core politische Maßnahmen
- AWS IoT Core Ressourcen für Aktionen
- AWS IoT Core Richtlinienvariablen
- Cross-service verwirrter Stellvertreter, Prävention
- AWS IoT Core Beispiele für Richtlinien
- Autorisierung mit Amazon-Cognito-Identitäten

AWS IoT Core politische Maßnahmen

Die folgenden Richtlinienaktionen werden von AWS IoT Core definiert:

MQTT-Richtlinienaktionen

`iot:Connect`

Stellt die Berechtigung dar, eine Verbindung zum AWS IoT Core Message Broker herzustellen. Die Berechtigung `iot:Connect` wird jedes Mal geprüft, wenn eine `CONNECT`-Anforderung an den Broker gesendet wird. Der Message Broker ermöglicht nicht die gleichzeitige Verbindung von zwei Clients mit der gleichen Client-ID. Nachdem sich der zweite Client verbunden hat, schließt der Broker die bestehende Verbindung. Verwenden Sie die `iot:Connect`-Berechtigung, um sicherzustellen, dass sich nur autorisierte Clients mit einer bestimmten Client-ID verbinden können.

`iot:DeleteConnection`

Stellt die Berechtigung dar, die Verbindung zu einem verbundenen MQTT-Client zu trennen. AWS IoT Core Die `iot:DeleteConnection` Berechtigung wird jedes Mal überprüft, wenn eine Anfrage gestellt wird, um die Verbindung zu einem Client gewaltsam zu trennen. Wenn Sie die Verbindung zu einem Client trennen, AWS IoT Core wird die Netzwerkverbindung des Clients geschlossen und optional der Sitzungsstatus gelöscht.

`iot:GetRetainedMessage`

Dies ist die Berechtigung zum Abrufen des Inhalts einer einzelnen beibehaltenen Nachricht. Bei gespeicherten Nachrichten handelt es sich um Nachrichten, die mit gesetztem RETAIN-Flag veröffentlicht und von AWS IoT Core gespeichert wurden. Informationen zur Berechtigung zum Abrufen einer Liste aller beibehaltenen Nachrichten des Kontos finden Sie unter `iot:ListRetainedMessages`.

`iot:ListRetainedMessages`

Dies ist die Berechtigung, zusammenfassende Informationen über die beibehaltenen Nachrichten des Kontos abzurufen, nicht jedoch den Inhalt der Nachrichten. Archivierte Nachrichten sind Nachrichten, die mit gesetztem RETAIN-Flag veröffentlicht und von gespeichert wurden AWS IoT Core. Der für diese Aktion angegebene Ressourcen-ARN muss * lauten. Informationen zur Berechtigung zum Abrufen des Inhalts einer einzelnen beibehaltenen Nachricht finden Sie unter `iot:GetRetainedMessage`.

`iot:Publish`

Dies ist die Berechtigung zum Veröffentlichen unter einem MQTT-Topic. Diese Berechtigung wird jedes Mal geprüft, wenn eine PUBLISH-Anforderung an den Broker gesendet wird. Sie können es verwenden, um Clients Nachrichten unter bestimmten Topic-Mustern veröffentlichen zu lassen.

> **Note**
>
> Um die Berechtigung `iot:Publish` zu erteilen, müssen Sie auch die Berechtigung `iot:Connect` erteilen.

`iot:Receive`

Stellt die Berechtigung dar, eine Nachricht von zu empfangen AWS IoT Core. Die `iot:Receive`-Berechtigung wird jedes Mal geprüft, wenn eine Nachricht an einen Client zugestellt wird. Da diese Berechtigung bei jedem Zustellvorgang geprüft wird, können Sie sie nutzen, um Clients, die zurzeit ein Topic abonnieren, Berechtigungen zu entziehen.

`iot:RetainPublish`

Dies ist die Berechtigung, eine MQTT-Nachricht mit gesetztem RETAIN-Flag zu veröffentlichen.

> **Note**
>
> Um die Berechtigung `iot:RetainPublish` zu erteilen, müssen Sie auch die Berechtigung `iot:Publish` erteilen.

`iot:Subscribe`

Dies ist die Berechtigung zum Abonnieren eines Topic-Filters. Diese Berechtigung wird jedes Mal geprüft, wenn eine SUBSCRIBE-Anforderung an den Broker gesendet wird. Verwenden Sie sie, um Clients das Abonnieren von Topics zu ermöglichen, die bestimmten Topic-Mustern entsprechen.

> **Note**
>
> Um die Berechtigung `iot:Subscribe` zu erteilen, müssen Sie auch die Berechtigung `iot:Connect` erteilen.

Aktionen für Geräteschattenrichtlinien

`iot:DeleteThingShadow`

Dies ist die Berechtigung zum Löschen des Geräteschattens eines Objekts. Die Berechtigung `iot:DeleteThingShadow` wird bei jeder Anforderung zum Löschen des Geräteschatteninhalts eines Objekts geprüft.

`iot:GetThingShadow`

Dies ist die Berechtigung zum Abrufen des Geräteschattens eines Objekts. Die Berechtigung `iot:GetThingShadow` wird bei jeder Anforderung zum Abrufen des Geräteschatteninhalts eines Objekts geprüft.

`iot:ListNamedShadowsForThing`

Dies ist die Berechtigung zum Auflisten der benannten Schatten eines Objekts. Die Berechtigung `iot:ListNamedShadowsForThing` wird bei jeder Anforderung zum Auflisten der benannten Schatten eines Objekts geprüft.

`iot:UpdateThingShadow`

Dies ist die Berechtigung zum Aktualisieren eines Device Shadow. Die Berechtigung `iot:UpdateThingShadow` wird bei jeder Anforderung zum Aktualisieren des Geräteschatteninhalts eines Objekts geprüft.

> **Note**
>
> Der Richtlinienaktionen für die Auftragsausführung gilt nur für den HTTP-TLS-Endpunkt. Wenn Sie den MQTT-Endpunkt verwenden, müssen Sie die in diesem Thema definierten MQTT-Richtlinienaktionen verwenden.
> Ein Beispiel für eine Richtlinie zur Auftragsausführung, die dies veranschaulicht, finden Sie unter the section called “Beispiel für grundlegende Auftragsrichtlinie”, das mit dem MQTT-Protokoll funktioniert.

Auftragsausführungen AWS IoT Core Richtlinienaktionen

`iotjobsdata:DescribeJobExecution`

Stellt die Berechtigung für den Abruf einer Auftragsausführung für ein bestimmtes Objekt dar. Die `iotjobsdata:DescribeJobExecution`-Berechtigung wird jedes Mal überprüft, wenn eine Anforderung zur Ausführung eines Auftrags gestellt wird.

`iotjobsdata:GetPendingJobExecutions`

Stellt die Berechtigung zum Abrufen der Liste der Aufträge dar, die sich nicht in einem Endstatus für ein Objekt befinden. Die Berechtigung `iotjobsdata:GetPendingJobExecutions` wird jedes Mal überprüft, wenn eine Anforderung zum Abrufen der Liste gestellt wird.

`iotjobsdata:UpdateJobExecution`

Stellt die Berechtigung zum Aktualisieren einer Auftragsausführung dar. Die Berechtigung `iotjobsdata:UpdateJobExecution` wird jedes Mal überprüft, wenn eine Anforderung zum Aktualisieren des Status einer Auftragsausführung gestellt wird.

`iotjobsdata:StartNextPendingJobExecution`

Stellt die Berechtigung zum Abrufen und Starten der nächsten ausstehenden Auftragsausführung für ein Objekt dar. (Um eine Auftragsausführung mit dem Status QUEUED auf IN_PROGRESS zu aktualisieren.) Die Berechtigung `iotjobsdata:StartNextPendingJobExecution` wird jedes Mal überprüft, wenn eine Anforderung zum Starten der nächsten ausstehenden Auftragsausführung gestellt wird.

AWS IoT Core Richtlinienaktion des Anmeldeinformationsanbieters

`iot:AssumeRoleWithCertificate`

Stellt die Berechtigung dar, den AWS IoT Core Anmeldeinformationsanbieter anzurufen, um eine IAM-Rolle mit zertifikatsbasierter Authentifizierung zu übernehmen. Die `iot:AssumeRoleWithCertificate` Berechtigung wird jedes Mal überprüft, wenn eine Anfrage an den AWS IoT Core Anmeldeinformationsanbieter gestellt wird, eine Rolle zu übernehmen.

AWS IoT Core Ressourcen für Aktionen

Um eine Ressource für eine AWS IoT Core Richtlinienaktion anzugeben, verwenden Sie den Amazon-Ressourcennamen (ARN) der Ressource. Alle Ressourcen-ARNs haben das folgende Format:

```
arn:partition:iot:region:AWS-account-ID:Resource-type/Resource-name
```

Die folgende Tabelle zeigt die Ressource, die für jeden Aktionstyp angegeben werden muss. Die ARN-Beispiele beziehen sich auf die Konto-ID `123456789012`, in der Partition `aws` und sind regionsspezifisch `us-east-1`. Weitere Informationen zu den Formaten für ARNs finden Sie im AWS Identity and Access Management Benutzerhandbuch unter Amazon Resource Names (ARNs).

Action	Ressourcentyp	Ressourcenname	ARN-Beispiel
`iot:Connect`	`client`	Die Client-ID des Clients	`arn:aws:iot:us-east-1:123456789012:client/myClientId`
`iot:DeleteConnection`	`client`	Die Client-ID des Clients	`arn:aws:iot:us-east-1:123456789012:client/myClientId`
`iot:DeleteThingShadow`	`thing`	Der Name des Objekts und gegebenenfalls der Name des Schattens	`arn:aws:iot:us-east-1:123456789012:thing/thingOne arn:aws:iot:us-east-1:123456789012:thing/thingOne/shadowOne`
`iotjobsdata:DescribeJobExecution`	`thing`	Der Name des Objekts	`arn:aws:iot:us-east-1:123456789012:thing/thingOne`
`iotjobsdata:GetPen`	`thing`	Der Name des Objekts	`arn:aws:iot:us-east-1:123456789012:thing/thingOne`

Action	Ressou ntyp	Ressourcenname	ARN-Beispiel
dingJobEx ecutions			
iot:GetRe tainedMes sage	topic	Ein beibehaltenes Nachricht en-Topic	arn:aws:iot:us-eas t-1:123456789012:topic/ myTopicName
iot:GetTh ingShadow	thing	Der Name des Objekts und gegebenenfalls der Name des Schattens	arn:aws:iot:us-eas t-1:123456789012:thing/ thingOne arn:aws:iot:us- east-1:123456789012:t hing/thingOne/shadowOne
iot:ListN amedShado wsForThing	Alle	Alle	*
iot:ListR etainedMe ssages	Alle	Alle	*
iot:Publish	topic	Eine Topic-Zeichenfolge	arn:aws:iot:us-eas t-1:123456789012:topic/ myTopicName
iot:Receive	topic	Eine Topic-Zeichenfolge	arn:aws:iot:us-eas t-1:123456789012:topic/ myTopicName
iot:Retai nPublish	topic	Ein Topic, das mit gesetztem RETAIN-Flag veröffentlicht werden soll	arn:aws:iot:us-eas t-1:123456789012:topic/ myTopicName

Action	Ressou ntyp	Ressourcenname	ARN-Beispiel
`iotjobsdata:StartNextPendingJobExecution`	`thing`	Der Name des Objekts	`arn:aws:iot:us-east-1:123456789012:thing/thingOne`
`iot:Subscribe`	`topicf er`	Eine Zeichenfolge für einen Topic-Filter	`arn:aws:iot:us-east-1:123456789012:topicfilter/myTopicFilter`
`iotjobsdata:UpdateJobExecution`	`thing`	Der Name des Objekts	`arn:aws:iot:us-east-1:123456789012:thing/thingOne`
`iot:UpdateThingShadow`	`thing`	Der Name des Objekts und gegebenenfalls der Name des Schattens	`arn:aws:iot:us-east-1:123456789012:thing/thingOne arn:aws:iot:us-east-1:123456789012:thing/thingOne/shadowOne`
`iot:AssumeRoleWithCertificate`	`rolea`	Das Rollenalias, das auf einen Rollen-ARN verweist	`arn:aws:iot:us-east-1:123456789012:rolealias/CredentialProviderRole_alias`

AWS IoT Core Richtlinienvariablen

AWS IoT Core definiert Richtlinienvariablen, die in AWS IoT Core Richtlinien im `Condition` Block `Resource` oder verwendet werden können. Bei Anwendung der Richtlinien werden die Variablen durch tatsächliche Werte ersetzt. Wenn beispielsweise ein Gerät mit der Client-ID 100-234-3456 mit dem AWS IoT Core Message Broker verbunden ist, wird die `iot:ClientId` Richtlinienvariable im Richtliniendokument durch 100-234-3456 ersetzt.

AWS IoT Core Richtlinien können Platzhalterzeichen verwenden und folgen einer ähnlichen Konvention wie IAM-Richtlinien. Ein in die Zeichenfolge eingefügtes * (Sternchen) kann als Platzhalter behandelt werden, der mit beliebigen Zeichen übereinstimmt. Sie können beispielsweise * verwenden, um mehrere MQTT-Topic-Namen im `Resource`-Attribut einer Richtlinie zu beschreiben. Die Zeichen + und # werden in einer Richtlinie als Literalzeichenfolgen behandelt. Ein Beispiel für eine Richtlinie, die die Verwendung von Platzhaltern veranschaulicht, finden Sie unter Verwendung von Platzhalterzeichen in MQTT und AWS IoT Core Richtlinien.

Sie können auch vordefinierte Richtlinienvariablen mit festen Werten verwenden, um Zeichen darzustellen, die andernfalls eine besondere Bedeutung haben. Zu diesen Sonderzeichen gehören `$(*)`, `$(?)` und `$($)`. Weitere Informationen zu Richtlinienvariablen und Sonderzeichen finden Sie unter IAM-Richtlinienelemente: Variablen und Tags und Erstellen einer Bedingung mit mehreren Schlüsseln oder Werten.

Themen

- Basic AWS IoT Core Richtlinienvariablen
- Objektrichtlinienvariablen
- X.509 Zertifikat AWS IoT Core Richtlinienvariablen

Basic AWS IoT Core Richtlinienvariablen

AWS IoT Core definiert die folgenden grundlegenden Richtlinienvariablen:

- `aws:SourceIp`: Die IP-Adresse des Clients, der mit dem AWS IoT Core Message Broker verbunden ist.
- `iot:ClientId`: Die Client-ID für die Verbindung zum AWS IoT Core Message Broker.
- `iot:DomainName`: Der Domainname des Clients, mit dem eine Verbindung hergestellt wurde AWS IoT Core.

Beispiele

- Beispiele für ClientIdund SourceIpRichtlinienvariablen
- Beispiele für die DomainName Richtlinienvariable iot:

Beispiele für **`ClientId`**und **`SourceIp`**Richtlinienvariablen

Die folgende AWS IoT Core Richtlinie zeigt eine Richtlinie, die Richtlinienvariablen verwendet. `aws:SourceIp`kann im Condition-Element Ihrer Richtlinie verwendet werden, um es Prinzipalen zu ermöglichen, API-Anfragen nur innerhalb eines bestimmten Adressbereichs zu stellen. Beispiele finden Sie unter Autorisieren Sie Benutzer und Cloud-Dienste zur Nutzung AWS IoT Jobs.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/clientid1"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/my/topic/${iot:ClientId}"
   ],
   "Condition": {
    "IpAddress": {
     "aws:SourceIp": "123.45.167.89"
    }
   }
  }
 ]
}
```

In diesen Beispielen `${iot:ClientId}` wird bei der Auswertung der Richtlinie durch die ID des Clients ersetzt, der mit dem AWS IoT Core Message Broker verbunden ist. Wenn Sie Richtlinienvariablen wie beispielsweise `${iot:ClientId}` verwenden, können Sie versehentlich

den Zugriff auf Themen ermöglichen. Wenn Sie beispielsweise eine Richtlinie nutzen, in der mit `${iot:ClientId}` ein bestimmter Topic-Filter angegeben wird:

```
{
 "Effect": "Allow",
 "Action": [
  "iot:Subscribe"
 ],
 "Resource": [
  "arn:aws:iot:us-east-1:123456789012:topicfilter/my/${iot:ClientId}/topic"
 ]
}
```

... kann ein Client mit der Client-ID + eine Verbindung zum &IoT; Message Broker herstellen. Damit könnte der Benutzer jedes Thema abonnieren, das dem Themenfilter `my/+/topic` entspricht. Nutzen Sie zum Schutz gegen solche Sicherheitslücken die Richtlinienaktion `iot:Connect`, mit der Sie steuern können, welche Client-IDs eine Verbindung herstellen dürfen. Mit dieser Richtlinie zum Beispiel dürfen nur die Clients mit der Client-ID `clientid1` eine Verbindung herstellen:

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/clientid"
   ]
  }
 ]
}
```

> **Note**
>
> Die Verwendung der Richtlinienvariable `${iot:ClientId}` mit `Connect` wird nicht empfohlen. Der `ClientId`-Wert wird nicht überprüft, sodass eine Anfügung mit einer anderen Client-ID die Validierung zwar bestehen, aber einen Verbindungsabbruch

verursachen kann. Da jede `ClientId` erlaubt ist, können mit einer zufälligen Client-ID die Richtlinien für Objektgruppen umgangen werden.

Beispiele für die DomainName Richtlinienvariable **`iot:`**

Sie können die `iot:DomainName` Richtlinienvariable hinzufügen, um einzuschränken, welche Domänen verwendet werden dürfen. Durch das Hinzufügen der `iot:DomainName` Richtlinienvariablen können Geräte nur eine Verbindung zu bestimmten konfigurierten Endpunkten herstellen.

Die folgende Richtlinie ermöglicht es Geräten, eine Verbindung mit der angegebenen Domäne herzustellen.

```
{
 "Version":"2012-10-17",
 "Statement": {
  "Sid": "AllowConnectionsToSpecifiedDomain",
  "Effect": "Allow",
  "Action": [
   "iot:Connect"
  ],
  "Resource": "arn:aws:iot:us-east-1:123456789012:client/clientid",
  "Condition": {
   "StringEquals": {
    "iot:DomainName": "d1234567890abcdefghij-ats.iot.us-east-1.amazonaws.com"
   }
  }
 }
}
```

Die folgende Richtlinie verweigert Geräten die Verbindung mit der angegebenen Domäne.

```
{
 "Version":"2012-10-17",
 "Statement": {
  "Sid": "DenyConnectionsToSpecifiedDomain",
  "Effect": "Deny",
  "Action": [
```

```
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/clientid",
   "Condition": {
    "StringEquals": {
     "iot:DomainName": "d1234567890abcdefghij-ats.iot.us-east-1.amazonaws.com"
    }
   }
  }
 }
```

Weitere Informationen zum Bedingungsoperator für Richtlinien finden Sie unter IAM-JSON-Richtlinienelemente: Bedingungsoperatoren. Weitere Informationen zu Domänenkonfigurationen finden Sie unter Was ist eine Domänenkonfiguration? .

Objektrichtlinienvariablen

Mithilfe von Dingrichtlinienvariablen können Sie AWS IoT Core Richtlinien schreiben, die Berechtigungen auf der Grundlage von Dingeigenschaften wie Dingnamen, Dingtypen und Dingattributwerten gewähren oder verweigern. Sie können Ding-Richtlinienvariablen verwenden, um dieselbe Richtlinie auf die Steuerung vieler AWS IoT Core Geräte anzuwenden. Weitere Informationen zur Gerätebereitstellung finden Sie unter Gerätebereitstellung.

Wenn Sie eine nicht ausschließliche Zuordnung von Dingen verwenden, kann dasselbe Zertifikat an mehrere Dinge angehängt werden. Um eine klare Zuordnung aufrechtzuerhalten und mögliche Konflikte zu vermeiden, müssen Sie Ihre Client-ID mit dem Namen des Dings abgleichen. In diesem Fall erhalten Sie den Namen des Dings aus der Client-ID in der `Connect` MQTT-Nachricht, die gesendet wird, wenn ein Ding eine Verbindung AWS IoT Core herstellt.

Beachten Sie Folgendes, wenn Sie Objektrichtlinienvariablen in AWS IoT Core -Richtlinien verwenden.

- Verwenden Sie die AttachThingPrincipalAPI, um Zertifikate oder Principals (authentifizierte Amazon Cognito Cognito-Identitäten) an eine Sache anzuhängen.
- Wenn eine nicht ausschließliche Ding-Assoziation vorhanden ist und Sie Ding-Namen durch Ding-Richtlinienvariablen ersetzen, muss der Wert von `clientId` in der MQTT-Verbindungsnachricht oder der TLS-Verbindung exakt mit dem Ding-Namen übereinstimmen.

Folgende Thing-Richtlinienvariablen stehen zur Verfügung:

- `iot:Connection.Thing.ThingName`

  Dies ergibt den Namen der Sache in der AWS IoT Core Registrierung, für die die Richtlinie ausgewertet wird. AWS IoT Core verwendet das Zertifikat, das das Gerät bei der Authentifizierung vorlegt, um zu ermitteln, welches Objekt zur Überprüfung der Verbindung verwendet werden soll. Diese Richtlinienvariable ist nur verfügbar, wenn ein Gerät eine Verbindung über MQTT oder MQTT über das Protokoll herstellt. WebSocket

- `iot:Connection.Thing.ThingTypeName`

  Dies wird in den Objekttyp aufgelöst, auf das die Richtlinie angewendet wurde. Die Client-ID der MQTT/WebSocket Verbindung muss mit dem Namen der Sache identisch sein. Diese Richtlinienvariable ist nur verfügbar, wenn eine Verbindung über MQTT oder MQTT über das WebSocket Protokoll hergestellt wird.

- `iot:Connection.Thing.Attributes[`*`attributeName`*`]`

  Dies wird in den Wert des angegebenen Attributs aufgelöst, das mit dem Thing verknüpft ist, auf das die Richtlinie angewendet wurde. Ein Thing kann bis zu 50 Attribute aufweisen. Jedes Attribut ist als Richtlinienvariable verfügbar: `iot:Connection.Thing.Attributes[`*`attributeName`*`]` wo *`attributeName`* ist der Name des Attributs. Die Client-ID der MQTT/WebSocket Verbindung muss mit dem Namen des Dings identisch sein. Diese Richtlinienvariable ist nur verfügbar, wenn eine Verbindung über MQTT oder MQTT über das WebSocket Protokoll hergestellt wird.

- `iot:Connection.Thing.IsAttached`

  `iot:Connection.Thing.IsAttached: ["true"]`erzwingt, dass nur die Geräte, die sowohl im AWS IoT Principal registriert als auch mit diesem verbunden sind, auf die in der Richtlinie enthaltenen Berechtigungen zugreifen können. Sie können diese Variable verwenden, um zu verhindern, dass ein Gerät eine Verbindung herstellt, AWS IoT Core wenn es ein Zertifikat vorlegt, das nicht an ein IoT-Ding in der AWS IoT Core Registrierung angehängt ist. Diese Variable hat Werte `true` oder `false` gibt an, dass das verbindende Ding über die API an das Zertifikat oder die Amazon Cognito Cognito-Identität in der Registrierung angehängt ist. AttachThingPrincipal Der Objektname wird als Client-ID verwendet.

Wenn Ihre Client-ID mit Ihrem Ding-Namen übereinstimmt oder wenn Sie Ihr Zertifikat ausschließlich an eine Sache anhängen, kann die Verwendung von Richtlinienvariablen in der Richtliniendefinition die Richtlinienverwaltung vereinfachen. Anstatt individuelle Richtlinien für jedes IoT-Ding zu erstellen, können Sie mithilfe der Ding-Richtlinienvariablen eine einzelne Richtlinie definieren. Diese Richtlinie

kann dynamisch auf alle Geräte angewendet werden. Im Folgenden finden Sie ein Beispiel für eine Richtlinie, die zeigt, wie sie funktioniert. Weitere Informationen finden Sie unter ???.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Condition": {
    "StringLike": {
     "iot:ClientId": "*${iot:Connection.Thing.Attributes[envType]}"
    }
   },
   "Effect": "Allow",
   "Action": "iot:Connect",
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/*"
  }
 ]
}
```

In diesem Richtlinienbeispiel können Dinge eine Verbindung herstellen, AWS IoT Core wenn ihre Client-ID mit dem Wert ihres `envType` Attributs endet. Nur Dinge mit einem passenden Client-ID-Muster dürfen eine Verbindung herstellen.

X.509 Zertifikat AWS IoT Core Richtlinienvariablen

X.509 Zertifikatsrichtlinienvariablen helfen beim Schreiben von AWS IoT Core Richtlinien. Diese Richtlinien gewähren Berechtigungen auf der Grundlage von X.509 Zertifikatsattributen. In den folgenden Abschnitten wird beschrieben, wie diese Zertifikatsrichtlinienvariablen verwendet werden.

> **Important**
>
> Wenn Ihr X.509 Zertifikat kein bestimmtes Zertifikatattribut enthält, die entsprechende Zertifikatsrichtlinienvariable jedoch in Ihrem Richtliniendokument verwendet wird, kann die Richtlinienbewertung zu unerwartetem Verhalten führen.

CertificateId

In der RegisterCertificateAPI `certificateId` erscheint das im Antworttext. Um Informationen zu Ihrem Zertifikat zu erhalten, verwenden Sie das `certificateId` in DescribeCertificate.

Ausstellerattribute

Die folgenden AWS IoT Core Richtlinienvariablen unterstützen das Zulassen oder Verweigern von Berechtigungen auf der Grundlage von Zertifikatsattributen, die vom Zertifikataussteller festgelegt wurden.

- `iot:Certificate.Issuer.DistinguishedNameQualifier`
- `iot:Certificate.Issuer.Country`
- `iot:Certificate.Issuer.Organization`
- `iot:Certificate.Issuer.OrganizationalUnit`
- `iot:Certificate.Issuer.State`
- `iot:Certificate.Issuer.CommonName`
- `iot:Certificate.Issuer.SerialNumber`
- `iot:Certificate.Issuer.Title`
- `iot:Certificate.Issuer.Surname`
- `iot:Certificate.Issuer.GivenName`
- `iot:Certificate.Issuer.Initials`
- `iot:Certificate.Issuer.Pseudonym`
- `iot:Certificate.Issuer.GenerationQualifier`

Subject-Attribute

Die folgenden AWS IoT Core Richtlinienvariablen unterstützen das Erteilen oder Verweigern von Berechtigungen auf der Grundlage der vom Zertifikatsaussteller festgelegten Attributen des Zertifikatsinhabers.

- `iot:Certificate.Subject.DistinguishedNameQualifier`
- `iot:Certificate.Subject.Country`
- `iot:Certificate.Subject.Organization`
- `iot:Certificate.Subject.OrganizationalUnit`
- `iot:Certificate.Subject.State`
- `iot:Certificate.Subject.CommonName`
- `iot:Certificate.Subject.SerialNumber`
- `iot:Certificate.Subject.Title`

- `iot:Certificate.Subject.Surname`
- `iot:Certificate.Subject.GivenName`
- `iot:Certificate.Subject.Initials`
- `iot:Certificate.Subject.Pseudonym`
- `iot:Certificate.Subject.GenerationQualifier`

X.509 Zertifikate bieten diesen Attributen die Option, einen oder mehrere Werte zu enthalten. Standardmäßig wird bei Attributen mit mehreren Werten der erste Wert zurückgegeben. So kann beispielsweise das Attribut `Certificate.Subject.Country` eine Liste von Ländernamen enthalten, aber wenn es in einer Richtlinie ausgewertet wird, wird `iot:Certificate.Subject.Country` durch den ersten Ländernamen ersetzt.

Einen spezifischen Attributwert, der nicht dem ersten Wert entspricht, können Sie unter Verwendung eines 1-basierten Indexes anfordern. `iot:Certificate.Subject.Country.1` wird z. B. durch den zweiten Ländernamen im Attribut `Certificate.Subject.Country` ersetzt. Wenn Sie einen Index angeben, der nicht vorhanden ist (wenn Sie zum Beispiel einen dritten Wert anfordern, obwohl dem Attribut nur zwei Werte zugeordnet sind), findet keine Ersetzung statt und die Autorisierung schlägt fehl. Sie können das Suffix `.List` verwenden, um alle Werte des Attributs anzugeben.

Attribute des alternativen Ausstellernamens

Die folgenden AWS IoT Core Richtlinienvariablen unterstützen die Erteilung oder Verweigerung von Berechtigungen auf der Grundlage von Attributen mit alternativen Namen, die vom Zertifikataussteller festgelegt wurden.

- `iot:Certificate.Issuer.AlternativeName.RFC822Name`
- `iot:Certificate.Issuer.AlternativeName.DNSName`
- `iot:Certificate.Issuer.AlternativeName.DirectoryName`
- `iot:Certificate.Issuer.AlternativeName.UniformResourceIdentifier`
- `iot:Certificate.Issuer.AlternativeName.IPAddress`

Attribute des alternativen Subjektnamens

Die folgenden AWS IoT Core Richtlinienvariablen unterstützen die Erteilung oder Verweigerung von Berechtigungen auf der Grundlage von Attributen mit alternativen Namen, die vom Zertifikataussteller festgelegt wurden.

- `iot:Certificate.Subject.AlternativeName.RFC822Name`
- `iot:Certificate.Subject.AlternativeName.DNSName`
- `iot:Certificate.Subject.AlternativeName.DirectoryName`
- `iot:Certificate.Subject.AlternativeName.UniformResourceIdentifier`
- `iot:Certificate.Subject.AlternativeName.IPAddress`

Weitere Attribute

Sie können sie verwenden`iot:Certificate.SerialNumber`, um den Zugriff auf AWS IoT Core Ressourcen auf der Grundlage der Seriennummer eines Zertifikats zu erlauben oder zu verweigern. Die Richtlinienvariable `iot:Certificate.AvailableKeys` enthält die Namen aller Zertifikat-Richtlinienvariablen, die Werte enthalten.

Verwenden von X.509 Zertifikatsrichtlinienvariablen

Dieses Thema enthält Einzelheiten zur Verwendung von Zertifikatsrichtlinienvariablen. X.509 Zertifikatrichtlinienvariablen sind wichtig, wenn Sie AWS IoT Core Richtlinien erstellen, die Berechtigungen auf der Grundlage von X.509 Zertifikatattributen gewähren. Wenn Ihr X.509 Zertifikat kein bestimmtes Zertifikatattribut enthält, die entsprechende Zertifikatsrichtlinienvariable jedoch in Ihrem Richtliniendokument verwendet wird, kann die Richtlinienbewertung zu unerwartetem Verhalten führen. Dies liegt daran, dass die fehlende Richtlinienvariable in der Richtlinienerklärung nicht bewertet wird.

In diesem Thema:

- X.509 Beispiel für ein Zertifikat
- Verwendung von Zertifikatsausstellerattributen als Zertifikatsrichtlinienvariablen
- Verwendung von Zertifikatsantragsattributen als Variablen für die Zertifikatsrichtlinie
- Verwendung von alternativen Namensattributen des Zertifikatausstellers als Variablen für die Zertifikatsrichtlinie
- Verwendung von Attributen mit alternativen Namen für den Antragsteller des Zertifikats als Variablen für die Zertifikatsrichtlinie
- Verwendung eines anderen Zertifikatsattributs als Zertifikatsrichtlinienvariable
- X.509 Einschränkungen der Variablen der Zertifikatsrichtlinie
- Beispielrichtlinien, die Zertifikatsrichtlinienvariablen verwenden

X.509 Beispiel für ein Zertifikat

Ein typisches X.509 Zertifikat könnte wie folgt aussehen. Dieses Beispielzertifikat enthält Zertifikatattribute. Bei der Bewertung von AWS IoT Core Richtlinien werden die folgenden Zertifikatsattribute als Zertifikatsrichtlinienvariablen aufgefüllt: `Serial Number`,`Issuer`,`Subject`,`X509v3 Issuer Alternative Name`, und`X509v3 Subject Alternative Name`.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            92:12:85:cb:b7:a5:e0:86
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, O=IoT Devices, OU=SmartHome, ST=WA, CN=IoT Devices Primary CA,
    GN=Primary CA1/initials=XY/dnQualifier=Example corp,
    SN=SmartHome/ title=CA1/pseudonym=Primary_CA/generationQualifier=2/serialNumber=987

        Validity
            Not Before: Mar 26 03:25:40 2024 GMT
            Not After : Apr 28 03:25:40 2025 GMT
        Subject: C=US, O=IoT Devices, OU=LightBulb, ST=NY, CN=LightBulb Device Cert,
    GN=Bulb/initials=ZZ/dnQualifier=Bulb001,
    SN=Multi Color/title=RGB/pseudonym=RGB Device/generationQualifier=4/
serialNumber=123
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    << REDACTED >>
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Digital Signature, Non Repudiation, Key Encipherment
            X509v3 Subject Alternative Name:
                DNS:example.com, IP Address:1.2.3.4, URI:ResourceIdentifier001,
 email:device1@example.com, DirName:/C=US/O=IoT/OU=SmartHome/CN=LightBulbCert
            X509v3 Issuer Alternative Name:
                DNS:issuer.com, IP Address:5.6.7.8, URI:PrimarySignerCA,
 email:primary@issuer.com, DirName:/C=US/O=Issuer/OU=IoT Devices/CN=Primary Issuer CA
    Signature Algorithm: sha256WithRSAEncryption
```

```
        << REDACTED >>
```

Verwendung von Zertifikatsausstellerattributen als Zertifikatsrichtlinienvariablen

Die folgende Tabelle enthält Einzelheiten dazu, wie die Attribute des Zertifikatsausstellers in einer AWS IoT Core Richtlinie aufgefüllt werden.

Ausstellerattribute, die in einer Richtlinie ausgefüllt werden sollen

Attribute des Zertifikatsausstellers	Variablen für die Zertifikatsrichtlinie
• C=US • o=IoT-Geräte • OU= SmartHome • ST=WA • CN=Primäre CA für IoT-Geräte • GN=Primäre CA1 • Initialen=XY • dnqualifier=Beispiel Corp • SN= SmartHome • Titel=CA1 • Pseudonym= Primary_CA • Qualifikator für die Generation = 2 • Seriennummer = 987	• `iot:Certificate.Issuer.Country = US` • `iot:Certificate.Issuer.Organization = IoT Devices` • `iot:Certificate.Issuer.OrganizationalUnit = SmartHome` • `iot:Certificate.Issuer.State = WA` • `iot:Certificate.Issuer.CommonName = IoT Devices Primary CA` • `iot:Certificate.Issuer.GivenName = Primary CA1` • `iot:Certificate.Issuer.initials = XY` • `iot:Certificate.Issuer.DistinguishedNameQualifier = Example corp` • `iot:Certificate.Issuer.Surname = SmartHome` • `iot:Certificate.Issuer.Title = CA1` • `iot:Certificate.Issuer.Pseudonym = Primary_CA` • `iot:Certificate.Issuer.GenerationQualifier = 2` • `iot:Certificate.Issuer.SerialNumber = 987`

Verwendung von Zertifikatsantragsattributen als Variablen für die Zertifikatsrichtlinie

Die folgende Tabelle enthält Einzelheiten dazu, wie die Attribute des Zertifikatssubjekts in einer AWS IoT Core Richtlinie aufgefüllt werden.

Betreff-Attribute, die in einer Richtlinie ausgefüllt werden sollen

Betreff-Attribute des Zertifikats	Variablen für die Zertifikatsrichtlinie
• C=US • o=IoT-Geräte • ST=NY • CN= Gerätezertifikat LightBulb • GN=Glühbirne • Initialen=ZZ • DNQualifier = BULB001 • sn=Mehrfarbig • Titel=RGB • Pseudonym=RGB-Gerät • Qualifier für die Generierung = 4 • Seriennummer = 123	• `iot:Certificate.Subject.Country = US` • `iot:Certificate.Subject.Organization = IoT Devices` • `iot:Certificate.Subject.State = NY` • `iot:Certificate.Subject.CommonName = LightBulb Device Cert` • `iot:Certificate.Subject.GivenName = Bulb` • `iot:Certificate.Subject.initials = ZZ` • `iot:Certificate.Subject.DistinguishedNameQualifier = Bulb001` • `iot:Certificate.Subject.Surname = Multi Color` • `iot:Certificate.Subject.Title = RGB` • `iot:Certificate.Subject.Pseudonym = RGB Device` • `iot:Certificate.Subject.GenerationQualifier = 4` • `iot:Certificate.Subject.SerialNumber = 123`

Verwendung von alternativen Namensattributen des Zertifikatausstellers als Variablen für die Zertifikatsrichtlinie

Die folgende Tabelle enthält Einzelheiten darüber, wie alternative Namensattribute von Zertifikatsausstellern in eine AWS IoT Core Richtlinie eingefügt werden.

Attribute für alternative Namen des Ausstellers, die in eine Richtlinie eingetragen werden sollen

Alternativer Name des X509v3-Emittenten	Attribut in einer Richtlinie
• DNS:issuer.com • IP Address:5.6.7.8 • URI:PrimarySignerCA • E-Mail: primary@issuer.com • DirName: /C= US/O = Issuer/OU =IoT Devices/CN =Primäre Emittentin CA	• `iot:Certificate.Issuer.AlternativeName.DNSName = issuer.com` • `iot:Certificate.Issuer.AlternativeName.IPAddress = 5.6.7.8` • `iot:Certificate.Issuer.AlternativeName.UniformResourceIdentifier = PrimarySignerCA` • `iot:Certificate.Issuer.AlternativeName.RFC822Name = primary@issuer.com` • `iot:Certificate.Issuer.AlternativeName.DirectoryName = cn=Primary Issuer CA,ou=IoT Devices,o=Issuer,c=US`

Verwendung von Attributen mit alternativen Namen für den Antragsteller des Zertifikats als Variablen für die Zertifikatsrichtlinie

Die folgende Tabelle enthält Einzelheiten dazu, wie die Attribute für alternative Namen von Zertifikatsempfängern in einer AWS IoT Core Richtlinie aufgefüllt werden.

Attribute mit alternativen Namen für Antragsteller, die in einer Richtlinie ausgefüllt werden sollen

X509v3 Alternativer Name des Betreffs	Attribut in einer Richtlinie
• DNS:example.com • IP Address:1.2.3.4 • URI:ResourceIdentifier001 • E-Mail: device1@example.com	• `iot:Certificate.Subject.AlternativeName.DNSName = example.com` • `iot:Certificate.Subject.AlternativeName.IPAddress = 1.2.3.4` • `iot:Certificate.Subject.AlternativeName.UniformResourceIdentifier = ResourceIdentifier001`

X509v3 Alternativer Name des Betreffs	Attribut in einer Richtlinie
• DirName: /C= = US/O = IoT/OU SmartHome/CN LightBulbCert	• `iot:Certificate.Subject.AlternativeName.RFC822Name = device1@example.com` • `iot:Certificate.Subject.AlternativeName.DirectoryName = cn=LightBulbCert,ou=SmartHome,o=IoT,c=US`

Verwendung eines anderen Zertifikatsattributs als Zertifikatsrichtlinienvariable

Die folgende Tabelle enthält Einzelheiten dazu, wie andere Zertifikatsattribute in eine AWS IoT Core Richtlinie aufgenommen werden.

Andere Attribute, die in einer Richtlinie aufgefüllt werden sollen

Anderes Zertifikatsattribut	Richtlinienvariable für Zertifikate
`Serial Number: 92:12:85:cb:b7:a5:e0:86`	`iot:Certificate.SerialNumber = 105256223 89124227206`

X.509 Einschränkungen der Variablen der Zertifikatsrichtlinie

Die folgenden Einschränkungen gelten für X.509 Zertifikatsrichtlinienvariablen:

Fehlende Richtlinienvariablen

Wenn Ihr X.509 Zertifikat kein bestimmtes Zertifikatattribut enthält, die entsprechende Zertifikatsrichtlinienvariable jedoch in Ihrem Richtliniendokument verwendet wird, kann die Richtlinienbewertung zu unerwartetem Verhalten führen. Dies liegt daran, dass die fehlende Richtlinienvariable in der Richtlinienerklärung nicht bewertet wird.

SerialNumber Format des Zertifikats

AWS IoT Core behandelt die Seriennummer des Zertifikats als Zeichenkettendarstellung einer dezimalen Ganzzahl. Wenn eine Richtlinie beispielsweise nur Verbindungen zulässt, deren Client-ID mit der Seriennummer des Zertifikats übereinstimmt, muss die Client-ID die Seriennummer im Dezimalformat sein.

Platzhalter

Wenn Platzhalterzeichen in Zertifikatattributen vorhanden sind, wird die Richtlinienvariable nicht durch den Wert des Zertifikatsattributs ersetzt. Dadurch bleibt der `${policy-variable}` Text im Richtliniendokument erhalten. Dies kann zu einem Autorisierungsfehler führen. Die folgenden Platzhalterzeichen können verwendet werden: `*`, `$`, `+`, `?` und `#`.

Array-Felder

Arrays in Zertifikatattributen sind auf fünf Elemente beschränkt. Zusätzliche Elemente werden ignoriert.

Länge der Zeichenfolge

Alle Zeichenfolgenwerte sind auf maximal 1.024 Zeichen beschränkt. Wenn ein Zertifikatsattribut eine Zeichenfolge enthält, die länger als 1024 Zeichen ist, wird die Richtlinienvariable nicht durch den Wert des Zertifikatsattributs ersetzt. Dadurch bleibt das `${policy-variable}` im Richtliniendokument. Dies kann zu einem Autorisierungsfehler führen.

Sonderzeichen

Jedem Sonderzeichen, wie `,`, `'`, `\`, `+`, `=`, `<`, `>` und `;`, muss ein umgekehrter Schrägstrich (`\`) vorangestellt werden, wenn es in einer Richtlinienvariablen verwendet wird. Beispielsweise wird `Amazon Web Services O=Amazon.com Inc. L=Seattle ST=Washington C=US` zu `Amazon Web Service O\=Amazon.com Inc. L\=Seattle ST\=Washington C\=US`.

Beispielrichtlinien, die Zertifikatsrichtlinienvariablen verwenden

Das folgende Richtliniendokument ermöglicht Verbindungen mit einer Client-ID, die der Seriennummer des Zertifikats entspricht, und das Veröffentlichen zu dem Thema, das dem Muster entspricht:`${iot:Certificate.Subject.Organization}/device-stats/${iot:ClientId}/*`.

> **Important**
>
> Wenn Ihr X.509 Zertifikat kein bestimmtes Zertifikatattribut enthält, aber die entsprechende Zertifikatsrichtlinienvariable in Ihrem Richtliniendokument verwendet wird, kann die Richtlinienbewertung zu unerwartetem Verhalten führen. Dies liegt daran, dass die fehlende Richtlinienvariable in der Richtlinienerklärung nicht bewertet wird. Wenn Sie beispielsweise das folgende Richtliniendokument an ein Zertifikat anhängen, das das `iot:Certificate.Subject.Organization` Attribut nicht enthält, werden die

`iot:Certificate.Subject.Organization` Zertifikatsrichtlinienvariablen bei der Richtlinienbewertung nicht aufgefüllt.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Certificate.SerialNumber}"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/
${iot:Certificate.Subject.Organization}/device-stats/${iot:ClientId}/*"
   ]
  }
 ]
}
```

Sie können auch den Bedingungsoperator Null verwenden, um sicherzustellen, dass die in einer Richtlinie verwendeten Zertifikatsrichtlinienvariablen bei der Richtlinienbewertung aufgefüllt werden. Im folgenden Richtliniendokument sind Zertifikate nur `iot:Connect` zulässig, wenn die Attribute Certificate Serial Number und Certificate Subject Common Name vorhanden sind.

Alle Variablen der Zertifikatsrichtlinie haben Zeichenkettenwerte, sodass alle Bedingungsoperatoren vom Typ „Zeichenfolge“ unterstützt werden.

```
{
 "Version":"2012-10-17",
```

```
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/*"
   ],
   "Condition": {
    "Null": {
     "iot:Certificate.SerialNumber": "false",
     "iot:Certificate.Subject.CommonName": "false"
    }
   }
  }
 ]
}
```

Cross-service verwirrter Stellvertreter, Prävention

Das Confused-Deputy-Problem ist ein Sicherheitsproblem, bei dem eine Entität, die nicht über die Berechtigung zum Ausführen einer Aktion verfügt, eine Entität mit größeren Rechten zwingen kann, die Aktion auszuführen. Im AWS Fall eines dienststellenübergreifenden Identitätswechsels kann es zu einem Problem mit verwirrten Stellvertretern kommen. Cross-service Ein Identitätswechsel kann auftreten, wenn ein Dienst (der anrufende Dienst) einen anderen Dienst (den angerufenen Dienst) aufruft. Der Anruf-Service kann so manipuliert werden, dass er seine Berechtigungen verwendet, um auf die Ressourcen eines anderen Kunden zu reagieren, auf die er sonst nicht zugreifen dürfte. Um dies zu verhindern, bietet AWS Tools, mit denen Sie Ihre Daten für alle Services mit Serviceprinzipalen schützen können, die Zugriff auf Ressourcen in Ihrem Konto erhalten haben.

Um die Berechtigungen einzuschränken, die AWS IoT der Ressource einem anderen Dienst zugewiesen werden, empfehlen wir die Verwendung der Kontextschlüssel `aws:SourceArn` und der `aws:SourceAccount` globalen Bedingungsschlüssel in Ressourcenrichtlinien. Wenn Sie beide globalen Bedingungskontextschlüssel verwenden, müssen der `aws:SourceAccount`-Wert und das Konto im `aws:SourceArn`-Wert dieselbe Konto-ID verwenden, wenn sie in derselben Richtlinienanweisung verwendet werden.

Der effektivste Weg, um sich vor dem Verwirrter-Stellvertreter-Problem zu schützen, ist die Verwendung des `aws:SourceArn` globalen Bedingungskontextschlüssels mit dem vollständigen

Amazon-Ressourcenname (ARN) der Ressource. Für AWS IoT`aws:SourceArn` müssen Sie das Format einhalten: `arn:aws:iot:region:account-id:resource-type/resource-id` für ressourcenspezifische Berechtigungen oder`arn:aws:iot:region:account-id:*`. Die Ressourcen-ID kann der Name oder die ID der zulässigen Ressource oder eine Platzhalterangabe der zulässigen Ressourcen-IDs sein. Stellen Sie sicher, dass das *region* mit Ihrer AWS IoT Region und das mit Ihrer *account-id* Kundenkonto-ID übereinstimmt.

Das folgende Beispiel zeigt, wie Sie das Problem mit dem verwirrten Stellvertreter verhindern können, indem Sie die Kontextschlüssel `aws:SourceArn` und die `aws:SourceAccount` globale Bedingung in der AWS IoT Rollenvertrauensrichtlinie verwenden. Weitere Beispiele finden Sie unter Ausführliche Beispiele für die Prävention verwirrter Stellvertreter.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"iot.amazonaws.com"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"123456789012"
        },
            "ArnLike":{
               "aws:SourceArn":"arn:aws:iot:us-east-1:123456789012:*"
         }
          }
      }
   ]
}
```

> **Note**
>
> Wenn Sie die Fehlermeldung „Zugriff verweigert“ erhalten, kann dies daran liegen, dass die Dienstintegration mit AWS Security Token Service (STS) die Kontextschlüssel `aws:SourceArn` und die `aws:SourceAccount` Kontextschlüssel nicht unterstützt.

Ausführliche Beispiele für die Prävention verwirrter Stellvertreter

Dieser Abschnitt enthält ausführliche Beispiele dafür, wie das Problem der verwirrten Stellvertreter verhindert werden kann, indem die Kontextschlüssel aws:SourceArn und die aws:SourceAccount globalen Bedingungsschlüssel in der AWS IoT Rollenvertrauensrichtlinie verwendet werden.

- Flottenbereitstellung
- JITP
- Anbieter von Anmeldeinformationen

Flottenbereitstellung

Sie können die Flottenbereitstellung mithilfe einer Provisioning-Vorlagenressource konfigurieren. Wenn eine Bereitstellungsvorlage auf eine Bereitstellungsrolle verweist, kann die Vertrauensrichtlinie dieser Rolle die Bedingungsschlüssel und die `aws:SourceArn` Bedingungsschlüssel enthalten. `aws:SourceAccount` Diese Schlüssel begrenzen die Ressourcen, für die die Konfiguration die Anforderung aufrufen kann. `sts:AssumeRole`

Die Rolle mit der folgenden Vertrauensrichtlinie kann nur vom IoT-Prinzipal (`iot.amazonaws.com`) für die in der `SourceArn` angegebene Bereitstellungsvorlage übernommen werden.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"iot.amazonaws.com"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"123456789012"
        },
            "ArnLike":{
               "aws:SourceArn":"arn:aws:iot:us-
east-1:123456789012:provisioningtemplate/example_template"
        }
         }
      }
```

```
    ]
}
```

JITP

Bei der Just-in-Time-Bereitstellung (JITP) können Sie entweder die Bereitstellungsvorlage als separate Ressource von der CA verwenden oder den Vorlagentext und die Rolle als Teil der CA-Zertifikatskonfiguration definieren. Der Wert von `aws:SourceArn` in der AWS IoT Rollenvertrauensrichtlinie hängt davon ab, wie Sie die Bereitstellungsvorlage definieren.

Definieren Sie die Bereitstellungsvorlage als separate Ressource

Wenn Sie Ihre Bereitstellungsvorlage als separate Ressource definieren, `aws:SourceArn` kann der Wert von sein. "`arn:aws:iot:`*`region`*`:`*`account-id`*`:provisioningtemplate/`*`example_template`*"" Sie können dasselbe Richtlinienbeispiel in Flottenbereitstellung verwenden.

Definieren einer Bereitstellungsvorlage in einem CA-Zertifikat

Wenn Sie Ihre Bereitstellungsvorlage innerhalb einer CA-Zertifikatsressource definieren, `aws:SourceArn` kann der Wert von oder sein"`arn:aws:iot:`*`region`*`:`*`account-id`*`:cacert/`*`cert_id`*"". "`arn:aws:iot:`*`region`*`:`*`account-id`*`:cacert/`*"" Sie können einen Platzhalter verwenden, wenn die Ressourcenkennung, z. B. die ID eines CA-Zertifikats, zum Zeitpunkt der Erstellung unbekannt ist.

Die Rolle mit der folgenden Vertrauensrichtlinie kann nur vom IoT-Prinzipal (`iot.amazonaws.com`) für das in der angegebene CA-Zertifikat übernommen werden`SourceArn`.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"iot.amazonaws.com"
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
```

```
                "aws:SourceAccount":"123456789012"
            },
                "ArnLike":{
                    "aws:SourceArn":"arn:aws:iot:us-
east-1:123456789012:cacert/
8ecde6884f3d87b1125ba31ac3fcb13d7016de7f57cc904fe1cb97c6ae98196e"
            }
             }
        }
    ]
}
```

Beim Erstellen eines CA-Zertifikats können Sie in der Registrierungskonfiguration auf eine Bereitstellungsrolle verweisen. Die Vertrauensrichtlinie der Bereitstellungsrolle kann verwendet werden`aws:SourceArn`, um einzuschränken, für welche Ressourcen die Rolle verwendet werden kann. Beim ersten RegisterCACertificate-Aufruf zur Registrierung des CA-Zertifikats müssten Sie jedoch nicht den ARN des CA-Zertifikats in der Bedingung angeben. `aws:SourceArn`

Um dieses Problem zu umgehen, d. h. um die Vertrauensrichtlinie der Bereitstellungsrolle für das spezifische CA-Zertifikat anzugeben, mit dem registriert ist AWS IoT Core, können Sie wie folgt vorgehen:

- Rufen Sie zunächst RegisterCACertificate auf, ohne den Parameter anzugeben. `RegistrationConfig`
- Nachdem das CA-Zertifikat bei registriert wurde AWS IoT Core, rufen Sie UpdateCACertificate auf.

  Geben Sie im updateCACertificate-Aufruf eine an, `RegistrationConfig` die die Vertrauensrichtlinie für die Bereitstellungsrolle enthält, die auf den ARN des neu registrierten CA-Zertifikats `aws:SourceArn` festgelegt ist.

Anbieter von Anmeldeinformationen

Verwenden Sie für AWS IoT Core Credential Provider dasselbe, das AWS-Konto Sie zum Erstellen des Rollenalias in verwendenaws:SourceAccount, und geben Sie eine Anweisung an, die dem Ressourcen-ARN des Rolealias-Ressourcentyps in entspricht. `aws:SourceArn` Wenn Sie eine IAM-Rolle für die Verwendung mit dem AWS IoT Core Anmeldeinformationsanbieter erstellen, müssen Sie die ARNs aller Rollenaliase, die möglicherweise die Rolle übernehmen müssen, in die `aws:SourceArn` Bedingung aufnehmen, um so die dienstübergreifende Anfrage zu autorisieren. `sts:AssumeRole`

Die Rolle mit der folgenden Vertrauensrichtlinie kann nur vom Principal von AWS IoT Core Credential Provider (`credentials.iot.amazonaws.com`) für den in der angegebenen roleAlias übernommen werden. `SourceArn` Wenn ein Principal versucht, Anmeldeinformationen für einen anderen Rollenalias als den in der `aws:SourceArn` Bedingung angegebenen abzurufen, wird die Anfrage abgelehnt, auch wenn dieser andere Rollenalias auf dieselbe IAM-Rolle verweist.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:iot:us-
east-1:123456789012:rolealias/example_rolealias"
        }
      }
    }
  ]
}
```

AWS IoT Core Beispiele für Richtlinien

Die Beispielrichtlinien in diesem Abschnitt veranschaulichen die Richtliniendokumente, die zur Ausführung allgemeiner Aufgaben in AWS IoT Core verwendet werden. Sie können sie als Beispiele verwenden, um darauf aufbauend mit der Erstellung der Richtlinien für Ihre Lösungen zu beginnen.

Für die Beispiele in diesem Abschnitt werden die folgenden Richtlinienelemente verwendet:

- the section called “AWS IoT Core politische Maßnahmen”
- the section called “AWS IoT Core Ressourcen für Aktionen”
- the section called “Identity-based Beispiele für politische Maßnahmen”

- the section called “Basic AWS IoT Core Richtlinienvariablen”
- the section called “X.509 Zertifikat AWS IoT Core Richtlinienvariablen”

Richtlinienbeispiele in diesem Abschnitt:

- Beispiel für die Verbinden-Richtlinie
- Publish/Subscribe Beispiele für Richtlinien
- Beispiele zu den Verbinden- und Veröffentlichen-Richtlinien
- Beispielrichtlinien für beibehaltene Nachrichten
- Beispiele für die Zertifikatrichtlinie
- Beispiel für Objektrichtlinien
- Beispiel für grundlegende Auftragsrichtlinie
- Beispiele für Richtlinien für Direktnachrichten

Beispiel für die Verbinden-Richtlinie

Die folgende Richtlinie verweigert den Zugriff auf Client-IDs `client1` und das Herstellen einer Verbindung `client2` zu Client-IDs AWS IoT Core, ermöglicht aber Geräten, eine Verbindung über eine Client-ID herzustellen. Die Client-ID entspricht dem Namen einer Sache, die in der AWS IoT Core Registrierung registriert und dem Prinzipal zugeordnet ist, der für die Verbindung verwendet wird:

> **Note**
>
> Für registrierte Geräte empfehlen wir, Objektrichtlinienvariablen für `Connect`-Aktionen zu verwenden und das Objekt an den Prinzipal anzuhängen, der für die Verbindung verwendet wird.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Deny",
   "Action": [
    "iot:Connect"
```

```
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:client/client1",
     "arn:aws:iot:us-east-1:123456789012:client/client2"
    ]
   },
   {
    "Effect": "Allow",
    "Action": [
     "iot:Connect"
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
    ],
    "Condition": {
     "Bool": {
      "iot:Connection.Thing.IsAttached": "true"
     }
    }
   }
 ]
}
```

Die folgende Richtlinie erteilt die Erlaubnis, eine Verbindung AWS IoT Core mit der Client-ID herzustellen`client1`. Dieses Richtlinienbeispiel bezieht sich auf nicht registrierte Geräte.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/client1"
   ]
  }
 ]
}
```

Richtlinienbeispiele für persistente MQTT-Sitzungen

`connectAttributes` ermöglichen es Ihnen, in Ihren IAM-Richtlinien `PersistentConnect` und `LastWill` anzugeben, welche Attribute Sie in Ihrer Connect-Meldung verwenden möchten. Weitere Informationen finden Sie unter ConnectAttributes verwenden.

Die folgende Richtlinie ermöglicht, eine Verbindung mit dem `PersistentConnect`-Feature herzustellen:

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "PersistentConnect"
     ]
    }
   }
  }
 ]
}
```

Die folgende Richtlinie untersagt `PersistentConnect`, andere Features hingegen sind erlaubt:

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
```

```
   "Condition": {
    "ForAllValues:StringNotEquals": {
     "iot:ConnectAttributes": [
      "PersistentConnect"
     ]
    }
   }
  }
 ]
}
```

Die oben genannte Richtlinie kann auch mit `StringEquals` ausgedrückt werden, sodass alle anderen Features, einschließlich neuer Features, zulässig sind:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1"
        },
        {
            "Effect": "Deny",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
            }
        }
        }
    ]
}
```

Die folgende Richtlinie erlaubt das Herstellen einer Verbindung über beide Methoden, `PersistentConnect` und `LastWill`. Alle anderen neue Features sind nicht erlaubt:

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "PersistentConnect",
      "LastWill"
     ]
    }
   }
  }
 ]
}
```

Die folgende Richtlinie ermöglicht eine nahtlose Verbindung von Clients mit oder ohne `LastWill`. Andere anderen Features sind nicht zulässig:

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "iot:Connect"
        ],
        "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
        "Condition": {
            "StringEquals": {
                "iot:ConnectAttributes": "LastWill"
        }
        }
```

```
    }]
}
```

Die folgende Richtlinie erlaubt nur Verbindungen unter Verwendung von Standardfeatures:

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": []
    }
   }
  }
 ]
}
```

Die folgende Richtlinie erlaubt nur die Verbindung mit `PersistentConnect`. Jedes neue Feature ist zulässig, solange die Verbindung `PersistentConnect` verwendet:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
```

```
                    ]
                }
            }
            }
        ]
}
```

Die folgende Richtlinie besagt, dass die Verbindung sowohl `PersistentConnect` als auch `LastWill` verwendet werden muss. Neue Features sind nicht zulässig:

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "PersistentConnect",
      "LastWill"
     ]
    }
   }
  },
  {
   "Effect": "Deny",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "*",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "PersistentConnect"
     ]
    }
   }
```

```
  },
  {
   "Effect": "Deny",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "*",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "LastWill"
     ]
    }
   }
  },
  {
   "Effect": "Deny",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "*",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": []
    }
   }
  }
 ]
}
```

Die folgende Richtlinie darf `PersistentConnect` nicht verwenden, kann aber `LastWill` verwenden; alle anderen neuen Features sind nicht erlaubt:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "iot:Connect"
            ],
```

```
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
            }
        }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "iot:ConnectAttributes": [
                        "LastWill"
                    ]
            }
        }
        }
    ]
}
```

Die folgende Richtlinie erlaubt Verbindungen nur für Clients, die einen `LastWill` mit dem Topic `"my/lastwill/topicName"` haben; alle anderen Features sind erlaubt, solange sie das Topic `LastWill` verwenden:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
            "Condition": {
                "ArnEquals": {
```

```
                "iot:LastWillTopic": "arn:aws:iot:us-east-1:123456789012:topic/my/
lastwill/topicName"
            }
        }
        }
    ]
}
```

Die folgende Richtlinie erlaubt nur die Verbindung mit einem spezifischen `LastWillTopic`; andere Features sind zulässig, solange es das `LastWillTopic` verwendet:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
            "Condition": {
                "ArnEquals": {
                "iot:LastWillTopic": "arn:aws:iot:us-east-1:123456789012:topic/my/
lastwill/topicName"
            }
        }
        },
        {
            "Effect": "Deny",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
            }
        }
        }
```

```
    ]
}
```

Publish/Subscribe Beispiele für Richtlinien

Welche Richtlinie Sie verwenden, hängt davon ab, wie Sie eine Verbindung herstellen AWS IoT Core. Sie können eine Verbindung herstellen, AWS IoT Core indem Sie einen MQTT-Client, HTTP oder WebSocket verwenden. Wenn Sie eine Verbindung mit einem MQTT-Client herstellen, authentifizieren Sie sich mit einem Zertifikat. X.509 Wenn Sie eine Verbindung über HTTP oder das WebSocket Protokoll herstellen, authentifizieren Sie sich mit Signature Version 4 und Amazon Cognito.

> **Note**
>
> Für registrierte Geräte empfehlen wir, Objektrichtlinienvariablen für `Connect`-Aktionen zu verwenden und das Objekt an den Prinzipal anzuhängen, der für die Verbindung verwendet wird.

In diesem Abschnitt:

- Verwendung von Platzhalterzeichen in MQTT und AWS IoT Core Richtlinien
- Richtlinien zum Veröffentlichen, Abonnieren und Empfangen von Nachrichten zu to/from bestimmten Themen
- Richtlinien zum Veröffentlichen, Abonnieren und Empfangen von Nachrichten, Themen mit einem bestimmten Präfix to/from
- Richtlinien zum Veröffentlichen, Abonnieren und Empfangen von Nachrichten — Themen, die für jedes Gerät spezifisch sind to/from
- Richtlinien zum Veröffentlichen, Abonnieren und Empfangen von Nachrichten — to/from Themen, bei denen das Thing-Attribut im Themennamen enthalten ist
- Richtlinien, um die Veröffentlichung von Nachrichten zu untergeordneten Topics eines Topic-Namens zu verweigern
- Richtlinien, um den Empfang von Nachrichten von untergeordneten Topics eines Topic-Namens zu verweigern
- Richtlinien zum Abonnieren von Topics mit MQTT-Platzhalterzeichen
- Richtlinien für HTTP und Clients WebSocket

Verwendung von Platzhalterzeichen in MQTT und AWS IoT Core Richtlinien

MQTT und AWS IoT Core Richtlinien haben unterschiedliche Platzhalterzeichen, und Sie sollten sie nach reiflicher Überlegung auswählen. In MQTT # werden die Platzhalterzeichen + und in MQTT-Themenfiltern verwendet, um mehrere Themennamen zu abonnieren. AWS IoT Core Richtlinien verwenden * und ? als Platzhalterzeichen und folgen den Konventionen der IAM-Richtlinien. In einem Richtliniendokument steht * für eine beliebige Kombination von mehreren Zeichen und ein Fragezeichen ? entspricht einem beliebigen einzelnen Zeichen. In Richtliniendokumenten werden die MQTT-Platzhalterzeichen + und # als Zeichen ohne besondere Bedeutung behandelt. Verwenden Sie die Platzhalterzeichen * und ? anstelle der MQTT-Platzhalterzeichen, um mehrere Topic-Namen und -Filter im `resource`-Attribut einer Richtlinie zu beschreiben,

Beachten Sie bei der Auswahl der Platzhalterzeichen für die Verwendung in einem Richtliniendokument, dass das * Zeichen nicht auf eine einzelne Themenebene beschränkt ist. Das + Zeichen ist in einem MQTT-Themenfilter auf eine einzelne Themenebene beschränkt. Ziehen Sie die Verwendung mehrerer ?-Zeichen in Betracht, um eine Platzhalterspezifikation auf eine einzige MQTT-Topic-Filterebene zu beschränken, Weitere Informationen zur Verwendung von Platzhalterzeichen in einer Richtlinienressource und weitere Beispiele für deren Übereinstimmung finden Sie unter Verwenden von Platzhaltern in Ressourcen-ARNs.

Die folgende Tabelle zeigt die verschiedenen Platzhalterzeichen, die in MQTT und AWS IoT Core - Richtlinien für MQTT-Clients verwendet werden.

Platzhalterzeichen	Ist ein MQTT-Platzhalterzeichen	Beispiel in MQTT	Ist ein AWS IoT Core Richtlinien-Platzhalterzeichen	Beispiel für AWS IoT Core Richtlinien für MQTT-Clients
#	Ja	`some/#`	Nein	N/A
+	Ja	`some/+/topic`	Nein	N/A
*	Nein	N/A	Ja	`topicfilter/some/*/topic` `topicfilter/some/sensor*/topic`

Platzhalterzeichen	Ist ein MQTT-Platzhalterzeichen	Beispiel in MQTT	Ist ein AWS IoT Core Richtlinien-Platzhalterzeichen	Beispiel für AWS IoT Core Richtlinien für MQTT-Clients

?	Nein	N/A	Ja	`topic/some/?????/topic`
				
				`topicfilter/some/sensor???/topic`

Richtlinien zum Veröffentlichen, Abonnieren und Empfangen von Nachrichten zu to/from bestimmten Themen

Im Folgenden finden Sie Beispiele für registrierte und nicht registrierte Geräte zum Veröffentlichen, Abonnieren und Empfangen von Nachrichten to/from zum Thema „some_specific_topic“. In diesen Beispielen wird ebenfalls hervorgehiben, dass `Publish` und `Receive` „topic“ als Ressource und `Subscribe` „topicfilter“ als Ressource verwenden.

Registered devices

Für Geräte, die in der AWS IoT Core Registrierung registriert sind, ermöglicht die folgende Richtlinie, dass Geräte eine Verbindung mit einer clientId herstellen, die dem Namen einer Sache in der Registrierung entspricht. Sie bietet auch `Publish`-, `Subscribe`- und `Receive`-Berechtigungen für das Topic namens „some_specific_topic“.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
```

```
      "iot:Connection.Thing.IsAttached": "true"
     }
    }
   },
   {
    "Effect": "Allow",
    "Action": [
     "iot:Publish"
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:topic/some_specific_topic"
    ]
   },
   {
    "Effect": "Allow",
    "Action": [
     "iot:Subscribe"
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:topicfilter/some_specific_topic"
    ]
   },
   {
    "Effect": "Allow",
    "Action": [
     "iot:Receive"
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:topic/some_specific_topic"
    ]
   }
  ]
}
```

Unregistered devices

Für Geräte, die nicht in der AWS IoT Core Registrierung registriert sind, ermöglicht die folgende Richtlinie, dass Geräte entweder über ClientID1, ClientID2 oder ClientID3 eine Verbindung herstellen. Sie bietet auch `Publish`-, `Subscribe`- und `Receive`-Berechtigungen für das Topic namens „some_specific_topic“.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/some_specific_topic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/
some_specific_topic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/some_specific_topic"
            ]
        }
    ]
```

```
}
```

Richtlinien zum Veröffentlichen, Abonnieren und Empfangen von Nachrichten, Themen mit einem bestimmten Präfix to/from

Im Folgenden finden Sie Beispiele für registrierte und nicht registrierte Geräte zum Veröffentlichen, Abonnieren und Empfangen von to/from Nachrichtenthemen mit dem Präfix „topic_prefix“.

> **Note**
>
> Beachten Sie die Verwendung des Platzhalterzeichens in diesem Beispiel. * Es * ist zwar nützlich, Berechtigungen für mehrere Themennamen in einer einzigen Anweisung bereitzustellen, kann jedoch zu unbeabsichtigten Konsequenzen führen, da Geräten mehr Rechte gewährt werden als erforderlich. Wir empfehlen daher, das Platzhalterzeichen nur * nach reiflicher Überlegung zu verwenden.

Registered devices

Für Geräte, die in der AWS IoT Core Registrierung registriert sind, ermöglicht die folgende Richtlinie, dass Geräte eine Verbindung mit einer clientId herstellen, die dem Namen einer Sache in der Registrierung entspricht. Sie bietet `Publish`-, `Subscribe`- and `Receive`-Berechtigungen für Topics mit dem Präfix „topic_prefix“.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
```

```
    }
   },
   {
    "Effect": "Allow",
    "Action": [
     "iot:Publish",
     "iot:Receive"
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:topic/topic_prefix*"
    ]
   },
   {
    "Effect": "Allow",
    "Action": [
     "iot:Subscribe"
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:topicfilter/topic_prefix*"
    ]
   }
  ]
}
```

Unregistered devices

Für Geräte, die nicht in der AWS IoT Core Registrierung registriert sind, ermöglicht die folgende Richtlinie, dass Geräte entweder über ClientID1, ClientID2 oder ClientID3 eine Verbindung herstellen. Sie bietet `Publish`-, `Subscribe`- and `Receive`-Berechtigungen für Topics mit dem Präfix „topic_prefix“.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
```

```
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/topic_prefix*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/topic_prefix*"
            ]
        }
    ]
}
```

Richtlinien zum Veröffentlichen, Abonnieren und Empfangen von Nachrichten — Themen, die für jedes Gerät spezifisch sind to/from

Im Folgenden finden Sie Beispiele für registrierte und nicht registrierte Geräte zum Veröffentlichen, Abonnieren und Empfangen von Nachrichten to/from , die für das jeweilige Gerät spezifisch sind.

Registered devices

Für Geräte, die in der AWS IoT Core Registrierung registriert sind, ermöglicht die folgende Richtlinie, dass Geräte eine Verbindung mit einer clientId herstellen, die dem Namen einer Sache in der Registrierung entspricht. Sie berechtigt zum Veröffentlichen von Inhalten zum objektspezifischen Topic (`sensor/device/${iot:Connection.Thing.ThingName}`) sowie zum Abonnieren und Empfangen von Inhalten zum objektspezifischen Topic (`command/device/${iot:Connection.Thing.ThingName}`). Wenn der Name des Objekts in der Registrierung „thing1" lautet, kann das Gerät unter dem Thema" sensor/device /thing1“ veröffentlichen. Das

Gerät kann außerdem das Thema „/thing1“ command/device abonnieren und Inhalte daraus empfangen.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/sensor/device/
${iot:Connection.Thing.ThingName}"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Subscribe"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topicfilter/command/device/
${iot:Connection.Thing.ThingName}"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
```

```
    "iot:Receive"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/command/device/
${iot:Connection.Thing.ThingName}"
   ]
  }
 ]
}
```

Unregistered devices

Für Geräte, die nicht in der AWS IoT Core Registrierung registriert sind, ermöglicht die folgende Richtlinie, dass Geräte entweder über ClientID1, ClientID2 oder ClientID3 eine Verbindung herstellen. Sie berechtigt zum Veröffentlichen von Inhalten zum clientspezifischen Topic (`sensor/device/${iot:ClientId}`) sowie zum Abonnieren und Empfangen von Inhalten zum clientspezifischen Topic (`command/device/${iot:ClientId}`). Wenn das Gerät eine Verbindung mit clientId als ClientID1 herstellt, kann es unter dem Thema "/ClientID1" veröffentlichen. sensor/device Das Gerät kann das Thema auch abonnieren und Inhalte empfangen. `device/clientId1/command`

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
```

```
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/sensor/device/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/command/device/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/command/device/
${iot:Connection.Thing.ThingName}"
            ]
        }
    ]
}
```

Richtlinien zum Veröffentlichen, Abonnieren und Empfangen von Nachrichten — to/from Themen, bei denen das Thing-Attribut im Themennamen enthalten ist

Im Folgenden finden Sie ein Beispiel für registrierte Geräte zum Veröffentlichen, Abonnieren und Empfangen von Nachrichten. to/from Themen, deren Namen Ding-Attribute enthalten.

> **Note**
>
> Ding-Attribute existieren nur für Geräte, die in der AWS IoT Core Registrierung registriert sind. Es gibt kein entsprechendes Beispiel für nicht registrierte Geräte.

Registered devices

Für Geräte, die in der AWS IoT Core Registrierung registriert sind, ermöglicht die folgende Richtlinie, dass Geräte eine Verbindung mit einer clientId herstellen, die dem Namen einer Sache in der Registrierung entspricht. Sie berechtigt zum Veröffentlichen von Inhalten im Topic (`sensor/${iot:Connection.Thing.Attributes[version]}`) sowie zum Abonnieren und Empfangen von Inhalten zum Topic (`command/${iot:Connection.Thing.Attributes[location]}`), wenn der Name des Topics Objektattribute enthält. Wenn der Name des Objekts in der Registrierung den `version=v1` Wert und hat`location=Seattle`, kann das Gerät Beiträge zum Thema "" veröffentlichen und das Thema sensor/v1 "" command/Seattle abonnieren und Inhalte aus dem Thema empfangen.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/sensor/
${iot:Connection.Thing.Attributes[version]}"
   ]
  },
  {
   "Effect": "Allow",
```

```
   "Action": [
    "iot:Subscribe"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topicfilter/command/
${iot:Connection.Thing.Attributes[location]}"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Receive"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/command/
${iot:Connection.Thing.Attributes[location]}"
   ]
  }
 ]
}
```

Unregistered devices

Da Thing-Attribute nur für Geräte existieren, die in der AWS IoT Core Registrierung registriert sind, gibt es kein entsprechendes Beispiel für nicht registrierte Dinge.

Richtlinien, um die Veröffentlichung von Nachrichten zu untergeordneten Topics eines Topic-Namens zu verweigern

Im Folgenden finden Sie Beispiele für registrierte und nicht registrierte Geräte, mit denen Nachrichten zu allen Topics außer bestimmten untergeordneten Topics veröffentlicht werden können.

Registered devices

Für Geräte, die in der AWS IoT Core Registrierung registriert sind, ermöglicht die folgende Richtlinie, dass Geräte eine Verbindung mit einer clientId herstellen, die dem Namen einer Sache in der Registrierung entspricht. Sie berechtigt zur Veröffentlichung zu allen Themen, denen das Präfix „Abteilung/“ vorangestellt ist, nicht jedoch zum Unterthema "“. department/admins

```
{
```

```
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/department/*"
   ]
  },
  {
   "Effect": "Deny",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/department/admins"
   ]
  }
 ]
}
```

Unregistered devices

Für Geräte, die nicht in der AWS IoT Core Registrierung registriert sind, ermöglicht die folgende Richtlinie, dass Geräte entweder über ClientID1, ClientID2 oder ClientID3 eine Verbindung

herstellen. Sie berechtigt zur Veröffentlichung zu allen Themen, denen das Präfix „Abteilung/“ vorangestellt ist, nicht jedoch zum Unterthema "“. department/admins

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/department/*"
            ]
        },
        {
            "Effect": "Deny",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/department/admins"
            ]
        }
    ]
}
```

Richtlinien, um den Empfang von Nachrichten von untergeordneten Topics eines Topic-Namens zu verweigern

Im Folgenden finden Sie Beispiele für registrierte und nicht registrierte Geräte, die Nachrichten von allen Topics mit bestimmten Präfixen außer gewissen untergeordneten Topics abonnieren und empfangen.

Registered devices

Für Geräte, die in der AWS IoT Core Registrierung registriert sind, ermöglicht die folgende Richtlinie, dass Geräte eine Verbindung mit einer clientId herstellen, die dem Namen einer Sache in der Registrierung entspricht. Die Richtlinie ermöglicht es Geräten, jedes Topic mit dem Präfix „topic_prefix“ zu abonnieren. Durch die Verwendung `NotResource` in der Anweisung for ermöglichen wir dem Gerät`iot:Receive`, Nachrichten zu allen Themen zu empfangen, die das Gerät abonniert hat, mit Ausnahme der Themen mit dem Präfix ""“. topic_prefix/restricted Mit dieser Richtlinie können Geräte beispielsweise "" und sogar topic_prefix/topic1 "" abonnieren, sie empfangen jedoch nur Nachrichten zum Thema "topic_prefix/topic1" und keine Nachrichten zum Thema ""“topic_prefix/restricted. topic_prefix/restricted

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": "iot:Subscribe",
   "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/topic_prefix/*"
  },
```

```
  {
   "Effect": "Allow",
   "Action": "iot:Receive",
   "NotResource": "arn:aws:iot:us-east-1:123456789012:topic/topic_prefix/
restricted/*"
  }
 ]
}
```

Unregistered devices

Für Geräte, die nicht in der AWS IoT Core Registrierung registriert sind, ermöglicht die folgende Richtlinie, dass Geräte entweder über ClientID1, ClientID2 oder ClientID3 eine Verbindung herstellen. Die Richtlinie ermöglicht es Geräten, jedes Topic mit dem Präfix „topic_prefix“ zu abonnieren. Durch die Verwendung `NotResource` in der Anweisung for ermöglichen wir dem Gerät`iot:Receive`, Nachrichten zu allen Themen zu empfangen, die das Gerät abonniert hat, mit Ausnahme von Themen mit dem Präfix "“. topic_prefix/restricted Mit dieser Richtlinie können Geräte beispielsweise "" und sogar topic_prefix/topic1 "topic_prefix/restricted“ abonnieren. Sie erhalten jedoch nur Nachrichten zum Thema "topic_prefix/topic1" und keine Nachrichten zum Thema "topic_prefix/restricted“.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/
topic_prefix/*"
```

```
        },
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "NotResource": "arn:aws:iot:us-east-1:123456789012:topic/
topic_prefix/restricted/*"
        }
    ]
}
```

Richtlinien zum Abonnieren von Topics mit MQTT-Platzhalterzeichen

Die MQTT-Platzhalterzeichen + und # werden als Literalzeichenfolgen behandelt, sie werden jedoch nicht als Platzhalter behandelt, wenn sie in Richtlinien verwendet werden. AWS IoT Core In MQTT werden + und # nur als Platzhalter behandelt, wenn ein Topic-Filter abonniert wird. In anderen Zusammenhängen werden sie als wörtliche Zeichenfolge gehandhabt. Wir empfehlen, diese MQTT-Platzhalter nur nach sorgfältiger Prüfung als Teil von Richtlinien zu verwenden. AWS IoT Core

Im Folgenden finden Sie Beispiele für registrierte und nicht registrierte Dinge, die MQTT-Platzhalter in Richtlinien verwenden. AWS IoT Core Diese Platzhalter werden als Literalzeichenfolgen behandelt.

Registered devices

Für Geräte, die in der AWS IoT Core Registrierung registriert sind, ermöglicht die folgende Richtlinie, dass Geräte eine Verbindung mit einer clientId herstellen, die dem Namen einer Sache in der Registrierung entspricht. Die Richtlinie ermöglicht es Geräten, die Topics „department/+/employees“ und „location/#“ zu abonnieren. Da + und # in AWS IoT Core Richtlinien als wörtliche Zeichenketten behandelt werden, können Geräte das Thema „Abteilung/+/Mitarbeiter“ abonnieren, aber nicht das Thema "/Mitarbeiter“. department/engineering Ebenso können Geräte das Thema „Location/#“ abonnieren, aber nicht das Thema "“. location/Seattle Sobald das Gerät jedoch das Thema „Abteilung/+/Mitarbeiter“ abonniert hat, erlaubt es die Richtlinie, Nachrichten zum Thema "/employees" zu empfangen. department/engineering Ebenso empfängt das Gerät, sobald es das Thema „Location/#“ abonniert hat, auch Nachrichten zum Thema "". location/Seattle

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

```
    "Action": [
     "iot:Connect"
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
    ],
    "Condition": {
     "Bool": {
      "iot:Connection.Thing.IsAttached": "true"
     }
    }
   },
   {
    "Effect": "Allow",
    "Action": "iot:Subscribe",
    "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/department/+/
employees"
   },
   {
    "Effect": "Allow",
    "Action": "iot:Subscribe",
    "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/location/#"
   },
   {
    "Effect": "Allow",
    "Action": "iot:Receive",
    "Resource": "arn:aws:iot:us-east-1:123456789012:topic/*"
   }
  ]
}
```

Unregistered devices

Für Geräte, die nicht in der AWS IoT Core Registrierung registriert sind, ermöglicht die folgende Richtlinie, dass Geräte entweder über ClientID1, ClientID2 oder ClientID3 eine Verbindung herstellen. Mit der Richtlinie können Geräte die Topics „department/+/employees“ und „location/#“ abonnieren. Da + und # in AWS IoT Core Richtlinien als wörtliche Zeichenketten behandelt werden, können Geräte das Thema „Abteilung/+/Mitarbeiter“ abonnieren, aber nicht das Thema "/Mitarbeiter“. department/engineering In ähnlicher Weise können Geräte das Thema „Location/#“ abonnieren, aber nicht das Thema ""“. location/Seattle Sobald das Gerät jedoch das Thema „Abteilung/+/Mitarbeiter“ abonniert hat, erlaubt es die Richtlinie, Nachrichten vom Thema "/

employees" zu empfangen. department/engineering Ebenso empfängt das Gerät, sobald es das Thema „Location/#“ abonniert hat, auch Nachrichten zum Thema "". location/Seattle

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/
department/+/employees"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/location/
#"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topic/*"
        }
    ]
}
```

Richtlinien für HTTP und Clients WebSocket

Wenn Sie eine Verbindung über HTTP oder das WebSocket Protokoll herstellen, authentifizieren Sie sich mit Signature Version 4 und Amazon Cognito. Amazon-Cognito-Identitäten können authentifiziert

oder nicht authentifiziert sein. Authentifizierte Identitäten gehören zu Benutzer, die von einem unterstützten Identitätsanbieter authentifiziert werden. Nicht authentifizierte Identitäten gehören in der Regel Gastbenutzern, die sich nicht bei einem Identitätsanbieter authentifizieren. Amazon Cognito bietet eine eindeutige Kennung und AWS Anmeldeinformationen zur Unterstützung nicht authentifizierter Identitäten. Weitere Informationen finden Sie unter the section called “Autorisierung mit Amazon-Cognito-Identitäten”.

AWS IoT Core Verwendet für die folgenden Operationen AWS IoT Core Richtlinien, die über die API an Amazon Cognito Cognito-Identitäten angehängt sind. `AttachPolicy` Dadurch werden die Berechtigungen, die dem Amazon Cognito Cognito-Identitätspool mit authentifizierten Identitäten zugewiesen sind, eingeschränkt.

- `iot:Connect`
- `iot:Publish`
- `iot:Subscribe`
- `iot:Receive`
- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

Das bedeutet, dass eine Amazon Cognito Cognito-Identität die Genehmigung der IAM-Rollenrichtlinie und der AWS IoT Core Richtlinie benötigt. Sie fügen die IAM-Rollenrichtlinie über die API dem Pool und die AWS IoT Core Richtlinie der Amazon Cognito Identity hinzu. AWS IoT Core `AttachPolicy`

Authentifizierte und nicht authentifizierte Benutzer sind unterschiedliche Identitätstypen. Wenn Sie der Amazon Cognito Identity keine AWS IoT Richtlinie beifügen, schlägt ein authentifizierter Benutzer die Autorisierung fehl AWS IoT und hat keinen Zugriff auf AWS IoT Ressourcen und Aktionen.

> **Note**
>
> Bei anderen AWS IoT Core Vorgängen oder bei AWS IoT Core nicht authentifizierten Identitäten werden die mit der Amazon Cognito Cognito-Identitätspool-Rolle verbundenen Berechtigungen nicht eingeschränkt. Dies ist sowohl für authentifizierte als auch für nicht authentifizierte Identitäten die großzügigste Richtlinie, die Sie der Amazon-Cognito-Poolrolle hinzufügen sollten.

HTTP

Um nicht authentifizierten Amazon-Cognito-Identitäten zu ermöglichen, Nachrichten über HTTP zu einem für die Amazon-Cognito-Identität spezifischen Thema zu veröffentlichen, fügen Sie der Amazon-Cognito-Identitätspoolrolle die folgende IAM-Richtlinie an:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/${cognito-
identity.amazonaws.com:sub}"
            ]
        }
    ]
}
```

Um authentifizierte Benutzer zuzulassen, fügen Sie die obige Richtlinie der Amazon Cognito Identity-Pool-Rolle und der Amazon Cognito Identity mithilfe der API hinzu. AWS IoT Core AttachPolicy

> **Note**
>
> Bei der Autorisierung von Amazon Cognito Cognito-Identitäten werden beide Richtlinien AWS IoT Core berücksichtigt und die geringsten angegebenen Rechte gewährt. Eine Aktion ist nur zulässig, wenn beide Richtlinien die angeforderte Aktion zulassen. Wenn eine Richtlinie eine Aktion verbietet, wird diese Aktion nicht erlaubt.

MQTT

Um nicht authentifizierten Amazon Cognito Cognito-Identitäten die Veröffentlichung von MQTT-Nachrichten zu einem WebSocket Thema zu ermöglichen, das für die Amazon Cognito Cognito-Identität in Ihrem Konto spezifisch ist, fügen Sie der Amazon Cognito Cognito-Identitätspool-Rolle die folgende IAM-Richtlinie hinzu:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/${cognito-
identity.amazonaws.com:sub}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/${cognito-
identity.amazonaws.com:sub}"]
        }
    ]
}
```

Um authentifizierte Benutzer zuzulassen, fügen Sie die obige Richtlinie der Amazon Cognito Identity-Pool-Rolle und der Amazon Cognito Identity mithilfe der API hinzu. AWS IoT Core AttachPolicy

> **Note**
>
> Bei der Autorisierung von Amazon Cognito Cognito-Identitäten werden beide AWS IoT Core berücksichtigt und die geringsten angegebenen Rechte gewährt. Eine Aktion ist nur zulässig, wenn beide Richtlinien die angeforderte Aktion zulassen. Wenn eine Richtlinie eine Aktion verbietet, wird diese Aktion nicht erlaubt.

Beispiele zu den Verbinden- und Veröffentlichen-Richtlinien

Für Geräte, die als Dinge in der AWS IoT Core Registrierung registriert sind, gewährt die folgende Richtlinie die Erlaubnis, eine Verbindung AWS IoT Core mit einer Client-ID herzustellen, die dem Ding-Namen entspricht, und beschränkt das Gerät auf die Veröffentlichung zu einem Client-ID- oder Dingnamen-spezifischen MQTT-Thema. Damit eine Verbindung erfolgreich hergestellt werden kann,

muss der Name des Dings in der AWS IoT Core Registrierung registriert und mithilfe einer Identität oder eines Prinzipals authentifiziert werden, der dem Ding zugeordnet ist:

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/${iot:Connection.Thing.ThingName}"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
      ]
    }
  ]
}
```

Für Geräte, die nicht als Dinge in der AWS IoT Core Registrierung registriert sind, gewährt die folgende Richtlinie die Erlaubnis, eine Verbindung AWS IoT Core mit der Client-ID herzustellen, `client1` und beschränkt das Gerät auf die Veröffentlichung auf einem ID-specific MQTT-Thema auf einem Client:

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

```
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/${iot:ClientId}"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:client/client1"
      ]
    }
  ]
}
```

Beispielrichtlinien für beibehaltene Nachrichten

Die Verwendung beibehaltener Nachrichten erfordert spezielle Richtlinien. Archivierte Nachrichten sind MQTT-Nachrichten, bei denen das RETAIN-Flag gesetzt und von gespeichert wurde. AWS IoT Core Dieser Abschnitt enthält Beispielrichtlinien, die die allgemeine Verwendung von beibehaltenen Nachrichten ermöglichen.

In diesem Abschnitt:

- Richtlinie zum Verbinden und Veröffentlichen von beibehaltenen Nachrichten
- Richtlinie zum Verbinden und Veröffentlichen von beibehaltenen Will-Nachrichten
- Richtlinie zum Auflisten und Abrufen von beibehaltenen Nachrichten

Richtlinie zum Verbinden und Veröffentlichen von beibehaltenen Nachrichten

Damit ein Gerät beibehaltene Nachrichten veröffentlichen kann, muss das Gerät in der Lage sein, eine Verbindung herzustellen, (jede MQTT-Nachricht) zu veröffentlichen und beibehaltene MQTT-Nachrichten zu veröffentlichen. Die folgende Richtlinie gewährt dem Client **`device1`** diese Berechtigungen für das Topic `device/sample/configuration`. Ein weiteres Beispiel für die Berechtigung zum Herstellen einer Verbindung finden Sie unter the section called “Beispiele zu den Verbinden- und Veröffentlichen-Richtlinien”.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/device1"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish",
    "iot:RetainPublish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/device/sample/configuration"
   ]
  }
 ]
}
```

Richtlinie zum Verbinden und Veröffentlichen von beibehaltenen Will-Nachrichten

Clients können eine Nachricht konfigurieren, die veröffentlicht AWS IoT Core wird, wenn der Client die Verbindung unerwartet trennt. Im MQTT werden solche Nachrichten als Will-Nachrichten bezeichnet. Die Verbindungsberechtigung eines Clients muss um eine zusätzliche Bedingung erweitert werden, damit sie einbezogen werden können.

Das folgende Richtliniendokument gewährt allen Clients die Berechtigung, eine Verbindung herzustellen und eine Will-Nachricht mit dem Topic `will` zu veröffentlichen, die auch von AWS IoT Core beibehalten wird.

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

```
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/device1"
   ],
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "LastWill"
     ]
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish",
    "iot:RetainPublish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/will"
   ]
  }
 ]
}
```

Richtlinie zum Auflisten und Abrufen von beibehaltenen Nachrichten

Services und Anwendungen können auf beibehaltene Nachrichten zugreifen, ohne einen MQTT-Client unterstützen zu müssen, indem sie `ListRetainedMessages` und `GetRetainedMessage` aufrufen. Die Services und Anwendungen, die diese Aktionen aufrufen, müssen mithilfe einer Richtlinie wie dem folgenden Beispiel autorisiert werden.

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

```
                "iot:ListRetainedMessages"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/device1"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:GetRetainedMessage"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/foo"
            ]
        }
    ]
}
```

Beispiele für die Zertifikatrichtlinie

Für Geräte, die in der AWS IoT Core Registrierung registriert sind, gewährt die folgende Richtlinie die Erlaubnis, eine Verbindung AWS IoT Core mit einer Client-ID herzustellen, die einem Ding-Namen entspricht, und zu einem Thema zu veröffentlichen, dessen Name dem `certificateId` des Zertifikats entspricht, mit dem sich das Gerät authentifiziert hat:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/
${iot:CertificateId}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
```

```
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        }
    ]
}
```

Für Geräte, die nicht in der AWS IoT Core Registrierung registriert sind, gewährt die folgende Richtlinie die Erlaubnis, eine Verbindung AWS IoT Core mit Client-IDs`client1`,`client2`,, herzustellen `client3` und zu einem Thema zu veröffentlichen, dessen Name dem `certificateId` des Zertifikats entspricht, mit dem sich das Gerät authentifiziert hat:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/
${iot:CertificateId}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        }
    ]
}
```

Für Geräte, die in der AWS IoT Core Registrierung registriert sind, gewährt die folgende Richtlinie die Erlaubnis, eine Verbindung AWS IoT Core mit einer Client-ID herzustellen, die dem Namen des

Dings entspricht, und zu einem Thema zu veröffentlichen, dessen Name dem `CommonName` Feld des Antragstellers des Zertifikats entspricht, mit dem sich das Gerät authentifiziert hat:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/
${iot:Certificate.Subject.CommonName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        }
    ]
}
```

> **Note**
>
> In diesem Beispiel wird der Subject Common Name des Zertifikats als Topic-ID verwendet, mit der Annahme, dass der Subject Common Name für jedes registrierte Zertifikat eindeutig ist. Wenn die Zertifikate für mehrere Geräte gemeinsam genutzt werden, ist der gemeinsame Betreff-Name für alle Geräte, die dieses Zertifikat gemeinsam nutzen, identisch, sodass Veröffentlichungsrechte für dasselbe Thema von mehreren Geräten aus möglich sind (nicht empfohlen).

Für Geräte, die nicht in der AWS IoT Core Registrierung registriert sind, gewährt die folgende Richtlinie die Erlaubnis, eine Verbindung AWS IoT Core mit Client-IDs `client1client2`,, `client3` und herzustellen und zu einem Thema zu veröffentlichen, dessen Name dem `CommonName` Feld des Antragstellers des Zertifikats entspricht, mit dem sich das Gerät authentifiziert hat:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/
${iot:Certificate.Subject.CommonName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        }
    ]
}
```

> **Note**
>
> In diesem Beispiel wird der Subject Common Name des Zertifikats als Topic-ID verwendet, mit der Annahme, dass der Subject Common Name für jedes registrierte Zertifikat eindeutig ist. Wenn die Zertifikate für mehrere Geräte gemeinsam genutzt werden, ist der gemeinsame Betreff-Name für alle Geräte, die dieses Zertifikat gemeinsam nutzen, identisch, sodass Veröffentlichungsrechte für dasselbe Thema von mehreren Geräten aus möglich sind (nicht empfohlen).

Für Geräte, die in der AWS IoT Core Registrierung registriert sind, gewährt die folgende Richtlinie die Erlaubnis, eine Verbindung AWS IoT Core mit einer Client-ID herzustellen, die dem Namen des Dings entspricht, und zu einem Thema zu veröffentlichen, dessen Name mit einem Präfix versehen

ist, `admin/` wenn das `Subject.CommonName.2` Feld für das Zertifikat, das zur Authentifizierung des Geräts verwendet wird, auf gesetzt ist`Administrator`:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin/*"],
            "Condition": {
                "StringEquals": {
                    "iot:Certificate.Subject.CommonName.2": "Administrator"
            }
        }
        }
    ]
}
```

Für Geräte, die nicht in der AWS IoT Core Registrierung registriert sind, gewährt die folgende Richtlinie die Erlaubnis`client1`, eine Verbindung AWS IoT Core mit Client-IDs herzustellen `client3` und zu einem Thema zu veröffentlichen`client2`, dessen Name mit dem Präfix versehen ist, `admin/` wenn das `Subject.CommonName.2` Feld für das Zertifikat, das zur Authentifizierung des Geräts verwendet wird, auf gesetzt ist`Administrator`:

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

```
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin/*"],
            "Condition": {
                "StringEquals": {
                    "iot:Certificate.Subject.CommonName.2": "Administrator"
            }
        }
        }
    ]
}
```

Für Geräte, die in der AWS IoT Core Registrierung registriert sind, erlaubt die folgende Richtlinie einem Gerät, seinen Dingnamen zu verwenden`ThingName`, um zu einem bestimmten Thema zu veröffentlichen, das aus `admin/` folgenden Elementen besteht`Administrator`: `Subject.CommonName`

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin/
${iot:Connection.Thing.ThingName}"],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:Certificate.Subject.CommonName.List": "Administrator"
            }
        }
        }
    ]
}
```

Für Geräte, die nicht in der AWS IoT Core Registrierung registriert sind, gewährt die folgende Richtlinie die Erlaubnis `client1client2`, eine Verbindung AWS IoT Core mit Client-IDs herzustellen `client3` und zum Thema zu veröffentlichen, `admin` wenn für das Zertifikat, mit dem das Gerät authentifiziert wurde, eines der `Subject.CommonName` Felder wie folgt gesetzt ist: `Administrator`

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
```

```
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin"],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:Certificate.Subject.CommonName.List": "Administrator"
            }
        }
        }
    ]
}
```

Beispiel für Objektrichtlinien

Die folgende Richtlinie ermöglicht es einem Gerät, eine Verbindung herzustellen, wenn das zur Authentifizierung verwendete Zertifikat an das Objekt angehängt AWS IoT Core ist, für das die Richtlinie geprüft wird:

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Effect":"Allow",
            "Action":["iot:Connect"],
            "Resource":[ "*" ],
            "Condition": {
                "Bool": {
                    "iot:Connection.Thing.IsAttached": ["true"]
            }
        }
        }
    ]
}
```

Die folgende Richtlinie ermöglicht einem Gerät das Veröffentlichen von Inhalten, wenn das Zertifikat an ein Objekt mit einem bestimmten Objekttyp angehängt ist und das Objekt das Attribut `attributeName` mit dem Wert `attributeValue` hat. Weitere Informationen über Objektrichtlinienvariablen finden Sie unter Objektrichtlinienvariablen.

```
{
  "Version":"2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": "arn:aws:iot:us-east-1:123456789012:topic/device/stats",
      "Condition": {
        "StringEquals": {
          "iot:Connection.Thing.Attributes[attributeName]": "attributeValue",
          "iot:Connection.Thing.ThingTypeName": "Thing_Type_Name"
        },
        "Bool": {
          "iot:Connection.Thing.IsAttached": "true"
        }
      }
    }
  ]
}
```

Mit der folgenden Richtlinie kann ein Gerät zu einem Topic veröffentlichen, das mit einem Attribut des Objekts beginnt. Wenn das Gerätezertifikat nicht mit dem Objekt verknüpft ist, wird diese Variable nicht aufgelöst und der Fehler „Zugriff verweigert“ wird angezeigt. Weitere Informationen über Objektrichtlinienvariablen finden Sie unter Objektrichtlinienvariablen.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": "arn:aws:iot:us-east-1:123456789012:topic/
${iot:Connection.Thing.Attributes[attributeName]}/*"
    }
  ]
}
```

Beispiel für grundlegende Auftragsrichtlinie

In diesem Beispiel werden die für ein Auftragsziel erforderlichen Richtlinienanweisungen gezeigt, bei dem es sich um ein einzelnes Gerät handelt, das eine Jobanfrage empfängt und dem AWS IoT den Status der Auftragsausführung mitteilt.

*us-west-2:57EXAMPLE833*Ersetzen Sie es durch Ihr AWS-Region, einen Doppelpunkt (:) und Ihre 12-stellige AWS-Konto Zahl und *uniqueThingName* ersetzen Sie es dann durch den Namen der Ding-Ressource, in der das Gerät steht. AWS IoT

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:client/uniqueThingName"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/test/dc/pubtopic",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/events/job/*",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/events/jobExecution/
*",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/
things/uniqueThingName/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
```

```
                "arn:aws:iot:us-west-2:123456789012:topicfilter/test/dc/subtopic",
                "arn:aws:iot:us-west-2:123456789012:topicfilter/$aws/events/
jobExecution/*",
                "arn:aws:iot:us-west-2:123456789012:topicfilter/$aws/
things/uniqueThingName/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/test/dc/subtopic",
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/
things/uniqueThingName/jobs/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iotjobsdata:DescribeJobExecution",
                "iotjobsdata:GetPendingJobExecutions",
                "iotjobsdata:StartNextPendingJobExecution",
                "iotjobsdata:UpdateJobExecution"
            ],
            "Resource": [
                "arn:aws:iot:us-west-2:123456789012:topic/$aws/
things/uniqueThingName"
            ]
        }
    ]
}
```

Beispiele für Richtlinien für Direktnachrichten

Für die Verwendung von Direktnachrichten sind spezielle Richtlinien erforderlich. Direct Messaging verwendet die SendDirectMessage HTTP-API, um Nachrichten von einem Absender an einen einzelnen Empfänger zu übermitteln, der anhand seiner MQTT-Client-ID identifiziert wird. In diesem Abschnitt werden Beispiele für Richtlinien vorgestellt, die die allgemeine Verwendung von Direktnachrichten ermöglichen.

In diesem Abschnitt:

- Richtlinie zum Senden einer Direktnachricht an einen bestimmten Kunden zu bestimmten Themen
- Richtlinie zum Empfangen von Direktnachrichten
- Richtlinie zum Senden einer Direktnachricht an einen beliebigen Kunden zu bestimmten Themen
- Richtlinie zum Senden einer Direktnachricht an jeden Kunden zu einem beliebigen Thema

Richtlinie zum Senden einer Direktnachricht an einen bestimmten Kunden zu bestimmten Themen

Damit ein Absender Direktnachrichten senden kann, muss er die `iot:SendDirectMessage` Erlaubnis haben, die Ziel-Client-ID als Ressource zu verwenden. Der `iot:Topic` Bedingungsschlüssel (optional) schränkt ein, zu welchen Themen der Absender Nachrichten senden kann.

- Fügen Sie dies für SigV4-authenticated Backend-Server zu einer IAM-Richtlinie hinzu.
- Fügen Sie dies für X.509-authenticated IoT-Geräte einer AWS IoT Core Richtlinie hinzu.
- Für benutzerdefinierte, vom Authorizer authentifizierte Clients muss die Lambda-Funktion ein Policy-Dokument zurückgeben, das für die Ziel-Client-Ressource mit `iot:SendDirectMessage` dem Bedingungsschlüssel gewährt wird `iot:Topic`

Die folgende Richtlinie ermöglicht es dem Kunden`device1`, Direktnachrichten zu den Themen und an den Kunden `myDevice` zu senden. `commands/reboot` `commands/update`

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:SendDirectMessage",
            "Resource": "arn:aws:iot:us-west-2:123456789012:client/myDevice",
            "Condition": {
                "StringEquals": {
                    "iot:Topic": ["commands/reboot", "commands/update"]
                }
            }
        }
    ]
}
```

Richtlinie zum Empfangen von Direktnachrichten

Die Richtlinien des Empfängers müssen `iot:Receive` zu diesem Thema gelten. Der Empfänger benötigt keine `iot:Subscribe` Genehmigung — er AWS IoT Core übermittelt Direktnachrichten, ohne dass ein Themenabonnement erforderlich ist. Der Empfänger kann sich mithilfe eines X.509 Client-Zertifikats (AWS IoT Core Richtlinie) oder SigV4 (IAM-Richtlinie) authentifizieren. In beiden Fällen ist die `iot:Receive` Genehmigung für das empfangende Thema erforderlich.

Die folgende Richtlinie ermöglicht es dem Empfängerclient`myDevice`, Direktnachrichten zu den Themen `commands/reboot` und zu empfangen`commands/update`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": "arn:aws:iot:us-west-2:123456789012:topic/commands/reboot"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": "arn:aws:iot:us-west-2:123456789012:topic/commands/update"
        }
    ]
}
```

In der folgenden Richtlinie wird ein Platzhalter verwendet, damit der Empfänger Direktnachrichten zu jedem beliebigen Thema unter dem `commands/` Präfix empfangen kann.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": "arn:aws:iot:us-west-2:123456789012:topic/commands/*"
        }
    ]
}
```

> **Note**
>
> Der Empfänger muss eine MQTT-Verbindung zu herstellen, AWS IoT Core bevor er eine Direktnachricht empfängt. Direktnachrichten werden für Offline-Geräte nicht in die Warteschlange gestellt.

Richtlinie zum Senden einer Direktnachricht an einen beliebigen Kunden zu bestimmten Themen

Die folgende Richtlinie ermöglicht es dem Absender, Direktnachrichten an jeden Kunden zu senden, jedoch nur zu Themen, die dem `commands/*` Präfix entsprechen. Dies ist nützlich für Flottenmanagementdienste, die jedes Gerät erreichen müssen, jedoch nur für bestimmte Befehlsthemen.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:SendDirectMessage",
            "Resource": "arn:aws:iot:us-west-2:123456789012:client/*",
            "Condition": {
                "StringLike": {
                    "iot:Topic": "commands/*"
                }
            }
        }
    ]
}
```

> **Note**
>
> Der `iot:Topic` Bedingungsschlüssel unterstützt den Platzhalterabgleich mit dem `StringLike` Bedingungsoperator.

Richtlinie zum Senden einer Direktnachricht an jeden Kunden zu einem beliebigen Thema

Die folgende Richtlinie ermöglicht es dem Absender, Direktnachrichten zu einem beliebigen Thema an jeden Kunden zu senden. Dies eignet sich für administrative oder Flottenmanagement-Anwendungsfälle, in denen ein Backend-Service uneingeschränkten Zugriff benötigt.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:SendDirectMessage",
            "Resource": "arn:aws:iot:us-west-2:123456789012:client/*",
            "Condition": {
                "StringLike": {
                    "iot:Topic": "*"
                }
            }
        }
    ]
}
```

Autorisierung mit Amazon-Cognito-Identitäten

Es gibt zwei Arten von Amazon–Cognito-Identitäten: nicht authentifiziert und authentifiziert. Wenn Ihre App nicht authentifizierte Amazon-Cognito-Identitäten unterstützt, wird keine Authentifizierung durchgeführt, sodass Sie nicht wissen, wer der Benutzer:in ist.

Nicht authentifizierte Identitäten: Für nicht authentifizierte Amazon-Cognito-Identitäten gewähren Sie Berechtigungen, indem Sie eine IAM-Rolle an einen nicht authentifizierten Identitätspool anfügen. Wir empfehlen, nur jenen Ressourcen Zugriff gewähren, die Sie für unbekannte Benutzer verfügbar haben möchten.

> **Important**
>
> Für nicht authentifizierte Amazon Cognito Cognito-Benutzer, die eine Verbindung herstellen AWS IoT Core, empfehlen wir, in den IAM-Richtlinien Zugriff auf sehr begrenzte Ressourcen zu gewähren.

Authentifizierte Identitäten: Für authentifizierte Amazon-Cognito-Identitäten müssen Sie Berechtigungen an zwei Stellen angeben:

- Fügen Sie dem authentifizierten Amazon-Cognito-Identitätspool eine IAM-Richtlinie an und
- Hängen Sie der Amazon Cognito Identity (authentifizierter Benutzer) eine AWS IoT Core Richtlinie an.

Richtlinienbeispiele für nicht authentifizierte und authentifizierte Amazon Cognito Cognito-Benutzer, die eine Verbindung herstellen mit AWS IoT Core

Das folgende Beispiel zeigt Berechtigungen sowohl in der IAM-Richtlinie als auch in der IoT-Richtlinie einer Amazon-Cognito-Identität. Der authentifizierte Benutzer möchte zu einem gerätespezifischen Thema veröffentlichen (z. B. /status). device/DEVICE_ID

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/Client_ID"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/device/Device_ID/status"
            ]
        }
    ]
}
```

Das folgende Beispiel zeigt Berechtigungen sowohl in der IAM-Richtlinie einer nicht authentifizierten Amazon-Cognito-Rolle. Der nicht authentifizierte Benutzer möchte Inhalte zu nicht gerätespezifischen Topics veröffentlichen, für die keine Authentifizierung erforderlich ist.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/non_device_specific_topic"
            ]
        }
    ]
}
```

GitHub Beispiele

Die folgenden Beispiel-Webanwendungen GitHub zeigen, wie das Anhängen von Richtlinien an authentifizierte Benutzer in den Benutzeranmelde- und Authentifizierungsprozess integriert werden kann.

- Verwendung der MQTT publish/subscribe React-Webanwendung und AWS AmplifyAWS IoT Device SDK for JavaScript
- MQTT publish/subscribe React-Webanwendung mit AWS AmplifyAWS IoT Device SDK for JavaScript, der und einer Lambda-Funktion

Amplify ist eine Reihe von Tools und Diensten, mit denen Sie Web- und Mobilanwendungen erstellen können, die sich in AWS Dienste integrieren lassen. Weitere Informationen zu Amplify finden Sie in der Amplify-Framework-Dokumentation.

In beiden Beispielen werden die folgenden Schritte ausgeführt.

1. Wenn sich ein/e Benutzer:in für ein Konto anmeldet, erstellt die Anwendung einen Amazon-Cognito-Benutzerpool und eine Identität.
2. Wenn sich ein/e Benutzer:in authentifiziert, erstellt die Anwendung eine Richtlinie und fügt sie der Identität an. Dadurch erhält der/die Benutzer:in Veröffentlichungs- und Abonnementberechtigungen.
3. Der/Die Benutzer:in kann die Anwendung verwenden, um MQTT-Thopics zu veröffentlichen und zu abonnieren.

Im ersten Beispiel wird die `AttachPolicy`-API-Operation direkt in der Authentifizierungsoperation verwendet. Im folgenden Beispiel wird gezeigt, wie dieser API-Aufruf in einer React-Webanwendung implementiert wird, die Amplify und das AWS IoT Device SDK for JavaScript verwendet.

```
function attachPolicy(id, policyName) {
    var Iot = new AWS.Iot({region: AWSConfiguration.region, apiVersion:
 AWSConfiguration.apiVersion, endpoint: AWSConfiguration.endpoint});
    var params = {policyName: policyName, target: id};

    console.log("Attach IoT Policy: " + policyName + " with cognito identity id: " +
 id);
    Iot.attachPolicy(params, function(err, data) {
         if (err) {
               if (err.code !== 'ResourceAlreadyExistsException') {
                  console.log(err);
               }
          }
         else  {
            console.log("Successfully attached policy with the identity", data);
         }
     });
}
```

Dieser Code erscheint in der AuthDisplay.jsDatei.

Im zweiten Beispiel wird die `AttachPolicy`-API-Operation in einer Lambda-Funktion implementiert. Im folgenden Beispiel wird gezeigt, wie Lambda diesen API-Aufruf verwendet.

```
iot.attachPolicy(params, function(err, data) {
     if (err) {
           if (err.code !== 'ResourceAlreadyExistsException') {
              console.log(err);
              res.json({error: err, url: req.url, body: req.body});
           }
      }
     else  {
        console.log(data);
        res.json({success: 'Create and attach policy call succeed!', url: req.url,
 body: req.body});
     }
 });
```

Dieser Code erscheint in der `iot.GetPolicy`-Funktion in der Datei app.js.

> **Note**
>
> Wenn Sie die Funktion mit AWS Anmeldeinformationen aufrufen, die Sie über Amazon Cognito Identity-Pools erhalten, enthält das Kontextobjekt in Ihrer Lambda-Funktion einen Wert für. `context.cognito_identity_id` Weitere Informationen finden Sie unter den folgenden Topics.
>
> - AWS Lambda Kontextobjekt in Node.js
> - AWS Lambda Kontextobjekt in Python
> - AWS Lambda Kontextobjekt in Ruby
> - AWS Lambda Kontextobjekt in Java
> - AWS Lambda Kontextobjekt in Go
> - AWS Lambda Kontextobjekt in C#
> - AWS Lambda Kontextobjekt in PowerShell

Autorisieren direkter Anrufe an AWS Dienste unter Verwendung AWS IoT Core Anbieter von Anmeldeinformationen

Geräte können X.509 Zertifikate verwenden, um AWS IoT Core mithilfe von Protokollen für die gegenseitige TLS-Authentifizierung eine Verbindung herzustellen. Andere AWS Dienste unterstützen keine zertifikatsbasierte Authentifizierung, sie können jedoch mithilfe von AWS Anmeldeinformationen im AWS Signature Version 4-Format aufgerufen werden. Der Signature Version 4-Algorithmus erfordert normalerweise, dass der Anrufer über eine Zugriffsschlüssel-ID und einen geheimen Zugriffsschlüssel verfügt. AWS IoT Core verfügt über einen Anbieter für Anmeldeinformationen, mit dem Sie das integrierte X.509 Zertifikat als eindeutige Geräteidentität zur Authentifizierung AWS von Anfragen verwenden können. Damit ist es nicht mehr erforderlich, eine Zugriffsschlüssel-ID und einen geheimen Zugriffsschlüssel auf Ihrem Gerät zu speichern.

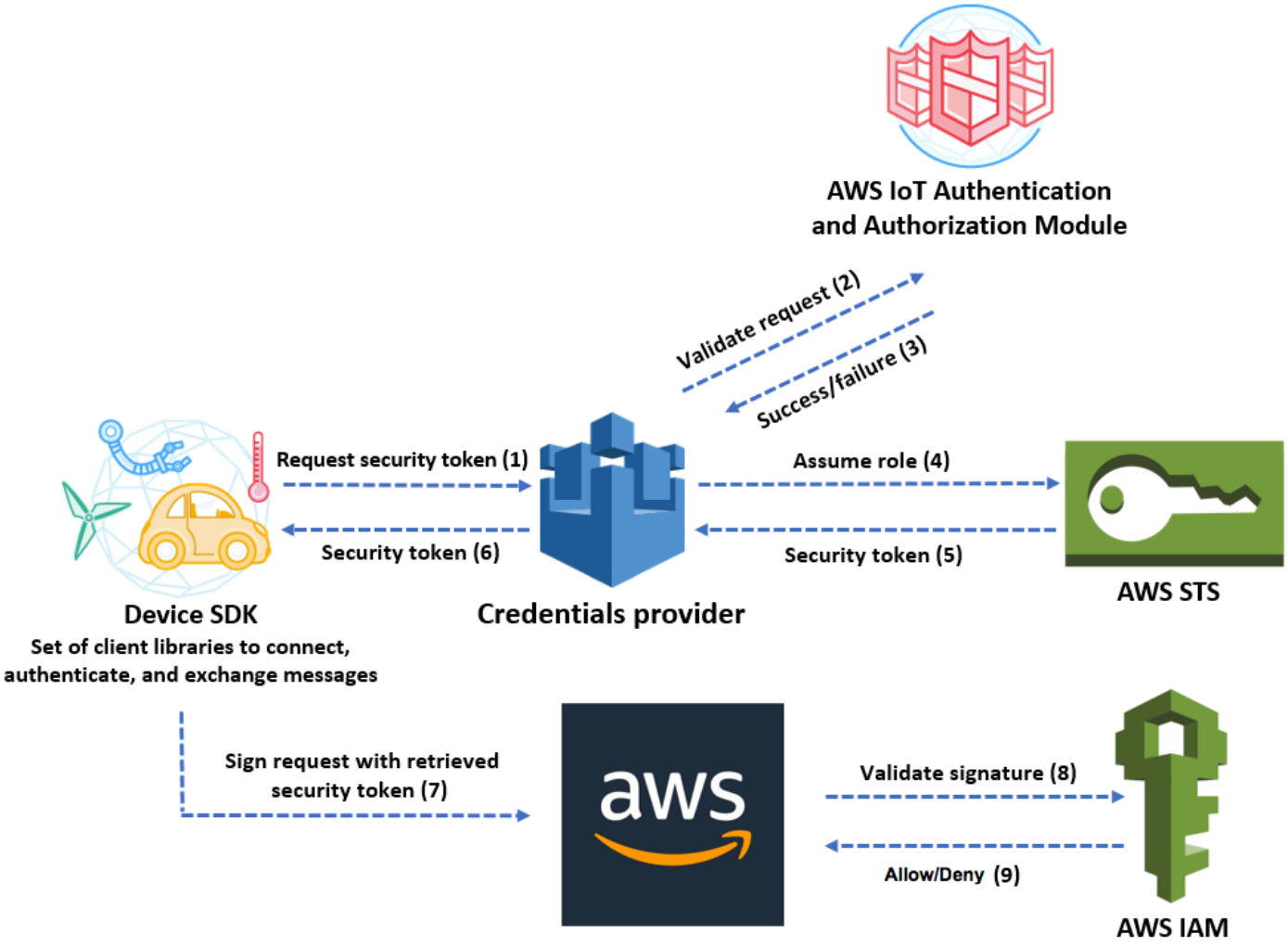
Der Anbieter für Anmeldeinformationen authentifiziert einen Anrufer mithilfe eines X.509 Zertifikats und stellt ein temporäres Sicherheitstoken mit eingeschränkten Rechten aus. Das Token kann verwendet werden, um jede Anfrage zu signieren und zu authentifizieren. AWS Für diese Art der Authentifizierung Ihrer AWS Anfragen müssen Sie eine AWS Identity and Access Management (IAM-) Rolle erstellen und konfigurieren und der Rolle entsprechende IAM-Richtlinien zuordnen, damit der Anbieter der Anmeldeinformationen die Rolle in Ihrem Namen übernehmen kann. Weitere Informationen zu AWS IoT Core und IAM finden Sie unter Identitäts- und Zugriffsmanagement für AWS IoT.

AWS IoT erfordert, dass Geräte die Server Name Indication (SNI) -Erweiterung an das Transport Layer Security (TLS) -Protokoll senden und die vollständige Endpunktadresse vor Ort angeben. `host_name` Im Feld `host_name`muss der Endpunkt angegeben sein. Dabei muss es sich um Folgendes handeln:

- Die von `aws iot describe-endpoint --endpoint-type iot:CredentialProvider` zurückgegebene `endpointAddress`.

Verbindungsversuche von Geräten ohne den richtigen Wert für `host_name` werden fehlschlagen.

Das folgende Diagramm veranschaulicht den Workflow des Anmeldeinformationsanbieters.



1. Das AWS IoT Core Gerät sendet eine HTTPS-Anfrage an den Anbieter für Anmeldeinformationen nach einem Sicherheitstoken. Die Anfrage enthält das X.509 Gerätezertifikat für die Authentifizierung.
2. Der Anbieter für Anmeldeinformationen leitet die Anfrage an das AWS IoT Core Authentifizierungs- und Autorisierungsmodul weiter, um das Zertifikat zu validieren und zu überprüfen, ob das Gerät berechtigt ist, das Sicherheitstoken anzufordern.
3. Wenn das Zertifikat gültig ist und berechtigt ist, ein Sicherheitstoken anzufordern, meldet das AWS IoT Core Authentifizierungs- und Autorisierungsmodul Erfolg. Andernfalls sendet es eine Ausnahme an das Gerät.
4. Nach der erfolgreichen Validierung des Zertifikats ruft der Anmeldeinformationsanbieter AWS - Security-Token-Service (AWS STS) auf, um die IAM-Rolle anzunehmen, die Sie für ihn erstellt haben.

5. AWS STS gibt ein temporäres Sicherheitstoken mit eingeschränkten Rechten an den Anmeldeinformationsanbieter zurück.
6. Der Anmeldeinformationsanbieter gibt das Sicherheits-Token an das Gerät zurück.
7. Das Gerät verwendet das Sicherheitstoken, um eine AWS Anfrage mit AWS Signature Version 4 zu signieren.
8. Der angeforderte Service ruft IAM auf, um die Signatur zu validieren und die Anforderung anhand der Zugriffsrichtlinien zu autorisieren, die der IAM-Rolle zugeordnet sind, die Sie für den Anmeldeinformationsanbieter erstellt haben.
9. Wenn IAM die Signatur erfolgreich validiert und die Anforderung autorisiert, ist die Anforderung erfolgreich. Andernfalls sendet IAM eine Ausnahme.

Im folgenden Abschnitt wird beschrieben, wie Sie ein Zertifikat verwenden, um ein Sicherheits-Token abzurufen. Es ist in der Annahme geschrieben, dass Sie bereits ein Gerät registriert haben und für es ein eigenes Zertifikat erstellt und aktiviert haben.

So verwenden Sie ein Zertifikat zum Abrufen eines Sicherheits-Tokens

1. Konfigurieren Sie die IAM-Rolle, die der Anmeldeinformationsanbieter im Namen Ihres Geräts annimmt. Fügen Sie der Rolle die folgende Vertrauensrichtlinie hinzu.

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Principal": {"Service": "credentials.iot.amazonaws.com"},
        "Action": "sts:AssumeRole"
    }
}
```

   Fügen Sie der Rolle für jeden AWS Dienst, den Sie aufrufen möchten, eine Zugriffsrichtlinie hinzu. Der Anmeldeinformationsanbieter unterstützt die folgenden Richtlinienvariablen:

   - `credentials-iot:ThingName`
   - `credentials-iot:ThingTypeName`
   - `credentials-iot:AwsCertificateId`

Wenn das Gerät den Objektnamen in seiner Anforderung an einen AWS -Service bereitstellt, fügt der Anmeldeinformationsanbieter dem Sicherheits-Token `credentials-iot:ThingName` und `credentials-iot:ThingTypeName` als Kontextvariablen hinzu. Der Anmeldeinformationsanbieter stellt `credentials-iot:AwsCertificateId` als Kontextvariable bereit, auch wenn das Gerät den Dingnamen nicht in der Anfrage angibt. Sie übergeben den Dingnamen als Wert des `x-amzn-iot-thingname` HTTP-Anfrage-Headers.

Diese drei Variablen funktionieren nur für IAM-Richtlinien, nicht für AWS IoT Core -Richtlinien.

2. Stellen Sie sicher, dass der Benutzer, der den nächsten Schritt ausführt (Anlegen eines Rollenalias), die Berechtigung hat, die neu erstellte Rolle an AWS IoT Core zu übergeben. Die folgende Richtlinie gewährt einem AWS Benutzer `iam:GetRole` sowohl `iam:PassRole` Berechtigungen als auch Berechtigungen. Die `iam:GetRole`-Berechtigung ermöglicht es dem Benutzer, Informationen über die Rolle zu abzurufen, die Sie gerade erstellt haben. Die `iam:PassRole` Berechtigung ermöglicht es dem Benutzer, die Rolle an einen anderen AWS Dienst zu übergeben.

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iam:GetRole",
            "iam:PassRole"
        ],
        "Resource": "arn:aws:iam::123456789012:role/your role name"
    }
}
```

3. Erstellen Sie einen AWS IoT Core Rollenalias. Das Gerät, das direkte Anrufe an AWS Dienste tätigen soll, muss wissen, zu welcher Rolle der ARN für die Verbindung verwendet AWS IoT Core werden soll. Hard-coding Der Rollen-ARN ist keine gute Lösung, da Sie das Gerät aktualisieren müssen, wenn sich der Rollen-ARN ändert. Eine bessere Lösung ist es, mit der `CreateRoleAlias`-API einen Rollenalias zu erstellen, der auf den Rollen-ARN verweist. Wenn sich der Rollen-ARN ändert, können Sie einfach den Rollenalias aktualisieren. Auf dem Gerät ist keine Änderung erforderlich. Diese API verwendet die folgenden Parameter:

`roleAlias`

Erforderlich Eine beliebige Zeichenfolge, die den Rollenalias identifiziert. Sie dient als Primärschlüssel im Rollenalias-Datenmodell. Sie hat 1-128 Zeichen und darf nur alphanumerische Zeichen und die Symbole =, @ und - enthalten. Großbuchstaben und Kleinbuchstaben sind zulässig. Bei Rollen-Aliasnamen wird zwischen Groß- und Kleinschreibung unterschieden.

`roleArn`

Erforderlich Der ARN der Rolle, auf den sich der Rollenalias bezieht.

`credentialDurationSeconds`

Optional. Die Gültigkeitsdauer (in Sekunden) der Anmeldeinformationen. Die Mindestwert beträgt 900 Sekunden (15 Minuten). Der Höchstwert beträgt 43.200 Sekunden (12 Stunden). Der Standardwert ist 3.600 Sekunden (1 Stunde).

> **Important**
>
> Der AWS IoT Core Credential Provider kann Anmeldeinformationen mit einer maximalen Lebensdauer von 43.200 Sekunden (12 Stunden) ausstellen. Wenn die Anmeldeinformationen bis zu 12 Stunden gültig sind, kann die Anzahl der Anrufe beim Anmeldeinformationsanbieter reduziert werden, da die Anmeldeinformationen länger zwischengespeichert werden.
> Der `credentialDurationSeconds`-Wert muss kleiner oder gleich der maximalen Sitzungsdauer der IAM-Rolle sein, auf die der Rollenalias verweist. Weitere Informationen finden Sie unter Ändern der maximalen Sitzungsdauer (AWS API) einer Rolle im AWS Identity and Access Management-Benutzerhandbuch.

Weitere Informationen über diese API finden Sie unter CreateRoleAlias.

4. Anfügen einer Richtlinie an das Gerätezertifikat. Die an das Gerätezertifikat angefügte Richtlinie muss dem Gerät die Berechtigung erteilen, die Rolle zu übernehmen. Dies erreichen Sie, indem Sie dem Rollenalias die Berechtigung für die Aktion `iot:AssumeRoleWithCertificate` erteilen, wie im folgenden Beispiel gezeigt.

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:AssumeRoleWithCertificate",
            "Resource": "arn:aws:iot:us-east-1:123456789012:rolealias/your role
 alias"
        }
    ]
}
```

5. Stellen Sie eine HTTPS-Anfrage an den Anmeldeinformationsanbieter, um ein Sicherheits-Token zu erhalten. Geben Sie die folgenden Informationen an:

   - Zertifikat: Da es sich um eine HTTP-Anfrage zur gegenseitigen TLS-Authentifizierung handelt, müssen Sie Ihrem Client bei der Anfrage das Zertifikat und den privaten Schlüssel zur Verfügung stellen. Verwenden Sie dasselbe Zertifikat und denselben privaten Schlüssel, mit dem Sie Ihr Zertifikat registriert haben AWS IoT Core.

     Um sicherzustellen, dass Ihr Gerät mit AWS IoT Core (und nicht mit einem Dienst, der sich als solches ausgibt) kommuniziert, finden Sie weitere Informationen unter Serverauthentifizierung. Folgen Sie den Links, um die entsprechenden CA-Zertifikate herunterzuladen, und kopieren Sie sie dann auf Ihr Gerät.
   - RoleAlias: Der Name des Rollenalias, den Sie für den Anbieter für Anmeldeinformationen erstellt haben. Bei Rollenaliasnamen wird zwischen Groß- und Kleinschreibung unterschieden und sie müssen mit dem Rollenalias übereinstimmen, der in erstellt wurde AWS IoT Core.
   - ThingName: Der Name des Dings, den Sie bei der Registrierung Ihres Dings erstellt AWS IoT Core haben. Dieser wird als Wert des `x-amzn-iot-thingname` HTTP-Headers übergeben. Dieser Wert ist nur erforderlich, wenn Sie Ding-Attribute als Richtlinienvariablen in AWS IoT Core oder IAM-Richtlinien verwenden.

     > **Note**
     >
     > Der Wert ThingName, den Sie angeben, `x-amzn-iot-thingname` muss mit dem Namen der Dingressource AWS IoT übereinstimmen, die einem Zertifikat zugewiesen ist. Wenn sie nicht übereinstimmen, wird ein 403-Fehler zurückgegeben.

Führen Sie den folgenden Befehl in der aus AWS CLI , um den Endpunkt des Anmeldeinformationsanbieters für Ihren AWS-Konto zu erhalten. Weitere Informationen über diese API finden Sie unter DescribeEndpoint. Informationen zu FIPS-enabled Endpunkten finden Sie unterAWS IoT Core - Endpunkte des Anbieters von Anmeldeinformationen.

```
aws iot describe-endpoint --endpoint-type iot:CredentialProvider
```

Das folgende JSON-Objekt ist die Beispielausgabe des describe-endpoint-Befehls. Es enthält die `endpointAddress` , die Sie verwenden, um ein Sicherheits-Token anzufordern.

```
{
    "endpointAddress": "your_aws_account_specific_prefix.credentials.iot.your
 region.amazonaws.com"
}
```

Verwenden Sie den Endpunkt, um eine HTTPS-Anfrage an den Anmeldeinformationsanbieter zu stellen, ein Sicherheits-Token zurückzugeben. Der folgende Beispielbefehl verwendet `curl`, aber Sie können jeden beliebigen HTTP-Client verwenden.

> **Note**
>
> Beim Namen roleAlias wird zwischen Groß- und Kleinschreibung unterschieden und er muss mit dem Rollenalias übereinstimmen, der in erstellt wurde. AWS IoT

```
curl --cert your certificate --key your private key -H "x-amzn-iot-thingname: your
 thing name" --cacert AmazonRootCA1.pem https://your endpoint /role-aliases/your
 role alias/credentials
```

Dieser Befehl gibt ein Sicherheits-Token-Objekt zurück, das einen `accessKeyId`, einen `secretAccessKey`, ein `sessionToken`, und einen Ablaufzeitpunkt enthält. Das folgende JSON-Objekt ist die Beispielausgabe des `curl`-Befehls.

```
   {"credentials":{"accessKeyId":"access key","secretAccessKey":"secret access
 key","sessionToken":"session token","expiration":"2018-01-18T09:18:06Z"}}
```

Anschließend können Sie die `sessionToken` Werte, und verwenden `accessKeyIdsecretAccessKey`, um Anfragen an Dienste zu signieren. AWS Eine umfassende Demonstration finden Sie im Blogbeitrag So vermeiden Sie Hard-Coded AWS Anmeldedaten auf Geräten mithilfe des AWS IoT Credential Providers im AWS Security Blog.

Kontenübergreifender Zugriff mit IAM

AWS IoT Core ermöglicht es Ihnen, einem Prinzipal die Möglichkeit zu geben, ein Thema zu veröffentlichen oder zu abonnieren, das AWS-Konto nicht dem Prinzipal gehört. Den kontoübergreifenden Zugriff konfigurieren Sie durch Erstellung einer IAM-Richtlinie und einer IAM-Rolle und das anschließende Anfügen der Richtlinie an die Rolle.

Erstellen Sie zunächst eine kundenseitig verwaltete IAM-Richtlinie wie in Erstellen von IAM-Richtlinien beschrieben, genauso wie für andere Benutzer und Zertifikate in Ihrem AWS-Konto.

Für Geräte, die in der AWS IoT Core Registrierung registriert sind, gewährt die folgende Richtlinie Geräten die Erlaubnis, eine Verbindung herzustellen, indem sie eine Client-ID AWS IoT Core verwenden, die dem Ding-Namen des Geräts entspricht, und dass sie dort veröffentlichen, `my/topic/`*`thing-name`* wo *`thing-name`* sich der Ding-Name des Geräts befindet:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
```

```
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/my/topic/
${iot:Connection.Thing.ThingName}"
            ]
        }
    ]
}
```

Für Geräte, die nicht in der AWS IoT Core Registrierung registriert sind, erteilt die folgende Richtlinie einem Gerät die Erlaubnis, den in der AWS IoT Core Registrierung Ihres Kontos (123456789012) `client1` registrierten Dingnamen zu verwenden, um eine Verbindung zu einem ID-specific Client-Thema herzustellen AWS IoT Core und dort zu veröffentlichen, dessen Name mit einem Präfix versehen ist: `my/topic/`

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/my/topic/${iot:ClientId}"
            ]
        }
    ]
}
```

Führen Sie als Nächstes die unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen IAM-Benutzer beschriebenen Schritte aus. Geben Sie die ID des AWS-Konto ein, für das gemeinsamer Zugriff gewährt werden soll. Fügen Sie dann im letzten Schritt die gerade erstellte Richtlinie an die Rolle an. Wenn Sie später die AWS -Konto-ID ändern müssen, auf die Sie den Zugriff gewährt haben, können Sie das folgende Vertrauensrichtlinienformat nutzen:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::567890123456:user/MyUser"
        },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Datenschutz in AWS IoT Core

Für den Datenschutz in AWS IoT Core gilt das Modell der geteilten Verantwortung das von AWS . Wie in diesem Modell beschrieben, AWS ist verantwortlich für den Schutz der globalen Infrastruktur, auf der AWS Cloud alle Sie sind dafür verantwortlich, die Kontrolle über Ihre in dieser Infrastruktur gehosteten Inhalte zu behalten. Sie sind auch für die Sicherheitskonfiguration und die Verwaltungsaufgaben für die von Ihnen verwendeten AWS-Services verantwortlich. Weitere Informationen zum Datenschutz finden Sie unter Häufig gestellte Fragen zum Datenschutz . Weitere Informationen zum Datenschutz in Europa finden Sie im Zentrum für die Datenschutz-Grundverordnung (DSGVO).

Aus Datenschutzgründen empfehlen wir, dass Sie AWS-Konto Anmeldeinformationen schützen und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einrichten. So erhält jeder Benutzer nur die Berechtigungen, die zum Durchführen seiner Aufgaben erforderlich sind. Außerdem empfehlen wir, die Daten mit folgenden Methoden schützen:

- Verwenden Sie für jedes Konto die Multi-Faktor-Authentifizierung (MFA).
- Wird verwendet SSL/TLS , um mit AWS Ressourcen zu kommunizieren. Wir benötigen TLS 1.2 und empfehlen TLS 1.3.

- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit ein AWS CloudTrail. Informationen zur Verwendung von CloudTrail Pfaden zur Erfassung von AWS Aktivitäten finden Sie unter Arbeiten mit CloudTrail Pfaden im AWS CloudTrail Benutzerhandbuch.
- Verwenden Sie AWS Verschlüsselungslösungen zusammen mit allen darin enthaltenen Standardsicherheitskontrollen AWS-Services.
- Verwenden Sie erweiterte verwaltete Sicherheitsservices wie Amazon Macie, die dabei helfen, in Amazon S3 gespeicherte persönliche Daten zu erkennen und zu schützen.
- Wenn Sie für den Zugriff AWS über eine Befehlszeilenschnittstelle oder eine API FIPS 140-3-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter Federal Information Processing Standard (FIPS) 140-3.

Wir empfehlen dringend, in Freitextfeldern, z. B. im Feld Name, keine vertraulichen oder sensiblen Informationen wie die E-Mail-Adressen Ihrer Kunden einzugeben. Dies gilt auch, wenn Sie mit der Konsole, der API AWS IoT oder den SDKs arbeiten oder diese anderweitig AWS-Services verwenden. AWS CLI AWS Alle Daten, die Sie in Tags oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet werden. Wenn Sie eine URL für einen externen Server bereitstellen, empfehlen wir dringend, keine Anmeldeinformationen zur Validierung Ihrer Anforderung an den betreffenden Server in die URL einzuschließen.

Weitere Informationen zum Datenschutz enthält der Blog-Beitrag AWS Shared Responsibility Model and GDPR im AWS -Sicherheitsblog.

AWS IoT Geräte sammeln Daten, manipulieren diese Daten und senden diese Daten dann an einen anderen Webdienst. Sie können einige Daten für einen kurzen Zeitraum auf Ihrem Gerät speichern. Sie sind dafür verantwortlich, den Datenschutz für diese Daten im Ruhezustand zu gewährleisten. Wenn Ihr Gerät Daten an sendet AWS IoT, geschieht dies über eine TLS-Verbindung, wie später in diesem Abschnitt beschrieben wird. AWS IoT Geräte können Daten an jeden AWS Dienst senden. Weitere Informationen zur Datensicherheit der einzelnen Dienste finden Sie in der Dokumentation zu diesem Dienst. AWS IoT kann so konfiguriert werden, dass CloudWatch Protokolle in Logs geschrieben und AWS IoT API-Aufrufe protokolliert werden AWS CloudTrail. Weitere Informationen zur Datensicherheit für diese Dienste finden Sie unter Authentifizierung und Zugriffskontrolle für Amazon CloudWatch und Verschlüsseln von CloudTrail Protokolldateien mit AWS KMS verwalteten Schlüsseln.

Datenverschlüsselung in AWS IoT

Standardmäßig sind alle AWS IoT Daten während der Übertragung und Speicherung verschlüsselt. Übertragene Daten werden mit TLS verschlüsselt, und Daten im Ruhezustand werden mit AWS eigenen Schlüsseln verschlüsselt. AWS IoT unterstützt vom Kunden verwaltete AWS KMS keys (KMS-Schlüssel) über den AWS Key Management Service (AWS KMS). Device Advisor und AWS IoT Wireless verwenden jedoch nur an AWS-eigener Schlüssel , um Kundendaten zu verschlüsseln.

Transportsicherheit in AWS IoT Core

TLS (Transport Layer Security) ist ein kryptografisches Protokoll, das für die sichere Kommunikation über ein Computernetzwerk ausgelegt ist. Beim AWS IoT Core Device Gateway müssen Kunden die gesamte Kommunikation während der Übertragung verschlüsseln, indem sie TLS für Verbindungen zwischen Geräten und dem Gateway verwenden. TLS wird verwendet, um die Vertraulichkeit der Anwendungsprotokolle (MQTT, HTTP und WebSocket) zu gewährleisten, die von unterstützt werden. AWS IoT Core TLS unterstützt verschiedene Programmiersprachen und Betriebssysteme. Die AWS darin enthaltenen Daten werden durch den jeweiligen AWS Dienst verschlüsselt. Weitere Informationen zur Datenverschlüsselung bei anderen AWS Diensten finden Sie in der Sicherheitsdokumentation für diesen Dienst.

Inhalt

- TLS-Protokolle
- Sicherheitsrichtlinien
- Wichtige Hinweise zur Verkehrssicherheit in AWS IoT Core
- Transportsicherheit für LoRa WAN-Wireless-Geräte

TLS-Protokolle

AWS IoT Core unterstützt die folgenden Versionen des TLS-Protokolls:

- TLS 1.3
- TLS 1.2

Mit AWS IoT Core können Sie die TLS-Einstellungen (für TLS 1.2 und TLS 1.3) in Domänenkonfigurationen konfigurieren. Weitere Informationen finden Sie unter ???.

Sicherheitsrichtlinien

Eine Sicherheitsrichtlinie ist eine Kombination aus TLS-Protokollen und ihren Verschlüsselungen, die bestimmen, welche Protokolle und Verschlüsselungen bei TLS-Verhandlungen zwischen einem Client und einem Server unterstützt werden. Sie können Ihre Geräte so konfigurieren, dass sie je nach Bedarf vordefinierte Sicherheitsrichtlinien verwenden. Beachten Sie, dass benutzerdefinierte Sicherheitsrichtlinien AWS IoT Core nicht unterstützt werden.

Sie können eine der vordefinierten Sicherheitsrichtlinien für Ihre Geräte auswählen, wenn Sie eine Verbindung herstellen AWS IoT Core. Die Namen der neuesten vordefinierten Sicherheitsrichtlinien AWS IoT Core enthalten Versionsinformationen, die auf dem Jahr und Monat basieren, in dem sie veröffentlicht wurden. Die vordefinierte Standardsicherheitsrichtlinie für AWS GovCloud (US) Nicht-Regionen ist `IoTSecurityPolicy_TLS13_1_2_2022_10` und `IoTSecurityPolicy_TLS13_1_2_2022_01` für AWS GovCloud (US) Regionen. Um eine Sicherheitsrichtlinie anzugeben, können Sie die AWS IoT Konsole oder die verwenden AWS CLI. Weitere Informationen finden Sie unter ???.

In der folgenden Tabelle werden die aktuellen vordefinierten Sicherheitsrichtlinien beschrieben, die von AWS IoT Core unterstützt werden. Die `IotSecurityPolicy_` wurde aus Richtliniennamen in der Überschriftenzeile entfernt, sodass sie passen.

Sicherheitsrichtlinie	TLS13_1_2022_10	TLS13_1_2022_10	TLS12_1_2022_10	TLS12_1_0_2016_01*		TLS12_1_0_2015_01*		TLS13_1_2022_01*
TCP-Port	443/8443/8883	443/8443/8883	443/8443/8883	443	8443/8883	443	8443/8883	443/8443/8883
TLS-Protokolle								
TLS 1.2		✓	✓	✓	✓	✓	✓	✓
TLS 1.3	✓	✓						✓
TLS-Verschlüsselungsverfahren								
TLS_AES_128_GCM_A256	✓	✓						✓

Sicherheitsrichtlinie	TLS13_1_2022_10	TLS13_1_2022_10	TLS12_1_2022_10	TLS12_1_0_2016_01*		TLS12_1_0_2015_01*		TLS13_1_2022_01*
TLS_AES_256_GCM_A384	✓	✓						✓
TLS_CHACHA20_POLY1305_SHA256	✓	✓						
ECDHE-RSA-AES128-GCM-SHA256		✓	✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES128-SHA256		✓	✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES128-SHA		✓	✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES256-GCM-SHA384		✓	✓	✓	✓	✓	✓	✓

Sicherheitsrichtlinie	TLS13_1_2022_10	TLS13_1_2022_10	TLS12_1_2022_10	TLS12_1_0_2016_01*		TLS12_1_0_2015_01*		TLS13_1_2022_01*
ECDHE-RSA-AES256-SHA384		✓	✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES256-SHA		✓	✓	✓	✓	✓	✓	✓
AES128-GCM-SHA256		✓	✓	✓	✓	✓	✓	
AES128-SHA256		✓	✓	✓		✓	✓	
AES128-SHA		✓	✓	✓	✓	✓	✓	✓
AES256-GCM-SHA384		✓	✓	✓	✓	✓	✓	
AES256-SHA256		✓	✓	✓	✓	✓	✓	
AES256-SHA		✓	✓	✓	✓	✓	✓	✓

Sicherheitsrichtlinie	TLS13_1_2022_10	TLS13_1_2022_10	TLS12_1_2022_10	TLS12_1_0_2016_01*		TLS12_1_0_2015_01*		TLS13_1_2022_01*
DHE-RSA-AES256-SHA						✓	✓	
ECDHE-ECDSA-AES128-GCM-SHA256		✓	✓	✓	✓	✓	✓	✓
ECDHE-ECDSA-AES128-SHA256		✓	✓	✓	✓	✓	✓	✓
ECDHE-ECDSA-AES128-SHA		✓	✓	✓	✓	✓	✓	✓
ECDHE-ECDSA-AES256-GCM-SHA384		✓	✓	✓	✓	✓	✓	✓

Sicherheitsrichtlinie	TLS13_1_2022_10	TLS13_1_2022_10	TLS12_1_2022_10	TLS12_1_0_2016_01*		TLS12_1_0_2015_01*		TLS13_1_2_2022_01*
ECDHE-ECDSA-AES256-SHA384		✓	✓	✓	✓	✓	✓	✓
ECDHE-ECDSA-AES256-SHA		✓	✓	✓	✓	✓	✓	✓

> **Note**
>
> `TLS12_1_0_2016_01`ist nur in den folgenden Sprachen verfügbar AWS-Regionen: ap-east-1, ap-northeast-2, ap-south-1, ap-southeast-2, ca-central-1, cn-north-1, cn-northwest-1, eu-north-1, eu-west-2, eu-west-3, me-south-1, sa-east-1, us-east-2, us-west-1.
> `TLS12_1_0_2015_01`ist nur in den folgenden Ländern verfügbar AWS-Regionen: ap-northeast-1, ap-southeast-1, eu-central-1, eu-west-1, us-east-1, us-west-2.
> `TLS13_1_2_2022_01`ist nur in AWS GovCloud (US) Regionen verfügbar.

Wichtige Hinweise zur Verkehrssicherheit in AWS IoT Core

Bei Geräten, die AWS IoT Core über MQTT eine Verbindung herstellen, verschlüsselt TLS die Verbindung zwischen den Geräten und dem Broker und AWS IoT Core verwendet die TLS-Client-Authentifizierung, um Geräte zu identifizieren. Weitere Informationen finden Sie unter Client-Authentifizierung. Bei Geräten, die AWS IoT Core über HTTP eine Verbindung herstellen, verschlüsselt TLS die Verbindung zwischen den Geräten und dem Broker, und die Authentifizierung wird an AWS Signature Version 4 delegiert. Weitere Informationen finden Sie unter Signieren von Anforderungen mit Signature Version 4 in der Allgemeinen Referenz zu AWS .

Wenn Sie Geräte mit verbinden AWS IoT Core, ist das Senden der SNI-Erweiterung (Server Name Indication) nicht erforderlich, wird aber dringend empfohlen. Um Funktionen wie die Registrierung mehrerer Konten, benutzerdefinierte Domänen, VPC-Endpunkte und konfigurierte TLS-Richtlinien zu verwenden, müssen Sie die SNI-Erweiterung verwenden und die vollständige Endpunktadresse in das Feld eingeben. `host_name` Im Feld `host_name` muss der Endpunkt angegeben sein, den Sie aufrufen. Dieser Endpunkt muss einer der folgenden sein:

- den von `aws iot describe-endpoint --endpoint-type iot:Data-ATS` zurückgegebenen `endpointAddress`
- den von `aws iot describe-domain-configuration --domain-configuration-name "domain_configuration_name"` zurückgegebenen `domainName`

Verbindungsversuche von Geräten mit dem falschen oder `host_name` ungültigen Wert schlagen fehl. AWS IoT Core protokolliert Fehler CloudWatch für den Authentifizierungstyp Benutzerdefinierte Authentifizierung.

AWS IoT Core unterstützt die SessionTicket TLS-Erweiterung nicht.

Transportsicherheit für LoRa WAN-Wireless-Geräte

LoRaWAN-Geräte folgen den in LoRaWAN™ SECURITY: A White Paper Prepared for the LoRa Alliance™ von Gemalto, Actility und Semtech beschriebenen Sicherheitspraktiken.

Weitere Informationen zur Transportsicherheit mit LoRa WAN-Geräten finden Sie unter LoRaWAN-Daten und Transportsicherheit.

Datenverschlüsselung in AWS IoT

Datenschutz bezieht sich auf den Schutz von Daten während der Übertragung (beim Hin- und Hersenden AWS IoT Core) und im Ruhezustand (während sie auf Geräten oder von anderen AWS Diensten gespeichert werden). Alle an AWS IoT Core gesendeten Daten werden über eine TLS-Verbindung unter Verwendung von MQTT, HTTPS und WebSocket Protokollen gesendet, sodass sie während der Übertragung standardmäßig sicher sind. AWS IoT Core sammelt Daten von Geräten und sendet sie dann zur weiteren Verarbeitung an andere AWS Dienste. Weitere Informationen zur Datenverschlüsselung für andere AWS -Services finden Sie in der Sicherheitsdokumentation des Services. Weitere Informationen finden Sie unter Datenverschlüsselung im Ruhezustand.

FreeRTOS bietet eine PKCS#11-Bibliothek, die Schlüsselspeicher abstrahiert, auf kryptografische Objekte zugreift und Sitzungen verwaltet. Es liegt in Ihrer Verantwortung, diese Bibliothek zu

verwenden, um Daten im Ruhezustand auf Ihren Geräten zu verschlüsseln. Weitere Informationen finden Sie unter FreeRTOS#11-Bibliothek (Public Key Cryptography Standard).

Datenverschlüsselung im Ruhezustand in AWS IoT Core

Standardmäßig werden alle AWS IoT Core Daten im Ruhezustand mit AWS eigenen Schlüsseln verschlüsselt. AWS IoT Core unterstützt auch symmetrische, vom Kunden verwaltete Schlüssel von AWS Key Management Service (AWS KMS). Mit vom Kunden verwalteten Schlüsseln können Sie die AWS KMS Schlüssel in Ihrem AWS Konto erstellen, besitzen und verwalten. AWS IoT Core verwendet Ihre KMS-Schlüssel, um Ihre Daten im Ruhezustand zu verschlüsseln. Sie haben die volle Kontrolle über diese KMS-Schlüssel, einschließlich der Erstellung und Verwaltung ihrer wichtigsten Richtlinien. Sie können auch IAM-Richtlinien für die Rollen konfigurieren, die Zugriff darauf haben, AWS KMS um die Berechtigungen für diese Schlüssel zu kontrollieren.

AWS eigene Schlüssel

AWS Eigene Schlüssel sind eine Sammlung von KMS-Schlüsseln, die ein AWS Dienst besitzt und verwaltet, sodass sie in mehreren AWS Konten verwendet werden können. AWS Dienste können AWS eigene Schlüssel verwenden, um Ihre Daten zu schützen. AWS IoT Core Verschlüsselt Daten im Ruhezustand standardmäßig mit AWS eigenen Schlüsseln. Diese Schlüssel werden vom Dienst verwaltet. Sie können AWS eigene Schlüssel nicht anzeigen, verwalten oder verwenden. Sie müssen jedoch keine Maßnahmen ergreifen, um diese Schlüssel zu schützen.

Weitere Informationen zu AWS eigenen Schlüsseln finden Sie unter AWS Eigene Schlüssel im AWS Key Management Service Entwicklerhandbuch.

Kundenseitig verwaltete Schlüssel

Von Kunden verwaltete Schlüssel sind KMS-Schlüssel in Ihrem AWS Konto, die Sie erstellen, besitzen und verwalten. Sie haben die volle Kontrolle über diese AWS KMS Schlüssel, einschließlich der Erstellung und Verwaltung ihrer wichtigsten Richtlinien. Sie können auch IAM-Richtlinien für die Rollen konfigurieren, die Zugriff darauf haben, AWS KMS um die Berechtigungen für diese Schlüssel zu kontrollieren. Sie können so konfigurieren AWS IoT Core , dass Ihre Daten mithilfe von kundenverwalteten KMS-Schlüsseln verschlüsselt werden.

Weitere Informationen über kundenverwaltete Schlüssel finden Sie unter Kundenverwaltete Schlüssel im AWS Key Management Service Developer Guide.

Gehen Sie wie folgt vor, um sich für vom Kunden verwaltete Schlüssel anzumelden: AWS IoT Core

Themen

- Schritt 1: Erstellen eines kundenverwalteten Schlüssels
- Schritt 2: Erstellen Sie eine IAM-Rolle, die Sie gewähren möchten AWS IoT Core Berechtigungen zur Verwendung des KMS-Schlüssels
- Schritt 3: Melden Sie sich für vom Kunden verwaltete Schlüssel an AWS IoT Core
- Schritt 4: Zusätzliche Berechtigungen sind erforderlich für AWS IoT Core Betrieb der Kontrollebene
- Schritt 5: Schlüssel verwalten
- Schritt 6: Überwachung des Zustands wichtiger Schlüssel

Schritt 1: Erstellen eines kundenverwalteten Schlüssels

Sie können einen symmetrischen, vom Kunden verwalteten Schlüssel mithilfe der AWS KMS Konsole oder der AWS KMS CLI-Befehle erstellen. Das `keySpec` muss sein `SYMMETRIC_DEFAULT` und das `keyUsage` muss sein. `ENCRYPT_DECRYPT`

> **Note**
>
> AWS IoT Core unterstützt nur AWS KMS Schlüssel mit `SYMMETRIC_DEFAULT` Schlüsselspezifikation und `ENCRYPT_DECRYPT` Schlüsselverwendung für vom Kunden verwaltete Schlüssel.

Im Folgenden finden Sie einen AWS CLI Beispielbefehl zum Erstellen eines KMS-Schlüssels, der zusammen mit AWS IoT Core vom Kunden verwalteten Schlüsseln verwendet werden kann.

```
aws kms create-key --key-spec SYMMETRIC_DEFAULT --key-usage ENCRYPT_DECRYPT --region
 us-west-2
```

Im Folgenden finden Sie ein Beispiel für die Ausgabe des Befehls.

```
{
    "KeyMetadata": {
        "AWSAccountId": "111122223333",
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
```

```
        "CreationDate": "2024-09-19T11:45:23.982000-07:00",
        "Enabled": true,
        "Description": "",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "Enabled",
        "Origin": "AWS_KMS",
        "KeyManager": "CUSTOMER",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "KeySpec": "SYMMETRIC_DEFAULT",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ],
        "MultiRegion": false
    }
}
```

Weitere Informationen finden Sie unter Creating a symmetric customer managed key im AWS Key Management Service Developer Guide.

Schlüsselrichtlinie

Bei der Erstellung eines vom Kunden verwalteten Schlüssels können Sie eine Schlüsselrichtlinie angeben. Schlüsselrichtlinien steuern den Zugriff auf den vom Kunden verwalteten Schlüssel. Jeder vom Kunden verwaltete Schlüssel muss über genau eine Schlüsselrichtlinie verfügen, die aussagt, wer den Schlüssel wie verwenden kann. Weitere Informationen finden Sie unter Wichtige Richtlinien im AWS Key Management Service Entwicklerhandbuch.

AWS IoT Core verwendet eine IAM-Rolle in Ihrem Konto, um auf Ihren vom Kunden verwalteten Schlüssel zuzugreifen. Wenn Sie eine benutzerdefinierte Schlüsselrichtlinie verwenden, stellen Sie sicher, dass die für diesen Schlüssel erstellte IAM-Rolle über die folgenden Berechtigungen verfügt:

- `kms:DescribeKey`
- `kms:Decrypt`
- `kms:Encrypt`
- `kms:GenerateDataKeyWithoutPlaintext`
- `kms:ReEncryptTo`
- `kms:ReEncryptFrom`

Schritt 2: Erstellen Sie eine IAM-Rolle, die Sie gewähren möchten AWS IoT Core Berechtigungen zur Verwendung des KMS-Schlüssels

AWS IoT Core Um den von Ihnen erstellten KMS-Schlüssel zum Verschlüsseln Ihrer Daten im Ruhezustand verwenden zu können, müssen Sie außerdem eine IAM-Rolle in Ihrem Konto erstellen, die davon ausgehen AWS IoT Core kann, auf den KMS-Schlüssel zuzugreifen.

Die Rolle muss über die folgende Vertrauensrichtlinie verfügen, damit sie diese AWS IoT Core Rolle übernehmen kann.

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Principal": {
            "Service": "iot.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
            "StringEquals": {
                "aws:SourceAccount": "111122223333"
            },
            "ArnLike": {
                "aws:SourceArn": "arn:aws:iot:us-west-2:111122223333:*"
            }
        }
    }
}
```

Stellen Sie sicher, dass die mit der IAM-Rolle verknüpften IAM-Richtlinien über die folgenden Berechtigungen für den KMS-Schlüssel verfügen:

- `kms:DescribeKey`
- `kms:Decrypt`
- `kms:Encrypt`
- `kms:GenerateDataKeyWithoutPlaintext`
- `kms:ReEncryptTo`
- `kms:ReEncryptFrom`

Im Folgenden finden Sie ein Beispiel für eine IAM-Richtlinie mit den erforderlichen Berechtigungen für vom Kunden verwaltete Schlüssel.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowIoTToAccessKMSResource",
            "Effect": "Allow",
            "Action": [
                "kms:DescribeKey",
                "kms:Decrypt",
                "kms:Encrypt",
                "kms:ReEncryptTo",
                "kms:ReEncryptFrom",
                "kms:GenerateDataKeyWithoutPlaintext"
            ],
            "Resource": [
                "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
            ],
            "Condition": {
                "StringEquals": {
                    "kms:EncryptionContext:aws-crypto-ec:vendor": "iot.amazonaws.com"
                }
            }
        }
    ]
}
```

Weitere Informationen finden Sie im Benutzerhandbuch unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen IAM-Benutzer.AWS Identity and Access Management

Schritt 3: Melden Sie sich für vom Kunden verwaltete Schlüssel an AWS IoT Core

Nachdem Sie alle vorherigen Schritte abgeschlossen haben, führen Sie den `update-encryption-configuration` CLI-Befehl aus, um sich mit vom Kunden verwalteten Schlüsseln anzumelden AWS IoT Core. Wenn Sie sich für vom Kunden verwaltete Schlüssel entscheiden, werden alle AWS IoT Core Ressourcen in Ihrem AWS Konto mit dem angegebenen AWS KMS Schlüssel verschlüsselt.

1. Führen Sie den `update-encryption-configuration` CLI-Befehl aus AWS CLI, um sich für die AWS IoT Core Verwendung von vom Kunden verwalteten Schlüsseln zu entscheiden.

```
aws iot update-encryption-configuration --encryption-type "CUSTOMER_MANAGED_KMS_KEY"
 \
--kms-access-role-arn "arn:aws:iam::111122223333:role/myrole" \
--kms-key-arn "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab" --region us-west-2
```

2. Führen Sie den `describe-encryption-configuration` CLI-Befehl aus, um zu überprüfen AWS CLI, ob vom Kunden verwaltete Schlüssel AWS IoT Core verwendet werden:

   ```
   aws iot describe-encryption-configuration --region us-west-2
   ```

   Wenn Sie vom Kunden verwaltete Schlüssel aktiviert haben AWS IoT Core, kann die Ausgabe wie folgt aussehen:

   ```
   {
       "encryptionType": "CUSTOMER_MANAGED_KMS_KEY",
       "kmsKeyArn": "arn:aws:kms:us-
   west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
       "kmsAccessRoleArn": "arn:aws:iam::111122223333:role/myrole",
       "configurationDetails": {
           "configurationStatus": "HEALTHY"
       },
       "lastModifiedDate": "2024-09-26T22:01:02.365000-07:00"
   }
   ```

   Das `lastModifiedDate` Feld gibt das Datum an, an dem die Verschlüsselungskonfiguration zuletzt aktualisiert wurde.

   Wenn Sie vom Kunden verwaltete Schlüssel nicht aktiviert haben, kann die Ausgabe wie folgt aussehen:

   ```
   {
       "encryptionType": "AWS_OWNED_KMS_KEY",
       "lastModifiedDate": "2024-09-26T22:01:02.365000-07:00"
   }
   ```

Schritt 4: Zusätzliche Berechtigungen sind erforderlich für AWS IoT Core Betrieb der Kontrollebene

Nachdem Sie sich für vom Kunden verwaltete Schlüssel entschieden haben, werden alle AWS IoT Core Ressourcen, die zu Ihrem AWS Konto gehören, mit dem bereitgestellten KMS-Schlüssel verschlüsselt. Für alle Operationen auf der Steuerungsebene benötigt der Anrufer nun zusätzlich zu den für den jeweiligen Vorgang auf der AWS IoT Core Ressource erforderlichen Berechtigungen auch Berechtigungen für den KMS-Schlüssel. `kms:Decrypt` Wenn der Aufrufer nicht über die `kms:Decrypt` entsprechende Berechtigung verfügt und einen API-Aufruf tätigt, für den Daten ver- oder entschlüsselt werden müssen (z. B.`GetPolicy`), erhält er eine. `UnauthorizedException`

Wenn Sie beispielsweise anrufen`GetPolicy`, benötigen Sie `iot:GetPolicy` sowohl beides als auch `kms:Decrypt` Berechtigungen für Ihren vom Kunden verwalteten KMS-Schlüssel, damit der API-Aufruf erfolgreich ist.

> **Note**
>
> Wenn Sie IAM-Benutzer oder -Rollen aktualisieren, um AWS KMS Berechtigungen für den für Ihre Verschlüsselungskonfiguration verwendeten Schlüssel zu gewähren, stellen Sie sicher, dass die KMS-Schlüsselrichtlinie auch den jeweiligen IAM-Benutzern oder -Rollen die erforderlichen Berechtigungen gewährt.

AWS KMS Berechtigungen für **`UpdateEncryptionConfiguration`**

Für den `UpdateEncryptionConfiguration` API-Aufruf sind die folgenden AWS KMS Berechtigungen für den KMS-Schlüssel erforderlich, damit Sie sich für vom Kunden verwaltete Schlüssel entscheiden oder die Schlüsselkonfiguration ändern können:

- `kms:DescribeKey`
- `kms:Decrypt`
- `kms:Encrypt`
- `kms:GenerateDataKeyWithoutPlaintext`
- `kms:ReEncryptTo`
- `kms:ReEncryptFrom`

AWS KMS Berechtigungen für alle anderen APIs der Steuerungsebene

Die meisten APIs der Kontrollebene erfordern `kms:Decrypt` Berechtigungen, wenn vom Kunden verwaltete Schlüssel aktiviert sind. Für bestimmte APIs sind diese zusätzlichen Berechtigungen jedoch nicht erforderlich:

APIs, für die keine AWS KMS Berechtigungen erforderlich sind

Die `Delete*` APIs `List*` und fallen nicht in diesen Bereich. Kunden können jederzeit eine beliebige API `List*` oder eine API auf `Delete*` Steuerungsebene aufrufen, und diese API-Aufrufe würden auch dann erfolgreich sein, wenn der Aufrufer nicht über `kms:Decrypt` die entsprechende Genehmigung verfügt. Diese API-Aufrufe sind auch dann erfolgreich, wenn Ihr vom Kunden verwalteter Schlüssel fehlerhaft ist, da `List* Delete*` APIs keine Entschlüsselung durchführen.

- List*-APIs — Alle Auflistungsvorgänge (z. B.,`ListThings`) `ListPolicies` `ListCertificates`
- Delete* APIs — Alle Löschvorgänge (z. B.,`DeleteThing`) `DeletePolicy` `DeleteCertificate`

Schritt 5: Schlüssel verwalten

AWS IoT Core überprüft regelmäßig die vom Kunden verwaltete Schlüsselkonfiguration, um sicherzustellen, dass die Verschlüsselungs- und Entschlüsselungsvorgänge nicht beeinträchtigt werden. Diese Integritätsprüfungen werden einmal pro Minute durchgeführt und überprüfen AWS IoT Core, ob auf den Schlüssel und die zugehörige IAM-Rolle zugegriffen und diese für Verschlüsselungs- und AWS KMS Entschlüsselungsvorgänge verwendet werden kann.

HEALTHY

AWS IoT Core kann über die angegebene IAM-Rolle erfolgreich auf den AWS KMS Schlüssel zugreifen und Operationen ausführen. encryption/decryption Alle Komponenten funktionieren einwandfrei.

UNHEALTHY

AWS IoT Core kann nicht auf den AWS KMS Schlüssel zugreifen oder ihn verwenden. Dadurch werden neue Verschlüsselungsvorgänge verhindert und die Funktionalität des Dienstes kann beeinträchtigt werden. Das `errorCode` Feld gibt an, ob das Problem mit dem Schlüssel oder der IAM-Rolle zusammenhängt.

Kundenaktionen, die sich auf den Zustand wichtiger Schlüssel auswirken können

Verschiedene Kundenaktionen können dazu führen, dass sich der Status des Schlüsselzustands von `HEALTHY` zu ändert`UNHEALTHY`:

Key-related Aktionen

- Löschen eines AWS KMS Schlüssels — Wenn Sie das Löschen eines Schlüssels planen, befindet er sich in einem `Pending deletion` Status und kann nicht verwendet werden
- AWS KMS Schlüssel deaktivieren — Wenn Sie einen KMS-Schlüssel deaktivieren, kann er nicht mehr für Verschlüsselungs-/Entschlüsselungsvorgänge verwendet werden
- Schlüssel für das Löschen planen — Der Schlüssel wird unbrauchbar, wenn der Löschvorgang abgeschlossen ist
- Änderung der Schlüsselrichtlinie — Entfernen der erforderlichen Zugriffsberechtigungen AWS IoT Core
- Ändern der Berechtigungen zur Schlüsselnutzung — Einschränkung der erforderlichen Aktionen AWS KMS

Rollenbezogene IAM-Aktionen

- Löschen der IAM-Rolle — es AWS IoT Core kann nicht angenommen werden, dass die Rolle auf den Schlüssel zugreift
- Rollenberechtigungen ändern — Erforderliche AWS KMS Berechtigungen aus der Rollenrichtlinie entfernen
- Änderung der Vertrauensrichtlinie — Verhindern, dass der AWS IoT Core Dienst die Rolle übernimmt
- Hinzufügen restriktiver Bedingungen — Bedingungen, die die Nutzung der Rolle AWS IoT Core verhindern

Account-level Aktionen

- Cross-account Änderungen am Schlüsselzugriff — Ändern der Berechtigungen für Schlüssel in verschiedenen Konten
- Service Control Policies (SCPs) — Organization-level Richtlinien, die den Zugriff einschränken AWS KMS
- Account-level IAM-Richtlinien — Richtlinien, die den Schlüsselzugriff außer Kraft setzen oder mit diesem in Konflikt geraten

> **Important**
>
> Alle Änderungen an AWS KMS Schlüsseln, IAM-Rollen oder Richtlinien, die von verwendet werden, AWS IoT Core sollten zuerst in Entwicklungsumgebungen getestet werden. Überwachen Sie den Zustand der Schlüssel genau, nachdem Sie Änderungen vorgenommen haben, um sicherzustellen, dass die AWS IoT Core Funktionalität nicht beeinträchtigt wird.

Die Verschlüsselungskonfiguration wird aktualisiert

Aktualisieren Sie Ihre Verschlüsselungskonfiguration AWS IoT Core , um von einem vom Kunden verwalteten Schlüssel zu einem anderen oder zwischen AWS eigenen Schlüsseln und kundenverwalteten Schlüsseln zu wechseln.

So ändern Sie die Konfiguration auf einen anderen vom Kunden verwalteten Schlüssel:

1. Erstellen Sie einen neuen vom Kunden verwalteten Schlüssel, indem Sie die Schritte unter befolgenthe section called “Schritt 1: Erstellen eines kundenverwalteten Schlüssels”.
2. Aktualisieren Sie Ihre IAM-Rollenrichtlinie so, dass sie während des Aktualisierungszeitraums Berechtigungen sowohl für die alten als auch für die neuen Schlüssel enthält.
3. Aktualisieren Sie Ihre Verschlüsselungskonfiguration, um den neuen Schlüssel zu verwenden:

```
aws iot update-encryption-configuration --encryption-type "CUSTOMER_MANAGED_KMS_KEY"
 \
--kms-access-role-arn "arn:aws:iam::111122223333:role/myrole" \
--kms-key-arn "arn:aws:kms:us-west-2:111122223333:key/new-key-id"
```

So ändern Sie die Konfiguration von vom Kunden verwalteten Schlüsseln zurück zu AWS eigenen Schlüsseln:

```
aws iot update-encryption-configuration --encryption-type "AWS_OWNED_KMS_KEY"
```

> **Note**
>
> Stellen Sie bei der Aktualisierung der Verschlüsselungskonfiguration für neue, vom Kunden verwaltete Schlüssel sicher, dass sowohl der alte als auch der neue Schlüssel weiterhin zugänglich sind, damit der Vorgang erfolgreich ist.

Häufige Ausfallszenarien und Auswirkungen

In der folgenden Tabelle werden häufige Fehlerszenarien beschrieben, wenn Schlüssel gelöscht oder deaktiviert werden:

Szenario	Sofortige Wirkung	Long-term Konsequenzen
Schlüssel deaktiviert	Alle neuen encryption/decryption Operationen schlagen sofort fehl	Betriebsunterbrechung, bis der Schlüssel wieder aktiviert oder ersetzt wird
Der Schlüssel soll gelöscht werden	Der Status des Schlüssels wird in „Ausstehend gelöscht“ geändert und alle encryption/decryption Vorgänge schlagen fehl	Automatischer Dienstfehler, wenn der Löschvorgang abgeschlossen ist
Der Schlüssel wurde dauerhaft gelöscht	Sofortiger und dauerhafter Ausfall aller Operationen	Dauerhafter Datenverlust und Unfähigkeit, verschlüsselte Daten wiederherzustellen
Wichtige Richtlinie wurde falsch geändert	AWS IoT Core verliert die Zugriffsberechtigungen für den Schlüssel	Der Dienst schlägt fehl, bis die Richtlinie korrigiert ist
Die IAM-Rolle wurde gelöscht	AWS IoT Core kann die Rolle für den Zugriffsschlüssel nicht übernehmen	Vollständiger Ausfall des Verschlüsselungsdienstes
Die IAM-Rolle wurde falsch geändert	AWS IoT Core kann keine Rolle übernehmen oder die Rolle für den Zugriff auf den Schlüssel verwenden	Der Dienst schlägt fehl, bis die IAM-Rolle behoben ist

Prävention und bewährte Verfahren

Um das versehentliche Löschen oder Deaktivieren von Schlüsseln zu verhindern und das Risiko von Serviceausfällen zu minimieren, gehen Sie wie folgt vor:

Implementieren Sie wichtige Lebenszyklusrichtlinien

Legen Sie klare Verfahren für die Schaffung, Rotation und Außerbetriebnahme von Schlüsseln fest. Dokumentieren Sie, welche Schlüssel von welchen AWS IoT Core Ressourcen verwendet werden, und führen Sie ein Inventar der aktiven Schlüssel.

Verwenden Sie IAM-Richtlinien, um das Löschen von Schlüsseln einzuschränken

Erstellen Sie IAM-Richtlinien, die verhindern, dass unbefugte Benutzer wichtige Verschlüsselungsschlüssel löschen oder deaktivieren. Verwenden Sie Bedingungen, um eine zusätzliche Genehmigung für das Löschen von Schlüsseln zu verlangen.

CloudTrail Protokollierung aktivieren

Überwachen Sie alle AWS KMS wichtigen Vorgänge, CloudTrail um unbefugte oder versehentliche Schlüsselverwaltungsaktivitäten zu erkennen. Richten Sie Warnmeldungen für das Löschen oder Deaktivieren von Schlüsseln oder für Richtlinienänderungen ein.

Testen Sie die Verfahren zum Austausch von Schlüsseln

Testen Sie Ihre wichtigsten Austauschverfahren regelmäßig in Umgebungen außerhalb der Produktionsumgebung, um sicherzustellen, dass Sie nach Ausfällen im Zusammenhang mit Schlüsseln schnell wieder einsatzbereit sind.

Pflegen Sie wichtige Backups

Sie können zwar kein AWS KMS Schlüsselmaterial exportieren, führen aber detaillierte Aufzeichnungen über wichtige ARNs, Richtlinien und zugehörige AWS IoT Core Konfigurationen, um bei Bedarf einen schnellen Austausch von Schlüsseln zu ermöglichen.

Überwachen Sie den Zustand wichtiger Schlüssel

Überwachen Sie die `CMK.Health` Metrik kontinuierlich und richten Sie automatische Benachrichtigungen für wichtige Änderungen des Gesundheitsstatus ein. Implementieren Sie automatisierte Antworten, um wichtige Probleme schnell zu lösen.

> **Important**
>
> Testen Sie wichtige Aktualisierungsverfahren immer in Entwicklungsumgebungen, bevor Sie sie in der Produktion implementieren. Halten Sie einen dokumentierten Rollback-Plan bereit und stellen Sie sicher, dass wichtige Ersatzverfahren in Notfällen schnell ausgeführt werden können.

Schritt 6: Überwachung des Zustands wichtiger Schlüssel

Im Rahmen der regelmäßigen AWS IoT Core Prüfläufe werden CloudWatch Messwerte und Protokolle ausgegeben, um einen Überblick über den Status Ihrer vom Kunden verwalteten Schlüsselkonfiguration zu erhalten

AWS IoT Core sendet die `CMK.Health` Metrik CloudWatch mindestens einmal pro Minute aus. Die Metrik liefert Informationen über den Zustand der vom Kunden verwalteten Schlüssel, die AWS IoT Core zum Verschlüsseln und Entschlüsseln Ihrer Daten verwendet werden.

Die `CMK.Health` Metrik kann die folgenden Werte haben:

- Der Wert lautet`1`: AWS IoT Core ist in der Lage, die Verschlüsselungsschlüssel erfolgreich zum Verschlüsseln und Entschlüsseln Ihrer Daten zu verwenden.
- Der Wert AWS IoT Core ist`0`: kann die Verschlüsselungsschlüssel nicht zum Verschlüsseln und Entschlüsseln Ihrer Daten verwenden.

AWS IoT Core gibt auch AWS IoT V2-Protokolle aus, wenn sich der Integritätsstatus der Verschlüsselungsschlüssel ändert. Diese Protokolle enthalten zusätzliche Informationen zur Aktualisierung des Gesundheitsstatus. Um diese Protokolle anzeigen zu können, müssen Sie AWS IoT V2-Protokolle aktivieren. Die `HEALTHY` Protokolle werden auf `INFO` Ebene ausgegeben, und die `UNHEALTHY` Protokolle werden auf `ERROR` Ebene 2 ausgegeben. Weitere Informationen zu den Protokollebenen finden Sie unter Protokollebenen.

Bei den folgenden Beispielen handelt es CloudWatch sich um Protokolleinträge, die von ausgegeben wurden AWS IoT Core , um auf die Aktualisierung des Integritätsstatus der vom Kunden verwalteten Schlüssel hinzuweisen.

Um wichtige Änderungen des Gesundheitsstatus effektiv zu überwachen und darauf zu reagieren:

1. Richten Sie CloudWatch Alarme für die `CMK.Health` Metrik ein:

```
aws cloudwatch put-metric-alarm --region us-west-2 \
  --alarm-name "IoTCore-CMK-Health-Alert" \
  --alarm-description "Alert when IoT Core CMK health is unhealthy" \
  --metric-name "CMK.Health" \
  --namespace "AWS/IoT" \
  --statistic "Minimum" \
  --period 300 \
  --evaluation-periods 1 \
```

```
  --threshold 1 \
  --comparison-operator "LessThanThreshold" \
  --alarm-actions "arn:aws:sns:us-west-2:111122223333:iot-alerts"
```

2. Aktivieren Sie die AWS IoT V2-Protokollierung, um detaillierte Ereignisse mit Änderungen des Gesundheitsstatus mit Fehlercodes und Meldungen zu erfassen.
3. Überprüfen Sie den Konfigurationsstatus zur Fehlerbehebung:

```
aws iot describe-encryption-configuration --region us-west-2
```

4. Untersuchen Sie den Status UNGESUND, indem Sie das `errorCode` Feld untersuchen:
   - `KMS_KEY_VALIDATION_ERROR`— Problem mit dem AWS KMS Schlüssel (deaktiviert, gelöscht oder Richtlinienprobleme)
   - `ROLE_VALIDATION_ERROR`— Problem mit der IAM-Rolle (gelöscht, Richtlinienprobleme oder Vertrauensprobleme)

Von UNGESUND zu GESUND

Wenn der Status der Verschlüsselungsschlüssel von `UNHEALTHY` bis aktualisiert wird`HEALTHY`, AWS IoT Core wird eine AWS IoT V2-Protokollnachricht im folgenden Format ausgegeben.

```
{
    "timestamp": "2017-08-10 15:37:23.476",
    "logLevel": "INFO",
    "traceId": "8421693b-f4f0-4e4a-9235-0cff8bab897d",
    "accountId": "111122223333",
    "status": "SUCCESS",
    "cmkStatus": "HEALTHY",
    "kmsKeyArn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "kmsAccessRoleArn": "arn:aws:iam::111122223333:role/myrole",
    "eventType": "CmkHealthCheck"
}
```

Von GESUND zu UNGESUND

Wenn der Status der Verschlüsselungsschlüssel von `HEALTHY` bis aktualisiert wird`UNHEALTHY`, AWS IoT Core wird eine AWS IoT V2-Protokollnachricht im folgenden Format ausgegeben.

```
{
```

```
    "timestamp": "2017-08-10 15:37:23.476",
    "logLevel": "ERROR",
    "traceId": "8421693b-f4f0-4e4a-9235-0cff8bab897d",
    "accountId": "111122223333",
    "status": "FAILURE",
    "cmkStatus": "UNHEALTHY",
    "errorCode": "KMS_KEY_VALIDATION_ERROR / ROLE_VALIDATION_ERROR",
    "errorMessage": "Error message on why there was a failure",
    "kmsKeyArn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "kmsAccessRoleArn": "arn:aws:iam::111122223333:role/myrole",
    "eventType": "CmkHealthCheck"
}
```

Warning

Wenn der Zustand des Schlüssels erreicht ist`UNHEALTHY`, schlagen die AWS IoT Core Operationen sofort fehl. Überprüfen Sie in diesem Fall Ihre Schlüsselkonfigurationen, IAM-Rollenberechtigungen und Richtlinien. Überwachen Sie die `CMK.Health` Metrik auf Statusänderungen. Wenn der Betrieb nach der Überprüfung Ihrer Konfigurationen weiterhin fehlschlägt, wenden Sie sich an Ihren Account Manager oder das AWS Support Center, um weitere Unterstützung zu erhalten.

Überwachung des Zustands der Schlüsselrotation in der Filiale

AWS IoT Core verwendet eine hierarchische Schlüsselstruktur, um Ihre Daten zu verschlüsseln. An der Spitze dieser Hierarchie steht Ihr AWS KMS Schlüssel (der vom Kunden verwaltete Schlüssel, den Sie konfigurieren). Darunter werden Filialschlüssel AWS IoT Core verwaltet — interne Schlüssel, die von Ihrem AWS KMS Schlüssel abgeleitet werden. AWS IoT Core verwendet dann diese Zweigschlüssel, um eindeutige Wrapping-Schlüssel zu generieren, die die eigentliche Ver- und Entschlüsselung Ihrer Daten durchführen. Dieser mehrschichtige Ansatz schränkt den Umfang jedes einzelnen Schlüssels ein und folgt den bewährten AWS Verschlüsselungsmethoden. AWS IoT Core verwaltet Filialschlüssel als internes Implementierungsdetail, getrennt von Ihrer AWS KMS Schlüsselrotation, die direkt AWS KMS verwaltet wird. Weitere Informationen finden Sie unter Rotation Ihres aktiven Filialschlüssels.

AWS IoT Core rotiert aus Sicherheitsgründen jährlich die Filialschlüssel. Durch Rotation wird eine neue Version des aktiven Branch-Schlüssels erstellt, wobei frühere Versionen zur Entschlüsselung vorhandener Daten beibehalten werden. Für diese Rotation ist die `kms:Encrypt` Berechtigung für

die IAM-Rolle erforderlich, die mit Ihrer vom Kunden verwalteten Schlüsselkonfiguration verknüpft ist. AWS IoT Core gibt die `CMK.BranchKeyRotationHealth` Metrik CloudWatch einmal pro Minute aus, um anzugeben, ob Ihre AWS KMS Zugriffsrolle über die für die Rotation erforderlichen Berechtigungen verfügt.

Die `CMK.BranchKeyRotationHealth` Metrik kann die folgenden Werte haben:

- Der Wert lautet`1`: Die AWS KMS Zugriffsrolle verfügt über die erforderlichen `kms:Encrypt` Berechtigungen und die Schlüsselrotation kann normal ablaufen.
- Der Wert ist`0`: Der AWS KMS Zugriffsrolle fehlt die `kms:Encrypt` Berechtigung und die Schlüsselrotation kann nicht durchgeführt werden. Ihre bestehenden Entschlüsselungsvorgänge werden weiterhin normal funktionieren, aber AWS IoT Core Sie können Ihre Zweigschlüssel erst wechseln, wenn die Berechtigung wiederhergestellt ist. Fügen Sie `kms:Encrypt` es zu Ihrer AWS KMS Zugriffsrolle hinzu, um die Rotation wieder zu aktivieren. Die vollständige Liste der erforderlichen Berechtigungen finden Sie unter Wichtige Richtlinien.

> **Note**
>
> Ein Wert von `0` bedeutet nicht, dass Ihr aktueller AWS IoT Core Betrieb unterbrochen wurde. Ihre Daten bleiben verschlüsselt und zugänglich. Sie sollten Ihre AWS KMS Zugriffsrollenberechtigungen jedoch umgehend aktualisieren, um sicherzustellen, dass die Filialschlüssel termingerecht ausgetauscht werden können.

Um benachrichtigt zu werden, wenn die Schlüsselrotation der Filialen nicht fortgesetzt werden kann, richten Sie einen CloudWatch Alarm für die `CMK.BranchKeyRotationHealth` Metrik ein:

```
aws cloudwatch put-metric-alarm --region us-west-2 \
  --alarm-name "IoTCore-CMK-BranchKeyRotationHealth-Alert" \
  --alarm-description "Alert when IoT Core branch key rotation health is degraded" \
  --metric-name "CMK.BranchKeyRotationHealth" \
  --namespace "AWS/IoT" \
  --statistic "Minimum" \
  --period 300 \
  --evaluation-periods 1 \
  --threshold 1 \
  --comparison-operator "LessThanThreshold" \
  --alarm-actions "arn:aws:sns:us-west-2:111122223333:iot-alerts"
```

AWS CloudTrail Veranstaltungen

Sie können auch die Verwendung AWS IoT Core des KMS-Schlüssels für Verschlüsselungs- und Entschlüsselungsvorgänge überwachen. AWS IoT Core führt`DescribeKey`, `DecryptReEncrypt`, und `GenerateDataKeyWithoutPlaintext` -Operationen mit Ihrem KMS-Schlüssel durch, um Daten zu verschlüsseln bzw. zu entschlüsseln, die zu Ihrem AWS Konto gehören und im Ruhezustand gespeichert sind.

Es gibt CloudTrail Ereignisse für`DescribeKey`, `DecryptReEncrypt`, und. `GenerateDataKeyWithoutPlaintext` Diese Ereignisse überwachen AWS KMS Vorgänge, die von aufgerufen werden AWS IoT Core , um auf Daten zuzugreifen, die mit Ihrem vom Kunden verwalteten Schlüssel verschlüsselt wurden.

Beispiel für das **`Entschlüsseln`**


```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
        "accountId": "111122223333",
        "accessKeyId": "**********************",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAIGDTESTANDEXAMPLE:Sampleuser01",
                "arn": "arn:aws:sts::111122223333:assumed-role/Admin/Sampleuser01",
                "accountId": "111122223333",
                "userName": "*****"
            },
            "attributes": {
                "creationDate": "2024-09-16T20:23:39Z",
                "mfaAuthenticated": "false"
            }
        },
        "invokedBy": "iot.amazonaws.com"
    },
    "eventTime": "2024-09-16T20:32:48Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "iot.amazonaws.com",
```

```
    "userAgent": "iot.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "kms-arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "aws-crypto-ec:vendor": "iot.amazonaws.com",
            "branch-key-id": "111122223333",
            "type": "branch:ACTIVE"
        },
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
        "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "requestID": "1afb6d98-8388-455d-8b48-e62c9e0cf7f4",
    "eventID": "b59a5f16-0d98-46d8-a590-0e040a48b39b",
    "readOnly": true,
    "resources": [
        {
            "accountId": "111122223333",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "eventCategory": "Management"
}
```


Identitäts- und Zugriffsmanagement für AWS IoT

AWS Identity and Access Management (IAM) hilft einem Administrator AWS-Service , den Zugriff auf AWS Ressourcen sicher zu kontrollieren. IAM-Administratoren kontrollieren, wer authentifiziert (angemeldet) und autorisiert werden kann (über Berechtigungen verfügt), um Ressourcen zu verwenden. AWS IoT IAM ist ein Programm AWS-Service , das Sie ohne zusätzliche Kosten nutzen können.

Themen

- Zielgruppe

- Authentifizierung mit IAM-Identitäten
- Verwalten des Zugriffs mit Richtlinien
- Wie AWS IoT funktioniert mit IAM
- AWS IoT Beispiele für identitätsbasierte Richtlinien
- AWS verwaltete Richtlinien für AWS IoT
- Fehlerbehebung AWS IoT Identität und Zugriff

Zielgruppe

Wie Sie AWS Identity and Access Management (IAM) verwenden, hängt von Ihrer Rolle ab:

- Servicebenutzer – Fordern Sie von Ihrem Administrator Berechtigungen an, wenn Sie nicht auf Features zugreifen können (siehe Fehlerbehebung AWS IoT Identität und Zugriff).
- Serviceadministrator – Bestimmen Sie den Benutzerzugriff und stellen Sie Berechtigungsanfragen (siehe Wie AWS IoT funktioniert mit IAM).
- IAM-Administrator – Schreiben Sie Richtlinien zur Zugriffsverwaltung (siehe AWS IoT Beispiele für identitätsbasierte Richtlinien).

Authentifizierung mit IAM-Identitäten

In AWS IoT Identitäten können Geräte (X.509) -Zertifikate, Amazon Cognito Cognito-Identitäten oder IAM-Benutzer oder -Gruppen enthalten sein. In diesem Thema werden nur IAM-Identitäten behandelt. Weitere Informationen zu den anderen Identitäten, die unterstützt werden, finden Sie unter. AWS IoT Client-Authentifizierung

Authentifizierung ist die Art und Weise, wie Sie sich AWS mit Ihren Identitätsdaten anmelden. Sie müssen sich als IAM-Benutzer authentifizieren oder eine IAM-Rolle annehmen. Root-Benutzer des AWS-Kontos

Sie können sich als föderierte Identität anmelden, indem Sie Anmeldeinformationen aus einer Identitätsquelle wie AWS IAM Identity Center (IAM Identity Center), Single Sign-On-Authentifizierung oder Anmeldeinformationen verwenden. Google/Facebook Weitere Informationen zum Anmelden finden Sie unter So melden Sie sich bei Ihrem AWS-Konto an im Benutzerhandbuch für AWS-Anmeldung .

AWS Bietet für den programmatischen Zugriff ein SDK und eine CLI zum kryptografischen Signieren von Anfragen. Weitere Informationen finden Sie unter AWS Signature Version 4 for API requests im IAM-Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie einen erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, dem sogenannten AWS-Konto Root-Benutzer, der vollständigen Zugriff auf alle AWS-Services Ressourcen hat. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Eine Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter Tasks that require root user credentials im IAM-Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein IAM-Benutzer ist eine Identität mit bestimmten Berechtigungen für eine einzelne Person oder Anwendung. Wir empfehlen die Verwendung temporärer Anmeldeinformationen anstelle von IAM-Benutzern mit langfristigen Anmeldeinformationen. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Erfordern, dass menschliche Benutzer für den Zugriff AWS mithilfe temporärer Anmeldeinformationen einen Verbund mit einem Identitätsanbieter verwenden müssen.

Eine IAM-Gruppe spezifiziert eine Sammlung von IAM-Benutzern und erleichtert die Verwaltung von Berechtigungen für große Gruppen von Benutzern. Weitere Informationen finden Sie unter Anwendungsfälle für IAM-Benutzer im IAM-Benutzerhandbuch.

IAM-Rollen

Eine IAM-Rolle ist eine Identität mit spezifischen Berechtigungen, die temporäre Anmeldeinformationen bereitstellt. Sie können eine Rolle übernehmen, indem Sie von einer Benutzer- zu einer IAM-Rolle (Konsole) wechseln AWS CLI oder einen AWS API-Vorgang aufrufen. Weitere Informationen finden Sie unter Methoden, um eine Rolle zu übernehmen im IAM-Benutzerhandbuch.

IAM-Rollen sind nützlich für den Verbundbenutzer-Zugriff, temporäre IAM-Benutzerberechtigungen, kontoübergreifenden Zugriff, serviceübergreifenden Zugriff und Anwendungen, die auf Amazon EC2 laufen. Weitere Informationen finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Verwalten des Zugriffs mit Richtlinien

Sie kontrollieren den Zugriff, AWS indem Sie Richtlinien erstellen und diese an AWS Identitäten oder Ressourcen anhängen. Eine Richtlinie definiert Berechtigungen, wenn sie mit einer Identität oder

Ressource verknüpft sind. AWS bewertet diese Richtlinien, wenn ein Principal eine Anfrage stellt. Die meisten Richtlinien werden AWS als JSON-Dokumente gespeichert. Weitere Informationen zu JSON-Richtliniendokumenten finden Sie unter Übersicht über JSON-Richtlinien im IAM-Benutzerhandbuch.

Mit Hilfe von Richtlinien legen Administratoren fest, wer Zugriff auf was hat, indem sie definieren, welches Prinzipal welche Aktionen auf welchen Ressourcenund unter welchen Bedingungendurchführen darf.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator erstellt IAM-Richtlinien und fügt sie zu Rollen hinzu, die die Benutzer dann übernehmen können. IAM-Richtlinien definieren Berechtigungen unabhängig von der Methode, die zur Ausführung der Operation verwendet wird.

Identity-based Richtlinien

Identity-based Richtlinien sind Richtliniendokumente für JSON-Berechtigungen, die Sie an eine Identität (Benutzer, Gruppe oder Rolle) anhängen. Diese Richtlinien steuern, welche Aktionen Identitäten für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien im IAM-Benutzerhandbuch.

Identity-based Richtlinien können Inline-Richtlinien (direkt in eine einzelne Identität eingebettet) oder verwaltete Richtlinien (eigenständige Richtlinien, die mehreren Identitäten zugeordnet sind) sein. Informationen dazu, wie Sie zwischen verwalteten und Inline-Richtlinien wählen, finden Sie unter Choose between managed policies and inline policies im IAM-Benutzerhandbuch.

Resource-based Richtlinien

Resource-based Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anhängen. Beispiele hierfür sind Vertrauensrichtlinien für IAM-Rollen und Amazon S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben.

Resource-based Richtlinien sind Inline-Richtlinien, die sich in diesem Dienst befinden. Sie können AWS verwaltete Richtlinien von IAM nicht in einer ressourcenbasierten Richtlinie verwenden.

Zugriffssteuerungslisten (ACLs)

Zugriffssteuerungslisten (ACLs) steuern, welche Prinzipale (Kontomitglieder, Benutzer oder Rollen) auf eine Ressource zugreifen können. ACLs sind ähnlich wie ressourcenbasierte Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

Amazon S3 und Amazon VPC sind Beispiele für Services, die ACLs unterstützen. AWS WAF Weitere Informationen“ zu ACLs finden Sie unter Zugriffskontrollliste (ACL) – Übersicht (Access Control List) im Amazon-Simple-Storage-Service-Entwicklerhandbuch.

Weitere Richtlinientypen

AWS unterstützt zusätzliche Richtlinientypen, mit denen die maximalen Berechtigungen festgelegt werden können, die durch gängigere Richtlinientypen gewährt werden:

- Berechtigungsgrenzen – Eine Berechtigungsgrenze legt die maximalen Berechtigungen fest, die eine identitätsbasierte Richtlinie einer IAM-Entität erteilen kann. Weitere Informationen finden Sie unter Berechtigungsgrenzen für IAM-Entitäten im -IAM-Benutzerhandbuch.
- Service-Kontrollrichtlinien (SCPs) – SCPs legen die maximalen Berechtigungen für eine Organisation oder Organisationseinheit in AWS Organizations fest. Weitere Informationen finden Sie unter Service-Kontrollrichtlinien im AWS Organizations -Benutzerhandbuch.
- Ressourcen-Kontrollrichtlinien (RCPs) – RCPs definieren die maximale Anzahl an Berechtigungen, die Ressourcen in Ihren Konten zur Verfügung stehen. Weitere Informationen finden Sie unter Ressourcen-Kontrollrichtlinien im AWS Organizations -Benutzerhandbuch.
- Sitzungsrichtlinien – Sitzungsrichtlinien sind erweiterte Richtlinien, die als Parameter übergeben werden, wenn Sie eine temporäre Sitzung für eine Rolle oder einen Verbundbenutzer erstellen. Weitere Informationen finden Sie unter Sitzungsrichtlinien im IAM-Benutzerhandbuch.

Mehrere Richtlinientypen

Wenn mehrere Arten von Richtlinien für eine Anfrage gelten, sind die sich daraus ergebenden Berechtigungen schwieriger zu verstehen. Informationen darüber, wie AWS bestimmt wird, ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, finden Sie unter Bewertungslogik für Richtlinien im IAM-Benutzerhandbuch.

Wie AWS IoT funktioniert mit IAM

Bevor Sie IAM verwenden, um den Zugriff auf zu verwalten AWS IoT, sollten Sie wissen, mit welchen IAM-Funktionen Sie verwenden können. AWS IoTEinen allgemeinen Überblick darüber, wie AWS IoT und andere AWS Dienste mit IAM funktionieren, finden Sie im IAM-Benutzerhandbuch unter AWS Services That Work with IAM.

Themen

- AWS IoT Identitätsbasierte Richtlinien
- AWS IoT Ressourcenbasierte Richtlinien
- Autorisierung basiert auf AWS IoT tags
- AWS IoT IAM-Rollen

AWS IoT Identitätsbasierte Richtlinien

Mit identitätsbasierten IAM-Richtlinien können Sie angeben, welche Aktionen und Ressourcen erteilt oder abgelehnt werden. Darüber hinaus können Sie die Bedingungen festlegen, unter denen Aktionen zugelassen oder abgelehnt werden. AWS IoT unterstützt bestimmte Aktionen, Ressourcen und Bedingungsschlüssel. Informationen zu sämtlichen Elementen, die Sie in einer JSON-Richtlinie verwenden, finden Sie in der IAM-Referenz für JSON-Richtlinienelemente im IAM-Benutzerhandbuch.

Aktionen

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer auf was Zugriff hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das Element `Action` einer JSON-Richtlinie beschreibt die Aktionen, mit denen Sie den Zugriff in einer Richtlinie zulassen oder verweigern können. Nehmen Sie Aktionen in eine Richtlinie auf, um Berechtigungen zur Ausführung des zugehörigen Vorgangs zu erteilen.

In der folgenden Tabelle sind die IAM-IoT-Aktionen, die zugehörige AWS IoT API und die Ressource aufgeführt, die durch die Aktion manipuliert wird.

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: AcceptCertificateTransfer	AcceptCertificateTransfer	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id* **Note** Das im ARN AWS-Konto angegebene Konto muss das Konto sein, auf das das Zertifikat übertragen wird.
IoT: AddThingToThingGroup	AddThingToThingGroup	`arn:aws:iot:` *region*`:`*account-id* `:thinggroup/` *thing-group-name* `arn:aws:iot:` *region*`:`*account-id* `:thing/`*thing-name*
IoT: AssociateTargetsWithJob	AssociateTargetsWithJob	Keine
IoT: AttachPolicy	AttachPolicy	`arn:aws:iot:` *region*`:`*account-id* `:thinggroup/` *thing-group-name* oder `arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
IoT: AttachPrincipalPolicy	AttachPrincipalPolicy	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
IoT: AttachSecurityProfile	AttachSecurityProfile	`arn:aws:iot:` *region*`:`*account-id* `:securityprofile/` *security-profile-name* `arn:aws:iot:` *region*`:`*account-id* `:dimension/` *dimension-name*

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: AttachThingPrincipal	AttachThingPrincipal	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
IoT: CancelCertificateTransfer	CancelCertificateTransfer	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id* **Note** Das im ARN AWS-Konto angegebene Konto muss das Konto sein, auf das das Zertifikat übertragen wird.
IoT: CancelJob	CancelJob	`arn:aws:iot:` *region*`:`*account-id* `:job/`*job-id*
IoT: CancelJobExecution	CancelJobExecution	`arn:aws:iot:` *region*`:`*account-id* `:job/`*job-id* `arn:aws:iot:` *region*`:`*account-id* `:thing/`*thing-name*
IoT: ClearDefaultAuthorizer	ClearDefaultAuthorizer	Keine
IoT: CreateAuthorizer	CreateAuthorizer	`arn:aws:iot:` *region*`:`*account-id* `:authorizer/` *authorizer-function-name*
IoT: CreateCertificateFromCsr	CreateCertificateFromCsr	*
IoT: CreateDimension	CreateDimension	`arn:aws:iot:` *region*`:`*account-id* `:dimension/` *dimension-name*

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: CreateJob	CreateJob	`arn:aws:iot:` *`region`*`:`*`account-id`* `:job/`*`job-id`* `arn:aws:iot:` *`region`*`:`*`account-id`* `:thinggroup/` *`thing-group-name`* `arn:aws:iot:` *`region`*`:`*`account-id`* `:thing/`*`thing-name`* `arn:aws:iot:` *`region`*`:`*`account-id`* `:jobtemplate/` *`job-template-id`*
IoT: CreateJobTemplate	CreateJobTemplate	`arn:aws:iot:` *`region`*`:`*`account-id`* `:job/`*`job-id`* `arn:aws:iot:` *`region`*`:`*`account-id`* `:jobtemplate/` *`job-template-id`*
IoT: CreateKeysAndCertificate	CreateKeysAndCertificate	*
IoT: CreatePolicy	CreatePolicy	`arn:aws:iot:` *`region`*`:`*`account-id`* `:policy/`*`policy-name`*
IoT: CreatePolicyVersion	CreatePolicyVersion	`arn:aws:iot:` *`region`*`:`*`account-id`* `:policy/`*`policy-name`* **Note** Dies muss eine AWS IoT Richtlinie sein, keine IAM-Richtlinie.
IoT: CreateRoleAlias	CreateRoleAlias	(Parameter: roleAlias) `arn:aws:iot:` *`region`*`:`*`account-id`* `:rolealias/` *`role-alias-name`*

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: CreateSecurityProfile	CreateSecurityProfile	arn:aws:iot: *region*:*account-id* :securityprofile/ *security-profile-name* arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
IoT: CreateThing	CreateThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
IoT: CreateThingGroup	CreateThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* für die Gruppe, die erstellt wird, und für die übergeordnete Gruppe, sofern verwendet
IoT: CreateThingType	CreateThingType	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
IoT: CreateTopicRule	CreateTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
IoT: DeleteAuthorizer	DeleteAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-name*
iot:DeleteCACertificate	DeleteCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*
IoT: DeleteCertificate	DeleteCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
IoT: DeleteDimension	DeleteDimension	arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
IoT: DeleteJob	DeleteJob	arn:aws:iot: *region*:*account-id* :job/*job-id*
IoT: DeleteJobTemplate	DeleteJobTemplate	arn:aws:iot: *region*:*account-id* :job/*job-template-id*

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: DeleteJobExecution	DeleteJobExecution	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
IoT: DeletePolicy	DeletePolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
IoT: DeletePolicyVersion	DeletePolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
IoT: DeleteRegistrationCode	DeleteRegistrationCode	*
IoT: DeleteRoleAlias	DeleteRoleAlias	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
IoT: DeleteSecurityProfile	DeleteSecurityProfile	arn:aws:iot: *region*:*account-id* :securityprofile/ *security-profile-name* arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
IoT: DeleteThing	DeleteThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
IoT: DeleteThingGroup	DeleteThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
IoT: DeleteThingType	DeleteThingType	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
IoT: DeleteTopicRule	DeleteTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
IoT: DeleteV2LoggingLevel	DeleteV2LoggingLevel	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: DeprecateThingType	DeprecateThingType	`arn:aws:iot:` *`region`*`:`*`account-id`* `:thingtype/` *`thing-type-name`*
IoT: DescribeAuthorizer	DescribeAuthorizer	`arn:aws:iot:` *`region`*`:`*`account-id`* `:authorizer/` *`authorizer-function-name`* (Parameter: authorizerName) Keine
iot:DescribeCACertificate	DescribeCACertificate	`arn:aws:iot:` *`region`*`:`*`account-id`* `:cacert/`*`cert-id`*
IoT: DescribeCertificate	DescribeCertificate	`arn:aws:iot:` *`region`*`:`*`account-id`* `:cert/`*`cert-id`*
IoT: DescribeDefaultAuthorizer	DescribeDefaultAuthorizer	Keine
IoT: DescribeEndpoint	DescribeEndpoint	*
IoT: DescribeEventConfigurations	DescribeEventConfigurations	Keine
IoT: DescribeIndex	DescribeIndex	`arn:aws:iot:` *`region`*`:`*`account-id`* `:index/`*`index-name`*
IoT: DescribeJob	DescribeJob	`arn:aws:iot:` *`region`*`:`*`account-id`* `:job/`*`job-id`*
IoT: DescribeJobExecution	DescribeJobExecution	Keine
IoT: DescribeJobTemplate	DescribeJobTemplate	`arn:aws:iot:` *`region`*`:`*`account-id`* `:job/`*`job-template-id`*

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: DescribeRoleAlias	DescribeRoleAlias	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
IoT: DescribeThing	DescribeThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
IoT: DescribeThingGroup	DescribeThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
IoT: DescribeThingRegistrationTask	DescribeThingRegistrationTask	Keine
IoT: DescribeThingType	DescribeThingType	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
IoT: DetachPolicy	DetachPolicy	arn:aws:iot: *region*:*account-id* :cert/*cert-id* oder arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
IoT: DetachPrincipalPolicy	DetachPrincipalPolicy	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
IoT: DetachSecurityProfile	DetachSecurityProfile	arn:aws:iot: *region*:*account-id* :securityprofile/ *security-profile-name* arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
IoT: DetachThingPrincipal	DetachThingPrincipal	arn:aws:iot: *region*:*account-id* :cert/*cert-id*

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: DisableTopicRule	DisableTopicRule	`arn:aws:iot:` *region*`:`*account-id* `:rule/`*rule-name*
IoT: EnableTopicRule	EnableTopicRule	`arn:aws:iot:` *region*`:`*account-id* `:rule/`*rule-name*
IoT: GetEffectivePolicies	GetEffectivePolicies	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
IoT: GetIndexingConfiguration	GetIndexingConfiguration	Keine
IoT: GetJobDocument	GetJobDocument	`arn:aws:iot:` *region*`:`*account-id* `:job/`*job-id*
IoT: GetLoggingOptions	GetLoggingOptions	*
IoT: GetPolicy	GetPolicy	`arn:aws:iot:` *region*`:`*account-id* `:policy/`*policy-name*
IoT: GetPolicyVersion	GetPolicyVersion	`arn:aws:iot:` *region*`:`*account-id* `:policy/`*policy-name*
IoT: GetRegistrationCode	GetRegistrationCode	*
IoT: GetTopicRule	GetTopicRule	`arn:aws:iot:` *region*`:`*account-id* `:rule/`*rule-name*
IoT: ListAttachedPolicies	ListAttachedPolicies	`arn:aws:iot:` *region*`:`*account-id* `:thinggroup/` *thing-group-name* oder `arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: ListAuthorizers	ListAuthorizers	Keine
iot:ListCACertificates	ListCACertificates	*
IoT: ListCertificates	ListCertificates	*
IoT: ListCertificatesByCA	ListCertificatesByCA	*
IoT: ListIndices	ListIndices	Keine
IoT: ListJobExecutionsForJob	ListJobExecutionsForJob	Keine
IoT: ListJobExecutionsForThing	ListJobExecutionsForThing	Keine
IoT: ListJobs	ListJobs	`arn:aws:iot:` *`region`*`:`*`account-id`* `:thinggroup/` *`thing-group-name`* wenn der GroupName Ding-Parameter verwendet wird
IoT: ListJobTemplates	ListJobs	Keine
IoT: ListOutgoingCertificates	ListOutgoingCertificates	*
IoT: ListPolicies	ListPolicies	*
IoT: ListPolicyPrincipals	ListPolicyPrincipals	*

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: ListPolicyVersions	ListPolicyVersions	`arn:aws:iot:` *region*`:`*account-id* `:policy/`*policy-name*
IoT: ListPrincipalPolicies	ListPrincipalPolicies	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
IoT: ListPrincipalThings	ListPrincipalThings	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
IoT: ListRoleAliases	ListRoleAliases	Keine
IoT: ListTargetsForPolicy	ListTargetsForPolicy	`arn:aws:iot:` *region*`:`*account-id* `:policy/`*policy-name*
IoT: ListThingGroups	ListThingGroups	Keine
IoT: ListThingGroupsForThing	ListThingGroupsForThing	`arn:aws:iot:` *region*`:`*account-id* `:thing/`*thing-name*
IoT: ListThingPrincipals	ListThingPrincipals	`arn:aws:iot:` *region*`:`*account-id* `:thing/`*thing-name*
IoT: ListThingRegistrationTaskReports	ListThingRegistrationTaskReports	Keine
IoT: ListThingRegistrationTasks	ListThingRegistrationTasks	Keine
IoT: ListThingTypes	ListThingTypes	*
IoT: ListThings	ListThings	*

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: ListThingsInThingGroup	ListThingsInThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
IoT: ListTopicRules	ListTopicRules	*
IoT: ListV2LoggingLevels	ListV2LoggingLevels	Keine
iot:RegisterCACertificate	RegisterCACertificate	*
IoT: RegisterCertificate	RegisterCertificate	*
IoT: RegisterThing	RegisterThing	Keine
IoT: RejectCertificateTransfer	RejectCertificateTransfer	arn:aws:iot: *region*:*account-id* :cert/*certificate-id*
IoT: RemoveThingFromThingGroup	RemoveThingFromThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
IoT: ReplaceTopicRule	ReplaceTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
IoT: SearchIndex	SearchIndex	arn:aws:iot: *region*:*account-id* :index/*index-id*
IoT: SetDefaultAuthorizer	SetDefaultAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-function-name*

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: SetDefaultPolicyVersion	SetDefaultPolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
IoT: SetLoggingOptions	SetLoggingOptions	arn:aws:iot: *region*:*account-id* :role/*role-name*
IoT: SetV2LoggingLevel	SetV2LoggingLevel	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
IoT: SetV2LoggingOptions	SetV2LoggingOptions	arn:aws:iot: *region*:*account-id* :role/*role-name*
IoT: StartThingRegistrationTask	StartThingRegistrationTask	Keine
IoT: StopThingRegistrationTask	StopThingRegistrationTask	Keine
IoT: TestAuthorization	TestAuthorization	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
IoT: TestInvokeAuthorizer	TestInvokeAuthorizer	Keine
IoT: TransferCertificate	TransferCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
IoT: UpdateAuthorizer	UpdateAuthorizer	arn:aws:iot: *region*:*account-id* :authorizerfunction/ *authorizer-function-name*
iot:UpdateCACertificate	UpdateCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*
IoT: UpdateCertificate	UpdateCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: UpdateDimension	UpdateDimension	arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
IoT: UpdateEventConfigurations	UpdateEventConfigurations	Keine
IoT: UpdateIndexingConfiguration	UpdateIndexingConfiguration	Keine
IoT: UpdateRoleAlias	UpdateRoleAlias	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
IoT: UpdateSecurityProfile	UpdateSecurityProfile	arn:aws:iot: *region*:*account-id* :securityprofile/ *security-profile-name* arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
IoT: UpdateThing	UpdateThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
IoT: UpdateThingGroup	UpdateThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
IoT: UpdateThingGroupsForThing	UpdateThingGroupsForThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name* arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*

Bei Richtlinienaktionen wird vor der Aktion das folgende Präfix AWS IoT verwendet:`iot:`. Um beispielsweise jemandem die Erlaubnis zu erteilen, alle IoT-Dinge aufzulisten, die in seiner AWS-Konto `ListThings` API registriert sind, nehmen Sie die `iot:ListThings` Aktion in seine Richtlinie auf. Richtlinienerklärungen müssen `Action` entweder ein `NotAction` Oder-Element enthalten. AWS

IoT definiert einen eigenen Satz von Aktionen, die Aufgaben beschreiben, die Sie mit diesem Dienst ausführen können.

Um mehrere Aktionen in einer einzigen Anweisung anzugeben, trennen Sie sie wie folgt durch Kommata:

```
"Action": [
      "ec2:action1",
      "ec2:action2"
```

Sie können auch Platzhalter verwenden, um mehrere Aktionen anzugeben. Beispielsweise können Sie alle Aktionen festlegen, die mit dem Wort `Describe` beginnen, einschließlich der folgenden Aktion:

```
"Action": "iot:Describe*"
```

Eine Liste der AWS IoT Aktionen finden Sie AWS IoT im IAM-Benutzerhandbuch unter Definierte Aktionen von.

Device-Advisor-Aktionen

Die folgende Tabelle listet die Device-Advisor-Aktionen im IAM-IoT, die zugehörige AWS IoT -Device-Advisor-API und die Ressource auf, die die Aktion bearbeitet.

Richtlinienaktionen	AWS IoT API	Ressourcen
Berater für IoT-Geräte: CreateSuiteDefinition	CreateSuiteDefinition	Keine
iotdeviceadvisor: DeleteSuiteDefinition	DeleteSuiteDefinition	`arn:aws:iotdeviceadvisor:region:account-id:suitedefinition/suite-definition-id`
iotdeviceadvisor: GetSuiteDefinition	GetSuiteDefinition	`arn:aws:iotdeviceadvisor:region:account-id:suitedefinition/suite-definition-id`

Richtlinienaktionen	AWS IoT API	Ressourcen
iotdeviceadvisor:GetSuiteRun	GetSuiteRun	arn:aws:iotdeviceadvisor:*region*:*account-id*:suitedefinition/*suite-run-id*
iotdeviceadvisor:GetSuiteRunReport	GetSuiteRunReport	arn:aws:iotdeviceadvisor:*region*:*account-id*:suiterun/*suite-definition-id*/*suite-run-id*
iotdeviceadvisor:ListSuiteDefinitions	ListSuiteDefinitions	Keine
iotdeviceadvisor:ListSuiteRuns	ListSuiteRuns	arn:aws:iotdeviceadvisor:*region*:*account-id*:suitedefinition/*suite-definition-id*
iotdeviceadvisor:ListTagsForResource	ListTagsForResource	arn:aws:iotdeviceadvisor:*region*:*account-id*:suitedefinition/*suite-definition-id* arn:aws:iotdeviceadvisor:*region*:*account-id*:suiterun/suite-definition-id/*suite-run-id*
iotdeviceadvisor:StartSuiteRun	StartSuiteRun	arn:aws:iotdeviceadvisor:*region*:*account-id*:suitedefinition/*suite-definition-id*
iotdeviceadvisor:TagResource	TagResource	arn:aws:iotdeviceadvisor:*region*:*account-id*:suitedefinition/*suite-definition-id* arn:aws:iotdeviceadvisor:*region*:*account-id*:suiterun/suite-definition-id/*suite-run-id*

Richtlinienaktionen	AWS IoT API	Ressourcen
iotdeviceadvisor:UntagResource	UntagResource	arn:aws:iotdeviceadvisor:*region*:*account-id*:suitedefinition/*suite-definition-id* arn:aws:iotdeviceadvisor:*region*:*account-id*:suiterun/suite-definition-id/*suite-run-id*
iotdeviceadvisor:UpdateSuiteDefinition	UpdateSuiteDefinition	arn:aws:iotdeviceadvisor:*region*:*account-id*:suitedefinition/*suite-definition-id*
iotdeviceadvisor:StopSuiteRun	StopSuiteRun	arn:aws:iotdeviceadvisor:*region*:*account-id*:suiterun/suite-definition-id/*suite-run-id*

Für Richtlinienaktionen in AWS IoT Device Advisor wird vor der Aktion das folgende Präfix verwendet:. `iotdeviceadvisor:` Um beispielsweise jemandem die Erlaubnis zu erteilen, alle in seiner AWS-Konto ListSuiteDefinitions API registrierten Suite-Definitionen aufzulisten, nehmen Sie die `iotdeviceadvisor:ListSuiteDefinitions` Aktion in seine Richtlinie auf.

Ressourcen

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das JSON-Richtlinienelement `Resource` gibt die Objekte an, auf welche die Aktion angewendet wird. Als Best Practice geben Sie eine Ressource mit dem zugehörigen Amazon-Ressourcennamen (ARN) an. Verwenden Sie für Aktionen, die keine Berechtigungen auf Ressourcenebene unterstützen, einen Platzhalter (*), um anzugeben, dass die Anweisung für alle Ressourcen gilt.

```
"Resource": "*"
```

AWS IoT Ressourcen

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: AcceptCertificateTransfer	AcceptCertificateTransfer	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id* **Note** Das im ARN AWS-Konto angegebene Konto muss das Konto sein, auf das das Zertifikat übertragen wird.
IoT: AddThingToThingGroup	AddThingToThingGroup	`arn:aws:iot:` *region*`:`*account-id* `:thinggroup/` *thing-group-name* `arn:aws:iot:` *region*`:`*account-id* `:thing/`*thing-name*
IoT: AssociateTargetsWithJob	AssociateTargetsWithJob	Keine
IoT: AttachPolicy	AttachPolicy	`arn:aws:iot:` *region*`:`*account-id* `:thinggroup/` *thing-group-name* oder `arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
IoT: AttachPrincipalPolicy	AttachPrincipalPolicy	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
IoT: AttachThingPrincipal	AttachThingPrincipal	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
IoT: CancelCertificateTransfer	CancelCertificateTransfer	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*

Richtlinienaktionen	AWS IoT API	Ressourcen
		Note Das im ARN AWS-Konto angegebene Konto muss das Konto sein, auf das das Zertifikat übertragen wird.
IoT: CancelJob	CancelJob	`arn:aws:iot:` *region*`:`*account-id* `:job/`*job-id*
IoT: CancelJob Execution	CancelJob Execution	`arn:aws:iot:` *region*`:`*account-id* `:job/`*job-id* `arn:aws:iot:` *region*`:`*account-id* `:thing/`*thing-name*
IoT: ClearDefaultAuthorizer	ClearDefaultAuthorizer	Keine
IoT: CreateAuthorizer	CreateAuthorizer	`arn:aws:iot:` *region*`:`*account-id* `:authorizer/` *authorizer-function-name*
IoT: CreateCertificateFromCsr	CreateCertificateFromCsr	*
IoT: CreateJob	CreateJob	`arn:aws:iot:` *region*`:`*account-id* `:job/`*job-id* `arn:aws:iot:` *region*`:`*account-id* `:thinggroup/` *thing-group-name* `arn:aws:iot:` *region*`:`*account-id* `:thing/`*thing-name* `arn:aws:iot:` *region*`:`*account-id* `:jobtemplate/` *job-template-id*

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: CreateJobTemplate	CreateJobTemplate	`arn:aws:iot:` *region*`:`*account-id*` :job/`*job-id* `arn:aws:iot:` *region*`:`*account-id*` :jobtemplate/` *job-template-id*
IoT: CreateKeysAndCertificate	CreateKeysAndCertificate	*
IoT: CreatePolicy	CreatePolicy	`arn:aws:iot:` *region*`:`*account-id*` :policy/`*policy-name*
CreatePolicyVersion	IoT: CreatePolicyVersion	`arn:aws:iot:` *region*`:`*account-id*` :policy/`*policy-name* **Note** Dies muss eine AWS IoT Richtlinie sein, keine IAM-Richtlinie.
IoT: CreateRoleAlias	CreateRoleAlias	(Parameter: roleAlias) `arn:aws:iot:` *region*`:`*account-id*` :rolealias/` *role-alias-name*
IoT: CreateThing	CreateThing	`arn:aws:iot:` *region*`:`*account-id*` :thing/`*thing-name*
IoT: CreateThingGroup	CreateThingGroup	`arn:aws:iot:` *region*`:`*account-id*` :thinggroup/` *thing-group-name* für die Gruppe, die erstellt wird, und für die übergeordnete Gruppe, sofern verwendet
IoT: CreateThingType	CreateThingType	`arn:aws:iot:` *region*`:`*account-id*` :thingtype/` *thing-type-name*

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: CreateTopicRule	CreateTopicRule	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
IoT: DeleteAuthorizer	DeleteAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-name*
iot:DeleteCACertificate	DeleteCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*
IoT: DeleteCertificate	DeleteCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
IoT: DeleteJob	DeleteJob	arn:aws:iot: *region*:*account-id* :job/*job-id*
IoT: DeleteJobExecution	DeleteJobExecution	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
IoT: DeleteJobTemplate	DeleteJobTemplate	arn:aws:iot: *region*:*account-id* :jobtemplate/ *job-template-id*
IoT: DeletePolicy	DeletePolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
IoT: DeletePolicyVersion	DeletePolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
IoT: DeleteRegistrationCode	DeleteRegistrationCode	*
IoT: DeleteRoleAlias	DeleteRoleAlias	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
IoT: DeleteThing	DeleteThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: DeleteThingGroup	DeleteThingGroup	`arn:aws:iot:` *region*`:`*account-id* `:thinggroup/` *thing-group-name*
IoT: DeleteThingType	DeleteThingType	`arn:aws:iot:` *region*`:`*account-id* `:thingtype/` *thing-type-name*
IoT: DeleteTopicRule	DeleteTopicRule	`arn:aws:iot:` *region*`:`*account-id* `:rule/`*rule-name*
IoT: DeleteV2LoggingLevel	DeleteV2LoggingLevel	`arn:aws:iot:` *region*`:`*account-id* `:thinggroup/` *thing-group-name*
IoT: DeprecateThingType	DeprecateThingType	`arn:aws:iot:` *region*`:`*account-id* `:thingtype/` *thing-type-name*
IoT: DescribeAuthorizer	DescribeAuthorizer	`arn:aws:iot:` *region*`:`*account-id* `:authorizer/` *authorizer-function-name* (Parameter: authorizerName) Keine
iot:DescribeCACertificate	DescribeCACertificate	`arn:aws:iot:` *region*`:`*account-id* `:cacert/`*cert-id*
IoT: DescribeCertificate	DescribeCertificate	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
IoT: DescribeDefaultAuthorizer	DescribeDefaultAuthorizer	Keine
IoT: DescribeEndpoint	DescribeEndpoint	*
IoT: DescribeEventConfigurations	DescribeEventConfigurations	Keine

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: DescribeIndex	DescribeIndex	arn:aws:iot: *region*:*account-id* :index/*index-name*
IoT: DescribeJob	DescribeJob	arn:aws:iot: *region*:*account-id* :job/*job-id*
IoT: DescribeJobExecution	DescribeJobExecution	Keine
IoT: DescribeJobTemplate	DescribeJobTemplate	arn:aws:iot: *region*:*account-id* :jobtemplate/ *job-template-id*
IoT: DescribeRoleAlias	DescribeRoleAlias	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
IoT: DescribeThing	DescribeThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
IoT: DescribeThingGroup	DescribeThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
IoT: DescribeThingRegistrationTask	DescribeThingRegistrationTask	Keine
IoT: DescribeThingType	DescribeThingType	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
IoT: DetachPolicy	DetachPolicy	arn:aws:iot: *region*:*account-id* :cert/*cert-id* oder arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
IoT: DetachPrincipalPolicy	DetachPrincipalPolicy	arn:aws:iot: *region*:*account-id* :cert/*cert-id*

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: DetachThingPrincipal	DetachThingPrincipal	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
IoT: DisableTopicRule	DisableTopicRule	`arn:aws:iot:` *region*`:`*account-id* `:rule/`*rule-name*
IoT: EnableTopicRule	EnableTopicRule	`arn:aws:iot:` *region*`:`*account-id* `:rule/`*rule-name*
IoT: GetEffectivePolicies	GetEffectivePolicies	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
IoT: GetIndexingConfiguration	GetIndexingConfiguration	Keine
IoT: GetJobDocument	GetJobDocument	`arn:aws:iot:` *region*`:`*account-id* `:job/`*job-id*
IoT: GetLoggingOptions	GetLoggingOptions	*
IoT: GetPolicy	GetPolicy	`arn:aws:iot:` *region*`:`*account-id* `:policy/`*policy-name*
IoT: GetPolicyVersion	GetPolicyVersion	`arn:aws:iot:` *region*`:`*account-id* `:policy/`*policy-name*
IoT: GetRegistrationCode	GetRegistrationCode	*
IoT: GetTopicRule	GetTopicRule	`arn:aws:iot:` *region*`:`*account-id* `:rule/`*rule-name*

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: ListAttachedPolicies	ListAttachedPolicies	`arn:aws:iot:` *`region`*`:`*`account-id`* `:thinggroup/` *`thing-group-name`* oder `arn:aws:iot:` *`region`*`:`*`account-id`* `:cert/`*`cert-id`*
IoT: ListAuthorizers	ListAuthorizers	Keine
iot:ListCACertificates	ListCACertificates	*
IoT: ListCertificates	ListCertificates	*
IoT: ListCertificatesByCA	ListCertificatesByCA	*
IoT: ListIndices	ListIndices	Keine
IoT: ListJobExecutionsForJob	ListJobExecutionsForJob	Keine
IoT: ListJobExecutionsForThing	ListJobExecutionsForThing	Keine
IoT: ListJobs	ListJobs	`arn:aws:iot:` *`region`*`:`*`account-id`* `:thinggroup/` *`thing-group-name`* wenn der GroupName Ding-Parameter verwendet wird
IoT: ListJobTemplates	ListJobTemplates	Keine

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: ListOutgoingCertificates	ListOutgoingCertificates	*
IoT: ListPolicies	ListPolicies	*
IoT: ListPolicyPrincipals	ListPolicyPrincipals	`arn:aws:iot:` *region*`:`*account-id* `:policy/`*policy-name*
IoT: ListPolicyVersions	ListPolicyVersions	`arn:aws:iot:` *region*`:`*account-id* `:policy/`*policy-name*
IoT: ListPrincipalPolicies	ListPrincipalPolicies	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
IoT: ListPrincipalThings	ListPrincipalThings	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
IoT: ListRoleAliases	ListRoleAliases	Keine
IoT: ListTargetsForPolicy	ListTargetsForPolicy	`arn:aws:iot:` *region*`:`*account-id* `:policy/`*policy-name*
IoT: ListThingGroups	ListThingGroups	Keine
IoT: ListThingGroupsForThing	ListThingGroupsForThing	`arn:aws:iot:` *region*`:`*account-id* `:thing/`*thing-name*
IoT: ListThingPrincipals	ListThingPrincipals	`arn:aws:iot:` *region*`:`*account-id* `:thing/`*thing-name*
IoT: ListThingRegistrationTaskReports	ListThingRegistrationTaskReports	Keine

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: ListThingRegistrationTasks	ListThingRegistrationTasks	Keine
IoT: ListThingTypes	ListThingTypes	*
IoT: ListThings	ListThings	*
IoT: ListThingsInThingGroup	ListThingsInThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
IoT: ListTopicRules	ListTopicRules	*
IoT: ListV2LoggingLevels	ListV2LoggingLevels	Keine
iot:RegisterCACertificate	RegisterCACertificate	*
IoT: RegisterCertificate	RegisterCertificate	*
IoT: RegisterThing	RegisterThing	Keine
IoT: RejectCertificateTransfer	RejectCertificateTransfer	arn:aws:iot: *region*:*account-id* :cert/*certificate-id*
IoT: RemoveThingFromThingGroup	RemoveThingFromThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* arn:aws:iot: *region*:*account-id* :thing/*thing-name*

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: ReplaceTopicRule	ReplaceTopicRule	`arn:aws:iot:` *region*`:`*account-id* `:rule/`*rule-name*
IoT: SearchIndex	SearchIndex	`arn:aws:iot:` *region*`:`*account-id* `:index/`*index-id*
IoT: SetDefaultAuthorizer	SetDefaultAuthorizer	`arn:aws:iot:` *region*`:`*account-id* `:authorizer/` *authorizer-function-name*
IoT: SetDefaultPolicyVersion	SetDefaultPolicyVersion	`arn:aws:iot:` *region*`:`*account-id* `:policy/`*policy-name*
IoT: SetLoggingOptions	SetLoggingOptions	*
IoT: SetV2LoggingLevel	SetV2LoggingLevel	*
IoT: SetV2LoggingOptions	SetV2LoggingOptions	*
IoT: StartThingRegistrationTask	StartThingRegistrationTask	Keine
IoT: StopThingRegistrationTask	StopThingRegistrationTask	Keine
IoT: TestAuthorization	TestAuthorization	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
IoT: TestInvokeAuthorizer	TestInvokeAuthorizer	Keine
IoT: TransferCertificate	TransferCertificate	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*

Richtlinienaktionen	AWS IoT API	Ressourcen
IoT: UpdateAuthorizer	UpdateAuthorizer	`arn:aws:iot:` *region*`:`*account-id* `:authorizerfunction/` *authorizer-function-name*
iot:UpdateCACertificate	UpdateCACertificate	`arn:aws:iot:` *region*`:`*account-id* `:cacert/`*cert-id*
IoT: UpdateCertificate	UpdateCertificate	`arn:aws:iot:` *region*`:`*account-id* `:cert/`*cert-id*
IoT: UpdateEventConfigurations	UpdateEventConfigurations	Keine
IoT: UpdateIndexingConfiguration	UpdateIndexingConfiguration	Keine
IoT: UpdateRoleAlias	UpdateRoleAlias	`arn:aws:iot:` *region*`:`*account-id* `:rolealias/` *role-alias-name*
IoT: UpdateThing	UpdateThing	`arn:aws:iot:` *region*`:`*account-id* `:thing/`*thing-name*
IoT: UpdateThingGroup	UpdateThingGroup	`arn:aws:iot:` *region*`:`*account-id* `:thinggroup/` *thing-group-name*
IoT: UpdateThingGroupsForThing	UpdateThingGroupsForThing	`arn:aws:iot:` *region*`:`*account-id* `:thing/`*thing-name*

Weitere Informationen zum Format von ARNs finden Sie unter Amazon Resource Names (ARNs) und AWS Service Namespaces.

Einige AWS IoT Aktionen, z. B. die zum Erstellen von Ressourcen, können nicht für eine bestimmte Ressource ausgeführt werden. In diesen Fällen müssen Sie den Platzhalter (*) verwenden.

```
"Resource": "*"
```

Eine Liste der AWS IoT Ressourcentypen und ihrer ARNs finden Sie AWS IoT im IAM-Benutzerhandbuch unter Defined by (Ressourcen definiert von). Informationen zu den Aktionen, mit denen Sie den ARN einzelner Ressourcen angeben können, finden Sie unter Von AWS IoT definierte Aktionen.

Device-Advisor-Ressourcen

Verwenden Sie die folgenden Ressourcen-ARN-Formate für Suite-Definitionen und Suite-Läufe, um Einschränkungen auf Ressourcenebene für AWS IoT Device Advisor IAM-Richtlinien zu definieren.

ARN–Format der Suite-Definitionsressource

```
arn:aws:iotdeviceadvisor:region:account-id:suitedefinition/suite-definition-id
```

ARN–Format der Suite-Ausführungsressource

```
arn:aws:iotdeviceadvisor:region:account-id:suiterun/suite-definition-id/suite-run-id
```

Bedingungsschlüssel

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das Element `Condition` gibt an, wann Anweisungen auf der Grundlage definierter Kriterien ausgeführt werden. Sie können bedingte Ausdrücke erstellen, die Bedingungsoperatoren verwenden, z. B. ist gleich oder kleiner als, damit die Bedingung in der Richtlinie mit Werten in der Anforderung übereinstimmt. Eine Übersicht aller AWS globalen Bedingungsschlüssel finden Sie unter Kontextschlüssel für AWS globale Bedingungen im IAM-Benutzerhandbuch.

AWS IoT definiert seinen eigenen Satz von Bedingungsschlüsseln und unterstützt auch die Verwendung einiger globaler Bedingungsschlüssel. Eine Übersicht aller AWS globalen Bedingungsschlüssel finden Sie unter AWS Globale Bedingungskontextschlüssel im IAM-Benutzerhandbuch.

AWS IoT Bedingungsschlüssel

AWS IoT Zustandsschlüssel	Beschreibung	Typ
`aws:RequestTag/${tag-key}`	Ein Tag-Schlüssel, der in der Anforderung vorhanden ist, die der Benutzer an AWS IoT stellt.	Zeichenfolge
`aws:ResourceTag/${tag-key}`	Die Schlüsselkomponente eines Tags, das an eine AWS IoT Ressource angehängt ist.	Zeichenfolge
`aws:TagKeys`	Liste aller Tag-Schlüsselnamen, die der Ressource in der Anforderung zugeordnet sind	Zeichenfolge

Eine Liste der AWS IoT Bedingungsschlüssel finden Sie unter Bedingungsschlüssel für AWS IoT im IAM-Benutzerhandbuch. Informationen zu den Aktionen und Ressourcen, mit denen Sie einen Bedingungsschlüssel verwenden können, finden Sie unter Definierte Aktionen von AWS IoT.

Beispiele

Beispiele für AWS IoT identitätsbasierte Richtlinien finden Sie unter. AWS IoT Beispiele für identitätsbasierte Richtlinien

AWS IoT Ressourcenbasierte Richtlinien

Resource-based Richtlinien sind JSON-Richtliniendokumente, die angeben, welche Aktionen ein bestimmter Prinzipal auf der AWS IoT Ressource ausführen kann und unter welchen Bedingungen.

AWS IoT unterstützt keine ressourcenbasierten IAM-Richtlinien. Es unterstützt AWS IoT jedoch ressourcenbasierte Richtlinien. Weitere Informationen finden Sie unter AWS IoT Core Richtlinien.

Autorisierung basiert auf AWS IoT tags

Sie können Tags an AWS IoT Ressourcen anhängen oder Tags in einer Anfrage an übergeben AWS IoT. Um den Zugriff auf der Grundlage von Tags zu steuern, geben Sie im Bedingungselement einer Richtlinie Tag-Informationen an, indem Sie die Schlüssel `iot:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, oder Bedingung `aws:TagKeys` verwenden. Weitere Informationen finden Sie unter Verwenden von Tags mit IAM-Richtlinien. Weitere Informationen zum Markieren von AWS IoT Ressourcen finden Sie unterTaggen Sie Ihre AWS IoT Ressourcen.

Ein Beispiel für eine identitätsbasierte Richtlinie zur Einschränkung des Zugriffs auf eine Ressource auf der Grundlage der Markierungen dieser Ressource finden Sie unter Wird angezeigt AWS IoT Ressourcen, die auf Tags basieren.

AWS IoT IAM-Rollen

Eine IAM-Rolle ist eine Entität innerhalb von Ihnen AWS-Konto , die über bestimmte Berechtigungen verfügt.

Verwenden temporärer Anmeldeinformationen mit AWS IoT

Sie können temporäre Anmeldeinformationen verwenden, um sich über einen Verbund anzumelden, eine IAM-Rolle anzunehmen oder eine kontenübergreifende Rolle anzunehmen. Sie erhalten temporäre Sicherheitsanmeldedaten, indem Sie AWS STS API-Operationen wie AssumeRoleoder aufrufen GetFederationToken.

AWS IoT unterstützt die Verwendung temporärer Anmeldeinformationen.

Service-linked Rollen

Service-linked Rollen ermöglichen es AWS Diensten, auf Ressourcen in anderen Diensten zuzugreifen, um eine Aktion in Ihrem Namen auszuführen. Service-linked Rollen werden in Ihrem IAM-Konto angezeigt und gehören dem Dienst. Ein IAM-Administrator kann die Berechtigungen für serviceverknüpfte Rollen anzeigen, aber nicht bearbeiten.

AWS IoT unterstützt keine dienstbezogenen Rollen.

Servicerollen

Dieses Feature ermöglicht einem Service das Annehmen einer Servicerolle in Ihrem Namen. Diese Rolle gewährt dem Service Zugriff auf Ressourcen in anderen Diensten, um eine Aktion in Ihrem Namen auszuführen. Servicerollen werden in Ihrem IAM-Konto angezeigt und gehören zum Konto. Dies bedeutet, dass ein IAM-Administrator die Berechtigungen für diese Rolle ändern kann. Dies kann jedoch die Funktionalität des Dienstes beeinträchtigen.

AWS IoT Beispiele für identitätsbasierte Richtlinien

IAM-Benutzer besitzen keine Berechtigungen zum Erstellen oder Ändern von AWS IoT -Ressourcen. Sie können auch keine Aufgaben mit der AWS-Managementkonsole AWS CLI, oder AWS API ausführen. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern und Rollen die Berechtigung zum Ausführen bestimmter API-Operationen für die angegebenen Ressourcen gewähren, die diese benötigen. Der Administrator muss diese Richtlinien anschließend den -Benutzern oder -Gruppen anfügen, die diese Berechtigungen benötigen.

Informationen dazu, wie Sie unter Verwendung dieser beispielhaften JSON-Richtliniendokumente eine identitätsbasierte IAM-Richtlinie erstellen, finden Sie unter Erstellen von Richtlinien auf der JSON-Registerkarte im IAM-Benutzerhandbuch.

Themen

- Best Practices für Richtlinien
- Verwendung der AWS IoT Konsole
- Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer
- Wird angezeigt AWS IoT Ressourcen, die auf Tags basieren
- Wird angezeigt AWS IoT Device Advisor-Ressourcen, die auf Tags basieren

Best Practices für Richtlinien

Identity-based Richtlinien legen fest, ob jemand AWS IoT Ressourcen in Ihrem Konto erstellen, darauf zugreifen oder sie löschen kann. Dies kann zusätzliche Kosten für Ihr verursachen AWS-Konto. Beachten Sie beim Erstellen oder Bearbeiten identitätsbasierter Richtlinien die folgenden Richtlinien und Empfehlungen:

- Erste Schritte mit AWS verwalteten Richtlinien und Umstellung auf Berechtigungen mit den geringsten Rechten — Verwenden Sie die AWS verwalteten Richtlinien, die Berechtigungen für viele gängige Anwendungsfälle gewähren, um damit zu beginnen, Ihren Benutzern und Workloads Berechtigungen zu gewähren. Sie sind in Ihrem verfügbar. AWS-Konto Wir empfehlen Ihnen, die Berechtigungen weiter zu reduzieren, indem Sie vom AWS Kunden verwaltete Richtlinien definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind. Weitere Informationen finden Sie unter Von AWS verwaltete Richtlinien oder Von AWS verwaltete Richtlinien für Auftragsfunktionen im IAM-Benutzerhandbuch.
- Anwendung von Berechtigungen mit den geringsten Rechten – Wenn Sie mit IAM-Richtlinien Berechtigungen festlegen, gewähren Sie nur die Berechtigungen, die für die Durchführung einer Aufgabe erforderlich sind. Sie tun dies, indem Sie die Aktionen definieren, die für bestimmte Ressourcen unter bestimmten Bedingungen durchgeführt werden können, auch bekannt als die geringsten Berechtigungen. Weitere Informationen zur Verwendung von IAM zum Anwenden von Berechtigungen finden Sie unter Richtlinien und Berechtigungen in IAM im IAM-Benutzerhandbuch.
- Verwenden von Bedingungen in IAM-Richtlinien zur weiteren Einschränkung des Zugriffs – Sie können Ihren Richtlinien eine Bedingung hinzufügen, um den Zugriff auf Aktionen und Ressourcen zu beschränken. Sie können beispielsweise eine Richtlinienbedingung schreiben, um festzulegen, dass alle Anforderungen mithilfe von SSL gesendet werden müssen. Sie können auch Bedingungen verwenden, um Zugriff auf Serviceaktionen zu gewähren, wenn diese für einen bestimmten Zweck verwendet werden AWS-Service, z. CloudFormation B. Weitere Informationen finden Sie unter IAM-JSON-Richtlinienelemente: Bedingung im IAM-Benutzerhandbuch.
- Verwenden von IAM Access Analyzer zur Validierung Ihrer IAM-Richtlinien, um sichere und funktionale Berechtigungen zu gewährleisten – IAM Access Analyzer validiert neue und vorhandene Richtlinien, damit die Richtlinien der IAM-Richtliniensprache (JSON) und den bewährten IAM-Methoden entsprechen. IAM Access Analyzer stellt mehr als 100 Richtlinienprüfungen und umsetzbare Empfehlungen zur Verfügung, damit Sie sichere und funktionale Richtlinien erstellen können. Weitere Informationen finden Sie unter Richtlinienvalidierung mit IAM Access Analyzer im IAM-Benutzerhandbuch.
- Multi-Faktor-Authentifizierung (MFA) erforderlich — Wenn Sie ein Szenario haben, das IAM-Benutzer oder einen Root-Benutzer in Ihrem System erfordert AWS-Konto, aktivieren Sie MFA für zusätzliche Sicherheit. Um MFA beim Aufrufen von API-Vorgängen anzufordern, fügen Sie Ihren Richtlinien MFA-Bedingungen hinzu. Weitere Informationen finden Sie unter Sicherer API-Zugriff mit MFA im IAM-Benutzerhandbuch.

Weitere Informationen zu bewährten Methoden in IAM finden Sie unter Best Practices für die Sicherheit in IAM im IAM-Benutzerhandbuch.

Verwendung der AWS IoT Konsole

Um auf die AWS IoT Konsole zugreifen zu können, benötigen Sie ein Mindestmaß an Berechtigungen. Diese Berechtigungen müssen es Ihnen ermöglichen, Details zu den AWS IoT Ressourcen in Ihrem aufzulisten und anzuzeigen AWS-Konto. Wenn Sie eine identitätsbasierte Richtlinie erstellen, die strenger ist als die mindestens erforderlichen Berechtigungen, funktioniert die Konsole nicht wie vorgesehen für Entitäten (Benutzer oder Rollen) mit dieser Richtlinie.

Um sicherzustellen, dass diese Entitäten die AWS IoT Konsole weiterhin verwenden können, fügen Sie den Entitäten außerdem die folgende AWS verwaltete Richtlinie hinzu:`AWSIoTFullAccess`. Weitere Informationen finden Sie unter Hinzufügen von Berechtigungen zu einem Benutzer im IAM-Benutzerhandbuch.

Sie müssen Benutzern, die nur die API AWS CLI oder die AWS API aufrufen, keine Mindestberechtigungen für die Konsole gewähren. Stattdessen sollten Sie nur Zugriff auf die Aktionen zulassen, die den API-Operation entsprechen, die Sie ausführen möchten.

Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer

In diesem Beispiel wird gezeigt, wie Sie eine Richtlinie erstellen, die IAM-Benutzern die Berechtigung zum Anzeigen der eingebundenen Richtlinien und verwalteten Richtlinien gewährt, die ihrer Benutzeridentität angefügt sind. Diese Richtlinie umfasst Berechtigungen zum Ausführen dieser Aktion auf der Konsole oder programmgesteuert mithilfe der API AWS CLI oder AWS .

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
```

```
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Wird angezeigt AWS IoT Ressourcen, die auf Tags basieren

Sie können in Ihrer identitätsbasierten Richtlinie Bedingungen für die Steuerung des Zugriffs auf AWS IoT -Ressourcen auf der Basis von Tags verwenden. Dieses Beispiel zeigt, wie Sie eine Richtlinie erstellen können, die das Anzeigen eines Things ermöglicht. Die Berechtigung wird jedoch nur erteilt, wenn das Things-Tag `Owner` den Wert des Benutzernamens dieses Benutzers hat. Diese Richtlinie gewährt auch die Berechtigungen, die für die Ausführung dieser Aktion auf der Konsole erforderlich sind.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ListBillingGroupsInConsole",
            "Effect": "Allow",
            "Action": "iot:ListBillingGroups",
            "Resource": "*"
        },
        {
            "Sid": "ViewBillingGroupsIfOwner",
            "Effect": "Allow",
           "Action": "iot:DescribeBillingGroup",
```

```
            "Resource": "arn:aws:iot:*:*:billinggroup/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/Owner": "${aws:username}"}
        }
        }
    ]
}
```

Sie können diese Richtlinie den IAM-Benutzern in Ihrem Konto anfügen. Wenn ein benannter Benutzer `richard-roe` versucht, eine AWS IoT Abrechnungsgruppe aufzurufen, muss die Abrechnungsgruppe mit `0wner=richard-roe` oder gekennzeichnet werden`owner=richard-roe`. Andernfalls wird der Zugriff abgelehnt. Der Tag-Schlüssel `0wner` der Bedingung stimmt sowohl mit `0wner` als auch mit `owner` überein, da die Namen von Bedingungsschlüsseln nicht zwischen Groß- und Kleinschreibung unterscheiden. Weitere Informationen finden Sie unter IAM-JSON-Richtlinienelemente: Bedingung im IAM-Benutzerhandbuch.

Wird angezeigt AWS IoT Device Advisor-Ressourcen, die auf Tags basieren

Sie können in Ihrer identitätsbasierten Richtlinie Bedingungen für die Steuerung des Zugriffs auf AWS IoT -Device-Advisor-Ressourcen auf der Basis von Tags verwenden. Im folgenden Beispiel wird gezeigt, wie Sie eine Richtlinie erstellen können, mit der eine bestimmte Suite-Definition angezeigt werden kann. Die Berechtigung wird jedoch nur erteilt, wenn `SuiteType` im Suite-Definitions-Tag auf den Wert von `MQTT` gesetzt ist. Diese Richtlinie gewährt auch die Berechtigungen, die für die Ausführung dieser Aktion auf der Konsole erforderlich sind.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ViewSuiteDefinition",
            "Effect": "Allow",
            "Action": "iotdeviceadvisor:GetSuiteDefinition",
            "Resource": "arn:aws:iotdeviceadvisor:*:*:suitedefinition/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/SuiteType": "MQTT"}
        }
        }
    ]
}
```

AWS verwaltete Richtlinien für AWS IoT

Um Benutzern, Gruppen und Rollen Berechtigungen hinzuzufügen, ist es einfacher, AWS verwaltete Richtlinien zu verwenden, als Richtlinien selbst zu schreiben. Es erfordert Zeit und Fachwissen, um von Kunden verwaltete IAM-Richtlinien zu erstellen, die Ihrem Team nur die benötigten Berechtigungen bieten. Um schnell loszulegen, können Sie unsere AWS verwalteten Richtlinien verwenden. Diese Richtlinien decken allgemeine Anwendungsfälle ab und sind in Ihrem AWS-Konto verfügbar. Weitere Informationen zu AWS verwalteten Richtlinien finden Sie im IAM-Benutzerhandbuch unter AWS Verwaltete Richtlinien.

AWS Dienste verwalten und aktualisieren AWS verwaltete Richtlinien. Sie können die Berechtigungen in AWS verwalteten Richtlinien nicht ändern. Services fügen einer von AWS verwalteten Richtlinien gelegentlich zusätzliche Berechtigungen hinzu, um neue Features zu unterstützen. Diese Art von Update betrifft alle Identitäten (Benutzer, Gruppen und Rollen), an welche die Richtlinie angehängt ist. Services aktualisieren eine von AWS verwaltete Richtlinie am ehesten, ein neues Feature gestartet wird oder neue Vorgänge verfügbar werden. Dienste entfernen keine Berechtigungen aus einer AWS verwalteten Richtlinie, sodass durch Richtlinienaktualisierungen Ihre bestehenden Berechtigungen nicht beeinträchtigt werden.

AWS Unterstützt außerdem verwaltete Richtlinien für Jobfunktionen, die sich über mehrere Dienste erstrecken. Die ReadOnlyAccess AWS verwaltete Richtlinie bietet beispielsweise schreibgeschützten Zugriff auf alle AWS Dienste und Ressourcen. Wenn ein Service ein neues Feature startet, fügt AWS schreibgeschützte Berechtigungen für neue Vorgänge und Ressourcen hinzu. Eine Liste und Beschreibungen der Richtlinien für Auftragsfunktionen finden Sie in Verwaltete AWS -Richtlinien für Auftragsfunktionen im IAM-Leitfaden.

> **Note**
>
> AWS IoT funktioniert sowohl mit IAM-Richtlinien als auch mit AWS IoT IAM-Richtlinien. In diesem Thema werden nur IAM-Richtlinien behandelt, die eine Richtlinienaktion für API-Operationen auf der Steuer- und Datenebene definieren. Siehe auch AWS IoT Core Richtlinien.

AWS verwaltete Richtlinie: AWSIoTConfigAccess

Sie können die `AWSIoTConfigAccess`-Richtlinie an Ihre IAM-Identitäten anfügen.

Diese Richtlinie erteilt die zugehörigen Identitätsberechtigungen, die Zugriff auf alle AWS IoT -Konfigurationsoperationen gewähren. Diese Richtlinie kann sich auf Datenverarbeitung und Speicher auswirken. Informationen zu dieser Richtlinie finden Sie AWS-Managementkonsole unter AWSIoTConfigAccess.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `iot`— Rufen Sie AWS IoT Daten ab und führen Sie IoT-Konfigurationsaktionen durch.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:AcceptCertificateTransfer",
                "iot:AddThingToThingGroup",
                "iot:AssociateTargetsWithJob",
                "iot:AttachPolicy",
                "iot:AttachPrincipalPolicy",
                "iot:AttachThingPrincipal",
                "iot:CancelCertificateTransfer",
                "iot:CancelJob",
                "iot:CancelJobExecution",
                "iot:ClearDefaultAuthorizer",
                "iot:CreateAuthorizer",
                "iot:CreateCertificateFromCsr",
                "iot:CreateJob",
                "iot:CreateKeysAndCertificate",
                "iot:CreateOTAUpdate",
                "iot:CreatePolicy",
```

```
                "iot:CreatePolicyVersion",
                "iot:CreateRoleAlias",
                "iot:CreateStream",
                "iot:CreateThing",
                "iot:CreateThingGroup",
                "iot:CreateThingType",
                "iot:CreateTopicRule",
                "iot:DeleteAuthorizer",
                "iot:DeleteCACertificate",
                "iot:DeleteCertificate",
                "iot:DeleteJob",
                "iot:DeleteJobExecution",
                "iot:DeleteOTAUpdate",
                "iot:DeletePolicy",
                "iot:DeletePolicyVersion",
                "iot:DeleteRegistrationCode",
                "iot:DeleteRoleAlias",
                "iot:DeleteStream",
                "iot:DeleteThing",
                "iot:DeleteThingGroup",
                "iot:DeleteThingType",
                "iot:DeleteTopicRule",
                "iot:DeleteV2LoggingLevel",
                "iot:DeprecateThingType",
                "iot:DescribeAuthorizer",
                "iot:DescribeCACertificate",
                "iot:DescribeCertificate",
                "iot:DescribeDefaultAuthorizer",
                "iot:DescribeEndpoint",
                "iot:DescribeEventConfigurations",
                "iot:DescribeIndex",
                "iot:DescribeJob",
                "iot:DescribeJobExecution",
                "iot:DescribeRoleAlias",
                "iot:DescribeStream",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:DescribeThingRegistrationTask",
                "iot:DescribeThingType",
                "iot:DetachPolicy",
                "iot:DetachPrincipalPolicy",
                "iot:DetachThingPrincipal",
                "iot:DisableTopicRule",
                "iot:EnableTopicRule",
```

```
                "iot:GetEffectivePolicies",
                "iot:GetIndexingConfiguration",
                "iot:GetJobDocument",
                "iot:GetLoggingOptions",
                "iot:GetOTAUpdate",
                "iot:GetPolicy",
                "iot:GetPolicyVersion",
                "iot:GetRegistrationCode",
                "iot:GetTopicRule",
                "iot:GetV2LoggingOptions",
                "iot:ListAttachedPolicies",
                "iot:ListAuthorizers",
                "iot:ListCACertificates",
                "iot:ListCertificates",
                "iot:ListCertificatesByCA",
                "iot:ListIndices",
                "iot:ListJobExecutionsForJob",
                "iot:ListJobExecutionsForThing",
                "iot:ListJobs",
                "iot:ListOTAUpdates",
                "iot:ListOutgoingCertificates",
                "iot:ListPolicies",
                "iot:ListPolicyPrincipals",
                "iot:ListPolicyVersions",
                "iot:ListPrincipalPolicies",
                "iot:ListPrincipalThings",
                "iot:ListRoleAliases",
                "iot:ListStreams",
                "iot:ListTargetsForPolicy",
                "iot:ListThingGroups",
                "iot:ListThingGroupsForThing",
                "iot:ListThingPrincipals",
                "iot:ListThingRegistrationTaskReports",
                "iot:ListThingRegistrationTasks",
                "iot:ListThings",
                "iot:ListThingsInThingGroup",
                "iot:ListThingTypes",
                "iot:ListTopicRules",
                "iot:ListV2LoggingLevels",
                "iot:RegisterCACertificate",
                "iot:RegisterCertificate",
                "iot:RegisterThing",
                "iot:RejectCertificateTransfer",
                "iot:RemoveThingFromThingGroup",
```

```
                "iot:ReplaceTopicRule",
                "iot:SearchIndex",
                "iot:SetDefaultAuthorizer",
                "iot:SetDefaultPolicyVersion",
                "iot:SetLoggingOptions",
                "iot:SetV2LoggingLevel",
                "iot:SetV2LoggingOptions",
                "iot:StartThingRegistrationTask",
                "iot:StopThingRegistrationTask",
                "iot:TestAuthorization",
                "iot:TestInvokeAuthorizer",
                "iot:TransferCertificate",
                "iot:UpdateAuthorizer",
                "iot:UpdateCACertificate",
                "iot:UpdateCertificate",
                "iot:UpdateEventConfigurations",
                "iot:UpdateIndexingConfiguration",
                "iot:UpdateRoleAlias",
                "iot:UpdateStream",
                "iot:UpdateThing",
                "iot:UpdateThingGroup",
                "iot:UpdateThingGroupsForThing",
                "iot:UpdateAccountAuditConfiguration",
                "iot:DescribeAccountAuditConfiguration",
                "iot:DeleteAccountAuditConfiguration",
                "iot:StartOnDemandAuditTask",
                "iot:CancelAuditTask",
                "iot:DescribeAuditTask",
                "iot:ListAuditTasks",
                "iot:CreateScheduledAudit",
                "iot:UpdateScheduledAudit",
                "iot:DeleteScheduledAudit",
                "iot:DescribeScheduledAudit",
                "iot:ListScheduledAudits",
                "iot:ListAuditFindings",
                "iot:CreateSecurityProfile",
                "iot:DescribeSecurityProfile",
                "iot:UpdateSecurityProfile",
                "iot:DeleteSecurityProfile",
                "iot:AttachSecurityProfile",
                "iot:DetachSecurityProfile",
                "iot:ListSecurityProfiles",
                "iot:ListSecurityProfilesForTarget",
                "iot:ListTargetsForSecurityProfile",
```

```
                "iot:ListActiveViolations",
                "iot:ListViolationEvents",
                "iot:ValidateSecurityProfileBehaviors"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS verwaltete Richtlinie: AWSIoTConfigReadOnlyAccess

Sie können die `AWSIoTConfigReadOnlyAccess`-Richtlinie an Ihre IAM-Identitäten anfügen.

Diese Richtlinie erteilt die zugehörigen Identitätsberechtigungen, die schreibgeschützten Zugriff auf alle AWS IoT -Konfigurationsoperationen gewähren. Informationen zu dieser Richtlinie finden Sie AWS-Managementkonsole unter AWSIoTConfigReadOnlyAccess.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `iot`: Führen Sie schreibgeschützte Operationen von IoT-Konfigurationsaktionen durch.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:DescribeAuthorizer",
                "iot:DescribeCACertificate",
                "iot:DescribeCertificate",
                "iot:DescribeDefaultAuthorizer",
                "iot:DescribeEndpoint",
                "iot:DescribeEventConfigurations",
                "iot:DescribeIndex",
```

```
                "iot:DescribeJob",
                "iot:DescribeJobExecution",
                "iot:DescribeRoleAlias",
                "iot:DescribeStream",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:DescribeThingRegistrationTask",
                "iot:DescribeThingType",
                "iot:GetEffectivePolicies",
                "iot:GetIndexingConfiguration",
                "iot:GetJobDocument",
                "iot:GetLoggingOptions",
                "iot:GetOTAUpdate",
                "iot:GetPolicy",
                "iot:GetPolicyVersion",
                "iot:GetRegistrationCode",
                "iot:GetTopicRule",
                "iot:GetV2LoggingOptions",
                "iot:ListAttachedPolicies",
                "iot:ListAuthorizers",
                "iot:ListCACertificates",
                "iot:ListCertificates",
                "iot:ListCertificatesByCA",
                "iot:ListIndices",
                "iot:ListJobExecutionsForJob",
                "iot:ListJobExecutionsForThing",
                "iot:ListJobs",
                "iot:ListOTAUpdates",
                "iot:ListOutgoingCertificates",
                "iot:ListPolicies",
                "iot:ListPolicyPrincipals",
                "iot:ListPolicyVersions",
                "iot:ListPrincipalPolicies",
                "iot:ListPrincipalThings",
                "iot:ListRoleAliases",
                "iot:ListStreams",
                "iot:ListTargetsForPolicy",
                "iot:ListThingGroups",
                "iot:ListThingGroupsForThing",
                "iot:ListThingPrincipals",
                "iot:ListThingRegistrationTaskReports",
                "iot:ListThingRegistrationTasks",
                "iot:ListThings",
                "iot:ListThingsInThingGroup",
```

```
                "iot:ListThingTypes",
                "iot:ListTopicRules",
                "iot:ListV2LoggingLevels",
                "iot:SearchIndex",
                "iot:TestAuthorization",
                "iot:TestInvokeAuthorizer",
                "iot:DescribeAccountAuditConfiguration",
                "iot:DescribeAuditTask",
                "iot:ListAuditTasks",
                "iot:DescribeScheduledAudit",
                "iot:ListScheduledAudits",
                "iot:ListAuditFindings",
                "iot:DescribeSecurityProfile",
                "iot:ListSecurityProfiles",
                "iot:ListSecurityProfilesForTarget",
                "iot:ListTargetsForSecurityProfile",
                "iot:ListActiveViolations",
                "iot:ListViolationEvents",
                "iot:ValidateSecurityProfileBehaviors"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS verwaltete Richtlinie: AWSIoTDataAccess

Sie können die `AWSIoTDataAccess`-Richtlinie an Ihre IAM-Identitäten anfügen.

Diese Richtlinie gewährt den zugehörigen Identitätsberechtigungen, die den Zugriff auf alle AWS IoT Datenoperationen ermöglichen. Bei den Datenoperationen werden Daten über das MQTT- oder HTTP-Protokoll gesendet. Informationen zum Anzeigen dieser Richtlinie in der AWS-Managementkonsole finden Sie unter `AWSIoTDataAccess`.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `iot`— Rufen Sie AWS IoT Daten ab und gewähren Sie vollen Zugriff auf AWS IoT Messaging-Aktionen.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect",
                "iot:Publish",
                "iot:Subscribe",
                "iot:Receive",
                "iot:GetThingShadow",
                "iot:UpdateThingShadow",
                "iot:DeleteThingShadow",
                "iot:ListNamedShadowsForThing"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS verwaltete Richtlinie: AWSIoTFullAccess

Sie können die `AWSIoTFullAccess`-Richtlinie an Ihre IAM-Identitäten anfügen.

Diese Richtlinie erteilt die zugehörigen Identitätsberechtigungen, die Zugriff auf alle AWS IoT -Konfigurations- und Messaging-Operationen gewähren. Informationen zu dieser Richtlinie finden Sie AWS-Managementkonsole unter `AWSIoTFullAccess`.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `iot`— Rufen Sie AWS IoT Daten ab und gewähren Sie vollen Zugriff auf AWS IoT Konfiguration und Nachrichtenaktionen.
- `iotjobsdata`— Rufen Sie AWS IoT Jobs-Daten ab und ermöglichen Sie den vollen Zugriff auf die API-Operationen auf der AWS IoT Jobs-Datenebene.

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

AWS verwaltete Richtlinie: AWSIoTLogging

Sie können die `AWSIoTLogging`-Richtlinie an Ihre IAM-Identitäten anfügen.

Diese Richtlinie gewährt die zugehörigen Identitätsberechtigungen, die den Zugriff auf die Erstellung von Amazon CloudWatch Logs-Gruppen und das Streamen von Protokollen an die Gruppen ermöglichen. Diese Richtlinie ist mit Ihrer CloudWatch Logging-Rolle verknüpft. Informationen zu dieser Richtlinie in der AWS-Managementkonsole finden Sie unter `AWSIoTLogging`.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `logs`— CloudWatch Protokolle abrufen. Ermöglicht auch die Erstellung von CloudWatch Protokollgruppen und das Streamen von Protokollen an die Gruppen.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:PutMetricFilter",
                "logs:PutRetentionPolicy",
                "logs:GetLogEvents",
                "logs:DeleteLogStream"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

AWS verwaltete Richtlinie: AWSIoTOTAUpdate

Sie können die `AWSIoTOTAUpdate`-Richtlinie an Ihre IAM-Identitäten anfügen.

Diese Richtlinie gewährt die zugehörigen Identitätsberechtigungen, die den Zugriff auf das Erstellen von AWS IoT Jobs und AWS IoT Codesignatur-Jobs und das Beschreiben von AWS Codesigner-Jobs ermöglichen. Informationen zu dieser Richtlinie finden Sie AWS-Managementkonsole unter`AWSIoTOTAUpdate`.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `iot`— AWS IoT Jobs und Codesignatur-Jobs erstellen.
- `signer`— Führen Sie die Erstellung von AWS Codesigner-Jobs durch.

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iot:CreateJob",
            "signer:DescribeSigningJob"
        ],
        "Resource": "*"
    }
}
```

AWS verwaltete Richtlinie: AWSIoTRuleActions

Sie können die `AWSIoTRuleActions`-Richtlinie an Ihre IAM-Identitäten anfügen.

Diese Richtlinie gewährt den zugehörigen Identitätsberechtigungen, die den Zugriff auf alle AWS-Service in AWS IoT der Regel unterstützten Aktionen ermöglichen. Informationen zu dieser Richtlinie in der AWS-Managementkonsole finden Sie unter `AWSIoTRuleActions`.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `iot`: Führen Sie Aktionen zum Veröffentlichen von Regelaktionsmeldungen durch.
- `dynamodb`: Fügen Sie eine Nachricht in eine DynamoDB-Tabelle ein oder teilen Sie eine Nachricht in mehrere Spalten einer DynamoDB-Tabelle auf.
- `s3`: Speichern Sie ein Objekt zu einem Amazon-S3-Bucket.
- `kinesis`: Senden Sie eine Nachricht an ein Amazon-Kinesis-Streamingobjekt.
- `firehose`- Fügt einen Datensatz in ein Firehose-Stream-Objekt ein.
- `cloudwatch`- Ändern Sie den CloudWatch Alarmstatus oder senden Sie Nachrichtendaten an die CloudWatch Metrik.
- `sns`: Führen Sie die Operation durch, um eine Benachrichtigung mit Amazon SNS zu veröffentlichen. Dieser Vorgang ist auf SNS-Themen beschränkt AWS IoT .

- `sqs`: Fügen Sie eine Nachricht ein, die der SQS-Warteschlange hinzugefügt werden soll.
- `es`- Senden Sie eine Nachricht an den OpenSearch Servicedienst.

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "dynamodb:PutItem",
            "kinesis:PutRecord",
            "iot:Publish",
            "s3:PutObject",
            "sns:Publish",
            "sqs:SendMessage*",
            "cloudwatch:SetAlarmState",
            "cloudwatch:PutMetricData",
            "es:ESHttpPut",
            "firehose:PutRecord"
        ],
        "Resource": "*"
    }
}
```

AWS verwaltete Richtlinie: AWSIoTThingsRegistration

Sie können die `AWSIoTThingsRegistration`-Richtlinie an Ihre IAM-Identitäten anfügen.

Diese Richtlinie gewährt die zugehörigen Identitätsberechtigungen, die die Massenregistrierung von Objekten mithilfe der `StartThingRegistrationTask`-API ermöglichen. Diese Richtlinie kann sich auf Datenverarbeitung und Speicher auswirken. Informationen zu dieser Richtlinie finden Sie AWS-Managementkonsole unter `AWSIoTThingsRegistration`.

Details zu Berechtigungen

Diese Richtlinie umfasst die folgenden Berechtigungen.

- `iot`: Führen Sie bei der Massenregistrierung Aktionen zum Erstellen von Objekten und zum Anhängen von Richtlinien und Zertifikaten durch.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:AddThingToThingGroup",
                "iot:AttachPolicy",
                "iot:AttachPrincipalPolicy",
                "iot:AttachThingPrincipal",
                "iot:CreateCertificateFromCsr",
                "iot:CreatePolicy",
                "iot:CreateThing",
                "iot:DescribeCertificate",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:DescribeThingType",
                "iot:DetachPolicy",
                "iot:DetachThingPrincipal",
                "iot:GetPolicy",
                "iot:ListAttachedPolicies",
                "iot:ListPolicyPrincipals",
                "iot:ListPrincipalPolicies",
                "iot:ListPrincipalThings",
                "iot:ListTargetsForPolicy",
                "iot:ListThingGroupsForThing",
                "iot:ListThingPrincipals",
                "iot:RegisterCertificate",
                "iot:RegisterThing",
                "iot:RemoveThingFromThingGroup",
                "iot:UpdateCertificate",
                "iot:UpdateThing",
                "iot:UpdateThingGroupsForThing",
                "iot:AddThingToBillingGroup",
                "iot:DescribeBillingGroup",
                "iot:RemoveThingFromBillingGroup"
            ],
```

```
            "Resource": [
                "*"
            ]
        }
    ]
}
```

AWS IoT Aktualisierungen zu AWS Verwaltete Richtlinien

Hier finden Sie Informationen zu Aktualisierungen der AWS verwalteten Richtlinien AWS IoT seit Beginn der Nachverfolgung dieser Änderungen durch diesen Dienst. Abonnieren Sie den RSS-Feed auf der Seite AWS IoT Dokumentenverlauf, um automatische Benachrichtigungen über Änderungen an dieser Seite zu erhalten.

Änderungen	Beschreibung	Datum
AWSIoTFullAccess – Aktualisierung auf eine bestehende Richtlinie	AWS IoT Es wurden neue Berechtigungen hinzugefügt, um Benutzern den Zugriff auf API-Operationen der AWS IoT Jobs-Datenebene mithilfe des HTTP-Protokolls zu ermöglichen. Ein neues IAM-Richtlinienpräfix,`iotjobsdata:` , bietet Ihnen eine detailliertere Zugriffskontrolle für den Zugriff auf Endpunkte der AWS IoT Jobs-Datenebene. Für API-Operationen auf der Steuerebene verwenden Sie weiterhin das Präfix `iot:`. Weitere Informationen finden Sie unter AWS IoT Core	11. Mai 2022

Änderungen	Beschreibung	Datum
	Richtlinien für das HTTPS-Protokoll.	
AWS IoT hat begonnen, Änderungen zu verfolgen	AWS IoT hat begonnen, Änderungen für die AWS verwalteten Richtlinien zu verfolgen.	11. Mai 2022

Fehlerbehebung AWS IoT Identität und Zugriff

Verwenden Sie die folgenden Informationen, um häufig auftretende Probleme zu diagnostizieren und zu beheben, die bei der Arbeit mit AWS IoT und IAM auftreten können.

Themen

- Ich bin nicht berechtigt, eine Aktion durchzuführen in AWS IoT
- Ich bin nicht berechtigt, iam auszuführen: PassRole
- Ich möchte Personen außerhalb meiner Umgebung zulassen AWS-Konto um auf meine zuzugreifen AWS IoT Ressourcen

Ich bin nicht berechtigt, eine Aktion durchzuführen in AWS IoT

Wenn Sie eine Fehlermeldung erhalten, dass Sie nicht zur Durchführung einer Aktion berechtigt sind, müssen Ihre Richtlinien aktualisiert werden, damit Sie die Aktion durchführen können.

Der folgende Beispielfehler tritt auf, wenn der/die IAM-Benutzer:in `mateojackson` versucht, über die Konsole Details zu einer Objektressource anzuzeigen, jedoch nicht über `iot:DescribeThing`-Berechtigungen verfügt.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 iot:DescribeThing on resource: MyIoTThing
```

In diesem Fall muss die Richtlinie für den/die Benutzer:in `mateojackson` aktualisiert werden, damit er/sie mit der `iot:DescribeThing`-Aktion auf die Objektressource zugreifen kann.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Verwenden von AWS IoT Device Advisor

Wenn Sie AWS IoT Device Advisor verwenden, tritt der folgende Beispielfehler auf, wenn der Benutzer `mateojackson` versucht, die Konsole zu verwenden, um Details zu einer Suite-Definition anzuzeigen, aber nicht über die `iotdeviceadvisor:`*`GetSuiteDefinition`* entsprechenden Berechtigungen verfügt.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 iotdeviceadvisor:GetSuiteDefinition on resource: MySuiteDefinition
```

In diesem Fall muss die Richtlinie für den Benutzer `mateojackson` aktualisiert werden, damit er mit der `iotdeviceadvisor:`*`GetSuiteDefinition`*-Aktion auf die *`MySuiteDefinition`*-Ressource zugreifen kann.

Ich bin nicht berechtigt, iam auszuführen: PassRole

Wenn Sie die Fehlermeldung erhalten, dass Sie nicht zum Durchführen der `iam:PassRole`-Aktion autorisiert sind, müssen Ihre Richtlinien aktualisiert werden, um eine Rolle an AWS IoTübergeben zu können.

Einige AWS-Services ermöglichen es Ihnen, eine bestehende Rolle an diesen Dienst zu übergeben, anstatt eine neue Servicerolle oder eine dienstverknüpfte Rolle zu erstellen. Hierzu benötigen Sie Berechtigungen für die Übergabe der Rolle an den Dienst.

Der folgende Beispielfehler tritt auf, wenn ein IAM-Benutzer mit dem Namen `marymajor` versucht, die Konsole zu verwenden, um eine Aktion in AWS IoT auszuführen. Die Aktion erfordert jedoch, dass der Service über Berechtigungen verfügt, die durch eine Servicerolle gewährt werden. Mary besitzt keine Berechtigungen für die Übergabe der Rolle an den Dienst.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In diesem Fall müssen die Richtlinien von Mary aktualisiert werden, um die Aktion `iam:PassRole` ausführen zu können.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich möchte Personen außerhalb meiner Umgebung zulassen AWS-Konto um auf meine zuzugreifen AWS IoT Ressourcen

Sie können eine Rolle erstellen, mit der Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation auf Ihre Ressourcen zugreifen können. Sie können festlegen, wem die Übernahme der Rolle anvertraut wird. Im Fall von Diensten, die ressourcenbasierte Richtlinien oder Zugriffskontrolllisten (Access Control Lists, ACLs) verwenden, können Sie diese Richtlinien verwenden, um Personen Zugriff auf Ihre Ressourcen zu gewähren.

Weitere Informationen dazu finden Sie hier:

- Informationen darüber, ob diese Funktionen AWS IoT unterstützt werden, finden Sie unterWie AWS IoT funktioniert mit IAM.
- Informationen dazu, wie Sie Zugriff auf Ihre Ressourcen gewähren können, AWS-Konten die Ihnen gehören, finden Sie im IAM-Benutzerhandbuch unter Gewähren des Zugriffs auf einen IAM-Benutzer in einem anderen AWS-Konto , den Sie besitzen.
- Informationen dazu, wie Sie Dritten Zugriff auf Ihre Ressourcen gewähren können AWS-Konten, finden Sie AWS-Konten im IAM-Benutzerhandbuch unter Gewähren des Zugriffs für Dritte.
- Informationen dazu, wie Sie über einen Identitätsverbund Zugriff gewähren, finden Sie unter Gewähren von Zugriff für extern authentifizierte Benutzer (Identitätsverbund) im IAM-Benutzerhandbuch.
- Informationen zum Unterschied zwischen der Verwendung von Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Protokollieren und Überwachen

Die Überwachung ist ein wichtiger Bestandteil der Aufrechterhaltung der Zuverlässigkeit, Verfügbarkeit und Leistung Ihrer AWS IoT AWS Lösungen. Sie sollten Überwachungsdaten aus allen Teilen Ihrer AWS Lösung sammeln, damit Sie einen etwaigen Ausfall an mehreren Stellen leichter debuggen können. Informationen zu Protokollierungs- und Überwachungsverfahren finden Sie unter Überwachen AWS IoT

Überwachungstools

AWS stellt Tools bereit, die Sie zur Überwachung verwenden können AWS IoT. Sie können einige dieser Tools für die Überwachung konfigurieren. Einige der Tools erfordern manuelle Eingriffe. Wir empfehlen, dass Sie die Überwachungsaufgaben möglichst automatisieren.

Automatisierte Überwachungstools

Sie können die folgenden automatisierten Überwachungstools verwenden, um zu beobachten AWS IoT und zu melden, wenn etwas nicht stimmt:

- Amazon CloudWatch Alarms — Überwachen Sie eine einzelne Metrik über einen von Ihnen angegebenen Zeitraum und führen Sie eine oder mehrere Aktionen aus, die auf dem Wert der Metrik im Verhältnis zu einem bestimmten Schwellenwert über mehrere Zeiträume basieren. Die Aktion ist eine Benachrichtigung, die an ein Amazon Simple Notification Service (Amazon SNS) -Thema oder eine Amazon EC2 Auto Scaling Scaling-Richtlinie gesendet wird. CloudWatch Alarme lösen keine Aktionen aus, nur weil sie sich in einem bestimmten Status befinden. Der Status muss sich geändert haben und für eine festgelegte Anzahl an Zeiträumen aufrechterhalten worden sein. Weitere Informationen finden Sie unter Überwachen AWS IoT Alarme und Metriken mithilfe von Amazon CloudWatch.
- Amazon CloudWatch Logs — Überwachen, speichern und greifen Sie auf Ihre Protokolldateien aus AWS CloudTrail oder anderen Quellen zu. Mit Amazon CloudWatch Logs können Sie auch wichtige Schritte, die AWS IoT Device Advisor-Testfälle ausführen, generierte Ereignisse und MQTT-Nachrichten verfolgen, die von Ihren Geräten oder AWS IoT Core während der Testausführung gesendet wurden. Mit diesen Protokollen können Sie Geräte debuggen und Korrekturmaßnahmen auf Ihren Geräten ergreifen. Weitere Informationen finden Sie unter Überwachen AWS IoT mithilfe von Protokollen CloudWatch Weitere Informationen zur Verwendung von Amazon CloudWatch finden Sie unter Überwachung von Protokolldateien im CloudWatch Amazon-Benutzerhandbuch.
- Amazon CloudWatch Events — Ordnen Sie Ereignisse zu und leiten Sie sie an eine oder mehrere Zielfunktionen oder Streams weiter, um Änderungen vorzunehmen, Statusinformationen zu erfassen und Korrekturmaßnahmen zu ergreifen. Weitere Informationen finden Sie unter Was ist Amazon CloudWatch Events im CloudWatch Amazon-Benutzerhandbuch.
- AWS CloudTrail Protokollüberwachung — Teilen Sie Protokolldateien zwischen Konten, überwachen CloudTrail Sie Protokolldateien in Echtzeit, indem Sie sie an CloudWatch Logs senden, schreiben Sie Protokollverarbeitungsanwendungen in Java und überprüfen Sie, ob sich Ihre Protokolldateien nach der Lieferung von nicht geändert haben CloudTrail. Weitere

Informationen finden Sie unter Protokollierung AWS IoT API-Aufrufe mit AWS CloudTrail und auch Arbeiten mit CloudTrail Protokolldateien im AWS CloudTrail Benutzerhandbuch.

Manuelle Überwachungstools

Ein weiterer wichtiger Teil der Überwachung ist AWS IoT die manuelle Überwachung der Elemente, die von den CloudWatch Alarmen nicht abgedeckt werden. Die Dashboards AWS IoT CloudWatch, und andere AWS Servicekonsolen-Dashboards bieten auf einen Blick einen Überblick über den Zustand Ihrer Umgebung. AWS Wir empfehlen, dass Sie auch die Protokolldateien unter überprüfen. AWS IoT

- AWS IoT Das Dashboard zeigt:
  - CA-Zertifikate
  - Zertifikate
  - Richtlinien
  - Regeln
  - Objekte
- CloudWatch Die Startseite zeigt:
  - Aktuelle Alarme und Status.
  - Diagramme mit Alarmen und Ressourcen.
  - Servicestatus.

  Sie können verwenden CloudWatch , um Folgendes zu tun:
  - Erstellen von benutzerdefinierten Dashboards zur Überwachung des gewünschten Services.
  - Aufzeichnen von Metrikdaten, um Probleme zu beheben und Trends zu erkennen.
  - Suchen und durchsuchen Sie alle Ihre AWS Ressourcenmetriken.
  - Erstellen und Bearbeiten von Alarmen, um über Probleme benachrichtigt zu werden.

Überprüfung der Einhaltung von Vorschriften für AWS IoT Core

Unser neues AWS Anmeldeerlebnis ist nicht für regulierte Workloads konzipiert. Wenn Sie unser neues AWS Anmeldeformular nutzen, es aber AWS für regulierte Workloads nutzen möchten, können Sie sich für AWS (erweiterte) Funktionen registrieren oder erweiterte Funktionen für Ihre Umgebung aktivieren. AWS

Informationen darüber, ob AWS-Service ein in den Geltungsbereich bestimmter Compliance-Programme fällt, finden Sie unter AWS-Services Umfang nach Compliance-Programm unter Umfang nach Compliance-Programm AWS-Services das Compliance-Programm aus, an dem Sie interessiert sind. Allgemeine Informationen finden Sie unter AWS Compliance-Programme AWS .

Sie können Prüfberichte von Drittanbietern unter herunterladen AWS Artifact. Weitere Informationen finden Sie unter Berichte herunterladen unter .

Ihre Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services hängt von der Vertraulichkeit Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. Weitere Informationen zu Ihrer Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services finden Sie in der AWS Sicherheitsdokumentation.

Resilienz in AWS IoT-Kern

Die AWS globale Infrastruktur basiert auf AWS-Region s und Availability Zones. AWS-Region s bieten mehrere physisch getrennte und isolierte Availability Zones, die über Netzwerke mit niedriger Latenz, hohem Durchsatz und hoher Redundanz miteinander verbunden sind. Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Availability Zones ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser hoch verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Weitere Informationen zu AWS-Region s und Availability Zones finden Sie unter AWS Globale Infrastruktur.

AWS IoT Core speichert Informationen zu Ihren Geräten in der Geräteregistrierung. Der Service speichert auch CA-Zertifikate, Gerätezertifikate und Geräteschattendaten. Diese Daten werden bei Hardware- oder Netzwerkausfällen automatisch in Availability Zones, nicht jedoch in Regionen repliziert.

AWS IoT Core veröffentlicht MQTT-Ereignisse, wenn die Geräteregistrierung aktualisiert wird. Sie können diese Nachrichten verwenden, um Ihre Registrierungsdaten zu sichern und sie irgendwo zu speichern (z. B. in einer DynamoDB-Tabelle). Sie sind dafür verantwortlich, Zertifikate zu speichern, AWS IoT Core die Sie für Sie oder die Sie selbst erstellen. Device Shadow speichert Statusdaten zu Ihren Geräten und kann erneut gesendet werden, wenn ein Gerät wieder online ist. AWS IoT Device Advisor speichert Informationen über Ihre Testsuite-Konfiguration. Diese Daten werden bei Hardware- oder Netzwerkausfällen automatisch repliziert.

AWS IoT Core Ressourcen werden Region-specific und werden nicht repliziert, AWS-Regionen es sei denn, Sie tun dies ausdrücklich.

Weitere Informationen zu bewährten Methoden für die Sicherheit finden Sie unter Bewährte Sicherheitsmethoden in AWS IoT Core.

Verwenden AWS IoT Core mit VPC-Endpunkten mit Schnittstelle

Mit AWS IoT Core können Sie IoT-Steuerebenen-Endpunkte und IoT-Datenendpunkte in Ihrer Virtual Private Cloud (VPC) mithilfe von Schnittstellen-VPC-Endpunkten erstellen. Schnittstellen-VPC-Endpoints basieren auf einer AWS Technologie AWS PrivateLink, mit der Sie mithilfe AWS von privaten IP-Adressen auf Dienste zugreifen können, auf denen sie ausgeführt werden. Weitere Informationen finden Sie unter Amazon Virtual Private Cloud.

Informationen zum Verbinden von Geräten vor Ort in Remote-Netzwerken, z. B. einem Unternehmensnetzwerk, mit Ihrer Amazon VPC, finden Sie in den Optionen, die in der Network-to-Amazon VPC-Konnektivitätsmatrix aufgeführt sind.

Inhalt

- VPC-Endpoints erstellen für AWS IoT Core Steuerebene
- VPC-Endpoints erstellen für AWS IoT Core Datenebene
- VPC-Endpoints erstellen für AWS IoT Core Anbieter von Anmeldeinformationen
- Erstellen eines Amazon-VPC-Schnittstellenendpunkts
- Konfigurieren Sie eine private gehostete Zone
- Steuern des Zugriffs auf AWS IoT Core über VPC-Endpunkte
- Einschränkungen
- Skalierung von VPC-Endpunkten mit AWS IoT Core
- Verwenden von benutzerdefinierten Domains mit VPC-Endpunkten
- Verfügbarkeit von VPC-Endpunkten für AWS IoT Core
- Verwenden AWS IoT Device Management sicheres Tunneling mit VPC-Endpunkten mit Schnittstellen

VPC-Endpoints erstellen für AWS IoT Core Steuerebene

Sie können einen VPC-Endpunkt für die AWS IoT Core Control Plane API erstellen, um Ihre Geräte mit AWS IoT Diensten und anderen AWS Diensten zu verbinden. Um mit VPC-Endpunkten zu

beginnen, erstellen Sie einen VPC-Schnittstellen-Endpunkt und wählen Sie ihn AWS IoT Core als Dienst aus. AWS Wenn Sie die CLI verwenden, rufen Sie zunächst describe-vpc-endpoint-services auf, um sicherzustellen, dass Sie eine Availability Zone auswählen, in der in AWS IoT Core Ihrem Bereich vorhanden ist. AWS-Region In us-east-1 sähe dieser Befehl zum Beispiel wie folgt aus:

```
aws ec2 describe-vpc-endpoint-services --service-name com.amazonaws.us-east-1.iot.api
```

Sehen Sie sich die detaillierten Anweisungen unten an, um einen Amazon VPC-Schnittstellenendpunkt für die AWS IoT Core Kontrollebene zu erstellen.

VPC-Endpoints erstellen für AWS IoT Core Datenebene

Sie können einen VPC-Endpunkt für die AWS IoT Core Datenebene-API erstellen, um Ihre Geräte mit AWS IoT Diensten und anderen AWS Diensten zu verbinden. Um mit VPC-Endpunkten zu beginnen, erstellen Sie einen VPC-Schnittstellen-Endpunkt und wählen Sie ihn AWS IoT Core als Dienst aus. AWS Wenn Sie die CLI verwenden, rufen Sie zunächst describe-vpc-endpoint-services auf, um sicherzustellen, dass Sie eine Availability Zone auswählen, in der in AWS IoT Core Ihrem Bereich vorhanden ist. AWS-Region In us-east-1 sähe dieser Befehl zum Beispiel wie folgt aus:

```
aws ec2 describe-vpc-endpoint-services --service-name com.amazonaws.us-east-1.iot.data
```

> **Note**
>
> Die VPC-Funktion zum automatischen Erstellen eines DNS-Eintrags ist deaktiviert. Erstellen Sie manuell einen privaten DNS-Eintrag, um eine Verbindung zu diesen Endpunkten herzustellen. Weitere Informationen über DNA-Einträge von privaten VPC finden Sie unter Private DNS für Schnittstellenendpunkte. Weitere Informationen zu AWS IoT Core VPC-Einschränkungen finden Sie unterEinschränkungen.

So verbinden Sie MQTT-Clients mit den VPC-Endpunktschnittstellen:

- Sie müssen manuell DNS-Einträge in einer privat gehosteten Zone erstellen, die an Ihre VPC angehängt ist. Informationen zu den ersten Schritten finden Sie unter Erstellen einer privat gehosteten Zone.

- Erstellen Sie in Ihrer privaten gehosteten Zone einen Alias-Datensatz für jede IP-Adresse der elastischen Netzwerkschnittstelle für den VPC-Endpunkt. Wenn Sie mehrere

Netzwerkschnittstellen-IPs für mehrere VPC-Endpunkte haben, erstellen Sie gewichtete DNS-Einträge mit gleicher Gewichtung für alle gewichteten Datensätze. Diese IP-Adressen sind im DescribeNetworkInterfacesAPI-Aufruf verfügbar, wenn sie nach der VPC-Endpunkt-ID im Beschreibungsfeld gefiltert werden.

Sehen Sie sich die detaillierten Anweisungen unten an, um einen Amazon VPC-Schnittstellenendpunkt zu erstellen und eine private gehostete Zone für die AWS IoT Core Datenebene zu konfigurieren.

VPC-Endpoints erstellen für AWS IoT Core Anbieter von Anmeldeinformationen

Sie können einen VPC-Endpunkt für den AWS IoT Core Anmeldeinformationsanbieter erstellen, um Geräte mithilfe der auf Client-Zertifikaten basierenden Authentifizierung zu verbinden und temporäre AWS Anmeldeinformationen im AWS Signature Version 4-Format abzurufen. Um mit VPC-Endpunkten für AWS IoT Core Credential Provider zu beginnen, führen Sie den CLI-Befehl create-vpc-endpoint aus, um einen VPC-Schnittstellen-Endpunkt zu erstellen, und wählen Sie Credential Provider als Dienst aus. AWS IoT Core AWS Um sicherzustellen, dass Sie eine Availability Zone auswählen, in der sich Ihre Availability Zone AWS IoT Core befindet, führen Sie zunächst den Befehl describe-vpc-endpoint-services aus. AWS-Region In us-east-1 sähe dieser Befehl zum Beispiel wie folgt aus:

```
aws ec2 describe-vpc-endpoint-services --service-name com.amazonaws.us-
east-1.iot.credentials
```

> **Note**
>
> Die VPC-Funktion zum automatischen Erstellen eines DNS-Eintrags ist deaktiviert. Erstellen Sie manuell einen privaten DNS-Eintrag, um eine Verbindung zu diesen Endpunkten herzustellen. Weitere Informationen über DNA-Einträge von privaten VPC finden Sie unter Private DNS für Schnittstellenendpunkte. Weitere Informationen zu AWS IoT Core VPC-Einschränkungen finden Sie unterEinschränkungen.

So verbinden Sie HTTP-Clients mit den VPC-Endpunktschnittstellen:

- Sie müssen manuell DNS-Einträge in einer privat gehosteten Zone erstellen, die an Ihre VPC angehängt ist. Informationen zu den ersten Schritten finden Sie unter Eine private gehostete Zone erstellen.

- Erstellen Sie in Ihrer privaten gehosteten Zone einen Alias-Datensatz für jede IP-Adresse der elastischen Netzwerkschnittstelle für den VPC-Endpunkt. Wenn Sie mehrere Netzwerkschnittstellen-IPs für mehrere VPC-Endpunkte haben, erstellen Sie gewichtete DNS-Einträge mit gleicher Gewichtung für alle gewichteten Datensätze. Diese IP-Adressen sind im DescribeNetworkInterfacesAPI-Aufruf verfügbar, wenn sie nach der VPC-Endpunkt-ID im Beschreibungsfeld gefiltert werden.

Sehen Sie sich die detaillierten Anweisungen unten an, um einen Amazon VPC-Schnittstellenendpunkt zu erstellen und eine private gehostete Zone für den AWS IoT Core Anmeldeinformationsanbieter zu konfigurieren.

Erstellen eines Amazon-VPC-Schnittstellenendpunkts

Sie können einen VPC-Schnittstellen-Endpunkt erstellen, um eine Verbindung zu AWS Diensten herzustellen, von AWS PrivateLink denen unterstützt wird. Gehen Sie wie folgt vor, um einen VPC-Schnittstellen-Endpunkt zu erstellen, der eine Verbindung zur AWS IoT Core Datenebene oder zum AWS IoT Core Anmeldeinformationsanbieter herstellt. Weitere Informationen finden Sie unter Zugreifen auf einen AWS Dienst über einen Schnittstellen-VPC-Endpunkt.

> **Note**
>
> Die Prozesse zum Erstellen eines Amazon VPC-Schnittstellenendpunkts für AWS IoT Core Datenebene und AWS IoT Core Anmeldeinformationsanbieter sind ähnlich, aber Sie müssen endpunktspezifische Änderungen vornehmen, damit die Verbindung funktioniert.

So erstellen Sie einen Schnittstellen-VPC-Endpunkt mit der VPC-Endpunkte-Konsole

1. Navigieren Sie zur VPC-Endpunkte-Konsole, wählen Sie im linken Menü unter Virtual Private Cloud die Option Endpunkte und dann Endpunkt erstellen aus.
2. Geben Sie auf der Seite Endpunkt erstellen die folgenden Informationen an:
   - Wählen Sie AWS-Service s als Servicekategorie aus.

- Suchen Sie nach dem Servicenamen, indem Sie das Schlüsselwort `iot` eingeben. Wählen Sie in der Liste der angezeigten `iot`-Services den Endpunkt aus.

  Wenn Sie einen VPC-Endpunkt für die AWS IoT Core Kontrollebene erstellen, wählen Sie den API-Endpunkt der AWS IoT Core Kontrollebene für Ihren AWS-Region. Der Endpunkt wird das Format `com.amazonaws.`*`region`*`.iot.api` haben.

  Wenn Sie einen VPC-Endpunkt für die AWS IoT Core Datenebene erstellen, wählen Sie den AWS IoT Core Datenebenen-API-Endpunkt für Ihre Region aus. Der Endpunkt wird das Format `com.amazonaws.`*`region`*`.iot.data` haben.

  Wenn Sie einen VPC-Endpunkt für den AWS IoT Core Credential Provider erstellen, wählen Sie den AWS IoT Core Credential Provider-Endpunkt für Ihre Region aus. Der Endpunkt wird das Format `com.amazonaws.`*`region`*`.iot.credentials` haben.

  Wenn Sie einen VPC-Endpunkt für FIPS-Regionen (Federal Information Processing Standard) erstellen, wählen Sie den FIPS-API-Endpunkt für Ihre. AWS-Region Der Endpunkt wird das Format `com.amazonaws.`*`region`*`.iot-fips.api` haben. Dies gilt nur für die Steuerungsebene.

  > **Note**
  >
  > Der Dienstname für die AWS IoT Core Datenebene in der Region China wird das folgende Format haben`cn.com.amazonaws.`*`region`*`.iot.data`. Der Dienstname für die AWS IoT Core Steuerebene in der Region China wird das folgende Format haben`com.amazonaws.`*`region`*`.iot.api`.

- Wählen Sie für VPC und Subnetze die VPC aus, in der Sie den Endpunkt erstellen möchten, und die Availability Zones (AZs), in denen Sie das Endpunktnetzwerk einrichten möchten.
- Stellen Sie sicher, dass für „DNS-Name aktivieren“ nicht die Option Für diesen Endpunkt aktivieren für AWS IoT Core Datenebene und AWS IoT Core Anmeldeinformationsanbieter ausgewählt ist. Weder die AWS IoT Core Datenebene noch der AWS IoT Core Anmeldeinformationsanbieter unterstützen bisher private DNS-Namen.

  Für die AWS IoT Core Steuerungsebene ist standardmäßig die Option DNS-Name aktivieren ausgewählt. Dadurch wird sichergestellt, dass alle Anfragen an die öffentlichen Endpunkte der AWS IoT Core Kontrollebene stattdessen über die VPC-Endpunkte weitergeleitet werden. Wenn dies aktiviert ist, müssen Sie keine private gehostete Zone konfigurieren.

- Wählen Sie für Sicherheitsgruppe die Sicherheitsgruppen aus, die Sie den Endpunktnetzwerkschnittstellen zuordnen möchten.
- Optional können Sie Tags hinzufügen oder entfernen. Tags sind Name-Wert-Paare, die Sie verwenden, um sie Ihrem Endpunkt zuzuordnen.

3. Wählen Sie VPC-Endpunkt erstellen, um den Schnittstellenendpunkt zu erstellen.

Nachdem Sie den AWS PrivateLink Endpunkt erstellt haben, sehen Sie auf der Registerkarte „Details“ Ihres Endpunkts eine Liste mit DNS-Namen. Sie können einen dieser DNS-Namen verwenden, die Sie in diesem Abschnitt erstellt haben, um Ihre private gehostete Zone zu konfigurieren. Wenn Sie die AWS IoT Core Steuerungsebene verwenden, müssen Sie keine private gehostete Zone konfigurieren.

Konfigurieren Sie eine private gehostete Zone

> **Note**
>
> Wenn Sie die AWS IoT Core Steuerungsebene verwenden und die Option DNS-Namen aktivieren ausgewählt haben, müssen Sie keine private gehostete Zone konfigurieren. Wenn Sie es deaktivieren, müssen Sie dieses Verfahren befolgen, um eine private gehostete Zone zu konfigurieren.

Sie können einen dieser DNS-Namen verwenden, die Sie im vorigen Abschnitt erstellt haben, um Ihre private gehostete Zone zu konfigurieren.

Für die AWS IoT Core Datenebene

Der DNS-Name muss der Name Ihrer Domainkonfiguration oder Ihr `IoT:Data-ATS`-Endpunkt sein. Ein mögliches Beispiel für einen DNS-Namen ist: *`xxx`*`-ats.data.iot.`*`region`*`.amazonaws.com`.

Für den AWS IoT Core Anbieter von Anmeldeinformationen

Der DNS-Name muss Ihr `iot:CredentialProvider`-Endpunkt sein. Ein möglicher DNS-Name ist: *`xxxx`*`.credentials.iot.`*`region`*`.amazonaws.com`.

Für die AWS IoT Core Kontrollebene

Der DNS-Name muss Ihr Endpunkt auf der AWS IoT Core Kontrollebene sein. Ein Beispiel für einen DNS-Namen für die AWS IoT Core Kontrollebene ist `xxxx.api.iot.region.amazonaws.com`.

> **Note**
>
> Die Verfahren zur Konfiguration der privaten gehosteten Zone für die AWS IoT Core Datenebene und den AWS IoT Core Anmeldeinformationsanbieter sind ähnlich, aber Sie müssen endpunktspezifische Änderungen vornehmen, damit die Verbindung funktioniert.

Erstellen einer privaten gehosteten Zone

So erstellen Sie eine privat gehostete Zone mit der Route-53-Konsole

1. Navigieren Sie zur Konsole Route 53 Gehostete Zonen und wählen Sie Gehostete Zone erstellen.
2. Geben Sie auf der Seite Gehostete Zone erstellen die folgenden Informationen an.
   - Geben Sie Domainname die Endpunktadresse für Ihren `iot:Data-ATS` oder `iot:CredentialProvider`-Endpunkt ein. Der folgende AWS -CLI-Befehl zeigt, wie der Endpunkt über ein öffentliches Netzwerk abgerufen wird: `aws iot describe-endpoint --endpoint-type iot:Data-ATS` oder `aws iot describe-endpoint --endpoint-type iot:CredentialProvider`.

     > **Note**
     >
     > Wenn Sie benutzerdefinierte Domains verwenden, finden Sie weitere Informationen unter Verwenden von benutzerdefinierten Domains mit VPC-Endpunkten. Benutzerdefinierte Domänen werden für den AWS IoT Core Anmeldeinformationsanbieter nicht unterstützt.

   - Wählen Sie in der Liste Typ die Option Privat gehostete Zone.
   - Optional können Sie Tags hinzufügen oder entfernen, um sie Ihrer gehosteten Zone zuzuordnen.
3. Wählen Sie Gehostete Zone erstellen, um Ihre private gehostete Zone zu erstellen.

Weitere Informationen finden Sie unter Erstellen einer privat gehosteten Zone.

Erstellen eines Datensatzes

Nachdem Sie eine private gehostete Zone eingerichtet haben, können Sie einen Eintrag erstellen, der dem DNS mitteilt, wie der Datenverkehr zu dieser Domain weitergeleitet werden soll.

So erstellen Sie einen Datensatz

1. Klicken Sie in der angezeigten Liste der gehosteten Zonen auf die privat gehostete Zone aus, die Sie zuvor erstellt haben, und wählen Sie Datensatz erstellen.
2. Erstellen Sie den Datensatz mithilfe des Assistenten. Wenn Ihnen in der Konsole die Methode Schnelle Erstellung angezeigt wird, wählen Sie Zum Assistenten wechseln aus.
3. Wählen Sie Einfaches Routing für Routing-Richtlinie und klicken Sie auf Weiter.
4. Wählen Sie unter Datensätze konfigurieren die Option Einfachen Datensatz definieren.
5. Gehen Sie auf der Seite Einfachen Datensatz definieren wie folgt vor:
   - Geben Sie als Datensatzname `iot:Data-ATS`-Endpunkt oder `iot:CredentialProvider`-Endpunkt ein. Dieser muss mit dem Namen der privaten gehosteten Zone übereinstimmen.
   - Wenn Sie für Record type nur IPv4-Unterstützung wünschen, behalten Sie den Wert als bei. `A - Routes traffic to an IPv4 address and some AWS resources` Wenn Sie nur IPv6-Unterstützung wünschen, behalten Sie den Wert als bei. `AAAA - Routes traffic to an IPv6 address and some AWS resources` Wenn Sie Dual-Stack-Unterstützung (sowohl IPv4 als auch IPv6) wünschen, erstellen Sie zwei Datensätze (**A**und zwar **AAAA** in der Hosting-Zone) mit demselben Datensatznamen und demselben Datenverkehr für. Value/Route
   - Wählen Sie für den Value/Route Datenverkehr die Option Alias zum VPC-Endpunkt aus. Wählen Sie wie unter ??? beschrieben Ihre Region und dann den Endpunkt aus, den Sie zuvor erstellt haben, aus der angezeigten Liste der Endpunkte aus.
6. Wählen Sie Einfachen Datensatz definieren, um Ihren Datensatz zu erstellen.

Steuern des Zugriffs auf AWS IoT Core über VPC-Endpunkte

Sie können den Gerätezugriff so einschränken, dass er nur über AWS IoT Core den VPC-Endpunkt erlaubt ist, indem Sie VPC-Bedingungskontextschlüssel verwenden. AWS IoT Core unterstützt die folgenden VPC-bezogenen Kontextschlüssel:

- SourceVpc
- SourceVpce
- VPCSourceIp

> **Note**
>
> AWS IoT Core unterstützt keine Endpunktrichtlinien für VPC-Endpunkte.

Die folgende Richtlinie gewährt beispielsweise die Erlaubnis, eine Verbindung AWS IoT Core mit einer Client-ID herzustellen, die dem Namen des Dings entspricht, und die Veröffentlichung zu einem beliebigen Thema, dem der Dingname vorangestellt ist, vorausgesetzt, dass das Gerät eine Verbindung zu einem VPC-Endpunkt mit einer bestimmten VPC-Endpunkt-ID herstellt. Diese Richtlinie verweigert Verbindungsversuche zu Ihrem öffentlichen IoT-Datenendpunkt.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:SourceVpce": "vpce-1a2b3c4d"
            }
        }

        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/
${iot:Connection.Thing.ThingName}/*"
            ]
        }
    ]
```

```
}
```

Einschränkungen

VPC-Endpunkte werden derzeit für AWS IoT Core Steuerungsebenen-Endpunkte,AWS IoT Core Datenendpunkteund AWS IoT Core Credential Provider-Endpunkte unterstützt. VPC-Endpunkte werden nur für FIPS-Endpunkte (Federal Information Processing Standard) unterstützt, wenn die Steuerungsebene verwendet wird. AWS IoT Core

Einschränkungen von VPC-Endpunkten auf IoT-Steuerungsebene

In diesem Abschnitt werden die Einschränkungen von VPC-Endpunkten auf IoT-Steuerungsebene behandelt.

- VPC-Endpunkte stellen nur ATS-Zertifikate bereit.
- Benutzerdefinierte Domänen werden für Endpunkte der Kontrollebene nicht unterstützt.
- Informationen zu FIPS-Sicherheitsrichtlinien finden Sie unter FIPS-Sicherheitsrichtlinien.

Einschränkungen von VPC-Endpunkten für IoT-Daten

In diesem Abschnitt werden die Einschränkungen von VPC-Endpunkten für IoT-Daten behandelt.

- Die Keep-Alive-Zeiträume von MQTT sind auf 230 Sekunden begrenzt. Längere Keep-Alive-Zeiträume werden automatisch auf 230 Sekunden reduziert.
- Jeder VPC-Endpunkt unterstützt insgesamt 100.000 gleichzeitig verbundene Geräte. Wenn Sie mehr Verbindungen benötigen, siehe. Skalierung von VPC-Endpunkten mit AWS IoT Core
- VPC-Endpunkte stellen nur ATS-Zertifikate bereit, mit Ausnahme von benutzerdefinierten Domains.
- VPC-Endpunktrichtlinien werden nicht unterstützt.
- Für VPC-Endpoints, die für die AWS IoT Core Datenebene erstellt wurden, unterstützt AWS IoT Core nicht die Verwendung zonaler oder regionaler öffentlicher DNS-Einträge.

Einschränkungen der Endpunkte von Anmeldeinformationsanbietern

In diesem Abschnitt werden die Einschränkungen von VPC-Endpunkten für Anmeldeinformationsanbieter behandelt.

- VPC-Endpunkte stellen nur ATS-Zertifikate bereit.
- VPC-Endpunktrichtlinien werden nicht unterstützt.
- Benutzerdefinierte Domains werden für die Endpunkte von Anmeldeinformationsanbietern nicht unterstützt.
- Für VPC-Endpoints, die für den AWS IoT Core Anmeldeinformationsanbieter erstellt wurden, unterstützt die Verwendung zonaler oder regionaler öffentlicher DNS-Einträge AWS IoT Core nicht.

Skalierung von VPC-Endpunkten mit AWS IoT Core

AWS IoT Core Schnittstellen-VPC-Endpunkte sind auf 100.000 verbundene Geräte über einen einzigen Schnittstellenendpunkt begrenzt. Wenn Ihr Anwendungsfall mehr gleichzeitige Verbindungen zum Broker erfordert, empfehlen wir, mehrere VPC-Endpunkte zu verwenden und Ihre Geräte manuell über Ihre Schnittstellenendpunkte zu routen. Achten Sie beim Erstellen von privaten DNS-Datensätzen für die Weiterleitung von Datenverkehr zu Ihren VPC-Endpunkten darauf, so viele gewichtete Datensätze zu erstellen, wie Sie VPC-Endpunkte haben, um den Verkehr auf Ihre verschiedenen Endpunkte zu verteilen.

Verwenden von benutzerdefinierten Domains mit VPC-Endpunkten

Wenn Sie benutzerdefinierte Domains mit VPC-Endpunkten verwenden möchten, müssen Sie Ihre benutzerdefinierten Domainnameneinträge in einer privaten gehosteten Zone und Routing-Datensätze in Route53 erstellen. Weitere Informationen finden Sie unter Erstellen einer privat gehosteten Zone.

> **Note**
>
> Benutzerdefinierte Domänen werden nur für AWS IoT Core Datenendpunkte unterstützt.

Verfügbarkeit von VPC-Endpunkten für AWS IoT Core

AWS IoT Core Schnittstellen-VPC-Endpunkte sind in allen AWS IoT Core unterstützten Regionen verfügbar. AWS IoT Core VPC-Schnittstellen-Endpunkte für den AWS IoT Core Anmeldeinformationsanbieter werden in der Region China und nicht unterstützt. AWS GovCloud (US) Regions

Verwenden AWS IoT Device Management sicheres Tunneling mit VPC-Endpunkten mit Schnittstellen

AWS IoT Device Management Secure Tunneling unterstützt Schnittstellen-VPC-Endpunkte. Sie können VPC-Endpunkte verwenden, um den Verkehr zwischen Ihrer VPC und AWS IoT Secure Tunneling innerhalb des AWS Netzwerks aufrechtzuerhalten, ohne dass ein Internet-Gateway, ein NAT-Gerät, eine VPN-Verbindung oder eine Verbindung erforderlich ist. AWS Direct Connect

Interface-VPC-Endpunkte basieren auf einer Technologie AWS PrivateLink, mit der Sie privat auf Dienste zugreifen können, indem Sie private IP-Adressen verwenden. Weitere Informationen finden Sie im AWS PrivateLink Handbuch unter Zugreifen auf einen AWS Dienst über einen Schnittstellen-VPC-Endpunkt.

Inhalt

- Voraussetzungen
- Empfangen von Tunnelbenachrichtigungen über VPC-Endpunkte
- VPC-Endpunkte für sicheres Tunneling erstellen
- Konfiguration von VPC-Endpunktrichtlinien auf dem Proxyserver
- Nächste Schritte

Voraussetzungen

Bevor Sie VPC-Endpoints für erstellen, stellen Sie sicher AWS IoT Secure Tunneling, dass Sie über Folgendes verfügen:

- Ein AWS Konto mit den erforderlichen Berechtigungen zum Erstellen von VPC-Endpoints.
- Eine VPC in Ihrem AWS Konto.
- Verständnis von Konzepten für AWS IoT Device Management sicheres Tunneling.
- Vertrautheit mit VPC-Endpunktrichtlinien und AWS Identity and Access Management (IAM)

Empfangen von Tunnelbenachrichtigungen über VPC-Endpunkte

Um Tunnelbenachrichtigungen über einen VPC-Endpunkt zu erhalten, können Ihre Geräte über einen VPC-Endpunkt eine Verbindung zur AWS IoT Core Datenebene herstellen und das reservierte MQTT-Thema für sicheres Tunneling abonnieren.

Anweisungen zum Erstellen und Konfigurieren eines VPC-Endpunkts in der AWS IoT Core Datenebene finden Sie unter VPC-Endpunkte AWS IoT Core mit der Schnittstelle verwenden im AWS IoT Entwicklerhandbuch.

VPC-Endpunkte für sicheres Tunneling erstellen

Sie können VPC-Endpunkte sowohl für die Secure Tunneling-Kontrollebene als auch für den Proxyserver erstellen.

So erstellen Sie einen VPC-Endpunkt für sicheres Tunneling

1. Folgen Sie den Schritten unter Erstellen eines Schnittstellenendpunkts im Amazon VPC Developer Guide
2. Wählen Sie für den Servicenamen je nach Endpunkttyp eine der folgenden Optionen aus:

   Steuerebene

   - Standard: `com.amazonaws.<region>.iot.tunneling.api`
   - FIPS (in FIPS-Regionen verfügbar): `com.amazonaws.<region>.iot-fips.tunneling.api`

   Proxy-Server

   - Standard: `com.amazonaws.<region>.iot.tunneling.data`
   - FIPS (in FIPS-Regionen verfügbar): `com.amazonaws.<region>.iot-fips.tunneling.data`

   Ersetze es durch *`<region>`* dein. AWS-Region Beispiel, `us-east-1`.
3. Führen Sie die verbleibenden Schritte im Prozess zur Erstellung von VPC-Endpunkten gemäß Ihren Netzwerkanforderungen aus.

Konfiguration von VPC-Endpunktrichtlinien auf dem Proxyserver

Zusätzlich zur tokenbasierten Autorisierung für den Clientzugriff, die zur Autorisierung von Verbindungen zu Tunneln verwendet wird, können Sie VPC-Endpunktrichtlinien verwenden, um weiter einzuschränken, wie Geräte einen VPC-Endpunkt verwenden können, um eine Verbindung

zum Secure Tunneling Proxy Server herzustellen. VPC-Endpunktrichtlinien folgen einer IAM-like Syntax und werden auf dem VPC-Endpunkt selbst konfiguriert.

Beachten Sie, dass die einzige unterstützte IAM-Aktion für Proxyserver-VPC-Endpunktrichtlinien ist. `iot:ConnectToTunnel`

Im Folgenden finden Sie Beispiele für verschiedene VPC-Endpunktrichtlinien.

Beispiele für VPC-Endpunktrichtlinien für Proxyserver

Die folgenden Beispiele zeigen die Konfigurationen der VPC-Endpunktrichtlinien für Proxyserver für allgemeine Anwendungsfälle.

Example- Standardrichtlinie

Diese Richtlinie ermöglicht es Geräten in Ihrer VPC, über jedes AWS Konto eine Verbindung zu jedem Tunnel herzustellen, der sich in demselben Tunnel befindet, in AWS-Region dem der Endpunkt erstellt wurde.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "*",
            "Resource": "*"
        }
    ]
}
```

Example- Beschränken Sie den Zugriff auf bestimmte AWS Konten

Diese Richtlinie ermöglicht es dem VPC-Endpunkt, sich nur mit Tunneln in bestimmten AWS Konten zu verbinden.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
            "Resource": [
```

```
                "arn:aws:iot:us-east-1:111122223333:tunnel/*",
                "arn:aws:iot:us-east-1:444455556666:tunnel/*"
            ]
        }
    ]
}
```

Example- Beschränken Sie Verbindungen nach Tunnelendpunkt

Sie können den VPC-Endpunktzugriff so einschränken, dass nur Geräte eine Verbindung zum Quell- oder Zielende eines Tunnels herstellen können.

Nur Quelle:

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iot:ClientMode": "source"
                }
            }
        }
    ]
}
```

Nur Ziel:

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iot:ClientMode": "destination"
                }
```

```
            }
        }
    ]
}
```

Example- Beschränken Sie den Zugriff auf der Grundlage von Ressourcen-Tags

Diese Richtlinie ermöglicht es dem VPC-Endpunkt, sich nur mit Tunneln zu verbinden, die mit einem bestimmten Schlüssel-Wert-Paar gekennzeichnet sind.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Environment": "Production"
                }
            }
        }
    ]
}
```

Example- Kombinierte Versicherungsbedingungen

Diese Politik zeigt die Kombination mehrerer politischer Elemente. Sie ermöglicht Verbindungen zu jedem Tunnel in einem bestimmten AWS Konto, aber nur, wenn der Tunnel mit `AllowConnectionsThroughPrivateLink` set to gekennzeichnet ist `true` und der Client keine Verbindung zum Zielende des Tunnels herstellt.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
            "Resource": [
                "arn:aws:iot:us-east-1:111122223333:tunnel/*"
            ],
```

```
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/AllowConnectionsThroughPrivateLink": "true"
                }
            }
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "iot:ConnectToTunnel",
            "Resource": [
                "arn:aws:iot:us-east-1:111122223333:tunnel/*"
            ],
            "Condition": {
                "StringEquals": {
                    "iot:ClientMode": "destination"
                }
            }
        }
    ]
}
```

Nächste Schritte

Nachdem Sie Ihre VPC-Endpoints für erstellt und konfiguriert haben AWS IoT Secure Tunneling, sollten Sie Folgendes beachten:

- Testen Sie Ihre VPC-Endpunktkonfiguration, indem Sie Geräte über den Endpunkt verbinden.
- Überwachen Sie die VPC-Endpunktnutzung anhand von Amazon CloudWatch Metriken.
- Überprüfen und aktualisieren Sie Ihre VPC-Endpunktrichtlinien nach Bedarf für Ihre Sicherheitsanforderungen.

Weitere Informationen zu AWS IoT Device Management Secure Tunneling finden Sie unter. AWS IoT Secure Tunneling

Sicherheit der Infrastruktur in AWS IoT

Als Sammlung verwalteter Services AWS IoT ist sie durch die AWS globalen Netzwerksicherheitsverfahren geschützt, die im Whitepaper Amazon Web Services: Sicherheitsprozesse im Überblick beschrieben sind.

Sie verwenden AWS veröffentlichte API-Aufrufe, um AWS IoT über das Netzwerk darauf zuzugreifen. Clients müssen Transport Layer Security (TLS) 1.2 oder höher unterstützen. Clients müssen außerdem Cipher Suites mit Perfect Forward Secrecy (PFS) wie Ephemeral (DHE) oder Elliptic Curve Ephemeral Diffie-Hellman (ECDHE) unterstützen. Diffie-Hellman Die meisten modernen Systeme, z. B. Java 7 und höher, unterstützen diese Modi. Weitere Informationen finden Sie unter Transportsicherheit in AWS IoT Core.

Anforderungen müssen mit einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel signiert sein, der mit einem IAM-Prinzipal verknüpft ist. Alternativ können Sie mit AWS -Security-Token-Service (AWS STS) temporäre Sicherheitsanmeldeinformationen erstellen, um die Anforderungen zu signieren.

Sicherheitsüberwachung von Produktionsflotten oder Geräten mit AWS IoT Core

IoT-Flotten können aus einer großen Anzahl von Geräten mit unterschiedlichsten Funktionen bestehen, sind langlebig und geografisch verteilt. Aufgrund dieser Merkmale ist die Flotteneinrichtung komplex und fehleranfällig. Und da Geräte bezüglich Rechenleistung, Arbeitsspeicher und Speicherkapazitäten eingeschränkt sind, können Verschlüsselung und andere Formen der Sicherheit auf den Geräten selbst nur limitiert eingesetzt werden. Außerdem verwenden Geräte häufig Software mit bekannten Schwachstellen. Diese Faktoren machen IoT-Flotten zu einem attraktiven Ziel für Hacker und erschweren die kontinuierliche Sicherung Ihrer Geräteflotte.

AWS IoT Device Defender begegnet diesen Herausforderungen durch die Bereitstellung von Tools zur Identifizierung von Sicherheitsproblemen und Abweichungen von bewährten Verfahren. Sie können AWS IoT Device Defender verwenden, um verbundene Geräte zu analysieren, zu prüfen und zu überwachen, um ungewöhnliches Verhalten zu erkennen und Sicherheitsrisiken zu minimieren. AWS IoT Device Defender kann Geräteflotten überprüfen, um sicherzustellen, dass sie sich an bewährte Sicherheitsmethoden halten, und um abnormales Verhalten auf Geräten zu erkennen. Auf diese Weise können Sie einheitliche Sicherheitsrichtlinien für Ihre gesamte AWS IoT Geräteflotte durchsetzen und schnell reagieren, wenn Geräte kompromittiert werden. Weitere Informationen finden Sie unter AWS IoT Device Defender.

AWS IoT Device Advisor veröffentlicht Updates und Patches für Ihre Flotte nach Bedarf. AWS IoT Device Advisor aktualisiert Testfälle automatisch. Bei den von Ihnen ausgewählten Testfällen handelt es sich immer um die aktuelle Version. Weitere Informationen finden Sie unter Device Advisor.

Bewährte Sicherheitsmethoden in AWS IoT Core

Dieser Abschnitt enthält Informationen zu bewährten Sicherheitsmethoden für AWS IoT Core. Informationen zu den Sicherheitsregeln für IoT-Lösungen finden Sie unter Zehn goldene Regeln in Bezug auf die Sicherheit für IoT-Lösungen.

Schutz von MQTT-Verbindungen in AWS IoT

AWS IoT Coreist ein verwalteter Cloud-Dienst, der es verbundenen Geräten ermöglicht, einfach und sicher mit Cloud-Anwendungen und anderen Geräten zu interagieren. AWS IoT Core unterstützt HTTP, und MQTT WebSocket, ein einfaches Kommunikationsprotokoll, das speziell für die Tolerierung intermittierender Verbindungen entwickelt wurde. Wenn Sie eine Verbindung AWS IoT über MQTT herstellen, muss jede Ihrer Verbindungen mit einer Kennung verknüpft sein, die als Client-ID bezeichnet wird. MQTT-Client-IDs identifizieren MQTT-Verbindungen eindeutig. Wenn eine neue Verbindung mit einer Client-ID hergestellt wird, die bereits für eine andere Verbindung beansprucht wurde, löscht der AWS IoT Message Broker die alte Verbindung, um die neue Verbindung zuzulassen. Client-IDs müssen innerhalb jeder einzelnen AWS-Konto ID eindeutig sein AWS-Region. Das bedeutet, dass Sie nicht die globale Eindeutigkeit von Kunden-IDs außerhalb Ihrer eigenen AWS-Konto oder regionsübergreifend innerhalb Ihrer AWS-Konto Region erzwingen müssen.

Die Auswirkungen und der Schweregrad von getrennten MQTT-Verbindungen auf Ihre Geräteflotte hängt von vielen Faktoren ab. Dazu zählen:

- Ihr Anwendungsfall (z. B. die Daten, an die Ihre Geräte senden AWS IoT, wie viele Daten und die Häufigkeit, mit der die Daten gesendet werden).
- Ihre MQTT-Client-Konfiguration (z. B. Einstellungen zum automatischen Wiederverbinden, zugehörige Backoff-Timings und die Verwendung von persistenten MQTT-Sitzungen).
- Einschränkungen für Geräteressourcen.
- Die Grundursache der Verbindungstrennungen, ihre Aggressivität und ihre Persistenz.

Um Client-ID-Konflikte und ihre möglichen negativen Auswirkungen zu vermeiden, stellen Sie sicher, dass jedes Gerät oder jede mobile Anwendung über eine AWS IoT oder IAM-Richtlinie verfügt, die einschränkt, welche Client-IDs für MQTT-Verbindungen zum Message Broker verwendet werden können. AWS IoT Sie können beispielsweise eine IAM-Richtlinie verwenden, um zu verhindern, dass ein Gerät unbeabsichtigt die Verbindung eines anderen Geräts schließt, indem es eine bereits verwendete Client-ID nutzt. Weitere Informationen finden Sie unter Autorisierung.

Alle Geräte in Ihrer Flotte müssen über Anmeldeinformationen mit Rechten verfügen, die nur beabsichtigte Aktionen autorisieren. Dazu gehören (aber nicht beschränkt auf) AWS IoT MQTT-Aktionen wie das Veröffentlichen von Nachrichten oder das Abonnieren von Themen mit einem bestimmten Umfang und Kontext. Die spezifischen Berechtigungsrichtlinien können für Ihre Anwendungsfälle variieren. Ermitteln Sie die Berechtigungsrichtlinien, die Ihren Geschäfts- und Sicherheitsanforderungen am besten entsprechen.

Verwenden Sie AWS IoT Core Richtlinienvariablen und IAM-Richtlinienvariablen, um die Erstellung und Verwaltung von Berechtigungsrichtlinien zu vereinfachen. Richtlinienvariablen können in einer Richtlinie platziert werden. Sie werden zu dem Zeitpunkt, zu dem die Richtlinie ausgewertet wird, durch Werte aus der Anforderung des Geräts ersetzt. Mithilfe von Richtlinienvariablen können Sie eine einzelne Richtlinie für die Erteilung von Berechtigungen für mehrere Geräte erstellen. Sie können die relevanten Richtlinienvariablen für Ihren Anwendungsfall anhand Ihrer AWS IoT Kontokonfiguration, Ihres Authentifizierungsmechanismus und Ihres Netzwerkprotokolls, das für die Verbindung zum AWS IoT Message Broker verwendet wird, identifizieren. Um jedoch die besten Berechtigungsrichtlinien zu schreiben, müssen Sie die Besonderheiten Ihres Anwendungsfalls und Ihr Bedrohungsmodell berücksichtigen.

Wenn Sie Ihre Geräte beispielsweise in der AWS IoT Registrierung registriert haben, können Sie Ding-Richtlinienvariablen in AWS IoT Richtlinien verwenden, um Berechtigungen auf der Grundlage von Dingeigenschaften wie Dingnamen, Dingtypen und Dingattributwerten zu gewähren oder zu verweigern. Der Name des Dings wird aus der Client-ID in der MQTT-Verbindungsnachricht abgerufen, die gesendet wird, wenn ein Ding eine Verbindung AWS IoT herstellt. Die Ding-Richtlinienvariablen werden ersetzt, wenn ein Ding AWS IoT über MQTT mithilfe der gegenseitigen TLS-Authentifizierung oder MQTT über WebSocket das Protokoll mithilfe authentifizierter Amazon Cognito Cognito-Identitäten eine Verbindung herstellt. Sie können die AttachThingPrincipalAPI verwenden, um Zertifikate und authentifizierte Amazon Cognito Cognito-Identitäten an eine Sache anzuhängen. `iot:Connection.Thing.ThingName`ist eine nützliche Richtlinienvariable, um Client-ID-Beschränkungen durchzusetzen. Die folgende AWS IoT Beispielrichtlinie erfordert, dass der Name einer registrierten Sache als Client-ID für MQTT-Verbindungen zum AWS IoT Message Broker verwendet wird:

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

```
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
    ]
   }
  ]
}
```

Wenn Sie laufende Client-ID-Konflikte identifizieren möchten, können Sie CloudWatch Logs for AWS IoT aktivieren und verwenden. Für jede MQTT-Verbindung, die der AWS IoT Message Broker aufgrund von Client-ID-Konflikten unterbricht, wird ein Protokolldatensatz generiert, der dem folgenden ähnelt:

```
{
    "timestamp": "2019-04-28 22:05:30.105",
    "logLevel": "ERROR",
    "traceId": "02a04a93-0b3a-b608-a27c-1ae8ebdb032a",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "Disconnect",
    "protocol": "MQTT",
    "clientId": "clientId01",
    "principalId": "1670fcf6de55adc1930169142405c4a2493d9eb5487127cd0091ca0193a3d3f6",
    "sourceIp": "203.0.113.1",
    "sourcePort": 21335,
    "reason": "DUPLICATE_CLIENT_ID",
    "details": "A new connection was established with the same client ID"
}
```

Sie können einen CloudWatch Protokollfilter verwenden, `{$.reason= "DUPLICATE_CLIENT_ID" }` um beispielsweise nach Instanzen von Client-ID-Konflikten zu suchen oder um CloudWatch Metrikfilter und entsprechende CloudWatch Alarme für die kontinuierliche Überwachung und Berichterstattung einzurichten.

Sie können AWS IoT Device Defender verwenden, um zu freizügige Richtlinien AWS IoT und IAM-Richtlinien zu identifizieren. AWS IoT Device Defender bietet auch eine Auditprüfung, die Sie benachrichtigt, wenn mehrere Geräte in Ihrer Flotte über dieselbe Client-ID eine Verbindung zum AWS IoT Message Broker herstellen.

Mit AWS IoT Device Advisor können Sie überprüfen, ob Ihre Geräte eine zuverlässige Verbindung zu den bewährten Sicherheitsmethoden herstellen AWS IoT Core und diese befolgen.

Weitere Informationen finden Sie auch unter

- AWS IoT Core
- AWS IoT-Sicherheitsfunktionen
- AWS IoT Core Richtlinienvariablen
- IAM-Richtlinienvariablen
- Amazon Cognito-Identität
- AWS IoT Device Defender
- CloudWatch Loggt sich für AWS IoT

Synchronisieren der internen Uhr eines Geräts

Es ist wichtig, dass Sie eine genaue Uhrzeit auf Ihrem Gerät haben. X.509 Zertifikate haben ein Ablaufdatum und eine Ablaufzeit. Die Uhr auf Ihrem Gerät wird verwendet, um sicherzustellen, dass ein Serverzertifikat noch gültig ist. Wenn Sie kommerzielle IoT-Geräte bauen, denken Sie daran, dass Ihre Produkte möglicherweise über einen längeren Zeitraum gelagert werden, bevor sie verkauft werden. Real-time Während dieser Zeit können Uhren schwanken und Batterien können sich entladen, sodass die werkseitige Einstellung der Zeit nicht ausreicht.

Für die meisten Systeme bedeutet dies, dass die Software des Geräts einen NTP-Client (Network Time Protocol) enthalten muss. Das Gerät sollte warten, bis es mit einem NTP-Server synchronisiert wird, bevor es versucht, eine Verbindung zu AWS IoT Core herzustellen. Wenn dies nicht möglich ist, sollte das System dem Benutzer die Möglichkeit geben, die Zeit des Gerätes so einzustellen, dass nachfolgende Verbindungen erfolgreich sind.

Nachdem das Gerät mit einem NTP-Server synchronisiert wurde, kann es eine Verbindung mit AWS IoT Core herstellen. Wie viel Taktversatz zulässig ist, hängt davon ab, was Sie mit der Verbindung tun möchten.

Überprüfen des Serverzertifikats

Das erste, was ein Gerät tut, um mit ihm zu interagieren, AWS IoT ist, eine sichere Verbindung herzustellen. Wenn Sie Ihr Gerät mit verbinden, stellen Sie sicher AWS IoT, dass Sie mit einem anderen Server sprechen AWS IoT und nicht mit einem anderen Server, der sich als solcher ausgibt. AWS IoT Jeder der AWS IoT Server ist mit einem Zertifikat ausgestattet, das für die Domain ausgestellt wurde. `iot.amazonaws.com` Dieses Zertifikat wurde AWS IoT von einer

vertrauenswürdigen Zertifizierungsstelle ausgestellt, die unsere Identität und unser Eigentum an der Domain verifizierte.

Wenn ein Gerät eine Verbindung herstellt, wird dem Gerät als Erstes ein Serverzertifikat gesendet. AWS IoT Core Geräte können überprüfen, ob sie eine Verbindung zu `iot.amazonaws.com` erwartet haben und ob der Server am Ende dieser Verbindung ein Zertifikat einer vertrauenswürdigen Autorität für diese Domäne hat.

TLS-Zertifikate haben ein X.509 Format und enthalten eine Vielzahl von Informationen wie den Namen, den Standort, den Domainnamen und eine Gültigkeitsdauer der Organisation. Der Gültigkeitszeitraum wird als ein Paar von Zeitwerten angegeben, die `notBefore` und `notAfter` genannt werden. Dienste wie AWS IoT Core verwenden begrenzte Gültigkeitszeiträume (z. B. ein Jahr) für ihre Serverzertifikate und beginnen, neue Zertifikate bereitzustellen, bevor die alten ablaufen.

Verwenden einer einzigen Identität pro Gerät

Verwenden Sie eine einzige Identität pro Client. Geräte verwenden in der Regel X.509 Client-Zertifikate. Web- und Mobilanwendungen verwenden Amazon Cognito Identity. Auf diese Weise können Sie detaillierte Berechtigungen auf Ihre Geräte anwenden.

Sie könnten beispielsweise eine Anwendung haben, die aus einem Mobiltelefon besteht, das Statusaktualisierungen von zwei verschiedenen Smart-Home-Objekten empfängt einer Glühbirne und einem Thermostat. Die Glühbirne sendet den Status ihres Batteriestands und ein Thermostat sendet Meldungen über die Temperatur.

AWS IoT authentifiziert Geräte einzeln und behandelt jede Verbindung einzeln. Sie können differenzierte Zugriffskontrollen mithilfe von Autorisierungsrichtlinien anwenden. Sie können eine Richtlinie für den Thermostat definieren, die es ihm ermöglicht, in einem Themenbereich zu veröffentlichen. Sie können eine separate Richtlinie für die Glühbirne definieren, die es ihr ermöglicht, in einem anderen Themenbereich zu veröffentlichen. Schließlich können Sie eine Richtlinie für die mobile App definieren, die es ihr nur erlaubt, sich mit den Themen für den Thermostat und die Glühbirne zu verbinden und diese zu abonnieren, um Nachrichten von diesen Geräten zu empfangen.

Wenden Sie das Prinzip der geringsten Privilegien an und beschränken Sie die Berechtigungen pro Gerät so weit wie möglich. Für alle Geräte oder Benutzer sollte eine AWS IoT Richtlinie gelten AWS IoT , die es ihnen nur erlaubt, eine Verbindung mit einer bekannten Client-ID herzustellen und bestimmte Themen zu veröffentlichen und zu abonnieren.

Verwenden Sie eine Sekunde AWS-Region als Backup

Erwägen Sie, eine Kopie Ihrer Daten in einer Sekunde AWS-Region als Backup zu speichern. Beachten Sie, dass die AWS Lösung mit dem Namen Disaster Recovery for nicht mehr verfügbar AWS IoT ist. Auf die zugehörige GitHubBibliothek kann zwar weiterhin zugegriffen werden, sie AWS wurde jedoch im Juli 2023 als veraltet eingestuft und bietet keine Wartung oder Unterstützung mehr für sie. Besuchen Sie Kontakt, um Ihre eigenen Lösungen zu implementieren oder zusätzliche Supportoptionen zu erkunden. AWS Wenn mit Ihrem Konto ein AWS Technical Account Manager verknüpft ist, wenden Sie sich an diesen, um Hilfe zu erhalten.

Just-in-Time-Bereitstellung nutzen

Die manuelle Erstellung und Bereitstellung jedes Geräts kann zeitaufwändig sein. AWS IoT bietet die Möglichkeit, eine Vorlage zu definieren, mit der Geräte bereitgestellt werden, wenn sie zum AWS IoT ersten Mal eine Verbindung herstellen. Weitere Informationen finden Sie unter Just-in-time Bereitstellung.

Berechtigungen zum Ausführen AWS IoT Device Advisor-Tests

Die folgende Richtlinienvorlage zeigt die Mindestberechtigungen und die IAM-Entität, die für die Ausführung von AWS IoT Device Advisor-Testfällen erforderlich sind. Sie müssen Amazon Resource Name (ARN) durch die Geräterolle ersetzen*your-device-role-arn*, die Sie unter den Voraussetzungen erstellt haben.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iot:us-east-1:123456789012:thinggroup/your-thing-
group",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "iotdeviceadvisor.amazonaws.com"
            }
        }
        },
        {
```

```
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke*",
                "iam:ListRoles",
                "iot:Connect",
                "iot:CreateJob",
                "iot:DeleteJob",
                "iot:DescribeCertificate",
                "iot:DescribeEndpoint",
                "iotjobsdata:DescribeJobExecution",
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iotjobsdata:GetPendingJobExecutions",
                "iot:GetPolicy",
                "iot:ListAttachedPolicies",
                "iot:ListCertificates",
                "iot:ListPrincipalPolicies",
                "iot:ListThingPrincipals",
                "iot:ListThings",
                "iot:Publish",
                "iotjobsdata:StartNextPendingJobExecution",
                "iotjobsdata:UpdateJobExecution",
                "iot:UpdateThingShadow",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:PutRetentionPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": "iotdeviceadvisor:*",
            "Resource": "*"
        }
    ]
}
```

Cross-service Confused Deputy Prevention für Device Advisor

Das Problem des verwirrten Stellvertreters ist ein Sicherheitsproblem, bei dem eine Entität, die keine Berechtigung zur Durchführung einer Aktion hat, eine privilegiertere Entität zur Durchführung der Aktion zwingen kann. In AWS kann ein dienstübergreifendes Identitätswechsels zu einem Problem mit dem verwirrten Stellvertreter führen. Cross-service Ein Identitätswechsel kann auftreten, wenn ein Dienst (der anrufende Dienst) einen anderen Dienst (den angerufenen Dienst) aufruft. Der Anruf-Dienst kann so manipuliert werden, dass er seine Berechtigungen verwendet, um auf die Ressourcen eines anderen Kunden zu reagieren, auf die er sonst nicht zugreifen dürfte. Um dies zu verhindern, werden Tools AWS bereitgestellt, mit denen Sie Ihre Daten für alle Dienste mit Dienstprinzipalen schützen können, denen Zugriff auf Ressourcen in Ihrem Konto gewährt wurde.

Wir empfehlen die Verwendung der globalen Bedingungskontext-Schlüssel `aws:SourceArn` und `aws:SourceAccount` in ressourcenbasierten Richtlinien, um die Berechtigungen, die Device Advisor einem anderen Service erteilt, auf eine bestimmte Ressource zu beschränken. Wenn Sie beide globalen Bedingungskontextschlüssel verwenden, müssen der `aws:SourceAccount`-Wert und das Konto im `aws:SourceArn`-Wert dieselbe Konto-ID verwenden, wenn sie in derselben Richtlinienanweisung verwendet werden.

Der Wert `aws:SourceArn` muss der ARN Ihrer Suite-Definitionsressource sein. Die Suite-Definitionsressource bezieht sich auf die Testsuite, die Sie mit Device Advisor erstellt haben.

Der effektivste Weg, um sich vor dem Confused-Deputy-Problem zu schützen, ist die Verwendung des globalen Bedingungskontext-Schlüssels `aws:SourceArn` mit dem vollständigen ARN der Ressource. Wenn Sie den vollständigen ARN der Ressource nicht kennen oder wenn Sie mehrere Ressourcen angeben, verwenden Sie den globalen Bedingungskontext-Schlüssel `aws:SourceArn` mit Platzhaltern (*) für die unbekannten Teile des ARN. Beispiel: `arn:aws:iotdeviceadvisor:*:`*`account-id`*`:suitedefinition/*`

Das folgende Beispiel zeigt, wie Sie die globalen Bedingungskontext-Schlüssel `aws:SourceArn` und `aws:SourceAccount` in Device Advisor verwenden können, um das Confused-Deputy-Problem zu vermeiden.

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

```
        "Principal": {
            "Service": "iotdeviceadvisor.amazonaws.com"
        },
        "Action": "sts:AssumeRole",
        "Condition": {
            "ArnLike": {
                "aws:SourceArn": "arn:aws:iotdeviceadvisor:us-
east-1:123456789012:suitedefinition/ygp6rxa3tzvn"
        },
            "StringEquals": {
                "aws:SourceAccount": "123456789012"
        }
        }
    }
}
```

AWS Schulung und Zertifizierung

Nehmen Sie an dem folgenden Kurs teil, um mehr über wichtige Sicherheitskonzepte zu erfahren: AWS IoT AWS IoT Security Primer.

Überwachen AWS IoT

Die Überwachung ist ein wichtiger Bestandteil der Aufrechterhaltung der Zuverlässigkeit, Verfügbarkeit AWS IoT und Leistung Ihrer AWS Lösungen.

Wir empfehlen Ihnen dringend, Überwachungsdaten aus allen Teilen Ihrer AWS Lösung zu sammeln, um das Debuggen eines Fehlers an mehreren Stellen zu erleichtern, falls einer auftritt. Erstellen Sie zunächst einen Überwachungsplan, der die folgenden Fragen beantwortet. Wenn Sie nicht sicher sind, wie Sie diese beantworten sollen, können Sie trotzdem die Protokollierung aktivieren und Ihre Leistungsgrundlagen festlegen.

- Was sind Ihre Überwachungsziele?
- Welche Ressourcen möchten Sie überwachen?
- Wie oft werden diese Ressourcen überwacht?
- Welche Überwachungs-Tools möchten Sie verwenden?
- Wer soll die Überwachungsaufgaben ausführen?
- Wer soll benachrichtigt werden, wenn Fehler auftreten?

Ihr nächster Schritt besteht darin, die Protokollierung zu aktivieren und einen Basiswert für die normale AWS IoT Leistung in Ihrer Umgebung festzulegen, indem Sie die Leistung zu verschiedenen Zeiten und unter verschiedenen Lastbedingungen messen. Bewahren Sie bei der Überwachung historische Überwachungsdaten auf, damit Sie sie mit aktuellen Leistungsdaten vergleichen können. AWS IoT Auf diese Weise können Sie normale Leistungsmuster und Leistungsanomalien identifizieren und Methoden zu deren Handhabung entwickeln.

Um Ihre Ausgangsleistung für zu ermitteln AWS IoT, sollten Sie diese Kennzahlen zunächst überwachen. Später können Sie immer noch weitere Metriken überwachen.

- `PublishIn.Success`
- `PublishOut.Success`
- `Subscribe.Success`
- `Ping.Success`
- `Connect.Success`
- `GetThingShadow.Accepted`
- `UpdateThingShadow.Accepted`

- `DeleteThingShadow.Accepted`
- `RulesExecuted`

Die Themen in diesem Abschnitt dienen als Einstieg in die Protokollierung und Überwachung von AWS IoT.

Themen

- Konfiguration AWS IoT Protokollierung
- Überwachen AWS IoT Alarme und Metriken mithilfe von Amazon CloudWatch
- Überwachen AWS IoT mithilfe von Protokollen CloudWatch
- Laden Sie geräteseitige Protokolle auf Amazon hoch CloudWatch
- Protokollierung AWS IoT API-Aufrufe mit AWS CloudTrail

Konfiguration AWS IoT Protokollierung

Sie müssen die Protokollierung mithilfe der AWS IoT Konsole, CLI oder API aktivieren, bevor Sie AWS IoT Aktivitäten überwachen und protokollieren können. Sie können die Protokollierung AWS IoT auf drei Ebenen konfigurieren: auf Kontoebene, auf Veranstaltungsebene oder auf ressourcenspezifischer Ebene. Event-level und ressourcenspezifische Protokollierung sind ausschließlich mit der V2-Protokollierung verfügbar. Kunden, die V1-Protokollierung verwenden, müssen eine Migration zu V2 durchführen, um auf diese Funktionen zugreifen zu können. Einzelheiten finden Sie hier.

Wenn Sie überlegen, wie Sie Ihre AWS IoT Protokollierung konfigurieren, bestimmt die Protokollierungskonfiguration auf Kontoebene, wie AWS IoT Aktivitäten protokolliert werden, sofern nicht anders angegeben. Zu Beginn möchten Sie möglicherweise detaillierte Protokolle mit der https://docs.aws.amazon.com/iot/latest/developerguide/configure-logging.html#log-level Standardprotokollebene INFO oder DEBUG abrufen. Nachdem Sie die ersten Protokolle überprüft haben, können Sie die Standardprotokollebene auf eine weniger ausführliche Ebene wie WARN oder ERROR auf Konto- oder Ereignisebene ändern und eine ausführlichere ressourcenspezifische Protokollebene für Ressourcen festlegen, die möglicherweise mehr Aufmerksamkeit erfordern. Protokollstufen können jederzeit geändert werden.

In diesem Thema wird die cloudseitige Anmeldung behandelt. AWS IoT Informationen zur geräteseitigen Protokollierung und Überwachung finden Sie unter Geräteseitige Protokolle hochladen auf. CloudWatch

Informationen zur Protokollierung und Überwachung finden Sie unter Protokollieren und Überwachen AWS IoT Greengrass in. AWS IoT Greengrass

Konfiguration der V2-Anmeldung AWS IoT

Ermitteln Sie Ihre Logging-Version

Die GetV2LoggingOptions API gibt eine zurück, NotConfiguredException wenn die V2-Protokollierung nicht aktiviert ist. Dieser Fehler tritt auf, wenn entweder die V1-Protokollierung verwendet wird oder keine Protokollierung konfiguriert wurde.

Grundlegendes zu den Protokollierungsfunktionen von

Die V2-Protokollierung bietet zwei wichtige Funktionen: die Protokollierung auf Veranstaltungsebene und die ressourcenspezifische Protokollierung. Event-levelDie Protokollierung ermöglicht eine gezielte Protokollierungskonfiguration mit anpassbaren Protokollebenen und Protokollgruppenzielen. CloudWatch Resource-specific Mithilfe der Protokollierung können Sie Protokolle nach Dinggruppe, Quell-IP, Client-ID oder Prinzipal-ID filtern. Zusammen bieten diese Funktionen eine detaillierte Kontrolle und einen umfassenden Einblick in den IoT-Betrieb, wodurch die Durchsuchbarkeit von Protokollen verbessert und die Kosten gesenkt werden, da unnötige Protokollierungsaktivitäten vermieden werden.

Migrieren von V1 auf V2

Sie können mithilfe der SetV2LoggingOptions API entweder über die AWS CLI oder das SDK zur V2-Protokollierung migrieren. Leitet alle Protokolle nach der Migration AWS IoT automatisch an die CloudWatch Protokollgruppe 'AWSIotLogsV2' weiter. Wichtig: Wenn Ihre Downstream-Anwendungen oder Ressourcen Informationen aus 'AWSIotLogs' verbrauchen, aktualisieren Sie sie, sodass sie den entsprechenden Protokollgruppenpfad verwenden.

Konfigurieren der Protokollierungsrolle und -richtlinie

Bevor Sie die Anmeldung aktivieren können AWS IoT, müssen Sie eine IAM-Rolle und eine Richtlinie erstellen, die Ihnen die AWS IoT Erlaubnis erteilt, AWS IoT Protokollaktivitäten in Ihrem Namen in CloudWatch Protokollgruppen zu schreiben. Sie können auch eine IAM-Rolle mit den erforderlichen Richtlinien im Abschnitt „Protokolle“ der AWS IoT Konsole generieren.

> **Note**
>
> Bevor Sie die AWS IoT Protokollierung aktivieren, stellen Sie sicher, dass Sie die Zugriffsberechtigungen für CloudWatch Protokolle verstanden haben. Benutzer mit Zugriff auf CloudWatch Protokolle können Debugging-Informationen von Ihren Geräten einsehen. Weitere Informationen finden Sie unter Authentifizierung und Zugriffskontrolle für Amazon CloudWatch Logs.
>
> Wenn Sie AWS IoT Core aufgrund von Lasttests mit einem hohen Verkehrsaufkommen rechnen, sollten Sie erwägen, die IoT-Protokollierung zu deaktivieren, um Drosselungen zu verhindern. Wenn ein hoher Datenverkehr festgestellt wird, deaktiviert unser Service möglicherweise die Protokollierung in Ihrem Konto.

Im Folgenden wird gezeigt, wie Sie eine Protokollierungsrolle und eine Richtlinie für Ressourcen erstellen. AWS IoT Core

Erstellen einer Protokollierungsrolle

Um eine Protokollierungsrolle zu erstellen, öffnen Sie den Rollen-Hub der IAM-Konsole und wählen Sie Rolle erstellen aus.

1. Wählen Sie unter Vertrauenswürdige Entität auswählen die Option AWS -Service aus. Wählen Sie dann die Option IoT unter Anwendungsfall aus. Wenn IoT nicht angezeigt wird, geben Sie IoT im Drop-down-Menü Anwendungsfälle für andere AWS -Services ein und suchen Sie danach. Klicken Sie auf Weiter.
2. Auf der Seite Berechtigungen hinzufügen sehen Sie die Richtlinien, die automatisch mit der Servicerolle verknüpft werden. Wählen Sie Weiter aus.
3. Geben Sie auf der Seite Name, Überprüfen und Erstellen einen Rollennamen und eine Rollenbeschreibung für die Rolle ein und wählen Sie dann Rolle erstellen.

Richtlinie für die Protokollierungsrolle

Die folgenden Richtliniendokumente enthalten die Rollen- und Vertrauensrichtlinien, die es AWS IoT ermöglichen, Protokolleinträge in Ihrem Namen CloudWatch an Dritte zu senden. Wenn Sie die Protokollierung auf Veranstaltungsebene mit einer benutzerdefinierten CloudWatch Protokollgruppe konfigurieren, müssen Sie die Rollenrichtlinie so aktualisieren, dass sie den benutzerdefinierten Ressourcen-ARN enthält.

Wenn Sie auch das Senden von Protokolleinträgen erlaubt AWS IoT Core für LoRaWAN haben, wird ein für Sie erstelltes Richtliniendokument angezeigt, in dem beide Aktivitäten protokolliert werden.

> **Note**
>
> Diese Dokumente wurden für Sie erstellt, als Sie die Protokollierungsrolle erstellt haben. Die Dokumente enthalten Variablen *${partition}*, *${region}*, und *${accountId}*, die Sie durch Ihre Werte ersetzen müssen.
>
> - Ersetzen Sie die Partition durch die Partition der Region.
> - Ersetzen Sie die Region durch die AWS Region, die Sie verwenden. Stellen Sie sicher, dass Sie dieselbe AWS Region verwenden, die Sie für die Konfiguration der AWS CLI auf Ihrem Gerät verwendet haben.
> - Ersetzen Sie Account-ID durch Ihre AWS Konto-ID.

Rollenrichtlinie:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:PutMetricFilter",
                "logs:PutRetentionPolicy",
                "iot:GetLoggingOptions",
                "iot:SetLoggingOptions",
                "iot:SetV2LoggingOptions",
                "iot:GetV2LoggingOptions",
                "iot:SetV2LoggingLevel",
                "iot:ListV2LoggingLevels",
                "iot:DeleteV2LoggingLevel"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:123456789012:log-group:AWSIotLogsV2:*"
            ]
```

```
        }
    ]
}
```

Vertrauensrichtlinie, um nur AWS IoT Core Aktivitäten zu protokollieren:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
        },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

AWS IoT Logging kann die Protokollierung möglicherweise aufgrund unzureichender CloudWatch IAM-Rollenberechtigungen nicht in Logs veröffentlichen. Wenn dies der Fall ist, überprüfen Sie die CloudWatch Logging-Metriken, um die Fehler zu untersuchen und zu beheben.

Konfigurieren Sie die Protokollierung im AWS IoT (Konsole)

In diesem Abschnitt wird beschrieben, wie Sie die AWS IoT Protokollierung mithilfe der AWS IoT Konsole konfigurieren. Sie können die Protokollierung auf Konto-, Ereignis- und ressourcenspezifische Protokollierung einrichten.

Um zu konfigurieren AWS IoT Protokollierung:

1. Loggen Sie sich in die AWS IoT Konsole ein. Weitere Informationen finden Sie unter Öffnen Sie AWS IoT Konsole.
2. Wählen Sie im linken Navigationsbereich Logs aus (zuvor ein Abschnitt unter Einstellungen).
3. Konfiguriere die Protokollierung auf Kontoebene: Die Protokollierung auf Kontoebene gilt für deine gesamte AWS IoT Flotte (Geräte oder Endpunkte), sofern sie nicht durch ereignisspezifische oder ressourcenspezifische Einstellungen außer Kraft gesetzt wird.

a. Wählen Sie unter Protokollierung Account-level die Option Protokollierung auf Kontoebene verwalten aus, um Aktualisierungen vorzunehmen.

b. Markieren Sie das Kontrollkästchen „Protokollierung aktivieren“, um mit dem Senden von Protokollen an diese Adresse zu beginnen. CloudWatch Wenn „Protokollierung aktivieren“ nicht markiert ist, AWS IoT werden unabhängig von den Protokollierungskonfigurationen auf Ereignis- oder Ressourcenebene keine Protokolle an CloudWatch Protokollgruppen gesendet.

c. Wählen Sie unter IAM-Protokollrolle eine vorhandene Rolle aus der Dropdownliste aus. Sie können Rollendetails anzeigen, um die Rollenberechtigungen zu überprüfen. Wählen Sie alternativ Neue Rolle erstellen aus, um eine neue IAM-Rolle einzurichten. Die Logging-Rolle bietet Richtlinien, die es AWS IoT ermöglichen, Logeinträge in Ihrem Namen CloudWatch an Dritte zu senden. Wenn Sie die Protokollierung auf Veranstaltungsebene mit einer benutzerdefinierten CloudWatch Protokollgruppe konfigurieren, müssen Sie die Rollenrichtlinie so aktualisieren, dass sie den ARN für diese Protokollgruppe enthält.

d. Wählen Sie die Standard-Protokollebene, die der https://docs.aws.amazon.com/iot/latest/developerguide/configure-logging.html#log-level Detailebene der Protokolleinträge entspricht, die in den CloudWatch Protokollen angezeigt werden sollen. Hinweis: Die Protokollebene „DEBUG“ bietet die meisten Details, erhöht jedoch die CloudWatch Kosten. CloudWatch Ziele für Protokollgruppen können nicht auf Kontoebene konfiguriert werden. Sie können jedoch benutzerdefinierte Protokollgruppen für einzelne Ereignistypen angeben, wie im folgenden Abschnitt beschrieben.

e. Wählen Sie Protokollierung aktualisieren, um Ihre Änderungen zu speichern.

4. Event-level Mit der Protokollierung können Sie selektiv Protokolle für relevante Ereignisse erfassen und sie an spezielle CloudWatch Protokollgruppen weiterleiten. Auf diese Weise haben Sie die Flexibilität, Protokolle zur besseren Auffindbarkeit nach Anwendungsfällen zu organisieren, sie für verschiedene Zielgruppen freizugeben und die CloudWatch Kosten zu senken, indem Sie Protokolle aktivieren und Protokollebenen auf der Grundlage der Ereigniskritik einrichten.

   Konfigurieren Sie die Protokollierung auf Ereignisebene: Die Protokollierung auf Ereignisebene erfasst bestimmte AWS IoT Ereignisse, wie z. B. Client-Authentifizierungsversuche. Diese Einstellungen überschreiben die Protokollierung. Account-Level

   a. Wählen Sie im Abschnitt Event-level Protokollierung die Option Protokollierung auf Veranstaltungsebene verwalten aus, um Aktualisierungen vorzunehmen.

b. Standardmäßig übernehmen Ereignistypen die Protokollierungskonfiguration auf Kontoebene. Hinweis: Wenn die ressourcenspezifische Protokollierung konfiguriert ist, überschreibt sie die Einstellungen auf Konto- und Veranstaltungsebene.

c. Um die Einstellungen für einzelne Ereignisse zu ändern, klicken Sie auf den Wert in der entsprechenden Ereigniszeile. Sie können sowohl die Protokollebene als auch das Ziel der CloudWatch Protokollgruppe anpassen. Wenn Sie ein benutzerdefiniertes CloudWatch Protokollgruppenziel angeben, müssen Sie sicherstellen, dass die IAM-Rollenrichtlinie Berechtigungen für die neue Protokollgruppe enthält. Wenn die Rollenrichtlinie nicht aktualisiert wird, können keine AWS IoT Protokolle in die benutzerdefinierte Protokollgruppe geschrieben werden. Nachdem Sie Ihre Auswahl getroffen haben, klicken Sie auf das Häkchen, um Ihre Auswahl zu bestätigen. In der Spalte „Ist geändert“ wird „Ja“ angezeigt, um auf ausstehende Änderungen hinzuweisen.

d. Klicken Sie auf Protokollierung aktualisieren, um Ihre Änderungen zu übernehmen, oder wählen Sie Abbrechen, um sie zu verwerfen.

5. Resource-specific Überschreibungen konfigurieren: Bei Resource-specific Überschreibungen wird eine Protokollierungseinstellung auf ausgewählte Ressourcen angewendet. Bei einer Ressource kann es sich um eine Dinggruppe, Quell-IP, Client-ID oder Prinzipal-ID handeln. Resource-specific Die Protokollierungskonfiguration überschreibt sowohl Einstellungen auf Konto- als auch auf Veranstaltungsebene. Wenn diese Option aktiviert ist, generiert sie Protokolle für alle Ereignistypen auf der konfigurierten Protokollierungsebene für die angegebenen Ressourcen. Sie können beispielsweise die Protokollierung auf Debug-Ebene für ein bestimmtes Ding einrichten und gleichzeitig die Protokollierung auf Infoebene für alle anderen Dinge beibehalten.

   a. Wählen Sie im Abschnitt Überschreibungen die Option Ressourcenspezifische Überschreibungen hinzufügen aus. Resource-specific

   b. Wählen Sie ein Protokollziel: Dinggruppe, Quell-IP, Client-ID oder Prinzipal-ID.

   c. Geben Sie den entsprechenden Log-Zielwert für den ausgewählten Zieltyp ein.

   d. Wählen Sie das gewünschte Log-Level aus dem Drop-down-Menü im Abschnitt Resource-specific Log Level aus.

   e. Klicken Sie auf Senden, um die Überschreibung hinzuzufügen, oder auf Abbrechen, um die Änderungen zu verwerfen.

   f. Um eine bestehende ressourcenspezifische Überschreibung zu ändern, markieren Sie das Kontrollkästchen neben der Ressource und klicken Sie auf „Entfernen“, um die Überschreibung zu löschen, oder auf „Bearbeiten“, um sie zu ändern.

Nachdem Sie die Protokollierung aktiviert haben, finden Sie unter Anzeigen AWS IoT loggt sich in der CloudWatch Konsole ein Informationen zum Anzeigen der Protokolleinträge.

Konto konfigurieren und anmelden Event-level AWS IoT (CLI)

In diesem Abschnitt wird beschrieben, wie Sie die globale Protokollierung AWS IoT mithilfe der CLI konfigurieren.

Sie können die Event-level Protokollierung optional konfigurieren. Event-level Bei der Protokollierung werden Protokollinformationen auf Ereignisebene erfasst, z. B. Authentifizierungs- und Autorisierungsereignisse oder Ereignisse zur Zertifikatserstellung. Sie können sowohl die Protokollebene als auch die Ziele der CloudWatch Protokollgruppe auf Ereignisebene anpassen. Event-level Die Protokollierung erfolgt auf einer gezielteren Ebene als die Protokollierung auf Kontoebene und überschreibt daher die Protokollierungseinstellungen auf Kontoebene. Dieser hierarchische Ansatz ermöglicht es Ihnen, je nach betrieblicher Bedeutung und Kostenaspekten unterschiedliche Protokollierungsstrategien für verschiedene Arten von Ereignissen beizubehalten.

> **Note**
>
> Sie benötigen den Amazon-Ressourcennamen (ARN) der Rolle, die Sie verwenden möchten. Wenn Sie eine Rolle für die Protokollierung erstellen müssen, beachten Sie zunächst Erstellen einer Protokollierungsrolle, bevor Sie weitermachen. Wenn Sie eine benutzerdefinierte CloudWatch Protokollgruppe für einen beliebigen Ereignistyp angeben, stellen Sie sicher, dass Ihre Protokollierungsrolle über die erforderlichen Berechtigungen für die Zielprotokollgruppe verfügt.
> Der zum Aufrufen der API verwendete Prinzipal benötigt Rollenberechtigungen weitergeben für Ihre Protokollierungsrolle.

Sie können dieses Verfahren auch mit der API ausführen, indem Sie die Methoden in der AWS API verwenden, die den hier gezeigten CLI-Befehlen entsprechen.

Um die CLI zum Konfigurieren der Standardprotokollierung für zu verwenden AWS IoT

1. Verwenden Sie den Befehl set-v2-logging-options, um die Protokollierungsoptionen für Ihr Konto festzulegen.

```
aws iot set-v2-logging-options \
    --event-configurations event-configuration-list \
```

```
    --role-arn logging-role-arn \
    --default-log-level log-level
```

Wobei:

--role-arn

Der Rollen-ARN, der die AWS IoT Schreibberechtigung für Ihre Logs in CloudWatch Logs gewährt. Role-arn Für die Ersteinrichtung ist eine Konfiguration erforderlich.

--default-log-level

Die zu verwendende Protokollstufe. Gültige Werte sind: `ERROR`,`WARN`,`INFO`,`DEBUG`, oder`DISABLED`. Default-log-level Für die Ersteinrichtung ist eine Konfiguration erforderlich.

--no-disable-all-logs

Ein optionaler Parameter, der die gesamte AWS IoT Protokollierung ermöglicht. Verwenden Sie diesen Parameter zum Aktivieren der Protokollierung, wenn sie derzeit deaktiviert ist.

--disable-all-logs

Ein optionaler Parameter, der die gesamte AWS IoT Protokollierung deaktiviert. Verwenden Sie diesen Parameter zum Deaktivieren der Protokollierung, wenn sie derzeit aktiviert ist.

--event-configurations

Dieser Parameter ist optional und ermöglicht es Ihnen, die Protokollierungseinstellungen für einzelne Ereignistypen anzupassen:

- EventType: Der Ereignistyp, der die Einstellung auf Kontoebene überschreibt
- LogLevel: Überschreiben Sie die Einstellung auf Kontoebene mit DEBUG, INFO, ERROR, WARN oder DISABLED
- LogDestination: Geben Sie eine benutzerdefinierte CloudWatch Protokollgruppe für die Protokollzustellung an

Sie können die Protokollierungsstufe und das Protokollziel für jeden Ereignistyp unabhängig voneinander konfigurieren. Wenn nicht angegeben, übernehmen Ereignisse die Einstellungen auf Kontoebene

```
aws iot set-v2-logging-options \
```

```
    --event-configurations "[{\"eventType\":\"Publish-In\",\"logLevel\":\"INFO\",
\"logDestination\":\"examplePublishInLogGroup\"}]"
```

2. Verwenden Sie den Befehl get-v2-logging-options, um Ihre aktuellen Protokollierungsoptionen abzurufen.

```
aws iot get-v2-logging-options \
    --verbose
```

Wobei:

--verbose

Ein optionaler Parameter, mit dem Sie alle EventTypes und ihre Konfigurationen abrufen können.

Nachdem Sie die Protokollierung aktiviert haben, finden Sie unter Anzeigen AWS IoT loggt sich in der CloudWatch Konsole ein Informationen zum Anzeigen der Protokolleinträge.

> **Note**
>
> AWS IoT unterstützt weiterhin ältere Befehle (set-logging-optionsundget-logging-options), mit denen Sie die globale Protokollierung für Ihr Konto einrichten und einrichten können. Beachten Sie, dass bei Verwendung dieser Befehle die ausgegebenen Protokolle nur Klartext enthalten und keine JSON-Nutzlasten. Zudem ist die Protokollierungslatenz in der Regel höher. Es werden keine weiteren Verbesserungen an der Implementierung dieser älteren Befehle vorgenommen. Wir empfehlen, dass Sie die „v2"-Versionen verwenden, um Ihre Protokollierungsoptionen zu konfigurieren und, wenn möglich, ältere Anwendungen zu ändern, die die älteren Versionen verwenden.

Konfigurieren Sie Resource-specific Überschreibungen in AWS IoT (CLI)

In diesem Abschnitt wird beschrieben, wie Sie Resource-specific Overrides für AWS IoT mithilfe der CLI konfigurieren. Resource-specific Overrides ermöglicht es Ihnen, eine Protokollierungsstufe für eine bestimmte Ressource anzugeben, die durch eine Dinggruppe, Client-ID, Quell-IP oder Prinzipal-ID identifiziert wird. Wenn die ressourcenspezifische Protokollierung aktiviert ist, überschreibt sie sowohl Einstellungen auf Konto- als auch auf Veranstaltungsebene. Alle Ereignistypen generieren

Protokolle für die angegebene Ressource auf der konfigurierten Protokollierungsebene, auch wenn diese Ereignisse in den Konfigurationen auf Ereignisebene deaktiviert sind.

Objektgruppen können andere Objektgruppen enthalten, um eine hierarchische Beziehung zu erstellen. In diesem Verfahren wird beschrieben, wie die Protokollierung einer einzelnen Objektgruppe konfiguriert wird. Sie können dieses Verfahren auf die übergeordnete Objektgruppe in einer Hierarchie anwenden, um die Protokollierung für alle Objektgruppen in der Hierarchie zu konfigurieren. Sie können dieses Verfahren auch auf eine untergeordnete Objektgruppe anwenden, um die Protokollierungskonfiguration des übergeordneten Elements zu überschreiben.

Ein Ding kann Mitglied einer Dinggruppe sein. Diese Mitgliedschaft ermöglicht es dem Ding, Konfigurationen, Richtlinien und Einstellungen zu erben, die auf die Dinggruppe angewendet wurden. Dinggruppen werden verwendet, um Einstellungen für mehrere Dinge gemeinsam zu verwalten und auf diese anzuwenden, anstatt jedes Ding einzeln zu behandeln. Wenn Ihre Client-ID mit dem Namen der Sache übereinstimmt, AWS IoT Core wird die Client-Sitzung automatisch der entsprechenden Ding-Ressource zugeordnet. Dadurch kann die Client-Sitzung die Konfigurationen und Einstellungen erben, die auf die Dinggruppen angewendet wurden, zu denen das Ding gehört, einschließlich der Protokollierungsstufen. Wenn Ihre Client-ID nicht mit dem Dingnamen übereinstimmt, können Sie den exklusiven Ding-Anhang aktivieren, um die Zuordnung herzustellen. Weitere Informationen finden Sie unter ???.

Neben Objektgruppen können Sie auch Ziele wie die Client-ID, die Quell-IP und die Prinzipal-ID eines Geräts protokollieren.

> **Note**
>
> Sie benötigen den Amazon-Ressourcennamen (ARN) der Rolle, die Sie verwenden möchten. Wenn Sie eine Rolle für die Protokollierung erstellen müssen, beachten Sie zunächst Erstellen einer Protokollierungsrolle, bevor Sie weitermachen.
> Der zum Aufrufen der API verwendete Prinzipal benötigt Rollenberechtigungen weitergeben für Ihre Protokollierungsrolle.

Sie können dieses Verfahren auch mit der API ausführen, indem Sie die Methoden in der AWS API verwenden, die den hier gezeigten CLI-Befehlen entsprechen.

Um die CLI zum Konfigurieren von Resource-specific Überschreibungen für zu verwenden AWS IoT

1. Aktivieren Sie die Protokollierung auf Kontoebene, bevor Sie die ressourcenspezifische Protokollierung mit dem folgenden Befehl konfigurieren: Befehl aws iot set-v2-logging-options
2. Verwenden set-v2-logging-level Sie den Befehl Resource-specific, um Overrides zu konfigurieren. Sehen Sie sich das folgende Beispiel für die Konfiguration von Dinggruppen an:

```
aws iot set-v2-logging-level \
              --log-target targetType=THING_GROUP,targetName=thing_group_name \
              --log-level log_level
```

--log-target

Typ und Name der Ressource, für die Sie die Protokollierung konfigurieren. Der Wert `targetType` muss einer der folgenden sein: `THING_GROUP` | `CLIENT_ID` | `SOURCE_IP` | `PRINCIPAL_ID`. Der Wert des Parameters log-target kann Text (wie im Befehlsbeispiel oben) oder eine JSON-Zeichenfolge wie im folgenden Beispiel sein.

```
aws iot set-v2-logging-level \
              --log-target '{"targetType": "THING_GROUP","targetName":
 "thing_group_name"}' \
              --log-level log_level
```

--log-level

Die Protokollierungsebene, die beim Generieren von Protokollen für die angegebene Ressource verwendet wird. Gültige Werte: DEBUG, INFO, ERROR, WARN und DISABLED.

3. Verwenden Sie den Befehl list-v2-logging-levels, um die aktuell konfigurierten Protokollierungsstufen aufzulisten.

```
aws iot list-v2-logging-levels
```

4. Verwenden Sie den Befehl delete-v2-logging-level, um eine ressourcenspezifische Protokollierungsstufe zu löschen, ähnlich dem folgenden Beispiel.

```
aws iot delete-v2-logging-level \
              --target-type "THING_GROUP" \
              --target-name "thing_group_name"
```

```
aws iot delete-v2-logging-level \
              --target-type=CLIENT_ID
              --target-name=ClientId1
```

--target-type

Der Wert `target-type` muss einer der folgenden sein: `THING_GROUP` | `CLIENT_ID` | `SOURCE_IP` | `PRINCIPAL_ID`.

--target-name

Der Name der Objektgruppe, für welche die Protokollierungsstufe entfernt werden soll.

Protokollstufen

Diese Protokollstufen bestimmen die Ereignisse, die protokolliert werden und die für Standard- und ressourcenspezifische Protokollstufen berücksichtigt werden.

ERROR

Jeder Fehler, der bewirkt, dass ein Vorgang fehlschlägt.

Beispiel: Das Gerät konnte aufgrund eines abgelaufenen Zertifikats nicht authentifiziert werden.

Protokolle enthalten ausschließlich Fehlerinformationen der Kategorie ERROR.

WARN

Alles, was zu Inkonsistenzen im System führen kann, aber möglicherweise nicht zum Fehlschlagen der Operation führt.

Beispiel: Annäherung an das Limit für die Nachrichtenrate.

Protokolle enthalten Informationen der Kategorien ERROR und WARN.

INFO

High-level Informationen über den Fluss der Dinge.

Beispiel: Ein Kunde hat erfolgreich ein MQTT-Thema abonniert.

Protokolle enthalten Informationen der Kategorien INFO, ERROR und WARN.

DEBUG

Informationen, die bei der Problembehebung hilfreich sein können.

Beispiel: Die IoT Rules Engine hat eine Nachricht erkannt, die unter dem Regelthema „rule/test“ veröffentlicht wurde, und die Ausführung wurde erfolgreich gestartet. Die Regel ist mit konfiguriert RepublishAction.

Protokolle enthalten Informationen der Kategorien DEBUG, INFO, ERROR und WARN.

DISABLED (DEAKTIVIERT)

Die gesamte Protokollierung ist deaktiviert.

Überwachen AWS IoT Alarme und Metriken mithilfe von Amazon CloudWatch

Sie können den AWS IoT Einsatz überwachen CloudWatch, der Rohdaten sammelt und verarbeitet, um sie AWS IoT in lesbare Metriken nahezu in Echtzeit umzuwandeln. Diese Statistiken werden für einen Zeitraum von zwei Wochen aufgezeichnet, damit Sie auf Verlaufsinformationen zugreifen können und einen besseren Überblick darüber erhalten, wie Ihre Webanwendung oder der Service ausgeführt werden. Standardmäßig werden AWS IoT Metrikdaten automatisch CloudWatch in Intervallen von einer Minute an diese gesendet. Weitere Informationen finden Sie unter Was sind Amazon CloudWatch, Amazon CloudWatch Events und Amazon CloudWatch Logs? im CloudWatch Amazon-Benutzerhandbuch.

Verwenden AWS IoT Metriken

Die von gemeldeten Metriken enthalten AWS IoT Informationen, die Sie auf unterschiedliche Weise analysieren können. Die folgenden Anwendungsfälle basieren auf einem Szenario, bei dem zehn Elemente einmal am Tag eine Verbindung zum Internet herstellen. Jeden Tag:

- Zehn Dinge sind ungefähr AWS IoT zur gleichen Zeit miteinander verbunden.
- Jedes Objekt meldet sich bei einem Themenfilter an und wartet eine Stunde, bevor des die Verbindung wieder trennt. Während dieser Zeit kommunizieren die Elemente miteinander und erfahren mehr über den Status der Welt.
- Jedes Objekt veröffentlicht eine Sichtweise, die es basierend auf den mit `UpdateThingShadow` neu erhaltenen Daten gebildet hat.

- Jedes Ding trennt sich von AWS IoT.

Um Ihnen den Einstieg zu erleichtern, werden in diesen Themen einige der Fragen behandelt, die Sie möglicherweise haben.

- Wie werde ich benachrichtigt, wenn meine Objekte nicht jeden Tag erfolgreich verbunden werden?
- Wie werde ich benachrichtigt, wenn meine Objekte nicht jeden Tag Daten veröffentlichen?
- Wie werde ich benachrichtigt, wenn die Schattenaktualisierungen meines Objekts jeden Tag abgelehnt werden?
- Wie kann ich einen CloudWatch Alarm für Jobs erstellen?

Mehr über CloudWatch Alarme und Metriken

- Zu CloudWatch überwachende Alarme erstellen AWS IoT
- AWS IoT Metriken und Dimensionen

Zu CloudWatch überwachende Alarme erstellen AWS IoT

Sie können einen CloudWatch Alarm erstellen, der eine Amazon SNS-Nachricht sendet, wenn sich der Status des Alarms ändert. Ein Alarm überwacht eine Metrik über einen bestimmten, von Ihnen festgelegten Zeitraum. Wenn der Wert der Metrik über eine Reihe von Zeiträumen einen bestimmten Schwellenwert überschreitet, wird mindestens eine Aktion ausgeführt. Die Aktion kann eine Benachrichtigung sein, die an ein Amazon-SNS-Thema oder eine Auto Scaling-Richtlinie gesendet wird. Alarme lösen Aktionen nur bei anhaltenden Zustandsänderungen aus. CloudWatch Alarme lösen keine Aktionen aus, nur weil sie sich in einem bestimmten Zustand befinden. Der Zustand muss sich geändert haben und für eine bestimmte Anzahl von Zeiträumen beibehalten worden sein.

In den folgenden Themen werden einige Beispiele zur Verwendung von CloudWatch-Alarmen beschrieben.

- Wie werde ich benachrichtigt, wenn meine Objekte nicht jeden Tag erfolgreich verbunden werden?
- Wie werde ich benachrichtigt, wenn meine Objekte nicht jeden Tag Daten veröffentlichen?
- Wie werde ich benachrichtigt, wenn die Schattenaktualisierungen meines Objekts jeden Tag abgelehnt werden?
- Wie kann ich einen CloudWatch Alarm für Jobs erstellen?

Sie können alle Metriken sehen, die CloudWatch Alarme überwachen könnenAWS IoT Metriken und Dimensionen.

Wie werde ich benachrichtigt, wenn meine Objekte nicht jeden Tag erfolgreich verbunden werden?

1. Erstellen Sie ein Amazon-SNS-Thema mit dem Namen `things-not-connecting-successfully` und zeichnen Sie dessen Amazon-Ressourcennamen (ARN) auf. In diesem Verfahren wird der ARN des Themas als *`sns-topic-arn`* angegeben.

   Weitere Informationen zum Erstellen einer Amazon-SNS-Benachrichtigung finden Sie unter Erste Schritte mit Amazon SNS.

2. Erstellen Sie den Alarm.

```
aws cloudwatch put-metric-alarm \
    --alarm-name ConnectSuccessAlarm \
    --alarm-description "Alarm when my Things don't connect successfully" \
    --namespace AWS/IoT \
    --metric-name Connect.Success \
    --dimensions Name=Protocol,Value=MQTT \
    --statistic Sum \
    --threshold 10 \
    --comparison-operator LessThanThreshold \
    --period 86400 \
    --evaluation-periods 1 \
    --alarm-actions sns-topic-arn
```

3. Testen Sie den Alarm.

```
aws cloudwatch set-alarm-state --alarm-name ConnectSuccessAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name ConnectSuccessAlarm --state-reason
 "initializing" --state-value ALARM
```

4. Stellen Sie sicher, dass der Alarm in der CloudWatch -Konsole angezeigt wird.

Wie werde ich benachrichtigt, wenn meine Objekte nicht jeden Tag Daten veröffentlichen?

1. Erstellen Sie ein Amazon-SNS-Thema mit dem Namen `things-not-publishing-data` und zeichnen Sie dessen Amazon-Ressourcennamen (ARN) auf. In diesem Verfahren wird der ARN des Themas als *`sns-topic-arn`* angegeben.

   Weitere Informationen zum Erstellen einer Amazon-SNS-Benachrichtigung finden Sie unter Erste Schritte mit Amazon SNS.

2. Erstellen Sie den Alarm.

```
aws cloudwatch put-metric-alarm \
    --alarm-name PublishInSuccessAlarm\
    --alarm-description "Alarm when my Things don't publish their data \
    --namespace AWS/IoT \
    --metric-name PublishIn.Success \
    --dimensions Name=Protocol,Value=MQTT \
    --statistic Sum \
    --threshold 10 \
    --comparison-operator LessThanThreshold \
    --period 86400 \
    --evaluation-periods 1 \
    --alarm-actions sns-topic-arn
```

3. Testen Sie den Alarm.

```
aws cloudwatch set-alarm-state --alarm-name PublishInSuccessAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name PublishInSuccessAlarm --state-reason
 "initializing" --state-value ALARM
```

4. Stellen Sie sicher, dass der Alarm in der CloudWatch -Konsole angezeigt wird.

Wie werde ich benachrichtigt, wenn die Schattenaktualisierungen meines Objekts jeden Tag abgelehnt werden?

1. Erstellen Sie ein Amazon-SNS-Thema mit dem Namen `things-shadow-updates-rejected` und zeichnen Sie dessen Amazon-Ressourcennamen (ARN) auf. In diesem Verfahren wird der ARN des Themas als *sns-topic-arn* angegeben.

   Weitere Informationen zum Erstellen einer Amazon-SNS-Benachrichtigung finden Sie unter Erste Schritte mit Amazon SNS.
2. Erstellen Sie den Alarm.

```
aws cloudwatch put-metric-alarm \
    --alarm-name UpdateThingShadowSuccessAlarm \
    --alarm-description "Alarm when my Things Shadow updates are getting rejected"
 \
    --namespace AWS/IoT \
    --metric-name UpdateThingShadow.Success \
    --dimensions Name=Protocol,Value=MQTT \
    --statistic Sum \
    --threshold 10 \
    --comparison-operator LessThanThreshold \
    --period 86400 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions sns-topic-arn
```

3. Testen Sie den Alarm.

```
aws cloudwatch set-alarm-state --alarm-name UpdateThingShadowSuccessAlarm --state-
reason "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name UpdateThingShadowSuccessAlarm --state-
reason "initializing" --state-value ALARM
```

4. Stellen Sie sicher, dass der Alarm in der CloudWatch -Konsole angezeigt wird.

Wie kann ich einen CloudWatch Alarm für Jobs erstellen?

Der Job-Service stellt CloudWatch Kennzahlen zur Verfügung, mit denen Sie Ihre Jobs überwachen können. Sie können CloudWatch Alarme erstellen, um alle zu überwachenJobs-Metriken.

Der folgende Befehl erzeugt einen CloudWatch Alarm, um die Gesamtzahl der fehlgeschlagenen Auftragsausführungen für Job zu überwachen, *SampleOTAJob* und benachrichtigt Sie, wenn mehr als 20 Auftragsausführungen fehlgeschlagen sind. Der Alarm überwacht die Jobs-Metrik `FailedJobExecutionTotalCount`, indem der gemeldete Wert alle 300 Sekunden überprüft wird. Er wird aktiviert, wenn ein einzelner gemeldeter Wert größer als 20 ist, was bedeutet, dass seit dem Start des Auftrags mehr als 20 fehlgeschlagene Auftragsausführungen aufgetreten sind. Wenn der Alarm auslöst, wird eine Benachrichtigung an das angegebene Amazon–SNS-Thema gesendet.

```
aws cloudwatch put-metric-alarm \
    --alarm-name TotalFailedJobExecution-SampleOTAJob \
    --alarm-description "Alarm when total number of failed job execution exceeds the
 threshold for SampleOTAJob" \
    --namespace AWS/IoT \
    --metric-name FailedJobExecutionTotalCount \
    --dimensions Name=JobId,Value=SampleOTAJob \
    --statistic Sum \
    --threshold 20 \
    --comparison-operator GreaterThanThreshold \
    --period 300 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:<AWS_REGION>:<AWS_ACCOUNT_ID>:SampleOTAJob-has-too-
many-failed-job-ececutions
```

Der folgende Befehl erzeugt einen CloudWatch Alarm, um die Anzahl der fehlgeschlagenen Auftragsausführungen für Job *SampleOTAJob* in einem bestimmten Zeitraum zu überwachen. Sie werden dann benachrichtigt, wenn mehr als fünf Auftragsausführungen in diesem Zeitraum fehlgeschlagen sind. Der Alarm überwacht die Jobs-Metrik `FailedJobExecutionCount`, indem der gemeldete Wert alle 3600 Sekunden überprüft wird. Sie wird aktiviert, wenn ein einzelner gemeldeter Wert größer als 5 ist, was bedeutet, dass in der letzten Stunde mehr als 5 fehlgeschlagene Auftragsausführungen aufgetreten sind. Wenn der Alarm auslöst, wird eine Benachrichtigung an das angegebene Amazon–SNS-Thema gesendet.

```
aws cloudwatch put-metric-alarm \
    --alarm-name FailedJobExecution-SampleOTAJob \
    --alarm-description "Alarm when number of failed job execution per hour exceeds the
 threshold for SampleOTAJob" \
    --namespace AWS/IoT \
    --metric-name FailedJobExecutionCount \
    --dimensions Name=JobId,Value=SampleOTAJob \
```

```
    --statistic Sum \
    --threshold 5 \
    --comparison-operator GreaterThanThreshold \
    --period 3600 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:<AWS_REGION>:<AWS_ACCOUNT_ID>:SampleOTAJob-has-too-
many-failed-job-ececutions-per-hour
```

AWS IoT Metriken und Dimensionen

Wenn Sie mit interagieren AWS IoT, sendet der Service Metriken und Dimensionen an CloudWatch jede Minute. Sie können diese Metriken verwenden AWS IoT, die CloudWatch Konsole verwenden oder AWS CLI sich diese ansehen.

Um Metriken mithilfe der CloudWatch Konsole anzuzeigen, öffnen Sie die CloudWatch Konsole. Wählen Sie im Navigationsbereich Metriken und dann Alle Metriken aus. Suchen Sie auf der Registerkarte „Durchsuchen“ nach, AWS IoT um die Liste der Metriken anzuzeigen. Metriken werden zunächst nach dem Service-Namespace und anschließend nach den verschiedenen Dimensionskombinationen in den einzelnen Namespaces gruppiert.

Führen Sie den folgenden Befehl aus AWS CLI, um Metriken mithilfe von anzuzeigen.

```
aws cloudwatch list-metrics --namespace "AWS/IoT"
```

CloudWatch zeigt die folgenden Gruppen von Metriken an für AWS IoT:

- AWS IoT Metriken
- AWS IoT Core Metriken für Anbieter von Anmeldeinformationen
- Authentifizierungsmetriken
- OCSP-Stapling-Metriken für Serverzertifikate
- Regelmetriken
- Regelaktionsmetriken
- Aktionsspezifische HTTP-Metriken
- Message Broker-Metriken
- Geräteschatten-Metriken
- Metriken protokollieren

- Jobs-Metriken
- Metriken für Device Defender-Prüfung
- Metriken für Device Defender-Erkennung
- Gerätebereitstellungsmetriken
- LoRaWAN-Metriken
- Metriken zur Flottenindizierung
- Dimensionen für Metriken

AWS IoT Metriken

Metrik	Description
`AddThingToDynamicThingGroupsFailed`	Die Anzahl der Fehlerereignisse, die mit dem Hinzufügen eines Objekts zu einer dynamischen Objektgruppe verbunden sind. Die Dimension `DynamicThingGroupName` enthält den Namen der dynamischen Gruppen, die Objekte nicht hinzufügen konnten.
`NumLogBatchesFailedToPublishThrottled`	Der einzelne Stapel der Protokollereignisse, der aufgrund von Drosselungsfehlern nicht veröffentlicht wurde.
`NumLogEventsFailedToPublishThrottled`	Die Anzahl der Protokollereignisse innerhalb des Stapels, die aufgrund von Drosselungsfehlern nicht veröffentlicht wurden.

AWS IoT Core Metriken für Anbieter von Anmeldeinformationen

Metrik	Description
`CredentialExchangeSuccess`	Die Anzahl der erfolgreichen `AssumeRoleWithCertificate` -Anforderungen an den Anbieter der AWS IoT Core Anmeldeinformationen.

Authentifizierungsmetriken

> **Note**
>
> Die Authentifizierungsmetriken werden in der CloudWatch Konsole unter Protokollmetriken angezeigt.

Metrik	Description
`Connection.AuthNError`	Die Anzahl der Verbindungsversuche, die aufgrund von Authentifizierungsfehlern AWS IoT Core abgelehnt werden. Diese Metrik berücksichtigt nur Verbindungen, die eine SNI-Zeichenfolge (Server Name Indication) senden, die einem Endpunkt Ihres entspricht. AWS-Konto Diese Metrik umfasst Verbindungsversuche aus externen Quellen wie Internet-Scantools oder Sondierungsaktivitäten. Die `Protocol` Dimension enthält das Protokoll, das zum Senden des Verbindungsversuchs verwendet wurde.

OCSP-Stapling-Metriken für Serverzertifikate

Metrik	Description
RetrieveOCSPStapleData.Success	Die OCSP-Antwort wurde empfangen und erfolgrei ch verarbeitet. Diese Antwort wird beim TLS-Hands hake für die konfigurierte Domain berücksichtigt. Die `DomainConfigurationName` Dimension enthält den Namen der konfigurierten Domain mit aktiviertem OCSP-Stapling für Serverzertifikate.

Regelmetriken

Metrik	Description
`ParseError`	Die Anzahl der JSON-Parsing-Fehler in Nachrichten zu einem Thema, das von einer Regel überwacht wird. Die Dimension `RuleName` enthält den Namen der Regel.
`RuleExecutionThrottled`	Die Anzahl der Nachrichten, die von der Regel-Engine aufgrund von bösartigem Verhalten gedrosselt wurden, z. B. aufgrund übermäßiger Ingest-Veröffentlichungsanfragen aufgrund nicht vorhandener Regeln. Die Dimension `RuleName` enthält den Namen der auszulösenden Regel.
`RuleEvaluation (Status=Throttled)`	Die Anzahl der Regelbewertungen, die gedrosselt wurden. Die `Status` Dimension gibt den Status der Anforderung an. Derzeit wird nur der Status unterstützt, der darauf hinweist`Throttled` , dass die Anfrage gedrosselt wurde.
`RuleNotFound`	Die auszulösende Regel konnte nicht gefunden werden. Die Dimension `RuleName` enthält den Namen der Regel.
`RulesExecuted`	Die Anzahl der ausgeführten AWS IoT Regeln.
`TopicMatch`	Die Anzahl der eingehenden Nachrichten zu einem Thema, das von einer Regel überwacht wird. Die Dimension `RuleName` enthält den Namen der Regel.

Regelaktionsmetriken

Metrik	Description
`Failure`	Die Anzahl fehlgeschlagener Aufrufe von Regelaktionen. Die Dimension `RuleName` enthält den Namen der Regel, welche die Aktion vorgibt. Die Dimension `ActionType` enthält den Aktionstyp, der aufgerufen wurde.
`Success`	Die Anzahl erfolgreicher Aufrufe von Regelaktionen. Die Dimension `RuleName` enthält den Namen der Regel, welche die Aktion vorgibt. Die Dimension `ActionType` enthält den Aktionstyp, der aufgerufen wurde.
`ErrorActionFailure`	Die Anzahl der fehlgeschlagenen Fehleraktionen. Die Dimension `RuleName` enthält den Namen der Regel, welche die Aktion vorgibt. Die Dimension `ActionType` enthält den Aktionstyp, der aufgerufen wurde.
`ErrorActionSuccess`	Die Anzahl der erfolgreichen Fehleraktionen. Die Dimension `RuleName` enthält den Namen der Regel, welche die Aktion vorgibt. Die Dimension `ActionType` enthält den Aktionstyp, der aufgerufen wurde.

Aktionsspezifische HTTP-Metriken

Metrik	Description
`HttpCode_Other`	Wird generiert, wenn der Statuscode der Antwort vom Downstream-Web nicht 2xx, 4xx oder 5xx service/application lautet.

Metrik	Description
`HttpCode_4XX`	Wird generiert, wenn der Statuscode der Antwort aus dem Downstream-Web zwischen 400 und 499 service/application liegt.
`HttpCode_5XX`	Wird generiert, wenn der Statuscode der Antwort vom Downstream-Web zwischen 500 und 599 service/application liegt.
`HttpInvalidUrl`	Wird generiert, wenn eine Endpunkt-URL nach dem Ersetzen von Ersetzungsvorlagen nicht mit `https://` beginnt.
`HttpRequestTimeout`	Wird generiert, wenn das Downstream-Web service/ application keine Antwort innerhalb des Zeitlimits für die Anforderung zurückgibt. Weitere Informationen finden Sie unter Service Quotas.
`HttpUnknownHost`	Wird generiert, wenn zwar die URL gültig ist, der Service jedoch nicht vorhanden oder nicht erreichbar ist.

Message Broker-Metriken

> **Note**
>
> Die Message Broker-Metriken werden in der CloudWatch Konsole unter Protocol Metrics angezeigt.

Metrik	Description
`Connect.AuthError`	Die Anzahl der Verbindungsanforderungen, die vom Message Broker nicht autorisiert werden konnten. Die Dimension `Protocol` enthält das zum Senden der `CONNECT`-Mitteilung verwendete Protokoll.

Metrik	Description
`Connect.ClientError`	Die Anzahl der Verbindungsanfragen, die abgelehnt wurden, weil die MQTT-Nachricht die unter definierten Anforderungen nicht AWS IoT Kontingente erfüllte. Die Dimension `Protocol` enthält das zum Senden der `CONNECT`-Mitteilung verwendete Protokoll.
`Connect.ClientIDThrottle`	Die Anzahl der Verbindungsanforderungen, die gedrosselt wurden, weil der Client die zugelassene Verbindungsanforderungsrate für eine bestimmte Client-ID überschritten hat. Die Dimension `Protocol` enthält das zum Senden der `CONNECT`-Mitteilung verwendete Protokoll.
`Connect.ServerError`	Die Anzahl der Verbindungsanforderungen, die aufgrund eines internen Fehlers fehlgeschlagen sind. Die Dimension `Protocol` enthält das zum Senden der `CONNECT`-Mitteilung verwendete Protokoll.
`Connect.Success`	Die Anzahl erfolgreicher Verbindungen zum Message Broker. Die Dimension `Protocol` enthält das zum Senden der `CONNECT`-Mitteilung verwendete Protokoll.
`Connect.Throttle`	Die Anzahl der Verbindungsanforderungen, die gedrosselt wurden, weil das Konto die zugelassene Verbindungsanforderungsrate überschritten hat. Die Dimension `Protocol` enthält das zum Senden der `CONNECT`-Mitteilung verwendete Protokoll.
`Ping.Success`	Die Anzahl der vom Message Broker empfangenen Ping-Meldungen. Die Dimension `Protocol` enthält das zum Senden der Ping-Meldung verwendete Protokoll.

Metrik	Description
`PublishIn.AuthError`	Die Anzahl der Veröffentlichungsanforderungen, die vom Message Broker nicht autorisiert werden konnten. Die Dimension `Protocol` enthält das zum Veröffentlichen der Mitteilung verwendete Protokoll. HTTP Publish unterstützt diese Metrik nicht.
`PublishIn.ClientError`	Die Anzahl der Veröffentlichungsanfragen, die vom Message Broker abgelehnt wurden, weil die Nachricht die unter definierten Anforderungen nicht AWS IoT Kontingente erfüllte. Die Dimension `Protocol` enthält das zum Veröffentlichen der Mitteilung verwendete Protokoll.
`PublishIn.ServerError`	Die Anzahl der Veröffentlichungsanforderungen, die vom Message Broker aufgrund eines internen Fehlers nicht verarbeitet werden konnten. Die Dimension `Protocol` enthält das zum Senden der `PUBLISH`-Mitteilung verwendete Protokoll.
`PublishIn.Success`	Die Anzahl der Veröffentlichungsanforderungen, die vom Message Broker erfolgreich verarbeitet wurden. Die Dimension `Protocol` enthält das zum Senden der `PUBLISH`-Mitteilung verwendete Protokoll.
`PublishIn.Throttle`	Die Anzahl der Veröffentlichungsanforderungen, die gedrosselt wurden, weil der Client die zugelassene Rate für eingehende Mitteilungen überschritten hat. Die Dimension `Protocol` enthält das zum Senden der `PUBLISH`-Mitteilung verwendete Protokoll. HTTP Publish unterstützt diese Metrik nicht.
`PublishOut.AuthError`	Die Anzahl der vom Message Broker vorgenomm enen Veröffentlichungsanforderungen, die von AWS IoT nicht autorisiert werden konnten. Die Dimension `Protocol` enthält das zum Senden der `PUBLISH`-Mitteilung verwendete Protokoll.

Metrik	Description
`PublishOut.ClientError`	Die Anzahl der vom Message Broker gestellten Veröffentlichungsanfragen, die abgelehnt wurden, weil die Nachricht die unter definierten Anforderungen nicht erfüllteAWS IoT Kontingente. Die Dimension `Protocol` enthält das zum Senden der `PUBLISH`-Mitteilung verwendete Protokoll.
`PublishOut.ServerError`	Die Anzahl der Veröffentlichungsanforderungen, die vom Message Broker aufgrund eines internen Fehlers nicht verarbeitet werden konnten. Die Dimension `Protocol` enthält das zum Senden der `PUBLISH`-Mitteilung verwendete Protokoll.
`PublishOut.Success`	Die Anzahl der vom Message Broker erfolgreich vorgenommenen Veröffentlichungsanforderungen. Die Dimension `Protocol` enthält das zum Senden der `PUBLISH`-Mitteilung verwendete Protokoll.
`PublishOut.Throttle`	Die Anzahl der Veröffentlichungsanforderungen, die gedrosselt wurden, weil der Client die zugelassene Rate für ausgehende Mitteilungen überschritten hat. Die Dimension `Protocol` enthält das zum Senden der `PUBLISH`-Mitteilung verwendete Protokoll.
`PublishRetained.AuthError`	Die Anzahl der Veröffentlichungsanforderungen mit dem aktivierten `RETAIN`-Flag, die vom Message Broker nicht autorisiert werden konnten. Die Dimension `Protocol` enthält das zum Senden der `PUBLISH`-Mitteilung verwendete Protokoll.
`PublishRetained.ServerError`	Die Anzahl der Veröffentlichungsanforderungen, die vom Message Broker aufgrund eines internen Fehlers nicht verarbeitet werden konnten. Die Dimension `Protocol` enthält das zum Senden der `PUBLISH`-Mitteilung verwendete Protokoll.

Metrik	Description
`PublishRetained.Success`	Die Anzahl der Veröffentlichungsanforderungen mit dem aktiven `RETAIN`-Flag, die vom Message Broker erfolgreich verarbeitet wurden. Die Dimension `Protocol` enthält das zum Senden der `PUBLISH`-Mitteilung verwendete Protokoll.
`PublishRetained.Throttle`	Die Anzahl der Veröffentlichungsanforderungen mit dem aktiven `RETAIN`-Flag, die gedrosselt wurden, weil der Client die zugelassene Rate für eingehend e Mitteilungen überschritten hat. Die Dimension `Protocol` enthält das zum Senden der `PUBLISH`-Mitteilung verwendete Protokoll.
`Queued.Success`	Die Anzahl der gespeicherten Nachrichten, die vom Message Broker erfolgreich verarbeitet wurden, für Clients, die von ihrer persistenten Sitzung getrennt wurden. Nachrichten mit einer QoS von 1 werden gespeichert, während ein Client mit einer persisten ten Sitzung getrennt wird.
`Queued.Throttle`	Die Anzahl der Nachrichten, die nicht gespeichert werden konnten und gedrosselt wurden, während Clients mit persistenten Sitzungen getrennt wurden. Dies tritt auf, wenn Clients das Limit für Nachricht en in der Warteschlange pro Sekunde pro Konto überschreiten. Nachrichten mit einer QoS von 1 werden gespeichert, während ein Client mit einer persistenten Sitzung getrennt wird.
`Queued.ServerError`	Die Anzahl der Nachrichten, die aufgrund eines internen Fehlers für eine persistente Sitzung nicht gespeichert wurden. Wenn Clients mit einer persisten ten Sitzung getrennt werden, dann werden Nachricht en mit einer Servicegüte (QoS) von 1 gespeichert.

Metrik	Description
`Subscribe.AuthError`	Die Anzahl der von einem Client vorgenommenen Abonnementanforderungen, die nicht autorisiert werden konnten. Die Dimension `Protocol` enthält das zum Senden der `SUBSCRIBE` -Mitteilung verwendete Protokoll.
`Subscribe.ClientError`	Die Anzahl der Abonnementanfragen, die abgelehnt wurden, weil die `SUBSCRIBE` Nachricht die unter definierten Anforderungen nicht erfüllteAWS IoT Kontingente. Die Dimension `Protocol` enthält das zum Senden der `SUBSCRIBE` -Mitteilung verwendete Protokoll.
`Subscribe.ServerError`	Anzahl der Abonnementanforderungen, die aufgrund eines internen Fehlers abgelehnt wurden. Die Dimension `Protocol` enthält das zum Senden der `SUBSCRIBE` -Mitteilung verwendete Protokoll.
`Subscribe.Success`	Anzahl der Abonnementanforderungen, die vom Message Broker erfolgreich verarbeitet wurden. Die Dimension `Protocol` enthält das zum Senden der `SUBSCRIBE` -Mitteilung verwendete Protokoll.
`Subscribe.Throttle`	Die Anzahl der Abonnementanfragen, die gedrosselt wurden, weil die zulässigen Abonnementanforderungsraten für Sie überschritten wurden. AWS-Konto Zu diesen Grenzwerten gehören Abonnements pro Sekunde pro Konto, Abonnements pro Konto und Abonnements pro Verbindung, die in den Grenzwerten und Kontingenten für AWS IoT Core Message Broker und Protokolle beschrieben werden. Die Dimension `Protocol` enthält das zum Senden der `SUBSCRIBE` -Mitteilung verwendete Protokoll.

Metrik	Description
`Throttle.Exceeded`	Diese Metrik wird angezeigt, CloudWatch wenn ein MQTT-Client hinsichtlich der Grenzwerte für Pakete pro Sekunde pro Verbindung gedrosselt wird. Diese Metrik gilt nicht für HTTP-Verbindungen.
`Unsubscribe.ClientError`	Die Anzahl der Abmeldeanfragen, die abgelehnt wurden, weil die `UNSUBSCRIBE` Nachricht die unter definierten Anforderungen nicht erfüllte. AWS IoT Kontingente Die Dimension `Protocol` enthält das zum Senden der `UNSUBSCRIBE` -Mitteilung verwendete Protokoll.
`Unsubscribe.ServerError`	Anzahl der Abmeldeanforderungen, die aufgrund eines internen Fehlers abgelehnt wurden. Die Dimension `Protocol` enthält das zum Senden der `UNSUBSCRIBE` -Mitteilung verwendete Protokoll.
`Unsubscribe.Success`	Anzahl der Abmeldeanforderungen, die vom Message Broker erfolgreich verarbeitet wurden. Die Dimension `Protocol` enthält das zum Senden der `UNSUBSCRIBE` -Mitteilung verwendete Protokoll.
`Unsubscribe.Throttle`	Anzahl der Abmeldeanforderungen, die abgelehnt wurden, weil der Client die zugelassene Abmeldean forderungsrate überschritten hat. Die Dimension `Protocol` enthält das zum Senden der `UNSUBSCRI BE` -Mitteilung verwendete Protokoll.

Geräteschatten-Metriken

> **Note**
>
> Die Geräteschattenmetriken werden in der CloudWatch Konsole unter Protokollmetriken angezeigt.

Metrik	Description
`DeleteThingShadow.Accepted`	Die Anzahl der erfolgreich verarbeiteten `DeleteThingShadow` -Anforderungen. Die Dimension `Protocol` enthält das zum Senden der Anforderung verwendete Protokoll.
`GetThingShadow.Accepted`	Die Anzahl der erfolgreich verarbeiteten `GetThingShadow` -Anforderungen. Die Dimension `Protocol` enthält das zum Senden der Anforderung verwendete Protokoll.
`ListThingShadow.Accepted`	Die Anzahl der erfolgreich verarbeiteten `ListThingShadow` -Anforderungen. Die Dimension `Protocol` enthält das zum Senden der Anforderung verwendete Protokoll.
`UpdateThingShadow.Accepted`	Die Anzahl der erfolgreich verarbeiteten `UpdateThingShadow` -Anforderungen. Die Dimension `Protocol` enthält das zum Senden der Anforderung verwendete Protokoll.

Metriken protokollieren

Metrik	Description
`CloudwatchLogs:LogGroupCreationFailed`	Die Anzahl der `CreateLogGroup` Anforderungsfehler. AWS IoT Logging erstellt eine CloudWatch Protokollgruppe, falls sie nicht existiert. Die `LogGroup` Dimension enthält die Protokollgruppe, die für die Anfrage verwendet wurde.
`CloudwatchLogs:LogStreamCreationFailed`	Die Anzahl der `CreateLogStream` Anforderungsfehler. AWS IoT Logging erstellt einen CloudWatch Log-Stream, um Log-Ereignisse zu übertragen. Die

Metrik	Description
	`LogGroup` Dimension enthält die Protokollgruppe, die für die Anfrage verwendet wurde.
`CloudwatchLogs:PutRetentionPolicyFailed`	Die Anzahl der `PutRetentionPolicy` Anforderungsfehler. AWS IoT Logging erstellt eine CloudWatch Protokollgruppe, falls sie nicht existiert, und die Aufbewahrungsrichtlinie ist auf 30 Tage festgelegt. Die `LogGroup` Dimension enthält die Protokollgruppe, die für die Anfrage verwendet wurde.
`CloudwatchLogs:PutLogEventsFailed`	Die Anzahl der `PutLogEvents` Anforderungsfehler. Die `LogGroup` Dimension enthält die CloudWatch Protokollgruppe, die für die Anforderung verwendet wurde.

Jobs-Metriken

Metrik	Description
`CanceledJobExecutionCount`	Die Anzahl der Jobausführungen, deren Status sich `CANCELED` innerhalb eines Zeitraums geändert hat, der bestimmt ist durch CloudWatch. (Weitere Informationen zu CloudWatch Metriken finden Sie unter CloudWatch Amazon-Metriken.) Die Dimension `JobId` enthält die ID des Auftrags.
`CanceledJobExecutionTotalCount`	Die Gesamtzahl der Auftragsausführungen mit dem Status `CANCELED` für den jeweiligen Auftrag. Die Dimension `JobId` enthält die ID des Auftrags.
`ClientErrorCount`	Die Anzahl der Clientfehler, die beim Ausführen des Auftrags aufgetreten sind. Die Dimension `JobId` enthält die ID des Auftrags.
`FailedJobExecutionCount`	Die Anzahl der Auftragsausführungen, deren Status sich `FAILED` innerhalb eines Zeitraums geändert

Metrik	Description
	hat, der von CloudWatch bestimmt wird. (Weitere Informationen zu CloudWatch Metriken finden Sie unter Amazon CloudWatch Metrics.) Die Dimension `JobId` enthält die ID des Auftrags.
`FailedJobExecutionTotalCount`	Die Gesamtzahl der Auftragsausführungen mit dem Status `FAILED` für den jeweiligen Auftrag. Die Dimension `JobId` enthält die ID des Auftrags.
`InProgressJobExecutionCount`	Die Anzahl der Auftragsausführungen, deren Status sich `IN_PROGRESS` innerhalb eines Zeitraums geändert hat, der von CloudWatch bestimmt wird. (Weitere Informationen zu CloudWatch Metriken finden Sie unter CloudWatch Amazon-Metriken.) Die Dimension `JobId` enthält die ID des Auftrags.
`InProgressJobExecutionTotalCount`	Die Gesamtzahl der Auftragsausführungen mit dem Status `IN_PROGRESS` für den jeweiligen Auftrag. Die Dimension `JobId` enthält die ID des Auftrags.
`RejectedJobExecutionTotalCount`	Die Gesamtzahl der Auftragsausführungen mit dem Status `REJECTED` für den jeweiligen Auftrag. Die Dimension `JobId` enthält die ID des Auftrags.
`RemovedJobExecutionTotalCount`	Die Gesamtzahl der Auftragsausführungen mit dem Status `REMOVED` für den jeweiligen Auftrag. Die Dimension `JobId` enthält die ID des Auftrags.
`QueuedJobExecutionCount`	Die Anzahl der Auftragsausführungen, deren Status sich `QUEUED` innerhalb eines Zeitraums geändert hat, der von CloudWatch bestimmt wird. (Weitere Informationen zu CloudWatch Metriken finden Sie unter Amazon CloudWatch Metrics.) Die Dimension `JobId` enthält die ID des Auftrags.

Metrik	Description
`QueuedJobExecutionTotalCount`	Die Gesamtzahl der Auftragsausführungen mit dem Status `QUEUED` für den jeweiligen Auftrag. Die Dimension `JobId` enthält die ID des Auftrags.
`RejectedJobExecutionCount`	Die Anzahl der Auftragsausführungen, deren Status sich `REJECTED` innerhalb eines Zeitraums geändert hat, der von CloudWatch bestimmt wird. (Weitere Informationen zu CloudWatch Metriken finden Sie unter CloudWatch Amazon-Metriken.) Die Dimension `JobId` enthält die ID des Auftrags.
`RemovedJobExecutionCount`	Die Anzahl der Auftragsausführungen, deren Status sich `REMOVED` innerhalb eines Zeitraums geändert hat, der von CloudWatch bestimmt wird. (Weitere Informationen zu CloudWatch Metriken finden Sie unter CloudWatch Amazon-Metriken.) Die Dimension `JobId` enthält die ID des Auftrags.
`ServerErrorCount`	Die Anzahl der Serverfehler, die beim Ausführen das Auftrags aufgetreten sind. Die Dimension `JobId` enthält die ID des Auftrags.
`SuccededJobExecutionCount`	Die Anzahl der Auftragsausführungen, deren Status sich `SUCCESS` innerhalb eines Zeitraums geändert hat, der von CloudWatch bestimmt wird. (Weitere Informationen zu CloudWatch Metriken finden Sie unter CloudWatch Amazon-Metriken.) Die Dimension `JobId` enthält die ID des Auftrags.
`SuccededJobExecutionTotalCount`	Die Gesamtzahl der Auftragsausführungen mit dem Status `SUCCESS` für den jeweiligen Auftrag. Die Dimension `JobId` enthält die ID des Auftrags.

Metriken für Device Defender-Prüfung

Metrik	Description
`NonCompliantResources`	Die Anzahl der Ressourcen, die bei einer Prüfung als nicht konform befunden wurden. Das System meldet die Anzahl der Ressourcen, die bei der Prüfung eines durchgeführten Audits nicht konform waren.
`ResourcesEvaluated`	Die Anzahl der Ressourcen, die auf Konformität geprüft wurden. Das System meldet die Anzahl der Ressourcen, die für die Prüfung der durchgeführten Audits ausgewertet wurden.
`MisconfiguredDeviceDefender Notification`	Benachrichtigt Sie, wenn Ihre SNS-Konfiguration für falsch konfiguriert AWS IoT Device Defender ist. Dimensions (Abmessungen)

Metriken für Device Defender-Erkennung

Metrik	Description
`NumOfMetricsExported`	Die Anzahl der Metriken, die für eine Cloud-seitige, geräteseitige oder benutzerdefinierte Metrik exportier t wurden. Das System meldet die Anzahl der für das Konto exportierten Metriken für eine bestimmte Metrik. Diese Metrik ist nur für Kunden verfügbar, die den Metrik-Export verwenden.
`NumOfMetricsSkipped`	Die Anzahl der Metriken, die für eine Cloud-sei tige, geräteseitige oder benutzerdefinierte Metrik übersprungen wurden. Das System meldet die Anzahl der Metriken, die für das Konto übersprun gen wurden, für eine bestimmte Metrik, da Device Defender Detect nicht genügend Berechtigungen zur Veröffentlichung im MQTT-Thema erteilt hat. Diese

Metrik	Description
	Metrik ist nur für Kunden verfügbar, die den Metrik-Export verwenden.
`NumOfMetricsExceedingSizeLimit`	Die Anzahl der Metriken, die beim Export für eine Cloud-seitige, geräteseitige oder benutzerdefinierte Metrik übersprungen wurden, weil die Größe die Größenbeschränkungen für MQTT-Nachrichten überschreitet. Das System meldet die Anzahl der Metriken, die für den Export für das Konto übersprungen wurden, und zwar für eine bestimmte Metrik, weil die Größe die Größenbeschränkungen von MQTT-Nachrichten überschreitet. Diese Metrik ist nur für Kunden verfügbar, die den Metrik-Export verwenden.
`Violations`	Die Anzahl der neuen Verletzungen des Sicherheitsprofils, die seit der letzten Auswertung festgestellt wurden. Das System meldet die Anzahl der neuen Verstöße für das Konto, für ein bestimmtes Sicherheitsprofil und für ein bestimmtes Verhalten eines bestimmten Sicherheitsprofils.
`ViolationsCleared`	Die Anzahl der Verstöße des Sicherheitsprofils, die seit der letzten Auswertung gelöst wurden. Das System meldet die Anzahl der gelösten Verstöße für das Konto, für ein bestimmtes Sicherheitsprofil und für ein bestimmtes Verhalten eines bestimmten Sicherheitsprofils.

Metrik	Description
`ViolationsInvalidated`	Die Anzahl der Verletzungen von Sicherheitsprofile n, für die seit der letzten Auswertung keine Informati onen mehr verfügbar sind (weil das meldende Gerät die Meldung eingestellt hat oder aus irgendeinem Grund nicht mehr überwacht wird). Das System meldet die Anzahl der ungültigen Verstöße für das gesamte Konto, für ein bestimmtes Sicherheitsprofil und für ein bestimmtes Verhalten eines bestimmten Sicherheitsprofils.
`MisconfiguredDeviceDefender Notification`	Benachrichtigt Sie, wenn Ihre SNS-Konfiguration für falsch konfiguriert ist. AWS IoT Device Defender Dimensions (Abmessungen)

Gerätebereitstellungsmetriken

AWS IoT Metriken zur Flottenbereitstellung

Metrik	Description
`ApproximateNumberOfThingsRe gistered`	Die Anzahl der Objekte, die von Flottenbereitstellung registriert wurden. Die Anzahl ist zwar im Allgemeinen korrekt, die verteilte Architektur von AWS IoT Core macht es jedoch schwierig, eine genaue Anzahl der registrie rten Objekte aufrechtzuerhalten. Die nützlichste Statistik für diese Metrik ist: • Max, um die Gesamtzahl der registrierten Objekte zu melden. Eine Anzahl der Dinge, die während des CloudWatch Aggregationsfensters registriert wurden, finden Sie in der `RegisterT hingFailed` Metrik.

Metrik	Description
	Maße: ClaimCertificateId
`CreateKeysAndCertificateFailed`	Die Anzahl der Fehler, die beim Aufrufen der `CreateKeysAndCertificate` -MQTT-API aufgetreten sind. Die Metrik wird sowohl bei Erfolg (Wert = 0) als auch bei Fehlschlag (Wert = 1) ausgegeben. Diese Metrik kann verwendet werden, um die Anzahl der Zertifikate zu verfolgen, die während der CloudWatch-supported Aggregationsfenster erstellt und registriert wurden, z. B. 5 Minuten oder 1 Stunde. Für diese Metrik sind folgende Statistiken verfügbar: • Summe zur Angabe der Anzahl fehlgeschlagener Aufrufe. • SampleCountum die Gesamtzahl der erfolgreichen und fehlgeschlagenen Aufrufe zu melden.
`CreateCertificateFromCsrFailed`	Die Anzahl der Fehler, die beim Aufrufen der `CreateCertificateFromCsr` -MQTT-API aufgetreten sind. Die Metrik wird sowohl bei Erfolg (Wert = 0) als auch bei Fehlschlag (Wert = 1) ausgegeben. Diese Metrik kann verwendet werden, um die Anzahl der Dinge zu verfolgen, die während der CloudWatch-supported Aggregationsfenster registriert wurden, z. B. 5 Minuten oder 1 Stunde. Für diese Metrik sind folgende Statistiken verfügbar: • Summe zur Angabe der Anzahl fehlgeschlagener Aufrufe. • SampleCountum die Gesamtzahl der erfolgreichen und fehlgeschlagenen Aufrufe zu melden.

Metrik	Description
`RegisterThingFailed`	Die Anzahl der Fehler, die beim Aufrufen der `RegisterThing` -MQTT-API aufgetreten sind. Die Metrik wird sowohl bei Erfolg (Wert = 0) als auch bei Fehlschlag (Wert = 1) ausgegeben. Diese Metrik kann verwendet werden, um die Anzahl der Dinge zu verfolgen, die während der CloudWatch-supported Aggregationsfenster registriert wurden, z. B. 5 Minuten oder 1 Stunde. Die Gesamtzahl der registrierten Objekte finden Sie in der `Approxima teNumberOfThingsRegistered` -Metrik. Für diese Metrik sind folgende Statistiken verfügbar: • Summe zur Angabe der Anzahl fehlgeschlagener Aufrufe. • SampleCountum die Gesamtzahl der erfolgreichen und fehlgeschlagenen Aufrufe zu melden. Maße: TemplateName

Just-in-time Metriken für die Bereitstellung

Metrik	Description
`ProvisionThing.ClientError`	Gibt an, wie oft ein Gerät aufgrund eines Client-Fe hlers nicht bereitgestellt werden konnte. Beispiels weise war die in der Vorlage angegebene Richtlinie nicht vorhanden.
`ProvisionThing.ServerError`	Gibt an, wie oft ein Gerät aufgrund eines Serverfeh lers nicht bereitgestellt werden konnte. Kunden können erneut versuchen, das Gerät bereitzustellen, nachdem sie gewartet haben und sie können sich

Metrik	Description
	an AWS IoT wenden, falls das Problem weiterhin besteht.
`ProvisionThing.Success`	Die Anzahl der erfolgreich bereitgestellten Geräte.

LoRaWAN-Metriken

Die folgende Tabelle zeigt die Metriken für AWS IoT Core für LoRaWAN. Weitere Informationen finden Sie unter AWS IoT Core für LoRaWAN Metriken.

AWS IoT Core für LoRaWAN Metriken

Metrik	Description
Aktiv devices/gateways	Die Anzahl der aktiven LoRa WAN-Geräte und Gateways in Ihrem Konto.
Anzahl der Uplink-Nachrichten	Die Anzahl der Uplink-Nachrichten, die innerhalb einer bestimmten Zeitdauer für alle aktiven Gateways und Geräte in Ihrem gesendet werden. AWS-Konto Uplink-Nachrichten sind Nachrichten, die von Ihrem Gerät an gesendet werden. AWS IoT Core für LoRaWAN
Anzahl der Downlink-Nachrichten	Die Anzahl der Downlink-Nachrichten, die innerhalb einer bestimmten Zeitdauer für alle aktiven Gateways und Geräte in Ihrem gesendet werden. AWS-Konto Downlink-Nachrichten sind Nachrichten, die von AWS IoT Core für LoRaWAN Ihrem Gerät gesendet werden.
Rate des Nachrichtenverlusts	Nachdem Sie Ihr Gerät hinzugefügt und eine Verbindung hergestellt haben AWS IoT Core für LoRaWAN, kann Ihr Gerät eine Uplink-Nachricht initiieren, um mit dem Nachrichtenaustausch mit der Cloud zu beginnen. Mit dieser Kennzahl können Sie

Metrik	Description
	dann die Rate der verlorenen Uplink-Nachrichten verfolgen.
Beitrittskennzahlen	Nachdem Sie Ihr Gerät und Ihr Gateway hinzugefügt haben, führen Sie einen Verbindungsvorgang durch, damit Ihr Gerät Uplink-Daten senden und mit ihnen kommunizieren kann. AWS IoT Core für LoRaWAN Sie können über diese Kennzahl Informationen zu Beitrittskennzahlen für alle aktiven Geräte in Ihrem AWS-Konto erhalten.
Anzeige der durchschnittlichen Empfangssignalstärke (RSSI)	Sie können mit dieser Kennzahl den durchschn ittlichen RSSI (Received Signal Strength Indicator) innerhalb des angegebenen Zeitraums zu überwache n. Mit der RSSI-Messung wird angegeben, ob das Signal stark genug für eine gute drahtlose Verbindun g ist. Dieser Wert ist negativ und muss für eine starke Verbindung näher an Null liegen.
Durchschnittliches Signal-Rausch-Verh ältnis (SNR)	Sie können diese Metrik verwenden, um das durchschnittliche SNR (Signal-to-noise Verhältnis) innerhalb der angegebenen Zeitdauer zu überwache n. Die SNR-Messung gibt an, ob das empfangen e Signal im Vergleich zum Geräuschpegel stark genug für eine gute Funkverbindung ist. Der SNR-Wert ist positiv und muss größer als Null sein, um anzuzeigen, dass die Signalleistung stärker ist als die Rauschleistung.
Gateway-Verfügbarkeit	Sie können über diese Kennzahl Informationen über die Verfügbarkeit dieses Gateways innerhalb eines bestimmten Zeitraums erhalten. Diese Kennzahl zeigt die Websocket-Verbindungszeit dieses Gateways für einen bestimmten Zeitraum an.

Just-in-time Metriken für die Bereitstellung

Metrik	Description
`ProvisionThing.ClientError`	Gibt an, wie oft ein Gerät aufgrund eines Client-Fehlers nicht bereitgestellt werden konnte. Beispielsweise war die in der Vorlage angegebene Richtlinie nicht vorhanden.
`ProvisionThing.ServerError`	Gibt an, wie oft ein Gerät aufgrund eines Serverfehlers nicht bereitgestellt werden konnte. Kunden können erneut versuchen, das Gerät bereitzustellen, nachdem sie gewartet haben und sie können sich an AWS IoT wenden, falls das Problem weiterhin besteht.
`ProvisionThing.Success`	Die Anzahl der erfolgreich bereitgestellten Geräte.

Metriken zur Flottenindizierung

AWS IoT Metriken zur Flottenindizierung

Metrik	Description
`NamedShadowCountForDynamicGroupQueryLimitExceeded`	In dynamischen Objektgruppen werden maximal 25 benannte Schatten pro Objekt für Abfragebegriffe verarbeitet, die nicht datenquellenspezifisch sind. Wird dieses Limit für ein Objekt überschritten, wird der Ereignistyp `NamedShadowCountForDynamicGroupQueryLimitExceeded` ausgegeben.

Dimensionen für Metriken

Metriken verwenden den Namespace und stellen Metriken für folgende Dimensionen bereit.

Dimension	Description
`ActionType`	Der Aktionstyp, der in der Regel festgelegt ist, welche die Anforderung ausgelöst hat.
`BehaviorName`	Der Name des Sicherheitsprofilverhaltens von Device Defender Detect, das überwacht wird.
`ClaimCertificateId`	Die `certificateId` des Anspruchs, der für die Bereitstellung der Geräte verwendet wurde.
`CheckName`	Der Name der Audit-Prüfung für Device Defender, deren Ergebnisse überwacht werden.
`JobId`	Die ID des Jobs, dessen Fortschritt oder Nachricht enverbindung überwacht success/failure wird.
`Protocol`	Das für die Anforderung verwendete Protokoll. Gültige Werte sind: MQTT oder HTTP
`RuleName`	Der Name der von der Anforderung ausgelösten Regel.
`ScheduledAuditName`	Der Name der geplanten Audit-Prüfung mit Device Defender, deren Ergebnisse überwacht werden. Dies hat den Wert `OnDemand`, wenn die gemeldete n Ergebnisse für ein Audit gelten, das auf Abruf durchgeführt wurde.
`SecurityProfileName`	Der Name des Sicherheitsprofils von Device Defender Detect, dessen Verhalten überwacht wird.
`TemplateName`	Der Name der Bereitstellungsvorlage.
`SourceArn`	Bezieht sich auf das Sicherheitsprofil für Detect oder das Konto arn für Audit.

Dimension	Description
`RoleArn`	Bezieht sich auf die Rolle, die Device Defender zu übernehmen versucht hat.
`TopicArn`	Bezieht sich auf das SNS-Thema, in dem Device Defender versucht hat, zu veröffentlichen.
`Error`	Enthält eine kurze Beschreibung des Fehlers, der beim Versuch, im SNS-Thema zu veröffentlichen, aufgetreten ist. Die möglichen Werte sind: • "KMSKeyNotFound„: gibt an, dass der KMS-Schlüssel für das Thema nicht existiert. • "InvalidTopicName„: zeigt an, dass das SNS-Thema nicht gültig ist. • "KMSAccessDenied„: gibt an, dass die Rolle keine Berechtigungen für den KMS-Schlüssel für das Thema hat. • "AuthorizationError„: gibt an, dass die angegebene Rolle Device Defender nicht autorisiert, Beiträge zum SNS-Thema zu veröffentlichen. • "SNSTopicNotFound„: zeigt an, dass das angegebene SNS-Thema nicht existiert. • "FailureToAssumeRole„: gibt an, dass die angegebene Rolle Device Defender nicht autorisiert, die Rolle zu übernehmen. • "CrossRegionSNSTopic„: gibt an, dass das SNS-Thema in einer anderen Region existiert.

Überwachen AWS IoT mithilfe von Protokollen CloudWatch

Wenn die AWS IoT Protokollierung aktiviert ist, werden Fortschrittsereignisse zu jeder Nachricht AWS IoT gesendet, die von Ihren Geräten über den Nachrichtenbroker und die Regel-Engine übertragen

wird. In der CloudWatch Konsole werden CloudWatch Protokolle in einer Protokollgruppe mit dem Namen angezeigt AWSIotLogs.

Weitere Informationen zu CloudWatch Protokollen finden Sie unter CloudWatch Protokolle. Hinweise zu unterstützten AWS IoT CloudWatch Protokollen finden Sie unterCloudWatch Protokolle AWS IoT Einträge protokollieren.

Anzeigen AWS IoT loggt sich in der CloudWatch Konsole ein

> **Note**
>
> Die `AWSIotLogsV2` Protokollgruppe ist in der CloudWatch Konsole erst sichtbar, wenn:
>
> - Sie haben die Anmeldung aktiviert AWS IoT. Weitere Informationen zur Aktivierung der Anmeldung finden Sie AWS IoT unter Konfiguration AWS IoT Protokollierung
> - Einige Logeinträge wurden von AWS IoT Operations geschrieben.

Um Ihre einzusehen AWS IoT loggt sich in der CloudWatch Konsole ein

1. Navigieren Sie zu https://console.aws.amazon.com/cloudwatch/. Wählen Sie im Navigationsbereich Protokollgruppen aus.
2. Geben Sie in das Textfeld Filter die Zeichenfolge **`AWSIotLogsV2`** ein und drücken Sie die Eingabetaste.
3. Double-click die `AWSIotLogsV2` Protokollgruppe.
4. Wählen Sie Alle durchsuchen aus. Eine vollständige Liste der für Ihr Konto generierten AWS IoT Protokolle wird angezeigt.
5. Wählen Sie das Erweiterungssymbol aus, um einen einzelnen Stream anzusehen.

Sie können auch eine Abfrage in das Textfeld Ereignisse filtern eingeben. Hier einige interessante Abfragen, die Sie ausprobieren können:

- `{ $.logLevel = "INFO" }`

  Suchen Sie alle Protokolle mit der Protokollebene `INFO`.
- `{ $.status = "Success" }`

  Suchen Sie alle Protokolle mit dem Status `Success`.

- `{ $.status = "Success" && $.eventType = "GetThingShadow" }`

  Suchen Sie alle Protokolle mit dem Status `Success` und dem Ereignistyp `GetThingShadow`.

Weitere Informationen zum Erstellen von Filterausdrücken finden Sie unter CloudWatch Protokollabfragen.

CloudWatch Protokolle AWS IoT Einträge protokollieren

Jede Komponente von AWS IoT generiert ihre eigenen Protokolleinträge. Jeder Protokolleintrag besitzt einen `eventType`, der die Operation angibt, die den Protokolleintrag generiert hat. In diesem Abschnitt werden die Protokolleinträge beschrieben, die von den folgenden AWS IoT -Komponenten generiert werden.

Themen

- Message Broker-Protokolleinträge
- OCSP-Protokolleinträge des Serverzertifikats
- Protokolleinträge „Geräteschatten“
- Protokolleinträge zur Regel-Engine
- Auftrag-Protokolleinträge
- Protokolleinträge für Gerätebereitstellung
- Protokolleinträge „Dynamische Objektgruppen“
- Protokolleinträge für die Flottenindizierung
- Allgemeine CloudWatch Log-Attribute

Message Broker-Protokolleinträge

Der AWS IoT Message Broker generiert Logeinträge für die folgenden Ereignisse:

Themen

- Protokolleintrag „Connect“
- Protokolleintrag „Disconnect“
- Ping-Logeintrag
- Connection.AuthNError Log-Eintrag
- DeleteConnection Protokolleintrag

- GetRetainedMessage Protokolleintrag
- ListRetainedMessage Log-Eintrag
- ListSubscriptions Log-Eintrag
- GetConnection Log-Eintrag
- Publish-In Log-Eintrag
- Publish-Out Protokolleintrag
- Protokolleintrag in der Warteschlange
- Protokolleintrag „Subscribe“
- Protokolleintrag abbestellen
- SendDirectMessage Log-Eintrag

Protokolleintrag „Connect“

Der AWS IoT Message Broker generiert einen Logeintrag mit einem `eventType` von, `Connect` wenn ein MQTT-Client eine Verbindung herstellt.

Beispiel für einen Connect-Erfolgsprotokolleintrag

```
{
    "timestamp": "2017-08-10 15:37:23.476",
    "logLevel": "INFO",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "accountId": "111122223333",
    "status": "Success",
    "eventType": "Connect",
    "protocol": "MQTT",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "203.0.113.11",
    "sourcePort": 13490
}
```

Beispiel für einen Protokolleintrag bei Verbindungsproblemen

```
{
    "timestamp": "2026-07-09 23:26:48.827",
    "logLevel": "ERROR",
```

```
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "accountId": "111122223333",
    "status": "Failure",
    "eventType": "Connect",
    "protocol": "MQTT",
    "clientId": "device-12345",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "203.0.113.1",
    "sourcePort": 11238,
    "reason": "throttled",
    "details": "Exceeded Connect requests per second per account limit"
}
```


Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `Connect` Protokolleinträge die folgenden Attribute:

clientId

Die ID des Clients, der die Anforderung stellt.

principalId

Die ID des Prinzipals, der die Anforderung stellt.

Protokoll

Das für die Anforderung verwendete Protokoll. Gültige Werte sind `MQTT` oder `HTTP`.

sourceIp

Die IP-Adresse, von der die Anforderung stammt.

sourcePort

Der Port, von dem die Anforderung stammt.

Grund

Der Grund für den `Connect` Betriebsausfall. Dieses Feld ist nur vorhanden, wenn das `status` ist`Failure`. Gültige Werte sind `throttled`, `client_id_throttled` oder `INTERNAL_SERVER_ERROR`.

Details

Eine kurze Erläuterung des Fehlers. Dieses Feld ist nur vorhanden, wenn das `status` ist`Failure`.

Codes für die Ursache

Die folgenden Ursachencodes und Details werden in den `Connect` Fehlerprotokolleinträgen angezeigt.

Grund	Details
`throttled`	Das Limit für Verbindungsanfragen pro Sekunde pro Konto wurde überschritten
`client_id_throttled`	Das Limit für Connect-Anfragen pro Sekunde pro Client-ID wurde überschritten

Protokolleintrag „Disconnect“

Der AWS IoT Message Broker generiert einen Logeintrag mit einem `eventType` von, `Disconnect` wenn ein MQTT-Client die Verbindung trennt.

Beispiel für den Protokolleintrag „Disconnect“

```
{
    "timestamp": "2017-08-10 15:37:23.476",
    "logLevel": "INFO",
    "traceId": "20b23f3f-d7f1-feae-169f-82263394fbdb",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "Disconnect",
    "protocol": "MQTT",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490,
    "reason": "DUPLICATE_CLIENT_ID",
    "details": "A new connection was established with the same client ID",
    "disconnectReason": "CLIENT_INITIATED_DISCONNECT"
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `Disconnect` Logeinträge die folgenden Attribute:

clientId

Die ID des Clients, der die Anforderung stellt.

principalId

Die ID des Prinzipals, der die Anforderung stellt.

Protokoll

Das für die Anforderung verwendete Protokoll. Gültige Werte sind `MQTT` oder `HTTP`.

sourceIp

Die IP-Adresse, von der die Anforderung stammt.

sourcePort

Der Port, von dem die Anforderung stammt.

Grund

Der Grund, warum der Client die Verbindung trennt.

Details

Eine kurze Erläuterung des Fehlers.

disconnectReason

Der Grund, warum der Client die Verbindung trennt.

Ping-Logeintrag

Der AWS IoT Message Broker generiert einen Logeintrag mit einem `eventType` oder, `Ping` wenn er einen MQTT-PINGREQ von einem verbundenen Client verarbeitet und den entsprechenden PINGRESP sendet. Pro Ping-Vorgang wird ein einzelner Logeintrag ausgegeben.

> **Important**
>
> Ping-Logeinträge sind standardmäßig aktiviert und deaktiviert. Der Emissionsschwellenwert entspricht dem Ergebnis pro Eintrag: Erfolgseinträge werden auf der `DEBUG` Ebene ausgegeben und Fehlereinträge werden auf der `ERROR` Ebene ausgegeben. Daher werden immer dann Ping-Fehlereinträge angezeigt, wenn die für die Verbindung geltende Ereignisebene oder die Resource-specific Außerkraftsetzung auf `ERROR` oder eine ausführlichere Stufe (`WARN`, `INFO`, oder`DEBUG`) festgelegt ist. Ping-Erfolgseinträge werden

jedoch nur veröffentlicht, wenn diese Stufe auf eingestellt ist. DEBUG Informationen zum Konfigurieren dieser Einstellungen finden Sie unter und. Konto konfigurieren und anmelden Event-level AWS IoT (CLI) Konfigurieren Sie Resource-specific Überschreibungen in AWS IoT (CLI)

Bei Erfolg `logLevel` gilt das ist `DEBUG` und das `status` ist`Success`. Bei einem Fehler (z. B. wenn die Client-Verbindung beendet wird, bevor der PINGRESP gesendet werden kann), `logLevel` ist `ERROR` und ist. `status` `Failure`

Beispiel für einen Ping-Logeintrag (erfolgreich)

```
{
    "timestamp": "2025-10-31 15:37:23.476",
    "logLevel": "DEBUG",
    "traceId": "20b23f3f-d7f1-feae-169f-82263394fbdb",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "Ping",
    "protocol": "MQTT",
    "latency": 128,
    "requestTimestamp": "2025-10-31 15:37:23.348",
    "responseTimestamp": "2025-10-31 15:37:23.476",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490
}
```

Beispiel für einen Ping-Logeintrag (Fehler)

```
{
    "timestamp": "2025-10-31 15:37:23.476",
    "logLevel": "ERROR",
    "traceId": "20b23f3f-d7f1-feae-169f-82263394fbdb",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "Ping",
    "protocol": "MQTT",
    "requestTimestamp": "2025-10-31 15:37:23.476",
    "clientId": "abf27092886e49a8a5c1922749736453",
```

```
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490,
    "reason": "CONNECTION_ALREADY_CLOSED",
    "details": "Connection has already closed"
}
```

Neben Allgemeine CloudWatch Log-Attribute enthalten `Ping`-Protokolleinträge die folgenden Attribute:

Protokoll

Das für die Anforderung verwendete Protokoll. Dieser Wert ist immer `MQTT`.

latency

Die Zeit in Millisekunden zwischen dem Empfang des PINGREQ und dem Senden des PINGRESP. Dieses Feld ist nur vorhanden, wenn das ist. `status` `Success`

RequestTimeStamp

Der Zeitstempel, zu dem der PINGREQ vom Client AWS IoT Core empfangen wurde.

Zeitstempel der Antwort

Der Zeitstempel, als der PINGRESP AWS IoT Core an den Client gesendet wurde. Dieses Feld ist nur vorhanden, wenn das ist. `status` `Success`

clientId

Die ID des Clients, der die Anforderung stellt.

principalId

Die ID des Prinzipals, der die Anforderung stellt.

sourceIp

Die IP-Adresse, von der die Anforderung stammt.

sourcePort

Der Port, von dem die Anforderung stammt.

Grund

Der Grund für den Ausfall des Ping-Vorgangs. Dieses Feld ist nur vorhanden, wenn das `status` ist`Failure`.

Details

Eine kurze Erläuterung des Fehlers. Dieses Feld ist nur vorhanden, wenn das `status` ist`Failure`.

Connection.AuthNError Log-Eintrag

Der AWS IoT Message Broker generiert einen Protokolleintrag mit einem `eventType` von`Connection.AuthNError`, wenn ein Client-Verbindungsversuch aufgrund eines Authentifizierungsfehlers abgelehnt wird. Das `logLevel` für diesen Logeintrag ist immer `ERROR` und das `status` ist immer`Failure`. Dieser Logeintrag entspricht der `Connection.AuthNError` CloudWatch Metrik.

> **Important**
>
> `Connection.AuthNError`Logeinträge sind standardmäßig aktiviert und deaktiviert. Sie werden auf `ERROR` Protokollebene ausgegeben, sodass ein `Connection.AuthNError` Protokolleintrag nur veröffentlicht wird, wenn die für die Verbindung geltende Ereignisebene oder die Resource-specific Außerkraftsetzung auf `ERROR` oder eine ausführlichere Stufe (`WARN`, `INFO` oder) gesetzt ist. `DEBUG` Informationen zum Konfigurieren dieser Einstellungen finden Sie unter und. Konto konfigurieren und anmelden Event-level AWS IoT (CLI) Konfigurieren Sie Resource-specific Überschreibungen in AWS IoT (CLI)

> **Note**
>
> Dieser Protokolleintrag wird auf demjenigen veröffentlicht AWS-Konto , dem die im TLS-Handshake bereitgestellte Server Name Indication (SNI) gehört. Wenn der Client kein SNI bereitstellt oder das SNI nicht zu einem Konto aufgelöst werden kann, wird kein Logeintrag oder keine Metrik ausgegeben.

Connection.AuthNError Beispiel für einen Logeintrag

```
{
    "timestamp": "2025-10-31 15:37:23.476",
    "logLevel": "ERROR",
    "traceId": "20b23f3f-d7f1-feae-169f-82263394fbdb",
    "accountId": "123456789012",
```

```
    "status": "Failure",
    "eventType": "Connection.AuthNError",
    "protocol": "MQTT",
    "authenticationType": "AWS_X509",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490,
    "targetIp": "52.94.236.10",
    "targetPort": 8883,
    "serverNameIndication": "a1b2c3d4e5f6g7-ats.iot.us-east-1.amazonaws.com",
    "reason": "DEVICE_CERTIFICATE_NOT_REGISTERED",
    "details": "Device certificate is not registered"
}
```


Neben Allgemeine CloudWatch Log-Attribute enthalten `Connection.AuthNError`-Protokolleinträge die folgenden Attribute:

Protokoll

Das für die Anforderung verwendete Protokoll. Gültige Werte sind `MQTT` oder `HTTP`.

AuthenticationType

Die Art der Authentifizierung, die für den Verbindungsversuch verwendet wird. Gültige Werte sind `AWS_X509`, `AWS_SIGV4`, `CUSTOM_AUTH` und `CUSTOM_AUTH_X509`.

clientId

Die ID des Clients, der die Anfrage stellt, falls verfügbar.

principalId

Die ID des Prinzipals, der die Anfrage stellt, falls verfügbar.

sourceIp

Die IP-Adresse, von der die Anforderung stammt.

sourcePort

Der Port, von dem die Anforderung stammt.

Ziel-IP

Die Ziel-IP-Adresse der Verbindung. Dieses Feld ist möglicherweise nicht in allen Protokolleinträgen vorhanden.

Zielport

Der Zielport der Verbindung.

Server NameIndication

Der vom Client bereitgestellte Hostname für die Erweiterung SNI (TLS Server Name Indication).

Grund

Der Grund, warum die Client-Authentifizierung fehlgeschlagen ist. Eine Liste der möglichen Werte finden Sie unterthe section called “Connection.AuthNError Fehlercodes”.

Details

Eine kurze Beschreibung des Authentifizierungsfehlers.

Connection.AuthNError Fehlercodes

In der folgenden Tabelle sind die `reason` Fehlercodes und die entsprechenden `details` Meldungen aufgeführt, die für einen `Connection.AuthNError` Protokolleintrag auftreten können. In Zukunft können weitere Fehlercodes hinzugefügt werden.

Fehlercode	Fehlermeldung
`CUSTOM_AUTHORIZER_LAMBDA_EXECUTION_ERROR`	Der Lambda-Aufruf des benutzerdefinierten Authorizers ist fehlgeschlagen
`CUSTOM_AUTHORIZER_NOT_FOUND`	Benutzerdefinierter Autorisierer konnte nicht gefunden werden
`CUSTOM_AUTHORIZER_PARAMETER_INVALID`	Der benutzerdefinierte Authentifizierungsparameter ist ungültig
`DEVICE_CERTIFICATE_INACTIVE`	Das Gerätezertifikat ist nicht aktiv
`DEVICE_CERTIFICATE_NOT_REGISTERED`	Das Gerätezertifikat ist nicht registriert
`DEVICE_CERTIFICATE_REVOKED`	Das Gerätezertifikat wurde gesperrt

Fehlercode	Fehlermeldung
DOMAIN_CONFIGURATION_DISABLED	Die Domänenkonfiguration für die bereitgestellte Server Name Indication (SNI) ist deaktiviert
DOMAIN_CONFIGURATION_INVALID	Der Endpunkttyp in der Domänenkonfiguration ist ungültig
INTERNAL_SERVER_ERROR	Interner Serverfehler
NOT_AUTHENTICATED	Die angegebenen Anmeldeinformationen konnten nicht authentifiziert werden
SECURITY_TOKEN_EXPIRED	Das Sicherheitstoken ist abgelaufen
SECURITY_TOKEN_INVALID	Das in der Anfrage enthaltene Sicherheitstoken ist ungültig
SECURITY_TOKEN_SIGNATURE_MISMATCH	Die Anforderungssignatur stimmt nicht mit der angegebenen Signatur überein
SNI_MISUSED	Der Kunde hat eine SNI verwendet, die nicht mit diesem Konto verknüpft ist

DeleteConnection Protokolleintrag

Der AWS IoT Message Broker generiert einen Logeintrag mit einem `eventType` von, `DeleteConnection` wenn eine MQTT-Client-Verbindung gelöscht wird.

DeleteConnection Beispiel für einen Logeintrag


```
{
    "timestamp": "2025-08-09 15:37:23.476",
    "logLevel": "INFO",
    "traceId": "20b23f3f-d7f1-feae-169f-82263394fbdb",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "DeleteConnection",
    "protocol": "HTTP",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
```

```
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `DeleteConnection` Protokolleinträge die folgenden Attribute:

clientId

Die ID des Clients, der getrennt wird.

principalId

Die ID des Prinzipals, der die Anforderung stellt. Informationen zur Identifizierung des Principals anhand der Principal-ID finden Sie im IAM-Benutzerhandbuch unter Vergleichen von IAM-Identitäten und -Anmeldeinformationen.

Protokoll

Das für die Anforderung verwendete Protokoll. Der gültige Wert lautet `HTTP`.

sourceIp

Die IP-Adresse, von der die Anforderung stammt.

sourcePort

Der Port, von dem die Anforderung stammt.

GetRetainedMessage Protokolleintrag

Der AWS IoT Message Broker generiert einen Logeintrag mit einem oder `eventTypeGetRetainedMessage`, wenn er aufgerufen `GetRetainedMessage` wird.

GetRetainedMessage Beispiel für einen Logeintrag

```
{
    "timestamp": "2017-08-07 18:47:56.664",
    "logLevel": "INFO",
    "traceId": "1a60d02e-15b9-605b-7096-a9f584a6ad3f",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "GetRetainedMessage",
```

```
    "protocol": "HTTP",
    "topicName": "a/b/c",
    "qos": "1",
    "lastModifiedDate": "2017-08-07 18:47:56.664"
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `GetRetainedMessage` Protokolleinträge die folgenden Attribute:

letzte ModifiedDate

Das Epochendatum und die Uhrzeit in Millisekunden, zu denen die aufbewahrte Nachricht gespeichert wurde. AWS IoT

Protokoll

Das für die Anforderung verwendete Protokoll. Zulässiger Wert: `HTTP`.

qos

Das QoS (Quality of Service)-Niveau, das in der Veröffentlichungsanforderung verwendet wird. Gültige Werte sind `0` oder 1.

topicName

Der Name des abonnierten Themas.

ListRetainedMessage Log-Eintrag

Der AWS IoT Message Broker generiert einen Logeintrag mit einem oder `eventTypeListRetainedMessage`, wenn er aufgerufen `ListRetainedMessages` wird.

ListRetainedMessage Beispiel für einen Logeintrag

```
{
    "timestamp": "2017-08-07 18:47:56.664",
    "logLevel": "INFO",
    "traceId": "1a60d02e-15b9-605b-7096-a9f584a6ad3f",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "ListRetainedMessage",
    "protocol": "HTTP"
}
```

Zusätzlich zu den enthalten Allgemeine CloudWatch Log-Attribute die `ListRetainedMessage` Protokolleinträge das folgende Attribut:

Protokoll

Das für die Anforderung verwendete Protokoll. Zulässiger Wert: `HTTP`.

ListSubscriptions Log-Eintrag

Der AWS IoT Message Broker generiert einen Logeintrag mit einem `eventType` von, `ListSubscriptions` wenn eine Anfrage zum Auflisten von Kundenabonnements eingeht.

ListSubscriptions Beispiel für einen Logeintrag

```
{
    "timestamp": "2025-08-22 19:57:46.587",
    "logLevel": "INFO",
    "traceId": "5bea658a-0752-bf20-770f-ad17c8dabb95",
    "accountId": "987654321",
    "status": "Success",
    "eventType": "ListSubscriptions",
    "protocol": "HTTP",
    "clientId": "abc",
    "principalId": "AIDAZPOK3C3545MCMYATF",
    "sourceIp": "52.94.133.137",
    "sourcePort": 13490,
    "maxResults": 100
}
```

Neben Allgemeine CloudWatch Log-Attribute enthalten `ListSubscriptions`-Protokolleinträge die folgenden Attribute:

clientId

Die ID des Clients, der die Anforderung stellt.

principalId

Die ID des Prinzipals, der die Anforderung stellt.

Protokoll

Das für die Anforderung verwendete Protokoll. Zulässiger Wert: `HTTP`.

sourceIp

Die IP-Adresse, von der die Anforderung stammt.

sourcePort

Der Port, von dem die Anforderung stammt.

maxResults

Die maximale Anzahl angeforderter Ergebnisse.

GetConnection Log-Eintrag

Der AWS IoT Message Broker generiert einen Logeintrag mit einem oder`GetConnection`, wenn eine Anfrage zum Abrufen `eventType` von Verbindungsinformationen eingeht.

GetConnection Beispiel für einen Logeintrag

```
{
    "timestamp": "2025-08-22 19:19:50.679",
    "logLevel": "INFO",
    "traceId": "7a04de1a-d1a3-50bc-3ca5-96b1ee293dae",
    "accountId": "987654321",
    "status": "Success",
    "eventType": "GetConnection",
    "protocol": "HTTP",
    "clientId": "aClient",
    "principalId": "AIDAZPOK3C3545MCMYATF",
    "sourceIp": "52.94.133.137",
    "sourcePort": 31983
}
```

Neben Allgemeine CloudWatch Log-Attribute enthalten `GetConnection`-Protokolleinträge die folgenden Attribute:

clientId

Die ID des Clients, der die Anforderung stellt.

principalId

Die ID des Prinzipals, der die Anforderung stellt.

Protokoll

Das für die Anforderung verwendete Protokoll. Zulässiger Wert: `HTTP`.

sourceIp

Die IP-Adresse, von der die API-Anfrage stammt.

sourcePort

Der Port, von dem die API-Anfrage stammt.

Publish-In Log-Eintrag

Wenn der AWS IoT Message Broker eine MQTT-Nachricht empfängt, generiert er einen Logeintrag mit einem `eventType` von`Publish-In`.

Publish-In Beispiel für einen Erfolgsprotokolleintrag

```
{
        "timestamp": "2017-08-10 15:39:30.961",
        "logLevel": "INFO",
        "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
        "accountId": "111122223333",
        "status": "Success",
        "eventType": "Publish-In",
        "protocol": "MQTT",
        "topicName": "$aws/things/MyThing/shadow/get",
        "clientId": "abf27092886e49a8a5c1922749736453",
        "principalId":
 "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
        "sourceIp": "203.0.113.12",
        "sourcePort": 13490,
        "retain": "True"
    }
```

Publish-In Beispiel für einen Fehlerprotokolleintrag

```
{
    "timestamp": "2026-07-09 22:01:12.858",
    "logLevel": "ERROR",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "accountId": "111122223333",
```

```
    "status": "Failure",
    "eventType": "Publish-In",
    "protocol": "MQTT",
    "topicName": "testing/topic",
    "clientId": "device-12345",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "203.0.113.4",
    "sourcePort": 10872,
    "reason": "throttled",
    "details": "Exceeded Inbound publish requests per second per account limit"
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `Publish-In` Protokolleinträge die folgenden Attribute:

clientId

Die ID des Clients, der die Anforderung stellt.

principalId

Die ID des Prinzipals, der die Anforderung stellt.

Protokoll

Das für die Anforderung verwendete Protokoll. Gültige Werte sind `MQTT` oder `HTTP`.

Beibehaltung

Das Attribut, das verwendet wird, wenn für eine Nachricht das RETAIN-Flag auf den Wert `True` festgelegt ist. Wenn für die Nachricht das RETAIN-Flag nicht festgelegt ist, erscheint dieses Attribut nicht im Protokolleintrag. Weitere Informationen finden Sie unter Beibehaltene MQTT-Meldungen.

sourceIp

Die IP-Adresse, von der die Anforderung stammt.

sourcePort

Der Port, von dem die Anforderung stammt.

topicName

Der Name des abonnierten Themas.

Grund

Der Grund für den `Publish-In` Betriebsausfall. Dieses Feld ist nur vorhanden, wenn das `status` ist`Failure`. Gültige Werte sind `throttled`, `connection_throttled`, `CLIENT_ERROR` oder `INTERNAL_SERVER_ERROR`.

Details

Eine kurze Erläuterung des Fehlers. Dieses Feld ist nur vorhanden, wenn das `status` ist`Failure`.

Codes für die Ursache

Die folgenden Ursachencodes und Details werden in den `Publish-In` Fehlerprotokolleinträgen angezeigt.

Grund	Details
`throttled`	Das Limit für eingehende Veröffentlichungsanfragen pro Sekunde und Konto wurde überschritten
`throttled`	Das Limit für ausgehende Veröffentlichungsanfragen pro Sekunde pro Konto wurde überschritten
`throttled`	Das Limit für Regelbewertungen pro Sekunde pro Konto wurde überschritten
`throttled`	Das Limit für eingehende Veröffentlichungsanfragen pro Sekunde pro Konto wurde überschritten
`throttled`	Das Limit für eingehende Veröffentlichungsanfragen pro Sekunde pro Thema wurde überschritten
`throttled`	Das Limit für die maximale Anzahl an aufbewahrten Nachrichten pro Konto wurde überschritten

Grund	Details
`connection_throttled`	Das Limit für Veröffentlichungsanfragen pro Sekunde pro Verbindung wurde überschritten
`CLIENT_ERROR`	Das maximale Limit für eingehende unbestätigte QoS 1-Veröffentlichungsanfragen wurde überschritten

Publish-Out Protokolleintrag

Wenn der Message Broker eine MQTT-Nachricht veröffentlicht, generiert er einen Protokolleintrag mit dem `eventType` `Publish-Out`

Publish-Out Beispiel für einen erfolgreichen Protokolleintrag

```
{
    "timestamp": "2017-08-10 15:39:30.961",
    "logLevel": "INFO",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "accountId": "111122223333",
    "status": "Success",
    "eventType": "Publish-Out",
    "protocol": "MQTT",
    "topicName": "$aws/things/MyThing/shadow/get",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "203.0.113.13",
    "sourcePort": 13490
}
```

Publish-Out Beispiel für einen Fehlerprotokolleintrag

```
{
    "timestamp": "2026-07-14 21:12:33.026",
    "logLevel": "ERROR",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "accountId": "111122223333",
    "status": "Failure",
    "eventType": "Publish-Out",
```

```
    "protocol": "MQTT",
    "topicName": "testing/topic",
    "clientId": "subscriberClientThrottled",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "reason": "connection_throttled",
    "details": "Exceeded Publish requests per second per connection limit"
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `Publish-Out` Protokolleinträge die folgenden Attribute:

clientId

Die ID des abonnierten Clients, der Nachrichten zu diesem MQTT-Thema empfängt.

principalId

Die ID des Prinzipals, der die Anforderung stellt.

Protokoll

Das für die Anforderung verwendete Protokoll. Gültige Werte sind `MQTT` oder `HTTP`.

sourceIp

Die IP-Adresse, von der die Anforderung stammt.

sourcePort

Der Port, von dem die Anforderung stammt.

topicName

Der Name des abonnierten Themas.

Grund

Der Grund für den `Publish-Out` Betriebsausfall. Dieses Feld ist nur vorhanden, wenn das `status` ist`Failure`. Gültige Werte sind `throttled`, `connection_throttled`, `CLIENT_ERROR` oder `INTERNAL_SERVER_ERROR`.

Details

Eine kurze Erläuterung des Fehlers. Dieses Feld ist nur vorhanden, wenn das `status` ist`Failure`.

Codes für die Ursache

Die folgenden Ursachencodes und Details werden in den `Publish-Out` Fehlerprotokolleinträgen angezeigt.

Grund	Details
`throttled`	Das Limit für ausgehende Veröffentlichungsanfragen pro Sekunde und Konto wurde überschritten
`connection_throttled`	Das Limit für Veröffentlichungsanfragen pro Sekunde pro Verbindung wurde überschritten
`connection_throttled`	Das maximale Limit für ausgehende unbestätigte QoS 1-Veröffentlichungsanfragen wurde überschritten
`CLIENT_ERROR`	Der Client-Endpunkt ist nicht beschreibbar
`CLIENT_ERROR`	Der Client ist nicht verbunden

Protokolleintrag in der Warteschlange

Wenn ein Gerät mit einer persistenten Sitzung getrennt wird, speichert der MQTT-Nachrichtenbroker die Nachrichten des Geräts und AWS IoT generiert Protokolleinträge mit einem EventType-Wert von. `Queued` Weitere Hinweise zu persistenten MQTT-Sitzungen finden Sie unter. Persistente MQTT-Sitzungen

Beispiel für einen Eintrag in das Fehlerprotokoll eines Servers in der Warteschlange

```
{
    "timestamp": "2022-08-10 15:39:30.961",
    "logLevel": "ERROR",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "accountId": "111122223333",
    "topicName": "$aws/things/MyThing/get",
    "clientId": "123123123",
    "qos": "1",
    "protocol": "MQTT",
```

```
    "eventType": "Queued",
    "status": "Failure",
    "details": "Server Error"
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `Queued` Serverfehlerprotokolleinträge die folgenden Attribute:

clientId

Die ID des Clients, für den sich die Nachricht in der Warteschlange befindet.

Details

`Serverfehler`

Aufgrund eines Serverfehlers konnte die Nachricht nicht gespeichert werden.

Protokoll

Das für die Anforderung verwendete Protokoll. Dieser Wert ist immer `MQTT`.

qos

Die QoS (Quality of Service)-Ebene der Anforderung. Der Wert ist immer 1, da die Nachrichten mit QoS von 0 nicht gespeichert werden.

topicName

Der Name des abonnierten Themas.

Beispiel für einen Eintrag in einem Erfolgsprotokoll in der Warteschlange

```
{
    "timestamp": "2022-08-10 15:39:30.961",
    "logLevel": "INFO",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "accountId": "111122223333",
    "topicName": "$aws/things/MyThing/get",
    "clientId": "123123123",
    "qos": "1",
    "protocol": "MQTT",
    "eventType": "Queued",
```

```
    "status": "Success"
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `Queued` Erfolgsprotokolleinträge die folgenden Attribute:

clientId

Die ID des Clients, für den sich die Nachricht in der Warteschlange befindet.

Protokoll

Das für die Anforderung verwendete Protokoll. Dieser Wert ist immer `MQTT`.

qos

Die QoS (Quality of Service)-Ebene der Anforderung. Der Wert ist immer 1, da die Nachrichten mit QoS von 0 nicht gespeichert werden.

topicName

Der Name des abonnierten Themas.

Beispiel für einen gedrosselten Protokolleintrag in der Warteschlange

```
{
    "timestamp": "2022-08-10 15:39:30.961",
    "logLevel": "ERROR",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "accountId": "111122223333",
    "topicName": "$aws/things/MyThing/get",
    "clientId": "123123123",
    "qos": "1",
    "protocol": "MQTT",
    "eventType": "Queued",
    "status": "Failure",
    "reason": "connection_throttled",
    "details": "Throttled while queueing offline message"
}
```

Zusätzlich zu den enthalten Allgemeine CloudWatch Log-Attribute `Queued` gedrosselte Protokolleinträge die folgenden Attribute:

clientId

Die ID des Clients, für den sich die Nachricht in der Warteschlange befindet.

Details

Eine kurze Erläuterung des Fehlers. Wenn `status` ist`Failure`, ist der Wert. `Throttled while queueing offline message`

Protokoll

Das für die Anforderung verwendete Protokoll. Dieser Wert ist immer `MQTT`.

qos

Die QoS (Quality of Service)-Ebene der Anforderung. Der Wert ist immer 1, da die Nachrichten mit QoS von 0 nicht gespeichert werden.

Grund

`connection_throttled`

Der Client hat das `Publish requests per second per connection` Limit überschritten, sodass die Nachricht nicht gespeichert wurde.

`gedrosselt`

Das Konto hat das `Queued messages per second per account` Limit überschritten, daher wurde die Nachricht nicht gespeichert.

topicName

Der Name des abonnierten Themas.

Protokolleintrag „Subscribe“

Der AWS IoT Message Broker generiert einen Logeintrag mit einem oder`eventType`, `Subscribe` wenn ein MQTT-Client ein Thema abonniert.

Beispiel für einen Erfolgsprotokolleintrag in MQTT 3 Subscribe

```
{
    "timestamp": "2017-08-10 15:39:04.413",
    "logLevel": "INFO",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
```

```
    "accountId": "111122223333",
    "status": "Success",
    "eventType": "Subscribe",
    "protocol": "MQTT",
    "topicName": "$aws/things/MyThing/shadow/#",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "203.0.113.14",
    "sourcePort": 13490
}
```

Beispiel für einen Fehlerprotokolleintrag abonnieren

```
{
    "timestamp": "2026-07-09 23:07:45.633",
    "logLevel": "ERROR",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "accountId": "111122223333",
    "status": "Failure",
    "eventType": "Subscribe",
    "protocol": "MQTT",
    "topicName": "subscribe/test",
    "clientId": "device-12345",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "203.0.113.2",
    "sourcePort": 10494,
    "reason": "throttled",
    "details": "Exceeded Subscriptions per second per account limit"
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `Subscribe` Protokolleinträge die folgenden Attribute:

clientId

Die ID des Clients, der die Anforderung stellt.

principalId

Die ID des Prinzipals, der die Anforderung stellt.

Protokoll

Das für die Anforderung verwendete Protokoll. Dieser Wert ist immer `MQTT`.

sourceIp

Die IP-Adresse, von der die Anforderung stammt.

sourcePort

Der Port, von dem die Anforderung stammt.

topicName

Der Name des abonnierten Themas.

Grund

Der Grund für den `Subscribe` Betriebsausfall. Dieses Feld ist nur vorhanden, wenn das `status` ist`Failure`. Gültige Werte sind `throttled`, `CLIENT_ERROR` oder `INTERNAL_SERVER_ERROR`.

Details

Eine kurze Erläuterung des Fehlers. Dieses Feld ist nur vorhanden, wenn das `status` ist`Failure`.

Beispiel für den Protokolleintrag in MQTT-5-Subscribe

```
{
 "timestamp": "2022-11-30 16:24:15.628",
 "logLevel": "INFO",
 "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
 "accountId": "111122223333",
 "status": "Success",
 "eventType": "Subscribe",
 "protocol": "MQTT",
 "topicName": "test/topic1,$invalid/reserved/topic",
 "subscriptions": [
  {
   "topicName": "test/topic1",
   "reasonCode": 1
  },
  {
   "topicName": "$invalid/reserved/topic",
   "reasonCode": 143
  }
 ],
 "clientId": "abf27092886e49a8a5c1922749736453",
 "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
```

```
 "sourceIp": "203.0.113.15",
 "sourcePort": 13490
}
```

Bei MQTT 5-Subscribe-Vorgängen enthalten MQTT 5-Logeinträge zusätzlich zu den Attributen Allgemeine CloudWatch Log-Attribute und den `Subscribe` Subscribe-Logeinträgen das folgende Attribut:

Abonnements

Eine Liste von Zuordnungen zwischen den angeforderten Themen in der Abonnement-Anforderung und dem individuellen MQTT 5-Ursachencode. Weitere Informationen finden Sie unter MQTT-Ursachencodes.

Codes für die Ursache

Die folgenden Ursachencodes und Details werden in den `Subscribe` Fehlerprotokolleinträgen angezeigt.

Grund	Details
`throttled`	Das Limit für Abonnements pro Sekunde pro Konto wurde überschritten
`throttled`	Das Limit der Abonnements pro Konto wurde überschritten
`throttled`	Das Limit für Abonnements pro Verbindung wurde überschritten
`throttled`	Das Limit für gemeinsame Abonnements pro Gruppe wurde überschritten
`throttled`	Das Limit für An- und Abmeldeanfragen pro Sekunde pro Gruppe wurde überschritten
`CLIENT_ERROR`	Beim Abonnement-Paket wird eine Paket-ID aus einer früheren unbestätigten Anfrage wiederverwendet

Grund	Details
CLIENT_ERROR	Der Client ist nicht verbunden
CLIENT_ERROR	Das Limit für maximale Abonnements pro Abonnementanfrage wurde überschritten
CLIENT_ERROR	Ungültiger Themenfilter

Protokolleintrag abbestellen

Der AWS IoT Message Broker generiert einen Logeintrag mit einem `eventType` von, `Unsubscribe` wenn sich ein MQTT-Client von einem MQTT-Thema abmeldet.

Beispiel für einen erfolgreichen Logeintrag beim Abbestellen

```
{
    "timestamp": "2024-08-20 22:53:32.844",
    "logLevel": "INFO",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "accountId": "111122223333",
    "status": "Success",
    "eventType": "Unsubscribe",
    "protocol": "MQTT",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "203.0.113.16",
    "sourcePort": 13490
}
```

Beispiel für einen Fehlerprotokolleintrag beim Abbestellen

```
{
    "timestamp": "2026-01-07 15:37:23.476",
    "logLevel": "ERROR",
    "traceId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "accountId": "111122223333",
    "status": "Failure",
    "eventType": "Unsubscribe",
    "protocol": "MQTT",
    "clientId": "device-12345",
```

```
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "203.0.113.3",
    "sourcePort": 26051,
    "reason": "throttled",
    "details": "Exceeded Unsubscribe requests per second per account limit"
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `Unsubscribe` Protokolleinträge die folgenden Attribute:

Protokoll

Das für die Anforderung verwendete Protokoll. Dieser Wert ist immer `MQTT`.

clientId

Die ID des Clients, der die Anforderung stellt.

principalId

Die ID des Prinzipals, der die Anforderung stellt.

sourceIp

Die IP-Adresse, von der die Anforderung stammt.

sourcePort

Der Port, von dem die Anforderung stammt.

Grund

Der Grund für den `Unsubscribe` Betriebsausfall. Dieses Feld ist nur vorhanden, wenn das `status` ist`Failure`. Gültige Werte sind `throttled`, `CLIENT_ERROR` oder `INTERNAL_SERVER_ERROR`.

Details

Eine kurze Erläuterung des Fehlers. Dieses Feld ist nur vorhanden, wenn das `status` ist`Failure`.

Codes für die Ursache

Die folgenden Ursachencodes und Details werden in den `Unsubscribe` Fehlerprotokolleinträgen angezeigt.

Grund	Details
`throttled`	Das Limit für Abmeldeanfragen pro Sekunde pro Konto wurde überschritten
`throttled`	Das Limit für An- und Abmeldeanfragen pro Sekunde pro Gruppe wurde überschritten
`CLIENT_ERROR`	Beim Abmelde-Paket wird eine Paket-ID aus einer früheren unbestätigten Anfrage wiederver wendet
`CLIENT_ERROR`	Der Client ist nicht verbunden

SendDirectMessage Log-Eintrag

Der AWS IoT Message Broker generiert einen Logeintrag mit einem `eventType` von`SendDirectMessage`, wenn eine SendDirectMessage API-Anfrage gestellt wird.

AWS IoT Core protokolliert SendDirectMessage Ereignisse in CloudWatch Logs, wenn Sie die IoT-Protokollierung aktivieren. Weitere Informationen zur Konfiguration der Protokollierung finden Sie unterKonfiguration AWS IoT Protokollierung.

SendDirectMessage Beispiel für einen Erfolgsprotokolleintrag


```
{
    "timestamp": "2026-01-20 20:44:00.123",
    "logLevel": "INFO",
    "traceId": "1a2b3c4d-1234-3214-abcd-1a2b3c4d5e6f",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "SendDirectMessage",
    "protocol": "HTTP",
    "topicName": "commands/reboot",
    "clientId": "mydevice-002",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "confirmation": true,
    "sourceIp": "203.0.113.45",
    "sourcePort": 52847,
    "timeout": 3
```

```
}
```

SendDirectMessage Beispiel für einen Fehlerprotokolleintrag

```
{
    "timestamp": "2026-01-15 20:30:00.123",
    "logLevel": "ERROR",
    "traceId": "1a2b3c4d-1234-5678-abcd-1a2b3c4d5e6f",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "SendDirectMessage",
    "protocol": "HTTP",
    "topicName": "commands/reboot",
    "clientId": "myDevice",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "confirmation": true,
    "sourceIp": "203.0.113.45",
    "sourcePort": 52847,
    "reason": "DELIVERY_CONFIRMATION_TIMEOUT",
    "details": "The delivery confirmation was not received within the specified timeout
 period",
    "timeout": 2
}
```

Neben Allgemeine CloudWatch Log-Attribute enthalten `SendDirectMessage`-Protokolleinträge die folgenden Attribute:

clientId

Die ID des Clients, der die Direktnachricht empfängt.

topicName

Das Thema, zu dem die Nachricht gesendet wurde.

Protokoll

Das für die Anforderung verwendete Protokoll. Zulässiger Wert: `HTTP`.

Bestätigung

Ob eine Lieferbestätigung angefordert wurde. Gültige Werte sind `true` oder `false`.

timeout

Maximale Wartezeit in Sekunden auf eine Lieferbestätigung des empfangenden Kunden. Nur anwesend, wenn `confirmation` es ist`true`.

Grund

Der Ursachencode für den Fehler.

Details

Eine kurze Erläuterung des Fehlers.

principalId

Die ID des Prinzipals, der die Anforderung stellt.

sourceIp

Die IP-Adresse, von der die Anforderung stammt.

sourcePort

Der Port, von dem die Anforderung stammt.

Ursachencodes

Die folgenden Ursachencodes und Details werden in den `SendDirectMessage` Fehlerprotokolleinträgen angezeigt.

Grund	Details
`MALFORMED_CLIENTID`	Diese Client-ID ist nicht gültig. Kunden-ID s dürfen 128 Zeichen nicht überschreiten und dürfen nicht mit einem Dollarzeichen ($) beginnen. Geben Sie eine andere Kunden-ID ein
`INVALID_TOPIC`	Dieses Thema ist nicht gültig. Themen müssen innerhalb von 256 Byte lang sein, sollten weniger als 9 Teile enthalten und dürfen nicht mit einem Dollarzeichen ($) beginnen

Grund	Details
INVALID_TIMEOUT_PARAMETER	Ungültiger Timeout-Wert. Muss eine Ganzzahl zwischen 1 und 15 sein
INVALID_CONFIRMATION_PARAMETER	Ungültiger Bestätigungswert. Muss 'wahr' oder 'falsch' sein
RECEIVE_AUTHORIZATION_FAILURE	Die Autorisierung für den Zielclient ist fehlgesch lagen. Stellen Sie sicher, dass Ihre Richtlinie die erforderlichen Berechtigungen gewährt
TARGET_CLIENT_ID_NOT_FOUND	Die Client-ID wurde nicht gefunden. Stellen Sie sicher, dass der Client verbunden ist, und versuchen Sie es erneut
TARGET_CLIENT_NOT_CONNECTED	Die Zielclient-ID ist nicht verbunden, es besteht jedoch eine aktive persistente Sitzung
SEND_DIRECT_MESSAGE_THROTTLED	Die Anfrage wurde abgelehnt, da die Anforderu ngsrate zum Senden von Direktnachrichten überschritten wurde
RECEIVE_SESSION_THROTTLED	Die Anfrage wurde abgelehnt, da die Anforderu ngsrate die ausgehenden Veröffentlichungsa nfragen pro Sekunde pro Verbindung überstieg
RECEIVER_MAX_UNACKED_QOS1_E XCEEDED	Der Empfängerclient hat die maximale Anzahl unbestätigter QoS 1-Nachrichten erreicht
DELIVERY_CONFIRMATION_TIMEOUT	Die Lieferbestätigung wurde nicht innerhalb des angegebenen Timeout-Zeitraums empfangen
DELIVERY_CONFIRMATION_SENDE R_DISCONNECT	Der Absender hat die Verbindung unterbroc hen, bevor die Lieferbestätigung empfangen wurde
CLIENT_ENDPOINT_NOT_WRITABLE	Der Endpunkt des Empfänger-Clients ist nicht beschreibbar

Grund	Details
INTERNAL_SERVER_ERROR	Interner Serverfehler

OCSP-Protokolleinträge des Serverzertifikats

AWS IoT Core generiert Protokolleinträge für das folgende Ereignis:

Themen

- RetrieveOCSPStapleData Log-Eintrag
- RetrieveOCSPStapleData Protokolleintrag für private Endpunkte

RetrieveOCSPStapleData Log-Eintrag

AWS IoT Core generiert einen Protokolleintrag mit einem oder, `RetrieveOCSPStapleData` wenn `eventType` der Server die OCSP-Stapeldaten abruft.

RetrieveOCSPStapleData Beispiele für Protokolleinträge

Im Folgenden finden Sie ein Beispiel für einen Protokolleintrag von`Success`.

```
{
 "timestamp": "2024-01-30 15:39:30.961",
 "logLevel": "INFO",
 "traceId": "180532b7-0cc7-057b-687a-5ca1824838f5",
 "accountId": "123456789012",
 "status": "Success",
 "eventType": "RetrieveOCSPStapleData",
 "domainConfigName": "test-domain-config-name",
 "connectionDetails": {
  "httpStatusCode": "200",
  "ocspResponderUri": "http://ocsp.example.com",
  "sourceIp": "205.251.233.181",
  "targetIp": "250.15.5.3"
 },
 "ocspRequestDetails": {
  "requesterName": "iot.amazonaws.com",
  "requestCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
 },
```

```
 "ocspResponseDetails": {
  "responseCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
  "ocspResponseStatus": "successful",
  "certStatus": "good",
  "signature":
 "4C:6F:63:61:6C:20:52:65:73:70:6F:6E:64:65:72:20:53:69:67:6E:61:74:75:72:65",
  "thisUpdateTime": "Jan 31 01:21:02 2024 UTC",
  "nextUpdateTime": "Feb 02 00:21:02 2024 UTC",
  "producedAtTime": "Jan 31 01:37:03 2024 UTC",
  "stapledDataPayloadSize": "XXX"
 }
}
```

Im Folgenden finden Sie ein Beispiel für einen Protokolleintrag für`Failure`.

```
{
 "timestamp": "2024-01-30 15:39:30.961",
 "logLevel": "ERROR",
 "traceId": "180532b7-0cc7-057b-687a-5ca1824838f5",
 "accountId": "123456789012",
 "status": "Failure",
 "reason": "A non 2xx HTTP response was received from the OCSP responder.",
 "eventType": "RetrieveOCSPStapleData",
 "domainConfigName": "test-domain-config-name",
 "connectionDetails": {
  "httpStatusCode": "444",
  "ocspResponderUri": "http://ocsp.example.com",
  "sourceIp": "205.251.233.181",
  "targetIp": "250.15.5.3"
 },
 "ocspRequestDetails": {
  "requesterName": "iot.amazonaws.com",
  "requestCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
 }
}
```

Für den `RetrieveOCSPStaple` Vorgang enthalten die Allgemeine CloudWatch Log-Attribute Protokolleinträge zusätzlich zu den die folgenden Attribute:

Grund

Der Grund, warum der Vorgang fehlschlägt.

Domäne ConfigName

Der Name Ihrer Domain-Konfiguration.

Angaben zur Verbindung

Eine kurze Erläuterung der Verbindungsdetails.

- http StatusCode

  HTTP-Statuscodes, die vom OCSP-Responder als Antwort auf die Anfrage des Clients an den Server zurückgegeben werden.
- ocsp ResponderUri

  Die OCSP-Responder-URI, die aus dem AWS IoT Core Serverzertifikat abruft.
- sourceIp

  Die Quell-IP-Adresse des Servers. AWS IoT Core
- Ziel-IP

  Die Ziel-IP-Adresse des OCSP-Responders.

ocsp RequestDetails

Details der OCSP-Anfrage.

- Name des Anforderers

  Der Bezeichner für den AWS IoT Core Server, der eine Anfrage an den OCSP-Responder sendet.
- Anfrage CertId

  Die Zertifikat-ID der Anfrage. Dies ist die ID des Zertifikats, für das die OCSP-Antwort angefordert wird.

ocsp ResponseDetails

Details der OCSP-Antwort.

- Antwort CertId

  Die Zertifikat-ID der OCSP-Antwort.

- ocsp ResponseStatus

  Der Status der OCSP-Antwort.
- CertStatus

  Der Status des Zertifikats.
- signature

  Die Signatur, die von einer vertrauenswürdigen Entität auf die Antwort angewendet wird.
- das UpdateTime

  Der Zeitpunkt, zu dem der angezeigte Status bekanntermaßen korrekt ist.
- als nächstes UpdateTime

  Der Zeitpunkt, zu dem oder vor dem neuere Informationen über den Status des Zertifikats verfügbar sein werden.
- produziert AtTime

  Der Zeitpunkt, zu dem der OCSP-Responder diese Antwort signiert hat.
- geheftet DataPayloadSize

  Die Nutzdatengröße der gehefteten Daten.

RetrieveOCSPStapleData Protokolleintrag für private Endpunkte

AWS IoT Core generiert einen Protokolleintrag mit einem oder, `RetrieveOCSPStapleData` wenn `eventType` der Server die OCSP-Stapeldaten abruft.

RetrieveOCSPStapleData Beispiele für Protokolleinträge für private Endpunkte

Im Folgenden finden Sie ein Beispiel für einen Protokolleintrag von. `Success`

```
{
 "timestamp": "2024-01-30 15:39:30.961",
 "logLevel": "INFO",
 "traceId": "180532b7-0cc7-057b-687a-5ca1824838f5",
 "accountId": "123456789012",
 "status": "Success",
 "eventType": "RetrieveOCSPStapleData",
 "domainConfigName": "test-domain-config-name",
```

```
        "lambdaDetails": {
        "lambdaArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
        "sourceArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
testDomainConfigure/6bzfg"
      },
        "authorizedResponderArn": "arn:aws:acm:us-west-2:123456789012:certificate/
certificate_ID",
 "ocspRequestDetails": {
  "requesterName": "iot.amazonaws.com",
  "requestCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
 },
 "ocspResponseDetails": {
  "responderId": "04:C1:3F:8F:27:D6:49:13:F8:DE:B2:36:9D:85:8E:F8:31:3B:A6:D0"
              "responseCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
  "ocspResponseStatus": "successful",
  "certStatus": "good",
  "signature":
 "4C:6F:63:61:6C:20:52:65:73:70:6F:6E:64:65:72:20:53:69:67:6E:61:74:75:72:65",
  "thisUpdateTime": "Jan 31 01:21:02 2024 UTC",
  "nextUpdateTime": "Feb 02 00:21:02 2024 UTC",
  "producedAtTime": "Jan 31 01:37:03 2024 UTC",
  "stapledDataPayloadSize": "XXX"
 }
}
```

Im Folgenden finden Sie ein Beispiel für einen Protokolleintrag für`Failure`.

```
{
 "timestamp": "2024-01-30 15:39:30.961",
 "logLevel": "ERROR",
 "traceId": "180532b7-0cc7-057b-687a-5ca1824838f5",
 "accountId": "123456789012",
 "status": "Failure",
 "reason": "The payload returned by the Lambda function exceeds the maximum response
 size of 7 kilobytes.",
 "eventType": "RetrieveOCSPStapleData",
 "domainConfigName": "test-domain-config-name",
        "lambdaDetails": {
        "lambdaArn": "arn:aws:lambda:us-west-2:123456789012:function:my-function",
        "sourceArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
testDomainConfigure/6bzfg"
```

```
      },
        "authorizedResponderArn": "arn:aws:acm:us-west-2:123456789012:certificate/
certificate_ID",
  "ocspRequestDetails": {
  "requesterName": "iot.amazonaws.com",
  "requestCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
 }
}
```

Für den `RetrieveOCSPStaple` Vorgang enthalten die RetrieveOCSPStapleData Protokolleinträge für private Endpunkte zusätzlich zu den Attributen Allgemeine CloudWatch Log-Attribute und den Attributen im Protokolleintrag die folgenden Attribute:

Lambda-Einzelheiten

Details der Lambda-Funktion.

- LambdaArn

  ARN der Lambda-Funktion.

- Quelle Arn

  Der ARN der Domänenkonfiguration.

autorisiert ResponderArn

Der ARN des Authorizer-Responders, falls in der Domänenkonfiguration einer konfiguriert ist.

Protokolleinträge „Geräteschatten“

Der AWS IoT Device Shadow-Dienst generiert Protokolleinträge für die folgenden Ereignisse:

Themen

- DeleteThingShadow Protokolleintrag
- GetThingShadow Eintrag protokollieren
- UpdateThingShadow Eintrag protokollieren

DeleteThingShadow Protokolleintrag

Der Geräteschatten-Service generiert einen Protokolleintrag `eventType` für `DeleteThingShadow`, wenn eine Anforderung zum Löschen eines Geräteschattens empfangen wird.

DeleteThingShadow Beispiel für einen Protokolleintrag

```
{
    "timestamp": "2017-08-07 18:47:56.664",
    "logLevel": "INFO",
    "traceId": "1a60d02e-15b9-605b-7096-a9f584a6ad3f",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "DeleteThingShadow",
    "protocol": "MQTT",
    "deviceShadowName": "Jack",
    "topicName": "$aws/things/Jack/shadow/delete"
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `DeleteThingShadow` Protokolleinträge die folgenden Attribute:

Gerät ShadowName

Der Name des zu aktualisierenden Schattens.

Protokoll

Das für die Anforderung verwendete Protokoll. Gültige Werte sind `MQTT` oder `HTTP`.

topicName

Der Name des Themas, in dem die Anforderung veröffentlicht wurde.

GetThingShadow Eintrag protokollieren

Der Geräteschatten-Service generiert einen Protokolleintrag mit `eventType` für `GetThingShadow`, wenn eine Abrufanforderung für einen Schatten empfangen wird.

GetThingShadow Beispiel für einen Protokolleintrag

```
{
```

```
    "timestamp": "2017-08-09 17:56:30.941",
    "logLevel": "INFO",
    "traceId": "b575f19a-97a2-cf72-0ed0-c64a783a2504",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "GetThingShadow",
    "protocol": "MQTT",
    "deviceShadowName": "MyThing",
    "topicName": "$aws/things/MyThing/shadow/get"
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `GetThingShadow` Protokolleinträge die folgenden Attribute:

Gerät ShadowName

Der Name des angefragten Schattens.

Protokoll

Das für die Anforderung verwendete Protokoll. Gültige Werte sind `MQTT` oder `HTTP`.

topicName

Der Name des Themas, in dem die Anforderung veröffentlicht wurde.

UpdateThingShadow Eintrag protokollieren

Der Geräteschatten-Service generiert einen Protokolleintrag durch `eventType` `UpdateThingShadow`, wenn eine Anforderung zum Aktualisieren eines Geräteschattens empfangen wird.

UpdateThingShadow Beispiel für einen Protokolleintrag

```
{
    "timestamp": "2017-08-07 18:43:59.436",
    "logLevel": "INFO",
    "traceId": "d0074ba8-0c4b-a400-69df-76326d414c28",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "UpdateThingShadow",
    "protocol": "MQTT",
```

```
    "deviceShadowName": "Jack",
    "topicName": "$aws/things/Jack/shadow/update"
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `UpdateThingShadow` Protokolleinträge die folgenden Attribute:

Gerät ShadowName

Der Name des zu aktualisierenden Schattens.

Protokoll

Das für die Anforderung verwendete Protokoll. Gültige Werte sind `MQTT` oder `HTTP`.

topicName

Der Name des Themas, in dem die Anforderung veröffentlicht wurde.

Protokolleinträge zur Regel-Engine

Die AWS IoT Regel-Engine generiert Protokolle für die folgenden Ereignisse:

Themen

- FunctionExecution Protokolleintrag
- RuleExecution Log-Eintrag
- RuleMatch Protokolleintrag
- RuleEvaluation gedrosselter Protokolleintrag
- RuleExecutionThrottled Log-Eintrag
- RuleNotFound Log-Eintrag
- StartingRuleExecution Log-Eintrag

FunctionExecution Protokolleintrag

Die Regel-Engine generiert einen Protokolleintrag mit `eventType` von `FunctionExecution`, wenn die SQL-Abfrage einer Regel eine externe Funktion aufruft. Eine externe Funktion wird aufgerufen, wenn die Aktion einer Regel eine HTTP-Anfrage an AWS IoT oder einen anderen Webdienst sendet (z. B. durch Aufrufen`get_thing_shadow`).

FunctionExecution Beispiel für einen Logeintrag

```
{
    "timestamp": "2017-07-13 18:33:51.903",
    "logLevel": "DEBUG",
    "traceId": "180532b7-0cc7-057b-687a-5ca1824838f5",
    "status": "Success",
    "eventType": "FunctionExecution",
    "clientId": "N/A",
    "topicName":"rules/test",
    "ruleName": "ruleTestPredict",
    "ruleAction": "MachinelearningPredict",
    "resources": {
        "ModelId": "predict-model"
    },
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `FunctionExecution` Protokolleinträge die folgenden Attribute:

clientId

N/A für `FunctionExecution`-Protokolle.

principalId

Die ID des Prinzipals, der die Anforderung stellt.

Ressourcen

Eine Sammlung von Ressourcen, die von den Aktionen der Regel verwendet werden.

ruleName

Der Name der übereinstimmenden Regel.

topicName

Der Name des abonnierten Themas.

RuleExecution Log-Eintrag

Wenn die AWS IoT Regel-Engine die Aktion einer Regel auslöst, generiert sie einen `RuleExecution` Protokolleintrag.

RuleExecution Beispiel für einen Logeintrag

```
{
    "timestamp": "2017-08-10 16:32:46.070",
    "logLevel": "INFO",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "RuleExecution",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "rules/test",
    "ruleName": "JSONLogsRule",
    "ruleAction": "RepublishAction",
    "resources": {
        "RepublishTopic": "rules/republish"
    },
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
}
```

RuleExecution Beispiel für einen Batch-Logeintrag

```
{
    "logLevel": "INFO",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "RuleExecution",
    "ruleName": "rule_test",
    "ruleAction": "HttpAction",
    "resources": {
        "Url": "https://example.com",
        "ConfirmationUrl": "https://example.com"
    },
    "details": "HttpAction made a request to the specified endpoint",
    "batchDetails": {
        "timestamps": [
            "1234567890123",
            "1234567890123",
            "1234567890123"
        ],
        "traceIds": [
            "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
            "30aa7ccc-1d23-0b97-aa7b-76196d83537c",
            "30aa7ccc-1d23-0b97-aa7b-76196d83537d"
```

```
        ],
        "clientIds": [
            "N/A",
            "N/A",
            "N/A"
        ],
        "topicNames": [
            "topic/ruletest",
            "topic/ruletest",
            "topic/ruletest"
        ],
        "principalIds": [
            "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
            "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
            "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
        ],
        "batchSize": 3,
        "batchSizeInBytes": 114
    }
}
```

Neben Allgemeine CloudWatch Log-Attribute enthalten `RuleExecution`-Protokolleinträge die folgenden Attribute:

clientId

Die ID des Clients, der die Anforderung stellt.

principalId

Die ID des Prinzipals, der die Anforderung stellt.

Ressourcen

Eine Sammlung von Ressourcen, die von den Aktionen der Regel verwendet werden.

ruleAction

Der Name der ausgelösten Aktion.

ruleName

Der Name der übereinstimmenden Regel.

topicName

Der Name des abonnierten Themas.

RuleMatch Protokolleintrag

Die AWS IoT Regel-Engine generiert einen Logeintrag mit einem `eventType` von`RuleMatch`, wenn der Message Broker eine Nachricht erhält, die einer Regel entspricht.

RuleMatch Beispiel für einen Logeintrag

```
{
    "timestamp": "2017-08-10 16:32:46.002",
    "logLevel": "INFO",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "RuleMatch",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "rules/test",
    "ruleName": "JSONLogsRule",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `RuleMatch` Protokolleinträge die folgenden Attribute:

clientId

Die ID des Clients, der die Anforderung stellt.

principalId

Die ID des Prinzipals, der die Anforderung stellt.

ruleName

Der Name der übereinstimmenden Regel.

topicName

Der Name des abonnierten Themas.

RuleEvaluation gedrosselter Protokolleintrag

Wenn AWS IoT die Rules Engine eine Regelauswertung für das Limit der Regelauswertungen pro Sekunde pro Konto drosselt, generiert sie einen Protokolleintrag mit einem „Von“ und einem Grund als`eventType`. `RuleEvaluation` `Throttled`

RuleEvaluation Beispiel für einen gedrosselten Protokolleintrag

```
{
    "timestamp": "2024-11-13 07:32:27.856",
    "logLevel": "ERROR",
    "traceId": "82995222-049b-ee5c-0cd4-15635f052ed3",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "RuleEvaluation",
    "clientId": "57b3b41a-c726-44ab-bcd4-2cc1c2cc4295",
    "topicName": "N/A",
    "ruleName": "N/A",
    "principalId": "N/A",
    "reason": "Throttled",
    "details": "Exceeded Rule Evaluations per second per account limit"
}
```

Neben Allgemeine CloudWatch Log-Attribute enthalten `RuleEvaluation`-Protokolleinträge die folgenden Attribute:

clientId

Die ID des Clients, der die Anforderung stellt.

Details

Eine kurze Erläuterung des Fehlers.

principalId

Die ID des Prinzipals, der die Anforderung stellt. Immer N/A für diesen Logeintrag.

Grund

Die Zeichenfolge „Throttled“.

ruleName

Der Name der Regel. Immer N/A für diesen Logeintrag.

topicName

Der Name des veröffentlichten Themas. Immer N/A für diesen Logeintrag.

RuleExecutionThrottled Log-Eintrag

Wenn eine Regelauswertung durch die Regel-Engine aufgrund von bösartigem Verhalten gedrosselt wird, z. B. aufgrund übermäßiger Ingest-Veröffentlichungsanfragen an nicht existierende Regeln, generiert die AWS IoT Regel-Engine einen Protokolleintrag mit dem Wert von. `eventType` `RuleExecutionThrottled`

RuleExecutionThrottled Beispiel für einen Protokolleintrag

```
{
    "timestamp": "2017-10-04 19:25:46.070",
    "logLevel": "ERROR",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "RuleExecutionThrottled",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "$aws/rules/example_rule",
    "ruleName": "example_rule",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "reason": "RuleExecutionThrottled",
    "details": "Exection of Rule example_rule throttled"
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `RuleExecutionThrottled` Protokolleinträge die folgenden Attribute:

clientId

Die ID des Clients, der die Anforderung stellt.

Details

Eine kurze Erläuterung des Fehlers.

principalId

Die ID des Prinzipals, der die Anforderung stellt.

Grund

Die Zeichenfolge "RuleExecutionThrottled“.

ruleName

Der Name der Regel, die ausgelöst werden soll.

topicName

Der Name des Themas, das veröffentlicht wurde.

RuleNotFound Log-Eintrag

Wenn die AWS IoT Regel-Engine keine Regel mit einem bestimmten Namen finden kann, generiert sie einen Protokolleintrag mit dem Wert `eventType` von`RuleNotFound`.

RuleNotFound Beispiel für einen Logeintrag

```
{
    "timestamp": "2017-10-04 19:25:46.070",
    "logLevel": "ERROR",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "RuleNotFound",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "$aws/rules/example_rule",
    "ruleName": "example_rule",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "reason": "RuleNotFound",
    "details": "Rule example_rule not found"
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `RuleNotFound` Protokolleinträge die folgenden Attribute:

clientId

Die ID des Clients, der die Anforderung stellt.

Details

Eine kurze Erläuterung des Fehlers.

principalId

Die ID des Prinzipals, der die Anforderung stellt.

Grund

Die Zeichenfolge "RuleNotFound“.

ruleName

Der Name der Regel, die nicht gefunden werden konnte.

topicName

Der Name des Themas, das veröffentlicht wurde.

StartingRuleExecution Log-Eintrag

Wenn die AWS IoT Regel-Engine beginnt, die Aktion einer Regel auszulösen, generiert sie einen Protokolleintrag mit dem Wert `eventType` von`StartingRuleExecution`.

StartingRuleExecution Beispiel für einen Logeintrag

```
{
    "timestamp": "2017-08-10 16:32:46.002",
    "logLevel": "DEBUG",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "StartingRuleExecution",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "rules/test",
    "ruleName": "JSONLogsRule",
    "ruleAction": "RepublishAction",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `rule-` Protokolleinträge die folgenden Attribute:

clientId

Die ID des Clients, der die Anforderung stellt.

principalId

Die ID des Prinzipals, der die Anforderung stellt.

ruleAction

Der Name der ausgelösten Aktion.

ruleName

Der Name der übereinstimmenden Regel.

topicName

Der Name des abonnierten Themas.

Auftrag-Protokolleinträge

Der AWS IoT Job-Service generiert Protokolleinträge für die folgenden Ereignisse. Protokolleinträge werden generiert, wenn eine MQTT- oder HTTP-Anforderung vom Gerät empfangen wird.

Themen

- DescribeJobExecution Log-Eintrag
- GetPendingJobExecution Log-Eintrag
- ReportFinalJobExecutionCount Log-Eintrag
- StartNextPendingJobExecution Log-Eintrag
- UpdateJobExecution Log-Eintrag

DescribeJobExecution Log-Eintrag

Der AWS IoT Jobs-Dienst generiert einen Protokolleintrag mit einem `eventType` von`DescribeJobExecution`, wenn der Dienst eine Anforderung zur Beschreibung einer Auftragsausführung erhält.

DescribeJobExecution Beispiel für einen Protokolleintrag

```
{
    "timestamp": "2017-08-10 19:13:22.841",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "DescribeJobExecution",
    "protocol": "MQTT",
    "clientId": "thingOne",
    "jobId": "002",
    "topicName": "$aws/things/thingOne/jobs/002/get",
    "clientToken": "myToken",
```

```
    "details": "The request status is SUCCESS."
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `GetJobExecution` Protokolleinträge die folgenden Attribute:

clientId

Die ID des Clients, der die Anforderung stellt.

clientToken

Ein eindeutiger Bezeichner, bei dem die Groß- und Kleinschreibung beachtet werden muss, um die Idempotenz der Anforderung sicherzustellen. Weitere Informationen finden Sie unter So wird Idempotenz sichergestellt.

Details

Weitere Informationen über den Jobs-Service.

jobId

Die Auftrags-ID für die Auftragsausführung.

Protokoll

Das für die Anforderung verwendete Protokoll. Gültige Werte sind `MQTT` oder `HTTP`.

topicName

Das für die Anforderung verwendete Thema.

GetPendingJobExecution Log-Eintrag

Der AWS IoT Jobs-Dienst generiert einen Protokolleintrag mit einem `eventType` oder`GetPendingJobExecution`, wenn der Dienst eine Anforderung zur Auftragsausführung erhält.

GetPendingJobExecution Beispiel für einen Protokolleintrag

```
{
    "timestamp": "2018-06-13 17:45:17.197",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
```

```
    "status": "Success",
    "eventType": "GetPendingJobExecution",
    "protocol": "MQTT",
    "clientId": "299966ad-54de-40b4-99d3-4fc8b52da0c5",
    "topicName": "$aws/things/299966ad-54de-40b4-99d3-4fc8b52da0c5/jobs/get",
    "clientToken": "24b9a741-15a7-44fc-bd3c-1ff2e34e5e82",
    "details": "The request status is SUCCESS."
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `GetPendingJobExecution` Protokolleinträge die folgenden Attribute:

clientId

Die ID des Clients, der die Anforderung stellt.

clientToken

Ein eindeutiger Bezeichner, bei dem die Groß- und Kleinschreibung beachtet werden muss, um die Idempotenz der Anforderung sicherzustellen. Weitere Informationen finden Sie unter So wird Idempotenz sichergestellt.

Details

Weitere Informationen über den Jobs-Service.

Protokoll

Das für die Anforderung verwendete Protokoll. Gültige Werte sind `MQTT` oder `HTTP`.

topicName

Der Name des abonnierten Themas.

ReportFinalJobExecutionCount Log-Eintrag

Der AWS IoT Jobs-Dienst generiert einen Protokolleintrag mit einem `entryType` oder`ReportFinalJobExecutionCount`, wenn ein Job abgeschlossen ist.

ReportFinalJobExecutionCount Beispiel für einen Logeintrag

```
{
    "timestamp": "2017-08-10 19:44:16.776",
```

```
    "logLevel": "INFO",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "ReportFinalJobExecutionCount",
    "jobId": "002",
    "details": "Job 002 completed. QUEUED job execution count: 0 IN_PROGRESS job
 execution count: 0 FAILED job execution count: 0 SUCCEEDED job execution count: 1
 CANCELED job execution count: 0 REJECTED job execution count: 0 REMOVED job execution
 count: 0"
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `ReportFinalJobExecutionCount` Protokolleinträge die folgenden Attribute:

Details

Weitere Informationen über den Jobs-Service.

jobId

Die Auftrags-ID für die Auftragsausführung.

StartNextPendingJobExecution Log-Eintrag

Wenn der AWS IoT Jobs-Dienst eine Anforderung erhält, die nächste ausstehende Auftragsausführung zu starten, generiert er einen Protokolleintrag mit dem Wert `eventType` von`StartNextPendingJobExecution`.

StartNextPendingJobExecution Beispiel für einen Protokolleintrag

```
{
    "timestamp": "2018-06-13 17:49:51.036",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "StartNextPendingJobExecution",
    "protocol": "MQTT",
    "clientId": "95c47808-b1ca-4794-bc68-a588d6d9216c",
    "topicName": "$aws/things/95c47808-b1ca-4794-bc68-a588d6d9216c/jobs/start-next",
    "clientToken": "bd7447c4-3a05-49f4-8517-dd89b2c68d94",
    "details": "The request status is SUCCESS."
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `StartNextPendingJobExecution` Protokolleinträge die folgenden Attribute:

clientId

Die ID des Clients, der die Anforderung stellt.

clientToken

Ein eindeutiger Bezeichner, bei dem die Groß- und Kleinschreibung beachtet werden muss, um die Idempotenz der Anforderung sicherzustellen. Weitere Informationen finden Sie unter So wird Idempotenz sichergestellt.

Details

Weitere Informationen über den Jobs-Service.

Protokoll

Das für die Anforderung verwendete Protokoll. Gültige Werte sind `MQTT` oder `HTTP`.

topicName

Das für die Anforderung verwendete Thema.

UpdateJobExecution Log-Eintrag

Der AWS IoT Jobs-Dienst generiert einen Protokolleintrag mit einem `eventType` oder`UpdateJobExecution`, wenn der Dienst eine Anforderung zur Aktualisierung einer Auftragsausführung erhält.

UpdateJobExecution Beispiel für einen Protokolleintrag

```
{
    "timestamp": "2017-08-10 19:25:14.758",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "UpdateJobExecution",
    "protocol": "MQTT",
    "clientId": "thingOne",
    "jobId": "002",
    "topicName": "$aws/things/thingOne/jobs/002/update",
```

```
    "clientToken": "myClientToken",
    "versionNumber": "1",
    "details": "The destination status is IN_PROGRESS. The request status is SUCCESS."
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `UpdateJobExecution` Protokolleinträge die folgenden Attribute:

clientId

Die ID des Clients, der die Anforderung stellt.

clientToken

Ein eindeutiger Bezeichner, bei dem die Groß- und Kleinschreibung beachtet werden muss, um die Idempotenz der Anforderung sicherzustellen. Weitere Informationen finden Sie unter So wird Idempotenz sichergestellt.

Details

Weitere Informationen über den Jobs-Service.

jobId

Die Auftrags-ID für die Auftragsausführung.

Protokoll

Das für die Anforderung verwendete Protokoll. Gültige Werte sind `MQTT` oder `HTTP`.

topicName

Das für die Anforderung verwendete Thema.

versionNumber

Die Version der Auftragsausführung.

Protokolleinträge für Gerätebereitstellung

Der AWS IoT Device Provisioning-Dienst generiert Protokolle für die folgenden Ereignisse.

Themen

- GetDeviceCredentials Protokolleintrag

- ProvisionDevice Eintrag protokollieren

GetDeviceCredentials Protokolleintrag

Der AWS IoT Device Provisioning-Dienst generiert einen Protokolleintrag mit einem `eventType` oder, `GetDeviceCredential` wenn ein Client anruft`GetDeviceCredential`.

GetDeviceCredentials Beispiel für einen Protokolleintrag

```
{
  "timestamp" : "2019-02-20 20:31:22.932",
  "logLevel" : "INFO",
  "traceId" : "8d9c016f-6cc7-441e-8909-7ee3d5563405",
  "accountId" : "123456789101",
  "status" : "Success",
  "eventType" : "GetDeviceCredentials",
  "deviceCertificateId" :
 "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
  "details" : "Additional details about this log."
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `GetDeviceCredentials` Protokolleinträge die folgenden Attribute:

Details

Eine kurze Erläuterung des Fehlers.

Gerät CertificateId

Die ID des Gerätezertifikats.

ProvisionDevice Eintrag protokollieren

Der AWS IoT Device Provisioning-Dienst generiert einen Protokolleintrag mit einem `eventType` oder, `ProvisionDevice` wenn ein Client anruft`ProvisionDevice`.

ProvisionDevice Beispiel für einen Protokolleintrag

```
{
  "timestamp" : "2019-02-20 20:31:22.932",
```

```
  "logLevel" : "INFO",
  "traceId" : "8d9c016f-6cc7-441e-8909-7ee3d5563405",
  "accountId" : "123456789101",
  "status" : "Success",
  "eventType" : "ProvisionDevice",
  "provisioningTemplateName" : "myTemplate",
  "deviceCertificateId" :
 "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
  "details" : "Additional details about this log."
 }
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `ProvisionDevice` Protokolleinträge die folgenden Attribute:

Details

Eine kurze Erläuterung des Fehlers.

Gerät CertificateId

Die ID des Gerätezertifikats.

Bereitstellung TemplateName

Der Name der Bereitstellungsvorlage.

Protokolleinträge „Dynamische Objektgruppen“

AWS IoT Dynamische Dinggruppen generieren Protokolle für das folgende Ereignis.

Themen

- AddThingToDynamicThingGroupsFailed Log-Eintrag

AddThingToDynamicThingGroupsFailed Log-Eintrag

Wenn AWS IoT den angegebenen dynamischen Gruppen nichts hinzugefügt werden konnte, wird ein Logeintrag mit dem Wert `eventType` von generiert`AddThingToDynamicThingGroupsFailed`. Dies geschieht, wenn ein Objekt die Kriterien für die Mitgliedschaft in der dynamischen Objektgruppe erfüllt, es aber der dynamischen Gruppe nicht hinzugefügt werden konnte oder es aus der dynamischen Gruppe entfernt wurde. Dies kann aus folgenden Gründen passieren:

- Das Objekt gehört bereits der maximalen Anzahl von Gruppen an.

- Mithilfe der Option --override-dynamic-groups wurde das Objekt zu einer statischen Objektgruppe hinzugefügt. Es wurde aus einer dynamischen Objektgruppe entfernt, um dies möglich zu machen.

Weitere Informationen finden Sie unter Dynamische Objektgruppen – Einschränkungen und Konflikte.

AddThingToDynamicThingGroupsFailed Beispiel für einen Logeintrag

In diesem Beispiel ist der Protokolleintrag für einen Fehler `AddThingToDynamicThingGroupsFailed` zu sehen. In diesem Beispiel TestThing erfüllte es die Kriterien, um zu den unter aufgeführten dynamischen Dinggruppen zu gehören`dynamicThingGroupNames`, konnte aber nicht zu diesen dynamischen Gruppen hinzugefügt werden, wie unter beschrieben`reason`.

```
{
 "timestamp": "2020-03-16 22:24:43.804",
 "logLevel": "ERROR",
 "traceId": "70b1f2f5-d95e-f897-9dcc-31e68c3e1a30",
 "accountId": "57EXAMPLE833",
 "status": "Failure",
 "eventType": "AddThingToDynamicThingGroupsFailed",
 "thingName": "TestThing",
 "dynamicThingGroupNames": [
  "DynamicThingGroup11",
  "DynamicThingGroup12",
  "DynamicThingGroup13",
  "DynamicThingGroup14"
 ],
 "reason": "The thing failed to be added to the given dynamic thing group(s) because
 the thing already belongs to the maximum allowed number of groups."
}
```

Zusätzlich zu den Allgemeine CloudWatch Log-Attribute enthalten `AddThingToDynamicThingGroupsFailed` Protokolleinträge die folgenden Attribute:

dynamisch ThingGroupNames

Ein Array der dynamischen Objektgruppen, denen das Objekt nicht hinzugefügt werden konnte.

Grund

Der Grund, warum das Objekt nicht zu den dynamischen Objektgruppen hinzugefügt werden konnte.

thingName

Der Name des Objekts, das keiner dynamischen Objektgruppe hinzugefügt werden konnte.

Protokolleinträge für die Flottenindizierung

AWS IoT Die Flottenindizierung generiert Protokolleinträge für die folgenden Ereignisse.

Themen

- NamedShadowCountForDynamicGroupQueryLimitExceeded Protokolleintrag

NamedShadowCountForDynamicGroupQueryLimitExceeded Protokolleintrag

Für Abfragebegriffe, die nicht datenquellenspezifisch sind, werden in dynamischen Gruppen maximal 25 benannte Schatten pro Objekt verarbeitet. Wird dieses Limit für ein Objekt überschritten, wird der Ereignistyp `NamedShadowCountForDynamicGroupQueryLimitExceeded` ausgegeben.

NamedShadowCountForDynamicGroupQueryLimitExceeded Beispiel für einen Protokolleintrag

In diesem Beispiel ist der Protokolleintrag für einen Fehler `NamedShadowCountForDynamicGroupQueryLimitExceeded` zu sehen. In diesem Beispiel können Ergebnisse, die ausschließlich auf Werten `DynamicGroup` basieren, ungenau sein, wie im Feld `reason` beschrieben.

```
{
"timestamp": "2020-03-16 22:24:43.804",
"logLevel": "ERROR",
"traceId": "70b1f2f5-d95e-f897-9dcc-31e68c3e1a30",
"accountId": "571032923833",
"status": "Failure",
"eventType": "NamedShadowCountForDynamicGroupQueryLimitExceeded",
"thingName": "TestThing",
"reason": "A maximum of 25 named shadows per thing are processed for non-data source
 specific query terms in dynamic groups."
}
```

Allgemeine CloudWatch Log-Attribute

Alle CloudWatch Log-Logeinträge enthalten die folgenden Attribute:

accountId

Ihre AWS-Konto ID.

eventType

Der Ereignistyp, für den das Protokoll generiert wurde. Der Wert des Ereignistyps hängt vom Ereignis ab, das den Protokolleintrag generiert hat. Jede Beschreibung des Protokolleintrags enthält den Wert von `eventType` für diesen Protokolleintrag.

logLevel

Die verwendete Protokollierungsebene. Weitere Informationen finden Sie unter the section called “Protokollstufen”.

Status

Der Status der Anforderung.

Zeitstempel

Der UNIX-Zeitstempel für den Zeitpunkt, an dem der Client eine Verbindung zum AWS IoT - Message Broker hergestellt hat.

traceId

Eine zufällig erstellte Kennung, die verwendet werden kann, um alle Protokolle für eine bestimmte Anforderung zu korrelieren.

Laden Sie geräteseitige Protokolle auf Amazon hoch CloudWatch

Sie können historische, geräteseitige Protokolle in Amazon hochladen, CloudWatch um die Aktivitäten eines Geräts vor Ort zu überwachen und zu analysieren. Device-side Protokolle können System-, Anwendungs- und Geräteprotokolldateien enthalten. Dieser Prozess verwendet einen Aktionsparameter für CloudWatch Protokollregeln, um geräteseitige Protokolle in einer vom Kunden definierten Protokollgruppe zu veröffentlichen.

Funktionsweise

Der Prozess beginnt, wenn ein AWS IoT Gerät MQTT-Nachrichten mit formatierten Protokolldateien an ein Thema sendet. AWS IoT Eine AWS IoT Regel überwacht das Nachrichtenthema und sendet die Protokolldateien an eine von Ihnen CloudWatch definierte Logs-Gruppe. Sie können die Informationen dann überprüfen und analysieren.

Themen

- MQTT-Themen
- Regelaktion

MQTT-Themen

Wählen Sie einen Namensraum für MQTT-Themen, den Sie für die Veröffentlichung der Protokolle verwenden werden. Wir empfehlen, das Format `$aws/rules/things/thing_name/logs` für den gemeinsamen Themenbereich und das Format `$aws/rules/things/thing_name/logs/errors` für Fehlerthemen zu verwenden Die Benennungsstruktur für Protokolle und Fehlerthemen wird empfohlen, ist aber nicht erforderlich. Weitere Informationen finden Sie in Der Entwurf von MQTT-Themen für AWS IoT Core.

Indem Sie den empfohlenen Bereich für allgemeine Themen verwenden, verwenden Sie für AWS IoT Basic Ingest reservierte Themen. AWS IoT Basic Ingest sendet Gerätedaten auf sichere Weise an die AWS Dienste, die durch AWS IoT Regelaktionen unterstützt werden. Dadurch wird der publish/subscribe Message Broker aus dem Aufnahmepfad entfernt, was ihn kostengünstiger macht. Weitere Informationen finden Sie unter Senken der Messaging-Kosten mit Basic Ingest.

Wenn Sie BatchMode zum Hochladen von Protokolldateien verwenden, müssen Ihre Nachrichten einem bestimmten Format folgen, das einen UNIX-Zeitstempel und eine UNIX-Nachricht enthält. Weitere Informationen finden Sie im Thema Anforderungen an das MQTT-Nachrichtenformat für BatchMode in der Protokollregelaktion. CloudWatch

Regelaktion

Beim AWS IoT Empfang der MQTT-Nachrichten von den Client-Geräten überwacht eine AWS IoT Regel das vom Kunden definierte Thema und veröffentlicht den Inhalt in einer von Ihnen definierten CloudWatch Protokollgruppe. Dieser Prozess verwendet eine CloudWatch Protokoll-Regelaktion, um MQTT auf Stapel von Protokolldateien zu überwachen. Weitere Informationen finden Sie unter der AWS IoT Regelaktion „CloudWatch Logs“.

Batch-Modus

`batchMode`ist ein boolescher Parameter innerhalb der Regelaktion „ AWS IoT CloudWatch Logs“. Dieser Parameter ist optional und standardmäßig deaktiviert (`false`). Um geräteseitige Protokolldateien stapelweise hochzuladen, müssen Sie diesen Parameter beim Erstellen der Regel aktivieren (`true`). AWS IoT Weitere Informationen finden Sie im Abschnitt „AWS IoT Regelaktionen“ unter CloudWatch Protokolle.

Laden Sie geräteseitige Protokolle hoch mit AWS IoT Regeln

Sie können die AWS IoT Regel-Engine verwenden, um Protokolldatensätze aus vorhandenen geräteseitigen Protokolldateien (System-, Anwendungs- und Geräte-Client-Protokolle) auf Amazon hochzuladen. CloudWatch Wenn geräteseitige Protokolle zu einem MQTT-Thema veröffentlicht werden, werden die Nachrichten mit der Aktion „Regeln CloudWatch protokollieren“ an Logs übertragen. CloudWatch Dieser Prozess beschreibt, wie Geräteprotokolle stapelweise hochgeladen werden, wobei der `batchMode`-Aktionsparameter „Regeln“ aktiviert ist (auf `true` festgelegt).

Erfüllen Sie die folgenden Voraussetzungen, um mit dem Hochladen von geräteseitigen Protokollen zu CloudWatch beginnen.

Voraussetzungen

Bevor Sie beginnen, führen Sie die folgenden Schritte aus:

- Erstellen Sie mindestens ein IoT-Zielgerät, bei dem Sie AWS IoT Core als Ding registriert sind. AWS IoT Weitere Informationen finden Sie unter Ein Objekt erstellen.
- Ermitteln Sie den MQTT-Themenbereich für Erfassung und Fehler. Weitere Informationen zu MQTT-Themen und empfohlenen Namenskonventionen finden Sie im Abschnitt MQTT-Themen MQTT-Themen unter Geräteseitige Protokolle auf Amazon hochladen. CloudWatch

Weitere Informationen zu diesen Voraussetzungen finden Sie unter Geräteseitige Protokolle hochladen auf. CloudWatch

Eine Protokollgruppe erstellen CloudWatch

Gehen Sie wie folgt vor, um eine CloudWatch Protokollgruppe zu erstellen. Wählen Sie die entsprechende Registerkarte aus, je nachdem, ob Sie die Schritte mit AWS-Managementkonsole oder mit AWS Command Line Interface (AWS CLI) ausführen möchten.

AWS-Managementkonsole

Um eine CloudWatch Protokollgruppe zu erstellen, verwenden Sie den AWS-Managementkonsole

1. Öffnen Sie AWS-Managementkonsole und navigieren Sie zu CloudWatch.
2. Wählen Sie im Navigationsbereich Protokolle und dann Protokollgruppen aus.
3. Wählen Sie Protokollgruppe erstellen aus.

4. Aktualisieren Sie den Namen der Protokollgruppe und aktualisieren Sie optional die Felder mit den Aufbewahrungseinstellungen.
5. Wählen Sie Erstellen aus.

AWS CLI

Um eine CloudWatch Protokollgruppe zu erstellen, verwenden Sie den AWS CLI

1. Führen Sie den folgenden Befehl aus, um die Protokollgruppe zu erstellen. Weitere Informationen finden Sie `create-log-group` in der AWS CLI v2-Befehlsreferenz.

   Ersetzen Sie den Namen der Protokollgruppe im Beispiel (`uploadLogsGroup`) durch Ihren bevorzugten Namen.

   ```
   aws logs create-log-group --log-group-name uploadLogsGroup
   ```

2. Um zu überprüfen, ob die Protokollgruppe korrekt erstellt wurde, führen Sie den folgenden Befehl aus.

   ```
   aws logs describe-log-groups --log-group-name-prefix uploadLogsGroup
   ```

   Beispielausgabe:

   ```
   {
       "logGroups": [
           {
               "logGroupName": "uploadLogsGroup",
               "creationTime": 1674521804657,
               "metricFilterCount": 0,
               "arn": "arn:aws:logs:us-east-1:111122223333:log-
   group:uploadLogsGroup:*",
               "storedBytes": 0
           }
       ]
   }
   ```

Erstellen einer Themenregel

Gehen Sie wie folgt vor, um eine AWS IoT Regel zu erstellen. Wählen Sie die entsprechende Registerkarte aus, je nachdem, ob Sie die Schritte mit AWS-Managementkonsole oder mit AWS Command Line Interface (AWS CLI) ausführen möchten.

AWS-Managementkonsole

Um eine Themenregel zu erstellen, verwenden Sie die AWS-Managementkonsole

1. Öffnen Sie den Regel-Hub.
   a. Öffnen Sie die AWS-Managementkonsole und navigieren Sie zu AWS IoT.
   b. Wählen Sie in der Navigationsleiste Nachrichtenweiterleitung und dann Regeln aus.
   c. Wählen Sie Regel erstellen aus.
2. Geben Sie die Regeleigenschaften ein.
   a. Geben Sie einen alphanumerischen Regelnamen ein.
   b. (Optional) Geben Sie eine Regelbeschreibung und Tags ein.
   c. Wählen Sie Weiter aus.
3. Geben Sie eine SQL-Anweisung ein.
   a. Geben Sie eine SQL-Anweisung ein, die das MQTT-Thema verwendet, das Sie für die Erfassung definiert haben.

      Beispiel: `SELECT * FROM '$aws/rules/things/thing_name/logs'`
   b. Wählen Sie Weiter aus.
4. Geben Sie Regelaktionen ein.
   a. Wählen Sie im Menü Aktion 1 die Option CloudWatch Protokolle aus.
   b. Wählen Sie den Protokoll-Gruppennamen aus und wählen Sie dann die Protokollgruppe aus, die Sie erstellt haben.
   c. Wählen Sie Batch-Modus verwenden aus.
   d. Geben Sie die IAM-Rolle für die Regel an.

      Führen Sie die folgenden Schritte aus, wenn Sie eine IAM-Rolle für die Regel besitzen.

      1. Wählen Sie im Menü IAM-Rolle Ihre IAM-Rolle aus.

Führen Sie die folgenden Schritte aus, wenn Sie noch keine IAM-Rolle für die Regel besitzen.

1. Klicken Sie auf Neue Rolle erstellen.
2. Geben Sie unter Rollenname einen eindeutigen Namen ein und wählen Sie Erstellen aus.
3. Vergewissern Sie sich, dass der IAM-Rollenname im Feld IAM-Rolle korrekt ist.

e. Wählen Sie Weiter aus.

5. Überprüfen Sie die Vorlagenkonfiguration.

a. Überprüfen Sie die Einstellungen für die Auftragsvorlage, um sicherzustellen, dass sie korrekt sind.

b. Wählen Sie anschließend Erstellen aus.

AWS CLI

Um eine IAM-Rolle und eine Themenregel zu erstellen, verwenden Sie die AWS CLI

1. Erstellen Sie eine IAM-Rolle, die Rechte für die AWS IoT Regel gewährt.

a. Erstellen Sie eine IAM-Richtlinie.

Führen Sie den folgenden Befehl aus, um eine IAM-Richtlinie zu erstellen. Stellen Sie sicher, dass Sie den Parameterwert `policy-name` aktualisieren. Weitere Informationen finden Sie `create-policy` in der AWS CLI v2-Befehlsreferenz.

> **Note**
>
> Wenn Sie ein Microsoft Windows-Betriebssystem verwenden, müssen Sie möglicherweise die Zeilenendemarkierung (\) durch ein Häkchen (`) oder ein anderes Zeichen ersetzen.

```
aws iam create-policy \
    --policy-name uploadLogsPolicy \
    --policy-document \
```

```
'{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iot:CreateTopicRule",
            "iot:Publish",
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents",
            "logs:GetLogEvents"
        ],
        "Resource": "*"
    }
}'
```

b. Kopieren Sie den Richtlinien-ARN aus Ihrer Ausgabe in einen Texteditor.

Beispielausgabe:

```
{
    "Policy": {
        "PolicyName": "uploadLogsPolicy",
        "PermissionsBoundaryUsageCount": 0,
        "CreateDate": "2023-01-23T18:30:10Z",
        "AttachmentCount": 0,
        "IsAttachable": true,
        "PolicyId": "AAABBBCCCDDDEEEFFFGGG",
        "DefaultVersionId": "v1",
        "Path": "/",
        "Arn": "arn:aws:iam::111122223333:policy/uploadLogsPolicy",
        "UpdateDate": "2023-01-23T18:30:10Z"
    }
}
```

c. Erstellen Sie eine IAM-Rolle und eine Vertrauensrichtlinie.

Führen Sie den folgenden Befehl aus, um eine IAM-Richtlinie zu erstellen. Stellen Sie sicher, dass Sie den Parameterwert `role-name` aktualisieren. Weitere Informationen finden Sie `create-role` in der AWS CLI v2-Befehlsreferenz.

```
aws iam create-role \
--role-name uploadLogsRole \
```

```
--assume-role-policy-document \
'{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}'
```

d. Hängen Sie die IAM-Richtlinie an die Rolle an.

Führen Sie den folgenden Befehl aus, um eine IAM-Richtlinie zu erstellen. Stellen Sie sicher, dass Sie die `role-name` und die Parameterwerte für `policy-arn` aktualisieren. Weitere Informationen finden Sie `attach-role-policy` in der AWS CLI v2-Befehlsreferenz.

```
aws iam attach-role-policy \
--role-name uploadLogsRole \
--policy-arn arn:aws:iam::111122223333:policy/uploadLogsPolicy
```

e. Überprüfen Sie die Rolle.

Um zu überprüfen, ob die IAM-Rolle korrekt erstellt wurde, führen Sie den folgenden Befehl aus. Stellen Sie sicher, dass Sie den Parameterwert `role-name` aktualisieren. Weitere Informationen finden Sie `get-role` in der AWS CLI v2-Befehlsreferenz.

```
aws iam get-role --role-name uploadLogsRole
```

Beispielausgabe:

```
{
    "Role": {
        "Path": "/",
        "RoleName": "uploadLogsRole",
        "RoleId": "AAABBBCCCDDDEEEFFFGGG",
        "Arn": "arn:aws:iam::111122223333:role/uploadLogsRole",
```

```
        "CreateDate": "2023-01-23T19:17:15+00:00",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Sid": "Statement1",
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "iot.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        },
        "Description": "",
        "MaxSessionDuration": 3600,
        "RoleLastUsed": {}
    }
}
```

2. Erstellen Sie eine AWS IoT Themenregel in der AWS CLI.

   a. Führen Sie den folgenden Befehl aus, um eine AWS IoT Themenregel zu erstellen. Stellen Sie sicher`--rule-name`, `sql` dass Sie die `logGroupName` Parameterwerte `descriptionroleARN` ,,, und aktualisieren. Weitere Informationen finden Sie unter create-topic-rule in der v2-Befehlsreferenz. AWS CLI

```
aws iot create-topic-rule \
--rule-name uploadLogsRule \
--topic-rule-payload \
 '{
  "sql":"SELECT * FROM 'rules/things/thing_name/logs'",
  "description":"Upload logs test rule",
  "ruleDisabled":false,
  "awsIotSqlVersion":"2016-03-23",
  "actions":[
   {"cloudwatchLogs":
    {"roleArn":"arn:aws:iam::111122223333:role/uploadLogsRole",
    "logGroupName":"uploadLogsGroup",
    "batchMode":true}
    }
   ]
```

```
}'
```

b. Um zu überprüfen, ob die Regel korrekt erstellt wurde, führen Sie den folgenden Befehl aus. Stellen Sie sicher, dass Sie den Parameterwert `role-name` aktualisieren. Weitere Informationen finden Sie unter https://awscli.amazonaws.com/v2/documentation/api/2.0.34/reference/iot/get-topic-rule.html get-topic-rule in der v2-Befehlsreferenz. AWS CLI

```
aws iot get-topic-rule --rule-name uploadLogsRule
```

Beispielausgabe:

```
{
    "ruleArn": "arn:aws:iot:us-east-1:111122223333:rule/uploadLogsRule",
    "rule": {
        "ruleName": "uploadLogsRule",
        "sql": "SELECT * FROM rules/things/thing_name/logs",
        "description": "Upload logs test rule",
        "createdAt": "2023-01-24T16:28:15+00:00",
        "actions": [
            {
                "cloudwatchLogs": {
                    "roleArn": "arn:aws:iam::111122223333:role/
uploadLogsRole",
                    "logGroupName": "uploadLogsGroup",
                    "batchMode": true
                }
            }
        ],
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23"
    }
}
```

Geräteseitige Protokolle werden gesendet an AWS IoT

Um geräteseitige Protokolle zu senden an AWS IoT

1. Um historische Protokolle an diese zu senden AWS IoT, kommunizieren Sie mit Ihren Geräten, um Folgendes sicherzustellen.

- Die Protokollinformationen werden an den richtigen Themen-Namespace gesendet, wie im Abschnitt Voraussetzungen dieses Verfahrens angegeben.

  Beispiel: `$aws/rules/things/thing_name/logs`

- Die Payload der MQTT-Nachricht ist korrekt formatiert. Weitere Informationen zum Thema MQTT und zur empfohlenen Namenskonvention finden Sie im darin enthaltenen MQTT-Themen Laden Sie geräteseitige Protokolle auf Amazon hoch CloudWatch Abschnitt.

2. Vergewissern Sie sich, dass die MQTT-Nachrichten im AWS IoT MQTT-Client empfangen werden.

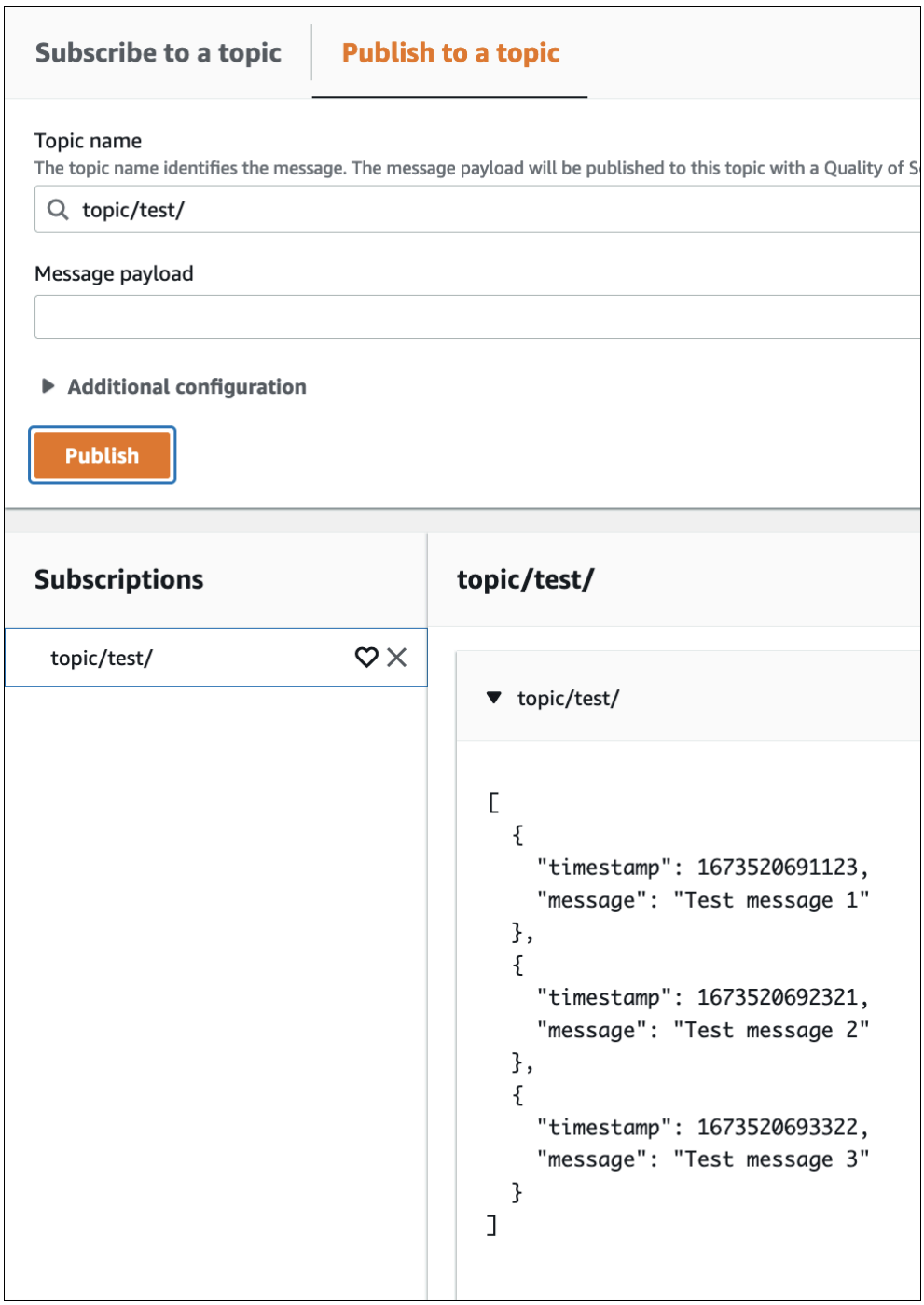
   a. Öffnen Sie das AWS-Managementkonsole und navigieren Sie zu. AWS IoT

   b. Um den MQTT-Testclient anzuzeigen, wählen Sie in der Navigationsleiste Test, MQTT-Testclient.

   c. Geben Sie unter Thema abonnieren, Themenfilter, den Themen-Namespace ein.

   d. Wählen Sie Subscribe (Abonnieren) aus.

      MQTT-Nachrichten werden in der Tabelle Abonnements und Themen angezeigt, wie im Folgenden dargestellt. Der Erscheinen dieser Nachrichten kann bis zu fünf Minuten in Anspruch nehmen.



Anzeige der Protokolldaten

Um Ihre Protokolleinträge in CloudWatch Logs zu überprüfen

1. Öffnen Sie AWS-Managementkonsole und navigieren Sie zu CloudWatch.
2. Wählen Sie in der Navigationsleiste Logs, Logs & Insights aus.

3. Wählen Sie im Menü „Protokollgruppe (n) auswählen“ die Protokollgruppe aus, die Sie in der AWS IoT Regel angegeben haben.
4. Wählen Sie auf der Seite Logs Insights die Option Abfrage ausführen aus.

Protokollierung AWS IoT API-Aufrufe mit AWS CloudTrail

AWS IoT ist in einen Dienst integriert AWS CloudTrail, der eine Aufzeichnung der Aktionen bereitstellt, die von einem Benutzer, einer Rolle oder einem AWS Dienst in ausgeführt wurden AWS IoT. CloudTrail erfasst alle API-Aufrufe AWS IoT als Ereignisse, einschließlich Aufrufe von der AWS IoT Konsole und von Codeaufrufen an die AWS IoT APIs. Wenn Sie einen Trail erstellen, können Sie die kontinuierliche Übertragung von CloudTrail Ereignissen an einen Amazon S3-Bucket aktivieren, einschließlich Ereignissen für AWS IoT. Wenn Sie keinen Trail konfigurieren, können Sie trotzdem die neuesten Ereignisse in der CloudTrail Konsole im Event-Verlauf einsehen. Anhand der gesammelten Informationen können Sie die Anfrage CloudTrail, an die die Anfrage gestellt wurde AWS IoT, die IP-Adresse, von der aus die Anfrage gestellt wurde, wer die Anfrage gestellt hat, wann sie gestellt wurde und andere Details ermitteln.

Weitere Informationen CloudTrail dazu finden Sie im AWS CloudTrail Benutzerhandbuch.

AWS IoT Informationen in CloudTrail

CloudTrail ist auf Ihrem aktiviert AWS-Konto , wenn Sie das Konto erstellen. Wenn eine Aktivität stattfindet AWS IoT, wird diese Aktivität zusammen mit anderen CloudTrail AWS Serviceereignissen im Veranstaltungsverlauf in einem Ereignis aufgezeichnet. Sie können aktuelle Ereignisse in Ihrem anzeigen, suchen und herunterladen AWS-Konto. Weitere Informationen finden Sie unter Ereignisse mit dem CloudTrail Veranstaltungsverlauf anzeigen.

Zur kontinuierlichen Aufzeichnung von Ereignissen in Ihrem AWS-Konto, einschließlich Ereignissen für AWS IoT, erstellen Sie einen Trail. Ein Trail ermöglicht CloudTrail die Übermittlung von Protokolldateien an einen Amazon S3-Bucket. Wenn Sie einen Trail in der Konsole erstellen, gilt der Trail standardmäßig für alle AWS-Region s. Der Trail protokolliert Ereignisse von allen AWS-Region s in der AWS Partition und übermittelt die Protokolldateien an den von Ihnen angegebenen Amazon S3-Bucket. Sie können andere AWS Dienste so konfigurieren, dass sie die in den CloudTrail Protokollen gesammelten Ereignisdaten weiter analysieren und darauf reagieren. Weitere Informationen finden Sie unter:

- Übersicht zum Erstellen eines Trails

- CloudTrail Unterstützte Dienste und Integrationen
- Konfiguration von Amazon SNS-Benachrichtigungen für CloudTrail
- Empfangen von CloudTrail Protokolldateien aus mehreren Regionen und Empfangen von CloudTrail Protokolldateien von mehreren Konten

> **Note**
>
> AWS IoT Aktionen auf Datenebene (geräteseitig) werden von nicht protokolliert CloudTrail. Wird verwendet CloudWatch , um diese Aktionen zu überwachen.

Im Allgemeinen werden Aktionen der AWS IoT Steuerungsebene, durch die Änderungen vorgenommen werden, von protokolliert CloudTrail. Anrufe wie CreateThing CreateKeysAndCertificate, und UpdateCertificate hinterlassen CloudTrail Einträge, Anrufe wie ListThings und ListTopicRules nicht.

Jeder Ereignis- oder Protokolleintrag enthält Informationen zu dem Benutzer, der die Anforderung generiert hat. Die Identitätsinformationen unterstützen Sie bei der Ermittlung der folgenden Punkte:

- Gibt an, ob die Anforderung mit Root- oder IAM-Benutzer-Anmeldeinformationen ausgeführt wurde.
- Gibt an, ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer gesendet wurde.
- Ob die Anfrage von einem anderen AWS Dienst gestellt wurde.

Weitere Informationen finden Sie unter CloudTrail userIdentity-Element.

AWS IoT Aktionen sind in der AWS IoT API-Referenz dokumentiert. AWS IoT Wireless Aktionen sind in der AWS IoT Wireless API-Referenz dokumentiert.

Verstehen AWS IoT Protokolldateieinträge

Ein Trail ist eine Konfiguration, die die Übertragung von Ereignissen als Protokolldateien an einen von Ihnen angegebenen Amazon S3-Bucket ermöglicht. CloudTrail Protokolldateien enthalten einen oder mehrere Protokolleinträge. Ein Ereignis stellt eine einzelne Anforderung aus einer beliebigen Quelle dar und enthält Informationen über die angeforderte Aktion, Datum und Uhrzeit der Aktion, Anforderungsparameter usw. CloudTrail Logdateien sind kein geordneter Stack-Trace der öffentlichen API-Aufrufe, daher erscheinen sie nicht in einer bestimmten Reihenfolge.

Das folgende Beispiel zeigt einen CloudTrail Logeintrag, der die `AttachPolicy` Aktion demonstriert.


```
{
    "timestamp":"1460159496",
    "AdditionalEventData":"",
    "Annotation":"",
    "ApiVersion":"",
    "ErrorCode":"",
    "ErrorMessage":"",
    "EventID":"8bff4fed-c229-4d2d-8264-4ab28a487505",
    "EventName":"AttachPolicy",
    "EventTime":"2016-04-08T23:51:36Z",
    "EventType":"AwsApiCall",
    "ReadOnly":"",
    "RecipientAccountList":"",
    "RequestID":"d4875df2-fde4-11e5-b829-23bf9b56cbcd",
    "RequestParamters":{
        "principal":"arn:aws:iot:us-
east-1:123456789012:cert/528ce36e8047f6a75ee51ab7beddb4eb268ad41d2ea881a10b67e8e76924d894",
        "policyName":"ExamplePolicyForIoT"
    },
    "Resources":"",
    "ResponseElements":"",
    "SourceIpAddress":"52.90.213.26",
    "UserAgent":"aws-internal/3",
    "UserIdentity":{
        "type":"AssumedRole",
        "principalId":"AKIAI44QH8DHBEXAMPLE",
        "arn":"arn:aws:sts::12345678912:assumed-role/iotmonitor-us-east-1-beta-
InstanceRole-1C5T1YCYMHPYT/i-35d0a4b6",
        "accountId":"222222222222",
        "accessKeyId":"access-key-id",
        "sessionContext":{
            "attributes":{
                "mfaAuthenticated":"false",
                "creationDate":"Fri Apr 08 23:51:10 UTC 2016"
            },
            "sessionIssuer":{
                "type":"Role",
                "principalId":"AKIAI44QH8DHBEXAMPLE",
                "arn":"arn:aws:iam::123456789012:role/executionServiceEC2Role/
iotmonitor-us-east-1-beta-InstanceRole-1C5T1YCYMHPYT",
                "accountId":"222222222222",
```

```
                "userName":"iotmonitor-us-east-1-InstanceRole-1C5T1YCYMHPYT"
            }
        },
        "invokedBy":{
            "serviceAccountId":"111111111111"
        }
    },
    "VpcEndpointId":""
}
```

Regeln für AWS IoT

Regeln geben Ihren Geräten die Möglichkeit, mit ihnen zu interagieren AWS-Services. Das Analysieren von Regeln sowie das Ausführen von Aktionen erfolgt ausgehend vom MQTT-Topic-Stream. Sie können Regeln verwenden, um die folgenden Aufgaben zu unterstützen:

- Von einem Gerät empfangene Daten ausweiten oder filtern
- Von einem Gerät empfangene Daten in eine Amazon DynamoDB-Datenbank schreiben
- Speichern Sie eine Datei auf Amazon S3.
- Senden Sie eine Push-Benachrichtigung an alle Benutzer, die Amazon SNS verwenden.
- Veröffentlichen Sie Daten in einer Amazon SQS-Warteschlange.
- Rufen Sie eine Lambda-Funktion zum Extrahieren von Daten auf.
- Nachrichten von einer großen Anzahl an Geräten mithilfe von Amazon Kinesis verarbeiten
- Senden Sie Daten an Amazon OpenSearch Service.
- Erfassen Sie eine CloudWatch Metrik.
- Ändern Sie einen CloudWatch Alarm.
- Senden Sie die Daten aus einer MQTT-Nachricht an Amazon SageMaker AI, um Vorhersagen auf der Grundlage eines Modells für maschinelles Lernen (ML) zu treffen.
- Starten Sie den Prozess eines Step Functions-Zustandsautomaten.
- Senden Sie Nachrichtendaten an eine Komponenteneigenschaft in AWS IoT SiteWise.
- Senden Sie Nachrichtendaten an eine Webanwendung oder einen Dienst.

Ihre Regeln können MQTT-Nachrichten verwenden, die das von der unterstützte publish/subscribe Protokoll durchlaufen. the section called “Gerätekommunikationsprotokolle” Sie können auch die Basic Ingest-Funktion verwenden, um Gerätedaten sicher an die zuvor AWS-Services genannten Adressen zu senden, ohne dass Nachrichtengebühren anfallen. Die Funktion Basic Ingest optimiert den Datenfluss, indem der publish/subscribe Message Broker aus dem Aufnahmepfad entfernt wird. Dies macht sie kostengünstig und behält gleichzeitig die Sicherheits- und Datenverarbeitungsfunktionen von bei. AWS IoT

Bevor Sie diese Aktionen ausführen AWS IoT können, müssen Sie dem Unternehmen die Erlaubnis erteilen, in Ihrem Namen auf Ihre AWS Ressourcen zuzugreifen. Wenn die Aktionen ausgeführt werden, fallen die Standardgebühren für die Aktionen an AWS-Services , die Sie verwenden.

Inhalt

- Zugriffskontrolle
- Rollenberechtigungen weitergeben
- Erstellen eines AWS IoT Regel
- Verwaltung eines AWS IoT Regel
- AWS IoT Regelaktionen
- Fehlerbehebung bei einer Regel
- Zugreifen auf kontoübergreifende Ressourcen mit AWS IoT Regeln
- Fehlerbehandlung (Fehleraktion)
- Senken der Messaging-Kosten mit Basic Ingest
- AWS IoT SQL-Referenz

Zugriffskontrolle

Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt

Verwenden Sie IAM-Rollen, um die AWS Ressourcen zu steuern, auf die jede Regel Zugriff hat. Bevor Sie eine Regel erstellen, müssen Sie eine IAM-Rolle mit einer Richtlinie erstellen, die den Zugriff auf die erforderlichen AWS Ressourcen ermöglicht. AWS IoT übernimmt diese Rolle bei der Implementierung einer Regel.

Führen Sie die folgenden Schritte aus, um die IAM-Rolle zu erstellen, und AWS IoT Richtlinie, die eine AWS IoT regeln Sie den Zugriff, den es benötigt (AWS CLI).

1. Speichern Sie das folgende Dokument mit der Vertrauensrichtlinie, das die AWS IoT Erlaubnis erteilt, die Rolle zu übernehmen, in einer Datei mit dem Name`iot-role-trust.json`.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
        },
            "Action": "sts:AssumeRole",
```

```
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
            },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:iot:us-east-1:123456789012:rule/
rulename"
            }
        }
        }
    ]
}
```

Erstellen Sie mit dem Befehl create-role eine IAM-Rolle und geben Sie die Datei `iot-role-trust.json` an:

```
aws iam create-role --role-name my-iot-role --assume-role-policy-document
 file://iot-role-trust.json
```

Die Ausgabe dieses Befehls sieht wie folgt aus:

```
{
 "Role": {
  "AssumeRolePolicyDocument": "url-encoded-json",
  "RoleId": "AKIAIOSFODNN7EXAMPLE",
  "CreateDate": "2015-09-30T18:43:32.821Z",
  "RoleName": "my-iot-role",
  "Path": "/",
  "Arn": "arn:aws:iam::123456789012:role/my-iot-role"
 }
}
```

2. Speichern Sie den folgenden JSON-Code in einer Datei mit dem Namen `my-iot-policy.json`.

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

```
    "Action": "dynamodb:*",
    "Resource": "*"
   }
  ]
}
```

Dieses JSON ist ein Beispiel für ein Richtliniendokument, das AWS IoT Administratorzugriff auf DynamoDB gewährt.

Verwenden Sie den Befehl create-policy, um AWS IoT Zugriff auf Ihre AWS Ressourcen zu gewähren, sobald Sie die Rolle übernommen haben, indem Sie die Datei übergeben: `my-iot-policy.json`

```
aws iam create-policy --policy-name my-iot-policy --policy-document file://my-iot-policy.json
```

Weitere Informationen darüber, wie Sie Zugriff auf AWS-Services In-Richtlinien für gewähren AWS IoT, finden Sie unter. Erstellen eines AWS IoT Regel

Die Ausgabe des Befehls create-policy enthält den ARN der Richtlinie. Anfügen der Richtlinie an eine Rolle.

```
{
 "Policy": {
  "PolicyName": "my-iot-policy",
  "CreateDate": "2015-09-30T19:31:18.620Z",
  "AttachmentCount": 0,
  "IsAttachable": true,
  "PolicyId": "ZXR6A36LTYANPAI7NJ5UV",
  "DefaultVersionId": "v1",
  "Path": "/",
  "Arn": "arn:aws:iam::123456789012:policy/my-iot-policy",
  "UpdateDate": "2015-09-30T19:31:18.620Z"
 }
}
```

3. Ordnen Sie die Richtlinie mit dem Befehl attach-role-policy Ihrer Rolle zu:

```
aws iam attach-role-policy --role-name my-iot-role --policy-arn
 "arn:aws:iam::123456789012:policy/my-iot-policy"
```

Widerrufen des Zugriffs auf die Rule Engine

Gehen Sie wie folgt vor, um den Zugriff auf die Rule-Engine sofort zu widerrufen:

1. Entfernen Sie iot.amazonaws.com aus der Vertrauensrichtlinie
2. Folgen Sie den Schritten, um IAM-Rollensitzungen zu widerrufen

Beschränkung des HTTP-Aktionsverkehrs auf die IPv6-Bereiche der Rules Engine

Endpunkte, die den eingehenden Datenverkehr anhand der Quelladresse einschränken, müssen den Bereich für Ihre Region aus der folgenden Tabelle zulassen. Dies gilt für HTTP-Aktionsendpunkte, die einzigen Ziele, die Sie so konfigurieren können, dass sie nur über IPv6 erreichbar sind. Bei anderen Aktionen stellt die Regel-Engine über IPv4 eine Verbindung zu Dual-Stack-Zielen her, sodass kein Bereich zulässig ist.

Diese Bereiche gelten nur, wenn die Regel-Engine über IPv6 eine Verbindung zu Ihrem Endpunkt herstellt. Dies ist der Fall, wenn Ihr Endpunkt AAAA-Einträge und keine A-Datensätze veröffentlicht. Da es keinen entsprechenden IPv4-Bereich gibt, ist für die Beschränkung des Regel-Engine-Datenverkehrs anhand der Quelladresse ein Endpunkt erforderlich, der nur über IPv6 erreichbar ist.

Jeder Bereich wird nur für Aktionen der Regel-Engine verwendet und ändert sich nicht. Bereiche für weitere Regionen werden dieser Tabelle hinzugefügt, sobald sie verfügbar sind. Die Schritte zum Empfangen von Daten über IPv6 finden Sie unterNetzwerkzugriff.

> **Note**
>
> Regionen, die nicht in dieser Tabelle aufgeführt sind, haben keinen IPv6-Bereich für die Rules Engine.

Region	IPv6-Adressbereich
us-east-1	`2600:1f33:8000::/48`
us-east-2	`2600:1f33:6000::/48`
us-west-1	`2600:1f33:c000::/48`

Region	IPv6-Adressbereich
us-west-2	2600:1f33:4000::/48
ap-east-1	2406:da33:e000::/48
ap-northeast-1	2406:da33:4000::/48
ap-northeast-2	2406:da33:2000::/48
ap-south-1	2406:da33:a000::/48
ap-southeast-1	2406:da33:8000::/48
ap-southeast-2	2406:da33:c000::/48
ap-southeast-5	2406:da33:800::/48
ca-central-1	2600:1f33:1000::/48
eu-central-1	2a05:d033:4000::/48
eu-north-1	2a05:d033:6000::/48
eu-south-1	2a05:d033:a000::/48
eu-south-2	2a05:d033:1000::/48
eu-west-1	2a05:d033:8000::/48
eu-west-2	2a05:d033:c000::/48
eu-west-3	2a05:d033:2000::/48
il-central-1	2a05:d033:5000::/48
sa-east-1	2600:1f33:e000::/48
us-gov-east-1	2600:1f33:5000::/48
us-gov-west-1	2600:1f33:2000::/48

In den folgenden Beispielen wird eingehender Datenverkehr auf diesen Bereich beschränkt. `us-east-1` Ersetzen Sie ihn durch den Bereich für Ihre Region.

Eine API-Gateway-Ressourcenrichtlinie, die die Regel-Engine zulässt und alles andere verweigert:

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Principal": "*",
   "Action": "execute-api:Invoke",
   "Resource": "execute-api:/*"
  },
  {
   "Effect": "Deny",
   "Principal": "*",
   "Action": "execute-api:Invoke",
   "Resource": "execute-api:/*",
   "Condition": {
    "NotIpAddress": {
     "aws:SourceIp": ["2600:1f33:8000::/48"]
    }
   }
  }
 ]
}
```

Fügen Sie für einen Endpunkt hinter einem Load Balancer oder auf Amazon EC2 der Sicherheitsgruppe eine Regel für eingehenden Datenverkehr hinzu, die TCP-Verkehr auf Port 443 aus dem Bereich für Ihre Region zulässt.

Rollenberechtigungen weitergeben

Zu einer Regeldefinition gehört eine IAM-Rolle, die die Berechtigung zum Zugriff auf Ressourcen gewährt, welche in der Aktion der Regel festgelegt sind. Die Regel-Engine übernimmt diese Rolle, wenn die Aktion der Regel aufgerufen wird. Die Rolle muss genauso definiert sein AWS-Konto wie die Regel.

Bei Erstellen oder Ersetzen einer Rolle übergeben Sie eine Rolle an die Regel-Engine. Die `iam:PassRole` Berechtigung ist erforderlich, um diesen Vorgang durchzuführen. Sie können

sicherstellen, dass Sie über diese Berechtigung verfügen, indem Sie eine Richtlinie erstellen, die die `iam:PassRole` Berechtigung gewährt, und diese an Ihren IAM-Benutzer anfügen. Die folgende Richtlinie zeigt, wie Sie die Berechtigung `iam:PassRole` für eine Rolle erlauben.

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Sid": "Stmt1",
   "Effect": "Allow",
   "Action": [
    "iam:PassRole"
   ],
   "Resource": [
    "arn:aws:iam::123456789012:role/myRole"
   ]
  }
 ]
}
```

In diesem Richtlinienbeispiel wird die Berechtigung `iam:PassRole` für die Rolle `myRole` gewährt. Die Rolle wird mit dem ARN der Rolle angegeben. Fügen Sie diese Richtlinie Ihrem IAM-Benutzer oder der Rolle zu, der Ihr Benutzer angehört. Weitere Informationen finden Sie unter Arbeiten mit verwalteten Richtlinien.

> **Note**
>
> Lambda-Funktionen verwenden eine ressourcenbasierte Richtlinie, wobei die Richtlinie direkt an die Lambda-Funktion selbst angefügt wird. Wenn Sie eine Regel erstellen, die eine Lambda-Funktion aufruft, übergeben Sie keine Rolle, sodass der Benutzer, der die Regel erstellt, die `iam:PassRole` Berechtigung nicht benötigt. Weitere Informationen zur Lambda-Funktionsautorisierung finden Sie unter Gewähren von Berechtigungen mit einer Ressourcenrichtlinie.

Erstellen eines AWS IoT Regel

Sie können AWS IoT Regeln erstellen, um Daten von Ihren verbundenen Geräten an die Interaktion mit anderen AWS Diensten weiterzuleiten. Eine AWS IoT Regel besteht aus den folgenden Komponenten:

Bestandteile einer Regel

Komponente	Description	Erforderlich oder optional
Regelname	Der Name der Regel. Beachten Sie, dass wir die Verwendung personenbezogener Daten in Ihren Regelnamen nicht empfehlen.	Erforderlich
Regelbeschreibung	Eine Textbeschreibung der Regel. Beachten Sie, dass wir die Verwendung personenbezogener Daten in Ihren Regelbeschreibungen nicht empfehlen.	Optional.
SQL-Anweisung	Eine vereinfachte SQL-Syntax zum Filtern von Nachrichten, die zu einem MQTT-Topic empfangen wurden, und zum Übertragen der Daten per Push an einen anderen Speicherort. Weitere Informationen finden Sie unter AWS IoT SQL-Referenz.	Erforderlich
SQL-Version	Die Version der SQL-Regel-Engine, die beim Auswerten der Regel verwendet wird. Obwohl diese Eigenschaft optional ist, empfehlen wir dringend, die SQL-Version anzugeben. Die AWS IoT Core Konsole legt diese Eigenschaft `2016-03-23` standardmäßig auf fest. Wenn diese Eigenschaft nicht festgelegt ist, z. B. in einem AWS CLI Befehl oder einer CloudFormation Vorlage, `2015-10-08` wird sie verwendet. Weitere Informationen finden Sie unter SQL-Versionen.	Erforderlich

Komponente	Description	Erforderlich oder optional
Eine oder mehrere Aktionen	Die Aktion wird AWS IoT ausgeführt, wenn die Regel in Kraft gesetzt wird. Sie können beispielsweise Daten in eine DynamoDB-Tabelle einfügen, Daten zu einem Amazon-S3-Bucket schreiben, Daten in einem Amazon-SNS-Thema veröffentlichen oder eine Lambda-Funktion aufrufen.	Erforderlich
Eine Fehleraktion	Die Aktion AWS IoT wird ausgeführt, wenn die Aktion einer Regel nicht ausgeführt werden kann.	Optional.

Bevor Sie eine AWS IoT Regel erstellen, müssen Sie eine IAM-Rolle mit einer Richtlinie erstellen, die den Zugriff auf die erforderlichen AWS Ressourcen ermöglicht. AWS IoT übernimmt diese Rolle bei der Implementierung einer Regel. Weitere Informationen finden Sie unter Einer AWS IoT Regel den erforderlichen Zugriff gewähren und Rollenberechtigungen weitergeben.

Beachten Sie beim Erstellen einer Regel, wie viele Daten Sie in Themen veröffentlichen. Wenn Sie Regeln erstellen, die ein Themenmuster mit Platzhaltern enthalten, stimmen diese möglicherweise mit einem großen Prozentsatz Ihrer Nachrichten überein. In diesem Fall müssen Sie möglicherweise die Kapazität der AWS Ressourcen erhöhen, die von den Zielaktionen verwendet werden. Wir empfehlen, Themenmuster mit Platzhaltern in Regeln zur erneuten Veröffentlichung zu vermeiden, um doppelte Verarbeitung zu vermeiden und die Kosten zu senken.

> **Note**
>
> Das Erstellen und Aktualisieren von Regeln sind Aktionen auf Administratorebene. Jeder Benutzer mit der Berechtigung zum Erstellen oder Aktualisieren von Regeln kann auf Daten zugreifen, die von den Regeln verarbeitet wurden.

Erstellen Sie eine Regel (Konsole)

Um eine Regel zu erstellen (AWS-Managementkonsole)

Verwenden Sie den AWS-ManagementkonsoleBefehl, um eine Regel zu erstellen:

1. Öffnen Sie die AWS IoT -Konsole.
2. Wählen Sie in der linken Navigationsleiste im Bereich Verwalten die Option Nachrichtenweiterleitung aus. Wählen Sie dann Regeln aus.
3. Wählen Sie auf der Seite Regeln die Option Regel erstellen aus.
4. Geben Sie auf der Seite Regeleigenschaften angeben einen Namen für Ihre Regel ein. Regelbeschreibung und Tags sind optional. Wählen Sie Weiter aus.
5. Wählen Sie auf der Seite „SQL-Anweisung konfigurieren“ eine SQL-Version aus und geben Sie eine SQL-Anweisung ein. Eine Beispiel-SQL-Anweisung kann sein`SELECT temperature FROM 'iot/topic' WHERE temperature > 50`. Weitere Informationen finden Sie unter SQL-Versionen und AWS IoT SQL-Referenz.
6. Fügen Sie auf der Seite „Regelaktionen anhängen“ Regelaktionen hinzu, um Daten an andere AWS Dienste weiterzuleiten.

    1. Wählen Sie unter Regelaktionen eine Regelaktion aus der Dropdownliste aus. Sie können beispielsweise Kinesis Stream wählen. Weitere Informationen zu Regelaktionen finden Sie unter AWS IoT Regelaktionen.
    2. Geben Sie je nach der ausgewählten Regelaktion die entsprechenden Konfigurationsdetails ein. Wenn Sie sich beispielsweise für Kinesis Stream entscheiden, müssen Sie eine Datenstream-Ressource auswählen oder erstellen und optional Konfigurationsdetails wie den Partitionsschlüssel eingeben, mit dem Daten in einem Stream nach Shard gruppiert werden.
    3. Wählen oder erstellen Sie in der IAM-Rolle eine Rolle, um AWS IoT Zugriff auf Ihren Endpunkt zu gewähren. Beachten Sie, dass dadurch AWS IoT automatisch eine Richtlinie mit dem Präfix `aws-iot-rule` unter Ihrer ausgewählten IAM-Rolle erstellt wird. Sie können „Ansicht“ wählen, um Ihre IAM-Rolle und die Richtlinie von der IAM-Konsole aus anzuzeigen. Die Aktion „Fehler“ ist optional. Weitere Informationen finden Sie unter Fehlerbehandlung (Fehleraktion). Weitere Informationen zum Erstellen einer IAM-Rolle für Ihre Regel finden Sie unter Gewähren Sie einer Regel den erforderlichen Zugriff. Wählen Sie Weiter aus.
7. Überprüfen Sie auf der Seite Überprüfen und erstellen die gesamte Konfiguration und nehmen Sie bei Bedarf Änderungen vor. Wählen Sie Erstellen aus.

Nachdem Sie eine Regel erfolgreich erstellt haben, wird die Regel auf der Seite Regeln aufgeführt. Sie können eine Regel auswählen, um die Detailseite zu öffnen, auf der Sie eine Regel anzeigen, bearbeiten, eine Regel deaktivieren und eine Regel löschen können.

Regel erstellen (CLI)

Um eine Regel zu erstellen (AWS CLI)

Verwenden Sie den Befehl create-topic-rule zum Erstellen einer Regel:

```
aws iot create-topic-rule --rule-name myrule --topic-rule-payload file://myrule.json
```

Im Folgenden finden Sie ein Beispiel für eine Nutzlastdatei mit einer Regel, die alle an das `iot/test` Thema gesendeten Nachrichten in die angegebene DynamoDB-Tabelle einfügt. Die SQL-Anweisung filtert die Nachrichten und der Rollen-ARN gewährt die AWS IoT Berechtigung, in die DynamoDB-Tabelle zu schreiben.

```
{
 "sql": "SELECT * FROM 'iot/test'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "dynamoDB": {
    "tableName": "my-dynamodb-table",
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role",
    "hashKeyField": "topic",
    "hashKeyValue": "${topic(2)}",
    "rangeKeyField": "timestamp",
    "rangeKeyValue": "${timestamp()}"
   }
  }
 ]
}
```

Im Folgenden finden Sie ein Beispiel für eine Nutzlastdatei mit einer Regel, die alle an das `iot/test` Thema gesendeten Nachrichten in den angegebenen S3-Bucket einfügt. Die SQL-Anweisung filtert die Nachrichten, und die Rolle ARN gewährt die AWS IoT Erlaubnis, in den Amazon S3 S3-Bucket zu schreiben.

```
{
 "awsIotSqlVersion": "2016-03-23",
 "sql": "SELECT * FROM 'iot/test'",
 "ruleDisabled": false,
```

```
 "actions": [
  {
   "s3": {
    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_s3",
    "bucketName": "amzn-s3-demo-bucket",
    "key": "myS3Key"
   }
  }
 ]
}
```

Im Folgenden finden Sie ein Beispiel für eine Payload-Datei mit einer Regel, die Daten an Amazon OpenSearch Service überträgt:

```
{
 "sql": "SELECT *, timestamp() as timestamp FROM 'iot/test'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "OpenSearch": {
    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_es",
    "endpoint": "https://my-endpoint",
    "index": "my-index",
    "type": "my-type",
    "id": "${newuuid()}"
   }
  }
 ]
}
```

Im Folgenden finden Sie ein Beispiel für eine Nutzlastdatei mit einer Regel, die eine Lambda-Funktion aufruft:

```
{
 "sql": "expression",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "lambda": {
    "functionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function"
```

```
    }
   }
  ]
}
```

Im Folgenden finden Sie ein Beispiel für eine Nutzlastdatei mit einer Regel, die in einem Amazon SNS-Thema veröffentlicht:

```
{
 "sql": "expression",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "sns": {
    "targetArn": "arn:aws:sns:us-west-2:123456789012:my-sns-topic",
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
   }
  }
 ]
}
```

Im Folgenden finden Sie ein Beispiel für eine Nutzlastdatei mit einer Regel, die in einem anderen MQTT-Topic erneut veröffentlicht:

```
{
 "sql": "expression",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "republish": {
    "topic": "my-mqtt-topic",
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
   }
  }
 ]
}
```

Im Folgenden finden Sie ein Beispiel für eine Payload-Datei mit einer Regel, die Daten in einen Amazon Data Firehose überträgt:

```
{
 "sql": "SELECT * FROM 'my-topic'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "firehose": {
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role",
    "deliveryStreamName": "my-stream-name"
   }
  }
 ]
}
```

Im Folgenden finden Sie ein Beispiel für eine Nutzlastdatei mit einer Regel, die eine Ausführung eines Step Functions-Zustandsautomaten startet.

```
{
 "sql": "expression",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "stepFunctions": {
    "stateMachineName": "myCoolStateMachine",
    "executionNamePrefix": "coolRunning",
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
   }
  }
 ]
}
```

Verwaltung eines AWS IoT Regel

Sie können die folgenden Aktionen verwenden, um Ihre AWS IoT Regeln zu verwalten.

In diesem Thema:

- Eine Regel taggen
- Eine Regel anzeigen
- Löschen einer Regel

Eine Regel taggen

Um Ihren neuen oder bestehenden Regeln eine weitere Ebene der Spezifität hinzuzufügen, können Sie sie mit Tags versehen. Beim Taggen werden Schlüssel-Wert-Paare in Ihren Regeln genutzt, sodass Sie besser kontrollieren können, wie und wo Ihre Regeln auf Ihre Ressourcen und Dienste angewendet werden. AWS IoT Sie können den Geltungsbereich Ihrer Regel beispielsweise so einschränken, dass sie nur in Ihrer Beta-Umgebung für Tests vor der Veröffentlichung gilt (`Key=environment, Value=beta`) oder alle Nachrichten, die nur von einem bestimmten Endpunkt an das `iot/test` Thema gesendet wurden, erfasst und in einem Amazon S3-Bucket speichert.

Beispiel für IAM-Richtlinien

Ein Beispiel, das zeigt, wie Tagging-Berechtigungen für eine Regel erteilt werden, stellen Sie sich einen Benutzer vor, der den folgenden Befehl ausführt, um eine Regel zu erstellen und sie so zu kennzeichnen, dass sie nur für seine Beta-Umgebung gilt.

Ersetzen Sie im Beispiel:

- *`MyTopicRuleName`*mit dem Namen der Regel.
- *`myrule.json`*mit dem Namen des Richtliniendokuments.

```
aws iot create-topic-rule
    --rule-name MyTopicRuleName
    --topic-rule-payload file://myrule.json
    --tags "environment=beta"
```

Für dieses Beispiel müssen Sie die folgende IAM-Richtlinie verwenden:

```
{
    "Version":"2012-10-17",
    "Statement":
    {
        "Action": [ "iot:CreateTopicRule", "iot:TagResource" ],
        "Effect": "Allow",
        "Resource": [
            "arn:aws:iot:us-east-1:123456789012:rule/MyTopicRuleName"
        ]
```

```
    }
}
```

Das obige Beispiel zeigt eine neu erstellte Regel namens `MyTopicRuleName`, die nur für Ihre Beta-Umgebung gilt. Die `iot:TagResource` in der Grundsatzerklärung mit `MyTopicRuleName` ausdrücklich genannte Option ermöglicht das Taggen bei der Erstellung oder Aktualisierung von `MyTopicRuleName`. Der bei der Erstellung der Regel verwendete Parameter `--tags "environment=beta"` beschränkt den Geltungsbereich von `MyTopicRuleName` nur auf Ihre Beta-Umgebung. Wenn Sie den Parameter `--tags "environment=beta"` entfernen, gilt `MyTopicRuleName` für alle Umgebungen.

Weitere Informationen zum Erstellen von IAM-Rollen und -Richtlinien für eine AWS IoT Regel finden Sie unter Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt

Allgemeine Informationen zum Tagging von Ressourcen finden Sie unter Taggen Sie Ihre AWS IoT Ressourcen.

Eine Regel anzeigen

Verwenden Sie den Befehl list-topic-rules zum Auflisten Ihrer Regeln:

```
aws iot list-topic-rules
```

Verwenden Sie den Befehl get-topic-rule zum Abrufen von Informationen zu einer Regel:

```
aws iot get-topic-rule --rule-name myrule
```

Löschen einer Regel

Wenn Sie eine Regel nicht mehr benötigen, können Sie sie löschen.

Um eine Regel zu löschen (AWS CLI)

Verwenden Sie den Befehl delete-topic-rule zum Löschen einer Regel:

```
aws iot delete-topic-rule --rule-name myrule
```

AWS IoT Regelaktionen

AWS IoT Regelaktionen geben an, was zu tun ist, wenn eine Regel aufgerufen wird. Sie können Aktionen definieren, um Daten an eine Amazon DynamoDB DynamoDB-Datenbank zu senden, Daten an Amazon Kinesis Data Streams zu senden, eine AWS Lambda Funktion aufzurufen usw. AWS IoT unterstützt die folgenden Aktionen, AWS-Regionen sofern der Service der Aktion verfügbar ist.

Regelaktion	Description	Name in der API
Apache Kafka	Sendet eine Nachricht an einen Apache-Kafka-Cluster.	`kafka`
CloudWatch Alarme	Ändert den Status eines CloudWatch Amazon-Alarms.	`cloudwatchAlarm`
CloudWatch Logs	Sendet eine Nachricht an Amazon CloudWatch Logs.	`cloudwatchLogs`
CloudWatch Metriken	Sendet eine Nachricht an eine CloudWatch Metrik.	`cloudwatchMetric`
DynamoDB	Sendet eine Nachricht an eine DynamoDB-Tabelle.	`dynamoDB`
DynamoDBv2	Sendet Nachrichtendaten an mehrere Spalten in einer DynamoDB-Tabelle.	`dynamoDBv2`
Elasticsearch	Sendet eine Nachricht an einen OpenSearch Endpunkt.	`OpenSearch`
HTTP	Sendet eine Nachricht an einen HTTPS-Endpunkt.	`http`
InfluxDB	Schreibt Zeitreihendaten von einem IoT-Gerät in InfluxDB.	`influxDB`

Regelaktion	Description	Name in der API
AWS IoT SiteWise	Sendet Nachrichtendaten an Asset-Eigenschaften. AWS IoT SiteWise	`iotSiteWise`
Firehose	Sendet eine Nachricht an einen Firehose-Lieferstream.	`firehose`
Kinesis Data Streams	Sendet eine Nachricht an einen Kinesis-Datenstrom.	`kinesis`
Lambda	Ruft eine Lambda-Funktion mit Nachrichtendaten als Eingabe auf.	`lambda`
Speicherort	Sendet Standortdaten an Amazon Location Service.	`location`
OpenSearch	Sendet eine Nachricht an einen Amazon OpenSearch Service-Endpunkt.	`OpenSearch`
Erneut veröffentlichen	Veröffentlicht eine Nachricht erneut in einem anderen MQTT-Thema.	`republish`
S3	Speichert eine Nachricht in einem Amazon Simple Storage Service (Amazon S3)-Bucket.	`s3`
SNS	Veröffentlichen einer Nachricht als Amazon Simple Notification Service (Amazon SNS) Push-Benachrichtigung.	`sns`

Regelaktion	Description	Name in der API
SQS	Sendet eine Nachricht an eine Amazon Simple Queue Service (Amazon SQS) Warteschlange.	`sqs`
Step Functions	Startet eine AWS Step Functions Zustandsmaschine.	`stepFunctions`
the section called “Timestream”	Sendet eine Nachricht an eine Amazon Timestream-Datenbanktabelle.	`timestream`

Hinweise

- Definieren Sie die Regel genauso AWS-Region wie die Ressource eines anderen Dienstes, sodass die Regelaktion mit dieser Ressource interagieren kann.
- Die AWS IoT Regel-Engine kann mehrere Versuche unternehmen, eine Aktion auszuführen, wenn zeitweise Fehler auftreten. Wenn alle Versuche fehlschlagen, wird die Nachricht verworfen und der Fehler ist in Ihren Protokollen verfügbar. CloudWatch Sie können eine Fehleraktion für jede Regel angeben, die aufgerufen wird, nachdem ein Fehler auftritt. Weitere Informationen finden Sie unter Fehlerbehandlung (Fehleraktion).
- Einige Regelaktionen lösen Aktionen in Services aus, die mit AWS Key Management Service (AWS KMS) integriert werden, um die Verschlüsselung von Daten im Ruhezustand zu unterstützen. Wenn Sie einen vom Kunden verwalteten AWS KMS key (KMS-Schlüssel) verwenden, um Daten im Ruhezustand zu verschlüsseln, muss der Dienst die Erlaubnis haben, den KMS-Schlüssel im Namen des Anrufers zu verwenden. Wie Sie die Berechtigungen für Ihren vom Kunden verwalteten KMS-Schlüssel verwalten können, erfahren Sie in den Themen zur Datenverschlüsselung im entsprechenden Servicehandbuch. Weitere Informationen über kundenverwaltete KMS-Schlüssel finden Sie unter AWS Key Management Service Konzepte im AWS Key Management Service Entwicklerhandbuch.

  Sie können jede Funktion oder Ersetzungsvorlage in der SQL-Anweisung einer Fehleraktion verwenden, einschließlich der externen Funktionen: `aws_lambda()`,

`get_dynamodb()`, `get_registry_data()`, `get_thing_shadow()get_secret()`, und `decode()`. Wenn eine Fehleraktion den Aufruf einer externen Funktion erfordert, kann der Aufruf der Fehleraktion zu einer zusätzlichen Rechnung für die externe Funktion führen.

Apache Kafka

Die Apache Kafka (Kafka) -Aktion sendet Nachrichten direkt an Ihr Amazon Managed Streaming for Apache Kafka (Amazon MSK), Apache Kafka-Cluster, die von Drittanbietern wie Confluent Cloud verwaltet werden, oder an selbstverwaltete Apache Kafka-Cluster. Mit Kafka-Regelaktionen können Sie Ihre IoT-Daten an Kafka-Cluster weiterleiten. Auf diese Weise können Sie leistungsstarke Datenpipelines für verschiedene Zwecke einrichten, z. B. für Streaming-Analysen, Datenintegration, Visualisierung und geschäftskritische Geschäftsanwendungen.

> **Note**
>
> Dieses Thema setzt Vertrautheit mit der Apache Kafka-Plattform und verwandten Konzepten voraus. Weitere Informationen zu Apache Kafka finden Sie unter Apache Kafka. MSK Serverless wird nicht unterstützt. Serverlose MSK-Cluster können nur über die IAM-Authentifizierung ausgeführt werden, die die Apache Kafka-Regelaktion derzeit nicht unterstützt. Weitere Informationen zur Konfiguration AWS IoT Core mit Confluent finden Sie unter Leveraging Confluent and Solve IoT Device and Data AWS Management Challenges.

Voraussetzungen

Diese Regelaktion hat die folgenden Anforderungen:

- Eine IAM-Rolle, die die Ausführung der,`ec2:CreateNetworkInterface`,,, `ec2:DescribeNetworkInterfaces` `ec2:CreateNetworkInterfacePermissionec2:DeleteNetworkInterface`, `ec2:DescribeSubnets` und -Operationen übernehmen AWS IoT kann. `ec2:DescribeVpcs` `ec2:DescribeVpcAttribute` `ec2:DescribeSecurityGroups` Diese Rolle erstellt und verwaltet ENIs zu Ihrer Amazon Virtual Private Cloud, um Ihren Kafka-Broker zu erreichen. Weitere Informationen finden Sie unter Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt.

  In der AWS IoT Konsole können Sie eine Rolle auswählen oder erstellen, um die Ausführung dieser Regelaktion AWS IoT Core zu ermöglichen.

Weitere Informationen zu Netzwerkschnittstellen finden Sie unter Elastic-Network-Schnittstellen im Amazon-EC2-Benutzerhandbuch.

Die Richtlinie, die der von Ihnen angegebenen Rolle zugewiesen ist, sollte wie im folgenden Beispiel aussehen.

```
{
    "Version":"2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "ec2:CreateNetworkInterface",
            "ec2:DescribeNetworkInterfaces",
            "ec2:CreateNetworkInterfacePermission",
            "ec2:DeleteNetworkInterface",
            "ec2:DescribeSubnets",
            "ec2:DescribeVpcs",
            "ec2:DescribeVpcAttribute",
            "ec2:DescribeSecurityGroups"
            ],
            "Resource": "*"
        }
    ]
}
```

- Wenn Sie die Anmeldeinformationen speichern AWS Secrets Manager , die für die Verbindung mit Ihrem Kafka-Broker erforderlich sind, müssen Sie eine IAM-Rolle erstellen, die die Ausführung der `secretsmanager:GetSecretValue` AND-Operationen übernehmen AWS IoT Core kann. `secretsmanager:DescribeSecret`

  Die Richtlinie, die der von Ihnen angegebenen Rolle zugewiesen ist, sollte wie im folgenden Beispiel aussehen.

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

```
                "secretsmanager:DescribeSecret"
            ],
            "Resource": [
                "arn:aws:secretsmanager:us-
east-1:123456789012:secret:kafka_client_truststore-*",
                "arn:aws:secretsmanager:us-
east-1:123456789012:secret:kafka_keytab-*"
            ]
        }
    ]
}
```

- Sie können Ihre Apache Kafka-Cluster in Amazon Virtual Private Cloud (Amazon VPC) ausführen. Sie müssen ein Apache Kafka Virtual Private Cloud (VPC) -Ziel erstellen und ein NAT-Gateway in Ihren Subnetzen verwenden, um Nachrichten von an einen öffentlichen AWS IoT Kafka-Cluster weiterzuleiten. Die AWS IoT Regel-Engine erstellt in jedem der im Ziel aufgelisteten Subnetze eine Netzwerkschnittstelle, um den Verkehr direkt an die VPC weiterzuleiten. Wenn Sie das Ziel erreichen, erstellt die AWS IoT Regel-Engine automatisch eine VPC-Regelaktion. Weitere Informationen zu VPC-Regelaktionen finden Sie unter Apache Kafka Virtual Private Cloud (VPC)-Ziele.
- Wenn Sie einen vom Kunden verwalteten AWS KMS key (KMS-Schlüssel) verwenden, um Daten im Ruhezustand zu verschlüsseln, muss der Dienst die Erlaubnis haben, den KMS-Schlüssel im Namen des Anrufers zu verwenden. Weitere Informationen finden Sie unter Amazon-MSK-Verschlüsselung im Entwicklerhandbuch für Amazon Managed Streaming for Apache Kafka.

Parameters

Wenn Sie mit dieser Aktion eine AWS IoT Regel erstellen, müssen Sie die folgenden Informationen angeben:

destinationArn

Der Amazon-Ressourcenname (ARN) des Apache Kafka Virtual Private Cloud (VPC) -Ziels. Informationen zum Erstellen eines Ziels finden Sie unter. Apache Kafka Virtual Private Cloud (VPC)-Ziele

Thema

Das Kafka-Thema für Nachrichten, die an den Kafka-Broker gesendet werden sollen.

Sie können dieses Feld durch eine Ersatzvorlage ersetzen. Weitere Informationen finden Sie unter the section called “Ersetzungsvorlagen”.

Schlüssel (optional)

Der Kafka-Nachrichtenschlüssel.

Sie können dieses Feld durch eine Ersatzvorlage ersetzen. Weitere Informationen finden Sie unter the section called “Ersetzungsvorlagen”.

Überschriften (optional)

Die Liste der Kafka-Header, die Sie festlegen. Jeder Header ist ein Schlüssel-Wert-Paar, das Sie beim Erstellen einer Kafka-Aktion angeben können. Sie können diese Header verwenden, um Daten von IoT-Clients an Downstream-Kafka-Cluster weiterzuleiten, ohne Ihre Nachrichtennutzlast zu ändern.

Sie können dieses Feld durch eine Ersatzvorlage ersetzen. Um zu verstehen, wie die Funktion einer Inline-Regel als Ersatzvorlage in den Header von Kafka Action übergeben wird, siehe Beispiele. Weitere Informationen finden Sie unter the section called “Ersetzungsvorlagen”.

> **Note**
>
> Header im Binärformat werden nicht unterstützt.

Partition (optional)

Die Kafka-Nachrichten-Partition.

Sie können dieses Feld durch eine Ersatzvorlage ersetzen. Weitere Informationen finden Sie unter the section called “Ersetzungsvorlagen”.

clientProperties

Ein Objekt, das die Eigenschaften des Apache Kafka-Producer-Clients definiert.

acks (optional)

Die Anzahl der Bestätigungen, die der Hersteller beim Server erhalten haben muss, bevor eine Anfrage als abgeschlossen betrachtet werden kann.

Wenn Sie 0 als Wert angeben, wartet der Producer nicht auf eine Bestätigung vom Server. Wenn der Server die Nachricht nicht empfängt, versucht der Producer nicht erneut, die Nachricht zu senden.

Zulässige Werte: `-1`, `0`, `1`, `all`. Der Standardwert ist `1`.

bootstrap.servers

Eine Liste von Host- und Port-Paaren (z. B.`host1:port1`, `host2:port2`), die verwendet wurden, um die erste Verbindung zu Ihrem Kafka-Cluster herzustellen.

compression.type (optional)

Der Komprimierungstyp für alle vom Hersteller generierten Daten.

Zulässige Werte: `none`, `gzip`, `snappy`, `lz4`, `zstd`. Der Standardwert ist `none`.

security.protocol

Das Sicherheitsprotokoll, das für die Verbindung mit Ihrem Kafka-Broker verwendet wird.

Zulässige Werte: `SSL`, `SASL_SSL`. Der Standardwert ist `SSL`.

key.serializer

Gibt an, wie die Schlüsselobjekte, die Sie mit `ProducerRecord` bereitstellen, in Bytes umgewandelt werden sollen.

Zulässiger Wert: `StringSerializer`.

value.serializer

Gibt an, wie Wertobjekte, die Sie mit dem `ProducerRecord` bereitstellen, in Bytes umgewandelt werden sollen.

Zulässiger Wert: `ByteBufferSerializer`.

ssl.truststore

Die Truststore-Datei im Base64-Format oder der Speicherort der Truststore-Datei in AWS Secrets Manager. Dieser Wert ist nicht erforderlich, wenn Ihr Truststore von den Amazon-Zertifizierungsstellen (CA) als vertrauenswürdig eingestuft wird.

Dieses Feld unterstützt Substitutionsvorlagen. Wenn Sie Secrets Manager verwenden, um die Anmeldeinformationen zu speichern, die für die Verbindung mit Ihrem Kafka-Broker erforderlich sind, können Sie die `get_secret` SQL-Funktion verwenden, um den Wert für

dieses Feld abzurufen. Weitere Informationen zu Ersatzvorlagen finden Sie unter the section called “Ersetzungsvorlagen”. Weitere Informationen zur `get_secret` SQL Funktion finden Sie unter the section called “get_secret (secretId, geheimer Typ, Schlüssel, roleArn)”. Wenn der Truststore in Form einer Datei vorliegt, verwenden Sie den `SecretBinary` Parameter. Wenn der Truststore die Form einer Zeichenfolge hat, verwenden Sie den `SecretString` Parameter.

Die maximale Größe dieses Wertes beträgt 65 KB.

ssl.truststore.password

Das Passwort für den Truststore Dieser Wert ist nur erforderlich, wenn Sie ein Passwort für den Truststore erstellt haben.

ssl.keystore

Die Keystore-Datei. Dieser Wert ist erforderlich, wenn Sie `SSL` als Wert für `security.protocol` angeben.

Dieses Feld unterstützt Substitutionsvorlagen. Verwenden Sie Secrets Manager, um die Anmeldeinformationen zu speichern, die für die Verbindung mit Ihrem Kafka-Broker erforderlich sind. Verwenden Sie die `get_secret` SQL-Funktion, um den Wert für dieses Feld abzurufen. Weitere Informationen zu Ersatzvorlagen finden Sie unter the section called “Ersetzungsvorlagen”. Weitere Informationen zur `get_secret` SQL Funktion finden Sie unter the section called “get_secret (secretId, geheimer Typ, Schlüssel, roleArn)”. Verwenden Sie den Parameter `SecretBinary`.

ssl.keystore.password

Das Speicherpasswort für die Keystore-Datei. Dieser Wert ist erforderlich, wenn Sie einen Wert für `ssl.keystore` angeben.

Der Wert dieses Feldes kann Klartext sein. Dieses Feld unterstützt auch Ersatzvorlagen. Verwenden Sie Secrets Manager, um die Anmeldeinformationen zu speichern, die für die Verbindung mit Ihrem Kafka-Broker erforderlich sind. Verwenden Sie die `get_secret` SQL-Funktion, um den Wert für dieses Feld abzurufen. Weitere Informationen zu Ersatzvorlagen finden Sie unter the section called “Ersetzungsvorlagen”. Weitere Informationen zur `get_secret` SQL Funktion finden Sie unter the section called “get_secret (secretId, geheimer Typ, Schlüssel, roleArn)”. Verwenden Sie den Parameter `SecretString`.

ssl.key.password

Das Passwort des privaten Schlüssels in Ihrer Keystore-Datei.

Dieses Feld unterstützt Substitutionsvorlagen. Verwenden Sie Secrets Manager, um die Anmeldeinformationen zu speichern, die für die Verbindung mit Ihrem Kafka-Broker erforderlich sind. Verwenden Sie die `get_secret` SQL-Funktion, um den Wert für dieses Feld abzurufen. Weitere Informationen zu Ersatzvorlagen finden Sie unter the section called “Ersetzungsvorlagen”. Weitere Informationen zur `get_secret` SQL Funktion finden Sie unter the section called “get_secret (secretId, geheimer Typ, Schlüssel, roleArn)”. Verwenden Sie den Parameter `SecretString`.

sasl.mechanik

Der Sicherheitsmechanismus, der für die Verbindung zu Ihrem Kafka-Broker verwendet wird. Dieser Wert ist erforderlich, wenn Sie `SASL_SSL` für `security.protocol` angeben.

Zulässige Werte: `PLAIN`, `SCRAM-SHA-512`, `GSSAPI`.

> **Note**
>
> `SCRAM-SHA-512` ist der einzige unterstützte Sicherheitsmechanismus in den -Regionen cn-north-1, cn-northwest-1, us-gov-east-1 und us-gov-west-1.

sasl.plain.username

Der Benutzername, der verwendet wird, um die geheime Zeichenfolge vom Secrets Manager abzurufen. Dieser Wert ist erforderlich, wenn Sie `SASL_SSL` für `security.protocol` und `PLAIN` für `sasl.mechanism` angeben.

sasl.plain.password

Das Passwort, das zum Abrufen der geheimen Zeichenfolge von Secrets Manager verwendet wird. Dieser Wert ist erforderlich, wenn Sie `SASL_SSL` für `security.protocol` und `PLAIN` für `sasl.mechanism` angeben.

sasl.scram.username

Der Benutzername, der verwendet wird, um die geheime Zeichenfolge vom Secrets Manager abzurufen. Dieser Wert ist erforderlich, wenn Sie `SASL_SSL` für `security.protocol` und `SCRAM-SHA-512` für `sasl.mechanism` angeben.

sasl.scram.password

Das Passwort, das zum Abrufen der geheimen Zeichenfolge von Secrets Manager verwendet wird. Dieser Wert ist erforderlich, wenn Sie `SASL_SSL` für `security.protocol` und `SCRAM-SHA-512` für `sasl.mechanism` angeben.

sasl.kerberos.keytab

Die Keytab-Datei für die Kerberos-Authentifizierung in Secrets Manager. Dieser Wert ist erforderlich, wenn Sie `SASL_SSL` für `security.protocol` und `GSSAPI` für `sasl.mechanism` angeben.

Dieses Feld unterstützt Substitutionsvorlagen. Verwenden Sie Secrets Manager, um die Anmeldeinformationen zu speichern, die für die Verbindung mit Ihrem Kafka-Broker erforderlich sind. Verwenden Sie die `get_secret` SQL-Funktion, um den Wert für dieses Feld abzurufen. Weitere Informationen zu Ersatzvorlagen finden Sie unter the section called “Ersetzungsvorlagen”. Weitere Informationen zur `get_secret` SQL Funktion finden Sie unter the section called “get_secret (secretId, geheimer Typ, Schlüssel, roleArn)”. Verwenden Sie den Parameter `SecretBinary`.

sasl.kerberos.service.name

Der Kerberos-Prinzipalname, unter dem Apache Kafka ausgeführt wird. Dieser Wert ist erforderlich, wenn Sie `SASL_SSL` für `security.protocol` und `GSSAPI` für `sasl.mechanism` angeben.

sasl.kerberos.krb5.kdc

Der Hostname des Key Distribution Center (KDC), mit dem Ihr Apache Kafka Producer-Client eine Verbindung herstellt. Dieser Wert ist erforderlich, wenn Sie `SASL_SSL` für `security.protocol` und `GSSAPI` für `sasl.mechanism` angeben.

sasl.kerberos.krb5.realm

Der Realm, mit dem Ihr Apache Kafka Producer-Client eine Verbindung herstellt. Dieser Wert ist erforderlich, wenn Sie `SASL_SSL` für `security.protocol` und `GSSAPI` für `sasl.mechanism` angeben.

sasl.kerberos.principal

Die eindeutige Kerberos-Identität, der Kerberos Tickets für den Zugriff auf Dienste zuweisen kann. Kerberos-aware Dieser Wert ist erforderlich, wenn Sie `SASL_SSL` für `security.protocol` und `GSSAPI` für `sasl.mechanism` angeben.

Beispiele

Das folgende JSON-Beispiel definiert eine Apache Kafka-Aktion in einer Regel. AWS IoT Im folgenden Beispiel wird die Inline-Funktion sourceIp () als Ersatzvorlage im Kafka Action-Header übergeben.

```
{
 "topicRulePayload": {
  "sql": "SELECT * FROM 'some/topic'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
   {
    "kafka": {
     "destinationArn": "arn:aws:iot:region:123456789012:ruledestination/vpc/
VPCDestinationARN",
     "topic": "TopicName",
     "clientProperties": {
      "bootstrap.servers": "kafka.com:9092",
      "security.protocol": "SASL_SSL",
      "ssl.truststore": "${get_secret('kafka_client_truststore',
 'SecretBinary','arn:aws:iam::123456789012:role/kafka-get-secret-role-name')}",
      "ssl.truststore.password": "kafka password",
      "sasl.mechanism": "GSSAPI",
      "sasl.kerberos.service.name": "kafka",
      "sasl.kerberos.krb5.kdc": "kerberosdns.com",
      "sasl.kerberos.keytab": "${get_secret('kafka_keytab','SecretBinary',
 'arn:aws:iam::123456789012:role/kafka-get-secret-role-name')}",
      "sasl.kerberos.krb5.realm": "KERBEROSREALM",
      "sasl.kerberos.principal": "kafka-keytab/kafka-keytab.com"
     },
     "headers": [
      {
       "key": "static_header_key",
       "value": "static_header_value"
      },
      {
       "key": "substitutable_header_key",
       "value": "${value_from_payload}"
      },
      {
       "key": "source_ip",
       "value": "${sourceIp()}"
```

```
      }
     ]
    }
   }
  ]
 }
}
```

Wichtige Hinweise zu Ihrem Kerberos-Setup

- Ihr Key Distribution Center (KDC) muss über ein privates Domain Name System (DNS) innerhalb Ihrer Ziel-VPC auflösbar sein. Ein möglicher Ansatz besteht darin, den KDC-DNS-Eintrag zu einer privaten gehosteten Zone hinzuzufügen. Weitere Informationen zu diesem Ansatz finden Sie unter Arbeiten mit privat gehosteten Zonen.
- Für jede VPC muss die DNS-Auflösung aktiviert sein. Weitere Informationen finden Sie unter Verwenden von DNS in Ihrer VPC.
- Sicherheitsgruppen für Netzwerkschnittstellen und Sicherheitsgruppen auf Instance-Ebene im VPC-Ziel müssen Datenverkehr aus Ihrer VPC an den folgenden Ports zulassen.
  - TCP-Verkehr auf dem Bootstrap-Broker-Listener-Port (häufig 9092, muss aber im Bereich 9000—9100 liegen)
  - TCP- und UDP-Verkehr auf Port 88 für das KDC
- `SCRAM-SHA-512` ist der einzige unterstützte Sicherheitsmechanismus in den -Regionen cn-north-1, cn-northwest-1, us-gov-east-1 und us-gov-west-1.

Apache Kafka Virtual Private Cloud (VPC)-Ziele

Die Apache Kafka-Regelaktion leitet Daten an einen Apache Kafka-Cluster in einer Amazon Virtual Private Cloud (Amazon VPC). Die von der Apache Kafka-Regelaktion verwendete VPC-Konfiguration wird automatisch aktiviert, wenn Sie das VPC-Ziel für Ihre Regelaktion angeben.

Ein Apache Kafka Virtual Private Cloud (VPC) -Ziel enthält eine Liste von Subnetzen innerhalb der VPC. Die Regel-Engine erstellt eine ENI in jedem Subnetz, das Sie in dieser Liste angeben. Weitere Informationen zu Netzwerkschnittstellen finden Sie unter Elastic-Network-Schnittstellen im Amazon-EC2-Benutzerhandbuch.

Anforderungen und Überlegungen

- Wenn Sie einen selbstverwalteten Apache Kafka-Cluster verwenden, auf den über einen öffentlichen Endpunkt im Internet zugegriffen wird:
  - Erstellen Sie ein NAT-Gateway für Instances in Ihren Subnetzen. Das NAT-Gateway verfügt über eine öffentliche IP-Adresse, die eine Verbindung zum Internet herstellen kann. Dadurch kann die Regel-Engine Ihre Nachrichten an den öffentlichen Kafka-Cluster weiterleiten.
  - Weisen Sie den Elastic Network Interfaces (ENIs), die vom Apache Kafka Virtual Private Cloud (VPC) -Ziel erstellt wurden, eine Elastic IP-Adresse zu. Die Sicherheitsgruppen, die Sie verwenden, müssen so konfiguriert sein, dass sie eingehenden Datenverkehr blockieren.

  > **Note**
  >
  > Wenn das Apache Kafka Virtual Private Cloud (VPC) -Ziel deaktiviert und dann wieder aktiviert wird, müssen Sie die Elastic IPs den neuen ENIs erneut zuordnen.

- Wenn ein Apache Kafka Virtual Private Cloud (VPC) -Ziel 7 Tage hintereinander keinen Traffic empfängt, wird es deaktiviert.
- Wenn sich Ressourcen ändern, die vom Apache Kafka Virtual Private Cloud (VPC) -Ziel verwendet werden, wird das Ziel deaktiviert und kann nicht verwendet werden.
- Zu den Änderungen, die ein Apache Kafka Virtual Private Cloud (VPC) -Ziel deaktivieren können, gehören:
  - Löschen der VPC, der Subnetze, Sicherheitsgruppen oder der verwendeten Rolle.
  - Änderung der Rolle, sodass sie nicht mehr über die erforderlichen Berechtigungen verfügt.
  - Wir haben fast die Subnetzkapazität erreicht, sodass wir kein FedRAMP-Patching mehr anwenden können.
  - Das Ziel wird deaktiviert.

Preisgestaltung

Aus Preisgründen wird eine VPC-Regelaktion zusätzlich zu der Aktion gemessen, die eine Nachricht an eine Ressource sendet, wenn sich die Ressource in Ihrer VPC befindet. Preisinformationen finden Sie unter AWS IoT Core Preise.

Apache Kafka Virtual Private Cloud (VPC) -Ziele erstellen

Sie erstellen ein Apache Kafka Virtual Private Cloud (VPC) -Ziel mithilfe der CreateTopicRuleDestinationAPI oder der AWS IoT Core Konsole.

Wenn Sie ein Ziel erstellen, müssen Sie die folgenden Informationen angeben.

vpcId

Die eindeutige ID der Amazon VPC.

subnetIds

Eine Liste von Subnetzen, in denen die Regel-Engine ENIs erstellt. Die Regel-Engine weist jedem Subnetz in der Liste eine einzelne Netzwerkschnittstelle zu.

> **Note**
>
> Sie können beim Erstellen eines Ziels keine Subnetze in der Verfügbarkeitszone use1-az3 verwenden.

securityGroups (optional)

Eine Liste von Sicherheitsgruppen, die auf die Netzwerkschnittstellen anzuwenden sind

roleArn

Der Amazon-Ressourcenname (ARN) einer Rolle, die die Berechtigung zum Erstellen von Netzwerkschnittstellen in Ihrem Namen erteilt.

Diesem ARN sollte eine Richtlinie angehängt sein, die wie im folgenden Beispiel aussieht.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeVpcs",
```

```
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeSecurityGroups"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ec2:CreateNetworkInterfacePermission",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ec2:ResourceTag/VPCDestinationENI": "true"
                }
        }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ec2:CreateAction": "CreateNetworkInterface",
                    "aws:RequestTag/VPCDestinationENI": "true"
                }
        }
        }
    ]
}
```

Erstellen eines Apache Kafka Virtual Private Cloud (VPC) -Ziels mithilfe von AWS CLI

Das folgende Beispiel zeigt, wie Sie ein Ziel mithilfe von erstellen. AWS CLI

```
aws --region regions iot create-topic-rule-destination --destination-configuration
 'vpcConfiguration={subnetIds=["subnet-
12345678910123045 6"],securityGroups=[],vpcId="vpc-
12345678910123045 6",roleArn="arn:aws:iam::123456789012:role/role-name"}'
```

Nachdem Sie diesen Befehl ausgeführt haben, lautet der ZielstatusIN_PROGRESS. Nach einigen Minuten ändert sich der Status entweder auf ERROR (falls der Befehl nicht erfolgreich ist) oder ENABLED. Wenn der Zielstatus ENABLED lautet, ist er einsatzbereit.

Sie können den folgenden Befehl verwenden, um den Status Ihres Apache Kafka Virtual Private Cloud (VPC) -Ziels abzurufen.

```
aws --region region iot get-topic-rule-destination --arn "VPCDestinationARN"
```

Erstellen eines Apache Kafka Virtual Private Cloud (VPC) -Ziels mithilfe der AWS IoT Core Konsole

In den folgenden Schritten wird beschrieben, wie Sie mithilfe der AWS IoT Core Konsole ein Ziel erstellen.

1. Navigieren Sie zur AWS IoT Core Konsole. Wählen Sie im linken Bereich auf der Registerkarte Aktion die Option Ziele aus.
2. Geben Sie Werte für folgende Felder ein.
   - VPC-ID
   - Subnetz-IDs
   - Sicherheitsgruppe
3. Wählen Sie eine Rolle aus, die über die erforderlichen Berechtigungen zum Erstellen von Netzwerkschnittstellen verfügt. Die obige Beispielrichtlinie enthält diese Berechtigungen.

Wenn der Zielstatus der Apache Kafka Virtual Private Cloud (VPC) AKTIVIERT ist, ist er einsatzbereit.

CloudWatch Alarme

Die Aktion CloudWatch alarm (`cloudWatchAlarm`) ändert den Status eines CloudWatch Amazon-Alarms. Sie können den Grund für die Zustandsänderung und den Wert in diesem Aufruf angeben.

Voraussetzungen

Diese Regelaktion hat die folgenden Anforderungen:

- Eine IAM-Rolle, die die Ausführung des `cloudwatch:SetAlarmState` Vorgangs übernehmen AWS IoT kann. Weitere Informationen finden Sie unter Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt.

  In der AWS IoT Konsole können Sie eine Rolle auswählen oder erstellen, um die Ausführung dieser Regelaktion AWS IoT zu ermöglichen.

Parameters

Wenn Sie eine AWS IoT Regel mit dieser Aktion erstellen, müssen Sie die folgenden Informationen angeben:

`alarmName`

Der Name des CloudWatch Alarms.

Unterstützt Substitutionsvorlagen: API und nur AWS CLI

`stateReason`

Der Grund für die Alarmänderung

Unterstützt Ersatzvorlagen: Ja

`stateValue`

Der Wert des Alarmzustands. Zulässige Werte: `OK`, `ALARM`, `INSUFFICIENT_DATA`.

Unterstützt Ersatzvorlagen: Ja

`roleArn`

Die IAM-Rolle, die den Zugriff auf den CloudWatch Alarm ermöglicht. Weitere Informationen finden Sie unter Voraussetzungen.

Unterstützt Ersatzvorlagen: Nein

Beispiele

Das folgende JSON-Beispiel definiert eine CloudWatch Alarmaktion in einer AWS IoT Regel.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
```

```
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "cloudwatchAlarm": {
                    "alarmName": "IotAlarm",
                    "stateReason": "Temperature stabilized.",
                    "stateValue": "OK",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw"
                }
            }
        ]
    }
}
```

Weitere Informationen finden Sie auch unter

- Was ist Amazon CloudWatch? im CloudWatch Amazon-Benutzerhandbuch
- Verwenden von CloudWatch Amazon-Alarmen im CloudWatch Amazon-Benutzerhandbuch

CloudWatch Logs

Die Aktion CloudWatch Logs (`cloudwatchLogs`) sendet Daten an Amazon CloudWatch Logs. Sie können `batchMode` verwenden, um mehrere Geräteprotokolldatensätze in einer Nachricht hochzuladen und mit einem Zeitstempel zu versehen. Sie können auch die Protokollgruppe angeben, an die die Aktion Daten sendet.

Voraussetzungen

Diese Regelaktion hat die folgenden Anforderungen:

- Eine IAM-Rolle, die die Ausführung der `logs:PutLogEvents` Operationen `logs:CreateLogStreamlogs:DescribeLogStreams`, und übernehmen AWS IoT kann. Weitere Informationen finden Sie unter Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt.

  In der AWS IoT Konsole können Sie eine Rolle auswählen oder erstellen, um die Ausführung dieser Regelaktion AWS IoT zu ermöglichen.
- Wenn Sie einen vom Kunden verwalteten AWS KMS key KMS-Schlüssel verwenden, um Protokolldaten in CloudWatch Logs zu verschlüsseln, muss der Dienst die Erlaubnis haben, den

KMS-Schlüssel im Namen des Anrufers zu verwenden. Weitere Informationen finden Sie unter Verschlüsseln von Protokolldaten in CloudWatch Logs using AWS KMS im Amazon CloudWatch Logs-Benutzerhandbuch.

Anforderungen an das MQTT-Nachrichtenformat für BatchMode

Wenn Sie die Regelaktion „ CloudWatch Logs“ bei `batchMode` ausgeschalteter Option verwenden, gibt es keine Anforderungen an die Formatierung von MQTT-Nachrichten. (Hinweis: Der Standardwert des `batchMode` Parameters ist `false`.) Wenn Sie jedoch die Regelaktion CloudWatch Logs mit `batchMode` aktivierter Option verwenden (der Parameterwert ist`true`), müssen MQTT-Nachrichten, die geräteseitige Logs enthalten, so formatiert werden, dass sie einen Zeitstempel und eine Nachrichten-Payload enthalten. Hinweis: `timestamp` stellt den Zeitpunkt dar, zu dem das Ereignis eingetreten ist, und wird als Anzahl der Millisekunden nach dem 1. Januar 1970 00:00:00 UTC ausgedrückt.

Nachfolgend sehen Sie ein Beispiel des Veröffentlichungsformats:

```
[
  {"timestamp": 1673520691093, "message": "Test message 1"},
  {"timestamp": 1673520692879, "message": "Test message 2"},
  {"timestamp": 1673520693442, "message": "Test message 3"}
]
```

Je nachdem, wie die geräteseitigen Protokolle generiert werden, müssen sie möglicherweise gefiltert und neu formatiert werden, bevor sie gesendet werden, um diese Anforderung zu erfüllen. Weitere Informationen finden Sie unter MQTT-Nachrichtennutzlast.

Unabhängig vom `batchMode` Parameter muss der `message` Inhalt den Größenbeschränkungen für Nachrichten entsprechen. AWS IoT Weitere Informationen finden Sie unter AWS IoT Core Endpunkte und -Kontingente.

Parameters

Wenn Sie mit dieser Aktion eine AWS IoT Regel erstellen, müssen Sie die folgenden Informationen angeben:

`logGroupName`

Die CloudWatch Protokollgruppe, in die die Aktion Daten sendet.

Unterstützt Ersatzvorlagen: API und nur AWS CLI

`roleArn`

Die IAM-Rolle, die den Zugriff auf die CloudWatch Protokollgruppe ermöglicht. Weitere Informationen finden Sie unter Voraussetzungen.

Unterstützt Ersatzvorlagen: Nein

(Optional) `batchMode`

Gibt an, ob Stapel von Protokolldatensätzen extrahiert und in diese hochgeladen werden. CloudWatch Zu den Werten gehört `true` oder `false` (Standard). Weitere Informationen finden Sie unter Voraussetzungen.

Unterstützt Ersatzvorlagen: Nein

Beispiele

Das folgende JSON-Beispiel definiert eine CloudWatch Logs-Aktion in einer AWS IoT Regel.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "cloudwatchLogs": {
                    "logGroupName": "IotLogs",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw",
                    "batchMode": false
                }
            }
        ]
    }
}
```

Weitere Informationen finden Sie auch unter

- Was ist Amazon CloudWatch Logs? im Amazon CloudWatch Logs-Benutzerhandbuch

CloudWatch Metriken

Die Aktion CloudWatch metric (`cloudwatchMetric`) erfasst eine CloudWatch Amazon-Metrik. Sie können den Namespace, den Namen, den Wert, die Einheit und den Zeitstempel der Metrik angeben.

Voraussetzungen

Diese Regelaktion hat die folgenden Anforderungen:

- Eine IAM-Rolle, die die Ausführung des `cloudwatch:PutMetricData` Vorgangs übernehmen AWS IoT kann. Weitere Informationen finden Sie unter Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt.

  In der AWS IoT Konsole können Sie eine Rolle auswählen oder erstellen, um die Ausführung dieser Regelaktion AWS IoT zu ermöglichen.

Parameters

Wenn Sie eine AWS IoT Regel mit dieser Aktion erstellen, müssen Sie die folgenden Informationen angeben:

`metricName`

Der Name der CloudWatch Metrik.

Unterstützt Ersatzvorlagen: Ja

`metricNamespace`

Der Name des CloudWatch Metrik-Namespaces.

Unterstützt Ersatzvorlagen: Ja

`metricUnit`

Die metrische Einheit, die von CloudWatch unterstützt wird.

Unterstützt Ersatzvorlagen: Ja

`metricValue`

Eine Zeichenfolge, die den CloudWatch metrischen Wert enthält.

Unterstützt Ersatzvorlagen: Ja

`metricTimestamp`

(Optional) Eine Zeichenfolge mit dem Zeitstempel, ausgedrückt in Sekunden der Unix-Epochenzeit. Standardmäßig wird die aktuelle Unix-Epochenzeit verwendet.

Unterstützt Ersatzvorlagen: Ja

`roleArn`

Die IAM-Rolle, die den Zugriff auf die CloudWatch Metrik ermöglicht. Weitere Informationen finden Sie unter Voraussetzungen.

Unterstützt Ersatzvorlagen: Nein

Beispiele

Das folgende JSON-Beispiel definiert eine CloudWatch Metrikaktion in einer AWS IoT Regel.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "cloudwatchMetric": {
                    "metricName": "IotMetric",
                    "metricNamespace": "IotNamespace",
                    "metricUnit": "Count",
                    "metricValue": "1",
                    "metricTimestamp": "1456821314",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw"
                }
            }
        ]
    }
}
```

Das folgende JSON-Beispiel definiert eine CloudWatch metrische Aktion mit Ersatzvorlagen in einer AWS IoT Regel.

```
{
    "topicRulePayload": {
```

```
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "cloudwatchMetric": {
                    "metricName": "${topic()}",
                    "metricNamespace": "${namespace}",
                    "metricUnit": "${unit}",
                    "metricValue": "${value}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw"
                }
            }
        ]
    }
}
```

Weitere Informationen finden Sie auch unter

- Was ist Amazon CloudWatch? im CloudWatch Amazon-Benutzerhandbuch
- Verwendung von CloudWatch Amazon-Metriken im CloudWatch Amazon-Benutzerhandbuch

DynamoDB

Die Aktion DynamoDB (`dynamoDB`) schreibt eine MQTT-Nachricht ganz oder teilweise in eine Amazon DynamoDB-Tabelle.

Sie können einem Tutorial folgen, das Ihnen veranschaulicht, wie Sie eine Regel mit einer DynamoDB-Aktion erstellen und testen. Weitere Informationen finden Sie unter Tutorial: Gerätedaten in einer DynamoDB-Tabelle speichern.

> **Note**
>
> Diese Regel schreibt Nicht-JSON-Daten als Binärdaten in DynamoDB. Die DynamoDB-Konsole zeigt die Daten als Base64-codierten Text an.

Voraussetzungen

Diese Regelaktion hat die folgenden Anforderungen:

- Eine IAM-Rolle, die die Ausführung des Vorgangs übernehmen AWS IoT kann. `dynamodb:PutItem` Weitere Informationen finden Sie unter Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt.

  In der AWS IoT Konsole können Sie eine Rolle auswählen oder erstellen, um die Ausführung dieser Regelaktion AWS IoT zu ermöglichen.
- Wenn Sie einen vom Kunden verwalteten AWS KMS key (KMS-Schlüssel) verwenden, um ruhende Daten in DynamoDB zu verschlüsseln, muss der Dienst über die Berechtigung verfügen, den KMS-Schlüssel im Namen des Anrufers zu verwenden. Weitere Informationen finden Sie unter Kundenverwalteter KMS-Schlüssel im Amazon DynamoDB Einführungshandbuch..

Parameters

Wenn Sie mit dieser Aktion eine AWS IoT Regel erstellen, müssen Sie die folgenden Informationen angeben:

`tableName`

Der Name der DynamoDB-Tabelle.

Unterstützt Ersatzvorlagen: API und nur AWS CLI

`hashKeyField`

Der Name des Hash-Schlüssels (auch Partitionsschlüssel genannt)

Unterstützt Ersatzvorlagen: API und nur AWS CLI

`hashKeyType`

(Optional) Der Datentyp des Hash-Schlüssels (auch Partitionsschlüssel genannt). Zulässige Werte: `STRING`, `NUMBER`.

Unterstützt Ersatzvorlagen: API und nur AWS CLI

`hashKeyValue`

Der Wert des Hash-Schlüssels Erwägen Sie die Verwendung einer Ersatzvorlage wie `${topic()}` oder `${timestamp()}`.

Unterstützt Ersatzvorlagen: Ja

`rangeKeyField`

(Optional) Der Name des Bereichsschlüssels (auch Sortierschlüssel genannt).

Unterstützt Ersatzvorlagen: API und nur AWS CLI

`rangeKeyType`

(Optional) Der Datentyp des Bereichsschlüssels (auch Sortierschlüssel genannt). Zulässige Werte: `STRING`, `NUMBER`.

Unterstützt Ersatzvorlagen: API und nur AWS CLI

`rangeKeyValue`

(Optional) Der Wert des Bereichsschlüssels. Erwägen Sie die Verwendung einer Ersatzvorlage wie `${topic()}` oder `${timestamp()}`.

Unterstützt Ersatzvorlagen: Ja

`payloadField`

(Optional) Der Name der Spalte, in die die Nutzlast geschrieben wird. Wenn Sie diesen Wert weglassen, wird die Nutzlast in die Spalte mit dem Namen `payload` geschrieben.

Unterstützt Ersatzvorlagen: Ja

`operation`

(Optional) Der Typ des auszuführenden Vorgangs. Zulässige Werte: `INSERT`, `UPDATE`, `DELETE`.

Unterstützt Ersatzvorlagen: Ja

`roleARN`

Die IAM-Rolle, die den Zugriff auf die DynamoDB-Tabelle erlaubt. Weitere Informationen finden Sie unter Voraussetzungen.

Unterstützt Ersatzvorlagen: Nein

Die in die DynamoDB-Tabelle geschriebenen Daten sind das Ergebnis der SQL-Anweisung der Regel.

Beispiele

Das folgende JSON-Beispiel definiert eine DynamoDB-Aktion in einer AWS IoT Regel.

```
{
    "topicRulePayload": {
        "sql": "SELECT * AS message FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "dynamoDB": {
                    "tableName": "my_ddb_table",
                    "hashKeyField": "key",
                    "hashKeyValue": "${topic()}",
                    "rangeKeyField": "timestamp",
                    "rangeKeyValue": "${timestamp()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamoDB"
                }
            }
        ]
    }
}
```

Weitere Informationen finden Sie auch unter

- Was ist Amazon DynamoDB? im Amazon DynamoDB Entwicklerhandbuch
- Erste Schritte mit DynamoDB im Amazon DynamoDB Entwicklerhandbuch
- Tutorial: Gerätedaten in einer DynamoDB-Tabelle speichern

DynamoDBv2

Die Aktion DynamoDBv2 (`dynamoDBv2`) schreibt eine MQTT-Nachricht ganz oder teilweise in eine Amazon DynamoDB-Tabelle. Jedes Attribut in der Nutzlast wird in eine separate Spalte in der DynamoDB-Datenbank geschrieben.

Voraussetzungen

Diese Regelaktion hat die folgenden Anforderungen:

- Eine IAM-Rolle, die die Ausführung des Vorgangs übernehmen kann. AWS IoT `dynamodb:PutItem` Weitere Informationen finden Sie unter Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt.

In der AWS IoT Konsole können Sie eine Rolle auswählen oder erstellen, um die Ausführung dieser Regelaktion AWS IoT zu ermöglichen.

- Die MQTT-Nachrichtnutzlast muss einen Schlüssel auf Stammebene enthalten, der mit dem primären Partitionsschlüssel der Tabelle übereinstimmt, sowie über einen Schlüssel auf Stammebene, der mit dem primären Sortierschlüssel der Tabelle (sofern definiert) übereinstimmt.
- Wenn Sie einen vom Kunden verwalteten AWS KMS key (KMS-Schlüssel) verwenden, um ruhende Daten in DynamoDB zu verschlüsseln, muss der Dienst über die Berechtigung verfügen, den KMS-Schlüssel im Namen des Anrufers zu verwenden. Weitere Informationen finden Sie unter Kundenverwalteter KMS-Schlüssel im Amazon DynamoDB Einführungshandbuch..

Parameters

Wenn Sie mit dieser Aktion eine AWS IoT Regel erstellen, müssen Sie die folgenden Informationen angeben:

`putItem`

Ein Objekt, das die DynamoDB-Tabelle angibt, in die die Nachrichtendaten geschrieben werden sollen. Dieses Objekt muss die folgenden Informationen enthalten:

`tableName`

Der Name der DynamoDB-Tabelle.

Unterstützt Ersatzvorlagen: API und nur AWS CLI

`roleARN`

Die IAM-Rolle, die den Zugriff auf die DynamoDB-Tabelle erlaubt. Weitere Informationen finden Sie unter Voraussetzungen.

Unterstützt Ersatzvorlagen: Nein

Die in die DynamoDB-Tabelle geschriebenen Daten sind das Ergebnis der SQL-Anweisung der Regel.

Beispiele

Das folgende JSON-Beispiel definiert eine DynamoDBv2-Aktion in einer Regel. AWS IoT

```
{
    "topicRulePayload": {
        "sql": "SELECT * AS message FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "dynamoDBv2": {
                    "putItem": {
                        "tableName": "my_ddb_table"
                    },
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamoDBv2",
                }
            }
        ]
    }
}
```

Das folgende JSON-Beispiel definiert eine DynamoDB-Aktion mit Ersatzvorlagen in einer Regel. AWS IoT

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2015-10-08",
        "actions": [
            {
                "dynamoDBv2": {
                    "putItem": {
                        "tableName": "${topic()}"
                    },
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamoDBv2"
                }
            }
        ]
    }
}
```

Weitere Informationen finden Sie auch unter

- Was ist Amazon DynamoDB? im Amazon DynamoDB Entwicklerhandbuch

- Erste Schritte mit DynamoDB im Amazon DynamoDB Entwicklerhandbuch

Elasticsearch

Die Elasticsearch (`elasticsearch`) -Aktion schreibt Daten aus MQTT-Nachrichten in eine Amazon OpenSearch Service-Domain. Sie können dann Tools wie OpenSearch Dashboards verwenden, um Daten in Service abzufragen und zu visualisieren. OpenSearch

> **Warning**
>
> Die `Elasticsearch` Aktion kann nur von vorhandenen Regelaktion verwendet werden. Um eine neue Regelaktion zu erstellen oder eine vorhandene Regelaktion zu aktualisieren, verwenden Sie stattdessen die `OpenSearch` Regelaktion. Weitere Informationen finden Sie unter OpenSearch.

Voraussetzungen

Diese Regelaktion hat die folgenden Anforderungen:

- Eine IAM-Rolle, die die Ausführung des `es:ESHttpPut` Vorgangs übernehmen AWS IoT kann. Weitere Informationen finden Sie unter Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt.

  In der AWS IoT Konsole können Sie eine Rolle auswählen oder erstellen, um die Ausführung dieser Regelaktion AWS IoT zu ermöglichen.
- Wenn Sie einen vom Kunden verwalteten AWS KMS key (KMS-Schlüssel) verwenden, um gespeicherte Daten zu verschlüsseln OpenSearch, muss der Dienst die Erlaubnis haben, den KMS-Schlüssel im Namen des Anrufers zu verwenden. Weitere Informationen finden Sie unter Verschlüsselung ruhender Daten für Amazon OpenSearch Service im Amazon OpenSearch Service Developer Guide.

Parameters

Wenn Sie eine AWS IoT Regel mit dieser Aktion erstellen, müssen Sie die folgenden Informationen angeben:

`endpoint`

Der Endpunkt Ihrer Service-Domain

Unterstützt Ersatzvorlagen: API und nur AWS CLI

`index`

Der Index, in dem Sie die Daten speichern möchten

Unterstützt Ersatzvorlagen: Ja

`type`

Der Typ des Dokuments, das Sie speichern

Unterstützt Ersatzvorlagen: Ja

`id`

Der eindeutige Bezeichner für jedes Dokument

Unterstützt Ersatzvorlagen: Ja

`roleARN`

Die IAM-Rolle, die den Zugriff auf die OpenSearch Service-Domain ermöglicht. Weitere Informationen finden Sie unter Voraussetzungen.

Unterstützt Ersatzvorlagen: Nein

Beispiele

Das folgende JSON-Beispiel definiert eine Elasticsearch-Aktion in einer AWS IoT Regel und zeigt, wie Sie die Felder für die `elasticsearch` Aktion angeben können. Weitere Informationen finden Sie unter ElasticsearchAction.

```
{
    "topicRulePayload": {
        "sql": "SELECT *, timestamp() as timestamp FROM 'iot/test'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "elasticsearch": {
                    "endpoint": "https://my-endpoint",
```

```
                    "index": "my-index",
                    "type": "my-type",
                    "id": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_es"
                }
            }
        ]
    }
}
```

Das folgende JSON-Beispiel definiert eine Elasticsearch-Aktion mit Ersatzvorlagen in einer Regel. AWS IoT

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "elasticsearch": {
                    "endpoint": "https://my-endpoint",
                    "index": "${topic()}",
                    "type": "${type}",
                    "id": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_es"
                }
            }
        ]
    }
}
```

Weitere Informationen finden Sie auch unter

- OpenSearch
- Was ist Amazon OpenSearch Service?

HTTP

Die Aktion HTTPS (`http`) sendet Daten von einer MQTT-Nachricht an einen HTTPS-Endpunkt, der auf eine Webanwendung oder einen Webdienst verweisen kann.

Voraussetzungen

Diese Regelaktion hat die folgenden Anforderungen:

- Sie müssen die HTTPS-Endpunkte bestätigen und aktivieren, bevor sie von der Regel-Engine verwendet werden können. Weitere Informationen finden Sie unter HTTP-Aktionsziele.

Parameters

Wenn Sie eine AWS IoT Regel mit dieser Aktion erstellen, müssen Sie die folgenden Informationen angeben:

`url`

Der HTTPS-Endpunkt, an den die Nachricht mithilfe der HTTP-POST-Methode gesendet wird. Wenn Sie eine IP-Adresse anstelle eines Hostnamens verwenden, muss es sich um eine IPv4-Adresse handeln. IPv6-Adressen werden anstelle eines Hostnamens nicht unterstützt. Um Daten über IPv6 zu empfangen, verwenden Sie einen Hostnamen, der AAAA-Einträge und keine A-Datensätze veröffentlicht.

Hinweise zur IP-Version, mit der die Rules Engine eine Verbindung zu Ihrem Endpunkt herstellt, finden Sie unter. Netzwerkzugriff

Unterstützt Ersatzvorlagen: Ja

`confirmationUrl`

(Optional) Falls angegeben, AWS IoT verwendet die Bestätigungs-URL, um ein passendes Ziel für die Themenregel zu erstellen. Sie müssen das HTTP-Aktionsziel aktivieren, bevor Sie es in einer HTTP-Aktion verwenden können. Weitere Informationen finden Sie unter HTTP-Aktionsziele. Wenn Sie Ersatzvorlagen verwenden, müssen Sie manuell ein HTTP-Aktionsziel erstellen, bevor die `http` Aktion verwendet werden kann. `confirmationUrl`muss ein Präfix von `url` sein.

Das Verhältnis zwischen `url` und `confirmationUrl` wird wie folgt beschrieben:

- Wenn `url` es fest codiert und nicht bereitgestellt `confirmationUrl` wird, behandeln wir das `url` Feld implizit als. `confirmationUrl` AWS IoT erstellt ein Ziel für eine Themenregel. `url`
- Falls `url` und fest codiert `confirmationUrl` sind, `url` muss mit `confirmationUrl` beginnen. AWS IoT erstellt ein Ziel für `confirmationUrl` eine Themenregel.
- Wenn `url` eine Ersetzungsvorlage enthält, müssen Sie `confirmationUrl` angeben und `url` muss mit `confirmationUrl` beginnen. Wenn es Ersatzvorlagen `confirmationUrl` enthält,

müssen Sie manuell ein HTTP-Aktionsziel erstellen, bevor die `http` Aktion verwendet werden kann. Wenn `confirmationUrl` es keine Ersatzvorlagen enthält, AWS IoT erstellt es ein Ziel für eine Themenregel. `confirmationUrl`

Unterstützt Ersatzvorlagen: Ja

`headers`

(Optional) Die Liste der Header, die in HTTP-Anfragen an den Endpunkt aufgenommen werden sollen. Jeder Header muss die folgenden Informationen enthalten:

`key`

Der Schlüssel des Headers.

Unterstützt Ersatzvorlagen: Nein

`value`

Der Wert des Headers

Unterstützt Ersatzvorlagen: Ja

> **Note**
>
> Der Standard-Inhaltstyp ist application/json , wenn die Payload im JSON-Format vorliegt. Andernfalls ist es application/octet -stream. Sie können ihn überschreiben, indem Sie im Header den genauen Inhaltstyp mit dem Schlüsselinhaltstyp angeben (ohne Berücksichtigung von Groß- und Kleinschreibung).

`auth`

(Optional) Die vom Regelmodul verwendete Authentifizierung, um eine Verbindung mit der im `url` Argument angegebenen Endpunkt-URL herzustellen. Derzeit ist Signature Version 4 der einzige unterstützte Authentifizierungstyp. Weitere Informationen finden Sie unter HTTP-Autorisierung.

Unterstützt Ersatzvorlagen: Nein

`enableBatching`

(Optional) Ob die HTTP-Aktionsnachrichten in einer einzigen Anfrage für eine bestimmte URL verarbeitet werden sollen. Der Wert kann wahr oder falsch sein. Weitere Informationen zur Batchverarbeitung finden Sie unter Batching von HTTP-Aktionsnachrichten.

Boolescher Wert

Unterstützt Ersatzvorlagen: Nein

`batchConfig`

(Optional) Konfigurationseinstellungen für die Batchverarbeitung. Nach der Aktivierung müssen `batchConfig` die Parameter angegeben werden. Wenn keine `batchConfig` Parameter angegeben werden, werden die Standardwerte verwendet.

`maxBatchOpenMs`

Die maximale Zeitspanne (in Millisekunden), für die eine ausgehende Nachricht darauf wartet, dass andere Nachrichten den Batch erstellen. Je höher die Einstellung, desto länger ist die Latenz der HTTP-Batch-Aktion.

Mindestwert: 5 ms. Maximalwert: 200 ms.

Standardwert: 20 ms

Unterstützt Ersatzvorlagen: Nein

`maxBatchSize`

Die maximale Anzahl von Nachrichten, die in einer einzigen Aktionsausführung gebündelt werden.

Unterstützt Ersatzvorlagen: Nein

Mindestwert: 2 Nachrichten. Höchstwert: 10 Nachrichten

Standardwert: 10 Nachrichten

`maxBatchSizeBytes`

Maximale Größe eines Nachrichtenstapels in Byte.

Mindestwert: 100 Byte. Maximalwert: 131.072 Byte

Standardwert: 5.120 Byte

Unterstützt Ersatzvorlagen: Nein

`batchAcrossTopics`

Gibt an, ob Nachrichten aus verschiedenen MQTT-Themen in einer einzigen HTTP-Anfrage gebündelt werden dürfen. Standardmäßig werden nur Nachrichten aus demselben

Thema gebündelt. Aktivieren Sie diesen Parameter für Routing-Anwendungsfälle, in denen Nachrichten von mehreren Gerätethemen für denselben HTTP-Endpunkt bestimmt sind.

Standardwert: falsch

Unterstützt Ersatzvorlagen: Nein

> **Note**
>
> Nachrichten werden immer im Rahmen desselben Kontos, desselben Regelnamens, derselben HTTP-Zielendpunkt-URL und derselben Abrechnungsgruppe gebündelt. Nachrichten, die sich in einem dieser Attribute unterscheiden, werden unabhängig von der `batchAcrossTopics` Einstellung niemals zu demselben Stapel zusammengefasst.

> **Note**
>
> Der Standard-Inhaltstyp ist application/json , wenn die Nutzdaten im JSON-Format vorliegen. Andernfalls ist es application/octet -stream. Sie können ihn überschreiben, indem Sie im Header den genauen Inhaltstyp mit dem Schlüsselinhaltstyp angeben (ohne Berücksichtigung von Groß- und Kleinschreibung).

Beispiele

Das folgende JSON-Beispiel definiert eine AWS IoT Regel mit einer HTTP-Aktion.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "http": {
                    "url": "https://www.example.com/subpath",
                    "confirmationUrl": "https://www.example.com",
                    "headers": [
                        {
                            "key": "static_header_key",
```

```
                            "value": "static_header_value"
                        },
                        {
                            "key": "substitutable_header_key",
                            "value": "${value_from_payload}"
                        }
                    ]
                }
            }
        ]
    }
}
```

```
"http": {
    "url": "https://www.example.com/subpath",
    "confirmationUrl": "https://www.example.com",
    "headers": [
        {
            "key": "Content-Type",
            "value": "application/json"
          }
    ],
    "enableBatching": true,
    "batchConfig": {
      "maxBatchOpenMs": 123,
      "maxBatchSize": 5,
      "maxBatchSizeBytes": 131072,
     }
 },
 "errorAction": {
        "http": {
            "url": "https://www.example.com/subpath",
            "confirmationUrl": "https://www.example.com"
            // batchConfig is not allowed here
        }
}
```

Logik zur Wiederholung von HTTP-Aktionen

Die AWS IoT Regel-Engine wiederholt die HTTP-Aktion gemäß den folgenden Regeln:

- Die Regel-Engine versucht mindestens einmal, eine Nachricht zu senden.

- Die Regel-Engine wiederholt den Versuch höchstens zweimal. Die maximale Anzahl von Versuchen ist drei.
- Die Regel-Engine unternimmt in folgenden Fällen keinen Wiederholungsversuch:
  - Der vorherige Versuch lieferte eine Antwort mit einer Größe von über 16.384 Bytes.
  - Der nachgeschaltete Webdienst oder die nachgeschaltete Anwendung schließt die TCP-Verbindung nach dem Versuch.
  - Die Gesamtzeit zum Abschließen einer Anforderung mit Wiederholungsversuchen hat das Zeitlimit für die Anforderung überschritten.
  - Die Anforderung gibt einen anderen HTTP-Statuscode als 429, 500–599 zurück.

> **Note**
>
> Für Wiederholungen fallen die Standardkosten für die Datenübertragung an.

Weitere Informationen finden Sie auch unter

- Batch-Verarbeitung von HTTP-Aktionsnachrichten
- HTTP-Aktionsziele
- Leiten Sie Daten direkt von AWS IoT Core zu Ihren Webdiensten im Internet der Dinge im Blog weiter AWS

Batch-Verarbeitung von HTTP-Aktionsnachrichten

Sie können Batching verwenden, um mehrere HTTP-Aktionsnachrichten in einer einzigen Anfrage zu senden.

-Übersicht

Durch Batching können Sie Nachrichten von der AWS IoT Core Rules Engine stapelweise an Ihre HTTP-Endpunkte senden. Mit dieser Funktion können Sie Ihre Kosten senken, indem Sie die Anzahl der ausgeführten HTTP-Aktionen verringern und die Effizienz verbessern, indem der mit der Einrichtung neuer Verbindungen verbundene Aufwand reduziert wird.

> **Note**
>
> Die HTTP-Aktion im Batchmodus wird als einzelne Aktion berechnet. Die Zählung erfolgt in Schritten von 5 KiB, basierend auf der Größe der ausgehenden Batch-Payload, die von der Rules Engine an den AWS IoT Core Downstream-Service gesendet wird. Weitere Informationen finden Sie in der AWS IoT Core Preisliste.

Wenn Sie Batching in der Definition Ihrer IoT-Regelaktion aktivieren, stehen die folgenden Parameter für die Konfiguration zur Verfügung:

`maxBatchOpenMs`

Die maximale Zeitspanne (in Millisekunden), für die eine ausgehende Nachricht darauf wartet, dass andere Nachrichten den Stapel erstellen. Je höher die Einstellung, desto länger ist die Latenz der HTTP-Batch-Aktion.

Mindestwert: 5 ms. Maximalwert: 200 ms.

Standardwert: 20 ms

Unterstützt Ersatzvorlagen: Nein

`maxBatchSize`

Die maximale Anzahl von Nachrichten, die in einer einzigen Ausführung einer IoT-Regelaktion gebündelt werden.

Mindestwert: 2 Nachrichten. Höchstwert: 10 Nachrichten

Standardwert: 10 Nachrichten

Unterstützt Ersatzvorlagen: Nein

`maxBatchSizeBytes`

Maximale Größe eines Nachrichtenstapels in Byte.

Mindestwert: 100 Byte. Maximalwert: 131.072 Byte

Standardwert: 5120 Byte

Unterstützt Ersatzvorlagen: Nein

`batchAcrossTopics`

Gibt an, ob Nachrichten aus verschiedenen MQTT-Themen in einer einzigen HTTP-Anfrage gebündelt werden dürfen. Standardmäßig werden nur Nachrichten aus demselben Thema gebündelt. Aktivieren Sie diesen Parameter für Routing-Anwendungsfälle, in denen Nachrichten von mehreren Gerätethemen für denselben HTTP-Endpunkt bestimmt sind.

Standardwert: falsch

Unterstützt Ersatzvorlagen: Nein

Important

Wenn Sie mehrere Batch-Parameter angeben, wird die Batchverarbeitung abgeschlossen, wenn das erste Limit erreicht ist. Wenn Sie beispielsweise 100 ms als maximale Batch-Öffnungszeit und 5 KiB als maximale Batch-Größe angeben und Rules Engine innerhalb von 100 ms nur 2 KiB stapelt, wird ein 2-KB-Batch erstellt und gesendet.

Note

Nachrichten werden immer im Rahmen desselben Kontos, desselben Regelnamens, derselben HTTP-Zielendpunkt-URL und derselben Abrechnungsgruppe gebündelt. Nachrichten, die sich in einem dieser Attribute unterscheiden, werden unabhängig von der `batchAcrossTopics` Einstellung niemals zu demselben Stapel zusammengefasst.

Verwendung von HTTP-Headern in einem Batch

Wenn Sie in Ihrer HTTP-Aktion Header verwenden, verwendet die Batch-Anfrage den Header-Wert der letzten Nachricht, die dem Batch hinzugefügt wurde (nicht unbedingt die letzte Nachricht, die Sie veröffentlicht haben). Wir empfehlen die Verwendung von Header-Werten, die entweder:

- Identisch für alle Nachrichten im Batch
- Gilt für alle Nachrichten (z. B. Authentifizierungsdaten)

Die Header werden mit der HTTP-Anfrage gesendet und sind nicht Teil des Nachrichtentexts.

> **Note**
>
> Wenn Batching aktiviert ist:
>
> - Die Batch-Anfrage enthält automatisch den `Content-Type: application/json` Header, da der Batch als JSON-Array gesendet wird.
> - Wir können nicht garantieren, dass die letzte Nachricht im Batch die letzte Nachricht ist, die Sie veröffentlicht haben. Es ist die letzte Nachricht, die es in den Stapel geschafft hat.

Beispiel für eine Nutzlast

Das folgende Beispiel zeigt die Struktur einer gebündelten Nachrichtennutzlast, die an Ihren HTTP-Endpunkt gesendet wird:

```
[
  {
    "user_id": "user1",
    "steps_today": 1000
  },
  {
    "user_id": "user2",
    "steps_today": 21000
  },
  {
    "user_id": "user8",
    "steps_today": 1500
  },
  ...
]
```

Einschränkungen

Im Folgenden sind Einschränkungen für die Batchverarbeitung aufgeführt:

- AWS IoT Core garantiert nicht die allgemeine Reihenfolge der Nachrichten. Die Batchverarbeitung erfolgt lokal auf jedem Host, was dazu führen kann, dass Nachrichten innerhalb eines Batches in einer anderen Reihenfolge verarbeitet werden, als sie empfangen wurden.

- AWS IoT Core bietet keine Unterstützung für die Nachrichtenverarbeitung auf der Empfängerseite. Sie sind dafür verantwortlich, dass Ihr Downstream-Service so konfiguriert ist, dass er Daten stapelweise akzeptiert und verarbeitet.
- Cross-account Batching wird nicht unterstützt, auch wenn Nachrichten für dieselbe Ressourcen-ID (HTTP-URL oder Ressourcen-ARN) bestimmt sind.
- AWS IoT Core garantiert nicht, dass die Batchgröße der von Ihnen angegebenen Konfiguration entspricht. Je nach Zeitpunkt und Nachrichtenfluss können Batches unter Ihren konfigurierten Grenzwerten liegen.
- Wenn Batching aktiviert ist, werden binäre Payloads (keine UTF-8 Daten) nicht unterstützt. Nur UTF-8 Textnutzlasten (wie JSON) werden akzeptiert. Um Binärdaten zu senden, kodieren Sie sie mit Base64, bevor Sie sie an die HTTP-Aktion senden, und dekodieren Sie sie dann an Ihrem Empfangsendpunkt. Sie können beispielsweise die Encode-Funktion in IoT-Regeln verwenden, um die binäre Nutzlast zu codieren. Alternativ können Sie die binäre Nutzlast in Ihrem IoT-Gerät codieren und auf veröffentlichen. AWS IoT Core

Fehleraktionen beim Batching

Sie können in Ihrer Fehleraktionsdefinition keine separate Batching-Logik definieren. Ihre Fehleraktion unterstützt jedoch die Batchverarbeitung, wenn Sie in Ihrer primären Aktion eine Batching-Logik definiert haben.

Wenn eine Batchanforderung fehlschlägt, folgt AWS IoT Core die Rules Engine der Wiederholungslogik für HTTP-Aktionen. Nach dem letzten Wiederholungsversuch wird eine Fehleraktion für den gesamten fehlgeschlagenen Batch aufgerufen.

Im Folgenden finden Sie ein Beispiel für eine Fehlermeldung mit aktivierter Batchverarbeitung:

```
{
    "ruleName": "FailedTopicRule",
    "topic": "topic/rulesengine",
    "payloadsWithMetadata": [
        {
            "id": 1,
            "cloudwatchTraceId": "bebd6d93-6d4a-899e-9e40-56e82252d2be",
            "clientId": "Test",
            "sourceIp": "10.0.0.0",
            "base64OriginalPayload":
 "eyJ1c2VyX2lkIjogInVzZXI1NjQ3IiwgInN0ZXBzX3RvZGF5IjogMTMzNjUsICJ0aW1lc3RhbXAiOiAiMjAyNS0xMC0wO
        },
```

```
        {
            "id": 2,
            "cloudwatchTraceId": "af94d3b8-0b18-1dbf-2c7d-513f5cb9e2e1",
            "clientId": "Test",
            "sourceIp": "10.0.0.0",
            "base64OriginalPayload":
"eyJ1c2VyX2lkIjogInVzZXI1NjQ3IiwgInN0ZXBzX3RvZGF5IjogMTMzNjUsICJ0aW1lc3RhbXAiOiAiMjAyNS0xMC0wO
        },
        {
            "id": 3,
            "cloudwatchTraceId": "ca441266-c2ce-c916-6aee-b9e5c7831675",
            "clientId": "Test",
            "sourceIp": "10.0.0.0",
            "base64OriginalPayload":
"eyJ1c2VyX2lkIjogInVzZXI1NjQ3IiwgInN0ZXBzX3RvZGF5IjogMTMzNjUsICJ0aW1lc3RhbXAiOiAiMjAyNS0xMC0wO
        }
    ],
    "failures": [
        {
            "affectedIds": [
                1,
                2,
                3
            ],
            "failedAction": "HttpAction",
            "failedResource": "https://example.foobar.com/HttpAction",
            "errorMessage": "HttpAction failed to make a request to the specified
endpoint. StatusCode: 500. Reason: Internal Server Error."
        },
        {
            "affectedIds": [
                3
            ],
            "failedAction": "S3Action",
            "failedResource": "amzn-s3-demo-bucket",
            "errorMessage": "Failed to put S3 object. The error received was The
specified bucket does not exist"
        },
        {
            "affectedIds": [
                3
            ],
            "failedAction": "LambdaAction",
            "failedResource": "arn:aws:lambda:us-west-2:123456789012:function:dummy",
```

```
            "errorMessage": "Failed to invoke lambda function. Received Server error
 from Lambda. The error code is 403"
        }
    ]
}
```

Wenn diese Option aktiviert `batchAcrossTopics` ist, ändert sich das Format der Payload für die Fehleraktion. Das `topic` Feld wird von der obersten Ebene in jeden `payloadsWithMetadata` Eintrag verschoben:

```
{
    "ruleName": "FailedTopicRule",
    "payloadsWithMetadata": [
        {
            "id": 1,
            "topic": "topic/sensor1",
            "cloudwatchTraceId": "bebd6d93-6d4a-899e-9e40-56e82252d2be",
            "clientId": "Test",
            "sourceIp": "10.0.0.0",
            "base64OriginalPayload":
 "eyJ1c2VyX2lkIjogInVzZXI1NjQ3IiwgInN0ZXBzX3RvZGF5IjogMTMzNjUsICJ0aW1lc3RhbXAiOiAiMjAyNS0xMC0wC
        },
        {
            "id": 2,
            "topic": "topic/sensor2",
            "cloudwatchTraceId": "af94d3b8-0b18-1dbf-2c7d-513f5cb9e2e1",
            "clientId": "Test",
            "sourceIp": "10.0.0.0",
            "base64OriginalPayload":
 "eyJ1c2VyX2lkIjogInVzZXI1NjQ3IiwgInN0ZXBzX3RvZGF5IjogMTMzNjUsICJ0aW1lc3RhbXAiOiAiMjAyNS0xMC0wC
        }
    ],
    "failures": [
        {
            "affectedIds": [1, 2],
            "failedAction": "HttpAction",
            "failedResource": "https://example.foobar.com/HttpAction",
            "errorMessage": "HttpAction failed to make a request to the specified
 endpoint. StatusCode: 500. Reason: Internal Server Error."
        }
    ]
}
```

> **Note**
>
> Fehler bei gebündelten Aktionen führen auch zu größeren Nutzlasten für Fehleraktionen, wodurch die Wahrscheinlichkeit, dass Fehleraktionen aufgrund ihrer Größe fehlschlagen, erhöht werden kann. Mithilfe der Metrik können Sie fehlgeschlagene Fehleraktionen überwachen. `ErrorActionFailure` Weitere Informationen finden Sie unter Regelaktionsmetriken.

Batching von HTTP-Aktionsnachrichten mit dem AWS CLI

Erstellen oder Aktualisieren einer Regelaktion mit Batching

1. Verwenden Sie den entsprechenden AWS CLI Befehl, um eine Regel zu erstellen oder zu aktualisieren:
   - Verwenden Sie den Befehl create-topic-rule, um eine neue Regel zu erstellen:

     ```
     aws iot create-topic-rule --rule-name myrule --topic-rule-payload
      file://myrule.json
     ```

   - Verwenden Sie den Befehl replace-topic-rule, um eine bestehende Regel zu aktualisieren:

     ```
     aws iot replace-topic-rule --rule-name myrule --topic-rule-payload
      file://myrule.json
     ```

2. Aktivieren Sie die Batching-Funktionen, indem Sie den Parameter enableBatching in der Payload Ihrer Themenregel auf true setzen:

```
{
        "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "http": {
                    "url": "https://www.example.com/subpath",
                    "confirmationUrl": "https://www.example.com",
                    "headers": [
                        {
                            "key": "static_header_key",
```

```
                        "value": "static_header_value"
                    },
                    {
                        "key": "substitutable_header_key",
                        "value": "${value_from_payload}"
                     }
                ],
                "enableBatching": true,
                "batchConfig": {
                   "maxBatchOpenMs": 100,
                   "maxBatchSize": 5,
                   "maxBatchSizeBytes": 1024,
                   "batchAcrossTopics": true
                }
            }
        }
    ]
}
```

3. Konfigurieren Sie die Batching-Parameter. Sie müssen nicht alle Batch-Parameter angeben. Sie können wählen, ob Sie 1, 2, 3 oder alle 4 Batch-Parameter angeben möchten. Wenn Sie keinen Batch-Parameter angeben, aktualisiert Rules Engine diesen Parameter mit den Standardwerten. Weitere Informationen zu Batching-Parametern und ihren Standardwerten finden Sie unter HTTP-Parameter.

HTTP-Aktionsziele

Ein HTTP-Aktionsziel ist ein Webdienst, an den die Regel-Engine Daten aus einer Themenregel weiterleiten kann. Eine AWS IoT Core Ressource beschreibt den Webdienst für AWS IoT. Zielressourcen können nach unterschiedlichen Regeln gemeinsam genutzt werden.

Bevor Daten an einen anderen Webdienst gesendet werden AWS IoT Core können, muss dieser bestätigen, dass er auf den Endpunkt des Dienstes zugreifen kann.

-Übersicht

Der Begriff „HTTP-Aktionsziel“ bezieht sich auf einen Webservice, der eine Bestätigungs-URL und eine oder mehrere Datensammlungs-URLs unterstützt. Die Zielressource enthält die Bestätigungs-URL Ihres Webdienstes. Wenn Sie eine HTTP-Aktion konfigurieren, geben Sie die tatsächliche URL des Endpunkts, der die Daten erhalten soll, zusammen mit der Bestätigungs-URL des Webdienstes

an. Nachdem Ihr Ziel bestätigt wurde, sendet die Themenregel das Ergebnis der SQL-Anweisung an den HTTPS-Endpunkt (und nicht an die Bestätigungs-URL).

Ein HTTP-Aktionsziel kann sich in einem der folgenden Zustände befinden:

ENABLED (AKTIVIERT)

Das Ziel wurde bestätigt und kann von einer Regelaktion verwendet werden. Ein Ziel muss den Zustand `ENABLED` aufweisen, damit es in einer Regel verwendet werden kann. Sie können nur Ziele im Status „DISABLED“ aktivieren.

DISABLED (DEAKTIVIERT)

Das Ziel wurde bestätigt, kann aber nicht von einer Regelaktion verwendet werden. Dies ist nützlich, wenn Sie Datenverkehr zu Ihrem Endpunkt vorübergehend aussetzen möchten, ohne den Bestätigungsvorgang erneut durchlaufen zu müssen. Sie können nur Ziele mit dem Status „ENABLED“ deaktivieren.

IN_PROGRESS

Die Bestätigung des Ziels wird ausgeführt.

ERROR

Zeitüberschreitung bei der Zielbestätigung.

Nachdem ein HTTP-Aktionsziel bestätigt und aktiviert wurde, kann es mit jeder Regel in Ihrem Konto verwendet werden.

Verwaltung von HTTP-Aktionszielen

Sie können die folgenden Operationen verwenden, um Ihre HTTP-Aktionsziele zu verwalten.

HTTP-Aktionsziele erstellen

Sie erstellen ein HTTP-Aktionsziel, indem Sie den `CreateTopicRuleDestination` Vorgang aufrufen oder die AWS IoT Konsole verwenden.

Nachdem Sie ein Ziel erstellt haben, AWS IoT sendet eine Bestätigungsanfrage an die Bestätigungs-URL. Die Bestätigungsanforderung hat das folgende Format:

```
HTTP POST {confirmationUrl}/?confirmationToken={confirmationToken}
```

```
Headers:
x-amz-rules-engine-message-type: DestinationConfirmation
x-amz-rules-engine-destination-arn:"arn:aws:iot:us-east-1:123456789012:ruledestination/
http/7a280e37-b9c6-47a2-a751-0703693f46e4"
Content-Type: application/json
Body:
{
    "arn":"arn:aws:iot:us-east-1:123456789012:ruledestination/http/7a280e37-b9c6-47a2-
a751-0703693f46e4",
    "confirmationToken": "AYADeMXLrPrNY2wqJAKsFNn-…NBJndA",
    "enableUrl": "https://iot.us-east-1.amazonaws.com/confirmdestination/
AYADeMXLrPrNY2wqJAKsFNn-…NBJndA",
    "messageType": "DestinationConfirmation"
}
```

Der Inhalt der Bestätigungsanforderung umfasst die folgenden Informationen:

arn

Der Amazon-Ressourcenname (ARN) für das zu bestätigende HTTP-Aktionsziel.

confirmationToken

Das Bestätigungstoken, gesendet von AWS IoT Core. Das Token im Beispiel ist gekürzt. Ihr tatsächliches Token ist länger. Sie benötigen dieses Token, um Ihr Ziel mit AWS IoT Core zu bestätigen.

enableUrl

Die URL, zu der Sie navigieren, um ein Themenregelziel zu bestätigen.

messageType

Der Nachrichtentyp.

Bestätigung der Ziele für HTTP-Aktionen

Wenn Sie den Endpunkt-Bestätigungsprozess verwenden, müssen Sie die folgenden Schritte ausführen AWS CLI, nachdem Ihre Bestätigungs-URL die Bestätigungsanfrage erhalten hat.

1. Vergewissern Sie sich, dass das Ziel bereit ist, Nachrichten zu empfangen

   Um zu bestätigen, dass das HTTP-Aktionsziel bereit ist, IoT-Nachrichten zu empfangen, rufen Sie entweder die `enableUrl` in der Bestätigungsanfrage auf oder führen Sie den

`ConfirmTopicRuleDestination` API-Vorgang `confirmationToken` aus und übergeben Sie die Bestätigungsanfrage.

2. Setzen Sie den Status der Themenregel auf aktiviert

   Nachdem Sie bestätigt haben, dass das Ziel Nachrichten empfangen kann, müssen Sie den `UpdateTopicRuleDestination` API-Vorgang ausführen, um den Status der Themenregel auf zu setzen`ENABLED`.

Wenn Sie die AWS IoT Konsole verwenden, kopieren Sie die `confirmationToken` und fügen Sie sie in das Bestätigungsdialogfeld des Ziels in der AWS IoT Konsole ein. Anschließend können Sie die Themenregel aktivieren.

Senden einer neuen Bestätigungsanforderung

Zum Aktivieren einer neuen Bestätigungsnachricht für ein Ziel rufen Sie `UpdateTopicRuleDestination` auf und legen den Zustand des Themenregelziels auf `IN_PROGRESS` fest.

Wiederholen Sie den Bestätigungsvorgang, nachdem Sie eine neue Bestätigungsanforderung gesendet haben.

Deaktivieren und Löschen eines HTTP-Aktionsziels

Zum Deaktivieren eines Ziels rufen Sie `UpdateTopicRuleDestination` auf und legen den Zustand des Themenregelziels auf `DISABLED` fest. Eine Themenregel mit dem Status DISABLED (DEAKTIVIERT) kann wieder aktiviert werden, ohne dass eine neue Bestätigungsanforderung gesendet werden muss.

Um ein HTTP-Aktionsziel zu löschen, rufen Sie an. `DeleteTopicRuleDestination`

Support durch Zertifizierungsstellen

> **Note**
>
> Self-signed Zertifikate werden nicht unterstützt.

HTTPS-Endpunkte in einem HTTP-Aktionsziel unterstützen Zertifikate, die sowohl von AWS Private Certificate Authority als auch von Lets Encrypt ausgestellt wurden.

Netzwerkzugriff

Die AWS IoT Regel-Engine stellt standardmäßig über IPv4 eine Verbindung zu Ihrem HTTPS-Endpunkt her. Wenn der Hostname Ihres Endpunkts nur in IPv6-Adressen aufgelöst wird, stellt die Regel-Engine stattdessen über IPv6 eine Verbindung zu ihm her.

Das passiert, wenn Ihr Hostname AAAA-Einträge und keine A-Einträge veröffentlicht. Es gibt keine Einstellung AWS IoT Core , die dies steuert. Der DNS Ihres Endpunkts bestimmt, welche Version die Regel-Engine verwendet.

> **Note**
>
> Für die Zielbestätigung wird immer IPv4 verwendet. Ihr Endpunkt muss über IPv4 erreichbar sein, wenn Sie ihn bestätigen, auch wenn Sie beabsichtigen, Daten über IPv6 zu empfangen.

Um Daten über IPv6 zu empfangen

1. Veröffentlichen Sie sowohl einen A-Eintrag als auch einen AAAA-Eintrag für den Hostnamen Ihres Endpunkts.
2. Erstellen Sie das Ziel und schließen Sie den Bestätigungsvorgang ab. Weitere Informationen erhalten Sie unter HTTP-Aktionsziele erstellen und Bestätigung der Ziele für HTTP-Aktionen.
3. Entfernen Sie den A-Eintrag, sodass nur der AAAA-Datensatz übrig bleibt. Ab diesem Zeitpunkt stellt die Regel-Engine eine Verbindung über IPv6 her.

Wenn Sie später den Status des Ziels auf `IN_PROGRESS` „Senden einer neuen Bestätigungsanfrage“ setzen, stellen Sie zuerst den A-Datensatz wieder her. Eine Bestätigungsanfrage kann derzeit keinen Endpunkt erreichen, der nur AAAA-Datensätze veröffentlicht.

Endpunkte, die den eingehenden Datenverkehr anhand der Quelladresse einschränken, müssen die Adressen der Rules Engine zulassen. Die Regel-Engine verwendet in jeder Region einen festen Bereich von IPv6-Adressen. Weitere Informationen finden Sie unter Beschränkung des HTTP-Aktionsverkehrs auf die IPv6-Bereiche der Rules Engine.

InfluxDB

Wählen Sie diese Aktion, wenn Sie die Gerätetelemetrie nach Betriebstrends abfragen oder Zeitreihendaten in Echtzeit überwachen möchten. Sie können die clientseitige oder serverseitige

Batchverarbeitung verwenden, um mehrere Punkte in einer Schreibanforderung zu kombinieren. AWS IoT Core konvertiert jede Nachricht in das InfluxDB-Zeilenprotokoll und schreibt sie in die angegebene Datenbank und Tabelle. Weitere Informationen zum Format finden Sie unter InfluxDB-Zeilenprotokoll in der Dokumentation. InfluxData Informationen zu verwalteten Clustern finden Sie unter Amazon Timestream for InfluxDB.

In diesem Thema:

- Voraussetzungen
- InfluxDB-Aktionsziel
- Zuordnung der InfluxDB-Terminologie
- Parameters
- Stapelverarbeitung
- Per-element Vorlagen für Array-Payloads
- Inhalt des InfluxDB-Datensatzes
- Aktionen bei Fehlern
- Beispiele

Voraussetzungen

Für diese Regelaktion gelten die folgenden Voraussetzungen:

- Ein InfluxDB-Aktionsziel — Erstellen Sie ein InfluxDB-Aktionsziel, das den Endpunkt, die Version und die Anmeldeinformationen für Ihre InfluxDB-Instance angibt. AWS IoT Core validiert den Besitz des Endpunkts, bevor Traffic gesendet wird. Siehe InfluxDB-Aktionsziel.
- Eine IAM-Rolle, die davon ausgehen AWS IoT kann, in Ihre InfluxDB-Datenbanken zu schreiben und den `GetSecretValue` Vorgang mit dem Secret durchzuführen, in dem Ihre InfluxDB-Anmeldeinformationen gespeichert sind. Weitere Informationen finden Sie unter Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt.
- InfluxDB-Anmeldeinformationen gespeichert in AWS Secrets Manager — Für InfluxDB V3 stellt Amazon Timestream for InfluxDB automatisch ein Secrets Manager-Geheimnis bereit, wenn Sie den InfluxDB-Cluster erstellen. Das Geheimnis enthält die Cluster-Anmeldeinformationen. Generieren Sie für InfluxDB V2 ein All-Access- oder ein benutzerdefiniertes API-Token aus Ihrer InfluxDB-Instance. Speichern Sie den Token-Wert als Klartext-Geheimnis in. AWS Secrets Manager Weitere Informationen zur Token-Erstellung finden Sie in der InfluxData Dokumentation unter Token erstellen. Zur Laufzeit ruft die Regelaktion auf, `secretsmanager:GetSecretValue`

um diese Anmeldeinformationen abzurufen, bevor Sie sich bei Ihrem InfluxDB-Endpunkt authentifizieren.

- Zeitstempel in der Nachrichtennutzlast — Jedes Objekt in der Nachrichtennutzlast muss einen Schlüssel mit dem Namen `timestamp` (Groß- und Kleinschreibung beachten) mit einem ganzzahligen Unix-Epochenwert enthalten. Die Zeitstempeleinheit kann auch in der IoT-Regeldefinition festgelegt werden. AWS IoT Core generiert keine Zeitstempel für die InfluxDB-Aktion, außer wenn InfluxDB als Fehleraktion verwendet wird.

  > **Note**
  >
  > Ein Alias wie `ts` oder `time` kann nicht anstelle von verwendet werden. `timestamp`

- HTTPS-Konnektivität — Ihre InfluxDB-Instance muss AWS IoT Core über HTTPS erreichbar sein. Unterstützte ausgehende Ports sind: 443, 8443, 8086 (Standardport für InfluxDB V2) und 8181 (Standardport für InfluxDB V3).
- JSON-format Payload — Die InfluxDB-Aktion verarbeitet nur JSON-Payloads. Wenn Ihre Geräte Binär- oder Protobuf-Daten veröffentlichen, verwenden Sie die `decode()`Funktion in Ihrer Regel-SQL, um sie in JSON zu konvertieren, bevor die Aktion ausgeführt wird.

InfluxDB-Aktionsziel

Bevor Sie die InfluxDB-Regelaktion verwenden können, müssen Sie ein InfluxDB-Aktionsziel erstellen. Das Ziel definiert die Verbindungsparameter für Ihre InfluxDB-Instanz. AWS IoT Core validiert dann den Besitz der Endgeräte.

Ein Ziel wird erstellt

Verwenden Sie die `CreateTopicRuleDestination` API, um ein InfluxDB-Ziel zu erstellen:

```
{
  "destinationConfiguration": {
    "influxDBConfiguration": {
      "endpoint": "https://my-instance.timestream-influxdb.us-
west-2.amazonaws.com:8086",
      "influxDBVersion": "V2",
      "secretId": "arn:aws:secretsmanager:us-west-2:111122223333:secret:my-influxdb-
credentials-AbCdEf"
    }
  }
```

```
}
```

Zielparameter

Parameter	Typ	Erforderlich	Beschreibung
`endpoint`	Zeichenfolge	Ja	Die HTTPS-Endpunkt-URL Ihrer InfluxDB-Instance. HTTP wird nicht unterstützt. Unterstützte Ports: 443, 8086, 8181, 8443.
`influxDBVersion`	Zeichenfolge	Ja	Die InfluxDB-Version. Zulässige Werte: `V2`, `V3`.
`secretId`	Zeichenfolge	Ja	Der Name oder ARN des AWS Secrets Manager Geheimnis ses, das Ihr InfluxDB-Token enthält.
`secretType`	Zeichenfolge	Nein	Der Typ des geheimen Werts. Zulässige Werte: `SecretString` , `SecretBinary` .
`secretKey`	Zeichenfolge	Nein	Der Schlüssel innerhalb des geheimen JSON-Codes, der das Authentifizierungs token enthält. Nur erforderlich, wenn das Geheimnis ein JSON-

Parameter	Typ	Erforderlich	Beschreibung
			Objekt mit mehreren Schlüsseln ist.

Überprüfung des Besitzes von Endpunkten

Wenn Sie ein InfluxDB-Aktionsziel erstellen, AWS IoT Core validiert es den Besitz des Endpunkts, indem es sich mit den von Ihnen angegebenen Anmeldeinformationen gegenüber der InfluxDB-API authentifiziert:

- InfluxDB V2: ruft den Endpunkt auf. AWS IoT Core `/api/v2/me`
- InfluxDB V3: AWS IoT Core Ruft den Endpunkt () der Listendatenbanken auf. `GET /api/v3/configure/database`

Eine erfolgreiche Antwort (2xx) setzt den Zielstatus auf. `ENABLED` Bei einem Fehler wird AWS IoT Core der Status auf gesetzt`ERROR`. Um die Validierung erneut zu versuchen, rufen Sie `UpdateTopicRuleDestination` mit dem Status auf an`IN_PROGRESS`.

Zuordnung der InfluxDB-Terminologie

Die folgenden Begriffe unterscheiden sich zwischen InfluxDB V2 und V3.

AWS IoT Core Parameter	Begriff InfluxDB V2	Begriff InfluxDB V3
`databaseName`	Bucket	Datenbank
`tableName`	Messung	Tabelle

> **Note**
>
> Wenn Sie von der Amazon Timestream-Regelaktion migrieren, `dimensions` wird der Parameter den Tags in der InfluxDB-Aktion zugeordnet, und die Attribute der Abfrageergebnisse werden Feldern zugeordnet.

Parameters

Wenn Sie eine AWS IoT Regel mit der InfluxDB-Aktion erstellen, müssen Sie die folgenden Informationen angeben:

`destinationArn`

Der ARN des InfluxDB-Aktionsziels. Siehe InfluxDB-Aktionsziel. Unterstützt Ersatzvorlagen: Nein

`roleArn`

Der ARN der IAM-Rolle, die die AWS IoT Berechtigung zum Zugriff auf das Secrets Manager Manager-Geheimnis gewährt. Siehe Voraussetzungen. Unterstützt Ersatzvorlagen: Nein

`databaseName`

Der Name der InfluxDB-Datenbank (in InfluxDB v2 als Bucket oder in InfluxDB v3 als Datenbank bezeichnet), in die Datensätze geschrieben werden sollen. Unterstützt Ersatzvorlagen: Nein. Um Daten an verschiedene Datenbanken weiterzuleiten, erstellen Sie separate Regelaktionen für jede Datenbank.

`tableName`

Der Name der Tabelle (in InfluxDB v2 als Messung oder in InfluxDB v3 als Tabelle bezeichnet), in die Datensätze geschrieben werden sollen. Unterstützt Ersatzvorlagen: Ja

`organization`

Der Name der InfluxDB-Organisation. Erforderlich für InfluxDB v2. Wenn Sie diesen Parameter für InfluxDB v3 einbeziehen, wird er ignoriert. Unterstützt Ersatzvorlagen: Nein.

`tags`

Metadaten für jeden Punkt, angegeben als Karte. Jeder Kartenschlüssel ist ein Tagname, und jeder Kartenwert ist der entsprechende Tag-Wert. Tags werden aus Gründen der Abfrageleistung indexiert.

- In InfluxDB V3 muss jeder Tagname innerhalb einer Tabelle eindeutig sein und darf keinen Feldnamen duplizieren.
- Tag-Werte unterstützen Vorlagen für den Nachrichtenbereich und die Ersetzung pro Element.

`timestampUnit`

Die Genauigkeit des Zeitstempelwerts in der Nutzlast. Gültige Werte: `s` (Sekunden) | (Millisekunden) | `ms` (Mikrosekunden) | `us` (Nanosekunden). `ns` Standard: `ms`. Unterstützt Substitutionsvorlagen: Nein.

`batchConfig`

(Optional) Server-side Batching-Konfiguration. Weitere Informationen finden Sie unter Stapelverarbeitung.

- `maxBatchSize`— Maximale Anzahl von Punkten in jedem Stapel. Gültiger Bereich: 1—500.
- `maxBatchOpenMs`— Maximale Zeit in Millisekunden, um einen Stapel geöffnet zu halten. Gültiger Bereich: 5—1.000.
- `maxBatchSizeBytes`— Maximale Gesamtgröße in Byte vor dem Leeren. Gültiger Bereich: 100—131.072.
- `batchAcrossTopics` – Boolesch. Wann`true`, der Stapel enthält Punkte aus Nachrichten zu unterschiedlichen Themen. Standard: `false`.

Stapelverarbeitung

Die InfluxDB-Aktion unterstützt zwei Batching-Modi.

Client-side Batching

Ihr IoT-Gerät stapelt Zeitreihendaten als JSON-Array und veröffentlicht sie als einzelne MQTT-Nachricht. Beim clientseitigen Batching wird jedes Array-Element zu einem Zeilenprotokollpunkt in einer einzigen Schreibanforderung. Sie benötigen keine zusätzliche Konfiguration.

Server-side Batching

Verwenden Sie serverseitiges Batching, um einzelne Nachrichten zu gruppieren, bevor Sie sie in InfluxDB schreiben. Konfigurieren Sie serverseitiges Batching mithilfe des Parameters. `batchConfig` Der Stapel wird geleert, wenn ein konfiguriertes Limit (`maxBatchSizemaxBatchOpenMs`, oder`maxBatchSizeBytes`) zuerst erreicht wird.

> **Note**
>
> Sowohl serverseitiges Batching als auch clientseitiges Batching (JSON-Array-Payloads) können gleichzeitig konfiguriert werden.

Die InfluxDB-Aktion wird basierend auf der Größe der ausgehenden Nutzlast in Schritten von 5 KiB gemessen. Line-protocol Konvertierungsfehler werden ebenfalls gemessen.

Per-element Vorlagen für Array-Payloads

Wenn Ihre IoT-Geräte gestapelte Zeitreihendaten als JSON-Array senden, können Sie die „Ersatzvorlagen pro Element“ verwenden, um Werte aus jedem einzelnen Array-Element aufzulösen. Dadurch wird jeder Datenpunkt an eine andere Tabelle weitergeleitet oder elementspezifische Tags angewendet. Weitere Informationen finden Sie unter Per-element Ersatz für Array-Payloads in AWS IoT Regeln.

Syntax von zwei Substitutionsvorlagen

- `${expression}`— Wird im Nachrichtenbereich anhand der eingehenden Gerätenachricht aufgelöst. Wird für jede Nachricht einmal ausgewertet; derselbe Wert gilt für jeden Punkt im Array.
- `@{expression}`— Wird im Elementbereich anhand eines einzelnen Elements der Nutzlast aufgelöst, die durch die SQL SELECT-Anweisung der Regel erzeugt wird. Re-evaluated für jedes Array-Element, sodass jeder Punkt einen anderen Wert erhalten kann. Weitere Informationen finden Sie in der `@{expression}`Referenz.

Verwenden Sie `@{...}` in `tableName` - und Tag-Werte, um den Ausdruck für jedes Array-Element einzeln aufzulösen.

Beispiel

Angesichts der folgenden Gerätenutzlast (JSON-Array):

```
[
  {"measurement_type": "temperature", "room": "kitchen", "timestamp": 1700000000000,
 "value": 23.5},
  {"measurement_type": "humidity", "room": "bedroom", "timestamp": 1700000001000,
 "value": 60.1}
]
```

Und die folgende Aktionskonfiguration:

```
{
  "influxDB": {
    "destinationArn": "arn:aws:iot:us-west-2:111122223333:ruledestination/influxdb/
abc123",
    "roleArn": "arn:aws:iam::111122223333:role/iot-influxdb-role",
    "databaseName": "sensor_data",
    "tableName": "@{measurement_type}",
```

```
    "tags": {
      "room": "@{room}"
    },
    "timestampUnit": "ms"
  }
}
```

Die resultierende Zeilenprotokollausgabe enthält zwei Punkte:

```
temperature,room=kitchen value=23.5 1700000000000
humidity,room=bedroom value=60.1 1700000001000
```

> **Note**
>
> Ein Feld, auf das von verwiesen `@{...}` wird, wird aus dem Linienprotokoll-Feldsatz entfernt, sodass es nicht auch als Feld angezeigt wird. In diesem Beispiel `measurement_type` wird es zum Tabellennamen und `room` zu einem Tag, sodass keines von beiden in der Feldgruppe vorkommt — `value` es ist das einzige Feld.

Einschränkungen

- `@{...}`wird nur in der InfluxDB-Aktionskonfiguration (`tableName`und den Tag-Werten) unterstützt.
- Sie können `@{...}` in der Regel kein SQL (SELECT/WHEREKlauseln), Fehleraktionsdefinitionen oder andere Regelaktionen verwenden.
- Nur Feldverweise werden intern `@{...}` unterstützt. Funktionen werden nicht unterstützt.
- Jeder Wert unterstützt höchstens eine `@{...}` Markierung. Mehrere Markierungen in einem einzigen Wert führen zu einer API-Ausnahme.
- Sie können den gleichen Wert nicht mischen`${...}`. `@{...}` Das Mischen erzeugt eine API-Ausnahme.
- Ein einzelnes JSON-Objekt wird als ein Array mit einem Element behandelt.

Inhalt des InfluxDB-Datensatzes

Für jeden Datensatz im Post-SQL-Abfrageergebnis enthält der resultierende InfluxDB-Zeilenprotokollpunkt die folgenden Komponenten:

Komponente	Quelle
Tabelle	Der Parameterwert `tableName`
Tags (Markierungen)	Die Schlüssel-Wert-Paare aus dem Parameter `tags`
Felder	Die verbleibenden Payload-Attribute werden nicht als Tags, Tabellenname oder Zeitstempel verwendet
Zeitstempel	Der aus der `timestamp` Payload extrahierte Schlüssel

Datentypkonvertierung

AWS IoT Core konvertiert JSON-Werte wie folgt in InfluxDB-Zeilenprotokolltypen:

JSON-Typ	Typ des Leitungsprotokolls	Beispiel
Ganzzahl (−2 ³ bis 2 ³ −1)	Ganzzahl mit `i` Vorzeichen (Suffix)	`42` → `42i`
Ganzzahl (2³ bis 2−1)	Ganzzahl ohne Vorzeichen (Suffix) u	`9223372036854775808` → `9223372036854775808u`
Ganzzahl außerhalb (−2³ bis 2−1)	Abgelehnt (Fehleraktion ausgelöst)	—
Float/Dezimal	IEEE-754 64-Bit-Float	`23.5` → `23.5`
Boolesch	`t` oder `f`	`true` → `t`
Zeichenfolge	Zeichenfolge in Anführung szeichen	`"active"` → `"active"`
Null	Ausgelassen (nicht geschrieb en)	—

JSON-Typ	Typ des Leitungsprotokolls	Beispiel
Objekt oder Array	Komprimierte JSON-Zeic henfolge (ohne Leerzeichen)	`{"a":1}` → `"{\"a\":1}"`

Einschränkungen bei der Benennung

- Feldtasten und Tag-Tasten dürfen nicht leer sein oder mit einem Unterstrich (_) beginnen.
- In InfluxDB V3 müssen Tabellenname und Tag-Schlüssel mit einem Buchstaben oder einer Ziffer beginnen.
- Kommas, Gleichheitszeichen und Leerzeichen in Feldschlüsseln, Tag-Schlüsseln und Tag-Werten werden automatisch maskiert.
- Maße: Kommas und Leerzeichen werden automatisch maskiert.
- Tags werden vor der Serialisierung alphabetisch nach Schlüsseln sortiert, um die Aufnahmeleistung zu verbessern.
- Leere Tag-Werte werden in der Ausgabe weggelassen.

Reservierte Schlüssel

Die folgenden Payload-Schlüssel werden bei der Konvertierung des Leitungsprotokolls aus dem Feldsatz entfernt:

- `timestamp`— Wird als Zeitstempel für den Zeitpunkt verwendet.
- Schlüssel, auf die `tableName` (über `${...}` oder`@{...}`) verwiesen wird — Wird als Tabellenname verwendet.
- Schlüssel, auf die durch den Tag-Wert verwiesen wird (über `${...}` oder`@{...}`) — Werden als Tag-Werte verwendet.

Aktionen bei Fehlern

Wenn die InfluxDB-Aktion fehlschlägt, wird die konfigurierte Fehleraktion ausgelöst.

InfluxDB als Fehleraktion verwenden

Sie können die InfluxDB-Aktion als Fehleraktion für jede Regel konfigurieren. Die gesamte Nutzlast wird `tableName` als ein Datensatz in einen einzigen Datensatz geschrieben. Message-scope Ersatzvorlagen (`${...}`) werden in den Fehleraktionen und unterstützt. `tableName` `tags`

Ausgabe der Fehleraktion

Wenn die Fehleraktion ausgelöst wird, enthält die Ausgabe-Payload:`ruleName`,`topic`,`cloudwatchTraceId`,`clientId`,`sourceIp`, `base64OriginalPayload` (Base64-encoded ursprüngliche Nachricht) und ein `failures` Array, in dem jeder Eintrag `failedActionfailedResource`, und `errorMessage` hat.

Beim serverseitigen Batching verwendet `payloadsWithMetadata` die Nutzlast einen Eintrag für jede einzelne eingehende MQTT-Nachricht. Die Werte jedes Fehlers beziehen sich auf die `affectedIds` `id` Werte dieser Nachrichteneinträge; es handelt sich nicht um Punktindizes. Ein clientseitiges JSON-Array ist eine eingehende Nachricht, obwohl es mehrere InfluxDB-Punkte erzeugt. Das vollständige Payload-Format finden Sie unter. Fehleraktionen beim Batching

Szenario eines Fehlers	Description
Ziel DEAKTIVIERT oder FEHLER	Das InfluxDB-Aktionsziel ist nicht aktiviert . Überprüfen Sie, ob die Überprüfung des Besitzes der Endgeräte erfolgreich
Ungültiger Ziel-ARN	Das angegebene Ziel ist nicht vorhanden.
Ungültiger Rollen-ARN	Die IAM-Rolle ist nicht vorhanden oder es fehlen Berechtigungen.
Fehler beim Abrufen des geheimen Abrufs	Das geheime oder konfigurierte Geheimnis ist `secretKey` nicht vorhanden, oder die Regelaktionsrolle kann das Geheimnis nicht abrufen oder entschlüsseln.
Fehlender Zeitstempel	Die Payload enthält keinen Schlüssel. `timestamp` Jedes Objekt in der Nutzlast muss ein `timestamp` Feld mit einem ganzzahligen Unix-Epochenwert enthalten.

Szenario eines Fehlers	Description
Ungültiger Zeitstempelwert	Der Zeitstempelwert ist keine Ganzzahl (z. B. eine Zeichenfolge, eine Fließkommazahl oder ein ISO-8601 Datum). Der Wert muss eine Ganzzahl für die Unix-Epoche in der von angegebenen Einheit sein. `timestampUnit`
Ungültige Nutzlast (keine Felder)	Die Payload enthält keine gültigen Felder für das Leitungsprotokoll, nachdem reservierte Schlüssel entfernt wurden.
Ungültiger Feldschlüssel	Ein Feldschlüssel ist leer oder beginnt mit_.
Der Client-Batch enthält einen ungültigen Punkt	Bei einem oder mehreren Elementen in einem JSON-Array ist die Zeilenprotokollvalidierung fehlgeschlagen.
Felder mit Tags (+) überschreiten das Spaltenli mit	Kombinierte Tag- und Feldschlüssel überschre iten die maximale Spaltenanzahl (250).
Verbindungsfehler	AWS IoT Core konnte keine Verbindung zum InfluxDB-Endpunkt herstellen.
Authentifizierungsfehler	Das InfluxDB-Token ist ungültig oder abgelaufe n. Aktualisieren Sie das Geheimnis in. AWS Secrets Manager
Die Ressource wurde nicht gefunden	Die angegebene Datenbank, Tabelle oder Organisation ist in InfluxDB nicht vorhanden.
Feldtypkonflikt	Ein oder mehrere Felder stehen in Konflikt mit dem vorhandenen Schema. Der gesamte Batch-Schreibvorgang schlägt fehl.
InfluxDB-Serverfehler	In InfluxDB ist ein interner Fehler aufgetreten.

Szenario eines Fehlers	Description
Der InfluxDB-Dienst ist nicht verfügbar	InfluxDB ist vorübergehend nicht verfügbar. Die Rules Engine versucht es erneut mit exponenti ellem Backoff.

> **Important**
>
> Ein Feldtypkonflikt an einem beliebigen Punkt in einem Batch führt dazu, dass der gesamte Batch-Schreibvorgang fehlschlägt. InfluxDB überträgt Punkte nicht teilweise — entweder sind alle Punkte erfolgreich oder der gesamte Schreibvorgang wird abgelehnt.

Wiederholbare Fehler (503) werden mit exponentiellem Backoff wiederholt. AWS IoT Core Lädt bei einer HTTP 401-Antwort das Token von neu und wiederholt die Anfrage einmal. AWS Secrets Manager Non-retryable Fehler (404, 422) lösen die Fehleraktion sofort aus. Die Grenzwerte für Wiederholungsversuche finden Sie unter AWS IoT Core Dienstkontingente.

Beispiele

Aktion der InfluxDB-Regel

```
{
  "topicRulePayload": {
    "sql": "SELECT * FROM 'devices/+/telemetry'",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "influxDB": {
          "destinationArn": "arn:aws:iot:us-west-2:111122223333:ruledestination/
influxdb/a1b2c3d4",
          "roleArn": "arn:aws:iam::111122223333:role/iot-influxdb-role",
          "organization": "my-org",
          "databaseName": "sensor_data",
          "tableName": "device_metrics",
          "tags": {
            "device_id": "${clientid()}",
            "location": "building-a"
          },
```

```
          "timestampUnit": "ms"
        }
      }
    ]
  }
}
```

Beispiel für eine Nutzlast:

```
{
  "timestamp": 1700000000000,
  "temperature": 23.5,
  "humidity": 60.1,
  "pressure": 1013.25,
  "battery_level": 87
}
```

Resultierendes Leitungsprotokoll:

```
device_metrics,device_id=myDevice123,location=building-a
 temperature=23.5,humidity=60.1,pressure=1013.25,battery_level=87i 1700000000000
```

Die Reihenfolge der Felder in der Ausgabe kann variieren — die Felder sind nicht alphabetisch sortiert.

Client-batched Array-Nutzlast mit Vorlagen pro Element

```
{
  "influxDB": {
    "destinationArn": "arn:aws:iot:us-west-2:111122223333:ruledestination/influxdb/
a1b2c3d4",
    "roleArn": "arn:aws:iam::111122223333:role/iot-influxdb-role",
    "organization": "my-org",
    "databaseName": "sensor_data",
    "tableName": "@{measurement_type}",
    "tags": {
      "sensor_id": "@{sensor_id}",
      "location": "${topic(2)}"
    },
    "timestampUnit": "ns"
  }
```

```
}
```

Beispiel für eine Nutzlast (veröffentlicht in): **`devices/floor3/telemetry`**

```
[
  {"measurement_type": "temperature", "sensor_id": "sensor-42", "timestamp":
 1700000000000000000, "value": 23.5},
  {"measurement_type": "humidity", "sensor_id": "sensor-42", "timestamp":
 1700000001000000000, "value": 60.1},
  {"measurement_type": "pressure", "sensor_id": "sensor-43", "timestamp":
 1700000002000000000, "value": 1013.25}
]
```

Resultierendes Leitungsprotokoll:

```
temperature,location=floor3,sensor_id=sensor-42 value=23.5 1700000000000000000
humidity,location=floor3,sensor_id=sensor-42 value=60.1 1700000001000000000
pressure,location=floor3,sensor_id=sensor-43 value=1013.25 1700000002000000000
```

Server-side Batching mit InfluxDB

Um serverseitiges Batching zu einer InfluxDB-Aktion hinzuzufügen, nehmen Sie Folgendes in die Aktionskonfiguration auf: `batchConfig`

```
"batchConfig": {
  "maxBatchSize": 50,
  "maxBatchOpenMs": 1000,
  "maxBatchSizeBytes": 65536,
  "batchAcrossTopics": false
}
```

IAM-Richtlinie für die Rolle „Regelaktion“

Vertrauensrichtlinie:

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

```
      "Principal": {"Service": "iot.amazonaws.com"},
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Genehmigungsrichtlinie:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "secretsmanager:GetSecretValue",
      "Resource": "arn:aws:secretsmanager:us-west-2:111122223333:secret:my-influxdb-
secret-a1b2c3"
    }
  ]
}
```

Kombiniertes clientseitiges und serverseitiges Batching (Neuordnung von Punkten)

Wenn Sie sowohl clientseitiges Batching (JSON-Array-Payloads) als auch serverseitiges Batching () aktivieren, beachten Sie, dass der serverseitige Batch Punkte aus einer clientseitigen Batch-Payload neu anordnen kann. `batchConfig` Die Rules Engine sammelt Punkte aus mehreren eingehenden Nachrichten in einem einzigen serverseitigen Batch. Da Nachrichten asynchron von unterschiedlichen Geräten oder Themen empfangen werden, können Punkte, die innerhalb der ursprünglichen Client-Nutzlast angeordnet waren, beim letzten Schreibvorgang mit Punkten aus anderen Nachrichten verschachtelt werden.

Konfiguration der Aktion:

```
{
  "influxDB": {
    "destinationArn": "arn:aws:iot:us-west-2:111122223333:ruledestination/influxdb/
abc123",
    "roleArn": "arn:aws:iam::111122223333:role/iot-influxdb-role",
    "databaseName": "sensor_data",
    "tableName": "@{measurement_type}",
    "tags": {
      "device_id": "${topic(2)}",
```

```
      "floor": "@{floor}"
    },
    "timestampUnit": "ms",
    "batchConfig": {
      "maxBatchSize": 100,
      "maxBatchOpenMs": 500,
      "maxBatchSizeBytes": 65536,
      "batchAcrossTopics": true
    }
  }
}
```

Gerät A veröffentlicht zum `devices/deviceA/telemetry` Zeitpunkt T:

```
[
  {"measurement_type": "temperature", "floor": "1", "timestamp": 1700000000000,
 "value": 22.1},
  {"measurement_type": "temperature", "floor": "2", "timestamp": 1700000000100,
 "value": 23.4},
  {"measurement_type": "humidity", "floor": "1", "timestamp": 1700000000200, "value":
 55.0}
]
```

Gerät B veröffentlicht zum `devices/deviceB/telemetry` Zeitpunkt T+10 ms:

```
[
  {"measurement_type": "temperature", "floor": "3", "timestamp": 1700000000050,
 "value": 21.8},
  {"measurement_type": "humidity", "floor": "3", "timestamp": 1700000000150, "value":
 62.3}
]
```

Beide Nachrichten kommen innerhalb des Batchfensters von 500 ms an (`maxBatchOpenMs`), sodass die Rules Engine alle fünf Punkte zu einem einzigen serverseitigen Batch zusammenfasst.

Resultierendes Zeilenprotokoll (serverseitiges Schreiben im Batch-Modus):

```
temperature,device_id=deviceB,floor=3 value=21.8 1700000000050
temperature,device_id=deviceA,floor=1 value=22.1 1700000000000
temperature,device_id=deviceA,floor=2 value=23.4 1700000000100
humidity,device_id=deviceB,floor=3 value=62.3 1700000000150
```

```
humidity,device_id=deviceA,floor=1 value=55.0 1700000000200
```

Beachten Sie, dass sich die Punkte nicht mehr in der Reihenfolge befinden, in der sie in den einzelnen Client-Nutzdaten erschienen sind. Die drei Punkte von Gerät A (Zeitstempel 1700000000000, 1700000000100, 1700000000200) sind mit den beiden Punkten von Gerät B (Zeitstempel 1700000000050, 1700000000150) verschachtelt. Der serverseitige Batch garantiert nicht die ursprüngliche Reihenfolge in jeder Nachricht. InfluxDB verwendet das Timestamp-Feld, um jeden Punkt auf der Timeline zu platzieren, sodass eine Neuordnung die Richtigkeit der Abfrage nicht beeinträchtigt. Wenn Ihre Anwendung jedoch auf Semantik der Schreibreihenfolge angewiesen ist (z. B. die Behandlung von Feldtypkonflikten oder die Deduplizierung von Last-Write-Wins innerhalb derselben Millisekunde), beachten Sie, dass die effektive Schreibreihenfolge von der Veröffentlichungsreihenfolge abweichen kann.

AWS IoT SiteWise

Die Aktion AWS IoT SiteWise (`iotSiteWise`) sendet Daten aus einer MQTT-Nachricht an die Asset-Eigenschaften in. AWS IoT SiteWise

Sie können einem Tutorial folgen, das Ihnen zeigt, wie Sie Daten von AWS IoT Dingen aufnehmen können. Weitere Informationen finden Sie im Tutorial Daten AWS IoT SiteWise aus AWS IoT Dingen aufnehmen oder im Abschnitt Daten mithilfe von AWS IoT Kernregeln aufnehmen im AWS IoT SiteWise Benutzerhandbuch.

Voraussetzungen

Diese Regelaktion hat die folgenden Anforderungen:

- Eine IAM-Rolle, die die Ausführung des Vorgangs übernehmen AWS IoT kann. `iotsitewise:BatchPutAssetPropertyValue` Weitere Informationen finden Sie unter Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt.

  Sie können die folgende Beispiel-Vertrauensrichtlinie mit der Rolle verknüpfen.

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

```
            "Resource": "*"
        }
    ]
}
```

Um die Sicherheit zu erhöhen, können Sie in der `Condition` Eigenschaft einen Pfad zur AWS IoT SiteWise Asset-Hierarchie angeben. Das folgende Beispiel ist eine Vertrauensrichtlinie, die einen Komponentenhierarchiepfad angibt.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotsitewise:BatchPutAssetPropertyValue",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iotsitewise:assetHierarchyPath": [
                        "/root node asset ID",
                        "/root node asset ID/*"
                    ]
            }
        }
        }
    ]
}
```

- Wenn Sie AWS IoT SiteWise mit dieser Aktion Daten an senden, müssen Ihre Daten die Anforderungen des `BatchPutAssetPropertyValue` Vorgangs erfüllen. Weitere Informationen finden Sie unter BatchPutAssetPropertyValue in der AWS IoT SiteWise -API-Referenz.

Parameters

Wenn Sie mit dieser Aktion eine AWS IoT Regel erstellen, müssen Sie die folgenden Informationen angeben:

`putAssetPropertyValueEntries`

Eine Liste mit Werteinträgen für die Komponenteneigenschaft, die jeweils die folgenden Informationen enthalten:

`propertyAlias`

(Optional) Der Eigenschaftenalias, der Ihrer Komponenteneigenschaft zugeordnet ist. Geben Sie entweder einen `propertyAlias` oder sowohl eine `assetId` als auch eine `propertyId` an. Weitere Informationen zu Eigenschaftsaliasnamen finden Sie unter Zuordnen von industriellen Datenströmen zu Komponenteneigenschaften im AWS IoT SiteWise -Benutzerhandbuch.

Unterstützt Ersatzvorlagen: Ja

`assetId`

(Optional) Die ID des AWS IoT SiteWise Assets. Geben Sie entweder einen `propertyAlias` oder sowohl eine `assetId` als auch eine `propertyId` an.

Unterstützt Ersatzvorlagen: Ja

`propertyId`

(Optional) Die ID der Komponenteneigenschaft. Geben Sie entweder einen `propertyAlias` oder sowohl eine `assetId` als auch eine `propertyId` an.

Unterstützt Ersatzvorlagen: Ja

`entryId`

(Optional) Ein eindeutiger Bezeichner für diesen Eintrag. Definieren Sie die `entryId`, um besser zu nachzuverfolgen, welche Nachricht ggf. einen Fehler verursacht hat. Standardmäßig wird eine neue UUID verwendet.

Unterstützt Ersatzvorlagen: Ja

`propertyValues`

Eine Liste der einzufügenden Eigenschaftswerte, die jeweils Zeitstempel, Qualität und Wert (Timestamp, Quality and Value, TQV) im folgenden Format enthalten:

`timestamp`

Eine Zeitstempelstruktur, die die folgenden Informationen enthält:

`timeInSeconds`

Eine Zeichenfolge, die die Zeit in Sekunden in der Unix-Epochenzeit enthält. Wenn Ihre Nachrichtennutzlast keinen Zeitstempel hat, können Sie timestamp() verwenden, der die aktuelle Zeit in Millisekunden zurückgibt. Um diese Zeit in Sekunden zu konvertieren,

können Sie die folgende Ersetzungsvorlage verwenden: **`${floor(timestamp() / 1E3)}`**.

Unterstützt Ersatzvorlagen: Ja

`offsetInNanos`

(Optional) Eine Zeichenfolge, die den Zeitversatz in Nanosekunden von der Zeit in Sekunden enthält. Wenn Ihre Nachrichtennutzlast keinen Zeitstempel hat, können Sie timestamp() verwenden, der die aktuelle Zeit in Millisekunden zurückgibt. Um den Nanosekunden-Zeitversatz von diesem Zeitpunkt zu berechnen, können Sie die folgende Ersetzungsvorlage verwenden: **`${(timestamp() % 1E3) * 1E6}`**.

Unterstützt Ersatzvorlagen: Ja

In Bezug auf die Unix-Epochenzeit werden nur Einträge AWS IoT SiteWise akzeptiert, die einen Zeitstempel von bis zu 7 Tagen in der Vergangenheit und bis zu 5 Minuten in der future haben.

`quality`

(Optional) Eine Zeichenfolge, die die Qualität des Werts beschreibt. Zulässige Werte: `GOOD`, `BAD`, `UNCERTAIN`.

Unterstützt Ersatzvorlagen: Ja

`value`

Eine Wertestruktur, die eines der folgenden Wertfelder enthält, je nach Datentyp der Komponenteneigenschaft:

`booleanValue`

(Optional) Eine Zeichenfolge, die den booleschen Wert des Werteintrags enthält.

Unterstützt Ersatzvorlagen: Ja

`doubleValue`

(Optional) Eine Zeichenfolge, die den doppelten Wert des Werteintrags enthält.

Unterstützt Ersatzvorlagen: Ja

`integerValue`

(Optional) Eine Zeichenfolge, die den Ganzzahlwert des Werteintrags enthält.

Unterstützt Ersatzvorlagen: Ja

`stringValue`

(Optional) Der Zeichenfolgenwert des Werteintrags.

Unterstützt Ersatzvorlagen: Ja

`roleArn`

Der ARN der IAM-Rolle, die die AWS IoT Berechtigung zum Senden eines Vermögenseigenschaftswerts an AWS IoT SiteWise erteilt. Weitere Informationen finden Sie unter Voraussetzungen.

Unterstützt Ersatzvorlagen: Nein

Beispiele

Das folgende JSON-Beispiel definiert eine grundlegende SiteWise IoT-Aktion in einer AWS IoT Regel.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "iotSiteWise": {
                    "putAssetPropertyValueEntries": [
                        {
                            "propertyAlias": "/some/property/alias",
                            "propertyValues": [
                                {
                                    "timestamp": {
                                        "timeInSeconds": "${my.payload.timeInSeconds}"
                                    },
                                    "value": {
                                        "integerValue": "${my.payload.value}"
                                    }
                                }
                            ]
                        }
```

```
                    ],
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sitewise"
                }
            }
        ]
    }
}
```

Das folgende JSON-Beispiel definiert eine SiteWise IoT-Aktion in einer AWS IoT Regel. In diesem Beispiel wird das Thema als Eigenschaftsalias und die `timestamp()` Funktion verwendet. Wenn Sie beispielsweise Daten in `/company/windfarm/3/turbine/7/rpm` veröffentlichen, sendet diese Aktion die Daten an die Komponenteneigenschaft mit einem Eigenschaftsalias, der dem von Ihnen angegebenen Thema entspricht.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM '/company/windfarm/+/turbine/+/+'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "iotSiteWise": {
                    "putAssetPropertyValueEntries": [
                        {
                            "propertyAlias": "${topic()}",
                            "propertyValues": [
                                {
                                    "timestamp": {
                                        "timeInSeconds": "${floor(timestamp() / 1E3)}",
                                        "offsetInNanos": "${(timestamp() % 1E3) * 1E6}"
                                    },
                                    "value": {
                                        "doubleValue": "${my.payload.value}"
                                    }
                                }
                            ]
                        }
                    ],
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sitewise"
                }
            }
        ]
    }
```

```
}
```

Weitere Informationen finden Sie auch unter

- Was ist AWS IoT SiteWise? im AWS IoT SiteWise Benutzerhandbuch
- Daten mithilfe von AWS IoT Core Regeln im AWS IoT SiteWise Benutzerhandbuch aufnehmen
- Daten aus AWS IoT Dingen AWS IoT SiteWise aus dem Benutzerhandbuch aufnehmen AWS IoT SiteWise
- Problembehandlung bei einer AWS IoT SiteWise Regelaktion im AWS IoT SiteWise Benutzerhandbuch

Firehose

Die Aktion Firehose (`firehose`) sendet Daten aus einer MQTT-Nachricht an einen Amazon Data Firehose-Stream.

Voraussetzungen

Diese Regelaktion hat die folgenden Anforderungen:

- Eine IAM-Rolle, die die Ausführung des AWS IoT Vorgangs übernehmen kann. `firehose:PutRecord` Weitere Informationen finden Sie unter Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt.

  In der AWS IoT Konsole können Sie eine Rolle auswählen oder erstellen, um die Ausführung dieser Regelaktion AWS IoT zu ermöglichen.
- Wenn Sie Firehose verwenden, um Daten an einen Amazon S3-Bucket zu senden, und Sie einen AWS KMS Kunden verwenden, der es geschafft hat, ruhende Daten in Amazon S3 AWS KMS key zu verschlüsseln, muss Firehose Zugriff auf Ihren Bucket und die Erlaubnis haben, diesen im Namen des AWS KMS key Anrufers zu verwenden. Weitere Informationen finden Sie unter Grant Firehose access to a Amazon S3 destination im Amazon Data Firehose Developer Guide.

Parameters

Wenn Sie eine AWS IoT Regel mit dieser Aktion erstellen, müssen Sie die folgenden Informationen angeben:

`batchMode`

(Optional) Ob der Firehose-Stream als Batch bereitgestellt werden soll, indem `PutRecordBatch` Der Standardwert ist `false`.

Wenn `batchMode` `true` ist und die SQL-Anweisung der Regel zu einem Array ausgewertet wird, bildet jedes Array-Element einen Datensatz in der `PutRecordBatch`-Anforderung. Das resultierende Array darf nicht mehr als 500 Datensätze enthalten.

Unterstützt Ersatzvorlagen: Nein

`deliveryStreamName`

Der Firehose-Stream, in den die Nachrichtendaten geschrieben werden sollen.

Unterstützt Ersatzvorlagen: API und nur AWS CLI

`separator`

(Optional) Ein Zeichentrennzeichen, das verwendet wird, um Datensätze zu trennen, die in den Firehose geschrieben wurden. Wenn Sie diesen Parameter auslassen, verwendet der Stream kein Trennzeichen. Gültige Werte: `,` (Komma), `\t` (Tab), `\n` (Newline), `\r\n` (Windows Newline).

Unterstützt Ersatzvorlagen: Nein

`roleArn`

Die IAM-Rolle, die den Zugriff auf den Firehose-Stream ermöglicht. Weitere Informationen finden Sie unter Voraussetzungen.

Unterstützt Ersatzvorlagen: Nein

Beispiele

Das folgende JSON-Beispiel definiert eine Firehose-Aktion in einer AWS IoT Regel.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
```

```
            {
                "firehose": {
                    "deliveryStreamName": "my_firehose_stream",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_firehose"
                }
            }
        ]
    }
}
```

Das folgende JSON-Beispiel definiert eine Firehose-Aktion mit Ersatzvorlagen in einer AWS IoT Regel.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "firehose": {
                    "deliveryStreamName": "${topic()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_firehose"
                }
            }
        ]
    }
}
```

Weitere Informationen finden Sie auch unter

- Was ist Amazon Data Firehose? im Amazon Data Firehose Developer Guide

Kinesis Data Streams

Die Aktion Kinesis Data Streams (`kinesis`) schreibt Daten aus einer MQTT-Nachricht in Amazon Kinesis Data Streams.

Voraussetzungen

Diese Regelaktion hat die folgenden Anforderungen:

- Eine IAM-Rolle, die die Ausführung des AWS IoT Vorgangs übernehmen kann. `kinesis:PutRecord` Weitere Informationen finden Sie unter Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt.

  In der AWS IoT Konsole können Sie eine Rolle auswählen oder erstellen, um die Ausführung dieser Regelaktion AWS IoT zu ermöglichen.
- Wenn Sie einen AWS KMS vom Kunden verwalteten AWS KMS key (KMS-Schlüssel) verwenden, um ruhende Daten in Kinesis Data Streams zu verschlüsseln, muss der Service über die Erlaubnis verfügen, den im Namen des AWS KMS key Anrufers zu verwenden. Weitere Informationen finden Sie unter Berechtigungen zur Verwendung von benutzergenerierten AWS KMS keys im Amazon Kinesis Data Streams Entwicklerhandbuch.

Parameters

Wenn Sie mit dieser Aktion eine AWS IoT Regel erstellen, müssen Sie die folgenden Informationen angeben:

`stream`

Der Kinesis Data Stream, in den die Daten geschrieben werden

Unterstützt Ersatzvorlagen: API und nur AWS CLI

`partitionKey`

Der Partitionsschlüssel, mit dem bestimmt wird, in welchen Shard die Daten geschrieben werden. Der Partitionsschlüssel besteht in der Regel aus einem Ausdruck (z. B. `${topic()}` oder `${timestamp()}`).

Unterstützt Ersatzvorlagen: Ja

`roleArn`

Der ARN der IAM-Rolle, die die AWS IoT Berechtigung zum Zugriff auf den Kinesis-Datenstream gewährt. Weitere Informationen finden Sie unter Voraussetzungen.

Unterstützt Ersatzvorlagen: Nein

Beispiele

Das folgende JSON-Beispiel definiert eine Kinesis Data Streams Streams-Aktion in einer AWS IoT Regel.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "kinesis": {
                    "streamName": "my_kinesis_stream",
                    "partitionKey": "${topic()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_kinesis"
                }
            }
        ]
    }
}
```

Das folgende JSON-Beispiel definiert eine Kinesis-Aktion mit Ersatzvorlagen in einer AWS IoT Regel.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "kinesis": {
                    "streamName": "${topic()}",
                    "partitionKey": "${timestamp()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_kinesis"
                }
            }
        ]
    }
}
```

Weitere Informationen finden Sie auch unter

- Was ist Amazon Kinesis Data Streams? im Entwicklerhandbuch für Amazon Kinesis Data Streams

Lambda

Eine Lambda (`lambda`) -Aktion ruft eine AWS Lambda Funktion auf und übergibt eine MQTT-Nachricht. AWS IoT ruft Lambda-Funktionen asynchron auf.

Sie können einem Tutorial folgen, das Ihnen veranschaulicht, wie Sie eine Regel mit einer Lambda-Aktion erstellen und testen. Weitere Informationen finden Sie unter Tutorial: Formatieren einer Benachrichtigung mithilfe einer AWS Lambda function.

Voraussetzungen

Diese Regelaktion hat die folgenden Anforderungen:

- AWS IoT Um eine Lambda-Funktion aufzurufen, müssen Sie eine Richtlinie konfigurieren, die die `lambda:InvokeFunction` Berechtigung dazu erteilt. AWS IoT Sie können nur eine Lambda-Funktion aufrufen, die in derselben AWS-Region definiert ist, in der Ihre Lambda-Richtlinie existiert. Lambda-Funktionen verwenden ressourcenbasierte Richtlinien, daher müssen Sie die Richtlinie an die Lambda-Funktion selbst anhängen.

  Verwenden Sie den folgenden AWS CLI Befehl, um eine Richtlinie anzuhängen, die die Berechtigung erteilt. `lambda:InvokeFunction` Ersetzen Sie in diesem Befehl:

  - *`function_name`*mit dem Namen der Lambda-Funktion. Sie fügen eine neue Berechtigung hinzu, um die Ressourcenrichtlinie der Funktion zu aktualisieren.
  - *`region`*mit dem AWS-Region der Funktion.
  - *`account-id`*mit der AWS-Konto Nummer, unter der die Regel definiert ist.
  - *`rule-name`*mit dem Namen der AWS IoT Regel, für die Sie die Lambda-Aktion definieren.
  - *`unique_id`*mit einer eindeutigen Anweisungskennung.

  > **Important**
  >
  > Wenn Sie eine Berechtigung für einen AWS IoT Prinzipal hinzufügen, ohne das `source-arn` oder anzugeben, kann jeder `source-account` AWS-Konto , der mit Ihrer Lambda-

Aktion eine Regel erstellt, Regeln aktivieren, von denen aus Ihre Lambda-Funktion aufgerufen wird. AWS IoT

Weitere Informationen finden Sie unter AWS Lambda Berechtigungen.

```
aws lambda add-permission \
    --function-name function_name \
    --region region \
    --principal iot.amazonaws.com \
    --source-arn arn:aws:iot:region:account-id:rule/rule_name \
    --source-account account-id
    --statement-id unique_id
    --action "lambda:InvokeFunction"
```

- Wenn Sie die AWS IoT Konsole verwenden, um eine Regel für die Lambda-Regelaktion zu erstellen, wird die Lambda-Funktion automatisch ausgelöst. Wenn Sie AWS CloudFormation stattdessen mit dem verwenden `AWS::IoT::TopicRule LambdaAction`, müssen Sie eine `AWS::lambda::Permission`Ressource hinzufügen. Die Ressource erteilt Ihnen dann die Erlaubnis, die Lambda-Funktion auszulösen.

  Der folgende Code zeigt ein Beispiel für das Hinzufügen dieser Ressource. Ersetzen Sie in diesem Beispiel:

  - *function_name*mit dem Namen der Lambda-Funktion.
  - *region*mit dem AWS-Region der Funktion.
  - *account-id*mit der AWS-Konto Nummer, unter der die Regel definiert ist.
  - *rule-name*mit dem Namen der AWS IoT Regel, für die Sie die Lambda-Aktion definieren.

```
Type: AWS::Lambda::Permission
Properties:
  Action: lambda:InvokeFunction
  FunctionName: !Ref function_name
  Principal: "iot.amazonaws.com"
  SourceAccount: account-id
  SourceArn: arn:aws:iot:region:account-id:rule/rule_name
```

- Wenn Sie einen AWS KMS Kunden verwenden, der es geschafft hat AWS KMS key , ruhende Daten in Lambda zu verschlüsseln, muss der Dienst über die Erlaubnis verfügen, den im

Namen des AWS KMS key Anrufers zu verwenden. Weitere Informationen finden Sie unter Verschlüsselung im Ruhezustand im Entwicklerhandbuch für AWS Lambda .

Parameters

Wenn Sie mit dieser Aktion eine AWS IoT Regel erstellen, müssen Sie die folgenden Informationen angeben:

`functionArn`

Der ARN der aufzurufenden Lambda-Funktion. AWS IoT muss die Erlaubnis haben, die Funktion aufzurufen. Weitere Informationen finden Sie unter Voraussetzungen.

Wenn Sie für Ihre Lambda-Funktion keine Version oder keinen Alias angeben, wird die neueste Version der Funktion heruntergefahren. Sie können eine Version oder einen Alias angeben, wenn Sie eine bestimmte Version Ihrer Lambda-Funktion herunterfahren möchten. Um eine Version oder einen Alias anzugeben, fügen Sie die Version oder den Alias dem ARN der Lambda-Funktion hinzu.

```
arn:aws:lambda:us-east-2:123456789012:function:myLambdaFunction:someAlias
```

Weitere Informationen über Versionsverwaltung und Aliasse finden Sie unter AWS Lambda Funktions-Versionsverwaltung und Aliasse.

Unterstützt Ersatzvorlagen: API und nur AWS CLI

Beispiele

Das folgende JSON-Beispiel definiert eine Lambda-Aktion in einer AWS IoT Regel.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "lambda": {
                    "functionArn": "arn:aws:lambda:us-
east-2:123456789012:function:myLambdaFunction"
```

```
                }
            }
        ]
    }
}
```

Das folgende JSON-Beispiel definiert eine Lambda-Aktion mit Ersatzvorlagen in einer AWS IoT Regel.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "lambda": {
                    "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:
${topic()}"
                }
            }
        ]
    }
}
```

Weitere Informationen finden Sie auch unter

- Was ist? AWS Lambda im AWS Lambda Developer Guide
- Tutorial: Formatieren einer Benachrichtigung mithilfe einer AWS Lambda function

Speicherort

Die Aktion Standort (`location`) sendet Ihre geografischen Standortdaten an Amazon Location Service.

Voraussetzungen

Diese Regelaktion hat die folgenden Anforderungen:

- Eine IAM-Rolle, die die Ausführung des `geo:BatchUpdateDevicePosition` Vorgangs übernehmen AWS IoT kann. Weitere Informationen finden Sie unter Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt.

  In der AWS IoT Konsole können Sie eine Rolle auswählen oder erstellen, um die Ausführung dieser Regelaktion AWS IoT zu ermöglichen.

Parameters

Wenn Sie eine AWS IoT Regel mit dieser Aktion erstellen, müssen Sie die folgenden Informationen angeben:

`deviceId`

Die eindeutige ID des Geräts, das die Standortdaten bereitstellt. Weitere Informationen finden Sie unter `DeviceId` in der Amazon Location Service API-Referenz.

Unterstützt Ersatzvorlagen: Ja

`latitude`

Eine Zeichenfolge, die einen doppelten Wert ergibt, der den Breitengrad des Gerätestandorts darstellt.

Unterstützt Ersatzvorlagen: Ja

`longitude`

Eine Zeichenfolge, die einen doppelten Wert ergibt, der den Längengrad des Gerätestandorts darstellt.

Unterstützt Ersatzvorlagen: Ja

`roleArn`

Die IAM-Rolle, die Zugriff auf die Amazon-Location-Service-Domain gewährt. Weitere Informationen finden Sie unter Voraussetzungen.

`timestamp`

Der Zeitpunkt, zu dem die Standortdaten erfasst wurden. Der Standardwert ist der Zeitpunkt, zu dem die MQTT-Nachricht verarbeitet wurde.

Der `timestamp` Wert besteht aus den folgenden beiden Werten:

- `value`: Ein Ausdruck, der einen Wert für lange Epochenzeit zurückgibt. Sie können die the section called “time_to_epoch (Zeichenfolge, Zeichenfolge)” Funktion verwenden, um aus einem Datums- oder Uhrzeitwert, der in der Nachrichtennutzlast übergeben wurde, einen gültigen Zeitstempel zu erstellen. Unterstützt Ersatzvorlagen: Ja
- `unit`: (Optional) Die Genauigkeit des Zeitstempelwerts, die sich aus dem unter `value` beschriebenen Ausdruck ergibt. Zulässige Werte: `SECONDS` | `MILLISECONDS` | `MICROSECONDS` | `NANOSECONDS`. Der Standardwert ist `MILLISECONDS`. Unterstützt Ersatzvorlagen: API und AWS CLI nur.

`trackerName`

Der Name der Tracker-Ressource in Amazon Location, in der der Standort aktualisiert wird. Weitere Informationen finden Sie unter Tracker im Amazon Location Service Entwicklerhandbuch.

Unterstützt Ersatzvorlagen: API und nur AWS CLI

Beispiele

Das folgende JSON-Beispiel definiert eine Location-Aktion in einer AWS IoT Regel.

```
{
 "topicRulePayload": {
  "sql": "SELECT * FROM 'some/topic'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
   {
    "location": {
     "roleArn": "arn:aws:iam::123454962127:role/service-role/ExampleRole",
     "trackerName": "MyTracker",
     "deviceId": "001",
     "sampleTime": {
      "value": "${timestamp()}",
      "unit": "MILLISECONDS"
     },
     "latitude": "-12.3456",
     "longitude": "65.4321"
    }
   }
  ]
 }
```

```
}
```

Das folgende JSON-Beispiel definiert eine Location-Aktion mit Ersatzvorlagen in einer AWS IoT Regel.

```
{
 "topicRulePayload": {
  "sql": "SELECT * FROM 'some/topic'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
   {
    "location": {
     "roleArn": "arn:aws:iam::123456789012:role/service-role/ExampleRole",
     "trackerName": "${TrackerName}",
     "deviceId": "${DeviceID}",
     "timestamp": {
      "value": "${timestamp()}",
      "unit": "MILLISECONDS"
     },
     "latitude": "${get(position, 0)}",
     "longitude": "${get(position, 1)}"
    }
   }
  ]
 }
}
```

Das folgende Beispiel für eine MQTT-Nutzlast zeigt, wie Ersatzvorlagen im vorherigen Beispiel auf Daten zugreifen. Sie können den get-device-position-history CLI-Befehl verwenden, um zu überprüfen, ob die MQTT-Nutzlastdaten in Ihrem Standort-Tracker geliefert werden.

```
{
 "TrackerName": "mytracker",
 "DeviceID": "001",
 "position": [
  "-12.3456",
  "65.4321"
 ]
}
```

```
aws location get-device-position-history --device-id 001 --tracker-name mytracker
```

```
{
 "DevicePositions": [
  {
   "DeviceId": "001",
   "Position": [
    -12.3456,
    65.4321
   ],
   "ReceivedTime": "2022-11-11T01:31:54.464000+00:00",
   "SampleTime": "2022-11-11T01:31:54.308000+00:00"
  }
 ]
}
```

Weitere Informationen finden Sie auch unter

- Was ist Amazon Location Service? im Amazon Location Service Entwicklerhandbuch.

OpenSearch

Die Aktion OpenSearch (`openSearch`) schreibt Daten aus MQTT-Nachrichten in eine Amazon OpenSearch Service-Domain. Sie können dann Tools wie OpenSearch Dashboards verwenden, um Daten in OpenSearch Service abzufragen und zu visualisieren.

Voraussetzungen

Diese Regelaktion hat die folgenden Anforderungen:

- Eine IAM-Rolle, die die Ausführung des `es:ESHttpPut` Vorgangs übernehmen AWS IoT kann. Weitere Informationen finden Sie unter Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt.

  In der AWS IoT Konsole können Sie eine Rolle auswählen oder erstellen, um die Ausführung dieser Regelaktion AWS IoT zu ermöglichen.
- Wenn Sie einen Kunden einsetzen, der es geschafft hat, ruhende Daten im OpenSearch Service AWS KMS key zu verschlüsseln, muss der Service über die Erlaubnis verfügen, den KMS-Schlüssel im Namen des Anrufers zu verwenden. Weitere Informationen finden Sie unter

Verschlüsselung ruhender Daten für Amazon OpenSearch Service im Amazon OpenSearch Service Developer Guide.

Parameters

Wenn Sie eine AWS IoT Regel mit dieser Aktion erstellen, müssen Sie die folgenden Informationen angeben:

`endpoint`

Der Endpunkt Ihrer Amazon OpenSearch Service-Domain.

Unterstützt Ersatzvorlagen: API und nur AWS CLI

`index`

Der OpenSearch Index, in dem Sie Ihre Daten speichern möchten.

Unterstützt Ersatzvorlagen: Ja

`type`

Der Typ des Dokuments, das Sie speichern

> **Note**
>
> Für OpenSearch Versionen nach 1.0 muss der Wert des `type` Parameters sein`_doc`. Weitere Informationen finden Sie in der OpenSearch -Dokumentation.

Unterstützt Ersatzvorlagen: Ja

`id`

Der eindeutige Bezeichner für jedes Dokument

Unterstützt Ersatzvorlagen: Ja

`roleARN`

Die IAM-Rolle, die den Zugriff auf die OpenSearch Service-Domäne ermöglicht. Weitere Informationen finden Sie unter Voraussetzungen.

Unterstützt Ersatzvorlagen: Nein

Einschränkungen

Die Aktion OpenSearch (`openSearch`) kann nicht verwendet werden, um Daten an VPC-Elasticsearch-Cluster zu liefern.

Beispiele

Das folgende JSON-Beispiel definiert eine OpenSearch Aktion in einer AWS IoT Regel und wie Sie die Felder für die `OpenSearch` Aktion angeben können. Weitere Informationen finden Sie unter OpenSearchAction.

```
{
    "topicRulePayload": {
        "sql": "SELECT *, timestamp() as timestamp FROM 'iot/test'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "openSearch": {
                    "endpoint": "https://my-endpoint",
                    "index": "my-index",
                    "type": "_doc",
                    "id": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_os"
                }
            }
        ]
    }
}
```

Das folgende JSON-Beispiel definiert eine OpenSearch Aktion mit Ersatzvorlagen in einer AWS IoT Regel.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "openSearch": {
                    "endpoint": "https://my-endpoint",
```

```
                    "index": "${topic()}",
                    "type": "${type}",
                    "id": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_os"
                }
            }
        ]
    }
}
```

> **Note**
>
> Das ersetzte `type` Feld funktioniert für OpenSearch Version 1.0. Für alle Versionen nach 1.0 `type` muss der Wert von sein`_doc`.

Weitere Informationen finden Sie auch unter

Was ist Amazon OpenSearch Service? im Amazon OpenSearch Service Developer Guide

Erneut veröffentlichen

Die Aktion neu veröffentlichen (`republish`) veröffentlicht eine MQTT-Nachricht erneut in einem anderen MQTT-Thema.

Voraussetzungen

Diese Regelaktion hat die folgenden Anforderungen:

- Eine IAM-Rolle, die die Ausführung des Vorgangs übernehmen AWS IoT kann. `iot:Publish` Weitere Informationen finden Sie unter Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt.

  In der AWS IoT Konsole können Sie eine Rolle auswählen oder erstellen, um die Ausführung dieser Regelaktion AWS IoT zu ermöglichen.

Parameters

Wenn Sie eine AWS IoT Regel mit dieser Aktion erstellen, müssen Sie die folgenden Informationen angeben:

`headers`

Header-Informationen für MQTT Version 5.0.

Weitere Informationen finden Sie unter RepublishAction und MqttHeaders in der AWS -API-Referenz.

`topic`

Das MQTT-Thema, für das die Nachricht erneut veröffentlicht werden soll

Um in einem reservierten Thema, das mit $ beginnt, erneut zu veröffentlichen, verwenden Sie stattdessen $$. Um zum Beispiel das Thema Geräteschatten `$aws/things/MyThing/shadow/update` neu zu veröffentlichen, geben Sie das Thema als `$$aws/things/MyThing/shadow/update` an.

> **Note**
>
> Das erneute Veröffentlichen zu reservierten Jobthemen wird nicht unterstützt.
> AWS IoT Device Defender Reserve-Themen unterstützen keine HTTP-Veröffentlichung.

Unterstützt Ersatzvorlagen: Ja

`qos`

(Optional) Das QoS (Quality of Service)-Niveau, das verwendet werden soll, wenn Nachrichten erneut veröffentlicht werden. Zulässige Werte: `0`, `1`. Der Standardwert ist `0`. Weitere Informationen zu MQTT QoS finden Sie unter MQTT.

Unterstützt Ersatzvorlagen: Nein

`roleArn`

Die IAM-Rolle, die das Veröffentlichen im MQTT-Thema ermöglicht AWS IoT . Weitere Informationen finden Sie unter Voraussetzungen.

Unterstützt Ersatzvorlagen: Nein

Beispiele

Das folgende JSON-Beispiel definiert eine Aktion zur erneuten Veröffentlichung in einer Regel. AWS IoT

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "republish": {
                    "topic": "another/topic",
                    "qos": 1,
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish"
                }
            }
        ]
    }
}
```

Das folgende JSON-Beispiel definiert eine Aktion zur erneuten Veröffentlichung mit Ersatzvorlagen in einer Regel. AWS IoT

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "republish": {
                    "topic": "${topic()}/republish",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish"
                }
            }
        ]
    }
}
```

Das folgende JSON-Beispiel definiert eine Aktion zur erneuten Veröffentlichung `headers` in einer Regel. AWS IoT

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
```

```
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "republish": {
                    "topic": "${topic()}/republish",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish",
                    "headers": {
                        "payloadFormatIndicator": "UTF8_DATA",
                        "contentType": "rule/contentType",
                        "correlationData": "cnVsZSBjb3JyZWxhdGlvbiBkYXRh",
                        "userProperties": [
                            {
                                "key": "ruleKey1",
                                "value": "ruleValue1"
                            },
                            {
                                "key": "ruleKey2",
                                "value": "ruleValue2"
                            }
                        ]
                    }
                }
            }
        ]
    }
}
```

> **Note**
>
> Die ursprüngliche Quell-IP wird bei der Aktion zur erneuten Veröffentlichung nicht weitergegeben.

S3

Die S3 (`s3`) Aktion schreibt die Daten aus einer MQTT-Nachricht in einen Amazon Simple Storage Service (Amazon S3)-Bucket.

Voraussetzungen

Diese Regelaktion hat die folgenden Anforderungen:

- Eine IAM-Rolle, die die Ausführung des `s3:PutObject` Vorgangs übernehmen AWS IoT kann. Weitere Informationen finden Sie unter Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt.

  In der AWS IoT Konsole können Sie eine Rolle auswählen oder erstellen, um die Ausführung dieser Regelaktion AWS IoT zu ermöglichen.
- Wenn Sie einen AWS KMS Kunden verwenden, der AWS KMS key zur Verschlüsselung ruhender Daten in Amazon S3 verwaltet wird, muss der Service über die Erlaubnis verfügen, den im Namen des AWS KMS key Anrufers zu verwenden. Weitere Informationen finden Sie unter AWS Verwaltet AWS KMS keys und vom Kunden verwaltet AWS KMS keys im Amazon Simple Storage Service Developer Guide.

Parameters

Wenn Sie eine AWS IoT Regel mit dieser Aktion erstellen, müssen Sie die folgenden Informationen angeben:

`bucket`

Der Amazon S3-Bucket, in den die Daten geschrieben werden

Unterstützt Ersatzvorlagen: API und nur AWS CLI

`cannedacl`

(Optional) Die vordefinierte Amazon S3-ACL, der den Zugriff auf das Objekt steuert, das vom Objektschlüssel identifiziert wurde. Weitere Informationen, einschließlich der zulässigen Werte, finden Sie unter Vordefinierter ACL.

Unterstützt Ersatzvorlagen: Nein

`key`

Der Pfad zur Datei, in die die Daten geschrieben werden.

Nehmen wir ein Beispiel, bei dem dieser Parameter `${topic()}/${timestamp()}` ist und die Regel eine Nachricht mit dem Thema `some/topic` empfängt. Wenn der aktuelle Zeitstempel `1460685389` ist, dann schreibt diese Aktion die Daten in eine Datei namens `1460685389` im Ordner `some/topic` des S3-Buckets.

> **Note**
>
> Wenn Sie einen statischen Schlüssel verwenden, wird bei jedem Aufruf der Regel eine einzelne Datei AWS IoT überschrieben. Wir empfehlen Ihnen, den Zeitstempel der Nachricht oder einen anderen eindeutigen Nachrichtenbezeichner zu verwenden, damit für jede empfangene Nachricht eine neue Datei in Amazon S3 gespeichert wird.

Unterstützt Ersatzvorlagen: Ja

`roleArn`

Die IAM-Rolle, die den Zugriff auf den Amazon S3-Bucket ermöglicht. Weitere Informationen finden Sie unter Voraussetzungen.

Unterstützt Ersatzvorlagen: Nein

Beispiele

Das folgende JSON-Beispiel definiert eine S3-Aktion in einer AWS IoT Regel.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "s3": {
                    "bucketName": "amzn-s3-demo-bucket",
                    "cannedacl": "public-read",
                    "key": "${topic()}/${timestamp()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_s3"
                }
            }
        ]
    }
}
```

Weitere Informationen finden Sie auch unter

- Was ist Amazon S3? im Entwicklerhandbuch für Amazon Simple Storage Service

SNS

Die Aktion SNS (`sns`) sendet die Daten aus einer MQTT-Nachricht als Amazon Simple Notification Service (Amazon SNS) Push-Nachricht.

Sie können einem Tutorial folgen, das Ihnen veranschaulicht, wie Sie eine Regel mit einer SNS-Aktion erstellen und testen. Weitere Informationen finden Sie unter Tutorial: Senden einer Amazon SNS-Benachrichtigung.

> **Note**
>
> Die SNS-Aktion unterstützt keine Amazon SNS FIFO () First-In-First-Out -Themen. Da die Regeln-Engine ein vollständig verteilter Dienst ist, gibt es keine Garantie für die Reihenfolge der Nachrichten, wenn die SNS-Aktion aufgerufen wird.

Voraussetzungen

Diese Regelaktion hat die folgenden Anforderungen:

- Eine IAM-Rolle, die die Ausführung des AWS IoT Vorgangs übernehmen kann. `sns:Publish` Weitere Informationen finden Sie unter Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt.

  In der AWS IoT Konsole können Sie eine Rolle auswählen oder erstellen, um die Ausführung dieser Regelaktion AWS IoT zu ermöglichen.
- Wenn Sie einen vom AWS KMS Kunden verwalteten Dienst AWS KMS key zur Verschlüsselung ruhender Daten in Amazon SNS verwenden, muss der Service über die Erlaubnis verfügen, den im Namen des Anrufers AWS KMS key zu verwenden. Weitere Informationen dazu erhalten Sie unter Schlüsselverwaltung im Entwicklerhandbuch für Amazon Simple Notification Service.

Parameters

Wenn Sie mit dieser Aktion eine AWS IoT Regel erstellen, müssen Sie die folgenden Informationen angeben:

`targetArn`

Das SNS-Thema oder das individuelle Gerät, an das die Pushbenachrichtigung gesendet wird

Unterstützt Ersatzvorlagen: API und nur AWS CLI

`messageFormat`

(Optional) Das Nachrichtenformat. Amazon SNS verwendet diese Einstellung, um zu bestimmen, ob die Nutzlast analysiert werden soll und ob die relevanten plattformspezifischen Teile der Nutzlast extrahiert werden. Zulässige Werte: `JSON`, `RAW`. Standardeinstellung: `RAW`.

Unterstützt Ersatzvorlagen: Nein

`roleArn`

Die IAM-Rolle, die den Zugriff auf SNS ermöglicht. Weitere Informationen finden Sie unter Voraussetzungen.

Unterstützt Ersatzvorlagen: Nein

Beispiele

Das folgende JSON-Beispiel definiert eine SNS-Aktion in einer AWS IoT Regel.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "sns": {
                    "targetArn": "arn:aws:sns:us-east-2:123456789012:my_sns_topic",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sns"
                }
            }
```

```
        ]
    }
}
```

Das folgende JSON-Beispiel definiert eine SNS-Aktion mit Ersatzvorlagen in einer Regel. AWS IoT

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "sns": {
                    "targetArn": "arn:aws:sns:us-east-1:123456789012:${topic()}",
                    "messageFormat": "JSON",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sns"
                }
            }
        ]
    }
}
```

Weitere Informationen finden Sie auch unter

- Was ist Amazon Simple Notification Service? im Amazon Simple Notification Service-Entwicklerhandbuch
- Tutorial: Senden einer Amazon SNS-Benachrichtigung

SQS

Die SQS (`sqs`) Aktion sendet Daten von einer MQTT-Nachricht an eine Amazon Simple Queue Service (Amazon SQS) Warteschlange.

> **Note**
>
> Die SQS-Aktion unterstützt keine Amazon SQS FIFO () First-In-First-Out -Warteschlangen. Da die Regeln-Engine ein vollständig verteilter Service ist, gibt es keine Garantie für die Nachrichtenreihenfolge, wenn die SQS-Aktion ausgelöst wird.

Voraussetzungen

Diese Regelaktion hat die folgenden Anforderungen:

- Eine IAM-Rolle, die die Ausführung des Vorgangs übernehmen AWS IoT kann. `sqs:SendMessage` Weitere Informationen finden Sie unter Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt.

  In der AWS IoT Konsole können Sie eine Rolle auswählen oder erstellen, um die Ausführung dieser Regelaktion AWS IoT zu ermöglichen.
- Wenn Sie einen AWS KMS Kunden einsetzen, der es geschafft hat AWS KMS key , ruhende Daten in Amazon SQS zu verschlüsseln, muss der Service über die Erlaubnis verfügen, den im Namen des AWS KMS key Anrufers zu verwenden. Weitere Informationen finden Sie unter Schlüsselverwaltung im Amazon Simple Queue Service Entwicklerhandbuch.

Parameters

Wenn Sie mit dieser Aktion eine AWS IoT Regel erstellen, müssen Sie die folgenden Informationen angeben:

`queueUrl`

Die URL der Amazon SQS-Warteschlange, in die die Daten geschrieben werden Die Region in dieser URL muss nicht mit Ihrer AWS-Region AWS IoT Regel identisch sein.

> **Note**
>
> Bei AWS-Regionen Verwendung der SQS-Regelaktion können zusätzliche Gebühren für die grenzüberschreitende Datenübertragung anfallen. Weitere Informationen finden Sie unter Amazon SQS SQS-Preise.

Unterstützt Ersatzvorlagen: API und nur AWS CLI

`useBase64`

Setzen Sie diesen Parameter auf `true`, um die Regelaktion so zu konfigurieren, dass sie die Nachrichtendaten Base64-kodiert, bevor sie in die Amazon SQS-Warteschlange geschrieben werden. Standardeinstellung: `false`.

Unterstützt Ersatzvorlagen: Nein

`roleArn`

Die IAM-Rolle, die den Zugriff auf die Amazon SQS-Warteschlange ermöglicht. Weitere Informationen finden Sie unter Voraussetzungen.

Unterstützt Ersatzvorlagen: Nein

Beispiele

Das folgende JSON-Beispiel definiert eine SQS-Aktion in einer AWS IoT Regel.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "sqs": {
                    "queueUrl": "https://sqs.us-east-2.amazonaws.com/123456789012/
my_sqs_queue",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sqs"
                }
            }
        ]
    }
}
```

Das folgende JSON-Beispiel definiert eine SQS-Aktion mit Ersatzvorlagen in einer Regel. AWS IoT

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "sqs": {
                    "queueUrl": "https://sqs.us-east-2.amazonaws.com/123456789012/
${topic()}",
```

```
                    "useBase64": true,
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sqs"
                }
            }
        ]
    }
}
```

Weitere Informationen finden Sie auch unter

- Was ist Amazon Simple Queue Service? im Amazon Simple Queue Service-Entwicklerhandbuch?

Step Functions

Die Aktion Step Functions (`stepFunctions`) startet eine AWS Step Functions Zustandsmaschine.

Voraussetzungen

Diese Regelaktion hat die folgenden Anforderungen:

- Eine IAM-Rolle, die die Ausführung des `states:StartExecution` Vorgangs übernehmen AWS IoT kann. Weitere Informationen finden Sie unter Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt.

  In der AWS IoT Konsole können Sie eine Rolle auswählen oder erstellen, um die Ausführung dieser Regelaktion AWS IoT zu ermöglichen.

Parameters

Wenn Sie eine AWS IoT Regel mit dieser Aktion erstellen, müssen Sie die folgenden Informationen angeben:

`stateMachineName`

Der Name des zu startenden Step Functions-Zustandsautomaten.

Unterstützt Ersatzvorlagen: API und nur AWS CLI

`executionNamePrefix`

(Optional) Der Name, der der Ausführung des Zustandsautomaten gegeben wird, besteht aus diesem Präfix, gefolgt von einer UUID. Step Functions erstellt einen eindeutigen Namen für jede Zustandsautomaten-Ausführung, sofern keiner angegeben wird.

Unterstützt Ersatzvorlagen: Ja

`roleArn`

Der ARN der Rolle, die die AWS IoT Erlaubnis zum Starten der Zustandsmaschine erteilt. Weitere Informationen finden Sie unter Voraussetzungen.

Unterstützt Ersatzvorlagen: Nein

Beispiele

Das folgende JSON-Beispiel definiert eine Step Functions Functions-Aktion in einer AWS IoT Regel.

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "stepFunctions": {
                    "stateMachineName": "myStateMachine",
                    "executionNamePrefix": "myExecution",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_step_functions"
                }
            }
        ]
    }
}
```

Weitere Informationen finden Sie auch unter

- Was ist AWS Step Functions? im AWS Step Functions Developer Guide

Timestream

Die Timestream-Regelaktion schreibt Attribute (Kennzahlen) aus einer MQTT-Nachricht in eine Amazon Timestream-Tabelle. Für weitere Informationen über Amazon Timestream, siehe Was ist Amazon Timestream?

> **Note**
>
> Amazon Timestream ist nicht in allen AWS-Region s verfügbar. Wenn Amazon Timestream in Ihrer Region nicht verfügbar ist, wird es nicht in der Liste der Regelaktionen angezeigt.

Die Attribute, die diese Regel in der Timestream-Datenbank speichert, sind diejenigen, die sich aus der Abfrageanweisung der Regel ergeben. Der Wert jedes Attributs im Ergebnis der Abfrageanweisung wird analysiert, um auf seinen Datentyp zu schließen (wie bei einer the section called “DynamoDBv2” Aktion). Der Wert jedes Attributs wird in einen eigenen Datensatz in der Timestream-Tabelle geschrieben. Um den Datentyp eines Attributs anzugeben oder zu ändern, verwenden Sie die `cast()` Funktion in der Abfrageanweisung. Weitere Informationen zum Inhalt der einzelnen Timestream-Datensätze finden Sie unter the section called “Timestream-Datensatzinhalte”.

> **Note**
>
> Mit SQL V2 (23.03.2016) werden numerische Werte, die ganze Zahlen sind, wie z. B. `10.0`, in ihre Integer-Darstellung (`10`) umgewandelt. Sie explizit in einen `Decimal` Wert umzuwandeln, z. B. mithilfe der Funktion cast (), verhindert dieses Verhalten nicht — das Ergebnis ist immer noch ein `Integer` Wert. Dies kann zu Typkonflikten führen, die verhindern, dass Daten in der Timestream-Datenbank aufgezeichnet werden. Um ganzzahlige numerische Werte als `Decimal` Werte zu verarbeiten, verwenden Sie SQL V1 (2015-10-08) für die Regelabfrageanweisung.

> **Note**
>
> Die maximale Anzahl von Werten, die eine Timestream-Regelaktion in eine Amazon Timestream-Tabelle schreiben kann, ist 100. Weitere Informationen finden Sie unter Amazon Timestream Quota's Referenz.

Voraussetzungen

Diese Regelaktion hat die folgenden Anforderungen:

- Eine IAM-Rolle, die die Ausführung der `timestream:DescribeEndpoints` AND-Operationen übernehmen AWS IoT kann. `timestream:WriteRecords` Weitere Informationen finden Sie unter Gewährung eines AWS IoT regeln Sie den Zugriff, den es benötigt.

  In der AWS IoT Konsole können Sie eine Rolle auswählen, aktualisieren oder erstellen, um die Ausführung dieser Regelaktion AWS IoT zu ermöglichen.
- Wenn Sie einen Kunden verwenden AWS KMS , um Daten im Ruhezustand in Timestream zu verschlüsseln, muss der Dienst die Erlaubnis haben, das im Namen des AWS KMS key Anrufers zu verwenden. Weitere Informationen finden Sie unter So verwenden AWS Dienste KMS. AWS

Parameters

Wenn Sie mit dieser Aktion eine AWS IoT Regel erstellen, müssen Sie die folgenden Informationen angeben:

`databaseName`

Der Name einer Amazon-Timestream-Datenbank, die über die Tabelle verfügt, in der die von dieser Aktion erstellten Datensätze empfangen werden sollen. Siehe auch **`tableName`**.

Unterstützt Ersatzvorlagen: API und nur AWS CLI

`dimensions`

Metadatenattribute der Zeitreihen, die in jedem Messdatensatz geschrieben werden. Beispielsweise sind der Name und die Availability Zone einer EC2-Instance oder der Name des Herstellers einer Windkraftanlage Dimensionen.

`name`

Der Name der Metadatendimension. Dies ist der Name der Spalte im Datensatz der Datenbanktabelle.

Dimensionen können nicht benannt werden: `measure_name`, `measure_value`, oder `time`. Diese Namen sind vorbehalten. Dimensionsnamen dürfen nicht mit `ts_` oder `measure_value` beginnen und dürfen keinen Doppelpunkt (`:`) enthalten.

Unterstützt Ersatzvorlagen: Nein

`value`

Der Wert, der in diese Spalte des Datenbankdatensatzes geschrieben werden soll.

Unterstützt Ersatzvorlagen: Ja

`roleArn`

Der Amazon-Ressourcenname (ARN) der Rolle, die AWS IoT die Berechtigung zum Schreiben in die Timestream-Datenbanktabelle gewährt. Weitere Informationen finden Sie unter Voraussetzungen.

Unterstützt Ersatzvorlagen: Nein

`tableName`

Der Name der Datenbanktabelle, in die die Messdatensätze geschrieben werden sollen. Siehe auch **`databaseName`**.

Unterstützt Ersatzvorlagen: API und nur AWS CLI

`timestamp`

Der Wert, der für den Zeitstempel des Eintrags verwendet werden soll. Wenn das Feld leer ist, wird die Zeit verwendet, zu der der Eintrag verarbeitet wurde.

`unit`

Die Genauigkeit des Zeitstempelwerts, die sich aus dem unter `value` beschriebenen Ausdruck ergibt.

Zulässige Werte: `SECONDS` | `MILLISECONDS` | `MICROSECONDS` | `NANOSECONDS`. Der Standardwert ist `MILLISECONDS`.

`value`

Ein Ausdruck, der einen Wert für lange Epochenzeit zurückgibt.

Sie können die the section called “time_to_epoch (Zeichenfolge, Zeichenfolge)” Funktion verwenden, um einen gültigen Zeitstempel aus einem Datums- oder Uhrzeitwert zu erstellen, der in der Nachrichtennutzlast übergeben wurde.

Timestream-Datensatzinhalte

Die durch diese Aktion in die Amazon Timestream-Tabelle geschriebenen Daten umfassen einen Zeitstempel, Metadaten aus der Timestream-Regelaktion und das Ergebnis der Abfrageanweisung der Regel.

Für jedes Attribut (Kennzahl) im Ergebnis der Abfrageanweisung schreibt diese Regelaktion einen Datensatz mit diesen Spalten in die angegebene Timestream-Tabelle.

Name der Spalte	Attribut Typ	Wert	Kommentare
dimension-name	DIMENSION	Der im Aktionseintrag für die Timestream-Regel angegebene Wert.	Jede im Regelaktionseintrag angegebene Dimension erstellt eine Spalte in der Timestream-Datenbank mit dem Namen der Dimension.
measure_name	MEASURE_NAME	Der Name des Attributs	Der Name des Attributs im Ergebnis der Abfrageanweisung, dessen Wert in der `measure_value::` *data-type* Spalte angegeben ist.
Messwert:: *data-type*	MESS_WERT	Der Wert des Attributs im Ergebnis der Abfrageanweisung. Der Name des Attributs steht in der `measure_name` Spalte.	Der Wert wird interpretiert* und als die am besten geeignete Übereinstimmung von: `bigint`, `boolean`, `double`, oder `varchar` gewertet. Amazon Timestream erstellt für jeden Datentyp eine

Name der Spalte	Attribut Typ	Wert	Kommentare
			separate Spalte. Der Wert in der Nachricht kann mithilfe der `cast()` Funktion in der Abfragean weisung der Regel in einen anderen Datentyp umgewande lt werden.
time	TIMESTAMP (ZEITSTEMPEL)	Das Datum und die Uhrzeit des Datensatz es in der Datenbank.	Dieser Wert wird von der Regel-Engine oder der `timestamp` Eigenschaft zugewiesen, sofern sie definiert ist.

* Der aus der Nachrichtennutzlast gelesene Attributwert wird wie folgt interpretiert. Eine Veranschaulichung der einzelnen Fälle finden Sie in der the section called “Beispiele”.

- Ein Wert von `true` oder `false` ohne Anführungszeichen wird als `boolean` Typ interpretiert.
- Eine dezimale Zahl wird als `double` Typ interpretiert.
- Ein numerischer Wert ohne Dezimalpunkt wird als `bigint` Typ interpretiert.
- Eine Zeichenfolge in Anführungszeichen wird als `varchar` Typ interpretiert.
- Objekte und Array-Werte werden in JSON-Zeichenketten konvertiert und als `varchar` Typ gespeichert.

Beispiele

Das folgende JSON-Beispiel definiert eine Timestream-Regelaktion mit einer Ersatzvorlage in einer Regel. AWS IoT

```
{
  "topicRulePayload": {
```

```
    "sql": "SELECT * FROM 'iot/topic'",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "timestream": {
          "roleArn": "arn:aws:iam::123456789012:role/aws_iot_timestream",
          "tableName": "devices_metrics",
          "dimensions": [
            {
              "name": "device_id",
              "value": "${clientId()}"
            },
            {
              "name": "device_firmware_sku",
              "value": "My Static Metadata"
            }
          ],
          "databaseName": "record_devices"
        }
      }
    ]
  }
}
```

Die Verwendung der im vorherigen Beispiel definierten Timestream-Themenregelaktion mit der folgenden Nachrichtennutzlast führt zu den Amazon Timestream-Datensätzen, die in der folgenden Tabelle aufgeführt sind.

```
{
  "boolean_value": true,
  "integer_value": 123456789012,
  "double_value": 123.456789012,
  "string_value": "String value",
  "boolean_value_as_string": "true",
  "integer_value_as_string": "123456789012",
  "double_value_as_string": "123.456789012",
  "array_of_integers": [23,36,56,72],
  "array of strings": ["red", "green","blue"],
  "complex_value": {
    "simple_element": 42,
    "array_of_integers": [23,36,56,72],
    "array of strings": ["red", "green","blue"]
```

```
  }
}
```

In der folgenden Tabelle werden die Datenbankspalten und Datensätze angezeigt, die durch die Verwendung der angegebenen Themenregelaktion zur Verarbeitung der vorherigen Nachrichtennutzlast erstellt wurden. Die `device_firmware_sku` und `device_id` Spalten sind die DIMENSIONEN, die in der Themenregelaktion definiert sind. Die Timestream-Themenregelaktion erstellt die `time` Spalte und die `measure_name` und `measure_value::*` Spalten, die sie mit den Werten aus dem Ergebnis der Abfrageanweisung der Themenregelaktion füllt.

device_firmware_sku	Gerät_ID	measure_name	Messwert::bigint	Messwert::varchar	Messwert::doppelt	Messwert::boolean	time
Meine statischen Metadaten	iotconsole-159EXAMPLE78-0	komplexer_Wert	-	{"simple_element":42,"array_of_integers":[23,36,56,72],"array of strings":["red","green","blue"]}	-	-	2020-08-26 22:42:16.423000000
Meine statischen Metadaten	IoT-Konsole-159EXAMPLE78-0	Integer_Wert_als_Zeichenfolge	-	123456789012	-	-	2020-08-26 22:42:16.423000000
Meine statischen Metadaten	IoT-Konsole-159EXAMPLE78-0	boolescher_Wert	-	-	-	TRUE	2020-08-26 22:42:16.423000000

device_firmware_sku	Gerät_ID	measure_name	Messwert::bigint	Messwert::varchar	Messwert::doppelt	Messwert::boolean	time
Meine statischen Metadaten	IoT-Konsole-159 EXAMPLE78-0	Integer_Wert	123456789012	-	-	-	2020-08-26 22:42:16.423000000
Meine statischen Metadaten	IoT-Konsole-159 EXAMPLE78-0	Zeichenfolge_Wert	-	Zeichenfolge_Wert	-	-	2020-08-26 22:42:16.423000000
Meine statischen Metadaten	IoT-Konsole-159 EXAMPLE78-0	Array_von_ganzen_Zahlen	-	[23,36,56,72]	-	-	2020-08-26 22:42:16.423000000
Meine statischen Metadaten	IoT-Konsole-159 EXAMPLE78-0	Zeichenfolgen-Array	-	["red","green","blue"]	-	-	2020-08-26 22:42:16.423000000
Meine statischen Metadaten	IoT-Konsole-159 EXAMPLE78-0	boolescher_Wert_als_Zeichenfolge	-	TRUE	-	-	2020-08-26 22:42:16.423000000

device_firmware_sku	Gerät_ID	measure_name	Messwert::bigint	Messwert::varchar	Messwert::doppelt	Messwert::boolean	time
Meine statischen Metadaten	IoT-Konsole-159 EXAMPLE78-0	doppelter_Wert	-	-	123,456789012	-	2020-08-26 22:42:16.423000000
Meine statischen Metadaten	IoT-Konsole-159 EXAMPLE78-0	doppelter Wert_als_Zeichenfolge	-	123,45679	-	-	2020-08-26 22:42:16.423000000

Fehlerbehebung bei einer Regel

Wenn Sie ein Problem mit Ihren Regeln haben, empfehlen wir Ihnen, Logs zu aktivieren. CloudWatch Sie können Ihre Protokolle analysieren, um festzustellen, ob es sich um ein Berechtigungsproblem handelt oder ob z. B. eine WHERE-Klausel-Bedingung nicht zutraf. Weitere Informationen finden Sie unter CloudWatch Protokolle einrichten.

Zugreifen auf kontoübergreifende Ressourcen mit AWS IoT Regeln

Sie können AWS IoT Regeln für den kontoübergreifenden Zugriff konfigurieren, sodass Daten, die zu MQTT-Themen eines Kontos aufgenommen wurden, an die AWS Dienste wie Amazon SQS und Lambda eines anderen Kontos weitergeleitet werden können. Im Folgenden wird erklärt, wie AWS IoT Regeln für die kontoübergreifende Datenaufnahme eingerichtet werden, und zwar von einem MQTT-Thema in einem Konto bis hin zu einem Ziel in einem anderen Konto.

Cross-account Regeln können mithilfe von ressourcenbasierten Berechtigungen für die Zielressource konfiguriert werden. Daher können nur Ziele, die ressourcenbasierte Berechtigungen unterstützen, für den kontenübergreifenden Zugriff mit Regeln aktiviert werden. AWS IoT Zu den unterstützten Zielen gehören Amazon SQS, Amazon SNS, Amazon S3 und AWS Lambda.

> **Note**
>
> Für die unterstützten Ziele, mit Ausnahme von Amazon SQS, müssen Sie die Regel in derselben Weise AWS-Region wie die Ressource eines anderen Dienstes definieren, damit die Regelaktion mit dieser Ressource interagieren kann. Weitere Informationen zu AWS IoT Regelaktionen finden Sie unter AWS IoT Regelaktionen. Weitere Informationen zur SQS-Aktion einer Regel finden Sie unter???.

Voraussetzungen

- Vertrautheit mit AWS IoT Regeln
- Ein Verständnis von IAM-Benutzern, -Rollen und ressourcenbasierten Berechtigungen
- Nach der AWS CLI Installation

Cross-account Einrichtung für Amazon SQS

Szenario: Konto A sendet Daten aus einer MQTT-Nachricht an die Amazon SQS-Warteschlange von Konto B.

AWS-Konto	Konto bezeichnet als	Description
1111-1111-1111	Konto A	Regelaktion: `sqs:SendMessage`
2222-2222-2222	Konto B	Amazon-SQS-Warteschlange • ARN: *arn:aws:sqs:region:2222-2222-2222:ExampleQueue* • URL: *https://sqs.region.amazonaws.com/2222-2222-2222/ExampleQueue*

> **Note**
>
> Ihre Amazon SQS SQS-Zielwarteschlange muss sich nicht in derselben Warteschlange AWS-Region wie Ihre AWS IoT Regel befinden. Weitere Informationen zur SQS-Aktion der Regel finden Sie unter. ???

Erledigen der Aufgaben von Konto A

> **Hinweis**
>
> Um die folgenden Befehle auszuführen, muss Ihr IAM-Benutzer über die Berechtigung `iot:CreateTopicRule` mit dem Amazon Resource Name (ARN) der Regel als Ressource und über die Berechtigung `iam:PassRole` Aktion mit einer Ressource als ARN der Rolle verfügen.

1. Konfigurieren Sie AWS CLI unter Verwendung des IAM-Benutzers von Konto A.
2. Erstellen Sie eine IAM-Rolle, die der AWS IoT Regel-Engine vertraut, und fügen Sie eine Richtlinie hinzu, die den Zugriff auf die Amazon SQS SQS-Warteschlange von Konto B ermöglicht. Beispielbefehle und Richtliniendokumente finden Sie unter Gewährung AWS IoT des erforderlichen Zugriffs.
3. Um eine Regel zu erstellen, die an ein Thema angehängt ist, führen Sie den Befehl create-topic-rule aus.

   ```
   aws iot create-topic-rule --rule-name myRule --topic-rule-payload file://./my-rule.json
   ```

   Im Folgenden finden Sie ein Beispiel für eine Nutzlastdatei mit einer Regel, die alle an das `iot/test` Thema gesendeten Nachrichten in die angegebene Amazon SQS-Warteschlange einfügt. Die SQL-Anweisung filtert die Nachrichten und der Rollen-ARN gewährt AWS IoT die Berechtigung, die Nachricht zur Amazon SQS-Warteschlange hinzuzufügen.

   ```
   {
    "sql": "SELECT * FROM 'iot/test'",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
   ```

```
  {
   "sqs": {
    "queueUrl": "https://sqs.region.amazonaws.com/2222-2222-2222/ExampleQueue",
    "roleArn": "arn:aws:iam::1111-1111-1111:role/my-iot-role",
    "useBase64": false
   }
  }
 ]
}
```

Weitere Informationen zur Definition einer Amazon SQS-Aktion in einer AWS IoT Regel finden Sie unter AWS IoT Regelaktionen — Amazon SQS.

Erledigen der Aufgaben von Konto B

1. Konfigurieren Sie AWS CLI mithilfe des IAM-Benutzers von Konto B.
2. Um Konto A Berechtigungen für die Amazon SQS-Warteschlangenressource zu erteilen, führen Sie den Befehl add-permission aus.

```
aws sqs add-permission --queue-url https://sqs.region.amazonaws.com/2222-2222-2222/
ExampleQueue --label SendMessagesToMyQueue --aws-account-ids 1111-1111-1111 --
actions SendMessage
```

Cross-account Einrichtung für Amazon SNS

Szenario: Konto A sendet Daten von einer MQTT-Nachricht an ein Amazon SNS Thema von Konto B.

AWS-Konto	Konto bezeichnet als	Description
1111-1111-1111	Konto A	Regelaktion: `sns:Publish`
2222-2222-2222	Konto B	ARN des Amazon-SNS-Themas: *arn:aws:sns:region:2222-2222-2222:ExampleTopic*

Erledigen der Aufgaben von Konto A

> **Hinweise**
>
> Um die folgenden Befehle auszuführen, sollte Ihr IAM-Benutzer über Berechtigungen für die `iot:CreateTopicRule` Regel ARN als Ressource und über Berechtigungen für die `iam:PassRole` Aktion mit einer Ressource als Rolle ARN verfügen.

1. Konfigurieren Sie AWS CLI unter Verwendung des IAM-Benutzers von Konto A.
2. Erstellen Sie eine IAM-Rolle, die der AWS IoT Regel-Engine vertraut, und fügen Sie eine Richtlinie hinzu, die den Zugriff auf das Amazon SNS SNS-Thema von Konto B ermöglicht. Befehle und Richtliniendokumente finden Sie beispielsweise unter Gewährung AWS IoT des erforderlichen Zugriffs.
3. Um eine Regel zu erstellen, die an ein Thema angehängt ist, führen Sie den Befehl create-topic-rule aus.

   ```
   aws iot create-topic-rule --rule-name myRule --topic-rule-payload file://./my-rule.json
   ```

   Im Folgenden finden Sie ein Beispiel für eine Nutzlastdatei mit einer Regel, die alle an das `iot/test` Thema gesendeten Nachrichten in die angegebene -Amazon SNS Thema einfügt. Die SQL-Anweisung filtert die Nachrichten, und die Rollen-ARN gewährt AWS IoT Berechtigungen zum Senden der Nachricht an das Amazon SNS Thema.

   ```
   {
    "sql": "SELECT * FROM 'iot/test'",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
     {
      "sns": {
       "targetArn": "arn:aws:sns:region:2222-2222-2222:ExampleTopic",
       "roleArn": "arn:aws:iam::1111-1111-1111:role/my-iot-role"
      }
     }
    ]
   }
   ```

Weitere Informationen zur Definition einer Amazon SNS-Aktion in einer AWS IoT Regel finden Sie unter AWS IoT Regelaktionen — Amazon SNS.

Erledigen der Aufgaben von Konto B

1. Konfigurieren Sie AWS CLI mithilfe des IAM-Benutzers von Konto B.
2. Um Konto A die Erlaubnis für die Amazon SNS-Themenressource zu erteilen, führen Sie den Befehl add-permission aus.

```
aws sns add-permission --topic-arn arn:aws:sns:region:2222-2222-2222:ExampleTopic
 --label Publish-Permission --aws-account-id 1111-1111-1111 --action-name Publish
```

Cross-account Einrichtung für Amazon S3

Szenario: Konto A sendet Daten von einer MQTT-Nachricht an einen Amazon-S3-Bucket von Konto B.

AWS-Konto	Konto bezeichnet als	Description
1111-1111-1111	Konto A	Regelaktion: `s3:PutObject`
2222-2222-2222	Konto B	Amazon S3-Bucket ARN: *arn:aws:s3:::amzn-s3-demo-bucket*

Erledigen der Aufgaben von Konto A

> **Hinweis**
>
> Um die folgenden Befehle auszuführen, sollte Ihr IAM-Benutzer über Berechtigungen für `iot:CreateTopicRule` mit der Regel ARN als Ressource und über Berechtigungen für `iam:PassRole` Aktionen mit einer Ressource als Rolle ARN verfügen.

1. Konfigurieren Sie AWS CLI unter Verwendung des IAM-Benutzers von Konto A.

2. Erstellen Sie eine IAM-Rolle, die der AWS IoT Regel-Engine vertraut, und fügen Sie eine Richtlinie hinzu, die den Zugriff auf den Amazon S3 S3-Bucket von Konto B ermöglicht. Befehle und Richtliniendokumente finden Sie beispielsweise unter Gewährung AWS IoT des erforderlichen Zugriffs.
3. Um eine Regel zu erstellen, die an Ihren Ziel-S3-Bucket angehängt ist, führen Sie den Befehl create-topic-rule aus.

   ```
   aws iot create-topic-rule --rule-name my-rule --topic-rule-payload file://./my-rule.json
   ```

   Im Folgenden finden Sie ein Beispiel für eine Nutzlastdatei mit einer Regel, die alle an das `iot/test` Thema gesendeten Nachrichten in den angegebenen Amazon S3-Bucket einfügt. Die SQL-Anweisung filtert die Nachrichten und der Rollen-ARN gewährt AWS IoT die Berechtigung zum Hinzufügen der Nachricht zum Amazon S3-Bucket.

   ```
   {
    "sql": "SELECT * FROM 'iot/test'",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
     {
      "s3": {
       "bucketName": "amzn-s3-demo-bucket",
       "key": "${topic()}/${timestamp()}",
       "roleArn": "arn:aws:iam::1111-1111-1111:role/my-iot-role"
      }
     }
    ]
   }
   ```

   Weitere Informationen zur Definition einer Amazon S3-Aktion in einer AWS IoT Regel finden Sie unter AWS IoT Regelaktionen — Amazon S3.

Erledigen der Aufgaben von Konto B

1. Konfigurieren Sie AWS CLI mithilfe des IAM-Benutzers von Konto B.
2. Erstellen Sie eine Bucket-Richtlinie, die dem Prinzipal von Konto A vertraut.

Im Folgenden finden Sie ein Beispiel für eine Payload-Datei, die eine Bucket-Richtlinie definiert, die dem Prinzipal eines anderen Kontos vertraut.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AddCannedAcl",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "arn:aws:iam::123456789012:root"
                ]
        },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/*"
        }
    ]
}
```

Weitere Informationen finden Sie unter Beispiele für Bucket-Richtlinien.

3. Um die Bucket-Richtlinie an den angegebenen Bucket anzuhängen, führen Sie den Befehl put-bucket-policy aus.

```
aws s3api put-bucket-policy --bucket amzn-s3-demo-bucket --policy file://./amzn-s3-demo-bucket-policy.json
```

4. Damit der kontoübergreifende Zugriff funktioniert, stellen Sie sicher, dass Sie die richtigen Einstellungen für Blockieren des gesamten öffentlichen Zugangs ausgewählt haben. Weitere Informationen finden Sie unter Bewährte Methoden für die Sicherheit in Amazon S3.

Cross-account einrichten für AWS Lambda

Szenario: Konto A ruft eine AWS Lambda Funktion von Konto B auf und übergibt eine MQTT-Nachricht.

AWS-Konto	Konto bezeichnet als	Description
1111-1111-1111	Konto A	Regelaktion: `lambda:InvokeFunction`
2222-2222-2222	Konto B	ARN der Lambda-Funktion: *arn:aws:lambda:region:2222-2222-2222:function:example-function*

Erledigen der Aufgaben von Konto A

> **Hinweise**
>
> Um die folgenden Befehle ausführen zu können, sollte Ihr IAM-Benutzer über Berechtigungen für `iot:CreateTopicRule` mit der Regel ARN als Ressource und über Berechtigungen für `iam:PassRole` Aktionen mit der Ressource als Rolle ARN verfügen.

1. Konfigurieren Sie AWS CLI unter Verwendung des IAM-Benutzers von Konto A.
2. Führen Sie den Befehl create-topic-rule aus, um eine Regel zu erstellen, die den kontoübergreifenden Zugriff auf die Lambda-Funktion von Konto B definiert.

   ```
   aws iot create-topic-rule --rule-name my-rule --topic-rule-payload file://./my-rule.json
   ```

   Im Folgenden finden Sie ein Beispiel für eine Nutzlastdatei mit einer Regel, die alle an das `iot/test` Thema gesendeten Nachrichten in die angegebene Lambda-Funktion einfügt. Die SQL-Anweisung filtert die Nachrichten und die Rolle ARN erteilt AWS IoT die Berechtigung, die Daten an die Lambda-Funktion zu übergeben.

   ```
   {
    "sql": "SELECT * FROM 'iot/test'",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
     {
      "lambda": {
       "functionArn": "arn:aws:lambda:region:2222-2222-2222:function:example-function"
   ```

```
    }
   }
  ]
}
```

Weitere Informationen zum Definieren einer AWS Lambda Aktion in einer AWS IoT Regel finden Sie unter AWS IoT Regelaktionen — Lambda.

Erledigen der Aufgaben von Konto B

1. Konfigurieren Sie AWS CLI mithilfe des IAM-Benutzers von Konto B.
2. Führen Sie den Befehl add-permission von Lambda aus, um AWS IoT Regeln die Erlaubnis zur Aktivierung der Lambda-Funktion zu erteilen. Um den folgenden Befehl auszuführen, sollte Ihr IAM-Benutzer über die entsprechende Berechtigung für `lambda:AddPermission` Aktion verfügen.

```
aws lambda add-permission --function-name example-function --region us-east-1 --
principal iot.amazonaws.com --source-arn arn:aws:iot:region:1111-1111-1111:rule/
example-rule --source-account 1111-1111-1111 --statement-id "unique_id" --action
 "lambda:InvokeFunction"
```

Optionen:

--Prinzipal

Dieses Feld gibt AWS IoT (dargestellt durch`iot.amazonaws.com`) die Erlaubnis, die Lambda-Funktion aufzurufen.

--source-arn

Dieses Feld bestätigt, dass diese Lambda-Funktion nur `arn:aws:iot:region:1111-1111-1111:rule/example-rule` in AWS IoT Triggern und keine andere Regel in demselben oder einem anderen Konto diese Lambda-Funktion aktivieren kann.

--source-account

Dieses Feld bestätigt, dass diese Lambda-Funktion nur im Namen des `1111-1111-1111` Kontos AWS IoT aktiviert wird.

> **Hinweise**
>
> Wenn Sie in der Konsole Ihrer AWS Lambda Funktion unter Konfiguration die Fehlermeldung „Die Regel konnte nicht gefunden werden“ sehen, ignorieren Sie die Fehlermeldung und fahren Sie mit dem Testen der Verbindung fort.

Fehlerbehandlung (Fehleraktion)

Wenn eine Nachricht von einem Gerät AWS IoT empfangen wird, prüft die Regel-Engine, ob die Nachricht einer Regel entspricht. Ist dies der Fall, wird die Abfrageanweisung der Regel evaluiert und die Aktionen der Regel aktiviert. Dabei wird das Ergebnis der Abfrageanweisung übergeben.

Wenn bei der Aktivierung einer Aktion ein Problem auftritt, aktiviert die Regel-Engine eine Fehleraktion, falls eine solche für die Regel angegeben ist. Dies kann der Fall sein, wenn:

- Eine Regel verfügt nicht über die Berechtigung, auf einen Amazon S3-Bucket zuzugreifen.
- Ein Benutzerfehler führt dazu, dass der für DynamoDB bereitgestellte Durchsatz überschritten wird.

> **Note**
>
> Die in diesem Thema behandelte Fehlerbehandlung bezieht sich auf Regelaktionen. Um SQL-Probleme, einschließlich externer Funktionen, zu debuggen, können Sie die AWS IoT Protokollierung einrichten. Weitere Informationen finden Sie unter ???.

Nachrichtenformat für Fehleraktion

Eine einzelne Nachricht wird pro Regel und Nachricht generiert. Wenn beispielsweise zwei Regelaktionen in derselben Regel fehlschlagen, empfängt die Fehleraktion eine Nachricht, die beide Fehler enthält.

Die Fehlermeldung der Aktion kann in etwa wie im folgenden Beispiel aussehen.

```
{
  "ruleName": "TestAction",
  "topic": "testme/action",
```

```
  "cloudwatchTraceId": "7e146a2c-95b5-6caf-98b9-50e3969734c7",
  "clientId": "iotconsole-1511213971966-0",
  "base64OriginalPayload":
 "ewogICJtZXNzYWdlIjogIkhlbGxvIHZyb20gQVdTIElvVCBjb25zb2xlIgp9",
  "failures": [
    {
      "failedAction": "S3Action",
      "failedResource": "us-east-1-s3-verify-user",
      "errorMessage": "Failed to put S3 object. The error received was The
 specified bucket does not exist (Service: Amazon S3; Status Code: 404; Error
 Code: NoSuchBucket; Request ID: 9DF5416B9B47B9AF; S3 Extended Request ID:
 yMah1cwPhqTH267QLPhTKeVPKJB8BO5ndBHzOmWtxLTM6uAvwYYuqieAKyb6qRPTxP1tHXCoR4Y=).
 Message arrived on: error/action, Action: s3, Bucket: us-
east-1-s3-verify-user, Key: \"aaa\". Value of x-amz-id-2:
 yMah1cwPhqTH267QLPhTKeVPKJB8BO5ndBHzOmWtxLTM6uAvwYYuqieAKyb6qRPTxP1tHXCoR4Y="
    }
  ]
}
```

ruleName

Der Name der Regel, die die Fehleraktion ausgelöst hat.

Thema

Das Thema, in dem die ursprüngliche Nachricht empfangen wurde.

Cloudwatch TraceId

Eine eindeutige Identität, die sich auf den Fehler bezieht, meldet sich an CloudWatch.

clientId

Die Client-ID des Herausgebers der Nachricht.

Base64 OriginalPayload

Die Nutzdaten Base64-encoded der ursprünglichen Nachricht.

failures

failedAction

Der Name der Aktion, die nicht abgeschlossen werden konnte (z. B. „S3Action”).

failedResource

Der Name der Ressource (z. B. der Name eines S3-Buckets).

errorMessage

Die Beschreibung und Erläuterung des Fehlers.

Beispiel für Fehleraktion

Hier finden Sie ein Beispiel für eine Regel, der eine Fehleraktion hinzugefügt wurde. Die folgende Regel weist eine Aktion, die Nachrichtendaten in eine DynamoDB-Tabelle schreibt, und eine Fehleraktion auf, die Daten in einen Amazon S3-Bucket schreibt:

```
{
    "sql" : "SELECT * FROM ..."
    "actions" : [{
        "dynamoDB" : {
            "table" : "PoorlyConfiguredTable",
            "hashKeyField" : "AConstantString",
            "hashKeyValue" : "AHashKey"}}
    ],
    "errorAction" : {
        "s3" : {
            "roleArn": "arn:aws:iam::123456789012:role/aws_iot_s3",
            "bucketName" : "message-processing-errors",
            "key" : "${replace(topic(), '/', '-') + '-' + timestamp() + '-' +
 newuuid()}"
        }
    }
}
```

Sie können jede Funktion oder Ersetzungsvorlage in der SQL-Anweisung einer Fehleraktion verwenden, einschließlich der externen Funktionen: `aws_lambda()`, `get_dynamodb()`, `get_registry_data()`, `get_thing_shadow()get_secret()`, und `decode()`. Wenn eine Fehleraktion den Aufruf einer externen Funktion erfordert, kann der Aufruf der Fehleraktion zu einer zusätzlichen Rechnung für die externe Funktion führen.

Weitere Informationen zu Regeln und zur Angabe einer Fehleraktion finden Sie unter AWS IoT Regel erstellen.

Weitere Informationen CloudWatch zur Überwachung des Erfolgs oder Fehlers von Regeln finden Sie unterAWS IoT Metriken und Dimensionen.

Senken der Messaging-Kosten mit Basic Ingest

Sie können Basic Ingest verwenden, um Gerätedaten sicher an das von Ihnen AWS-Services unterstützte Gerät zu sendenAWS IoT Regelaktionen, ohne dass Messaging-Kosten anfallen. https://aws.amazon.com/iot-core/pricing/ Basic Ingest optimiert den Datenfluss, indem der publish/subscribe Message Broker aus dem Aufnahmepfad entfernt wird.

Basic Ingest kann Nachrichten von Ihren Geräten oder Anwendungen senden. Die Nachrichten verfügen über Themennamen, die für die ersten drei Ebenen mit `$aws/rules/rule_name` beginnen, wobei *`rule_name`* der Name der AWS IoT -Regel ist, die Sie aufrufen möchten.

Sie können eine vorhandene Regel mit Basic Ingest verwenden, indem Sie das Basic Ingest-Präfix (`$aws/rules/rule_name`) dem Nachrichtenthema hinzufügen, mit dem Sie die Regel normalerweise aufrufen. Wenn Sie beispielsweise eine Regel mit dem Namen `BuildingManager` haben, die bei Nachrichten mit Themen wie `Buildings/Building5/Floor2/Room201/Lights` (`"sql": "SELECT * FROM 'Buildings/#'"`) aufgerufen wird, können Sie dieselbe Regel mit Basic Ingest durch Senden einer Nachricht mit dem Thema `$aws/rules/BuildingManager/Buildings/Building5/Floor2/Room201/Lights` aufrufen.

> **Note**
>
> - Ihre Geräte und Regeln können keine reservierten Basic Ingest-Themen abonnieren. Beispielsweise werden die AWS IoT Device Defender `num-messages-received` Metrik-Metriken nicht ausgegeben, da das Abonnieren von Themen nicht unterstützt wird. Weitere Informationen finden Sie unter Reservierte Themen.
> - Wenn Sie einen publish/subscribe Broker benötigen, um Nachrichten an mehrere Abonnenten zu verteilen (z. B. um Nachrichten an andere Geräte und die Rules Engine zu senden), sollten Sie den AWS IoT Message Broker weiterhin für die Nachrichtenverteilung verwenden. Stellen Sie jedoch sicher, dass Sie Ihre Nachrichten zu anderen Themen als Basic Ingest-Themen veröffentlichen.

Verwenden von Basic Ingest

Stellen Sie vor der Verwendung von Basic Ingest sicher, dass Ihr Gerät oder Ihre Anwendung eine Richtlinie mit Veröffentlichungsberechtigungen auf `$aws/rules/*` nutzt. Oder Sie können die Berechtigungen für einzelne Regeln `$aws/rules/rule_name/*` in der Richtlinie angeben.

Andernfalls können Ihre Geräte und Anwendungen weiterhin ihre bestehenden Verbindungen mit AWS IoT Core nutzen.

Wenn die Nachricht die Regel-Engine erreicht, besteht kein Unterschied hinsichtlich der Implementierung oder Fehlerbehandlung zwischen von Basic Ingest und von Message Broker-Abonnements aufgerufenen Regeln.

Sie können Regeln für die Verwendung mit Basic Ingest erstellen. Beachten Sie Folgendes:

- Das erste Präfix eines Basic Ingest-Themas (`$aws/rules/`*`rule_name`*) ist für die topic(Decimal)-Funktion nicht verfügbar.
- Wenn Sie eine Regel definieren, die nur mit Basic Ingest aufgerufen wird, ist die `FROM`-Klausel im `sql`-Feld der `rule`-Definition optional. Sie ist weiterhin erforderlich, wenn die Regel auch von anderen Regeln aufgerufen wird, die über den Message Broker gesendet werden müssen (etwa weil diese anderen Nachrichten an mehrere Abonnenten verteilt werden müssen). Weitere Informationen finden Sie unter AWS IoT SQL-Referenz.
- Die ersten drei Ebenen des Basic Ingest-Themas (`$aws/rules/`*`rule_name`*) zählen nicht für die Längenbeschränkung von 8 Segmenten oder von insgesamt 256 Zeichen für ein Thema. Davon abgesehen gelten dieselben Einschränkungen, wie in den AWS IoT -Einschränkungen dokumentiert.
- Wenn eine Nachricht mit einem Basic Ingest-Thema eingeht, das eine inaktive Regel oder eine Regel angibt, die nicht existiert, wird in einem Amazon-Protokoll ein CloudWatch Fehlerprotokoll erstellt, das Ihnen beim Debuggen hilft. Weitere Informationen finden Sie unter Protokolleinträge zur Regel-Engine. Es wird eine `RuleNotFound`-Metrik angezeigt, und Sie können dafür Alarme erstellen. Weitere Informationen finden Sie unter „Regelmetriken“ in Regelmetriken.
- Sie können nach wie vor mit QoS 1 auf Basic Ingest-Themen veröffentlichen. Sie erhalten eine PUBACK, sobald die Nachricht erfolgreich an die Regel-Engine übergeben wurde. Der Empfang einer PUBACK-Nachricht bedeutet nicht, dass Ihre Regelaktionen erfolgreich abgeschlossen wurden. Sie können bei der Ausführung einer Aktion eine Fehleraktion zur Fehlerbehebung konfigurieren. Weitere Informationen finden Sie unter Fehlerbehandlung (Fehleraktion).

AWS IoT SQL-Referenz

In AWS IoT werden Regeln mithilfe einer SQL-like Syntax definiert. SQL-Anweisungen bestehen aus drei Typen von Klauseln:

SET

(Optional) Definiert Variablen, die Sie in SQL-Anweisungen und Ersatzvorlagen wiederverwenden können. Weisen Sie Variablen mithilfe von Ausdrücken Werte zu. Verweisen Sie in SELECT- und WHERE-Klauseln sowie in Vorlagen zur Aktionsersetzung auf diese Variablen.

Die SET-Klausel unterstützt DatentypenOperatoren,Funktionen,Literale,Case-Anweisungen,JSON-Erweiterungen, Variablen und. Verschachtelte Objektabfragen

SELECT

(Erforderlich) Extrahiert Informationen aus der Nutzlast einer eingehenden Nachricht und führt Veränderungen an den Informationen durch. Die zu verwendenden Nachrichten werden durch den in der FROM-Klausel angegebenen Themenfilter identifiziert.

Die SELECT-Klausel unterstützt Datentypen Operatoren FunktionenLiterale,Case-Anweisungen,JSON-Erweiterungen,Ersetzungsvorlagen,,, VariablenVerschachtelte Objektabfragen, undBinäre Nutzlasten.

FROM

Der MQTT-Themenfilter, der Nachrichten identifiziert, aus denen Daten extrahiert werden sollen. Die Regel wird für jede Meldung aktiviert, die an ein MQTT-Topic gesendet wird, das dem hier angegebenen Themenfilter entspricht. Erforderlich für Regeln, die durch Meldungen aktiviert werden, die den Message Broker durchlaufen. Optional für Regeln, die nur unter Verwendung von Basic Ingest ausgelöst werden.

WHERE

(Optional) Fügt eine Bedingungslogik hinzu, die bestimmt, ob die von einer Regel angegebenen Aktionen ausgeführt werden.

Die WHERE-Klausel unterstützt Datentypen Operatoren FunktionenLiterale,Case-Anweisungen,JSON-Erweiterungen,,, Variablen undVerschachtelte Objektabfragen.

Ein Beispiel für eine SQL-Anweisung sieht folgendermaßen aus:

```
SELECT color AS rgb FROM 'topic/subtopic' WHERE temperature > 50
```

Ein Beispiel für eine MQTT-Nachricht (auch eingehende Nutzlast genannt) sieht folgendermaßen aus:

```
{
    "color":"red",
```

```
    "temperature":100
}
```

Wenn diese Nachricht im Topic `'topic/subtopic'` veröffentlicht wird, wird die Regel ausgelöst und die SQL-Anweisung wird ausgewertet. Die SQL-Anweisung extrahiert den Wert der Eigenschaft `color`, wenn die Eigenschaft `"temperature"` größer als 50 ist. Die WHERE-Klausel legt die Bedingung `temperature > 50` fest. Das Schlüsselwort `AS` benennt die Eigenschaft `"color"` in `"rgb"` um. Das Ergebnis (auch ausgehende Nutzlast genannt) sieht folgendermaßen aus:

```
{
    "rgb":"red"
}
```

Diese Daten werden anschließend an die Regelaktion weitergeleitet, die die Daten für eine weitere Verarbeitung versendet. Weitere Informationen zu Regelaktionen unter AWS IoT Regelaktionen.

> **Note**
>
> Kommentare werden derzeit in der AWS IoT SQL-Syntax nicht unterstützt.
> Attributnamen mit Leerzeichen können nicht als Feldnamen in der SQL-Anweisung verwendet werden. Die eingehende Nutzlast kann zwar Attributnamen mit Leerzeichen enthalten, solche Namen können jedoch nicht in der SQL-Anweisung verwendet werden. Sie werden jedoch an die ausgehende Nutzlast weitergegeben, wenn Sie einen Platzhalter (*) für den Feldnamen verwenden.

SELECT-Klausel

Die AWS IoT SELECT-Klausel entspricht im Wesentlichen der ANSI SQL SELECT-Klausel, mit einigen geringfügigen Unterschieden.

Die SELECT-Klausel unterstützt Datentypen OperatorenFunktionen,Literale,Case-Anweisungen,JSON-Erweiterungen,, VariablenVerschachtelte Objektabfragen, undBinäre Nutzlasten.

Sie können die SELECT-Klausel verwenden, um Informationen aus eingehenden MQTT-Nachrichten zu extrahieren. Sie können `SELECT *` auch zum Abrufen der gesamten Nutzlast einer eingehenden Nachricht verwenden. Beispiel:

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
```

```
SQL statement: SELECT * FROM 'topic/subtopic'
Outgoing payload: {"color":"red", "temperature":50}
```

Wenn die Nutzlast ein JSON-Objekt ist, können Sie auf Schlüssel im Objekt verweisen. die ausgehende Nutzlast enthält das Schlüssel-Wert-Paar. Beispiel:

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL statement: SELECT color FROM 'topic/subtopic'
Outgoing payload: {"color":"red"}
```

Sie können Schlüssel mithilfe des Schlüsselworts „AS“ umbenennen. Beispiel:

```
Incoming payload published on topic 'topic/subtopic':{"color":"red", "temperature":50}
SQL:SELECT color AS my_color FROM 'topic/subtopic'
Outgoing payload: {"my_color":"red"}
```

Sie können mehrere Elemente durch ein Komma getrennt auswählen. Beispiel:

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT color as my_color, temperature as fahrenheit FROM 'topic/subtopic'
Outgoing payload: {"my_color":"red","fahrenheit":50}
```

Sie können mehrere Elemente auswählen, indem Sie „*“ einbeziehen, um die Elemente zur eingehenden Nutzlast hinzuzufügen. Beispiel:

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT *, 15 as speed FROM 'topic/subtopic'
Outgoing payload: {"color":"red", "temperature":50, "speed":15}
```

Sie können das Schlüsselwort `"VALUE"` verwenden, um ausgehende Nutzlasten zu erzeugen, die keine JSON-Objekte sind. Mit der SQL-Version `2015-10-08` können Sie nur ein Element auswählen. Mit der SQL-Version `2016-03-23` oder höher können Sie auch ein Array auswählen, das als Objekt der obersten Ebene ausgegeben werden soll.

Example

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT VALUE color FROM 'topic/subtopic'
Outgoing payload: "red"
```

Mit der Syntax `'.'` können Sie verschachtelte JSON-Objekte in der eingehenden Nutzlast analysieren. Beispiel:

```
Incoming payload published on topic 'topic/subtopic': {"color":
{"red":255,"green":0,"blue":0}, "temperature":50}
SQL: SELECT color.red as red_value FROM 'topic/subtopic'
Outgoing payload: {"red_value":255}
```

Informationen über die Verwendung von JSON-Objekt- und Eigenschaftsnamen, die reservierte Zeichen wie Zahlen oder den Bindestrich (Minuszeichen) enthalten, unter JSON-Erweiterungen

Mit Funktionen (siehe Funktionen) können Sie die eingehende Nutzlast umwandeln. Sie können Klammern zum Gruppieren verwenden. Beispiel:

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT (temperature - 32) * 5 / 9 AS celsius, upper(color) as my_color FROM
 'topic/subtopic'
Outgoing payload: {"celsius":10,"my_color":"RED"}
```

Sie können Objektliterale in der SELECT-Klausel verwenden, um JSON-Objekte mit Schlüsseln zu erstellen, die Zeichen enthalten, die von Standardbezeichnern nicht unterstützt werden, z. B. Bindestriche. Beispiel:

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT {"key-with-hyphen": color, "123-numeric-start": temperature} FROM 'topic/
subtopic'
Outgoing payload: {"key-with-hyphen":"red","123-numeric-start":50}
```

FROM-Klausel

Die FROM-Klausel abonniert für die Regel ein Thema oder einen Themenfilter. Schließen Sie das Thema oder den Themenfilter in einfache Anführungszeichen (') ein. Die Regel wird für jede Nachricht ausgelöst, die an ein MQTT-Topic gesendet wird, das mit dem hier angegebenen Topic-Filter übereinstimmt. Sie können eine Gruppe ähnlicher Themen mithilfe eines Themenfilters abonnieren.

Beispiel:

Eingehende Nutzlast veröffentlicht für Topic `'topic/subtopic'`: `{temperature: 50}`

Eingehende Nutzlast veröffentlicht für Topic `'topic/subtopic-2'`: `{temperature: 50}`

SQL: `"SELECT temperature AS t FROM 'topic/subtopic'"`.

Die Regel wird für `'topic/subtopic'` abonniert, daher wird die eingehende Nutzlast an die Regel übergeben. Die ausgehende Nutzlast, die an die Regelaktionen übergeben wird, lautet: `{t: 50}`. Die Regel hat `'topic/subtopic-2'` nicht abonniert, sodass die Regel nicht für die Nachricht ausgelöst wird, die für `'topic/subtopic-2'` veröffentlicht wird.

Beispiel:# Platzhalter

Sie können das Platzhalterzeichen „#“ (mehrere Ebenen) verwenden, um mehrere bestimmte Pfadelemente abzugleichen.

Eingehende Nutzlast veröffentlicht für Topic `'topic/subtopic'`: `{temperature: 50}`

Eingehende Nutzlast veröffentlicht für Topic `'topic/subtopic-2'`: `{temperature: 60}`

Eingehende Nutzlast veröffentlicht für Topic `'topic/subtopic-3/details'`: `{temperature: 70}`

Eingehende Nutzlast veröffentlicht für Topic `'topic-2/subtopic-x'`: `{temperature: 80}`

SQL: `"SELECT temperature AS t FROM 'topic/#'"`.

Die Regel abonniert jedes Thema`'topic'`, das mit 1 beginnt. Sie wird also dreimal ausgeführt, wobei ausgehende Nutzdaten von `{t: 50}` (für topic/subtopic), `{t: 60}` (für topic/subtopic -2) und `{t: 70}` (für topic/subtopic -3/details) an ihre Aktionen gesendet werden. Sie wird nicht auf `'topic-2/subtopic-x'` abonniert, so dass die Regel nicht für die `{temperature: 80}`-Nachricht ausgelöst wird.

Beispiel: +-Platzhalter

Sie können das Platzhalterzeichen „+“ (einzelne Ebene) verwenden, um ein beliebiges Pfadelement abzugleichen:

Eingehende Nutzlast veröffentlicht für Topic `'topic/subtopic'`: `{temperature: 50}`

Eingehende Nutzlast veröffentlicht für Topic `'topic/subtopic-2'`: `{temperature: 60}`

Eingehende Nutzlast veröffentlicht für Topic `'topic/subtopic-3/details'`: `{temperature: 70}`

Eingehende Nutzlast veröffentlicht für Topic `'topic-2/subtopic-x'`: `{temperature: 80}`

SQL: `"SELECT temperature AS t FROM 'topic/+'"`.

Für die Regel sind alle Topics mit zwei Pfadelementen abonniert, bei denen `'topic'` das erste Element ist. Die Regel wird für Nachrichten ausgeführt, die an `'topic/subtopic'` und `'topic/subtopic-2'` gesendet werden, aber nicht an `'topic/subtopic-3/details'` (sie hat mehr Ebenen als der Themenfilter) oder `'topic-2/subtopic-x'` (sie beginnt nicht mit `topic`).

SET-Klausel

Verwenden Sie die SET-Klausel, um Variablen zu definieren, die Ausdrucksergebnisse speichern. Sie können diese Variablen in SELECT- und WHERE-Klauseln sowie in Ersatzvorlagen wiederverwenden. Auf diese Weise können Sie das Duplizieren komplexer Ausdrücke vermeiden und die Anzahl der Funktionsaufrufen in Ihrer SQL-Anweisung reduzieren.

Die SET-Klausel unterstützt Datentypen OperatorenFunktionen,Literale,Case-Anweisungen,JSON-Erweiterungen,, Variablen undVerschachtelte Objektabfragen.

Syntax der SET-Klau

Die SET-Klausel muss in Ihrer SQL-Anweisung vor der SELECT-Klausel stehen. Verwenden Sie die folgende Syntax:

```
SET @variable_name = expression [, @variable_name2 = expression2]
```

Syntaxregeln:

- Beginnen Sie Variablennamen mit `@`
- Variablennamen können Buchstaben, Zahlen und Unterstriche enthalten
- Variablennamen können bis zu 64 Zeichen lang sein
- In einer einzigen SET-Klausel können mehrere Variablen, getrennt durch Kommas, festgelegt werden
- Jede Variable kann nur einmal zugewiesen werden (Variablen sind unveränderlich)
- Das SET-Schlüsselwort kann nur einmal pro SQL-Anweisung verwendet werden

Verwenden von Variablen

Nachdem Sie Variablen definiert haben, können Sie sie verwenden in:

- SELECT-Klauseln
- WHERE-Klauseln

- Andere SET-Variablenzuweisungen
- Vorlagen für die Ersetzung von Aktionen
- Vorlagen für die Ersetzung von Aktionen sind fehlerhaft
- Verschachtelte SELECT-Abfragen
- Funktionsparameter (bestimmte Parameter wie Rolearn-Parameter und Parameter, die den Modus einer Funktion wechseln, unterstützen ähnlich wie `transform("enrichArray", attributes, values)` keine Variablen)

Variablen werden mit derselben `@variable_name` Syntax referenziert, die in der SET-Klausel verwendet wird. Sie können auch die JSON-Erweiterungssyntax verwenden, um auf Eigenschaften von Variablen zuzugreifen, die Objekte enthalten, wie `@variable_name.property` z.

Beispiele für SET-Klausel

Grundlegende Verwendung von Variablen

Das folgende Beispiel zeigt eine Payload, die zum Thema `device/data` veröffentlicht wurde: `{"temp_fahrenheit": 75, "humidity": 60}`

SQL-Anweisung:

```
SET @temp_celsius = (temp_fahrenheit - 32) * 5 / 9
SELECT @temp_celsius AS celsius, humidity FROM 'device/data'
```

Ausgehende Payload: `{"celsius": 23.89, "humidity": 60}`

Greifen Sie auf Mitglieder in eingebetteten JSON-Objekten zu

Das folgende Beispiel zeigt eine Payload, die zum Thema `device/data` veröffentlicht wurde: `{"device1": {"deviceId":"weather_sensor", "deviceData": {"sensors": {"temp_fahrenheit": 75, "humidity": 60}, "location": [47.606,-122.332]}}}`

SQL-Anweisung:

```
SET @device_sensor_data = device1.deviceData.sensors
SELECT @device_sensor_data.temp_fahrenheit AS temp_fahrenheit,
 @device_sensor_data.humidity as humidity, device1.deviceId as deviceId FROM 'device/
data'
```

Ausgehende Payload:
`{"temp_fahrenheit":75,"humidity":60,"deviceId":"weather_sensor"}`

Weitere Informationen zur Arbeit mit JSON-Erweiterungen finden Sie unter JSON-Erweiterungen

Vermeidung doppelter Funktionsaufrufen

SET-Variablen tragen dazu bei, das Duplizieren komplexer Dekodierungsoperationen zu vermeiden:

```
SET @decoded_data = decode(encode(*, 'base64'), 'proto', 'schema', 'schema.desc',
 'message.proto', 'Message')
SELECT @decoded_data.sensor_id, @decoded_data.reading FROM 'device/protobuf'
WHERE @decoded_data.reading > 100
```

Ohne SET-Variablen müssten Sie die Dekodierungsfunktion dreimal wiederholen, was die Grenzwerte für Funktionsaufrufe überschreitet.

Mehrere Variablen

Sie können mehrere Variablen in einer einzigen SET-Klausel definieren, indem Sie sie durch Kommas trennen:

```
SET @user_data = get_user_properties(device_id), @threshold = 50
SELECT @user_data.name, temp_fahrenheit FROM 'sensors/+'
WHERE temp_fahrenheit > @threshold AND @user_data.active = true
```

Verwenden von Variablen in Ersatzvorlagen

Variablen können auch in Vorlagen zur Aktionsersetzung verwendet werden, sodass Sie berechnete Werte sowohl in der SQL-Anweisung als auch in den Regelaktionen wiederverwenden können.

SQL-Anweisung:

```
SET @temp_celsius = (temp_fahrenheit - 32) * 5 / 9
SELECT @temp_celsius AS celsius, humidity FROM 'device/data'
```

Aktionskonfiguration:

```
{
  "s3": {
    "roleArn": "arn:aws:iam::123456789012:role/testRuleRole",
```

```
    "bucketName": "bucket",
    "key": "temperature-data/${device_id}/temp-${@temp_celsius}C.json"
  }
}
```

In diesem Beispiel `@temp_celsius` wird die SET-Variable in einer Ersatzvorlage verwendet, um das Schlüsselfeld der S3-Aktion zu erstellen.

Non-JSON Nutzung der Nutzlast

SET-Variablen unterstützen keine direkten Nutzlasten, die nicht von JSON stammen, daher muss die Nutzlast zuerst codiert oder dekodiert werden:

```
SET @encoded_payload = encode(*, 'base64')
SELECT @encoded_payload AS raw_data FROM 'device/binary'
```

Weitere Informationen zur Arbeit mit Nicht-JSON-Payloads finden Sie unter Arbeiten mit binären Nutzlasten

Grenzen der SET-Klausel

Die folgenden Grenzwerte gelten für SET-Variablen:

- Maximal 10 eindeutige Variablen pro SQL-Anweisung
- Maximale Größe des Variablenwerts von 128 KiB (minimierte UTF-8 JSON-Zeichenfolge)
- Maximale Gesamtwertgröße von 128 KiB für alle Variablen
- Variablennamen sind auf 64 Zeichen begrenzt
- Variablen können JSON-Nutzlasten direkt so akzeptieren, wie sie sind (Nicht-JSON-Nutzlasten müssen es zuerst sein) encoded/decoded

WHERE-Klausel

Die WHERE-Klausel bestimmt, ob die durch eine Regel angegeben Aktionen ausgeführt werden. Wenn die WHERE-Klausel wahr ist, werden die Regelaktionen ausgeführt. Andernfalls werden die Regelaktionen nicht ausgeführt.

Die WHERE-Klausel unterstützt Datentypen Operatoren FunktionenLiterale,Case-Anweisungen,JSON-Erweiterungen,,, Variablen undVerschachtelte Objektabfragen.

Beispiel:

Eingehende Nutzlast veröffentlicht für `topic/subtopic`: `{"color":"red", "temperature":40}`

SQL: `SELECT color AS my_color FROM 'topic/subtopic' WHERE temperature > 50 AND color <> 'red'`.

In diesem Fall wird die Regel ausgelöst. Die durch die Regel angegebenen Aktionen werden jedoch nicht ausgeführt. Es gibt keine ausgehende Nutzlast.

In der WHERE-Klausel können Sie Funktionen und Operatoren verwenden. Sie können jedoch nicht auf Aliase verweisen, die in SELECT mit dem Schlüsselwort AS erstellt wurden. Die WHERE-Klausel wird zuerst ausgewertet, um zu bestimmen, ob SELECT ausgewertet wurde.

Beispiel mit Nicht-JSON-Nutzdaten:

Eingehende Nicht-JSON-Nutzdaten, veröffentlicht am ``: topic/subtopic `80`

SQL: `` `SELECT decode(encode(*, 'base64'), 'base64') AS value FROM 'topic/subtopic' WHERE decode(encode(*, 'base64'), 'base64') > 50 ``

In diesem Fall wird die Regel ausgelöst und durch die Regel angegebene Aktionen werden ausgeführt. Die ausgehende Nutzlast wird durch die SELECT-Klausel in eine JSON-Nutzlast umgewandelt. `{"value":80}`

Datentypen

Die AWS IoT Regel-Engine unterstützt alle JSON-Datentypen.

Unterstützte Datentypen

Typ	Bedeutung
`Int`	Eine separate `Int`. Maximal 34 Ziffern.
`Decimal`	A `Decimal` mit einer Genauigkeit von 34 Ziffern, mit einer Mindestgröße ungleich Null von 1 E-999 und einer maximalen Größe von 9,999... E999.

Typ	Bedeutung
	Note Einige Funktionen geben `Decimal`-Werte mit doppelter Genauigkeit anstelle von genau 34 Zeichen aus. Mit SQL V2 (23.03.2016) werden numerische Werte, die ganze Zahlen sind, wie z. B. `10.0`, als `Int` Wert (`10`) statt als erwarteter `Decimal` Wert () verarbeitet. `10.0` Um ganzzahlige numerische Werte zuverlässig als `Decimal`-Werte zu verarbeiten, verwenden Sie SQL V1 (2015-10-08) für die Regelabfrageanweisung.
`Boolean`	`True` oder `False`.
`String`	Eine Zeichenfolge. UTF-8
`Array`	Eine Serie von Werten, die nicht den gleichen Typ aufweisen müssen
`Object`	Ein JSON-Wert, der aus einem Schlüssel und einem Wert besteht. Schlüssel müssen Zeichenfolgen sein. Werte können jeden Typ aufweisen.
`Null`	`Null` wie von JSON definiert. Dies ist ein tatsächlicher Wert, der die Abwesenheit eines Werts darstellt. Sie können einen `Null`-Wert explizit erstellen, indem Sie das Schlüsselwort `Null` in Ihrer SQL-Anweisung verwenden. Beispiel: `"SELECT NULL AS n FROM 'topic/subtopic'"`

Typ	Bedeutung

`Undefined`	Kein Wert. Dies kann in JSON nicht explizit dargestellt werden, außer durch Auslassen des Werts. Z. B. im Objekt `{"foo": null}` gibt der Schlüssel "foo" NULL zurück, der Schlüssel "bar" jedoch `Undefined` . Intern behandelt die SQL-Sprache `Undefined` als Wert, kann jedoch nicht in JSON dargestellt werden. Bei einer Serialisierung in JSON sind die Ergebnisse daher `Undefined` .
	`{"foo":null, "bar":undefined}`
	wird in JSON serialisiert als:
	`{"foo":null}`
	Dementsprechend wird `Undefined` in eine leere Zeichenfolge konvertiert, wenn es selbst konvertiert wird. Funktionen, die mit ungültigen Argumenten aufgerufen werden (z. B. falsche Typen, falsche Anzahl an Argumenten usw.), geben `Undefined` zurück.

Konversionen

Die folgende Tabelle listet die Ergebnisse auf, wenn ein Wert eines Typs in einen anderen Typ konvertiert wird (wenn ein Wert mit dem falschen Typ an eine Funktion übergeben wird). Wenn beispielsweise der absoluten Wertfunktion "abs" (die `Int` oder `Decimal` erwartet) ein `String`-Wert übergeben wird, versucht diese, den `String`-Wert nach diesen Regeln in einen `Decimal`-Wert umzuwandeln. In diesem Fall wird "abs("-5.123")" als "abs(-5.123)" behandelt.

> **Note**
>
> Konversionen in `Array`, `Object`, `Null` oder `Undefined` werden nicht versucht.

In Dezimalwerte

Argumenttyp	Ergebnis
`Int`	Ein Wert vom Typ `Decimal` ohne Dezimaltr ennzeichen
`Decimal`	Der Quellwert
`Boolean`	`Undefined` . (Sie können die cast-Funktion explizit zum Umwandeln von true = 1.0, false = 0.0 verwenden.)
`String`	Die SQL-Engine versucht, die Zeichenfo lge als `Decimal` zu analysieren. AWS IoT versucht, Zeichenketten zu analysieren, die dem regulären Ausdruck entsprechen:`^-?\d +(\.\d+)?((?i)E-?\d+)?$` . „0", „-1.2", „5" E-12 sind Beispiele für Zeichenketten, die automatisch in s konvertiert werden. `Decimal`
Array	`Undefined` .
Object	`Undefined` .
Null	`Null`.
Undefined	`Undefined` .

In Ganzzahlen

Argumenttyp	Ergebnis
`Int`	Der Quellwert
`Decimal`	Der Quellwert, auf den nächsten `Int`-Wert gerundet.

Argumenttyp	Ergebnis
`Boolean`	`Undefined` . (Sie können die cast-Funktion explizit zum Umwandeln von true = 1.0, false = 0.0 verwenden.)
`String`	Die SQL-Engine versucht, die Zeichenfolge als zu analysieren. `Decimal` AWS IoT versucht, Zeichenketten zu analysieren, die dem regulären Ausdruck entsprechen:`^-?\d+(\.\d+)?((?i)E-?\d+)?$` . „0", „-1.2", „5E-12“ sind alles Beispiele für Zeichenketten, die automatisch in `Decimal` s konvertiert werden. AWS IoT Versucht, die in a `String` zu konvertieren`Decimal`, und schneidet dann die Dezimalstellen ab, um eine zu bilden. `Decimal` `Int`
Array	`Undefined` .
Object	`Undefined` .
Null	`Null`.
Undefined	`Undefined` .

In Boolesche Werte

Argumenttyp	Ergebnis
`Int`	`Undefined` . (Sie können die `cast`-Funktion explizit zum Umwandeln von 0 = False, any_nonzero_value = True verwenden.)
`Decimal`	`Undefined` . (Sie können die cast-Funktion explizit zum Umwandeln von 0 = False, any_nonzero_value = True verwenden.)

Argumenttyp	Ergebnis
`Boolean`	Der ursprüngliche Wert
`String`	"true"=wahr und "false"=falsch (ohne Beachtung der Groß- und Kleinschreibung). Andere Zeichenfolgenwerte sind `Undefined` .
Array	`Undefined` .
Object	`Undefined` .
Null	`Undefined` .
Undefined	`Undefined` .

In Zeichenfolgen

Argumenttyp	Ergebnis
`Int`	Eine Zeichenfolgendarstellung des `Int`-Werts in Standardnotation
`Decimal`	Eine Zeichenfolge, die den `Decimal`-Wert in Standardnotation darstellt
`Boolean`	"true" oder "false". Alles in Kleinbuchstaben.
`String`	Der ursprüngliche Wert
Array	Das in JSON serialisierte `Array`. Die resultier ende Zeichenfolge ist eine durch Kommata getrennte Liste in eckigen Klammern. Ein `String` ist von Anführungszeichen umschloss en. `Decimal`, `Int`, `Boolean` und `Null` sind dies nicht.
Objekt	Das in JSON serialisierte Objekt. Die resultier ende Zeichenfolge ist eine durch Kommata

Argumenttyp	Ergebnis
	getrennte Liste von Schlüssel-Wert-Paaren, die mit geschweiften Klammern beginnt und endet. Ein `String` ist von Anführungszeichen umschlossen. `Decimal`, `Int`, `Boolean` und `Null` sind dies nicht.
Null	`Undefined` .
Undefined	Undefined

Operatoren

Die folgenden Operatoren können in SELECT- und WHERE-Klauseln verwendet werden.

AND-Operator

Gibt ein Ergebnis vom Typ `Boolean` zurück. Führt einen logischen AND-Vorgang aus. Gibt "true" zurück, wenn die Operanden links und rechts wahr sind. Ansonsten wird „false“ zurückgegeben. Operanden vom Typ `Boolean` oder die Zeichenfolgenoperanden „true“ oder „false“ (ohne Beachtung der Groß- und Kleinschreibung) sind erforderlich.

Syntax: *`expression`* `AND` *`expression`*.

AND-Operator

Left Operator	Right operator	Output
`Boolean`	`Boolean`	`Boolean`. True, wenn beide Operanden True sind. Ansonsten „false“.
`String/Boolean`	`String/Boolean`	Wenn alle Zeichenfolgen „true“ oder „false“ (ohne Beachtung der Groß- und Kleinschreibung) sind, werden sie in den Typ `Boolean` konvertiert und normal als *`boolean`* `AND` *`boolean`* verarbeitet.
Anderer Wert	Anderer Wert	`Undefined` .

OR-Operator

Gibt ein Ergebnis vom Typ `Boolean` zurück. Führt einen logischen OR-Vorgang aus. Gibt „true" zurück, wenn der linke oder der rechte Operand wahr ist. Ansonsten wird „false" zurückgegeben. Operanden vom Typ `Boolean` oder die Zeichenfolgenoperanden „true" oder „false" (ohne Beachtung der Groß- und Kleinschreibung) sind erforderlich.

Syntax: *`expression`* `OR` *`expression`*.

OR-Operator

Left operator	Right operator	Output
`Boolean`	`Boolean`	`Boolean`. True, wenn einer der Operanden True ist. Ansonsten „false".
`String/Boolean`	`String/Boolean`	Wenn alle Zeichenfolgen „true" oder „false" (ohne Beachtung der Groß- und Kleinschreibung) sind, werden sie in boolesche Werte konvertiert und normal als *`boolean`* `OR` *`boolean`* verarbeitet.
Anderer Wert	Anderer Wert	`Undefined` .

NOT-Operator

Gibt ein Ergebnis vom Typ `Boolean` zurück. Führt einen logischen NOT-Vorgang aus. Gibt „true" zurück, wenn der Operand falsch ist. Gibt andernfalls „true" zurück. Ein `Boolean` Operand oder der Zeichenfolgenoperand „true" oder „false" (ohne Beachtung der Groß- und Kleinschreibung) ist erforderlich.

Syntax: `NOT` *`expression`*.

NOT-Operator

Operand	Ausgabe
`Boolean`	`Boolean`. True, wenn der Operand False ist. Verwenden Sie andernfalls "true".

Operand	Ausgabe
`String`	Ist die Zeichenfolge „true“ oder „false“ (ohne Beachtung der Groß- und Kleinschreibung), wird sie in den entsprechenden booleschen Wert konvertiert und der gegenteilige Wert wird zurückgegeben.
Anderer Wert	`Undefined` .

IN-Operator

Gibt ein Ergebnis vom Typ `Boolean` zurück. Sie können den IN-Operator in einer WHERE-Klausel verwenden, um zu überprüfen, ob ein Wert mit einem Wert in einem Array übereinstimmt. Er gibt „true“ zurück, wenn die Übereinstimmung gefunden wird, andernfalls „false“.

Syntax: *`expression`* `IN` *`expression`*.

IN-Operator

Left operator	Right operator	Output
`Int/Decimal/String/A`	`Array`	Stimmt, wenn das `Object` Element `Integer` `Decimal` `StringArray////`im Array gefunden wird. Ansonsten „false“.

Beispiel:

```
SQL: "select * from 'a/b' where 3 in arr"

JSON: {"arr":[1, 2, 3, "three", 5.7, null]}
```

In diesem Beispiel `where 3 in arr` wird die Bedingungsklausel als wahr ausgewertet, da 3 in dem genannten Array vorhanden ist`arr`. Daher `select * from 'a/b'` wird in der SQL-Anweisung ausgeführt. Dieses Beispiel zeigt auch, dass das Array heterogen sein kann.

EXISTS-Operator

Gibt ein Ergebnis vom Typ `Boolean` zurück. Sie können den EXISTS-Operator in einer Bedingungsklausel verwenden, um zu testen, ob Elemente in einer Unterabfrage vorhanden sind. Er gibt true zurück, wenn die Unterabfrage ein oder mehrere Elemente zurückgibt, und false, wenn die Unterabfrage keine Elemente zurückgibt.

Syntax: *`expression`*.

Beispiel:

```
SQL: "select * from 'a/b' where exists (select * from arr as a where a = 3)"

JSON: {"arr":[1, 2, 3]}
```

In diesem Beispiel `where exists (select * from arr as a where a = 3)` wird die Bedingungsklausel als wahr ausgewertet, weil 3 in dem genannten Array vorhanden ist. `arr` Daher `select * from 'a/b'` wird in der SQL-Anweisung ausgeführt.

Beispiel:

```
SQL: select * from 'a/b' where exists (select * from e as e where foo = 2)

JSON: {"foo":4,"bar":5,"e":[{"foo":1},{"foo":2}]}
```

In diesem Beispiel `where exists (select * from e as e where foo = 2)` wird die Bedingungsklausel als wahr ausgewertet, da das Array e innerhalb des JSON-Objekts das Objekt enthält`{"foo":2}`. Daher `select * from 'a/b'` wird in der SQL-Anweisung ausgeführt.

>-Operator

Gibt ein Ergebnis vom Typ `Boolean` zurück. Gibt "true" zurück, wenn der linke Operand größer ist als der rechte Operand. Beide Operanden werden in den Typ `Decimal` konvertiert und anschließend verglichen.

Syntax: *`expression`* `>` *`expression`*.

>-Operator

Left operator	Right operator	Output
`Int/Decimal`	`Int/Decimal`	`Boolean`. „true“, wenn der linke Operand größer ist als der rechte Operand. Ansonsten „false“.
`String/Int/Deci`	`String/Int/Deci`	Wenn alle Zeichenfolgen in den Typ `Decimal` konvertiert werden können: `Boolean`. Gibt "true" zurück, wenn der linke Operand größer ist als der rechte Operand. Ansonsten „false“.
Anderer Wert	`Undefined .`	`Undefined .`

>= Operator

Gibt ein Ergebnis vom Typ `Boolean` zurück. Gibt "true" zurück, wenn der linke Operand mindestens genauso groß ist wie der rechte Operand. Beide Operanden werden in den Typ `Decimal` konvertiert und anschließend verglichen.

Syntax: *`expression`* `>=` *`expression`*.

>= Betreiber

Left operator	Right operator	Output
`Int/Decimal`	`Int/Decimal`	`Boolean`. „true“, wenn der linke Operand mindestens genauso groß ist wie der rechte Operand. Ansonsten „false“.
`String/Int/Deci`	`String/Int/Deci`	Wenn alle Zeichenfolgen in den Typ `Decimal` konvertiert werden können: `Boolean`. Gibt "true" zurück, wenn der linke Operand mindestens genauso groß ist wie der rechte Operand. Ansonsten „false“.
Anderer Wert	`Undefined .`	`Undefined .`

<-Operator

Gibt ein Ergebnis vom Typ `Boolean` zurück. Gibt "true" zurück, wenn der linke Operand kleiner ist als der rechte Operand. Beide Operanden werden in den Typ `Decimal` konvertiert und anschließend verglichen.

Syntax: *`expression`* `<` *`expression`*.

<-Operator

Left operator	Right operator	Output
`Int/Decimal`	`Int/Decimal`	`Boolean`. „true“, wenn der linke Operand kleiner ist als der rechte Operand. Ansonsten „false“.
`String/Int/Deci`	`String/Int/Deci`	Wenn alle Zeichenfolgen in den Typ `Decimal` konvertiert werden können: `Boolean`. Gibt "true" zurück, wenn der linke Operand kleiner ist als der rechte Operand. Ansonsten „false“.
Anderer Wert	`Undefined`	`Undefined`

<= Betreiber

Gibt ein Ergebnis vom Typ `Boolean` zurück. Gibt "true" zurück, wenn der linke Operand höchstens genauso groß ist wie der rechte Operand. Beide Operanden werden in den Typ `Decimal` konvertiert und anschließend verglichen.

Syntax: *`expression`* `<=` *`expression`*.

<= Betreiber

Left operator	Right operator	Output
`Int/Decimal`	`Int/Decimal`	`Boolean`. „true“, wenn der linke Operand höchstens genauso groß ist wie der rechte Operand. Ansonsten „false“.
`String/Int/Deci`	`String/Int/Deci`	Wenn alle Zeichenfolgen in den Typ `Decimal` konvertiert werden können: `Boolean`. Gibt "true" zurück, wenn der

Left operator	Right operator	Output
		linke Operand höchstens genauso groß ist wie der rechte Operand. Ansonsten „false“.
Anderer Wert	`Undefined`	`Undefined`

<>-Operator

Gibt ein Ergebnis vom Typ `Boolean` zurück. Gibt „true” zurück, wenn die Operanden links und rechts nicht gleich sind. Ansonsten wird "false" zurückgegeben.

Syntax: *`expression`* `<>` *`expression`*.

<>-Operator

Left operator	Right operator	Output
`Int`	`Int`	"True", wenn der linke Operand und der rechte Operand nicht gleich sind; ansonsten "false". Ansonsten „false“.
`Decimal`	`Decimal`	"True", wenn der linke Operand und der rechte Operand nicht gleich sind; ansonsten "false". Ansonsten "false". Der Typ `Int` wird vor dem Vergleichen in den Typ `Decimal` umgewandelt.
`String`	`String`	"True", wenn der linke Operand und der rechte Operand nicht gleich sind; ansonsten "false". Ansonsten „false“.
Array	Array	"True", wenn die Elemente in den einzelnen Operanden nicht gleich sind und nicht in der gleichen Reihenfolge vorliegen. Ansonsten "false".
Objekt	Objekt	"True", wenn die Schlüssel und Werte der einzelnen Operanden nicht gleich sind. Ansonsten „false“. Die Reihenfolge von keys/values ist unwichtig.
Null	Null	Falsch.
Beliebiger Wert	`Undefined`	Undefined

Left operator	Right operator	Output
`Undefined`	Beliebiger Wert	Undefined
Falscher Typ	Falscher Typ	true

=-Operator

Gibt ein Ergebnis vom Typ `Boolean` zurück. Gibt „true” zurück, wenn die Operanden links und rechts gleich sind. Ansonsten wird "false" zurückgegeben.

Syntax: *`expression`* `=` *`expression`*.

=-Operator

Left operator	Right operator	Output
`Int`	`Int`	"True", wenn der linke Operand und der rechte Operand nicht gleich sind. Ansonsten „false“.
`Decimal`	`Decimal`	"True", wenn der linke Operand und der rechte Operand nicht gleich sind. Ansonsten "false". Der Typ `Int` wird vor dem Vergleichen in den Typ `Decimal` umgewandelt.
`String`	`String`	"True", wenn der linke Operand und der rechte Operand nicht gleich sind. Ansonsten „false“.
Array	Array	"True", wenn die Elemente in den einzelnen Operanden gleich sind und in der gleichen Reihenfolge vorliegen. Ansonsten „false“.
Objekt	Objekt	"True", wenn die Schlüssel und Werte der einzelnen Operanden gleich sind. Ansonsten „false“. Die Reihenfol ge von keys/values ist unwichtig.
Beliebiger Wert	`Undefined`	`Undefined` .
`Undefined`	Beliebiger Wert	`Undefined` .
Falscher Typ	Falscher Typ	"false"

+-Operator

"+" ist ein überladener Operator. Er kann zum Verketten von Zeichenfolgen oder zum Addieren verwendet werden.

Syntax: *`expression`* `+` *`expression`*.

+-Operator

Left operator	Right operator	Output
`String`	Beliebiger Wert	Konvertiert den rechten Operanden in eine Zeichenfolge und hängt ihn an den linken Operanden an
Beliebiger Wert	`String`	Konvertiert den linken Operanden in eine Zeichenfolge und hängt den rechten Operanden an den konvertierten linken Operanden an
`Int`	`Int`	`Int` Wert. Addiert Operanden.
`Int/Decimal`	`Int/Decimal`	`Decimal` Wert. Addiert Operanden.
Anderer Wert	Anderer Wert	`Undefined` .

--Operator

Subtrahiert den rechten Operanden vom linken Operanden

Syntax: *`expression`* `-` *`expression`*.

--Operator

Left operator	Right operator	Output
`Int`	`Int`	`Int` Wert. Subtrahiert den rechten Operanden vom linken Operanden.
`Int/Decimal`	`Int/Decimal`	`Decimal` Wert. Subtrahiert den rechten Operanden vom linken Operanden.

Left operator	Right operator	Output
`String/Int/Deci`	`String/Int/Deci`	Wenn alle Zeichenfolgen korrekt zu Dezimalwerten konvertiert wurden, wird ein `Decimal`-Wert zurückgeg eben. Subtrahiert den rechten Operanden vom linken Operanden. Gibt andernfalls `Undefined` zurück.
Anderer Wert	Anderer Wert	`Undefined` .
Anderer Wert	Anderer Wert	`Undefined` .

*-Operator

Multipliziert den linken Operanden mit dem rechten Operanden

Syntax: *`expression`* * *`expression`*.

*-Operator

Left operator	Right operator	Output
`Int`	`Int`	`Int` Wert. Multipliziert den linken Operanden mit dem rechten Operanden
`Int/Decimal`	`Int/Decimal`	`Decimal` Wert. Multipliziert den linken Operanden mit dem rechten Operanden
`String/Int/Deci`	`String/Int/Deci`	Wenn alle Zeichenfolgen korrekt zu Dezimalwerten konvertiert wurden, wird ein `Decimal`-Wert zurückgeg eben. Multipliziert den linken Operanden mit dem rechten Operanden Gibt andernfalls `Undefined` zurück.
Anderer Wert	Anderer Wert	`Undefined` .

/-Operator

Dividiert den linken Operanden durch den rechten Operanden

Syntax: *`expression`* / *`expression`*.

/-Operator

Left operator	Right operator	Output
`Int`	`Int`	`Int` Wert. Dividiert den linken Operanden durch den rechten Operanden
`Int/Decimal`	`Int/Decimal`	`Decimal` Wert. Dividiert den linken Operanden durch den rechten Operanden
`String/Int/Deci`	`String/Int/Deci`	Wenn alle Zeichenfolgen korrekt zu Dezimalwerten konvertiert wurden, wird ein `Decimal`-Wert zurückgeg eben. Dividiert den linken Operanden durch den rechten Operanden Gibt andernfalls `Undefined` zurück.
Anderer Wert	Anderer Wert	`Undefined` .

%-Operator

Gibt den Rest zurück, der beim Dividieren des linken Operanden durch den rechten Operanden entsteht.

Syntax: *`expression`* `%` *`expression`*.

%-Operator

Left operator	Right operator	Output
`Int`	`Int`	`Int` Wert. Gibt den Rest zurück, der beim Dividieren des linken Operanden durch den rechten Operanden entsteht.
`String/Int/Deci`	`String/Int/Deci`	Wenn alle Zeichenfolgen korrekt zu Dezimalwerten konvertiert wurden, wird ein `Decimal`-Wert zurückgeg eben. Gibt den Rest zurück, der beim Dividieren des linken Operanden durch den rechten Operanden entsteht. Andernfalls `Undefined` .
Anderer Wert	Anderer Wert	`Undefined` .

Funktionen

Verwenden Sie die folgenden integrierten Funktionen in den SELECT- oder WHERE-Klauseln Ihrer SQL-Ausdrücke.

Die folgenden externen Funktionen werden wie eine Regelaktion abgerechnet: `aws_lambda`, `get_dynamodb()get_registry_data()`, und. `get_thing_shadow()` Außerdem wird Ihnen die `decode()` Funktion nur dann in Rechnung gestellt, wenn Sie eine Protobuf-Nachricht in JSON dekodieren. Weitere Informationen finden Sie auf der AWS IoT Core Seite mit den Preisen.

abs(Decimal)

Gibt den absoluten Wert einer Zahl zurück. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `abs(-5)` gibt 5 zurück.

Argumenttyp	Ergebnis
`Int`	`Int`, der absolute Wert des Arguments
`Decimal`	`Decimal`, der absolute Wert des Arguments
`Boolean`	`Undefined` .
`String`	`Decimal`: das Ergebnis ist der absolute Wert des Arguments. Wenn die Zeichenfolge nicht konvertiert werden kann, ist das Ergebnis `Undefined` .
Array	`Undefined` .
Object	`Undefined` .
Null	`Undefined` .
Undefined	`Undefined` .

accountid()

Gibt die ID des Kontos zurück, das diese Regel als `String` besitzt. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel:

`accountid()` = "123456789012"

acos(Decimal)

Gibt den umgekehrten Kosinus einer Zahl im Bogenmaß zurück. `Decimal`-Argumente werden vor dem Anwenden der Funktion auf doppelte Genauigkeit gerundet. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `acos(0)` = 1,5707963267948966

Argumenttyp	Ergebnis
`Int`	`Decimal` (mit doppelter Genauigkeit), der umgekehrte Kosinus des Arguments . Imaginäre Ergebnisse werden als `Undefined` zurückgegeben.
`Decimal`	`Decimal` (mit doppelter Genauigkeit), der umgekehrte Kosinus des Arguments . Imaginäre Ergebnisse werden als `Undefined` zurückgegeben.
`Boolean`	`Undefined` .
`String`	`Decimal`, der umgekehrte Kosinus des Arguments. Wenn die Zeichenfolge nicht konvertiert werden kann, ist das Ergebnis `Undefined` . Imaginäre Ergebnisse werden als `Undefined` zurückgegeben.
Array	`Undefined` .
Object	`Undefined` .

Argumenttyp	Ergebnis
Null	`Undefined` .
Undefined	`Undefined` .

asin(Decimal)

Gibt den umgekehrten Sinus einer Zahl im Bogenmaß zurück. `Decimal`-Argumente werden vor dem Anwenden der Funktion auf doppelte Genauigkeit gerundet. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `asin(0)` = 0,0

Argumenttyp	Ergebnis
`Int`	`Decimal` (mit doppelter Genauigkeit), der umgekehrte Sinus des Arguments . Imaginäre Ergebnisse werden als `Undefined` zurückgegeben.
`Decimal`	`Decimal` (mit doppelter Genauigkeit), der umgekehrte Sinus des Arguments . Imaginäre Ergebnisse werden als `Undefined` zurückgegeben.
`Boolean`	`Undefined` .
`String`	`Decimal` (mit doppelter Genauigkeit), der umgekehrte Sinus des Arguments. Wenn die Zeichenfolge nicht konvertiert werden kann, ist das Ergebnis `Undefined` . Imaginäre Ergebnisse werden als `Undefined` zurückgegeben.
Array	`Undefined` .
Object	`Undefined` .

Argumenttyp	Ergebnis
Null	`Undefined` .
Undefined	`Undefined` .

atan(Decimal)

Gibt den umgekehrten Tangens einer Zahl im Bogenmaß zurück. `Decimal`-Argumente werden vor Anwendung des Features auf doppelte Genauigkeit gerundet. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `atan(0)` = 0,0

Argumenttyp	Ergebnis
`Int`	`Decimal` (mit doppelter Genauigkeit), der umgekehrte Tangens des Arguments . Imaginäre Ergebnisse werden als `Undefined` zurückgegeben.
`Decimal`	`Decimal` (mit doppelter Genauigkeit), der umgekehrte Tangens des Arguments . Imaginäre Ergebnisse werden als `Undefined` zurückgegeben.
`Boolean`	`Undefined` .
`String`	`Decimal`, der umgekehrte Tangens des Arguments. Wenn die Zeichenfolge nicht konvertiert werden kann, ist das Ergebnis `Undefined` . Imaginäre Ergebnisse werden als `Undefined` zurückgegeben.
Array	`Undefined` .
Object	`Undefined` .

Argumenttyp	Ergebnis
Null	`Undefined` .
Undefined	`Undefined` .

atan2(Decimal, Decimal)

Gibt den Winkel im Bogenmaß zwischen der positiven x-Achse und dem Punkt (x, y) an, der in den beiden Argumenten definiert ist. Der Winkel ist positiv für Winkel gegen den Uhrzeigersinn (obere Halbebene, y > 0) und negativ für Winkel im Uhrzeigersinn (untere Halbebene, y < 0). `Decimal` Argumente werden vor dem Anwenden der Funktion auf doppelte Genauigkeit gerundet. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `atan2(1, 0)` = 1,5707963267948966

Argumenttyp	Argumenttyp	Ergebnis
`Int/Decimal`	`Int/Decimal`	`Decimal` (mit doppelter Genaui Winkel zwischen der x-Achse ur festgelegten Punkt (x, y)
`Int/Decimal/String`	`Int/Decimal/String`	`Decimal`, der umgekehrte Tang des beschriebenen Punkts. Wer Zeichenfolge nicht konvertiert w ist das Ergebnis `Undefined` .
Anderer Wert	Anderer Wert	`Undefined` .

aws_lambda(functionArn, inputJson)

Ruft die angegebene Lambda-Funktion. Dabei wird `inputJson` an die Lambda-Funktion übergeben und das von der Lambda-Funktion generierte JSON-Objekt zurückgegeben.

Argumente

Argument	Description
`functionArn`	Der ARN der aufzurufenden Lambda-Funktion. Die Lambda-Funktion muss JSON-Daten zurückgeben.
`inputJson`	Die an die Lambda-Funktion übergebene JSON-Eingabe. Um Abfragen und Literale verschachtelter Objekte zu übergeben, müssen Sie die SQL-Version 2016-03-23 verwenden.

Sie müssen AWS IoT `lambda:InvokeFunction` Berechtigungen erteilen, um die angegebene Lambda-Funktion aufzurufen. Das folgende Beispiel zeigt, wie die `lambda:InvokeFunction`-Berechtigung mit AWS CLI erteilt wird:

```
aws lambda add-permission --function-name "function_name"
--region "region"
--principal iot.amazonaws.com
--source-arn arn:aws:iot:us-east-1:account_id:rule/rule_name
--source-account "account_id"
--statement-id "unique_id"
--action "lambda:InvokeFunction"
```

Im Folgenden werden die Argumente für den Befehl add-permission aufgeführt:

--function-name

Der Name der Lambda-Funktion. Sie fügen eine neue Berechtigung hinzu, um die Ressourcenrichtlinie der Funktion zu aktualisieren.

--Region

Die AWS-Region Ihres Kontos.

--Prinzipal

Der Prinzipal, der die Berechtigung erhält. Dies sollte dazu dienen`iot.amazonaws.com`, die AWS IoT Erlaubnis zum Aufrufen einer Lambda-Funktion zu gewähren.

--source-arn

Der ARN der Regel. Sie können den get-topic-rule AWS CLI Befehl verwenden, um den ARN einer Regel abzurufen.

--source-account

Der AWS-Konto Ort, an dem die Regel definiert ist.

--statement-id

Ein eindeutiger Anweisungsbezeichner

--action

Die Lambda-Aktionen, die Sie in dieser Anweisung zulassen möchten. Um AWS IoT den Aufruf einer Lambda-Funktion zu erlauben, geben Sie `lambda:InvokeFunction` an.

> **Important**
>
> Wenn Sie eine Berechtigung für einen AWS IoT Prinzipal hinzufügen, ohne das `source-arn` oder anzugeben, kann jedes`source-account`, AWS-Konto die mit Ihrer Lambda-Aktion eine Regel erstellt, Regeln auslösen, von denen aus Ihre Lambda-Funktion aufgerufen wird. AWS IoT Weitere Informationen finden Sie unter Lambda-Berechtigungsmodell.

Bei einer JSON-Nachrichtennutzlast wie:

```
{
    "attribute1": 21,
    "attribute2": "value"
}
```

Die `aws_lambda`-Funktion kann verwendet werden, um die Lambda-Funktion wie folgt aufzurufen.

```
SELECT
aws_lambda("arn:aws:lambda:us-east-1:account_id:function:lambda_function",
 {"payload":attribute1}) as output FROM 'topic-filter'
```

Wenn Sie möchten, dass die vollständige MQTT-Nachrichtnutzlast übergeben wird, können Sie die JSON-Nutzlast mit „*“ angeben.

```
SELECT
aws_lambda("arn:aws:lambda:us-east-1:account_id:function:lambda_function", *) as output
 FROM 'topic-filter'
```

`payload.inner.element`wählt Daten aus Nachrichten aus, die zum Thema " veröffentlicht wurden. topic/subtopic

`some.value` wählt Daten aus der Ausgabe aus, die von der Lambda Funktion generiert wurde.

> **Note**
>
> Die Regeln-Engine begrenzt die Ausführungsdauer von Lambda-Funktionen. Aufrufe von Lambda-Funktionen über Regeln sollten innerhalb von 2.000 Millisekunden abgeschlossen werden.

bitand(Int, Int)

Führt Bit für Bit AND für die Bit-Darstellungen der beiden `Int`(-konvertierten) Argumente durch. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `bitand(13, 5)` = 5

Argumenttyp	Argumenttyp	Ergebnis
`Int`	`Int`	`Int`, ein bitweises AND der beid Argumente.
`Int/Decimal`	`Int/Decimal`	`Int`, ein bitweises AND der beid Argumente. Alle Zahlen, die nicl Typ `Int` sind, werden auf den n `Int`-Wert abgerundet. Wenn ein Argumente nicht in einen `Int`-W konvertiert werden kann, ist das `Undefined` .
`Int/Decimal/String`	`Int/Decimal/String`	`Int`, ein bitweises AND der beid Argumente. Alle Zeichenfolgen in Dezimal-Werte konvertiert un nächsten `Int`-Wert abgerundet. Konvertierung fehlschlägt, ist da `Undefined` .

Argumenttyp	Argumenttyp	Ergebnis
Anderer Wert	Anderer Wert	`Undefined` .

bitor(Int, Int)

Führt eine bBit für Bit einen OR-Vorgang für die Bit-Darstellungen der beiden Argumente durch. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `bitor(8, 5)` = 13

Argumenttyp	Argumenttyp	Ergebnis
`Int`	`Int`	`Int`, das bitweise OR der beide Argumente
`Int/Decimal`	`Int/Decimal`	`Int`, das bitweise OR der beide Argumente Alle Zahlen, die nich `Int` sind, werden auf den nächs Wert abgerundet. Wenn die Kor fehlschlägt, ist das Ergebnis Un
`Int/Decimal/String`	`Int/Decimal/String`	`Int`, das bitweise OR für die be Argumente. Alle Zeichenfolgen in Dezimal-Werte konvertiert un nächsten `Int`-Wert abgerundet. Konvertierung fehlschlägt, ist da `Undefined` .
Anderer Wert	Anderer Wert	`Undefined` .

bitxor(Int, Int)

Führt Bit für Bit eine XOR-Maßnahme für die Bit-Darstellungen der beiden `Int`(-konvertierten) Argumente durch. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `bitor(13, 5)` = 8

Argumenttyp	Argumenttyp	Ergebnis
`Int`	`Int`	`Int`, ein bitweises XOR für die b Argumente
`Int/Decimal`	`Int/Decimal`	`Int`, ein bitweises XOR für die b Argumente Zahlen, die nicht vor sind, werden auf den nächsten abgerundet.
`Int/Decimal/String`	`Int/Decimal/String`	`Int`, ein bitweises XOR für die b Argumente. Zeichenfolgen werd Dezimal-Werte konvertiert und a nächsten `Int`-Wert abgerundet. Konvertierung fehlschlägt, ist da `Undefined` .
Anderer Wert	Anderer Wert	`Undefined` .

bitnot(Int)

Führt Bit für Bit einen NOT-Vorgang für die Bit-Darstellungen des `Int`(-konvertierten) Arguments durch. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `bitnot(13)` = 2

Argumenttyp	Ergebnis
`Int`	`Int`, ein bitweises NOT des Arguments.
`Decimal`	`Int`, ein bitweises NOT des Arguments. Der `Decimal`-Wert wird auf den nächsten `Int`-Wert abgerundet.
`String`	`Int`, ein bitweises NOT des Arguments. Zeichenfolgen werden in Dezimal-Werte konvertiert und auf den nächsten `Int`-

Argumenttyp	Ergebnis
	Wert abgerundet. Wenn eine Konvertierung fehlschlägt, ist das Ergebnis `Undefined` .
Anderer Wert	Anderer Wert.

cast()

Konvertiert einen Wert von einem Datentyp in einen anderen. Diese Umwandlung verhält sich größtenteils wie die Standardkonvertierungen. Zusätzlich verfügt sie jedoch über die Möglichkeit, Zahlen von oder zu booleschen Werten umzuwandeln. Wenn AWS IoT nicht bestimmt werden kann, wie ein Typ in einen anderen umgewandelt werden soll, lautet das Ergebnis`Undefined`. Unterstützt von SQL Version 2015-10-08 und höher. Format: Besetzung (*`value`*als*`type`*).

Beispiel:

```
cast(true as Int) = 1
```

Die folgenden Schlüsselwörter können beim Aufrufen von `cast` unter Umständen nach „as“ angezeigt werden:

Für SQL-Version 2015-10-08 und 2016-03-23

Stichwort	Ergebnis
`String`	Wandelt einen Wert in `String` um.
Nvarchar	Wandelt einen Wert in `String` um.
Text	Wandelt einen Wert in `String` um.
Ntext	Wandelt einen Wert in `String` um.
varchar	Wandelt einen Wert in `String` um.
`Int`	Wandelt einen Wert in `Int` um.
Ganzzahl	Wandelt einen Wert in `Int` um.

Stichwort	Ergebnis
Double	Überträgt den Wert auf `Decimal` (mit doppelter Genauigkeit).

Außerdem für SQL-Version 2016-03-23

Stichwort	Ergebnis
`Decimal`	Wandelt einen Wert in `Decimal` um.
Bool	Wandelt einen Wert in `Boolean` um.
`Boolean`	Wandelt einen Wert in `Boolean` um.

Umwandlungsregeln:

In Dezimalwert umwandeln

Argumenttyp	Ergebnis
`Int`	Ein Wert vom Typ `Decimal` ohne Dezimaltrennzeichen
`Decimal`	Der Quellwert **Note** Bei SQL V2 (23.03.2016) geben numerische Werte, bei denen es sich um ganze Zahlen handelt, z. B. `10.0` einen `Int` Wert (`10`) anstelle des erwarteten `Decimal` Werts (`10.0`) zurück. Um ganzzahlige numerische Werte zuverlässig in `Decimal`-Werte umzuwandeln,

Argumenttyp	Ergebnis
	verwenden Sie SQL V1 (2015-10-08) für die Regelabfrageanweisung.
`Boolean`	"true" = 1,0, "false" = 0,0
`String`	Versucht, die Zeichenfolge als zu analysieren. `Decimal` AWS IoT versucht, Zeichenketten zu analysieren, die der Regex entsprechen: ^-? \ d+ (\. \ d+)? ((? (i) E-? \ d+)? $. „0", „-1.2", „5E-12" sind alles Beispiele für Zeichenketten, die automatisch in Dezimalzahlen umgewandelt werden.
Array	`Undefined` .
Object	`Undefined` .
Null	`Undefined` .
Undefined	`Undefined` .

In Ganzzahl umwandeln

Argumenttyp	Ergebnis
`Int`	Der Quellwert
`Decimal`	Der Quellwert, auf den nächsten `Int`-Wert abgerundet
`Boolean`	"true" = 1,0, "false" = 0,0
`String`	Versucht, die Zeichenfolge als zu analysieren. `Decimal` AWS IoT versucht, Zeichenketten zu analysieren, die der Regex entsprechen: ^-? \ d+ (\. \ d+)? ((? (i) E-? \ d+)? $. „0", „-1.2", „5E-12" sind alles

Argumenttyp	Ergebnis
	Beispiele für Zeichenketten, die automatisch in Dezimalzahlen umgewandelt werden. AWS IoT versucht, die Zeichenfolge in eine Zahl umzuwandeln `Decimal` und auf die nächste Zahl abzurunden. `Int`
Array	`Undefined` .
Object	`Undefined` .
Null	`Undefined` .
Undefined	`Undefined` .

In einen **`booleschen Wert umwandeln`**

Argumenttyp	Ergebnis
`Int`	0 = false, alle Werte außer Null = true
`Decimal`	0 = false, alle Werte außer Null = true
`Boolean`	Der Quellwert
`String`	"true" = wahr und "false" = falsch (ohne Beachtung der Groß- und Kleinschreibung). Andere Zeichenfolgenwerte sind = `Undefined` .
Array	`Undefined` .
Object	`Undefined` .
Null	`Undefined` .
Undefined	`Undefined` .

In Zeichenfolge umwandeln

Argumenttyp	Ergebnis
`Int`	Eine Zeichenfolgendarstellung des `Int`-Werts in Standardnotation
`Decimal`	Eine Zeichenfolge, die den `Decimal`-Wert in Standardnotation darstellt
`Boolean`	"true" oder "false", in Kleinbuchstaben
`String`	Der Quellwert
Array	Das in JSON serialisierte Array. Die resultierende Zeichenfolge ist eine durch Kommata getrennte Liste in eckigen Klammern. `String`-Werte werden mit Anführungszeichen angegeben. Dies ist bei `Decimal`-, `Int`-, `Boolean`-Werten nicht der Fall.
Object	Das in JSON serialisierte Objekt. Die JSON-Zeichenfolge ist eine durch Kommata getrennte Liste von Schlüssel-Wert-Paa ren, die mit geschweiften Klammern beginnt und endet. `String`-Werte werden mit Anführungszeichen angegeben. Dies ist bei `Decimal`-, `Int`-, – `Boolean`und `Null`-Werten nicht der Fall.
Null	`Undefined` .
Undefined	`Undefined` .

ceil(Decimal)

Rundet den angegebenen `Decimal`-Wert auf den nächsten `Int`-Wert auf. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiele:

`ceil(1.2)` = 2

`ceil(-1.2)` = -1

Argumenttyp	Ergebnis
`Int`	`Int`, der Argumentwert
`Decimal`	`Int`, der `Decimal`-Wert, auf den nächsten `Int`-Wert gerundet
`String`	`Int`. Die Zeichenfolge wird in `Decimal` konvertiert und auf den nächsten `Int` aufgerundet. Wenn die Zeichenfolge nicht in einen `Decimal`-Wert konvertiert werden kann, ist das Ergebnis `Undefined` .
Anderer Wert	`Undefined` .

chr(String)

Gibt das ASCII-Zeichen zurück, das dem angegebenen `Int`-Argument entspricht. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiele:

`chr(65)` = "A".

`chr(49)` = "1".

Argumenttyp	Ergebnis
`Int`	Das Zeichen, das dem angegebenen ASCII-Wert entspricht. Wenn das Argument kein gültiger ASCII-Wert ist, ist das Ergebnis `Undefined` .

Argumenttyp	Ergebnis
`Decimal`	Das Zeichen, das dem angegebenen ASCII-Wert entspricht. Das `Decimal`-Argument wird auf den nächsten `Int`-Wert abgerundet. Wenn das Argument kein gültiger ASCII-Wert ist, ist das Ergebnis `Undefined` .
`Boolean`	`Undefined` .
`String`	Wenn der `String`-Wert nicht in einen `Decimal`-Wert konvertiert werden kann, wird er auf den nächsten `Int`-Wert abgerundet. Wenn das Argument kein gültiger ASCII-Wert ist, ist das Ergebnis `Undefined` .
Array	`Undefined` .
Object	`Undefined` .
Null	`Undefined` .
Anderer Wert	`Undefined` .

clientid()

Gibt die ID des MQTT-Clients zurück, der die Nachricht sendet, oder `n/a`, wenn die Nachricht nicht über MQTT gesendet wurde. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel:

```
clientid() = "123456789012"
```

concat()

Hängt Arrays oder Zeichenfolgen aneinander an. Diese Funktion akzeptiert eine beliebige Anzahl von Argumenten und gibt einen – `String`oder `Array`-Wert zurück. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiele:

`concat()` = `Undefined`.

`concat(1)` = "1".

`concat([1, 2, 3], 4)` = [1, 2, 3, 4].

`concat([1, 2, 3], "hello")` = [1, 2, 3, "hello"]

`concat("con", "cat")` = "concat"

`concat(1, "hello")` = "1hello"

`concat("he","is","man")` = "heisman"

`concat([1, 2, 3], "hello", [4, 5, 6])` = [1, 2, 3, "hello", 4, 5, 6]

Anzahl der Argumente	Ergebnis
0	`Undefined` .
1	Das Argument wird unverändert zurückgeg eben.
2+	Wenn ein Argument ein `Array` ist, ist das Ergebnis ein einzelnes Array, das alle Argumente enthält. Wenn kein Argument den Typ Array hat, und mindestens ein Argument den Typ `String` hat, ist das Ergebnis eine Verkettung der `String`-Darstellungen aller Argumente. Argumente werden mithilfe von oben genannten Standardkonvertierungen in Zeichenfolgen konvertiert.

cos(Decimal)

Gibt den Kosinus einer Zahl im Bogenmaß zurück. `Decimal`-Argumente werden vor dem Anwenden der Funktion auf doppelte Genauigkeit gerundet. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel:

`cos(0)` = 1.

Argumenttyp	Ergebnis
`Int`	`Decimal` (mit doppelter Genauigkeit), der Kosinus des Arguments. Imaginäre Ergebnisse werden als `Undefined` zurückgegeben.
`Decimal`	`Decimal` (mit doppelter Genauigkeit), der Kosinus des Arguments. Imaginäre Ergebnisse werden als `Undefined` zurückgegeben.
`Boolean`	`Undefined` .
`String`	`Decimal` (mit doppelter Genauigkeit), der Kosinus des Arguments. Wenn die Zeichenfolge nicht in einen `Decimal`-Wert konvertiert werden kann, ist das Ergebnis `Undefined` . Imaginäre Ergebnisse werden als `Undefined` zurückgegeben.
Array	`Undefined` .
Object	`Undefined` .
Null	`Undefined` .
Undefined	`Undefined` .

cosh(Decimal)

Gibt den hyperbolischen Kosinus einer Zahl im Bogenmaß zurück. `Decimal`-Argumente werden vor dem Anwenden der Funktion auf doppelte Genauigkeit gerundet. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `cosh(2.3)` = 5,037220649268761.

Argumenttyp	Ergebnis
`Int`	`Decimal` (mit doppelter Genauigkeit), der hyperbolische Kosinus des Arguments . Imaginäre Ergebnisse werden als `Undefined` zurückgegeben.
`Decimal`	`Decimal` (mit doppelter Genauigkeit), der hyperbolische Kosinus des Arguments . Imaginäre Ergebnisse werden als `Undefined` zurückgegeben.
`Boolean`	`Undefined` .
`String`	`Decimal` (mit doppelter Genauigkeit), der hyperbolische Kosinus des Arguments . Wenn die Zeichenfolge nicht in einen `Decimal`-Wert konvertiert werden kann, ist das Ergebnis `Undefined` . Imaginäre Ergebnisse werden als `Undefined` zurückgegeben.
Array	`Undefined` .
Object	`Undefined` .
Null	`Undefined` .
Undefined	`Undefined` .

decode(value, decodingScheme)

Verwenden Sie die `decode` Funktion, um einen kodierten Wert zu dekodieren. Wenn es sich bei der dekodierten Zeichenfolge um ein JSON-Dokument handelt, wird ein adressierbares Objekt zurückgegeben. Andernfalls wird die dekodierte Zeichenfolge als Zeichenfolge zurückgegeben. Die Funktion gibt NULL zurück, wenn die Zeichenfolge nicht dekodiert werden kann. Diese Funktion unterstützt die Dekodierung von Base64-kodierten Zeichenketten und das Nachrichtenformat Protocol Buffer (protobuf).

Unterstützt von der SQL Version vom 23.03.2016 und höher.

value

Ein Zeichenkettenwert oder einer der gültigen Ausdrücke, wie unter AWS IoT SQL-Referenz definiert, die eine Zeichenfolge zurückgeben.

decodingScheme

Eine Literalzeichenfolge, die das Schema darstellt, das zur Dekodierung des Wertes verwendet wurde. Aktuell werden nur `'base64'` und `'proto'` unterstützt.

Dekodierung von base64-verschlüsselten Strings

In diesem Beispiel enthält die Nachrichtennutzlast einen kodierten Wert.

```
{
    encoded_temp: "eyAidGVtcGVyYXR1cmUiOiAzMyB9Cg=="
}
```

Die `decode`-Funktion in dieser SQL-Anweisung dekodiert den Wert in der Nachrichtennutzlast.

```
SELECT decode(encoded_temp,"base64").temperature AS temp from 'topic/subtopic'
```

Die Dekodierung des `encoded_temp`-Wertes führt zu dem folgenden gültigen JSON-Dokument, das es der SELECT-Anweisung ermöglicht, den Temperaturwert zu lesen.

```
{ "temperature": 33 }
```

Das Ergebnis der SELECT-Anweisung in diesem Beispiel wird hier gezeigt.

```
{ "temp": 33 }
```

Ist der dekodierte Wert kein gültiges JSON-Dokument, wird der dekodierte Wert als Zeichenfolge zurückgegeben.

Payloads der protobuf-Nachrichten entschlüsseln

Sie können SQL-Funktion dekodieren, um eine Regel zu konfigurieren, die die protobuf-Nachrichtennutzlast dekodieren kann. Weitere Informationen finden Sie unter Payloads von protobuf-Nachrichten dekodieren.

> **Important**
>
> Wenn Sie `source-account` bei der Festlegung von Berechtigungen für einen AWS IoT Prinzipal das `source-arn` oder weglassen, AWS-Konto kann jeder Ihre Decode-Funktion über andere Regeln aufrufen. AWS IoT Informationen zur Sicherung Ihrer Funktion finden Sie unter Bucket-Richtlinien im Amazon Simple Storage Service-Benutzerhandbuch.

Die Signatur der Funktion sieht wie folgt aus:

```
decode(<ENCODED DATA>, 'proto', '<S3 BUCKET NAME>', '<S3 OBJECT KEY>', '<PROTO NAME>',
 '<MESSAGE TYPE>')
```

`ENCODED DATA`

Gibt die protobuf-kodierten Daten an, die dekodiert werden sollen. Wenn es sich bei der gesamten an die Regel gesendeten Nachricht um protobuf-kodierte Daten handelt, können Sie die eingehende rohe binäre Nutzlast mit * referenzieren. Andernfalls muss es sich bei diesem Feld um eine Base-64-kodierte JSON-Zeichenfolge handeln, und ein Verweis auf die Zeichenfolge kann direkt übergeben werden.

1) Um eine eingehende rohe binäre protobuf-Nutzlast zu dekodieren:

```
decode(*, 'proto', ...)
```

2) Um eine protobuf-kodierte Nachricht zu dekodieren, die durch eine Base64-kodierte Zeichenfolge 'a.b' dargestellt wird:

```
decode(a.b, 'proto', ...)
```

`proto`

Spezifiziert die zu dekodierenden Daten in einem protobuf-Nachrichtenformat. Wenn Sie `base64` statt `proto` angeben, dekodiert diese Funktion Base64-kodierte Zeichenketten als JSON.

`S3 BUCKET NAME`

Der Name des Amazon S3 Buckets, in den Sie die `FileDescriptorSet`-Datei hochgeladen haben.

`S3 OBJECT KEY`

Der Objektschlüssel, das die `FileDescriptorSet`-Datei im Amazon S3 S3-Bucket spezifiziert.

`PROTO NAME`

Der Name der `.proto`-Datei (ohne Erweiterung), aus der die `FileDescriptorSet`-Datei generiert wurde.

`MESSAGE TYPE`

Der Name der protobuf-Nachrichtenstruktur innerhalb der `FileDescriptorSet`-Datei, der die zu dekodierenden Daten entsprechen sollen.

Ein Beispiel für einen SQL-Ausdruck, der die SQL-Funktion decode verwendet, kann wie folgt aussehen:

```
SELECT VALUE decode(*, 'proto', 's3-bucket', 'messageformat.desc', 'myproto',
 'messagetype') FROM 'some/topic'
```

- `*`

  Stellt eine binäre eingehende Nutzlast dar, die dem aufgerufenen protobuf-Nachrichtentyp mit dem Namen `mymessagetypeen`tspricht.

- `messageformat.desc`

  Die in einem Amazon S3 S3-Bucket gespeicherte `FileDescriptorSet`-Datei mit dem Namen `s3-bucket`.

- `myproto`

Die `.proto`-Originaldatei, die zur Generierung der `FileDescriptorSet`-Datei mit dem Namen `myproto.proto` verwendet wurde.

- `messagetype`

  Der aufgerufene Nachrichtentyp `messagetype` (zusammen mit allen importierten Abhängigkeiten), wie in definiert`myproto.proto`.

encode(value, encodingScheme)

Verwenden Sie die Funktion `encode`, um die Nutzlast, bei der es sich möglicherweise um Nicht-JSON-Daten handelt, auf der Grundlage des Kodierungsschemas in ihre String-Darstellung zu kodieren. Unterstützt von der SQL Version vom 23.03.2016 und höher.

value

Einer der gültigen Ausdrücke, wie in AWS IoT SQL-Referenz definiert. Sie können „*“ angeben, um die gesamte Nutzlast zu verschlüsseln, unabhängig davon, ob sie das JSON-Format oder ein anderes Format aufweist. Wenn Sie einen Ausdruck angeben, wird das Ergebnis der Auswertung vor dem Verschlüsseln in eine Zeichenfolge umgewandelt.

encodingScheme

Eine Literalzeichenfolge, die das zu verwendende Verschlüsselungsschema darstellt. Derzeit wird nur `'base64'` unterstützt.

endswith(String, String)

Gibt einen Wert vom Typ `Boolean` zurück, der angibt, ob das erste `String`-Argument mit dem zweiten `String`-Argument endet. Wenn ein Argument `Null` oder `Undefined` ist, ist das Ergebnis `Undefined`. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `endswith("cat","at")` = true.

Argumenttyp 1	Argumenttyp 2	Ergebnis
`String`	`String`	"True", wenn das erste Argumer zweite Argument endet. Ansons

Argumenttyp 1	Argumenttyp 2	Ergebnis
Anderer Wert	Anderer Wert	Beide Argumente werden mithilf Standardkonvertierungsregeln z lgen konvertiert. "True", wenn da Argument auf das zweite Argum Ansonsten „false“. Wenn ein Arg `Null` oder `Undefined` ist, ist Ergebnis `Undefined` .

exp(Decimal)

Gibt "e" zurück, potenziert mit dem `Decimal`-Argument. `Decimal`-Argumente werden vor dem Anwenden der Funktion auf doppelte Genauigkeit gerundet. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `exp(1)` = e.

Argumenttyp	Ergebnis
`Int`	`Decimal` (mit doppelter Genauigkeit), e ^ Argument.
`Decimal`	`Decimal` (mit doppelter Genauigkeit), e ^ Argument.
`String`	`Decimal` (mit doppelter Genauigkeit), e ^ Argument. Wenn der `String`-Wert nicht in einen `Decimal`-Wert konvertiert werden kann, ist das Ergebnis `Undefined` .
Anderer Wert	`Undefined` .

floor(Decimal)

Rundet den angegebenen `Decimal`-Wert auf den nächsten `Int`-Wert ab. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiele:

`floor(1.2)` = 1

`floor(-1.2)` = -2

Argumenttyp	Ergebnis
`Int`	`Int`, der Argumentwert
`Decimal`	`Int`, der auf den nächsten `Int` abgerunde te `Decimal`-Wert.
`String`	`Int`. Die Zeichenfolge wird konvertiert in `Decimal` und auf den nächsten `Int`-Wert abgerundet. Wenn die Zeichenfolge nicht in einen `Decimal`-Wert konvertiert werden kann, ist das Ergebnis `Undefined` .
Anderer Wert	`Undefined` .

get

Extrahiert einen Wert aus einem sammlungsartigen Typ (Array, Zeichenfolge, Objekt). Auf das erste Argument wird keine Konvertierung angewendet. Die Konvertierung wird wie in der Tabelle dokumentiert auf das zweite Argument angewendet. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiele:

`get(["a", "b", "c"], 1)` = "b"

`get({"a":"b"}, "a")` = "b"

`get("abc", 0)` = „a“.

Argumenttyp 1	Argumenttyp 2	Ergebnis
Array	Beliebiger Typ (konvertiert in `Int`)	Das Element am 0-basierten Ind `Array`-Werts, der vom zweiten

Argumenttyp 1	Argumenttyp 2	Ergebnis
		angegeben wird (konvertiert in I Wenn die Konvertierung fehlsch das Ergebnis `Undefined` . We der Index außerhalb von `Array` (negativ oder >= array.length), i Ergebnis `Undefined` .
Zeichenfolge	Beliebiger Typ (konvertiert in `Int`)	Das Zeichen am 0-basierten Ind Zeichenfolge, der vom zweiten A angegeben wird (konvertiert in I die Konvertierung fehlschlägt, is Ergebnis `Undefined` . Wenn s Index außerhalb der Zeichenfolg (negativ oder >= string.length), i Ergebnis `Undefined` .
Object	`String` (keine Konvertierung wird angewendet)	Der im ersten Argumentobjekt g rte Wert, der dem Zeichenfolger ssel entspricht, der als zweites A angegeben wurde.
Anderer Wert	Beliebiger Wert	`Undefined` .

get_dynamodb (tableName, Partition, Partition, Sortierung KeyName KeyValueKeyName, roleArn) KeyValue

Ruft Daten aus einer DynamoDB-Tabelle ab. `get_dynamodb()` ermöglicht es Ihnen, eine DynamoDB-Tabelle abzufragen, während eine Regel evaluiert wird. Sie können den Nachrichten-Payload mit Hilfe von Daten aus DynamoDB filtern oder erweitern. Unterstützt von der SQL Version vom 23.03.2016 und höher.

`get_dynamodb()` verwendet folgenden Parameter:

tableName

Der Name der DynamoDB-Tabelle für die Abfrage.

Partition KeyName

Der Name des Partitionsschlüssels. Weitere Informationen finden Sie unter DynamoDB Keys.

Partition KeyValue

Der Wert des Partitionsschlüssels, der zur Identifizierung eines Datensatzes verwendet wird. Weitere Informationen finden Sie unter DynamoDB Keys.

sortieren KeyName

(Optional) Der Name des Sortierschlüssels. Dieser Parameter ist nur erforderlich, wenn die abgefragte DynamoDB-Tabelle einen zusammengesetzten Schlüssel verwendet. Weitere Informationen finden Sie unter DynamoDB Keys.

sortieren KeyValue

Der Wert des Sortierschlüssels. Dieser Parameter ist nur erforderlich, wenn die abgefragte DynamoDB-Tabelle einen zusammengesetzten Schlüssel verwendet. Weitere Informationen finden Sie unter DynamoDB Keys.

roleArn

Der ARN der IAM-Rolle, der den Zugriff auf die DynamoDB-Tabelle gewährt. Die Regel-Engine übernimmt diese Rolle, um in Ihrem Namen auf die DynamoDB-Tabelle zuzugreifen. Vermeiden Sie die Verwendung einer zu großzügigen Rolle. Erteilen Sie der Rolle nur die von der Regel benötigten Berechtigungen. Nachfolgend finden Sie eine Beispielrichtlinie, die Zugriff auf eine DynamoDB-Tabelle gewährt.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:GetItem",
            "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/table-
name"
        }
    ]
}
```

Als Beispiel dafür, wie Sie `get_dynamodb()` verwenden können, nehmen wir an, Sie haben eine DynamoDB-Tabelle, die Geräte-ID- und Standortinformationen für alle mit AWS IoT verbundenen

Geräte enthält. Die folgende SELECT-Anweisung verwendet die `get_dynamodb()`-Funktion, um den Standort für die angegebene Geräte-ID abzurufen:

```
SELECT *, get_dynamodb("InServiceDevices", "deviceId", id,
"arn:aws:iam::12345678910:role/getdynamo").location AS location FROM 'some/
topic'
```

> **Note**
>
> - Sie können `get_dynamodb()` maximal einmal pro SQL-Anweisung aufrufen. Der mehrfache Aufruf von `get_dynamodb()` in einer einzigen SQL-Anweisung führt dazu, dass die Regel ohne Aufruf von Aktionen beendet wird.

get_mqtt_property(name)

Verweist auf einen der folgenden MQTT5-Header: `contentType`, `payLoadFormatIndicator`, `responseTopic` und `correlationData` Diese Funktion verwendet eine der folgenden Literalzeichenfolgen als Argument:`content_type`, `format_indicator`. `response_topic` und `correlation_data` Weitere Informationen in der folgenden Tabelle mit Funktionsargumenten.

contentType

Zeichenfolge: Eine UTF-8 kodierte Zeichenfolge, die den Inhalt der Veröffentlichungsnachricht beschreibt.

zahlen LoadFormatIndicator

Zeichenfolge: Ein Enum-Zeichenkettenwert, der angibt, ob die Nutzlast als formatiert ist. UTF-8 Gültige Werte sind `UNSPECIFIED_BYTES` und `UTF8_DATA`.

responseTopic

Zeichenfolge: Eine UTF-8 codierte Zeichenfolge, die als Themenname für eine Antwortnachricht verwendet wird. Das Antwortthema wird verwendet, um das Thema zu beschreiben, das der Empfänger im Rahmen des Ablaufs Anforderung-Antwort veröffentlichen soll. Das Thema darf keine Platzhalterzeichen enthalten.

correlationData

Zeichenfolge: Die base64-kodierten Binärdaten, die vom Absender der Request Message verwendet werden, um zu identifizieren, für welche Anforderung die Response Message bestimmt ist, wenn sie empfangen wird.

Die folgende Tabelle zeigt die zulässigen Funktionsargumente und die zugehörigen Rückgabetypen für die `get_mqtt_property`-Funktion:

Funktionsargumente

SQL	Zurückgegebener Datentyp (falls vorhanden)	Zurückgegebener Datentyp (falls vorhanden)
`get_mqtt_property( "format_indicator")`	Zeichenfolge (UNSPECIF IED_BYTES oder UTF8_DATA)	Zeichenfolge (UNSPECIF IED_BYTES)
`get_mqtt_property( "content_type")`	Zeichenfolge	Undefined
`get_mqtt_property( "response_topic")`	Zeichenfolge	Undefined
`get_mqtt_property( "correlation_data")`	base64-kodierte Zeichenfo lge	Undefined
`get_mqtt_property( "some_invalid_name ")`	Undefined	Undefined

Das folgende Beispiel für Rules SQL verweist auf einen der folgenden MQTT5-Header:,, und. `contentType` `payLoadFormatIndicator` `responseTopic` `correlationData`

```
SELECT *, get_mqtt_property('content_type') as contentType,
          get_mqtt_property('format_indicator') as payloadFormatIndicator,
          get_mqtt_property('response_topic') as responseTopic,
          get_mqtt_property('correlation_data') as correlationData
FROM 'some/topic'
```

get_or_default (Ausdruck, defaultValue)

Gibt den Standardwert im zweiten Parameter zurück, falls angegeben, oder gibt undefined zurück, wenn der Ausdruck im ersten Parameter Null, undefiniert zurückgibt oder fehlschlägt. Unterstützt von der SQL Version vom 23.03.2016 und höher.

> **Important**
>
> `get_or_default`unterstützt Payloads, die nicht als JSON gelten, nicht direkt. Wenn Sie eine Nicht-JSON-Nutzlast verwenden, verwenden Sie die Funktionen oder. `encode` `decode`

`get_or_default()` verwendet folgenden Parameter:

Ausdruck

Jeder gültige AusdruckDatentypen,,, Variablen FunktionenLiterale,, oder enthält. Verschachtelte Objektabfragen JSON-Erweiterungen

defaultValue

(Optional) Jeder gültige AusdruckDatentypen, derFunktionen,Literale, VariablenVerschachtelte Objektabfragen, oder enthältJSON-Erweiterungen. Dies ist der Wert, der immer dann zurückgegeben werden muss, wenn das erste Argument Null oder undefiniert zurückgibt oder fehlschlägt.

> **Note**
>
> Funktionen, die Daten aus kundeneigenen Ressourcen abrufen, wie get_secret, get_dynamodb, aws_lambda, get_thing_shadow und decode-protobuf, sind für den DefaultValue-Parameter nicht zulässig.

Die folgende Tabelle zeigt akzeptable Funktionsargumente für jedes Argument und die zugehörigen Ausgaben:

Erstes Argument	Zweites Argument	Ausgabe
Erfolgreiche Bewertung	Beliebiger Wert oder nicht angegeben	Der erste Argumentwert.

Erstes Argument	Zweites Argument	Ausgabe
Undefiniert, Null oder Fehler	Jeder Wert, einschließlich Undefiniert oder Null	Der zweite Argumentwert.
Undefiniert, Null oder Fehler	nicht angegeben	`Undefined`

Beispiele:

Beispiel 1:

Das folgende Beispiel stellt einen DefaultValue-Wert bereit, falls eine DynamoDB-Tabelle oder -Abfrage fehlschlägt:

```
SELECT
    device_id,
    get_or_default(
        get_dynamodb("DeviceConfig", "deviceId", nonExistentId,
 "arn:aws:iam::123456789012:role/ROLE_NAME"),
        {"mode": "standard", "timeout": 30, "enabled": true }
    ) as config
FROM 'device/telemetry'
```

Beispiel 2:

Das folgende Beispiel stellt den sicheren Standardwert „UNKNOWN“ bereit, wenn der Status undefiniert ist:

```
SELECT
  get_or_default( CASE status
    WHEN 'active' THEN 'GOOD'
    WHEN 'inactive' THEN 'BAD'/
    ELSE 'UNKNOWN'
  END, 'UNKNOWN') as status_category
FROM 'topic/subtopic'
```

Beispiel 3:

Das folgende Beispiel zeigt, wie Sie get_or_default auch mit einem einzigen Parameter verwenden können. Dies ist nützlich in Szenarien, in denen Sie möglicherweise keinen eindeutigen Standardwert haben, Sie aber nicht möchten, dass Ihre Regelausführung fehlschlägt.

```
SELECT
  get_dynamodb("DeviceConfig", "deviceId", nonExistentId,
 "arn:aws:iam::123456789012:role/ROLE_NAME") as config
FROM 'device/telemetry'
```

Wenn die DynamoDB-Suche fehlschlägt, schlägt die Regelausführung fehl und es werden keine Aktionen ausgeführt. Wenn stattdessen das folgende SQL verwendet wird:

```
SELECT
  get_or_default(get_dynamodb("DeviceConfig", "deviceId", nonExistentId,
 "arn:aws:iam::123456789012:role/ROLE_NAME")) as config
FROM 'device/telemetry'
```

Die get_or_default-Anweisung wird als ausgewertet. In diesem Beispiel wird also die SELECT-Anweisung insgesamt als ausgewertet `{}` und es werden alle Regelaktionen versucht. `Undefined`

> **Important**
>
> Wir empfehlen, die folgenden bewährten Methoden zu befolgen, um die Sicherheit bei der Verwendung dieser Funktion zu gewährleisten:
>
> - Vermeiden Sie die Verwendung von hartcodierten Geheimnissen in Regeldefinitionen, einschließlich Standardwerten
> - Wird AWS Secrets Manager für die Verwaltung vertraulicher Informationen verwendet

get_registry_data (Registrierungs-API, thingName, RoleRN)

Ruft Ding-Registrierungsdaten in einer Regel ab. AWS IoT AWS IoT Sie können Registrierungsdaten (wie Attribute, Dingtyp und Dinggruppen, zu denen ein Gerät gehört) lesen und diese Informationen verwenden, um Nachrichten zu filtern, anzureichern oder dynamisch weiterzuleiten. Unterstützt von der SQL Version vom 23.03.2016 und höher.

`get_registry_data()` verwendet folgenden Parameter:

Registrierungs-API

Die Registrierungs-API wird aufgerufen. Gültige Werte sind `DescribeThing` und `ListThingGroupsForThing`. Bei diesen Werten muss es sich um konstante Zeichenketten handeln.

thingName

Zeichenfolge: Der Name der Sache, deren Registrierungsdaten Sie abrufen möchten.

roleArn

Zeichenfolge: Ein Rollen-ARN mit `iot:DescribeThing` and/or `iot:ListThingGroupsForThing` Berechtigungsberechtigungen, die auf der aufgerufenen API basieren.

Das Antwortformat der `get_registry_data` Funktion entspricht dem der aufgerufenen Registrierungs-API. Weitere Informationen finden Sie unter den ListThingGroupsForThingAPIs DescribeThingund.

Beispiel:

Sie können Informationen zum Dingtyp abrufen, um die AWS IoT Core Lebenszyklusereignismeldungen nach Dingen filtern zu können (wobei der Name des Dings mit der MQTT-Client-ID übereinstimmt), für die sich der Typ des Dings befindet`testenv`.

```
SELECT *
FROM '$aws/events/lifecycle/+'
WHERE
    get_registry_data("DescribeThing",clientId,[roleArn]).thingTypeName='testenv'
```

Beispiel:

Sie können die Ding-Attribute für ein Gerät mit dem Ding-Namen `sensor1` für alle Nachrichten abrufen, die von seinem Gateway-Gerät `gateway1` gesendet wurden.

```
SELECT *, get_registry_data("DescribeThing","sensor1",
[roleArn]).attributes.temperature_threhold AS device1_tempthreshold
FROM home1/gateway1/sensor1/#
```

> **Note**
>
> - Sie können `get_registry_data()` maximal einmal pro SQL-Anweisung und Ersatzvorlagen für Aktionen und Fehleraktionen aufrufen.

get_secret (secretId, geheimer Typ, Schlüssel, roleArn)

Ruft den Wert des verschlüsselten `SecretString` oder `SecretBinary`-Feldes der aktuellen Version eines Secrets in AWS Secrets Manager ab. Weitere Hinweise zum Erstellen und Verwalten von Geheimnissen finden Sie unter CreateSecretUpdateSecret, und PutSecretValue.

`get_secret()` verwendet folgenden Parameter:

secretId

Zeichenfolge: Der Amazon-Ressourcenname (ARN) oder der Anzeigename des Secrets, das abgerufen werden soll.

SecretType

Zeichenfolge: Der Secret-Typ. Zulässige Werte: `SecretString` | `SecretBinary`.

SecretString

- Informationen zu Geheimnissen, die Sie als JSON-Objekte mithilfe der APIs AWS CLI, der oder der AWS Secrets Manager Konsole erstellen:
  - Wenn Sie einen Wert für den `key`-Parameter angeben, gibt diese Funktion den Wert des angegebenen Schlüssels zurück.
  - Wenn Sie keinen Wert für den `key`-Parameter angeben, gibt diese Funktion das gesamte JSON-Objekt zurück.
- Für Geheimnisse, die Sie als Nicht-JSON-Objekte erstellen, indem Sie die APIs verwenden oder: AWS CLI
  - Wenn Sie einen Wert für den `key`-Parameter angeben, schlägt diese Funktion mit einer Ausnahme fehl.
  - Wenn Sie keinen Wert für den `key`-Parameter angeben, gibt diese Funktion den Inhalt des Secrets zurück.

SecretBinary

- Wenn Sie einen Wert für den `key`-Parameter angeben, schlägt diese Funktion mit einer Ausnahme fehl.
- Wenn Sie keinen Wert für den `key` Parameter angeben, gibt diese Funktion den geheimen Wert als Base64-kodierte UTF-8 Zeichenfolge zurück.

Schlüssel

(Optional) Zeichenfolge: Der Schlüsselname in einem JSON-Objekt, das im `SecretString`-Feld eines Geheimnisses gespeichert ist. Verwenden Sie diesen Wert, wenn Sie statt des gesamten JSON-Objekts nur den Wert eines in einem geheimen Schlüssel gespeicherten Schlüssels abrufen möchten.

Wenn Sie einen Wert für diesen Parameter angeben und das Geheimnis kein JSON-Objekt in seinem `SecretString`-Feld enthält, schlägt diese Funktion mit einer Ausnahme fehl.

roleArn

Zeichenfolge: Eine Rollen-ARN mit den Berechtigungen `secretsmanager:GetSecretValue` und `secretsmanager:DescribeSecret`.

> **Note**
>
> Diese Funktion gibt immer die aktuelle Version des Geheimnisses zurück (die Version mit dem `AWSCURRENT` Tag). Die AWS IoT Regel-Engine speichert jedes Geheimnis bis zu 15 Minuten lang im Cache. Daher kann es bis zu 15 Minuten dauern, bis die Regel-Engine ein Geheimnis aktualisiert. Das heißt, wenn Sie ein Geheimnis bis zu 15 Minuten nach einem Update mit abrufen AWS Secrets Manager, gibt diese Funktion möglicherweise die vorherige Version zurück.
> Diese Funktion ist nicht kostenpflichtig, es AWS Secrets Manager fallen jedoch Gebühren an. Aufgrund des geheimen Caching-Mechanismus ruft die Regel-Engine gelegentlich AWS Secrets Manager auf. Da es sich bei der Regel-Engine um einen vollständig verteilten Dienst handelt, können Sie während des 15-minütigen Caching-Fensters mehrere Secrets Manager Manager-API-Aufrufe von der Regel-Engine sehen.

Beispiele:

Sie können die `get_secret`-Funktion in einem Authentifizierungsheader in einer HTTPS-Regelaktion verwenden, wie im folgenden Beispiel für die API-Schlüsselauthentifizierung.

```
"API_KEY": "${get_secret('API_KEY', 'SecretString', 'API_KEY_VALUE',
 'arn:aws:iam::12345678910:role/getsecret')}"
```

Weitere Informationen zu Regelaktionen finden Sie unter the section called “HTTP”.

get_thing_shadow (DingName, shadowName, roleArn)

Gibt den angegebenen Schatten des angegebenen Geräts zurück. Unterstützt von der SQL Version vom 23.03.2016 und höher.

thingName

Zeichenfolge: Der Name des Geräts, dessen Schatten Sie abrufen möchten.

shadowName

(Optionale) Zeichenfolge: Der Name des Schattens. Dieser Parameter ist nur erforderlich, wenn auf benannte Schatten verwiesen wird.

roleArn

Zeichenfolge: Eine Rollen-ARN mit der Berechtigung `iot:GetThingShadow`.

Beispiele:

Geben Sie bei Verwendung mit einem benannten Schatten den `shadowName`-Parameter an.

```
SELECT * from 'topic/subtopic'
WHERE
    get_thing_shadow("MyThing","MyThingShadow","arn:aws:iam::123456789012:role/
AllowsThingShadowAccess")
    .state.reported.alarm = 'ON'
```

Lassen Sie bei Verwendung mit einem benannten Schatten den `shadowName`-Parameter aus.

```
SELECT * from 'topic/subtopic'
```

```
WHERE
    get_thing_shadow("MyThing","arn:aws:iam::123456789012:role/
AllowsThingShadowAccess")
    .state.reported.alarm = 'ON'
```

get_user_properties PropertyKey (Benutzer)

Verweist auf Benutzereigenschaften, eine Art von Eigenschaftsheadern, die in MQTT5 unterstützt werden.

userProperty

Zeichenfolge: Eine Benutzereigenschaft ist ein Schlüssel-Wert-Paar. Diese Funktion verwendet den Schlüssel als Argument und gibt ein Array mit allen Werten zurück, die dem zugehörigen Schlüssel entsprechen.

Funktionsargumente

Für die folgenden Benutzereigenschaften in den Nachrichtenkopfzeilen:

Key (Schlüssel)	Value (Wert)
ein Schlüssel	ein Wert
ein anderer Schlüssel	ein anderer Wert
ein Schlüssel	Wert mit doppeltem Schlüssel

Die folgende Tabelle zeigt das erwartete SQL-Verhalten:

SQL	Ein Rückgabedatentyp	Zurückgegebener Datenwert
get_user_properties ('irgendein Schlüssel')	Zeichenfolgen-Array	`['some value', 'value with duplicate key']`
get_user_properties ('anderer Schlüssel')	Zeichenfolgen-Array	`['a different value']`

SQL	Ein Rückgabedatentyp	Zurückgegebener Datenwert
get_user_properties ()	Array von Schlüssel-Wert-Paar-Objekten	`[{'"some key": "some value"'}, {"other key": "a different value"}, {"some key": "value with duplicate key"}]`
get_user_properties ('Schlüssel nicht vorhandener Schlüssel')	Undefined	

Das folgende Beispiel für Rules SQL referenziert Benutzereigenschaften (eine Art MQTT5-Eigenschaftsheader) in die Nutzlast:

```
SELECT *, get_user_properties('user defined property key') as userProperty
FROM 'some/topic'
```

Hashfunktionen

AWS IoT bietet die folgenden Hashing-Funktionen:

- md2
- md5
- sha1
- sha224
- sha256
- sha384
- sha512

Alle Hashfunktionen erwarten ein Zeichenfolgenargument. Das Ergebnis ist der Hashwert dieser Zeichenfolge. Standard-Zeichenfolgenkonvertierungen gelten für alle Argumente, die keine Zeichenfolgen sind. Alle Hashfunktionen werden von der SQL Version 2015-10-08 und höher unterstützt.

Beispiele:

`md2("hello")` = "a9046c73e00331af68917d3804f70655"

`md5("hello")` = "5d41402abc4b2a76b9719d911017c592"

indexof(String, String)

Gibt den ersten Index (0-basiert) des zweiten Arguments als Teilzeichenfolge des ersten Arguments zurück. Beide Argumente werden als Zeichenfolgen erwartet. Argumente, die keine Zeichenfolgen sind, unterliegen den Standardregeln für die Zeichenfolgenkonvertierung. Diese Funktion gilt nicht für Arrays, nur für Zeichenfolgen. Unterstützt von der SQL Version vom 23.03.2016 und höher.

Beispiele:

`indexof("abcd", "bc")` = 1

isNull()

Gibt „true“ zurück, wenn das Argument der Wert `Null` ist. Unterstützt von der SQL Version vom 23.03.2016 und höher.

Beispiele:

`isNull(5)` = false.

`isNull(Null)` = true.

Argumenttyp	Ergebnis
`Int`	false
`Decimal`	false
`Boolean`	false
`String`	false
`Array`	false
`Object`	false
`Null`	true
`Undefined`	false

isUndefined()

Gibt „true“ zurück, wenn das Argument `Undefined` ist. Unterstützt von der SQL Version vom 23.03.2016 und höher.

Beispiele:

`isUndefined(5)` = false.

`isUndefined(floor([1,2,3]))` = true.

Argumenttyp	Ergebnis
`Int`	false
`Decimal`	false
`Boolean`	false
`String`	false
`Array`	false
`Object`	false
`Null`	false
`Undefined`	true

length(String)

Gibt die Anzahl der Zeichen in der angegebenen Zeichenfolge zurück. Für andere Argumente als `String` gelten Standardkonvertierungsregeln. Unterstützt von der SQL Version vom 23.03.2016 und höher.

Beispiele:

`length("hi")` = 2

`length(false)` = 5

ln(Decimal)

Gibt den natürlichen Logarithmus des Arguments zurück. `Decimal`-Argumente werden vor dem Anwenden der Funktion auf doppelte Genauigkeit gerundet. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `ln(e)` = 1.

Argumenttyp	Ergebnis
`Int`	`Decimal` (mit doppelter Genauigkeit), der natürliche Logarithmus des Arguments
`Decimal`	`Decimal` (mit doppelter Genauigkeit), der natürliche Logarithmus des Arguments
`Boolean`	`Undefined` .
`String`	`Decimal` (mit doppelter Genauigkeit), der natürliche Logarithmus des Arguments Wenn die Zeichenfolge nicht in einen `Decimal`-Wert konvertiert werden kann, ist das Ergebnis `Undefined` .
Array	`Undefined` .
Object	`Undefined` .
Null	`Undefined` .
Undefined	`Undefined` .

log(Decimal)

Gibt den Logarithmus des Arguments zur Basis 10 zurück. `Decimal`-Argumente werden vor dem Anwenden der Funktion auf doppelte Genauigkeit gerundet. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `log(100)` = 2.0.

Argumenttyp	Ergebnis
`Int`	`Decimal` (mit doppelter Genauigkeit), der Logarithmus des Arguments zur Basis 10
`Decimal`	`Decimal` (mit doppelter Genauigkeit), der Logarithmus des Arguments zur Basis 10
`Boolean`	`Undefined` .
`String`	`Decimal` (mit doppelter Genauigkeit), der Logarithmus des Arguments zur Basis 10 Wenn der `String`-Wert nicht in einen `Decimal`-Wert konvertiert werden kann, ist das Ergebnis `Undefined` .
Array	`Undefined` .
Object	`Undefined` .
Null	`Undefined` .
Undefined	`Undefined` .

lower(String)

Gibt die Kleinbuchstabenversion des angegebenen Objekts zurück. `String` Non-stringArgumente werden unter Verwendung der Standardkonvertierungsregeln in Zeichenketten umgewandelt. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiele:

`lower("HELLO")` = „hello“

`lower(["HELLO"])` = "[\"hello\"]"

lpad(String, Int)

Gibt das Argument `String` zurück, das auf der linken Seite mit der vom zweiten Argument festgelegten Anzahl an Leerzeichen aufgefüllt wurde. Das Argument `Int` muss zwischen 0 und 1000

liegen. Wenn der angegebene Wert außerhalb des gültigen Bereichs liegt, wird das Argument auf den nächsten gültigen Wert (0 oder 1000) festgelegt. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiele:

`lpad("hello", 2) = "  hello"`.

`lpad(1, 3) = "   1"`

Argumenttyp 1	Argumenttyp 2	Ergebnis
`String`	`Int`	`String`, der angegebene `Stri` der auf der linken Seite mit der A an Leerzeichen aufgefüllt wurde angegebenen `Int`-Wert entspri
`String`	`Decimal`	Das `Decimal`-Argument wird au nächsten `Int`-Wert abgerundet `String` wird links mit der angeg Anzahl an Leerzeichen aufgefül
`String`	`String`	Das zweite Argument wird in ein `Decimal`-Wert konvertiert, der a nächsten `Int`-Wert abgerundet und der `String`-Wert wird links angegebenen Anzahl an Leerze aufgefüllt. Wenn das zweite Arg in einen `Int`-Wert konvertiert w ist das Ergebnis `Undefined` .
Anderer Wert	`Int`/`Decimal`/`String`	Der erste Wert wird mit der Star onvertierungen in `String` konv anschließend wird die LPAD-Fu diesen `String` angewendet. W konvertiert werden kann, ist das `Undefined` .
Beliebiger Wert	Anderer Wert	`Undefined` .

ltrim(String)

Entfernt alle führenden Leerzeichen (Tabulatoren und Leerzeichen) aus dem angegebenen `String`-Wert. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel:

`Ltrim(" h i ")` = "hi "

Argumenttyp	Ergebnis
`Int`	Die `String`-Darstellung des `Int`-Werts nach dem Entfernen der führenden Leerzeichen.
`Decimal`	Die `String`-Darstellung des `Decimal`-Werts nach dem Entfernen der führenden Leerzeichen.
`Boolean`	Die `String`-Darstellung des boolesche n Werts („true“ oder „false“) nach dem Entfernen der führenden Leerzeichen.
`String`	Das Argument nach dem Entfernen der führenden Leerzeichen.
Array	Die `String`-Darstellung dvon es `Array` (mit Standardkonvertierungsregeln) nach dem Entfernen der führenden Leerzeichen.
Object	Die `String`-Darstellung des Objekts (mit Standardkonvertierungsregeln) nach dem Entfernen der führenden Leerzeichen.
Null	`Undefined` .
Undefined	`Undefined` .

mod(Decimal, Decimal)

Gibt den Rest zurück, der beim Teilen des ersten Arguments durch das zweite Argument entstanden ist. Äquivalent mit remainder(Decimal, Decimal). Sie können auch "%" als infix-Operator für die gleiche Modulfunktionalität verwenden. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `mod(8, 3)` = 2

Left operator	Right operator	Output
`Int`	`Int`	`Int`, das erste Argumentmodulo zweiten Arguments
`Int/Decimal`	`Int/Decimal`	`Decimal`, das erste Argumentm zweiten Operanden
`String/Int/Decimal`	`String/Int/Decimal`	Wenn alle Zeichenfolgen in Dez erte konvertiert werden, ist das das erste Argumentmodulo des Arguments. Andernfalls `Undefi`
Anderer Wert	Anderer Wert	`Undefined` .

nanvl (AnyValue, AnyValue)

Gibt das erste Argument zurück, wenn es ein gültiger `Decimal`-Wert ist. Andernfalls wird das zweite Argument zurückgegeben. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `Nanvl(8, 3)` = 8.

Argumenttyp 1	Argumenttyp 2	Ausgabe
Undefined	Beliebiger Wert	Das zweite Argument
Null	Beliebiger Wert	Das zweite Argument
`Decimal` (NaN)	Beliebiger Wert	Das zweite Argument
`Decimal` (nicht NaN)	Beliebiger Wert	Das erste Argument

Argumenttyp 1	Argumenttyp 2	Ausgabe
Anderer Wert	Beliebiger Wert	Das erste Argument

newuuid()

Gibt eine zufällige 16-Byte-UUID zurück. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `newuuid()` = `123a4567-b89c-12d3-e456-789012345000`

numbytes(String)

Gibt die Anzahl der Byte in der UTF-8 Kodierung der angegebenen Zeichenfolge zurück. Für andere Argumente als `String` gelten Standardkonvertierungsregeln. Unterstützt von der SQL Version vom 23.03.2016 und höher.

Beispiele:

`numbytes("hi")` = 2

`numbytes("€")` = 3

parse_time(String, Long, [String])

Verwenden Sie die `parse_time` Funktion, um einen Zeitstempel in ein für Menschen date/time lesbares Format zu formatieren. Unterstützt von der SQL Version vom 23.03.2016 und höher. Informationen zur Konvertierung einer Zeitstempelzeichenfolge in Millisekunden finden Sie unter time_to_epoch (Zeichenfolge, Zeichenfolge).

Die `parse_time`-Funktion verwendet folgende Argumente:

pattern

(Zeichenfolge) Ein date/time Muster, das Formaten folgt. Joda-Time

Zeitstempel

(Long) Die zu formatierende Zeit in Millisekunden seit dem Startdatum der Unixzeit. Siehe Funktion timestamp().

timezone

(Zeichenfolge) Die Zeitzone des formatierten. date/time Der Standardwert ist "UTC". Die Funktion unterstützt Joda-Time Zeitzonen. Dieses Argument ist optional.

Beispiele:

Wenn diese Nachricht unter dem Thema 'A/B' veröffentlicht wird, `{"ts": "1970.01.01 AD at 21:46:40 CST"}` wird die Nutzlast an den S3-Bucket gesendet:

```
{
    "ruleArn": "arn:aws:iot:us-east-2:ACCOUNT_ID:rule/RULE_NAME",
    "topicRulePayload": {
        "sql": "SELECT parse_time(\"yyyy.MM.dd G 'at' HH:mm:ss z\", 100000000,
 'America/Belize' ) as ts FROM 'A/B'",

        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "s3": {
                    "roleArn": "arn:aws:iam::ACCOUNT_ID:rule:role/ROLE_NAME",
                    "bucketName": "BUCKET_NAME",
                    "key": "KEY_NAME"
                }
            }
        ],
        "ruleName": "RULE_NAME"
    }
}
```

Wenn diese Nachricht unter dem Thema 'A/B' veröffentlicht wird, wird eine ähnliche Nutzlast `{"ts": "2017.06.09 AD at 17:19:46 UTC"}` (aber mit der aktuellen date/time) an den S3-Bucket gesendet:

```
{
    "ruleArn": "arn:aws:iot:us-east-2:ACCOUNT_ID:rule/RULE_NAME",
    "topicRulePayload": {
        "sql": "SELECT parse_time(\"yyyy.MM.dd G 'at' HH:mm:ss z\", timestamp() ) as ts
 FROM 'A/B'",
        "awsIotSqlVersion": "2016-03-23",
```

```
        "ruleDisabled": false,
        "actions": [
            {
                "s3": {
                    "roleArn": "arn:aws:iam::ACCOUNT_ID:rule:role/ROLE_NAME",
                    "bucketName": "BUCKET_NAME",
                    "key": "KEY_NAME"
                }
            }
        ],
        "ruleName": "RULE_NAME"
    }
}
```

`parse_time()` kann auch als Ersatz-Vorlage verwendet werden. Wenn diese Nachricht beispielsweise unter dem Thema 'A/B' veröffentlicht wird, wird die Nutzlast mit dem Schlüssel = „2017" an den S3-Bucket gesendet:

```
{
    "ruleArn": "arn:aws:iot:us-east-2:ACCOUNT_ID:rule/RULE_NAME",
    "topicRulePayload": {
        "sql": "SELECT * FROM 'A/B'",
        "awsIotSqlVersion": "2016-03-23",
        "ruleDisabled": false,
        "actions": [{
            "s3": {
                "roleArn": "arn:aws:iam::ACCOUNT_ID:rule:role/ROLE_NAME",
                "bucketName": "BUCKET_NAME",
                "key": "${parse_time('yyyy', timestamp(), 'UTC')}"
            }
        }],
        "ruleName": "RULE_NAME"
    }
}
```

power(Decimal, Decimal)

Gibt das erste Argument, potenziert mit dem zweiten Argument, zurück. `Decimal`-Argumente werden vor dem Anwenden der Funktion auf doppelte Genauigkeit gerundet. Unterstützt von SQL Version 2015-10-08 und höher. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `power(2, 5)` = 32,0

Argumenttyp 1	Argumenttyp 2	Ausgabe
`Int/Decimal`	`Int/Decimal`	Ein `Decimal`-Wert (mit doppelte Genauigkeit), das erste Argume potenziert mit dem zweiten Argu
`Int/Decimal/String`	`Int/Decimal/String`	Ein `Decimal`-Wert (mit doppelte Genauigkeit), das erste Argume potenziert mit dem zweiten Argu Zeichenfolgen werden in Dezim konvertiert. Wenn `String`-Wert in `Decimal`-Werte umgewande können, ist das Ergebnis `Undef`
Anderer Wert	Anderer Wert	`Undefined` .

Prinzipal()

Gibt den Prinzipal zurück, den das Gerät für die Authentifizierung verwendet, basierend darauf, wie die auslösende Nachricht veröffentlicht wurde. Die folgende Tabelle beschreibt den Prinzipal, der für jede Veröffentlichungsmethode und jedes Protokoll zurückgegeben wird.

So wird die Nachricht veröffentlicht	Protocol (Protokoll)	Anmeldeinformationstyp
MQTT-Client	MQTT	X.509 Gerätezertifikat
AWS IoT MQTT-Client für die Konsole	MQTT	IAM-Benutzer oder Rolle
AWS CLI	HTTP	IAM-Benutzer oder Rolle
AWS IoT Geräte-SDK	MQTT	X.509 Gerätezertifikat
AWS IoT Geräte-SDK	MQTT vorbei WebSocket	IAM-Benutzer oder Rolle

Die folgenden Beispiele zeigen die verschiedenen Arten von Werten, die von `principal()` zurückgegeben werden:

- X.509 Fingerabdruck des Zertifikats: `ba67293af50bf2506f5f93469686da660c7c844e7b3950bfb16813e0d31e9373`
- IAM-Rollen-ID und Sitzungsname: `ABCD1EFG3HIJK2LMNOP5:my-session-name`
- gibt eine Benutzer-ID zurück: `ABCD1EFG3HIJK2LMNOP5`

rand()

Gibt einen pseudozufälligen, einheitlich verteilten doppelten Wert zwischen 0,0 und 1,0 zurück. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel:

`rand()` = 0.8231909191640703

regexp_matches(String, String)

Gibt „true“ zurück, wenn die Zeichenfolge (erstes Argument) eine Übereinstimmung für den regulären Ausdruck (zweites Argument) enthält. Wenn Sie `|` den regulären Ausdruck verwenden, verwenden Sie ihn mit `()`.

Beispiele:

`regexp_matches("aaaa", "a{2,}")` = true.

`regexp_matches("aaaa", "b")` = false.

`regexp_matches("aaa", "(aaa|bbb)")` = true.

`regexp_matches("bbb", "(aaa|bbb)")` = true.

`regexp_matches("ccc", "(aaa|bbb)")` = false.

Erstes Argument:

Argumenttyp	Ergebnis
`Int`	Die `String`-Darstellung des `Int`-Werts

Argumenttyp	Ergebnis
`Decimal`	Die `String`-Darstellung des `Decimal`-Werts
`Boolean`	Die `String`-Darstellung des booleschen Werts („true“ oder „false“)
`String`	Das Tool `String`.
Array	Die `String`-Darstellung des `Array`-Werts (mit Standardkonvertierungsregeln)
Object	Die `String`-Darstellung des Objekts (mit Standardkonvertierungsregeln)
Null	`Undefined` .
Undefined	`Undefined` .

Zweites Argument:

Muss ein gültiger Regex-Ausdruck sein. Non-string Typen werden `String` mithilfe der Standardkonvertierungsregeln in diese umgewandelt. Je nach Typ ist die resultierende Zeichenfolge unter Umständen kein gültiger regulärer Ausdruck. Wenn das (konvertierte) Argument kein gültiger regulärer Ausdruck ist, ist das Ergebnis `Undefined`.

regexp_replace(String, String, String)

Ersetzt alle Vorkommen des zweiten Arguments (regulärer Ausdruck) im ersten Argument durch das dritte Argument. Verweist auf Erfassungsgruppen mit "$". Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel:

`regexp_replace("abcd", "bc", "x")` = "axd"

`regexp_replace("abcd", "b(.*)d", "$1")` = "ac"

Erstes Argument:

Argumenttyp	Ergebnis
`Int`	Die `String`-Darstellung des `Int`-Werts
`Decimal`	Die `String`-Darstellung des `Decimal`-Werts
`Boolean`	Die `String`-Darstellung des booleschen Werts („true“ oder „false“)
`String`	Der Quellwert
Array	Die `String`-Darstellung des `Array`-Werts (mit Standardkonvertierungsregeln)
Object	Die `String`-Darstellung des Objekts (mit Standardkonvertierungsregeln)
Null	`Undefined` .
Undefined	`Undefined` .

Zweites Argument:

Muss ein gültiger Regex-Ausdruck sein. Non-string Typen werden `String` mithilfe der Standardkonvertierungsregeln in diese umgewandelt. Je nach Typ ist die resultierende Zeichenfolge unter Umständen kein gültiger regulärer Ausdruck. Wenn das (konvertierte) Argument kein gültiger regulärer Ausdruck ist, ist das Ergebnis `Undefined`.

Drittes Argument:

Muss eine gültige RegEx-Ersetzungszeichenfolge sein. (Kann auf Capture-Gruppen verweisen.) Non-string Typen werden `String` mithilfe der Standardkonvertierungsregeln in diese umgewandelt. Wenn das (konvertierte) Argument keine gültige RegEx-Ersetzungszeichenfolge ist, ist das Ergebnis `Undefined`.

regexp_substr(String, String)

Findet die erste Übereinstimmung des zweiten Parameters (regex) im ersten Parameter. Verweist auf Erfassungsgruppen mit "$". Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel:

`regexp_substr("hihihello", "hi")` = "hi"

`regexp_substr("hihihello", "(hi)*")` = "hihi"

Erstes Argument:

Argumenttyp	Ergebnis
`Int`	Die `String`-Darstellung des `Int`-Werts
`Decimal`	Die `String`-Darstellung des `Decimal`-Werts
`Boolean`	Die `String`-Darstellung des booleschen Werts („true“ oder „false“)
`String`	Das `String`-Argument
Array	Die `String`-Darstellung des `Array`-Werts (mit Standardkonvertierungsregeln)
Object	Die `String`-Darstellung des Objekts (mit Standardkonvertierungsregeln)
Null	`Undefined` .
Undefined	`Undefined` .

Zweites Argument:

Muss ein gültiger Regex-Ausdruck sein. Non-string Typen werden `String` mithilfe der Standardkonvertierungsregeln in diese umgewandelt. Je nach Typ ist die resultierende Zeichenfolge unter Umständen kein gültiger regulärer Ausdruck. Wenn das (konvertierte) Argument kein gültiger regulärer Ausdruck ist, ist das Ergebnis `Undefined`.

remainder(Decimal, Decimal)

Gibt den Rest zurück, der beim Teilen des ersten Arguments durch das zweite Argument entstanden ist. Äquivalent mit mod(Decimal, Decimal). Sie können auch "%" als infix-Operator für die gleiche Modulfunktionalität verwenden. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `remainder(8, 3)` = 2

Left operator	Right operator	Output
`Int`	`Int`	`Int`, das erste Argumentmodulo zweiten Arguments
`Int/Decimal`	`Int/Decimal`	`Decimal`, das erste Argumentm zweiten Operanden
`String/Int/Decimal`	`String/Int/Decimal`	Wenn alle Zeichenfolgen in Dez erte konvertiert werden, ist das das erste Argumentmodulo des Arguments. Andernfalls `Undefi`
Anderer Wert	Anderer Wert	`Undefined` .

replace(String, String, String)

Ersetzt alle Vorkommen des zweiten Arguments im ersten Argument durch das dritte Argument. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel:

`replace("abcd", "bc", "x")` = `"axd"`.

`replace("abcdabcd", "b", "x")` = `"axcdaxcd"`.

Alle Argumente

Argumenttyp	Ergebnis
`Int`	Die `String`-Darstellung des `Int`-Werts

Argumenttyp	Ergebnis
`Decimal`	Die `String`-Darstellung des `Decimal`-Werts
`Boolean`	Die `String`-Darstellung des booleschen Werts („true“ oder „false“)
`String`	Der Quellwert
Array	Die `String`-Darstellung des `Array`-Werts (mit Standardkonvertierungsregeln)
Object	Die `String`-Darstellung des Objekts (mit Standardkonvertierungsregeln)
Null	`Undefined` .
Undefined	`Undefined` .

rpad(String, Int)

Gibt das Zeichenfolgenargument zurück, das auf der rechten Seite mit der im zweiten Argument festgelegten Anzahl an Leerzeichen aufgefüllt wurde. Das Argument `Int` muss zwischen 0 und 1000 liegen. Wenn der angegebene Wert außerhalb des gültigen Bereichs liegt, wird das Argument auf den nächsten gültigen Wert (0 oder 1000) festgelegt. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiele:

`rpad("hello", 2)` = "`hello  `".

`rpad(1, 3)` = "`1  `".

Argumenttyp 1	Argumenttyp 2	Ergebnis
`String`	`Int`	Der `String`-Wert, der auf der rechten Seite mit der Anzahl an Leerzeichen aufgefüll

Argumenttyp 1	Argumenttyp 2	Ergebnis
		t wurde, die dem angegebenen `Int`-Wert entspricht
`String`	`Decimal`	Das `Decimal`-Argument wird auf den nächsten `Int`-Wert abgerundet und die Zeichenfolge wird auf der rechten Seite mit der Anzahl an Leerzeichen aufgefüllt, die dem angegebenen `Int`-Wert entspricht.
`String`	`String`	Das zweite Argument wird in einen `Decimal`-Wert konvertiert, der auf den nächsten `Int`-Wert abgerundet wird. Der `String`-Wert wird auf der rechten Seite mit der Anzahl an Leerzeichen aufgefüll t, die dem `Int`-Wert entspricht.

Argumenttyp 1	Argumenttyp 2	Ergebnis
Anderer Wert	`Int`/`Decimal`/`String`	Der erste Wert wird mit der Standardk onvertierungen in `String` konvertie rt und anschließend wird die rpad-Funk tion auf `String` angewendet. Wenn er nicht konvertiert werden kann, ist das Ergebnis `Undefined` .
Beliebiger Wert	Anderer Wert	`Undefined` .

round(Decimal)

Runden den angegebenen `Decimal`-Wert auf den nächsten `Int`-Wert. Wenn `Decimal` gleich weit von zwei `Int`-Werten entfernt ist (z. B. 0,5), wird `Decimal` aufgerundet. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `Round(1.2)` = 1.

`Round(1.5)` = 2.

`Round(1.7)` = 2.

`Round(-1.1)` = -1.

`Round(-1.5)` = -2.

Argumenttyp	Ergebnis
`Int`	Das Argument
`Decimal`	`Decimal` wird auf den nächsten `Int`-Wert abgerundet.

Argumenttyp	Ergebnis
`String`	`Decimal` wird auf den nächsten `Int`-Wert abgerundet. Wenn die Zeichenfolge nicht in einen `Decimal`-Wert konvertiert werden kann, ist das Ergebnis `Undefined` .
Anderer Wert	`Undefined` .

rtrim(String)

Entfernt alle nachstehenden Leerzeichen (Tabulatoren und Leerzeichen) aus dem angegebenen `String`-Wert. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiele:

`rtrim(" h i ") = " h i"`

Argumenttyp	Ergebnis
`Int`	Die `String`-Darstellung des `Int`-Werts
`Decimal`	Die `String`-Darstellung des `Decimal`-Werts
`Boolean`	Die `String`-Darstellung des booleschen Werts („true“ oder „false“)
Array	Die `String`-Darstellung des `Array`-Werts (mit Standardkonvertierungsregeln)
Object	Die `String`-Darstellung des Objekts (mit Standardkonvertierungsregeln)
Null	`Undefined` .
Undefined	`Undefined`

sign(Decimal)

Gibt das Vorzeichen der angegebenen Zahl zurück. Wenn das Vorzeichen des Arguments positiv ist, wird 1 zurückgegeben. Wenn das Vorzeichen des Arguments negativ ist, wird -1 zurückgegeben. Wenn das Argument 0 ist, wird 0 zurückgegeben. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiele:

`sign(-7)` = -1.

`sign(0)` = 0.

`sign(13)` = 1.

Argumenttyp	Ergebnis
`Int`	`Int`, das Vorzeichen des `Int`-Werts
`Decimal`	`Int`, das Vorzeichen des `Decimal`-Werts
`String`	`Int`, das Vorzeichen des `Decimal`-Werts Die Zeichenfolge wird in einen `Decimal`-Wert konvertiert und das Vorzeichen des `Decimal`-Werts wird zurückgegeben. Wenn der `String`-Wert nicht in einen `Decimal`-Wert konvertiert werden kann, ist das Ergebnis `Undefined` . Unterstützt von SQL Version 2015-10-08 und höher.
Anderer Wert	`Undefined` .

sin(Decimal)

Gibt den Sinus einer Zahl im Bogenmaß zurück. `Decimal`-Argumente werden vor dem Anwenden der Funktion auf doppelte Genauigkeit gerundet. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `sin(0)` = 0,0

Argumenttyp	Ergebnis
`Int`	`Decimal` (mit doppelter Genauigkeit), der Sinus des Arguments.
`Decimal`	`Decimal` (mit doppelter Genauigkeit), der Sinus des Arguments.
`Boolean`	`Undefined` .
`String`	`Decimal` (mit doppelter Genauigkeit), der Sinus des Arguments. Wenn die Zeichenfolge nicht in einen `Decimal`-Wert konvertiert werden kann, ist das Ergebnis `Undefined` .
Array	`Undefined` .
Object	`Undefined` .
Null	`Undefined` .
`Undefined`	`Undefined` .

sinh(Decimal)

Gibt den hyperbolischen Sinus einer Zahl zurück. `Decimal`-Werte werden vor dem Anwenden der Funktion auf doppelte Genauigkeit gerundet. Das Ergebnis ist ein `Decimal`-Wert mit doppelter Genauigkeit. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `sinh(2.3)` = 4,936961805545957

Argumenttyp	Ergebnis
`Int`	`Decimal` (mit doppelter Genauigkeit), der hyperbolische Sinus des Arguments.

Argumenttyp	Ergebnis
`Decimal`	`Decimal` (mit doppelter Genauigkeit), der hyperbolische Sinus des Arguments.
`Boolean`	`Undefined` .
`String`	`Decimal` (mit doppelter Genauigkeit), der hyperbolische Sinus des Arguments . Wenn die Zeichenfolge nicht in einen `Decimal`-Wert konvertiert werden kann, ist das Ergebnis `Undefined` .
Array	`Undefined` .
Object	`Undefined` .
Null	`Undefined` .
Undefined	`Undefined` .

SourceIp

Ruft die IP-Adresse eines Geräts oder des Routers ab, der eine Verbindung zu diesem Gerät herstellt. Wenn Ihr Gerät direkt mit dem Internet verbunden ist, gibt die Funktion die Quell-IP-Adresse des Geräts zurück. Wenn Ihr Gerät mit einem Router verbunden ist, der eine Verbindung zum Internet herstellt, gibt die Funktion die Quell-IP-Adresse des Routers zurück. Unterstützt von der SQL-Version 2016-03-23. `sourceip()` benötigt keine Parameter.

> **Important**
>
> Die öffentliche Quell-IP-Adresse eines Geräts ist häufig die IP-Adresse des letzten Network Address Translation (NAT)-Gateways, z. B. des Routers oder des Kabelmodems Ihres Internetdienstanbieters.

Beispiele:

```
sourceip()="192.158.1.38"
```

```
sourceip()="1.102.103.104"

sourceip()="2001:db8:ff00::12ab:34cd"
```

SQL-Beispiel

```
SELECT *, sourceip() as deviceIp FROM 'some/topic'
```

Beispiele für die Verwendung der Funktion sourceip () in AWS IoT Core Regelaktionen:

Beispiel 1

Das folgende Beispiel zeigt, wie die Funktion () als Ersatzvorlage in einer DynamoDB-Maßnahme aufgerufen wird.

```
{
 "topicRulePayload": {
  "sql": "SELECT * AS message FROM 'some/topic'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
   {
    "dynamoDB": {
     "tableName": "my_ddb_table",
     "hashKeyField": "key",
     "hashKeyValue": "${sourceip()}",
     "rangeKeyField": "timestamp",
     "rangeKeyValue": "${timestamp()}",
     "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamoDB"
    }
   }
  ]
 }
}
```

Beispiel 2

Das folgende Beispiel zeigt, wie die Funktion sourceip() als MQTT-Benutzereigenschaft mithilfe von Ersetzungsvorlagen hinzugefügt wird.

```
{
 "topicRulePayload": {
```

```
  "sql": "SELECT * FROM 'some/topic'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
   {
    "republish": {
     "topic": "${topic()}/republish",
     "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish",
     "headers": {
      "payloadFormatIndicator": "UTF8_DATA",
      "contentType": "rule/contentType",
      "correlationData": "cnVsZSBjb3JyZWxhdGlvbiBkYXRh",
      "userProperties": [
       {
        "key": "ruleKey1",
        "value": "ruleValue1"
       },
       {
        "key": "sourceip",
        "value": "${sourceip()}"
       }
      ]
     }
    }
   }
  ]
 }
}
```

Sie können die Quell-IP-Adresse aus Nachrichten abrufen, die sowohl über den Message Broker- als auch über den Basic-Ingest-Pfad an AWS IoT Core Regeln weitergeleitet werden. Sie können die Quell-IP auch für IPv4- und IPv6-Nachrichten abrufen. Die Quell-IP wird wie folgt angezeigt:

IPv6: `yyyy:yyyy:yyyy::yyyy:yyyy`

IPv4: `xxx.xxx.xxx.xxx`

> **Note**
>
> Die ursprüngliche Quell-IP wird bei der Maßnahme Erneut veröffentlichen nicht weitergegeben.

substring(String, Int[, Int])

Erwartet einen `String`-Wert, gefolgt von einem oder zwei `Int`-Werten. Für einen `String`-Wert und ein einzelnes `Int`-Argument gibt diese Funktion die Teilzeichenfolge des angegebenen `String`-Werts vom angegebenen `Int`-Index (0-basiert, inklusive) am Ende des `String`-Werts zurück. Für einen `String`-Wert und ein zwei `Int`-Argumente gibt diese Funktion die Teilzeichenfolge des angegebenen `String`-Werts vom ersten `Int`-Indexargument (0-basiert, inklusive) an das zweite`Int`-Indexargument (0-basiert, exklusive) zurück. Indizes, die kleiner als Null sind, werden auf Null festgelegt. Indizes, die größer sind als die `String`-Länge, werden auf die `String`-Länge festgelegt. Wenn bei drei Argumenten der erste Index mindestens genauso groß ist wie der zweite Index, ist das Ergebnis der leere `String`-Wert.

Wenn die angegebenen Argumente nicht (*`String`*,*`Int`*) oder (,,) lauten *`StringInt`*, *`Int`* werden die Standardkonvertierungen auf die Argumente angewendet, um zu versuchen, sie in die richtigen Typen zu konvertieren. Wenn die Typen nicht konvertiert werden können, ist das Ergebnis der Funktion `Undefined`. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiele:

`substring("012345", 0)` = "012345".

`substring("012345", 2)` = "2345".

`substring("012345", 2.745)` = "2345".

`substring(123, 2)` = "3".

`substring("012345", -1)` = "012345".

`substring(true, 1.2)` = "true".

`substring(false, -2.411E247)` = "false".

`substring("012345", 1, 3)` = "12".

`substring("012345", -50, 50)` = "012345".

`substring("012345", 3, 1)` = "".

sql_version()

Gibt die in dieser Regel angegebene SQL Version zurück. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel:

`sql_version()` = "2016-03-23"

sqrt(Decimal)

Gibt die Quadratwurzel einer Zahl zurück. `Decimal`-Argumente werden vor dem Anwenden der Funktion auf doppelte Genauigkeit gerundet. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `sqrt(9)` = 3.0.

Argumenttyp	Ergebnis
`Int`	Die Quadratwurzel des Arguments.
`Decimal`	Die Quadratwurzel des Arguments.
`Boolean`	`Undefined` .
`String`	Die Quadratwurzel des Arguments. Wenn die Zeichenfolge nicht in einen `Decimal`-Wert konvertiert werden kann, ist das Ergebnis `Undefined` .
Array	`Undefined` .
Object	`Undefined` .
Null	`Undefined` .
Undefined	`Undefined` .

startswith(String, String)

Gibt einen Wert vom Typ `Boolean` zurück, der angibt, ob das erste Zeichenfolgenargument mit dem zweiten Zeichenfolgenargument beginnt. Wenn ein Argument `Null` oder `Undefined` ist, ist das Ergebnis `Undefined`. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel:

`startswith("ranger","ran")` = true

Argumenttyp 1	Argumenttyp 2	Ergebnis
`String`	`String`	Ob die erste Zeichenfolge mit d Zeichenfolge beginnt
Anderer Wert	Anderer Wert	Beide Argumente werden mithilf Standardkonvertierungsregeln z lgen konvertiert. Gibt „true“ zurü die erste Zeichenfolge mit der z Zeichenfolge beginnt. Wenn ein `Null` oder `Undefined` ist, ist Ergebnis `Undefined` .

tan(Decimal)

Gibt den Tangens einer Zahl im Bogenmaß zurück. `Decimal`-Werte werden vor dem Anwenden der Funktion auf doppelte Genauigkeit gerundet. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `tan(3)` = -0,1425465430742778

Argumenttyp	Ergebnis
`Int`	`Decimal` (mit doppelter Genauigkeit), der Tangens des Arguments.
`Decimal`	`Decimal` (mit doppelter Genauigkeit), der Tangens des Arguments.
`Boolean`	`Undefined` .
`String`	`Decimal` (mit doppelter Genauigkeit), der Tangens des Arguments. Wenn die Zeichenfolge nicht in einen `Decimal`-Wert konvertiert werden kann, ist das Ergebnis `Undefined` .
Array	`Undefined` .

Argumenttyp	Ergebnis
Object	`Undefined .`
Null	`Undefined .`
Undefined	`Undefined .`

tanh(Decimal)

Gibt den hyperbolischen Tangens einer Zahl im Bogenmaß zurück. `Decimal`-Werte werden vor dem Anwenden der Funktion auf doppelte Genauigkeit gerundet. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `tanh(2.3)` = 0,9800963962661914

Argumenttyp	Ergebnis
`Int`	`Decimal` (mit doppelter Genauigkeit), der hyperbolische Tangens des Arguments
`Decimal`	`Decimal` (mit doppelter Genauigkeit), der hyperbolische Tangens des Arguments
`Boolean`	`Undefined .`
`String`	`Decimal` (mit doppelter Genauigkeit), der hyperbolische Tangens des Arguments Wenn die Zeichenfolge nicht in einen `Decimal`-Wert konvertiert werden kann, ist das Ergebnis `Undefined .`
Array	`Undefined .`
Object	`Undefined .`
Null	`Undefined .`
Undefined	`Undefined .`

time_to_epoch (Zeichenfolge, Zeichenfolge)

Verwenden Sie die `time_to_epoch`-Funktion, um eine Zeitstempelzeichenfolge in eine Anzahl von Millisekunden in der Unix-Zeit umzuwandeln. Unterstützt von der SQL Version vom 23.03.2016 und höher. Informationen zur Konvertierung von Millisekunden in eine formatierte Zeitstempelzeichenfolge finden Sie unter parse_time(String, Long, [String]).

Die `time_to_epoch`-Funktion verwendet folgende Argumente:

Zeitstempel

(Zeichenfolge) Die Zeitstempelzeichenfolge, die seit der Unix-Epoche in Millisekunden konvertiert werden soll. Wenn die Zeitstempelzeichenfolge keine Zeitzone angibt, verwendet die Funktion die UTC-Zeitzone.

pattern

(Zeichenfolge) Ein date/time Muster, das den JDK11-Zeitformaten folgt.

Beispiele:

`time_to_epoch("2020-04-03 09:45:18 UTC+01:00", "yyyy-MM-dd HH:mm:ss VV")` = 1585903518000

`time_to_epoch("18 December 2015", "dd MMMM yyyy")` = 1450396800000

`time_to_epoch("2007-12-03 10:15:30.592 America/Los_Angeles", "yyyy-MM-dd HH:mm:ss.SSS z")` = 1196705730592

timestamp()

Gibt den aktuellen Zeitstempel in Millisekunden ab 00:00:00 Uhr Coordinated Universal Time (UTC), Donnerstag, 1. Januar 1970, zurück, wie er von der Regel-Engine beobachtet wurde. AWS IoT Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel: `timestamp()` = `1481825251155`

topic(Decimal)

Gibt das Topic zurück, an das die Nachricht gesendet wurde, welche die Regel ausgelöst hat. Wenn kein Parameter angegeben ist, wird das gesamte Topic zurückgegeben. Der Parameter `Decimal`

wird verwendet, um ein bestimmtes Themensegment anzugeben, wobei 1 das erste Segment bezeichnet. Für das Thema `foo/bar/baz` gibt topic(1) `foo` zurück, topic(2) gibt `bar` zurück usw. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiele:

`topic()`= "things/myThings/Ding thingOne“

`topic(1)` = "things"

Wenn Basic Ingest verwendet wird, ist das anfängliche Präfix des Themas (`$aws/rules/`*`rule-name`*) nicht für die Thema()-Funktion verfügbar. Angenommen, das Thema ist:

`$aws/rules/BuildingManager/Buildings/Building5/Floor2/Room201/Lights`

`topic()` = "Buildings/Building5/Floor2/Room201/Lights"

`topic(3)` = "Floor2" („Etage2”)

traceid()

Gibt die Ablaufverfolgungs-ID (UUID) der MQTT-Nachricht zurück oder `Undefined`, wenn die Nachricht nicht über MQTT gesendet wurde. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel:

`traceid()` = "12345678-1234-1234-1234-123456789012"

transformieren (Zeichenfolge, Objekt, Array)

Gibt ein Array von Objekten zurück, das das Ergebnis der angegebenen Transformation des `Object` Parameters für den `Array`-Parameter enthält.

Unterstützt von der SQL Version vom 23.03.2016 und höher.

Zeichenfolge

> Der zu verwendende Transformationsmodus. In der folgenden Tabelle finden Sie Informationen zu den unterstützten Transformationsmodi und dazu, wie sie die `Array`-Parameter `Result` aus den `Object` und erstellen.

Object

Ein Objekt, das die Attribute enthält, die auf jedes Element von `Array` angewendet werden sollen.

Array

Eine Reihe von Objekten, auf die die Attribute von `Object` angewendet werden.

Jedes Objekt in diesem Array entspricht einem Objekt in der Antwort der Funktion. Jedes Objekt in der Antwort der Funktion enthält die Attribute, die im ursprünglichen Objekt vorhanden sind, und die Attribute, die von `Object` bereitgestellt werden, wie durch den in `String` angegebenen Transformationsmodus bestimmt.

`String` Parameter	**`Object`** Parameter	**`Array`** Parameter	Ergebnis
`enrichArray`	Objekt	Array von -Objekten.	Ein Array von Objekten, in dem jedes Objekt die Attribute eines Elements aus dem `Array`-Parameter und die Attribute des `Object`-Parameters enthält.
Jeder andere Wert	Beliebiger Wert	Beliebiger Wert	Undefined

> **Note**
>
> Das von dieser Funktion zurückgegebene Array ist auf 128 KiB begrenzt.

Beispiel 1 für die Transformationsfunktion

Dieses Beispiel zeigt, wie die transform()-Funktion aus einem Datenobjekt und einem Array ein einzelnes Array von Objekten erzeugt.

In diesem Beispiel wird folgende Nachricht im MQTT-Thema `A/B` veröffentlicht.

```
{
    "attributes": {
        "data1": 1,
        "data2": 2
    },
    "values": [
        {
            "a": 3
        },
        {
            "b": 4
        },
        {
            "c": 5
        }
    ]
}
```

Diese SQL-Anweisung für eine Themenregelaktion verwendet die transform()-Funktion mit dem Wert `String` von `enrichArray`. In diesem Beispiel `Object` handelt es sich um die `attributes`-Eigenschaft aus der Nachrichtennutzlast und um `Array` das `values` Array, das drei Objekte enthält.

```
select value transform("enrichArray", attributes, values) from 'A/B'
```

Nach dem Empfang der Nachrichtennutzdaten wird die SQL-Anweisung zu der folgenden Antwort ausgewertet.

```
[
  {
    "a": 3,
    "data1": 1,
    "data2": 2
  },
  {
    "b": 4,
    "data1": 1,
    "data2": 2
  },
  {
    "c": 5,
    "data1": 1,
    "data2": 2
```

```
  }
]
```

Beispiel 2 für die Transformationsfunktion

Dieses Beispiel zeigt, wie die transform()-Funktion Literalwerte verwenden kann, um einzelne Attribute aus der Nachrichtennutzlast einzubeziehen und umzubenennen.

In diesem Beispiel wird folgende Nachricht im MQTT-Thema `A/B` veröffentlicht. Dies ist dieselbe Nachricht, die in the section called “Beispiel 1 für die Transformationsfunktion” verwendet wurde.

```
{
    "attributes": {
        "data1": 1,
        "data2": 2
    },
    "values": [
        {
            "a": 3
        },
        {
            "b": 4
        },
        {
            "c": 5
        }
    ]
}
```

Diese SQL-Anweisung für eine Themenregelaktion verwendet die transform()-Funktion mit dem Wert `String` von `enrichArray`. Die `Object` in der transform()-Funktion hat ein einzelnes Attribut, das `key` mit dem Wert von `attributes.data1` in der Nachrichtennutzlast benannt ist, und `Array` ist das `values` Array, das dieselben drei Objekte enthält, die im vorherigen Beispiel verwendet wurden.

```
select value transform("enrichArray", {"key": attributes.data1}, values) from 'A/B'
```

Nach dem Empfang der Nachrichtennutzdaten wird diese SQL-Anweisung zu der folgenden Antwort ausgewertet. Beachten Sie, wie die `data1`-Eigenschaft `key` in der Antwort benannt ist.

```
[
  {
    "a": 3,
```

```
    "key": 1
  },
  {
    "b": 4,
    "key": 1
  },
  {
    "c": 5,
    "key": 1
  }
]
```

Beispiel 3 für die Transformationsfunktion

Dieses Beispiel zeigt, wie die transform()-Funktion in verschachtelten SELECT-Klauseln verwendet werden kann, um mehrere Attribute auszuwählen und neue Objekte für die nachfolgende Verarbeitung zu erstellen.

In diesem Beispiel wird folgende Nachricht im MQTT-Thema `A/B` veröffentlicht.

```
{
  "data1": "example",
  "data2": {
    "a": "first attribute",
    "b": "second attribute",
    "c": [
      {
        "x": {
          "someInt": 5,
          "someString": "hello"
        },
        "y": true
      },
      {
        "x": {
          "someInt": 10,
          "someString": "world"
        },
        "y": false
      }
    ]
  }
}
```

Das `Object` für diese Transformationsfunktion ist das Objekt, das von der SELECT-Anweisung zurückgegeben wurde, die die Elemente a und b des `data2`-Objekts der Nachricht enthält. Der `Array`-Parameter besteht aus den beiden Objekten aus dem `data2.c` Array in der ursprünglichen Nachricht.

```
select value transform('enrichArray', (select a, b from data2), (select value c from
 data2)) from 'A/B'
```

Bei der vorherigen Nachricht ergibt die SQL-Anweisung die folgende Antwort.

```
[
  {
    "x": {
      "someInt": 5,
      "someString": "hello"
    },
    "y": true,
    "a": "first attribute",
    "b": "second attribute"
  },
  {
    "x": {
      "someInt": 10,
      "someString": "world"
    },
    "y": false,
    "a": "first attribute",
    "b": "second attribute"
  }
]
```

Das in dieser Antwort zurückgegebene Array könnte mit Themenregelaktionen verwendet werden, die `batchMode` unterstützen.

trim(String)

Entfernt alle führenden und nachfolgenden Leerzeichen aus dem angegebenen `String`-Wert. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiel:

`Trim(" hi ")` = "hi"

Argumenttyp	Ergebnis
`Int`	Die `String`-Darstellung des `Int`-Werts nach dem Entfernen der führenden und nachfolgenden Leerzeichen.
`Decimal`	Die `String`-Darstellung des `Decimal`-Werts nach dem Entfernen der führenden und nachfolgenden Leerzeichen.
`Boolean`	Die `String`-Darstellung des `Boolean`-Werts ("true" oder "false") nach dem Entfernen der führenden und nachfolge nden Leerzeichen.
`String`	Der `String`-Wert nach dem Entfernen der führenden und nachfolgenden Leerzeichen.
Array	Die `String`-Darstellung des `Array`-Werts mit Standardkonvertierungsregeln
Object	Die `String`-Darstellung des Objekts mit Standardkonvertierungsregeln
Null	`Undefined` .
Undefined	`Undefined` .

trunc(Decimal, Int)

Schneidet das erste Argument auf die Anzahl von `Decimal`-Stellen ab, die vom zweiten Argument festgelegt wurden. Wenn das zweite Argument kleiner ist als Null, wird es auf Null festgelegt. Wenn das zweite Argument größer ist als 34, wird es auf 34 festgelegt. Nachfolgende Nullen werden aus dem Ergebnis entfernt. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiele:

`trunc(2.3, 0)` = 2.

`trunc(2.3123, 2)` = 2,31.

`trunc(2.888, 2)` = 2,88.

`trunc(2.00, 5)` = 2.

Argumenttyp 1	Argumenttyp 2	Ergebnis
`Int`	`Int`	Der Quellwert
`Int/Decimal`	`Int/Decimal`	Das erste Argument wird auf die abgeschnitten, die vom zweiten beschrieben wird. Wenn das zw Argument kein `Int`-Wert ist, wir den nächsten `Int`-Wert abgeru
`Int/Decimal/String`	`Int/Decimal`	Das erste Argument wird auf die abgeschnitten, die vom zweiten beschrieben wird. Wenn das zw Argument kein `Int`-Wert ist, wir den nächsten `Int`-Wert abgeru `String`-Wert wird zu einem De konvertiert. Wenn die Konvertie Zeichenfolge fehlschlägt, ist das `Undefined` .
Anderer Wert		`Undefined` .

upper(String)

Gibt die großgeschriebene Version des angegebenen `String`-Werts zurück. Andere Argumente als `String` werden mit der Standardkonvertierungsregeln in ,`String` konvertiert. Unterstützt von SQL Version 2015-10-08 und höher.

Beispiele:

`upper("hello")` = "HELLO"

`upper(["hello"])` = "[\"HELLO\"]"

Literale

Sie können Literalobjekte in den SELECT- und WHERE-Klauseln Ihrer Regel-SQL direkt angeben, was zum Weitergeben von Informationen praktisch ist.

> **Note**
>
> Literale sind nur verfügbar, wenn Sie die SQL-Version vom 23.03.2016 oder höher verwenden.

Die JSON-Objektsyntax wird verwendet (Schlüssel-Wert-Paare, durch Trennzeichen getrennt, wobei Schlüssel Zeichenfolgen und Wert JSON-Werte sind, eingeschlossen in geschweifte Klammern: {}). Beispiel:

Eingehende Nutzlast veröffentlicht für Topic `topic/subtopic`: `{"lat_long": [47.606,-122.332]}`

SQL-Anweisung: `SELECT {'latitude': get(lat_long, 0),'longitude':get(lat_long, 1)} as lat_long FROM 'topic/subtopic'`

Die resultierende ausgehende Nutzlast ist: `{"lat_long": {"latitude":47.606,"longitude":-122.332}}`.

Sie können auch Arrays in den SELECT- und WHERE-Klauseln Ihrer Regel-SQL direkt angeben, was Ihnen das Gruppieren von Informationen ermöglicht. Die JSON-Syntax wird verwendet (setzen Sie durch Trennzeichen getrennte Elemente zwischen eckige Klammern, [], um ein Arrayliteral zu erstellen). Beispiel:

Eingehende Nutzlast veröffentlicht für Topic `topic/subtopic`: `{"lat": 47.696, "long": -122.332}`

SQL-Anweisung: `SELECT [lat,long] as lat_long FROM 'topic/subtopic'`

Verwenden Sie eine Austausch-Vorlage, um die JSON-Daten zu erhöhen, die zurückgegeben werden, wenn eine Regel ausgelöst wird und `{"lat_long": [47.606,-122.332]}` eine Maßnahme durchführt.

Case-Anweisungen

Case-Anweisungen können zum Branchen verwendet werden, wie etwa eine switch-Anweisung.

Syntax:

```
CASE v WHEN t[1] THEN r[1]
  WHEN t[2] THEN r[2] ...
  WHEN t[n] THEN r[n]
  ELSE r[e] END
```

Der Ausdruck *v* wird ausgewertet und die Übereinstimmung mit dem Wert jeder *t[i]*-Klausel abgeglichen. `WHEN` Bei einer Übereinstimmung wird der entsprechende *r[i]*-Ausdruck zum Ergebnis der `CASE`-Anweisung. Die `WHEN`-Klauseln werden der Reihe nach ausgewertet, sodass, wenn es mehr als eine übereinstimmende Klausel gibt, das Ergebnis der ersten übereinstimmenden Klausel zum Ergebnis der `CASE`-Anweisung wird. Wenn es keine Treffer gibt, ist *r[e]* der`ELSE` Klausel das Ergebnis. Wenn es keine Übereinstimmung und keine `ELSE`-Klausel gibt, ist das Ergebnis `Undefined`.

Für `CASE`-Anweisungen ist mindestens eine `WHEN`-Klausel erforderlich. Eine `ELSE`-Klausel ist optional.

Beispiel:

Eingehende Nutzlast veröffentlicht für Topic `topic/subtopic`:

```
{
    "color":"yellow"
}
```

SQL-Anweisung:

```
SELECT CASE color
        WHEN 'green' THEN 'go'
        WHEN 'yellow' THEN 'caution'
        WHEN 'red' THEN 'stop'
        ELSE 'you are not at a stop light' END as instructions
    FROM 'topic/subtopic'
```

Die resultierende Ausgabenutzlast ist:

```
{
    "instructions":"caution"
}
```

> **Note**
>
> Wenn *v* `Undefined` ist, ist das Ergebnis der Case-Anweisung `Undefined`.

JSON-Erweiterungen

Sie können die folgenden Erweiterungen für die ANSI SQL-Syntax verwenden, um die Arbeit mit verschachtelten JSON-Objekten zu vereinfachen.

"." Operator

Dieser Operator greift auf Elemente in eingebetteten JSON-Objekten zu und funktioniert genauso wie ANSI SQL und. JavaScript Beispiel:

```
SELECT foo.bar AS bar.baz FROM 'topic/subtopic'
```

wählt den Wert der `bar`-Eigenschaft im `foo`-Objekt aus der folgenden Nachrichten-Payload aus, die an das `topic/subtopic`-Thema gesendet wurde.

```
{
  "foo": {
    "bar": "RED",
    "bar1": "GREEN",
    "bar2": "BLUE"
  }
}
```

Die resultierende ausgehende Nutzlast ist:

```
{
  "bar": {
    "baz": "RED"
  }
}
```

Wenn ein JSON-Eigenschaftsname einen Bindestrich oder numerische Zeichen enthält, funktioniert die Notation „Punkt“ nicht. Stattdessen müssen Sie die Funktion get verwenden, um den Wert der Eigenschaft zu extrahieren.

In diesem Beispiel wird die folgende Nachricht an das `iot/rules`-Thema gesendet.

```
{
  "mydata": {
    "item2": {
      "0": {
        "my-key": "myValue"
      }
    }
  }
}
```

Normalerweise wird der Wert von `my-key` wie in dieser Abfrage identifiziert.

```
SELECT * from iot/rules WHERE mydata.item2.0.my-key= "myValue"
```

Da der Eigenschaftsname `my-key` jedoch einen Bindestrich und ein numerisches Zeichen `item2` enthält, muss die Funktion get verwendet werden, wie die folgende Abfrage zeigt.

```
SELECT * from 'iot/rules' WHERE get(get(get(mydata,"item2"),"0"),"my-key") = "myValue"
```

*-Operator

Dieser funktioniert genauso wie der Platzhalter * in ANSI SQL. Er wird nur in der SELECT-Klausel verwendet und erstellt ein neues JSON-Objekt, das die Nachrichtdaten enthält. Wenn die Nachrichtnutzlast nicht im JSON-Format vorliegt, gibt * die gesamte Nachrichtnutzlast als Rohbytes zurück. Beispiel:

```
SELECT * FROM 'topic/subtopic'
```

Anwenden einer Funktion auf einen Attributwert

Im Folgenden finden Sie ein Beispiel für eine JSON-Nutzlast, die von einem Gerät veröffentlicht werden kann:

```
{
    "deviceid" : "iot123",
    "temp" : 54.98,
    "humidity" : 32.43,
    "coords" : {
        "latitude" : 47.615694,
        "longitude" : -122.3359976
```

```
    }
}
```

Das folgende Beispiel wendet eine Funktion auf einen Attributwert in einer JSON-Nutzlast an:

```
SELECT temp, md5(deviceid) AS hashed_id FROM topic/#
```

Das Ergebnis dieser Abfrage ist das folgende JSON-Objekt:

```
{
   "temp": 54.98,
   "hashed_id": "e37f81fb397e595c4aeb5645b8cbbbd1"
}
```

Ersetzungsvorlagen

Sie können eine Ersatzvorlage verwenden, um die JSON-Daten zu erweitern, die zurückgegeben werden, wenn eine Regel ausgelöst wird und AWS IoT eine Aktion ausführt. Die Syntax für eine Substitutionsvorlage lautet *${* Ausdruck*}*, wobei Ausdruck ein beliebiger Ausdruck sein kann, der von INSELECT-Klauseln, AWS IoT WHERE-Klauseln und unterstützt wird. AWS IoT Regelaktionen Dieser Ausdruck kann in ein Aktionsfeld für eine Regel eingefügt werden, sodass Sie eine Aktion dynamisch konfigurieren können. Tatsächlich ersetzt diese Funktion eine Information in einer Aktion. Dazu gehören Funktionen, Operatoren und Informationen aus der ursprünglichen Nachrichtennutzlast.

> **Important**
>
> Da ein Ausdruck in einer Ersatzvorlage separat von dem „SELECT...“-Ausdruck evaluiert wird, können Sie nicht auf ein mit der AS-Klausel erstelltes Alias verweisen. Sie können zu den unterstützten Funktionen und Operatoren nur in der ursprünglichen Nutzlast vorhandene Informationen referenzieren.

Weitere Informationen zu unterstützten Ausdrücken unter AWS IoT SQL-Referenz.

Die folgenden Regelaktionen unterstützen Ersetzungsvorlagen. Jede Aktion unterstützt verschiedene Felder, die ersetzt werden können.

- Apache Kafka

- CloudWatch Alarme
- CloudWatch Logs
- CloudWatch Metriken
- DynamoDB
- DynamoDBv2
- Elasticsearch
- HTTP
- InfluxDB
- AWS IoT SiteWise
- Kinesis Data Streams
- Firehose
- Lambda
- Speicherort
- OpenSearch
- Erneut veröffentlichen
- S3
- SNS
- SQS
- Step Functions
- Timestream

Ersetzungsvorlagen werden in den Aktionsparametern innerhalb einer Regel angezeigt:

```
{
    "sql": "SELECT *, timestamp() AS timestamp FROM 'my/iot/topic'",
    "ruleDisabled": false,
    "actions": [{
        "republish": {
            "topic": "${topic()}/republish",
            "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
        }
    }]
}
```

Wenn diese Regel vom folgenden in `my/iot/topic` veröffentlichten JSON ausgelöst wird:

```
{
    "deviceid": "iot123",
    "temp": 54.98,
    "humidity": 32.43,
    "coords": {
        "latitude": 47.615694,
        "longitude": -122.3359976
    }
}
```

Anschließend veröffentlicht diese Regel das folgende JSON-Format`my/iot/topic/republish`, das AWS IoT durch Folgendes ersetzt wird: `${topic()}/republish`

```
{
    "deviceid": "iot123",
    "temp": 54.98,
    "humidity": 32.43,
    "coords": {
        "latitude": 47.615694,
        "longitude": -122.3359976
    },
    "timestamp": 1579637878451
}
```

Per-element Ersatz für Array-Payloads in AWS IoT Regeln

Mit AWS IoT können Sie den ${ Standardausdruck } der Substitutionsvorlage verwenden. Diese Vorlage wird einmal pro Nachricht anhand der Nachrichtennutzdaten, die das Gerät veröffentlicht hat, aufgelöst, bevor die SQL-Anweisung der Regel sie transformiert. Wenn es sich bei Ihrer Nachrichtennutzlast um ein JSON-Array handelt, verwenden Sie stattdessen das Sequenzfeld pro Element. `@{ }` Diese Sequenz wird für jedes Element der Nutzlast, das die SQL-Anweisung der Regel erzeugt, einmal aufgelöst. Die Regelaktion schreibt einen InfluxDB Zeilenprotokollpunkt pro Element. Sie verwendet den Wert aus dem passenden Element.

> **Note**
>
> Die beiden Sequenzen lesen unterschiedliche Payloads, sodass sie für denselben Feldnamen in unterschiedliche Werte aufgelöst werden können. Wenn die SQL-Anweisung

der Regel Felder umbenennt oder umformt, müssen } Sie innerhalb des @{ Felds auf den Feldnamen verweisen, so wie er in der SQL-Ausgabe erscheint. Wenn ein Element das Feld nicht enthält, wird die Sequenz in einen leeren Wert aufgelöst. Sie greift nicht auf einen Wert auf Nachrichtenebene zurück und AWS IoT löscht einen leeren Tag-Wert aus dem Zeilenprotokoll, das sie schreibt.

Important

Die @{ }Feldsequenz funktioniert nur in der AWS IoT InfluxDB Regelaktionskonfiguration. An diesen Orten funktioniert sie nicht:

- Die SQL-Anweisung für Regeln (die SELECT-, WHERE- oder SET-Klauseln)
- Jede andere Regelaktion
- Eine -Funktion Die Sequenz darf keinen enthalten, wie `@{upper(room)}` z. Sie können die Sequenz auch nicht an eine Ersatzvorlagenfunktion übergeben, da ein einzelner Konfigurationswert nicht `@{...}` sowohl `${...}` als auch enthalten kann.
- Die Fehleraktion der Regel, auch wenn es sich bei dieser Fehleraktion um eine InfluxDB Aktion handelt. Die Fehleraktion wird einmal für die Nachricht ausgeführt, sodass sie nie pro Element behoben wird.

Wenn Sie sie an einer dieser Stellen verwenden, schlagen die `ReplaceTopicRule` Operationen `CreateTopicRule` und mit einem Fehler fehl.

Für das @{ Feld „}Sequenz pro Element“ gelten die folgenden Anforderungen:

- Muss auf ein einzelnes Feld verweisen, z. B. `@{room}` oder `@{meta.sensor_id}`
- Darf keine Funktionen, Arithmetik, Vergleiche oder Literale enthalten
- Muss höchstens einmal in einem Konfigurationswert vorkommen
- Darf nicht im selben Konfigurationswert wie eine `${...}` Ersatzvorlage vorkommen
- Muss in einem Konfigurationswert vorkommen, nicht in einem Schlüssel (z. B. in einem InfluxDB Tagwert, aber nicht in einem Tagschlüssel)

Ordnen Sie Array-Elemente zu InfluxDB Punkte

In diesem Beispiel setzt eine InfluxDB Regelaktion den Messnamen (`tableName`) auf `@{measurement_type}` und ein `location` Tag auf: `@{room}`

```
{
  "topicRulePayload": {
    "sql": "SELECT * FROM 'iot/topic'",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "influxDB": {
          "destinationArn": "arn:aws:iot:us-east-1:123456789012:ruledestination/
influxdb/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
          "roleArn": "arn:aws:iam::123456789012:role/aws_iot_influxdb",
          "databaseName": "device_metrics",
          "tableName": "@{measurement_type}",
          "organization": "my_organization",
          "tags": {
            "location": "@{room}"
          },
          "timestampUnit": "ms"
        }
      }
    ]
  }
}
```

Die folgende Array-Nutzlast löst die Regel aus:

```
[
    { "measurement_type": "temperature", "room": "kitchen", "value": 5,  "timestamp":
 1700000000000 },
    { "measurement_type": "humidity",    "room": "bedroom", "value": 15, "timestamp":
 1700000001000 }
]
```

Die Aktion schreibt einen InfluxDB Zeilenprotokollpunkt für jedes Element des Arrays. Jedes Element erhält `@{measurement_type}` seinen eigenen Messnamen und `@{room}` sein eigenes `location` Tag:

```
temperature,location=kitchen value=5i 1700000000000
```

```
humidity,location=bedroom value=15i 1700000001000
```

> **Note**
>
> AWS IoT schließt jedes Feld, auf das eine `@{ }`Feldsequenz verweist, aus der Feldgruppe aus. AWS IoT verwendet den Wert dieses Felds als Messnamen oder Tag-Wert. `measurement_type`Liefert in diesem Beispiel die Messung und `room` liefert das `location` Tag. AWS IoT schließt beide aus, sodass jeder Punkt nur `value` als Feld gilt.

Im vorherigen Beispiel wird verwendet`SELECT *`, dass die Payload, die die Aktion empfängt, mit der Payload übereinstimmt, die das Gerät veröffentlicht hat. Wenn die SQL-Anweisung ein Feld umbenennt, verwenden Sie den Namen, der in der SQL-Ausgabe erscheint. Wenn die Anweisung beispielsweise Folgendes enthält`room AS location_name`, verwenden Sie `@{location_name}` anstelle von`@{room}`.

Entspricht einer Nutzlast, die die SQL-Anweisung erzeugt

Die Payload, die die Aktion empfängt, ist nicht immer die Payload, die das Gerät veröffentlicht hat. In diesem Beispiel veröffentlicht das Gerät ein Objekt, das eine Reihe von Messwerten enthält:

```
{
    "device_id": "sensor-1",
    "readings": [
        { "measurement_type": "temperature", "room": "kitchen", "value": 5,
 "timestamp": 1700000000000 },
        { "measurement_type": "humidity",    "room": "bedroom", "value": 15,
 "timestamp": 1700000001000 }
    ]
}
```

Die Regel verwendet`SELECT VALUE`, um das `readings` Array als Payload der obersten Ebene zurückzugeben. Weitere Informationen finden Sie unter the section called “Gibt ein `Array` als Objekt der obersten Ebene aus”.

```
SELECT VALUE readings FROM 'iot/topic'
```

Die Aktion empfängt das folgende Array, obwohl das Gerät ein Objekt veröffentlicht hat. Sie schreibt dieselben zwei Punkte wie im vorherigen Beispiel, weil jedes Element immer noch `measurement_typeroom`, und enthält`value`.

```
[
    { "measurement_type": "temperature", "room": "kitchen", "value": 5,  "timestamp":
 1700000000000 },
    { "measurement_type": "humidity",    "room": "bedroom", "value": 15, "timestamp":
 1700000001000 }
]
```

> **Note**
>
> Das `device_id` Feld befindet sich in der Payload, die das Gerät veröffentlicht hat, aber nicht in dem Array, das die SQL-Anweisung erzeugt. Aus diesem Grund wird `@{device_id}` es in dieser Regel nicht aufgelöst. Verweisen Sie nur auf Felder, die in den Elementen des erzeugten Arrays vorkommen.

Verschachtelte Objektabfragen

Sie können verschachtelte SELECT-Klauseln verwenden, um Attribute in Arrays und inneren JSON-Objekten abzufragen. Unterstützt von der SQL Version vom 23.03.2016 und höher.

Betrachten Sie die folgende MQTT-Meldung:

```
{
    "e": [
        { "n": "temperature", "u": "Cel", "t": 1234, "v": 22.5 },
        { "n": "light", "u": "lm", "t": 1235, "v": 135 },
        { "n": "acidity", "u": "pH", "t": 1235, "v": 7 }
    ]
}
```

Example

Sie können Werte in ein neues Array mit der folgenden Regel konvertieren.

```
SELECT (SELECT VALUE n FROM e) as sensors FROM 'my/topic'
```

Die Regel generiert folgenden Output:

```
{
    "sensors": [
```

```
        "temperature",
        "light",
        "acidity"
    ]
}
```

Example

Mit derselben MQTT-Meldung können Sie auch einen bestimmten Wert innerhalb eines verschachtelten Objekts mit der folgenden Regel abfragen.

```
SELECT (SELECT v FROM e WHERE n = 'temperature') as temperature FROM 'my/topic'
```

Die Regel generiert folgenden Output:

```
{
    "temperature": [
        {
            "v": 22.5
        }
    ]
}
```

Example

Sie können die Ausgabe auch mit einer komplizierteren Regel glätten.

```
SELECT get((SELECT v FROM e WHERE n = 'temperature'), 0).v as temperature FROM 'topic'
```

Die Regel generiert folgenden Output:

```
{
    "temperature": 22.5
}
```

Arbeiten mit binären Nutzlasten

Wenn Nachrichtennutzlasten als unformatierte binäre Daten behandelt werden sollen (und nicht als JSON-Objekt), können Sie den Operator * verwenden, um darauf in einer SELECT-Klausel zu verweisen.

In diesem Thema:

- Beispiele für binäre Nutzlasten
- Payloads von protobuf-Nachrichten entschlüsseln

Beispiele für binäre Nutzlasten

Wenn Sie * verwenden, um die Nachrichtennutzlast als unformatierte Binärdaten zu bezeichnen, können Sie der Regel Daten hinzufügen. Wenn Sie eine leere oder eine JSON-Nutzlast haben, können der resultierenden Nutzlast mithilfe der Regel Daten hinzugefügt werden. Es folgen Beispiele unterstützter `SELECT`-Klauseln.

- Sie können die folgenden `SELECT`-Klauseln nur mit einem Sternchen (*) für binäre Payloads verwenden.
  - ```
    SELECT * FROM 'topic/subtopic'
    ```
  - ```
    SELECT * FROM 'topic/subtopic' WHERE timestamp() % 12 = 0
    ```
- Sie können auch Daten hinzufügen und die folgenden `SELECT`-Klauseln verwenden.
  - ```
    SELECT *, principal() as principal, timestamp() as time FROM 'topic/subtopic'
    ```
  - ```
    SELECT encode(*, 'base64') AS data, timestamp() AS ts FROM 'topic/subtopic'
    ```
- Sie können diese `SELECT`-Klauseln auch mit binären Payloads verwenden.
  - Das Folgende bezieht sich auf `device_type` in der WHERE-Klausel.

    ```
    SELECT * FROM 'topic/subtopic' WHERE device_type = 'thermostat'
    ```
  - Folgendes wird ebenfalls unterstützt.

    ```
    {
     "sql": "SELECT * FROM 'topic/subtopic'",
     "actions": [
      {
       "republish": {
        "topic": "device/${device_id}"
       }
      }
     ]
    ```

```
}
```

Die folgenden Regelaktionen unterstützen keine binären Payloads, sodass Sie sie dekodieren müssen.

- Für Regelaktionen, die die Eingabe binärer Nutzlast nicht unterstützen, wie z. B. die -Maßnahme, müssen Sie binäre Nutzlasten dekodieren. Die Lambda-Regelaktion kann binäre Daten empfangen, wenn sie base64-kodiert und in einer JSON-Nutzlast enthalten sind. Dazu müssen Sie die Regel folgendermaßen ändern:

  ```
  SELECT encode(*, 'base64') AS data FROM 'my_topic'
  ```

- Die SQL-Anweisung unterstützt keine Zeichenfolge als Eingabe. Um eine String-Eingabe in JSON zu konvertieren, können Sie den folgenden Befehl ausführen.

  ```
  SELECT decode(encode(*, 'base64'), 'base64') AS payload FROM 'topic'
  ```

Payloads von protobuf-Nachrichten entschlüsseln

Protocol Buffers (protobuf) ist ein Open-Source-Datenformat, das zur Serialisierung strukturierter Daten in kompakter, binärer Form verwendet wird. Es wird verwendet, um Daten über Netzwerke zu übertragen oder in Dateien zu speichern. Mit Protobuf können Sie Daten in kleinen Paketgrößen und mit einer schnelleren Geschwindigkeit als mit anderen Nachrichtenformaten senden. AWS IoT Core Regeln unterstützen Protobuf, indem sie die SQL-Funktion decode (value, decodingScheme) bereitstellen, mit der Sie Protobuf-kodierte Nachrichtennutzdaten in das JSON-Format dekodieren und an nachgeschaltete Dienste weiterleiten können. In diesem Abschnitt wird Schritt für Schritt beschrieben, wie die protobuf-Decodierung in AWS IoT Core -Regeln konfiguriert wird.

In diesem Abschnitt:

- Voraussetzungen
- Deskriptordateien erstellen
- So laden Sie die Dateien zu einem S3-Bucket hoch
- Konfigurieren Sie die protobuf-Dekodierung in Regeln.
- Einschränkungen
- Bewährte Methoden

Voraussetzungen

- Ein grundlegendes Verständnis von Protocol Buffers (protobuf)
- Die `.proto`-Dateien, die Nachrichtentypen und zugehörige Abhängigkeiten definieren
- Installieren Sie den protobuf Compiler (protoc) auf Ihrem System.

Deskriptordateien erstellen

Wenn Sie bereits über die Deskriptordateien verfügen, können Sie diesen Schritt überspringen. Eine Deskriptordatei (`.desc`) ist eine kompilierte Version einer `.proto`-Datei, bei der es sich um eine Textdatei handelt, die die Datenstrukturen und Nachrichtentypen definiert, die bei einer protobuf-Serialisierung verwendet werden sollen. Um eine Deskriptordatei zu generieren, müssen Sie eine `.proto`-Datei definieren und sie mit dem protoc-Compiler kompilieren.

1. Erstellen Sie `.proto`-Dateien, die die Nachrichtentypen definieren. Ein Beispiel `.proto` könnte folgendermaßen aussehen:

   ```
   syntax = "proto3";

   message Person {
     optional string name = 1;
     optional int32 id = 2;
     optional string email = 3;
   }
   ```

   In dieser `.proto`-Beispieldatei verwenden Sie die Proto3-Syntax und definieren den Nachrichtentyp `Person`. Die `Person`-Nachrichtendefinition spezifiziert drei Felder (Name, ID und E-Mail). Weitere Informationen zu `.proto`-Dateiformaten im Language Guide (proto3).

2. Verwenden Sie den Protoc-Compiler, um die `.proto` Dateien zu kompilieren und eine Deskriptordatei zu generieren. Ein Beispielbefehl zum Erstellen einer Deskriptordatei (`.desc`) kann der folgende sein:

   ```
   protoc --descriptor_set_out=<FILENAME>.desc \
       --proto_path=<PATH_TO_IMPORTS_DIRECTORY> \
       --include_imports \
       <PROTO_FILENAME>.proto
   ```

Dieser Beispielbefehl generiert eine Deskriptordatei`<FILENAME>.desc`, mit der AWS IoT Core Rules Protobuf-Nutzlasten dekodieren kann, die der in definierten Datenstruktur entsprechen. `<PROTO_FILENAME>.proto`

- `--descriptor_set_out`

  Gibt den Namen der Deskriptordatei (`<FILENAME>.desc`) an, die generiert werden soll.

- `--proto_path`

  Gibt die Speicherorte aller importierten `.proto`-Dateien an, auf die in der kompilierten Datei verwiesen wird. Sie können das Kennzeichen mehrfach angeben, wenn Sie mehrere importierte `.proto`-Dateien mit unterschiedlichen Speicherorten haben.

- `--include_imports`

  Gibt an, dass alle importierten `.proto`-Dateien ebenfalls kompiliert und in die `<FILENAME>.desc`-Deskriptordatei aufgenommen werden sollen.

- `<PROTO_FILENAME>.proto`

  Gibt den Namen der `.proto`-Datei an, die Sie kompilieren möchten.

Weitere Informationen über die Protoc-Referenz finden Sie unter API-Referenz.

So laden Sie die Dateien zu einem S3-Bucket hoch

Nachdem Sie Ihre Deskriptordateien erstellt haben`<FILENAME>.desc`, laden Sie die Deskriptordateien `<FILENAME>.desc` mithilfe der AWS API, des AWS SDK oder der in einen Amazon S3 S3-Bucket hoch. AWS-Managementkonsole

Wichtige Überlegungen

- Stellen Sie sicher, dass Sie die Deskriptordateien in einen Amazon S3 S3-Bucket AWS-Konto in derselben Umgebung hochladen AWS-Region , in der Sie Ihre Regeln konfigurieren möchten.
- Stellen Sie sicher, dass Sie AWS IoT Core Zugriff auf das Lesen `FileDescriptorSet` von S3 gewähren. Wenn in Ihrem S3-Bucket die serverseitige Verschlüsselung (SSE) deaktiviert ist oder wenn Ihr S3-Bucket mit S3-managed Amazon-Schlüsseln (SSE-S3) verschlüsselt ist, sind keine zusätzlichen Richtlinienkonfigurationen erforderlich. Dies kann mit der Beispiel-Bucket-Richtlinie erreicht werden:

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Sid": "Statement1",
   "Effect": "Allow",
   "Principal": {
    "Service": "iot.amazonaws.com"
   },
   "Action": "s3:Get*",
                    "Resource": "arn:aws:s3:::<BUCKET NAME>/<FILENAME>.desc"
  }
 ]
}
```

- Wenn Ihr S3-Bucket mit einem AWS Key Management Service Schlüssel (SSE-KMS) verschlüsselt ist, stellen Sie sicher, dass Sie beim Zugriff auf Ihren S3-Bucket die AWS IoT Core Erlaubnis zur Verwendung des Schlüssels erteilen. Dazu müssen Sie diese Erklärung zu Ihrer Schlüsselrichtlinie hinzufügen:

```
{
 "Sid": "Statement1",
 "Effect": "Allow",
 "Principal": {
  "Service": "iot.amazonaws.com"
 },
 "Action": [
  "kms:Decrypt",
  "kms:GenerateDataKey*",
  "kms:DescribeKey"
 ],
        "Resource": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"

}
```

Konfigurieren Sie die protobuf-Dekodierung in Regeln.

Nachdem Sie die Deskriptordateien in Ihren Amazon S3 S3-Bucket hochgeladen haben, konfigurieren Sie eine Regel, die Ihr protobuf-Nachrichtennutzdatenformat mit der SQL-Funktion

decode (value, decodingScheme) dekodieren kann. Eine ausführliche Funktionssignatur und ein Beispiel finden Sie in der SQL-Funktion decode (value, decodingScheme) der SQL-Referenz.AWS IoT

Im Folgenden finden Sie ein Beispiel für einen SQL-Ausdruck, der die Funktion decode (value, decodingScheme) verwendet:

```
SELECT VALUE decode(*, 'proto', '<BUCKET NAME>', '<FILENAME>.desc', '<PROTO_FILENAME>',
 '<PROTO_MESSAGE_TYPE>') FROM '<MY_TOPIC>'
```

In diesem Beispielausdruck:

- Sie verwenden die SQL-Funktion decode (value, DecodingScheme), um die Nutzdaten der binären Nachricht zu dekodieren, auf die von * verwiesen wird. Dies kann eine binäre protobuf-kodierte Nutzlast oder eine JSON-Zeichenfolge sein, die eine Base64-kodierte protobuf-Nutzlast darstellt.
- Die bereitgestellte Nachrichtennutzlast ist mit dem Nachrichtentyp `Person` definierten in `PROTO_FILENAME.proto` kodiert.
- Der benannte Amazon S3 S3-Bucket `BUCKET NAME` enthält das `FILENAME.desc` generierte von `PROTO_FILENAME.proto`.

Nachdem Sie die Konfiguration abgeschlossen haben, veröffentlichen Sie eine Nachricht zu dem AWS IoT Core Thema, das die Regel abonniert hat.

Einschränkungen

AWS IoT Core Regeln unterstützen Protobuf mit den folgenden Einschränkungen:

- Die Dekodierung von protobuf-Nachrichtennutzlasten innerhalb von Ersatzvorlagen wird nicht unterstützt.
- Beim Dekodieren von protobuf-Nachrichtennutzlasten können Sie die Funktion „SQL dekodieren“ innerhalb eines einzelnen SQL-Ausdrucks bis zu zweimal verwenden.
- Die maximale Größe der eingehenden Nutzlast beträgt 128 KiB (1 = 1024 Byte), die maximale Größe der ausgehenden Nutzlast beträgt 128 KiB und die maximale Größe für ein `FileDescriptorSet`-Objekt, das in einem Amazon S3-Bucket gespeichert ist, beträgt 32 KiB.
- Mit SSE-C Verschlüsselung verschlüsselte Amazon S3 S3-Buckets werden nicht unterstützt.

Bewährte Methoden

Nachfolgend Best Practices und Tipps zur Fehlerbehebung.

- Laden Sie die proto-Dateien in den Amazon S3 Bucket hoch.

  Es empfiehlt sich, die proto-Dateien zu sichern, falls etwas schief geht. Wenn Sie z. B. bei der Ausführung von protoc die proto-Dateien fälschlicherweise ohne Sicherungen ändern, kann dies zu Problemen in Ihrem Produktionsstapel führen. Es gibt mehrere Möglichkeiten, Dateien in einem Amazon S3-Bucket zu sichern. Sie können beispielsweise die Versionierung in S3-Buckets verwenden. Weitere Informationen zum Sichern von Dateien in Amazon S3-Buckets im Amazon S3 Developer Guide.
- Konfigurieren Sie die AWS IoT Protokollierung, um Protokolleinträge anzuzeigen.

  Es empfiehlt sich, die AWS IoT Protokollierung so zu konfigurieren, dass Sie die AWS IoT Protokolle für Ihr Konto überprüfen können CloudWatch. Wenn die SQL-Abfrage einer Regel eine externe Funktion aufruft, generiert AWS IoT Core Rules einen Protokolleintrag mit dem Wert `eventType` of`FunctionExecution`, der das Feld „Grund“ enthält, das Ihnen bei der Behebung von Fehlern hilft. Zu den möglichen Fehlern gehören ein Amazon S3 S3-Objekt, das nicht gefunden wurde, oder ein ungültiger protobuf-Dateideskriptor. Weitere Informationen zur Konfiguration der AWS IoT -Protokollierung und zum Anzeigen der Protokolleinträge unter AWS IoT Protokollierung konfigurieren und Protokolleinträge der Rules Engine.
- Aktualisieren Sie `FileDescriptorSet` mit einem neuen Objektschlüssel und aktualisieren Sie den Objektschlüssel in der Regel.

  Zum Aktualisieren von `FileDescriptorSet` laden Sie eine aktualisierte Deskriptordatei in den Amazon S3 S3-Bucket hoch. Es kann bis zu 15 Minuten dauern, bis `FileDescriptorSet`-Aktualisierungen angezeigt werden. Es empfiehlt sich, die aktualisierte `FileDescriptorSet` mit einem neuen Objektschlüssel hochzuladen und den Objektschlüssel in der Regel zu aktualisieren.

SQL-Versionen

Die AWS IoT Regel-Engine verwendet eine SQL-like Syntax, um Daten aus MQTT-Nachrichten auszuwählen. Die SQL-Anweisungen werden auf Grundlage einer SQL-Version interpretiert, die mit der Eigenschaft `awsIotSqlVersion` in einem JSON-Dokument angegeben ist, welches die Regel beschreibt. Weitere Informationen zur Struktur von JSON-Regeldokumenten unter Erstellen einer Regel. Mit der `awsIotSqlVersion` Eigenschaft können Sie angeben, welche Version der AWS IoT SQL-Regel-Engine Sie verwenden möchten. Wenn eine neue Version bereitgestellt wird, können Sie

weiterhin eine frühere Version verwenden oder die Regel ändern, damit sie die neue Version nutzt. Die aktuellen Regeln verwenden weiterhin die Version, mit der sie erstellt wurden.

Das folgende JSON-Beispiel zeigt, wie die SQL-Version mit der Eigenschaft `awsIotSqlVersion` angegeben wird:

```
{
    "sql": "expression",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [{
        "republish": {
            "topic": "my-mqtt-topic",
            "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
        }
    }]
}
```

AWS IoT unterstützt derzeit die folgenden SQL-Versionen:

- `2016-03-23` – Die SQL-Version von 23.03.2016 (empfohlen).
- `2015-10-08` – Die ursprüngliche SQL-Version von 08.10.2015.
- `beta` – Die neueste Beta-SQL-Version. Mit dieser Version können Sie die Regeln grundlegend ändern.

Neuerungen in der Version der SQL-Regel-Engine vom 23.03.2016

- Lösungen für das Auswählen verschachtelter JSON-Objekte
- Lösungen für Array-Abfragen
- Intra-object Unterstützung für Abfragen. Weitere Informationen finden Sie unter Verschachtelte Objektabfragen.
- Unterstützung zum Ausgeben eines Arrays als oberstes Objekt
- Hinzufügen der `encode(value, encodingScheme)`-Funktion, die auf JSON- und Nicht-JSON-Formatdaten angewendet werden kann. Weitere Informationen finden Sie unter Encode-Funktion.

Gibt ein **`Array`** als Objekt der obersten Ebene aus

Diese Funktion ermöglicht eine Regel, mit der ein Array als oberstes Objekt zurückgegeben wird. Als Beispiel dient etwa die folgende MQTT-Nachricht:

```
{
    "a": {"b":"c"},
    "arr":[1,2,3,4]
}
```

Und die folgende Regel:

```
SELECT VALUE arr FROM 'topic'
```

Die Regel generiert folgenden Output:

```
[1,2,3,4]
```

Reservierte Zeichenfolge in AWS IoT Regelaktionen

AWS IoT Regeln reservieren die ^{ Zeichenfolge. Sie können es nicht in Konfigurationsfeldern für Regelaktionen verwenden. Wenn ein Konfigurationsfeld diese Sequenz enthält, schlagen die `ReplaceTopicRule` Operationen `CreateTopicRule` und mit einem Fehler fehl. Entfernen oder ändern Sie die Reihenfolge und senden Sie die Anfrage dann erneut.

AWS IoT Geräteschatten-Service

Der AWS IoT Device Shadow-Dienst fügt AWS IoT Dingobjekten Schatten hinzu. Schatten können den Status eines Geräts für Apps und andere Dienste verfügbar machen, unabhängig davon, ob das Gerät mit dem Gerät verbunden ist AWS IoT oder nicht. AWS IoT Ding-Objekte können mehrere benannte Shadows haben, sodass Ihre IoT-Lösung mehr Optionen zum Verbinden Ihrer Geräte mit anderen Apps und Diensten bietet.

AWS IoT Ding-Objekte haben keine Schatten, bis sie explizit erstellt werden. Schatten können mithilfe der AWS IoT Konsole erstellt, aktualisiert und gelöscht werden. Geräte, andere Webclients und Services können Schatten erstellen, aktualisieren und löschen mithilfe von MQTT und den reservierten MQTT-Themen; HTTP mithilfe der Geräteschatten-REST-API; und der AWS CLI für AWS IoT Da Schatten AWS in der Cloud gespeichert werden, können sie Gerätestatusdaten von Apps und anderen Cloud-Diensten sammeln und melden, unabhängig davon, ob das Gerät angeschlossen ist oder nicht.

Verwendung von Schatten

Schatten bieten einen zuverlässigen Datenspeicher für Geräte, Anwendungen und andere Cloud-Services, um Daten gemeinsam zu nutzen. Sie ermöglichen es Geräten, Anwendungen und anderen Cloud-Services, eine Verbindung herzustellen und zu trennen, ohne den Status eines Geräts zu verlieren.

Geräte, Apps und andere Cloud-Dienste sind zwar mit Geräten, Apps und anderen Cloud-Diensten verbunden AWS IoT, können aber über dessen Schatten auf den aktuellen Zustand eines Geräts zugreifen und ihn kontrollieren. Beispielsweise kann eine App eine Änderung des Gerätezustands anfordern, indem sie einen Shadow aktualisiert. AWS IoT veröffentlicht eine Meldung, die auf die Änderung des Geräts hinweist. Das Gerät empfängt diese Nachricht, aktualisiert seinen Status so, dass er übereinstimmt, und veröffentlicht eine Nachricht mit dem aktualisierten Status. Der Geräteschatten-Service spiegelt diesen aktualisierten Status im entsprechenden Schatten wider. Die Anwendung kann das Update des Schattens abonnieren oder den Schatten nach seinem aktuellen Status abfragen.

Wenn ein Gerät offline geht, kann eine App immer noch mit AWS IoT den Schatten des Geräts kommunizieren. Wenn das Gerät erneut eine Verbindung herstellt, erhält es den aktuellen Status seiner Schatten. Das Gerät kann seinen Status so aktualisieren, dass er mit dem seiner Schatten

übereinstimmt, und dann eine Nachricht mit dem aktualisierten Status veröffentlichen. Wenn eine Anwendung offline ist und sich der Gerätestatus ändert, während sie offline ist, hält das Gerät den Schatten aktualisiert, damit die Anwendung die Schatten nach seinem aktuellen Status abfragen kann, wenn sie erneut eine Verbindung herstellt.

Wenn die Geräte häufig offline sind und Sie die Geräte so konfigurieren möchten, dass sie Deltameldungen empfangen, nachdem sie wieder eine Verbindung hergestellt haben, können Sie die Funktion für persistente Sitzungen verwenden. Weitere Informationen zum Ablauf der persistenten Sitzung finden Sie unter Ablauffrist für persistente Sitzungen.

Auswählen der Verwendung benannter oder unbenannter Schatten

Der Geräteschatten-Service unterstützt benannte und unbenannte (klassische) Schatten. Ein Objekt kann mehrere benannte Schatten und nicht mehr als einen unbenannten, klassischen Schatten haben. Das Objekt kann auch einen reservierten benannten Schatten haben, der ähnlich wie ein benannter Schatten funktioniert, außer dass Sie seinen Namen nicht aktualisieren können. Weitere Informationen finden Sie unter Reservierter benannter Schatten.

Ein Objekt kann sowohl benannte als auch unbenannte Schatten gleichzeitig haben. Die API, die für den jeweiligen Zugriff verwendet wird, unterscheidet sich jedoch leicht. Daher ist es möglicherweise effizienter, zu entscheiden, welcher Schattentyp für Ihre Lösung am besten geeignet ist, und nur diesen Typ zu verwenden. Weitere Informationen zur API für den Zugriff auf die Schatten finden Sie unter Schatten-Themen.

Mit benannten Schatten können Sie verschiedene Ansichten des Status eines Objekts erstellen. Beispielsweise könnten Sie ein Objekt mit vielen Eigenschaften in Schatten mit logischen Eigenschaftengruppen unterteilen, die jeweils durch einen Schattennamen gekennzeichnet sind. Sie können den Zugriff auf Eigenschaften auch einschränken, indem Sie sie in verschiedene Schatten gruppieren und Richtlinien verwenden, um den Zugriff zu steuern. Weitere Informationen zu Richtlinien, die mit Geräteschatten verwendet werden können, finden Sie unter Aktionen, Ressourcen und Bedingungsschlüssel für AWS IoT und AWS IoT Core Richtlinien.

Die klassischen, unbenannten Schatten sind einfacher, aber etwas stärker eingeschränkt als die benannten Schatten. Jedes AWS IoT Objekt kann nur einen unbenannten Schatten haben. Wenn Sie erwarten, dass Ihre IoT-Lösung nur einen begrenzten Bedarf an Schattendaten hat, sollten Sie auf diese Weise mit der Verwendung von Schatten beginnen. Wenn Sie jedoch der Meinung sind, dass Sie in Zukunft weitere Schatten hinzufügen möchten, sollten Sie von Anfang an benannte Schatten verwenden.

Die Flottenindizierung unterstützt unbenannte Schatten und benannte Schatten unterschiedlich. Weitere Informationen finden Sie unter Verwalten der Flottenindizierung.

Zugreifen auf Schatten

Jeder Schatten verfügt über ein reserviertes MQTT-Thema und eine HTTP-URL, die die `get`-, `update`- und `delete`-Aktionen für den Schatten unterstützt.

Schatten verwenden JSON-Schattendokumente zum Speichern und Abrufen von Daten. Das Dokument eines Schattens enthält eine Statuseigenschaft, die die folgenden Aspekte des Gerätezustands beschreibt:

- `desired`

  Apps geben die gewünschten Status der Geräteeigenschaften an, indem sie das `desired`-Objekt aktualisieren.
- `reported`

  Geräte melden ihren aktuellen Status im `reported`-Objekt.
- `delta`

  AWS IoT meldet Unterschiede zwischen dem gewünschten und dem gemeldeten Status im `delta` Objekt.

Die in einem Schatten gespeicherten Daten werden durch die Statuseigenschaft des Nachrichtentexts der Aktualisierungsaktion bestimmt. Nachfolgende Aktualisierungsaktionen können die Werte eines vorhandenen Datenobjekts ändern und Schlüssel und andere Elemente im Zustandsobjekt des Schattens hinzufügen und löschen. Weitere Informationen zum Zugriff auf Schatten finden Sie unter Verwenden von Schatten in Geräten und Verwenden von Schatten in Apps und Services.

> **Important**
>
> Die Berechtigung, Aktualisierungsanforderungen zu stellen, sollte auf vertrauenswürdige Apps und Geräte beschränkt sein. Dadurch wird verhindert, dass die Statuseigenschaft des Schattens unerwartet geändert wird. Andernfalls sollten die Geräte und Apps, die den Schatten verwenden, so konzipiert werden, dass sie erwarten, dass sich die Schlüssel in der Statuseigenschaft ändern.

Verwenden von Schatten in Geräten, Apps und anderen Cloudservices

Die Verwendung von Schatten in Geräten, Apps und anderen Cloudservices erfordert Konsistenz und Koordination zwischen all diesen Services. Der AWS IoT Device Shadow-Dienst speichert den Schattenstatus, sendet Nachrichten, wenn sich der Schattenstatus ändert, und reagiert auf Nachrichten, die seinen Status ändern. Die Geräte, Apps und anderen Cloud-Services in Ihrer IoT-Lösung müssen ihren Status verwalten und den Status des Geräteschattens konsistent halten.

Die Schattenstatusdaten sind dynamisch und können von Geräten, Apps und anderen Cloud-Services mit Berechtigung zum Zugriff auf den Schatten geändert werden. Aus diesem Grund ist es wichtig zu berücksichtigen, wie jedes Gerät, jede App und jeder andere Cloud-Service mit dem Schatten interagiert. Beispiel:

- Geräte sollten nur in die `reported`-Eigenschaft des Schattenstatus schreiben, wenn Statusdaten an den Schatten übertragen werden.
- Apps und andere Cloud-Services sollten nur in die `desired`-Eigenschaft schreiben, wenn Statusänderungsanforderungen über den Schatten an das Gerät übermittelt werden.

> **Important**
>
> Die in einem Schattendatenobjekt enthaltenen Daten sind unabhängig von anderen Schatten und anderen Objekteigenschaften, z. B. den Attributen eines Objekts und dem Inhalt von MQTT-Nachrichten, die das Gerät eines Objekts veröffentlichen könnte. Ein Gerät kann jedoch bei Bedarf dieselben Daten in verschiedenen MQTT-Themen und -Schatten melden. Ein Gerät, das mehrere Schatten unterstützt, muss die Konsistenz der Daten aufrechterhalten, die es in den verschiedenen Schatten meldet.

Nachrichtenreihenfolge

Es gibt keine Garantie dafür, dass Nachrichten vom AWS IoT Dienst in einer bestimmten Reihenfolge auf dem Gerät ankommen. Das folgende Szenario zeigt, was in diesem Fall passiert.

Ursprüngliches Statusdokument:

```
{
  "state": {
    "reported": {
```

```
      "color": "blue"
    }
  },
  "version": 9,
  "timestamp": 123456776
}
```

Aktualisierung 1:

```
{
  "state": {
    "desired": {
      "color": "RED"
    }
  },
  "version": 10,
  "timestamp": 123456777
}
```

Aktualisierung 2:

```
{
  "state": {
    "desired": {
      "color": "GREEN"
    }
  },
  "version": 11,
  "timestamp": 123456778
}
```

Endgültiges Statusdokument:

```
{
  "state": {
    "reported": {
      "color": "GREEN"
    }
  },
  "version": 12,
  "timestamp": 123456779
}
```

Dies führt zu zwei Delta-Nachrichten:

```
{
  "state": {
    "color": "RED"
  },
  "version": 11,
  "timestamp": 123456778
}
```

```
{
  "state": {
    "color": "GREEN"
  },
  "version": 12,
  "timestamp": 123456779
}
```

Das Gerät kann diese Nachrichten ohne feste Reihenfolge erhalten. Da der Status in diesen Nachrichten kumulativer Natur ist, kann ein Gerät Nachrichten, die eine Versionsnummer enthalten, die älter ist als die, die es verfolgt, sicher verwerfen. Erhält das Gerät das Delta für die Version 12 vor der Version 11 kann es die Nachricht zur Version 11 sicher verwerfen.

Kürzen von Shadow-Nachrichten

Um die Größe der Schatten-Nachrichten, die an Ihr Gerät gesendet werden, zu reduzieren, legen Sie eine Regel fest, mit der nur die Felder, die Ihr Gerät benötigt, ausgewählt und die Nachricht in einem MQTT-Thema, das Ihr Gerät überwacht, erneut veröffentlicht werden.

Die Regel wird in der JSON vorgegeben und sollte wie folgt aussehen:

```
{
  "sql": "SELECT state, version FROM '$aws/things/+/shadow/update/delta'",
  "ruleDisabled": false,
  "actions": [
    {
      "republish": {
        "topic": "${topic(3)}/delta",
        "roleArn": "arn:aws:iam:123456789012:role/my-iot-role"
      }
    }
  ]
```

```
}
```

Mit der SELECT-Anweisung wird festgelegt, welche Felder der Nachricht im vorgegebenen Topic erneut veröffentlicht werden. Der Platzhalter "+" wird verwendet, um allen Schattennamen zu entsprechen. Mit der Regel wird festgelegt, dass alle passenden Nachrichten im vorgegebenen Topic erneut veröffentlicht werden sollen. In diesem Fall wird die Funktion `"topic()"` verwendet, um das Thema anzugeben, in dem erneut eine Veröffentlichung erfolgen soll. `topic(3)` ermittelt den Objektnamen im Ursprungs-Thema. Weitere Informationen zum Erstellen von Regeln finden Sie unter Regeln für AWS IoT.

Verwenden von Schatten in Geräten

In diesem Abschnitt wird die Gerätekommunikation mit Shadows mithilfe von MQTT-Nachrichten beschrieben. Dies ist die bevorzugte Methode für Geräte, um mit dem AWS IoT Device Shadow-Dienst zu kommunizieren.

Die Shadow-Kommunikation emuliert ein request/response Modell unter Verwendung des publish/subscribe Kommunikationsmodells von MQTT. Jede Schattenaktion besteht aus einem Anforderungsthema, einem Thema für erfolgreiche Antworten (`accepted`) und einem Thema für Fehlerantworten (`rejected`).

Informationen dazu, ob Apps und Services feststellen können, ob ein Gerät verbunden ist, finden Sie unter Erkennt, ob ein Gerät angeschlossen ist.

> **Important**
>
> Da MQTT ein publish/subscribe Kommunikationsmodell verwendet, sollten Sie die Antwortthemen abonnieren, bevor Sie ein Anforderungsthema veröffentlichen. Wenn Sie dies nicht tun, erhalten Sie keine Antwort auf die Anfrage, die Sie veröffentlichen. Wenn Sie eine AWS IoT Device SDK zum Aufrufen der Geräteschatten-Dienst-APIs verwenden, wird dies für Sie erledigt.

Die Beispiele in diesem Abschnitt verwenden eine abgekürzte Form des Themas, wobei sich das Thema entweder auf einen benannten oder einen unbenannten Shadow beziehen *`ShadowTopicPrefix`* kann, wie in dieser Tabelle beschrieben.

Schatten können benannt oder unbenannt sein (klassisch). Die jeweils verwendeten Themen unterscheiden sich nur durch das Themenpräfix. In dieser Tabelle wird das Themenpräfix angezeigt, das von jedem Schattentyp verwendet wird.

ShadowTopicPrefix Wert	Schattentyp
`$aws/things/ thingName /shadow`	Unbenannter (klassischer) Schatten
`$aws/things/ thingName /shadow/name/ shadowName`	Benannter Schatten

> **Important**
>
> Stellen Sie sicher, dass die Verwendung der Schatten durch Ihre App oder Ihren Service konsistent ist und von den entsprechenden Implementierungen auf Ihren Geräten unterstützt wird. Bedenken Sie beispielsweise, wie Schatten erstellt, aktualisiert und gelöscht werden. Berücksichtigen Sie auch, wie Updates auf dem Gerät und in den Apps oder Services behandelt werden, die über einen Schatten auf das Gerät zugreifen. Ihr Design sollte klar angeben, wie der Zustand des Geräts aktualisiert und gemeldet wird und wie Ihre Apps und Services mit dem Gerät und seinen Schatten interagieren.

Um ein vollständiges Thema zu erstellen, wählen Sie *ShadowTopicPrefix* als Schattentyp aus, auf den Sie verweisen möchten. Ersetzen Sie *thingName* und *shadowName* mit den entsprechenden Werten, wenn zutreffend. Fügen Sie anschließend den Themen-Stub an wie in der folgenden Tabelle dargestellt. Denken Sie daran, dass bei Themen zwischen Groß- und Kleinschreibung unterschieden wird.

Weitere Informationen zu den reservierten Themen für Schatten finden Sie unter Schatten-Themen.

Initialisierung des Geräts bei der ersten Verbindung mit AWS IoT

Nachdem sich ein Gerät bei uns registriert hat AWS IoT, sollte es diese MQTT-Nachrichten für die Schatten abonnieren, die es unterstützt.

Thema	Bedeutung	Aktion, die ein Gerät ausführen sollte, wenn dieses Thema empfangen wird
ShadowTopicPrefix / delete/accepted	Die delete Anfrage wurde akzeptiert und der Shadow wurde AWS IoT gelöscht.	Die Aktionen, die für den gelöschten Schatten erforderl ich sind, z. B. das Beenden der Veröffentlichung von Aktualisierungen.
ShadowTopicPrefix / delete/rejected	Die delete Anfrage wurde von abgelehnt AWS IoT und der Shadow wurde nicht gelöscht. Der Nachrichtentext enthält die Fehlerinformatione n.	Reagieren Sie auf die Fehlermeldung im Nachricht entext.
ShadowTopicPrefix / get/accepted	Die get Anfrage wurde von akzeptiert AWS IoT, und der Nachrichtentext enthält das aktuelle Shadow-Dokument.	Die Aktionen, die erforderl ich sind, um das Statusdok ument im Nachrichtentext zu verarbeiten.
ShadowTopicPrefix / get/rejected	Die get Anfrage wurde von abgelehnt AWS IoT, und der Nachrichtentext enthält die Fehlerinformationen.	Reagieren Sie auf die Fehlermeldung im Nachricht entext.
ShadowTopicPrefix / update/accepted	Die update Anfrage wurde von akzeptiert AWS IoT, und der Nachrichtentext enthält das aktuelle Schattend okument.	Bestätigen Sie, dass die aktualisierten Daten im Nachrichtentext mit dem Gerätestatus übereinstimmen.
ShadowTopicPrefix / update/rejected	Die update Anfrage wurde von abgelehnt AWS IoT, und der Nachrichtentext enthält die Fehlerinformationen.	Reagieren Sie auf die Fehlermeldung im Nachricht entext.

Thema	Bedeutung	Aktion, die ein Gerät ausführen sollte, wenn dieses Thema empfangen wird
ShadowTopicPrefix /update/delta	Das Schattendokument wurde durch eine Anfrage an aktualisiert AWS IoT, und der Nachrichtentext enthält die angeforderten Änderungen.	Aktualisieren Sie den Gerätestatus so, dass er mit dem gewünschten Status im Nachrichtentext übereinst immt.
ShadowTopicPrefix /update/documents	Eine Aktualisierung des Schattens wurde kürzlich abgeschlossen, und der Nachrichtentext enthält das aktuelle Schattendokument.	Bestätigen Sie, dass der aktualisierte Status im Nachrichtentext mit dem Gerätestatus übereinstimmt.

Nach dem Abonnieren der Nachrichten in der obigen Tabelle für jeden Schatten sollte das Gerät testen, ob die Schatten, die es unterstützt, bereits erstellt wurden, indem es ein `/get`-Thema in jedem Schatten veröffentlicht. Wenn eine `/get/accepted`-Nachricht empfangen wird, enthält der Nachrichtentext das Schattendokument, mit dem das Gerät seinen Status initialisieren kann. Wenn eine `/get/rejected`-Nachricht empfangen wird, sollte der Schatten erstellt werden, indem eine `/update`-Nachricht mit dem aktuellen Gerätestatus veröffentlicht wird.

Nehmen wir zum Beispiel an, Sie haben ein Objekt `My_IoT_Thing`, das keine klassischen oder benannten Schatten hat. Wenn Sie jetzt eine `/get`-Anfrage zum reservierten Thema `$aws/things/My_IoT_Thing/shadow/get` veröffentlichen, erhält das `$aws/things/My_IoT_Thing/shadow/get/rejected` Thema einen Fehler, da das Objekt keine Schatten hat. Um diesen Fehler zu beheben, veröffentlichen Sie zunächst eine `/update`-Nachricht, indem Sie das `$aws/things/My_IoT_Thing/shadow/update` Thema mit dem aktuellen Gerätestatus verwenden, z. B. die folgende Payload.

```
{
 "state": {
  "reported": {
   "welcome": "aws-iot",
   "color": "yellow"
  }
 }
```

```
}
```

Für das Objekt wird jetzt ein klassischer Schatten erstellt, und die Nachricht wird auf dem `$aws/things/My_IoT_Thing/shadow/update/accepted`-Thema veröffentlicht. Wenn Sie zu dem Thema `$aws/things/My_IoT_Thing/shadow/get` veröffentlichen, wird eine Antwort auf das `$aws/things/My_IoT_Thing/shadow/get/accepted`-Thema mit dem Gerätestatus zurückgegeben.

Bei benannten Schatten müssen Sie zuerst den benannten Schatten erstellen oder ein Update mit dem Schatten-Namen veröffentlichen, bevor Sie die GET-Anfrage verwenden können. Um beispielsweise einen benannten Schatten `namedShadow1` zu erstellen, müssen Sie zunächst die Informationen zum Gerätestatus für das Thema veröffentlichen`$aws/things/My_IoT_Thing/shadow/name/namedShadow1/update`. Um die Statusinformationen abzurufen, verwenden Sie die `/get`-Anfrage für den benannten Schatten, `$aws/things/My_IoT_Thing/shadow/name/namedShadow1/get`.

Nachrichten werden verarbeitet, während das Gerät angeschlossen ist AWS IoT

Solange ein Gerät angeschlossen ist AWS IoT, kann es update/delta /-Meldungen empfangen und sollte dafür sorgen, dass der Gerätestatus an die Änderungen in seinen Schatten angepasst wird, und zwar durch:

1. Alle empfangenen update/delta /-Meldungen werden gelesen und der Gerätestatus wird synchronisiert, damit er dem entspricht.
2. Veröffentlichen einer /update-Nachricht mit einem `reported`-Nachrichtentext, der den aktuellen Status des Geräts hat, wenn sich der Gerätestatus ändert.

Solange ein Gerät angeschlossen ist, sollte es diese Meldungen veröffentlichen, wenn angezeigt.

Indikation	Thema	Nutzlast
Der Zustand des Geräts hat sich geändert.	*ShadowTopicPrefix* `/update`	Ein Schattendokument mit der `reported`-Eigenschaft.

Indikation	Thema	Nutzlast
Das Gerät wird möglicher weise nicht mit dem Schatten synchronisiert.	*ShadowTopicPrefix* /get	(empty)
Eine Aktion auf dem Gerät zeigt an, dass ein Schatten vom Gerät nicht mehr unterstützt wird, z. B. wenn das Gerät entfernt oder ersetzt wird.	*ShadowTopicPrefix* /delete	(empty)

Verarbeitung von Nachrichten, wenn das Gerät wieder eine Verbindung herstellt AWS IoT

Wenn ein Gerät mit einem oder mehreren Schatten eine Verbindung herstellt AWS IoT, sollte es seinen Status mit dem aller unterstützten Schatten synchronisieren, indem es:

1. Es werden alle empfangenen update/delta /-Nachrichten gelesen und der Gerätestatus wird synchronisiert, damit er übereinstimmt.
2. Veröffentlichen einer /update-Nachricht mit einem `reported`-Nachrichtentext, der den aktuellen Status des Geräts hat.

Verwenden von Schatten in Apps und Services

In diesem Abschnitt wird beschrieben, wie eine App oder ein Dienst mit dem AWS IoT Device Shadow-Dienst interagiert. In diesem Beispiel wird davon ausgegangen, dass die App oder der Service nur mit dem Schatten und darüber dem Gerät interagiert. In diesem Beispiel sind keine Verwaltungsaktionen enthalten, z. B. das Erstellen oder Löschen von Schatten.

In diesem Beispiel wird die REST-API des AWS IoT Device Shadow-Dienstes verwendet, um mit Shadows zu interagieren. Im Gegensatz zu dem in verwendeten BeispielVerwenden von Schatten in Geräten, das ein publish/subscribe Kommunikationsmodell verwendet, verwendet dieses Beispiel das request/response Kommunikationsmodell der REST-API. Das bedeutet, dass die App oder der Dienst eine Anfrage stellen muss, bevor sie eine Antwort von erhalten kann AWS IoT. Ein

Nachteil dieses Modells ist jedoch, dass es keine Benachrichtigungen unterstützt. Wenn für Ihre App oder Ihr Dienst zeitnahe Benachrichtigungen über Änderungen des Gerätestatus erforderlich sind, ziehen Sie die Protokolle MQTT oder MQTT over WSS in Betracht, die das publish/subscribe Kommunikationsmodell unterstützen, wie unter beschrieben. Verwenden von Schatten in Geräten

> **Important**
>
> Stellen Sie sicher, dass die Verwendung der Schatten Ihrer App oder Ihres Services mit den entsprechenden Implementierungen in Ihren Geräten konsistent ist und von diesen unterstützt wird. Berücksichtigen Sie beispielsweise, wie Schatten erstellt, aktualisiert und gelöscht werden und wie Updates auf dem Gerät und in den Apps oder Services, die auf den Schatten zugreifen, gehandhabt werden. In Ihrem Design sollte klar angegeben werden, wie der Zustand des Geräts aktualisiert und gemeldet wird und wie Ihre Apps und Services mit dem Gerät und seinen Schatten interagieren.

Die URL der REST-API für einen benannten Schatten lautet:

```
https://endpoint/things/thingName/shadow?name=shadowName
```

und für einen unbenannten Schatten:

```
https://endpoint/things/thingName/shadow
```

Wobei:

Endpunkt

Der vom CLI-Befehl zurückgegebene Endpunkt:

```
aws iot describe-endpoint --endpoint-type IOT:Data-ATS
```

thingName

Der Name des Objekts, zu dem der Schatten gehört

shadowName

Der Name des benannten Schattens. Dieser Parameter wird mit unbenannten Schatten nicht verwendet.

Initialisierung der App oder des Dienstes beim Herstellen einer Verbindung zu AWS IoT

Wenn die App zum ersten Mal eine Verbindung herstellt AWS IoT, sollte sie eine HTTP GET-Anfrage an die URLs der Shadows senden, die sie verwendet, um den aktuellen Status der verwendeten Shadows abzurufen. Dies ermöglicht es, die App oder den Service mit dem Schatten zu synchronisieren.

Der Verarbeitungsstatus ändert sich, während die App oder der Dienst verbunden ist AWS IoT

Während die App oder der Dienst verbunden ist AWS IoT, kann sie den aktuellen Status in regelmäßigen Abständen abfragen, indem sie eine HTTP GET-Anfrage an die URLs der verwendeten Shadows sendet.

Wenn ein Endbenutzer mit der App oder dem Service interagiert, um den Status des Geräts zu ändern, kann die App oder der Service eine HTTP-POST-Anforderung an die URLs der Schatten senden, mit denen der `desired`-Status des Schattens aktualisiert wird. Diese Anforderung gibt die Änderung zurück, die akzeptiert wurde, aber Sie müssen möglicherweise den Schatten abfragen, indem Sie HTTP-GET-Anforderungen vornehmen, bis das Gerät den Schatten mit seinem neuen Status aktualisiert hat.

Erkennt, ob ein Gerät angeschlossen ist

Um festzustellen, ob ein Gerät derzeit verbunden ist, schließen Sie eine `connected`-Eigenschaft in das Schattendokument ein und verwenden eine MQTT Last Will and Testament (LWT)-Meldung, um die `connected`-Eigenschaft auf `false` festzulegen, wenn ein Gerät aufgrund eines Fehlers getrennt wird.

> **Note**
>
> MQTT-LWT-Nachrichten, die an AWS IoT reservierte Themen gesendet werden (Themen, die mit $ beginnen), werden vom AWS IoT Device Shadow-Dienst ignoriert. Sie werden jedoch von abonnierten Clients und von der AWS IoT Regel-Engine verarbeitet. Sie müssen also eine LWT-Nachricht erstellen, die an ein nicht reserviertes Thema gesendet wird, und eine Regel, die die MQTT-LWT-Nachricht als Shadow-

Aktualisierungsnachricht im reservierten Aktualisierungsthema des Shadows erneut veröffentlicht. *ShadowTopicPrefix*/update

So senden Sie dem Device Shadow-Service eine LWT-Nachricht

1. Erstellen Sie eine Regel, die die MQTT LWT-Nachricht im reservierten Thema erneut veröffentlicht. Das folgende Beispiel ist eine Regel, die auf Nachrichten zu dem my/things/myLightBulb/update-Thema wartet und sie erneut in $aws/things/myLightBulb/shadow/update veröffentlicht.

```
{
    "rule": {
    "ruleDisabled": false,
    "sql": "SELECT * FROM 'my/things/myLightBulb/update'",
    "description": "Turn my/things/ into $aws/things/",
    "actions": [
        {
        "republish": {
            "topic": "$$aws/things/myLightBulb/shadow/update",
            "roleArn": "arn:aws:iam:123456789012:role/aws_iot_republish"
            }
        }
     ]
   }
}
```

2. Wenn das Gerät eine Verbindung zu einem nicht reservierten Thema herstellt AWS IoT, registriert es eine LWT-Nachricht, die von der Regel zur erneuten Veröffentlichung erkannt wird. In diesem Beispiel ist dieses Thema my/things/myLightBulb/update, und die verbundene Eigenschaft wird auf false festgelegt.

```
{
    "state": {
        "reported": {
            "connected":"false"
        }
    }
}
```

3. Nach der Verbindung veröffentlicht das Gerät eine Nachricht zu seinem Shadow-Update-Thema, `$aws/things/myLightBulb/shadow/update`, um seinen aktuellen Status zu melden, einschließlich der Einstellung seiner `connected`-Eigenschaft auf `true`.

```
{
     "state": {
        "reported": {
            "connected":"true"
        }
    }
}
```

4. Bevor das Gerät die Verbindung ordnungsgemäß trennt, veröffentlicht es eine Nachricht zu seinem Schattenaktualisierungsthema, `$aws/things/myLightBulb/shadow/update`, um seinen neuesten Status zu melden, einschließlich der Einstellung seiner `connected`-Eigenschaft auf `false`.

```
{
    "state": {
        "reported": {
            "connected":"false"
        }
    }
}
```

5. Wenn das Gerät aufgrund eines Fehlers die Verbindung trennt, veröffentlicht der AWS IoT Message Broker die LWT-Nachricht des Geräts im Namen des Geräts. Die Regel zum erneuten Veröffentlichen erkennt diese Nachricht und veröffentlicht die Schattenaktualisierungsmeldung, um die `connected`-Eigenschaft des Geräteschattens zu aktualisieren.

> **Note**
>
> Aufgrund der asynchronen Verarbeitung von Verbindungsabbrüchen kann nicht garantiert werden, dass LWT-Nachrichten bei der Wiederherstellung der Verbindung in der richtigen Reihenfolge versendet werden. Wir empfehlen, Lebenszyklusereignisse zu verwenden, um die Genauigkeit der Erkennung des Verbindungsstatus zu verbessern, da diese Ereignisse Attribute für die Verwaltung von Ereignissen bereitstellen, die nicht in der richtigen Reihenfolge liegen.

Simulieren der Device Shadow-Servicekommunikation

In diesem Thema wird veranschaulicht, wie der Device Shadow-Service als Vermittler fungiert und es Geräten und Apps ermöglicht, einen Schatten zum Aktualisieren, Speichern und Abrufen des Status eines Geräts zu verwenden.

Um die in diesem Thema beschriebene Interaktion zu demonstrieren und sie weiter zu untersuchen, benötigen Sie ein AWS-Konto System, auf dem Sie das ausführen können. AWS CLI Wenn Sie diese nicht haben, können Sie die Interaktion immer anhand der Codebeispiele untersuchen.

In diesem Beispiel steht die AWS IoT Konsole für das Gerät. Das AWS CLI steht für die App oder den Dienst, der über den Shadow auf das Gerät zugreift. Die AWS CLI Schnittstelle ist der API sehr ähnlich, mit AWS IoT der eine App möglicherweise kommuniziert. Das Gerät in diesem Beispiel ist eine intelligente Glühlampe, und die App zeigt den Status der Glühlampe an und kann ihren Status ändern.

Einrichten der Simulation

Diese Verfahren initialisieren die Simulation, indem Sie die AWS IoT -Konsole öffnen, die Ihr Gerät simuliert, und das Befehlszeilenfenster, das Ihre App simuliert.

So richten Sie Ihre Simulationsumgebung ein:

1. Sie benötigen eine AWS-Konto , um die Beispiele aus diesem Thema selbst ausführen zu können. Wenn Sie noch keine haben AWS-Konto, erstellen Sie eine, wie unter beschriebenEinrichten AWS-Konto.
2. Öffnen Sie die AWS IoT -Konsole und wählen Sie im linken Menü Test, um den MQTT-Client zu öffnen.
3. Öffnen Sie in einem anderen Fenster ein Terminalfenster auf einem System, auf dem das AWS CLI installiert ist.

Sie sollten zwei Fenster geöffnet haben: eines mit der AWS IoT Konsole auf der Testseite und eines mit einer Befehlszeilenaufforderung.

Initialisieren des Geräts

In dieser Simulation arbeiten wir mit einem Objekt namens my SimulatedThing und seinem Schatten namens SimShadow1.

Erstellen des Objekts und seiner IoT-Richtlinie

Um ein Objekt zu erstellen, gehen Sie in der AWS IoT Konsole wie folgt vor:

1. Wählen Sie Verwalten und dann Things.
2. Klicken Sie auf die Schaltfläche Erstellen, wenn Dinge aufgeführt sind, andernfalls klicken Sie auf Einzelnes Objekt registrieren, um ein einzelnes Objekt zu erstellen. AWS IoT
3. Geben Sie den Namen `mySimulatedThing` ein, behalten Sie die Standardeinstellungen für andere Einstellungen bei und klicken Sie dann auf Weiter.
4. Generieren Sie mithilfe der Zertifikatserstellung mit nur einem Klick die Zertifikate, mit denen die Verbindung des Geräts mit AWS IoT authentifiziert wird. Klicken Sie auf Aktivieren, um das Zertifikat zu aktivieren.
5. Sie können die Richtlinie `My_IoT_Policy` anhängen, die dem Gerät die Erlaubnis erteilt, die reservierten MQTT-Themen zu veröffentlichen und zu abonnieren. Eine detailliertere Anleitung zum Erstellen einer AWS IoT Sache und zum Erstellen dieser Richtlinie finden Sie unterDies erstellt ein Objekt.

Erstellen Sie einen benannten Schatten für das Objekt.

Sie können einen benannten Schatten für ein Objekt erstellen, indem Sie eine Aktualisierungsanfrage für das Thema `$aws/things/mySimulatedThing/shadow/name/simShadow1/update`, wie unten beschrieben, veröffentlichen.

Oder um einen benannten Schatten zu erstellen:

1. Wählen Sie in der AWS IoT -Konsole Ihr Objekt in der Liste der angezeigten Objekte aus und wählen Sie dann Schatten.
2. Wählen Sie Schatten hinzufügen, geben Sie den Namen `simShadow1` ein und wählen Sie dann Erstellen, um den benannten Schatten hinzuzufügen.

Abonnieren und veröffentlichen Sie reservierte MQTT-Themen

Abonnieren Sie in der Konsole die reservierten MQTT-Schatten-Themen. Diese Themen sind die Antworten auf die Aktionen `get`, `update` und `delete`, damit Ihr Gerät bereit ist, die Antworten zu empfangen, nachdem es eine Aktion veröffentlicht hat.

Um ein MQTT-Thema zu abonnieren, finden Sie im MQTT-Client

1. Wählen Sie im MQTT-Client die Option In einem Thema veröffentlichen aus.
2. Geben Sie `get`, `update`, und `delete` Themen ein, die Sie abonnieren möchten. Kopieren Sie jeweils ein Thema aus der folgenden Liste, fügen Sie es in das Feld Themenfilter ein und klicken Sie dann auf Abonnieren. Die Themen müssten dann unter Abonnements aufgeführt werden.

   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/delete/accepted`
   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/delete/rejected`
   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/get/accepted`
   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/get/rejected`
   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/update/accepted`
   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/update/rejected`
   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/update/delta`
   - `$aws/things/mySimulatedThing/shadow/name/simShadow1/update/documents`

   An diesem Punkt ist Ihr simuliertes Gerät bereit, die Themen zu erhalten, wie sie von AWS IoT veröffentlicht werden.

Um zu einem MQTT-Thema zu veröffentlichen, finden Sie im MQTT-Client

Nachdem ein Gerät sich selbst initialisiert und die Antwortthemen abonniert hat, sollte Abfragen nach den unterstützten Schatten durchführen. Diese Simulation unterstützt nur einen Schatten, nämlich den Schatten, der ein Objekt namens my SimulatedThing, named simShadow1 unterstützt.

Um den aktuellen Schattenstatus aus dem MQTT-Client

1. Wählen Sie im MQTT-Client die Option Publish to a topic (In einem Thema veröffentlichen) aus.
2. Geben Sie unter Veröffentlichen folgendes Thema ein und löschen Sie alle Inhalte aus dem Nachrichtentextfenster, in dem Sie das Thema für GET eingegeben haben. Sie können dann In Thema veröffentlichen auswählen, um die Anfrage zu veröffentlichen. `$aws/things/mySimulatedThing/shadow/name/simShadow1/get`.

   Wenn Sie den benannten Schatten nicht erstellt haben, `simShadow1`, erhalten Sie eine Nachricht im `$aws/things/mySimulatedThing/shadow/name/simShadow1/get/`

`rejected` Thema, und der `code` ist `404`, wie in diesem Beispiel, und da der Schatten nicht erstellt wurde, erstellen wir ihn als Nächstes.

```
{
  "code": 404,
  "message": "No shadow exists with name: 'simShadow1'"
}
```

So erstellen Sie einen Schatten mit dem aktuellen Status des Geräts:

1. Wählen Sie im MQTT-Client die Option In einem Thema veröffentlichen aus.

   ```
   $aws/things/mySimulatedThing/shadow/name/simShadow1/update
   ```

2. Geben Sie im Nachrichtentextfenster, in dem Sie das Thema eingegeben haben, dieses Schattendokument ein, um anzuzeigen, dass das Gerät seine ID und seine aktuelle Farbe in RGB-Werten meldet. Wählen Sie Veröffentlichen, um die Anfrage zu veröffentlichen.

   ```
   {
     "state": {
       "reported": {
         "ID": "SmartLamp21",
         "ColorRGB": [
           128,
           128,
           128
         ]
       }
     },
     "clientToken": "426bfd96-e720-46d3-95cd-014e3ef12bb6"
   }
   ```

Wenn Sie eine Nachricht zum Thema erhalten:

- `$aws/things/mySimulatedThing/shadow/name/simShadow1/update/accepted`: Das bedeutet, dass der Schatten erstellt wurde und der Nachrichtentext das aktuelle Schattendokument enthält.
- `$aws/things/mySimulatedThing/shadow/name/simShadow1/update/rejected`: Überprüfen Sie den Fehler im Nachrichtentext.

- `$aws/things/mySimulatedThing/shadow/name/simShadow1/get/accepted`: Der Schatten ist bereits vorhanden und der Nachrichtentext hat den aktuellen Schattenstatus, wie in diesem Beispiel. Damit können Sie Ihr Gerät einstellen oder bestätigen, dass es mit dem Schattenstatus übereinstimmt.

```
{
  "state": {
    "reported": {
      "ID": "SmartLamp21",
      "ColorRGB": [
        128,
        128,
        128
      ]
    }
  },
  "metadata": {
    "reported": {
      "ID": {
        "timestamp": 1591140517
      },
      "ColorRGB": [
        {
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
        }
      ]
    }
  },
  "version": 3,
  "timestamp": 1591140517,
  "clientToken": "426bfd96-e720-46d3-95cd-014e3ef12bb6"
}
```

Senden einer Aktualisierung von der App

In diesem Abschnitt wird das verwendet AWS CLI , um zu demonstrieren, wie eine App mit einem Schatten interagieren kann.

Um den aktuellen Zustand des Schattens abzurufen, verwenden Sie den AWS CLI

Geben Sie in der Befehlszeile den folgenden Befehl ein.

```
aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name simShadow1 /
dev/stdout
```

Auf Windows-Plattformen können Sie `con` anstelle von `/dev/stdout` verwenden.

```
aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name simShadow1
 con
```

Da der Schatten vorhanden ist und vom Gerät initialisiert wurde, um seinen aktuellen Zustand wiederzugeben, sollte das folgende Schattendokument zurückgegeben werden.

```
{
  "state": {
    "reported": {
      "ID": "SmartLamp21",
      "ColorRGB": [
        128,
        128,
        128
      ]
    }
  },
  "metadata": {
    "reported": {
      "ID": {
        "timestamp": 1591140517
      },
      "ColorRGB": [
        {
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
```

```
        },
        {
          "timestamp": 1591140517
        }
      ]
    }
  },
  "version": 3,
  "timestamp": 1591141111
}
```

Die App kann diese Antwort verwenden, um die Darstellung des Gerätestatus zu initialisieren.

Wenn die App den Status aktualisiert, z. B. wenn ein Endbenutzer die Farbe unserer intelligenten Glühlampe zu Gelb ändert, sendet die App einen update-thing-shadow-Befehl. Dieser Befehl entspricht der `UpdateThingShadow`-REST-API.

So aktualisieren Sie einen Schatten aus einer App:

Geben Sie in der Befehlszeile den folgenden Befehl ein.

AWS CLI v2.x

```
aws iot-data update-thing-shadow --thing-name mySimulatedThing --shadow-name
 simShadow1 \
    --cli-binary-format raw-in-base64-out \
    --payload '{"state":{"desired":{"ColorRGB":
[255,255,0]}},"clientToken":"21b21b21-bfd2-4279-8c65-e2f697ff4fab"}' /dev/stdout
```

AWS CLI v1.x

```
aws iot-data update-thing-shadow --thing-name mySimulatedThing --shadow-name
 simShadow1 \
    --payload '{"state":{"desired":{"ColorRGB":
[255,255,0]}},"clientToken":"21b21b21-bfd2-4279-8c65-e2f697ff4fab"}' /dev/stdout
```

Wenn dieser Befehl erfolgreich ist, sollte das folgende Schattendokument zurückgegeben werden.

```
{
  "state": {
    "desired": {
      "ColorRGB": [
```

```
        255,
        255,
        0
      ]
    }
  },
  "metadata": {
    "desired": {
      "ColorRGB": [
        {
          "timestamp": 1591141596
        },
        {
          "timestamp": 1591141596
        },
        {
          "timestamp": 1591141596
        }
      ]
    }
  },
  "version": 4,
  "timestamp": 1591141596,
  "clientToken": "21b21b21-bfd2-4279-8c65-e2f697ff4fab"
}
```

Reaktion auf eine Aktualisierung im Gerät

Wenn Sie in der AWS Konsole zum MQTT-Client zurückkehren, sollten Sie die Nachrichten sehen, die AWS IoT veröffentlicht wurden, um den im vorherigen Abschnitt ausgegebenen Aktualisierungsbefehl widerzuspiegeln.

Um die Aktualisierungsnachrichten in der MQTT-Client

Wählen Sie im MQTT-Client $aws/things//mySimulatedThing/shadowname/simShadow1/update/deltain der Spalte Abonnements aus. Wenn der Themenname abgeschnitten wird, können Sie ihn anhalten, um das vollständige Thema anzuzeigen. Im Themenprotokoll zu diesem Thema sollten Sie eine `/delta` Meldung sehen, die dieser ähnelt.

```
{
  "version": 4,
  "timestamp": 1591141596,
```

```
  "state": {
    "ColorRGB": [
      255,
      255,
      0
    ]
  },
  "metadata": {
    "ColorRGB": [
      {
        "timestamp": 1591141596
      },
      {
        "timestamp": 1591141596
      },
      {
        "timestamp": 1591141596
      }
    ]
  },
  "clientToken": "21b21b21-bfd2-4279-8c65-e2f697ff4fab"
}
```

Ihr Gerät verarbeitet den Inhalt dieser Nachricht, um den Gerätestatus so festzulegen, dass er mit dem `desired`-Status in der Nachricht übereinstimmt.

Nachdem das Gerät den Status so aktualisiert hat, dass er mit dem `desired` Status in der Nachricht übereinstimmt, muss es den neu gemeldeten Status zurücksenden, AWS IoT indem es eine Aktualisierungsnachricht veröffentlicht. Dieses Verfahren simuliert dies im MQTT-Client.

So aktualisieren Sie den Schatten vom Gerät aus:

1. Wählen Sie im MQTT-Client die Option Publish to a topic (In einem Thema veröffentlichen) aus.
2. Geben Sie im Nachrichtentextfenster im Themenfeld über dem Nachrichtentext das Schattenthema ein, gefolgt von der `/update`-Aktion: `$aws/things/mySimulatedThing/shadow/name/simShadow1/update` und geben Sie im Nachrichtentext dieses aktualisierte Shadow-Dokument ein, das den aktuellen Status des Geräts beschreibt. Wählen Sie Veröffentlichen, um den aktualisierten Gerätestatus zu veröffentlichen.

   ```
   {
     "state": {
       "reported": {
   ```

```
      "ColorRGB": [255,255,0]
      }
  },
  "clientToken": "a4dc2227-9213-4c6a-a6a5-053304f60258"
}
```

Wenn die Nachricht erfolgreich von empfangen wurde AWS IoT, sollten Sie im $aws/things//mySimulatedThing/shadowname/simShadow1/update/accepted-Meldungsprotokoll des MQTT-Clients eine neue Antwort mit dem aktuellen Status des Shadows sehen, wie in diesem Beispiel.

```
{
  "state": {
    "reported": {
      "ColorRGB": [
        255,
        255,
        0
      ]
    }
  },
  "metadata": {
    "reported": {
      "ColorRGB": [
        {
          "timestamp": 1591142747
        },
        {
          "timestamp": 1591142747
        },
        {
          "timestamp": 1591142747
        }
      ]
    }
  },
  "version": 5,
  "timestamp": 1591142747,
  "clientToken": "a4dc2227-9213-4c6a-a6a5-053304f60258"
}
```

Ein erfolgreiches Update auf den gemeldeten Zustand des Geräts führt auch AWS IoT dazu, dass eine umfassende Beschreibung des Shadow-Status in einer Nachricht an das `update/documents` Thema gesendet wird, z. B. dieser Nachrichtentext, der aus dem Shadow-Update resultiert, das das Gerät im vorherigen Verfahren durchgeführt hat.

```
{
  "previous": {
    "state": {
      "desired": {
        "ColorRGB": [
          255,
          255,
          0
        ]
      },
      "reported": {
        "ID": "SmartLamp21",
        "ColorRGB": [
          128,
          128,
          128
        ]
      }
    },
    "metadata": {
      "desired": {
        "ColorRGB": [
          {
            "timestamp": 1591141596
          },
          {
            "timestamp": 1591141596
          },
          {
            "timestamp": 1591141596
          }
        ]
      },
      "reported": {
        "ID": {
          "timestamp": 1591140517
        },
        "ColorRGB": [
```

```
          {
            "timestamp": 1591140517
          },
          {
            "timestamp": 1591140517
          },
          {
            "timestamp": 1591140517
          }
        ]
      }
    },
    "version": 4
  },
  "current": {
    "state": {
      "desired": {
        "ColorRGB": [
          255,
          255,
          0
        ]
      },
      "reported": {
        "ID": "SmartLamp21",
        "ColorRGB": [
          255,
          255,
          0
        ]
      }
    },
    "metadata": {
      "desired": {
        "ColorRGB": [
          {
            "timestamp": 1591141596
          },
          {
            "timestamp": 1591141596
          },
          {
            "timestamp": 1591141596
          }
```

```
        ]
      },
      "reported": {
        "ID": {
          "timestamp": 1591140517
        },
        "ColorRGB": [
          {
            "timestamp": 1591142747
          },
          {
            "timestamp": 1591142747
          },
          {
            "timestamp": 1591142747
          }
        ]
      }
    },
    "version": 5
  },
  "timestamp": 1591142747,
  "clientToken": "a4dc2227-9213-4c6a-a6a5-053304f60258"
}
```

Beobachten Sie das Update in der App

Die App kann jetzt den Schatten nach dem aktuellen Status abfragen, wie vom Gerät gemeldet.

Um den aktuellen Status des Schattens abzurufen, verwenden Sie den AWS CLI

1. Geben Sie in der Befehlszeile den folgenden Befehl ein.

   ```
   aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name
    simShadow1 /dev/stdout
   ```

   Auf Windows-Plattformen können Sie `con` anstelle von `/dev/stdout` verwenden.

   ```
   aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name
    simShadow1 con
   ```

2. Da der Schatten gerade vom Gerät aktualisiert wurde, um seinen aktuellen Zustand wiederzugeben, sollte er das folgende Schattendokument zurückgeben.


```
{
  "state": {
    "desired": {
      "ColorRGB": [
        255,
        255,
        0
      ]
    },
    "reported": {
      "ID": "SmartLamp21",
      "ColorRGB": [
        255,
        255,
        0
      ]
    }
  },
  "metadata": {
    "desired": {
      "ColorRGB": [
        {
          "timestamp": 1591141596
        },
        {
          "timestamp": 1591141596
        },
        {
          "timestamp": 1591141596
        }
      ]
    },
    "reported": {
      "ID": {
        "timestamp": 1591140517
      },
      "ColorRGB": [
        {
          "timestamp": 1591142747
        },
```

```
        {
          "timestamp": 1591142747
        },
        {
          "timestamp": 1591142747
        }
      ]
    }
  },
  "version": 5,
  "timestamp": 1591143269
}
```

Über die Simulation hinaus

Experimentieren Sie mit der Interaktion zwischen der AWS CLI (für die App) und der Konsole (für das Gerät), um Ihre IoT-Lösung zu modellieren.

Interaktion mit Schatten

In diesem Thema werden die Nachrichten beschrieben, die jeder der drei Methoden zugeordnet sind, die AWS IoT für das Arbeiten mit Schatten bereitstellt. Zu diesen Methoden gehören die folgenden:

`UPDATE`

Erstellt einen Schatten, wenn er nicht vorhanden ist, oder aktualisiert den Inhalt eines vorhandenen Schattens mit den Statusinformationen, die im Nachrichtentext bereitgestellt werden. AWS IoT zeichnet mit jeder Aktualisierung einen Zeitstempel auf, um anzugeben, wann der Status zuletzt aktualisiert wurde. Wenn sich der Status des Shadows ändert, werden `/delta` Nachrichten mit dem Unterschied zwischen den `reported` Zuständen `desired` und an alle MQTT-Abonnenten AWS IoT gesendet. Geräte oder Apps, die eine `/delta`-Nachricht empfangen, können basierend auf dem Unterschied Aktionen durchführen. Bei einem Gerät kann z. B. der Status auf den Sollstatus oder bei einer Anwendung die Benutzeroberfläche aktualisiert werden, um die Änderung des Gerätestatus zu reflektieren.

`GET`

Ruft ein aktuelles Schattendokument ab, das den vollständigen Status des Schattens einschließlich Metadaten enthält.

`DELETE`

Löscht den Geräteschatten und seinen gesamten Inhalt.

Sie können ein gelöschtes Geräteschatten-Dokument nicht wiederherstellen, aber Sie können ein neues Geräteschatten-Dokument mit dem Namen eines gelöschten Geräteschatten-Dokuments erstellen. Wenn Sie ein Geräteschatten-Dokument erstellen, das denselben Namen hat wie eines, das innerhalb der letzten 48 Stunden gelöscht wurde, folgt die Versionsnummer des neuen Geräteschatten-Dokuments der Versionsnummer des gelöschten. Wenn ein Geräteschatten-Dokument länger als 48 Stunden gelöscht wurde, lautet die Versionsnummer eines neuen Geräteschatten-Dokuments mit demselben Namen 0.

Protokollunterstützung

AWS IoT unterstützt MQTT und eine REST-API über HTTPS-Protokolle zur Interaktion mit Shadows. AWS IoT bietet eine Reihe von reservierten Anfrage- und Antwortthemen für MQTT-Veröffentlichungs- und Abonnementaktionen. Geräte und Apps sollten die Antwortthemen abonnieren, bevor sie sie zu einem Anfragethema veröffentlichen, um Informationen darüber zu erhalten, wie die Anfrage AWS IoT bearbeitet wurde. Weitere Informationen erhalten Sie unter MQTT-Themen für Geräteschatten und Geräteschatten-REST-API.

Anforderungs- und Meldestatus

Wenn Sie Ihre IoT-Lösung mithilfe von AWS IoT und Shadows entwerfen, sollten Sie festlegen, welche Apps oder Geräte Änderungen anfordern und welche diese implementieren. In der Regel implementiert und meldet ein Gerät Änderungen an den Schatten zurück, und Apps und Services reagieren auf Änderungen im Schatten und fordern Änderungen an. Ihre Lösung könnte davon abweichen, aber die Beispiele in diesem Thema gehen jedoch davon aus, dass die Client-App oder der Service Änderungen im Schatten anfordert und das Gerät die Änderungen durchführt und sie an den Schatten zurückmeldet.

Aktualisieren eines Shadows

Ihre App oder Ihr Service kann den Status eines Schattens mithilfe der UpdateThingShadow-API oder durch Veröffentlichung im /update-Thema aktualisieren. Aktualisierungen betreffen lediglich die in der Anfrage angegebenen Felder.

Aktualisieren eines Schattens, wenn ein Client eine Statusänderung anfordert

Wenn ein Client eine Statusänderung in einem Schatten mithilfe des MQTT-Protokolls anfordert

1. Der Client sollte über ein aktuelles Schattendokument verfügen, damit er die zu ändernden Eigenschaften identifizieren kann. Weitere Informationen zum Abrufen des aktuellen Schattendokuments finden Sie unter der Aktion /get.
2. Der Client abonniert diese MQTT-Themen:

   - `$aws/things/thingName/shadow/name/shadowName/update/accepted`
   - `$aws/things/thingName/shadow/name/shadowName/update/rejected`
   - `$aws/things/thingName/shadow/name/shadowName/update/delta`
   - `$aws/things/thingName/shadow/name/shadowName/update/documents`
3. Der Client veröffentlicht ein `$aws/things/thingName/shadow/name/shadowName/update`-Anforderungsthema mit einem Statusdokument, das den gewünschten Status des Schattens enthält. Nur die zu ändernden Eigenschaften müssen in das Dokument aufgenommen werden. Dies ist ein Beispiel für ein Dokument mit dem gewünschten Status.

   ```
   {
     "state": {
       "desired": {
         "color": {
           "r": 10
         },
         "engine": "ON"
       }
     }
   }
   ```
4. Wenn die Aktualisierungsanforderung gültig ist, wird der gewünschte Status im Shadow AWS IoT aktualisiert und Meldungen zu den folgenden Themen veröffentlicht:

   - `$aws/things/thingName/shadow/name/shadowName/update/accepted`
   - `$aws/things/thingName/shadow/name/shadowName/update/delta`

   Die `/update/accepted`-Nachricht enthält ein Antwortstatusdokument „/accepted“-Schattendokument, und die `/update/delta`-Nachricht enthält ein Antwortstatusdokument „/delta“-Schattendokument.

5. Wenn die Aktualisierungsanforderung nicht gültig ist, AWS IoT veröffentlicht eine Nachricht mit dem `$aws/things/thingName/shadow/name/shadowName/update/rejected` Thema zusammen mit einem Fehlerantwortdokument Schattendokument, das den Fehler beschreibt.

Wenn ein Client eine Statusänderung in einem Schatten mithilfe der API anfordert

1. Der Client ruft die `UpdateThingShadow`-API mit einem Anfragestatusdokument-Statusdokument als Nachrichtentext auf.
2. Wenn die Anforderung gültig war, wird ein HTTP-Erfolgsantwortcode und ein Antwortstatusdokument „/accepted“ Shadow-Dokument als Hauptteil der Antwortnachricht AWS IoT zurückgegeben.

   AWS IoT veröffentlicht außerdem eine MQTT-Nachricht zum `$aws/things/thingName/shadow/name/shadowName/update/delta` Thema mit einem Antwortstatusdokument „/delta“ Shadow-Dokument für alle Geräte oder Clients, die es abonnieren.
3. Wenn die Anfrage nicht gültig war, wird ein HTTP-Fehlerantwortcode Fehlerantwortdokument als Antwortnachrichtentext AWS IoT zurückgegeben.

Wenn das Gerät den `/desired`-Status zum `/update/delta`-Thema erhält, nimmt es die gewünschten Änderungen im Gerät vor. Anschließend wird eine Nachricht an das `/update`-Thema gesendet, um den aktuellen Status an den Schatten zu melden.

Aktualisieren eines Schattens, wenn ein Gerät seinen aktuellen Status meldet

Wenn ein Gerät seinen aktuellen Status an den Schatten mithilfe des MQTT-Protokolls meldet

1. Das Gerät sollte diese MQTT-Themen abonnieren, bevor Sie den Schatten aktualisieren:

   - `$aws/things/thingName/shadow/name/shadowName/update/accepted`
   - `$aws/things/thingName/shadow/name/shadowName/update/rejected`
   - `$aws/things/thingName/shadow/name/shadowName/update/delta`
   - `$aws/things/thingName/shadow/name/shadowName/update/documents`
2. Das Gerät meldet seinen aktuellen Status, indem es eine Nachricht zum `$aws/things/thingName/shadow/name/shadowName/update`-Thema veröffentlicht, das den aktuellen Status meldet, wie etwa in diesem Beispiel.

   ```
   {
   ```

```
    "state": {
        "reported" : {
            "color" : { "r" : 10 },
            "engine" : "ON"
        }
    }
}
```

3. Wenn das Update AWS IoT akzeptiert wird, veröffentlicht es eine Nachricht zu den `$aws/things/thingName/shadow/name/shadowName/update/accepted` Themen mit einem Antwortstatusdokument „/accepted“ Schattendokument.
4. Wenn die Aktualisierungsanforderung nicht gültig ist, wird eine Nachricht mit dem `$aws/things/thingName/shadow/name/shadowName/update/rejected` Thema zusammen mit einem Fehlerantwortdokument Schattendokument AWS IoT veröffentlicht, das den Fehler beschreibt.

Wenn ein Gerät den aktuellen Status mithilfe der API an den Schatten meldet

1. Das Gerät ruft die `UpdateThingShadow`-API mit einem Anfragestatusdokument-Statusdokument als Nachrichtentext auf.
2. Wenn die Anforderung gültig war, wird der Shadow AWS IoT aktualisiert und ein HTTP-Erfolgsantwortcode mit einem Antwortstatusdokument „/accepted“ Shadow-Dokument als Nachrichtentext zurückgegeben.

   AWS IoT veröffentlicht außerdem eine MQTT-Nachricht zum `$aws/things/thingName/shadow/name/shadowName/update/delta` Thema mit einem Antwortstatusdokument „/delta“ Shadow-Dokument für alle Geräte oder Clients, die es abonnieren.
3. Wenn die Anfrage nicht gültig war, wird ein HTTP-Fehlerantwortcode Fehlerantwortdokument als Antwortnachrichtentext AWS IoT zurückgegeben.

Optimistische Sperre

Sie können die Version des Statusdokuments verwenden, um sicherzustellen, dass Sie die neueste Version eines Geräteschattendokuments aktualisieren. Geben Sie bei einer Aktualisierungsanfrage eine Version an, lehnt der Service die Anfrage mit einem Konflikt-Antwortcode HTTP 409 ab, wenn die aktuelle Version des Statusdokuments nicht der angegebenen Version entspricht. Der Konfliktreaktionscode kann auch in jeder API vorkommen, die Änderungen am `ThingShadow` vornimmt, darunter `DeleteThingShadow`:

Beispiel:

Ausgangsdokument:

```
{
  "state": {
    "desired": {
      "colors": [
        "RED",
        "GREEN",
        "BLUE"
      ]
    }
  },
  "version": 10
}
```

Aktualisierung: (die Versionen stimmen nicht überein; die Anfrage wird abgelehnt)

```
{
  "state": {
    "desired": {
      "colors": [
        "BLUE"
      ]
    }
  },
  "version": 9
}
```

Ergebnis:

```
{
  "code": 409,
  "message": "Version conflict",
  "clientToken": "426bfd96-e720-46d3-95cd-014e3ef12bb6"
}
```

Aktualisierung: (die Versionen stimmen überein; die Anfrage wird angenommen)

```
{
```

```
  "state": {
    "desired": {
      "colors": [
        "BLUE"
      ]
    }
  },
  "version": 10
}
```

Endzustand:

```
{
  "state": {
    "desired": {
      "colors": [
        "BLUE"
      ]
    }
  },
  "version": 11
}
```

Abrufen eines Shadow-Dokuments

Sie können ein Schattendokument abrufen, indem Sie die GetThingShadow-API verwenden oder indem Sie das /get-Thema abonnieren und dazu veröffentlichen. Hierdurch wird ein vollständiges Schattendokument einschließlich aller Unterschiede zwischen den Statusarten `desired` und `reported` abgerufen. Die Vorgehensweise für diese Aufgabe ist unabhängig davon, ob das Gerät oder ein Client die Anforderung durchführt.

So rufen Sie ein Schattendokument mithilfe des MQTT-Protokolls ab:

1. Das Gerät oder der Client sollte diese MQTT-Themen abonnieren, bevor der Schatten aktualisiert wird:
   - `$aws/things/thingName/shadow/name/shadowName/get/accepted`
   - `$aws/things/thingName/shadow/name/shadowName/get/rejected`
2. Das Gerät oder der Client veröffentlicht eine Nachricht mit einem leeren Nachrichtentext zum `$aws/things/thingName/shadow/name/shadowName/get`-Thema.

3. Wenn die Anfrage erfolgreich ist, AWS IoT veröffentlicht eine Nachricht zum `$aws/things/thingName/shadow/name/shadowName/get/accepted` Thema mit einem Antwortstatusdokument „/accepted“ im Nachrichtentext.
4. Wenn die Anfrage nicht gültig war, AWS IoT veröffentlicht eine Nachricht zum `$aws/things/thingName/shadow/name/shadowName/get/rejected` Thema mit einem Fehlerantwortdokument im Nachrichtentext.

So rufen Sie ein Schattendokument mithilfe einer REST-API ab:

1. Das Gerät oder Client ruft die `GetThingShadow`-API mit einem leeren Nachrichtentext auf.
2. Wenn die Anfrage gültig ist, wird ein HTTP-Erfolgsantwortcode mit einem Antwortstatusdokument „/accepted“ Shadow-Dokument als Nachrichtentext der Antwort AWS IoT zurückgegeben.
3. Wenn die Anforderung nicht gültig ist, wird ein HTTP-Fehlerantwortcode an Fehlerantwortdokument als Antwortnachrichtentext AWS IoT zurückgegeben.

Löschen von Schattendaten

Es gibt zwei Möglichkeiten, Schattendaten zu löschen: Sie können bestimmte Eigenschaften im Schattendokument löschen oder den Schatten vollständig löschen.

- Um bestimmte Eigenschaften aus einem Schatten zu löschen, aktualisieren Sie den Schatten. Legen Sie jedoch den Wert der Eigenschaften, die Sie löschen möchten, auf `null` fest. Felder mit dem Wert `null` werden aus dem Schattendokument entfernt.
- Um den gesamten Schatten zu löschen, verwenden Sie die DeleteThingShadow-API oder veröffentlichen Sie zum /delete-Thema.

> **Note**
>
> Durch das Löschen eines Shadows wird seine Versionsnummer nicht sofort auf Null zurückgesetzt. Er wird nach 48 Stunden auf null zurückgesetzt.

Löschen einer Eigenschaft aus einem Schattendokument

So löschen Sie eine Eigenschaft aus einem Schatten mithilfe des MQTT-Protokolls:

1. Das Gerät oder der Client sollte über ein aktuelles Schattendokument verfügen, damit es/er die zu ändernden Eigenschaften identifizieren kann. Weitere Informationen zum Abrufen des aktuellen Schattendokuments finden Sie unter Abrufen eines Shadow-Dokuments.
2. Das Gerät oder der Client abonniert diese MQTT-Themen:
   - `$aws/things/thingName/shadow/name/shadowName/update/accepted`
   - `$aws/things/thingName/shadow/name/shadowName/update/rejected`
3. Das Gerät oder der Client veröffentlicht ein `$aws/things/thingName/shadow/name/shadowName/update`-Anforderungsthema mit einem Statusdokument, das den Eigenschaften des zu löschenden Schattens `null`-Werte zuweist. Nur die zu ändernden Eigenschaften müssen in das Dokument aufgenommen werden. Dies ist ein Beispiel für ein Dokument, das die `engine`-Eigenschaft löscht.

   ```
   {
     "state": {
       "desired": {
         "engine": null
       }
     }
   }
   ```
4. Wenn die Aktualisierungsanforderung gültig ist, AWS IoT werden die angegebenen Eigenschaften im Shadow gelöscht und eine Nachricht mit dem `$aws/things/thingName/shadow/name/shadowName/update/accepted` Thema mit einem Antwortstatusdokument „/accepted“ Shadow-Dokument im Nachrichtentext veröffentlicht.
5. Wenn die Aktualisierungsanforderung nicht gültig ist, AWS IoT veröffentlicht eine Nachricht mit dem `$aws/things/thingName/shadow/name/shadowName/update/rejected` Thema zusammen mit einem Fehlerantwortdokument Schattendokument, das den Fehler beschreibt.

So löschen Sie eine Eigenschaft aus einem Schatten mithilfe der REST-API:

1. Das Gerät oder der Client ruft die `UpdateThingShadow`-API mit einer Anfragestatusdokument auf die den Eigenschaften des zu löschenden Schattens `null`-Werte zuweist. Fügen Sie nur

die Eigenschaften, die Sie löschen möchten, in das Dokument ein. Dies ist ein Beispiel für ein Dokument, das die `engine`-Eigenschaft löscht.

```
{
  "state": {
    "desired": {
      "engine": null
    }
  }
}
```

2. Wenn die Anforderung gültig war, wird ein HTTP-Erfolgsantwortcode und ein Antwortstatusdokument „/accepted“ Shadow-Dokument als Hauptteil der Antwortnachricht AWS IoT zurückgegeben.
3. Wenn die Anforderung nicht gültig war, wird ein HTTP-Fehlerantwortcode an Fehlerantwortdokument als Antwortnachrichtentext AWS IoT zurückgegeben.

Löschen eines Shadows

Im Folgenden werden einige Aspekte beschrieben, die beim Löschen des Schattens eines Geräts berücksichtigt werden sollten.

- Wenn Sie den Schattenstatus des Geräts auf `null` festlegen, wird der Schatten nicht gelöscht. Die Schattenversion wird beim nächsten Update erhöht.
- Das Löschen eines Geräteschattens löscht das Objekt nicht. Das Löschen eines Objekts löscht den entsprechenden Geräteschatten nicht.
- Durch das Löschen eines Shadows wird seine Versionsnummer nicht sofort auf Null zurückgesetzt. Er wird nach 48 Stunden auf null zurückgesetzt.

So löschen Sie einen Schatten mithilfe des MQTT-Protokolls:

1. Das Gerät oder der Client abonniert diese MQTT-Themen:
   - `$aws/things/thingName/shadow/name/shadowName/delete/accepted`
   - `$aws/things/thingName/shadow/name/shadowName/delete/rejected`
2. Das Gerät oder der Client veröffentlicht eine `$aws/things/thingName/shadow/name/shadowName/delete`-mit einem leeren Nachrichtenpuffer.

3. Wenn die Löschanforderung gültig ist, wird der Shadow AWS IoT gelöscht und eine Nachricht mit dem `$aws/things/thingName/shadow/name/shadowName/delete/accepted` Thema und einem abgekürzten Antwortstatusdokument „/accepted“ Shadow-Dokument im Nachrichtentext veröffentlicht. Dies ist ein Beispiel für eine akzeptierte Löschmeldung:

```
{
  "version": 4,
  "timestamp": 1591057529
}
```

4. Wenn die Aktualisierungsanforderung nicht gültig ist, AWS IoT veröffentlicht eine Nachricht mit dem `$aws/things/thingName/shadow/name/shadowName/delete/rejected` Thema zusammen mit einem Fehlerantwortdokument Shadow-Dokument, das den Fehler beschreibt.

So löschen Sie einen Schatten mithilfe der REST-API:

1. Das Gerät oder der Client ruft die `DeleteThingShadow`-API mit einem leeren Nachrichtenpuffer auf.
2. Wenn die Anfrage gültig war, wird ein HTTP-Erfolgsantwortcode Antwortstatusdokument „/accepted“ und ein abgekürztes Antwortstatusdokument „/accepted“ Shadow-Dokument im Nachrichtentext AWS IoT zurückgegeben. Dies ist ein Beispiel für eine akzeptierte Löschmeldung:

```
{
  "version": 4,
  "timestamp": 1591057529
}
```

3. Wenn die Anfrage nicht gültig war, wird ein HTTP-Fehlerantwortcode an Fehlerantwortdokument als Antwortnachrichtentext AWS IoT zurückgegeben.

Geräteschatten-REST-API

Ein Schatten stellt den folgenden URI für die Aktualisierung der Statusinformationen bereit:

```
https://account-specific-prefix-ats.iot.region.amazonaws.com/things/thingName/shadow
```

Der Endpunkt ist spezifisch für Ihre AWS-Konto. Um Ihren Endpunkt zu finden, können Sie:

- den Befehl describe-endpoint aus dem AWS CLI verwenden.
- Verwenden Sie die AWS IoT Konsoleneinstellungen. In den Einstellungen ist der Endpunkt unter Benutzerdefinierter Endpunkt aufgeführt
- Benutze die Seite mit den Details der AWS IoT Konsole. In der Konsole:
  1. Öffnen Sie Verwalten und wählen Sie unter Verwalten die Option Objekte aus.
  2. Wählen Sie in der Liste der Objekte das Objekt aus, für das Sie den Endpunkt-URI abrufen möchten.
  3. Wählen Sie die Registerkarte Geräteschatten und wählen Sie Ihren Schatten aus. Sie können den Endpunkt-URI im Abschnitt Geräteschatten-URL der Geräteschatten-Detailseite einsehen.

Der Endpunkt hat folgendes Format:

```
identifier.iot.region.amazonaws.com
```

Die Shadow-REST-API folgt denselben protocols/port HTTPS-Zuordnungen wie unter beschrieben. Gerätekommunikationsprotokolle

> **Note**
>
> Um die APIs verwenden zu können, müssen Sie `iotdevicegateway` als Dienstnamen den Namen der Authentifizierung verwenden. Weitere Informationen finden Sie unter IoTDataPlane.

API-Aktionen

- GetThingShadow
- UpdateThingShadow
- DeleteThingShadow
- ListNamedShadowsForThing

Sie können die API auch verwenden, um einen benannten Schatten zu erstellen, indem Sie `name=shadowName` als Teil des Abfrageparameters der API angeben.

GetThingShadow

Ruft das Schattengerät für das angegebene Objekt ab.

Das Antwort-Statusdokument enthält das Delta zwischen dem Status `desired` (Soll) und dem Status `reported` (gemeldet).

Anforderung

Die Anfrage enthält die Standard-HTTP-Kopfzeilen sowie das folgende URI:

```
HTTP GET https://endpoint/things/thingName/shadow?name=shadowName
Request body: (none)
```

Der `name`-Abfrageparameter ist für unbenannte (klassische) Schatten nicht erforderlich.

Antwort

Bei erfolgreicher Anfrage enthält die Antwort die Standard-HTTP-Kopfzeilen sowie den folgenden Code und Text:

```
HTTP 200
Response Body: response state document
```

Weitere Informationen finden Sie im Antwort-Statusdokumentenbeispiel.

Autorisierung

Für das Abrufen eine Schattens ist eine Richtlinie erforderlich, die es dem Aufrufer erlaubt, die Aktion `iot:GetThingShadow` durchzuführen. Der Geräteschatten-Service akzeptiert zwei Arten der Authentifizierung: die Signatur-Version 4 mit IAM-Anmeldeinformationen oder die gegenseitige TLS-Authentifizierung mit einem Client-Zertifikat.

Es folgt ein Beispiel für eine Richtlinie, die es dem Aufrufer erlaubt, einen Geräteschatten abzurufen:

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

```
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/thing"
            ]
        }
    ]
}
```

UpdateThingShadow

Aktualisiert das Schattengerät für das angegebene Objekt.

Aktualisierungen betreffen lediglich die im Anfragestatusdokument angegebenen Felder. Ein Feld mit dem Wert `null` (Null) wird aus dem Geräteschatten entfernt.

Anforderung

Die Anfrage enthält die Standard-HTTP-Kopfzeilen sowie das/den folgende(n) URI und Text:

```
HTTP POST https://endpoint/things/thingName/shadow?name=shadowName
Request body: request state document
```

Der `name`-Abfrageparameter ist für unbenannte (klassische) Schatten nicht erforderlich.

Weitere Informationen finden Sie im Anfragestatusdokumentenbeispiel.

Antwort

Bei erfolgreicher Anfrage enthält die Antwort die Standard-HTTP-Kopfzeilen sowie den folgenden Code und Text:

```
HTTP 200
Response body: response state document
```

Weitere Informationen finden Sie im Antwort-Statusdokumentenbeispiel.

Autorisierung

Für das Aktualisieren eines Schattens ist eine Richtlinie erforderlich, die es dem Aufrufer erlaubt, die Aktion `iot:UpdateThingShadow` durchzuführen. Der Geräteschatten-Service akzeptiert zwei Arten der Authentifizierung: die Signatur-Version 4 mit IAM-Anmeldeinformationen oder die gegenseitige TLS-Authentifizierung mit einem Client-Zertifikat.

Es folgt ein Beispiel für eine Richtlinie, die es dem Aufrufer erlaubt, einen Geräteschatten zu aktualisieren:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:UpdateThingShadow",
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/thing"
            ]
        }
    ]
}
```

DeleteThingShadow

Löscht das Schattengerät für das angegebene Objekt.

Anforderung

Die Anfrage enthält die Standard-HTTP-Kopfzeilen sowie das folgende URI:

```
HTTP DELETE https://endpoint/things/thingName/shadow?name=shadowName
Request body: (none)
```

Der `name`-Abfrageparameter ist für unbenannte (klassische) Schatten nicht erforderlich.

Antwort

Bei erfolgreicher Anfrage enthält die Antwort die Standard-HTTP-Kopfzeilen sowie den folgenden Code und Text:

```
HTTP 200
Response body: Empty response state document
```

Beachten Sie, dass durch das Löschen eines Shadows seine Versionsnummer nicht auf 0 zurückgesetzt wird.

Autorisierung

Für das Löschen eine Geräteschattens ist eine Richtlinie erforderlich, die es dem Aufrufer erlaubt, die Aktion `iot:DeleteThingShadow` durchzuführen. Der Geräteschatten-Service akzeptiert zwei Arten der Authentifizierung: die Signatur-Version 4 mit IAM-Anmeldeinformationen oder die gegenseitige TLS-Authentifizierung mit einem Client-Zertifikat.

Es folgt ein Beispiel für eine Richtlinie, die es dem Aufrufer erlaubt, einen Geräteschatten zu löschen:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:DeleteThingShadow",
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/thing"
            ]
        }
    ]
}
```

ListNamedShadowsForThing

Listet die Schatten für das angegebene Objekt auf.

Anforderung

Die Anfrage enthält die Standard-HTTP-Kopfzeilen sowie das folgende URI:

```
HTTP GET /api/things/shadow/ListNamedShadowsForThing/thingName?
nextToken=nextToken&pageSize=pageSize
Request body: (none)
```

nextToken

Das Token zum Abruf des nächsten Ergebnissatzes.

Dieser Wert wird für nach Seiten organisierte Ergebnisse zurückgegeben und in dem Aufruf verwendet, der die nächste Seite zurückgibt.

pageSize

Die Anzahl der Schattennamen, die bei jedem Aufruf zurückgegeben werden sollen. Siehe auch `nextToken`.

thingName

Der Name des Objekts, für das die benannten Schatten aufgelistet werden sollen.

Antwort

Falls erfolgreich, enthält die Antwort die Standard-HTTP-Kopfzeilen sowie den folgenden Antwortcode und eine Antwortdokument für die Schattennamenliste.

> **Note**
>
> Der unbenannte (klassische) Schatten wird in dieser Liste nicht angezeigt. Die Antwort ist eine leere Liste, wenn Sie nur einen klassischen Schatten haben oder wenn der von `thingName` Ihnen angegebene Schatten nicht existiert.

```
HTTP 200
Response body: Shadow name list document
```

Autorisierung

Für das Löschen eines Geräteschattens ist eine Richtlinie erforderlich, die es dem Aufrufer erlaubt, die Aktion `iot:ListNamedShadowsForThing` durchzuführen. Der Geräteschatten-Service akzeptiert zwei Arten der Authentifizierung: die Signatur-Version 4 mit IAM-Anmeldeinformationen oder die gegenseitige TLS-Authentifizierung mit einem Client-Zertifikat.

Es folgt ein Beispiel für eine Richtlinie, die es dem Aufrufer erlaubt, die benannten Schatten eines Objekts zu aktualisieren:

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

```
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/thing"
            ]
        }
    ]
}
```

MQTT-Themen für Geräteschatten

Der Device Shadow-Service verwendet reservierte MQTT-Themen, um es Anwendungen und Geräten zu ermöglichen, die Statusinformationen für ein Gerät (Schatten) abzurufen, zu aktualisieren oder zu löschen.

Das Veröffentlichen in und Abonnieren von Schattengerätethemen erfordert eine themenbasierte Autorisierung. AWS IoT behält sich das Recht vor, der vorhandenen Themenstruktur neue Themen hinzuzufügen. Aus diesem Grund empfehlen wir, Abonnements mit Platzhaltern von Schattengeräte-Topics zu vermeiden. Vermeiden Sie es beispielsweise, Themenfilter zu abonnieren, `$aws/things/thingName/shadow/#` da die Anzahl der Themen, die diesem Themenfilter entsprechen, mit der AWS IoT Einführung neuer Shadow-Themen zunehmen könnte. Beispiele für Nachrichten, die zu diesen Topics veröffentlichten wurden, finden Sie unter Interaktion mit Schatten.

Schatten können benannt oder unbenannt sein (klassisch). Die jeweils verwendeten Themen unterscheiden sich nur durch das Themenpräfix. In dieser Tabelle wird das Themenpräfix angezeigt, das von jedem Schattentyp verwendet wird.

ShadowTopicPrefix Wert	Schattentyp
`$aws/things/` *thingName* `/shadow`	Unbenannter (klassischer) Schatten
`$aws/things/` *thingName* `/shadow/name/` *shadowName*	Benannter Schatten

Um ein vollständiges Thema zu erstellen, wählen Sie die *ShadowTopicPrefix* für den Schattentyp aus, auf den Sie verweisen möchten, ersetzen Sie *thingName* und gegebenenfalls *shadowName* durch die entsprechenden Werte und fügen Sie diese dann an den Themen-Stub an, wie in den folgenden Abschnitten dargestellt.

Nachstehend finden Sie die MQTT-Themen, die für die Interaktion mit Schatten verwendet wurden.

Topics

- /get
- /get/accepted
- /get/rejected
- /update
- /update/delta
- /update/accepted
- /update/documents
- /update/rejected
- /delete
- /delete/accepted
- /delete/rejected

/get

Veröffentlichen Sie eine leere Nachricht in diesem Thema, um den Geräteschatten abzurufen:

```
ShadowTopicPrefix/get
```

AWS IoT reagiert mit der Veröffentlichung auf entweder /get/accepted oder/get/rejected.

Beispielrichtline

Im Folgenden finden Sie ein Beispiel für die erforderliche Richtlinie:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/get"
```

```
            ]
        }
    ]
}
```

/get/accepted

AWS IoT veröffentlicht ein Antwort-Schattendokument zu diesem Thema, wenn der Shadow des Geräts zurückgegeben wird:

```
ShadowTopicPrefix/get/accepted
```

Weitere Informationen finden Sie unter Antwortstatusdokumente.

Beispielrichtline

Im Folgenden finden Sie ein Beispiel für die erforderliche Richtlinie:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/get/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/get/accepted"
            ]
```

```
        }
    ]
}
```

/get/rejected

AWS IoT veröffentlicht ein Fehlerantwortdokument zu diesem Thema, wenn der Shadow des Geräts nicht zurückgegeben werden kann:

```
ShadowTopicPrefix/get/rejected
```

Weitere Informationen finden Sie unter Fehlerantwortdokument.

Beispielrichtline

Im Folgenden finden Sie ein Beispiel für die erforderliche Richtlinie:

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/thingName/
shadow/get/rejected"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/shadow/get/
rejected"
      ]
    }
  ]
```

```
}
```

/update

Veröffentlichen Sie in diesem Thema ein Anfragestatusdokument, um den Geräteschatten zu aktualisieren:

```
ShadowTopicPrefix/update
```

Der Nachrichtentext enthält ein partielles Anfragestatusdokument.

Ein Client, der versucht, den Status eines Geräts zu aktualisieren, sendet ein JSON-Anfragestatusdokument mit einer `desired`-Eigenschaft wie der folgenden:

```
{
  "state": {
    "desired": {
      "color": "red",
      "power": "on"
    }
  }
}
```

Ein Gerät, das seinen Schatten aktualisiert, würde ein JSON-Anfragestatusdokument mit der `reported`-Eigenschaft senden, z. B.:

```
{
  "state": {
    "reported": {
      "color": "red",
      "power": "on"
    }
  }
}
```

AWS IoT reagiert mit der Veröffentlichung auf entweder /update/accepted oder/update/rejected.

Beispielrichtline

Im Folgenden finden Sie ein Beispiel für die erforderliche Richtlinie:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/update"
            ]
        }
    ]
}
```

/update/delta

AWS IoT veröffentlicht ein Antwortstatusdokument zu diesem Thema, wenn es eine Änderung für den Schatten des Geräts akzeptiert, und das Antwortstatusdokument enthält verschiedene Werte für `desired` und `reported` Status:

```
ShadowTopicPrefix/update/delta
```

Der Nachrichtenpuffer enthält eine Antwortstatusdokument „/delta“.

Nachrichtentextdetails

- Eine in `update/delta` veröffentlichte Nachricht umfasst nur die gewünschten Attribute, die sich zwischen dem Abschnitt `desired` (Soll) und dem Abschnitt `reported` (gemeldet) Abschnitt unterscheiden. Sie enthält alle diese Attribute, unabhängig davon, ob diese in der Nachricht zur aktuellen Aktualisierung enthalten waren oder bereits in AWS IoT gespeichert wurden. Attribute, die sich nicht zwischen dem Abschnitt `desired` (Soll) und dem Abschnitt `reported` (gemeldet) Abschnitt unterscheiden, sind nicht enthalten.
- Wenn sich ein Attribut im Abschnitt `reported` (gemeldet) befindet, jedoch kein Pendant im Abschnitt `desired` (Soll), dann ist es nicht enthalten.
- Ist ein Attribut im Abschnitt `desired` (Soll) vorhanden, besitzt jedoch kein Pendant im Abschnitt `reported` (gemeldet), dann ist es enthalten.

- Wenn ein Attribut aus dem Abschnitt `reported` (gemeldet) gelöscht wurde, aber sich nach wie vor im Abschnitt `desired` (Soll) befindet, dann ist es enthalten.

Beispielrichtline

Im Folgenden finden Sie ein Beispiel für die erforderliche Richtlinie:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/update/delta"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/update/delta"
            ]
        }
    ]
}
```

/update/accepted

AWS IoT veröffentlicht ein Antwortstatusdokument zu diesem Thema, wenn es eine Änderung für den Schatten des Geräts akzeptiert:

```
ShadowTopicPrefix/update/accepted
```

Der Nachrichtenpuffer enthält eine Antwortstatusdokument „/accepted“.

Beispielrichtline

Im Folgenden finden Sie ein Beispiel für die erforderliche Richtlinie:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/update/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/update/accepted"
            ]
        }
    ]
}
```

/update/documents

AWS IoT veröffentlicht ein Statusdokument zu diesem Thema, wenn eine Aktualisierung des Shadows erfolgreich durchgeführt wurde:

```
ShadowTopicPrefix/update/documents
```

Der Nachrichtentext enthält eine /Dokumente, Antwortstatusdokument.

Beispielrichtline

Im Folgenden finden Sie ein Beispiel für die erforderliche Richtlinie:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/update/documents"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/update/documents"
            ]
        }
    ]
}
```

/update/rejected

AWS IoT veröffentlicht ein Fehlerantwortdokument zu diesem Thema, wenn eine Änderung für den Shadow des Geräts abgelehnt wird:

```
ShadowTopicPrefix/update/rejected
```

Der Nachrichtentext enthält eine Fehlerantwortdokument.

Beispielrichtline

Im Folgenden finden Sie ein Beispiel für die erforderliche Richtlinie:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/update/rejected"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/update/rejected"
            ]
        }
    ]
}
```

/delete

Um einen Geräteschatten zu löschen, veröffentlichen Sie im Löschthema eine leere Nachricht.

```
ShadowTopicPrefix/delete
```

Der Inhalt der Nachricht wird ignoriert.

Beachten Sie, dass durch das Löschen eines Shadows seine Versionsnummer nicht auf 0 zurückgesetzt wird.

AWS IoT reagiert mit der Veröffentlichung auf entweder /delete/accepted oder/delete/rejected.

Beispielrichtline

Im Folgenden finden Sie ein Beispiel für die erforderliche Richtlinie:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/delete"
            ]
        }
    ]
}
```

/delete/accepted

AWS IoT veröffentlicht eine Nachricht zu diesem Thema, wenn der Schatten eines Geräts gelöscht wird:

```
ShadowTopicPrefix/delete/accepted
```

Beispielrichtline

Im Folgenden finden Sie ein Beispiel für die erforderliche Richtlinie:

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

```
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/delete/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/delete/accepted"
            ]
        }
    ]
}
```

/delete/rejected

AWS IoT veröffentlicht ein Dokument mit einer Fehlermeldung zu diesem Thema, wenn der Shadow des Geräts nicht gelöscht werden kann:

```
ShadowTopicPrefix/delete/rejected
```

Der Nachrichtentext enthält eine Fehlerantwortdokument.

Beispielrichtline

Im Folgenden finden Sie ein Beispiel für die erforderliche Richtlinie:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
```

```
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
things/thingName/shadow/delete/rejected"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thingName/
shadow/delete/rejected"
            ]
        }
    ]
}
```

Dokumente des Device Shadow-Services

Der Device Shadow-Service befolgt alle Regeln der JSON-Spezifikation. Werte, Objekte und Arrays werden im Geräteschatten-Dokument gespeichert.

Inhalt

- Beispiele für Schatten-Dokumente
- Dokumenteigenschaften
- Delta-Status
- Versioning von Schattendokumenten
- Client-Tokens in Schattendokumenten
- Eigenschaften des leeren Schattendokuments
- Array-Werte in Schattendokumenten

Beispiele für Schatten-Dokumente

Der Device Shadow-Dienst verwendet diese Dokumente in UPDATE-, GET- und DELETE-Vorgängen mithilfe der REST-API oder Pub/Sub MQTT-Nachrichten.

Beispiele

- Anfragestatusdokument
- Antwortstatusdokumente
- Fehlerantwortdokument
- Antwortdokument für die Schattennamenliste

Anfragestatusdokument

Ein Anfragestatusdokument hat das folgende Format:

```
{
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        },
        "reported": {
            "attribute1": integer1,
            "attribute2": "string1",
            ...
            "attributeN": boolean1
        }
    },
    "clientToken": "token",
    "version": version
}
```

- `state` – Aktualisierungen betreffen lediglich die angegebenen Felder. In der Regel verwenden Sie entweder die `desired`- oder die `reported`-Eigenschaft, aber nicht beide, in derselben Anforderung.
  - `desired` – Die Statuseigenschaften und Werte, deren Aktualisierung im Gerät angefordert wurde.
  - `reported` – Die vom Gerät gemeldeten Zustandseigenschaften und -werte.
- `clientToken` – Falls verwendet, können Sie die Anfrage und die entsprechende Antwort anhand des Client-Tokens abgleichen.
- `version` — Bei Verwendung verarbeitet der Device Shadow-Service nur dann die Aktualisierung, wenn die angegebene Version mit seiner neuesten Version übereinstimmt.

Antwortstatusdokumente

Antwortstatusdokumente haben je nach Antworttyp das folgende Format.

Antwortstatusdokument „/accepted“

```
{
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        }
    },
    "metadata": {
        "desired": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        }
    },
    "timestamp": timestamp,
    "clientToken": "token",
    "version": version
}
```

Antwortstatusdokument „/delta“

```
{
    "state": {
        "attribute1": integer2,
        "attribute2": "string2",
        ...
        "attributeN": boolean2
    },
```

```
    "metadata": {
        "attribute1": {
            "timestamp": timestamp
        },
        "attribute2": {
            "timestamp": timestamp
        },
        ...
        "attributeN": {
            "timestamp": timestamp
        }
    },
    "timestamp": timestamp,
    "clientToken": "token",
    "version": version
}
```

/Dokumente, Antwortstatusdokument

```
{
  "previous" : {
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        },
        "reported": {
            "attribute1": integer1,
            "attribute2": "string1",
            ...
            "attributeN": boolean1
        }
    },
    "metadata": {
        "desired": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
```

```
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        },
        "reported": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        }
    },
    "version": version-1
},
"current": {
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        },
        "reported": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        }
    },
    "metadata": {
        "desired": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
```

```
            "attributeN": {
                "timestamp": timestamp
            }
        },
        "reported": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        }
    },
    "version": version
  },
  "timestamp": timestamp,
  "clientToken": "token"
}
```

Eigenschaften des Antwortstatusdokuments

- `previous` – Enthält nach einer erfolgreichen Aktualisierung den `state` des Objekts vor dem Update.
- `current` – Enthält nach einer erfolgreichen Aktualisierung den `state` des Objekts nach dem Update.
- `state`
  - `reported` – Liegt nur dann vor, wenn ein Objekt Daten im `reported`-Abschnitt gemeldet hat, und enthält nur Felder, die im Anfragestatusdokument enthalten waren.
  - `desired` – Liegt nur dann vor, wenn ein Gerät Daten im `desired`-Abschnitt gemeldet hat, und enthält nur Felder, die im Anfragestatusdokument enthalten waren.
- `metadata` –`desired` Enthält für jedes Attribut in den Abschnitten und `reported` die Zeitstempel, sodass Sie feststellen können, wann der Status aktualisiert wurde.
- `timestamp`— Datum und Uhrzeit der Generierung der Antwort in Epoch. AWS IoT
- `clientToken` – Liegt nur dann vor, wenn bei der Veröffentlichung einer gültigen JSON im Thema `/update` ein Client-Token verwendet wurde.

- `version` – Die aktuelle Version des Dokuments für den Geräteschatten, freigegeben in AWS IoT. Sie erhöht sich zur vorherigen Versionsnummer des Dokuments um die Zahl eins.

Fehlerantwortdokument

Ein Fehlerantwortdokument hat das folgende Format:

```
{
    "code": error-code,
    "message": "error-message",
    "timestamp": timestamp,
    "clientToken": "token"
}
```

- `code` – Einen HTTP-Antwortcode, der auf die Art des Fehlers hinweist.
- `message` – Eine Textnachricht mit zusätzlichen Informationen.
- `timestamp`— Datum und Uhrzeit der Generierung der Antwort. AWS IoT Diese Eigenschaft ist nicht in allen Fehlerantwortdokumenten vorhanden.
- `clientToken` – Liegt nur dann vor, wenn ein Client-Token in der veröffentlichten Nachricht verwendet wurde.

Weitere Informationen finden Sie unter Device Shadow-Fehlermeldungen.

Antwortdokument für die Schattennamenliste

Ein Antwortdokument für die Schattennamenliste hat das folgende Format:

```
{
    "results": [
        "shadowName-1",
        "shadowName-2",
        "shadowName-3",
        "shadowName-n"
    ],
    "nextToken": "nextToken",
    "timestamp": timestamp
}
```

- `results` – Das Array von Schattennamen.

- `nextToken` – Der Token-Wert, der in nach Seiten organisierten Anforderungen verwendet wird, um die nächste Seite in der Sequenz abzurufen. Diese Eigenschaft ist nicht vorhanden, wenn keine weiteren Schattennamen zurückgegeben werden sollen.
- `timestamp`— Datum und Uhrzeit der Generierung der Antwort AWS IoT.

Dokumenteigenschaften

Ein Geräteschatten-Dokument besitzt die folgenden Eigenschaften:

`state`

`desired`

Den gewünschte Status des Geräts. Anwendungen können diesen Teil des Dokuments beschreiben, um den Status eines Geräts zu aktualisieren, ohne direkt mit diesem verbunden zu sein.

`reported`

Der gemeldete Status des Geräts. Geräte schreiben in diesen Teil des Dokuments, um ihren neuen Status zu melden. Apps lesen diesen Teil des Dokuments, um den zuletzt gemeldeten Zustand des Geräts zu bestimmen.

`metadata`

Informationen über die im Abschnitt `state` (Status) des Dokuments gespeicherten Daten. Dazu zählen Zeitstempel, in Epoche-Uhrzeit, für das jeweilige Attribut im Abschnitt `state` (Status), anhand derer Sie den Zeitpunkt ermitteln können, zu dem sie aktualisiert wurden.

> **Note**
>
> Metadaten zählen nicht zur Dokumentengröße für Service Limits oder Preise. Weitere Informationen finden Sie unter Service Limits AWS IoT.

`timestamp`

Gibt an, wann die Nachricht gesendet wurde AWS IoT. Durch Verwendung des Zeitstempels in der Nachricht und der Zeitstempel für einzelne Attribute im Abschnitt `desired` oder `reported` kann ein Gerät das Alter einer Eigenschaft bestimmen, selbst wenn das Gerät über keine interne Uhr verfügt.

`clientToken`

Eine für das Gerät einmalige Zeichenfolge, die es Ihnen ermöglicht, Anfragen in einer MQTT-Umgebung Antworten zuzuordnen.

`version`

Die Dokumentversion. Jedes Mal, wenn das Dokument aktualisiert wird, erhöht sich diese Versionsnummer. Wird verwendet, um sicherzustellen, dass die Versionsnummer des aktualisierten Dokuments die neueste ist.

Weitere Informationen finden Sie unter Beispiele für Schatten-Dokumente.

Delta-Status

Der Delta-Status ist ein virtueller Typ eines Status, in dem der Unterschied zwischen dem Status `desired` (Soll) Status und dem Status `reported` (gemeldet) enthalten ist. Felder im Abschnitt `desired` (Soll), die nicht im Abschnitt `reported` (gemeldet) enthalten sind, sind im Delta enthalten. Felder, die im Abschnitt `reported` (gemeldet) und nicht im Abschnitt `desired` (Soll) enthalten sind, sind nicht im Delta enthalten. Das Delta enthält Metadaten und seine Werte entsprechen den Metadaten im Feld `desired` (Soll). Beispiel:

```
{
  "state": {
    "desired": {
      "color": "RED",
      "state": "STOP"
    },
    "reported": {
      "color": "GREEN",
      "engine": "ON"
    },
    "delta": {
      "color": "RED",
      "state": "STOP"
    }
  },
  "metadata": {
    "desired": {
      "color": {
        "timestamp": 12345
      },
```

```
      "state": {
        "timestamp": 12345
      }
      },
      "reported": {
        "color": {
          "timestamp": 12345
        },
        "engine": {
          "timestamp": 12345
        }
      },
      "delta": {
        "color": {
          "timestamp": 12345
        },
        "state": {
          "timestamp": 12345
        }
      }
    },
    "version": 17,
    "timestamp": 123456789
  }
}
```

Wenn die verschachtelten Objekte differieren, enthält das Delta den Pfad bis hin zum Stammverzeichnis.

```
{
  "state": {
    "desired": {
      "lights": {
        "color": {
          "r": 255,
          "g": 255,
          "b": 255
        }
      }
    },
    "reported": {
      "lights": {
        "color": {
```

```
          "r": 255,
          "g": 0,
          "b": 255
        }
      }
    },
    "delta": {
      "lights": {
        "color": {
          "g": 255
        }
      }
    }
  },
  "version": 18,
  "timestamp": 123456789
}
```

Der Device Shadow-Service berechnet das Delta, indem er jedes einzelne Feld im Status `desired` (Soll) durchläuft und mit dem Status `reported` (gemeldet) abgleicht.

Arrays werden wie Werte behandelt. Stimmt ein Array im Abschnitt `desired` (Soll) nicht mit dem Array im Abschnitt `reported` (gemeldet) überein, dann wird das gesamte "Soll”-Array in das Delta kopiert.

Versioning von Schattendokumenten

Der Device Shadow-Service unterstützt das Versioning für jede Aktualisierungsnachricht, sowohl Anforderung als auch Antwort. Dies bedeutet, dass bei jeder Aktualisierung eines Schattens die Version des JSON-Dokuments erhöht wird. Dies gewährleistet Folgendes:

- Ein Client kann eine Fehlermeldung empfangen, wenn versucht wird, einen Schatten mit einer älteren Versionsnummer zu überschreiben. Der Client wurde darüber informiert, dass eine neue Synchronisation erfolgen muss, bevor ein Geräteschatten aktualisiert werden kann.
- Ein Client kann entscheiden, bei einer erhaltenen Nachricht nicht tätig zu werden, wenn die Nachricht über eine niedrigere Version verfügt als die vom Client gespeicherte Version.

Ein Client kann den Versionsabgleich umgehen, indem er keine Version in das Schattendokument aufnimmt.

Client-Tokens in Schattendokumenten

Sie können ein Client-Token zusammen mit MQTT-based Nachrichten verwenden, um zu überprüfen, ob dasselbe Client-Token in einer Anfrage und einer Anforderungsantwort enthalten ist. Dies stellt sicher, dass Antwort und Anfrage miteinander verknüpft sind.

> **Note**
>
> Das Client-Token darf nicht länger als 64 Bytes sein. Ein Client-Token, das länger als 64 Bytes ist, verursacht eine 400er-Antwort (Bad Request) und die Fehlermeldung Invalid clientToken (Ungültiges Client-Token).

Eigenschaften des leeren Schattendokuments

Die Eigenschaften `reported` und `desired` in einem Schattendokument können leer sein oder weggelassen werden, wenn sie nicht für den aktuellen Schattenstatus gelten. Ein Schattendokument enthält beispielsweise nur dann eine `desired`-Eigenschaft, wenn es einen gewünschten Status hat. Der folgende Code ist ein gültiges Beispiel für ein Statusdokument ohne `desired`-Eigenschaft:

```
{
    "reported" : { "temp": 55 }
}
```

Die `reported`-Eigenschaft kann auch leer sein, z. B. wenn der Schatten nicht vom Gerät aktualisiert wurde:

```
{
    "desired" : { "color" : "RED" }
}
```

Wenn eine Aktualisierung bewirkt, dass die Eigenschaft `desired` oder `reported` null wird, wird diese aus dem Dokument entfernt. Nachfolgend wird gezeigt, wie Sie die `desired`-Eigenschaft entfernen, indem Sie sie auf `null` festlegen. Sie können dies beispielsweise tun, wenn ein Gerät seinen Status aktualisiert.

```
{
```

```
    "state": {
        "reported": {
            "color": "red"
        },
        "desired": null
    }
}
```

Ein Schattendokument kann auch keine `desired`- oder `reported`-Eigenschaft haben, wodurch das Schattendokument leer wird. Dies ist ein Beispiel für ein leeres, aber gültiges Schattendokument.

```
{
}
```

Array-Werte in Schattendokumenten

Schatten unterstützen zwar Arrays, können diese jedoch so als normale Werte behandeln, dass bei einer Aktualisierung eines Arrays das gesamte Array ersetzt wird. Es ist nicht möglich, einen Teil eines Arrays zu aktualisieren.

Ursprungszustand:

```
{
    "desired" : { "colors" : ["RED", "GREEN", "BLUE" ] }
}
```

Aktualisieren:

```
{
    "desired" : { "colors" : ["RED"] }
}
```

Endzustand:

```
{
    "desired" : { "colors" : ["RED"] }
}
```

Arrays können keine Nullwerte besitzen. Das folgende Array ist z. B. ungültig und wird abgelehnt.

```
{
    "desired" : {
        "colors" : [ null, "RED", "GREEN" ]
    }
}
```

Device Shadow-Fehlermeldungen

Der Device Shadow-Service veröffentlicht (über MQTT) eine Meldung im Thema "Fehler”, wenn ein Versuch, das Statusdokument zu ändern, fehlschlägt. Diese Meldung wird lediglich als Antwort auf eine Veröffentlichungsanfrage zu einem der reservierten `$aws`-Topics ausgegeben. Aktualisiert der Client das Dokument mittels REST-API, erhält er den HTTP-Fehlercode als Teil seiner Antwort, und es werden keine MQTT-Fehlermeldungen ausgegeben.

HTTP-Fehlercode	Fehlermeldungen
400 (Ungültige Anfrage)	• Ungültige JSON • Erforderlicher Knoten fehlt: Status • Der Statusknoten muss ein Objekt sein • Der "Soll”-Knoten muss ein Objekt sein • Der "Gemeldet”-Knoten muss ein Objekt sein • Ungültige Version • Ungültiger Client-Token **Note** Ein Client-Token, das länger als 64 Bytes ist, verursacht diese Antwort. • Die JSON enthält zu viele Verschachtelungsebenen; maximal 6 sind zulässig • Der Status enthält einen ungültigen Knoten
401 (Unauthorized) (Unautorisiert)	• Nicht autorisiert
403 (Forbidden) (Unzulässig)	• Forbidden

HTTP-Fehlercode	Fehlermeldungen
404 (Not Found) (Nicht gefunden)	• Gerät nicht gefunden • Es existiert kein Shadow mit dem Namen: *shadowName*
409 (Conflict) (Konflikt)	• Versionskonflikt
413 (Payload Too Large) (Nutzlast zu hoch)	• Die Nutzlast überschreitet die maximal zulässige Größe
415 (Unsupported Media Type) (Nicht unterstützter Medientyp)	• Die dokumentierte Kodierung wird nicht unterstützt; die unterstützte Kodierung ist UTF-8
429 (Too Many Requests) (Zu viele Anfragen)	• Bei mehr als 10 übertragenen Anfragen auf eine einzelne Verbindung erzeugt der Geräteschatten-Service diese Fehlermeldung. Bei einer laufenden Anfrage handelt es sich um eine Anfrage in Bearbeitung, die zwar gestartet, aber noch nicht abgeschlossen wurde.
500 (Internal Server Error) (Interner Serverfehler)	• Interner Service-Fehler

AWS IoT Device Management Softwarepaket-Katalog

Mit dem AWS IoT Device Management Softwarepaketkatalog können Sie ein Inventar der Softwarepakete und ihrer Versionen verwalten. Sie können Paketversionen einzelnen Dingen und AWS IoT dynamischen Dinggruppen zuordnen und sie über interne Prozesse oder AWS IoT Jobs bereitstellen.

Ein Softwarepaket enthält eine oder mehrere Paketversionen, eine Sammlung von Dateien, die als einzelne Einheit bereitgestellt werden können. Paketversionen können Firmware, Betriebssystemupdates, Geräteanwendungen, Konfigurationen und Sicherheitspatches enthalten. Da sich die Software im Laufe der Zeit weiterentwickelt, können Sie eine neue Paketversion erstellen und sie in Ihrer Flotte einsetzen.

Der AWS IoT Softwarepaket-Hub befindet sich darin AWS IoT Core. Sie können den Hub verwenden, um Ihr Softwarepaketinventar und Ihre Metadaten zentral zu registrieren und zu verwalten, wodurch ein Katalog von Softwarepaketen und deren Versionen erstellt wird. Sie können Geräte auf der Grundlage von Softwarepaketen und Paketversionen gruppieren, die auf dem Gerät bereitgestellt werden. Diese Funktion bietet die Möglichkeit, das geräteseitige Paketinventar als benannten Schatten zu verwalten, Geräte anhand von Versionen zuzuordnen und zu gruppieren und die Verteilung der Paketversionen innerhalb der Flotte anhand von Flottenmetriken zu visualisieren.

Wenn Sie ein internes Softwarebereitstellungssystem eingerichtet haben, können Sie diesen Prozess weiterhin für die Bereitstellung Ihrer Paketversionen verwenden. Wenn Sie noch keinen Bereitstellungsprozess eingerichtet haben oder wenn Sie es vorziehen, empfehlen wir, AWS IoT Jobs zu verwenden, um die Funktionen im Softwarepaket-Katalog zu verwenden. Weitere Informationen finden Sie unter AWS IoT Jobs vorbereiten.

Dieses Kapitel enthält die folgenden Abschnitte:

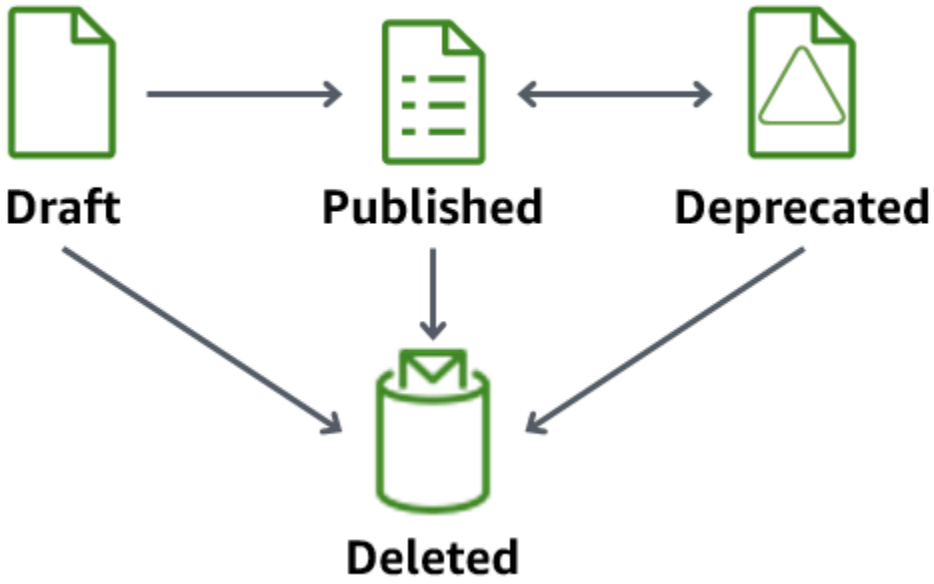
- Vorbereitung der Verwendung des Softwarepaket-Katalogs
- Vorbereitung der Sicherheit
- Vorbereitung der Flottenindizierung
- Schalten Sie die &Snowball;-Appliance ein, indem Sie auf den Ein-/Aus-Schalter über dem E-Ink-Display drücken. AWS IoT Jobs
- Erste Schritte mit dem Softwarepaket-Katalog

Vorbereitung der Verwendung des Softwarepaket-Katalogs

Der folgende Abschnitt bietet einen Überblick über den Lebenszyklus der Paketversion sowie Informationen zur Verwendung des AWS IoT Device Management Softwarepaketkatalogs.

Lebenszyklus der Paketversion

Eine Paketversion kann sich in den folgenden Lebenszyklusstatus weiterentwickeln: `draft`, `published`, und `deprecated`. Sie kann auch `deleted` sein.



- Entwurf

  Wenn Sie eine Paketversion erstellen, befindet sie sich in einem `draft` Status. Dieser Status zeigt an, dass das Softwarepaket gerade vorbereitet wird oder unvollständig ist.

  Solange sich die Paketversion in diesem Zustand befindet, können Sie sie nicht bereitstellen. Sie können die Beschreibung, Attribute und Tags der Paketversion bearbeiten.

  Sie können eine Paketversion, die sich im `draft` Status befindet, mithilfe der Konsole `published` oder `deleted` durch Ausführen der DeletePackageVersion API-Operationen in den Status „UpdatePackageVersionoder“ umstellen.
- Veröffentlicht

  Wenn Ihre Paketversion bereit für die Bereitstellung ist, stellen Sie die Paketversion in einen `published` Status um. In diesem Zustand können Sie wählen, ob Sie die Paketversion als

Standardversion identifizieren möchten, indem Sie das Softwarepaket in der Konsole bearbeiten oder den UpdatePackage API-Vorgang ausführen. In diesem Zustand können Sie nur die Beschreibung und die Tags bearbeiten.

Sie können eine Paketversion, die sich im `published` Status befindet, auf `deprecated` oder in der sich befinden, `deleted` mithilfe der Konsole UpdatePackageVersion oder der DeletePackageVersion API-Operationen umstellen.

- Als veraltet gekennzeichnet

  Wenn eine neue Paketversion verfügbar ist, können Sie frühere Paketversionen auf `deprecated` umstellen. Sie können weiterhin Jobs mit einer veralteten Paketversion bereitstellen. Sie können auch eine veraltete Paketversion als Standardversion benennen und nur die Beschreibung und die Tags bearbeiten.

  Erwägen Sie, eine Paketversion auf eine Version umzustellen, `deprecated` wenn die Version veraltet ist, Sie aber immer noch Geräte im Feld haben, die die ältere Version verwenden oder sie aufgrund von Laufzeitabhängigkeit pflegen müssen.

  Sie können eine Paketversion, die sich im `deprecated` Status befindet, auf die Konsole `deleted` umstellen `published` oder entweder die UpdatePackageVersion oder DeletePackageVersion API-Operationen ausführen.

- Deleted (Gelöscht)

  Wenn Sie eine Paketversion nicht mehr verwenden möchten, können Sie sie löschen, indem Sie die Konsole verwenden oder die DeletePackageVersion API-Operation ausführen.

  > **Note**
  >
  > Wenn Sie eine Paketversion löschen, während noch ausstehende Aufträge darauf verweisen, erhalten Sie eine Fehlermeldung, wenn der Auftrag erfolgreich abgeschlossen wurde und versucht wird, den reservierten Named Shadow zu aktualisieren.
  > Wenn die Softwarepaketversion, die Sie löschen möchten, als Standardpaketversion benannt ist, müssen Sie das Paket zuerst aktualisieren, um eine andere Version als Standardversion zu benennen, oder das Feld unbenannt lassen. Sie können dies tun, indem Sie die Konsole oder den UpdatePackageVersion API-Vorgang verwenden. (Um eine benannte Paketversion standardmäßig zu entfernen, setzen Sie den DefaultVersion Parameter unset auf true, wenn Sie den UpdatePackage API-Vorgang ausführen.)

Wenn Sie ein Softwarepaket über die Konsole löschen, werden alle mit diesem Paket verknüpften Paketversionen gelöscht, sofern nicht eine als Standardversion benannt ist.

Namenskonventionen für Paketversionen

Wenn Sie Paketversionen benennen, ist es wichtig, eine logische Benennungsstrategie zu planen und anzuwenden, damit Sie und andere leicht die neueste Paketversion und den Versionsverlauf erkennen können. Sie müssen bei der Erstellung der Paketversion einen Versionsnamen angeben, aber die Strategie und das Format hängen weitgehend von Ihrem Geschäftsszenario ab.

Als bewährte Methode empfehlen wir die Verwendung des Semantic SemVer Versioning-Formats. Zum Beispiel, `1.2.3` wo `1` die Hauptversion für funktionell inkompatible Änderungen ist, `2` die Hauptversion für funktionell kompatible Änderungen und `3` die Patch-Version (für Fehlerbehebungen) ist. Weitere Informationen finden Sie unter Semantic Versioning 2.0.0. Weitere Informationen zu den Anforderungen an den Paketversionsnamen finden Sie unter VersionName im AWS IoT API-Referenzhandbuch.

Standardversion

Das Festlegen einer Version als Standard ist optional. Sie können Standard-Paketversionen hinzufügen oder entfernen. Sie können auch eine Paketversion bereitstellen, die nicht als Standardversion benannt ist.

Wenn Sie eine Paketversion erstellen, wird sie in einen `draft` Status versetzt und kann erst dann als Standardversion bezeichnet werden, wenn Sie die Paketversion auf veröffentlicht umstellen. Der Softwarepaket-Katalog wählt nicht automatisch eine Version als Standard aus oder aktualisiert eine neuere Paketversion als Standard. Sie müssen die von Ihnen gewählte Paketversion bewusst über die Konsole oder durch Ausführen der UpdatePackageVersion API-Operation benennen.

Versionsattribute

Versionsattribute und ihre Werte enthalten wichtige Informationen über Ihre Paketversionen. Wir empfehlen Ihnen, allgemeine Attribute für ein Paket oder eine Paketversion zu definieren. Sie können beispielsweise ein Name-Wert-Paar für Plattform, Architektur, Betriebssystem, Veröffentlichungsdatum, Autor oder Amazon-S3-URL erstellen.

Wenn Sie einen AWS IoT Job mit einem Jobdokument erstellen, können Sie auch eine Substitutionsvariable (`$parameter`) verwenden, die auf den Wert eines Attributs verweist. Weitere

Informationen finden Sie unter AWS IoT Jobs https://docs.aws.amazon.com/iot/latest/developerguide/preparing-jobs-for-service-package-catalog.html vorbereiten.

Versionsattribute, die in Paketversionen verwendet werden, werden nicht automatisch zum reservierten benannten Schatten hinzugefügt und können nicht direkt über Fleet Indexing indexiert oder abgefragt werden. Um Paketversionsattribute über Fleet Indexing zu indizieren oder abzufragen, können Sie das Versionsattribut im reservierten benannten Schatten auffüllen.

Wir empfehlen, dass der Versionsattributparameter im reservierten Schatten die vom Gerät gemeldeten Eigenschaften erfasst, z. B. das Betriebssystem und die Installationszeit. Sie können auch über Fleet Indexing indexiert und abgefragt werden.

Für Versionsattribute ist es nicht erforderlich, dass sie einer bestimmten Benennungskonvention entsprechen. Sie können Name-Wert-Paare erstellen, um Ihren Geschäftsanforderungen gerecht zu werden. Die Gesamtgröße aller Attribute in einer Paketversion ist auf 3 KB begrenzt. Weitere Informationen finden Sie unter Beschränkungen für Softwarepakete und Paketversionen im Softwarepaket-Katalog.

Alle Attribute in einem Jobdokument verwenden

Sie können festlegen, dass alle Attribute der Paketversion automatisch zu Ihrer Auftragsbereitstellung für ausgewählte Geräte hinzugefügt werden. Informationen zur automatischen Verwendung aller Paketversionsattribute programmgesteuert in einem API- oder CLI-Befehl finden Sie im folgenden Beispiel für ein Jobdokument:

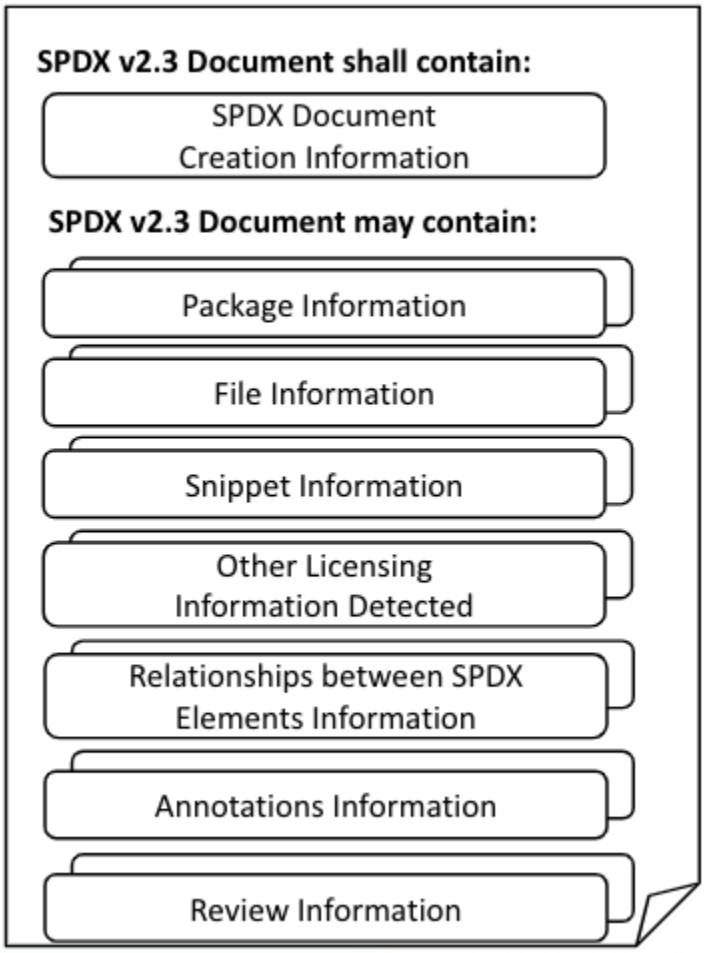
```
"TestPackage": "${aws:iot:package:TestPackage:version:PackageVersion:attributes}"
```

Software-Stückliste

Die Softwareliste (SBOM) bietet eine zentrale Ablage für alle Aspekte Ihres Softwarepakets. Zusätzlich zum Speichern von Softwarepaketen und Paketversionen können Sie die jeder Paketversion zugeordnete Softwareliste (SBOM) im AWS IoT Device Management Softwarepaketkatalog speichern. Ein Softwarepaket enthält eine oder mehrere Paketversionen, und jede Paketversion besteht aus einer oder mehreren Komponenten. Jede dieser Komponenten, die die Zusammensetzung einer bestimmten Paketversion unterstützen, kann mithilfe einer Softwareliste beschrieben und katalogisiert werden. Die unterstützten Industriestandards für Softwarelisten sind SPDX und CyclonedX. Wenn eine SBOM zum ersten Mal erstellt wird, wird sie anhand des SPDX- und CyclonedX-Industriestandardformats validiert. Weitere Informationen zu SPDX finden Sie unter

System Package Data Exchange. https://spdx.dev/ Weitere Informationen zu CyclonedX finden Sie unter CyclonedX. https://cyclonedx.org/

Die Softwareliste beschreibt alle Aspekte der Komponenten einer bestimmten Paketversion, wie Paketinformationen, Dateiinformationen und andere relevante Metadaten. Sehen Sie sich das folgende Beispiel für eine Dokumentstruktur einer Softwareliste im SPDX-Format an:



Vorteile der Softwareliste

Einer der wichtigsten Vorteile, wenn Sie Ihre Softwareliste für eine Paketversion zum Softwarepaketkatalog hinzufügen, ist das Schwachstellenmanagement.

Verwaltung von Sicherheitslücken

Die Bewertung und Minderung Ihrer Sicherheitsanfälligkeit gegenüber offensichtlichen Sicherheitsrisiken in Softwarekomponenten ist nach wie vor von entscheidender Bedeutung für den Schutz der Integrität Ihrer Geräteflotte. Durch Hinzufügen der Softwareliste, die für jede Paketversion im Softwarepaketkatalog gespeichert ist, können Sie Sicherheitslücken proaktiv aufdecken, indem

Sie anhand der Paketversion und der SBOM wissen, welche Geräte gefährdet sind, und zwar mithilfe Ihrer eigenen internen Schwachstellenmanagement-Lösung. Sie können Fixes für die betroffenen Geräte bereitstellen und Ihre Geräteflotte schützen.

Aufbewahrung der Stückliste der Software

Die Softwareliste (SBOM) für jede Softwarepaketversion wird mithilfe der Amazon S3-Versionierungsfunktion in einem Amazon S3-Bucket gespeichert. Der Amazon S3-Bucket, in dem die SBOM gespeichert ist, muss sich in derselben Region befinden, in der die Paketversion erstellt wurde. Ein Amazon S3-Bucket, der die Versionierungsfunktion verwendet, verwaltet mehrere Varianten eines Objekts im selben Bucket. Weitere Informationen zur Verwendung der Versionierung in einem Amazon S3-Bucket finden Sie unter Verwenden der Versionierung in Amazon S3-Buckets.

> **Note**
>
> An jede Softwarepaketversion können mehrere SBOM-Dateien angehängt werden, die SBOM-Dateien müssen jedoch in einer einzigen ZIP-Archivdatei gespeichert werden.

Der spezifische Amazon S3-Schlüssel und die Versions-ID für Ihren Bucket werden verwendet, um jede Version einer Software-Stückliste für eine Paketversion eindeutig zu identifizieren.

> **Note**
>
> Für eine Paketversion mit einer einzelnen SBOM-Datei können Sie diese SBOM-Datei in Ihrem Amazon S3-Bucket als ZIP-Archivdatei speichern.
> Für eine Paketversion mit mehreren SBOM-Dateien müssen Sie alle SBOM-Dateien in einer einzigen ZIP-Archivdatei platzieren und diese Zip-Archivdatei dann in Ihrem Amazon S3-Bucket speichern.
> Alle SBOM-Dateien, die in beiden Szenarien in der einzelnen ZIP-Archivdatei gespeichert sind, sind entweder als SPDX- oder CyclonedX-JSON-Dateien formatiert.

Richtlinien für Berechtigungen

Damit AWS IoT Sie als angegebener Principal auf die im Amazon S3-Bucket gespeicherten SBOM-ZIP-Archivdateien zugreifen können, benötigen Sie eine ressourcenbasierte Berechtigungsrichtlinie. Im folgenden Beispiel finden Sie die richtige ressourcenbasierte Berechtigungsrichtlinie:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "iot.amazonaws.com"
                ]
        },
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::bucketName/*"
        }
    ]
}
```

Weitere Informationen zu ressourcenbasierten Berechtigungsrichtlinien finden Sie unter AWS IoT Ressourcenbasierte Richtlinien

Aktualisierung der SBOM

Sie können die Softwareliste so oft wie nötig aktualisieren, um Ihre Geräteflotte zu schützen und zu erweitern. Jedes Mal, wenn die Software-Stückliste in Ihrem Amazon S3-Bucket aktualisiert wird, ändert sich die Versions-ID und Sie müssen die neue Amazon S3-Bucket-URL mit der entsprechenden Softwarepaketversion verknüpfen. Sie finden die neue Versions-ID in der Spalte Amazon S3-Objektversions-ID auf der Seite mit der Paketversion im AWS-Managementkonsole. Darüber hinaus können Sie die API-Operation `GetPackageVersion` oder den CLI-Befehl verwenden`get-package-version`, um die neue Versions-ID anzuzeigen.

> **Note**
>
> Die Aktualisierung Ihrer Softwareliste, die zu einer neuen Versions-ID führt, führt nicht dazu, dass eine neue Paketversion erstellt wird.

Weitere Informationen zu Amazon S3-Objektschlüsseln finden Sie unter Objektschlüsselnamen erstellen.

Aktivieren AWS IoT -Flottenindizierung

Die Aktivierung der AWS IoT Flottenindizierung ist eine Voraussetzung für die Verwendung des AWS IoT Device Management Softwarepaketkatalogs. Um die AWS IoT Flottenindizierung mit dem Softwarepaketkatalog zu nutzen, legen Sie den reservierten Namen Shadow (`$package`) als Datenquelle für jedes Gerät fest, auf dem Sie indizieren und Metriken sammeln möchten. Weitere Informationen zu reservierten benannten Shadows finden Sie unterReservierter benannter Schatten.

Die Flottenindizierung bietet Unterstützung, die es ermöglicht, AWS IoT Dinge anhand dynamischer Dinggruppen zu gruppieren, die nach Softwarepaketversion gefiltert werden. Mit der Flottenindizierung können beispielsweise Objekte identifiziert werden, für die eine bestimmte Paketversion installiert ist oder für die keine Paketversionen installiert sind oder die bestimmten Name-Wert-Paaren entsprechen. Schließlich bietet die Flottenindizierung standardmäßige und benutzerdefinierte Metriken, anhand derer Sie sich einen Überblick über den Zustand Ihrer Geräteflotte verschaffen können. Weitere Informationen finden Sie unter Vorbereitung der Flottenindizierung.

> **Note**
>
> Die Aktivierung der Flottenindizierung für den Softwarepaket-Katalog verursacht Standard-Servicekosten. Weitere Informationen finden Sie unter AWS IoT Device Management-Preisgestaltung.

Reservierter benannter Schatten

Der reservierte, benannte Schatten, `$package`, gibt den Status der auf dem Gerät installierten Softwarepakete und Paketversionen wieder. Die Flottenindizierung verwendet den reservierten benannten Schatten als Datenquelle, um Standard- und benutzerdefinierte Messwerte zu erstellen, mit denen Sie den Status Ihrer Flotte abfragen können. Weitere Informationen finden Sie unter Vorbereitung der Flottenindizierung

Ein reservierter benannter Schatten ähnelt einem benannten Schatten mit der Ausnahme, dass sein Name vordefiniert ist und Sie ihn nicht ändern können. Darüber hinaus aktualisiert sich der reservierte benannte Schatten nicht mit Metadaten und verwendet nur die Schlüsselwörter `version` und `attributes`.

Bei Aktualisierungsanfragen, die andere Stichwörter enthalten, wie z. B. `description`, wird unter dem `rejected` Thema eine Fehlermeldung angezeigt. Weitere Informationen finden Sie unter Geräteschatten-Fehlermeldungen.

Sie kann erstellt werden, wenn Sie über die Konsole AWS IoT etwas erstellen, wenn ein AWS IoT Job erfolgreich abgeschlossen und der Shadow aktualisiert wird und wenn Sie die `UpdateThingShadow` API-Operation ausführen. Weitere Informationen finden Sie UpdateThingShadow im AWS IoT Core Entwicklerhandbuch.

> **Note**
>
> Die Indizierung des reservierten benannten Schattens wird nicht auf die Anzahl der benannten Schatten angerechnet, die von der Flottenindizierung indexiert werden können. Weitere Informationen finden Sie unter AWS IoT Device Management Flottenindizierungsgrenzen und -quoten. Wenn Sie außerdem festlegen, dass AWS IoT Jobs den reservierten Named Shadow aktualisieren, wenn ein Job erfolgreich abgeschlossen wird, wird der API-Aufruf auf Ihre Device Shadow- und Registrierungsvorgänge angerechnet und kann mit Kosten verbunden sein. Weitere Informationen finden Sie unter Grenzwerte und Kontingente für AWS IoT Device Management Aufträge und zum IndexingFilter API-Datentyp.

Struktur des **`$package`** Schattens

Der reservierte benannte Schatten enthält Folgendes:

```
{
    "state": {
        "reported": {
            "<packageName>": {
                "version": "",
                "attributes": {
                }
            }
        }
    },
    "version" : 1
    "timestamp" : 1672531201
}
```

Die Schatteneigenschaften werden mit den folgenden Informationen aktualisiert:

- `<packageName>`: Der Name des installierten Softwarepakets, das mit dem PackageName-Parameter aktualisiert wird.
- `version`: Der Name der installierten Paketversion, die mit dem VersionName-Parameter aktualisiert wird.
- `attributes`: Optionale Metadaten, die vom Gerät gespeichert und durch Flottenindizierung indexiert werden. Auf diese Weise können Kunden ihre Indizes auf der Grundlage der gespeicherten Daten abfragen.
- `version`: Die Versionsnummer des Schattens. Sie wird jedes Mal, wenn der Schatten aktualisiert wird, automatisch erhöht und beginnt bei 1.
- `timestamp`: Gibt an, wann der Schatten zuletzt aktualisiert wurde, und wird in Unix-Zeit aufgezeichnet.

Weitere Informationen zum Format und Verhalten eines benannten Schattens finden Sie unter AWS-IoT-Device-Shadow-Service Reihenfolge der Nachrichten.

Löschen eines Softwarepakets und seiner Paketversionen

Führen Sie vor dem Löschen eines Softwarepakets die folgenen Schritte aus:

- Vergewissern Sie sich, dass das Paket und seine Versionen nicht aktiv bereitgestellt werden.
- Löschen Sie zuerst alle zugehörigen Versionen. Wenn eine der Versionen als Standardversion gekennzeichnet ist, müssen Sie die benannte Standardversion aus dem Paket entfernen. Da die Angabe einer Standardversion optional ist, besteht kein Konflikt darin, sie zu entfernen. Um die Standardversion aus dem Softwarepaket zu entfernen, bearbeiten Sie das Paket über die Konsole oder verwenden Sie die UpdatePackageVersion API-Operation.

Solange es keine benannte Standardpaketversion gibt, können Sie die Konsole verwenden, um ein Softwarepaket zu löschen. Alle zugehörigen Paketversionen werden ebenfalls gelöscht. Wenn Sie einen API-Aufruf zum Löschen von Softwarepaketen verwenden, müssen Sie zuerst die Paketversionen und dann das Softwarepaket löschen.

Vorbereitung der Sicherheit

In diesem Abschnitt werden die wichtigsten Sicherheitsanforderungen für den AWS IoT Device Management Softwarepaketkatalog beschrieben.

Resource-based authentication

Der Softwarepaket-Katalog verwendet ressourcenbasierte Autorisierung, um zusätzliche Sicherheit bei der Aktualisierung von Software auf Ihrer Flotte zu bieten. Das bedeutet, dass Sie eine AWS Identity and Access Management (IAM-) Richtlinie erstellen müssen, die Rechte zum Ausführen von`create`, `read` `update``delete`, und `list` Aktionen für Softwarepakete und Paketversionen gewährt, und in diesem `Resources` Abschnitt auf die spezifischen Softwarepakete und Paketversionen verweisen müssen, die Sie bereitstellen möchten. Sie benötigen diese Rechte auch, damit Sie den reservierten benannten Schatten aktualisieren können. Sie verweisen auf die Softwarepakete und Paketversionen, indem Sie einen Amazon-Ressourcennamen (ARN) für jede Entität angeben.

> **Note**
>
> Wenn Sie beabsichtigen, mit der Richtlinie Rechte für API-Aufrufe der Paketversion (wie CreatePackageVersion,, DeletePackageVersion) zu gewähren UpdatePackageVersion, müssen Sie sowohl die ARNs für das Softwarepaket als auch für die Paketversion in die Richtlinie aufnehmen. Wenn Sie beabsichtigen, mit der Richtlinie Rechte für API-Aufrufe von Softwarepaketen (wie CreatePackage UpdatePackage, und DeletePackage) zu gewähren, müssen Sie nur den Softwarepaket-ARN in die Richtlinie aufnehmen.

Strukturieren Sie die ARNs für das Softwarepaket und die Paketversion wie folgt:

- Softwarepaket: `arn:aws:iot:<region>:<accountID>:package/<packageName>/package`
- Paketversion: `arn:aws:iot:<region>:<accountID>:package/<packageName>/version/<versionName>`

> **Note**
>
> Es gibt weitere verwandte Rechte, die Sie möglicherweise in diese Richtlinie aufnehmen. Sie könnten beispielsweise einen ARN für `job`, `thinggroup`, und `jobtemplate` angeben. Weitere Informationen und eine vollständige Liste der Richtlinienoptionen finden Sie unter Schützen von Benutzern und Geräten mit AWS IoT Jobs.

Wenn Sie beispielsweise über ein Softwarepaket und eine Paketversion verfügen, die wie folgt benannt sind:

- AWS IoT Sache: `myThing`
- Paketname: `samplePackage`
- Version `1.0.0`

Die Richtlinie könnte wie das folgende Beispiel aussehen:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:createPackage",
                "iot:createPackageVersion",
                "iot:updatePackage",
                "iot:updatePackageVersion"
            ],
            "Resource": [
               "arn:aws:iot:us-east-1:111122223333:package/samplePackage",
               "arn:aws:iot:us-east-1:111122223333:package/samplePackage/
version/1.0.0"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
                "iot:UpdateThingShadow"
```

```
            ],
            "Resource": "arn:aws:iot:us-east-1:111122223333:thing/myThing/$package"
        }
    ]
}
```

AWS IoT Arbeitsrechte zum Bereitstellen von Paketversionen

Aus Sicherheitsgründen ist es wichtig, dass Sie Rechte zur Bereitstellung von Paketen und Paketversionen gewähren und die spezifischen Pakete und Paketversionen benennen, die sie bereitstellen dürfen. Zu diesem Zweck erstellen Sie eine IAM-Rolle und eine IAM-Richtlinie, die die Erlaubnis zum Bereitstellen von Aufträgen mit Paketversionen erteilt. Die Richtlinie muss die Zielpaketversionen als Ressource angeben.

IAM-Richtlinie

Die IAM-Richtlinie gewährt das Recht, einen Job zu erstellen, der das Paket und die Version enthält, die im `Resource` Abschnitt genannt werden.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:CreateJob",
                "iot:CreateJobTemplate"
            ],
            "Resource":[
                "arn:aws:iot:*:111122223333:job/<jobId>",
                "arn:aws:iot:*:111122223333:thing/<thingName>/$package",
                "arn:aws:iot:*:111122223333:thinggroup/<thingGroupName>",
                "arn:aws:iot:*:111122223333:jobtemplate/<jobTemplateName>",
                "arn:aws:iot:*:111122223333:package/<packageName>/
version/<versionName>"
            ]
        }
    ]
}
```

> **Note**
>
> Wenn Sie einen Auftrag bereitstellen möchten, der ein Softwarepaket und eine Paketversion deinstalliert, müssen Sie einen ARN autorisieren, in dem sich die Paketversion `$null` befindet, z. B. im Folgenden:

```
arn:aws:iot:<regionCode>:111122223333:package/<packageName>/version/$null
```

AWS IoT Jobrechte zur Aktualisierung des reservierten benannten Shadows

Damit Aufträge den reservierten Namensschatten des Objekts aktualisieren können, wenn der Auftrag erfolgreich abgeschlossen wurde, müssen Sie eine IAM-Rolle und eine Richtlinie erstellen. Dafür gibt es zwei Möglichkeiten in der AWS IoT Konsole. Die erste Möglichkeit besteht darin, ein Softwarepaket in der Konsole zu erstellen. Wenn das Dialogfeld Abhängigkeiten für die Paketverwaltung aktivieren angezeigt wird, können Sie wählen, ob Sie eine vorhandene Rolle verwenden oder eine neue Rolle erstellen möchten. Oder wählen Sie in der AWS IoT Konsole Einstellungen, Indizierung verwalten und dann Indizierung für Gerätepakete und Versionen verwalten aus.

> **Note**
>
> Wenn Sie sich dafür entscheiden, dass der AWS IoT Job Service den reservierten Named Shadow aktualisiert, wenn ein Job erfolgreich abgeschlossen wurde, wird der API-Aufruf auf Ihre Device Shadow- und Registrierungsvorgänge angerechnet und kann mit Kosten verbunden sein. Weitere Informationen finden Sie unter AWS IoT Core Preise.

Wenn Sie die Option Rolle erstellen verwenden, beginnt der Name der generierten Rolle mit `aws-iot-role-update-shadows` und enthält die folgenden Richtlinien:

Einrichten einer Rolle

Berechtigungen

Die Berechtigungsrichtlinie gewährt die Rechte, den Objektschatten abzufragen und zu aktualisieren. Der `$package` Parameter im Ressourcen-ARN zielt auf den reservierten benannten Schatten ab.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:DescribeEndpoint",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
                "iot:UpdateThingShadow"
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:111122223333:thing/<thingName>/$package"
            ]
        }
    ]
}
```

Vertrauensstellung

Zusätzlich zur Berechtigungsrichtlinie erfordert die Rolle eine Vertrauensbeziehung mit AWS IoT Core , sodass die Entität die Rolle übernehmen und den reservierten benannten Schatten aktualisieren kann.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
        },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Eine Benutzerrichtlinie einrichten

iam: Erlaubnis PassRole

Schließlich müssen Sie die Berechtigung haben, die Rolle zu übergeben, AWS IoT Core wenn Sie den UpdatePackageConfiguration API-Vorgang aufrufen.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole",
                "iot:UpdatePackageConfiguration"
            ],
            "Resource": "arn:aws:iam::111122223333:role/<roleName>"
        }
    ]
}
```

AWS IoT Jobs: Berechtigungen zum Herunterladen von Amazon S3

Das Auftragsdokument wird in Amazon S3 gespeichert. Sie verweisen auf diese Datei, wenn Sie den Versand über AWS IoT Jobs durchführen. Sie müssen AWS IoT Jobs die Rechte zum Herunterladen der Datei gewähren (`s3:GetObject`). Sie müssen auch eine Vertrauensbeziehung zwischen Amazon S3 und AWS IoT Jobs einrichten. Anweisungen zum Erstellen dieser Richtlinien finden Sie unter Vorsignierte URLs unter Jobs verwalten.

Berechtigungen zur Aktualisierung der Softwareliste für eine Paketversion

Um die Software-Stückliste für eine Paketversion im Status`Draft`,`Published`, oder `Deprecated` Lebenszyklus zu aktualisieren, benötigen Sie eine AWS Identity and Access Management Rolle und Richtlinien, um die neue Software-Stückliste in Amazon S3 zu finden und die Paketversion in zu aktualisieren AWS IoT Core.

Zunächst platzieren Sie die aktualisierte Software-Stückliste in Ihrem versionierten Amazon S3-Bucket und rufen den `UpdatePackageVersion` API-Vorgang mit dem enthaltenen `sboms` Parameter auf. Als Nächstes übernimmt Ihr autorisierter Principal die von Ihnen erstellte IAM-Rolle, sucht die aktualisierte Software-Stückliste in Amazon S3 und aktualisiert die Paketversion im Software Package AWS IoT Core Catalog.

Die folgenden Richtlinien sind erforderlich, um dieses Update durchzuführen:

Richtlinien

- Vertrauensrichtlinie, die eine Vertrauensbeziehung mit dem autorisierten Principal herstellt, der die IAM-Rolle übernimmt, sodass dieser die aktualisierte Software-Stückliste aus Ihrem versionierten Bucket in Amazon S3 finden und die Paketversion in aktualisieren kann. AWS IoT Core
  -
    ```
    {
        "Version":"2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {
                    "Service": "s3.amazonaws.com"
            },
                "Action": "sts:AssumeRole"
            }
        ]
    }
    ```
  -
    ```
    {
        "Version":"2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {
                    "Service": "iot.amazonaws.com"
            },
                "Action": "sts:AssumeRole"
            }
        ]
    }
    ```
- Berechtigungsrichtlinie: Richtlinie für den Zugriff auf den Amazon S3-Bucket, in dem die Softwareliste für eine Paketversion gespeichert ist, und für die Aktualisierung der Paketversion in. AWS IoT Core
  -
    ```
    {
        "Version":"2012-10-17",
        "Statement": [
    ```

```
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::awsexamplebucket1"
            ]
        }
    ]
}
```

-

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:UpdatePackageVersion"
            ],
            "Resource": [
                "arn:aws:iot:*:111122223333:package/<packageName>/
version/<versionName>"
            ]
        }
    ]
}
```

- Rollenberechtigungen weitergeben: Richtlinie, die die Erlaubnis erteilt, die IAM-Rolle an Amazon S3 zu übergeben und AWS IoT Core wenn Sie den UpdatePackageVersion API-Vorgang aufrufen.

  -

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole",
        "s3:GetObject"
      ],
      "Resource": [
```

```
        "arn:aws:s3:::awsexamplebucket1"
      ]
    }
  ]
}
```

- ```
  {
      "Version":"2012-10-17",
      "Statement": [
          {
              "Effect": "Allow",
              "Action": [
                  "iam:PassRole",
                  "iot:UpdatePackageVersion"
              ],
              "Resource": "arn:aws:iam::111122223333:role/<roleName>"
          }
      ]
  }
  ```

> **Note**
>
> Sie können die Softwareliste einer Paketversion, die in den `Deleted` Lebenszyklusstatus übergegangen ist, nicht aktualisieren.

Weitere Informationen zum Erstellen einer IAM-Rolle für einen AWS Service finden Sie unter Erstellen einer Rolle zum Delegieren von Berechtigungen an einen Service. AWS

Weitere Informationen zum Erstellen eines Amazon S3-Buckets und zum Hochladen von Objekten in diesen Bucket finden Sie unter Einen Bucket erstellen und Objekte hochladen.

Vorbereitung der Flottenindizierung

Mit der AWS IoT Flottenindizierung können Sie Daten suchen und aggregieren, indem Sie den reservierten Namen shadow () verwenden. `$package` Sie können AWS IoT Dinge auch gruppieren, indem Sie die Reservierter benannter Schatten und die dynamischen Dinggruppen abfragen. Sie können beispielsweise Informationen darüber finden, welche AWS IoT Dinge eine bestimmte Paketversion verwenden, keine bestimmte Paketversion installiert haben oder bei denen keine
  ```

Paketversion installiert ist. Sie können weitere Erkenntnisse gewinnen, indem Sie Attribute kombinieren. Zum Beispiel die Identifizierung von Objekten, die eine bestimmte Version und einen bestimmten Objekttyp haben (wie Version 1.0.0 und den Objekttyp pump_sensor). Weitere Informationen finden Sie unter Flottenindizierung.

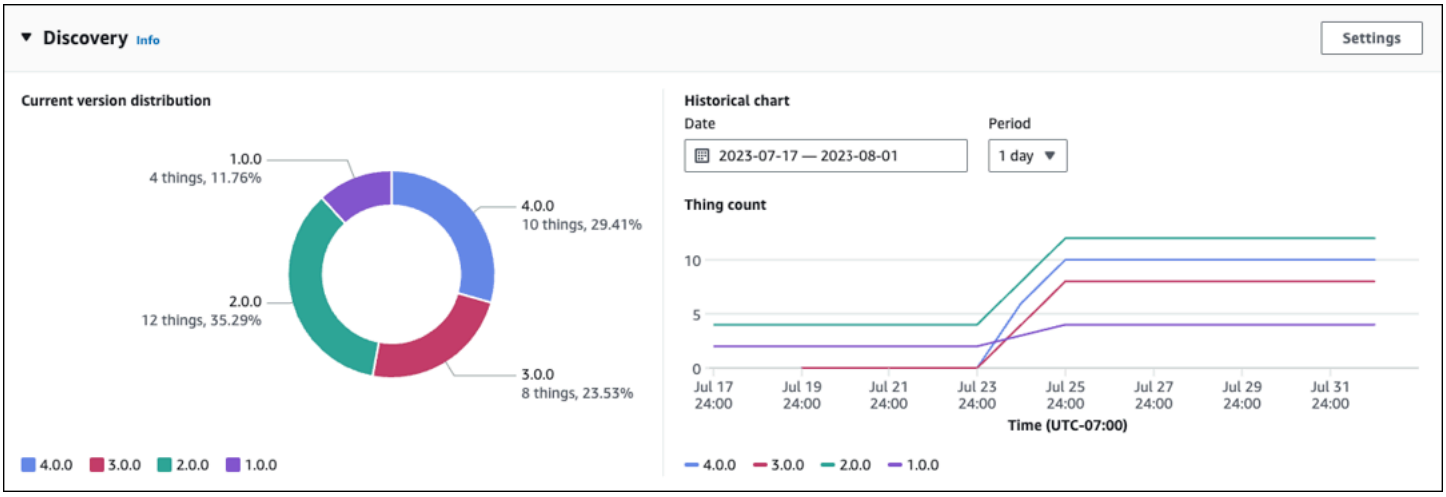
Den **$package** Shadow als Datenquelle festlegen

Um die Flottenindizierung mit dem Softwarepaket-Katalog zu verwenden, müssen Sie die Flottenindizierung aktivieren, den benannten Schatten als Datenquelle festlegen und `$package` als benannten Schattenfilter definieren. Wenn Sie die Flottenindizierung nicht aktiviert haben, können Sie sie im Rahmen dieses Vorgangs aktivieren. Öffnen Sie von AWS IoT Core in der Konsole Einstellungen, wählen Sie Indizierung verwalten, dann Benannte Schatten hinzufügen, Gerätesoftwarepakete und -versionen hinzufügen und Aktualisieren. Weitere Informationen finden Sie unter Verwalten der Objektindizierung.

Alternativ können Sie die Flottenindizierung aktivieren, wenn Sie Ihr erstes Paket erstellen. Wenn das Dialogfeld Abhängigkeiten für die Paketverwaltung aktivieren angezeigt wird, wählen Sie die Option, Gerätesoftwarepakete und -versionen als Datenquellen zur Flottenindizierung hinzuzufügen. Durch Auswahl dieser Option aktivieren Sie auch die Flottenindizierung.

> **Note**
>
> Die Aktivierung der Flottenindizierung für den Softwarepaket-Katalog verursacht Standard-Servicekosten. Weitere Informationen finden Sie unter AWS IoT Device Management-Preisgestaltung.

In der Konsole dargestellte Metriken



Auf der Detailseite des AWS IoT Konsolen-Softwarepakets werden im Discovery-Bereich Standardmetriken angezeigt, die über den `$package` Shadow aufgenommen wurden.

- Das Verteilungsdiagramm „Aktuelle Version“ zeigt die Anzahl der Geräte und den Prozentsatz der 10 neuesten Paketversionen, die einem AWS IoT Objekt aus allen Geräten zugeordnet sind, die diesem Softwarepaket zugeordnet sind. Hinweis: Wenn das Softwarepaket mehr Paketversionen als die in der Tabelle angegebenen enthält, finden Sie diese unter Andere gruppiert.
- Das Verlaufsdiagramm zeigt die Anzahl der Geräte, die den ausgewählten Paketversionen über einen bestimmten Zeitraum zugeordnet sind. Das Diagramm ist zunächst leer, bis Sie bis zu 5 Paketversionen auswählen und den Datumsbereich und das Zeitintervall definieren. Um die Parameter des Diagramms auszuwählen, wählen Sie Einstellungen. Die im Verlaufsdiagramm angezeigten Daten unterscheiden sich möglicherweise vom Verteilungsdiagramm der aktuellen Version. Dies liegt an der unterschiedlichen Anzahl der angezeigten Paketversionen und auch daran, dass Sie im Verlaufsdiagramm auswählen können, welche Paketversionen analysiert werden sollen. Hinweis: Wenn Sie eine Paketversion zur Visualisierung auswählen, wird diese auf die maximale Anzahl von Flottenkennzahlen angerechnet. Weitere Informationen finden Sie unter Flottenindizierungsgrenzen und -quoten.

Eine weitere Methode, um einen Einblick in die Erfassung der Paketversionsverteilung zu erhalten, finden Sie unter Erfassung der Paketversionsverteilung durch `getBucketsAggregation`.

Abfragemuster

Die Flottenindizierung mit dem Softwarepaketkatalog verwendet die meisten unterstützten Standardfunktionen (wie Begriffe, Phrasen und Suchfelder). Vergleichsoperatoren (z. B. kleiner als < und größer als>) und `range` Abfragen sind jedoch für den reservierten Named Shadow (`$package`) `version` -Schlüssel nicht verfügbar. Diese Abfragen sind für den `attributes` Schlüssel verfügbar. Weitere Informationen finden Sie unter Abfragesyntax.

Beispiel für Daten

Hinweis: Informationen zum reservierten benannten Schatten und seiner Struktur finden Sie unter Reservierter benannter Schatten.

In diesem Beispiel wird ein erstes Gerät `AnyThing` benannt und es sind die folgenden Pakete installiert:

- Softwarepaket: `SamplePackage`

  Paketversion: `1.0.0`

  Paket-ID: `1111`

Der Schatten sieht wie folgt aus:

```
{
    "state": {
        "reported": {
            "SamplePackage": {
                "version": "1.0.0",
                "attributes": {
                    "s3UrlForSamplePackage": "https://EXAMPIEBUCKET.s3.us-
west-2.amazonaws.com/exampleCodeFile1",
                    "packageID": "1111"
                    }
            }
        }
    }
}
```

Ein zweites Gerät wird `AnotherThing` benannt und hat das folgende Paket installiert:

- Softwarepaket: `SamplePackage`

  Paketversion: `1.0.0`

  Paket-ID: `1111`
- Softwarepaket: `OtherPackage`

  Paketversion: `1.2.5`

  Paket-ID: `2222`

Der Schatten sieht wie folgt aus:

```
{
    "state": {
        "reported": {
            "SamplePackage": {
                "version": "1.0.0",
                "attributes": {
                    "s3UrlForSamplePackage": "https://EXAMPIEBUCKET.s3.us-
west-2.amazonaws.com/exampleCodeFile1",
                    "packageID": "1111"
                }
            },
            "OtherPackage": {
                "version": "1.2.5",
                "attributes": {
                    "s3UrlForOtherPackage": "https://EXAMPIEBUCKET.s3.us-
west-2.amazonaws.com/exampleCodeFile2",
                    "packageID": "2222"
                    }
            },
        }
    }
}
```

Beispielabfragen

In der folgenden Tabelle sind Beispielabfragen aufgeführt, die auf den Geräteschatten für `AnyThing` und `AnotherThing` basieren. Weitere Informationen finden Sie unter Beispiel-Objektabfragen.

Aktuelle Version von AWS IoT Device Tester für FreeRTOS

Angeforderte Informationen	Abfrage	Ergebnis
Objekte, auf denen eine bestimmte Paketversion installiert ist	`shadow.name.$package.reported.SamplePackage.version:1.0.0`	`AnyThing`, `OtherThing`
Objekte, auf denen keine bestimmte Paketversion installiert ist	`NOT shadow.name.$package.reported.OtherPackage.version:1.2.5`	`AnyThing`
Jedes Gerät, das eine Paketversion verwendet, deren Paket-ID größer als 1500 ist	`shadow.name.$package.reported.*.attributes.packageID>1500"`	`OtherThing`
Objekte, auf denen ein bestimmtes Paket installiert ist und auf denen mehr als ein Paket installiert ist	`shadow.name.$package.reported.SamplePackage.version:1.0.0 AND shadow.name.$package.reported.totalCount:2`	`OtherThing`

Erfassung der Paketversionsverteilung über Get **`BucketsAggregation`**

Zusätzlich zum Discovery-Bedienfeld in der AWS IoT Konsole können Sie mithilfe der `GetBucketsAggregation` API-Operation auch Informationen zur Verteilung der Paketversion abrufen. Um die Distributionsinformationen der Paketversion zu erhalten, gehen Sie wie folgt vor:

- Definieren Sie in der Flottenindizierung für jedes Softwarepaket ein benutzerdefiniertes Feld. Hinweis: Die Erstellung benutzerdefinierter Felder wird auf die AWS IoT Service Quotas für die Flottenindexierung angerechnet.
- Formatieren Sie das benutzerdefinierte Feld wie folgt:

  `shadow.name.$package.reported.<packageName>.version`

Weitere Informationen finden Sie im Abschnitt Benutzerdefinierte Felder in der AWS IoT Flottenindizierung.

Schalten Sie die &Snowball;-Appliance ein, indem Sie auf den Ein-/Aus-Schalter über dem E-Ink-Display drücken. AWS IoT Jobs

AWS IoT Device Management Der Softwarepaketkatalog erweitert AWS IoT Jobs um Substitutionsparameter und die Integration mit AWS IoT Flottenindizierung, dynamischen Dinggruppen und dem AWS IoT reservierten Objekt mit dem Namen Shadow.

> **Note**
>
> Um alle Funktionen des Softwarepaketkatalogs nutzen zu können, müssen Sie diese AWS Identity and Access Management (IAM-) Rollen und Richtlinien erstellen: AWS IoT Job-Rechte für die Bereitstellung von Paketversionen und AWS IoT Job-Rechte für die Aktualisierung des reservierten benannten Shadows. Weitere Informationen finden Sie unter Vorbereitung der Sicherheit

Substitutionsparameter für AWS IoT jobs

Sie können Substitutionsparameter als Platzhalter in Ihrem AWS IoT Jobdokument verwenden. Wenn der Auftragsservice auf einen Substitutionsparameter stößt, verweist er den Auftrag auf das Attribut einer benannten Softwareversion für den Parameterwert. Sie können diesen Prozess verwenden, um ein einzelnes Auftragsdokument zu erstellen und die Metadaten über allgemeine Attribute an den Auftrag zu übergeben. Sie könnten z. B. eine Amazon Simple Storage Service (Amazon S3) -URL, einen Amazon-Ressourcennamen (ARN) eines Softwarepakets (ARN) oder eine Signatur über Paketversionsattribute an das Auftragsdokument übergeben.

Die Substitutionsparameter sollten im Jobdokument wie folgt formatiert sein:

- Name und Version des Softwarepakets
  - Die leere Zeichenfolge dazwischen `package::version` steht für den Substitutionsparameter für den Softwarepaketnamen. Die leere Zeichenfolge dazwischen `version::attribute` steht für den Substitutionsparameter für die Softwarepaketversion. Sehen Sie sich das folgende Beispiel für die Verwendung des

Paketnamens und der Substitutionsparameter für die Paketversion in einem Jobdokument an:. `${aws:iot:package::version::attributes:<attributekey>}`

- Das Jobdokument füllt diese Substitutionsparameter automatisch aus, wobei der Versions-ARN aus den Paketversionsdetails verwendet wird. Wenn Sie mithilfe eines API- oder CLI-Befehls einen Job oder eine Auftragsvorlage für eine Bereitstellung mit einem einzelnen Paket erstellen, wird der Versions-ARN für eine Paketversion durch den `destinationPackageVersions` Parameter in und dargestellt. `CreateJob` `DescribeJob`

- Alle Attribute für eine Softwarepaketversion
  - Sehen Sie sich das folgende Beispiel für die Verwendung aller Attribute eines Substitutionsparameters für eine Softwarepaketversion in einem Jobdokument an: `${aws:iot:package:<packageName>:version:<versionName>:attributes}`

> **Note**
>
> Der Paketname, die Paketversion und alle Substitutionsparameter für Attribute können zusammen verwendet werden. Sehen Sie sich das folgende Beispiel für die Verwendung aller drei Substitutionsparameter in einem Jobdokument an: `${aws:iot:package::version::attributes}`

Im folgenden Beispiel gibt es ein Softwarepaket mit dem Namen `samplePackage` und einer Paketversion mit `2.1.5` den folgenden Attributen:

- Name: `s3URL`, Wert: `https://EXAMPIEBUCKET.s3.us-west-2.amazonaws.com/exampleCodeFile`
  - Dieses Attribut identifiziert den Speicherort der Codedatei, die in Amazon S3 gespeichert ist.
- Name: `signature`, Wert: `aaaaabbbbbccccddddeeeeeffffgggggghhhhhiiiiijjjj`
  - Dieses Attribut stellt einen Wert für die Codesignatur bereit, den das Gerät als Sicherheitsmaßnahme benötigt. Weitere Informationen finden Sie unter Codesignatur für Aufträge. Hinweis: Dieses Attribut ist ein Beispiel und nicht als Teil des Softwarepaket-Katalogs oder von Aufträgen erforderlich.

Für `s3URL` wird der Auftragsdokumentparameter wie folgt geschrieben:

```
{
```

```
"samplePackage": "${aws:iot:package:samplePackage1:version:2.1.5:attributes:s3URL}"
}
```

Für `signature` wird der Auftragsdokumentparameter wie folgt geschrieben:

```
{
"samplePackage": "${aws:iot:package:samplePackage1:version:2.1.5:attributes:signature}"
}
```

Das vollständige Auftragsdokument ist wie folgt geschrieben:

```
{
  ...
  "Steps": {
    "uninstall": ["samplePackage"],
    "download": [
      {
        "samplePackage":
 "${aws:iot:package:samplePackage1:version:2.1.5:attributes:s3URL}"
      },
    ],
    "signature": [
      "samplePackage" :
 "${aws:iot:package:samplePackage1:version:2.1.5:attributes:signature}"
    ]
  }
}
```

Nachdem die Ersetzung vorgenommen wurde, wird das folgende Auftragsdokument auf den Geräten bereitgestellt:

```
{
  ...
  "Steps": {
    "uninstall": ["samplePackage"],
    "download": [
      {
        "samplePackage": "https://EXAMPIEBUCKET.s3.us-west-2.amazonaws.com/
exampleCodeFile"
      },
    ],
    "signature": [
```

```
      "samplePackage" : "aaaaabbbbbcccccdddddeeeeeffffffgggggghhhhhiiiiijjjj"
    ]
  }
}
```

Substitutionsparameter (Vorher- und Nachher-Ansicht)

Substitutionsparameter vereinfachen die Erstellung eines Jobdokuments mithilfe verschiedener Flags, z. B. `$default` für die Standard-Paketversion. Dadurch entfällt die Notwendigkeit, spezifische Paketversions-Metadaten für jede Auftragsbereitstellung manuell einzugeben, da diese Flags automatisch mit den Metadaten gefüllt werden, auf die in der jeweiligen Paketversion verwiesen wird. Weitere Informationen zu Paketversionsattributen, z. B. `$default` für die Standard-Paketversion, finden Sie unter. Vorbereitung des Auftragsdokuments und der Paketversion für die Bereitstellung

Klicken Sie während einer Auftragsbereitstellung für eine Paketversion im Fenster des Editors für die Bereitstellungsanweisungsdatei auf die Schaltfläche „Vorschau der Substitution“, um das Auftragsdokument mit und ohne die Substitutionsparameter anzuzeigen. AWS-Managementkonsole

Mithilfe des Parameters „before-substitution“ in den `GetJobDocument` APIs `DescribeJob` und können Sie die API-Antwort vor und nach dem Entfernen der Substitutionsparameter anzeigen. Sehen Sie sich die folgenden Beispiele mit den APIs und an: `DescribeJob` `GetJobDocument`

- `DescribeJob`
  - Standardansicht

    ```
    {
        "jobId": "<jobId>",
        "description": "<description>",
        "destinationPackageVersions": ["arn:aws:iot:us-west-2:123456789012:package/
    TestPackage/version/1.0.2"]
    }
    ```

  - Ansicht vor der Substitution

    ```
    {
        "jobId": "<jobId>",
        "description": "<description>",
        "destinationPackageVersions": ["arn:aws:iot:us-west-2:123456789012:package/
    TestPackage/version/$default"]
    }
    ```

- `GetJobDocument`
  - Standardansicht

```
{
    "attributes": {
        "location": "prod-artifacts.s3.us-east-1.amazonaws.com/mqtt-core",
        "signature": "IQoJb3JpZ2luX2VjEIrwEaCXVzLWVhc3QtMSJHMEUCIAofPNPpZ9cI",
        "streamName": "mqtt-core",
        "fileId": "0"
    },
}
```

  - Ansicht vor der Substitution

```
{
    "attributes": "${aws:iot:package:TestPackage:version:$default:attributes}",
}
```

Weitere Informationen über AWS IoT Jobs, das Erstellen von Jobdokumenten und das Bereitstellen von Jobs finden Sie unter Jobs.

Vorbereitung des Auftragsdokuments und der Paketversion für die Bereitstellung

Wenn eine Paketversion erstellt wird, befindet sie sich in einem `draft` Zustand, der darauf hinweist, dass sie für die Bereitstellung vorbereitet wird. Um die Paketversion für die Bereitstellung vorzubereiten, müssen Sie ein Auftragsdokument erstellen, das Dokument an einem Ort speichern, auf den der Job zugreifen kann (z. B. Amazon S3), und bestätigen, dass die Paketversion die Attributwerte enthält, die das Jobdokument verwenden soll. (Hinweis: Sie können Attribute für eine Paketversion nur aktualisieren, solange sie sich im aktuellen `draft` Status befindet.)

Wenn Sie einen AWS IoT Job oder eine Job-Vorlage für eine Einzelpaket-Bereitstellung erstellen, haben Sie die folgenden Optionen, um Ihr Jobdokument anzupassen:

Datei mit Anweisungen zur Bereitstellung () **`recipe`**

- Die Bereitstellungsanweisungsdatei für eine Paketversion enthält die Bereitstellungsanweisungen, einschließlich eines Inline-Jobdokuments, für die Bereitstellung einer Paketversion auf mehreren

Geräten. Die Datei ordnet einer Paketversion spezifische Bereitstellungsanweisungen zu, um eine schnelle und effiziente Auftragsbereitstellung zu ermöglichen.

In der AWS-Managementkonsole können Sie die Datei im Vorschaufenster mit den Bereitstellungsanweisungen auf der Registerkarte „Konfigurationen für die Versionsbereitstellung“ des Workflows „Neues Paket erstellen“ erstellen. Sie können die automatische Generierung einer Anweisungsdatei aus den Attributen Ihrer Paketversion nutzen AWS IoT , indem Sie die Option Von AWS IoT empfohlener Datei starten verwenden oder Ihr vorhandenes Auftragsdokument verwenden, das in einem Amazon S3-Bucket gespeichert ist, indem Sie Ihre eigene Bereitstellungsanweisungsdatei verwenden verwenden.

> **Note**
>
> Wenn Sie Ihr eigenes Auftragsdokument verwenden, können Sie es direkt im Vorschaufenster der Bereitstellungsanweisungsdatei aktualisieren. Ihr ursprüngliches Auftragsdokument, das in Ihrem Amazon S3-Bucket gespeichert ist, wird jedoch nicht automatisch aktualisiert.

Wenn Sie AWS CLI oder verwenden, `recipe` steht ein API-Befehl wie `CreatePackageVersion` `GetPackageVersionUpdatePackageVersion`, oder für die Bereitstellungsanweisungsdatei, die ein integriertes Jobdokument enthält.

Weitere Informationen darüber, was ein Jobdokument ist, finden Sie unterGrundkonzepte.

Im folgenden Beispiel finden Sie die Datei mit den Bereitstellungsanweisungen, wie sie dargestellt wird durch`recipe`:

```
{
    "packageName": "sample-package-name",
    "versionName": "sample-package-version",
    ...
    "recipe": "{...}"
}
```

> **Note**
>
> Die Bereitstellungsanweisungsdatei, wie sie durch dargestellt wird, `recipe` kann aktualisiert werden, wenn sich eine Paketversion im `published` Statusstatus befindet, da

sie von den Metadaten der Paketversion getrennt ist. Sie wird während des Einsatzes am Arbeitsplatz unveränderlich.

`Artifact`Versionsattribut

- Mithilfe des Versionsattributs `artifact` in Ihrer Softwarepaketversion können Sie den Amazon S3-Speicherort für Ihre Paketversionsartefakte hinzufügen. Wenn mithilfe von AWS IoT Jobs eine Auftragsbereitstellung für Ihre Paketversion ausgelöst wird, wird der vorsignierte URL-Platzhalter `${aws:iot:package:<packageName>:version:<versionName>:artifact-location:s3-presigned-url}` im Jobdokument mithilfe des Amazon S3-Buckets, des Bucket-Schlüssels und der Version der im Amazon S3-Bucket gespeicherten Datei aktualisiert. Der Amazon-S3-Bucket, in dem die Paketversionsartefakte gespeichert sind, muss sich in derselben Region befinden, in der die Paketversion erstellt wurde.

  > **Note**
  >
  > Um mehrere Objektversionen derselben Datei in Ihrem Amazon S3-Bucket zu speichern, müssen Sie die Versionierung in Ihrem Bucket aktivieren. Weitere Informationen finden Sie unter Versionierung für Buckets aktivieren.

  Um auf die Artefakte der Paketversion im Amazon S3-Bucket zuzugreifen, wenn Sie den `CreatePackageVersion` oder `UpdatePackageVersion` API-Vorgang verwenden, benötigen Sie die folgenden Berechtigungen:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObjectVersion",
            "Resource": "arn:aws:s3:::bucket-name/key-name"
        }
    ]
}
```

Weitere Informationen zum Versionsattribut `artifact` in den `CreatePackageVersion` und `UpdatePackageVersion` -API-Vorgängen finden Sie unter CreatePackageVersion und UpdatePackageVersion.

Sehen Sie sich das folgende Beispiel an, das das Versionsattribut zeigt, das den Speicherort des Artefakts in Amazon S3 `artifact` unterstützt, wenn Sie eine neue Paketversion erstellen:

```
{
    "packageName": "sample package name",
    "versionName": "1.0",
    "artifact": {
        "s3Location": {
            "bucket": "firmware",
            "key": "image.bin",
            "version": "12345"
        }
    }
}
```

> **Note**
>
> Wenn eine Paketversion von einem `draft` Statusstatus in einen Statusstatus wechselt, werden die Attribute der Paketversion und der Speicherort der Artefakte unveränderlich. `published` Um diese Informationen zu aktualisieren, müssen Sie eine neue Paketversion erstellen und diese Aktualisierungen durchführen, während Sie sich im Statusstatus befinden. `draft`

Paket-Version

- Eine Standardversion des Softwarepakets kann in den verfügbaren Versionen des Softwarepakets angegeben werden, die eine sichere und stabile Paketversion bieten. Dies dient als Basisversion des Softwarepakets, wenn Sie mithilfe AWS IoT von Jobs die Standardpaketversion für Ihre Geräteflotte bereitstellen. Wenn Sie einen Auftrag zur Bereitstellung der `$default` Paketversion für ein Softwarepaket erstellen, müssen die Paketversion im Auftragsdokument und in der neuen Auftragsbereitstellung mit der Version übereinstimmen`$default`. Die Paketversion in der Auftragsbereitstellung wird durch API `destinationPackageVersions` - und CLI-Befehle und

`VersionARN` in der dargestellt AWS-Managementkonsole. Die Paketversion im Jobdokument wird durch den folgenden Platzhalter für das Jobdokument dargestellt, der unten abgebildet ist:

```
arn:aws:iot:<regionCode>:111122223333:package/<packageName>/version/$default
```

Um einen Job oder eine Jobvorlage mit der Standardpaketversion zu erstellen, verwenden Sie das `$default` Flag im Befehl `CreateJob` or `CreateJobTemplate` API, wie unten dargestellt:

```
"$ aws iot create-job \
    --destination-package-versions "arn:aws:iot:us-west-2:123456789012:package/
TestPackage/version/$default"
    --document file://jobdoc.json
```

> **Note**
>
> Das `$default` Paketversionsattribut, das auf die Standardversion verweist, ist ein optionales Attribut, das nur erforderlich ist, wenn für eine Auftragsbereitstellung über AWS IoT Jobs auf die Standardpaketversion verwiesen wird.

Wenn Sie mit der Paketversion zufrieden sind, veröffentlichen Sie sie entweder über die Seite mit den Softwarepaketdetails in der AWS IoT Konsole oder indem Sie den UpdatePackageVersion API-Vorgang ausführen. Sie können dann auf die Paketversion verweisen, wenn Sie den Job entweder über die AWS IoT Konsole erstellen oder indem Sie den CreateJob API-Vorgang ausführen.

Benennen der Pakete und Versionen bei der Bereitstellung

Um eine Softwarepaketversion auf einem Gerät bereitzustellen, stellen Sie sicher, dass das Softwarepaket und die Paketversion, auf die im Jobdokument verwiesen wird, mit dem Softwarepaket und der Paketversion übereinstimmen, die im `destinationPackageVersions` Parameter im `CreateJob` API-Vorgang angegeben sind. Wenn sie nicht übereinstimmen, erhalten Sie eine Fehlermeldung, in der Sie aufgefordert werden, beide Referenzen abzugleichen. Weitere Informationen zu Fehlermeldungen im Softwarepaketkatalog finden Sie unterAllgemeine Fehlermeldungen zur Fehlerbehebung.

Zusätzlich zu den Softwarepaketen und Paketversionen, auf die im Jobdokument verwiesen wird, können Sie zusätzliche Softwarepakete und Paketversionen in den `destinationPackageVersions` Parameter des `CreateJob` API-Vorgangs aufnehmen, auf die

im Jobdokument nicht verwiesen wird. Stellen Sie sicher, dass das Jobdokument die erforderlichen Installationsinformationen enthält, damit die Geräte die zusätzlichen Softwarepaketversionen ordnungsgemäß installieren können. Weitere Informationen zum `CreateJob` API-Betrieb finden Sie unter CreateJob.

Zielgruppenzuweisung für Jobs durch AWS IoT Dynamische Objektgruppen

Der Softwarepaketkatalog arbeitet mit Flottenindizierung, AWS IoT Aufträgen und AWS IoT dynamischen Objektgruppen, um Geräte innerhalb Ihrer Flotte zu filtern und gezielt auszuwählen, um auszuwählen, welche Paketversion auf Ihren Geräten bereitgestellt werden soll. Sie können eine Flottenindizierungsabfrage auf der Grundlage der aktuellen Paketinformationen Ihres Geräts ausführen und diese Dinge für einen AWS IoT Auftrag als Targeting auswählen. Sie können auch Softwareupdates veröffentlichen, jedoch nur für geeignete Zielgeräte. Sie können beispielsweise angeben, dass Sie eine Konfiguration nur für die Geräte bereitstellen möchten, auf denen derzeit die `iot-device-client 1.5.09` ausgeführt wird. Weitere Informationen finden Sie unter Erstellen einer dynamischen Objektgruppe.

Reservierte benannte Schatten- und Paketversionen

Falls konfiguriert, können AWS IoT Jobs den reservierten Namen shadow (`$package`) einer Sache aktualisieren, wenn der Job erfolgreich abgeschlossen wurde. In diesem Fall müssen Sie dem reservierten benannten Schatten eines Objekts keine Paketversion manuell zuordnen.

In den folgenden Situationen können Sie sich dafür entscheiden, eine Paketversion manuell dem reservierten benannten Schatten des Objekts zuzuordnen oder zu aktualisieren:

- Sie registrieren ein Ding unter, AWS IoT Core ohne die installierte Paketversion zuzuordnen.
- AWS IoT Jobs ist nicht so konfiguriert, dass es den reservierten Namen Shadow des Dings aktualisiert.
- Sie verwenden einen internen Prozess, um Paketversionen an Ihre Flotte zu versenden, und dieser Prozess wird nicht aktualisiert, AWS IoT Core wenn er abgeschlossen ist.

> **Note**
>
> Wir empfehlen Ihnen, AWS IoT Jobs zu verwenden, um die Paketversion im reservierten Shadow (`$package`) zu aktualisieren. Das Aktualisieren des Versionsparameters im `$package` Shadow durch andere Prozesse (z. B. manuelle oder programmatische API-

Aufrufe), wenn AWS IoT Jobs ebenfalls für die Aktualisierung des Shadows konfiguriert ist, kann zu Inkonsistenzen zwischen der tatsächlichen Version auf dem Gerät und der Version führen, die an den reservierten benannten Shadow gemeldet wurde.

Sie können eine Paketversion zum reservierten Schatten (`$package`) eines Objekts über die Konsole oder den `UpdateThingShadow`-API-Vorgang hinzufügen oder aktualisieren. Weitere Informationen finden Sie unter Einer Sache eine Paketversion zuordnen. AWS IoT

> **Note**
>
> Durch das Zuordnen einer Paketversion zu einer AWS IoT Sache wird die Gerätesoftware nicht direkt aktualisiert. Sie müssen die Paketversion auf dem Gerät bereitstellen, um die Gerätesoftware zu aktualisieren.

Deinstallation eines Softwarepakets und seiner Paketversion

`$null`ist ein reservierter Platzhalter, der den AWS IoT Jobs-Dienst auffordert, das vorhandene Softwarepaket und die Paketversion aus dem reservierten benannten Shadow des Geräts zu entfernen. `$package` Weitere Informationen finden Sie unter Reservierter benannter Schatten.

Um diese Funktion zu verwenden, ersetzen Sie den Versionsnamen am Ende des PackageVersion Amazon-Ressourcennamens (ARN) des Ziels durch`$null`. Anschließend müssen Sie Ihren Service anweisen, die Software vom Gerät zu entfernen.

Der autorisierte ARN verwendet das folgende Format:

```
arn:aws:iot:<regionCode>:111122223333:package/<packageName>/version/$null
```

Zum Beispiel

```
$ aws iot create-job \
    ... \
    --destinationPackageVersions ["arn:aws:iot:us-east-1:111122223333:package/
samplePackage/version/$null"]
```

Erste Schritte mit dem Softwarepaket-Katalog

Sie können den AWS IoT Device Management Softwarepaketkatalog über die AWS-Managementkonsole AWS IoT Core API-Operationen und AWS Command Line Interface (AWS CLI) erstellen und verwalten.

> **Note**
>
> Die Aktivierung der AWS IoT Flottenindizierung ist eine Voraussetzung für die Verwendung des Softwarepaketkatalogs. Grundlegende Operationen wie das Erstellen einer Softwarepaketversion im AWS-Managementkonsole und das Verwenden des `CreatePackage` API-Befehls schlagen fehl, ohne dass die AWS IoT Flottenindizierung aktiviert ist.
> Weitere Informationen zur Verwendung der AWS IoT Flottenindizierung mit dem Softwarepaketkatalog finden Sie unter. Vorbereitung der Flottenindizierung

Verwenden der Konsole

Um den zu verwenden AWS-Managementkonsole, melden Sie sich in Ihrem AWS Konto an und navigieren Sie zu AWS IoT Core. Wählen Sie im Navigationsbereich Softwarepakete aus. In diesem Abschnitt können Sie dann Pakete und deren Versionen erstellen und verwalten.

Verwenden von API- oder CLI-Operationen

Sie können die AWS IoT Core API-Operationen verwenden, um Funktionen des Softwarepaketkatalogs zu erstellen und zu verwalten. Weitere Informationen finden Sie unter AWS IoT API-Referenz und AWS SDKs und Toolkits. Mit den AWS CLI Befehlen wird auch Ihr Katalog verwaltet. Weitere Informationen finden Sie in der AWS IoT CLI Befehlsreferenz.

Dieses Kapitel enthält die folgenden Abschnitte:

- Ein Softwarepaket und eine Paketversion erstellen
- Bereitstellung einer Paketversion über AWS IoT jobs
- Zuordnen einer Paketversion zu einer AWS IoT Objekt

Ein Softwarepaket und eine Paketversion erstellen

Sie können die folgenden Schritte verwenden, um ein Paket und eine erste Version über die AWS-Managementkonsole zu erstellen.

So erstellen Sie ein Softwarepaket

1. Loggen Sie sich in Ihr AWS Konto ein und navigieren Sie zur AWS IoT Konsole.
2. Wählen Sie im Navigationsbereich Softwarepakete aus.
3. Wählen Sie auf der Seite AWS IoT Softwarepaket die Option Paket erstellen aus. Das Dialogfeld Abhängigkeiten für die Paketverwaltung aktivieren wird angezeigt.
4. Wählen Sie unter Flottenindizierung die Option Gerätesoftwarepakete und Version hinzufügen aus. Dies ist für den Softwarepaket-Katalog erforderlich und bietet Flottenindizierung und Kennzahlen zu Ihrer Flotte.
5. [Optional] Wenn AWS IoT Jobs den reservierten benannten Shadow aktualisieren sollen, wenn Jobs erfolgreich abgeschlossen wurden, wählen Sie Shadows aus Jobs automatisch aktualisieren. Wenn Sie nicht möchten, dass AWS IoT Jobs diese Aktualisierung durchführen, lassen Sie dieses Kontrollkästchen deaktiviert.
6. [Optional] Um AWS IoT Jobs die Rechte zum Aktualisieren des reservierten benannten Shadows zu gewähren, wählen Sie unter Rolle auswählen die Option Rolle erstellen aus. Wenn Sie nicht möchten, dass AWS IoT Jobs diese Aktualisierung vornehmen, ist diese Rolle nicht erforderlich.
7. Erstellen oder wählen Sie eine Rolle aus.

   a. Wenn Sie keine Rolle für diesen Zweck haben: Wenn das Dialogfeld Rolle erstellen angezeigt wird, geben Sie einen Rollennamen ein, und wählen Sie dann Erstellen aus.

   b. Falls Sie über eine Rolle für diesen Zweck verfügen: Wählen Sie unter Rolle auswählen Ihre Rolle aus und stellen Sie dann sicher, dass das Kontrollkästchen Richtlinie an IAM-Rolle anhängen aktiviert ist.
8. Wählen Sie Bestätigen aus. Die Seite Neues Paket erstellen wird angezeigt.
9. Geben Sie unter Paketdetails einen Paketnamen ein.
10. Geben Sie unter Paketbeschreibung Informationen ein, die Ihnen helfen, dieses Paket zu identifizieren und zu verwalten.
11. [Optional] Mit Tags können Sie dieses Paket besser kategorisieren und verwalten. Um Tags hinzuzufügen, erweitern Sie Tags, wählen Sie Tag hinzufügen und geben Sie ein Schlüssel-

Wert-Paar ein. Sie können bis zu 50 Tags eingeben. Weitere Informationen finden Sie unter Ihre AWS IoT Ressourcen https://docs.aws.amazon.com/iot/latest/developerguide/tagging-iot.html taggen.

Um beim Erstellen eines neuen Pakets eine Paketversion hinzuzufügen

1. Geben Sie unter Erste Version einen Versionsnamen ein.

   Wir empfehlen, das SemVer Format (z. B. `1.0.0.0`) zu verwenden, um Ihre Paketversion eindeutig zu identifizieren. Sie können auch eine andere Formatierungsstrategie verwenden, die besser zu Ihrem Anwendungsfall passt. Weitere Informationen finden Sie unter Lebenszyklus der Paketversion.
2. Geben Sie unter Versionsbeschreibung Informationen ein, anhand derer Sie diese Paketversion identifizieren und verwalten können.

   > **Note**
   >
   > Das Kontrollkästchen Standardversion ist deaktiviert, da Paketversionen in einem bestimmten `draft` Status erstellt werden. Sie können die Standardversion benennen, nachdem Sie die Paketversion erstellt haben und wenn Sie den Status in ändern`published`. Weitere Informationen finden Sie unter Lebenszyklus der Paketversion.
3. [Optional] Um Ihnen bei der Verwaltung dieser Version zu helfen oder Informationen an Ihre Geräte zu übermitteln, geben Sie ein oder mehrere Name-Wert-Paare für Versionsattribute ein. Wählen Sie für jedes Name-Wert-Paar, das Sie eingeben, die Option Attribut hinzufügen aus. Weitere Informationen finden Sie unter Versionsattribute.
4. [Optional] Mit Tags können Sie dieses Paket besser kategorisieren und verwalten. Um Tags hinzuzufügen, erweitern Sie Tags, wählen Sie Tag hinzufügen und geben Sie ein Schlüssel-Wert-Paar ein. Sie können bis zu 50 Tags eingeben. Weitere Informationen finden Sie unter Ihre AWS IoT Ressourcen https://docs.aws.amazon.com/iot/latest/developerguide/tagging-iot.html taggen.
5. Wählen Sie Weiter aus.

Ordnen Sie die Software-Stückliste einer Paketversion zu (optional)

1. Wählen Sie in Schritt 3: Versions-SBOMs (optional) im Fenster „SBOM-Konfigurationen“ das Standard-SBOM-Dateiformat und den Validierungsmodus aus, mit denen Ihre Software-Stückliste validiert wird, bevor sie Ihrer Paketversion zugeordnet wird.
2. Geben Sie im Fenster SBOM-Datei hinzufügen den Amazon-Ressourcennamen (ARN) ein, der Ihren versionierten Amazon S3-Bucket darstellt, und das bevorzugte SBOM-Dateiformat, falls der Standardtyp nicht funktioniert.

   > **Note**
   >
   > Sie können entweder eine einzelne SBOM-Datei oder eine einzelne ZIP-Datei mit mehreren SBOMs hinzufügen, wenn Sie mehr als eine Software-Stückliste für Ihre Paketversion haben.

3. Im Fenster „SBOM-Datei hinzugefügt“ können Sie die SBOM-Datei sehen, die Sie für Ihre Paketversion hinzugefügt haben.
4. Wählen Sie Paket und Version erstellen. Die Seite mit der Paketversion wird angezeigt, und Sie können den Validierungsstatus Ihrer SBOM-Datei im Fenster „SBOM-Datei hinzugefügt“ sehen. Der Anfangsstatus ist, wenn die SBOM-Datei einer `In progress` Validierung unterzogen wird.

   > **Note**
   >
   > Der Validierungsstatus der SBOM-Datei lautet`Invalid file`,,, `Not started`In `progress`, `Validated (SPDX)` und die Gründe für das Fehlschlagen der Überprüfung. `Validated (CycloneDX)`

Bereitstellung einer Paketversion über AWS IoT jobs

Sie können die folgenden Schritte ausführen, um eine Paketversion über den AWS-Managementkonsole bereitzustellen.

Voraussetzungen:

Bevor Sie beginnen, führen Sie die folgenden Schritte aus:

- AWS IoT Dinge registrieren bei AWS IoT Core. Eine Anleitung zum Hinzufügen Ihrer Geräte zu AWS IoT Core finden Sie unter Ein Ding-Objekt erstellen.

- [Optional] Erstellen Sie eine AWS IoT Dinggruppe oder dynamische Dinggruppe als Ziel für die Geräte, auf denen Sie die Paketversion bereitstellen werden. Anweisungen zum Erstellen einer Objektgruppe finden Sie unter Erstellen einer statischen Objektgruppe. Anweisungen zum Erstellen einer dynamischen Objektgruppe finden Sie unter Erstellen einer dynamischen Objektgruppe.
- Erstellen Sie ein Softwarepaket und eine Paketversion. Weitere Informationen finden Sie unter Ein Softwarepaket und eine Paketversion erstellen.
- Erstellen eines Auftragsdokuments. Weitere Informationen finden Sie unter Vorbereiten des Auftragsdokuments und der Paketversion für die Bereitstellung.

Um eine bereitzustellen AWS IoT Auftrag

1. Wählen Sie auf der AWS IoT Konsole Softwarepakete aus.
2. Wählen Sie das Softwarepaket aus, das Sie bereitstellen möchten. Die Seite mit den Details zum Softwarepaket wird angezeigt.
3. Wählen Sie unter Versionen die Paketversion aus, die Sie bereitstellen möchten, und wählen Sie dann Auftragsversion bereitstellen aus.
4. Wenn Sie zum ersten Mal einen Auftrag über dieses Portal bereitstellen, wird ein Dialogfeld mit einer Beschreibung der Anforderungen angezeigt. Prüfen Sie die Informationen und wählen Sie Bestätigen aus.
5. Geben Sie einen Namen für die Bereitstellung ein, oder lassen Sie den automatisch generierten Namen im Feld Name stehen.
6. [Optional] Geben Sie im Feld Beschreibung eine Beschreibung ein, die den Zweck oder den Inhalt der Bereitstellung identifiziert, oder lassen Sie die automatisch generierten Informationen übrig.

   Hinweis: Wir empfehlen, dass Sie in den Feldern Auftragsname und Beschreibung keine personenbezogenen Daten verwenden.
7. [Optional] Fügen Sie alle Tags hinzu, die mit diesem Auftrag verknüpft werden sollen.
8. Wählen Sie Weiter aus.
9. Wählen Sie unter Auftragsziele die Objekte oder Objektgruppen aus, die den Auftrag erhalten sollen.
10. Geben Sie im Feld Auftragsdatei die JSON-Datei des Auftragsdokuments an.
11. Öffnen Sie Auftragsintegration mit dem Paketkatalogdienst.
12. Wählen Sie die Pakete und Versionen aus, die in Ihrem Auftragsdokument angegeben sind.

> **Note**
>
> Sie müssen dieselben Pakete und Paketversionen auswählen, die im Auftragsdokument angegeben sind. Sie können mehr hinzufügen, aber der Auftrag gibt Anweisungen nur für die Pakete und Versionen aus, die im Auftragsdokument enthalten sind. Weitere Informationen finden Sie unter Benennen der Pakete und Versionen bei der Bereitstellung.

13. Wählen Sie Weiter aus.
14. Wählen Sie auf der Seite Auftragskonfiguration im Dialogfeld Auftragskonfiguration einen der folgenden Auftragstypen aus:
    - Snapshot-Auftrag: Ein Snapshot-Auftrag ist abgeschlossen, wenn er abgeschlossen ist. Er wird auf den Zielgeräten und -gruppen ausgeführt.
    - Kontinuierlicher Auftrag: Ein kontinuierlicher Auftrag gilt für Objektgruppen und wird auf jedem Gerät ausgeführt, das Sie später zu einer bestimmten Zielgruppe hinzufügen.
15. Überprüfen Sie im Dialogfeld Zusätzliche Konfigurationen — optional die folgenden optionalen Auftragskonfigurationen und treffen Sie Ihre Auswahl entsprechend. Weitere Informationen finden Sie unter Konfigurationen für Auftrags-Rollout, Planung und Abbruch sowie Konfigurationen für Timeout und Wiederholungsversuche bei der Auftragsausführung.
    - Konfiguration des Rollouts
    - Konfiguration des Zeitplans
    - Konfiguration des Timeouts für Auftragsausführungen
    - Auftragsausführungen: Konfiguration wiederholen
    - Abbruch der Konfiguration
16. Überprüfen Sie die Auftragsauswahl und klicken Sie dann auf Absenden.

Nachdem Sie den Auftrag erstellt haben, generiert die Konsole eine JSON-Signatur und setzt sie in Ihr Auftragsdokument. Sie können die AWS IoT Konsole verwenden, um den Status eines Jobs anzuzeigen oder einen Job abzubrechen oder zu löschen. Gehen Sie zum Auftrags-Hub der Konsole, um Aufträge zu verwalten.

Zuordnen einer Paketversion zu einer AWS IoT Objekt

Nachdem Sie Software auf Ihrem Gerät installiert haben, können Sie einer reservierten AWS IoT Sache mit dem Namen Shadow eine Paketversion zuordnen. Wenn AWS IoT Jobs so konfiguriert wurden, dass der reservierte benannte Shadow des Dings aktualisiert wird, nachdem der Job bereitgestellt und erfolgreich abgeschlossen wurde, müssen Sie dieses Verfahren nicht ausführen. Weitere Informationen finden Sie unter Reservierter benannter Schatten.

Voraussetzungen:

Bevor Sie beginnen, führen Sie die folgenden Schritte aus:

- Erstellen Sie ein AWS IoT Ding oder Dinge und richten Sie die Telemetrie ein. AWS IoT Core Weitere Informationen finden Sie unter Erste Schritte mit AWS IoT Core.
- Erstellen Sie ein Softwarepaket und eine Paketversion. Weitere Informationen finden Sie unter Ein Softwarepaket und eine Paketversion erstellen.
- Installieren Sie die Software der Paketversion auf dem Gerät.

> **Note**
>
> Durch das Zuordnen einer Paketversion zu AWS IoT einem Objekt wird keine Software auf dem physischen Gerät aktualisiert oder installiert. Die Paketversion muss auf dem Gerät bereitgestellt werden.

Um eine Paketversion einer zuzuordnen AWS IoT Objekt

1. Erweitern Sie im Navigationsbereich der AWS IoT -Konsole das Menü Alle Geräte und wählen Sie Objekte aus.
2. Identifizieren Sie das AWS IoT Ding, das Sie aktualisieren möchten, aus der Liste und wählen Sie den Namen des Dings aus, um die zugehörige Detailseite anzuzeigen.
3. Wählen Sie im Abschnitt Details die Option Pakete und Versionen aus.
4. Wählen Sie Zu Paket und Version hinzufügen.
5. Wählen Sie unter Gerätepaket auswählen das gewünschte Softwarepaket aus.
6. Wählen Sie unter Version auswählen die gewünschte Softwareversion aus.
7. Wählen Sie Gerätepaket hinzufügen.

Das Paket und die Version werden in der Liste Ausgewählte Pakete und Versionen angezeigt.

8. Wiederholen Sie diese Schritte für jedes Paket und jede Version, die Sie diesem Objekt zuordnen möchten.
9. Wenn Sie fertig sind, wählen Sie Paket- und Versionsdetails hinzufügen. Die Seite mit den Objektdetails wird geöffnet, und Sie können das neue Paket und die neue Version in der Liste sehen.

AWS IoT Jobs

Verwenden Sie AWS IoT Jobs, um eine Reihe von Fernoperationen zu definieren, die an ein oder mehrere verbundene Geräte gesendet und auf diesen ausgeführt werden können AWS IoT. Sie können beispielsweise einen Auftrag definieren, der eine Reihe von Geräten anweist, Anwendungen herunterzuladen und zu installieren, Firmware-Updates auszuführen, einen Neustart vorzunehmen, die Zertifikate zu rotieren oder Remote-Fehlerbehebungsvorgänge auszuführen.

Zugriff AWS IoT jobs

Sie können mit AWS IoT Jobs beginnen, indem Sie die Konsole oder die AWS IoT Core API verwenden.

Verwenden der Konsole

Melden Sie sich bei der AWS-Managementkonsole an und gehen Sie zur AWS IoT Konsole. Wählen Sie im Navigationsbereich Manage (Verwalten) und dann Jobs (Aufträge) aus. In diesem Bereich können Sie Aufträge erstellen und verwalten. Wenn Sie Auftragsvorlagen erstellen und verwalten möchten, wählen Sie im Navigationsbereich die Option Auftragsvorlagen aus. Weitere Informationen finden Sie unter Erstellen und verwalten Sie Jobs mithilfe der AWS-Managementkonsole.

Verwenden der API oder CLI

Sie können beginnen, indem Sie die AWS IoT Core API-Operationen verwenden. Weitere Informationen finden Sie unter AWS IoT -API-Referenz. Die AWS IoT Core API, auf der AWS IoT Jobs basiert, wird vom AWS SDK unterstützt. Weitere Informationen finden Sie unter AWS SDKs und Toolkits.

Sie können das verwenden AWS CLI , um Befehle zum Erstellen und Verwalten von Jobs und Jobvorlagen auszuführen. Weitere Informationen finden Sie in der AWS IoT CLI-Referenz.

AWS IoT Jobs, Regionen und Endpunkte

AWS IoT Jobs unterstützt API-Endpunkte auf Steuerungs- und Datenebene, die für Sie spezifisch sind. AWS-Region Die API-Endpunkte der Datenebene sind spezifisch für Ihr AWS-Konto und. AWS-Region Weitere Informationen zu den AWS IoT Jobs-Endpunkten finden Sie unter AWS IoT Device Management — Jobs-Datenendpunkte in der AWS Allgemeinen Referenz.

Was ist eine Fernsteuerung?

> **Note**
>
> Over-the-air (OTA) -Updates sind in den folgenden Regionen nicht verfügbar:
>
> - Europa (Spain)
> - Asien-Pazifik (Malaysia)

Ein Fernvorgang ist jede Aktualisierung oder Aktion, die Sie auf einem physischen Gerät, virtuellen Gerät oder Endpunkt ausführen können und die remote ausgeführt werden können, ohne dass die physische Anwesenheit eines Bedieners oder Technikers erforderlich ist. Die Fernsteuerung erfolgt mithilfe eines Over-the-Air (OTA)-Updates, sodass Ihre Geräte nicht physisch anwesend sein müssen. Wenn Sie Ihre Geräteflotte in verwalten, AWS Cloud können Sie Fernoperationen auf Ihren Geräten ausführen, wenn diese registriert sind AWS IoT Core.

AWS IoT Device Management Jobs bietet einen skalierbaren Ansatz für die Durchführung von Fernaktionen auf Ihren Geräten, bei denen Sie registriert sind AWS IoT Core. Ein Job wird in der erstellt AWS Cloud und mithilfe eines OTA-Updates über das MQTT- oder HTTP-Protokoll an alle Zielgeräte verteilt.

AWS IoT Device Management Jobs bieten Ihnen die Möglichkeit, Fernoperationen wie Werkseinstellungen, Geräteneustarts und Software-OTA-Updates auf sichere, skalierbare und kostengünstigere Weise durchzuführen.

Weitere Informationen dazu finden Sie unter AWS IoT Core. Was ist AWS IoT?

Weitere Informationen zu AWS IoT Device Management Jobs finden Sie unter Was ist AWS IoT Jobs?.

Vorteile der Verwendung AWS IoT Device Management Jobs für Fernoperationen

Wenn Sie AWS IoT Device Management Jobs für Ihre Fernoperationen verwenden, wird die Verwaltung Ihrer Geräteflotte optimiert. In der folgenden Liste sind einige der wichtigsten Vorteile aufgeführt, die sich aus der Verwendung von AWS IoT Device Management Jobs für die Ausführung Ihrer Fernoperationen ergeben:

- Nahtlose Integration mit anderen AWS-Services
  - AWS IoT Device Management Jobs ist eng mit den folgenden Mehrwerten AWS-Services und Funktionen integriert:
    - Amazon S3: Speichern Sie Ihre Anweisungen zur Remote-Steuerung in einem sicheren Amazon S3-Bucket, in dem Sie die Zugriffsberechtigungen für diese Inhalte kontrollieren. Die Verwendung eines Amazon S3-Buckets bietet eine skalierbare und langlebige Speicherlösung, die nativ in den AWS IoT Gerätemanagement-Softwarepaketkatalog integriert ist, sodass AWS IoT Device Management Jobs in Aktualisierungsanweisungen referenzieren und diese ersetzen kann. Weitere Informationen finden Sie unter Was ist Amazon S3?.
    - Amazon CloudWatch: Überwachen und protokollieren Sie den Status der Auftragsausführung für jedes Gerät sowie andere Geräteaktivitäten, um die allgemeine Auftragsleistung in Jobs nachzuverfolgen und zu analysieren. AWS IoT Device Management Weitere Informationen finden Sie unter Was ist Amazon CloudWatch? Überwachung von Auftragsprotokollen und Erfassung historischer Daten zur Fehlerbehebung. So funktioniert es mit Aufträgen.
    - AWS-IoT-Device-Shadow-Service: Sorgen Sie mithilfe von AWS IoT Device Management Jobs für eine digitale Darstellung Ihres Objekts mithilfe AWS IoT eines Geräteschattens, sodass der Status Ihres Geräts unabhängig von der Gerätekonnektivität für Anwendungen und andere Dienste verfügbar ist. Weitere Informationen finden Sie unter AWS-IoT-Device-Shadow-Service.
- Bewährte Methoden für die Gewährleistung der Sicherheit
  - Berechtigungskontrolle: Steuern Sie mithilfe von Amazon S3 die Zugriffsberechtigungen für Ihre Fernbedienungsanweisungen und legen Sie mithilfe von AWS IoT Richtlinien und IAM-Benutzerrollen fest, welche IAM-Benutzer Ihre Fernbedienungsanweisungen für Ihre Geräteflotte bereitstellen können.
    - Weitere Informationen zu AWS IoT Richtlinien finden Sie unter. Erstellen Sie ein AWS IoT policy
    - Weitere Informationen zu IAM-Benutzerrollen finden Sie unter Identitäts- und Zugriffsmanagement für AWS IoT.
- Skalierbarkeit
  - Gezielte Auftragsbereitstellung: Steuern Sie, welche Geräte das Auftragsdokument eines Auftrags mit einer gezielten Auftragsverteilung erhalten, indem Sie bei der Erstellung des Auftrags bestimmte Kriterien für die Gerätegruppierung in Ihrem Auftragsdokument eingeben. Wenn Sie für jedes Gerät AWS IoT etwas erstellen und diese Informationen in der AWS IoT Registrierung speichern, können Sie mithilfe der Flottenindizierung gezielte Suchen

durchführen. Sie können auf der Grundlage der Suchergebnisse zur Flottenindizierung benutzerdefinierte Gruppen erstellen, um die Bereitstellung Ihres Zielauftrags zu unterstützen. Weitere Informationen finden Sie unter Geräte verwalten mit AWS IoT. Verwenden Sie Aufträge, um Snapshot-Aufträge im Vergleich zu kontinuierlichen Aufträgen auszuführen.

  - Auftragsstatus: Verfolgen Sie den Status des Rollouts des Auftragsdokuments für Ihre Geräteflotte und den gesamten Auftragsstatus auf Geräteflottenebene sowie den individuellen Implementierungsstatus des Auftragsdokuments auf jedem Gerät. Weitere Informationen finden Sie unter Aufträge und Status der Auftragsausführung.
  - Skalierbarkeit neuer Geräte: Stellen Sie Ihr Auftragsdokument ganz einfach auf einem neuen Gerät bereit, indem Sie es einer vorhandenen, benutzerdefinierten Gruppe hinzufügen, die mithilfe der Flottenindizierung über einen kontinuierlichen Auftrag erstellt wurde. Dadurch sparen Sie Zeit, anstatt das Auftragsdokument auf jedem neuen Gerät separat bereitstellen zu müssen. Oder Sie können einen gezielteren Ansatz mit einem Snapshot verwenden, indem Sie ein Auftragsdokument einmal auf eine vorgegebene Gruppe von Geräten verteilen und dann der Job abgeschlossen ist.
- Flexibilität
  - Auftragskonfigurationen: Passen Sie Ihren Auftrag und Ihr Auftragsdokument mit den optionalen Auftragskonfigurationen Rollout, Terminplanung, Abbruch, Timeout und Wiederholung an Ihre spezifischen Anforderungen an. Weitere Informationen finden Sie unter Aufgabe configurations.
- Kosteneffektiv
  - Führen Sie eine effizientere Kostenstruktur für die Wartung Ihrer Geräteflotte ein, indem Sie AWS IoT Device Management Jobs nutzen, um wichtige Updates bereitzustellen und routinemäßige Wartungsaufgaben durchzuführen. Eine Do-it-Yourself-Lösung (DIY) zur Wartung Ihrer Geräteflotte beinhaltet wiederkehrende, variable Kosten wie die für das Hosten und Verwalten der DIY-Lösung erforderliche Infrastruktur, die Arbeitskosten für die Entwicklung, Wartung und Skalierung der DIY-Lösung sowie die Kosten für die Datenübertragung. Dank der transparenten, festen Kostenstruktur von AWS IoT Device Management Jobs wissen Sie genau, was jede Auftragsausführung für ein Gerät zusätzlich zu den Kosten für die Datenübertragung kostet, die erforderlich sind, um die Bereitstellung der Auftragsdokumente auf Ihrer Geräteflotte zu erleichtern und den Ausführungsstatus der Aufträge für jedes Gerät zu verfolgen. Weitere Informationen finden Sie unter AWS IoT Core Preise.

Was ist AWS IoT Jobs?

Verwenden Sie AWS IoT Jobs, um eine Reihe von Fernoperationen zu definieren, die an ein oder mehrere verbundene Geräte gesendet und auf diesen ausgeführt werden können AWS IoT.

Um Aufträge zu erstellen, definieren Sie zunächst ein Auftragsdokument, das eine Liste von Anweisungen enthält, in denen die Operationen beschrieben werden, die das Gerät aus der Ferne ausführen muss. Um diese Operationen auszuführen, geben Sie eine Liste von Zielen an, bei denen es sich um einzelne Objekte, Objektgruppen oder beides handelt. Das Arbeitsdokument und die Ziele bilden zusammen eine Bereitstellung.

Jede Bereitstellung kann zusätzliche Konfigurationen haben:

- Rollout: Definiert, wie viele Geräte das Auftragsdokument pro Minute erhalten.
- Abbrechen: Wenn eine bestimmte Anzahl von Geräten die Auftragsbenachrichtigung nicht erhält, verwenden Sie diese Konfiguration, um den Auftrag abzubrechen. Dadurch wird vermieden, dass ein fehlerhaftes Update an eine ganze Flotte gesendet wird.
- Timeout: Wenn innerhalb eines bestimmten Zeitraums keine Antwort von Ihren Auftragszielen eingeht, kann der Auftrag fehlschlagen. Sie können den Auftrag verfolgen, der auf diesen Geräten ausgeführt wird.
- Erneut versuchen: Wenn ein Gerät einen Ausfall meldet oder eine Zeitüberschreitung bei einem Job auftritt, können Sie AWS IoT Jobs verwenden, um das Auftragsdokument automatisch erneut an das Gerät zu senden.
- Planung: Mit dieser Konfiguration können Sie einen Auftrag für ein zukünftiges Datum und eine zukünftige Uhrzeit planen. Sie können damit auch wiederkehrende Wartungsfenster einrichten, in denen Geräte während vordefinierter Zeiträume mit geringem Datenverkehr aktualisiert werden.

AWS IoT Jobs sendet eine Nachricht, um die Ziele darüber zu informieren, dass ein Job verfügbar ist. Das Ziel startet die Ausführung des Jobs, indem es das Auftragsdokument herunterlädt, die von ihm angegebenen Operationen ausführt und seinen Fortschritt meldet AWS IoT. Sie können den Fortschritt eines Jobs für ein bestimmtes Ziel oder für alle Ziele verfolgen, indem Sie Befehle ausführen, die von AWS IoT Jobs bereitgestellt werden. Wenn ein Auftrag gestartet wird, hat er den Status In Bearbeitung. Die Geräte melden dann inkrementelle Aktualisierungen und zeigen diesen Status an, bis der Auftrag erfolgreich ist, fehlschlägt oder das Zeitlimit überschritten wird.

In den folgenden Themen werden einige wichtige Konzepte von Aufträgen und der Lebenszyklus von Aufträgen und Auftragsausführungen beschrieben.

Themen

- Wichtige Konzepte von Jobs
- Aufträge und Status der Auftragsausführung

Wichtige Konzepte von Jobs

Die folgenden Konzepte enthalten Einzelheiten zu AWS IoT Jobs und zur Erstellung und Bereitstellung von Aufträgen zur Ausführung von Remote-Vorgängen auf Ihren Geräten.

Grundkonzepte

Im Folgenden finden Sie grundlegende Konzepte, die Sie kennen müssen, wenn Sie AWS IoT Jobs verwenden.

Job

Ein Auftrag ist eine Remote-Operation, die an ein oder mehrere mit AWS IoT verbundene Geräte gesendet und dort ausgeführt werden. Sie können beispielsweise einen Auftrag definieren, der eine Reihe von Geräten anweist, eine Anwendung herunterzuladen und zu installieren oder Firmware-Updates auszuführen, einen Neustart vorzunehmen, die Zertifikate zu rotieren oder Remote-Fehlerbehebungsvorgänge auszuführen.

Auftragsdokument

Um einen Auftrag zu erstellen, müssen Sie zunächst ein Auftragsdokument erstellen, das eine Beschreibung der Remote-Operationen, die von den Geräten ausgeführt werden sollen.

Jobdokumente sind UTF-8 kodierte JSON-Dokumente und enthalten Informationen, die Ihre Geräte zur Ausführung eines Jobs benötigen. Ein Auftragsdokument enthält eine oder mehrere URLs, unter denen das Gerät ein Update oder andere Daten herunterladen kann. Das Auftragsdokument kann in einem Amazon S3-Bucket gespeichert sein oder inline in dem Befehl enthalten sein, der den Auftrag erstellt.

> **Note**
>
> In MQTT-Antworten ist das `jobDocument` Feld ein JSON-Objekt. In HTTP-Antworten ist es eine Zeichenkettendarstellung des JSON-Objekts.

Ziel

Beim Anlegen eines Auftrags geben Sie eine Liste von Zielen an, nämlich die Geräte, die die Operationen ausführen sollen. Bei den Zielen kann es sich um Objekte und/oder Objektgruppen handeln. Der AWS IoT Jobs-Dienst sendet eine Nachricht an jedes Ziel, um es darüber zu informieren, dass ein Job verfügbar ist.

Bereitstellung

Nachdem Sie einen Auftrag erstellt haben, indem Sie das Auftragsdokument bereitgestellt und Ihre Zielliste angegeben haben, wird das Auftragsdokument dann auf den Remote-Zielgeräten bereitgestellt, für die Sie das Update durchführen möchten. Bei Snapshot-Aufträgen wird der Auftrag nach der Bereitstellung auf den Zielgeräten abgeschlossen. Bei kontinuierlichen Aufträgen wird ein Auftrag für eine Gruppe von Geräten bereitgestellt, sobald diese den Gruppen hinzugefügt werden.

Auftragsausführung

Eine Auftragsausführung ist eine Instance eines Auftrags auf einem Zielgerät. Das Ziel startet eine Ausführung eines Auftrags durch Herunterladen des Auftragsdokuments. Anschließend führt er die im Dokument angegebenen Operationen aus und meldet seinen Fortschritt an AWS IoT. Eine Ausführungsnummer ist eine eindeutige Kennung einer Auftragsausführung auf einem bestimmten Ziel. Der AWS IoT Jobs-Dienst stellt Befehle zur Verfügung, um den Fortschritt einer Auftragsausführung auf einem Ziel und den Fortschritt eines Jobs auf allen Zielen zu verfolgen.

Auftragstypen, Konzepte

Die folgenden Konzepte können Ihnen helfen, mehr über die verschiedenen Arten von Aufträgen zu erfahren, die Sie mit AWS IoT Jobs erstellen können.

Snapshot-Auftrag

Standardmäßig wird ein Auftrag an alle Ziele gesendet, die Sie bei der Erstellung des Auftrags angeben. Wenn diese Ziele den Auftrag abgeschlossen haben (oder melden, dass sie dazu nicht in der Lage sind), ist der Auftrag abgeschlossen.

Kontinuierlicher Auftrag

Kontinuierliche Aufträge werden an alle Ziele gesendet, die Sie bei der Erstellung des Auftrags angeben. Er weiter ausgeführt und an alle neuen Geräten (Objekte) gesendet, die der Zielgruppe hinzugefügt werden. Beispiel: Ein kontinuierlicher Auftrag kann zum Onboarding und zur

Aktualisierung von Geräten, die einer Gruppe hinzugefügt werden, verwendet werden. Sie können einen Auftrag kontinuierlich machen, indem Sie bei der Erstellung des Auftrags einen optionalen Parameter setzen.

> **Note**
>
> Wenn Sie Ihre IoT-Flotte mit dynamischen Objektgruppen ins Visier nehmen, empfehlen wir, kontinuierliche Aufträge anstelle von Snapshot-Aufträgen zu verwenden. Durch die Verwendung kontinuierlicher Aufträge erhalten Geräte, die der Gruppe beitreten, die Auftragsausführung auch dann, wenn der Auftrag erstellt wurde.

Vorsignierte URLs

Für einen sicheren, zeitlich begrenzten Zugriff auf Daten, die nicht im Auftragsdokument enthalten sind, können Sie vorsignierte Amazon S3-URLs verwenden. Platzieren Sie Ihre Daten in einen Amazon S3-Bucket und fügen einen Platzhalterlink zu den Daten in dem Auftragsdokument hinzu. Wenn AWS IoT Jobs eine Anfrage für das Jobdokument erhält, analysiert es das Jobdokument, indem es nach den Platzhalter-Links sucht und die Links dann durch vorsignierte Amazon S3-URLs ersetzt.

Der Platzhalterlink hat das folgende Format:

```
${aws:iot:s3-presigned-url:https://s3.amazonaws.com/bucket/key}
```

wo *bucket* ist Ihr Bucket-Name und *key* ist das Objekt im Bucket, auf das Sie verlinken.

In den Regionen Peking und Ningxia funktionieren vorsignierte URLs nur, wenn der Eigentümer der Ressource über eine ICP-Lizenz (Internet Content Provider) verfügt. Weitere Informationen finden Sie unter Amazon Simple Storage Service in der Dokumentation Erste Schritte mit AWS Services in China.

Auftragskonfigurationskonzepte

Die folgenden Konzepte können Ihnen helfen, die Konfiguration von Aufträgen zu verstehen.

Rollouts

Sie können angeben, wie schnell die Ziele über eine ausstehende Auftragsausführung benachrichtigt werden. Auf diese Weise können Sie einen schrittweisen Rollout erstellen,

um Updates, Neustarts und andere Operationen besser zu verwalten. Sie können eine Rolloutkonfiguration erstellen, indem Sie entweder eine statische Rollout-Rate oder eine exponentielle Rollout-Rate verwenden. Verwenden Sie eine statische Rollout-Rate, um die maximale Anzahl von Auftragszielen festzulegen, die pro Minute informiert werden sollen.

Beispiele für die Festlegung von Rollout-Raten und weitere Informationen zur Konfiguration von Auftrags-Rollouts finden Sie unter. Konfigurationen für Auftrags-Rollout, Planung und Abbruch

Planung

Auftragsplanung ermöglicht es Ihnen, den Rollout-Zeitraum eines Auftragsdokuments auf allen Geräten der Zielgruppe für kontinuierliche Aufträge und Snapshot-Aufträge zu planen. Darüber hinaus können Sie ein optionales Wartungsfenster einrichten, das bestimmte Daten und Uhrzeiten enthält, zu denen das Auftragsdokument im Rahmen eines Auftrags auf allen Geräten in der Zielgruppe bereitgestellt wird. Ein Wartungsfenster ist eine wiederkehrende Instance mit einer Häufigkeit von täglichen, wöchentlichen, monatlichen oder benutzerdefinierten Daten und Uhrzeiten, die bei der ersten Erstellung des Auftrags oder der Auftragsvorlage ausgewählt wurden. Nur fortlaufende Aufträge können so geplant werden, dass sie während eines Wartungsfensters einen Rollout durchführen.

Die Planung von Aufträgen ist spezifisch für Ihren Auftrag. Einzelne Auftragsausführungen können nicht geplant werden. Weitere Informationen finden Sie unter Konfigurationen für Auftrags-Rollout, Planung und Abbruch.

Abbrechen

Sie können eine Reihe von Bedingungen für den Abbruch von Rollouts erstellen, wenn bestimmte von Ihnen angegebenen Kriterien erfüllt sind. Weitere Informationen finden Sie unter Konfigurationen für Auftrags-Rollout, Planung und Abbruch.

Timeouts

Auftrags-Timeouts benachrichtigen Sie, wenn eine Auftragsbereitstellung für eine unerwartet lange Zeit im Status `IN_PROGRESS` stecken bleibt. Es gibt zwei Arten von Timern: Timer für „In Bearbeitung“ und Timer für „Schritt“. Wenn der Auftrag `IN_PROGRESS` ist, können Sie den Fortschritt Ihrer Auftragsbereitstellung überwachen und verfolgen.

Rollouts und Abbruchkonfigurationen sind spezifisch für Ihren Auftrag, wohingegen die Timeout-Konfiguration spezifisch für eine Auftragsbereitstellung ist. Weitere Informationen finden Sie unter Timeout bei der Auftragsausführung und Wiederholungskonfigurationen.

Wiederholversuche

Auftragswiederholungen ermöglichen es, die Auftragsausführung erneut zu versuchen, wenn ein Auftrag fehlschlägt, eine Zeitüberschreitung eintritt oder beides auftritt. Sie können über bis zu 10 Wiederholungsversuche für die Auftragsausführung verfügen. Sie können den Fortschritt Ihres Wiederholungsversuchs überwachen und verfolgen, ob die Auftragsausführung erfolgreich war.

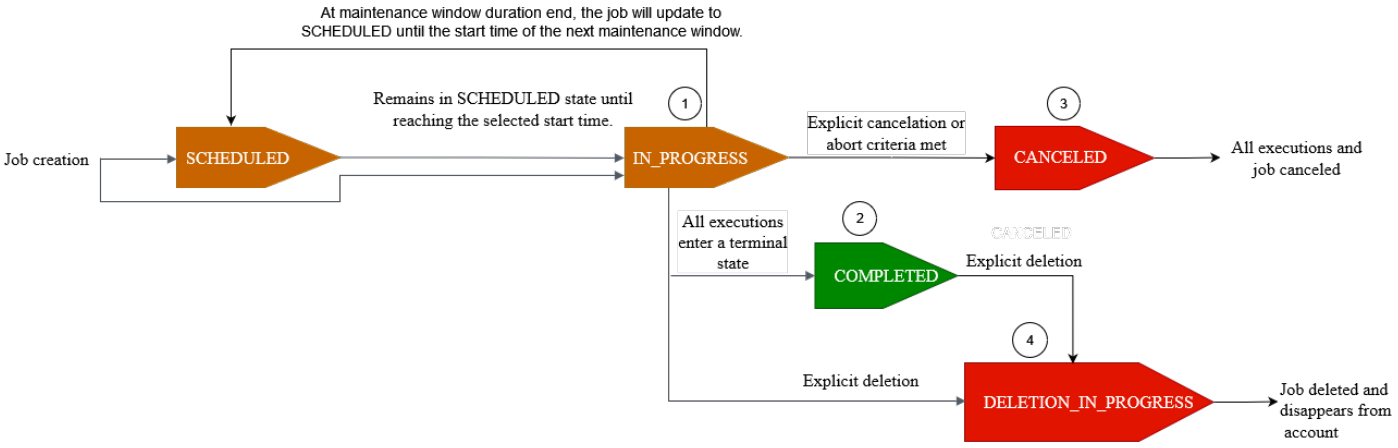
Rollouts und Abbruchkonfigurationen sind spezifisch für Ihren Auftrag, während die Timeout- und Wiederholungs-Konfigurationen spezifisch für eine Auftragsausführung sind. Weitere Informationen finden Sie unter Timeout bei der Auftragsausführung und Wiederholungskonfigurationen.

Aufträge und Status der Auftragsausführung

In den folgenden Abschnitten werden der Lebenszyklus eines AWS IoT Jobs und der Lebenszyklus einer Auftragsausführung beschrieben.

Status des Jobs

Das folgende Diagramm zeigt die verschiedenen Status eines AWS IoT Jobs.



Ein Job, den Sie mithilfe von AWS IoT Jobs erstellen, kann sich in einem der folgenden Zustände befinden:

- GEPLANT

  Bei der ersten Erstellung des Jobs oder der Jobvorlage mithilfe der AWS IoT Konsole, CreateJob der CreateJobTemplate API oder der API können Sie die optionale Planungskonfiguration in der AWS IoT Konsole oder `SchedulingConfig` in der CreateJob API oder CreateJobTemplate API

auswählen. Wenn Sie einen geplanten Auftrag starten, der ein bestimmtes `startTime`, `endTime` und `endBehavior` erhält, wird der Auftragsstatus aktualisiert auf `SCHEDULED`. Wenn der Auftrag das von Ihnen gewählte `startTime` oder `startTime` des nächsten Wartungsfensters erreicht (falls Sie den Auftrags-Rollout während eines Wartungsfensters ausgewählt haben), wird der Status von `SCHEDULED` zu `IN_PROGRESS` aktualisiert und mit dem Rollout des Auftragsdokuments auf allen Geräten in der Zielgruppe begonnen.

- IN_PROGRESS

  Wenn Sie einen Job mithilfe der AWS IoT Konsole oder der CreateJob API erstellen, wird der Auftragsstatus auf aktualisiert`IN_PROGRESS`. Während der Auftragserstellung beginnt AWS IoT Jobs mit der Bereitstellung von Auftragsausführungen auf den Geräten Ihrer Zielgruppe. Nachdem alle Auftragsausführungen eingeführt wurden, wartet AWS IoT Jobs darauf, dass die Geräte die Remote-Aktion abgeschlossen haben.

  Informationen zur Parallelität und zu den Beschränkungen, die für laufende Aufträge gelten, finden Sie unter AWS IoT Grenzwerte für Aufträge.

  > **Note**
  >
  > Wenn ein `IN_PROGRESS`-Auftrag das Ende des aktuellen Wartungsfensters erreicht, wird der Rollout des Auftragsdokuments beendet. Der Auftrag wird `SCHEDULED` bis zum `startTime` nächsten Wartungsfenster aktualisiert.

- COMPLETED

  Ein kontinuierlicher Auftrag wird auf eine der folgenden Arten behandelt:

  - Bei einem kontinuierlichen Auftrag, bei dem die optionale Planungskonfiguration nicht ausgewählt wurde, ist er immer in Bearbeitung und wird für alle neuen Geräte, die der Zielgruppe hinzugefügt werden, weiterhin ausgeführt. Es wird niemals den Status `COMPLETED` erreichen.
  - Für einen kontinuierlichen Auftrag mit der ausgewählten optionalen Planungskonfiguration gilt Folgendes:
    - Wenn ein `endTime` angegeben wurde, erreicht ein kontinuierlicher Auftrag den Status `COMPLETED`, wenn er `endTime` bestanden hat und alle Auftragsausführungen den Terminal-Zustand erreicht haben.
    - Wenn in der optionalen Planungskonfiguration kein bereitgestellt *`endTime`* wurde, führt der fortlaufende Auftrag weiterhin den Rollout des Auftragsdokuments durch.

Bei einem Snapshot-Auftrag ändert sich der Auftragsstatus in den `COMPLETED`-Zustand, in dem alle zugehörigen Auftragsausführungen in einen Terminal-Zustand übergehen, z. B. `SUCCEEDED`, `FAILED`, `TIMED_OUT`, `REMOVED` oder `CANCELED`.

- CANCELED

  Wenn Sie einen Job mithilfe der AWS IoT Konsole, der CancelJob API oder der stornierenKonfiguration des Auftragsabbruchs, ändert sich der Auftragsstatus in`CANCELED`. Während der Auftragsstornierung beginnt AWS IoT Jobs, zuvor erstellte Auftragsausführungen abzubrechen.

  Informationen zur Gleichzeitigkeit und zu den Grenzen, die für abgebrochene Aufträge gelten, finden Sie unter AWS IoT Grenzwerte für Aufträge.

- DELETION_IN_PROGRESS

  Wenn Sie einen Job mithilfe der AWS IoT Konsole oder der DeleteJob API löschen, ändert sich der Jobstatus in. `DELETION_IN_PROGRESS` Während des Auftragslöschens beginnt AWS IoT Jobs mit dem Löschen zuvor erstellter Auftragsausführungen. Nachdem alle Auftragsausführungen gelöscht wurden, verschwindet der Job aus Ihrem AWS Konto.

Auftragsausführungsstatus

Die folgende Tabelle zeigt die verschiedenen Status einer AWS IoT Jobausführung und zeigt, ob die Statusänderung vom Gerät oder von AWS IoT Jobs initiiert wurde.

Status und Quelle der Auftragsausführung

Auftrags-Ausführungsstatus	Vom Gerät initiiert?	Von AWS IoT Jobs initiiert?	Terminal-Status?	Kann erneut versucht werden?
`QUEUED`	Nein	Ja	Nein	Nicht zutreffend
`IN_PROGRESS`	Ja	Nein	Nein	Nicht zutreffend
`SUCCEEDED`	Ja	Nein	Ja	Nicht zutreffend
`FAILED`	Ja	Nein	Ja	Ja
`TIMED_OUT`	Nein	Ja	Ja	Ja

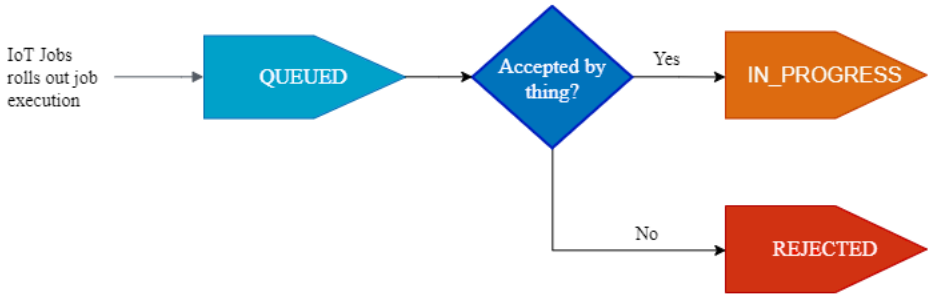
Auftrags-Ausführungsstatus	Vom Gerät initiiert?	Von AWS IoT Jobs initiiert?	Terminal-Status?	Kann erneut versucht werden?
`REJECTED`	Ja	Nein	Ja	Nein
`REMOVED`	Nein	Ja	Ja	Nein
`CANCELED`	Nein	Ja	Ja	Nein

Der folgende Abschnitt beschreibt mehr über die Status einer Jobausführung, die eingeführt wird, wenn Sie einen Job mit AWS IoT Jobs erstellen.

- IN WARTESCHLANGE

  Wenn AWS IoT Jobs eine Auftragsausführung für ein Zielgerät einführt, wird der Status der Auftragsausführung auf gesetzt`QUEUED`. Die Auftragsausführung bleibt so lange im Status `QUEUED`, bis:

  - Ihr Gerät empfängt die Auftragsausführung und ruft die Jobs API-Operationen auf und meldet den Status als `IN_PROGRESS`.
  - Sie brechen den Auftrag oder die Auftragsausführung ab, oder wenn die von Ihnen angegebenen Abbruchkriterien erfüllt sind und der Status sich auf `CANCELED` ändert.
  - Ihr Gerät wird aus der Zielgruppe entfernt und der Status ändert sich auf `REMOVED`.



- IN_PROGRESS

  Wenn Ihr IoT-Gerät das reservierte Auftragsthemen `$notify` und `$notify-next` Ihr Gerät entweder die `StartNextPendingJobExecution` API oder die `UpdateJobExecution` API mit einem Status von aufruft`IN_PROGRESS`, setzt AWS IoT Jobs den Status der Auftragsausführung auf. `IN_PROGRESS`

Die `UpdateJobExecution`-API kann mehrfach mit dem Status `IN_PROGRESS` aufgerufen werden. Sie können mithilfe des `statusDetails`-Objekts zusätzliche Details zu den Ausführungsschritten angeben.

> **Note**
>
> Wenn Sie mehrere Jobs für jedes Gerät erstellen, garantieren AWS IoT Jobs und das MQTT-Protokoll nicht die Reihenfolge der Lieferung.

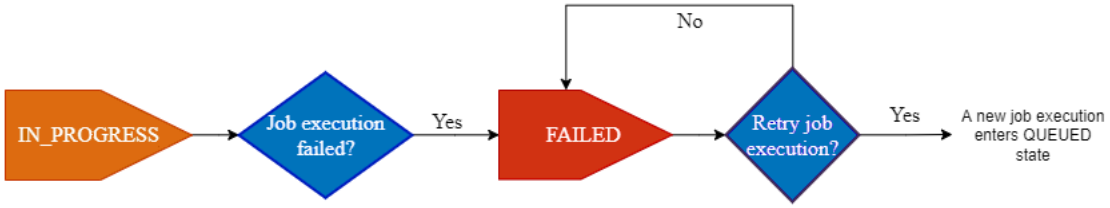
- SUCCEEDED

  Wenn Ihr Gerät den Fernvorgang erfolgreich abgeschlossen hat, muss das Gerät die `UpdateJobExecution` API mit dem Status aufrufen, `SUCCEEDED` um anzuzeigen, dass die Auftragsausführung erfolgreich war. AWS IoT Jobs wird dann aktualisiert und gibt den Status der Auftragsausführung als `SUCCEEDED` zurück.



- FEHLGESCHLAGEN

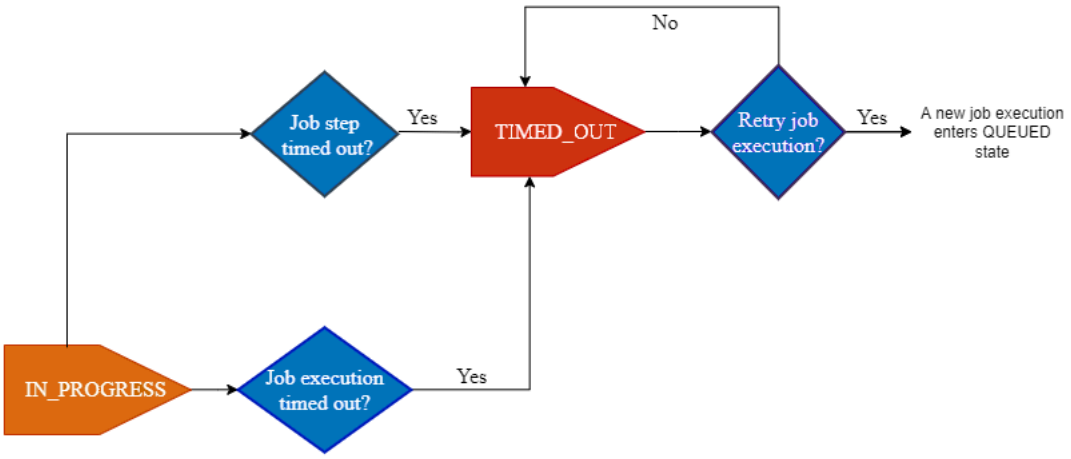
  Wenn Ihr Gerät den Fernvorgang nicht abschließt, muss das Gerät die `UpdateJobExecution` API mit dem Status aufrufen, `Failed` um anzuzeigen, dass die Auftragsausführung fehlgeschlagen ist. AWS IoT Jobs wird dann aktualisiert und gibt den Status der Auftragsausführung als `Failed` zurück. Sie können versuchen, diese Auftragsausführung für das Gerät mittels des Auftragsausführung: Konfiguration wiederholen erneut auszuführen.



- TIMED_OUT

  Wenn Ihr Gerät einen Auftragsschritt nicht abschließt, wenn der Status lautet`IN_PROGRESS`, oder wenn es den Fernvorgang nicht innerhalb der Zeitüberschreitungsdauer des in Bearbeitung befindlichen Timers abschließt, setzt AWS IoT Jobs den Status der Auftragsausführung auf.

`TIMED_OUT` Sie haben außerdem einen Schrittzeitgeber für jeden Auftragsschritt eines laufenden Auftrags, der nur für die Auftragsausführung gilt. Die Dauer des in Bearbeitung befindlichen Timers wird mithilfe der `inProgressTimeoutInMinutes`-Eigenschaft von Timeout-Konfiguration für die Auftragsausführung angegeben. Sie können versuchen, diese Auftragsausführung für das Gerät mittels des Auftragsausführung: Konfiguration wiederholen erneut auszuführen.



- ABGELEHNT

  Wenn Ihr Gerät eine ungültige oder inkompatible Anfrage erhält, muss das Gerät die `UpdateJobExecution` API mit dem Status aufrufen. `REJECTED` AWS IoT Jobs wird dann aktualisiert und gibt den Status der Auftragsausführung als `REJECTED` zurück.

- ENTFERNT

  Wenn Ihr Gerät kein gültiges Ziel für die Auftragsausführung mehr ist, z. B. wenn es von einer dynamischen Objektgruppe getrennt ist, setzt AWS IoT Jobs den Status der Auftragsausführung auf `REMOVED`. Sie können das Objekt wieder an Ihre Zielgruppe anhängen und die Auftragsausführung für das Gerät neu starten.

- CANCELED

  Wenn Sie einen Job oder die Ausführung eines Jobs mithilfe der Konsole oder der `CancelJobExecution` API abbrechen `CancelJob` oder wenn die mit der angegebenen Abbruchkriterien erfüllt sind, bricht AWS IoT Jobs den Konfiguration des Auftragsabbruchs Job ab und setzt den Auftragsausführungsstatus auf. `CANCELED`

Verwalten von Aufträgen

Verwenden Sie Aufträge, um Geräte über ein Software- oder Firmware-Update zu informieren. Sie können die AWS IoT Konsole, die API-Operationen zur Auftragsverwaltung und -steuerung, das AWS Command Line Interface oder die AWS SDKs zum Erstellen und Verwalten von Aufträgen verwenden.

Codesignatur für Aufgaben

Wenn Sie Code an Geräte senden, damit Geräte erkennen können, ob der Code während der Übertragung geändert wurde, empfehlen wir, die Codedatei mit dem AWS CLI zu signieren. Anweisungen finden Sie unter Erstellen und Verwalten von Aufträgen mithilfe von AWS CLI.

Weitere Informationen finden Sie unter Wozu dient Code Signing AWS IoT? .

Auftragsdokument

Bevor Sie einen Auftrag erstellen, müssen Sie ein Auftragsdokument erstellen. Wenn Sie Code Signing für verwenden AWS IoT, müssen Sie Ihr Auftragsdokument in einen versionierten Amazon S3-Bucket hochladen. Weitere Informationen zum Erstellen eines Amazon S3-Buckets und zum Hochladen von Dateien in diesen Bucket finden Sie unter Erste Schritte mit Amazon Simple Storage Service im Amazon S3 Einführungshandbuch.

> **Tip**
>
> Beispiele für Jobdokumente finden Sie im Beispiel jobs-agent.js im AWS IoT SDK für JavaScript.

Vorsignierte URLs

Ihr Auftragsdokument kann eine vorsignierte Amazon S3-URL enthalten, die auf Ihre Code-Datei (oder eine andere Datei) verweist. Vorsignierte Amazon S3-URLs gelten nur für einen begrenzten Zeitraum und werden generiert, wenn ein Gerät ein Auftragsdokument anfordert. Da die vorsignierte URL nicht erstellt wird, wenn Sie das Auftragsdokument erstellen, setzen Sie stattdessen eine Platzhalter-URL in Ihr Auftragsdokument. Eine Platzhalter-URL sieht wie folgt aus:

```
${aws:iot:s3-presigned-url-v2:https://
s3.region.amazonaws.com/<bucket>/<code file>}
```

Wobei:

- *bucket*ist der Amazon S3-Bucket, der die Codedatei enthält.
- *code file*ist der Amazon S3-Schlüssel der Codedatei.

Wenn ein Gerät das Jobdokument anfordert, AWS IoT generiert es die vorsignierte URL und ersetzt die Platzhalter-URL durch die vorsignierte URL. Ihr Auftragsdokument wird dann an das Gerät gesendet.

IAM-Rolle zur Erteilung der Erlaubnis für den Download von Dateien von S3

Wenn Sie einen Auftrag erstellen, der vorsignierte Amazon S3-URLs verwendet, müssen Sie eine IAM-Rolle angeben. Die Rolle muss die Berechtigung zum Herunterladen von Dateien aus dem Amazon S3-Bucket gewähren, in dem die Daten oder Updates gespeichert sind. Die Rolle muss auch die Berechtigung für AWS IoT zur Übernahme der Rolle gewähren.

Sie können optional ein Timeout für die vorsignierte URL angeben. Weitere Informationen finden Sie unter CreateJob.

Gewährung AWS IoT Jobs — Erlaubnis, Ihre Rolle zu übernehmen

1. Gehen Sie zum Rollen-Hub der IAM-Konsole und wählen Sie Ihre Rolle.
2. Wählen Sie auf der Registerkarte Vertrauensbeziehungen die Option Vertrauensstellung bearbeiten aus und ersetzen Sie das Richtliniendokument durch das folgende JSON-Dokument. Wählen Sie Update Trust Policy (Trust Policy aktualisieren).

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "iot.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
```

```
}
```

3. Verwenden Sie die globalen Konditionsschlüssel `aws:SourceArn` und `aws:SourceAccount` für die Richtlinie, um vor dem Confused-Deputy-Problem zu schützen.

> **Important**
>
> Ihr `aws:SourceArn` muss das Format einhalten: `arn:aws:iot:`*`region`*`:`*`account-id`*`:*`. Vergewissern Sie sich, dass dies *`region`* Ihrer AWS IoT Region und Ihrer Kundenkonto-ID *`account-id`* entspricht. Weitere Informationen finden Sie unter Cross-service Confused Deputy Prevention.

```
{
  "Effect": "Allow",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service":
          "iot.amazonaws.com"
       },
      "Action": "sts:AssumeRole",
      "Condition": {
         "StringEquals": {
            "aws:SourceAccount": "123456789012"
         },
         "ArnLike": {
              "aws:SourceArn": "arn:aws:iot:*:123456789012:job/*"
         }
       }
     }
   ]
}
```

4. Wenn Ihr Auftrag ein Auftragsdokument verwendet, das ein Amazon S3-Objekt ist, wählen Sie Berechtigungen und verwenden Sie das folgende JSON. Dadurch wird eine Richtlinie hinzugefügt, die das Herunterladen von Dateien aus Ihrem Amazon S3-Bucket gestattet:

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::your_S3_bucket/*"
        }
    ]
}
```

Vorsignierte URL für den Datei-Upload

Wenn Ihre Geräte während einer Auftragsbereitstellung Dateien in einen Amazon S3-Bucket hochladen müssen, können Sie den folgenden vorsignierten URL-Platzhalter in Ihr Jobdokument aufnehmen:

```
${aws:iot:s3-presigned-url-upload:https://s3.region.amazonaws.com/<bucket>/<key>}
```

Sie können innerhalb des `key` Attributs in der Platzhalter-URL für das Hochladen von `${thingName}` Dateien in Ihrem Jobdokument jeweils maximal zwei der Schlüsselwörter, und `${executionNumber}` als reservierte Schlüsselwörter verwenden. `${jobId}` Der lokale Platzhalter, der diese reservierten Schlüsselwörter im `key` Attribut darstellt, wird analysiert und ersetzt, wenn die Auftragsausführung erstellt wird. Die Verwendung eines lokalen Platzhalters mit reservierten Schlüsselwörtern, die für jedes Gerät spezifisch sind, stellt sicher, dass jede von einem Gerät hochgeladene Datei für dieses Gerät spezifisch ist und nicht durch eine ähnliche hochgeladene Datei von einem anderen Gerät überschrieben wird, für das dieselbe Jobbereitstellung vorgesehen ist. Informationen zur Problembehandlung bei lokalen Platzhaltern innerhalb eines vorsignierten URL-Platzhalters beim Hochladen von Dateien während einer Jobbereitstellung finden Sie unter. Allgemeine Fehlermeldungen zur Fehlerbehebung

> **Note**
>
> Der Name des Amazon S3-Buckets darf nicht den lokalen Platzhalter enthalten, der die reservierten Schlüsselwörter für die hochgeladene Datei darstellt. Der lokale Platzhalter muss sich im Attribut befinden. `key`

Dieser vorsignierte URL-Platzhalter wird in Ihrem Jobdokument in eine vorsignierte Amazon S3-Upload-URL umgewandelt, wenn ein Gerät ihn empfängt. Ihre Geräte verwenden dies, um Dateien in einen Amazon S3-Ziel-Bucket hochzuladen.

> **Note**
>
> Wenn der Amazon S3-Bucket und der Schlüssel nicht in der obigen Platzhalter-URL angegeben sind, generiert AWS IoT Jobs automatisch einen Schlüssel für jedes Gerät, wobei jeweils maximal zwei der Optionen `${thingName}${jobId}`, und `${executionNumber}` verwendet werden.

Vorsignierte URL unter Verwendung der Amazon S3-Versionierung

Der Schutz der Integrität einer Datei, die in einem Amazon S3-Bucket gespeichert ist, ist entscheidend für die sichere Bereitstellung von Aufträgen, die diese Datei verwenden, für Ihre Geräteflotte. Mithilfe der Amazon S3-Versionierung können Sie für jede Variante der Datei, die in Ihrem Amazon S3-Bucket gespeichert ist, eine Versionskennung hinzufügen, um jede Version der Datei nachzuverfolgen. Dies gibt Aufschluss darüber, welche Version der Datei mithilfe AWS IoT von Jobs für Ihre Geräteflotte bereitgestellt wird. Weitere Informationen zu Amazon S3-Buckets, die Versionierung verwenden, finden Sie unter Verwenden der Versionierung in Amazon S3-Buckets.

Wenn die Datei in Amazon S3 gespeichert ist und das Jobdokument einen vorsignierten URL-Platzhalter enthält, generiert AWS IoT Jobs eine vorsignierte URL im Jobdokument unter Verwendung des Amazon S3-Buckets, des Bucket-Schlüssels und der Version der im Amazon S3-Bucket gespeicherten Datei. Diese im Jobdokument generierte vorsignierte URL ersetzt den vorsignierten URL-Platzhalter, der ursprünglich im Jobdokument enthalten war. Wenn Sie die in Ihrem Amazon S3-Bucket gespeicherte Datei aktualisieren, `versionId` werden eine neue und nachfolgende Version der Datei erstellt, um die vorgenommenen Aktualisierungen zu signalisieren und die Möglichkeit zu bieten, diese bestimmte Datei bei zukünftigen Auftragsbereitstellungen als Ziel auszuwählen.

Anhand der folgenden Beispiele können Sie sich die vorsignierten Amazon S3-URLs in Ihrem Jobdokument vor und während der Bearbeitung ansehen. Verwenden Sie dazu: `versionId`

Amazon S3-Platzhalter für vorsignierte URLs (vor der Auftragsbereitstellung)

```
//Virtual-hosted style URL
```

```
${aws:iot:s3-presigned-url-v2:https://bucket-name.s3.region-code.amazonaws.com/key-name%3FversionId%3Dversion-id}

//Path-style URL
${aws:iot:s3-presigned-url-v2:https://s3.region-code.amazonaws.com/bucket-name/key-name%3FversionId%3Dversion-id}
```

Vorsignierte Amazon S3-URL (während der Auftragsbereitstellung)

```
//Virtual-hosted style URL
${aws:iot:s3-presigned-url-v2:https://sample-bucket-name.s3.us-west-2.amazonaws.com/sample-code-file.png%3FversionId%3Dversion1}

//Path-style
${aws:iot:s3-presigned-url-v2:https://s3.us-west-2.amazonaws.com/sample-bucket-name/sample-code-file.png%3FversionId%3Dversion1}
```

Weitere Informationen zu virtuell gehosteten Amazon S3-Objekt-URLs und Objekt-URLs im Pfadstil finden Sie unter Anfragen und Anfragen. Virtual-hosted-style Path-style

> **Note**
>
> Wenn Sie eine vorsignierte Amazon S3-URL anhängen `versionId` möchten, muss diese der Unterstützung für die URL-Codierung entsprechen. AWS SDK for Java 2.x Weitere Informationen finden Sie unter Änderungen beim Parsen von Amazon S3-URIs von Version 1 auf Version 2.

Versionsunterschiede zwischen vorsignierten URL-Platzhaltern in Amazon S3

In der folgenden Liste sind die Unterschiede zwischen den vorsignierten URL-Platzhaltern von Amazon S3 `${aws:iot:s3-presigned-url-v1` (Version 1) und `${aws:iot:s3-presigned-url-v2` (Version 2) aufgeführt:

- Der vorsignierte URL-Platzhalter `${aws:iot:s3-presigned-url-v1` von Amazon S3 wird nicht unterstützt. `version-id`
- Der vorsignierte URL-Platzhalter von Amazon S3 `${aws:iot:s3-presigned-url-v1` empfängt die Amazon S3-URL unverschlüsselt. Für den Amazon S3-Platzhalter für vorsignierte URLs `${aws:iot:s3-presigned-url-v2` muss die Amazon S3-URL so codiert sein, dass sie dem Amazon S3-SDK-Standard entspricht.

Themen

- Erstellen und verwalten Sie Jobs mithilfe der AWS-Managementkonsole
- Erstellen und verwalten Sie Jobs mithilfe der AWS CLI

Erstellen und verwalten Sie Jobs mithilfe der AWS-Managementkonsole

In diesem Abschnitt wird beschrieben, wie Sie Jobs von der AWS IoT Konsole aus erstellen und verwalten können. Nachdem Sie einen Job erstellt haben, können Sie auf der Detailseite Informationen zum Job anzeigen und den Job verwalten.

> **Note**
>
> Wenn Sie Codesignaturen für AWS IoT Jobs ausführen möchten, verwenden Sie die AWS CLI. Weitere Informationen finden Sie unter Erstellen und Verwalten von Aufträgen mithilfe von AWS CLI.

Themen

- Erstellen Sie Verwaltungsaufträge mithilfe der AWS-Managementkonsole
- Jobs anzeigen und verwalten, indem Sie AWS-Managementkonsole

Erstellen Sie Verwaltungsaufträge mithilfe der AWS-Managementkonsole

Um einen Job zu erstellen, melden Sie sich bei der AWS IoT Konsole an und gehen Sie im Bereich Remote-Aktionen zum Jobs-Hub. Führen Sie dann die folgenden Schritte aus.

1. Wählen Sie auf der Seite „Jobs“ im Dialogfeld „Jobs“ die Option „Job erstellen“ aus.
2. Je nachdem, welches Gerät Sie verwenden, können Sie einen benutzerdefinierten Job, einen FreeRTOS OTA-Aktualisierungsjob oder einen AWS IoT Greengrass Job erstellen. Wählen Sie für dieses Beispiel Einen benutzerdefinierten Auftragerstellen aus. Wählen Sie Weiter aus.
3. Geben Sie auf der Seite Benutzerdefinierte Auftragseigenschaften im Dialogfeld Auftragseigenschaften Ihre Informationen für die folgenden Felder ein:
   - Name: Geben Sie einen eindeutigen, alphanumerischen Auftragsnamen ein.
   - Beschreibung – optional: Geben Sie optional eine Beschreibung Ihres Auftrags ein.

- Markierungen – optional:

> **Note**
>
> Wir empfehlen Ihnen, in Ihren Auftrags-IDs und -beschreibungen keine PII zu verwenden.

Wählen Sie Weiter aus.

4. Wählen Sie auf der Seite Dateikonfiguration im Dialogfeld Auftragsziele die Objekte oder Objektgruppen aus, für die Sie diesen Auftrag ausführen möchten.

   Wählen Sie im Dialogfeld Auftragsdokument eine der folgenden Optionen aus:

   - Aus Datei: Eine JSON-Auftragsdatei, die Sie zuvor in einen Amazon S3-Bucket hochgeladen haben
     - Codesignatur

       In dem Auftragsdokument, das sich in Ihrer Amazon S3-URL befindet, ist `${aws:iot:code-sign-signature:s3://region.bucket/code-file@code-file-version-id}` als Platzhalter erforderlich, bis er mithilfe Ihres Codesignaturprofils durch den Pfad der signierten Codedatei ersetzt wird. Die neue signierte Codedatei wird zunächst in einem `SignedImages`-Ordner in Ihrem Amazon S3-Quell-Bucket angezeigt. Es wird ein neues Auftragsdokument mit einem `Codesigned_`-Präfix erstellt, wobei der Pfad der signierten Codedatei den Platzhalter für das Codezeichen ersetzt und in Ihre Amazon S3-URL eingefügt wird, um einen neuen Auftrag zu erstellen.
     - Pre-sign Ressourcen-URLs

       Wählen Sie im Pre-signing Rollen-Drop-down-Menü die IAM-Rolle aus, die Sie unter Vorsignierte URLs erstellt haben. Die Verwendung von `${aws:iot:s3-presigned-url:` zum Vorsignieren URLs für Objekte, die sich in Amazon S3 befinden, ist eine bewährte Sicherheitsmethode für Geräte, die Objekte von Amazon S3 herunterladen.

       Wenn Sie vorsignierte URLs für einen Codesignatur-Platzhalter verwenden möchten, verwenden Sie die folgende Beispielvorlage:

       ```
       ${aws:iot:s3-presigned-url:${aws:iot:code-sign-signature:<S3 URL>}
       ```

- Aus Vorlage: Eine Auftragsvorlage, die ein Auftragsdokument und Auftragskonfigurationen enthält. Bei der Jobvorlage kann es sich um eine benutzerdefinierte Jobvorlage handeln, die Sie erstellt haben, oder um eine AWS verwaltete Vorlage.

  Wenn Sie einen Job erstellen, um häufig verwendete Remote-Aktionen wie den Neustart Ihres Geräts auszuführen, können Sie eine AWS verwaltete Vorlage verwenden. Diese Vorlagen wurden bereits für die Verwendung vorkonfiguriert. Weitere Informationen erhalten Sie unter Erstellen einer benutzerdefinierten Auftragsvorlage und Erstellen Sie benutzerdefinierte Jobvorlagen aus verwalteten Vorlagen.

5. Wählen Sie auf der Seite Auftragskonfiguration im Dialogfeld Auftragskonfiguration einen der folgenden Auftragstypen aus:

   - Snapshot-Auftrag: Ein Snapshot-Auftrag ist abgeschlossen, wenn er abgeschlossen ist. Er wird auf den Zielgeräten und -gruppen ausgeführt.
   - Kontinuierlicher Auftrag: Ein kontinuierlicher Auftrag gilt für Objektgruppen und wird auf jedem Gerät ausgeführt, das Sie später zu einer bestimmten Zielgruppe hinzufügen.

6. Überprüfen Sie im Dialogfeld Zusätzliche Konfigurationen – optional die folgenden optionalen Auftragskonfigurationen und treffen Sie Ihre Auswahl entsprechend.

   - Rollout-Konfiguration
   - Konfiguration des Zeitplans
   - Konfiguration des Timeouts für Auftragsausführungen
   - Auftragsausführungen: Konfiguration wiederholen – neu
   - Konfiguration abbrechen

   In den folgenden Abschnitten finden Sie weitere Informationen zu Auftragskonfigurationen:

   - Konfigurationen für Auftrags-Rollout, Planung und Abbruch
   - Timeout bei der Auftragsausführung und Wiederholungskonfigurationen

   Überprüfen Sie alle Ihre Auftragsauswahlen und wählen Sie dann Absenden, um Ihren Auftrag zu erstellen.

Jobs anzeigen und verwalten, indem Sie AWS-Managementkonsole

Nachdem Sie den Auftrag erstellt haben, generiert die Konsole eine JSON-Signatur und setzt sie in Ihr Auftragsdokument. Sie können die AWS IoT -Konsole verwenden, um den Status anzuzeigen, oder um einen Auftrag abzubrechen oder zu löschen.

Wenn Sie den Job auswählen, den Sie erstellt haben, finden Sie:

- Allgemeine Auftragsdetails wie Jobname, Beschreibung, Typ, Zeitpunkt der Erstellung, letzte Aktualisierung und geschätzte Startzeit.
- Alle von Ihnen angegebenen Auftragskonfigurationen und deren Status.
- Das Auftragsdokument.
- Die Auftragsausführungen und alle optionalen Tags, die Sie angegeben haben.

Um Jobs zu verwalten, gehen Sie zum Job-Hub der Konsole und wählen Sie aus, ob Sie den Job bearbeiten, löschen oder abbrechen möchten.

Erstellen und verwalten Sie Jobs mithilfe der AWS CLI

In diesem Abschnitt wird beschrieben, wie Sie Aufträge erstellen und verwalten.

Erstellen von Aufträgen

Verwenden Sie den CreateJob Befehl, um einen AWS IoT Job zu erstellen. Der Auftrag wird in die Warteschlange für die Ausführung auf den Zielen (Objekten oder Objektgruppen) gesetzt, die Sie angeben. Um einen AWS IoT Job zu erstellen, benötigen Sie ein Jobdokument, das in den Text der Anfrage oder als Link zu einem Amazon S3-Dokument aufgenommen werden kann. Wenn der Job das Herunterladen von Dateien mithilfe vorsignierter Amazon S3-URLs beinhaltet, benötigen Sie eine IAM-Rolle (Amazon Resource Name, ARN), die zum Herunterladen der Datei berechtigt ist und dem AWS IoT Jobs-Service die Erlaubnis erteilt, die Rolle zu übernehmen.

Weitere Informationen zur Syntax bei der Eingabe von Datum und Uhrzeit mithilfe eines API-Befehls oder des finden Sie unter AWS CLI Timestamp.

Codesignatur mit Aufträgen

Wenn Sie Code Signing für verwenden AWS IoT, müssen Sie einen Codesignatur-Job starten und die Ausgabe in Ihr Jobdokument aufnehmen. Dadurch wird der Platzhalter für die

Codezeichensignatur in Ihrem Auftragsdokument ersetzt, der als Platzhalter erforderlich ist, bis er mithilfe Ihres Codesignaturprofils durch den Pfad der signierten Codedatei ersetzt wird. Der Platzhalter für die Codesignatur sieht wie folgt aus:

```
    ${aws:iot:code-sign-signature:s3://region.bucket/code-file@code-file-version-id}
```

Verwenden Sie den Befehl start-signing-job, um einen Codesignaturauftrag zu erstellen. `start-signing-job` gibt eine Auftrags-ID aus. Verwenden Sie den Befehl describe-signing-job, um den Amazon S3-Ort abzurufen, an dem die Signatur gespeichert ist. Anschließend können Sie die Signatur von Amazon S3 herunterladen. Weitere Informationen zu Codesignaturaufträgen finden Sie unter Codesignatur für AWS IoT.

Ihr Auftragsdokument muss einen vorsignierten URL Platzhalter für Ihre Code-Datei haben, und die JSON-Signaturausgabe muss in einem Amazon S3-Bucket unter Verwendung des start-signing-job-Befehls platziert sein:

```
{
    "presign": "${aws:iot:s3-presigned-url:https://s3.region.amazonaws.com/bucket/
image}",
}
```

Erstellen eines Auftrags mit einem Auftragsdokument

Der folgende Befehl zeigt, wie Sie einen Job mithilfe eines in einem Amazon S3-Bucket (*job-document.json*) gespeicherten Jobdokuments (*jobBucket*) und einer Rolle mit der Berechtigung zum Herunterladen von Dateien von Amazon S3 (*S3DownloadRole*) erstellen.

```
aws iot create-job  \
      --job-id 010  \
      --targets arn:aws:iot:us-east-1:123456789012:thing/thingOne  \
      --document-source https://s3.amazonaws.com/amzn-s3-demo-bucket/job-document.json
 \
      --timeout-config inProgressTimeoutInMinutes=100 \
      --job-executions-rollout-config "{ \"exponentialRate\": { \"baseRatePerMinute
\": 50, \"incrementFactor\": 2, \"rateIncreaseCriteria\": { \"numberOfNotifiedThings
\": 1000, \"numberOfSucceededThings\": 1000}}, \"maximumPerMinute\": 1000}" \
      --abort-config "{ \"criteriaList\": [ { \"action\": \"CANCEL\", \"failureType
\": \"FAILED\", \"minNumberOfExecutedThings\": 100, \"thresholdPercentage\": 20},
 { \"action\": \"CANCEL\", \"failureType\": \"TIMED_OUT\", \"minNumberOfExecutedThings
\": 200, \"thresholdPercentage\": 50}]}" \
```

```
      --presigned-url-config "{\"roleArn\":\"arn:aws:iam::123456789012:role/
S3DownloadRole\", \"expiresInSec\":3600}"
```

Der Job wird am ausgeführt*thingOne*.

Der optionale `timeout-config`-Parameter gibt die Dauer an, die jedes Gerät für den Abschluss der Ausführung des Auftrags hat. Der Timer wird gestartet, wenn der Status der Auftragsausführung auf `IN_PROGRESS` gesetzt wird. Wird der Status der Auftragsausführung vor Ablauf der Zeit nicht auf einen anderen Terminal-Zustand festgelegt, wird er automatisch auf `TIMED_OUT` festgelegt.

Der Timer für „In Bearbeitung“ kann nicht aktualisiert werden und gilt für alle Auftragsausführungen für den Auftrag. Immer wenn eine Auftragsausführung länger als dieses Intervall im `IN_PROGRESS` Status verbleibt, schlägt sie fehl und wechselt in den `TIMED_OUT` Terminalstatus. AWS IoT veröffentlicht auch eine MQTT-Benachrichtigung.

Weitere Informationen zum Konfigurieren von Auftragsrollouts und -abbrüchen finden Sie unter Auftragsrollout- und Abbruchkonfiguration.

> **Note**
>
> Auftragsdokumente, die als Amazon S3-Dateien angegeben werden, werden zum Zeitpunkt der Erstellung des Auftrags abgerufen. Wenn Sie den Inhalt der Amazon S3-Datei, die Sie als Quelle Ihres Auftragsdokuments verwendet haben, ändern, nachdem Sie das Auftragsdokument erstellt haben, ändert sich das, was an die Auftragsziele gesendet wird, nicht.

Aktualisieren eines Auftrags

Um einen Auftrag zu aktualisieren, verwenden Sie den Befehl UpdateJob. Sie können die Felder `description`, `presignedUrlConfig`, `jobExecutionsRolloutConfig`, `abortConfig` und `timeoutConfig` eines Auftrags aktualisieren.

```
aws iot update-job  \
  --job-id 010  \
  --description "updated description" \
  --timeout-config inProgressTimeoutInMinutes=100 \
  --job-executions-rollout-config "{ \"exponentialRate\": { \"baseRatePerMinute\": 50,
 \"incrementFactor\": 2, \"rateIncreaseCriteria\": { \"numberOfNotifiedThings\": 1000,
 \"numberOfSucceededThings\": 1000}, \"maximumPerMinute\": 1000}}" \
```

```
  --abort-config "{ \"criteriaList\": [ { \"action\": \"CANCEL\", \"failureType
\": \"FAILED\", \"minNumberOfExecutedThings\": 100, \"thresholdPercentage\": 20},
 { \"action\": \"CANCEL\", \"failureType\": \"TIMED_OUT\", \"minNumberOfExecutedThings
\": 200, \"thresholdPercentage\": 50}]}" \
  --presigned-url-config "{\"roleArn\":\"arn:aws:iam::123456789012:role/
S3DownloadRole\", \"expiresInSec\":3600}"
```

Weitere Informationen finden Sie unter Rollout von Aufträgen und Abbruchskonfiguration.

Abbrechen eines Auftrags

Um einen Auftrag abzubrechen, verwenden Sie den Befehl CancelJob. Durch das Abbrechen eines Jobs AWS IoT werden keine neuen Jobausführungen für den Job bereitgestellt. Außerdem werden alle Jobausführungen storniert, die sich in einem bestimmten Status befinden. `QUEUED` AWS IoT Alle Auftragsausführungen im Terminalzustand bleiben unberührt, da das Gerät den Job bereits abgeschlossen hat. Wenn der Status einer Auftragsausführung `IN_PROGRESS` ist, wird sie auch nicht verändert, es sei denn, Sie verwenden den optionalen `--force`-Parameter.

Der folgende Befehl zeigt, wie Sie einen Auftrag mit der ID 010 abbrechen.

```
aws iot cancel-job --job-id 010
```

Die Ausgabe des Befehls sieht wie folgt aus:

```
{
    "jobArn": "string",
    "jobId": "string",
    "description": "string"
}
```

Wenn Sie einen Auftrag abbrechen, werden Auftragsausführungen mit dem Status `QUEUED` abgebrochen. Auftragsausführungen mit dem Status `IN_PROGRESS` werden abgebrochen, aber nur, wenn Sie den optionalen `--force`-Parameter angeben. Auftragsausführungen mit einem Terminal-Zustand werden nicht abgebrochen.

> **Warning**
>
> Der Abbruch eines Auftrags im Status `IN_PROGRESS` (durch Setzen des `--force`-Parameters) bricht alle Auftragsausführungen ab, die sich in Ausführung befinden, was dazu führt, dass das Gerät, das den Auftrag ausführt, den Status der Auftragsausführung nicht

aktualisieren kann. Seien Sie vorsichtig, und stellen Sie sicher, dass jedes Gerät, das einen abgebrochenen Auftrag ausführt, in einen gültigen Status zurückkehren kann.

Der Status eines stornierten Jobs oder einer seiner Auftragsausführungen ist letztendlich konsistent. AWS IoT beendet so schnell wie möglich die Planung neuer `QUEUED` Auftragsausführungen und Auftragsausführungen für diesen Job auf Geräten. Die Änderung des Status einer Auftragsausführung zu `CANCELED` kann jedoch je nach der Anzahl der Geräte und anderen Faktoren einige Zeit dauern.

Wenn ein Auftrag abgebrochen wurde, da er die von einem `AbortConfig`-Objekt definierten Kriterien erfüllt hat, fügt der Service automatisch eingegebene Werte für die Felder `comment` und `reasonCode` hinzu. Sie können Ihre eigenen Werte für `reasonCode` erstellen, wenn der Abbruch des Auftrags vom Benutzer gesteuert wird.

Abbrechen einer Auftragsausführung

Um eine Auftragsausführung auf einem Gerät abzubrechen, verwenden Sie den CancelJobExecution-Befehl. Damit wird eine Auftragsausführung abgebrochen, die sich im Status `QUEUED` befindet. Wenn Sie eine Auftragsausführung abbrechen möchten, die sich in Ausführung befindet, müssen Sie den `--force`-Parameter verwenden.

Der folgende Befehl zeigt, wie die Ausführung von Auftrag 010 auf `myThing` abgebrochen wird.

```
aws iot cancel-job-execution --job-id 010 --thing-name myThing
```

Der Befehl zeigt keine Ausgabe an.

Eine Auftragsausführung, die sich im Status `QUEUED` befindet, wird abgebrochen. Eine Auftragsausführung mit dem Status `IN_PROGRESS` wird abgebrochen, aber nur, wenn Sie den optionalen `--force`-Parameter angeben. Auftragsausführungen mit einem Terminal-Zustand können nicht abgebrochen werden.

> **Warning**
>
> Der Abbruch einer Auftragsausführung mit dem Status `IN_PROGRESS` führt dazu, dass das Gerät den Ausführungsstatus für den Auftrag nicht aktualisieren kann. Seien Sie vorsichtig, und stellen Sie sicher, dass das Gerät in der Lage ist, in einen gültigen Status zurückzukehren.

Wenn sich die Auftragsausführung in einem Terminal-Zustand oder im Status `IN_PROGRESS` befindet und der `--force`-Parameter nicht auf `true` gesetzt ist, führt dieser Befehl zu einer `InvalidStateTransitionException`.

Der Status einer abgebrochenen Auftragsausführungen bleibt schließlich konsistent. Das Ändern des Status einer Auftragsausführung zu `CANCELED` kann aufgrund verschiedener Faktoren einige Zeit in Anspruch nehmen.

Löschen eines Auftrags

Um einen Auftrag und dessen Auftragsausführungen zu löschen, verwenden Sie den DeleteJob-Befehl. Standardmäßig können Sie nur einen Auftrag löschen, der sich in einem Terminal-Zustand (`SUCCEEDED` oder `CANCELED`) befindet. Andernfalls tritt eine Ausnahme auf. Sie können einen Auftrag im Status `IN_PROGRESS` löschen, jedoch nur, wenn der `force`-Parameter auf `true` gesetzt ist.

Um einen Auftrag zu löschen, führen Sie den folgenden Befehl aus:

```
aws iot delete-job --job-id 010 --force|--no-force
```

Der Befehl zeigt keine Ausgabe an.

> **Warning**
>
> Wenn Sie einen Auftrag löschen, der sich im `IN_PROGRESS`-Status befindet, kann das Gerät, das den Auftrag umsetzt, nicht auf Auftragsinformationen zugreifen oder den Status der Auftragsausführung aktualisieren. Seien Sie vorsichtig, und stellen Sie sicher, dass jedes Gerät, das einen gelöschten Auftrag umsetzt, in einen gültigen Status zurückkehren kann.

Das Löschen eines Auftrags kann einige Zeit in Anspruch nehmen, abhängig von der Anzahl der Auftragsausführungen für den Auftrag und anderen Faktoren. Während der Auftrag gelöscht wird, wird `DELETION_IN_PROGRESS` als Status des Auftrags angezeigt. Es tritt ein Fehler auf, wenn Sie versuchen, einen Auftrag abzubrechen oder zu löschen, dessen Status bereits `DELETION_IN_PROGRESS` ist.

Nur 10 Aufträge können gleichzeitig den Status `DELETION_IN_PROGRESS` haben. Andernfalls tritt eine `LimitExceededException` auf.

Abrufen eines Auftragsdokuments

Um ein Auftragsdokument für einen Auftrag abzurufen, verwenden Sie den Befehl GetJobDocument. Ein Auftragsdokument ist eine Beschreibung der Remoteoperationen, die von den Geräten ausgeführt werden sollen.

Um ein Auftragsdokument zu erhalten, führen Sie den folgenden Befehl aus:

```
aws iot get-job-document --job-id 010
```

Der Befehl gibt das Auftragsdokument für den angegebenen Auftrag aus:

```
{
    "document": "{\n\t\"operation\":\"install\",\n\t\"url\":\"http://amazon.com/
firmWareUpate-01\",\n\t\"data\":\"${aws:iot:s3-presigned-url:https://s3.amazonaws.com/
amzn-s3-demo-bucket/datafile}\"\n}"
}
```

> **Note**
>
> Wenn Sie mit diesem Befehl ein Auftragsdokument abrufen, werden Platzhalter-URLs nicht durch vorsignierte Amazon S3-URLs ersetzt. Wenn ein Gerät den GetPendingJobExecutions API-Vorgang aufruft, werden die Platzhalter-URLs im Jobdokument durch vorsignierte Amazon S3-URLs ersetzt.

Auflisten von Aufträgen

Um eine Liste aller Jobs in Ihrem zu erhalten AWS-Konto, verwenden Sie den ListJobs Befehl. Auftragsdaten und Auftragsausführungsdaten werden für eine begrenzte Zeitaufbewahrt. Führen Sie den folgenden Befehl aus, um alle Jobs in Ihrem aufzulisten AWS-Konto:

```
aws iot list-jobs
```

Der Befehl listet alle Aufträge in Ihrem Konto nach Auftragsstatus sortiert auf:

```
{
    "jobs": [
        {
            "status": "IN_PROGRESS",
```

```
            "lastUpdatedAt": 1486687079.743,
            "jobArn": "arn:aws:iot:us-east-1:123456789012:job/013",
            "createdAt": 1486687079.743,
            "targetSelection": "SNAPSHOT",
            "jobId": "013"
        },
        {
            "status": "SUCCEEDED",
            "lastUpdatedAt": 1486685868.444,
            "jobArn": "arn:aws:iot:us-east-1:123456789012:job/012",
            "createdAt": 1486685868.444,
            "completedAt": 148668789.690,
            "targetSelection": "SNAPSHOT",
            "jobId": "012"
        },
        {
            "status": "CANCELED",
            "lastUpdatedAt": 1486678850.575,
            "jobArn": "arn:aws:iot:us-east-1:123456789012:job/011",
            "createdAt": 1486678850.575,
            "targetSelection": "SNAPSHOT",
            "jobId": "011"
        }
    ]
}
```

Beschreiben eines Auftrags

Um den Status eines Auftrags abzurufen, führen Sie den Befehl DescribeJob aus. Der folgende Befehl zeigt, wie Sie einen Auftrag beschreiben:

```
$ aws iot describe-job --job-id 010
```

Der Auftrag gibt den Status des angegebenen Auftrags aus. Beispiel:

```
{
    "documentSource": "https://s3.amazonaws.com/amzn-s3-demo-bucket/job-
document.json",
    "job": {
        "status": "IN_PROGRESS",
        "jobArn": "arn:aws:iot:us-east-1:123456789012:job/010",
        "targets": [
            "arn:aws:iot:us-east-1:123456789012:thing/myThing"
```

```
        ],
        "jobProcessDetails": {
            "numberOfCanceledThings": 0,
            "numberOfFailedThings": 0,
            "numberOfInProgressThings": 0,
            "numberOfQueuedThings": 0,
            "numberOfRejectedThings": 0,
            "numberOfRemovedThings": 0,
            "numberOfSucceededThings": 0,
            "numberOfTimedOutThings": 0,
            "processingTargets": [
                arn:aws:iot:us-east-1:123456789012:thing/thingOne,
                arn:aws:iot:us-east-1:123456789012:thinggroup/thinggroupOne,
                arn:aws:iot:us-east-1:123456789012:thing/thingTwo,
                arn:aws:iot:us-east-1:123456789012:thinggroup/thinggroupTwo
            ]
        },
        "presignedUrlConfig": {
            "expiresInSec": 60,
            "roleArn": "arn:aws:iam::123456789012:role/S3DownloadRole"
        },
        "jobId": "010",
        "lastUpdatedAt": 1486593195.006,
        "createdAt": 1486593195.006,
        "targetSelection": "SNAPSHOT",
        "jobExecutionsRolloutConfig": {
            "exponentialRate": {
                "baseRatePerMinute": integer,
                "incrementFactor": integer,
                "rateIncreaseCriteria": {
                    "numberOfNotifiedThings": integer, // Set one or the other
                    "numberOfSucceededThings": integer // of these two values.
                },
            "maximumPerMinute": integer
         }
        },
        "abortConfig": {
            "criteriaList": [
                {
                    "action": "string",
                    "failureType": "string",
                    "minNumberOfExecutedThings": integer,
                    "thresholdPercentage": integer
                }
```

```
            ]
        },
        "timeoutConfig": {
           "inProgressTimeoutInMinutes": number
          }
    }
}
```

Auflisten von Ausführungen für einen Auftrag

Ein Auftrag, der auf einem bestimmten Gerät ausgeführt wird, wird durch ein Auftragsausführungsobjekt repräsentiert. Mit dem Befehl ListJobExecutionsForJob listen Sie alle Auftragsausführungen für einen Auftrag auf. Nachfolgend sehen Sie, wie Sie die Ausführungen für einen Auftrag auflisten können:

```
aws iot list-job-executions-for-job --job-id 010
```

Der Befehl gibt eine Liste von Auftragsausführungen zurück:

```
{
    "executionSummaries": [
    {
        "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/thingOne",
        "jobExecutionSummary": {
            "status": "QUEUED",
            "lastUpdatedAt": 1486593196.378,
            "queuedAt": 1486593196.378,
            "executionNumber": 1234567890
        }
    },
    {
        "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/thingTwo",
        "jobExecutionSummary": {
            "status": "IN_PROGRESS",
            "lastUpdatedAt": 1486593345.659,
            "queuedAt": 1486593196.378,
            "startedAt": 1486593345.659,
            "executionNumber": 4567890123
        }
    }
    ]
}
```

Auflisten von Auftragsausführungen für ein Objekt

Mit dem Befehl ListJobExecutionsForThing listen Sie alle Auftragsausführungen auf einem Objekt auf. Nachfolgend sehen Sie, wie Sie die Auftragsausführungen für ein Objekt auflisten können:

```
aws iot list-job-executions-for-thing --thing-name thingOne
```

Der Befehl gibt eine Liste der Auftragsausführungen aus, die auf dem angegebenen Objekt ausgeführt werden oder wurden:


```
{
    "executionSummaries": [
    {
        "jobExecutionSummary": {
            "status": "QUEUED",
            "lastUpdatedAt": 1486687082.071,
            "queuedAt": 1486687082.071,
            "executionNumber": 9876543210
        },
        "jobId": "013"
    },
    {
        "jobExecutionSummary": {
            "status": "IN_PROGRESS",
            "startAt": 1486685870.729,
            "lastUpdatedAt": 1486685870.729,
            "queuedAt": 1486685870.729,
            "executionNumber": 1357924680
        },
        "jobId": "012"
    },
    {
        "jobExecutionSummary": {
            "status": "SUCCEEDED",
            "startAt": 1486678853.415,
            "lastUpdatedAt": 1486678853.415,
            "queuedAt": 1486678853.415,
            "executionNumber": 4357680912
        },
        "jobId": "011"
    },
    {
        "jobExecutionSummary": {
```

```
            "status": "CANCELED",
            "startAt": 1486593196.378,
            "lastUpdatedAt": 1486593196.378,
            "queuedAt": 1486593196.378,
            "executionNumber": 2143174250
        },
        "jobId": "010"
    }
    ]
}
```

Beschreiben der Auftragsausführung

Mit dem Befehl DescribeJobExecution rufen Sie den Status einer Auftragsausführung ab. Sie müssen eine Auftrags-ID und den Namen eines Objekts sowie optional eine Ausführungsnummer angeben, um die Auftragsausführung zu identifizieren. Nachfolgend sehen Sie, wie Sie eine Auftragsausführung beschreiben können:

```
aws iot describe-job-execution --job-id 017 --thing-name thingOne
```

Der Befehl gibt den `JobExecution` zurück: Beispiel:

```
{
    "execution": {
        "jobId": "017",
        "executionNumber": 4516820379,
        "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/thingOne",
        "versionNumber": 123,
        "createdAt": 1489084805.285,
        "lastUpdatedAt": 1489086279.937,
        "startedAt": 1489086279.937,
        "status": "IN_PROGRESS",
        "approximateSecondsBeforeTimedOut": 100,
        "statusDetails": {
            "status": "IN_PROGRESS",
            "detailsMap": {
                "percentComplete": "10"
            }
        }
    }
}
```

Löschen einer Auftragsausführung

Mit dem DeleteJobExecution-Befehl löschen Sie eine Auftragsausführung. Sie müssen eine Auftrags-ID und den Namen eines Objekts und eine Ausführungsnummer angeben, um die Auftragsausführung zu identifizieren. Nachfolgend sehen Sie, wie Sie eine Auftragsausführung löschen können:

```
aws iot delete-job-execution --job-id 017 --thing-name thingOne --execution-number
 1234567890 --force|--no-force
```

Der Befehl zeigt keine Ausgabe an.

Standardmäßig muss der Status der Auftragsausführung `QUEUED` oder ein Terminal-Zustand (`SUCCEEDED`, `FAILED`, `REJECTED`, `TIMED_OUT`, `REMOVED` oder `CANCELED`) sein. Andernfalls tritt ein Fehler auf. Um eine Auftragsausführung mit dem Status `IN_PROGRESS` zu löschen, können Sie den `force`-Parameter auf `true` setzen.

> **Warning**
>
> Wenn Sie eine Auftragsausführung löschen, die sich im `IN_PROGRESS`-Status befindet, kann das Gerät, das den Auftrag ausführt, nicht auf Auftragsinformationen zugreifen oder den Status der Auftragsausführung aktualisieren. Seien Sie vorsichtig, und stellen Sie sicher, dass das Gerät in der Lage ist, in einen gültigen Status zurückzukehren.

Auftragsvorlagen

Verwenden Sie Auftragsvorlagen, um Aufträge vorzukonfigurieren, die Sie auf mehreren Gruppen von Zielgeräten bereitstellen können. Zum Bereitstellen häufig ausgeführter Remote-Aktionen wie das Neustarten oder Installieren einer Anwendung auf Ihren Geräten können Sie Vorlagen verwenden, um Standardkonfigurationen zu definieren. Zum Durchführen von Vorgängen wie die Bereitstellung von Sicherheitspatches und Bugfixes können Sie Vorlagen aus vorhandenen Aufträgen erstellen.

Geben Sie beim Erstellen einer Auftragsvorlage die folgenden zusätzlichen Konfigurationen und Ressourcen an.

- Auftragseigenschaften
- Auftragsunterlagen und Ziele

- Kriterien für Rollout, Planung und Abbrechen
- Kriterien für Zeitüberschreibung und Wiederholung

Benutzerdefiniert und AWS verwaltete Vorlagen

Je nachdem, welche Remote-Aktion Sie ausführen möchten, können Sie entweder eine benutzerdefinierte Auftragsvorlage erstellen oder eine AWS verwaltete Vorlage verwenden. Verwenden Sie benutzerdefinierte Auftragsvorlagen, um Ihr eigenes benutzerdefiniertes Auftragsdokument bereitzustellen und wiederverwendbare Jobs zu erstellen, die Sie auf Ihren Geräten bereitstellen können. AWS Verwaltete Vorlagen sind Jobvorlagen, die von AWS IoT Jobs für häufig ausgeführte Aktionen bereitgestellt werden. Diese Vorlagen enthalten ein vordefiniertes Auftragsdokument für einige Remote-Aktionen, sodass Sie kein eigenes Auftragsdokument erstellen müssen. Verwaltete Vorlagen helfen Ihnen dabei, wiederverwendbare Aufträge zu erstellen, damit Sie diese Aufträge schneller auf Ihren Geräten starten können.

Themen

- Verwenden Sie AWS verwaltete Vorlagen zur Bereitstellung gängiger Remote-Operationen
- Erstellen von benutzerdefinierten Auftragsvorlagen

Verwenden Sie AWS verwaltete Vorlagen zur Bereitstellung gängiger Remote-Operationen

AWS Verwaltete Vorlagen sind Auftragsvorlagen, die von bereitgestellt werden AWS. Sie werden für häufig ausgeführte Remote-Aktionen wie den Neustart, das Herunterladen einer Datei oder die Installation einer Anwendung auf Ihren Geräten verwendet. Diese Vorlagen verfügen über ein vordefiniertes Auftragsdokument für jede Remote-Aktion, sodass Sie kein eigenes Auftragsdokument erstellen müssen.

Sie können aus einer Reihe vordefinierter Konfigurationen auswählen und Aufträge mit diesen Vorlagen erstellen, ohne zusätzlichen Code schreiben zu müssen. Mithilfe verwalteter Vorlagen können Sie das Auftragsdokument einsehen, das für Ihre Flotten bereitgestellt wurde. Sie können mithilfe dieser Vorlagen einen Auftrag erstellen und eine benutzerdefinierte Auftragsvorlage erstellen, die Sie für Ihre Remoteoperationen wiederverwenden können.

Was enthalten verwaltete Vorlagen?

Jede AWS verwaltete Vorlage enthält:

- Die Umgebung, in der die Befehle im Auftragsdokument ausgeführt werden.
- Ein Auftragsdokument, das den Namen und die Parameter der Operation angibt. Wenn Sie beispielsweise eine Vorlage für Download Datei verwenden, lautet der Name der Operation Datei herunterladen und die Parameter können wie folgt lauten:
  - Die URL der Datei, die Sie auf Ihr Gerät herunterladen möchten. Dies kann eine Internetressource oder eine öffentliche oder vorsignierte Amazon Simple Storage Service (Amazon S3)-URL sein.
  - Ein lokaler Dateipfad auf dem Gerät zum Speichern der heruntergeladenen Datei.

  Weitere Information zum Auftragsdokument und den Parametern finden Sie unter Verwaltete Vorlagen für Remote-Aktionen und Auftragsdokumente.

Voraussetzungen

Damit Ihre Geräte die Remote-Aktionen ausführen können, die im verwalteten Auftragsvorlagendokument festgelegt sind, müssen Sie Folgendes tun:

- Installieren Sie die bestimmte Software auf Ihrem Gerät.

  Verwenden Sie Ihre eigene Gerätesoftware und Ihre eigenen Job-Handler oder den AWS IoT Device Client. Abhängig von Ihrem Geschäftsfall können Sie beide auch so ausführen, dass sie unterschiedliche Funktionen durchführen.

  - Verwenden Sie Ihre eigene Gerätesoftware und Auftragshandler

    Sie können Ihren eigenen Code für Geräte schreiben, indem Sie AWS IoT Device SDK und die zugehörigen Handler-Bibliotheken verwenden, mit denen die Remoteoperationen unterstützt werden. Stellen Sie sicher, dass zum Bereitstellen und Ausführen von Aufträgen die Geräteagent-Bibliotheken korrekt installiert wurden und auf den Geräten ausgeführt werden.

    Sie können auch Ihre eigenen Handler verwenden, mit denen die Remoteoperationen unterstützt werden. Weitere Informationen finden Sie unter Beispiel-Job-Handler im AWS IoT Device GitHub Client-Repository.

  - Verwenden Sie den AWS IoT Device Client

    Sie können den AWS IoT Device Client auch auf Ihren Geräten installieren und ausführen, da er standardmäßig die Verwendung aller verwalteten Vorlagen direkt von der Konsole aus unterstützt.

Der Device Client ist eine in C++ geschriebene Open-Source-Software, die Sie kompilieren und auf Ihren eingebetteten Linux-based IoT-Geräten installieren können. Der Device Client verfügt über einen Basisclient und separate clientseitige Funktionen. Der Basisclient stellt die Konnektivität AWS IoT über das MQTT-Protokoll her und kann eine Verbindung zu den verschiedenen clientseitigen Funktionen herstellen.

Verwenden Sie die clientseitige Auftragsfunktion des Geräteclients, um Remoteoperationen auf Ihren Geräten durchzuführen. Diese Funktion enthält einen Parser zum Empfangen des Auftragsdokuments und Auftragshandler, mit denen die im Auftragsdokument angegebenen Remote-Aktionen implementiert sind. Weitere Informationen zum Device Client und dessen zugehörige Funktionen finden unter AWS IoT Device Client.

Bei der Ausführung auf Geräten empfängt der Device Client das Auftragsdokument und verfügt über eine plattformspezifische Implementierung, mit der er Befehle im Dokument ausführt. Weitere Informationen zum Einrichten des Device Client und zur Verwendung der Funktion Aufträge finden Sie unter AWS IoT -Tutorials.

- Verwenden Sie eine unterstützte Umgebung

Für jede verwaltete Vorlage finden Sie Informationen über die Umgebung, in der Sie die Remote-Aktionen ausführen können. Wir empfehlen, dass Sie die Vorlage in einer unterstützten Linux-Umgebung verwenden, so wie das in der Vorlage angegeben ist. Verwenden Sie den AWS IoT Geräteclient, um die verwalteten Template-Remote-Aktionen auszuführen, da er gängige Mikroprozessoren und Linux-Umgebungen wie Debian und Ubuntu unterstützt.

Verwaltete Vorlagen für Remote-Aktionen und Auftragsdokumente

Der folgende Abschnitt listet die verschiedenen AWS verwalteten Vorlagen für AWS IoT Jobs auf und beschreibt die Fernaktionen, die auf den Geräten ausgeführt werden können. Der folgende Abschnitt enthält Informationen zum Auftragsdokument und eine Beschreibung der Parameter für das Auftragsdokument für jede Remote-Aktion. Ihre geräteseitige Software verwendet zum Durchführen der Remote-Aktion den Vorlagennamen und die Parameter.

AWS Verwaltete Vorlagen akzeptieren Eingabeparameter, für die Sie beim Erstellen eines Jobs mithilfe der Vorlage einen Wert angeben. Allen verwalteten Vorlagen haben zwei gemeinsame optionale Eingabeparameter: `runAsUser` und `pathToHandler`. Mit Ausnahme der `AWS-Reboot`-Vorlage benötigen die Vorlagen zusätzliche Eingabeparameter, für die Sie einen Wert angeben müssen, wenn Sie einen Auftrag mithilfe der Vorlage erstellen. Die erforderlichen Eingabeparameter

variieren je nach der von Ihnen ausgewählten Vorlage. Wenn Sie beispielsweise die `AWS-Download-File`-Vorlage auswählen, müssen Sie eine Liste der zu installierenden Pakete und eine URL angeben, von der Dateien heruntergeladen werden sollen.

Geben Sie einen Wert für die Eingabeparameter an, wenn Sie die AWS IoT Konsole verwenden, oder AWS Command Line Interface (AWS CLI), um einen Job zu erstellen, der eine verwaltete Vorlage verwendet. Wenn Sie die CLI verwenden, geben Sie diese Werte mithilfe des `document-parameters`-Objekts an. Weitere Informationen finden Sie unter documentParameters.

> **Note**
>
> Verwenden Sie `document-parameters` nur beim Erstellen von Aufträgen AWS -verwaltete Vorlagen. Dieser Parameter kann nicht mit benutzerdefinierten Auftragsvorlagen oder zum Erstellen von Aufträgen aus diesen Auftragsvorlagen verwendet werden.

Im Folgenden finden Sie eine Beschreibung der gängigen optionalen Eingabeparameter. Im nächsten Abschnitt finden Sie eine Beschreibung der anderen Eingabeparameter, die für jede verwaltete Vorlage erforderlich sind.

`runAsUser`

Dieser Parameter gibt an, ob der Auftragshandler unter einem anderen Benutzer ausgeführt werden soll. Wird er bei der Auftragserstellung nicht angegeben, wird der Auftragshandler unter demselben Benutzer wie der Device Client ausgeführt. Wenn Sie den Auftragshandler unter einem anderen Benutzer ausführen, geben Sie einen Zeichenkettenwert an, der nicht länger als 256 Zeichen ist.

`pathToHandler`

Der Pfad zum Auftragshandler, der auf dem Gerät ausgeführt wird. Wenn der Pfad bei der Auftragserstellung nicht angegeben wurde, verwendet der Device Client das aktuelle Arbeitsverzeichnis.

Im Folgenden werden die verschiedenen Remote-Aktionen, ihre Auftragsdokumente und die von ihnen akzeptierten Parameter dargestellt. Alle diese Vorlagen unterstützen die Linux-Umgebung für die Ausführung des Remote-Operation auf dem Gerät.

AWS–Download-Datei

Name der Vorlage

`AWS-Download-File`

Vorlagenbeschreibung

Eine verwaltete Vorlage, die von AWS zum Herunterladen einer Datei bereitgestellt wird.

Eingabeparameter

Diese Vorlage verfügt über die folgenden erforderlichen Parameter. Sie können auch die optionalen Parameter `runAsUser` und `pathToHandler` angeben.

`downloadUrl`

Die URL, von der die Datei heruntergeladen werden soll. Dies kann eine Internetressource sein, ein Objekt in Amazon S3, auf das öffentlich zugegriffen werden kann, oder ein Objekt in Amazon S3, auf das Ihr Gerät nur über eine vorsignierte URL zugreifen kann. Weitere Informationen zur Verwendung von vorsignierten URLs und zum Gewähren von Berechtigungen finden Sie unter Vorsignierte URLs.

`filePath`

Ein lokaler Dateipfad, der den Speicherort auf dem Gerät anzeigt, an dem die heruntergeladene Datei gespeichert wird.

Geräteverhalten

Das Gerät lädt die Datei vom angegebenen Speicherort herunter, überprüft, ob der Download abgeschlossen ist und speichert sie lokal.

Auftragsdokument

Im Folgenden werden das Auftragsdokument und die zugehörige neueste Version angezeigt. Die Vorlage zeigt den Pfad zum Auftragshandler und zum Shell-Skript `download-file.sh`, die der Auftragshandler ausführen muss, um die Datei herunterzuladen. Es zeigt auch die erforderlichen Parameter `downloadUrl` und `filePath` an.

```
{
  "version": "1.0",
  "steps": [
```

```
    {
      "action": {
        "name": "Download-File",
        "type": "runHandler",
        "input": {
          "handler": "download-file.sh",
          "args": [
            "${aws:iot:parameter:downloadUrl}",
            "${aws:iot:parameter:filePath}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS-Install-Application

Name der Vorlage

`AWS-Install-Application`

Vorlagenbeschreibung

Eine verwaltete Vorlage, die von AWS für die Installation einer oder mehrerer Anwendungen bereitgestellt wird.

Eingabeparameter

Diese Vorlage hat den folgenden erforderlichen Parameter, `packages`. Sie können auch die optionalen Parameter `runAsUser` und `pathToHandler` angeben.

`packages`

Eine durch Leerzeichen getrennte Liste mit einer oder mehreren Anwendungen, die installiert werden sollen.

Geräteverhalten

Das Gerät installiert die Anwendungen, wie im Auftragsdokument angegeben.

Auftragsdokument

Im Folgenden werden das Auftragsdokument und die zugehörige neueste Version angezeigt. Die Vorlage zeigt den Pfad zum Auftragshandler und zum Shell-Skript `install-packages.sh`, die der Auftragshandler ausführen muss, um die Datei herunterzuladen. Es zeigt auch den erforderlichen Parameter `packages` an.

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Install-Application",
        "type": "runHandler",
        "input": {
          "handler": "install-packages.sh",
          "args": [
            "${aws:iot:parameter:packages}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS-Neustart

Name der Vorlage

`AWS-Reboot`

Vorlagenbeschreibung

Eine verwaltete Vorlage, die von AWS für den Neustart Ihres Geräts bereitgestellt wird.

Eingabeparameter

Dieser Befehl hat keine erforderlichen Parameter. Sie können die optionalen Parameter `runAsUser` und `pathToHandler` angeben.

Geräteverhalten

Das Gerät wird erfolgreich neu gestartet.

Auftragsdokument

Im Folgenden werden das Auftragsdokument und die zugehörige neueste Version angezeigt. Die Vorlage zeigt den Pfad zum Auftragshandler und zum Shell-Skript `reboot.sh`, die der Auftragshandler ausführen muss, um das Gerät neu zu starten.

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Reboot",
        "type": "runHandler",
        "input": {
          "handler": "reboot.sh",
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS-Remove-Application

Name der Vorlage

`AWS-Remove-Application`

Vorlagenbeschreibung

Eine verwaltete Vorlage, die AWS für die Deinstallation einer oder mehrerer Anwendungen bereitgestellt wird.

Eingabeparameter

Diese Vorlage hat den folgenden erforderlichen Parameter, `packages`. Sie können auch die optionalen Parameter `runAsUser` und `pathToHandler` angeben.

`packages`

Eine durch Leerzeichen getrennte Liste mit einer oder mehreren Anwendungen, die deinstalliert werden sollen.

Geräteverhalten

Das Gerät deinstalliert die Anwendungen, wie im Jobdokument angegeben.

Auftragsdokument

Im Folgenden werden das Auftragsdokument und die zugehörige neueste Version angezeigt. Die Vorlage zeigt den Pfad zum Auftragshandler und zum Shell-Skript `remove-packages.sh`, die der Auftragshandler ausführen muss, um die Datei herunterzuladen. Es zeigt auch den erforderlichen Parameter `packages` an.

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Remove-Application",
        "type": "runHandler",
        "input": {
          "handler": "remove-packages.sh",
          "args": [
            "${aws:iot:parameter:packages}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS–Restart–Application

Name der Vorlage

`AWS-Restart-Application`

Vorlagenbeschreibung

Eine verwaltete Vorlage, die von AWS zum Beenden und Neustarten eines oder mehrerer Dienste bereitgestellt wird.

Eingabeparameter

Diese Vorlage hat den folgenden erforderlichen Parameter, `services`. Sie können auch die optionalen Parameter `runAsUser` und `pathToHandler` angeben.

Dienstleistungen

Eine durch Leerzeichen getrennte Liste mit einer oder mehreren Anwendungen, die neu gestartet werden sollen.

Geräteverhalten

Die angegebenen Anwendungen werden gestoppt und dann auf dem Gerät neu gestartet.

Auftragsdokument

Im Folgenden werden das Auftragsdokument und die zugehörige neueste Version angezeigt. Die Vorlage zeigt den Pfad zum Auftragshandler und zum Shell-Skript `restart-services.sh`, die der Auftragshandler ausführen muss, um die Systemdienste neu zu starten. Es zeigt auch den erforderlichen Parameter `services` an.

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Restart-Application",
        "type": "runHandler",
        "input": {
          "handler": "restart-services.sh",
          "args": [
            "${aws:iot:parameter:services}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS-Start-Application

Name der Vorlage

`AWS-Start-Application`

Vorlagenbeschreibung

Eine verwaltete Vorlage, AWS die von zum Starten eines oder mehrerer Dienste bereitgestellt wird.

Eingabeparameter

Diese Vorlage hat den folgenden erforderlichen Parameter, `services`. Sie können auch die optionalen Parameter `runAsUser` und `pathToHandler` angeben.

`services`

Eine durch Leerzeichen getrennte Liste mit einer oder mehreren Anwendungen, die gestartet werden sollen.

Geräteverhalten

Die angegebenen Anwendungen werden auf dem Gerät ausgeführt.

Auftragsdokument

Im Folgenden werden das Auftragsdokument und die zugehörige neueste Version angezeigt. Die Vorlage zeigt den Pfad zum Auftragshandler und zum Shell-Skript `start-services.sh`, die der Auftragshandler ausführen muss, um die Systemdienste zu starten. Es zeigt auch den erforderlichen Parameter `services` an.

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Start-Application",
        "type": "runHandler",
        "input": {
          "handler": "start-services.sh",
          "args": [
            "${aws:iot:parameter:services}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
```

```
  ]
}
```

AWS-Stop-Application

Name der Vorlage

`AWS-Stop-Application`

Vorlagenbeschreibung

Eine verwaltete Vorlage, die von AWS zum Beenden eines oder mehrerer Dienste bereitgestellt wird.

Eingabeparameter

Diese Vorlage hat den folgenden erforderlichen Parameter, `services`. Sie können auch die optionalen Parameter `runAsUser` und `pathToHandler` angeben.

`services`

Eine durch Leerzeichen getrennte Liste mit einer oder mehreren Anwendungen, die beendet werden sollen.

Geräteverhalten

Die angegebenen Anwendungen werden auf dem Gerät nicht mehr ausgeführt.

Auftragsdokument

Im Folgenden werden das Auftragsdokument und die zugehörige neueste Version angezeigt. Die Vorlage zeigt den Pfad zum Auftragshandler und zum Shell-Skript `stop-services.sh`, die der Auftragshandler ausführen muss, um die Systemservices zu beenden. Es zeigt auch den erforderlichen Parameter `services` an.

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Stop-Application",
        "type": "runHandler",
        "input": {
          "handler": "stop-services.sh",
          "args": [
```

```
            "${aws:iot:parameter:services}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS-Run-Command

Name der Vorlage

`AWS-Run-Command`

Vorlagenbeschreibung

Eine verwaltete Vorlage, die von AWS für die Ausführung eines Shell-Befehls bereitgestellt wird.

Eingabeparameter

Diese Vorlage hat den folgenden erforderlichen Parameter, `command`. Sie können auch den anderen optionalen Parameter `runAsUser` angeben.

`command`

Eine durch Kommas getrennte Befehlsfolge. Jedes Komma, das im Befehl selbst enthalten ist, muss maskiert werden.

Geräteverhalten

Das Gerät führt den Shell-Befehl wie im Auftragsdokument angegeben aus.

Auftragsdokument

Im Folgenden werden das Auftragsdokument und die zugehörige neueste Version angezeigt. Die Vorlage zeigt den Pfad zum Auftragsbefehl und den von Ihnen angegebenen Befehl, der auf dem Gerät ausgeführt werden soll.

```
{
  "version": "1.0",
  "steps": [
    {
```

```
      "action": {
        "name": "Run-Command",
        "type": "runCommand",
        "input": {
          "command": "${aws:iot:parameter:command}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

Themen

- Erstellen Sie einen Job von AWS verwaltete Vorlagen mithilfe der AWS-Managementkonsole
- Erstellen Sie einen Job von AWS verwaltete Vorlagen mithilfe der AWS CLI

Erstellen Sie einen Job von AWS verwaltete Vorlagen mithilfe der AWS-Managementkonsole

Verwenden Sie die AWS-Managementkonsole , um Informationen über AWS verwaltete Vorlagen abzurufen und mithilfe dieser Vorlagen einen Job zu erstellen. Anschließend können Sie den von Ihnen erstellten Auftrag als Ihre eigene benutzerdefinierte Vorlage speichern.

Abrufen von Details von verwalteten Vorlagen

Informationen zu den verschiedenen verwalteten Vorlagen, die zur Verwendung zur Verfügung stehen, finden Sie in der AWS IoT Konsole.

1. Um Ihre verfügbaren verwalteten Vorlagen zu sehen, wechseln Sie in der AWS IoT Konsole zum Hub für Jobvorlagen und wählen Sie den Tab Verwaltete Vorlagen.
2. Zum Anzeigen von Details wählen Sie eine verwaltete Vorlage aus.

Die Seite Details enthält die folgenden Informationen:

- Name, Beschreibung und Amazon-Ressourcenname (ARN) der verwalteten Vorlage.
- Die Umgebung, in der die Remote-Operationen ausgeführt werden können, z. B. Linux.
- Das JSON-Auftragsdokument, das den Pfad zum Auftragshandler und zu den Befehlen angibt, die auf dem Gerät ausgeführt werden sollen. Im Folgenden wird beispielsweise ein Beispiel

für ein Jobdokument für die AWS-Reboot Vorlage gezeigt. Die Vorlage zeigt den Pfad zum Auftragshandler und zum Shell-Skript `reboot.sh`, die der Auftragshandler ausführen muss, um das Gerät neu zu starten.

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Reboot",
        "type": "runHandler",
        "input": {
          "handler": "reboot.sh",
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

Weitere Informationen zum Auftragsdokument und den zugehörigen Parametern für verschiedene Remote-Aktionen finden Sie unter Verwaltete Vorlagen für Remote-Aktionen und Auftragsdokumente.

- Die neueste Version des Auftragsdokuments.

Erstellen Sie einen Auftrag mit verwalteten Vorlagen

Sie können die AWS Managementkonsole verwenden, um eine AWS verwaltete Vorlage auszuwählen, die Sie zum Erstellen eines Jobs verwenden möchten. In diesem Abschnitt erfahren Sie mehr darüber.

Sie können auch den Workflow zur Auftragserstellung starten und dann die AWS verwaltete Vorlage auswählen, die Sie beim Erstellen des Jobs verwenden möchten. Weitere Informationen zu diesem Workflow finden Sie unter Erstellen und verwalten Sie Jobs mithilfe der AWS-Managementkonsole.

1. Wählen Sie Ihre AWS verwaltete Vorlage

   Gehen Sie in der AWS IoT Konsole zum Hub für Jobvorlagen, wählen Sie den Tab Verwaltete Vorlagen und wählen Sie dann Ihre Vorlage aus.

2. Erstellen Sie einen Auftrag mit Ihrer verwalteten Vorlage

    1. Wählen Sie auf der Seite Details Ihrer Vorlage die Option Auftrag erstellen aus.

        Die Konsole wechselt zum Schritt Benutzerdefinierte Auftragseigenschaften des Workflows Auftrag erstellen, in dem Ihre Vorlagenkonfiguration hinzugefügt wurde.

    2. Geben Sie einen eindeutigen alphanumerischen Auftragsnamen sowie optional eine Beschreibung und Tags ein und wählen Sie dann Weiter aus.

    3. Wählen Sie die Objekte oder Objektgruppen als Auftragsziele aus, die Sie in diesem Auftrag ausführen möchten.

    4. Im Bereich Auftragsdokument wird Ihre Vorlage mit ihren Konfigurationseinstellungen und Eingabeparametern angezeigt. Geben Sie Werte für die Eingabeparameter in die von Ihnen ausgewählten Vorlage ein. Wenn Sie beispielsweise die Download-File Vorlage AWS-ausgewählt haben:

        - Geben Sie für DownloadUrl die URL der Datei ein, die heruntergeladen werden soll, zum Beispiel: `https://example.com/index.html`.
        - Geben Sie für FilePath den Pfad auf dem Gerät ein, in dem die heruntergeladene Datei gespeichert werden soll, zum Beispiel: *`path/to/file`*.

        Sie können optional auch Werte für die Parameter `runAsUser` und `pathToHandler` eingeben. Weitere Informationen zu den Eingabeparametern für jede Vorlage finden Sie unter Verwaltete Vorlagen für Remote-Aktionen und Auftragsdokumente.

    5. Wählen Sie auf der Seite Auftragskonfiguration den Auftragstyp als kontinuierlichen Auftrag oder Snapshot-Auftrag aus. Ein Snapshot-Auftrag ist abgeschlossen, wenn er seine Ausführung auf den Zielgeräten und Zielgruppen abgeschlossen hat. Ein kontinuierlicher Auftrag gilt für Objektgruppen und wird auf jedem Gerät ausgeführt, das Sie einer bestimmten Zielgruppe hinzufügen.

    6. Fügen Sie weitere Konfigurationen für Ihren Auftrag hinzu und überprüfen und erstellen Sie dann Ihren Auftrag. Weitere Informationen zum Arbeiten mit zusätzlichen Konfigurationen finden Sie unter:

        - Konfigurationen für Auftrags-Rollout, Planung und Abbruch
        - Timeout bei der Auftragsausführung und Wiederholungskonfigurationen

Erstellen Sie benutzerdefinierte Jobvorlagen aus verwalteten Vorlagen

Sie können eine AWS verwaltete Vorlage und einen benutzerdefinierten Job als Ausgangspunkt verwenden, um Ihre eigene benutzerdefinierte Auftragsvorlage zu erstellen. Um eine benutzerdefinierte Auftragsvorlage zu erstellen, erstellen Sie zunächst einen Job aus Ihrer AWS verwalteten Vorlage, wie im vorherigen Abschnitt beschrieben.

Anschließend können Sie den benutzerdefinierten Auftrag als Vorlage speichern, um Ihre eigene benutzerdefinierte Auftragsvorlage zu erstellen. Zum Speichern als Vorlage:

1. Gehen Sie zum Job-Hub der AWS IoT Konsole und wählen Sie den Job aus, der Ihre verwaltete Vorlage enthält.
2. Wählen Sie die Option Als Auftragsvorlage speichern und erstellen Sie dann Ihre benutzerdefinierte Auftragsvorlage. Weitere Informationen zum Erstellen einer benutzerdefinierten Auftragsvorlage finden Sie unter Erstellen einer Auftragsvorlage anhand eines vorhandenen Auftrags.

Erstellen Sie einen Job von AWS verwaltete Vorlagen mithilfe der AWS CLI

Verwenden Sie die AWS CLI , um Informationen über AWS verwaltete Vorlagen abzurufen und mithilfe dieser Vorlagen einen Job zu erstellen. Anschließend können Sie den Auftrag als Vorlage speichern und anschließend Ihre eigene benutzerdefinierte Vorlage erstellen.

Auflisten von verwalteten Vorlagen

Der `list-managed-job-templates` AWS CLI Befehl listet alle Jobvorlagen in Ihrem auf AWS-Konto.

```
aws iot list-managed-job-templates
```

Wenn Sie diesen Befehl ausführen, werden standardmäßig alle verfügbaren AWS verwalteten Vorlagen und ihre Details angezeigt.

```
{
    "managedJobTemplates": [
        {
            "templateArn": "arn:aws:iot:region::jobtemplate/AWS-Reboot:1.0",
            "templateName": "AWS-Reboot",
```

```
            "description": "A managed job template for rebooting the device.",
            "environments": [
                "LINUX"
            ],
            "templateVersion": "1.0"
        },
        {
            "templateArn": "arn:aws:iot:region::jobtemplate/AWS-Remove-
Application:1.0",
            "templateName": "AWS-Remove-Application",
            "description": "A managed job template for uninstalling one or more
 applications.",
            "environments": [
                "LINUX"
            ],
            "templateVersion": "1.0"
        },
        {
            "templateArn": "arn:aws:iot:region::jobtemplate/AWS-Stop-Application:1.0",
            "templateName": "AWS-Stop-Application",
            "description": "A managed job template for stopping one or more system
 services.",
            "environments": [
                "LINUX"
            ],
            "templateVersion": "1.0"
        },

        ...

        {
            "templateArn": "arn:aws:iot:us-east-1::jobtemplate/AWS-Restart-
Application:1.0",
            "templateName": "AWS-Restart-Application",
            "description": "A managed job template for restarting one or more system
 services.",
            "environments": [
                "LINUX"
            ],
            "templateVersion": "1.0"
        }
    ]
}
```

Weitere Informationen finden Sie unter ListManagedJobTemplates.

Abrufen von Details über eine verwaltete Vorlage

`describe-managed-job-template` AWS CLI Mit dem Befehl werden Details zu einer angegebenen Auftragsvorlage abgerufen. Geben Sie den Namen der Auftragsvorlage und eine optionale Vorlagenversion an. Wenn die Vorlagenversion nicht angegeben ist, wird die vordefinierte Standardversion zurückgegeben. Um beispielsweise Details über die `AWS-Download-File`-Vorlage zu sehen, führen Sie den folgenden Befehl aus.

```
aws iot describe-managed-job-template \
    --template-name AWS-Download-File
```

Der Befehl zeigt die Vorlagendetails und den ARN, das zugehörige Auftragsdokument und den `documentParameters`-Parameter an, bei dem es sich um eine Liste von Schlüssel-Wert-Paaren von Eingabeparametern der Vorlage handelt. Weitere Informationen zu den verschiedenen Vorlagen und Eingabeparametern finden Sie unter Verwaltete Vorlagen für Remote-Aktionen und Auftragsdokumente.

> **Note**
>
> Das `documentParameters` Objekt, das zurückgegeben wird, wenn Sie diese API verwenden, darf nur verwendet werden, wenn Sie Jobs aus AWS verwalteten Vorlagen erstellen. Das Objekt darf nicht für benutzerdefinierte ‚Auftragsvorlagen verwendet werden. Ein Beispiel, das zeigt, wie dieser Parameter verwendet wird, finden Sie unter Erstellen eines Auftrags mithilfe verwalteter Vorlagen.

```
{
    "templateName": "AWS-Download-File",
    "templateArn": "arn:aws:iot:region::jobtemplate/AWS-Download-File:1.0",
    "description": "A managed job template for downloading a file.",
    "templateVersion": "1.0",
    "environments": [
        "LINUX"
    ],
    "documentParameters": [
        {
```

```
            "key": "downloadUrl",
            "description": "URL of file to download.",
            "regex": "(.*?)",
            "example": "http://www.example.com/index.html",
            "optional": false
        },
        {
            "key": "filePath",
            "description": "Path on the device where downloaded file is written.",
            "regex": "(.*?)",
            "example": "/path/to/file",
            "optional": false
        },
        {
            "key": "runAsUser",
            "description": "Execute handler as another user. If not specified, then
 handler is executed as the same user as device client.",
            "regex": "(.){0,256}",
            "example": "user1",
            "optional": true
        },
        {
            "key": "pathToHandler",
            "description": "Path to handler on the device. If not specified, then
 device client will use the current working directory.",
            "regex": "(.){0,4096}",
            "example": "/path/to/handler/script",
            "optional": true
        }
    ],
    "document": "{\"version\":\"1.0\",\"steps\":[{\"action\":{\"name
\":\"Download-File\",\"type\":\"runHandler\",\"input\":{\"handler\":
\"download-file.sh\",\"args\":[\"${aws:iot:parameter:downloadUrl}\",
\"${aws:iot:parameter:filePath}\"],\"path\":\"${aws:iot:parameter:pathToHandler}\"},
\"runAsUser\":\"${aws:iot:parameter:runAsUser}\"}}]}"
}
```

Weitere Informationen finden Sie unter DescribeManagedJobTemplate.

Erstellen eines Auftrags mithilfe verwalteter Vorlagen

Der `create-job` AWS CLI Befehl kann verwendet werden, um einen Job aus einer Jobvorlage zu erstellen. Es zielt auf ein Gerät mit dem Namen `thing0ne` und gibt den Amazon-Ressourcennamen

(ARN) der verwalteten Vorlage an, die als Grundlage für den Auftrag verwendet werden soll. Sie können erweiterte Konfigurationen wie Timeout- und Abbruchkonfigurationen überschreiben, indem Sie die zugehörigen Parameter des Befehls `create-job` übergeben.

Das Beispiel zeigt, wie ein Auftrag erstellt wird, der die Vorlage `AWS-Download-File` verwendet. Außerdem wird gezeigt, wie die Eingabeparameter der Vorlage mithilfe des Parameters `document-parameters` angegeben werden.

> **Note**
>
> Verwenden Sie das `document-parameters` Objekt nur mit AWS verwalteten Vorlagen. Dieses Objekt darf nicht mit benutzerdefinierten Auftragsvorlagen verwendet werden.

```
aws iot create-job \
   --targets arn:aws:iot:region:account-id:thing/thingOne \
   --job-id "new-managed-template-job" \
   --job-template-arn arn:aws:iot:region::jobtemplate/AWS-Download-File:1.0 \
   --document-parameters downloadUrl=https://example.com/index.html,filePath=path/to/
file
```

Wobei:

- *region*ist der AWS-Region.
- *account-id*ist die eindeutige AWS-Konto Zahl.
- *thingOne* ist der Name des IoT-Objekts, für das der Auftrag bestimmt ist.
- *AWS-Download-File:1.0*ist der Name der verwalteten Vorlage.
- `https://`*example.com/index*`.html` ist die URL, von der die Datei heruntergeladen werden soll.
- `https://`*pathto/file/index* ist der Pfad auf dem Gerät, in dem die heruntergeladene Datei gespeichert werden soll.

Führen Sie den folgenden Befehl aus, um einen Auftrag für die Vorlage *AWS-Download-File* zu erstellen.

```
{
    "jobArn": "arn:aws:iot:region:account-id:job/new-managed-template-job",
    "jobId": "new-managed-template-job",
```

```
    "description": "A managed job template for downloading a file."
}
```

Erstellen Sie eine benutzerdefinierte Auftragsvorlage aus verwalteten Vorlagen

1. Erstellen Sie einen Auftrag mit einer verwalteten Vorlage, wie im vorherigen Abschnitt beschrieben.
2. Erstellen Sie eine benutzerdefinierte Auftragsvorlage, indem Sie den ARN des Auftrags verwenden, den Sie erstellt haben. Weitere Informationen finden Sie unter Erstellen einer Auftragsvorlage anhand eines vorhandenen Auftrags.

Erstellen von benutzerdefinierten Auftragsvorlagen

Sie können Jobvorlagen mithilfe der AWS CLI und AWS IoT -Konsole erstellen. Sie können Jobs auch anhand von Jobvorlagen erstellen, indem Sie die Webanwendungen AWS CLI, die AWS IoT Konsole und Fleet Hub for AWS IoT Device Management verwenden. Weitere Informationen zum Arbeiten mit Jobvorlagen in Fleet Hub-Anwendungen finden Sie unter Arbeiten mit Jobvorlagen in Fleet Hub for AWS IoT Device Management.

> **Note**
>
> Die Gesamtzahl der Substitutionsmuster in einem Auftragsdokument sollte höchstens zehn betragen.

Themen

- Erstellen Sie benutzerdefinierte Jobvorlagen mithilfe der AWS-Managementkonsole
- Erstellen Sie benutzerdefinierte Jobvorlagen mithilfe der AWS CLI

Erstellen Sie benutzerdefinierte Jobvorlagen mithilfe der AWS-Managementkonsole

In diesem Thema wird erklärt, wie Sie Jobvorlagen mithilfe der AWS IoT Konsole erstellen, löschen und Details zu ihnen anzeigen.

Erstellen einer benutzerdefinierten Auftragsvorlage

Sie können entweder eine ursprüngliche benutzerdefinierte Auftragsvorlage oder eine Auftragsvorlage aus einem vorhandenen Auftrag erstellen. Sie können auch eine benutzerdefinierte

Jobvorlage aus einem vorhandenen Job erstellen, der mithilfe einer AWS verwalteten Vorlage erstellt wurde. Weitere Informationen finden Sie unter Erstellen Sie benutzerdefinierte Jobvorlagen aus verwalteten Vorlagen.

Erstellen einer ursprüngliche Auftragsvorlage

1. Beginnen mit der Erstellung Ihrer Jobvorlage

   1. Gehen Sie in der AWS IoT Konsole zum Hub „Jobvorlagen“ und wählen Sie die Registerkarte Benutzerdefinierte Vorlagen.
   2. Wählen Sie Auftragsvorlage erstellen.

   > **Note**
   >
   > Sie können auch von der Seite Verwandte Services unter Fleet Hub zur Seite Auftragsvorlagen navigieren.

2. Angeben der Eigenschaften der Auftragsvorlage

   Geben Sie auf der Seite Auftragsvorlage erstellen eine alphanumerische Kennung für Ihren Auftragsnamen und eine alphanumerische Beschreibung ein, um zusätzliche Informationen zur Vorlage bereitzustellen.

   > **Note**
   >
   > Es wird nicht empfohlen, für Ihre Auftrags-IDs oder Beschreibungen personenbezogene Informationen zu verwenden.

3. Angeben des Auftragsdokuments

   Stellen Sie eine JSON-Auftragsdatei bereit, die entweder in einem S3-Bucket oder als Inline-Auftragsdokument gespeichert ist, das im Auftrag angegeben ist. Diese Auftragsdatei wird zum Auftragsdokument, wenn Sie einen Auftrag mit dieser Vorlage erstellen.

   Wenn die Auftragsdatei in einem S3-Bucket gespeichert ist, geben Sie die S3-URL ein oder wählen Sie S3 durchsuchen, navigieren Sie dann zu Ihrem Auftragsdokument und wählen Sie es aus.

> **Note**
>
> Sie können nur S3-Buckets in Ihrer aktuellen Region auswählen.

4. Fügen Sie weitere Konfigurationen für Ihren Auftrag hinzu und überprüfen und erstellen Sie dann Ihren Auftrag. Weitere Informationen zu den zusätzlichen optionalen Konfigurationen finden Sie unter den folgenden Links:

   - Konfigurationen für Auftrags-Rollout, Planung und Abbruch
   - Timeout bei der Auftragsausführung und Wiederholungskonfigurationen

Erstellen einer Auftragsvorlage anhand eines vorhandenen Auftrags

1. Wählen Sie Ihren Schlüssel aus.

   1. Gehen Sie zum Job-Hub der AWS IoT Konsole und wählen Sie den Job aus, den Sie als Grundlage für Ihre Jobvorlage verwenden möchten.
   2. Wählen Sie Als Auftragsvorlage speichern aus.

> **Note**
>
> Optional können Sie ein anderes Auftragsdokument auswählen oder die erweiterten Konfigurationen des ursprünglichen Auftrags bearbeiten und dann die Option Jobvorlage erstellen auswählen. Ihre neue Auftragsvorlage wird auf der Seite Auftragsvorlagen angezeigt.

2. Angeben der Eigenschaften der Auftragsvorlage

   Geben Sie auf der Seite Auftragsvorlage erstellen eine alphanumerische Kennung für Ihren Auftragsnamen und eine alphanumerische Beschreibung ein, um zusätzliche Informationen zur Vorlage bereitzustellen.

> **Note**
>
> Das Auftragsdokument ist die Auftragsdatei, die Sie bei der Erstellung der Vorlage angegeben haben. Wenn das Auftragsdokument innerhalb des Auftrags und nicht an

einem S3-Speicherort angegeben ist, können Sie das Auftragsdokument auf der Seite Details für diesen Auftrag sehen.

3. Fügen Sie weitere Konfigurationen für Ihren Auftrag hinzu und überprüfen und erstellen Sie dann Ihren Auftrag. Weitere Informationen zum Arbeiten mit zusätzlichen Konfigurationen finden Sie unter:

   - Konfigurationen für Auftrags-Rollout, Planung und Abbruch
   - Timeout bei der Auftragsausführung und Wiederholungskonfigurationen

Erstellen eines Auftrags anhand einer benutzerdefinierten Auftragsvorlage

Sie können einen Auftrag anhand einer benutzerdefinierten Auftragsvorlage erstellen, indem Sie die Seite Details Ihrer Auftragsvorlage aufrufen, wie in diesem Thema beschrieben. Sie können auch einen Auftrag erstellen oder die Auftragsvorlage auswählen, die Sie bei der Ausführung des Workflows zur Auftragserstellung verwenden möchten. Weitere Informationen finden Sie unter Erstellen und verwalten Sie Jobs mithilfe der AWS-Managementkonsole.

In diesem Thema wird gezeigt, wie Sie einen Auftrag von der Seite Details einer benutzerdefinierten Auftragsvorlage aus erstellen. Sie können einen Job auch aus einer AWS verwalteten Vorlage erstellen. Weitere Informationen finden Sie unter Erstellen Sie einen Auftrag mit verwalteten Vorlagen.

1. Auswählen Ihrer benutzerdefinierten Auftragsvorlage

   Gehen Sie in der AWS IoT Konsole zum Hub „Jobvorlagen“ und wählen Sie die Registerkarte „Benutzerdefinierte Vorlagen“ und wählen Sie dann Ihre Vorlage aus.

2. Erstellen eines Auftrags mit Ihrer benutzerdefinierten Vorlage

   So erstellen Sie einen Auftrag:

   1. Wählen Sie auf der Seite Details Ihrer Vorlage die Option Auftrag erstellen aus.

      Die Konsole wechselt zum Schritt Benutzerdefinierte Auftragseigenschaften des Workflows Auftrag erstellen, in dem Ihre Vorlagenkonfiguration hinzugefügt wurde.

   2. Geben Sie einen eindeutigen alphanumerischen Auftragsnamen sowie optional eine Beschreibung und Tags ein und wählen Sie dann Weiter aus.

3. Wählen Sie die Objekte oder Objektgruppen als Auftragsziele aus, die Sie in diesem Auftrag ausführen möchten.

   Im Bereich Auftragsdokument wird Ihre Vorlage mit ihren Konfigurationseinstellungen angezeigt. Wenn Sie ein anderes Auftragsdokument verwenden möchten, wählen Sie Durchsuchen und wählen Sie einen anderen Bucket und ein anderes Dokument aus. Wählen Sie Weiter aus.
4. Wählen Sie auf der Seite Auftragskonfiguration den Auftragstyp als kontinuierlichen Auftrag oder Snapshot-Auftrag aus. Ein Snapshot-Auftrag ist abgeschlossen, wenn er seine Ausführung auf den Zielgeräten und Zielgruppen abgeschlossen hat. Ein kontinuierlicher Auftrag gilt für Objektgruppen und wird auf jedem Gerät ausgeführt, das Sie einer bestimmten Zielgruppe hinzufügen.
5. Fügen Sie weitere Konfigurationen für Ihren Auftrag hinzu und überprüfen und erstellen Sie dann Ihren Auftrag. Weitere Informationen zum Arbeiten mit zusätzlichen Konfigurationen finden Sie unter:
   - Konfigurationen für Auftrags-Rollout, Planung und Abbruch
   - Timeout bei der Auftragsausführung und Wiederholungskonfigurationen

> **Note**
>
> Wenn ein anhand einer Auftragsvorlage erstellter Auftrag die vorhandenen Parameter aktualisiert, die in der Auftragsvorlage bereitgestellt werden, überschreiben diese aktualisierten Parameter die vorhandenen Parameter, die in der Auftragsvorlage für diesen Auftrag bereitgestellt wurden.

Mit Fleet Hub-Webanwendungen können Sie Aufträge auch anhand von Auftragsvorlagen erstellen. Informationen zum Erstellen von Jobs in Fleet Hub finden Sie unter Arbeiten mit Jobvorlagen in Fleet Hub for AWS IoT Device Management.

Löschen einer Auftragsvorlage

Um eine Jobvorlage zu löschen, gehen Sie zunächst in der AWS IoT Konsole zum Hub für Jobvorlagen und wählen Sie den Tab Benutzerdefinierte Vorlagen. Wählen Sie die zu löschende Vorlage aus und klicken Sie auf Löschen.

> **Note**
>
> Eine Löschung ist dauerhaft und die Auftragsvorlage wird nicht mehr auf der Registerkarte Benutzerdefinierte Vorlagen angezeigt.

Erstellen Sie benutzerdefinierte Jobvorlagen mithilfe der AWS CLI

In diesem Thema wird erklärt, wie Sie Details erstellen und löschen können und wie Sie Details zu Auftragsvorlagen mithilfe von AWS CLI abrufen können.

Erstellen einer von Grund auf neuen Auftragsvorlage

Der folgende AWS CLI Befehl zeigt, wie Sie einen Job mithilfe eines in einem S3-Bucket gespeicherten Jobdokuments (*job-document.json*) und einer Rolle mit der Berechtigung zum Herunterladen von Dateien von Amazon S3 (*S3DownloadRole*) erstellen.

```
aws iot create-job-template  \
      --job-template-id 010  \
      --description "My custom job template for updating the device firmware"
      --document-source https://s3.amazonaws.com/amzn-s3-demo-bucket/job-document.json
 \
      --timeout-config inProgressTimeoutInMinutes=100 \
      --job-executions-rollout-config "{ \"exponentialRate\": { \"baseRatePerMinute\":
 50, \"incrementFactor\": 2, \"rateIncreaseCriteria\": { \"numberOfNotifiedThings\":
 1000, \"numberOfSucceededThings\": 1000}}, \"maximumPerMinute\": 1000}" \
      --abort-config "{ \"criteriaList\": [ { \"action\": \"CANCEL\", \"failureType
\": \"FAILED\", \"minNumberOfExecutedThings\": 100, \"thresholdPercentage\": 20},
 { \"action\": \"CANCEL\", \"failureType\": \"TIMED_OUT\", \"minNumberOfExecutedThings
\": 200, \"thresholdPercentage\": 50}]}" \
      --presigned-url-config "{\"roleArn\":\"arn:aws:iam::123456789012:role/
S3DownloadRole\", \"expiresInSec\":3600}"
```

Der optionale Parameter `timeout-config` gibt die Dauer an, die jedes Gerät für den Abschluss der Ausführung des Auftrags hat. Der Timer wird gestartet, wenn der Status der Auftragsausführung auf `IN_PROGRESS` gesetzt wird. Wird der Status der Auftragsausführung vor Ablauf der Zeit nicht auf einen anderen abschließenden Status festgelegt, wird er automatisch auf `TIMED_OUT` festgelegt.

Der Timer „In Bearbeitung“ kann nicht aktualisiert werden und gilt für alle Auftragsausführungen für den Auftrag. Immer wenn ein Jobstart länger als dieses Intervall im `IN_PROGRESS` Status verbleibt, schlägt der Jobstart fehl und wechselt in den `TIMED_OUT` Terminalstatus. AWS IoT veröffentlicht auch eine MQTT-Benachrichtigung.

Weitere Informationen zum Konfigurieren von Auftragsrollouts und Auftragsabbrüchen finden Sie unter Auftragsrollout- und Abbruchkonfiguration.

> **Note**
>
> Auftragsdokumente, die als Amazon-S3-Dateien angegeben sind, werden zum Zeitpunkt der Erstellung des Auftrags abgerufen. Änderungen der Inhalte der Amazon-S3-Datei, die als Quelle Ihres Auftragsdokuments verwendet werden, nachdem Sie den Auftrag erstellt haben, ändern nicht, was an die Ziele des Auftrags gesendet wird.

Erstellen einer Auftragsvorlage anhand eines vorhandenen Auftrags

Mit dem folgenden AWS CLI Befehl wird eine Jobvorlage erstellt, indem der Amazon-Ressourcenname (ARN) eines vorhandenen Jobs angegeben wird. Die neue Auftragsvorlage verwendet alle im Auftrag angegebenen Konfigurationen. Optional können Sie jede der Konfigurationen im vorhandenen Auftrag ändern, indem Sie einen der optionalen Parameter verwenden.

```
aws iot create-job-template  \
      --job-arn arn:aws:iot:region:123456789012:job/job-name  \
      --timeout-config inProgressTimeoutInMinutes=100
```

Abrufen von Details einer Aufgabenvorlage

Mit dem folgenden AWS CLI Befehl werden Details zu einer angegebenen Jobvorlage abgerufen.

```
aws iot describe-job-template \
      --job-template-id template-id
```

Die Ausgabe des Befehls sieht wie folgt aus:

```
{
   "abortConfig": {
      "criteriaList": [
         {
            "action": "string",
            "failureType": "string",
            "minNumberOfExecutedThings": number,
            "thresholdPercentage": number
         }
      ]
   },
   "createdAt": number,
   "description": "string",
   "document": "string",
   "documentSource": "string",
   "jobExecutionsRolloutConfig": {
      "exponentialRate": {
         "baseRatePerMinute": number,
         "incrementFactor": number,
         "rateIncreaseCriteria": {
            "numberOfNotifiedThings": number,
            "numberOfSucceededThings": number
         }
      },
      "maximumPerMinute": number
   },
   "jobTemplateArn": "string",
   "jobTemplateId": "string",
   "presignedUrlConfig": {
      "expiresInSec": number,
      "roleArn": "string"
   },
   "timeoutConfig": {
      "inProgressTimeoutInMinutes": number
   }
}
```

Auflisten der Auftragsvorlagen

Der folgende AWS CLI Befehl listet alle Jobvorlagen in Ihrem auf AWS-Konto.

```
aws iot list-job-templates
```

Die Ausgabe des Befehls sieht wie folgt aus:

```
{
   "jobTemplates": [
      {
         "createdAt": number,
         "description": "string",
         "jobTemplateArn": "string",
         "jobTemplateId": "string"
      }
   ],
   "nextToken": "string"
}
```

Verwenden Sie den Wert des Feldes `nextToken`, um weitere Ergebnisseiten abzurufen.

Löschen einer Auftragsvorlage

Mit dem folgenden AWS CLI Befehl wird eine angegebene Jobvorlage gelöscht.

```
aws iot delete-job-template \
      --job-template-id template-id
```

Der Befehl zeigt keine Ausgabe an.

Erstellen eines Auftrags anhand einer benutzerdefinierten Auftragsvorlage

Mit dem folgenden AWS CLI Befehl wird ein Auftrag aus einer benutzerdefinierten Auftragsvorlage erstellt. Es zielt auf ein Gerät mit dem Namen `thing0ne` und gibt den Amazon-Ressourcennamen (ARN) der Aufgabenvorlage an, die als Grundlage für den Auftrag verwendet werden soll. Sie können erweiterte Konfigurationen wie Timeout- und Abbruchkonfigurationen überschreiben, indem Sie die zugehörigen Parameter des Befehls `create-job` übergeben.

> **Warning**
>
> Das Objekt `document-parameters` darf nur zusammen mit dem Befehl `create-job` verwendet werden, wenn Aufträge aus AWS -verwalteten Vorlagen erstellt werden. Dieses Objekt darf nicht mit benutzerdefinierten Auftragsvorlagen verwendet werden. Ein Beispiel, das zeigt, wie Aufträge mit diesem Parameter erstellt werden, finden Sie unter Erstellen eines Auftrags mithilfe verwalteter Vorlagen.

```
aws iot create-job \
      --targets arn:aws:iot:region:123456789012:thing/thingOne  \
      --job-template-arn arn:aws:iot:region:123456789012:jobtemplate/template-id
```

Aufgabe configurations

Sie können für jeden Auftrag, den Sie für die angegebenen Ziele bereitstellen, die folgenden zusätzlichen Konfigurationen verwenden.

- Rollout: Definiert, wie viele Geräte das Auftragsdokument pro Minute erhalten.
- Planung: Plant einen Auftrag für ein future Datum und eine zukünftige Uhrzeit und verwendet zusätzlich wiederkehrende Wartungsfenster.
- Abbrechen: Bricht einen Auftrag ab, z. B. wenn einige Geräte die Auftragsbenachrichtigung nicht erhalten oder wenn Ihre Geräte bei der Auftragsausführung einen Fehler melden.
- Timeout: Wenn Ihre Auftragsziele innerhalb einer bestimmten Zeit nach Beginn der Auftragsausführung keine Antwort erhalten, kann der Auftrag fehlschlagen.
- Erneut versuchen: Wiederholt die Auftragsausführung, wenn Ihr Gerät beim Versuch, eine Auftragsausführung abzuschließen, einen Fehler meldet oder wenn bei der Auftragsausführung ein Timeout auftritt.

Mithilfe dieser Konfigurationen können Sie den Status Ihrer Auftragsausführung überwachen und verhindern, dass ein fehlerhaftes Update an die gesamte Flotte gesendet wird.

Themen

- Wie funktionieren Auftragskonfigurationen
- Zusätzliche Konfigurationen angeben

Wie funktionieren Auftragskonfigurationen

Sie verwenden die Rollout- und Abbruchkonfigurationen, wenn Sie einen Auftrag bereitstellen, und die Timeout- und Wiederholungskonfigurationen für die Auftragsausführung. In den folgenden Abschnitten finden Sie weitere Informationen zur Funktionsweise dieser Konfigurationen.

Themen

- Konfigurationen für Auftrags-Rollout, Planung und Abbruch
- Timeout bei der Auftragsausführung und Wiederholungskonfigurationen

Konfigurationen für Auftrags-Rollout, Planung und Abbruch

Mithilfe der Konfigurationen für Auftrags-Rollout, Planung und Abbruch können Sie festlegen, wie viele Geräte das Auftragsdokument erhalten, einen Auftrags-Rollout planen und die Kriterien für das Abbrechen eines Auftrags festlegen.

Konfiguration des Auftrags-Rollouts

Sie können angeben, wie schnell die Ziele über eine ausstehende Auftragsausführung benachrichtigt werden. Sie können auch ein gestaffeltes Rollout erstellen, um Updates, Neustarts und andere Vorgänge zu verwalten. Um festzulegen, wie Ihre Ziele benachrichtigt werden, verwenden Sie die Auftrags-Rollout-Raten.

Auftragsrolloutraten

Sie können eine Rollout-Konfiguration erstellen, indem Sie entweder eine konstante Rollout-Rate oder eine exponentielle Rollout-Rate verwenden. Verwenden Sie eine konstante Rollout-Rate, um die maximale Anzahl von Auftragszielen festzulegen, die pro Minute informiert werden sollen.

AWS IoT Jobs können mithilfe exponentieller Rollout-Raten bereitgestellt werden, wenn verschiedene Kriterien und Schwellenwerte erfüllt werden. Wenn die Anzahl der fehlgeschlagenen Aufträge einer Reihe von Kriterien entspricht, die Sie angeben, können Sie den Auftrags-Rollout abbrechen. Sie legen die Kriterien für die Auftrags-Rollout-Rate fest, wenn Sie mithilfe des `JobExecutionsRolloutConfig`-Objekts einen Auftrag erstellen. Sie legen auch die Kriterien für den Auftragsabbruch bei der Auftragserstellung fest, indem Sie das `AbortConfig`-Objekt verwenden.

Das folgende Beispiel zeigt, wie die Rollout-Raten funktionieren. Beispielsweise würde ein Auftrags-Rollout mit einer Basisrate von 50 pro Minute, einem Inkrementfaktor von 2 und einer Anzahl von

jeweils 1.000 Geräten, die benachrichtigt und erfolgreich waren, wie folgt funktionieren: Der Auftrag beginnt mit einer Geschwindigkeit von 50 Auftragsausführungen pro Minute und wird mit dieser Geschwindigkeit fortgesetzt, bis entweder 1.000 Objekte Benachrichtigungen zur Auftragsausführung erhalten haben oder 1.000 erfolgreiche Auftragsausführungen stattgefunden haben.

Die folgende Tabelle zeigt, wie der Rollout über die ersten vier Inkremente verläuft.

Rolloutrate pro Minute	50	100	200	400
Anzahl der benachrichtigten Geräte oder der erfolgreichen Auftragsausführungen, um einer erhöhten Rate gerecht zu werden	1.000	2.000	3,000	4.000

> **Note**
>
> Wenn Sie Ihr maximales Limit von 500 Aufträgen (`isConcurrent = True`) erreicht haben, behalten alle aktiven Aufträge den Status `IN-PROGRESS` und führen keine neuen Auftragsausführungen durch, bis die Anzahl der gleichzeitigen Aufträge 499 oder weniger beträgt (`isConcurrent = False`). Dies gilt für Snapshots und kontinuierliche Aufträge. Falls `isConcurrent = True`, führt der Auftrag derzeit Auftragsausführungen auf allen Geräten Ihrer Zielgruppe durch. Falls `isConcurrent = False`, hat der Auftrag den Rollout aller Auftragsausführungen auf allen Geräten in Ihrer Zielgruppe abgeschlossen. Der Status wird aktualisiert, sobald alle Geräte in Ihrer Zielgruppe einen Terminal-Zustand erreicht haben, oder wenn Sie eine Konfiguration für den Auftragsabbruch ausgewählt haben, einen bestimmten Schwellenwert für Ihre Zielgruppe erreicht haben. Der Status auf Auftragsebene `isConcurrent = False` gibt für `isConcurrent = True` und beide `IN_PROGRESS` an. Weitere Informationen zu den Limits für aktive und gleichzeitige Aufträge finden Sie unter Limits für aktive und gleichzeitige Aufträge.

Auftrags-Rollout-Raten für kontinuierliche Aufträge mit dynamischen Objektgruppen

Wenn Sie einen kontinuierlichen Job verwenden, um den Fernbetrieb in Ihrer Flotte einzuführen, führt AWS IoT Jobs die Ausführung von Aufträgen für Geräte in Ihrer Zielgruppe durch.
Bei neuen Geräten, die der dynamischen Objektgruppe hinzugefügt werden, werden diese

Auftragsausführungen auch nach der Erstellung des Auftrags weiterhin auf diesen Geräten ausgeführt.

Mit der Rollout-Konfiguration können die Rollout-Raten nur für Geräte gesteuert werden, die der Gruppe hinzugefügt werden, bis der Auftrag erstellt wird. Nachdem ein Auftrag erstellt wurde, werden die Auftragsausführungen für alle neuen Geräte nahezu in Echtzeit erstellt, sobald die Geräte der Zielgruppe beitreten.

Konfiguration der Auftragsplanung

Sie können einen kontinuierlichen Auftrag oder einen Snapshot-Auftrag bis zu einem Jahr im Voraus planen und dabei eine vorher festgelegte Startzeit, Endzeit und ein Endverhalten angeben, was mit jeder Auftragsausführung nach Erreichen der Endzeit geschehen soll. Darüber hinaus können Sie ein optionales wiederkehrendes Wartungsfenster mit flexibler Häufigkeit, Startzeit und Dauer für fortlaufende Aufträge einrichten, um ein Auftragsdokument auf allen Geräten innerhalb der Zielgruppe bereitzustellen.

Konfigurationen für die Auftragsplanung

Startzeit

Die Startzeit eines geplanten Auftrags ist das zukünftige Datum und die Uhrzeit, zu der der Auftrag mit der Bereitstellung des Auftragsdokuments auf allen Geräten in der Zielgruppe beginnt. Die Startzeit für einen geplanten Auftrag gilt für fortlaufende Aufträge und Snapshot-Aufträge. Wenn ein geplanter Auftrag zum ersten Mal erstellt wird, behält er den Status `SCHEDULED`. Sobald das von Ihnen gewählte `startTime` erreicht ist, wird es aktualisiert auf `IN_PROGRESS` und der Rollout des Auftragsdokuments wird gestartet. Der Zeitraum zwischen dem ursprünglichen Datum und der Uhrzeit, zu der Sie den geplanten Auftrag erstellt haben, `startTime` darf nicht länger als ein Jahr sein.

Weitere Informationen zur Syntax für die `startTime` Verwendung eines API-Befehls oder des AWS CLI finden Sie unter https://docs.aws.amazon.com/cli/latest/userguide/cli-usage-parameters-types.html#parameter-type-timestamp Timestamp.

Bei einem Auftrag mit der optionalen Planungskonfiguration, der während eines wiederkehrenden Wartungsfensters an einem Ort mit Sommerzeit ausgeführt wird, ändert sich die Uhrzeit um eine Stunde, wenn von Sommerzeit auf Normalzeit und von Normalzeit auf Sommerzeit umgestellt wird.

> **Note**
>
> Die in der angezeigte Zeitzone AWS-Managementkonsole ist Ihre aktuelle Systemzeitzone. Diese Zeitzonen werden jedoch im System in UTC umgerechnet.

Endzeit

Die Endzeit eines geplanten Auftrags ist das zukünftige Datum und die Uhrzeit, zu der der Auftrag das Rollout des Auftragsdokuments an alle verbleibenden Geräte in der Zielgruppe beendet. Die Endzeit für einen geplanten Auftrag gilt für fortlaufende Aufträge und Snapshot-Aufträge. Wenn ein geplanter Auftrag beim ausgewählten `endTime` eintrifft und alle Auftragsausführungen einen Terminal-Zustand erreicht haben, wird sein Status von `IN_PROGRESS` auf `COMPLETED` aktualisiert. Der Zeitraum zwischen dem ursprünglichen Datum und der Uhrzeit, zu der Sie den geplanten Auftrag erstellt haben, `endTime` darf nicht länger als ein Jahr sein. Die Mindestdauer zwischen `startTime` und `endTime` beträgt 30 Minuten. Wiederholungsversuche bei der Auftragsausführung werden durchgeführt, bis der Auftrag den `endTime` erreicht, dann bestimmt der `endBehavior`, wie weiter vorzugehen ist.

Weitere Informationen zur Syntax für die `endTime` Verwendung eines API-Befehls oder des AWS CLI finden Sie unter Timestamp.

Bei einem Auftrag mit der optionalen Planungskonfiguration, der während eines wiederkehrenden Wartungsfensters an einem Ort mit Sommerzeit ausgeführt wird, ändert sich die Uhrzeit um eine Stunde, wenn von Sommerzeit auf Normalzeit und von Normalzeit auf Sommerzeit umgestellt wird.

> **Note**
>
> Die in der angezeigte Zeitzone AWS-Managementkonsole ist Ihre aktuelle Systemzeitzone. Diese Zeitzonen werden jedoch im System in UTC umgerechnet.

Verhalten beenden

Das Endverhalten eines geplanten Auftrags bestimmt, was mit dem Auftrag und allen noch nicht abgeschlossenen Auftragsausführungen passiert, wenn der Auftrag den ausgewählten `endTime` erreicht.

Im Folgenden sind die Endverhaltensweisen aufgeführt, aus denen Sie bei der Erstellung des Auftrags oder der Auftragsvorlage wählen können:

- `STOP_ROLLOUT`
  - `STOP_ROLLOUT` stoppt den Rollout des Auftragsdokuments auf allen verbleibenden Geräten in der Zielgruppe für den Auftrag. Darüber hinaus werden alle `QUEUED`- und `IN_PROGRESS`-Aufträge so lange ausgeführt, bis sie den Terminal-Zustand erreicht haben. Dies ist das standardmäßige Endverhalten, sofern Sie nicht `CANCEL` oder `FORCE_CANCEL` auswählen.
- `CANCEL`
  - `CANCEL` stoppt den Rollout des Auftragsdokuments auf allen verbleibenden Geräten in der Zielgruppe für den Auftrag. Darüber hinaus werden alle `QUEUED` Auftragsausführungen abgebrochen, während alle `IN_PROGRESS` Auftragsausführungen fortgesetzt werden, bis sie einen Terminal-Zustand erreichen.
- `FORCE_CANCEL`
  - `FORCE_CANCEL` stoppt den Rollout des Auftragsdokuments auf allen verbleibenden Geräten in der Zielgruppe für den Auftrag. Darüber hinaus werden alle `QUEUED`- und `IN_PROGRESS`-Auftragsausführungen storniert.

> **Note**
>
> Um eine auszuwählen`endbehavior`, müssen Sie eine auswählen `endtime`

Max. Dauer

Die Höchstdauer eines geplanten Auftrags muss unabhängig von `startTime` und `endTime` unter zwei Jahren liegen.

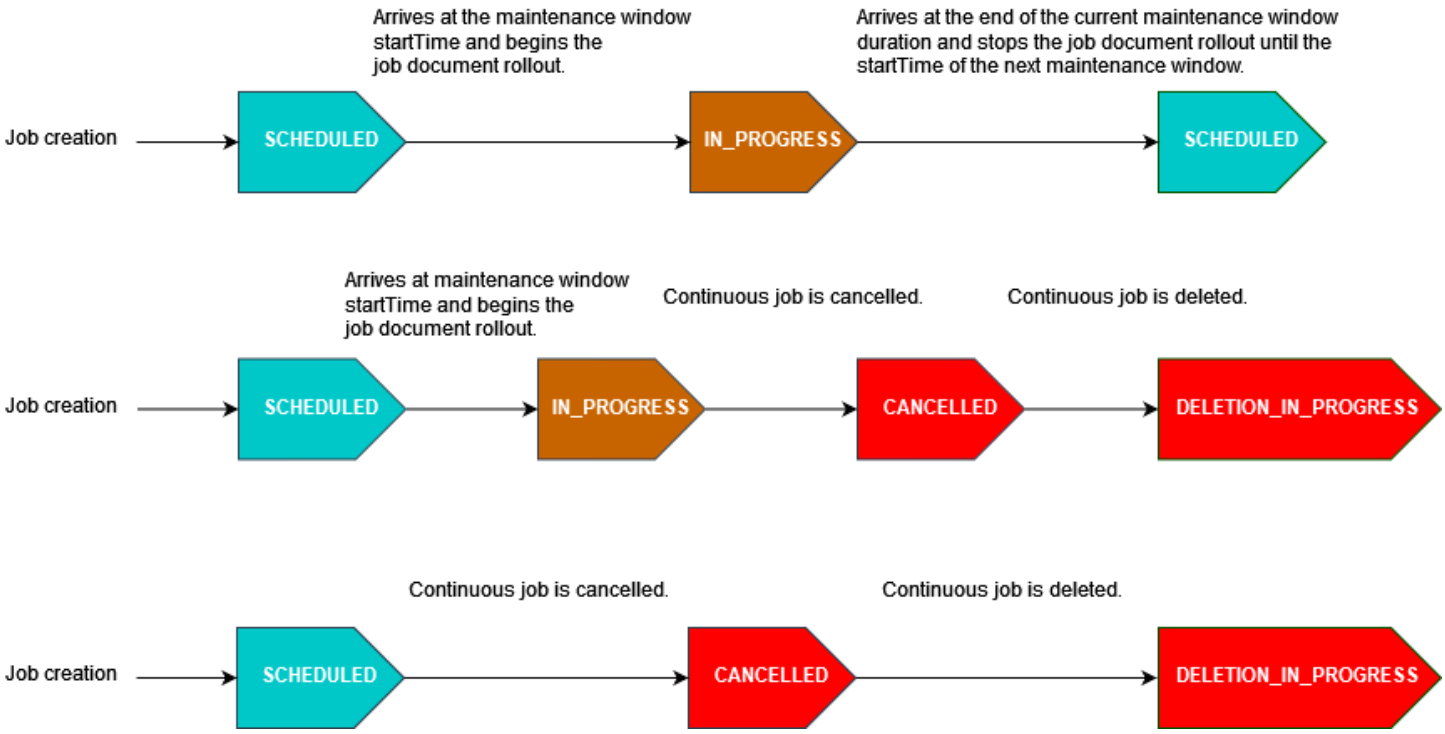
In der folgenden Tabelle sind die häufigsten Dauerszenarien für einen geplanten Auftrag aufgeführt:

Beispielnummer für einen geplanten Auftrag	startTime	endTime	Max. Dauer
1	Unmittelbar nach der ersten Auftragsrstellung.	Ein Jahr nach der ersten Auftragsrstellung.	Ein Jahr
2	Ein Monat nach der ersten Schaffung von Arbeitsplätzen.	13 Monate nach der ersten Schaffung von Arbeitsplätzen.	Ein Jahr
3	Ein Jahr nach der ersten Auftragsrstellung.	Zwei Jahre nach der ersten Auftragsrstellung.	Ein Jahr
4	Unmittelbar nach der ersten Auftragsrstellung.	Zwei Jahre nach der ersten Auftragsrstellung.	Zwei Jahre

Wiederkehrendes Wartungsfenster

Das Wartungsfenster ist eine optionale Konfiguration innerhalb der Planungskonfiguration der `CreateJob` und der `CreateJobTemplate` APIs. AWS-Managementkonsole `SchedulingConfig` Sie können ein wiederkehrendes Wartungsfenster mit einer festgelegten Startzeit, Dauer und Häufigkeit (täglich, wöchentlich oder monatlich) einrichten. Wartungsfenster gelten nur für fortlaufende Aufträge. Die maximale Dauer eines wiederkehrenden Wartungsfensters beträgt 23 Stunden und 50 Minuten.

Das folgende Diagramm zeigt den Status der Aufträge für verschiedene geplante Auftragsszenarien mit einem optionalen Wartungsfenster:



Weitere Informationen zu Auftragsstatus finden Sie unter [Aufträge und Status der Auftragsausführung](#).

> **Note**
>
> Wenn ein Auftrag `endTime` während eines Wartungsfensters am eingeht, wird er von `IN_PROGRESS` bis `COMPLETED` aktualisiert. Darüber hinaus folgen alle verbleibenden Auftragsausführungen dem `endBehavior` für den Auftrag.

Cron-Ausdrücke

Bei geplanten Aufträgen, bei denen das Auftragsdokument während eines Wartungsfensters mit einer benutzerdefinierten Häufigkeit ausgeführt wird, wird die benutzerdefinierte Häufigkeit mithilfe eines Cron-Ausdrucks eingegeben. Ein Cron-Ausdruck verfügt über sechs Pflichtfelder, die durch Leerzeichen voneinander getrennt sind.

Syntax

```
cron(fields)
```

Feld	Werte	Platzhalter
Minuten	0-59	, - * /
Stunden	0-23	, - * /
Day-of-month	1-31	, - * ? / L W
Monat	1—12 oder JAN-DEC	, - * /
Day-of-week	1-7 oder SUN-SAT	, - * ? / L #
Jahr	1970-2199	, - * /

Platzhalter

- Das Platzhalterzeichen , (Komma) schließt zusätzliche Werte ein. Im Feld "Monat" steht JAN, FEB, MAR für Januar, Februar und März.
- Das Platzhalterzeichen - (Bindestrich) gibt einen Bereich an. Im Feld "Tag" steht 1-15 für die Tage 1 bis 15 des angegebenen Monats.
- Das Platzhalterzeichen * (Sternchen) steht für alle Werte im Feld. Im Feld für die Stundenangaben steht * für alle Stunden. Sie können nicht in beiden Feldern als auch in den Day-of-month Day-of-week Feldern * verwenden. Wenn Sie es in einem der Felder eingeben, müssen Sie im anderen Feld ein ? verwenden.
- Das Platzhalterzeichen / (Schrägstrich) steht für schrittweise Steigerungen. Im Feld Minuten könnten Sie angeben, dass jede zehnte Minute, beginnend mit der ersten Minute der Stunde, angegeben werden soll (z. B. die 11., 21. und 31. Minute usw.). 1/10
- Das Platzhalterzeichen ? (Fragezeichen) steht für einen Wert. In das Day-of-month Feld könnten Sie 7 eingeben, und wenn es Ihnen egal wäre, welcher Wochentag der 7. ist, könnten Sie eingeben? im Day-of-week Feld.
- Der Platzhalter L in den Day-of-week Feldern Day-of-month oder gibt den letzten Tag des Monats oder der Woche an.
- Der **W** Platzhalter im Day-of-month Feld gibt einen Wochentag an. **3W**Gibt in dem Day-of-month Feld den Wochentag an, der dem dritten Tag des Monats am nächsten liegt.
- Der Platzhalter # im Day-of-week Feld gibt eine bestimmte Instanz des angegebenen Wochentags innerhalb eines Monats an. Beispiel: 3#2 steht für den zweiten Dienstag des Monats: Die 3 bezieht

sich auf Dienstag, da dies der dritte Tag jeder Woche ist, und die 2 bezieht sich auf den zweiten Tag dieses Typs innerhalb des Monats.

> **Note**
>
> Wenn Sie ein '#' -Zeichen verwenden, können Sie nur einen Ausdruck im Wochentag-Feld definieren. Beispiel, `"3#1,6#3"` ist ungültig, da es als zwei Ausdrücke interpretiert wird.

Einschränkungen

- Sie können die Day-of-week Felder Day-of-month und nicht in demselben Cron-Ausdruck angeben. Wenn Sie einen Wert (oder einen *) in einem der Felder angeben, müssen Sie in dem anderen Feld ein ? eingeben.

Beispiele

Wenn Sie einen Cron-Ausdruck für ein wiederkehrendes Wartungsfenster verwenden, können Sie sich auf die folgenden Beispiele für `startTime` beziehen.

Minuten	Stunden	Tag des Monats	Monat	Wochentag	Jahr	Bedeutung
0	10	*	*	?	*	Ausführung jeden Tag um 10:00 Uhr (UTC)
15	12	*	*	?	*	Ausführung jeden Tag um 12:15 Uhr (UTC)
0	18	?	*	MON-FRI	*	Ausführung jeden Montag bis

Minuten	Stunden	Tag des Monats	Monat	Wochentag	Jahr	Bedeutung
						Freitag um 18:00 Uhr (UTC)
0	8	1	*	?	*	Ausführung jeden 1. Tag des Monats um 08:00 Uhr (UTC)

Logik für die Dauer des wiederkehrenden Wartungsfensters

Wenn ein Auftrags-Rollout während eines Wartungsfensters das Ende des aktuellen Wartungsfensters erreicht, werden die folgenden Aktionen ausgeführt:

- Der Auftrag beendet alle Rollouts des Auftragsdokuments auf allen verbleibenden Geräten in Ihrer Zielgruppe. Es wird mit dem `startTime` des nächsten Wartungsfensters fortgesetzt.
- Alle Auftragsausführungen mit dem Status `QUEUED` bleiben `QUEUED` bis zum `startTime` des nächsten Wartungsfenster. Im nächsten Fenster können sie zu dem `IN_PROGRESS` wechseln, zu dem das Gerät bereit ist, mit der Ausführung der im Auftragsdokument angegebenen Aktionen zu beginnen.
- Alle Auftragsausführungen mit dem Status `IN_PROGRESS` setzen die Ausführung der im Auftragsdokument angegebenen Aktionen fort, bis sie den Terminal-Zustand erreichen. Alle Wiederholungsversuche, wie unter `JobExecutionsRetryConfig` beschrieben, finden mit dem `startTime` des nächsten Wartungsfensters statt.

Konfiguration des Auftragsabbruchs

Verwenden Sie diese Konfiguration, um Kriterien für das Abbrechen eines Auftrags zu erstellen, wenn ein bestimmter Prozentsatz von Geräten diese Kriterien erfüllt. Mit dieser Konfiguration können Sie beispielsweise einen Auftrag in den folgenden Fällen stornieren:

- Wenn ein bestimmter Prozentsatz der Geräte keine Benachrichtigungen zur Auftragsausführung erhält, z. B. wenn Ihr Gerät für ein Over-The-Air (OTA-) Update nicht kompatibel ist. In diesem Fall kann Ihr Gerät einen `REJECTED`-Status melden.
- Wenn ein bestimmter Prozentsatz von Geräten Fehler bei der Auftragsausführung meldet, z. B. wenn Ihr Gerät beim Versuch, das Auftragsdokument von einer Amazon S3-URL herunterzuladen, auf eine Verbindungsunterbrechung stößt. In solchen Fällen muss Ihr Gerät so programmiert sein, dass es den `FAILURE`-Status an AWS IoT meldet.
- Wenn ein `TIMED_OUT`-Status gemeldet wird, weil die Auftragsausführung nach dem Start der Auftragsausführung für einen bestimmten Prozentsatz von Geräten ein Timeout überschreitet.
- Wenn mehrere Wiederholungsversuche fehlgeschlagen sind. Wenn Sie eine Wiederholungskonfiguration hinzufügen, können für jeden erneuten Versuch zusätzliche Kosten für Ihr AWS-Konto anfallen. In solchen Fällen kann das Abbrechen des Auftrags die Ausführung von Aufträgen in der Warteschlange abbrechen und Wiederholungsversuche für diese Ausführungen verhindern. Weitere Informationen zur Wiederholungskonfiguration und deren Verwendung zusammen mit der Abbruchkonfiguration finden Sie unter Timeout bei der Auftragsausführung und Wiederholungskonfigurationen.

Sie können mithilfe der AWS IoT Konsole oder der Jobs-API eine Bedingung für den AWS IoT Jobabbruch einrichten.

Timeout bei der Auftragsausführung und Wiederholungskonfigurationen

Verwenden Sie die Timeout-Konfiguration für die Auftragsausführung, um Ihnen eine Auftragsbenachrichtigungen zu senden, wenn eine Auftragsausführung länger als die festgelegte Dauer andauert. Verwenden Sie die Konfiguration für die Wiederholung der Auftragsausführung, um die Ausführung erneut zu versuchen, wenn der Auftrag fehlschlägt oder das Timeout überschritten wird.

Timeout-Konfiguration für die Auftragsausführung

Verwenden Sie die Konfiguration für die Zeitüberschreitung bei der Auftragsausführung, um Sie zu benachrichtigen, wenn ein Auftrag für einen unerwartet langen Zeitraum im Status `IN_PROGRESS` stecken bleibt. Wenn der Auftrag `IN_PROGRESS` ist, können Sie den Fortschritt Ihrer Auftragsausführung überwachen.

Timer für Auftrags-Timeouts

Es gibt zwei Arten von Timern: Timer für „In Bearbeitung“ und Timer für „Schritt“.

In-progress Zeitschaltuhren

Wenn Sie einen Auftrag oder eine Auftragsvorlage erstellen, können Sie einen Wert für den Timer in Bearbeitung angeben, der zwischen 1 Minute und 7 Tagen liegt. Sie können den Wert dieses Timers bis zum Start der Auftragsausführung aktualisieren. Nachdem Ihr Timer gestartet wurde, kann er nicht mehr aktualisiert werden, und der Timerwert gilt für alle Auftragsausführungen für den Auftrag. Immer wenn eine Auftragsausführung länger als dieses Intervall im `IN_PROGRESS` Status verbleibt, schlägt die Auftragsausführung fehl und wechselt in den `TIMED_OUT` Terminalstatus. AWS IoT veröffentlicht auch eine MQTT-Benachrichtigung.

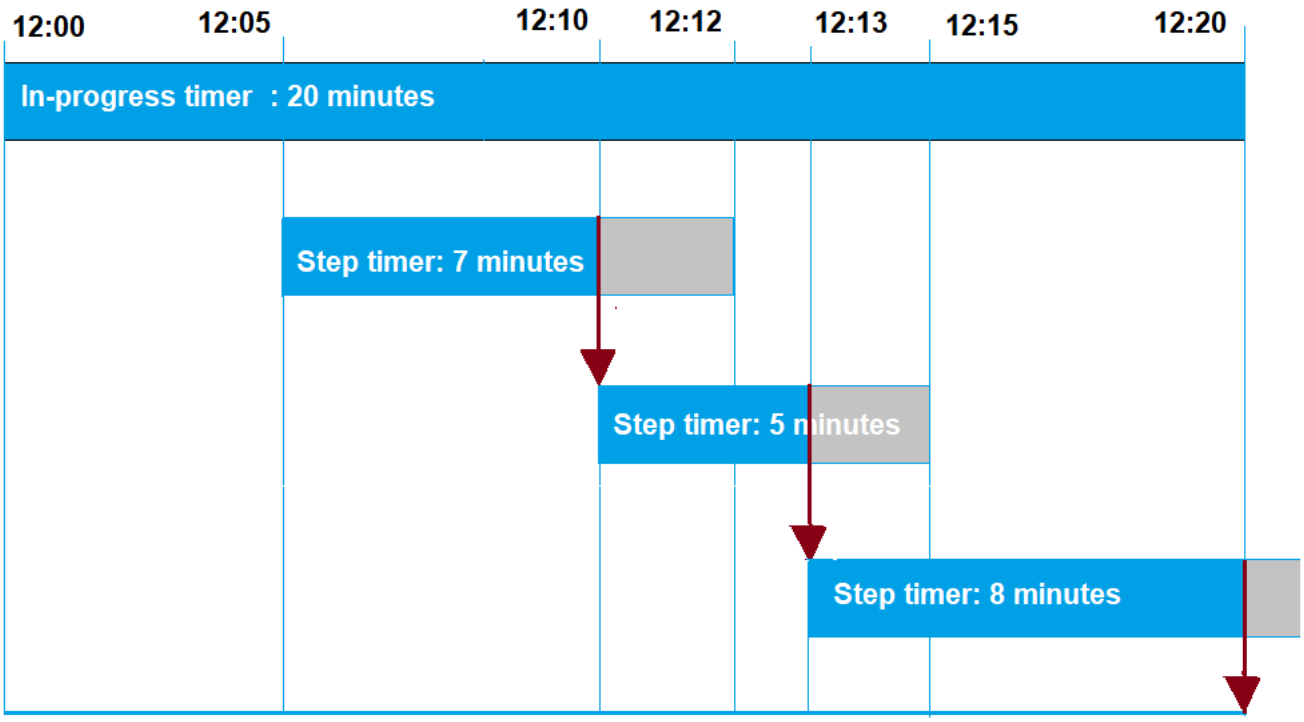
Schritt-Timer

Sie können auch einen Schritt-Timer festlegen, der nur für die Auftragsausführung gilt, die Sie aktualisieren möchten. Dieser Timer hat keine Auswirklungen auf den Timer, der gerade bearbeitet wird. Jedes Mal, wenn Sie eine Auftragsausführung aktualisieren, können Sie einen neuen Wert für den Schritt-Timer festlegen. Sie können auch einen neuen Schritt-Timer erstellen, wenn Sie die nächste ausstehende Auftragsausführung für ein Objekt starten. Falls die Auftragsausführung länger als das Schritt-Timer-Intervall im Status `IN_PROGRESS` bleibt, schlägt sie fehl und wechselt in den terminalen Status `TIMED_OUT`.

> **Note**
>
> Sie können den Timer für den laufenden Vorgang mithilfe der AWS IoT Konsole oder der AWS IoT Jobs-API festlegen. Verwenden Sie die API, um den Schritt-Timer anzugeben.

So funktionieren Timer für Auftrags-Timeouts

Nachfolgend sehen Sie, wie die Zeitüberschreitungen während eines 20-minütigen Zeitlimits und die Schrittzeitüberschreitungen miteinander interagieren.



Im Folgenden werden die einzelnen Schritte erläutert:

1. 12:00

   Beim Erstellen eines Auftrags wird ein neuer Auftrag erstellt und ein Timer für die Dauer von 20 Minuten in Bearbeitung gestartet. Der Timer für die Bearbeitung beginnt zu laufen und die Auftragsausführung wechselt in den Status `IN_PROGRESS`.

2. 12:05

   Ein neuer Schritt-Timer mit einem Wert von 7 Minuten wird erstellt. Die Auftragsausführung läuft jetzt um 12:12 Uhr ab.

3. 12:10

   Ein neuer Schritt-Timer mit einem Wert von 5 Minuten wird erstellt. Wenn ein neuer Schritt-Timer erstellt wird, wird der vorherige Step-Timer verworfen und die Auftragsausführung läuft nun um 12:15 Uhr ab.

4. 12:13

   Ein neuer Schritt-Timer mit einem Wert von 9 Minuten wird erstellt. Der vorherige Schritt-Timer wird verworfen und die Auftragsausführung läuft jetzt um 12:20 Uhr ab, da der Timer für die

Bearbeitung des laufenden Timers um 12:20 Uhr abläuft. Der Schritt-Timer kann die absolute Grenze des laufenden Timers nicht überschreiten.

Auftragsausführung: Konfiguration wiederholen

Sie können die Wiederholungskonfiguration verwenden, um die Ausführung des Auftrags erneut zu versuchen, wenn bestimmte Kriterien erfüllt sind. Ein Wiederholungsversuch kann versucht werden, wenn bei einem Auftrag das Timeout überschritten wird oder wenn das Gerät ausfällt. Um die Ausführung aufgrund eines Timeout-Fehlers erneut zu versuchen, müssen Sie die Timeout-Konfiguration aktivieren.

Wie man Konfiguration wiederholen verwendet

Führen Sie die folgenden Schritte aus, um die Konfiguration zu wiederholen:

1. Bestimmen Sie, ob Sie die Wiederholungskonfiguration für `FAILED`, `TIMED_OUT` oder beide Fehlerkriterien verwenden möchten. Für den `TIMED_OUT` Status versucht AWS IoT Jobs nach der Statusmeldung automatisch erneut, den Job für das Gerät auszuführen.
2. Prüfen Sie für den `FAILED`-Status, ob Ihr Fehler bei der Auftragsausführung erneut versucht werden kann. Wenn ein erneuter Versuch möglich ist, programmieren Sie Ihr Gerät so, dass es einen `FAILURE`-Status an AWS IoT meldet. Im folgenden Abschnitt erfahren Sie mehr über wiederholbare und nicht wiederholbare Fehler.
3. Geben Sie anhand der obigen Informationen die Anzahl der Wiederholungen an, die für jeden Fehlertyp verwendet werden sollen. Für ein einzelnes Gerät können Sie bis zu 10 Wiederholungsversuche für beide Fehlertypen zusammen angeben. Die Wiederholungsversuche werden automatisch beendet, wenn eine Ausführung erfolgreich ist oder wenn die angegebene Anzahl von Versuchen erreicht wird.
4. Fügen Sie eine Abbruchkonfiguration hinzu, um den Auftrag abzubrechen, wenn wiederholte Wiederholungsversuche fehlschlagen, um zu vermeiden, dass bei einer großen Anzahl von Wiederholungsversuchen zusätzliche Kosten anfallen.

> **Note**
>
> Wenn ein Auftrag das Ende eines wiederkehrenden Wartungsfensters erreicht, führen alle `IN_PROGRESS`-Auftragsausführungen weiterhin die im Auftragsdokument angegebenen Aktionen aus, bis sie den Terminal-Zustand erreichen. Wenn die Ausführung eines Auftrags außerhalb eines Wartungsfensters den Terminal-Zustand `FAILED` oder `TIMED_OUT` erreicht,

wird im nächsten Fenster ein erneuter Versuch unternommen, wenn die Versuche nicht erschöpft sind. Beim `startTime` des Wartungsfensters wird eine neue Auftragsausführung erstellt, die in den Status `QUEUED` wechselt, bis das Gerät startbereit ist.

Versuchen Sie es erneut und brechen Sie die Konfiguration ab

Bei jedem erneuten Versuch fallen zusätzliche Gebühren für Sie an. AWS-Konto Um zu vermeiden, dass zusätzliche Gebühren aufgrund wiederholter Wiederholungsversuche anfallen, empfehlen wir, eine Abbruchkonfiguration hinzuzufügen. Weitere Informationen zur Preisgestaltung finden Sie unter AWS IoT Device Management – Preise.

Es kann vorkommen, dass mehrere Wiederholungsversuche fehlschlagen, wenn bei einem hohen Prozentsatz Ihrer Geräte ein Timeout auftritt oder ein Fehler gemeldet wird. In diesem Fall können Sie die Abbruchkonfiguration verwenden, um den Auftrag abzubrechen und so zu vermeiden, dass Aufträge in der Warteschlange ausgeführt oder weitere Versuche wiederholt werden.

> **Note**
>
> Wenn die Abbruchkriterien für den Abbruch einer Auftragsausführung erfüllt sind, werden nur `QUEUED`-Auftragsausführungen abgebrochen. Wiederholungen in der Warteschlange für das Gerät werden nicht versucht. Aktuelle Auftragsausführungen, die einen `IN_PROGRESS`-Status haben, werden jedoch nicht abgebrochen.

Bevor Sie eine fehlgeschlagene Auftragsausführung erneut versuchen, empfehlen wir Ihnen außerdem, zu überprüfen, ob die fehlgeschlagene Auftragsausführung erneut versucht werden kann, wie im folgenden Abschnitt beschrieben.

Versuchen Sie es erneut, wenn der Fehlertyp FEHLGESCHLAGEN ist

Um Wiederholungsversuche für den Fehlertyp `FAILED` zu versuchen, müssen Ihre Geräte so programmiert sein, dass sie den `FAILURE`.Status einer fehlgeschlagenen Auftragsausführung an AWS IoT melden. Legen Sie die Wiederholungskonfiguration mit den Kriterien fest, nach denen die `FAILED` Auftragsausführung wiederholt werden soll, und geben Sie die Anzahl der auszuführenden Wiederholungen an. Wenn AWS IoT Jobs den `FAILURE` Status erkennt, versucht das Programm automatisch, die Auftragsausführung für das Gerät zu wiederholen. Die Wiederholungsversuche werden fortgesetzt, bis die Auftragsausführung erfolgreich ist oder die maximale Anzahl von Wiederholungsversuchen erreicht ist.

Sie können jeden Wiederholungsversuch und den Auftrag, der auf diesen Geräten ausgeführt wird, verfolgen. Indem Sie den Ausführungsstatus verfolgen, können Sie nach der angegebenen Anzahl von Wiederholungsversuchen Ihr Gerät verwenden, um Fehler zu melden und einen weiteren Wiederholungsversuch einzuleiten.

Fehler, die wiederholt werden können und die nicht wiederholt werden können

Ihr Fehler bei der Auftragsausführung kann wiederholt oder nicht wiederholt werden. Für jeden Wiederholungsversuch können Gebühren für Ihr AWS-Konto anfallen. Um zu vermeiden, dass bei mehreren Wiederholungsversuchen zusätzliche Gebühren anfallen, sollten Sie zunächst prüfen, ob Ihr Fehler bei der Auftragsausführung erneut versucht werden kann. Ein Beispiel für einen erneuten Versuch ist ein Verbindungsfehler, der auf Ihrem Gerät auftritt, wenn versucht wird, das Auftragsdokument von einer Amazon S3-URL herunterzuladen. Wenn Ihr Fehler bei der Auftragsausführung erneut versucht werden kann, programmieren Sie Ihr Gerät so, dass es einen `FAILURE`-Status meldet, falls die Auftragsausführung fehlschlägt. Stellen Sie dann die Wiederholungskonfiguration so ein, dass `FAILED`-Ausführungen erneut versucht werden.

Wenn die Ausführung nicht erneut versucht werden kann, empfehlen wir, das Gerät so zu programmieren, dass es einen `REJECTED`-Status an AWS IoT meldet, um einen erneuten Versuch zu vermeiden und Ihrem Konto möglicherweise zusätzliche Gebühren aufzuerlegen. Ein Fehler, der nicht wiederholt werden kann, ist zum Beispiel, wenn Ihr Gerät keine Auftragsaktualisierung empfangen kann oder wenn bei der Ausführung eines Auftrags ein Speicherfehler auftritt. In diesen Fällen versucht AWS IoT Jobs die Auftragsausführung nicht erneut, da die Auftragsausführung nur dann erneut versucht wird, wenn ein `FAILED` OR-Status erkannt wird. `TIMED_OUT`

Wenn Sie festgestellt haben, dass ein Fehler bei der Auftragsausführung erneut versucht werden kann, sollten Sie die Geräteprotokolle überprüfen, falls ein erneuter Versuch immer noch fehlschlägt.

> **Note**
>
> Wenn ein Auftrag mit der optionalen Planungskonfiguration seinen `endTime` erreicht, stoppt `endBehavior` die Auswahl die Bereitstellung des Auftragsdokuments auf allen verbleibenden Geräten in der Zielgruppe und bestimmt, wie mit den verbleibenden Auftragsausführungen fortgefahren werden soll. Die Versuche werden wiederholt, wenn sie über die Wiederholungskonfiguration ausgewählt wurden.

Versuchen Sie es erneut, wenn der Fehlertyp TIMEOUT lautet

Wenn Sie beim Erstellen eines Jobs das Timeout aktivieren, versucht AWS IoT Jobs, die Auftragsausführung für das Gerät zu wiederholen, wenn sich der Status von zu ändert. `IN_PROGRESS` `TIMED_OUT` Diese Statusänderung kann auftreten, wenn für den Timer in Bearbeitung eine Zeitüberschreitung eintritt oder wenn ein von Ihnen festgelegter Schritt-Timer in `IN_PROGRESS` ist und dann das Zeitlimit überschritten wird. Die Wiederholungsversuche werden fortgesetzt, bis die Auftragsausführung erfolgreich ist oder bis die maximale Anzahl von Wiederholungsversuchen für diesen Fehlertyp erreicht ist.

Fortlaufende Aktualisierungen von Aufträgen und Mitgliedschaften in Objektgruppen

Bei kontinuierlichen Aufträgen mit dem Auftragsstatus `IN_PROGRESS` wird die Anzahl der Wiederholungsversuche auf Null zurückgesetzt, wenn die Gruppenmitgliedschaft eines Objekts aktualisiert wird. Nehmen wir beispielsweise an, Sie haben fünf Wiederholungsversuche angegeben und drei Wiederholungsversuche wurden bereits durchgeführt. Wenn ein Ding jetzt aus der Objektgruppe entfernt wird und dann wieder der Gruppe beitritt, z. B. bei dynamischen Objektgruppen, wird die Anzahl der Wiederholungsversuche auf Null zurückgesetzt. Sie können jetzt fünf Wiederholungsversuche für Ihre Objektgruppe durchführen, anstatt der beiden verbleibenden Versuche. Wenn ein Objekt aus der Objektgruppe entfernt wird, werden außerdem weitere Wiederholungsversuche abgebrochen.

Zusätzliche Konfigurationen angeben

Wenn Sie einen Auftrag oder eine Auftragsvorlage erstellen, können Sie diese zusätzlichen Konfigurationen angeben. Im Folgenden wird gezeigt, wann Sie diese Konfigurationen angeben können.

- Wenn Sie eine benutzerdefinierte Auftragsvorlage erstellen. Die zusätzlichen Konfigurationseinstellungen, die Sie angeben, werden gespeichert, wenn Sie einen Auftrag anhand der Vorlage erstellen.
- Beim Erstellen eines benutzerdefinierten Auftrags mithilfe einer Auftragsdatei. Die Auftragsdatei kann eine JSON-Datei sein, die in einen S3-Bucket hochgeladen wird.
- Beim Erstellen eines benutzerdefinierten Auftrags mithilfe einer benutzerdefinierten Auftragsvorlage. Wenn für die Vorlage diese Einstellungen bereits angegeben sind, können Sie sie entweder wiederverwenden oder sie überschreiben, indem Sie neue Konfigurationseinstellungen angeben.
- Wenn Sie einen benutzerdefinierten Job mithilfe einer AWS verwalteten Vorlage erstellen.

Themen

- Geben Sie Auftragskonfigurationen an, indem Sie den AWS-Managementkonsole
- Geben Sie Auftragskonfigurationen an, indem Sie den AWS IoT Auftrags-API

Geben Sie Auftragskonfigurationen an, indem Sie den AWS-Managementkonsole

Sie können die verschiedenen Konfigurationen für Ihren Job mithilfe der AWS IoT Konsole hinzufügen. Nachdem Sie einen Auftrag erstellt haben, können Sie die Statusdetails Ihrer Auftrags-Konfigurationen auf der Seite mit den Auftragsdetails sehen. Weitere Informationen über die verschiedenen Konfigurationen und deren Funktionsweise finden Sie unter Wie funktionieren Auftragskonfigurationen.

Fügen Sie die Auftragskonfigurationen hinzu, wenn Sie einen Auftrag oder eine Auftragsvorlage erstellen.

Wenn Sie eine benutzerdefinierte Auftragsvorlage erstellen

Um die Rollout-Konfiguration bei der Erstellung einer benutzerdefinierten Auftragsvorlage anzugeben

1. Gehen Sie in der AWS IoT Konsole zum Hub „Jobvorlagen“ und wählen Sie „Jobvorlage erstellen“.
2. Geben Sie die Eigenschaften der Auftragsvorlage an, stellen Sie das Auftragsdokument bereit, erweitern Sie die Konfiguration, die Sie hinzufügen möchten, und geben Sie dann die Konfigurationsparameter an.

Beim Erstellen eines benutzerdefinierten Auftrags

Um die Rollout-Konfiguration bei der Erstellung eines benutzerdefinierten Auftrags anzugeben

1. Gehen Sie zum Job-Hub der AWS IoT Konsole und wählen Sie Job erstellen.
2. Wählen Sie „Benutzerdefinierten Auftrag erstellen“ und geben Sie die Auftragseigenschaften und Ziele an und geben Sie an, ob eine Auftragsdatei oder eine Vorlage für das Auftragsdokument verwendet werden soll. Sie können eine benutzerdefinierte Vorlage oder eine AWS verwaltete Vorlage verwenden.
3. Wählen Sie die Auftragskonfiguration aus und erweitern Sie dann die Rollout-Konfiguration, um anzugeben, ob eine konstante Rate oder eine exponentielle Rate verwendet werden soll. Geben Sie dann die Konfigurationsparameter an.

Der nächste Abschnitt zeigt die Parameter, die Sie für jede Konfiguration angeben können.

Rollout-Konfiguration

Sie können angeben, ob Sie eine konstante Rollout-Rate oder eine exponentielle Rate verwenden möchten.

- Stellen Sie eine konstante Rollout-Rate ein

  Um eine konstante Rate für die Ausführung von Aufträgen festzulegen, wählen Sie Konstante Rate und geben Sie dann das Maximum pro Minute als Obergrenze für die Rate an. Dieser Wert ist optional und reicht von 1 bis 1000. Wenn Sie ihn nicht festlegen, wird 1000 als Standardwert verwendet.
- Legen Sie eine exponentielle Rollout-Rate fest

  Um eine exponentielle Rate festzulegen, wählen Sie Exponentielle Rate und geben Sie dann die folgenden Parameter an:

  - Basistarif pro Minute

    Die Geschwindigkeit, mit der die Aufträge ausgeführt werden, bis der Schwellenwert für die Anzahl der gemeldeten Geräte oder die Anzahl der erfolgreichen Geräte für die Kriterien zur Erhöhung der Rate erreicht ist.
  - Inkrementfaktor

    Der exponentielle Faktor, um den sich die Rollout-Rate erhöht, wenn der Schwellenwert „Anzahl der benachrichtigten Geräte“ oder „Anzahl der erfolgreichen Geräte“ für die Kriterien „Ratenerhöhung“ erreicht ist.
  - Kriterien für die Ratenerhöhung

    Der Schwellenwert für entweder die Anzahl der gemeldeten Geräte oder die Anzahl der erfolgreichen Geräte.

Konfiguration abbrechen

Wählen Sie Neue Konfiguration hinzufügen und geben Sie für jede Konfiguration die folgenden Parameter an:

- Art des Fehlers

Gibt die Fehlertypen an, die einen Auftragsabbruch auslösen. Dazu gehören FAILED, REJECTED, TIMED_OUT oder ALL.

- Inkrementfaktor

  Gibt die Anzahl der abgeschlossenen Auftragsausführungen an, die erfolgen müssen, bevor das Auftragsabbruchkriterium erfüllt ist.

- Schwellenwert-Prozentsatz

  Gibt die Gesamtanzahl der ausgeführten Objekte an, die einen Auftragsabbruch auslöst.

Konfiguration des Zeitplans

Jeder Auftrag kann sofort nach der ersten Erstellung beginnen, zu einem späteren Zeitpunkt beginnen oder während eines wiederkehrenden Wartungsfensters ausgeführt werden.

Wählen Sie Neue Konfiguration hinzufügen und geben Sie für jede Konfiguration die folgenden Parameter an:

- Beginn des Auftrags

  Geben Sie das Datum und die Uhrzeit an, zu der der Auftrag gestartet wird.

- Wiederkehrendes Wartungsfenster

  Ein wiederkehrendes Wartungsfenster definiert das genaue Datum und die Uhrzeit, zu der ein Auftrag das Auftragsdokument auf den Zielgeräten des Auftrags bereitstellen kann. Das Wartungsfenster kann täglich, wöchentlich, monatlich oder mit einer benutzerdefinierten Wiederholung von Tag und Uhrzeit wiederholt werden.

- Ende des Auftrags

  Geben Sie das Datum und die Uhrzeit an, zu der der Auftrag beendet wird.

- Verhalten bei Auftragsende

  Wählen Sie ein Endverhalten für alle noch nicht abgeschlossenen Auftragsausführungen aus, wenn der Auftrag beendet ist.

> **Note**
>
> Wenn ein Auftrag mit der optionalen Zeitplanungskonfiguration und der gewählten Endzeit die Endzeit erreicht, stoppt der Auftrag das Rollout auf alle verbleibenden Geräte in der Zielgruppe. Außerdem wird anhand des ausgewählten Endverhaltens entschieden, wie mit den verbleibenden Auftragsausführungen und deren Wiederholungsversuchen gemäß der Wiederholungskonfiguration fortgefahren werden soll.

Timeout-Konfiguration

Standardmäßig gibt es kein Timeout und Ihr Auftrag wird abgebrochen oder gelöscht. Um Timeouts zu verwenden, wählen Sie Timeout aktivieren und geben Sie dann einen Timeout-Wert zwischen 1 Minute und 7 Tagen an.

Wiederholungs-Konfiguration

> **Note**
>
> Nachdem ein Auftrag erstellt wurde, kann die Anzahl der Wiederholungen nicht aktualisiert werden. Sie können die Wiederholungskonfiguration nur für alle Fehlertypen entfernen. Wenn Sie einen Auftrag erstellen, sollten Sie die entsprechende Anzahl von Wiederholungsversuchen berücksichtigen, die Sie für Ihre Konfiguration verwenden. Fügen Sie eine Abbruchkonfiguration hinzu, um übermäßige Kosten aufgrund potenzieller Wiederholungsfehler zu vermeiden.

Wählen Sie Neue Konfiguration hinzufügen und geben Sie für jede Konfiguration die folgenden Parameter an:

- Art des Fehlers

  Gibt die Fehlertypen an, die einen erneuten Versuch der Auftragsausführung auslösen sollen. Dazu gehören Fehlgeschlagen, Timeout und Alle.
- Anzahl der Wiederholungen

  Gibt die Anzahl der Wiederholungen für den ausgewählten Fehlertyp an. Für beide Fehlertypen zusammen können bis zu 10 Wiederholungsversuche versucht werden.

Geben Sie Auftragskonfigurationen an, indem Sie den AWS IoT Auftrags-API

Sie können die CreateJob oder die CreateJobTemplate API verwenden, um die verschiedenen Jobkonfigurationen anzugeben. In den folgenden Abschnitten wird beschrieben, wie Sie diese Konfigurationen hinzufügen. Nachdem Sie die Konfigurationen hinzugefügt haben, können Sie sie verwenden JobExecutionSummary und, JobExecutionSummaryForJob um ihren Status einzusehen.

Weitere Informationen über die verschiedenen Konfigurationen und deren Funktionsweise finden Sie unter Wie funktionieren Auftragskonfigurationen.

Rollout-Konfiguration

Sie können eine konstante Rollout-Rate oder eine exponentielle Rollout-Rate für Ihre Rollout-Konfiguration angeben.

- Stellen Sie eine konstante Rollout-Rate ein

  Um eine konstante Rollout-Rate festzulegen, verwenden Sie das `JobExecutionsRolloutConfig`-Objekt, um den `maximumPerMinute`-Parameter zur `CreateJob`-Anforderung hinzuzufügen. Dieser Parameter gibt die obere Grenze der Rate an, mit der Auftragsausführungen durchgeführt werden können. Dieser Wert ist optional und reicht von 1 bis 1000. Wenn Sie den Wert nicht festlegen, wird 1000 als Standardwert verwendet.

  ```
      "jobExecutionsRolloutConfig": {
          "maximumPerMinute": 1000
      }
  ```

- Legen Sie eine exponentielle Rollout-Rate fest

  Verwenden Sie das `JobExecutionsRolloutConfig`-Objekt, um eine variable Auftrags-Rollout-Rate festzulegen. Sie können die `ExponentialRolloutRate`-Eigenschaft konfigurieren, wenn Sie den `CreateJob`-API-Vorgang ausführen. Im folgenden Beispiel wird eine variable Rolloutrate mit dem Parameter `exponentialRate` eingerichtet. Weitere Informationen zu den Parametern finden Sie unter `ExponentialRolloutRate`.

  ```
  {
  ...
    "jobExecutionsRolloutConfig": {
      "exponentialRate": {
  ```

```
      "baseRatePerMinute": 50,
      "incrementFactor": 2,
      "rateIncreaseCriteria": {
        "numberOfNotifiedThings": 1000,
        "numberOfSucceededThings": 1000
      },
      "maximumPerMinute": 1000
    }
  }
...
}
```

Wo der Parameter:

baseRatePerMinute

Gibt die Rate an, mit der die Aufträge ausgeführt werden, bis der `numberOfNotifiedThings`- oder der `numberOfSucceededThings`-Schwellenwert erreicht wird.

incrementFactor

Gibt den exponentiellen Faktor an, um den die Rolloutrate erhöht wird, nachdem der `numberOfNotifiedThings`- oder der `numberOfSucceededThings`-Schwellenwert erreicht wird.

bewerten IncreaseCriteria

Gibt entweder den `numberOfNotifiedThings`- oder den `numberOfSucceededThings`-Schwellenwert an.

Konfiguration abbrechen

Um diese Konfiguration mithilfe der API hinzuzufügen, geben Sie den `AbortConfig`-Parameter an, wenn Sie den `CreateJob` oder den `CreateJobTemplate` API-Vorgang ausführen. Das folgende Beispiel zeigt eine Abbruchkonfiguration für einen Auftrags-Rollout, bei dem es, wie bei der `CreateJob` API-Operation angegeben, zu mehreren fehlgeschlagenen Ausführungen kam.

> **Note**
>
> Das Löschen einer Auftragsausführung wirkt sich auf die Berechnung des Werts der gesamten abgeschlossenen Ausführungen aus. Wenn ein Auftrag abgebrochen wird, erstellt

der Service einen automatisierten `comment` und `reasonCode` zur Differenzierung zwischen eines vom Benutzer ausgelösten Abbruchs und eines automatischen Auftragsabbruchs.

```
  "abortConfig": {
     "criteriaList": [
        {
           "action": "CANCEL",
           "failureType": "FAILED",
           "minNumberOfExecutedThings": 100,
           "thresholdPercentage": 20
        },
        {
           "action": "CANCEL",
           "failureType": "TIMED_OUT",
           "minNumberOfExecutedThings": 200,
           "thresholdPercentage": 50
        }
     ]
   }
```

Wo der Parameter:

Aktion

Gibt die Aktion an, die durchgeführt werden soll, wenn das Abbruchkriterium erfüllt ist. Dieser Parameter ist erforderlich, und `CANCEL` ist der einzige gültige Wert.

failureType

Gibt an, welche Fehlertypen einen Auftragsabbruch auslösen sollen. Gültige Werte sind `FAILED`, `REJECTED`, `TIMED_OUT` und `ALL`.

Min.NumberOfExecutedThings

Gibt die Anzahl der abgeschlossenen Auftragsausführungen an, die erfolgen müssen, bevor das Auftragsabbruchkriterium erfüllt ist. In diesem Beispiel prüft AWS IoT erst dann, ob ein Auftragsabbruch erfolgen soll, wenn mindestens 100 Geräte Auftragsausführungen abgeschlossen haben.

thresholdPercentage

Gibt die Gesamtzahl der Objekte an, für die Aufträge ausgeführt werden, die einen Auftragsabbruch auslösen können. In diesem Beispiel AWS IoT prüft es sequentiell und leitet einen Jobabbruch ein, wenn der Schwellenwert in Prozent erreicht wird. Wenn mindestens 20 % der vollständigen Ausführungen fehlschlagen, nachdem 100 Ausführungen abgeschlossen sind, wird der Auftrags-Rollout abgebrochen. Wenn dieses Kriterium nicht erfüllt ist, AWS IoT wird überprüft, ob bei mindestens 50% der abgeschlossenen Ausführungen nach Abschluss von 200 Ausführungen ein Timeout aufgetreten ist. Wenn dies der Fall ist, wird der Auftrags-Rollout abgebrochen.

Konfiguration des Zeitplans

Um diese Konfiguration mithilfe der API hinzuzufügen, geben Sie den optionalen `SchedulingConfig` an, wenn Sie den `CreateJob` oder den `CreateJobTemplate` API-Vorgang ausführen.

```
  "SchedulingConfig": {
    "endBehavior": string
    "endTime": string
    "maintenanceWindows": string
    "startTime": string
  }
```

Wo der Parameter:

startTime

Gibt das Datum und die Uhrzeit an, zu der der Auftrag gestartet wird.

endTime

Gibt das Datum und die Uhrzeit an, zu der der Auftrag beendet wird.

maintenanceWindows

Gibt an, ob für den geplanten Auftrag ein optionales Wartungsfenster ausgewählt wurde, um das Auftragsdokument auf alle Geräte in der Zielgruppe auszurollen. Das Zeichenkettenformat für `maintenanceWindow` steht YYYY/MM/DD für das Datum und hh:mm für die Uhrzeit.

endBehavior

Gibt das Auftragsverhalten für einen geplanten Auftrag beim Erreichen des `endTime` an.

> **Note**
>
> Die Option `SchedulingConfig` für einen Auftrag ist in den `DescribeJobTemplate`- und `DescribeJob`-APIs einsehbar.

Timeout-Konfiguration

Um diese Konfiguration mithilfe der API hinzuzufügen, geben Sie den `TimeoutConfig`-Parameter an, wenn Sie den `CreateJob` oder den `CreateJobTemplate` API-Vorgang ausführen.

Um die Timeout-Konfiguration zu verwenden

1. Um beim Erstellen eines Jobs oder einer Auftragsvorlage den Timer für die Bearbeitung festzulegen, legen Sie einen Wert für die `inProgressTimeoutInMinutes` Eigenschaft des optionalen Objekts fest. TimeoutConfig

```
    "timeoutConfig": {
      "inProgressTimeoutInMinutes": number
   }
```

2. Um einen Schrittzeitgeber für die Ausführung eines Jobs anzugeben, legen Sie einen Wert für den `stepTimeoutInMinutes` Zeitpunkt des Aufrufs UpdateJobExecution fest. Der Schritt Timer gilt nur für die Auftragsausführung, die Sie aktualisieren. Sie können einen neuen Wert für diesen Timer bei jeder Aktualisierung einer Auftragsausführung einrichten.

   > **Note**
   >
   > `UpdateJobExecution` kann einen bereits erstellten Schritt-Timer verwerfen, indem er einen neuen Schritt-Timer mit dem Wert -1 erstellt.

```
{
   ...
    "statusDetails": {
      "string" : "string"
   },
   "stepTimeoutInMinutes": number
}
```

3. Um einen neuen Schrittzeitgeber zu erstellen, können Sie auch die StartNextPendingJobExecution API-Operation aufrufen.

Wiederholungs-Konfiguration

> **Note**
>
> Wenn Sie einen Auftrag erstellen, sollten Sie die entsprechende Anzahl von Wiederholungsversuchen berücksichtigen, die Sie für Ihre Konfiguration verwenden. Fügen Sie eine Abbruchkonfiguration hinzu, um übermäßige Kosten aufgrund potenzieller Wiederholungsfehler zu vermeiden. Nachdem ein Auftrag erstellt wurde, kann die Anzahl der Wiederholungen nicht aktualisiert werden. Sie können die Anzahl der Wiederholungen nur mithilfe der UpdateJob API-Operation auf 0 setzen.

Um diese Konfiguration mithilfe der API hinzuzufügen, geben Sie den `jobExecutionsRetryConfig`-Parameter an, wenn Sie den `CreateJob` oder den `CreateJobTemplate` API-Vorgang ausführen.

```
{
...
  "jobExecutionsRetryConfig": {
      "criteriaList": [
         {
            "failureType": "string",
            "numberOfRetries": number
         }
      ]
  }
...
}
```

Wobei criteriaList ein Array ist, das die Kriterienliste angibt, die die Anzahl der zulässigen Wiederholungen für jeden Fehlertyp für einen Auftrag bestimmt.

Geräte und Aufträge

Geräte können mithilfe von MQTT, HTTP Signature Version 4 oder HTTP TLS mit AWS IoT Jobs kommunizieren. Führen Sie den Befehl aus, um den Endpunkt zu ermitteln, der verwendet werden

soll, wenn Ihr Gerät mit AWS IoT Jobs kommuniziert. DescribeEndpoint Wenn Sie z. B. den folgenden Befehl ausführen:

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

Das Ergebnis sieht in etwa wie folgt aus:

```
{
    "endpointAddress": "a1b2c3d4e5f6g7-ats.iot.us-west-2.amazonaws.com"
}
```

Verwendung des MQTT-Protokolls

Geräte können mithilfe des MQTT-Protokolls mit AWS IoT Jobs kommunizieren. Geräte abonnieren MQTT-Themen, um über neue Jobs informiert zu werden und Antworten vom AWS IoT Jobs-Dienst zu erhalten. Geräte veröffentlichen auf MQTT-Themen, um den Status eines Auftrags-Launch abzufragen oder zu aktualisieren. Jedes Gerät verfügt über ein allgemeines MQTT-Thema. Für weitere Informationen zur Veröffentlichung und zum Abonnement von MQTT-Themen vgl. Gerätekommunikationsprotokolle.

Bei dieser Kommunikationsmethode verwendet Ihr Gerät sein gerätespezifisches Zertifikat und seinen privaten Schlüssel, um sich bei Jobs zu authentifizieren. AWS IoT

Ihre Geräte können die folgenden Themen abonnieren. `thing-name` ist der Name des Objekts, das mit dem Gerät verknüpft ist.

- **$aws/things/*Name einer Sache*/jobs/notify**

  Abonnieren Sie dieses Thema, um Sie zu benachrichtigen, wenn ein Auftragsstart zur Liste der ausstehenden Auftragsstarts hinzugefügt oder daraus entfernt wird.

- **$aws/things/*Dingname*/-weiter jobs/notify**

  Abonnieren Sie dieses Thema, um Sie zu benachrichtigen, wenn sich die nächste ausstehende Auftragsausführung geändert hat.

- **$aws/things/*Dingname*/jobs/*Name der Anfrage*/akzeptiert**

  Der AWS IoT Jobs-Dienst veröffentlicht Erfolgs- und Fehlernachrichten zu einem MQTT-Thema. Das Thema wird durch Anhängen von `accepted` oder `rejected` an das Thema erstellt, mit dem

die Anforderung getätigt wurde. Hier `request-name` ist der Name einer Anfrage wie `Get` und das Thema kann lauten:`$aws/things/myThing/jobs/get`. AWS IoT Jobs veröffentlicht dann Erfolgsmeldungen zum `$aws/things/myThing/jobs/get/accepted` Thema.

- **`$aws/things/`*`Name der Sache`*`/jobs/`*`Name der Anfrage`*`/abgelehnt`**

  Hier ist `request-name` der Name Ihrer Region, z. B. `Get`. Wenn die Anfrage fehlgeschlagen ist, veröffentlicht AWS IoT Jobs Fehlermeldungen zu dem `$aws/things/myThing/jobs/get/rejected` Thema.

Sie können auch die folgenden HTTPS API-Operationen verwenden:

- Den Status einer Auftragsausführung durch Aufruf der `UpdateJobExecution`-API aktualisieren.
- Den Status einer Auftragsausführung durch Aufruf der `DescribeJobExecution`-API abfragen.
- Eine Liste der ausstehenden Auftragsausführung durch Aufruf der `GetPendingJobExecutions`-API abrufen.
- Die nächste ausstehende Auftragsausführung durch Aufruf der `DescribeJobExecution`-API mit `jobId` als `$next` abrufen.
- Die nächste ausstehende Auftragsausführung durch Aufruf der `StartNextPendingJobExecution`-API abrufen und starten.

Verwendung der HTTP-Signaturversion 4

Geräte können mithilfe der HTTP-Signatur Version 4 auf Port 443 mit AWS IoT Jobs kommunizieren. Dies ist die Methode, die von den AWS SDKs und der CLI verwendet wird. Weitere Informationen zu diesen Tools finden Sie unter AWS CLI Befehlsreferenz: iot-jobs-data oder AWS SDKs und Tools sowie im IotJobsDataPlane Abschnitt für Ihre bevorzugte Sprache.

Bei dieser Kommunikationsmethode verwendet Ihr Gerät IAM-Anmeldeinformationen, um sich bei Jobs zu authentifizieren. AWS IoT

Für diese Methode stehen die folgenden Befehle zur Verfügung:

- DescribeJobExecution

  ```
  aws iot-jobs-data describe-job-execution ...
  ```

- GetPendingJobExecutions

```
aws iot-jobs-data get-pending-job-executions ...
```

- StartNextPendingJobExecution

```
aws iot-jobs-data start-next-pending-job-execution ...
```

- UpdateJobExecution

```
aws iot-jobs-data update-job-execution ...
```

Verwendung von HTTP TLS

Geräte können mithilfe eines Drittanbieter-Softwareclients, der dieses Protokoll unterstützt, über HTTP TLS an Port 8443 mit AWS IoT Jobs kommunizieren.

Bei dieser Methode verwendet Ihr Gerät eine X.509 zertifikatsbasierte Authentifizierung (z. B. das gerätespezifische Zertifikat und der private Schlüssel).

Für diese Methode stehen die folgenden Befehle zur Verfügung:

- DescribeJobExecution
- GetPendingJobExecutions
- StartNextPendingJobExecution
- UpdateJobExecution

Programmieren von Geräten zur Arbeit mit Aufträgen

Die Beispiele in diesem Abschnitt verwenden MQTT, um zu veranschaulichen, wie ein Gerät mit dem AWS IoT Jobs-Service zusammenarbeitet. Sie können auch die entsprechenden API- oder CLI-Befehle verwenden. Bei diesen Beispielen wird davon ausgegangen, dass ein Gerät mit der Bezeichnung `MyThing` die folgenden MQTT-Themen abonniert hat:

- `$aws/things/MyThing/jobs/notify` (oder `$aws/things/MyThing/jobs/notify-next`)
- `$aws/things/MyThing/jobs/get/accepted`
- `$aws/things/MyThing/jobs/get/rejected`
- `$aws/things/MyThing/jobs/jobId/get/accepted`
- `$aws/things/MyThing/jobs/jobId/get/rejected`

Wenn Sie die Codesignatur für verwenden AWS IoT, muss Ihr Gerätecode die Signatur Ihrer Codedatei überprüfen. Die Signatur befindet sich in dem Auftragsdokument in der `codesign`-Eigenschaft. Weitere Informationen zum Verifizieren einer Codedatei-Signatur finden Sie unter Geräte-Agent-Beispiel.

Themen

- Geräteworkflow
- Arbeitsablauf für Aufträge
- Auftragsbenachrichtigungen

Geräteworkflow

Ein Gerät kann Aufträge, die es ausführt, auf eine der folgenden Arten verarbeiten.

- Holen Sie sich den nächsten Auftrag

  1. Wenn ein Gerät online geht, sollte es das `notify-next`-Thema des Gerätes abonnieren.
  2. Rufen Sie die DescribeJobExecution-MQTT-API mit jobId `$next` auf, um den nächsten Auftrag, sein Auftragsdokument und andere Details abzurufen, einschließlich eines eventuell in `statusDetails` gespeicherten Status. Wenn das Auftragsdokument eine Codedateisignatur hat, müssen Sie die Signatur verifizieren, bevor Sie mit der Verarbeitung des Auftrags fortfahren.
  3. Rufen Sie die UpdateJobExecution-MQTT-API auf, um den Auftragsstatus zu aktualisieren. Zur Kombination dieses und des vorherigen Schrittes in einem Aufruf kann das Gerät auch StartNextPendingJobExecution aufrufen.
  4. (Optional) Sie können einen Schritt-Timer hinzufügen, indem Sie einen Wert für `stepTimeoutInMinutes` angeben, wenn Sie UpdateJobExecution oder StartNextPendingJobExecution aufrufen.
  5. Führen Sie die in dem Auftragsdokument angegebenen Aktionen mit der UpdateJobExecution-MQTT-API durch, um über den Fortschritt des Auftrags zu berichten.
  6. Überwachen Sie die Auftragsausführung durch Aufruf der DescribeJobExecution MQTT-API mit dieser jobId. Wenn die Auftragsausführung gelöscht wird, gibt ein DescribeJobExecution einen `ResourceNotFoundException` aus.

     Das Gerät sollte in der Lage sein, einen gültigen Zustand wiederherzustellen, wenn die Auftragsausführung abgebrochen oder gelöscht wird, während das Gerät den Auftrag ausführt.

7. Rufen Sie die UpdateJobExecution-MQTT-API erneut auf, wenn Sie den Auftrag abgeschlossen haben, um den Auftragsstatus zu aktualisieren und Erfolg oder Fehlschlag zu melden.
8. Da der Auftragsstatus dieses Auftrags zu einem terminalen Status geändert wurde, ändert sich der nächste zur Ausführung anstehende Auftrag (falls vorhanden). Das Gerät wird benachrichtigt, dass sich die nächste ausstehende Auftragsausführung geändert hat. An diesem Punkt sollte das Gerät wie in Schritt 2 beschrieben fortfahren.

Wenn das Gerät online bleibt, erhält es weiterhin Benachrichtigungen über die nächste ausstehende Auftragsausführung. Dazu gehören auch die Daten zur Auftragsausführung, wenn ein Auftrag abgeschlossen ist oder wenn eine neue ausstehende Auftragsausführung hinzugefügt wird. Wenn dies eintritt, fährt das Gerät wie in Schritt 2 beschrieben fort.

- Wählen Sie aus verfügbaren Aufträgen

1. Wenn ein Gerät online geht, sollte es das `notify`-Thema des Objekts abonnieren.
2. Rufen Sie die GetPendingJobExecutions-MQTT-API auf, um eine Liste der ausstehenden Auftragsausführungen abzurufen.
3. Wenn die Liste eine oder mehrere Auftragsausführungen enthält, wählen Sie eine davon.
4. Rufen Sie die DescribeJobExecution-MQTT-API ab, um das Auftragsdokument und andere Details abzurufen, einschließlich eventueller in `statusDetails` gespeicherter Status.
5. Rufen Sie die UpdateJobExecution-MQTT-API auf, um den Auftragsstatus zu aktualisieren. Wenn das Feld `includeJobDocument` in diesem Befehl auf `true` gesetzt ist, kann das Gerät den vorherigen Schritt übergehen und an diesem Punkt das Auftragsdokument abrufen.
6. Optional können Sie einen Schritt-Timer hinzufügen, indem Sie einen Wert für `stepTimeoutInMinutes` angeben, wenn Sie UpdateJobExecution aufrufen.
7. Führen Sie die in dem Auftragsdokument angegebenen Aktionen mit der UpdateJobExecution-MQTT-API durch, um über den Fortschritt des Auftrags zu berichten.
8. Überwachen Sie die Auftragsausführung durch Aufruf der DescribeJobExecution MQTT-API mit dieser jobId. Wenn die Auftragsausführung abgebrochen oder gelöscht werden, während das Gerät den Auftrag ausführt, sollte das Gerät in der Lage sein, die Wiederherstellung in einen gültigen Zustand durchzuführen.
9. Rufen Sie die UpdateJobExecution-MQTT-API erneut auf, wenn Sie den Auftrag abgeschlossen haben, um den Auftragsstatus zu aktualisieren und Erfolg oder Fehlschlag zu melden.

Wenn das Gerät online bleibt, wird es über alle ausstehenden Auftragsausführungen benachrichtigt, wenn neue ausstehende Auftragsausführungen verfügbar werden. Wenn dies eintritt, kann das Gerät wie in Schritt 2 beschrieben fortfahren.

Wenn das Gerät den Auftrag nicht ausführen kann, sollte es die UpdateJobExecution-MQTT-API aufruft, um den Auftragsstatus zu `REJECTED` aktualisieren.

Arbeitsablauf für Aufträge

Im Folgenden werden die verschiedenen Schritte des Auftragsworkflows vom Starten eines neuen Auftrags bis hin zur Meldung des Abschlussstatus einer Auftragsausführung dargestellt.

Starten Sie einen neuen Auftrag

Wenn ein neuer Job erstellt wird, veröffentlicht AWS IoT Jobs für jedes Zielgerät eine Nachricht `$aws/things/`*`thing-name`*`/jobs/notify` zum Thema.

Die Nachricht enthält die folgenden Informationen:

```
{
    "timestamp":1476214217017,
    "jobs":{
        "QUEUED":[{
            "jobId":"0001",
            "queuedAt":1476214216981,
            "lastUpdatedAt":1476214216981,
            "versionNumber" : 1
        }]
    }
}
```

Das Gerät erhält diese Nachricht auf dem `'$aws/things/`*`thingName`*`/jobs/notify'`-Thema, wenn die Auftragsausführung in die Warteschlange gesetzt wird.

> **Note**
>
> Bei Aufträgen mit dem optionalen `SchedulingConfig` behält der Auftrag den ursprünglichen `SCHEDULED`-Status bei. Wenn der Auftrag den ausgewählten `startTime` erreicht, passiert Folgendes:

- Der Status des Auftragsstatus wird auf `IN_PROGRESS` aktualisiert.
- Der Auftrag beginnt mit dem Rollout des Auftragsdokuments auf alle Geräte der Zielgruppe.

Auftragsinformationen abrufen

Um weitere Informationen zu einer Auftragsausführung abzurufen, ruft das Gerät die DescribeJobExecution-MQTT-API mit dem Wert `true` in dem Feld `includeJobDocument` ab (Standard).

Wenn die Anfrage erfolgreich ist, veröffentlicht der AWS IoT Jobs-Dienst eine Nachricht zum `$aws/things/MyThing/jobs/0023/get/accepted` Thema:

```
{
    "clientToken" : "client-001",
    "timestamp" : 1489097434407,
    "execution" : {
        "approximateSecondsBeforeTimedOut": number,
        "jobId" : "023",
        "status" : "QUEUED",
        "queuedAt" : 1489097374841,
        "lastUpdatedAt" : 1489097374841,
        "versionNumber" : 1,
        "jobDocument" : {
            < contents of job document >
        }
    }
}
```

Schlägt die Anfrage fehl, veröffentlicht der AWS IoT Jobs-Service eine Nachricht zum `$aws/things/MyThing/jobs/0023/get/rejected` Thema.

Das Gerät verfügt jetzt über das Auftragsdokument, das es verwenden kann, um die Remoteoperationen für den Auftrag durchzuführen. Wenn das Auftragsdokument eine vorsignierte Amazon S3-URL enthält, kann das Gerät diese URL zum Download aller für den Auftrag erforderlicher Dateien verwenden.

Melden des Status der Auftragsausführung

Wenn das Gerät den Auftrag ausführt, kann es die UpdateJobExecution-MQTT-API aufrufen, um den Status der Auftragsausführung zu aktualisieren.

Beispielsweise kann ein Gerät den Status der Auftragsausführung zu `IN_PROGRESS` aktualisieren, indem es die folgende Nachricht auf dem Thema `$aws/things/MyThing/jobs/0023/update` veröffentlicht:

```
{
    "status":"IN_PROGRESS",
    "statusDetails": {
        "progress":"50%"
    },
    "expectedVersion":"1",
    "clientToken":"client001"
}
```

Jobs reagiert mit der Veröffentlichung einer Nachricht auf das Thema `$aws/things/MyThing/jobs/0023/update/accepted` oder `$aws/things/MyThing/jobs/0023/update/rejected`:

```
{
    "clientToken":"client001",
    "timestamp":1476289222841
}
```

Das Gerät kann die beiden vorherigen Anforderungen durch den Aufruf von StartNextPendingJobExecution kombinieren. Dadurch wird die nächste ausstehende Auftragsausführung abgerufen und gestartet, und das Gerät kann den Auftragsausführungsstatus aktualisieren. Diese Anforderung gibt auch das Auftragsdokument aus, wenn eine Auftragsausführung aussteht.

Wenn der Job eine enthält, beginnt der Timer TimeoutConfig, der gerade läuft, zu laufen. Sie können auch einen Schritttimer für die Auftragsausführung festlegen, indem Sie einen Wert für den `stepTimeoutInMinutes` Zeitpunkt des Aufrufs UpdateJobExecution festlegen. Der Schritt Timer gilt nur für die Auftragsausführung, die Sie aktualisieren. Sie können einen neuen Wert für diesen Timer bei jeder Aktualisierung einer Auftragsausführung einrichten. Sie können auch einen Schritttimer erstellen, wenn Sie anrufen StartNextPendingJobExecution. Falls die Auftragsausführung länger als das Schritt-Timer-Intervall im Status `IN_PROGRESS` bleibt, schlägt sie

fehl und wechselt in den terminalen Status `TIMED_OUT`. Der Schritt-Timer hat keine Auswirkungen auf den Timer „In Bearbeitung“, den Sie beim Erstellen eines Auftrags festlegen.

Das Feld `status` kann auf `IN_PROGRESS`, `SUCCEEDED` oder `FAILED` gesetzt werden. Der Status einer bereits im Status „Terminal“ befindlichen Auftragsausführung kann nicht aktualisiert werden.

Meldung des Abschlusses der Ausführung

Wenn das Gerät die Ausführung des Auftrags abgeschlossen hat, ruft es die UpdateJobExecution-MQTT-API ab. Wenn der Auftrag erfolgreich war, setzen Sie `status` auf `SUCCEEDED`, und fügen Sie in `statusDetails` in der Nachrichtnutzlast weitere Informationen zu dem Auftrag als Name/Wert-Paare hinzu. Die Timer „In Bearbeitung“ und „Schritt“ enden, wenn die Auftragsausführung abgeschlossen ist.

Beispiel:

```
{
    "status":"SUCCEEDED",
    "statusDetails": {
        "progress":"100%"
    },
    "expectedVersion":"2",
    "clientToken":"client-001"
}
```

Wenn der Auftrag nicht erfolgreich war, setzen Sie `status` auf `FAILED`, und fügen Sie in `statusDetails`Informationen zu dem aufgetretenen Fehler hinzu:

```
{
    "status":"FAILED",
    "statusDetails": {
        "errorCode":"101",
        "errorMsg":"Unable to install update"
    },
    "expectedVersion":"2",
    "clientToken":"client-001"
}
```

> **Note**
>
> Das Attribut `statusDetails` kann eine beliebige Zahl von Name/Wert-Paaren enthalten.

Wenn der AWS IoT Job-Service dieses Update erhält, veröffentlicht er eine Meldung zum `$aws/things/MyThing/jobs/notify` Thema, um darauf hinzuweisen, dass die Auftragsausführung abgeschlossen ist:

```
{
    "timestamp":1476290692776,
    "jobs":{}
}
```

Zusätzliche Aufträge

Wenn für das Gerät weitere Auftragsausführungen ausstehen, sind diese in der Nachricht enthalten, die für `$aws/things/MyThing/jobs/notify` veröffentlicht wurde.

Beispiel:

```
{
    "timestamp":1476290692776,
    "jobs":{
        "QUEUED":[{
            "jobId":"0002",
            "queuedAt":1476290646230,
            "lastUpdatedAt":1476290646230
        }],
        "IN_PROGRESS":[{
            "jobId":"0003",
            "queuedAt":1476290646230,
            "lastUpdatedAt":1476290646230
        }]
    }
}
```

Auftragsbenachrichtigungen

Der AWS IoT Jobs-Dienst veröffentlicht MQTT-Nachrichten in reservierten Themen, wenn Jobs ausstehen oder wenn sich die erste Jobausführung in der Liste ändert. Geräte können ausstehende Aufträge verfolgen, indem sie diese Themen abonnieren.

Auftrag-Benachrichtigungstypen

Auftragsbenachrichtigungen werden als JSON-Nutzlasten in MQTT-Themen veröffentlicht. Es gibt zwei Arten von Benachrichtigungen:

ListNotification

Eine `ListNotification` enthält eine Liste von höchstens 15 ausstehenden Auftragsausführungen. Sie nach Status (`IN_PROGRESS`-Auftragsausführungen vor `QUEUED`-Auftragsausführungen) und dann nach dem Zeitpunkt sortiert, an dem sie in die Warteschlange gestellt wurden.

Eine `ListNotification` wird veröffentlicht, sobald eines der folgenden Kriterien erfüllt ist.

- Eine neue Auftragsausführung wird in die Warteschlange gestellt oder ändert sich in einen Nicht-Endstatus (`IN_PROGRESS` oder `QUEUED`).
- Eine alte Auftragsausführung ändert sich in einen Endstatus (`FAILED`, `SUCCEEDED`, `CANCELED`, `TIMED_OUT`, `REJECTED` oder `REMOVED`).

Weitere Hinweise zu den Grenzwerten mit und ohne die Schedulingkonfiguration finden Sie unterGrenzwerte für die Ausführung von Aufträgen.

NextNotification

- Eine `NextNotification` enthält zusammenfassende Informationen über die Auftragsausführung, die sich an der nächsten Stelle in der Warteschlange befindet.

  Eine `NextNotification` wird veröffentlicht, sobald sich die erste Auftragsausführung in der Liste ändert.

  - Eine neue Auftragsausführung wird der Liste als `QUEUED` hinzugefügt und ist die erste in der Liste.
  - Der Status einer vorhandenen Auftragsausführung, die nicht die erste in der Liste war, ändert sich von `QUEUED` in `IN_PROGRESS` und wird zur ersten in der Liste. (Dies passiert, wenn

sich in der Liste keine anderen `IN_PROGRESS`-Auftragsausführungen befinden und wenn die Auftragsausführung, deren Status sich von `QUEUED` in `IN_PROGRESS` geändert hat, früher als jede andere `IN_PROGRESS`-Auftragsausführung in der Liste in die Warteschlange gestellt wurde.)

- Der Status der Auftragsausführung, die sich an erster Stelle in der Liste befindet, ändert sich in den Endstatus und wird aus der Liste entfernt.

Für weitere Informationen zur Veröffentlichung und zum Abonnement von MQTT-Themen vgl. the section called “Gerätekommunikationsprotokolle”.

> **Note**
>
> Benachrichtigungen sind nicht verfügbar, wenn Sie die HTTP-Signaturversion 4 oder HTTP TLS für die Kommunikation mit Aufträgen verwenden.

Ausstehender Auftrag

Der AWS IoT Jobs-Dienst veröffentlicht eine Nachricht zu einem MQTT-Thema, wenn ein Job zur Liste der ausstehenden Jobausführungen für eine Sache hinzugefügt oder aus ihr entfernt wird oder wenn sich die erste Jobausführung in der Liste ändert:

- `$aws/things/`*`thingName`*`/jobs/notify`
- `$aws/things/`*`thingName`*`/jobs/notify-next`

Die Nachricht enthält die folgende Beispielnutzlast:

`$aws/things/`*`thingName`*`/jobs/notify`:

```
{
  "timestamp" : 10011,
  "jobs" : {
    "IN_PROGRESS" : [ {
      "jobId" : "other-job",
      "queuedAt" : 10003,
      "lastUpdatedAt" : 10009,
      "executionNumber" : 1,
      "versionNumber" : 1
    } ],
```

```
    "QUEUED" : [ {
      "jobId" : "this-job",
      "queuedAt" : 10011,
      "lastUpdatedAt" : 10011,
      "executionNumber" : 1,
      "versionNumber" : 0
    } ]
  }
}
```

Wenn die aufgerufene Auftragsausführung `this-job` aus einem Auftrag stammt, für den die optionale Planungskonfiguration ausgewählt wurde und der Rollout des Auftragsdokuments während eines Wartungsfensters geplant ist, wird sie nur in einem wiederkehrenden Wartungsfenster angezeigt. Außerhalb eines Wartungsfensters `this-job` wird der aufgerufene Auftrag aus der Liste der ausstehenden Auftragsausführungen ausgeschlossen, wie im folgenden Beispiel gezeigt.

```
{
  "timestamp" : 10011,
  "jobs" : {
    "IN_PROGRESS" : [ {
      "jobId" : "other-job",
      "queuedAt" : 10003,
      "lastUpdatedAt" : 10009,
      "executionNumber" : 1,
      "versionNumber" : 1
    } ],
    "QUEUED" : []
  }
}
```

`$aws/things/`*`thingName`*`/jobs/notify-next`:

```
{
  "timestamp" : 10011,
  "execution" : {
    "jobId" : "other-job",
    "status" : "IN_PROGRESS",
    "queuedAt" : 10009,
    "lastUpdatedAt" : 10009,
    "versionNumber" : 1,
    "executionNumber" : 1,
    "jobDocument" : {"c":"d"}
```

```
  }
}
```

Wenn die aufgerufene Auftragsausführung `other-job` aus einem Auftrag stammt, für den die optionale Planungskonfiguration ausgewählt wurde und der Rollout des Auftragsdokuments während eines Wartungsfensters geplant ist, wird sie nur in einem wiederkehrenden Wartungsfenster angezeigt. Außerhalb eines Wartungsfensters wird der `other-job` aufgerufene Auftrag nicht als nächste Auftragsausführung aufgeführt, wie im folgenden Beispiel gezeigt.

```
{} //No other pending jobs
```

```
{
  "timestamp" : 10011,
  "execution" : {
      "jobId" : "this-job",
      "queuedAt" : 10011,
      "lastUpdatedAt" : 10011,
      "executionNumber" : 1,
      "versionNumber" : 0,
      "jobDocument" : {"a":"b"}
  }
} // "this-job" is pending next to "other-job"
```

Mögliche Statuswerte für die Auftragsausführungen sind `QUEUED`, `IN_PROGRESS`, `FAILED`, `SUCCEEDED`, `CANCELED`, `TIMED_OUT`, `REJECTED` und `REMOVED`.

Die folgende Reihe von Beispielen zeigt die in jedem Thema veröffentlichten Benachrichtigungen, wenn Auftragsausführungen angelegt und von einem Status in einen anderen geändert werden.

Zuerst wird ein Auftrag mit dem Namen `job1` erstellt. Diese Benachrichtigung wird im `jobs/notify`-Thema veröffentlicht:

```
{
  "timestamp": 1517016948,
  "jobs": {
    "QUEUED": [
      {
        "jobId": "job1",
        "queuedAt": 1517016947,
        "lastUpdatedAt": 1517016947,
```

```
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
  }
}
```

Diese Benachrichtigung wird im `jobs/notify-next`-Thema veröffentlicht:

```
{
  "timestamp": 1517016948,
  "execution": {
    "jobId": "job1",
    "status": "QUEUED",
    "queuedAt": 1517016947,
    "lastUpdatedAt": 1517016947,
    "versionNumber": 1,
    "executionNumber": 1,
    "jobDocument": {
      "operation": "test"
    }
  }
}
```

Wenn ein weiterer Auftrag erstellt wird (`job2`), wird diese Benachrichtigung im `jobs/notify`-Thema veröffentlicht:

```
{
  "timestamp": 1517017192,
  "jobs": {
    "QUEUED": [
      {
        "jobId": "job1",
        "queuedAt": 1517016947,
        "lastUpdatedAt": 1517016947,
        "executionNumber": 1,
        "versionNumber": 1
      },
      {
        "jobId": "job2",
        "queuedAt": 1517017191,
        "lastUpdatedAt": 1517017191,
        "executionNumber": 1,
```

```
        "versionNumber": 1
      }
    ]
  }
}
```

Es wird keine Benachrichtigung im `jobs/notify-next`-Thema veröffentlicht, da sich der nächste Auftrag in der Warteschlange (`job1`) nicht geändert hat. Wenn die Ausführung von `job1` beginnt, ändert sich der Status zu `IN_PROGRESS`. Es werden keine Benachrichtigungen veröffentlicht, da sich die Liste der Aufträge und der nächste Auftrag in der Warteschlange nicht geändert haben.

Wenn ein dritter Auftrag hinzugefügt wird (`job3`), wird diese Benachrichtigung im `jobs/notify`-Thema veröffentlicht:

```
{
  "timestamp": 1517017906,
  "jobs": {
    "IN_PROGRESS": [
      {
        "jobId": "job1",
        "queuedAt": 1517016947,
        "lastUpdatedAt": 1517017472,
        "startedAt": 1517017472,
        "executionNumber": 1,
        "versionNumber": 2
      }
    ],
    "QUEUED": [
      {
        "jobId": "job2",
        "queuedAt": 1517017191,
        "lastUpdatedAt": 1517017191,
        "executionNumber": 1,
        "versionNumber": 1
      },
      {
        "jobId": "job3",
        "queuedAt": 1517017905,
        "lastUpdatedAt": 1517017905,
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
```

```
  }
}
```

Es wird keine Benachrichtigung im `jobs/notify-next`-Thema veröffentlicht, da der nächste Auftrag in der Warteschlange noch `job1` ist.

Wenn `job1` abgeschlossen ist, ändert sich sein Status zu `SUCCEEDED`. Diese Benachrichtigung wird im `jobs/notify`-Thema veröffentlicht:

```
{
  "timestamp": 1517186269,
  "jobs": {
    "QUEUED": [
      {
        "jobId": "job2",
        "queuedAt": 1517017191,
        "lastUpdatedAt": 1517017191,
        "executionNumber": 1,
        "versionNumber": 1
      },
      {
        "jobId": "job3",
        "queuedAt": 1517017905,
        "lastUpdatedAt": 1517017905,
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
  }
}
```

Jetzt wurde `job1` aus der Warteschlange entfernt, und die nächste auszuführenden Aufgabe ist `job2`. Diese Benachrichtigung wird im `jobs/notify-next`-Thema veröffentlicht:

```
{
  "timestamp": 1517186269,
  "execution": {
    "jobId": "job2",
    "status": "QUEUED",
    "queuedAt": 1517017191,
    "lastUpdatedAt": 1517017191,
    "versionNumber": 1,
```

```
    "executionNumber": 1,
    "jobDocument": {
      "operation": "test"
    }
  }
}
```

Wenn `job3` vor der Ausführung vor `job2` beginnen muss (was nicht empfehlenswert ist), kann der Status von `job3` zu `IN_PROGRESS` geändert werden. Wenn dies geschieht, ist `job2` nicht mehr der nächste Auftrag in der Warteschlange. Diese Benachrichtigung wird im `jobs/notify-next`-Thema veröffentlicht:

```
{
  "timestamp": 1517186779,
  "execution": {
    "jobId": "job3",
    "status": "IN_PROGRESS",
    "queuedAt": 1517017905,
    "startedAt": 1517186779,
    "lastUpdatedAt": 1517186779,
    "versionNumber": 2,
    "executionNumber": 1,
    "jobDocument": {
      "operation": "test"
    }
  }
}
```

Es wird keine Benachrichtigung im `jobs/notify`-Thema veröffentlicht, da kein Auftrag hinzugefügt oder entfernt wurde.

Wenn das Gerät `job2` ablehnt und seinen Status zu `REJECTED` aktualisiert, wird diese Benachrichtigung im `jobs/notify`-Thema veröffentlicht:

```
{
  "timestamp": 1517189392,
  "jobs": {
    "IN_PROGRESS": [
      {
        "jobId": "job3",
        "queuedAt": 1517017905,
        "lastUpdatedAt": 1517186779,
```

```
        "startedAt": 1517186779,
        "executionNumber": 1,
        "versionNumber": 2
      }
    ]
  }
}
```

Wenn das Löschen von `job3` (der noch ausgeführt wird) erzwungen wird, wird diese Benachrichtigung im `jobs/notify`-Thema veröffentlicht:

```
{
  "timestamp": 1517189551,
  "jobs": {}
}
```

An diesem Punkt ist die Warteschlange leer. Diese Benachrichtigung wird im `jobs/notify-next`-Thema veröffentlicht:

```
{
  "timestamp": 1517189551
}
```

AWS IoT Jobs API-Operationen

AWS IoT Die Jobs-API kann für jede der folgenden Kategorien verwendet werden:

- Administrative Aufgaben wie die Verwaltung und Kontrolle von Aufträgen. Dies ist die Steuerebene.
- Geräte, die diese Aufgaben ausführen. Dies ist die Datenebene, mit der Sie Daten senden und empfangen können.

Die Auftragsverwaltung und die Auftragssteuerung verwenden eine HTTPS-Protokoll-API. Geräte können eine MQTT- oder eine HTTPS-Protokoll-API verwenden. Die Steuerebenen-API ist für eine geringe Anzahl von Aufrufen ausgelegt, wie sie für die Erstellung und Verfolgung von Aufträgen typisch sind. In der Regel öffnet sie eine Verbindung für eine einzelne Anforderung und schließt diese dann nach dem Eingang der Antwort. Die Datenebene HTTPS und die MQTT-API ermöglichen lange Abfragen. Diese API-Vorgänge sind für große Datenmengen ausgelegt, die auf Millionen von Geräten skaliert werden können.

Jede AWS IoT Jobs-HTTPS-API hat einen entsprechenden Befehl, mit dem Sie die API über AWS Command Line Interface (AWS CLI) aufrufen können. Die Befehle werden in Kleinbuchstaben mit Bindestrichen zwischen den Wörtern, aus denen die API besteht, ausgedrückt. Beispielsweise können Sie die `CreateJob`-API in der CLI aufrufen, indem Sie Folgendes eingeben:

```
aws iot create-job ...
```

Wenn während eines Vorgangs ein Fehler auftritt, erhalten Sie eine Fehlerantwort, die Informationen über den Fehler enthält.

ErrorResponse

Enthält Informationen zu einem Fehler, der während einer Operation des AWS IoT Jobs-Service aufgetreten ist.

Das folgende Beispiel zeigt die Syntax dieser Operation:

```
{
    "code": "ErrorCode",
    "message": "string",
    "clientToken": "string",
    "timestamp": timestamp,
    "executionState": JobExecutionState
}
```

Im Folgenden finden Sie eine Beschreibung dieser `ErrorResponse`:

`code`

ErrorCode kann wie folgt gesetzt werden:

InvalidTopic

Die Anfrage wurde an ein Thema im AWS IoT Jobs-Namespace gesendet, das keiner API-Operation zugeordnet ist.

InvalidJson

Der Inhalt der Anfrage konnte nicht als gültiges UTF-8-encoded JSON interpretiert werden.

InvalidRequest

Der Inhalt der Anforderung war nicht gültig. Dieser Code wird beispielsweise ausgegeben, wenn eine `UpdateJobExecution`-Anforderung ungültige Statusdetails enthält. Die Mitteilung enthält Einzelheiten zu dem Fehler.

InvalidStateTransition

Bei einem Update wurde versucht, die Auftragsausführung in einen Status zu ändern, der aufgrund des aktuellen Status der Auftragsausführung nicht gültig ist. Beispiel: Ein Versuch, eine Anforderung im Status SUCCEED in den Status IN_PROGRESS zu ändern. In diesem Fall enthält der Text der Fehlermeldung auch das Feld `executionState`.

ResourceNotFound

Die von dem Anforderungsthema angegebene `JobExecution` ist nicht vorhanden.

VersionMismatch

Die in der Anforderung angegebene erwartete Version entspricht nicht der Version der Auftragsausführung im AWS IoT Jobs-Service. In diesem Fall enthält der Text der Fehlermeldung auch das Feld `executionState`.

InternalError

Bei der Verarbeitung der Anforderung ist ein interner Fehler aufgetreten.

RequestThrottled

Die Anforderung wurde gedrosselt.

TerminalStateReached

Tritt auf, wenn ein Befehl zum Beschreiben eines Auftrags für einen Auftrag im Status „Terminal“ durchgeführt wird.

`message`

Eine Fehlermeldungszeichenfolge.

`clientToken`

Eine beliebige Zeichenfolge für die Korrelierung einer Anforderung mit der jeweiligen Antwort.

`timestamp`

Die seit der Epoche vergangene Zeit (in Sekunden).

`executionState`

Ein `JobExecutionState`-Objekt. Dieses Feld ist nur enthalten, wenn das Feld `code` den Wert `InvalidStateTransition` oder `VersionMismatch` hat. Dadurch ist es in diesen Fällen nicht erforderlich, eine separate `DescribeJobExecution`-Anforderung durchzuführen, um die Daten zum Status der aktuellen Auftragsausführung abzurufen.

Im Folgenden werden die Operationen und Datentypen der Jobs API aufgeführt.

- Auftragsverwaltung und -steuerung API und Datentypen
- Aufträge, Geräte, MQTT- und HTTPS-API-Operationen und Datentypen

Auftragsverwaltung und -steuerung API und Datentypen

Die folgenden Befehle sind für die Verwaltung und Steuerung von Aufträgen in der CLI und über das HTTPS-Protokoll verfügbar.

- Datentypen für Auftragsverwaltung und -steuerung
- API-Operationen zur Auftragsverwaltung und -steuerung

Führen Sie diesen Befehl aus, um den *endpoint-url* Parameter für Ihre CLI-Befehle zu ermitteln.

```
aws iot describe-endpoint --endpoint-type=iot:Jobs
```

Dieser Befehl gibt die folgende Ausgabe zurück.

```
{
"endpointAddress": "account-specific-prefix.jobs.iot.aws-region.amazonaws.com"
}
```

> **Note**
>
> Der Jobs-Endpunkt unterstützt ALPN `x-amzn-http-ca` nicht.
> Wenn Sie Dual-Stack-Endpunkte (IPv6 und IPv6) verwenden, verwenden Sie den Endpunkt. `iot:Data-ATS` Der `iot:Jobs` Endpunkt unterstützt nur IPv4.

Datentypen für Auftragsverwaltung und -steuerung

Die folgenden Datentypen werden von Verwaltungs- und Steuerungsanwendungen für die Kommunikation mit AWS IoT Jobs verwendet.

Aufgabe

Das Objekt `Job` enthält Details zu einem Auftrag. Im folgenden Beispiel wird die Syntax dargestellt:

```
{
    "jobArn": "string",
    "jobId": "string",
    "status": "IN_PROGRESS|CANCELED|SUCCEEDED",
    "forceCanceled": boolean,
    "targetSelection": "CONTINUOUS|SNAPSHOT",
    "comment": "string",
    "targets": ["string"],
    "description": "string",
    "createdAt": timestamp,
    "lastUpdatedAt": timestamp,
    "completedAt": timestamp,
    "jobProcessDetails": {
        "processingTargets": ["string"],
        "numberOfCanceledThings": long,
        "numberOfSucceededThings": long,
        "numberOfFailedThings": long,
        "numberOfRejectedThings": long,
        "numberOfQueuedThings": long,
        "numberOfInProgressThings": long,
        "numberOfRemovedThings": long,
        "numberOfTimedOutThings": long
    },
    "presignedUrlConfig": {
        "expiresInSec": number,
        "roleArn": "string"
    },
    "jobExecutionsRolloutConfig": {
        "exponentialRate": {
           "baseRatePerMinute": integer,
           "incrementFactor": integer,
           "rateIncreaseCriteria": {
              "numberOfNotifiedThings": integer, // Set one or the other
              "numberOfSucceededThings": integer // of these two values.
           },
```

```
            "maximumPerMinute": integer
      }
    },
    "abortConfig": {
       "criteriaList": [
          {
             "action": "string",
             "failureType": "string",
             "minNumberOfExecutedThings": integer,
             "thresholdPercentage": integer
          }
       ]
    },
    "SchedulingConfig": {
      "startTime": string
      "endTime": string
      "timeZone": string


      "endTimeBehavior": string

   },
    "timeoutConfig": {
        "inProgressTimeoutInMinutes": long
    }
}
```

Für weitere Informationen siehe Job oder job.

JobSummary

Das Objekt JobSummary enthält eine Auftragszusammenfassung. Im folgenden Beispiel wird die Syntax dargestellt:

```
{
    "jobArn": "string",
    "jobId": "string",
    "status": "IN_PROGRESS|CANCELED|SUCCEEDED|SCHEDULED",
    "targetSelection": "CONTINUOUS|SNAPSHOT",
    "thingGroupId": "string",
    "createdAt": timestamp,
    "lastUpdatedAt": timestamp,
    "completedAt": timestamp
```

```
}
```

Für weitere Informationen siehe JobSummary oder job-summary.

JobExecution

Das Objekt JobExecution repräsentiert die Ausführung eines Auftrags auf einem Gerät. Im folgenden Beispiel wird die Syntax dargestellt:

> **Note**
>
> Wenn Sie die API-Operationen der Steuerebene verwenden, enthält der Datentyp JobExecution kein JobDocument-Feld. Um diese Informationen zu erhalten, können Sie die GetJobDocument API-Operation oder den get-job-document CLI-Befehl verwenden.

```
{
    "approximateSecondsBeforeTimedOut": 50,
    "executionNumber": 1234567890,
    "forceCanceled": true|false,
    "jobId": "string",
    "lastUpdatedAt": timestamp,
    "queuedAt": timestamp,
    "startedAt": timestamp,
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|
REMOVED",
    "forceCanceled": boolean,
    "statusDetails": {
        "detailsMap": {
            "string": "string" ...
        },
        "status": "string"
    },
    "thingArn": "string",
    "versionNumber": 123
}
```

Für weitere Informationen siehe JobExecution oder job-execution.

JobExecutionSummary

Das `JobExecutionSummary`-Objekt enthält zusammenfassende Informationen zur Auftragsausführung. Im folgenden Beispiel wird die Syntax dargestellt:

```
{
    "executionNumber": 1234567890,
    "queuedAt": timestamp,
    "lastUpdatedAt": timestamp,
    "startedAt": timestamp,
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|REMOVED"
}
```

Für weitere Informationen siehe `JobExecutionSummary` oder `job-execution-summary`.

JobExecutionSummaryForJob

Das Objekt `JobExecutionSummaryForJob` enthält eine Zusammenfassung der Informationen zu Auftragsausführungen für einen bestimmten Auftrag. Im folgenden Beispiel wird die Syntax dargestellt:

```
{
    "executionSummaries": [
        {
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyThing",
            "jobExecutionSummary": {
                "status": "IN_PROGRESS",
                "lastUpdatedAt": 1549395301.389,
                "queuedAt": 1541526002.609,
                "executionNumber": 1
            }
        },
        ...
    ]
}
```

Für weitere Informationen siehe `JobExecutionSummaryForJob` oder `job-execution-summary-for-job`.

JobExecutionSummaryForThing

Das Objekt `JobExecutionSummaryForThing` enthält eine Zusammenfassung der Informationen zu einer Auftragsausführung für ein bestimmtes Objekt. Im folgenden Beispiel wird die Syntax dargestellt:

```
{
    "executionSummaries": [
        {
            "jobExecutionSummary": {
                "status": "IN_PROGRESS",
                "lastUpdatedAt": 1549395301.389,
                "queuedAt": 1541526002.609,
                "executionNumber": 1
            },
            "jobId": "MyThingJob"
        },
        ...
    ]
}
```

Für weitere Informationen siehe `JobExecutionSummaryForThing` oder `job-execution-summary-for-thing`.

API-Operationen zur Auftragsverwaltung und -steuerung

Verwenden Sie die folgenden API-Vorgänge oder CLI-Befehle:

AssociateTargetsWithJob

Weist eine Gruppe einem kontinuierlichen Auftrag zu. Die folgenden Kriterien müssen erfüllt sein:

- Der Auftrag muss mit der Einstellung des Feldes `targetSelection` auf `CONTINUOUS` erstellt worden sein.
- Der Auftrag muss den Status `IN_PROGRESS` haben.
- Die Gesamtzahl der mit einem Auftrag verbundenen Ziele darf 100 nicht überschreiten.

HTTPS request

```
POST /jobs/jobId/targets
```

```
{
"targets": [ "string" ],
"comment": "string"
}
```

Weitere Informationen finden Sie unter AssociateTargetsWithJob.

CLI syntax

```
aws iot  associate-targets-with-job \
--targets <value> \
--job-id <value> \
[--comment <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"targets": [
"string"
],
"jobId": "string",
"comment": "string"
}
```

Weitere Informationen finden Sie unter associate-targets-with-job.

CancelJob

Bricht einen Auftrag ab.

HTTPS request

```
PUT /jobs/jobId/cancel

{
"force": boolean,
"comment": "string",
"reasonCode": "string"
```

```
}
```

Weitere Informationen finden Sie unter CancelJob.

CLI syntax

```
aws iot cancel-job \
    --job-id <value> \
    [--force <value>]  \
    [--comment <value>]  \
    [--reasonCode <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
    "jobId": "string",
    "force": boolean,
    "comment": "string"
}
```

Weitere Informationen finden Sie unter cancel-job.

CancelJobExecution

Bricht eine Auftragsausführung auf einem Gerät ab.

HTTPS request

```
PUT /things/thingName/jobs/jobId/cancel

{
"force": boolean,
"expectedVersion": "string",
"statusDetails": {
    "string": "string"
    ...
}
}
```

Weitere Informationen finden Sie unter CancelJobExecution.

CLI syntax

```
aws iot cancel-job-execution \
--job-id <value> \
--thing-name <value> \
[--force | --no-force] \
[--expected-version <value>] \
[--status-details <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"jobId": "string",
"thingName": "string",
"force": boolean,
"expectedVersion": long,
"statusDetails": {
"string": "string"
}
}
```

Weitere Informationen finden Sie unter `cancel-job-execution`.

CreateJob

Erstellt einen Auftrag. Sie können das Auftragsdokument als Link zu einer Datei in einem Amazon S3-Bucket (Parameter `documentSource`) oder im Text der Anfrage (Parameter `document`) bereitstellen.

Ein Auftrag kann kontinuierlich gemacht werden, indem Sie den optionalen Parameter `targetSelection` auf `CONTINUOUS` setzen (der Standard ist `SNAPSHOT`). Ein kontinuierlicher Auftrag kann verwendet werden, um Geräte zu integrieren oder zu aktualisieren, wenn sie zu einer Gruppe hinzugefügt werden, da er weiterhin ausgeführt wird und für neu hinzugefügte Objekte gestartet wird. Dies kann auch dann der Fall sein, wenn die Objekte, die sich zum Zeitpunkt der Auftragserstellung in der Gruppe befanden, den Auftrag abgeschlossen haben.

Ein Job kann einen optionalen Wert haben TimeoutConfig, der den Wert des in Bearbeitung befindlichen Timers festlegt. Der Timer für „In Bearbeitung“ kann nicht aktualisiert werden und gilt für alle Ausführungen des Auftrags.

Die folgenden Validierungen werden auf Argumenten der `CreateJob`-API durchgeführt:

- Das Argument `targets` muss eine Liste gültiger Objekt- oder Objektgruppen-ARNs sein. Alle Dinge und Dinggruppen müssen in deinem AWS-Konto sein.
- Das Argument `documentSource` muss eine gültige Amazon S3-URL zu einem Auftragsdokument sein. Amazon S3-URLs haben das folgende Format: `https://s3.amazonaws.com/`*`bucketName`*`/`*`objectName`*.
- Das Dokument, das in der durch das `documentSource` Argument angegebenen URL gespeichert ist, muss ein UTF-8 codiertes JSON-Dokument sein.
- Die Größe eines Auftragsdokuments ist aufgrund der Größenbeschränkung für eine MQTT-Nachricht (128 KB) und die Verschlüsselung auf 32 KB begrenzt.
- Das `jobId` muss in Ihrem AWS-Konto einzigartig sein.

HTTPS request

```
PUT /jobs/jobId

{
"targets": [ "string" ],
"document": "string",
"documentSource": "string",
"description": "string",
"jobTemplateArn": "string",
"presignedUrlConfigData": {
    "roleArn": "string",
    "expiresInSec": "integer"
},
"targetSelection": "CONTINUOUS|SNAPSHOT",
"jobExecutionsRolloutConfig": {
    "exponentialRate": {
       "baseRatePerMinute": integer,
       "incrementFactor": integer,
       "rateIncreaseCriteria": {
          "numberOfNotifiedThings": integer, // Set one or the other
          "numberOfSucceededThings": integer // of these two values.
       },
       "maximumPerMinute": integer
  }
},
"abortConfig": {
```

```
   "criteriaList": [
      {
         "action": "string",
         "failureType": "string",
         "minNumberOfExecutedThings": integer,
         "thresholdPercentage": integer
      }
   ]
},
"SchedulingConfig": {
    "startTime": string
    "endTime": string
    "timeZone": string


    "endTimeBehavior": string

   }
"timeoutConfig": {
  "inProgressTimeoutInMinutes": long
}
}
```

Weitere Informationen finden Sie unter `CreateJob`.

CLI syntax

```
aws iot create-job \
    --job-id <value> \
    --targets <value> \
    [--document-source <value>] \
    [--document <value>] \
    [--description <value>] \
    [--job-template-arn <value>] \
    [--presigned-url-config <value>] \
    [--target-selection <value>] \
    [--job-executions-rollout-config <value>] \
    [--abort-config <value>] \
    [--timeout-config <value>] \
    [--document-parameters <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
    "jobId": "string",
    "targets": [ "string" ],
    "documentSource": "string",
    "document": "string",
    "description": "string",
    "jobTemplateArn": "string",
    "presignedUrlConfig": {
        "roleArn": "string",
        "expiresInSec": long
     },
    "targetSelection": "string",
    "jobExecutionsRolloutConfig": {
          "exponentialRate": {
              "baseRatePerMinute": integer,
              "incrementFactor": integer,
              "rateIncreaseCriteria": {
                 "numberOfNotifiedThings": integer, // Set one or the other
                 "numberOfSucceededThings": integer // of these two values.
              },
      "maximumPerMinute": integer
      }
    },
    "abortConfig": {
    "criteriaList": [
        {
           "action": "string",
           "failureType": "string",
           "minNumberOfExecutedThings": integer,
           "thresholdPercentage": integer
         }
      ]
    },
    "timeoutConfig": {
          "inProgressTimeoutInMinutes": long
    },
    "documentParameters": {
    "string": "string"
    }
}
```

Weitere Informationen finden Sie unter [create-job](#).

DeleteJob

Löscht einen Auftrag und die damit verbundenen Auftragsausführungen.

Das Löschen eines Auftrags kann einige Zeit in Anspruch nehmen, abhängig von der Anzahl der Auftragsausführungen für den Auftrag und verschiedenen anderen Faktoren. Während der Auftrag gelöscht wird, wird der Status des Auftrags als „DELETION_IN_PROGRESS“ angezeigt. Der Versuch, einen Auftrag zu löschen oder abzubrechen, dessen Status bereits „DELETION_IN_PROGRESS“ ist, führt zu einem Fehler.

HTTPS request

```
DELETE /jobs/jobId?force=force
```

Weitere Informationen finden Sie unter DeleteJob.

CLI syntax

```
aws iot  delete-job \
--job-id <value> \
[--force | --no-force]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"jobId": "string",
"force": boolean
}
```

Weitere Informationen finden Sie unter delete-job.

DeleteJobExecution

Löscht eine Auftragsausführung.

HTTPS request

```
DELETE /things/thingName/jobs/jobId/executionNumber/executionNumber?force=force
```

Weitere Informationen finden Sie unter DeleteJobExecution.

CLI syntax

```
aws iot  delete-job-execution \
--job-id <value> \
--thing-name <value> \
--execution-number <value> \
[--force | --no-force]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"jobId": "string",
"thingName": "string",
"executionNumber": long,
"force": boolean
}
```

Weitere Informationen finden Sie unter delete-job-execution.

DescribeJob

Ruft die Details der Auftragsausführung ab.

HTTPS request

```
GET /jobs/jobId
```

Weitere Informationen finden Sie unter DescribeJob.

CLI syntax

```
aws iot describe-job \
--job-id <value>  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"jobId": "string"
}
```

Weitere Informationen finden Sie unter [describe-job](#).

DescribeJobExecution

Ruft Details einer Auftragsausführung ab. Der Ausführungsstatus des Auftrags muss `SUCCEEDED` oder `FAILED` sein.

HTTPS request

```
GET /things/thingName/jobs/jobId?executionNumber=executionNumber
```

Weitere Informationen finden Sie unter [DescribeJobExecution](#).

CLI syntax

```
aws iot  describe-job-execution \
--job-id <value> \
--thing-name <value> \
[--execution-number <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"jobId": "string",
"thingName": "string",
"executionNumber": long
}
```

Weitere Informationen finden Sie unter [describe-job-execution](#).

GetJobDocument

Ruft das Auftragsdokument für einen Auftrag ab.

> **Note**
>
> Platzhalter-URLs werden nicht durch vorsignierte Amazon S3-URLs in dem ausgegebenen Dokument ersetzt. Vorsignierte URLs werden nur generiert, wenn der AWS IoT Jobs-Dienst eine Anfrage über MQTT erhält.

HTTPS request

```
GET /jobs/jobId/job-document
```

Weitere Informationen finden Sie unter GetJobDocument.

CLI syntax

```
aws iot get-job-document \
--job-id <value>  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"jobId": "string"
}
```

Weitere Informationen finden Sie unter get-job-document.

ListJobExecutionsForJob

Ruft eine Liste der Auftragsausführungen für einen Auftrag ab.

HTTPS request

```
GET /jobs/jobId/things?status=status&maxResults=maxResults&nextToken=nextToken
```

Weitere Informationen finden Sie unter ListJobExecutionsForJob.

CLI syntax

```
aws iot  list-job-executions-for-job \
```

```
--job-id <value> \
[--status <value>] \
[--max-results <value>] \
[--next-token <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"jobId": "string",
"status": "string",
"maxResults": "integer",
"nextToken": "string"
}
```

Weitere Informationen finden Sie unter `list-job-executions-for-job`.

ListJobExecutionsForThing

Ruft eine Liste der Auftragsausführungen für ein Objekt ab.

HTTPS request

```
GET /things/thingName/jobs?status=status&maxResults=maxResults&nextToken=nextToken
```

Weitere Informationen finden Sie unter `ListJobExecutionsForThing`.

CLI syntax

```
aws iot list-job-executions-for-thing \
--thing-name <value> \
[--status <value>] \
[--max-results <value>] \
[--next-token <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
```

```
"thingName": "string",
"status": "string",
"maxResults": "integer",
"nextToken": "string"
}
```

Weitere Informationen finden Sie unter list-job-executions-for-thing.

ListJobs

Ruft eine Liste der Jobs in Ihrem ab. AWS-Konto

HTTPS request

```
GET /jobs?
status=status&targetSelection=targetSelection&thingGroupName=thingGroupName&thingGroupId=thi
```

Weitere Informationen finden Sie unter ListJobs.

CLI syntax

```
aws iot list-jobs \
[--status <value>] \
[--target-selection <value>] \
[--max-results <value>] \
[--next-token <value>] \
[--thing-group-name <value>] \
[--thing-group-id <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"status": "string",
"targetSelection": "string",
"maxResults": "integer",
"nextToken": "string",
"thingGroupName": "string",
"thingGroupId": "string"
}
```

Weitere Informationen finden Sie unter `list-jobs`.

UpdateJob

Aktualisiert unterstützte Felder des angegebenen Auftrags. Aktualisierte Werte für `timeoutConfig` werden nur für neu begonnen Launches wirksam. Derzeit werden laufende Starts weiterhin mit der vorherigen Timeout-Konfiguration gestartet.

HTTPS request

```
PATCH /jobs/jobId
{
"description": "string",
"presignedUrlConfig": {
  "expiresInSec": number,
  "roleArn": "string"
},
"jobExecutionsRolloutConfig": {
  "exponentialRate": {
     "baseRatePerMinute": number,
     "incrementFactor": number,
     "rateIncreaseCriteria": {
        "numberOfNotifiedThings": number,
        "numberOfSucceededThings": number
     },
  "maximumPerMinute": number
  },
"abortConfig": {
  "criteriaList": [
     {
        "action": "string",
        "failureType": "string",
        "minNumberOfExecutedThings": number,
        "thresholdPercentage": number
     }
  ]
},
"timeoutConfig": {
  "inProgressTimeoutInMinutes": number
}
}
```

Weitere Informationen finden Sie unter `UpdateJob`.

CLI syntax

```
aws iot  update-job \
--job-id <value> \
[--description <value>] \
[--presigned-url-config <value>] \
[--job-executions-rollout-config <value>] \
[--abort-config <value>] \
[--timeout-config <value>] \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"description": "string",
"presignedUrlConfig": {
  "expiresInSec": number,
  "roleArn": "string"
},
"jobExecutionsRolloutConfig": {
  "exponentialRate": {
     "baseRatePerMinute": number,
     "incrementFactor": number,
     "rateIncreaseCriteria": {
        "numberOfNotifiedThings": number,
        "numberOfSucceededThings": number
     }
  },
  "maximumPerMinute": number
},
"abortConfig": {
  "criteriaList": [
     {
        "action": "string",
        "failureType": "string",
        "minNumberOfExecutedThings": number,
        "thresholdPercentage": number
     }
  ]
},
"timeoutConfig": {
```

```
  "inProgressTimeoutInMinutes": number
}
}
```

Weitere Informationen finden Sie unter [update-job](#).

Aufträge, Geräte, MQTT- und HTTPS-API-Operationen und Datentypen

Die folgenden Befehle stehen über die Protokolle MQTT und HTTPS zur Verfügung. Verwenden Sie diese API-Operationen auf der Datenebene für Geräte, die die Aufträge ausführen.

Geräte, MQTT- und HTTPS-Datentypen

Die folgenden Datentypen werden für die Kommunikation mit dem AWS IoT Jobs-Dienst über die Protokolle MQTT und HTTPS verwendet.

JobExecution

Das Objekt `JobExecution` repräsentiert die Ausführung eines Auftrags auf einem Gerät. Im folgenden Beispiel wird die Syntax dargestellt:

> **Note**
>
> Wenn Sie die API-Operationen MQTT und HTTP für die Datenebene verwenden, enthält der Datentyp `JobExecution` ein `JobDocument`-Feld. Ihre Geräte können diese Informationen verwenden, um das Auftragsdokument aus einer Auftragsausführung abzurufen.

```
{
    "jobId" : "string",
    "thingName" : "string",
    "jobDocument" : "string",
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|
REMOVED",
    "statusDetails": {
        "string": "string"
    },
    "queuedAt" : "timestamp",
    "startedAt" : "timestamp",
    "lastUpdatedAt" : "timestamp",
```

```
    "versionNumber" : "number",
    "executionNumber": long
}
```

Für weitere Informationen siehe JobExecution oder job-execution.

JobExecutionState

Das JobExecutionState enthält Informationen über den Status einer Auftragsausführung. Im folgenden Beispiel wird die Syntax dargestellt:

```
{
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|
REMOVED",
    "statusDetails": {
        "string": "string"
        ...
    }
    "versionNumber": "number"
}
```

Für weitere Informationen siehe JobExecutionState oder job-execution-state.

JobExecutionSummary

Enthält einen Teil der Informationen zu einer Auftragsausführung. Im folgenden Beispiel wird die Syntax dargestellt:

```
{
    "jobId": "string",
    "queuedAt": timestamp,
    "startedAt": timestamp,
    "lastUpdatedAt": timestamp,
    "versionNumber": "number",
    "executionNumber": long
}
```

Für weitere Informationen siehe JobExecutionSummary oder job-execution-summary.

In den folgenden Abschnitten erfahren Sie mehr über die MQTT- und HTTPS-API-Operationen:

- Führt MQTT-API-Operationen auf dem Gerät aus

- HTTP-API für Aufträge-Geräte

Führt MQTT-API-Operationen auf dem Gerät aus

Sie können Gerätebefehle für Aufträge ausgeben, indem Sie MQTT-Nachrichten in den für Auftragsbefehle verwendeten reservierten Themen veröffentlichen.

Ihr geräteseitiger Client muss die Antwortnachrichten-Themen dieser Befehle abonniert haben. Wenn Sie den AWS IoT Device Client verwenden, abonniert Ihr Gerät automatisch die Antwortthemen. Das bedeutet, dass der Message Broker die Themen der Antwortnachricht auf dem Client veröffentlicht, der die Befehlsnachricht veröffentlicht hat, unabhängig davon, ob Ihr Client die Themen der Antwortnachricht abonniert hat oder nicht. Diese Antwortnachrichten werden nicht durch den Message Broker weitergeleitet und können auch nicht von anderen Clients oder Regeln abonniert werden.

Wenn Sie den Auftrag und die `jobExecution` Ereignisthemen für Ihre Flottenüberwachungslösung abonnieren, aktivieren Sie zunächst die Aufgaben- und Auftragsausführungsereignisse, um alle Ereignisse auf der Cloud-Seite zu empfangen. Auftragsfortschrittsnachrichten, die über den Message Broker verarbeitet werden und von AWS IoT -Regeln verwendet werden können, werden veröffentlicht als Auftragsereignisse. Da der Message Broker Antwortnachrichten auch ohne ausdrückliches Abonnement veröffentlicht, muss Ihr Client so konfiguriert sein, dass er die empfangenen Nachrichten empfängt und identifiziert. Ihr Client muss außerdem bestätigen, dass das Thema *`thingName`* in der eingehenden Nachricht für den Namen des Clients gilt, bevor der Client auf die Nachricht reagiert.

> **Note**
>
> Nachrichten, die als Antwort auf MQTT Jobs API-Befehlsnachrichten AWS IoT gesendet werden, werden Ihrem Konto belastet, unabhängig davon, ob Sie sie explizit abonniert haben oder nicht.

Im Folgenden werden die MQTT-API-Operationen und ihre Anfrage- und Antwortsyntax gezeigt. Alle MQTT-API-Operationen haben die folgenden Parameter:

clientToken

Ein optionaler Client-Token zur Korrelierung von Anforderungen und Antworten. Geben Sie hier einen beliebigen Wert ein, und dieser wird in der Antwort reflektiert.

`timestamp`

Die Zeit in Sekunden seit der Epoche, in der die Nachricht gesendet wurde, vergangene Zeit.

GetPendingJobExecutions

Ruft die Liste aller Aufträge für ein bestimmtes Objekt ab, die sich nicht in einem Terminal-Zustand befinden.

Veröffentlichen Sie zum Aufruf dieser API eine Nachricht auf `$aws/things/thingName/jobs/get`.

Anforderungsnutzlast:

```
{ "clientToken": "string" }
```

Der Message Broker veröffentlicht `$aws/things/thingName/jobs/get/accepted` und `$aws/things/thingName/jobs/get/rejected` auch ohne ein bestimmtes Abonnement. Damit Ihr Kunde die Nachrichten empfangen kann, muss er sie jedoch abhören. Weitere Informationen finden Sie in der Anmerkung zu Jobs API-Nachrichten.

Antwortnutzlast:

```
{
"inProgressJobs" : [ JobExecutionSummary ... ],
"queuedJobs" : [ JobExecutionSummary ... ],
"timestamp" : 1489096425069,
"clientToken" : "client-001"
}
```

Wenn `inProgressJobs` und `queuedJobs` eine Liste von JobExecutionSummary Objekten zurückgibt, die den Status `IN_PROGRESS` oder `QUEUED` haben.

StartNextPendingJobExecution

Ruft die nächste anstehende Auftragsausführung für ein Objekt ab und startet sie (Status `IN_PROGRESS` oder `QUEUED`).

- Alle Auftragsausführungen mit dem Status `IN_PROGRESS` werden zuerst zurückgegeben.
- Auftragsausführungen werden in der Reihenfolge zurückgegeben, in der sie zur Warteschlange hinzugefügt wurden. Wenn der Zielgruppe für Ihren Auftrag etwas hinzugefügt oder daraus entfernt

wird, überprüfen Sie die Rollout-Reihenfolge aller neuen Auftragsausführungen im Vergleich zu bestehenden Auftragsausführungen.

- Wenn die nächste ausstehende Auftragsausführung den Status `QUEUED` hat, wechselt ihr Status zu `IN_PROGRESS`, und die Statusdetails der Auftragsausführung werden wie angegeben eingerichtet.
- Wenn die nächste ausstehende Auftragsausführung bereits den Status `IN_PROGRESS` hat, werden ihre Statusdetails nicht geändert.
- Wenn keine Auftragsausführungen ausstehen, enthält die Antwort das Feld `execution` nicht.
- Optional können Sie einen Schritt-Timer erstellen, indem Sie einen Wert für die `stepTimeoutInMinutes`-Eigenschaft angeben. Falls Sie den Wert dieser Eigenschaft nicht aktualisieren, indem Sie `UpdateJobExecution` ausführen, läuft die Auftragsausführung ab, wenn der Schritt-Timer abläuft.

Veröffentlichen Sie zum Aufruf dieser API eine Nachricht auf `$aws/things/`*`thingName`*`/jobs/start-next`.

Anforderungsnutzlast:

```
{
"statusDetails": {
    "string": "job-execution-state"
    ...
},
"stepTimeoutInMinutes": long,
"clientToken": "string"
}
```

`statusDetails`

Eine Sammlung von Name/Wert-Paaren, die den Status der Auftragsausführung beschreiben. Wenn nicht angegeben, sind die `statusDetails` nicht geändert.

`stepTimeOutInMinutes`

Gibt die Dauer an, die dieses Gerät für den Abschluss der Ausführung dieses Auftrags hat. Wenn der Status der Auftragsausführung vor Ablauf oder Zurücksetzen des Timers (durch Aufrufen von `UpdateJobExecution`, Setzen des Status auf `IN_PROGRESS` und Angeben eines neuen Zeitüberschreitungswerts im Feld `stepTimeoutInMinutes`) auf keinen Terminal-Zustand gesetzt wird, wird der Status der Auftragsausführung auf `TIMED_OUT` gesetzt. Das Festlegen dieser Zeitüberschreitung hat keinen Einfluss auf die Zeitüberschreitung für

die Auftragsausführung, die möglicherweise beim Erstellen des Auftrags festgelegt wurde (`CreateJob` mithilfe des Feldes `timeoutConfig`).

Gültige Werte für diesen Parameter liegen im Bereich von 1 bis 10 080 (1 Minute bis 7 Tage). Ein Wert von -1 ist ebenfalls gültig und storniert den aktuellen Step-Timer (der durch eine frühere Verwendung von UpdateJobExecutionRequest erstellt wurde).

Der Message Broker veröffentlicht `$aws/things/thingName/jobs/start-next/accepted` und `$aws/things/thingName/jobs/start-next/rejected` auch ohne ein bestimmtes Abonnement. Damit Ihr Kunde die Nachrichten empfangen kann, muss er sie jedoch abhören. Weitere Informationen finden Sie in der Anmerkung zu Jobs API-Nachrichten.

Antwortnutzlast:

```
{
"execution" : JobExecutionData,
"timestamp" : timestamp,
"clientToken" : "string"
}
```

Wo `execution` ein JobExecution-Objekt ist. Beispiel:

```
{
"execution" : {
    "jobId" : "022",
    "thingName" : "MyThing",
    "jobDocument" : "< contents of job document >",
    "status" : "IN_PROGRESS",
    "queuedAt" : 1489096123309,
    "lastUpdatedAt" : 1489096123309,
    "versionNumber" : 1,
    "executionNumber" : 1234567890
},
"clientToken" : "client-1",
"timestamp" : 1489088524284,
}
```

DescribeJobExecution

Ruft detaillierte Informationen zu einer Auftragsausführung ab.

Sie können die `jobId` auf `$next` setzen, um die nächste ausstehende Auftragsausführung für ein Objekt (mit Status `IN_PROGRESS` oder `QUEUED`) zurückzugeben.

Veröffentlichen Sie zum Aufruf dieser API eine Nachricht auf `$aws/things/`*`thingName`*`/jobs/`*`jobId`*`/get`.

Anforderungsnutzlast:

```
{
"jobId" : "022",
"thingName" : "MyThing",
"executionNumber": long,
"includeJobDocument": boolean,
"clientToken": "string"
}
```

`thingName`

Der Name des dem Gerät zugeordneten Objekts.

`jobId`

Die eindeutige Kennung, die diesem Auftrag bei seiner Erstellung zugewiesen wurde.

Sie können `$next` verwenden, um die nächste ausstehende Auftragsausführung für ein Objekt (mit Status `IN_PROGRESS` oder `QUEUED`) zurückzugeben. In diesem Fall werden alle Auftragsausführungen mit dem Status `IN_PROGRESS` zuerst zurückgegeben. Auftragsausführungen werden in der Reihenfolge zurückgegeben, in der sie erstellt wurden.

`executionNumber`

(Optional) Eine Nummer, die eine Auftragsausführung auf einem Gerät identifiziert. Wenn nicht angegeben, wird die letzte Auftragsausführung zurückgegeben.

`includeJobDocument`

(Optional) Sofern nicht auf `false` gesetzt, enthält die Antwort das Auftragsdokument. Der Standardwert ist `true`.

Der Message Broker veröffentlicht `$aws/things/`*`thingName`*`/jobs/`*`jobId`*`/get/accepted` und `$aws/things/`*`thingName`*`/jobs/`*`jobId`*`/get/rejected` auch ohne ein bestimmtes Abonnement. Damit Ihr Kunde die Nachrichten empfangen kann, muss er sie jedoch abhören. Weitere Informationen finden Sie in der Anmerkung zu Jobs API-Nachrichten.

Antwortnutzlast:

```
{
"execution" : JobExecutionData,
"timestamp": "timestamp",
"clientToken": "string"
}
```

Wo `execution` ein JobExecution-Objekt ist.

UpdateJobExecution

Aktualisiert den Status einer Auftragsausführung. Sie können optional einen Schritt-Timer erstellen, indem Sie einen Wert für die `stepTimeoutInMinutes`-Eigenschaft angeben. Falls Sie den Wert dieser Eigenschaft nicht aktualisieren, indem Sie `UpdateJobExecution` erneut ausführen, läuft die Auftragsausführung ab, wenn der Schritt-Timer abläuft.

Veröffentlichen Sie zum Aufruf dieser API eine Nachricht auf `$aws/things/`*`thingName`*`/jobs/`*`jobId`*`/update`.

Anforderungsnutzlast:

```
{
"status": "job-execution-state",
"statusDetails": {
    "string": "string"
    ...
},
"expectedVersion": "number",
"executionNumber": long,
"includeJobExecutionState": boolean,
"includeJobDocument": boolean,
"stepTimeoutInMinutes": long,
"clientToken": "string"
}
```

`status`

Der neue Status für die Auftragsausführung (`IN_PROGRESS`, `FAILED`, `SUCCEEDED` oder `REJECTED`). Dieser muss bei jeder Aktualisierung angegeben werden.

`statusDetails`

Eine Sammlung von Name/Wert-Paaren, die den Status der Auftragsausführung beschreiben. Wenn nicht angegeben, sind die `statusDetails` nicht geändert.

`expectedVersion`

Die erwartete aktuelle Version der Auftragsausführung. Bei jeder Aktualisierung der Auftragsausführung wird die Version erhöht. Wenn die im Jobs-Dienst gespeicherte Version der AWS IoT Auftragsausführung nicht übereinstimmt, wird das Update mit einem `VersionMismatch` Fehler abgelehnt. Eine ErrorResponse, die den aktuellen Status der Auftragsausführung enthält, wird ebenfalls zurückgegeben. (Dadurch ist es nicht erforderlich, eine separate `DescribeJobExecution`-Anforderung durchzuführen, um die Daten zum Status der Auftragsausführung abzurufen.)

`executionNumber`

(Optional) Eine Nummer, die eine Auftragsausführung auf einem Gerät identifiziert. Wenn nicht angegeben, wird die letzte Auftragsausführung verwendet.

`includeJobExecutionState`

(Optional) Wenn enthalten und auf `true` gesetzt, enthält die Antwort das Feld `JobExecutionState`. Der Standardwert ist `false`.

`includeJobDocument`

(Optional) Wenn enthalten und auf `true` gesetzt, enthält die Antwort das `JobDocument`. Der Standardwert ist `false`.

`stepTimeoutInMinutes`

Gibt die Dauer an, die dieses Gerät für den Abschluss der Ausführung dieses Auftrags hat. Wenn der Status der Auftragsausführung nicht in einen Terminal-Zustand gesetzt wird, bevor dieser Timer abläuft oder bevor der Timer zurückgesetzt wird, wird der Status der Auftragsausführung auf `TIMED_OUT` gesetzt. Das Festlegen oder Zurücksetzen dieser Zeitüberschreitung hat keinen Einfluss auf die Zeitüberschreitung der Auftragsausführung, die möglicherweise beim Erstellen des Auftrags festgelegt wurde.

Der Message Broker veröffentlicht `$aws/things/`*`thingName`*`/jobs/`*`jobId`*`/update/accepted` und `$aws/things/`*`thingName`*`/jobs/`*`jobId`*`/update/rejected` auch ohne ein bestimmtes Abonnement. Damit Ihr Kunde die Nachrichten empfangen kann, muss er sie jedoch abhören. Weitere Informationen finden Sie in der Anmerkung zu Jobs API-Nachrichten.

Antwortnutzlast:

```
{
"executionState": JobExecutionState,
"jobDocument": "string",
"timestamp": timestamp,
"clientToken": "string"
}
```

`executionState`

Ein JobExecutionState-Objekt.

`jobDocument`

Ein -Auftragsdokument-Objekt.

> **Note**
>
> In MQTT-Antworten ist das `jobDocument` Feld ein JSON-Objekt. In HTTP-Antworten ist es eine Zeichenkettendarstellung des JSON-Objekts.

`timestamp`

Die Zeit in Sekunden seit der Epoche, in der die Nachricht gesendet wurde, vergangene Zeit.

`clientToken`

Ein Client-Token zur Korrelierung von Anforderungen und Antworten.

Wenn Sie das MQTT-Protokoll verwenden, können Sie auch die folgenden Updates durchführen:

JobExecutionsChanged

Wird gesendet, wenn eine Auftragsausführung der Liste ausstehender Auftragsausführungen für ein Objekt hinzugefügt oder daraus entfernt wird.

Verwenden Sie das -Thema:

`$aws/things/thingName/jobs/notify`

Nachrichtennutzlast:

```
{
"jobs" : {
    "JobExecutionState": [ JobExecutionSummary ... ]
         },
    "timestamp": timestamp
}
```

NextJobExecutionChanged

Wird gesendet, wenn sich ändert, welche Auftragsausführung die nächste auf der Liste der ausstehenden Auftragsausführungen für ein Objekt ist, wie für `DescribeJobExecution` mit `jobId` `$next` definiert. Diese Nachricht wird nicht gesendet, wenn sich die Details der nächsten Auftragsausführung ändern, sondern nur, wenn sich der nächste Auftrag geändert hat, der von `DescribeJobExecution` mit `jobId` `$next` ausgegeben würde. Nehmen wir als Beispiel die Auftragsausführungen J1 und J2 mit dem Status `QUEUED`. J1 ist die nächste ausstehende Auftragsausführung auf der Liste. Wenn sich der Status von J2 zu `IN_PROGRESS` ändert und der Status von J1 unverändert bleibt, wird diese Benachrichtigung gesendet und enthält Details von J2.

Verwenden Sie das -Thema:

`$aws/things/`*`thingName`*`/jobs/notify-next`

Nachrichtennutzlast:

```
{
"execution" : JobExecution,
"timestamp": timestamp,
}
```

HTTP-API für Aufträge-Geräte

Geräte können mithilfe der HTTP-Signatur Version 4 auf Port 443 mit AWS IoT Jobs kommunizieren. Dies ist die Methode, die von den AWS SDKs und der CLI verwendet wird. Weitere Informationen zu diesen Tools finden Sie unter AWS CLI Befehl Reference:iot-jobs-data oder AWS SDKs und Tools.

Für Geräte, die die Aufträge ausführen, sind die folgenden Befehle verfügbar. Hinweise zur Verwendung von API-Operationen mit dem MQTT-Protokoll finden Sie unter Führt MQTT-API-Operationen auf dem Gerät aus.

GetPendingJobExecutions

Ruft die Liste aller Aufträge für ein bestimmtes Objekt ab, die sich nicht in einem Terminal-Zustand befinden.

HTTPS request

```
GET /things/thingName/jobs
```

Antwort:

```
{
"inProgressJobs" : [ JobExecutionSummary ... ],
"queuedJobs" : [ JobExecutionSummary ... ]
}
```

Weitere Informationen finden Sie unter GetPendingJobExecutions.

CLI syntax

```
aws iot-jobs-data get-pending-job-executions \
--thing-name <value>  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"thingName": "string"
}
```

Weitere Informationen finden Sie unter get-pending-job-executions.

StartNextPendingJobExecution

Ruft die nächste anstehende Auftragsausführung für ein Objekt ab und startet sie (mit Status `IN_PROGRESS` oder `QUEUED`).

- Alle Auftragsausführungen mit dem Status `IN_PROGRESS` werden zuerst zurückgegeben.
- Auftragsausführungen werden in der Reihenfolge zurückgegeben, in der sie erstellt wurden.

- Wenn die nächste ausstehende Auftragsausführung den Status `QUEUED` hat, wechselt ihr Status zu `IN_PROGRESS`, und die Statusdetails der Auftragsausführung werden wie angegeben eingerichtet.
- Wenn die nächste ausstehende Auftragsausführung bereits den Status `IN_PROGRESS` hat, ändern sich ihre Statusdetails nicht.
- Wenn keine Auftragsausführungen ausstehen, enthält die Antwort das Feld `execution` nicht.
- Optional können Sie einen Schritt-Timer erstellen, indem Sie einen Wert für die `stepTimeoutInMinutes`-Eigenschaft angeben. Falls Sie den Wert dieser Eigenschaft nicht aktualisieren, indem Sie `UpdateJobExecution` ausführen, läuft die Auftragsausführung ab, wenn der Schritt-Timer abläuft.

HTTPS request

Im folgenden Beispiel wird die Anfragesyntax dargestellt:

```
PUT /things/thingName/jobs/$next
{
"statusDetails": {
    "string": "string"
    ...
},
"stepTimeoutInMinutes": long
}
```

Weitere Informationen finden Sie unter [StartNextPendingJobExecution](#).

CLI syntax

Syntax:

```
aws iot-jobs-data  start-next-pending-job-execution \
--thing-name <value> \
{--step-timeout-in-minutes <value>] \
[--status-details <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
```

```
"thingName": "string",
"statusDetails": {
"string": "string"
},
"stepTimeoutInMinutes": long
}
```

Weitere Informationen finden Sie unter start-next-pending-job-execution.

DescribeJobExecution

Ruft detaillierte Informationen zu einer Auftragsausführung ab.

Sie können die `jobId` auf `$next` setzen, um die nächste ausstehende Auftragsausführung für ein Objekt zurückzugeben. Der Ausführungsstatus des Auftrags muss `QUEUED` oder `IN_PROGRESS` sein.

HTTPS request

Anforderung:

```
GET /things/thingName/jobs/jobId?
executionNumber=executionNumber&includeJobDocument=includeJobDocument
```

Antwort:

```
{
"execution" : JobExecution,
}
```

Weitere Informationen finden Sie unter DescribeJobExecution.

CLI syntax

Syntax:

```
aws iot-jobs-data  describe-job-execution \
--job-id <value> \
--thing-name <value> \
[--include-job-document | --no-include-job-document] \
[--execution-number <value>]  \
```

```
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"jobId": "string",
"thingName": "string",
"includeJobDocument": boolean,
"executionNumber": long
}
```

Weitere Informationen finden Sie unter `describe-job-execution`.

UpdateJobExecution

Aktualisiert den Status einer Auftragsausführung. Optional können Sie einen Schritt-Timer erstellen, indem Sie einen Wert für die `stepTimeoutInMinutes`-Eigenschaft angeben. Falls Sie den Wert dieser Eigenschaft nicht aktualisieren, indem Sie `UpdateJobExecution` erneut ausführen, läuft die Auftragsausführung ab, wenn der Schritt-Timer abläuft.

HTTPS request

Anforderung:

```
POST /things/thingName/jobs/jobId
{
"status": "job-execution-state",
"statusDetails": {
    "string": "string"
    ...
},
"expectedVersion": "number",
"includeJobExecutionState": boolean,
"includeJobDocument": boolean,
"stepTimeoutInMinutes": long,
"executionNumber": long
}
```

Weitere Informationen finden Sie unter `UpdateJobExecution`.

CLI syntax

Syntax:

```
aws iot-jobs-data  update-job-execution \
--job-id <value> \
--thing-name <value> \
--status <value> \
[--status-details <value>] \
[--expected-version <value>] \
[--include-job-execution-state | --no-include-job-execution-state] \
[--include-job-document | --no-include-job-document] \
[--execution-number <value>]  \
[--cli-input-json <value>] \
[--step-timeout-in-minutes <value>] \
[--generate-cli-skeleton]
```

`cli-input-json` format:

```
{
"jobId": "string",
"thingName": "string",
"status": "string",
"statusDetails": {
"string": "string"
},
"stepTimeoutInMinutes": number,
"expectedVersion": long,
"includeJobExecutionState": boolean,
"includeJobDocument": boolean,
"executionNumber": long
}
```

Weitere Informationen finden Sie unter `update-job-execution`.

Schutz von Benutzern und Geräten mit AWS IoT Jobs

Um Benutzern die Verwendung von AWS IoT Jobs mit ihren Geräten zu gestatten, müssen Sie ihnen mithilfe von IAM-Richtlinien Berechtigungen gewähren. Die Geräte müssen dann mithilfe von AWS IoT Core Richtlinien autorisiert werden, um eine sichere Verbindung herzustellen AWS IoT, Auftragsausführungen zu empfangen und den Ausführungsstatus zu aktualisieren.

Erforderlicher Richtlinientyp für AWS IoT Jobs

Die folgende Tabelle zeigt die verschiedenen Richtlinientypen, die Sie für die Autorisierung verwenden müssen. Weitere Informationen über die erforderliche Richtlinie finden Sie unter Autorisierung.

Erforderlicher Richtlinientyp

Anwendungsfall	Protokoll	Authentifizierung	Kontrolliere die plane/data Ebene	Identitätstyp	Erforderlicher Richtlinientyp
Autorisieren Sie einen Administrator, Operator oder Cloud-Service, um sicher mit Aufträgen zu arbeiten	HTTPS	AWS Signaturauthentifizierung der Version 4 (Port 443)	Sowohl Steuerebene als auch Datenebene	Amazon Cognito Identity, IAM oder Verbundbenutzer	IAM-Richtlinie
Autorisieren Sie Ihr IoT-Gerät, um sicher mit Aufträgen zu arbeiten	MQTT/HTTPS	Gegenseitige TCP- oder TLS-Authentifizierung (Port 8883 oder 443)	Datenebene	X.509 Zertifikate	AWS IoT Core Richtlinie

Um AWS IoT Jobs-Operationen zu autorisieren, die sowohl auf der Steuerungsebene als auch auf der Datenebene ausgeführt werden können, müssen Sie IAM-Richtlinien verwenden. Die Identitäten müssen für AWS IoT authentifiziert sein, um diese Operationen ausführen zu können. Dabei muss es sich um Amazon-Cognito-Identitäten oder IAM-Benutzer, -Gruppen und -Rollen handeln. Weitere Informationen über die Authentifizierung finden Sie unter Authentifizierung.

Die Geräte müssen jetzt auf der Datenebene mithilfe von AWS IoT Core Richtlinien autorisiert werden, um eine sichere Verbindung zum Geräte-Gateway herzustellen. Das Geräte-Gateway ermöglicht es Geräten, sicher mit ihnen zu kommunizieren AWS IoT, Auftragsausführungen zu

empfangen und den Status der Auftragsausführung zu aktualisieren. Die Gerätekommunikation wird mithilfe von sicheren MQTT- oder HTTPS veröffentlichen-Kommunikationsprotokollen gesichert. Diese Protokolle verwenden X.509 Client-Zertifikate die von zur Verfügung gestellten Protokolle, AWS IoT um die Geräteverbindungen zu authentifizieren.

Im Folgenden wird gezeigt, wie Sie Ihre Benutzer, Cloud-Dienste und Geräte zur Nutzung AWS IoT von Jobs autorisieren. Weitere Informationen zu API-Operationen der Steuerebene und Datenebene finden Sie unter AWS IoT Jobs API-Operationen.

Themen

- Autorisieren Sie Benutzer und Cloud-Dienste zur Nutzung AWS IoT Jobs
- Autorisieren Sie Ihre Geräte zur sicheren Verwendung AWS IoT Jobs auf der Datenebene

Autorisieren Sie Benutzer und Cloud-Dienste zur Nutzung AWS IoT Jobs

Um Ihre Benutzer und Cloud-Services zu autorisieren, müssen Sie IAM-Richtlinien sowohl auf der Steuerebene als auch auf der Datenebene verwenden. Die Richtlinien müssen mit dem HTTPS-Protokoll verwendet werden und müssen die AWS Signature Version 4-Authentifizierung (Port 443) verwenden, um Benutzer zu authentifizieren.

> **Note**
>
> AWS IoT Core Richtlinien dürfen nicht auf der Steuerungsebene verwendet werden. Für die Autorisierung von Benutzern oder Cloud-Services werden nur IAM-Richtlinien verwendet. Weitere Informationen über die Verwendung der erforderlichen Richtlinie finden Sie unter Erforderlicher Richtlinientyp für AWS IoT Jobs.

IAM-Richtlinien sind JSON-Dokumente, die Richtlinienanweisungen enthalten. Richtlinienanweisungen verwenden die Elemente Effekt, Aktion und Ressource, um Ressourcen festzulegen, Aktionen zuzulassen oder abzulehnen und Bedingungen, bei denen Aktionen zugelassen oder abgelehnt werden. Weitere Informationen finden Sie in der Referenz zu IAM-JSON-Richtlinienelementen im IAM-Benutzerhandbuch.

> **Warning**
>
> Es wird empfohlen, keine Platzhalterberechtigungen zu verwenden, wie z. B. `"Action": ["iot:*"]` in Ihren IAM-Richtlinien oder AWS IoT Core -Richtlinien. Die Verwendung von

Platzhalterberechtigungen ist keine empfohlene, bewährte Sicherheitsmethode. Weitere Informationen finden Sie unter AWS IoT Richtlinie zu freizügig.

IAM-Richtlinien auf der Steuerebene

Auf der Steuerebene verwenden IAM-Richtlinien das Präfix `iot:` mit der Aktion, um den entsprechenden API-Vorgang für Aufträge zu autorisieren. Beispielsweise gewährt die Richtlinienaktion `iot:CreateJob` dem Benutzer die Erlaubnis, die `CreateJob`-API zu verwenden.

Richtlinienaktionen

In der folgenden Tabelle wird eine Liste der IAM-Richtlinienaktionen und Berechtigungen für die Verwendung der API-Aktionen dargestellt. Informationen zu Ressourcentypen finden Sie unter Ressourcentypen, definiert von. AWS IoT Weitere Informationen zu AWS IoT Aktionen finden Sie unter Aktionen definiert von AWS IoT.

IAM-Richtlinienaktionen auf Steuerebene

Richtlinienaktionen	API-Operation	Ressourcentypen	Description
`iot:AssociateTargetsWithJob`	`AssociateTargetsWithJob`	• Auftrag • Objekt • thinggroup	Steht für die Erlaubnis zum Verknüpfen einer Gruppe mit einem kontinuierlichen Auftrag. Die Berechtigung `iot:AssociateTargetsWithJob` wird jedes Mal überprüft, wenn eine Anforderung zur Verknüpfung eines Auftrags gestellt wird.
`iot:CancelJob`	`CancelJob`	Auftrag	Stellt die Berechtigung zum Abbrechen eines Auftrags dar. Die Berechtigung `iot:CancelJob` wird jedes Mal überprüft, wenn eine Anforderung zum Abbrechen eines Auftrags gestellt wird.
`iot:CancelJobExecution`	`CancelJobExecution`	• Auftrag • Objekt	Stellt die Berechtigung zum Abbruch einer Auftragsausführung dar. Die Berechtigung `iot: CancelJobExecution` wird jedes Mal überprüft, wenn eine Anforderu

Richtlinienaktionen	API-Operation	Ressourcentypen	Description
			ng zum Abbrechen einer Auftragsausführung gestellt wird.
iot:CreateJob	CreateJob	• Auftrag • Objekt • thinggroup • jobtemplate • package	Stellt die Berechtigung zum Erstellen eines Auftrags dar. Die Berechtigung iot:CreateJob wird jedes Mal überprüft, wenn eine Anforderung zum Erstellen eines Auftrags gestellt wird.
iot:CreateJobTemplate	CreateJobTemplate	• Auftrag • jobtemplate • package	Stellt die Berechtigung zum Erstellen einer Auftragsvorlage dar. Die Berechtigung iot: CreateJobTemplate wird jedes Mal überprüft, wenn eine Anforderung zum Erstellen einer Auftragsvorlage gestellt wird.
iot:DeleteJob	DeleteJob	Auftrag	Stellt die Berechtigung zum Löschen eines Auftrags dar. Die Berechtigung iot:DeleteJob wird jedes Mal überprüft, wenn eine Anforderung zum Löschen eines Auftrags gestellt wird.
iot:DeleteJobTemplate	DeleteJobTemplate	jobtemplate	Stellt die Berechtigung zum Löschen einer Auftragsvorlage dar. Die Berechtigung iot: CreateJobTemplate wird jedes Mal überprüft, wenn eine Anforderung zum Löschen einer Auftragsvorlage gestellt wird.

Richtlinienaktionen	API-Operation	Ressourcentypen	Description
`iot:DeleteJobExecution`	DeleteJobTemplate	• Auftrag • Objekt	Stellt die Berechtigung zum Löschen einer Auftragsausführung dar. Die Berechtigung `iot: DeleteJobExecution` wird jedes Mal überprüft, wenn eine Anforderung zum Löschen eines Auftragsausführung gestellt wird.
`iot:DescribeJob`	DescribeJob	Auftrag	Stellt die Berechtigung zum Beschreiben eines Auftrags dar. Die Berechtigung `iot: DescribeJob` wird jedes Mal überprüft, wenn eine Anforderung zum Beschreiben eines Auftrags gestellt wird.
`iot:DescribeJobExecution`	DescribeJobExecution	• Auftrag • Objekt	Stellt die Berechtigung zum Beschreiben einer Auftragsausführung dar. Die Berechtigung `iot: DescribeJobExecution` wird jedes Mal überprüft, wenn eine Anforderung zum Beschreiben einer Auftragsausführung gestellt wird.
`iot:DescribeJobTemplate`	DescribeJobTemplate	jobtemplate	Stellt die Berechtigung zum Beschreiben einer Auftragsvorlage dar. Die Berechtigung `iot: DescribeJobTemplate` wird jedes Mal überprüft, wenn eine Anforderung zum Beschreiben einer Auftragsvorlage gestellt wird.
`iot:DescribeManagedJobTemplate`	DescribeManagedJobTemplate	jobtemplate	Stellt die Berechtigung zum Beschreiben einer verwalteten Auftragsvorlage dar. Die Berechtigung `iot: DescribeManagedJobTemplate` wird jedes Mal überprüft, wenn eine Anfrage zur Beschreibung einer verwalteten Auftragsvorlage gestellt wird.

Richtlinienaktionen	API-Operation	Ressourcentypen	Description
iot:GetJobDocument	GetJobDocument	Auftrag	Stellt die Berechtigung zum Abrufen eines Auftragsdokuments für einen Auftrag dar. Die Berechtigung iot:GetJobDocument wird jedes Mal überprüft, wenn eine Anforderung zum Abrufen eines Auftragsdokuments gestellt wird.
iot:ListJobExecutionsForJob	ListJobExecutionsForJob	Auftrag	Stellt die Berechtigung zum Auflisten von Auftragsausführungen für einen Auftrag dar. Die Berechtigung iot:ListJobExecutionsForJob wird jedes Mal überprüft, wenn eine Anforderung zum Auflisten der Auftragsausführungen für einen Auftrag gestellt wird.
iot:ListJobExecutionsForThing	ListJobExecutionsForThing	Objekt	Stellt die Berechtigung zum Auflisten von Auftragsausführungen für einen Auftrag dar. Die Berechtigung iot:ListJobExecutionsForThing wird jedes Mal überprüft, wenn eine Anforderung zum Auflisten der Auftragsausführungen für ein Objekt gestellt wird.
iot:ListJobs	ListJobs	Keine	Stellt die Berechtigung zum Auflisten der Aufträge dar. Die Berechtigung iot:ListJobs wird jedes Mal überprüft, wenn eine Anforderung zum Auflisten der Aufträge gestellt wird.
iot:ListJobTemplates	ListJobTemplates	Keine	Stellt die Berechtigung zum Auflisten der Auftragsvorlagen dar. Die Berechtigung iot:ListJobTemplates wird jedes Mal überprüft, wenn eine Anforderung zum Auflisten der Auftragsvorlagen gestellt wird.

Richtlinienaktionen	API-Operation	Ressourcentypen	Description
`iot:ListManagedJobTemplates`	ListManagedJobTemplates	Keine	Stellt die Berechtigung zum Auflisten der verwalteten Jobvorlagen dar. Die Berechtigung `iot:ListManagedJobTemplates` wird jedes Mal überprüft, wenn eine Anforderung zum Auflisten der verwalteten Auftragsvorlagen gestellt wird.
`iot:UpdateJob`	UpdateJob	Auftrag	Stellt die Berechtigung zum Aktualisieren eines Auftrags dar. Die Berechtigung `iot:UpdateJob` wird jedes Mal überprüft, wenn eine Anforderung zum Aktualisieren des Auftrags gestellt wird.
`iot:TagResource`	TagResource	• Auftrag • jobtemplate • Objekt	Gewährt die Berechtigung zum Markieren einer bestimmten Ressource.
`iot:UntagResource`	UntagResource	• Auftrag • jobtemplate • Objekt	Gewährt die Berechtigung zum Aufheben der Markierung einer bestimmten Ressource.

Grundlegendes Beispiel für IAM-Richtlinien

Das folgende Beispiel zeigt eine IAM-Richtlinie, die dem Benutzer die Berechtigung erteilt, die folgenden Aktionen für Ihr IoT-Objekt und Ihre Objektgruppe auszuführen.

Ersetzen Sie im Beispiel:

- *`region`*mit Ihrem AWS-Region, wie z. `us-east-1` B.
- *`account-id`*mit Ihrer AWS-Konto Nummer, wie z. `57EXAMPLE833` B.
- *`thing-group-name`*mit dem Namen Ihrer IoT-Dinggruppe, für die Sie Jobs ins Visier nehmen, wie zum Beispiel`FirmwareUpdateGroup`.

- *thing-name*mit dem Namen Ihrer IoT-Sache, für die Sie Jobs ins Visier nehmen, wie zum BeispielMyIoTThing.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "iot:CreateJobTemplate",
                "iot:CreateJob"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:us-east-1:123456789012:thinggroup/thing-group-
name"
        },
        {
            "Action": [
                "iot:DescribeJob",
                "iot:CancelJob",
                "iot:DeleteJob"
            ],
            "Effect": "Allow",
     "Resource": "arn:aws:iot:us-east-1:123456789012:job/*"
        },
        {
            "Action": [
                "iot:DescribeJobExecution",
                "iot:CancelJobExecution",
                "iot:DeleteJobExecution"
            ],
            "Effect": "Allow",
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:thing/thing-123",
                "arn:aws:iot:us-east-1:123456789012:job/*"
            ]
        }
    ]
}
```

Beispiel für eine IAM-Richtlinie für eine IP-basierte Autorisierung

Sie können Prinzipale daran hindern, von bestimmten IP-Adressen aus API-Aufrufe an den Endpunkt Ihrer Steuerebene zu tätigen. Um die IP-Adressen anzugeben, die zugelassen werden können, verwenden Sie im Condition-Element Ihrer IAM-Richtlinie den globalen Bedingungsschlüssel `aws:SourceIp`.

Die Verwendung dieses Bedingungsschlüssels kann auch anderen Benutzern AWS-Service den Zugriff verwehren, diese API-Aufrufe in Ihrem Namen zu tätigen, z. AWS CloudFormation B. Um den Zugriff auf diese Dienste zu ermöglichen, verwenden Sie den `aws:ViaAWSService` globalen Bedingungsschlüssel zusammen mit dem SourceIp Schlüssel aws:. Dadurch wird sichergestellt, dass die Zugriffsbeschränkung für die Quell-IP-Adresse nur für Anforderungen gilt, die direkt von einem Prinzipal stammen. Weitere Informationen finden Sie unter AWS: Verweigert den Zugriff auf AWS basierend auf der Quell-IP.

Das folgende Beispiel zeigt, wie nur eine bestimmte IP-Adresse zugelassen wird, die API-Aufrufe an den Endpunkt der Steuerebene tätigen kann. Der Schlüssel `aws:ViaAWSService` ist auf `true` eingestellt, sodass andere Services API-Aufrufe in Ihrem Namen tätigen können.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:CreateJobTemplate",
                "iot:CreateJob"
            ],
            "Resource": [
                "*"
            ],
            "Condition": {
                 "IpAddress": {
                     "aws:SourceIp": "123.45.167.89"
                 },
                 "Bool": {
                     "aws:ViaAWSService": "false"
        }
        }
    }]
```

```
}
```

IAM-Richtlinien auf der Datenebene

IAM-Richtlinien auf der Datenebene verwenden das Präfix `iotjobsdata:` zum Autorisieren von API-Vorgängen, die Benutzer ausführen können. Auf der Datenebene können Sie einem Benutzer die Berechtigung zur Verwendung der `DescribeJobExecution`-API mithilfe der Richtlinienaktion `iotjobsdata:DescribeJobExecution` erteilen.

> **Warning**
>
> Die Verwendung von IAM-Richtlinien auf der Datenebene wird nicht empfohlen, wenn Sie AWS IoT -Jobs für Ihre Geräte vergeben möchten. Wir empfehlen, dass Sie IAM-Richtlinien auf der Steuerebene verwenden, damit Benutzer Aufträge erstellen und verwalten können. Verwenden Sie auf der Datenebene AWS IoT Core Richtlinien für das HTTPS-Protokoll, um Geräte zum Abrufen von Auftragsausführungen und zum Aktualisieren des Ausführungsstatus zu autorisieren.

Grundlegendes Beispiel für IAM-Richtlinien

Die API-Operationen, die autorisiert werden müssen, werden normalerweise ausgeführt, indem Sie CLI-Befehle eingeben. Nachfolgend wird ein Beispiel für einen Benutzer gezeigt, der eine Operation `DescribeJobExecution` ausführt.

Ersetzen Sie im Beispiel:

- *`region`*mit Ihrem AWS-Region, z. B. `us-east-1`
- *`account-id`*mit Ihrer AWS-Konto Nummer, wie z. `57EXAMPLE833` B.
- *`thing-name`*mit dem Namen Ihrer IoT-Sache, für die Sie Jobs ins Visier nehmen, wie zum Beispiel`myRegisteredThing`.
- *`job-id`* ist die eindeutige Kennung für den zu vergebenden Auftrag, für den die API verwendet wird.

```
aws iot-jobs-data describe-job-execution \
    --endpoint-url "https://account-id.jobs.iot.region.amazonaws.com" \
    --job-id jobID --thing-name thing-name
```

Nachfolgend wird das Beispiel einer IAM-Richtlinie angezeigt, mit der diese Aktion autorisiert wird:

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": [
            "iotjobsdata:DescribeJobExecution"
        ],
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:thing/thing-123"
    }
}
```

Beispiele für IAM-Richtlinien mit IP-basierter Autorisierung

Sie können den Zugriff der Prinzipale einschränken, von bestimmten IP-Adressen aus API-Aufrufe an den Endpunkt Ihrer Datenebene zu vergeben. Um die IP-Adressen anzugeben, die zugelassen werden können, verwenden Sie im Condition-Element Ihrer IAM-Richtlinie den globalen Bedingungsschlüssel `aws:SourceIp`.

Die Verwendung dieses Bedingungsschlüssels kann auch anderen Benutzern AWS-Service den Zugriff verwehren, diese API-Aufrufe in Ihrem Namen zu tätigen, z. AWS CloudFormation B. Um den Zugriff auf diese Services zu ermöglichen, verwenden Sie den globalen Bedingungsschlüssel `aws:ViaAWSService` zusammen mit dem Bedingungsschlüssel `aws:SourceIp`. Dadurch wird sichergestellt, dass die Zugriffsbeschränkung für IP-Adressen nur für Anfragen gilt, die direkt vom Prinzipal gestellt werden. Weitere Informationen finden Sie unter AWS: Verweigert den Zugriff auf AWS basierend auf der Quell-IP.

Das folgende Beispiel zeigt, wie nur eine bestimmte IP-Adresse zugelassen wird, die API-Aufrufe an den Endpunkt der Datenebene vergeben kann.

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

```
        ],
        "Resource": [
            "*"
        ],
        "Condition": {
            "IpAddress": {
                "aws:SourceIp": "123.45.167.89"
        },
            "Bool": {
                "aws:ViaAWSService": "false"
        }
        }
    }]
}
```

Das folgende Beispiel zeigt, wie Sie für bestimmte IP-Adressen oder Adressbereiche das Vergeben von API-Aufrufen an den Endpunkt der Datenebene einschränken können.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "iotjobsdata:*"
            ],
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": [
                        "123.45.167.89",
                        "192.0.2.0/24",
                        "203.0.113.0/24"
                    ]
            }
        },
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Beispiel einer IAM-Richtlinie für Steuerebene und Datenebene

Wenn Sie eine API-Operation sowohl auf der Steuerebene als auch auf der Datenebene ausführen, muss Ihre Richtlinienaktion auf der Steuerebene das Präfix `iot:` verwenden, und Ihre Richtlinienaktion auf der Datenebene muss das Präfix `iotjobsdata:` verwenden.

Beispielsweise kann die API `DescribeJobExecution` sowohl auf der Steuerebene als auch auf der Datenebene verwendet werden. Auf der Steuerungsebene wird die DescribeJobExecution API verwendet, um die Ausführung eines Jobs zu beschreiben. Auf der Datenebene wird die DescribeJobExecution API verwendet, um Details einer Auftragsausführung abzurufen.

Die folgende IAM-Richtlinie autorisiert einen Benutzer, die API `DescribeJobExecution` sowohl auf der Steuerebene als auch auf der Datenebene zu verwenden.

Ersetzen Sie im Beispiel:

- *`region`*mit Ihrem AWS-Region, wie z. `us-east-1` B.
- *`account-id`*mit Ihrer AWS-Konto Nummer, wie z. `57EXAMPLE833` B.
- *`thing-name`*mit dem Namen Ihrer IoT-Sache, für die Sie Jobs ins Visier nehmen, wie zum Beispiel`MyIoTThing`.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "iotjobsdata:DescribeJobExecution"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:us-east-1:123456789012:thing/thing-123"
        },
        {
            "Action": [
                "iot:DescribeJobExecution",
                "iot:CancelJobExecution",
                "iot:DeleteJobExecution"
            ],
            "Effect": "Allow",
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:thing/thing-123",
```

```
            "arn:aws:iot:us-east-1:123456789012:job/*"
                ]
            }
        ]
}
```

Das Markieren von IoT-Ressourcen autorisieren

Um eine bessere Kontrolle über Aufträge und Auftragsvorlagen zu erhalten, die Sie erstellen, ändern oder verwenden können, haben Sie die Möglichkeit, Markierungen an die Aufträge oder Auftragsvorlagen anzuhängen. Markierungen helfen Ihnen außerdem bei der Unterscheidung von Eigentumsrechten, der Zuweisung und Zuordnung von Kosten, indem Sie diese Abrechnungsgruppen zuordnen und mit Markierungen versehen.

Wenn ein Benutzer seine Jobs oder Jobvorlagen, die er mit AWS-Managementkonsole oder erstellt hat, taggen möchte AWS CLI, muss Ihre IAM-Richtlinie dem Benutzer Berechtigungen zum Taggen gewähren. Zum Erteilen von Berechtigungen muss Ihre IAM-Richtlinie die Aktion `iot:TagResource` verwenden.

> **Note**
>
> Wenn Ihre IAM-Richtlinie die Aktion `iot:TagResource` nicht beinhaltet, wird jede Aktion `CreateJob` oder `CreateJobTemplate` mit einer Markierung einen Fehler `AccessDeniedException` zurückgeben.

Wenn Sie Ihre Jobs oder Jobvorlagen, die Sie mithilfe von oder erstellt haben, taggen AWS-Managementkonsole möchten AWS CLI, muss Ihre IAM-Richtlinie die Erlaubnis zum Taggen gewähren. Zum Erteilen von Berechtigungen muss Ihre IAM-Richtlinie die Aktion `iot:TagResource` verwenden.

Allgemeine Informationen zum Tagging von Ressourcen finden Sie unter Taggen Sie Ihre AWS IoT Ressourcen.

Beispiel für IAM-Richtlinien

Sehen Sie sich die folgenden Beispiele für IAM-Richtlinien zur Gewährung von Berechtigungen zum Markieren an:

Beispiel 1

Ein Benutzer, der den folgenden Befehl ausführt, um einen Auftrag zu erstellen und ihn einer bestimmten Umgebung zuzuordnen.

Ersetzen Sie in diesem Beispiel:

- *region*mit Ihrem AWS-Region, z. B. `us-east-1`
- *account-id*mit Ihrer AWS-Konto Nummer, wie z. `57EXAMPLE833` B.
- *thing-name*mit dem Namen Ihrer IoT-Sache, für die Sie Jobs ins Visier nehmen, wie zum Beispiel`MyIoTThing`.

```
aws iot create-job
    --job-id test_job
    --targets "arn:aws:iot:region:account-id:thing/thingOne"
    --document-source "https://s3.amazonaws.com/amzn-s3-demo-bucket/job-document.json"
    --description "test job description"
    --tags Key=environment,Value=beta
```

In diesem Beispiel müssen Sie die folgende IAM-Richtlinie verwenden:

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": [
            "iot:CreateJob",
            "iot:CreateJobTemplate",
            "iot:TagResource"
        ],
        "Effect": "Allow",
        "Resource": [
           "arn:aws:iot:us-east-1:123456789012:job/*",
           "arn:aws:iot:us-east-1:123456789012:jobtemplate/*"
        ]
    }
}
```

Autorisieren Sie Ihre Geräte zur sicheren Verwendung AWS IoT Jobs auf der Datenebene

Um Ihre Geräte für die sichere Interaktion mit AWS IoT Jobs auf der Datenebene zu autorisieren, müssen Sie AWS IoT Core Richtlinien verwenden. AWS IoT Core Richtlinien für Jobs sind JSON-Dokumente, die Richtlinienerklärungen enthalten. Diese Richtlinien verwenden auch die Elemente Effekt, Aktion und Ressource und folgen einer ähnlichen Konvention wie IAM-Richtlinien. Weitere Informationen finden Sie in der Referenz für IAM-JSON-Richtlinienelemente im IAM-Benutzerhandbuch.

Die Richtlinien können sowohl mit MQTT- als auch mit HTTPS-Protokollen verwendet werden und müssen zur Autorisierung der Geräte die gegenseitige Authentifizierung über TCP oder TLS verwenden. Im Folgenden wird gezeigt, wie diese Richtlinien in den verschiedenen Kommunikationsprotokollen verwendet werden.

> **Warning**
>
> Es wird empfohlen, keine Platzhalterberechtigungen zu verwenden, wie dies beispielsweise `"Action": ["iot:*"]` in Ihren IAM-Richtlinien oder AWS IoT Core Richtlinien der Fall ist. Die Verwendung von Platzhalterberechtigungen ist keine empfohlene, bewährte Sicherheitsmethode. Weitere Informationen finden Sie unter AWS IoT Richtlinie zu freizügig.

AWS IoT Core Richtlinien für das MQTT-Protokoll

AWS IoT Core Richtlinien für das MQTT-Protokoll gewähren Ihnen Berechtigungen zur Verwendung der MQTT-API-Aktionen des Job-Geräts. Die MQTT-API-Operationen werden verwendet, um mit MQTT-Themen zu arbeiten, die für Auftragsbefehle reserviert sind. Weitere Informationen zu diesen API-Operationen finden Sie unter Führt MQTT-API-Operationen auf dem Gerät aus.

MQTT-Richtlinien verwenden Richtlinienaktionen wie `iot:Connect`, `iot:Publish`, `iot:Subscribe` und `iot:Receieve`, um mit den Auftragsthemen zu arbeiten. Diese Richtlinien ermöglichen es Ihnen, eine Verbindung zum Message Broker herzustellen, die MQTT-Themen für Aufträge zu abonnieren und MQTT-Nachrichten zwischen Ihren Geräten und der Cloud zu senden und zu empfangen. Weitere Informationen zu diesen Aktionen finden Sie unter AWS IoT Core politische Maßnahmen.

Informationen zu Themen für AWS IoT Jobs finden Sie unter. Auftragsthemen

Beispiel für grundlegende MQTT-Richtlinien

Das folgende Beispiel zeigt, wie Sie `iot:Publish` und `iot:Subscribe` zum Veröffentlichen und Abonnieren von Aufträgen und Auftragsausführungen verwenden können.

Ersetzen Sie im Beispiel:

- *`region`*mit Ihrem AWS-Region, z. `us-east-1` B.
- *`account-id`*mit Ihrer AWS-Konto Nummer, wie z. `57EXAMPLE833` B.
- *`thing-name`*mit dem Namen Ihrer IoT-Sache, für die Sie Jobs ins Visier nehmen, wie zum Beispiel`MyIoTThing`.

```
{
    "Version":"2012-10-17",

    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Subscribe"
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topic/$aws/events/job/*",
    "arn:aws:iot:us-east-1:123456789012:topic/$aws/events/jobExecution/*",
    "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/thing-123/jobs/*"
            ]
        }
    ]
}
```

AWS IoT Core Richtlinien für das HTTPS-Protokoll

AWS IoT Core Richtlinien auf der Datenebene können auch das HTTPS-Protokoll mit dem TLS-Authentifizierungsmechanismus verwenden, um Ihre Geräte zu autorisieren. Auf der Datenebene verwenden Richtlinien das Präfix `iotjobsdata:`, um API-Aufträge zu autorisieren, die Ihre Geräte ausführen können. Beispielsweise gewährt die Richtlinienaktion `iotjobsdata:DescribeJobExecution` dem Benutzer die Erlaubnis, die `DescribeJobExecution`-API zu verwenden.

> **Note**
>
> Die Richtlinienaktionen auf der Datenebene müssen das Präfix `iotjobsdata:` verwenden. Auf der Steuerebene müssen die Aktionen das Präfix `iot:` verwenden. Ein Beispiel für eine IAM-Richtlinie, bei der sowohl Richtlinienaktionen auf der Steuerebene als auch auf der Datenebene verwendet werden, finden Sie unter Beispiel einer IAM-Richtlinie für Steuerebene und Datenebene.

Richtlinienaktionen

Die folgende Tabelle enthält eine Liste der AWS IoT Core Richtlinienaktionen und Berechtigungen für die Autorisierung von Geräten zur Verwendung der API-Aktionen. Eine Liste der API-Operationen, die Sie auf der Datenebene ausführen können, finden Sie unter HTTP-API für Aufträge-Geräte.

> **Note**
>
> Der Richtlinienaktionen für die Auftragsausführung gilt nur für den HTTP-TLS-Endpunkt. Wenn Sie den MQTT-Endpunkt verwenden, müssen Sie die in diesem Thema definierten MQTT-Richtlinienaktionen verwenden.

AWS IoT Core politische Maßnahmen auf Datenebene

Richtlinienaktionen	API-Operation	Ressou ntypen	Description
`iotjobsda ta:Descri beJobExecution`	`DescribeJ obExecution`	• Auftra • Objek	Stellt die Berechtigung für den Abruf einer Auftragsausführung dar. Die Berechtigung `iotjobsda ta:DescribeJobExecution` wird jedes Mal überprüft, wenn eine Anforderung zum Abruf einer Auftragsausführung gestellt wird.
`iotjobsda ta:GetPen`	`GetPendin gJobExecutions`	Objekt	Stellt die Berechtigung zum Abrufen der Liste der Aufträge dar, die sich nicht in einem Endstatus für ein Objekt befinden. Die Berechtig

Richtlinienaktionen	API-Operation	Ressourcentypen	Description
dingJobExecutions			ung iotjobsdata:GetPendingJobExecutions wird jedes Mal überprüft, wenn eine Anforderung zum Abrufen der Liste gestellt wird.
iotjobsdata:StartNextPendingJobExecution	StartNextPendingJobExecution	Objekt	Stellt die Berechtigung zum Abrufen und Starten der nächsten ausstehenden Auftragsausführung für ein Objekt dar. Das heißt, um eine Auftragsausführung mit dem Status QUEUED auf IN_PROGRESS zu aktualisieren. Die Berechtigung iotjobsdata:StartNextPendingJobExecution wird jedes Mal überprüft, wenn eine Anforderung zum Starten der nächsten ausstehenden Auftragsausführung gestellt wird.
iotjobsdata:UpdateJobExecution	UpdateJobExecution	Objekt	Stellt die Berechtigung zum Aktualisieren einer Auftragsausführung dar. Die Berechtigung iotjobsdata:UpdateJobExecution wird jedes Mal überprüft, wenn eine Anforderung zum Aktualisieren des Status einer Auftragsausführung gestellt wird.

Grundlegende Beispiele für Richtlinien

Das Folgende zeigt ein Beispiel AWS IoT Core für eine Richtlinie, die die Erlaubnis erteilt, die Aktionen auf der Datenebene API-Operationen für jede Ressource auszuführen. Sie können Ihre Richtlinie auf eine bestimmte Ressource beschränken, wie z. B. auf ein IoT-Objekt. Ersetzen Sie in Ihrem Beispiel:

- *region*mit deinem AWS-Region wieus-east-1.
- *account-id*mit deiner AWS-Konto Nummer, wie zum Beispiel57EXAMPLE833.
- *thing-name*mit dem Namen der IoT-Sache, wie zum BeispielMyIoTthing.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "iotjobsdata:GetPendingJobExecutions",
                "iotjobsdata:StartNextPendingJobExecution",
                "iotjobsdata:DescribeJobExecution",
                "iotjobsdata:UpdateJobExecution"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:us-east-1:123456789012:thing/thing-123"
        }
    ]
}
```

Ein Beispiel dafür, wann Sie diese Richtlinien verwenden müssen, kann sein, wenn Ihre IoT-Geräte eine AWS IoT Core -Richtlinie für den Zugriff auf eine dieser API-Operationen verwenden, wie z. B. das folgende Beispiel für die API `DescribeJobExecution`:

```
GET /things/thingName/jobs/jobId?
executionNumber=executionNumber&includeJobDocument=includeJobDocument&namespaceId=namespaceId
 HTTP/1.1
```

AWS IoT Grenzwerte für Aufträge

AWS IoT Für Jobs gelten Dienstkontingente oder Grenzwerte, die der maximalen Anzahl von Serviceressourcen oder Vorgängen für Sie entsprechen AWS-Konto.

Themen

- Grenzwerte für die Ausführung von Aufträgen
- Limits für aktive und gleichzeitige Aufträge

Grenzwerte für die Ausführung von Aufträgen

Dieser Abschnitt enthält Informationen zu den Grenzwerten für die Auftragsausführung für AWS IoT Device Management.

> **Note**
>
> Diese Grenzwerte sind nicht Teil der Servicekontingente, die Sie in der Dokumentation zu den AWS IoT Device Management Servicekontingenten finden.

Um Informationen über die Anzahl der ausstehenden Jobausführungen zu erhalten, können Sie entweder die `GetPendingJobExecutions` API verwenden oder die reservierten MQTT-Themen für AWS IoT Jobs and Receive abonnieren. Auftrag-Benachrichtigungstypen

Die Anzahl der ausstehenden Auftragsausführungen in Ihrem Konto kann variieren, je nachdem, ob Sie die Planungskonfiguration aktiviert haben und ein wiederkehrendes Wartungsfenster verwenden.

Maximale Anzahl ausstehender Auftragsausführungen

API/notification Name	Description	Ohne Planungskonfiguration	Mit Planungskonfiguration
`ListNotification`	A `ListNotification` wird immer dann veröffentlicht, wenn eine alte Auftragsausführung in den Terminalstatus übergeht oder wenn eine neue Auftragsausführung in die Warteschlange gestellt wird oder in einen Nicht-Terminal-Status wechselt. Es kann bis zu 15 ausstehende Auftragsausführungen anzeigen, die entweder oder sind. `QUEUED` `IN_PROGRESS`	10	15 (Bis zu 5 Auftragsausführungen werden nur im Fenster `ListNotification` während einer Wartung angezeigt).
`GetPendingJobExecutions`	Wenn Sie die `GetPendingJobExecutions` API aufrufen, gibt sie eine Liste der Auftragsausführungen zurück, die noch nicht gestartet wurden und	10	15

API/notif ication Name	Description	Ohne Planungsk onfiguration	Mit Planungsk onfiguration
	nach dem API-Aufruf gestartet werden können. Die API kann bis zu 10 ausstehende Auftragsausführungen zurückgeben. • Von den 10 ausstehenden Auftragsa usführungen `IN_PROGRESS` werden die Ausführungen, die noch ausstehen , aus dem Ergebnis herausgefiltert. • Wenn sich die Jobs der 10 ausstehen den Auftragsausführungen im `SCHEDULED` Status befinden, werden sie aus dem Ergebnis herausgefiltert.		

Limits für aktive und gleichzeitige Aufträge

In diesem Abschnitt erfahren Sie mehr über aktive und gleichzeitige Aufträge und die für sie geltenden Beschränkungen.

Aktive Aufträge und Limit für aktive Aufträge

Wenn Sie einen Job mithilfe der AWS IoT Konsole oder der `CreateJob` API erstellen, ändert sich der Auftragsstatus in`IN_PROGRESS`. Alle laufenden Aufträge sind aktive Aufträge und werden auf das Limit für aktive Aufträge angerechnet. Dazu gehören Aufträge, bei denen entweder neue Auftragsausführungen eingeführt werden, oder Aufträge, die darauf warten, dass Geräte ihre Auftragsausführungen abschließen. Dieses Limit gilt sowohl für kontinuierliche als auch für Snapshot-Aufträge.

Gleichzeitige Aufträge und Limit für die gleichzeitige Ausführung von Aufträgen

In-progress Jobs, die entweder neue Auftragsausführungen bereitstellen oder Jobs, die zuvor erstellte Auftragsausführungen abbrechen, sind gleichzeitige Jobs und werden auf das Limit für die gleichzeitige Ausführung von Jobs angerechnet. AWS IoT Jobs können schnell mit einer Geschwindigkeit von 1000 Geräten pro Minute eingeführt und storniert werden. Jeder Auftrag ist

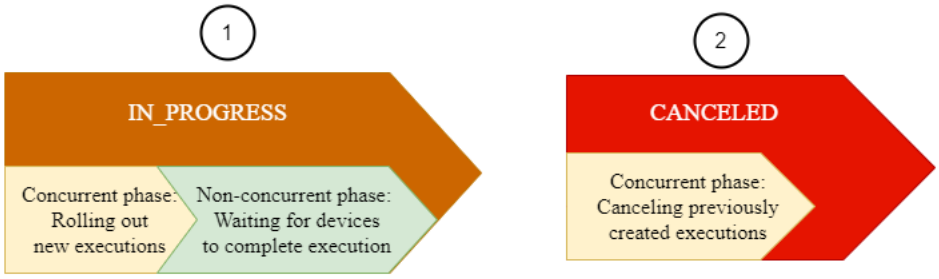
`concurrent` und wird nur für kurze Zeit auf das Limit für die gleichzeitige Ausführung von Aufträgen angerechnet. Nachdem die Auftragsausführungen eingeführt oder abgebrochen wurden, ist der Auftrag nicht mehr gleichzeitig und wird nicht auf das Limit für die gleichzeitige Ausführung von Aufträgen angerechnet. Sie können die Auftragsparallelität verwenden, um eine große Anzahl von Aufträgen zu erstellen und gleichzeitig darauf zu warten, dass die Geräte die Auftragsausführung abschließen.

> **Note**
>
> Wenn ein Auftrag mit der optionalen Planungskonfiguration und dem Rollout des Auftragsdokuments, der während eines Wartungsfensters stattfinden soll, den ausgewählten `startTime` erreicht und Sie Ihr maximales Limit für die gleichzeitige Ausführung von Aufträgen erreicht haben, wechselt dieser geplante Auftrag in den Statusstatus `CANCELED`.

Um festzustellen, ob ein Auftrag gleichzeitig ausgeführt wird, können Sie die `IsConcurrent` Eigenschaft eines Jobs in der AWS IoT Konsole oder mithilfe der OR-API verwenden. `DescribeJob` `ListJob` Dieses Limit gilt sowohl für kontinuierliche als auch für Snapshot-Aufträge.

Informationen zu den aktiven Aufträgen und den Grenzwerten für die Parallelität von Aufträgen sowie andere für Sie AWS IoT geltende Kontingente für Aufträge AWS-Konto und zum Beantragen einer Erhöhung des Limits finden Sie unter Endpunkte und Kontingente für die AWS IoT Geräteverwaltung im. Allgemeine AWS-Referenz

Das folgende Diagramm zeigt, wie sich die gleichzeitige Ausführung von Aufträgen auf Aufträge bezieht, die gerade bearbeitet werden, und auf Aufträge, die gerade storniert werden.



> **Note**
>
> Neue Aufträge mit dem optionalen `SchedulingConfig` behalten ihren ursprünglichen Status bei `SCHEDULED` und werden bei Erreichen des ausgewählten `startTime` auf

`IN_PROGRESS` aktualisiert. Sobald der neue Auftrag mit der Option `SchedulingConfig` den ausgewählten `startTime` erreicht und auf `IN_PROGRESS` aktualisiert wird, wird er auf das Limit für aktive Aufträge und das Limit für die gleichzeitige Ausführung von Aufträgen angerechnet. Aufträge mit einem Status von `SCHEDULED` werden auf das Limit für aktive Aufträge angerechnet, nicht aber auf das Limit für die gleichzeitige Ausführung von Aufträgen.

Die folgende Tabelle zeigt die Grenzwerte, die für aktive und gleichzeitige Aufträge gelten, sowie die gleichzeitigen und nicht gleichzeitigen Phasen der Auftragsstatus.

Limits für aktive und gleichzeitige Aufträge

Status des Jobs	Phase	Limit für aktive Aufträge	Grenzwert für die Auftrags-Gleichzeitigkeit
`SCHEDULED`	Non-concurrent Phase: AWS IoT Jobs wartet darauf, dass der für den Job geplante Zeitpunkt beginnt, um mit `startTime` der Auftragsausführung zu beginnen. Benachrichtigungen an Ihre Geräte werden gesendet. Aufträge in dieser Phase werden nur auf das Limit für aktive Aufträge angerechnet und die `IsConcurrent` -Eigenschaft ist auf „Falsch“ gesetzt.	Trifft zu	Trifft nicht zu
`IN_PROGRESS`	Gleichzeitige Phase: AWS IoT Jobs akzeptiert die Anfrage zur Auftragserstellung und beginnt, Benachrichtigungen über die Auftragsausführung an Ihre Geräte zu senden. Aufträge in dieser Phase sind parallel, wie durch die `IsConcurrent` -Eigenschaft angegeben, die auf „true“ gesetzt ist, und werden sowohl auf die aktiven Aufträgen als auch auf die Grenzwerte für die	Trifft zu	Trifft zu

Status des Jobs	Phase	Limit für aktive Aufträge	Grenzwert für die Auftrags-Gleichzeitigkeit
	gleichzeitige Ausführung von Aufträgen angerechnet.		
	Non-concurrent Phase: AWS IoT Jobs warten darauf, dass Geräte die Ergebnisse ihrer Auftragsausführung melden. Aufträge in dieser Phase werden nur auf das Limit für aktive Aufträge angerechnet und die `IsConcurrent` -Eigenschaft ist auf „Falsch“ gesetzt.	Trifft zu	Trifft nicht zu
`Canceled`	Gleichzeitige Phase: AWS IoT Jobs akzeptiert die Anfrage zum Stornieren des Jobs und beginnt, Jobausführungen abzubrechen, die zuvor für Ihre Geräte erstellt wurden. Jobs in dieser Phase sind gleichzeitig und die Eigenschaft `IsConcurrent` wird auf „true“ gesetzt. Sobald der Auftrag und die Auftragsausführungen abgebrochen wurden, ist der Auftrag nicht mehr gleichzeitig und wird nicht auf das Limit für die gleichzeitige Ausführung von Aufträgen angerechnet.	Trifft nicht zu	Trifft zu

> **Note**
>
> Die maximale Dauer eines wiederkehrenden Wartungsfensters beträgt 23 Stunden und 50 Minuten.

AWS IoT Device Management commands

> **Important**
>
> In dieser Dokumentation wird beschrieben, wie Sie die Befehlsfunktion in verwenden können AWS IoT Device Management. Informationen zur Verwendung dieser Funktion für finden Sie AWS IoT FleetWise unter Fernbefehle.
> Sie sind allein dafür verantwortlich, Befehle auf sichere und den geltenden Gesetzen entsprechende Weise bereitzustellen. Weitere Informationen zu Ihren Verantwortlichkeiten finden Sie in den AWS Servicebedingungen für AWS IoT Dienste.

Verwenden Sie AWS IoT Device Management Befehle, um eine Anweisung aus der Cloud an ein Gerät zu senden, mit dem Sie verbunden sind AWS IoT. Befehle zielen jeweils auf ein Gerät ab und können für Anwendungen mit niedriger Latenz und hohem Durchsatz verwendet werden, z. B. zum Abrufen von geräteseitigen Protokollen oder zum Initiieren einer Änderung des Gerätezustands.

Der Befehl ist eine wiederverwendbare Ressource, die von verwaltet wird. AWS IoT Device Management Er enthält Konfigurationen, die angewendet werden, bevor sie auf dem Gerät veröffentlicht werden. Sie können eine Reihe von Befehlen für bestimmte Anwendungsfälle vordefinieren, z. B. das Einschalten einer Glühbirne oder das Entriegeln einer Fahrzeugtür.

Die AWS IoT Befehlsfunktion ermöglicht Ihnen:

- Erstellen Sie wiederverwendbare Befehlsvorlagen mit statischen oder dynamischen Payloads und führen Sie sie dann auf bestimmten Geräten aus.
- Zielgeräte, die als AWS IoT Dinge registriert sind, oder nicht registrierte MQTT-Clients.
- Führen Sie mehrere Befehle gleichzeitig auf demselben Gerät aus.
- Aktivieren Sie Ereignisbenachrichtigungen und verfolgen Sie den Ausführungsstatus, während Geräte Befehle verarbeiten.

In den folgenden Themen erfahren Sie, wie Sie Befehle erstellen, sie an Ihr Gerät senden und den vom Gerät gemeldeten Status abrufen.

Themen

- Schnellstart

- Befehle, Konzepte und Status
- High-level Arbeitsablauf für Befehle
- Erstellen und Verwalten von Befehlen
- Starten und Überwachen von Befehlsausführungen
- Kennzeichnen einer Befehlsressource als veraltet

Schnellstart

Gehen Sie wie folgt vor, um Ihren ersten Befehl zu senden:

1. Voraussetzungen — Stellen Sie sicher, dass Sie bei IoT Core, einem mit IoT Core verbundenen Gerät, IAM-Berechtigungen für Commands-API-Operationen und einer auf Ihrem Gerät installierten MQTT-Clientbibliothek angemeldet sind.
2. Befehl erstellen — Definieren Sie einen Befehl mit einer statischen Nutzlast oder einer Payload-Vorlage, die Geräteaktionen spezifiziert. Siehe Erstellen Sie eine Befehlsressource.
3. Identifizieren Sie Ihr Gerät — Registrieren Sie Ihr Gerät als IoT-Sache oder notieren Sie sich seine MQTT-Client-ID. Siehe Überlegungen zum Zielgerät.
4. Berechtigungen konfigurieren — Richten Sie IAM-Richtlinien ein, die es Ihrem Gerät ermöglichen, Befehle zu empfangen und Antworten zu veröffentlichen.
5. Themen abonnieren — Konfigurieren Sie Ihr Gerät so, dass es Themen zur Anforderung und Beantwortung von Befehlen abonniert. Siehe Wählen Sie das Zielgerät für Ihre Befehle und abonnieren Sie MQTT-Themen.
6. Befehl ausführen — Starten Sie die Befehlsausführung auf Ihrem Zielgerät. Siehe Starten Sie eine Befehlsausführung.

Häufige Anwendungsfälle:

- Ferndiagnose — Rufen Sie Protokolle ab, führen Sie Diagnosen aus, überprüfen Sie den Zustand des Geräts
- Konfigurationsupdates — Einstellungen ändern, Parameter aktualisieren, Funktionen umschalten
- Gerätesteuerung - Lock/unlock, Stromversorgung on/off, Modusänderungen

Befehle, Konzepte und Status

Verwenden Sie AWS IoT Befehle, um Anweisungen aus der Cloud an verbundene Geräte zu senden. Um dieses Feature zu verwenden:

1. Erstellen Sie einen Befehl mit einer Nutzlast, die die Konfigurationen enthält, die für die Ausführung auf dem Gerät erforderlich sind.
2. Geben Sie das Zielgerät an, das die Payload empfangen soll, und führen Sie die Aktionen aus.
3. Führen Sie den Befehl auf dem Zielgerät aus und rufen Sie Statusinformationen ab. Informationen zur Behebung von Problemen finden Sie in den CloudWatch Protokollen.

Weitere Informationen zu diesem Workflow finden Sie unter High-level Arbeitsablauf für Befehle.

Themen

- Befehle und wichtige Konzepte
- Befehlsstatus
- Status der Befehlsausführung

Befehle und wichtige Konzepte

Die folgenden Schlüsselkonzepte helfen Ihnen, die Befehlsfunktion zu verstehen. Begriffe werden in dieser Dokumentation einheitlich verwendet:

- Befehl — Eine wiederverwendbare Vorlage, die Geräteanweisungen definiert
- Ausführung — Eine Instanz eines Befehls, der auf einem Gerät ausgeführt wird
- Name der Sache — Kennung für Geräte, die in der IoT-Registrierung registriert sind
- Client-ID — MQTT-ID für nicht registrierte Geräte
- Payload — Die an Geräte gesendeten Anweisungsdaten
- Thema — MQTT-Kanal für die Befehlskommunikation

Befehle

Befehle sind Anweisungen, die als MQTT-Nachrichten von der Cloud an Ihre IoT-Geräte gesendet werden. Nach Erhalt der Payload verarbeiten die Geräte die Anweisungen und ergreifen

entsprechende Maßnahmen, z. B. das Ändern der Konfigurationseinstellungen, das Übertragen von Sensorwerten oder das Hochladen von Protokollen. Die Geräte geben dann die Ergebnisse an die Cloud zurück und ermöglichen so die Fernüberwachung und -steuerung.

Namespace

Wenn Sie einen Befehl erstellen, geben Sie dessen Namespace an. Verwenden Sie für AWS IoT Device Management Befehle den `AWS-IoT` Standard-Namespace und geben Sie entweder eine Payload oder eine PayloadTemplate an. Verwenden Sie für AWS IoT FleetWise Befehle den Namespace. `AWS-IoT-FleetWise` Weitere Informationen finden Sie im AWS IoT FleetWise Entwicklerhandbuch unter Remote-Befehle.

Nutzlast

Geben Sie beim Erstellen eines Befehls eine statische Nutzlast an, die die Aktionen definiert, die das Gerät ausführen muss. Die Payload kann jedes unterstützte Format verwenden. Um sicherzustellen, dass Geräte die Payload korrekt interpretieren, empfehlen wir, den Payload-Formattyp anzugeben. Geräte, die das MQTT5-Protokoll verwenden, können dem MQTT-Standard folgen, um das Format zu identifizieren. Formatindikatoren für JSON oder CBOR sind im Thema zur Befehlsanforderung verfügbar.

Payload-Vorlage

Eine Payload-Vorlage definiert eine Befehlsnutzlast mit Platzhaltern, die zur Laufzeit auf der Grundlage der von Ihnen angegebenen Parameterwerte unterschiedliche Nutzlasten generieren. Anstatt beispielsweise separate Payloads für verschiedene Temperaturwerte zu erstellen, erstellen Sie eine Vorlage mit einem Temperatur-Platzhalter und geben Sie den Wert bei der Ausführung an. Dadurch entfällt die Verwaltung mehrerer ähnlicher Nutzlasten.

Zielgerät

Um einen Befehl auszuführen, geben Sie ein Zielgerät entweder mit seinem Ding-Namen (für Geräte, die registriert sind AWS IoT) oder mit der MQTT-Client-ID (für nicht registrierte Geräte) an. Die Client-ID ist eine eindeutige Kennung, die in dem MQTT Protokoll definiert ist, mit dem Geräte verbunden werden. AWS IoT Details hierzu finden Sie unter Überlegungen zum Zielgerät.

Befehle, Themen

Bevor ein Befehl ausgeführt wird, müssen Geräte das Thema für die Befehlsanforderung abonnieren. Wenn Sie einen Befehl ausführen, wird die Payload zu diesem Thema an das Gerät gesendet. Nach der Ausführung veröffentlichen Geräte Ergebnisse und Status im Antwortthema „Befehle“. Weitere Informationen finden Sie unter Befehle, Themen.

Ausführung von Befehlen

Eine Ausführung ist eine Instanz eines Befehls, der auf einem Zielgerät ausgeführt wird. Wenn Sie eine Ausführung starten, wird die Nutzlast an das Gerät übermittelt und eine eindeutige Ausführungs-ID wird generiert. Das Gerät führt den Befehl aus und meldet den Fortschritt an. AWS IoT Device-side Die Logik bestimmt das Ausführungsverhalten und die Statusberichterstattung zu reservierten Themen.

Bedingungen für Parameterwerte

Definieren Sie beim Erstellen von Befehlen mit Payload-Vorlagen Wertbedingungen, um Parameterwerte vor der Ausführung zu überprüfen. Wertbedingungen stellen sicher, dass die Parameter die Anforderungen erfüllen, und verhindern so ungültige Ausführungen.

Unterstützte Operatoren nach Typ CommandParameterValue

Numerische Typen (INTEGER, LONG, DOUBLE, UNSIGNEDLONG)

- `EQUALS`- Der Wert muss der angegebenen Zahl entsprechen
- `NOT_EQUALS`- Der Wert darf nicht der angegebenen Zahl entsprechen
- `GREATER_THAN`- Der Wert muss größer als die angegebene Zahl sein
- `GREATER_THAN_EQUALS`- Der Wert muss größer oder gleich der angegebenen Zahl sein
- `LESS_THAN`- Der Wert muss kleiner als die angegebene Zahl sein
- `LESS_THAN_EQUALS`- Der Wert muss kleiner oder gleich der angegebenen Zahl sein
- `IN_RANGE`- Der Wert muss innerhalb des angegebenen Bereichs (einschließlich) liegen
- `NOT_IN_RANGE`- Der Wert muss außerhalb des angegebenen Bereichs (einschließlich) liegen
- `IN_SET`- Der Wert muss mit einer der angegebenen Zahlen übereinstimmen
- `NOT_IN_SET`- Der Wert darf mit keiner der angegebenen Zahlen übereinstimmen

Zeichenfolgentyp (STRING)

- `EQUALS`- Der Wert muss der angegebenen Zeichenfolge entsprechen
- `NOT_EQUALS`- Der Wert darf nicht der angegebenen Zeichenfolge entsprechen
- `IN_SET`- Der Wert muss mit einer der angegebenen Zeichenketten übereinstimmen
- `NOT_IN_SET`- Der Wert darf mit keiner der angegebenen Zeichenketten übereinstimmen

Boolescher Typ

- Wertbedingungen werden nicht unterstützt

Binärer Typ

- Wertbedingungen werden nicht unterstützt

Beispiel: Befehl zur Temperatursteuerung

```
{
  "commandId": "SetTemperature",
  "namespace": "AWS-IoT",
  "payloadTemplate": "{\"temperature\":
 \"${aws:iot:commandexecution::parameter:temperature}\"}",
  "parameters": [
    {
      "name": "temperature",
      "type": "INTEGER",
      "valueConditions": [
        {
          "comparisonOperator": "IN_RANGE",
          "operand": {
            "numberRange": {
              "min": "60",
              "max": "80"
            }
          }
        }
      ]
    }
  ]
}
```

In diesem Beispiel muss der `temperature` Parameter zwischen 60 und 80 (einschließlich) liegen. Ausführungsanforderungen mit Werten außerhalb dieses Bereichs schlagen bei der Überprüfung fehl.

> **Note**
>
> Wertbedingungen werden beim Aufruf der StartCommandExecution API ausgewertet. Fehlgeschlagene Validierungen geben einen Fehler zurück und verhindern die Erstellung der Ausführung.

Priorität und Auswertung von Parameterwerten

Wenn Befehlsausführungen mit Payload-Vorlagen gestartet werden, werden Parameterwerte mit der folgenden Priorität aufgelöst:

1. Parameter der Ausführungsanforderung — Die in der `StartCommandExecution` Anfrage angegebenen Werte haben die höchste Priorität
2. Befehlsstandardwerte — Wenn in der Ausführungsanforderung kein Parameter angegeben ist, `defaultValue` wird der Parameter des Parameters verwendet
3. Kein Wert — Wenn keiner angegeben wird, schlägt die Ausführung fehl, da der Parameter zum Generieren der Ausführungsanforderung erforderlich ist

Die Wertbedingungen werden anhand des endgültigen Parameterwerts ausgewertet, der oben anhand der Priorität und vor der Erstellung der Ausführung abgeleitet wurde. Schlägt die Validierung fehl, gibt die Ausführungsanforderung einen Fehler zurück.

Beispiel: SetTemperature Befehl mit **`defaultValue`**

```
{
  "parameters": [
    {
      "name": "temperature",
      "type": "INTEGER",
      "defaultValue": {"I": 72},
      "valueConditions": [
        {
          "comparisonOperator": "IN_RANGE",
          "operand": {"numberRange": {"min": "60", "max": "80"}}
        }
      ]
    }
  ]
}
```

Beim Start der Ausführung:

- Wenn Sie `"temperature": {"I": 75}` in der Anfrage angeben, wird 75 verwendet
- Wenn Sie den Temperaturparameter weglassen, wird der Standardwert 72 verwendet
- Beide Werte werden anhand der Bereichsbedingung [60,80] validiert

Befehlsstatus

Befehle in Ihrem AWS-Konto können sich in einem von drei Zuständen befinden: Verfügbar, Veraltet oder Ausstehend.

Verfügbar

Nach erfolgreicher Erstellung befindet sich ein Befehl im Status Verfügbar und kann auf Geräten ausgeführt werden.

Als veraltet gekennzeichnet

Markieren Sie Befehle als veraltet, wenn sie nicht mehr benötigt werden. Veraltete Befehle können keine neuen Ausführungen starten, aber ausstehende Ausführungen werden bis zum Abschluss fortgesetzt. Um neue Ausführungen zu ermöglichen, setzen Sie den Befehl auf Verfügbar zurück.

Löschen ausstehend

Wenn Sie einen Befehl zum Löschen markieren, wird er automatisch gelöscht, wenn er länger als der maximale Timeout veraltet ist (Standardeinstellung: 12 Stunden). Diese Aktion ist permanent. Wenn der Befehl nicht veraltet ist oder vor Ablauf des Timeouts als veraltet gilt, wechselt er in den Status Ausstehender Löschvorgang und wird nach Ablauf des Timeouts entfernt.

Status der Befehlsausführung

Wenn Sie eine Ausführung auf einem Zielgerät starten, wechselt das Gerät in den `CREATED` Status und kann aufgrund von Geräteberichten in einen anderen Status übergehen. Sie können Statusinformationen abrufen und Ausführungen verfolgen.

> **Note**
>
> Sie können mehrere Befehle gleichzeitig auf einem Gerät ausführen. Verwenden Sie die Parallelitätssteuerung, um die Ausführung pro Gerät zu begrenzen und eine Überlastung zu verhindern. Informationen zur maximalen Anzahl gleichzeitiger Ausführungen pro Gerät finden Sie unter Befehlskontingente. AWS IoT Device Management

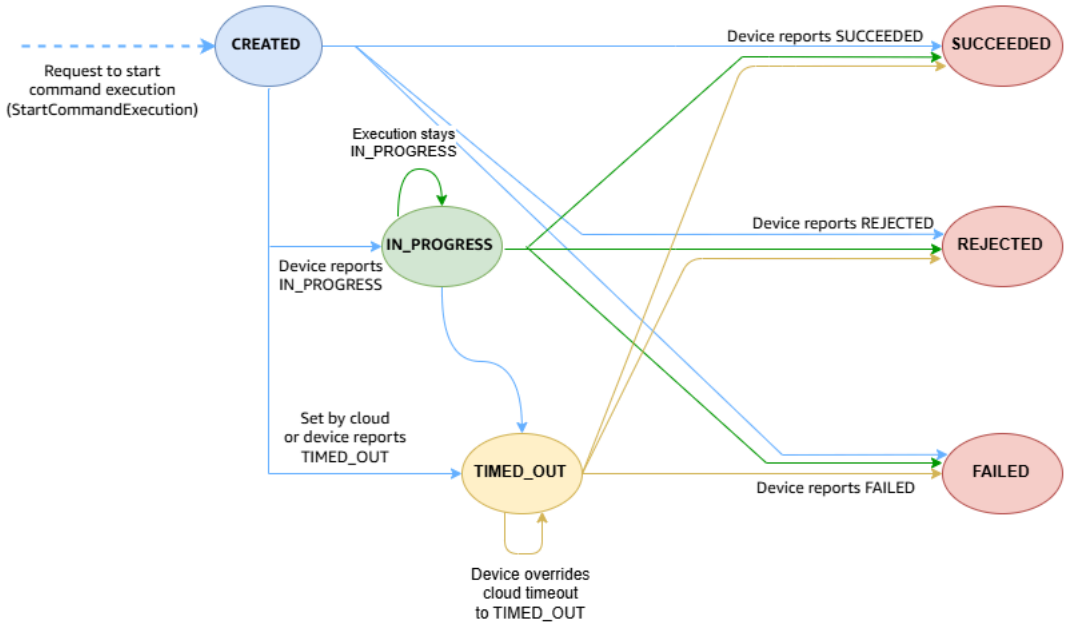
Die folgende Tabelle zeigt den Ausführungsstatus und ihre Übergänge auf der Grundlage des Ausführungsfortschritts.

Status und Quelle der Befehlsausführung

Status der Befehlsausführung	Initiiert von device/cloud?	Terminale Ausführung?	Erlaubte Statusübergänge
`CREATED`	Cloud	Nein	• IN_PROGRESS • SUCCEEDED • FEHLGESCHLAGEN • ABGELEHNT • TIMED_OUT
`IN_PROGRESS`	Gerät	Nein	• IN_PROGRESS • SUCCEEDED • FEHLGESCHLAGEN • ABGELEHNT • ZEITLIMIT ÜBERSCHRITTEN
`TIMED_OUT`	Gerät und Cloud	Nein	• SUCCEEDED • FEHLGESCHLAGEN • ABGELEHNT • TIMED_OUT
`SUCCEEDED`	Gerät	Ja	Nicht zutreffend
`FAILED`	Gerät	Ja	Nicht zutreffend
`REJECTED`	Gerät	Ja	Nicht zutreffend

Geräte können Status- und Ergebnisaktualisierungen jederzeit mithilfe von Befehlen veröffentlichen, die MQTT-Themen vorbehalten sind. Um zusätzlichen Kontext bereitzustellen, können Geräte `reasonCode` und `reasonDescription` Felder im `statusReason` Objekt verwenden.

Das folgende Diagramm zeigt Übergänge zum Ausführungsstatus.



> **Note**
>
> Wenn innerhalb des Timeout-Zeitraums keine Reaktion des Geräts AWS IoT festgestellt wird, wird ein temporärer Status festgelegt`TIMED_OUT`, der Wiederholungen und Statusänderungen ermöglicht. Wenn sich Ihr Gerät explizit meldet`TIMED_OUT`, wird dies in einen Terminalstatus ohne weitere Übergänge umgewandelt. Weitere Informationen finden Sie unter Non-terminal Befehlsausführungen.

In den folgenden Abschnitten werden Terminalausführungen und andere Terminalausführungen sowie deren Status beschrieben.

Themen

- Non-terminal Befehlsausführungen
- Ausführung von Terminal-Befehlen

Non-terminal Befehlsausführungen

Eine Ausführung ist nicht terminal, wenn sie Aktualisierungen von Geräten annehmen kann. Non-terminal Ausführungen werden als aktiv betrachtet. Die folgenden Status sind nicht terminal:

- CREATED

Wenn Sie eine Ausführung von der AWS IoT Konsole aus starten oder die `StartCommandExecution` API verwenden, ändern erfolgreiche Anfragen den Status in. `CREATED` Ab diesem Status können Ausführungen in einen beliebigen anderen Status übergehen, der kein Terminal oder ein Terminal ist.

- IN_PROGRESS

  Nach Erhalt der Payload können Geräte mit der Ausführung von Anweisungen und der Ausführung bestimmter Aktionen beginnen. Während der Ausführung können Geräte Antworten auf das Antwortthema der Befehle veröffentlichen und den Status von aktualisieren. `IN_PROGRESS` Ab `IN_PROGRESS` können Ausführungen in einen beliebigen Terminal- oder Nicht-Terminal-Status übergehen, mit Ausnahme von. `CREATED`

  > **Note**
  >
  > Die `UpdateCommandExecution` API kann mehrfach mit Status aufgerufen werden. `IN_PROGRESS` Geben Sie mithilfe des `statusReason` Objekts zusätzliche Ausführungsdetails an.

- TIMED_OUT

  Sowohl die Cloud als auch das Gerät können diesen Status auslösen. Der `IN_PROGRESS` Status von Ausführungen in `CREATED` oder kann sich aus den folgenden Gründen in ändern: `TIMED_OUT`

  - Nach dem Senden des Befehls startet ein Timer. Wenn das Gerät innerhalb der angegebenen Dauer nicht reagiert, ändert sich der Status der Cloud in`TIMED_OUT`. In diesem Fall erfolgt die Ausführung nicht terminal.
  - Das Gerät kann den Status auf einen beliebigen Terminalstatus überschreiben oder einen Timeout melden und den Status auf setzen. `TIMED_OUT` In diesem Fall bleibt der Status erhalten`TIMED_OUT`, aber die `StatusReason` Objektfelder ändern sich je nach Geräteinformationen. Die Ausführung wird zum Endpunkt.

  Weitere Informationen finden Sie unter Timeout-Wert und _TIMED_OUT-Ausführungsstatus.

Ausführung von Terminal-Befehlen

Eine Ausführung wird zum Terminal, wenn sie keine Updates mehr von Geräten akzeptiert. Die folgenden Status haben den Status „Terminal“. Ausführungen können von jedem Status, der kein Terminal ist, in den Terminalstatus übergehen:`CREATED`, oder. `IN_PROGRESS` `TIMED_OUT`

- SUCCEEDED

  Wenn das Gerät die Ausführung erfolgreich abgeschlossen hat, kann es eine Antwort auf das Antwortthema und die Statusaktualisierung der Befehle veröffentlichen. `SUCCEEDED`

- FEHLGESCHLAGEN

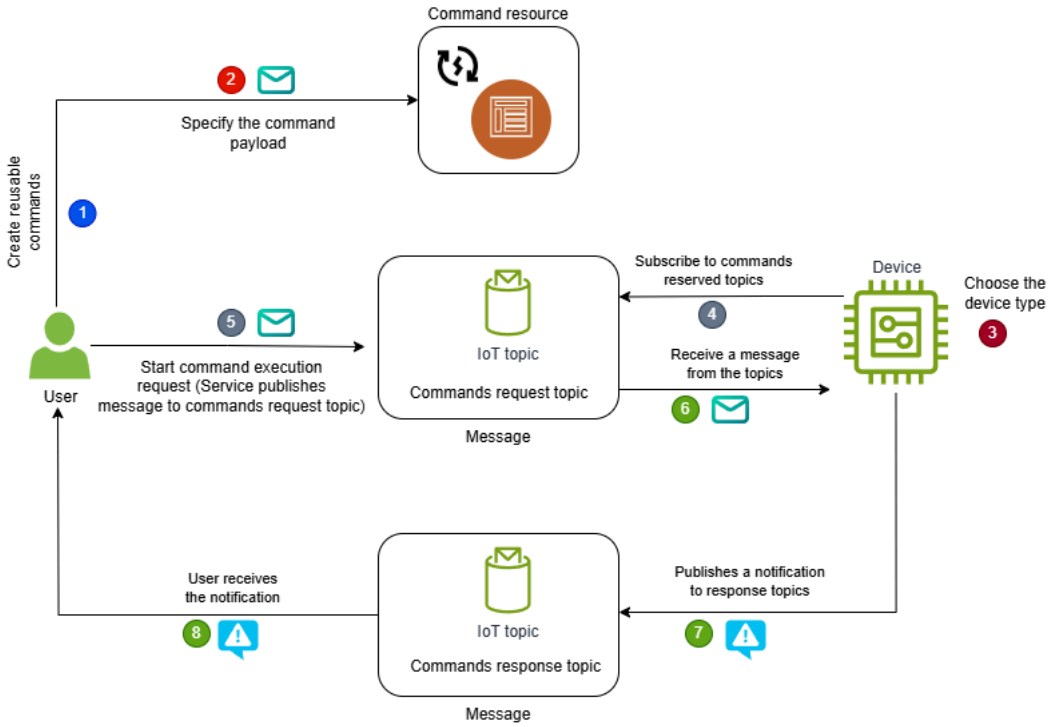
  Wenn ein Gerät die Ausführung nicht abschließen kann, kann es eine Antwort auf das Antwortthema und den Aktualisierungsstatus der Befehle veröffentlichen`FAILED`. Verwenden Sie die `reasonDescription` Felder `reasonCode` und im `statusReason` Objekt oder in den CloudWatch Protokollen, um Fehler zu beheben.

- ABGELEHNT

  Wenn ein Gerät eine ungültige oder inkompatible Anfrage erhält, kann es die `UpdateCommandExecution` API mit Status `REJECTED` aufrufen. Verwenden Sie die `reasonDescription` Felder `reasonCode` und im `statusReason` Objekt oder in den CloudWatch Protokollen, um Probleme zu beheben.

High-level Arbeitsablauf für Befehle

Dieser Arbeitsablauf zeigt, wie Geräte mit AWS IoT Device Management Befehlen interagieren. Alle HTTP-API-Anfragen verwenden [Sigv4-Anmeldeinformationen](#) zum Signieren.



Workflow-Übersicht

- Erstellen und Verwalten von Befehlen
- Wählen Sie das Zielgerät für Ihre Befehle und abonnieren Sie MQTT-Themen
- Starten und überwachen Sie die Befehlsausführung für Ihr Zielgerät
- (Optional) Aktivieren Sie Benachrichtigungen für Befehlsereignisse

Erstellen und Verwalten von Befehlen

Gehen Sie wie folgt vor, um Befehle für Ihre Geräte zu erstellen und zu verwalten.

1. Erstellen Sie eine Befehlsressource

   Erstellen Sie einen Befehl über den Command Hub oder mithilfe der `CreateCommand` API.

2. Geben Sie die Nutzlast an

   Stellen Sie eine Payload in einem beliebigen Format bereit. Geben Sie den Inhaltstyp an, um eine korrekte Geräteinterpretation sicherzustellen.

   Bei dynamischen Befehlen mit Payload-Vorlagen wird die endgültige Nutzlast zur Laufzeit mithilfe der von Ihnen angegebenen Parameter generiert. Vorlagen unterstützen nur das JSON-Format, aber die generierte Payload kann als JSON oder CBOR gesendet werden.

3. (Optional) Verwalte die erstellten Befehle

   Aktualisieren Sie den Anzeigenamen und die Beschreibung nach der Erstellung. Markieren Sie Befehle als veraltet, wenn sie nicht mehr benötigt werden, oder löschen Sie sie dauerhaft. Um Payload-Informationen zu ändern, erstellen Sie einen neuen Befehl.

Wählen Sie das Zielgerät für Ihre Befehle und abonnieren Sie MQTT-Themen

Wählen Sie Ihr Zielgerät und konfigurieren Sie MQTT-Themen für den Empfang von Befehlen und das Veröffentlichen von Antworten.

1. Wählen Sie das Zielgerät für Ihren Befehl

   Wählen Sie ein Zielgerät aus, um den Befehl zu empfangen und auszuführen. Verwenden Sie einen Dingnamen für registrierte Geräte oder eine Client-ID für nicht registrierte Geräte. Weitere Informationen finden Sie unter Überlegungen zum Zielgerät.

2. Konfigurieren Sie das AWS IoT Geräterichtlinie

   Konfigurieren Sie eine IAM-Richtlinie, die Berechtigungen zum Empfangen von Ausführungen und zum Veröffentlichen von Updates gewährt. Beispiele Beispiel einer IAM-Richtlinie für Richtlinien finden Sie unter.

3. Stellen Sie eine MQTT-Verbindung her

   Verbinden Sie Geräte mit dem Message Broker und abonnieren Sie Anfrage- und Antwortthemen. Geräte benötigen eine `iot:Connect` Genehmigung. Finden Sie Ihren Endpunkt auf der Datenebene mithilfe des `DescribeEndpoint` API- oder `describe-endpoint` CLI-Befehls:

   ```
   aws iot describe-endpoint --endpoint-type iot:Data-ATS
   ```

   Wenn Sie diesen Befehl ausführen, wird der kontospezifische Endpunkt der Datenebene zurückgegeben, wie unten dargestellt.

   ```
   account-specific-prefix.iot.region.amazonaws.com
   ```

4. Abonnieren Sie die Themen zu Befehlen

   Abonnieren Sie das Thema zur Anforderung von Befehlen. Wenn Sie eine Ausführung starten, veröffentlicht der Message Broker die Payload in diesem Topic. Ihr Gerät empfängt und verarbeitet den Befehl.

   (Optional) Abonnieren Sie die Antwortthemen (`accepted`oder`rejected`), um eine Bestätigung zu erhalten, ob der Cloud-Dienst die Antwort des Geräts akzeptiert oder abgelehnt hat.

   Ersetzen Sie in diesem Beispiel:

   - *<device>*mit `thing` oder `client` je nachdem, ob das Gerät, auf das Sie abzielen, als IoT-Ding registriert oder als MQTT-Client angegeben wurde.
   - *<DeviceID>*mit der eindeutigen Kennung Ihres Zielgeräts. Diese ID kann die eindeutige MQTT-Client-ID oder ein Dingname sein.

> **Note**
>
> Wenn der Nutzlasttyp nicht JSON oder CBOR ist, ist das *<PayloadFormat>* Feld möglicherweise nicht im Thema der Befehlsanforderung enthalten. Um das Payload-Format abzurufen, empfehlen wir, MQTT5 zu verwenden, um die Formatinformationen aus den MQTT-Nachrichten-Headern abzurufen. Weitere Informationen finden Sie unter Befehle, Themen.

```
$aws/commands/<devices>/<DeviceID>/executions/+/request/<PayloadFormat>
$aws/commands/<devices>/<DeviceID>/executions/+/response/accepted/<PayloadFormat>
$aws/commands/<devices>/<DeviceID>/executions/+/response/rejected/<PayloadFormat>
```

Starten und überwachen Sie die Befehlsausführung für Ihr Zielgerät

Nachdem Sie die Befehle erstellt und die Ziele für den Befehl angegeben haben, können Sie die Ausführung auf dem Zielgerät starten, indem Sie die folgenden Schritte ausführen.

1. Starten Sie die Befehlsausführung auf dem Zielgerät

   Starten Sie die Ausführung vom Command Hub aus oder verwenden Sie die `StartCommandExecution` API mit Ihrem kontospezifischen Endpunkt. Nur `iot:Data-ATS` für Dual-Stack (IPv4/IPv6) oder `iot:Jobs` für IPv4 verwenden.

   Die API veröffentlicht die Payload im Commands Request Topic.

   > **Note**
   >
   > Wenn das Gerät offline ist und persistente MQTT-Sitzungen verwendet, wartet der Befehl beim Message Broker. Wenn das Gerät vor dem Timeout die Verbindung wiederherstellt, kann es den Befehl verarbeiten und Ergebnisse veröffentlichen. Wenn das Timeout abläuft, läuft die Ausführung ab und die Payload wird verworfen.

2. Aktualisieren Sie das Ergebnis der Befehlsausführung

   Das Gerät empfängt die Payload, verarbeitet den Befehl, führt die angegebenen Aktionen aus und veröffentlicht die Ergebnisse mithilfe der `UpdateCommandExecution` MQTT-basierten API im Commands Response Topic. Wenn das Gerät akzeptierte und abgelehnte Themen abonniert hat, erhält es eine Bestätigung, ob der Cloud-Dienst die Antwort akzeptiert oder abgelehnt hat.

   Je nachdem, was Sie im Anforderungsthema angegeben haben, *<devices>* kann es sich entweder um Dinge oder Kunden handeln und dies *<DeviceID>* kann Ihr AWS IoT Dingname oder die MQTT-Client-ID sein.

   > **Note**
   >
   > Das *<PayloadFormat>* kann nur JSON oder CBOR im Antwortthema des Befehls sein.

   ```
   $aws/commands/<devices>/<DeviceID>/executions/<ExecutionId>/
   response/<PayloadFormat>
   ```

3. (Optional) Rufen Sie das Ergebnis der Befehlsausführung ab

   Rufen Sie die Ausführungsergebnisse von der AWS IoT Konsole ab oder verwenden Sie`GetCommandExecution`. Das Gerät muss die Ergebnisse im Antwort-Thema „Befehle“ veröffentlichen, um die neuesten Informationen zu erhalten. Sehen Sie sich zusätzliche Details an, einschließlich Uhrzeit der letzten Aktualisierung, Ergebnis und Abschlusszeit.

(Optional) Aktivieren Sie Benachrichtigungen für Befehlsereignisse

Abonnieren Sie Befehlsereignisse für Benachrichtigungen, wenn sich der Ausführungsstatus ändert. Detaillierte Informationen zu Ereignissen bei der Befehlsausführung, einschließlich des Nachrichtenformats und der Nutzlastattribute, finden Sie unterEreignisse bei der Befehlsausführung.

1. Erstellen einer Themenregel

   Abonnieren Sie das Thema Befehlsereignisse für Benachrichtigungen über Statusänderungen. Erstellen Sie eine Themenregel, um Gerätedaten mithilfe der AWS IoT Konsole oder Erstellen eines AWS IoT Regel an andere AWS IoT Dienste wie AWS Lambda Amazon SQS und AWS Step Functions weiterzuleiten.

Ersetzen Sie in diesem Beispiel *<CommandID>* durch den Bezeichner des Befehls, für den Sie Benachrichtigungen erhalten möchten, und *<CommandExecutionStatus>* durch den Status der Befehlsausführung.

```
$aws/events/commandExecution/<CommandID>/<CommandExecutionStatus>
```

> **Note**
>
> Um Benachrichtigungen für alle Befehle und den Status der Befehlsausführung zu erhalten, können Sie Platzhalterzeichen verwenden und das folgende Thema abonnieren.

```
$aws/events/commandExecution/+/#
```

2. Empfangen und Verarbeiten von Befehlsereignissen

   Verwalten Sie Befehle, Push-Benachrichtigungen und erstellen Sie Anwendungen mithilfe der abonnierten Ereignisse.

Der folgende Code zeigt eine Beispielnutzlast für die Benachrichtigungen über Befehlsereignisse, die Sie erhalten.

```
{
    "executionId": "2bd65c51-4cfd-49e4-9310-d5cbfdbc8554",
    "status":"FAILED",
    "statusReason": {
         "reasonCode": "DEVICE_TOO_BUSY",
         "reasonDescription": ""
    },
    "eventType": "COMMAND_EXECUTION",
    "commandArn":"arn:aws:iot:us-east-1:123456789012:command/0b9d9ddf-
e873-43a9-8e2c-9fe004a90086",
    "targetArn":"arn:aws:iot:us-east-1:123456789012:thing/5006c3fc-
de96-4def-8427-7eee36c6f2bd",
    "timestamp":1717708862107
}
```

Erstellen und Verwalten von Befehlen

Verwenden Sie AWS IoT Device Management Befehle, um wiederverwendbare Fernaktionen zu konfigurieren oder sofortige Anweisungen an Geräte zu senden. Erstellen und verwalten Sie Befehle über die AWS IoT Konsole oder mithilfe der AWS CLI.

Befehle und Operationen erstellen und verwalten

- Erstellen Sie eine Befehlsressource
- Ruft Informationen zu einem Befehl ab
- Listen Sie Befehle in Ihrem auf AWS-Konto
- Aktualisieren Sie eine Befehlsressource
- Verwerfen Sie eine Befehlsressource oder stellen Sie sie wieder her
- Löschen Sie eine Befehlsressource

Erstellen Sie eine Befehlsressource

Geben Sie beim Erstellen eines Befehls die folgenden Informationen an:

- Allgemeine Informationen

  Geben Sie eine eindeutige Befehls-ID an, um den Befehl zu identifizieren, wenn Sie ihn auf Zielgeräten ausführen. Geben Sie optional einen Anzeigenamen, eine Beschreibung und Tags für die Verwaltung an.

- Nutzlast

  Geben Sie für statische Befehle eine Payload an, die Geräteaktionen definiert. Geben Sie den Payload-Formattyp für eine korrekte Geräteinterpretation an.

  Dynamische Befehle finden Sie im Payload-Vorlagenattribut.

- Payload-Vorlage

  Stellen Sie für dynamische Befehle ein PayloadTemplate mit Platzhaltern und Parametern bereit. Geben Sie optional auch Bedingungen an`defaultValue`. AWS IoT Device Management Befehle ersetzen Platzhalter zur Laufzeit. Fehlende Parameter verwenden ihren DefaultValue. Alle Werte müssen definierte Bedingungen erfüllen.

Die folgenden Platzhaltertypen werden unterstützt, bei denen zwischen Groß- und Kleinschreibung unterschieden wird:

- `${aws:iot:commandexecution::parameter:`*`parameter1`*`}`— Ein Platzhalter für den Wert eines Parameters mit dem Namen. `parameter1`
- `${aws:iot:commandexecution::executionTimeoutSec}`— Ein Platzhalter für den Timeout-Parameter für die Befehlsausführung, der zur Laufzeit bereitgestellt wird.

Themen zu Payload und Befehlen

Befehle vorbehalten Topics verwenden ein Format, das auf dem Payload-Formattyp basiert.

- Verwenden Sie für `application/json` `application/cbor` unsere Inhaltstypen dieses Anforderungsthema:

```
$aws/commands/<devices>/<DeviceID>/executions/+/request/<PayloadFormat>
```

- Verwenden Sie für andere Inhaltstypen oder ein nicht spezifiziertes Format dieses Anforderungsthema. Das Payload-Format wird im MQTT-Nachrichtenheader angezeigt.

```
$aws/commands/<devices>/<DeviceID>/executions/+/request
```

Das Antwort-Topic für Befehle verwendet `json` unser `cbor` Format unabhängig vom Payload-Typ. *`<PayloadFormat>`*muss `json` oder `cbor` sein:

```
$aws/commands/<devices>/<DeviceID>/executions/<ExecutionId>/response/<PayloadFormat>
```

Erstellen Sie eine Befehlsressource (Konsole)

In den folgenden Abschnitten werden Überlegungen zum Payload-Format und die Befehlserstellung über die Konsole beschrieben.

Themen

- Payload-Format für statische Befehle
- Dynamisches Payload-Format für Befehle
- Wie erstelle ich einen Befehl (Konsole)

Payload-Format für statische Befehle

Die Payload unterstützt jedes Format bis zu 32 KB. Geben Sie den Payload-Formattyp für eine sichere und korrekte Geräteinterpretation an.

Geben Sie den Payload-Formattyp mithilfe des `type/subtype` Formats an (z. B. `application/json` oder`application/cbor`). Standard: `application/octet-stream`. Unterstützte Formate finden Sie unter Allgemeine MIME-Typen.

Dynamisches Payload-Format für Befehle

Das PayloadTemplate muss ein gültiges JSON mit mindestens einem Platzhalter sein, bis zu 32 KB.

Für den `AwsJsonSubstitution` Präprozessor sendet AWS IoT Device Management Commands Benachrichtigungen im JSON- oder CBOR-Format, basierend auf der Präprozessorkonfiguration.

Wie erstelle ich einen Befehl (Konsole)

Um einen Befehl von der Konsole aus zu erstellen, gehen Sie zu Command Hub und gehen Sie wie folgt vor:

1. Wählen Sie Befehl erstellen.
2. Geben Sie eine eindeutige Befehls-ID an.
3. (Optional) Geben Sie Anzeigenamen, Beschreibung und Tags an.
4. Laden Sie die Payload-Datei hoch, die Geräteaktionen enthält. Geben Sie den Payload-Formattyp für die korrekte Geräteinterpretation an.
5. (Optional) Bei JSON-Payload-Vorlagen mit Platzhaltern werden die Parameter in der Inline-Tabelle zur Bearbeitung vorab ausgefüllt.
6. (Optional) Konfigurieren Sie den Parameterwerttyp (erforderlich), den Standardwert und die Bedingungen.
7. Wählen Sie Befehl „Erstellen“.

Erstellen Sie eine Befehlsressource (CLI)

Verwenden Sie den `CreateCommand` API- oder `create-command` CLI-Befehl, um einen Befehl zu erstellen.

Themen

- Nutzlast des Befehls
- Beispiel einer IAM-Richtlinie
- Beispiel für die Erstellung statischer Befehle
- Beispiel für die dynamische Befehlserstellung

Nutzlast des Befehls

Stellen Sie eine statische Nutzlast oder eine Payload-Vorlage bereit. Statische Payloads sind Base64-kodiert. Bei Payload-Vorlagen wird die endgültige Payload zur Laufzeit mithilfe von Parameterwerten generiert. Geräte verarbeiten die Payload und führen bestimmte Aktionen aus. Geben Sie den Payload-Inhaltstyp für den korrekten Geräteempfang an.

> **Note**
>
> Payloads können nach der Befehlserstellung nicht geändert werden. Erstellen Sie einen neuen Befehl, um die Payload zu ändern.

Beispiel einer IAM-Richtlinie

Bevor Sie diesen API-Vorgang verwenden, stellen Sie sicher, dass Ihre IAM-Richtlinie Sie autorisiert, diese Aktion auf dem Gerät durchzuführen. Das folgende Beispiel zeigt eine IAM-Richtlinie, die dem Benutzer die Berechtigung zum Ausführen der Aktion gewährt. `CreateCommand`

Ersetzen Sie in diesem Beispiel:

- *`region`*mit Ihrem AWS-Region, z. B. *`us-east-1`*
- *`account-id`*mit Ihrer AWS-Konto Nummer, wie z. *`123456789012`* B.
- *`command-id`*mit einer eindeutigen Kennung für Ihre AWS IoT Befehls-ID, z. *`LockDoor`* B. Wenn Sie mehr als einen Befehl senden möchten, können Sie diese Befehle in der IAM-Richtlinie im Abschnitt Ressource angeben.

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": "iot:CreateCommand",
```

```
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:command/command-id"
    }
}
```

Beispiel für die Erstellung statischer Befehle

Das folgende Beispiel zeigt, wie Sie einen statischen Befehl erstellen können. Ersetzen Sie je nach Anwendung:

- *<command-id>*mit einer eindeutigen Kennung für den Befehl. Um beispielsweise die Tür Ihres Hauses zu verriegeln, können Sie Folgendes angeben*LockDoor*. Wir empfehlen, dass Sie UUID verwenden. Sie können auch alphanumerische Zeichen wie „-“ und „_“ verwenden.
- (Optional) *<display-name>* und*<description>*, das sind optionale Felder, die Sie verwenden können, um einen benutzerfreundlichen Namen und eine aussagekräftige Beschreibung für den Befehl anzugeben, z. B. *Lock the doors of my home*
- `namespace`, mit denen Sie den Namespace des Befehls angeben können. Das muss es sein. `AWS-IoT` Informationen zur Verwendung dieser Funktion für AWS IoT FleetWise finden Sie unter Remote-Befehle
- `payload`enthält Informationen über die Payload, die Sie bei der Ausführung des Befehls verwenden möchten, und über deren Inhaltstyp.

```
aws iot create-command \
    --command-id <command-id> \
    --display-name <display-name> \
    --description <description> \
    --namespace AWS-IoT \
    --payload
 '{"content":"eyAibWVzc2FnZSI6ICJIZWxsbyBJb1QiIH0=","contentType":"application/json"}'
```

Wenn Sie diesen Befehl ausführen, wird eine Antwort generiert, die die ID und den ARN (Amazon-Ressourcenname) des Befehls enthält. Wenn Sie den *LockDoor* Befehl beispielsweise bei der Erstellung angegeben haben, zeigt das Folgende eine Beispielausgabe für die Ausführung des Befehls.

```
{
    "commandId": "LockDoor",
```

```
    "commandArn": "arn:aws:iot:us-east-1:123456789012:command/LockDoor"
}
```

Beispiel für die dynamische Befehlserstellung

Das folgende Beispiel zeigt, wie Sie einen dynamischen Befehl erstellen können. Ersetzen Sie je nach Anwendung:

- *<command-id>*mit einer eindeutigen Kennung für den Befehl. Um beispielsweise den Lichtstromstatus festzulegen, können Sie Folgendes angeben*Light_Power_Status*. Wir empfehlen die Verwendung von UUID. Sie können auch alphanumerische Zeichen wie „-“ und „_“ verwenden.
- (Optional) *<display-name>* und*<description>*, das sind optionale Felder, die Sie verwenden können, um einen benutzerfreundlichen Namen und eine aussagekräftige Beschreibung für den Befehl anzugeben, z. B. *Turn a light ON or OFF*
- `namespace`, mit denen Sie den Namespace des Befehls angeben können. Das muss es sein. `AWS-IoT` Informationen zur Verwendung dieser Funktion für AWS IoT FleetWise finden Sie unter Remote-Befehle
- `payloadTemplate`enthält die JSON-formatierte Playload-Vorlage mit Platzhaltern.
- `preprocessor`enthält die Konfiguration für den Präprozessor, die bestimmt, wie das PayloadTemplate verarbeitet werden muss.
- `mandatoryParameter`enthält die Parameter, die den Platzhaltern im PayloadTemplate entsprechen, ihren Typ, ihre Standardwerte und Bedingungen.

```
aws iot create-command \
    --command-id Light_Power_Status \
    --description "Turn a light ON or OFF" \
    --namespace AWS-IoT \
    --payload-
template '{"powerStatus":"${aws:iot:commandexecution::parameter:powerStatus}"}' \
    --preprocessor awsJsonSubstitution={outputFormat=JSON} \
    --mandatory-parameters "name=powerStatus, defaultValue={S=OFF},
 valueConditions=[{comparisonOperator=IN_SET, operand={strings=['ON','OFF']}}]"
```

Wenn Sie diesen Befehl ausführen, wird eine Antwort generiert, die die ID und den ARN (Amazon-Ressourcenname) des Befehls enthält. Wenn Sie den *Light_Power_Status* Befehl beispielsweise

bei der Erstellung angegeben haben, zeigt das Folgende eine Beispielausgabe für die Ausführung des Befehls.

```
{
    "commandId": "Light_Power_Status",
    "commandArn": "arn:aws:iot:us-east-1:123456789012:command/Light_Power_Status"
}
```

Ruft Informationen zu einem Befehl ab

Nachdem Sie einen Befehl erstellt haben, können Sie Informationen dazu von der AWS IoT Konsole abrufen und den verwenden AWS CLI. Sie können die folgenden Informationen abrufen.

- Die Befehls-ID, der Amazon-Ressourcenname (ARN), ein beliebiger Anzeigename und eine Beschreibung, die Sie für den Befehl angegeben haben.
- Der Befehlsstatus, der angibt, ob ein Befehl auf dem Zielgerät ausgeführt werden kann oder ob er veraltet ist oder gelöscht wurde.
- Die Payload oder PayloadTemplate, die Sie bereitgestellt haben.
- Der Präprozessor, den Sie bereitgestellt haben.
- Die obligatorischen Parameter, die Sie angegeben haben.
- Die Uhrzeit, zu der der Befehl erstellt und zuletzt aktualisiert wurde.

Ruft eine Befehlsressource (Konsole) ab

Um einen Befehl von der Konsole abzurufen, gehen Sie zum Command Hub der AWS IoT Konsole und wählen Sie dann den Befehl aus, den Sie erstellt haben, um dessen Details anzuzeigen.

Zusätzlich zu den Befehlsdetails können Sie den Befehlsverlauf einsehen, der Informationen über die Ausführung des Befehls auf dem Zielgerät enthält. Nachdem Sie diesen Befehl auf dem Gerät ausgeführt haben, finden Sie auf dieser Registerkarte Informationen zu den Ausführungen.

Rufen Sie eine Befehlsressource (CLI) ab

Verwenden Sie den API-Vorgang der `GetCommand` HTTP-Steuerungsebene oder den `get-command` AWS CLI Befehl, um Informationen über eine Befehlsressource abzurufen. Sie müssen den Befehl bereits mit der `CreateCommand` API-Anforderung oder der `create-command` CLI erstellt haben.

Beispiel einer IAM-Richtlinie

Bevor Sie diesen API-Vorgang verwenden, stellen Sie sicher, dass Ihre IAM-Richtlinie Sie autorisiert, diese Aktion auf dem Gerät auszuführen. Das folgende Beispiel zeigt eine IAM-Richtlinie, die dem Benutzer die Berechtigung zum Ausführen der Aktion gewährt. `GetCommand`

Ersetzen Sie in diesem Beispiel:

- *`region`*mit Ihrem AWS-Region, z. B. `us-east-1`
- *`account-id`*mit Ihrer AWS-Konto Nummer, wie z. *`123456789023`* B.
- *`command-id`*mit Ihrer AWS IoT eindeutigen Befehlskennung, z. B. *`LockDoor`* oder*`Light_Power_Status`*. Wenn Sie mehr als einen Befehl abrufen möchten, können Sie diese Befehle in der IAM-Richtlinie im Abschnitt Ressource angeben.

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": "iot:GetCommand",
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:command/command-id"
    }
}
```

Rufen Sie ein Befehlsbeispiel ab (AWS CLI)

Das folgende Beispiel zeigt, wie Sie Informationen zu einem Befehl mithilfe von abrufen `get-command` AWS CLI. Ersetzen Sie je nach Anwendung *`<command-id>`* durch den Bezeichner für den Befehl, für den Sie Informationen abrufen möchten. Sie können diese Informationen der Antwort der `create-command` CLI entnehmen.

```
aws iot get-command --command-id <command-id>
```

Wenn Sie diesen Befehl ausführen, wird eine Antwort generiert, die Informationen über den Befehl, die Nutzlast und den Zeitpunkt seiner Erstellung und letzten Aktualisierung enthält. Es enthält auch Informationen, die angeben, ob ein Befehl veraltet ist oder gelöscht wird.

Der folgende Code zeigt beispielsweise eine Beispielantwort.

```
{
    "commandId": "LockDoor",
    "commandArn": "arn:aws:iot:<region>:<account>:command/LockDoor",
    "namespace": "AWS-IoT",
    "payload":{
        "content": "eyAibWVzc2FnZSI6ICJIZWxsbyBJb1QiIH0=",
        "contentType": "application/json"
    },
    "createdAt": "2024-03-23T00:50:10.095000-07:00",
    "lastUpdatedAt": "2024-03-23T00:50:10.095000-07:00",
    "deprecated": false,
    "pendingDeletion": false
}
```

Listen Sie Befehle in Ihrem auf AWS-Konto

Nachdem Sie Befehle erstellt haben, können Sie die Befehle, die Sie in Ihrem Konto erstellt haben, einsehen. In der Liste finden Sie Informationen zu:

- Die Befehls-ID und alle Anzeigenamen, die Sie für die Befehle angegeben haben.
- Der Amazon-Ressourcenname (ARN) der Befehle.
- Der Befehlsstatus, der angibt, ob die Befehle auf dem Zielgerät ausgeführt werden können oder ob sie veraltet sind.

> **Note**
>
> In der Liste werden nicht angezeigt, welche aus Ihrem Konto gelöscht werden. Wenn die Befehle noch gelöscht werden müssen, können Sie die Details für diese Befehle weiterhin anhand ihrer Befehls-ID anzeigen.

- Die Uhrzeit, zu der die Befehle erstellt und zuletzt aktualisiert wurden.

Listet Befehle in Ihrem Konto auf (Konsole)

In der AWS IoT Konsole finden Sie die Liste der Befehle, die Sie erstellt haben, und deren Details, indem Sie zum Command Hub gehen.

Listen Sie die Befehle in Ihrem Konto auf (CLI)

Verwenden Sie die `ListCommands` API-Operation oder die `list-commands` CLI, um die von Ihnen erstellten Befehle aufzulisten.

Beispiel einer IAM-Richtlinie

Bevor Sie diesen API-Vorgang verwenden, stellen Sie sicher, dass Ihre IAM-Richtlinie Sie autorisiert, diese Aktion auf dem Gerät auszuführen. Das folgende Beispiel zeigt eine IAM-Richtlinie, die dem Benutzer die Berechtigung zum Ausführen der Aktion gewährt. `ListCommands`

Ersetzen Sie in diesem Beispiel:

- *`region`*mit Ihrem AWS-Region, z. B. `us-east-1`
- *`account-id`*mit Ihrer AWS-Konto Nummer, wie z. *`123456789012`* B.

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": "iot:ListCommands",
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:command/*"
    }
}
```

Listen Sie die Befehle in Ihrem Kontobeispiel auf

Der folgende Befehl zeigt, wie Sie Befehle in Ihrem Konto auflisten.

```
aws iot list-commands --namespace "AWS-IoT"
```

Wenn Sie diesen Befehl ausführen, wird eine Antwort generiert, die eine Liste der von Ihnen erstellten Befehle, den Zeitpunkt der Erstellung der Befehle und die letzte Aktualisierung enthält. Außerdem werden Informationen zum Befehlsstatus bereitgestellt, die angeben, ob ein Befehl veraltet ist oder auf dem Zielgerät ausgeführt werden kann. Weitere Informationen zu den verschiedenen Status und den Gründen für den Status finden Sie unter. Status der Befehlsausführung

Aktualisieren Sie eine Befehlsressource

Nachdem Sie einen Befehl erstellt haben, können Sie den Anzeigenamen und die Beschreibung des Befehls aktualisieren.

> **Note**
>
> Die Payload für den Befehl kann nicht aktualisiert werden. Um diese Informationen zu aktualisieren oder eine geänderte Nutzlast zu verwenden, müssen Sie einen neuen Befehl erstellen.

Aktualisieren Sie eine Befehlsressource (Konsole)

Um einen Befehl von der Konsole aus zu aktualisieren, wechseln Sie zum Command Hub der AWS IoT Konsole und führen Sie die folgenden Schritte aus.

1. Um eine vorhandene Befehlsressource zu aktualisieren, wählen Sie den Befehl aus, den Sie aktualisieren möchten, und wählen Sie dann unter Aktionen die Option Bearbeiten aus.
2. Geben Sie den Anzeigenamen und die Beschreibung an, die Sie verwenden möchten, sowie alle Name-Wert-Paare als Tags für Ihren Befehl.
3. Wählen Sie Bearbeiten, um den Befehl mit den neuen Einstellungen zu speichern.

Aktualisieren Sie eine Befehlsressource (CLI)

Verwenden Sie die API-Operation der `UpdateCommand` Steuerungsebene oder die `update-command` AWS CLI , um eine Befehlsressource zu aktualisieren. Mit dieser API können Sie:

- Bearbeiten Sie den Anzeigenamen und die Beschreibung eines Befehls, den Sie erstellt haben.
- Verwerfen Sie eine Befehlsressource oder stellen Sie einen Befehl wieder her, der bereits veraltet ist.

Beispiel einer IAM-Richtlinie

Bevor Sie diesen API-Vorgang verwenden, stellen Sie sicher, dass Ihre IAM-Richtlinie Sie autorisiert, diese Aktion auf dem Gerät durchzuführen. Das folgende Beispiel zeigt eine IAM-Richtlinie, die dem Benutzer die Berechtigung zum Ausführen der Aktion gewährt. `UpdateCommand`

Ersetzen Sie in diesem Beispiel:

- *region*mit Ihrem AWS-Region, z. B. `us-east-1`
- *account-id*mit Ihrer AWS-Konto Nummer, wie z. *123456789012* B.
- *command-id*mit Ihrer AWS IoT eindeutigen Befehlskennung, z. *LockDoor* B. Wenn Sie mehr als einen Befehl abrufen möchten, können Sie diese Befehle in der IAM-Richtlinie im Abschnitt Ressource angeben.

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": "iot:UpdateCommand",
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:command/command-id"
    }
}
```

Aktualisieren Sie die Informationen zu einem Befehl — Beispiele (AWS CLI)

Das folgende Beispiel zeigt, wie Sie Informationen zu einem Befehl mithilfe des `update-command` AWS CLI Befehls aktualisieren. Hinweise dazu, wie Sie diese API verwenden können, um eine Befehlsressource zu deaktivieren oder wiederherzustellen, finden Sie unter. Aktualisieren Sie eine Befehlsressource (CLI)

Das Beispiel zeigt, wie Sie den Anzeigenamen und die Beschreibung eines Befehls aktualisieren können. Ersetzen Sie ihn je nach Anwendung *<command-id>* durch den Bezeichner für den Befehl, für den Sie Informationen abrufen möchten.

```
aws iot update-command \
    --command-id <command-id>
    --displayname <display-name> \
    --description <description>
```

Wenn Sie diesen Befehl ausführen, wird eine Antwort generiert, die die aktualisierten Informationen über den Befehl und die Uhrzeit der letzten Aktualisierung enthält. Der folgende Code zeigt ein Beispiel für eine Anforderung und Antwort zur Aktualisierung des Anzeigenamens und der Beschreibung eines Befehls, der den AC ausschaltet.

```
aws iot update-command \
    --command-id <LockDoor> \
    --displayname <Secondary lock door> \
    --description <Locks doors to my home>
```

Wenn Sie diesen Befehl ausführen, wird die folgende Antwort generiert.

```
{
    "commandId": "LockDoor",
    "commandArn": "arn:aws:iot:us-east-1:123456789012:command/LockDoor",
    "displayName": "Secondary lock door",
    "description": "Locks doors to my home",
    "lastUpdatedAt": "2024-05-09T23:15:53.899000-07:00"
}
```

Verwerfen Sie eine Befehlsressource oder stellen Sie sie wieder her

Wenn Sie einen Befehl erstellt haben und den Befehl nicht mehr verwenden möchten, können Sie ihn als veraltet markieren. Wenn Sie einen Befehl als veraltet markieren, werden alle ausstehenden Befehlsausführungen auf dem Zielgerät weiter ausgeführt, bis sie den Terminalstatus erreichen. Sobald ein Befehl veraltet ist und Sie ihn verwenden möchten, z. B. um eine neue Befehlsausführung an das Zielgerät zu senden, müssen Sie ihn wiederherstellen.

> **Note**
>
> Sie können einen veralteten Befehl nicht bearbeiten oder neue Ausführungen dafür ausführen. Um neue Befehle auf dem Gerät auszuführen, müssen Sie es wiederherstellen, sodass sich der Befehlsstatus in Verfügbar ändert.

Weitere Informationen zum Verwerfen und Wiederherstellen eines Befehls sowie Überlegungen dazu finden Sie unter. Kennzeichnen einer Befehlsressource als veraltet

Löschen Sie eine Befehlsressource

Wenn Sie einen Befehl nicht mehr verwenden möchten, können Sie ihn dauerhaft aus Ihrem Konto entfernen. Wenn die Löschaktion erfolgreich ist:

- Wenn der Befehl für eine Dauer, die länger als das maximale Timeout von 12 Stunden ist, veraltet ist, wird der Befehl sofort gelöscht.

- Wenn der Befehl nicht veraltet ist oder für eine kürzere Dauer als das maximale Timeout veraltet ist, befindet sich der Befehl in einem Status. `pending deletion` Er wird nach Ablauf des maximalen Timeouts von 12 Stunden automatisch von Ihrem Konto entfernt.

> **Note**
>
> Der Befehl wird möglicherweise gelöscht, auch wenn Befehlsausführungen ausstehen. Der Befehl befindet sich im Status Ausstehender Löschvorgang und wird automatisch aus Ihrem Konto entfernt.

Löschen Sie eine Befehlsressource (Konsole)

Um einen Befehl von der Konsole zu löschen, wechseln Sie zum Command Hub der AWS IoT Konsole und führen Sie die folgenden Schritte aus.

1. Wählen Sie den Befehl aus, den Sie löschen möchten, und wählen Sie dann unter Aktionen die Option Löschen aus.
2. Bestätigen Sie, dass Sie den Befehl löschen möchten, und wählen Sie dann Löschen aus.

Der Befehl wird zum Löschen markiert und nach 12 Stunden dauerhaft aus Ihrem Konto entfernt.

Löschen Sie eine Befehlsressource (CLI)

Verwenden Sie den API-Vorgang der `DeleteCommand` HTTP-Steuerungsebene oder den `delete-command` AWS CLI Befehl, um eine Befehlsressource zu löschen. Wenn die Löschaktion erfolgreich ist, wird ein HTTP-Wert `statusCode` von 204 oder 202 angezeigt, und der Befehl wird nach Ablauf der maximalen Timeout-Dauer von 12 Stunden automatisch aus Ihrem Konto gelöscht. Im Fall des Status 204 bedeutet dies, dass der Befehl gelöscht wurde.

Beispiel einer IAM-Richtlinie

Bevor Sie diesen API-Vorgang verwenden, stellen Sie sicher, dass Ihre IAM-Richtlinie Sie autorisiert, diese Aktion auf dem Gerät auszuführen. Das folgende Beispiel zeigt eine IAM-Richtlinie, die dem Benutzer die Berechtigung zum Ausführen der Aktion gewährt. `DeleteCommand`

Ersetzen Sie in diesem Beispiel:

- *`region`*mit Ihrem AWS-Region, z. B. `us-east-1`

- *account-id*mit Ihrer AWS-Konto Nummer, wie z. *123456789012* B.
- *command-id*mit Ihrer AWS IoT eindeutigen Befehlskennung, z. *LockDoor* B. Wenn Sie mehr als einen Befehl abrufen möchten, können Sie diese Befehle in der IAM-Richtlinie im Abschnitt Ressource angeben.

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": "iot:DeleteCommand",
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:command/command-id"
    }
}
```

Löschen Sie ein Beispiel für einen Befehl (AWS CLI)

Die folgenden Beispiele zeigen, wie Sie einen Befehl mithilfe des `delete-command` AWS CLI Befehls löschen. Ersetzen Sie ihn je nach Anwendung *<command-id>* durch den Bezeichner für den Befehl, den Sie löschen.

```
aws iot delete-command --command-id <command-id>
```

Wenn die API-Anfrage erfolgreich ist, generiert der Befehl den Statuscode 202 oder 204. Sie können die `GetCommand` API verwenden, um zu überprüfen, ob der Befehl in Ihrem Konto nicht mehr existiert.

Starten und Überwachen von Befehlsausführungen

Nachdem Sie einen Befehl erstellt haben, starten Sie eine Ausführung auf dem Zielgerät. Das Gerät aktualisiert die Ergebnisse und veröffentlicht den Status in den für MQTT reservierten Themen. Rufen Sie den Ausführungsstatus von Ihrem Konto ab und überwachen Sie ihn.

Starten und überwachen Sie Befehle mit der AWS IoT Konsole oder AWS CLI.

Befehle und Operationen starten und überwachen

- Starten Sie eine Befehlsausführung

- Aktualisieren des Ergebnisses einer Befehlsausführung
- Ruft eine Befehlsausführung ab
- Befehlsaktualisierungen mit dem MQTT-Testclient anzeigen
- Listen Sie die Befehlsausführungen in Ihrem auf AWS-Konto
- Löschen Sie eine Befehlsausführung

Starten Sie eine Befehlsausführung

> **Important**
>
> Sie sind allein dafür verantwortlich, Befehle auf sichere und den geltenden Gesetzen entsprechende Weise bereitzustellen.

Bevor Sie eine Ausführung starten, stellen Sie Folgendes sicher:

- Sie haben im AWS IoT Namespace einen Befehl mit Payload-Informationen erstellt. Wenn die Ausführung gestartet wird, verarbeitet das Gerät Payload-Anweisungen und führt bestimmte Aktionen aus. Informationen Erstellen Sie eine Befehlsressource zur Befehlserstellung finden Sie unter.
- Ihr Gerät hat für MQTT reservierte Themen für Befehle abonniert. Wenn Sie mit der Ausführung beginnen, werden Payload-Informationen in dieser reservierten MQTT-Anforderung veröffentlicht. Thema:

  *<devices>*können Things- oder MQTT-Clients sein. *<DeviceID>*ist der Name des Dings oder die Client-ID. Unterstützte *<PayloadFormat>* Werte: JSON und CBOR. Weitere Informationen finden Sie unter Befehle, Themen.

  ```
  $aws/commands/<devices>/<DeviceID>/executions/+/request/<PayloadFormat>
  ```

  Verwenden Sie für „Nicht JSON/CBOR *<PayloadFormat>* -“ das folgende Befehlsthemenformat:

  ```
  $aws/commands/<devices>/<DeviceID>/executions/+/request
  ```

Überlegungen zum Zielgerät

Geben Sie das Zielgerät an, das den Befehl empfangen und ausführen soll. Verwenden Sie einen Dingnamen für registrierte Geräte oder eine Client-ID für nicht registrierte Geräte. Nach Erhalt der Payload führt das Gerät den Befehl aus und führt die angegebenen Aktionen aus.

AWS IoT Objekt

Zielgeräte können Dinge sein, die in der AWS IoT Registrierung registriert sind. Dinge vereinfachen die Gerätesuche und -verwaltung.

Registriere Geräte auf der Connect-Geräteseite oder mithilfe von „Dinge“ `CreateThing`. Finden Sie vorhandene Dinge über Thing Hub oder mithilfe von `DescribeThing`. Einzelheiten zur Registrierung finden Sie unter Dinge mit der Registrierung https://docs.aws.amazon.com/iot/latest/developerguide/thing-registry verwalten.

Client-ID

Verwenden Sie für nicht registrierte Geräte die Client-ID.

Die Client-ID ist eine eindeutige Kennung, die Sie Geräten zuweisen. Sie ist im MQTT-Protokoll definiert und enthält alphanumerische Zeichen, Unterstriche oder Bindestriche. Jedes Gerät, mit dem eine Verbindung hergestellt wird, AWS IoT benötigt eine eindeutige Client-ID.

> **Note**
>
> - Bei registrierten Dingen kann die Client-ID mit dem Namen des Dings übereinstimmen.
> - Wenn Geräte eine bestimmte Client-ID als Targeting verwenden, müssen sie AWS IoT mithilfe dieser Client-ID eine Verbindung herstellen, um die Payload zu empfangen.

Die Client-ID ist die MQTT-Client-ID, mit der Geräte eine Verbindung herstellen. AWS IoT Core AWS IoT verwendet diese ID, um Geräte zu identifizieren und Verbindungen und Abonnements zu verwalten.

Überlegungen zum Timeout bei der Befehlsausführung

Timeout gibt die Dauer (in Sekunden) an, für die Geräte Ausführungsergebnisse bereitstellen.

Nach dem Erstellen einer Ausführung startet ein Timer. Wenn das Gerät offline geht oder innerhalb des Timeouts keine Ergebnisse meldet, wird für die Ausführung ein Timeout mit Status angezeigt`TIMED_OUT`.

Standard: 10 Sekunden. Maximum: 12 Stunden.

Timeout-Wert und TIMED_OUT-Ausführungsstatus

Sowohl die Cloud als auch das Gerät können einen Timeout melden.

Nach dem Senden des Befehls startet ein Timer. Wenn innerhalb des Timeouts keine Geräteantwort eingeht, setzt die Cloud den Ausführungsstatus auf `TIMED_OUT` mit Ursachencode`$NO_RESPONSE_FROM_DEVICE`.

Dies tritt auf, wenn:

- Das Gerät ging während der Ausführung offline.
- Das Gerät konnte die Ausführung nicht innerhalb des Timeouts abschließen.
- Das Gerät konnte den Status innerhalb des Timeouts nicht melden.

In diesem Fall erfolgt die Befehlsausführung nicht `TIMED_OUT` terminalgesteuert, wenn der Ausführungsstatus von aus der Cloud gemeldet wird. Ihr Gerät kann eine Antwort veröffentlichen, die den Status für jeden der Terminalstatus außer Kraft setzt:`SUCCEEDED`, `FAILED` oder. `REJECTED` Die Befehlsausführung wird dann zum Terminal und akzeptiert keine weiteren Aktualisierungen.

Ihr Gerät kann auch einen von der Cloud initiierten `TIMED_OUT` Status aktualisieren, indem es meldet, dass bei der Ausführung des Befehls ein Timeout aufgetreten ist. In diesem Fall bleibt der Ausführungsstatus des Befehls unverändert`TIMED_OUT`, aber das `statusReason` Objekt wird auf der Grundlage der vom Gerät gemeldeten Informationen aktualisiert. Die Befehlsausführung wird dann abgeschlossen, und es werden keine weiteren Aktualisierungen akzeptiert.

Verwendung persistenter MQTT-Sitzungen

Sie können persistente MQTT-Sitzungen für die Verwendung mit der AWS IoT Device Management Befehlsfunktion konfigurieren. Diese Funktion ist besonders nützlich, wenn Ihr Gerät beispielsweise offline geht und Sie sicherstellen möchten, dass das Gerät den Befehl auch dann empfängt, wenn es vor Ablauf der Zeitüberschreitung wieder online ist, und die angegebenen Anweisungen ausführt.

Standardmäßig ist die Ablaufzeit einer persistenten MQTT-Sitzung auf 60 Minuten festgelegt. Wenn Ihr Timeout für die Befehlsausführung auf einen Wert konfiguriert ist, der diese Dauer überschreitet,

können Befehlsausführungen, die länger als 60 Minuten dauern, vom Message Broker abgelehnt werden und schlagen fehl. Um Befehle auszuführen, die länger als 60 Minuten dauern, können Sie eine Verlängerung der Ablaufzeit der persistenten Sitzung anfordern.

> **Note**
>
> Um sicherzustellen, dass Sie die Funktion für persistente MQTT-Sitzungen korrekt verwenden, setzen Sie das Clean Start-Flag auf Null. Weitere Informationen finden Sie unter Persistente MQTT-Sitzungen.

Starten Sie eine Befehlsausführung (Konsole)

Um die Ausführung des Befehls von der Konsole aus zu starten, rufen Sie die Command Hub-Seite der AWS IoT Konsole auf und führen Sie die folgenden Schritte aus.

1. Um den Befehl auszuführen, den Sie erstellt haben, wählen Sie Befehl ausführen.
2. Überprüfen Sie die Informationen zu dem Befehl, den Sie erstellt haben, einschließlich der reservierten MQTT-Themen und Parameter, falls zutreffend.

   Geben Sie für dynamische Befehle die Parameterwerte ein, oder belassen Sie sie bei den Standardwerten. Für Parameter, die keinen Standardwert haben, müssen Sie einen Wert angeben, der im Rahmen dieser Ausführung gesendet wird.
3. Geben Sie das Zielgerät an, das den Befehl empfangen und ausführen soll. Das Gerät kann als Objekt AWS IoT angegeben werden, wenn es registriert wurde AWS IoT, oder mithilfe der Client-ID, wenn Ihr Gerät noch nicht registriert wurde. Weitere Informationen finden Sie unter Überlegungen zum Zielgerät.
4. (Optional) Konfigurieren Sie einen Timeout-Wert für den Befehl, der die Dauer bestimmt, für die der Befehl ausgeführt werden soll, bevor das Zeitlimit überschritten wird. Wenn Ihr Befehl länger als 60 Minuten ausgeführt werden muss, müssen Sie möglicherweise die Ablaufzeit für persistente MQTT-Sitzungen verlängern. Weitere Informationen finden Sie unter Überlegungen zum Timeout bei der Befehlsausführung.
5. Wählen Sie Befehl ausführen aus.

Starten Sie eine Befehlsausführung (AWS CLI)

Verwenden Sie die API-Operation für die `StartCommandExecution` HTTP-Datenebene, um eine Befehlsausführung zu starten. Die API-Anfrage und -Antwort werden durch die Befehlsausführungs-

ID korreliert. Nachdem das Gerät die Ausführung des Befehls abgeschlossen hat, kann es den Status und das Ausführungsergebnis an die Cloud melden, indem es eine Nachricht zum Antwortthema des Befehls veröffentlicht. Bei einem benutzerdefinierten Antwortcode können Anwendungscodes, die Sie besitzen, die Antwortnachricht verarbeiten und das Ergebnis an senden AWS IoT.

Wenn Ihre Geräte das Thema für die Befehlsanfrage abonniert haben, veröffentlicht die `StartCommandExecution` API die Payload-Nachricht zum Thema. Die Payload kann ein beliebiges Format Ihrer Wahl verwenden. Weitere Informationen finden Sie unter Nutzlast des Befehls.

```
$aws/commands/<devices>/<DeviceID>/executions/+/request/<PayloadFormat>
```

Wenn das Payload-Format nicht JSON oder CBOR ist, wird im Folgenden das Format des Anforderungsthemas für Befehle angezeigt.

```
$aws/commands/<devices>/<DeviceID>/executions/+/request
```

Beispiel einer IAM-Richtlinie

Bevor Sie diesen API-Vorgang verwenden, stellen Sie sicher, dass Ihre IAM-Richtlinie Sie autorisiert, diese Aktion auf dem Gerät auszuführen. Das folgende Beispiel zeigt eine IAM-Richtlinie, die dem Benutzer die Berechtigung zum Ausführen der Aktion gewährt. `StartCommandExecution`

Ersetzen Sie in diesem Beispiel:

- *`region`*mit Ihrem AWS-Region, z. B. `us-east-1`
- *`account-id`*mit Ihrer AWS-Konto Nummer, wie z. *`123456789012`* B.
- *`command-id`*mit einer eindeutigen Kennung für Ihren AWS IoT Befehl, z. *`LockDoor`* B. Wenn Sie mehr als einen Befehl senden möchten, können Sie diese Befehle in der IAM-Richtlinie angeben.
- *`devices`*entweder mit `thing` oder `client` je nachdem, ob Ihre Geräte als AWS IoT Dinge registriert wurden oder als MQTT-Clients angegeben sind.
- *`device-id`*mit deinem AWS IoT `thing-name` oder`client-id`.

```
{
  "Effect": "Allow",
  "Action": [
      "iot:StartCommandExecution"
  ],
  "Resource": [
```

```
      "arn:aws:iot:region:account-id:command/command-id",
      "arn:aws:iot:region:account-id:devices/device-id"
  ]
}
```

Eine Liste der unterstützten Bedingungsschlüssel finden Sie AWS IoT im IAM-Benutzerhandbuch unter Bedingungsschlüssel für. `StartCommandExecution`

Rufen Sie den kontospezifischen Endpunkt auf der Datenebene ab

Bevor Sie den API-Befehl ausführen, müssen Sie die kontospezifische Endpunkt-URL für den Endpunkt abrufen. Wenn Sie Dual-Stack-Endpunkte (IPv4 und IPv6) verwenden, verwenden Sie die. `iot:Data-ATS` Der `iot:Jobs` Endpunkt ist nur für IPv4 vorgesehen. Wenn Sie z. B. den folgenden Befehl ausführen:

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

Es gibt die kontospezifische Endpunkt-URL zurück, wie in der Beispielantwort unten gezeigt.

```
{
    "endpointAddress":
    "<account-specific-prefix>-ats.iot.<region>.api.com"
}
```

Starten Sie ein Beispiel für die Befehlsausführung (AWS CLI)

Das folgende Beispiel zeigt, wie die Ausführung eines Befehls mithilfe des `start-command-execution` AWS CLI Befehls gestartet wird.

Ersetzen Sie in diesem Beispiel:

- *<command-arn>*mit dem ARN für den Befehl, den Sie ausführen möchten. Sie können diese Informationen der Antwort auf den `create-command` CLI-Befehl entnehmen. Wenn Sie beispielsweise den Befehl zum Ändern des Lenkradmodus ausführen, verwenden Sie`arn:aws:iot:`*region*`:`*account-id*`:command/`*SetComfortSteeringMode*.
- *<target-arn>*mit dem Thing-ARN für das Zielgerät, das ein IoT-Ding oder ein MQTT-Client sein kann, für das Sie den Befehl ausführen möchten. Wenn Sie beispielsweise den Befehl für das Zielgerät ausführen`myRegisteredThing`, verwenden Sie`arn:aws:iot:`*region*`:`*account-id*`:thing/`*myRegisteredThing*.

- *<endpoint-url>*mit dem kontospezifischen Endpunkt, den Sie erhalten habenRufen Sie den kontospezifischen Endpunkt auf der Datenebene ab, mit dem Präfix. `https://` Beispiel, `https://`*123456789012abcd*`.jobs.iot.`*us-east-1*`.amazonaws.com`.
- (Optional) Sie können auch einen zusätzlichen Parameter angeben`executionTimeoutSeconds`, wenn Sie den `StartCommandExecution` API-Vorgang ausführen. Dieses optionale Feld gibt die Zeit in Sekunden an, innerhalb derer das Gerät die Ausführung des Befehls abschließen muss. In der Standardeinstellung ist der Wert 10 Sekunden. Wenn der Befehlsausführungsstatus lautet`CREATED`, startet ein Timer. Wenn das Ergebnis der Befehlsausführung nicht vor Ablauf des Timers empfangen wird, wechselt der Status automatisch zu`TIMED_OUT`.
- ```
  aws iot-jobs-data start-command-execution \
      --command-arn <command-arn>  \
      --target-arn <target-arn> \
      --endpoint <endpoint-url> \
      --execution-timeout-seconds 900
  ```
- (Optional) Geben Sie bei dynamischen Befehlen die Parameter und ihre Werte an, die für die Ersetzung verwendet werden sollen. Sie müssen einen Wert für Parameter angeben, für die bei der Befehlserstellung kein DefaultValue festgelegt wurde. Wenn ein Parameter einen DefaultValue hat, hat der hier angegebene Parameterwert Vorrang. Bei Parametern, für die valueConditions festgelegt sind, muss der hier angegebene Parameterwert die Bedingung erfüllen.

  Basierend auf einem `Light_Power_Status` dynamischen Befehlsbeispiel:
- ```
  aws iot-jobs-data start-command-execution \
      --command-arn arn:aws:iot:us-east-1:123456789012:command/Light_Power_Status  \
      --target-arn arn:aws:iot:us-east-1:123456789012:thing/exampleThing \
      --endpoint <endpoint-url> \
      --execution-timeout-seconds 900 \
      --parameters "powerStatus={S=ON}"
  ```

Wenn Sie diesen Befehl ausführen, wird eine Befehlsausführungs-ID zurückgegeben. Sie können diese ID verwenden, um den Befehlsausführungsstatus, die Details und den Verlauf der Befehlsausführung abzufragen.

> **Note**
>
> Wenn der Befehl veraltet ist, schlägt die `StartCommandExecution` API-Anforderung mit einer Validierungsausnahme fehl. Um diesen Fehler zu beheben, stellen Sie zuerst

den Befehl mithilfe der `UpdateCommand` API wieder her und führen Sie dann die `StartCommandExecution` Anfrage aus.

```
{
    "executionId": "07e4b780-7eca-4ffd-b772-b76358da5542"
}
```

Aktualisieren des Ergebnisses einer Befehlsausführung

Verwenden Sie den API-Vorgang der `UpdateCommandExecution` MQTT-Datenebene, um den Status oder das Ergebnis einer Befehlsausführung zu aktualisieren.

> **Note**
>
> Bevor Sie diese API verwenden:
>
> - Ihr Gerät muss eine MQTT-Verbindung hergestellt und die Themen für Befehlsanfragen und Antworten abonniert haben. Weitere Informationen finden Sie unter High-level Arbeitsablauf für Befehle.
> - Sie müssen diesen Befehl bereits mit der `StartCommandExecution` API-Operation ausgeführt haben.

Beispiel einer IAM-Richtlinie

Bevor Sie diesen API-Vorgang verwenden, stellen Sie sicher, dass Ihre IAM-Richtlinie Ihr Gerät zur Ausführung dieser Aktionen autorisiert. Im Folgenden wird eine Beispielrichtlinie gezeigt, die Ihr Gerät zur Ausführung der Aktion autorisiert. Weitere Beispiele für IAM-Richtlinien, die dem Benutzer die Berechtigung zum Ausführen der `UpdateCommandExecution` MQTT-Aktion gewähren, finden Sie unter. Beispiele zu den Verbinden- und Veröffentlichen-Richtlinien

Ersetzen Sie in diesem Beispiel:

- *`Region`*mit Ihrem AWS-Region, z. B. `us-east-1`
- *`AccountID`*mit deiner AWS-Konto Nummer, wie zum Beispiel*`123456789012`*.
- *`ThingName`*mit dem Namen Ihres Dings AWS IoT , für das Sie die Befehlsausführung ins Visier nehmen, wie zum Beispiel*`myRegisteredThing`*.

- *commands-request-topic*und *commands-response-topic* mit den Namen Ihrer AWS IoT Befehlsanfrage- und Antwortthemen. Weitere Informationen finden Sie unter High-level Arbeitsablauf für Befehle.

Beispiel für eine IAM-Richtlinie für die MQTT-Client-ID

Der folgende Code zeigt ein Beispiel für eine Geräterichtlinie bei Verwendung der MQTT-Client-ID.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Publish",
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/response",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/response/json"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Receive",
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/request",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/response/accepted",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/response/rejected",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/request/json",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/response/accepted/json",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/clients/
${iot:ClientId}/executions/*/response/rejected/json"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
```

```
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/clients/
${iot:ClientId}/executions/+/request",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/clients/
${iot:ClientId}/executions/+/response/accepted",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/clients/
${iot:ClientId}/executions/+/response/rejected",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/clients/
${iot:ClientId}/executions/+/request/json",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/clients/
${iot:ClientId}/executions/+/response/accepted/json",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/clients/
${iot:ClientId}/executions/+/response/rejected/json"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "arn:aws:iot:us-east-1:123456789012:client/${iot:ClientId}"
    }
  ]
}
```

Beispiel für eine IAM-Richtlinie für eine IoT-Sache

Der folgende Code zeigt ein Beispiel für eine Geräterichtlinie bei der Verwendung AWS IoT eines Dings.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Publish",
      "Resource": "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/response"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Receive",
      "Resource": [
```

```
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/request",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/response/accepted",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/response/rejected",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/request/json",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/response/accepted/json",
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/*/response/rejected/json"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/+/request",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/+/response/accepted",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/+/response/rejected",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/+/request/json",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/+/response/accepted/json",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/commands/things/
${iot:Connection.Thing.ThingName}/executions/+/response/rejected/json"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "arn:aws:iot:us-east-1:123456789012:client/${iot:ClientId}"
    }
  ]
}
```

Wie benutzt man die **`UpdateCommandExecution`** API

Nachdem die Befehlsausführung im Anforderungsthema eingegangen ist, verarbeitet das Gerät den Befehl. Anschließend verwendet es die `UpdateCommandExecution` API, um den Status und das Ergebnis der Befehlsausführung auf das folgende Antwortthema zu aktualisieren.

```
$aws/commands/<devices>/<DeviceID>/executions/<ExecutionId>/response/<PayloadFormat>
```

In diesem Beispiel *<DeviceID>* ist das die eindeutige Kennung Ihres Zielgeräts und *<execution-id>* die Kennung der Befehlsausführung auf dem Zielgerät. Das *<PayloadFormat>* kann JSON oder CBOR sein.

> **Note**
>
> Wenn Sie Ihr Gerät nicht registriert haben AWS IoT, können Sie die Client-ID als Ihre Kennung anstelle eines Objektnamens verwenden.

```
$aws/commands/clients/<ClientID>/executions/<ExecutionId>/response/<PayloadFormat>
```

Das Gerät hat Aktualisierungen des Ausführungsstatus gemeldet

Ihre Geräte können die API verwenden, um jede der folgenden Statusaktualisierungen für die Befehlsausführung zu melden. Weitere Informationen zu diesen Status finden Sie unterStatus der Befehlsausführung.

- `IN_PROGRESS`: Wenn das Gerät mit der Ausführung des Befehls beginnt, kann es den Status auf `IN_PROGRESS` aktualisieren.
- `SUCCEEDED`: Wenn das Gerät den Befehl erfolgreich verarbeitet und die Ausführung abgeschlossen hat, kann das Gerät eine Nachricht im Antwortthema veröffentlichen als`SUCCEEDED`.
- `FAILED`: Wenn das Gerät den Befehl nicht ausführen konnte, kann es eine Nachricht im Antwortthema veröffentlichen als`FAILED`.
- `REJECTED`: Wenn das Gerät den Befehl nicht akzeptiert hat, kann es eine Nachricht im Antwortthema veröffentlichen als`REJECTED`.
- `TIMED_OUT`: Der Ausführungsstatus des Befehls kann sich `TIMED_OUT` aus einem der folgenden Gründe in ändern.

- Das Ergebnis der Befehlsausführung wurde nicht empfangen. Dies kann daran liegen, dass die Ausführung nicht innerhalb der angegebenen Dauer abgeschlossen wurde oder dass das Gerät die Statusinformationen zum Antwortthema nicht veröffentlicht hat.
- Das Gerät meldet, dass beim Versuch, den Befehl auszuführen, ein Timeout aufgetreten ist.

Weitere Informationen zum `TIMED_OUT` Status finden Sie unterTimeout-Wert und TIMED_OUT-Ausführungsstatus.

Überlegungen bei der Verwendung der **`UpdateCommandExecution`** API

Im Folgenden finden Sie einige wichtige Überlegungen bei der Verwendung der `UpdateCommandExecution` API.

- Ihre Geräte können ein optionales `statusReason` Objekt verwenden, um zusätzliche Informationen zur Ausführung bereitzustellen. Wenn Ihre Geräte dieses Objekt bereitstellen, ist das `reasonCode` Feld des Objekts erforderlich, das `reasonDescription` Feld ist jedoch optional.
- Wenn Ihre Geräte das `statusReason` Objekt verwenden, `reasonCode` müssen sie das Muster verwenden `[A-Z0-9_-]+` und dürfen eine Länge von 64 Zeichen nicht überschreiten. Wenn Sie das angeben`reasonDescription`, stellen Sie sicher, dass es eine Länge von 1.024 Zeichen nicht überschreitet. Es können alle Zeichen außer Steuerzeichen wie Zeilenumbrüchen verwendet werden.
- Ihre Geräte können ein optionales `result` Objekt verwenden, um Informationen über das Ergebnis der Befehlsausführung bereitzustellen, z. B. den Rückgabewert eines Remote-Funktionsaufrufs. Wenn Sie das angeben`result`, muss mindestens ein Eintrag erforderlich sein.
- In dem `result` Feld geben Sie die Einträge als Schlüssel-Wert-Paare an. Für jeden Eintrag müssen Sie die Datentypinformationen als Zeichenfolge, boolesch oder binär angeben. Ein Zeichenfolgendatentyp muss den Schlüssel verwendens, ein boolescher Datentyp verwendet den Schlüssel b und ein binärer Datentyp muss den Schlüssel verwenden. `bin` Stellen Sie sicher, dass diese Schlüssel in Kleinbuchstaben geschrieben sind.
- Wenn Sie beim Ausführen der `UpdateCommandExecution` API auf einen Fehler stoßen, können Sie den Fehler in der `AWSIoTLogsV2` Protokollgruppe in Amazon CloudWatch einsehen. Informationen zum Aktivieren der Protokollierung und zum Anzeigen der Protokolle finden Sie unterKonfiguration AWS IoT Protokollierung.

`UpdateCommandExecution`API-Beispiel

Der folgende Code zeigt ein Beispiel dafür, wie Ihr Gerät die `UpdateCommandExecution` API verwenden kann, um den Ausführungsstatus zu melden, das `statusReason` Feld, um zusätzliche Informationen zum Status bereitzustellen, und das Ergebnisfeld, um Informationen über das Ergebnis der Ausführung bereitzustellen, in diesem Fall beispielsweise den Prozentsatz der Autobatterie.

```
{
  "status": "IN_PROGRESS",
  "statusReason": {
    "reasonCode": "200",
    "reasonDescription": "Execution_in_progress"
  },
  "result": {
        "car_battery": {
            "s": "car battery at 50 percent"
        }
    }
}
```

> **Note**
>
> Wenn die `UpdateCommandExecution` MQTT-Anforderung fehlschlägt, veröffentlicht der Dienst eine Fehlerantwort auf das `/rejected` Thema. Eine vollständige Liste der Fehlercodes und Anleitungen zur Problembehandlung finden Sie unterAWS IoT Befehle zur Problembehandlung.

Ruft eine Befehlsausführung ab

Nachdem Sie einen Befehl ausgeführt haben, können Sie Informationen über die Befehlsausführung von der AWS IoT Konsole abrufen und die verwenden AWS CLI. Sie können die folgenden Informationen abrufen.

> **Note**
>
> Um den aktuellen Status der Befehlsausführung abzurufen, muss Ihr Gerät die Statusinformationen mithilfe der `UpdateCommandExecution` MQTT-API im Antwortthema veröffentlichen, wie unten beschrieben. Bis das Gerät eine Veröffentlichung zu diesem

Thema veröffentlicht, meldet die `GetCommandExecution` API den Status als `CREATED` oder`TIMED_OUT`.

Jede Befehlsausführung, die Sie erstellen, hat:

- eine Ausführungs-ID, die eine eindeutige Kennung der Befehlsausführung darstellt.
- den Status der Befehlsausführung. Wenn Sie den Befehl auf dem Zielgerät ausführen, geht die Befehlsausführung in den Zustand `CREATED` über. Sie kann dann in andere Befehlsausführungsstatus übergehen, wie unten beschrieben.
- Das Ergebnis der Befehlsausführung.
- Die eindeutige Befehls-ID und das Zielgerät, für das Ausführungen erstellt wurden.
- Das Startdatum, das die Uhrzeit angibt, zu der die Befehlsausführung erstellt wurde.

Ruft eine Befehlsausführung ab (Konsole)

Sie können eine Befehlsausführung mit einer der folgenden Methoden von der Konsole abrufen.

- Von der Command-Hub-Seite

  Gehen Sie zur Command Hub-Seite der AWS IoT Konsole und führen Sie die folgenden Schritte aus.

  1. Wählen Sie den Befehl aus, für den Sie eine Ausführung auf dem Zielgerät erstellt haben.
  2. Auf der Seite mit den Befehlsdetails auf der Registerkarte Befehlsverlauf werden die von Ihnen erstellten Ausführungen angezeigt. Wählen Sie die Ausführung aus, für die Sie Informationen abrufen möchten.
  3. Wenn Ihre Geräte die `UpdateCommandExecution` API zur Bereitstellung der Ergebnisinformationen verwendet haben, finden Sie diese Informationen dann auf der Registerkarte Ergebnisse auf dieser Seite.
- Von der Thing-Hub-Seite

  Wenn Sie bei der Ausführung des Befehls ein AWS IoT Ding als Zielgerät ausgewählt haben, können Sie die Ausführungsdetails auf der Ding-Hub-Seite einsehen.

1. Gehen Sie in der AWS IoT Konsole zur https://console.aws.amazon.com/iot/home#/thinghub Thing-Hub-Seite und wählen Sie das Ding aus, für das Sie die Befehlsausführung erstellt haben.
2. Auf der Seite mit den Ding-Details sehen Sie im Befehlsverlauf die Ausführungen, die Sie erstellt haben. Wählen Sie die Ausführung aus, für die Sie Informationen abrufen möchten.
3. Wenn Ihre Geräte die `UpdateCommandExecution` API zur Bereitstellung der Ergebnisinformationen verwendet haben, finden Sie diese Informationen dann auf der Registerkarte Ergebnisse auf dieser Seite.

Rufen Sie eine Befehlsausführung (CLI) ab

Verwenden Sie den HTTP-API-Vorgang der `GetCommandExecution` AWS IoT Core Steuerungsebene, um Informationen über eine Befehlsausführung abzurufen. Sie müssen diesen Befehl bereits mit der `StartCommandExecution` API-Operation ausgeführt haben.

Beispiel einer IAM-Richtlinie

Bevor Sie diesen API-Vorgang verwenden, stellen Sie sicher, dass Ihre IAM-Richtlinie Sie autorisiert, diese Aktion auf dem Gerät auszuführen. Das folgende Beispiel zeigt eine IAM-Richtlinie, die dem Benutzer die Berechtigung zum Ausführen der Aktion gewährt. `GetCommandExecution`

Ersetzen Sie in diesem Beispiel:

- *`region`*mit Ihrem AWS-Region, z. B. `us-east-1`
- *`account-id`*mit Ihrer AWS-Konto Nummer, wie z. *`123456789012`* B.
- *`command-id`*mit Ihrer eindeutigen AWS IoT Befehlskennung, z. *`LockDoor`* B.
- *`devices`*entweder mit `thing` oder `client` je nachdem, ob Ihre Geräte als AWS IoT Dinge registriert wurden oder als MQTT-Clients angegeben sind.
- *`device-id`*mit deinem AWS IoT `thing-name` oder`client-id`.

```
{
  "Effect": "Allow",
  "Action": [
      "iot:GetCommandExecution"
  ],
  "Resource": [
```

```
      "arn:aws:iot:region:account-id:command/command-id",
      "arn:aws:iot:region:account-id:devices/device-id"
  ]
}
```

Rufen Sie ein Beispiel für die Befehlsausführung ab

Das folgende Beispiel zeigt, wie Sie Informationen zu einem Befehl abrufen, der mit dem `start-command-execution` AWS CLI Befehl ausgeführt wurde. Das folgende Beispiel zeigt, wie Sie Informationen zu einem Befehl abrufen können, der ausgeführt wurde, um den Lenkradmodus auszuschalten.

Ersetzen Sie in diesem Beispiel:

- *<execution-id>*mit dem Bezeichner für die Befehlsausführung, für die Sie Informationen abrufen möchten.
- *<target-arn>*mit der Amazon-Ressourcennummer (ARN) des Geräts, für das Sie die Ausführung anstreben. Sie können diese Informationen der Antwort auf den `start-command-execution` CLI-Befehl entnehmen.
- Wenn Ihre Geräte die `UpdateCommandExection` API zur Bereitstellung des Ausführungsergebnisses verwendet haben, können Sie optional angeben, ob das Ergebnis der Befehlsausführung in die Antwort der `GetCommandExecution` API mithilfe der `GetCommandExecution` API aufgenommen werden soll.

```
aws iot get-command-execution
    --execution-id <execution-id> \
    --target-arn <target-arn> \
    --include-result
```

Wenn Sie diesen Befehl ausführen, wird eine Antwort generiert, die Informationen über den ARN der Befehlsausführung, den Ausführungsstatus und die Uhrzeit, zu der die Ausführung begonnen und abgeschlossen wurde, enthält. Es stellt auch ein `statusReason` Objekt bereit, das zusätzliche Informationen zum Status enthält. Weitere Informationen zu den verschiedenen Status und dem Grund für den Status finden Sie unterStatus der Befehlsausführung.

Der folgende Code zeigt ein Beispiel für eine Antwort auf die API-Anforderung.

> **Note**
>
> Das `completedAt` Feld in der Ausführungsantwort entspricht dem Zeitpunkt, zu dem das Gerät einen Terminalstatus an die Cloud meldet. Im Falle eines `TIMED_OUT` Status wird dieses Feld nur gesetzt, wenn das Gerät einen Timeout meldet. Wenn der `TIMED_OUT` Status von der Cloud festgelegt wird, wird der `TIMED_OUT` Status nicht aktualisiert. Weitere Informationen zum Timeout-Verhalten finden Sie unterÜberlegungen zum Timeout bei der Befehlsausführung.

```
{
    "executionId": "07e4b780-7eca-4ffd-b772-b76358da5542",
    "commandArn": "arn:aws:iot:us-east-1:123456789012:command/LockDoor",
    "targetArn": "arn:aws:iot:us-east-1:123456789012:thing/myRegisteredThing",
    "status": "SUCCEEDED",
    "statusReason": {
        "reasonCode": "DEVICE_SUCCESSFULLY_EXECUTED",
        "reasonDescription": "SUCCESS"
    },
    "result": {
        "sn": { "s": "ABC-001" },
        "digital": { "b": true }
    },
    "createdAt": "2024-03-23T00:50:10.095000-07:00",
    "completedAt": "2024-03-23T00:50:10.095000-07:00"
}
```

Befehlsaktualisierungen mit dem MQTT-Testclient anzeigen

Sie können den MQTT-Testclient verwenden, um den Nachrichtenaustausch über MQTT anzuzeigen, wenn Sie die Befehlsfunktion verwenden. Nachdem Ihr Gerät eine MQTT-Verbindung mit hergestellt hat AWS IoT, können Sie einen Befehl erstellen, die Nutzlast angeben und ihn dann auf dem Gerät ausführen. Wenn Sie den Befehl ausführen und Ihr Gerät das reservierte MQTT-Anforderungsthema für Befehle abonniert hat, wird die Payload-Meldung angezeigt, die zu diesem Thema veröffentlicht wurde.

Das Gerät empfängt dann die Payload-Anweisungen und führt die angegebenen Operationen auf dem Gerät aus. AWS IoT Anschließend verwendet es die `UpdateCommandExecution` API, um das Ergebnis der Befehlsausführung und die Statusinformationen in den für Befehle reservierten

MQTT-Antwortthemen zu veröffentlichen. AWS IoT Device Management hört sich Aktualisierungen zu den Antwortthemen an und speichert die aktualisierten Informationen und veröffentlicht Protokolle an AWS CloudTrail und Amazon. CloudWatch Sie können dann die neuesten Informationen zur Befehlsausführung von der Konsole oder mithilfe der `GetCommandExecution` API abrufen.

Die folgenden Schritte zeigen, wie Sie den MQTT-Testclient verwenden, um Nachrichten zu beobachten.

1. Öffnen Sie den MQTT-Testclient in der AWS IoT Konsole.
2. Geben Sie auf der Registerkarte Abonnieren das folgende Thema ein und wählen Sie dann Abonnieren. Dabei *<thingId>* befindet sich der Name des Geräts, bei AWS IoT dem Sie sich registriert haben.

   > **Note**
   >
   > Den Dingnamen für Ihr Gerät finden Sie auf der https://console.aws.amazon.com/iot/home#/thinghub Thing-Hub-Seite der AWS IoT Konsole. Wenn Sie Ihr Gerät nicht als Ding registriert haben, können Sie es registrieren, wenn Sie über die Seite „Gerät verbinden“ eine Verbindung herstellen. AWS IoT

   ```
   $aws/commands/things/<thingId>/executions/+/request
   ```

3. (Optional) Auf der Registerkarte Abonnieren können Sie auch die folgenden Themen eingeben und Abonnieren auswählen.

   ```
   $aws/commands/things/+/executions/+/response/accepted/json
   $aws/commands/things/+/executions/+/response/rejected/json
   ```

4. Wenn Sie eine Befehlsausführung starten, wird die Nutzlast der Nachricht unter Verwendung des Anforderungsthemas, das das Gerät abonniert hat, an das Gerät gesendet. `$aws/commands/things/<thingId>/executions/+/request` Im MQTT-Testclient sollten Sie die Befehlspayload sehen, die die Anweisungen für das Gerät zur Verarbeitung des Befehls enthält.
5. Nachdem das Gerät mit der Ausführung des Befehls begonnen hat, kann es Statusaktualisierungen zum folgenden reservierten MQTT-Antwortthema für Befehle veröffentlichen.

```
$aws/commands/<devices>/<device-id>/executions/<executionId>/response/json
```

Stellen Sie sich zum Beispiel einen Befehl vor, den Sie ausgeführt haben, um die Klimaanlage Ihres Autos einzuschalten, um die Temperatur auf einen gewünschten Wert zu senken. Die folgende JSON-Datei zeigt eine Beispielnachricht, die das Fahrzeug zum Antwortthema veröffentlicht hat und aus der hervorgeht, dass der Befehl nicht ausgeführt werden konnte.

```
{
  "deviceId": "My_Car",
  "executionId": "07e4b780-7eca-4ffd-b772-b76358da5542",
  "status": "FAILED",
  "statusReason": {
    "reasonCode": "CAR_LOW_ON_BATTERY",
    "reasonDescription": "Car battery is lower than 5 percent"
  }
}
```

In diesem Fall können Sie die Batterie Ihres Autos aufladen und den Befehl dann erneut ausführen.

Listen Sie die Befehlsausführungen in Ihrem auf AWS-Konto

Nachdem Sie einen Befehl ausgeführt haben, können Sie Informationen über die Befehlsausführung von der AWS IoT Konsole abrufen und die AWS CLI verwenden. Sie können die folgenden Informationen abrufen.

- eine Ausführungs-ID, die eine eindeutige Kennung der Befehlsausführung darstellt.
- den Status der Befehlsausführung. Wenn Sie den Befehl auf dem Zielgerät ausführen, geht die Befehlsausführung in den Zustand `CREATED` über. Sie kann dann in andere Befehlsausführungsstatus übergehen, wie unten beschrieben.
- Die eindeutige Befehls-ID und das Zielgerät, für das Ausführungen erstellt wurden.
- Das Startdatum, das die Uhrzeit angibt, zu der die Befehlsausführung erstellt wurde.

Listet die Befehlsausführungen in Ihrem Konto auf (Konsole)

Sie können alle Befehlsausführungen von der Konsole aus mit einer der folgenden Methoden sehen.

- Von der Command-Hub-Seite

  Gehen Sie zur Command Hub-Seite der AWS IoT Konsole und führen Sie die folgenden Schritte aus.

  1. Wählen Sie den Befehl aus, für den Sie eine Ausführung auf dem Zielgerät erstellt haben.
  2. Gehen Sie auf der Seite mit den Befehlsdetails zur Registerkarte Befehlsverlauf. Dort wird eine Liste der von Ihnen erstellten Ausführungen angezeigt.
- Von der Thing-Hub-Seite

  Wenn Sie bei der Ausführung des Befehls ein AWS IoT Ding als Zielgerät ausgewählt und mehrere Befehlsausführungen für ein einzelnes Gerät erstellt haben, können Sie die Ausführungen für das Gerät auf der Ding-Hub-Seite anzeigen.

  1. Gehen Sie in der AWS IoT Konsole zur https://console.aws.amazon.com/iot/home#/thinghub Thing-Hub-Seite und wählen Sie das Ding aus, für das Sie die Ausführungen erstellt haben.
  2. Auf der Seite mit den Ding-Details sehen Sie im Befehlsverlauf eine Liste der Ausführungen, die Sie für das Gerät erstellt haben.

Listet die Befehlsausführungen in Ihrem Konto auf (CLI)

Verwenden Sie den HTTP-API-Vorgang der `ListCommandExecutions` AWS IoT Core Steuerungsebene, um alle Befehlsausführungen in Ihrem Konto aufzulisten.

Beispiel einer IAM-Richtlinie

Bevor Sie diesen API-Vorgang verwenden, stellen Sie sicher, dass Ihre IAM-Richtlinie Sie autorisiert, diese Aktion auf dem Gerät auszuführen. Das folgende Beispiel zeigt eine IAM-Richtlinie, die dem Benutzer die Berechtigung zum Ausführen der Aktion gewährt. `ListCommandExecutions`

Ersetzen Sie in diesem Beispiel:

- *`region`*mit Ihrem AWS-Region, z. B. `us-east-1`
- *`account-id`*mit Ihrer AWS-Konto Nummer, wie z. *`123456789012`* B.
- *`command-id`*mit Ihrer eindeutigen AWS IoT Befehlskennung, z. *`LockDoor`* B.

```
{
  "Effect": "Allow",
```

```
  "Action": "iot:ListCommandExecutions",
  "Resource": *
}
```

Beispiel für die Ausführung von Befehlen auflisten

Das folgende Beispiel zeigt Ihnen, wie Sie Befehlsausführungen in Ihrem auflisten. AWS-Konto

Wenn Sie den Befehl ausführen, müssen Sie angeben, ob die Liste so gefiltert werden soll, dass nur Befehlsausführungen angezeigt werden, die für ein bestimmtes Gerät mithilfe von erstellt wurden`targetArn`, oder Ausführungen für einen bestimmten Befehl, der mit dem angegeben wurde. `commandArn`

Ersetzen Sie in diesem Beispiel:

- *<target-arn>*mit der Amazon-Ressourcennummer (ARN) des Geräts, für das Sie die Ausführung anstreben, z. B. `arn:aws:iot:`*us-east-1*`:`*123456789012*`:thing/`*b8e4157c98f332cffb37627f*
- *<target-arn>*mit der Amazon-Ressourcennummer (ARN) des Geräts, für das Sie die Ausführung anstreben, z. `arn:aws:iot:`*us-east-1*`:`*123456789012*`:thing/`*b8e4157c98f332cffb37627f* B.
- *<after>*mit der Zeit, nach der Sie die Ausführungen auflisten möchten, die erstellt wurden, `2024-11-01T03:00` z. B.

```
aws iot list-command-executions \
--target-arn <target-arn> \
--started-time-filter '{after=<after>}' \
--sort-order "ASCENDING"
```

Wenn Sie diesen Befehl ausführen, wird eine Antwort generiert, die eine Liste der von Ihnen erstellten Befehlsausführungen sowie die Uhrzeit, zu der die Ausführung der Ausführungen begonnen und abgeschlossen wurde, enthält. Es enthält auch Statusinformationen und das `statusReason` Objekt, das zusätzliche Informationen zum Status enthält.

```
{
    "commandExecutions": [
        {
            "commandArn": "arn:aws:iot:us-east-1:123456789012:command/TestMe002",
            "executionId": "b2b654ca-1a71-427f-9669-e74ae9d92d24",
```

```
            "targetArn": "arn:aws:iot:us-east-1:123456789012:thing/
b8e4157c98f332cffb37627f",
            "status": "TIMED_OUT",
            "createdAt": "2024-11-24T14:39:25.791000-08:00",
            "startedAt": "2024-11-24T14:39:25.791000-08:00"
        },
        {
            "commandArn": "arn:aws:iot:us-east-1:123456789012:command/TestMe002",
            "executionId": "34bf015f-ef0f-4453-acd0-9cca2d42a48f",
            "targetArn": "arn:aws:iot:us-east-1:123456789012:thing/
b8e4157c98f332cffb37627f",
            "status": "IN_PROGRESS",
            "createdAt": "2024-11-24T14:05:36.021000-08:00",
            "startedAt": "2024-11-24T14:05:36.021000-08:00"
        }
    ]
}
```


Weitere Informationen zu den verschiedenen Status und dem Grund für den Status finden Sie unterStatus der Befehlsausführung.

Löschen Sie eine Befehlsausführung

Wenn Sie eine Befehlsausführung nicht mehr verwenden möchten, können Sie sie dauerhaft aus Ihrem Konto entfernen.

> **Note**
>
> - Eine Befehlsausführung kann nur gelöscht werden, wenn sie einen Terminalstatus wie `SUCCEEDED`FAILED, oder erreicht hat`REJECTED`.
> - Dieser Vorgang kann nur mithilfe der AWS IoT Core API oder der ausgeführt werden AWS CLI. Er ist nicht über die Konsole verfügbar.

Beispiel einer IAM-Richtlinie

Bevor Sie diesen API-Vorgang verwenden, stellen Sie sicher, dass Ihre IAM-Richtlinie Ihr Gerät autorisiert, diese Aktionen auszuführen. Im Folgenden wird eine Beispielrichtlinie gezeigt, die Ihr Gerät zur Ausführung der Aktion autorisiert.

Ersetzen Sie in diesem Beispiel:

- *Region*mit Ihrem AWS-Region, wie z. B. `us-east-1`
- *AccountID*mit Ihrer AWS-Konto Nummer, wie z. *123456789012* B.
- *CommandID*mit der Kennung des Befehls, für den Sie die Ausführung löschen möchten.
- *devices*entweder mit `thing` oder `client` je nachdem, ob Ihre Geräte als AWS IoT Dinge registriert wurden oder als MQTT-Clients angegeben sind.
- *device-id*mit deinem AWS IoT `thing-name` oder`client-id`.

```
{
  "Effect": "Allow",
  "Action": [
      "iot:DeleteCommandExecution"
  ],
  "Resource": [
      "arn:aws:iot:region:account-id:command/command-id",
      "arn:aws:iot:region:account-id:devices/device-id"
  ]
}
```

Löschen Sie ein Beispiel für die Befehlsausführung

Das folgende Beispiel zeigt, wie Sie einen Befehl mithilfe des `delete-command` AWS CLI Befehls löschen. Ersetzen Sie je nach Anwendung *<execution-id>* durch den Bezeichner für die Befehlsausführung, den Sie löschen, und den *<target-arn>* durch den ARN Ihres Zielgeräts.

```
aws iot delete-command-execution \
--execution-id <execution-id> \
--target-arn <target-arn>
```

Wenn die API-Anfrage erfolgreich ist, generiert die Befehlsausführung den Statuscode 200. Sie können die `GetCommandExecution` API verwenden, um zu überprüfen, ob die Befehlsausführung in Ihrem Konto nicht mehr existiert.

Kennzeichnen einer Befehlsressource als veraltet

Verwerfen Sie Befehle als veraltet, um anzuzeigen, dass sie veraltet sind und nicht verwendet werden sollten. Veraltete Befehle werden beispielsweise nicht mehr aktiv verwaltet oder wenn neuere Befehle mit derselben ID, aber unterschiedlichen Nutzlasten erstellt werden.

Wesentliche Überlegungen

Wichtige Überlegungen beim Ablehnen von Befehlen:

- Wenn ein Befehl als veraltet eingestuft wird, wird er nicht gelöscht. Sie können den Befehl mithilfe seiner ID abrufen und zur Wiederverwendung wiederherstellen.
- Wenn Sie versuchen, veraltete Befehle erneut auszuführen, wird ein Fehler generiert, der die Verwendung veralteter Befehle verhindert.
- Um einen veralteten Befehl auszuführen, müssen Sie ihn zunächst wiederherstellen. Nach der Wiederherstellung steht der Befehl zur regelmäßigen Verwendung und Ausführung auf Zielgeräten zur Verfügung.
- Wenn Sie einen Befehl während der Ausführung als veraltet markieren, werden sie bis zum Abschluss weiter ausgeführt. Sie können den Ausführungsstatus weiterhin abrufen.

Verwerfen Sie eine Befehlsressource (Konsole)

Um einen Befehl von der Konsole als veraltet zu erklären, rufen Sie den Command Hub der AWS IoT Konsole auf und führen Sie die folgenden Schritte aus.

1. Wählen Sie den Befehl aus, den Sie ablehnen möchten, und wählen Sie dann unter Aktionen die Option Verwerfen aus.
2. Bestätigen Sie, dass Sie den Befehl als veraltet ansehen möchten, und wählen Sie dann Verwerfen aus.

Verwerfen Sie eine Befehlsressource (CLI)

Markieren Sie Befehle mithilfe der CLI als veraltet. `update-command` Sie müssen einen Befehl vor dem Löschen als veraltet markieren. Um einen veralteten Befehl zu verwenden, stellen Sie ihn zuerst wieder her.

```
aws iot update-command \
    --command-id <command-id> \
    --deprecated
```

Wenn Sie beispielsweise den *ACSwitch* Befehl, den Sie im obigen Beispiel aktualisiert haben, als veraltet markiert haben, zeigt der folgende Code eine Beispielausgabe der Ausführung des Befehls.

```
{
    "commandId": "turnOffAc",
    "deprecated": true,
    "lastUpdatedAt": "2024-05-09T23:16:51.370000-07:00"
}
```

Überprüfen Sie die Uhrzeit und den Status der veralteten Version

Verwenden Sie die `GetCommand` API, um festzustellen, ob ein Befehl veraltet ist und wann er zuletzt veraltet war.

```
aws iot get-command --command-id <turnOffAC>
```

Dieser Befehl generiert eine Antwort mit Befehlsinformationen, einschließlich der Zeitstempel für Erstellung und Verfall, aus dem zuletzt aktualisierten Feld. Dies hilft bei der Bestimmung der Lebensdauer des Befehls und ob er gelöscht oder wiederverwendet werden soll. Das Folgende zeigt eine Beispielantwort für den *`turnOffAc`* Befehl:

```
{
    "commandId": "turnOffAC",
    "commandArn": "arn:aws:iot:us-east-1:123456789012:command/turnOffAC",
    "namespace": "AWS-IoT",
    "payload": {
        "content": "testPayload.json",
        "contentType": "application/json"
    },
    "createdAt": "2024-03-23T00:50:10.095000-07:00",
    "lastUpdatedAt": "2024-05-09T23:16:51.370000-07:00",
    "deprecated": false
}
```

Stellen Sie eine Befehlsressource wieder her

Um den `ACSwitch` Befehl zu verwenden oder an Ihr Gerät zu senden, stellen Sie ihn zuerst wieder her.

Um einen Befehl von der Konsole aus wiederherzustellen, wechseln Sie zum Command Hub der AWS IoT Konsole, wählen Sie den Befehl aus, den Sie wiederherstellen möchten, und wählen Sie dann unter Aktionen die Option Wiederherstellen aus.

Um einen Befehl mithilfe der AWS IoT Core API oder der wiederherzustellen AWS CLI, verwenden Sie die `UpdateCommand` API-Operation oder die `update-command` CLI. Der folgende Code zeigt ein Beispiel für eine Anfrage und Antwort.

```
aws iot update-command \
    --command-id <command-id>
    --no-deprecated
```

Der folgende Code zeigt eine Beispielausgabe.

```
{
    "commandId": "ACSwitch",
    "deprecated": false,
    "lastUpdatedAt": "2024-05-09T23:17:21.954000-07:00"
}
```

AWS IoT sicheres Tunneln

Wenn Geräte hinter eingeschränkten Firewalls an Remote-Standorten bereitgestellt werden, brauchen Sie für die Fehlerbehebung, Konfigurationsupdates und andere betriebliche Aufgaben eine Zugriffsmöglichkeit auf diese Geräte. Verwenden Sie Secure Tunneling, um eine bidirektionale Kommunikation mit Remote-Geräten über eine sichere Verbindung herzustellen, die von verwaltet wird. AWS IoT Für Secure Tunneling sind keine Aktualisierungen der vorhandenen eingehenden Firewallregel erforderlich, sodass Sie die gleiche Sicherheitsstufe beibehalten können, die von Firewallregeln an einem Remote-Standort bereitgestellt wird.

Beispiel: Ein Sensorgerät, das sich in einer Fabrik befindet, die ein paar hundert Kilometer entfernt ist, hat Probleme bei der Messung der Temperatur in der Fabrik. Sie können Secure Tunneling verwenden, um eine Sitzung mit diesem Sensorgerät zu öffnen und schnell zu starten. Nachdem Sie das Problem identifiziert haben (z. B. eine fehlerhafte Konfigurationsdatei), können Sie die Datei zurücksetzen und das Sensorgerät über dieselbe Sitzung neu starten. Im Vergleich zu herkömmlichen Fehlerbehebungen (z. B. beim Senden eines Technikers zur Fabrik, um das Sensorgerät zu untersuchen) senkt Secure Tunneling die Reaktionszeit, die Wiederherstellungszeit und die Betriebskosten.

Was ist Secure Tunneling?

Verwenden Sie Secure Tunneling, um auf Geräte zuzugreifen, die hinter portbeschränkten Firewalls an entfernten Standorten installiert sind. Sie können von Ihrem Laptop oder Desktop-Computer als Quellgerät aus eine Verbindung zum Zielgerät herstellen, indem Sie den AWS Cloud verwenden. Quelle und Ziel kommunizieren über einen lokalen Open-Source-Proxy, der auf jedem Gerät ausgeführt wird. Der lokale Proxy kommuniziert mit dem über einen offenen Port, der AWS Cloud von der Firewall zugelassen wird, in der Regel 443. Daten, die durch den Tunnel übertragen werden, werden mittels Transported Layer Security (TLS) verschlüsselt.

Themen

- Secure Tunneling-Konzepte
- Wie funktioniert Secure Tunneling
- Sicherer Tunnellebenszyklus

Secure Tunneling-Konzepte

Die folgenden Begriffe werden beim Secure Tunneling verwendet, wenn die Kommunikation mit Remote-Geräten hergestellt wird. Weitere Informationen zum Einrichten eines sicheren Tunnels finden Sie unter Wie funktioniert Secure Tunneling.

Client-Zugriffstoken (CAT)

Ein Token-Paar, das durch Secure Tunneling generiert wird, wenn ein neuer Tunnel erstellt wird. Das CAT wird von den Quell- und Zielgeräten verwendet, um eine Verbindung mit dem Secure-Tunneling-Service herzustellen. Das CAT kann nur einmal verwendet werden, um eine Verbindung zum Tunnel herzustellen. Um die Verbindung zum Tunnel wiederherzustellen, rotieren Sie die Client-Zugriffstoken mithilfe der RotateTunnelAccessToken API-Operation oder des CLI-Befehls https://docs.aws.amazon.com/cli/latest/reference/iotsecuretunneling/rotate-tunnel-access-token.html rotate-tunnel-access-token.

Client-Token

Ein vom Client generierter eindeutiger Wert, den AWS IoT Secure Tunneling für alle nachfolgenden Wiederholungsverbindungen zum gleichen Tunnel verwenden kann. Dies ist ein optionales Feld. Wenn das Client-Token nicht bereitgestellt wird, kann das Client-Zugriffstoken (CAT) nur einmal für denselben Tunnel verwendet werden. Nachfolgende Verbindungsversuche mit demselben CAT werden abgewiesen. Weitere Informationen zur Verwendung von Client-Token finden Sie in der lokalen Proxy-Referenzimplementierung unter. GitHub

Zielanwendung

Die Anwendung, die auf dem Zielgerät ausgeführt wird. Beispielsweise kann die Zielanwendung ein SSH-Daemon für die Einrichtung einer SSH-Sitzung mit Secure Tunneling sein.

Zielgerät

Das Remote-Gerät, auf das Sie zugreifen möchten.

Geräte-Agent

Eine IoT-Anwendung, die eine Verbindung zum AWS IoT Geräte-Gateway herstellt und über MQTT auf neue Tunnelbenachrichtigungen wartet. Weitere Informationen finden Sie unter IoT-Agent-Snippet.

Lokaler Proxy

Ein Software-Proxy, der auf den Quell- und Zielgeräten ausgeführt wird und einen Datenstrom zwischen dem Secure Tunneling-Service und der Geräteanwendung ermöglicht. Der lokale Proxy

kann im Quell- oder Zielmodus ausgeführt werden. Weitere Informationen finden Sie unter Lokaler Proxy.

Quellgerät

Das Gerät, das ein Bediener verwendet, um eine Sitzung mit dem Zielgerät zu initiieren, normalerweise ein Laptop oder Desktop-Computer.

Tunnel

Ein logischer Pfad, der AWS IoT die bidirektionale Kommunikation zwischen einem Quellgerät und einem Zielgerät ermöglicht.

Wie funktioniert Secure Tunneling

Im Folgenden wird gezeigt, wie Secure Tunneling eine Verbindung zwischen Ihrem Quell- und Zielgerät herstellt. Informationen zu den verschiedenen Begriffen wie Client-Zugriffstoken (CAT) finden Sie unter. Secure Tunneling-Konzepte

1. Tunnel öffnen

   Um einen Tunnel für die Initiierung einer Sitzung mit Ihrem Remote-Zielgerät zu öffnen, können Sie den AWS-Managementkonsole Befehl AWS CLI open-tunnel oder die API verwenden. OpenTunnel
2. Client-Zugriffstokenpaar herunterladen

   Nachdem Sie einen Tunnel geöffnet haben, können Sie das Client-Zugriffstoken (CAT) für Ihre Quelle und Ihr Ziel herunterladen und auf Ihrem Quellgerät speichern. Sie müssen das CAT jetzt abrufen und speichern, bevor Sie den lokalen Proxy starten können.
3. Lokalen Proxy im Zielmodus starten

   Der IoT-Agent, der auf Ihrem Zielgerät installiert wurde und läuft, abonniert das reservierte MQTT-Thema `$aws/things/thing-name/tunnels/notify` und erhält das CAT. Hier *`thing-name`* ist der Name der AWS IoT Sache, die Sie für Ihr Ziel erstellen. Weitere Informationen finden Sie unter Themen zu Secure Tunneling.

   Der IoT-Agent verwendet dann das CAT, um den lokalen Proxy im Zielmodus zu starten und eine Verbindung auf der Zielseite des Tunnels einzurichten. Weitere Informationen finden Sie unter IoT-Agent-Snippet.
4. Lokalen Proxy im Quellmodus starten

AWS IoT Device Management Stellt nach dem Öffnen des Tunnels den CAT für die Quelle bereit, den Sie auf das Quellgerät herunterladen können. Sie können das CAT verwenden, um den lokalen Proxy im Quellmodus zu starten, der dann die Quellseite des Tunnels verbindet. Weitere Informationen zum lokalen Proxy finden Sie unter Lokaler Proxy.

5. Eine SSH-Sitzung öffnen

   Da beide Seiten des Tunnels verbunden sind, können Sie eine SSH-Sitzung starten, indem Sie den lokalen Proxy auf der Quellseite verwenden.

Weitere Informationen zur Verwendung des zum Öffnen eines Tunnels und AWS-Managementkonsole zum Starten einer SSH-Sitzung finden Sie unterÖffnen Sie einen Tunnel und starten Sie eine SSH-Sitzung zum Remote-Gerät.

Das folgende Video beschreibt, wie Secure Tunneling funktioniert, und führt Sie durch den Prozess der Einrichtung einer SSH-Sitzung zu einem Raspberry Pi-Gerät.

Sicherer Tunnellebenszyklus

Tunnel können den Status `OPEN` oder `CLOSED` haben. Verbindungen zum Tunnel können den Status `CONNECTED` oder `DISCONNECTED` haben. Im Folgenden wird gezeigt, wie die verschiedenen Tunnel- und Verbindungsstatus funktionieren.

1. Wenn Sie einen Tunnel öffnen, hat er den Status `OPEN`. Der Quell- und Zielverbindungsstatus des Tunnels ist auf `DISCONNECTED` eingestellt.
2. Wenn sich ein Gerät (Quelle oder Ziel) mit dem Tunnel verbindet, ändert sich der zugehörige Verbindungsstatus in `CONNECTED`.
3. Wenn ein Gerät die Verbindung zum Tunnel trennt, während der Tunnelstatus `OPEN` bleibt, wechselt der entsprechende Verbindungsstatus wieder auf `DISCONNECTED`. Ein Gerät kann wiederholt eine Verbindung zu einem Tunnel herstellen und diese trennen, solange der Tunnel `OPEN` bleibt.

> **Note**
>
> Das Client-Zugriffstoken (CAT) kann nur einmal verwendet werden, um eine Verbindung zum Tunnel herzustellen. Um die Verbindung zum Tunnel wiederherzustellen, rotieren Sie die Client-Zugriffstoken mithilfe der RotateTunnelAccessToken API-Operation oder des CLI-Befehls https://docs.aws.amazon.com/cli/latest/reference/iotsecuretunneling/rotate-

tunnel-access-token.html rotate-tunnel-access-token. Beispiele finden Sie unter Auflösen AWS IoT Probleme mit der Tunneling-Konnektivität durch rotierendes Rotieren der Client-Zugriffstoken.

4. Wenn Sie `CloseTunnel` aufrufen oder der Tunnel `OPEN` verbleibt und zwar länger als der `MaxLifetimeTimeout` -Wert, wird der Status eines Tunnels zu `CLOSED`. Sie können `MaxLifetimeTimeout` beim Aufruf von `OpenTunnel` konfigurieren. `MaxLifetimeTimeout` ist standardmäßig 12 Stunden, wenn Sie keinen Wert angeben.

> **Note**
>
> Ein Tunnel kann nicht wieder geöffnet werden, wenn er `CLOSED` ist.

5. Während der Tunnel sichtbar ist, können Sie `DescribeTunnel` und `ListTunnels` aufrufen, um Tunnelmetadaten anzuzeigen. Der Tunnel kann mindestens drei Stunden in der AWS IoT Konsole sichtbar sein, bevor er gelöscht wird.

AWS IoT Tutorials zum sicheren Tunneln

AWS IoT Secure Tunneling hilft Kunden dabei, eine bidirektionale Kommunikation mit Remote-Geräten herzustellen, die sich hinter einer Firewall befinden, und zwar über eine sichere Verbindung, die von verwaltet wird. AWS IoT

Um AWS IoT sicheres Tunneling zu demonstrieren, verwenden Sie unsere AWS IoT Demo zum sicheren Tunneling unter. GitHub

In den folgenden Tutorials erfahren Sie, wie Sie mit Secure Tunneling beginnen und wie Sie es verwenden können. Sie lernen, wie Sie:

1. Einen sicheren Tunnel mithilfe des Quick Setups und der manuellen Einrichtung für den Zugriff auf das Remote-Gerät erstellen.
2. Den lokalen Proxy konfigurieren, wenn Sie die manuelle Einrichtungsmethode verwenden, und eine Verbindung zum Tunnel herstellen, um auf das Zielgerät zuzugreifen.
3. Über einen Browser eine SSH-Verbindung zum Remote-Gerät herstellen, ohne den lokalen Proxy konfigurieren zu müssen.
4. Konvertieren Sie einen Tunnel, der mit der AWS CLI oder mit der manuellen Einrichtungsmethode erstellt wurde, um die Schnellinstallationsmethode zu verwenden.

Tutorials in diesem Abschnitt

Die Tutorials in diesem Abschnitt konzentrieren sich auf das Erstellen eines Tunnels mithilfe der AWS-Managementkonsole und der AWS IoT API-Referenz. In der AWS IoT Konsole können Sie einen Tunnel von der Hub-Seite Tunnels oder von der Detailseite eines Objekts, das Sie erstellt haben, aus erstellen. Weitere Informationen finden Sie unter Methoden zur Tunnelerstellung in AWS IoT Konsole.

Dieser Abschnitt enthält die folgenden Tutorials:

- Öffnen Sie einen Tunnel und verwenden Sie browserbasiertes SSH, um auf das Remote-Gerät zuzugreifen

  Dieses Tutorial zeigt, wie Sie mithilfe von Quick Setup einen Tunnel von der Tunnel-Hub-Seite aus öffnen. Sie lernen auch, wie Sie browserbasiertes SSH verwenden, um über eine kontextabhängige Befehlszeilenschnittstelle in der Konsole auf das AWS IoT Remote-Gerät zuzugreifen.

- Öffnen Sie mithilfe der manuellen Einrichtung einen Tunnel und stellen Sie eine Verbindung zum Remote-Gerät her

  Dieses Tutorial zeigt, wie Sie mithilfe der manuellen Einrichtungsmethode einen Tunnel von der Tunnel-Hub-Seite aus öffnen. Sie lernen auch, wie Sie den lokalen Proxy von einem Terminal auf Ihrem Quellgerät aus konfigurieren und starten und eine Verbindung zum Tunnel herstellen.
- Öffnen Sie einen Tunnel und verwenden Sie browserbasiertes SSH, um auf das Remote-Gerät zuzugreifen

  Dieses Tutorial zeigt, wie Sie einen Tunnel von der Detailseite eines Objekts aus öffnen, das Sie erstellt haben. Sie lernen, wie Sie einen neuen Tunnel erstellen und einen vorhandenen verwenden. Der bestehende Tunnel entspricht dem letzten offenen Tunnel, der für das Gerät erstellt wurde. Sie können auch das browserbasierte SSH verwenden, um auf das Remote-Gerät zuzugreifen.

AWS IoT Tutorials zum sicheren Tunneling

- Öffnen Sie einen Tunnel und starten Sie eine SSH-Sitzung zum Remote-Gerät
- Öffnen Sie einen Tunnel und verwenden Sie browserbasiertes SSH, um auf das Remote-Gerät zuzugreifen

Öffnen Sie einen Tunnel und starten Sie eine SSH-Sitzung zum Remote-Gerät

In diesen Tutorials lernen Sie, wie Sie per Fernzugriff auf ein Gerät zugreifen können, das sich hinter einer Firewall befindet. Sie können keine direkte SSH-Sitzung auf dem Gerät starten, da die Firewall den gesamten eingehenden Datenverkehr blockiert. Die Tutorials zeigen Ihnen, wie Sie einen Tunnel öffnen und diesen Tunnel dann verwenden können, um eine SSH-Sitzung zu einem Remote-Gerät zu starten.

Voraussetzungen für das Tutorial

Die Voraussetzungen für die Ausführung des Tutorials können variieren, je nachdem, ob Sie die manuelle oder die Quick Setup Methode für das Öffnen eines Tunnels und den Zugriff auf das Remote-Gerät verwenden.

> **Note**
>
> Für beide Einrichtungsmethoden müssen Sie ausgehenden Datenverkehr auf Port 443 zulassen.

- Informationen zu den Voraussetzungen für das Tutorial zur Quick Setup Methode finden Sie unter Voraussetzungen für die Quick Setup Methode.
- Informationen zu den Voraussetzungen für das Tutorial zur manuellen Einrichtungsmethode finden Sie unter Voraussetzungen für die manuelle Einrichtungsmethode. Wenn Sie diese Einrichtungsmethode verwenden, müssen Sie den lokalen Proxy auf Ihrem Quellgerät konfigurieren. Informationen zum Herunterladen des lokalen Proxy-Quellcodes finden Sie unter Implementierung der lokalen Proxy-Referenz unter. GitHub

Methoden zur Tunneleinrichtung

In diesen Tutorials erfahren Sie mehr über die manuellen und Quick Setup Methoden zum Öffnen eines Tunnels und zum Herstellen einer Verbindung zum Remote-Gerät. In der folgenden Tabelle wird der Unterschied zwischen den Einrichtungsmethoden dargestellt. Nachdem Sie den Tunnel erstellt haben, können Sie über eine Befehlszeilen-Schnittstelle im Browser eine SSH-Verbindung zum Remote-Gerät herstellen. Wenn Sie die Token verlegen oder der Tunnel unterbrochen wird, können Sie neue Zugriffstoken senden, um die Verbindung zum Tunnel wiederherzustellen.

Schnelle und manuelle Einrichtungsmethoden

Kriterien	Quick Setup	Manuelle Einrichtung
Tunnel-Erstellung	Erstellen Sie einen neuen Tunnel mit editierbaren Standardkonfigurationen. Um auf Ihr Remote-Gerät zuzugreifen, können Sie SSH nur als Zieldienst verwenden.	Erstellen Sie einen Tunnel, indem Sie die Tunnelkonfigurationen manuell angeben. Sie können diese Methode verwenden, um über andere Dienste als SSH eine Verbindung zum Remote-Gerät herzustellen.
Zugriffstoken	Das Zielzugriffstoken wird Ihrem Gerät automatisch zum reservierten MQTT-Thema zugestellt, wenn bei der Erstellung des Tunnels ein Objektname angegeben wurde. Sie müssen das Token nicht auf Ihr Quellgerät herunterladen oder dort verwalten.	Sie müssen das Token manuell auf Ihrem Quellgerät herunterladen und verwalten. Das Zielzugriffstoken wird Ihrem Gerät automatisch zum reservierten MQTT-Thema zugestellt, wenn bei der Erstellung des Tunnels ein Objektname angegeben wurde.
Lokaler Proxy	Für die Interaktion mit dem Gerät wird automatisch ein webbasierter lokaler Proxy für Sie konfiguriert. Sie müssen den lokalen Proxy nicht manuell konfigurieren.	Sie müssen den lokalen Proxy manuell konfigurieren und starten. Um den lokalen Proxy zu konfigurieren, können Sie entweder den AWS IoT Device Client verwenden oder die Referenzimplementierung für den lokalen Proxy unter herunterladen GitHub.

Methoden zur Tunnelerstellung in AWS IoT Konsole

Die Tutorials in diesem Abschnitt zeigen Ihnen, wie Sie einen Tunnel mithilfe der AWS-Managementkonsole und der OpenTunnel API erstellen. Wenn Sie das Ziel beim Erstellen eines Tunnels konfigurieren, übermittelt AWS IoT Secure Tunneling das Zugriffstoken für den Ziel-Client über MQTT und das reservierte MQTT-Thema () an das Remote-Gerät. `$aws/things/RemoteDeviceA/tunnels/notify` Nach Erhalt der MQTT-Nachricht startet der IoT-Agent auf dem Remote-Gerät den lokalen Proxy im Zielmodus. Weitere Informationen finden Sie unter Reservierte Themen.

> **Note**
>
> Sie können die Zielkonfiguration weglassen, wenn Sie das Zugriffstoken für den Zielclient über eine andere Methode an das Remote-Gerät senden möchten. Weitere Informationen finden Sie unter Ein Remote-Gerät konfigurieren und IoT-Agent verwenden.

In der AWS IoT Konsole können Sie einen Tunnel mit einer der folgenden Methoden erstellen. Informationen zu Tutorials, in denen Sie lernen, wie Sie mit diesen Methoden einen Tunnel erstellen, finden Sie unter Tutorials in diesem Abschnitt.

- Tunnel-Hub

  Bei der Erstellung des Tunnels können Sie angeben, ob Sie Quick Setup oder die manuelle Einrichtung für die Erstellung des Tunnels verwenden möchten, und die optionalen Details zur Tunnelkonfiguration angeben. Zu den Konfigurationsdetails gehören auch der Name des Zielgeräts und der Dienst, den Sie für die Verbindung mit dem Gerät verwenden möchten. Nachdem Sie einen Tunnel erstellt haben, können Sie entweder SSH im Browser verwenden oder ein Terminal außerhalb der AWS IoT Konsole öffnen, um auf Ihr Remote-Gerät zuzugreifen.

- Seite mit Objektdetails

  Bei der Erstellung des Tunnels können Sie außerdem angeben, ob Sie den zuletzt geöffneten Tunnel verwenden oder einen neuen Tunnel für das Gerät erstellen möchten. Außerdem können Sie die Einrichtungsmethoden auswählen und alle optionalen Details zur Tunnelkonfiguration angeben. Sie können die Konfigurationsdetails eines vorhandenen Tunnels nicht bearbeiten. Sie können die Quick Setup Methode verwenden, um die Zugriffstoken zu rotieren und über den Browser eine SSH-Verbindung zum Remote-Gerät herzustellen. Um einen Tunnel mit dieser Methode zu öffnen, müssen Sie (z. B.`RemoteDeviceA`) ein IoT-Ding in der AWS IoT Registrierung erstellt haben. Weitere Informationen finden Sie unter Registrieren eines Geräts in der AWS IoT Registrierung.

Tutorials in diesem Abschnitt

- Öffnen Sie einen Tunnel und verwenden Sie browserbasiertes SSH, um auf das Remote-Gerät zuzugreifen
- Öffnen Sie mithilfe der manuellen Einrichtung einen Tunnel und stellen Sie eine Verbindung zum Remote-Gerät her

Öffnen Sie einen Tunnel und verwenden Sie browserbasiertes SSH, um auf das Remote-Gerät zuzugreifen

Sie können die Quick Setup Methode oder die manuelle Einrichtungsmethode verwenden, um einen Tunnel zu erstellen. Dieses Tutorial zeigt, wie Sie einen Tunnel mit der Quick Setup Methode öffnen und das browserbasierte SSH verwenden, um eine Verbindung zum Remote-Gerät herzustellen. Ein Beispiel, das zeigt, wie ein Tunnel mithilfe der manuellen Einrichtungsmethode geöffnet wird, finden Sie unter Öffnen Sie mithilfe der manuellen Einrichtung einen Tunnel und stellen Sie eine Verbindung zum Remote-Gerät her.

Mithilfe der Quick Setup Methode können Sie einen neuen Tunnel mit Standardkonfigurationen erstellen, die bearbeitet werden können. Ein webbasierter lokaler Proxy wird für Sie konfiguriert und das Zugriffstoken wird automatisch über MQTT an Ihr Remote-Zielgerät gesendet. Nachdem Sie einen Tunnel erstellt haben, können Sie über eine Befehlszeilen-Schnittstelle innerhalb der Konsole mit Ihrem Remote-Gerät interagieren.

Bei der Quick Setup Methode müssen Sie SSH als Zieldienst für den Zugriff auf das Remote-Gerät verwenden. Weitere Informationen zu den verschiedenen Einrichtungsmethoden finden Sie unter Methoden zur Tunneleinrichtung.

Voraussetzungen für die Quick Setup Methode

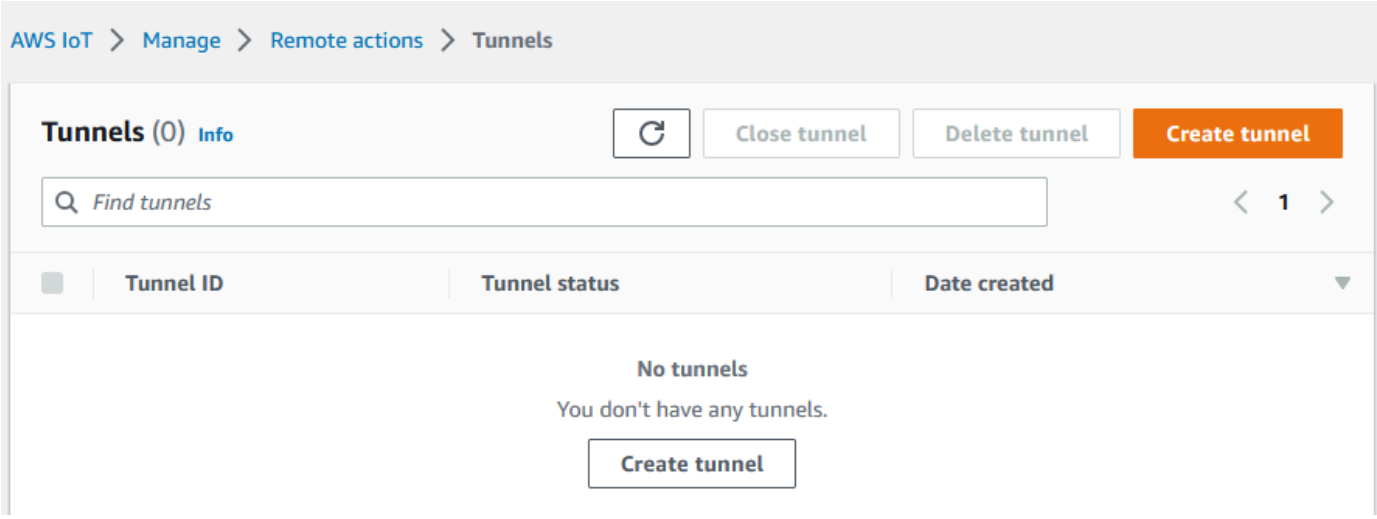
- Die Firewalls, hinter denen sich das Remote-Gerät befindet, müssen ausgehenden Datenverkehr an Port 443 zulassen. Der Tunnel, den Sie erstellen, verwendet diesen Port, um eine Verbindung zum Remote-Gerät herzustellen.
- Auf dem Remote-Gerät läuft ein IoT-Geräteagent (siehe IoT-Agent-Snippet), der eine Verbindung zum Geräte-Gateway herstellt und mit einem MQTT-Themenabonnement konfiguriert ist. AWS IoT Weitere Informationen finden Sie unter Ein Gerät mit dem AWS IoT Geräte-Gateway https://docs.aws.amazon.com/iot/latest/developerguide/sdk-tutorials.html verbinden.
- Auf dem Remote-Gerät muss ein SSH-Daemon ausgeführt werden.

Tunnel öffnen

Sie können einen sicheren Tunnel mit der AWS-Managementkonsole, der AWS IoT API-Referenz oder der öffnen AWS CLI. Sie können optional einen Zielnamen konfigurieren, der für dieses Tutorial jedoch nicht erforderlich ist. Wenn Sie das Ziel konfigurieren, übermittelt Secure Tunneling das Zugriffstoken automatisch mithilfe von MQTT an das Remote-Gerät. Weitere Informationen finden Sie unter Methoden zur Tunnelerstellung in AWS IoT Konsole.

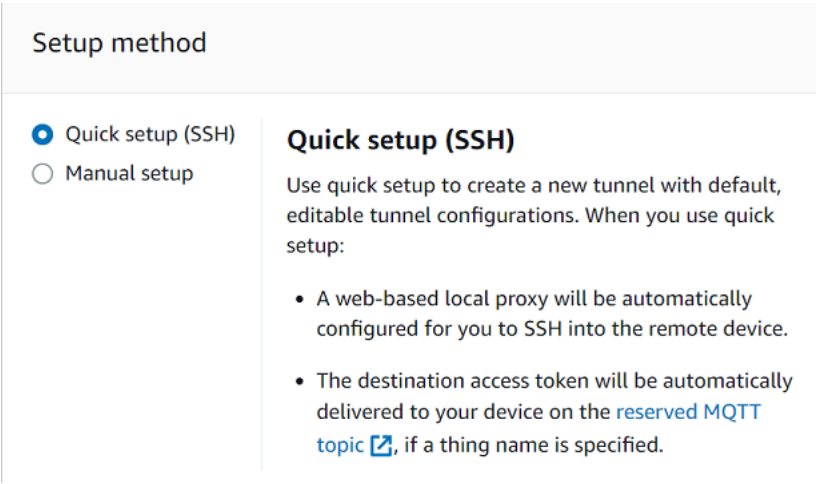
Um einen Tunnel mit der Konsole zu öffnen,

1. Gehen Sie zum Tunnel-Hub der AWS IoT -Konsole und wählen Sie Tunnel erstellen.



2. Wählen Sie für dieses Tutorial die Quick Setup Methode zur Tunnelerstellung und wählen Sie dann Weiter.

> **Note**
>
> Wenn Sie auf der Detailseite eines von Ihnen erstellten Objekts einen sicheren Tunnel erstellen, können Sie wählen, ob Sie einen neuen Tunnel erstellen oder einen vorhandenen verwenden möchten. Weitere Informationen finden Sie unter Öffnen Sie einen Tunnel und verwenden Sie browserbasiertes SSH, um auf das Remote-Gerät zuzugreifen.

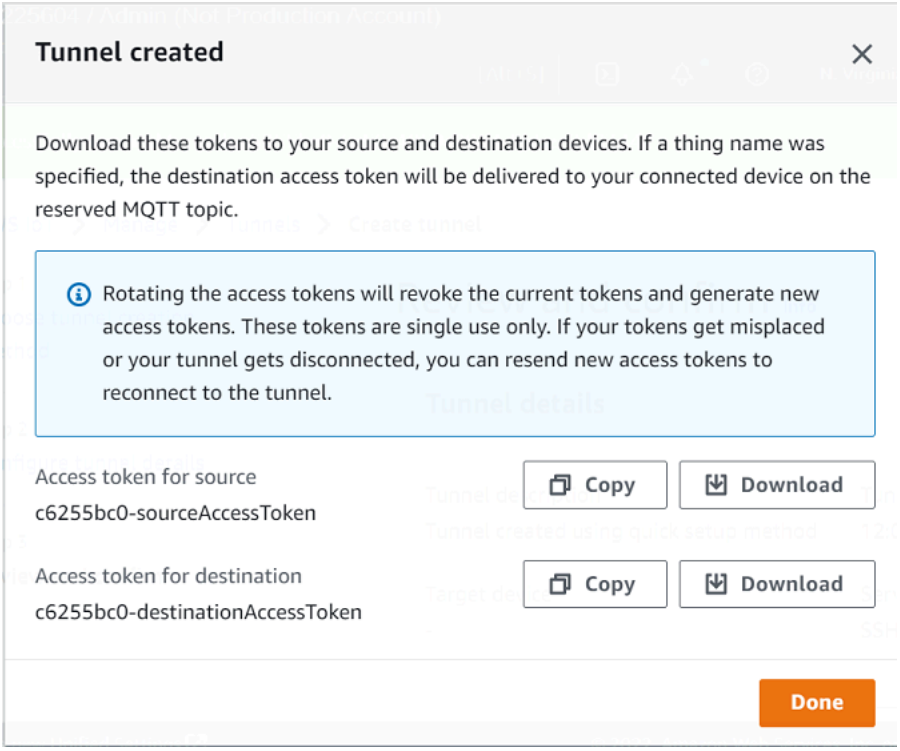


3. Überprüfen und bestätigen Sie die Details der Tunnelkonfiguration. Um einen Tunnel zu erstellen, wählen Sie Bestätigen und erstellen. Wenn Sie die Details bearbeiten möchten, wählen Sie Zurück, um zur vorherigen Seite zurückzukehren. Bestätigen Sie dann und erstellen Sie den Tunnel.

   > **Note**
   >
   > Wenn Sie die Quick Setup Methode verwenden, kann der Dienstname nicht bearbeitet werden. Sie müssen SSH als Dienst verwenden.

4. Um den Tunnel zu erstellen, wählen Sie Fertig.

   Für dieses Tutorial müssen Sie die Quell- oder Zielzugriffstoken nicht herunterladen. Diese Token können nur einmal verwendet werden, um eine Verbindung zum Tunnel herzustellen. Wenn Ihr Tunnel unterbrochen wird, können Sie neue Token generieren und an Ihr Remote-Gerät senden, um die Verbindung zum Tunnel wiederherzustellen. Weitere Informationen finden Sie unter Tunnelzugriffstoken erneut senden.



So öffnen Sie einen Tunnel über die API

Um einen neuen Tunnel zu öffnen, können Sie die OpenTunnel API-Operation verwenden.

> **Note**
>
> Sie können einen Tunnel mit der Quick Setup Methode nur von der AWS IoT -Konsole aus erstellen. Wenn Sie die AWS IoT API-Referenz-API oder die verwenden AWS CLI, wird die manuelle Einrichtungsmethode verwendet. Sie können den vorhandenen Tunnel, den Sie erstellt haben, öffnen und dann die Einrichtungsmethode des Tunnels ändern, um die Quick Setup Methode zu verwenden. Weitere Informationen finden Sie unter Öffnen eines vorhandenen Tunnels und Verwenden von browserbasiertem SSH.

Im Folgenden wird ein Beispiel gezeigt, wie dieser API-Vorgang ausgeführt wird. Wenn Sie optional den Objektnamen und den Zieldienst angeben möchten, verwenden Sie den `DestinationConfig`-Parameter. Ein Beispiel, das zeigt, wie dieser Parameter verwendet wird, finden Sie unterÖffnen Sie einen neuen Tunnel für das Remote-Gerät.

```
aws iotsecuretunneling open-tunnel
```

Wenn Sie diesen Befehl ausführen, wird ein neuer Tunnel erstellt und Sie erhalten die Quell- und Zielzugriffstoken.

```
{
    "tunnelId": "01234567-89ab-0123-4c56-789a01234bcd",
    "tunnelArn": "arn:aws:iot:us-
east-1:123456789012:tunnel/01234567-89ab-0123-4c56-789a01234bcd",
    "sourceAccessToken": "<SOURCE_ACCESS_TOKEN>",
    "destinationAccessToken": "<DESTINATION_ACCESS_TOKEN>"
}
```

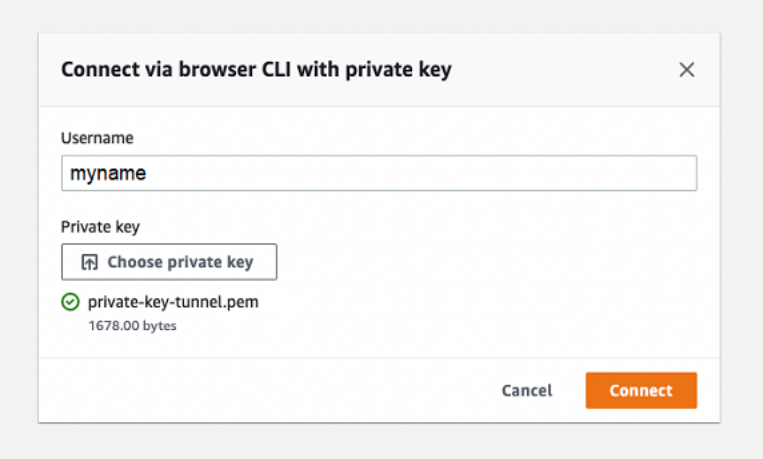
Verwenden von browserbasiertem SSH

Nachdem Sie mit der Quick Setup Methode einen Tunnel erstellt haben und Ihr Zielgerät eine Verbindung zum Tunnel hergestellt hat, können Sie über ein browserbasiertes SSH auf das Remote-Gerät zugreifen. Mithilfe des browserbasierten SSH können Sie direkt mit dem Remote-Gerät kommunizieren, indem Sie Befehle in eine kontextinterne Befehlszeilen-Schnittstelle in der Konsole eingeben. Diese Funktion erleichtert Ihnen die Interaktion mit dem Remote-Gerät, da Sie kein Terminal außerhalb der Konsole öffnen oder den lokalen Proxy konfigurieren müssen.

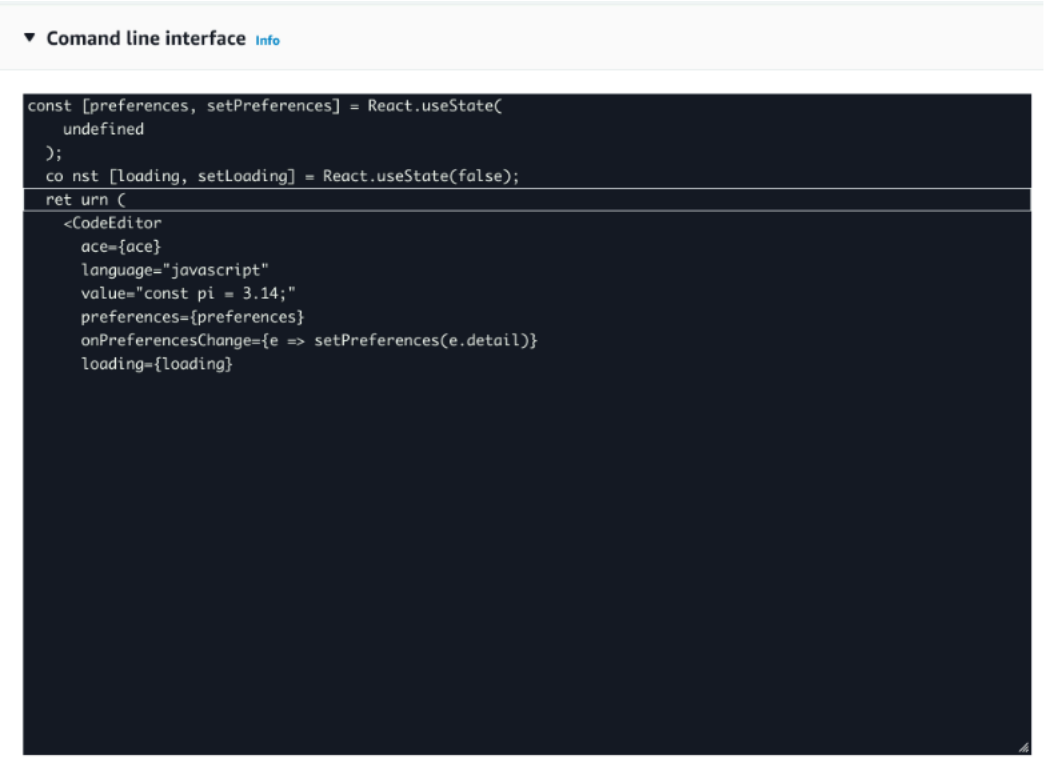
Browserbasiertes SSH verwenden

1. Gehen Sie zum Tunnel-Hub der AWS IoT -Konsole und wählen Sie den von Ihnen erstellten Tunnel aus, um seine Details anzuzeigen.
2. Erweitern Sie den Abschnitt Secure Shell (SSH) und wählen Sie dann Verbinden.
3. Wählen Sie aus, ob Sie sich bei der SSH-Verbindung authentifizieren möchten, indem Sie Ihren Benutzernamen und Ihr Passwort angeben, oder ob Sie für eine sicherere Authentifizierung den privaten Schlüssel Ihres Geräts verwenden können. Wenn Sie sich mit dem privaten Schlüssel authentifizieren, beachten Sie, dass nur PEM-formatierte (256 und 512) RSA-Schlüssel mit der SSH-Konsole funktionieren. AWS IoT Secure Tunneling

   - Um mit Ihrem Benutzernamen und Passwort eine Verbindung herzustellen, wählen Sie Passwort verwenden. Sie können dann Ihren Benutzernamen und Ihr Passwort eingeben und das In-Browser-CLI verwenden.
   - Um mit dem privaten Schlüssel Ihres Zielgeräts eine Verbindung herzustellen, wählen Sie Privaten Schlüssel verwenden aus. Geben Sie Ihren Benutzernamen an, laden Sie die private Schlüsseldatei des Geräts hoch und wählen Sie dann Verbinden, um mit der Verwendung des CLI im Browser zu beginnen.



Nachdem Sie sich bei der SSH-Verbindung authentifiziert haben, können Sie schnell mit der Eingabe von Befehlen beginnen und über das Browser-CLI mit dem Gerät interagieren, da der lokale Proxy bereits für Sie konfiguriert wurde.



Wenn die Browser-CLI nach Ablauf der Tunneldauer geöffnet bleibt, kann es zu einem Timeout kommen, wodurch die Befehlszeilen-Schnittstelle unterbrochen wird. Sie können den Tunnel duplizieren und eine weitere Sitzung starten, um mit dem Remote-Gerät in der Konsole selbst zu interagieren.

Behebung von Problemen bei der Verwendung des browserbasierten SSH

Im Folgenden wird gezeigt, wie Sie einige Probleme beheben können, auf die Sie bei der Verwendung von browserbasiertem SSH stoßen könnten.

- Anstelle der Befehlszeilen-Schnittstelle wird ein Fehler angezeigt.

  Möglicherweise wird der Fehler angezeigt, weil die Verbindung zu Ihrem Zielgerät unterbrochen wurde. Sie können Neue Zugriffstoken generieren wählen, um neue Zugriffstoken zu generieren und die Token mithilfe von MQTT an Ihr Remote-Gerät zu senden. Die neuen Token können verwendet werden, um die Verbindung zum Tunnel wiederherzustellen. Beim erneuten Herstellen der Verbindung mit dem Tunnel wird der Verlauf gelöscht und die Befehlszeilensitzung aktualisiert.
- Bei der Authentifizierung mit einem privaten Schlüssel wird ein Fehler angezeigt, dass die Verbindung zum Tunnel unterbrochen wurde

Möglicherweise wird der Fehler angezeigt, weil Ihr privater Schlüssel vom Zielgerät vielleicht nicht akzeptiert wurde. Um diesen Fehler zu beheben, überprüfen Sie die Datei mit dem privaten Schlüssel, die Sie zur Authentifizierung hochgeladen haben. Wenn Sie immer noch einen Fehler sehen, überprüfen Sie Ihre Geräteprotokolle. Sie können auch versuchen, erneut eine Verbindung zum Tunnel herzustellen, indem Sie neue Zugriffstoken an Ihr Remote-Gerät senden.

- Ihr Tunnel wurde geschlossen, als Sie die Sitzung verwendet haben

  Wenn Ihr Tunnel geschlossen wurde, weil er länger als die angegebene Dauer geöffnet blieb, wird Ihre Befehlszeilensitzung möglicherweise unterbrochen. Ein Tunnel kann nicht wieder geöffnet werden, wenn er einmal geschlossen wurde. Um die Verbindung wieder herzustellen, müssen Sie einen weiteren Tunnel zum Gerät öffnen.

  Sie können einen Tunnel duplizieren, um einen neuen Tunnel mit denselben Konfigurationen wie der geschlossene Tunnel zu erstellen. Sie können einen geschlossenen Tunnel von der Konsole aus duplizieren. AWS IoT Um den Tunnel zu duplizieren, wählen Sie den Tunnel aus, der geschlossen wurde, um seine Details anzuzeigen, und wählen Sie dann Tunnel duplizieren aus. Geben Sie die Tunneldauer an, die Sie verwenden möchten, und erstellen Sie dann den neuen Tunnel.

Bereinigen

- Tunnel schließen

  Wir empfehlen, den Tunnel zu schließen, wenn Sie ihn nicht mehr verwenden. Ein Tunnel kann auch geschlossen werden, wenn er länger als die angegebene Tunneldauer geöffnet blieb. Ein Tunnel kann nicht wieder geöffnet werden, wenn er einmal geschlossen wurde. Sie können einen Tunnel trotzdem duplizieren, indem Sie den geschlossenen Tunnel und dann Tunnel duplizieren auswählen. Geben Sie die Tunneldauer an, die Sie verwenden möchten, und erstellen Sie dann den neuen Tunnel.

  - Um einen einzelnen Tunnel oder mehrere Tunnel von der AWS IoT -Konsole aus zu schließen, wechseln Sie zum Tunnel-Hub, wählen Sie die Tunnel aus, die Sie schließen möchten, und wählen Sie dann Tunnel schließen aus.
  - Verwenden Sie die API, um einen einzelnen Tunnel oder mehrere Tunnel mithilfe der AWS IoT CloseTunnel API-Referenz-API zu schließen.

    ```
    aws iotsecuretunneling close-tunnel \
    ```

```
    --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
```

- Löschen eines Tunnels

  Sie können einen Tunnel dauerhaft aus Ihrem löschen AWS-Konto.

  > **Warning**
  >
  > Dieser Löschvorgang ist dauerhaft und kann nicht rückgängig gemacht werden.

  - Um einen einzelnen Tunnel oder mehrere Tunnel von der AWS IoT -Konsole aus zu löschen, wechseln Sie zum Tunnel-Hub, wählen Sie die Tunnel aus, die Sie löschen möchten, und wählen Sie dann Tunnel löschen aus.
  - Verwenden Sie die API, um einen einzelnen Tunnel oder mehrere Tunnel mithilfe der AWS IoT CloseTunnel API-Referenz-API zu löschen. Wenn Sie die API verwenden, setzen Sie das `delete`-Flag auf `true`.

    ```
    aws iotsecuretunneling close-tunnel \
        --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
        --delete true
    ```

Öffnen Sie mithilfe der manuellen Einrichtung einen Tunnel und stellen Sie eine Verbindung zum Remote-Gerät her

Wenn Sie einen Tunnel öffnen, können Sie die Quick Setup Methode oder die manuelle Einrichtungsmethode wählen, um einen Tunnel zum Remote-Gerät zu öffnen. Dieses Tutorial zeigt, wie Sie einen Tunnel mithilfe der manuellen Einrichtungsmethode öffnen und den lokalen Proxy konfigurieren und starten, um eine Verbindung zum Remote-Gerät herzustellen.

Wenn Sie die manuelle Einrichtungsmethode verwenden, müssen Sie die Tunnelkonfigurationen bei der Erstellung des Tunnels manuell angeben. Nachdem Sie den Tunnel erstellt haben, können Sie SSH im Browser verwenden oder ein Terminal außerhalb der AWS IoT Konsole öffnen. Dieses Tutorial zeigt, wie Sie das Terminal außerhalb der Konsole verwenden, um auf das Remote-Gerät zuzugreifen. Sie erfahren auch, wie Sie den lokalen Proxy konfigurieren und dann eine Verbindung zum lokalen Proxy herstellen, um mit dem Remote-Gerät zu interagieren. Um eine Verbindung zum lokalen Proxy herzustellen, müssen Sie beim Erstellen des Tunnels das Quellzugriffstoken herunterladen.

Sie können diese Methode verwenden, um über andere Dienste als SSH eine Verbindung zum Remote-Gerät herzustellen. Weitere Informationen zu den verschiedenen Einrichtungsmethoden finden Sie unter Methoden zur Tunneleinrichtung.

Voraussetzungen für die manuelle Einrichtungsmethode

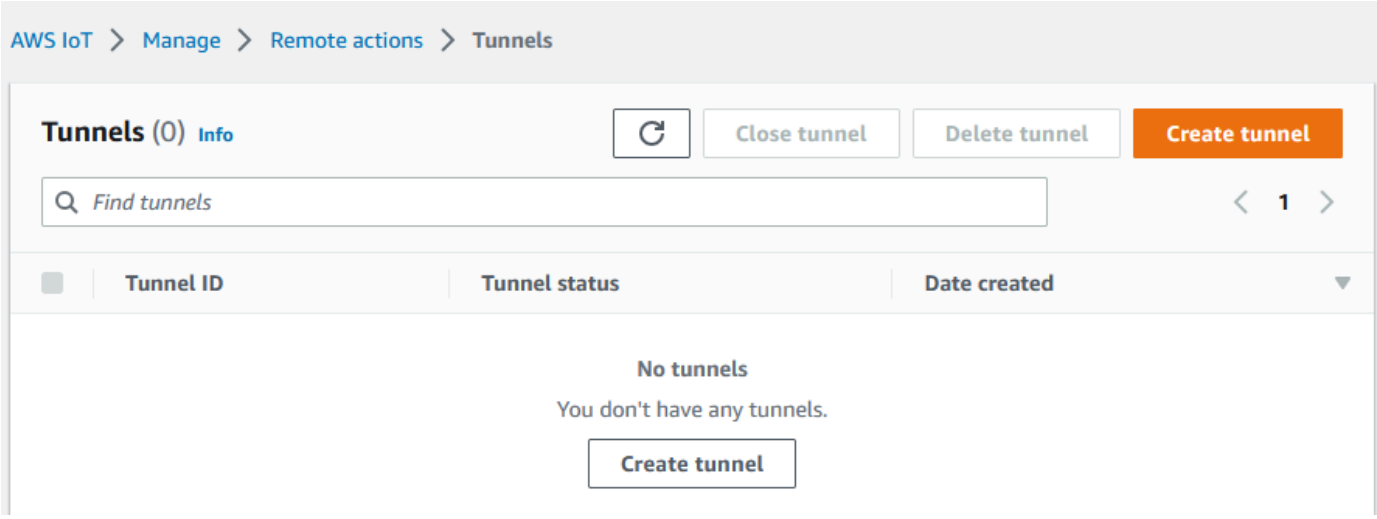
- Die Firewalls, hinter denen sich das Remote-Gerät befindet, müssen ausgehenden Datenverkehr an Port 443 zulassen. Der Tunnel, den Sie erstellen, verwendet diesen Port, um eine Verbindung zum Remote-Gerät herzustellen.
- Auf dem Remote-Gerät läuft ein IoT-Geräteagent (siehe IoT-Agent-Snippet), der eine Verbindung zum AWS IoT Geräte-Gateway herstellt und mit einem MQTT-Thema-Abonnement konfiguriert ist. Weitere Informationen finden Sie unter Ein Gerät mit dem AWS IoT Geräte-Gateway https://docs.aws.amazon.com/iot/latest/developerguide/sdk-tutorials.html verbinden.
- Auf dem Remote-Gerät muss ein SSH-Daemon ausgeführt werden.
- Sie haben den lokalen Proxy-Quellcode von heruntergeladen GitHub und für die Plattform Ihrer Wahl erstellt. In diesem Tutorial verweisen wir auf die erstellte lokale ausführbare Proxy-Datei als `localproxy`.

Tunnel öffnen

Sie können einen sicheren Tunnel mit der AWS-Managementkonsole, der AWS IoT API-Referenz oder der öffnen AWS CLI. Sie können optional einen Zielnamen konfigurieren, der für dieses Tutorial jedoch nicht erforderlich ist. Wenn Sie das Ziel konfigurieren, übermittelt Secure Tunneling das Zugriffstoken automatisch mithilfe von MQTT an das Remote-Gerät. Weitere Informationen finden Sie unter Methoden zur Tunnelerstellung in AWS IoT Konsole.

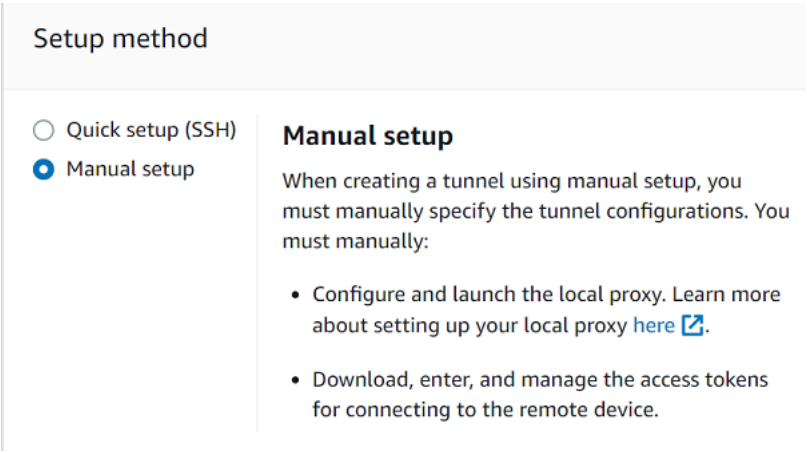
Um einen Tunnel mit der Konsole zu öffnen,

1. Gehen Sie zum Tunnel-Hub der AWS IoT -Konsole und wählen Sie Tunnel erstellen.



2. Wählen Sie für dieses Tutorial Manuelle Einrichtungsmethode zur Erstellung des Tunnels und wählen Sie dann Weiter. Informationen zur Verwendung der Quick Setup Methode zum Erstellen eines Tunnels finden Sie unter Öffnen Sie einen Tunnel und verwenden Sie browserbasiertes SSH, um auf das Remote-Gerät zuzugreifen.

> **Note**
>
> Wenn Sie auf der Detailseite eines von Ihnen erstellten Objekts einen sicheren Tunnel erstellen, können Sie wählen, ob Sie einen neuen Tunnel erstellen oder einen vorhandenen verwenden möchten. Weitere Informationen finden Sie unter Öffnen Sie einen Tunnel und verwenden Sie browserbasiertes SSH, um auf das Remote-Gerät zuzugreifen.



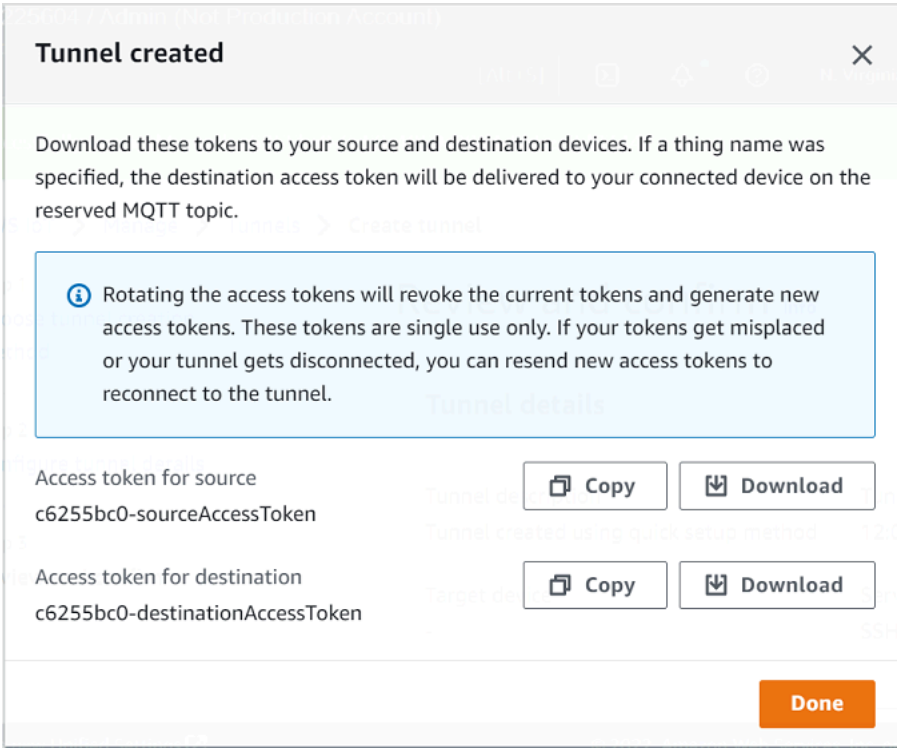
3. (Optional) Geben Sie die Konfigurationseinstellungen für Ihren Tunnel ein. Sie können diesen Schritt auch überspringen und mit dem nächsten Schritt fortfahren, um einen Tunnel zu erstellen.

   Geben Sie eine Tunnelbeschreibung, eine Dauer des Tunnel-Timeouts und Ressourcen-Tags als Schlüssel-Wert-Paare ein, um Ihre Ressource leichter identifizieren zu können. Für dieses Tutorial können Sie die Zielkonfiguration überspringen.

   > **Note**
   >
   > Für die Dauer der Offenhaltung eines Tunnels werden Ihnen keine Gebühren in Rechnung gestellt. Gebühren fallen nur an, wenn Sie einen neuen Tunnel erstellen. Preisinformationen finden Sie im Abschnitt Secure Tunneling unter AWS IoT Device Management -Preise.

4. Laden Sie die Client-Zugriffstoken herunter und wählen Sie dann Fertig aus. Die Token können nicht heruntergeladen werden, nachdem Sie Fertig ausgewählt haben.

   Diese Token können nur einmal verwendet werden, um eine Verbindung zum Tunnel herzustellen. Wenn Sie die Token verlegen oder die Verbindung zum Tunnel unterbrochen wird, können Sie neue Token erzeugen und an Ihr Remote-Gerät senden, um die Verbindung zum Tunnel wiederherzustellen.



So öffnen Sie einen Tunnel über die API

Um einen neuen Tunnel zu öffnen, können Sie die OpenTunnel API-Operation verwenden. Sie können mithilfe der API auch zusätzliche Konfigurationen angeben, z. B. die Tunneldauer und die Zielkonfiguration.

```
aws iotsecuretunneling open-tunnel \
    --region us-east-1 \
    --endpoint https://api.us-east-1.tunneling.iot.amazonaws.com
```

Wenn Sie diesen Befehl ausführen, wird ein neuer Tunnel erstellt und Sie erhalten die Quell- und Zielzugriffstoken.

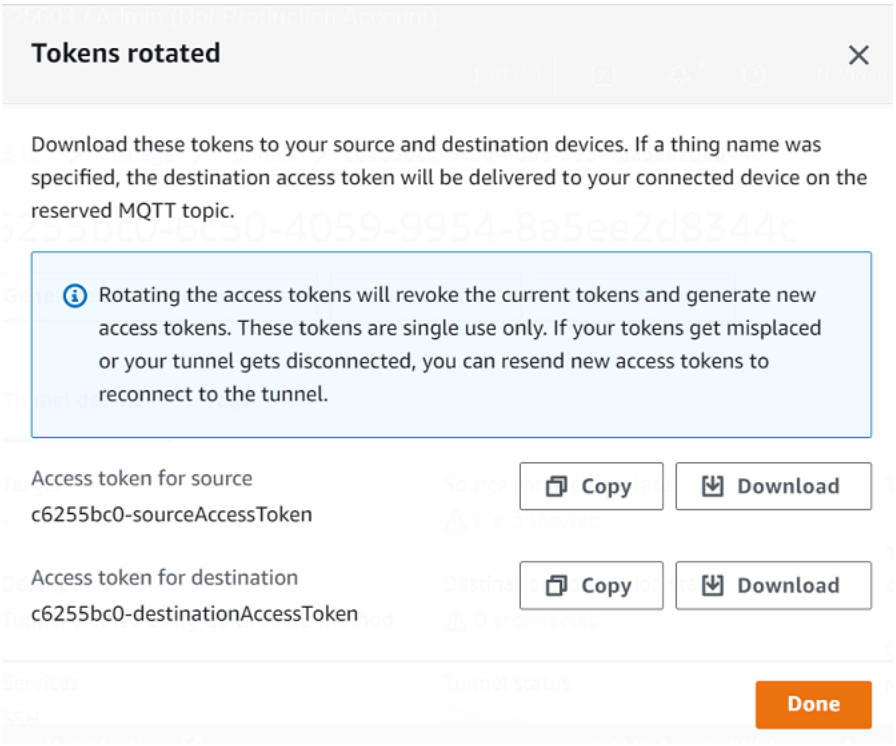
```
{
    "tunnelId": "01234567-89ab-0123-4c56-789a01234bcd",
    "tunnelArn": "arn:aws:iot:us-
east-1:123456789012:tunnel/01234567-89ab-0123-4c56-789a01234bcd",
    "sourceAccessToken": "<SOURCE_ACCESS_TOKEN>",
    "destinationAccessToken": "<DESTINATION_ACCESS_TOKEN>"
}
```

Tunnelzugriffstoken erneut senden

Die Token, die Sie beim Erstellen eines Tunnels erhalten haben, können nur einmal verwendet werden, um eine Verbindung zum Tunnel herzustellen. Wenn Sie das Zugriffstoken verlegen oder der Tunnel unterbrochen wird, können Sie mithilfe von MQTT ohne zusätzliche Kosten erneut neue Zugriffstoken an das Remote-Gerät senden. AWS IoT Secure Tunneling widerruft die aktuellen Token und gibt neue Zugriffstoken für die erneute Verbindung zum Tunnel zurück.

Um die Token von der Konsole aus zu rotieren,

1. Gehen Sie zum Tunnels-Hub der AWS IoT Konsole und wählen Sie den Tunnel aus, den Sie erstellt haben.
2. Wählen Sie auf der Seite mit den Tunneldetails die Option Neue Zugriffstoken generieren und dann Weiter aus.
3. Laden Sie die neuen Zugriffstoken für Ihren Tunnel herunter und wählen Sie Fertig. Diese Token können nur einmal verwendet werden. Wenn Sie die Token verlegen oder der Tunnel unterbrochen wird, können Sie neue Zugriffstoken senden, um die Verbindung zum Tunnel wiederherzustellen.



So rotieren Sie Zugriffstokens über die API

Um die Tunnelzugriffstoken zu rotieren, können Sie den RotateTunnelAccessToken API-Vorgang verwenden, um die aktuellen Token zu widerrufen und neue Zugriffstoken für die Wiederverbindung mit dem Tunnel zurückzugeben. Mit dem folgenden Befehl werden beispielsweise die Zugriffstoken für das Zielgerät rotiert, *RemoteThing1*.

```
aws iotsecuretunneling rotate-tunnel-access-token \
    --tunnel-id <tunnel-id> \
    --client-mode DESTINATION \
    --destination-config thingName=<RemoteThing1>,services=SSH \
    --region <region>
```

Wenn Sie diesen Befehl ausführen, wird das neue Zugriffstoken generiert, siehe folgendes Beispiel. Das Token wird dann über MQTT an das Gerät übermittelt, um eine Verbindung zum Tunnel herzustellen, sofern der Geräteagent korrekt eingerichtet ist.

```
{
    "destinationAccessToken": "destination-access-token",
    "tunnelArn": "arn:aws:iot:region:account-id:tunnel/tunnel-id"
}
```

Beispiele, die zeigen, wie und wann die Zugriffstoken rotiert werden müssen, finden Sie unter Auflösen AWS IoT Probleme mit der Tunneling-Konnektivität durch rotierendes Rotieren der Client-Zugriffstoken.

Konfigurieren und Starten des lokalen Proxys

Um eine Verbindung zum Remote-Gerät herzustellen, öffnen Sie ein Terminal auf Ihrem Laptop und konfigurieren und starten Sie den lokalen Proxy. Der lokale Proxy überträgt Daten, die von der auf dem Quellgerät laufenden Anwendung gesendet wurden, mithilfe von sicherem Tunneling über eine sichere Verbindung. WebSocket Sie können die lokale Proxyquelle von herunterladen. GitHub

Öffnen Sie ein Terminal auf Ihrem Laptop, kopieren Sie das Zugriffstoken für den Quell-Client und starten Sie den lokalen Proxy im Quellmodus. Im Folgenden finden Sie einen Beispielbefehl zum Starten des lokalen Proxys. Im folgenden Befehl ist der lokale Proxy so konfiguriert, dass er auf neue Verbindungen an Port 5555 wartet. In diesem Befehl gilt Folgendes:

- `-r`gibt die AWS-Region an. Dabei muss es sich um dieselbe Region handeln, in der Ihr Tunnel erstellt wurde.
- `-s` gibt den Port an, zu dem der Proxy eine Verbindung herstellen soll.
- `-t` gibt den Client-Token-Text an.

```
./localproxy -r us-east-1 -s 5555 -t source-client-access-token
```

Wenn Sie diesen Befehl ausführen, wird der lokale Proxy im Quellmodus gestartet. Wenn Sie nach der Ausführung des Befehls die folgende Fehlermeldung erhalten, richten Sie den CA-Pfad ein. Weitere Informationen finden Sie unter Lokaler Proxy für sicheres Tunneling auf. GitHub

```
Could not perform SSL handshake with proxy server: certificate verify failed
```

Im Folgenden finden Sie eine Beispielausgabe für die Ausführung des lokalen Proxys im `source`-Modus.

```
...
...

Starting proxy in source mode
```

```
Attempting to establish web socket connection with endpoint wss://
data.tunneling.iot.us-east-1.amazonaws.com:443
Resolved proxy  server IP: 10.10.0.11
Connected successfully with proxy server
Performing SSL handshake with proxy server
Successfully completed SSL handshake with proxy server
HTTP/1.1 101 Switching Protocols

...

Connection: upgrade
channel-id: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
upgrade: websocket

...

Web socket session ID: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
Web socket subprotocol selected: aws.iot.securetunneling-2.0
Successfully established websocket connection with proxy server: wss://
data.tunneling.iot.us-east-1.amazonaws.com:443
Setting up web socket pings for every 5000 milliseconds
Scheduled next read:

...

Starting web socket read loop continue reading...
Resolved bind IP: 127.0.0.1
Listening for new connection on port 5555
```

Starten einer SSH-Sitzung

Öffnen Sie ein anderes Terminal und verwenden Sie den folgenden Befehl, um eine neue SSH-Sitzung zu starten, indem Sie eine Verbindung zum lokalen Proxy auf Port 5555 herstellen.

```
ssh username@localhost -p 5555
```

Möglicherweise werden Sie zur Eingabe eines Passworts für die SSH-Sitzung aufgefordert. Wenn Sie mit der SSH-Sitzung fertig sind, geben Sie **`exit`** ein, um die Sitzung zu schließen.

Bereinigen

- Tunnel schließen

Wir empfehlen, den Tunnel zu schließen, wenn Sie ihn nicht mehr verwenden. Ein Tunnel kann auch geschlossen werden, wenn er länger als die angegebene Tunneldauer geöffnet blieb. Ein Tunnel kann nicht wieder geöffnet werden, wenn er einmal geschlossen wurde. Sie können einen Tunnel trotzdem duplizieren, indem Sie den geschlossenen Tunnel öffnen und dann Tunnel duplizieren auswählen. Geben Sie die Tunneldauer an, die Sie verwenden möchten, und erstellen Sie dann den neuen Tunnel.

- Um einen einzelnen Tunnel oder mehrere Tunnel von der AWS IoT -Konsole aus zu schließen, wechseln Sie zum Tunnel-Hub, wählen Sie die Tunnel aus, die Sie schließen möchten, und wählen Sie dann Tunnel schließen aus.
- Verwenden Sie den API-Vorgang, um einen einzelnen Tunnel oder mehrere Tunnel mithilfe der AWS IoT API-Referenz-API zu schließen. CloseTunnel

```
aws iotsecuretunneling close-tunnel \
    --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
```

- Löschen eines Tunnels

  Sie können einen Tunnel dauerhaft aus Ihrem löschen AWS-Konto.

  > **Warning**
  >
  > Dieser Löschvorgang ist dauerhaft und kann nicht rückgängig gemacht werden.

  - Um einen einzelnen Tunnel oder mehrere Tunnel von der AWS IoT -Konsole aus zu löschen, wechseln Sie zum Tunnel-Hub, wählen Sie die Tunnel aus, die Sie löschen möchten, und wählen Sie dann Tunnel löschen aus.
  - Verwenden Sie den API-Vorgang, um einen einzelnen Tunnel oder mehrere Tunnel mithilfe der AWS IoT CloseTunnel API-Referenz-API zu löschen. Wenn Sie die API verwenden, setzen Sie das `delete`-Flag auf `true`.

```
aws iotsecuretunneling close-tunnel \
    --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
    --delete true
```

Öffnen Sie einen Tunnel und verwenden Sie browserbasiertes SSH, um auf das Remote-Gerät zuzugreifen

Von der AWS IoT Konsole aus können Sie einen Tunnel entweder vom Tunnel-Hub aus oder von der Detailseite eines von Ihnen erstellten IoT-Dings aus erstellen. Wenn Sie einen Tunnel vom Tunnel-Hub aus erstellen, können Sie mithilfe von Quick Setup oder der manuellen Einrichtung angeben, ob Sie einen Tunnel erstellen möchten. Ein Tutorial finden Sie unter Öffnen Sie einen Tunnel und starten Sie eine SSH-Sitzung zum Remote-Gerät.

Wenn Sie einen Tunnel von der Ding-Detailseite der AWS IoT Konsole aus erstellen, können Sie auch angeben, ob ein neuer Tunnel erstellt oder ein vorhandener Tunnel für dieses Ding geöffnet werden soll, wie in diesem Tutorial dargestellt. Wenn Sie einen vorhandenen Tunnel auswählen, können Sie auf den neuesten, offenen Tunnel zugreifen, den Sie für dieses Gerät erstellt haben. Sie können dann die Befehlszeilen-Schnittstelle im Terminal verwenden, um eine SSH-Verbindung zum Gerät herzustellen.

Voraussetzungen

- Die Firewalls, hinter denen sich das Remote-Gerät befindet, müssen ausgehenden Datenverkehr an Port 443 zulassen. Der Tunnel, den Sie erstellen, verwendet diesen Port, um eine Verbindung zum Remote-Gerät herzustellen.
- Sie haben (zum Beispiel`RemoteDevice1`) ein IoT-Ding in der AWS IoT Registrierung erstellt. Dieses Objekt entspricht der Darstellung Ihres Remote-Geräts in der Cloud. Weitere Informationen finden Sie unter Registrierung eines Geräts in der AWS IoT -Registry.
- Auf dem Remote-Gerät läuft ein IoT-Geräteagent (siehe IoT-Agent-Snippet), der eine Verbindung zum Geräte-Gateway herstellt und mit einem MQTT-Themenabonnement konfiguriert ist. AWS IoT Weitere Informationen finden Sie unter Ein Gerät mit dem AWS IoT Geräte-Gateway https://docs.aws.amazon.com/iot/latest/developerguide/sdk-tutorials.html verbinden.
- Auf dem Remote-Gerät muss ein SSH-Daemon ausgeführt werden.

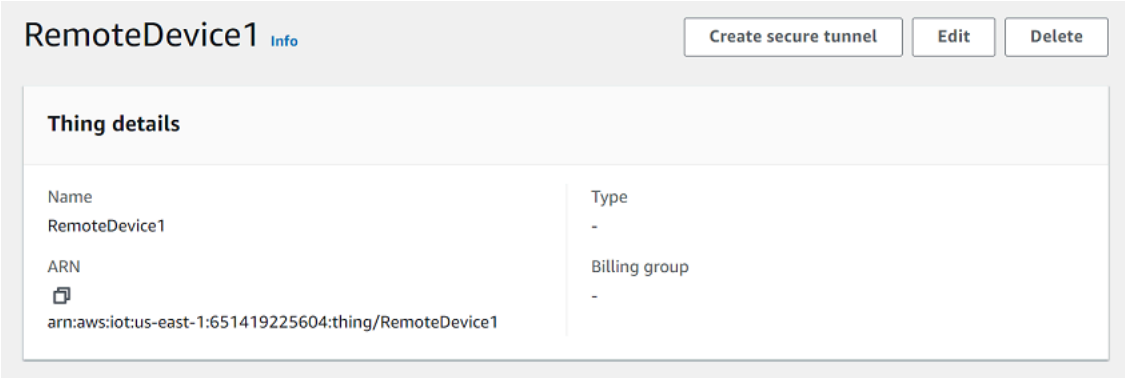
Öffnen Sie einen neuen Tunnel für das Remote-Gerät

Angenommen, Sie möchten einen Tunnel zu Ihrem Remote-Gerät `RemoteDevice1` öffnen. Erstellen Sie zunächst ein IoT-Objekt mit dem Namen `RemoteDevice1` in der AWS IoT -Registry. Sie können dann einen Tunnel mithilfe der AWS-Managementkonsole, der AWS IoT API Reference API oder der erstellen AWS CLI.

Wenn Sie das Ziel beim Erstellen des Tunnels konfigurieren, liefert der Secure Tunneling-Service das Zugriffstoken für den Zielclient über MQTT an das Remote-Gerät und das reservierte MQTT-Thema (`$aws/things/RemoteDeviceA/tunnels/notify`). Weitere Informationen finden Sie unter Methoden zur Tunnelerstellung in AWS IoT Konsole.

Um von der Konsole aus einen Tunnel für ein Remote-Gerät zu erstellen,

1. Wählen Sie das Objekt `RemoteDevice1` aus, um seine Details anzuzeigen, und wählen Sie dann Sicheren Tunnel erstellen aus.



2. Wählen Sie aus, ob Sie einen neuen Tunnel erstellen oder einen vorhandenen Tunnel öffnen möchten. Um einen neuen Tunnel zu erstellen, wählen Sie Neuen Tunnel erstellen. Sie können die Quick Setup Methode oder die manuelle Einrichtungsmethode verwenden, um einen Tunnel zu erstellen. Weitere Informationen erhalten Sie unter Öffnen Sie mithilfe der manuellen Einrichtung einen Tunnel und stellen Sie eine Verbindung zum Remote-Gerät her und Öffnen Sie einen Tunnel und verwenden Sie browserbasiertes SSH, um auf das Remote-Gerät zuzugreifen.

So erstellen Sie einen Tunnel für ein Remote-Gerät mithilfe der API

Um einen neuen Tunnel zu öffnen, können Sie die OpenTunnel API-Operation verwenden. Der folgende Code zeigt ein Beispiel für die Ausführung dieses Befehls.

```
aws iotsecuretunneling open-tunnel \
    --region us-east-1 \
    --endpoint https://api.us-east-1.tunneling.iot.amazonaws.com
    --cli-input-json file://input.json
```

Im Folgenden werden die Inhalte der `input.json`-Datei angezeigt. Sie können den `destinationConfig`-Parameter verwenden, um den Namen des Zielgeräts (z. B. *`RemoteDevice1`*) und den Dienst anzugeben, den Sie für den Zugriff auf das Zielgerät verwenden

möchten, z. B. *SSH*. Optional können Sie auch zusätzliche Parameter wie die Tunnelbeschreibung und Tags angeben.

Inhalt von input.json

```
{
   "description": "Tunnel to remote device1",
   "destinationConfig": {
      "services": [ "SSH" ],
      "thingName": "RemoteDevice1"
   }
}
```

Wenn Sie diesen Befehl ausführen, wird ein neuer Tunnel erstellt und Sie erhalten die Quell- und Zielzugriffstoken.

```
{
    "tunnelId": "01234567-89ab-0123-4c56-789a01234bcd",
    "tunnelArn": "arn:aws:iot:us-
east-1:123456789012:tunnel/01234567-89ab-0123-4c56-789a01234bcd",
    "sourceAccessToken": "<SOURCE_ACCESS_TOKEN>",
    "destinationAccessToken": "<DESTINATION_ACCESS_TOKEN>"
}
```

Öffnen eines vorhandenen Tunnels und Verwenden von browserbasiertem SSH

Angenommen, `RemoteDevice1` Sie haben den Tunnel für Ihr Remote-Gerät mithilfe der manuellen Einrichtungsmethode oder mithilfe der AWS IoT API-Referenz-API erstellt. Sie können dann den vorhandenen Tunnel für das Gerät öffnen und dieQuick Setup Methode wählen, um die browserbasierte SSH-Funktion zu verwenden. Die Konfigurationen eines vorhandenen Tunnels können nicht bearbeitet werden, sodass Sie die manuelle Einrichtungsmethode nicht verwenden können.

Um die browserbasierte SSH-Funktion zu verwenden, müssen Sie weder das Quellzugriffstoken herunterladen noch den lokalen Proxy konfigurieren. Ein webbasierter lokaler Proxy wird automatisch für Sie konfiguriert, sodass Sie mit der Interaktion mit Ihrem Remote-Gerät beginnen können.

So verwenden Sie die Quick Setup Methode und browserbasiertes SSH

1. Gehen Sie zur Detailseite des Objekts `RemoteDevice1`, das Sie erstellt haben, und klicken Sie auf Sicheren Tunnel erstellen.

2. Wählen Sie Bestehenden Tunnel verwenden, um den letzten offenen Tunnel zu öffnen, den Sie für das Remote-Gerät erstellt haben. Die Konfigurationen eines vorhandenen Tunnels können nicht bearbeitet werden, sodass Sie die manuelle Einrichtungsmethode nicht verwenden können. Um die Quick Setup Methode zu verwenden, wählen Sie Schnelle Einrichtung.
3. Überprüfen und bestätigen Sie die Details der Tunnelkonfiguration und erstellen Sie den Tunnel. Die Tunnelkonfigurationen können nicht bearbeitet werden.

   Wenn Sie den Tunnel erstellen, verwendet Secure Tunneling den RotateTunnelAccessToken API-Vorgang, um die ursprünglichen Zugriffstoken zu widerrufen und neue Zugriffstoken zu generieren. Wenn Ihr Remote-Gerät MQTT verwendet, werden diese Token automatisch zu dem MQTT-Thema, das es abonniert hat, an das Remote-Gerät gesendet. Sie können sich auch dafür entscheiden, diese Token manuell auf Ihr Quellgerät herunterzuladen.

Nachdem Sie den Tunnel erstellt haben, können Sie das browserbasierte SSH verwenden, um über die kontextinterne Befehlszeilen-Schnittstelle direkt von der Konsole aus mit dem Remote-Gerät zu interagieren. Um diese Befehlszeilenschnittstelle zu verwenden, wählen Sie den Tunnel für das Ding aus, das Sie erstellt haben, und erweitern Sie auf der Detailseite den Command-line Schnittstellenbereich. Da der lokale Proxy bereits für Sie konfiguriert wurde, können Sie mit der Eingabe von Befehlen beginnen, um schnell mit dem Zugriff auf und der Interaktion mit Ihrem Remote-Gerät `RemoteDevice1` zu beginnen.

Weitere Informationen zur Quick Setup Methode und zur Verwendung von browserbasiertem SSH finden Sie unter Öffnen Sie einen Tunnel und verwenden Sie browserbasiertes SSH, um auf das Remote-Gerät zuzugreifen.

Bereinigen

- Tunnel schließen

  Wir empfehlen, den Tunnel zu schließen, wenn Sie ihn nicht mehr verwenden. Ein Tunnel kann auch geschlossen werden, wenn er länger als die angegebene Tunneldauer geöffnet blieb. Ein Tunnel kann nicht wieder geöffnet werden, wenn er einmal geschlossen wurde. Sie können einen Tunnel trotzdem duplizieren, indem Sie den geschlossenen Tunnel öffnen und dann Tunnel duplizieren auswählen. Geben Sie die Tunneldauer an, die Sie verwenden möchten, und erstellen Sie dann den neuen Tunnel.

- Um einen einzelnen Tunnel oder mehrere Tunnel von der AWS IoT -Konsole aus zu schließen, wechseln Sie zum Tunnel-Hub, wählen Sie die Tunnel aus, die Sie schließen möchten, und wählen Sie dann Tunnel schließen aus.
- Verwenden Sie den API-Vorgang, um einen einzelnen Tunnel oder mehrere Tunnel mithilfe der AWS IoT CloseTunnel API-Referenz-API zu schließen.

```
aws iotsecuretunneling close-tunnel \
    --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
```

- Löschen eines Tunnels

  Sie können einen Tunnel dauerhaft aus Ihrem löschen AWS-Konto.

  > **Warning**
  >
  > Dieser Löschvorgang ist dauerhaft und kann nicht rückgängig gemacht werden.

  - Um einen einzelnen Tunnel oder mehrere Tunnel von der AWS IoT -Konsole aus zu löschen, wechseln Sie zum Tunnel-Hub, wählen Sie die Tunnel aus, die Sie löschen möchten, und wählen Sie dann Tunnel löschen aus.
  - Verwenden Sie den API-Vorgang, um einen einzelnen Tunnel oder mehrere Tunnel mithilfe der AWS IoT CloseTunnel API-Referenz-API zu löschen. Wenn Sie die API verwenden, setzen Sie das `delete`-Flag auf `true`.

```
aws iotsecuretunneling close-tunnel \
    --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
    --delete true
```

Lokaler Proxy

Der lokale Proxy überträgt Daten, die von der auf dem Quellgerät laufenden Anwendung gesendet wurden, mithilfe von sicherem Tunneling über eine WebSocket sichere Verbindung. Sie können die lokale Proxyquelle von herunterladen. GitHub

Der lokale Proxy kann in zwei Modi ausgeführt werden: `source` oder `destination`. Im Quellmodus wird der lokale Proxy auf demselben Gerät oder Netzwerk ausgeführt wie die Clientanwendung, die

die TCP-Verbindung initiiert. Im Zielmodus wird der lokale Proxy zusammen mit der Zielanwendung auf dem Remote-Gerät ausgeführt. Ein einzelner Tunnel kann mithilfe von Tunnelmultiplexing bis zu drei Datenströme gleichzeitig unterstützen. Für jeden Datenstrom werden beim Secure Tunneling mehrere TCP-Verbindungen verwendet, wodurch das Risiko eines Timeouts reduziert wird. Weitere Informationen finden Sie unter Multiplex-Datenströme und gleichzeitige Verwendung von TCP-Verbindungen in einem sicheren Tunnel.

Wie man den lokalen Proxy benutzt

Sie können den lokalen Proxy auf den Quell- und Zielgeräten ausführen, um Daten an die sicheren Tunneling-Endpunkte zu übertragen. Wenn sich Ihre Geräte in einem Netzwerk befinden, das einen Web-Proxy verwendet, kann der Web-Proxy die Verbindungen abfangen, bevor er sie an das Internet weiterleitet. In diesem Fall müssen Sie Ihren lokalen Proxy so konfigurieren, dass er den Web-Proxy verwendet. Weitere Informationen finden Sie unter Konfigurieren Sie den lokalen Proxy für Geräte, die einen Web-Proxy verwenden.

Workflow des lokalen Proxy

Die folgenden Schritte zeigen, wie der lokale Proxy auf den Quell- und Zielgeräten ausgeführt wird.

1. Lokalen Proxy mit Secure Tunneling verbinden

   Der lokale Proxy stellt zunächst eine Verbindung zum Secure Tunneling her. Wenn Sie den lokalen Proxy starten, verwenden Sie die folgenden Argumente:

   - Das `-r` Argument zur Angabe, AWS-Region in dem der Tunnel geöffnet wird.
   - Verwenden Sie das `-t`-Argument, um entweder das vom `OpenTunnel` zurückgegebene Quell- oder Zielclient-Zugriffstoken zu übergeben.

   > **Note**
   >
   > Zwei lokale Proxys, die dasselbe Client-Zugriffstoken verwenden, können nicht gleichzeitig verbunden werden.

2. Quell- oder Zielaktionen ausführen

   Nachdem die WebSocket Verbindung hergestellt wurde, führt der lokale Proxy je nach Konfiguration entweder Aktionen im Quell- oder im Zielmodus durch.

Standardmäßig versucht der lokale Proxy, die Verbindung zum sicheren Tunneling wiederherzustellen, wenn irgendwelche input/output (I/O) -Fehler auftreten oder wenn die WebSocket Verbindung unerwartet geschlossen wird. Dadurch wird die TCP-Verbindung geschlossen. Wenn TCP-Socket-Fehler auftreten, sendet der lokale Proxy eine Nachricht über den Tunnel, um die andere Seite zu benachrichtigen, dass die TCP-Verbindung geschlossen wird. Standardmäßig verwendet der lokale Proxy immer die SSL-Kommunikation.

3. Lokalen Proxy stoppen

   Nachdem Sie den Tunnel verwendet haben, ist es sicher, den lokalen Proxyprozess zu stoppen. Wir empfehlen Ihnen, den Tunnel explizit durch Aufrufen von `CloseTunnel` zu schließen. Aktive Tunnelclients werden möglicherweise nicht direkt nach dem Anruf geschlossen. `CloseTunnel`

Weitere Informationen zur Verwendung des zum Öffnen eines Tunnels und AWS-Managementkonsole zum Starten einer SSH-Sitzung finden Sie unterÖffnen Sie einen Tunnel und starten Sie eine SSH-Sitzung zum Remote-Gerät.

Bewährte Methoden des lokalen Proxy

Befolgen Sie beim Ausführen des lokalen Proxy die folgenden bewährten Methoden:

- Vermeiden Sie beim Übergeben von Zugriffstoken die Verwendung des lokalen Proxy-Arguments `-t`. Es wird empfohlen, dass Sie die `AWSIOT_TUNNEL_ACCESS_TOKEN`-Umgebungsvariable verwenden, um das Zugriffstoken für den lokalen Proxy festzulegen.
- Führen Sie die ausführbare Datei des lokalen Proxys mit den geringsten Berechtigungen im Betriebssystem oder in der Umgebung aus.
  - Führen Sie den lokalen Proxy nicht als Administrator unter Windows aus.
  - Führen Sie den lokalen Proxy als Root unter Linux und macOS aus.
- Erwägen Sie, den lokalen Proxy auf separaten Hosts, Containern, Sandboxen, Chroot-Jail oder einer virtualisierten Umgebung auszuführen.
- Erstellen Sie den lokalen Proxy mit relevanten Sicherheits-Flags, abhängig von Ihrer Toolchain.
- Verwenden Sie auf Geräten mit mehreren Netzwerkschnittstellen das `-b`-Argument, um den TCP-Socket an die Netzwerkschnittstelle zu binden, die zur Kommunikation mit der Zielanwendung verwendet wird.

Beispielbefehl und Ausgabe

Hier ein Beispiel für einen Befehl, den Sie ausführen, und die entsprechende Ausgabe. Das Beispiel zeigt, wie der lokale Proxy sowohl im `source`- als auch im `destination`-Modus konfiguriert werden kann. Der lokale Proxy aktualisiert das HTTPS-Protokoll auf, WebSockets um eine langlebige Verbindung herzustellen, und beginnt dann mit der Übertragung von Daten über die Verbindung an die Endpunkte des sicheren Tunnelgeräts.

Bevor Sie diese Befehle ausführen:

Müssen Sie einen Tunnel geöffnet und die Client-Zugriffstoken für die Quelle und das Ziel abgerufen haben. Müssen Sie auch den lokalen Proxy wie zuvor beschrieben erstellt haben. Um den lokalen Proxy zu erstellen, öffnen Sie den lokalen Proxy-Quellcode im GitHub Repository und folgen Sie den Anweisungen zum Erstellen und Installieren des lokalen Proxys.

> **Note**
>
> Die folgenden Befehle, die in den Beispielen verwendet werden, verwenden das `verbosity`-Flag, um einen Überblick über die verschiedenen zuvor beschriebenen Schritte zu veranschaulichen, nachdem Sie den lokalen Proxy ausgeführt haben. Wir empfehlen, dass Sie das Flag nur für Tests verwenden.

Lokalen Proxy im Quellmodus betreiben

Die folgenden Befehle zeigen, wie der lokale Proxy im Quellmodus ausgeführt wird.

Linux/macOS

Führen Sie unter Linux oder macOS die folgenden Befehle im Terminal aus, um den lokalen Proxy auf Ihrer Quelle zu konfigurieren und zu starten.

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
./localproxy -s 5555 -v 5 -r us-west-2
```

Wobei Folgendes gilt:

- `-s` ist der Quell-Listen-Port, der den lokalen Proxy im Quellmodus startet.
- `-v` ist die Ausführlichkeit der Ausgabe, die ein Wert zwischen null und dechs sein kann.

- `-r` ist die Endpunktregion, in der der Tunnel geöffnet ist.

Weitere Informationen zu den Parametern finden Sie unter Mit Befehlszeilenargumenten festgelegte Optionen.

Windows

In Windows konfigurieren Sie den lokalen Proxy ähnlich wie für Linux oder macOS, aber die Art und Weise, wie Sie die Umgebungsvariablen definieren, unterscheidet sich von den anderen Plattformen. Führen Sie die folgenden Befehle im `cmd`-Fenster aus, um den lokalen Proxy auf Ihrer Quelle zu konfigurieren und zu starten.

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
.\localproxy -s 5555 -v 5 -r us-west-2
```

Wobei Folgendes gilt:

- `-s` ist der Quell-Listen-Port, der den lokalen Proxy im Quellmodus startet.
- `-v` ist die Ausführlichkeit der Ausgabe, die ein Wert zwischen null und dechs sein kann.
- `-r` ist die Endpunktregion, in der der Tunnel geöffnet ist.

Weitere Informationen zu den Parametern finden Sie unter Mit Befehlszeilenargumenten festgelegte Optionen.

> **Note**
>
> Wenn Sie die neueste Version des lokalen Proxys im Quellmodus verwenden, müssen Sie den AWS CLI Parameter aus Gründen der Abwärtskompatibilität `--destination-client-type V1` auf dem Quellgerät angeben. Dies gilt, wenn Sie eine Verbindung zu einem dieser Zielmodi herstellen:
>
> - AWS IoT Geräte-Client
> - AWS IoT Sichere Tunneling-Komponente oder AWS IoT Greengrass Version 2 sichere Tunneling-Komponente
> - Jeder AWS IoT Secure Tunneling-Democode, der vor 2022 geschrieben wurde
> - 1.X-Versionen des lokalen Proxys

Dieser Parameter gewährleistet eine korrekte Kommunikation zwischen dem aktualisierten Quell-Proxy und älteren Zielclients. Weitere Informationen zu lokalen Proxyversionen finden Sie unter AWS IoT Secure Tunneling on. GitHub

Im Folgenden finden Sie eine Beispielausgabe für die Ausführung des lokalen Proxys im `source` Modus.

```
...
...

Starting proxy in source mode
Attempting to establish web socket connection with endpoint wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Resolved proxy  server IP: 10.10.0.11
Connected successfully with proxy server
Performing SSL handshake with proxy server
Successfully completed SSL handshake with proxy server
HTTP/1.1 101 Switching Protocols

...

Connection: upgrade
channel-id: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
upgrade: websocket

...

Web socket session ID: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
Web socket subprotocol selected: aws.iot.securetunneling-2.0
Successfully established websocket connection with proxy server: wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Setting up web socket pings for every 5000 milliseconds
Scheduled next read:

...

Starting web socket read loop continue reading...
Resolved bind IP: 127.0.0.1
Listening for new connection on port 5555
```

Lokalen Proxy im Zielmodus betreiben

Die folgenden Befehle zeigen, wie der lokale Proxy im Zielmodus ausgeführt wird.

Linux/macOS

Führen Sie unter Linux oder macOS die folgenden Befehle im Terminal aus, um den lokalen Proxy auf Ihrem Ziel zu konfigurieren und zu starten.

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
./localproxy -d 22 -v 5 -r us-west-2
```

Wobei Folgendes gilt:

- `-d` ist die Zielanwendung, die den lokalen Proxy im Zielmodus startet.
- `-v` ist die Ausführlichkeit der Ausgabe, die ein Wert zwischen null und dechs sein kann.
- `-r` ist die Endpunktregion, in der der Tunnel geöffnet ist.

Weitere Informationen zu den Parametern finden Sie unter Mit Befehlszeilenargumenten festgelegte Optionen.

Windows

In Windows konfigurieren Sie den lokalen Proxy ähnlich wie für Linux oder macOS, aber die Art und Weise, wie Sie die Umgebungsvariablen definieren, unterscheidet sich von den anderen Plattformen. Führen Sie die folgenden Befehle im `cmd`-Fenster aus, um den lokalen Proxy auf Ihrem Ziel zu konfigurieren und zu starten.

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
.\localproxy -d 22 -v 5 -r us-west-2
```

Wobei Folgendes gilt:

- `-d` ist die Zielanwendung, die den lokalen Proxy im Zielmodus startet.
- `-v` ist die Ausführlichkeit der Ausgabe, die ein Wert zwischen null und dechs sein kann.
- `-r` ist die Endpunktregion, in der der Tunnel geöffnet ist.

Weitere Informationen zu den Parametern finden Sie unter Mit Befehlszeilenargumenten festgelegte Optionen.

> **Note**
>
> Wenn Sie die neueste Version des lokalen Proxys im Zielmodus verwenden, müssen Sie den AWS CLI Parameter aus Gründen der Abwärtskompatibilität `--destination-client-type V1` auf dem Zielgerät angeben. Dies gilt, wenn Sie eine Verbindung zu einem dieser Quellmodi herstellen:
>
> - Browser-based Sicheres Tunneling von der AWS Konsole aus.
> - 1.X-Versionen des lokalen Proxys
>
> Dieser Parameter gewährleistet eine korrekte Kommunikation zwischen dem aktualisierten Ziel-Proxy und älteren Quellclients. Weitere Informationen zu lokalen Proxyversionen finden Sie unter AWS IoT Secure Tunneling on. GitHub

Im Folgenden finden Sie eine Beispielausgabe für die Ausführung des lokalen Proxys im `destination` Modus.

```
...
...

Starting proxy in destination mode
Attempting to establish web socket connection with endpoint wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Resolved proxy  server IP: 10.10.0.11
Connected successfully with proxy server
Performing SSL handshake with proxy server
Successfully completed SSL handshake with proxy server
HTTP/1.1 101 Switching Protocols

...

Connection: upgrade
channel-id: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
upgrade: websocket

...

Web socket session ID: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
Web socket subprotocol selected: aws.iot.securetunneling-2.0
```

```
Successfully established websocket connection with proxy server: wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Setting up web socket pings for every 5000 milliseconds
Scheduled next read:

...

Starting web socket read loop continue reading...
```

Konfigurieren Sie den lokalen Proxy für Geräte, die einen Web-Proxy verwenden

Sie können den lokalen Proxy auf AWS IoT Geräten verwenden, um mit AWS IoT sicheren Tunneling-APIs zu kommunizieren. Der lokale Proxy überträgt Daten, die von der Geräteanwendung gesendet werden, mithilfe von sicherem Tunneling über eine sichere Verbindung. WebSocket Der lokale Proxy kann im Modus `source` oder `destination` arbeiten. Im `source`-Modus läuft er auf demselben Gerät oder Netzwerk, das die TCP-Verbindung initiiert. Im `destination`-Modus wird der lokale Proxy zusammen mit der Zielanwendung auf dem Remote-Gerät ausgeführt. Weitere Informationen finden Sie unter Lokaler Proxy.

Der lokale Proxy muss eine direkte Verbindung zum Internet herstellen, um sicheres Tunneling verwenden zu können AWS IoT . Für eine langlebige TCP-Verbindung mit sicherem Tunneling aktualisiert der lokale Proxy die HTTPS-Anfrage, um eine Verbindung zu einem der WebSockets Verbindungsendpunkte des sicheren Tunnelgeräts herzustellen.

Wenn sich Ihre Geräte in einem Netzwerk befinden, das einen Web-Proxy verwendet, kann der Web-Proxy die Verbindungen abfangen, bevor er sie an das Internet weiterleitet. Um eine dauerhafte Verbindung zu den Verbindungsendpunkten für Secure Tunneling-Geräte herzustellen, konfigurieren Sie Ihren lokalen Proxy so, dass er den Web-Proxy verwendet, wie in der Websocket-Spezifikation beschrieben.

> **Note**
>
> AWS IoT Geräte-Client unterstützt keine Geräte, die einen Web-Proxy verwenden. Um mit dem Web-Proxy arbeiten zu können, müssen Sie einen lokalen Proxy verwenden und ihn so konfigurieren, dass er mit einem Web-Proxy funktioniert, wie unten beschrieben.

Die folgenden Schritte zeigen, wie der lokale Proxy mit einem Web-Proxy funktioniert.

1. Der lokale Proxy sendet eine HTTP `CONNECT`-Anfrage an den Web-Proxy, die die Remote-Adresse des Secure Tunneling-Service zusammen mit den Authentifizierungsinformationen für den Web-Proxy enthält.
2. Der Web-Proxy stellt dann eine langlebige Verbindung zu den Remote-Secure-Tunneling-Endpunkten her.
3. Die TCP-Verbindung ist hergestellt und der lokale Proxy funktioniert jetzt sowohl im Quell- als auch im Zielmodus für die Datenübertragung.

Führen Sie die folgenden Schritte aus, um dieses Verfahren abzuschließen.

- Erstellen Sie den lokalen Proxy
- Konfigurieren Sie Ihren Web-Proxy
- Konfigurieren und Starten des lokalen Proxys

Erstellen Sie den lokalen Proxy

Öffnen Sie den lokalen Proxy-Quellcode im GitHub Repository und folgen Sie den Anweisungen zum Erstellen und Installieren des lokalen Proxys.

Konfigurieren Sie Ihren Web-Proxy

Der lokale Proxy stützt sich auf den in der Spezifikation beschriebenen HTTP-Tunneling-Mechanismus. HTTP/1.1 Um den Spezifikationen zu entsprechen, muss Ihr Web-Proxy Geräten die Verwendung der `CONNECT`Methode ermöglichen.

Wie Sie Ihren Web-Proxy konfigurieren, hängt von dem von Ihnen verwendeten Web-Proxy und der Web-Proxy-Version ab. Um die korrekte Konfiguration des Web-Proxys sicherzustellen, lesen Sie die Anleitung Ihres Web-Proxys.

Zur Konfiguration Ihres Web-Proxys müssen Sie zunächst die URL Ihres Web-Proxys ermitteln und prüfen, ob Ihr Web-Proxy HTTP-Tunneling unterstützt. Die URL des Web-Proxys wird später verwendet, wenn Sie den lokalen Proxy konfigurieren und starten.

1. Identifizieren Sie Ihre URL des Web-Proxys

   Ihre URL für den Web-Proxy hat das folgende Format.

   ```
   protocol://web_proxy_host_domain:web_proxy_port
   ```

AWS IoT Secure Tunneling unterstützt nur die Standardauthentifizierung für den Webproxy. Um die Standardauthentifizierung zu verwenden, müssen Sie das **`username`** und **`password`** als Teil der URL des Web-Proxys angeben. Die URL für den Web-Proxy hat das folgende Format.

```
protocol://username:password@web_proxy_host_domain:web_proxy_port
```

- *`protocol`*kann oder sein. `http` `https` Wir empfehlen Ihnen, `https` zu verwenden.
- *`web_proxy_host_domain`*ist die IP-Adresse Ihres Webproxys oder ein DNS-Name, der in die IP-Adresse Ihres Webproxys aufgelöst wird.
- *`web_proxy_port`*ist der Port, auf dem der Webproxy lauscht.
- Der Web-Proxy verwendet dies **`username`** und **`password`**, um die Anfrage zu authentifizieren.

2. URL des Web-Proxys testen

   Um zu überprüfen, ob Ihr Web-Proxy TCP-Tunneling unterstützt, verwenden Sie einen `curl` Befehl und stellen Sie sicher, dass Sie eine Antwort `2xx` oder `3xx` erhalten.

   Wenn Ihre URL für den Web-Proxy beispielsweise lautet `https://server.com:1235`, verwenden Sie ein `proxy-insecure` Flag zusammen mit dem `curl` Befehl, da der Web-Proxy möglicherweise auf einem selbstsignierten Zertifikat basiert.

```
export HTTPS_PROXY=https://server.com:1235
curl -I https://aws.amazon.com --proxy-insecure
```

   Wenn Ihre URL für den Web-Proxy einen `http`-Port hat (z. B.`http://server.com:1234`), müssen Sie das `proxy-insecure`-Flag nicht verwenden.

```
export HTTPS_PROXY=http://server.com:1234
curl -I https://aws.amazon.com
```

Konfigurieren und Starten des lokalen Proxys

Um den lokalen Proxy für die Verwendung eines Web-Proxys zu konfigurieren, müssen Sie die `HTTPS_PROXY` Umgebungsvariable entweder mit den DNS-Domain-Namen oder den IP-Adressen und Portnummern konfigurieren, die Ihr Web-Proxy verwendet.

Nachdem Sie den lokalen Proxy konfiguriert haben, können Sie den lokalen Proxy verwenden, wie in diesem README-Dokument beschrieben.

> **Note**
>
> Bei der Deklaration Ihrer Umgebungsvariablen wird die Groß-/Kleinschreibung beachtet. Wir empfehlen, jede Variable einmal zu definieren, indem Sie entweder nur Groß- oder Kleinbuchstaben verwenden. In den folgenden Beispielen wird die Umgebungsvariable in Großbuchstaben angegeben. Wenn dieselbe Variable sowohl in Groß- als auch in Kleinbuchstaben angegeben wird, hat die in Kleinbuchstaben angegebene Variable Vorrang.

Die folgenden Befehle zeigen, wie Sie den lokalen Proxy, der auf Ihrem Ziel ausgeführt wird, so konfigurieren, dass er den Web-Proxy verwendet und den lokalen Proxy startet.

- `AWSIOT_TUNNEL_ACCESS_TOKEN`: Diese Variable enthält das Client-Zugriffstoken (CAT) für das Ziel.
- `HTTPS_PROXY`: Diese Variable enthält die URL des Web-Proxys oder die IP-Adresse für die Konfiguration des lokalen Proxys.

Die in den folgenden Beispielen gezeigten Befehle hängen vom verwendeten Betriebssystem ab und davon, ob der Web-Proxy einen HTTP- oder einen HTTPS-Port abhört.

Der Web-Proxy überwacht einen HTTP-Port

Wenn Ihr Web-Proxy einen HTTP-Port abhört, können Sie die URL des Web-Proxys oder IP-Adresse für die `HTTPS_PROXY` Variable angeben.

Linux/macOS

In Linux oder macOS führen Sie die folgenden Befehle im Terminal aus, um den lokalen Proxy auf Ihrem Ziel zu konfigurieren und zu starten, um einen Web-Proxy zu verwenden, der einen HTTP-Port abhört.

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http:proxy.example.com:1234
./localproxy -r us-east-1 -d 22
```

Wenn Sie sich beim Proxy authentifizieren müssen, müssen Sie ein **username** und **password** als Teil der HTTPS_PROXY-Variablen angeben.

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http://username:password@proxy.example.com:1234
./localproxy -r us-east-1 -d 22
```

Windows

In Windows konfigurieren Sie den lokalen Proxy ähnlich wie für Linux oder macOS, aber die Art und Weise, wie Sie die Umgebungsvariablen definieren, unterscheidet sich von den anderen Plattformen. Führen Sie die folgenden Befehle im cmd-Fenster aus, um den lokalen Proxy auf Ihrem Ziel so zu konfigurieren und zu starten, dass er einen Web-Proxy verwendet, der einen HTTP-Port abhört.

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
set HTTPS_PROXY=http://proxy.example.com:1234
.\localproxy -r us-east-1 -d 22
```

Wenn Sie sich beim Proxy authentifizieren müssen, müssen Sie ein **username** und **password** als Teil der HTTPS_PROXY-Variablen angeben.

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
set HTTPS_PROXY=http://username:password@10.15.20.25:1234
.\localproxy -r us-east-1 -d 22
```

Der Web-Proxy überwacht einen HTTPS-Port

Führen Sie die folgenden Befehle aus, wenn Ihr Web-Proxy einen HTTPS-Port abhört.

> **Note**
>
> Wenn Sie ein selbstsigniertes Zertifikat für den Web-Proxy verwenden oder wenn Sie den lokalen Proxy auf einem Betriebssystem ausführen, das keine native OpenSSL-Unterstützung und keine Standardkonfigurationen hat, müssen Sie Ihre Web-Proxy-Zertifikate einrichten, wie im Abschnitt Zertifikatseinrichtung im GitHub Repository beschrieben.

Die folgenden Befehle ähneln der Konfiguration Ihres Web-Proxys für einen HTTP-Proxy, mit der Ausnahme, dass Sie auch den Pfad zu den Zertifikatsdateien angeben, die Sie wie zuvor beschrieben installiert haben.

Linux/macOS

In Linux oder macOS führen Sie die folgenden Befehle im Terminal aus, um den lokalen Proxy auf Ihrem Ziel zu konfigurieren, um einen Web-Proxy zu verwenden, der einen HTTPS-Port abhört.

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http:proxy.example.com:1234
./localproxy -r us-east-1 -d 22 -c /path/to/certs
```

Wenn Sie sich beim Proxy authentifizieren müssen, müssen Sie ein **username** und **password** als Teil der HTTPS_PROXY-Variablen angeben.

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http://username:password@proxy.example.com:1234
./localproxy -r us-east-1 -d 22 -c /path/to/certs
```

Windows

In Windows führen Sie die folgenden Befehle im cmd-Fenster aus, um den lokalen Proxy auf Ihrem Ziel so zu konfigurieren und zu starten, dass er einen Web-Proxy verwendet, der einen HTTP-Port abhört.

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
set HTTPS_PROXY=http://proxy.example.com:1234
.\localproxy -r us-east-1 -d 22 -c \path\to\certs
```

Wenn Sie sich beim Proxy authentifizieren müssen, müssen Sie ein **username** und **password** als Teil der HTTPS_PROXY-Variablen angeben.

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
set HTTPS_PROXY=http://username:password@10.15.20.25:1234
.\localproxy -r us-east-1 -d 22 -c \path\to\certs
```

Beispielbefehl und Ausgabe

Das Folgende zeigt ein Beispiel für einen Befehl, den Sie auf einem Linux OS ausführen, und die entsprechende Ausgabe. Das Beispiel zeigt einen Web-Proxy, der einen HTTP-Port abhört, und wie der lokale Proxy so konfiguriert werden kann, dass er den Web-Proxy in beiden `source` `destination` Modi verwendet. Bevor Sie diese Befehle ausführen können, müssen Sie bereits einen Tunnel geöffnet und die Client-Zugriffstoken für die Quelle und das Ziel erhalten haben. Sie müssen auch den lokalen Proxy erstellt und Ihren Web-Proxy wie zuvor beschrieben konfiguriert haben.

Hier ist eine Übersicht über die Schritte, nachdem Sie den lokalen Proxy gestartet haben. Der lokale Proxy:

- Identifiziert die URL des Web-Proxys, so dass er die URL zur Verbindung mit dem Proxy-Server verwenden kann.
- Stellt eine TCP-Verbindung mit dem Web-Proxy her.
- Sendet eine `CONNECT` HTTP-Anfrage an den Web-Proxy und wartet auf die `HTTP/1.1 200`-Antwort, die darauf hinweist, dass die Verbindung hergestellt wurde.
- Aktualisiert das HTTPS-Protokoll auf, WebSockets um eine langlebige Verbindung herzustellen.
- Beginnt mit der Übertragung von Daten über die Verbindung zu den Endpunkten der Secure-Tunneling-Geräte.

> **Note**
>
> Die folgenden Befehle, die in den Beispielen verwendet werden, verwenden das `verbosity`-Flag, um einen Überblick über die verschiedenen zuvor beschriebenen Schritte zu veranschaulichen, nachdem Sie den lokalen Proxy ausgeführt haben. Wir empfehlen, dass Sie das Flag nur für Tests verwenden.

Lokalen Proxy im Quellmodus betreiben

Die folgenden Befehle veranschaulichen die Ausführung des lokalen Proxys im Quellmodus.

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http:username:password@10.15.10.25:1234
./localproxy -s 5555 -v 5 -r us-west-2
```

Im Folgenden finden Sie ein Beispiel für die Ausgabe der Ausführung des lokalen Proxys im `source`-Modus.

```
...

Parsed basic auth credentials for the URL
Found Web proxy information in the environment variables, will use it to connect via
 the proxy.

...

Starting proxy in source mode
Attempting to establish web socket connection with endpoint wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Resolved Web proxy IP: 10.10.0.11
Connected successfully with Web Proxy
Successfully sent HTTP CONNECT to the Web proxy
Full response from the Web proxy:
HTTP/1.1 200 Connection established
TCP tunnel established successfully
Connected successfully with proxy server
Successfully completed SSL handshake with proxy server
Web socket session ID: 0a109afffee745f5-00001341-000b8138-cc6c878d80e8adb0-f186064b
Web socket subprotocol selected: aws.iot.securetunneling-2.0
Successfully established websocket connection with proxy server: wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Seting up web socket pings for every 5000 milliseconds
Scheduled next read:

...

Starting web socket read loop continue reading...
Resolved bind IP: 127.0.0.1
Listening for new connection on port 5555
```

Lokalen Proxy im Zielmodus betreiben

Die folgenden Befehle veranschaulichen die Ausführung des lokalen Proxys im Zielmodus.

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http:username:password@10.15.10.25:1234
./localproxy -d 22 -v 5 -r us-west-2
```

Im Folgenden finden Sie ein Beispiel für die Ausgabe der Ausführung des lokalen Proxys im `destination`-Modus.

```
...

Parsed basic auth credentials for the URL
Found Web proxy information in the environment variables, will use it to connect via
 the proxy.

...

Starting proxy in destination mode
Attempting to establish web socket connection with endpoint wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Resolved Web proxy IP: 10.10.0.1
Connected successfully with Web Proxy
Successfully sent HTTP CONNECT to the Web proxy
Full response from the Web proxy:
HTTP/1.1 200 Connection established
TCP tunnel established successfully
Connected successfully with proxy server
Successfully completed SSL handshake with proxy server
Web socket session ID: 06717bfffed3fd05-00001355-000b8315-da3109a85da804dd-24c3d10d
Web socket subprotocol selected: aws.iot.securetunneling-2.0
Successfully established websocket connection with proxy server: wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Seting up web socket pings for every 5000 milliseconds
Scheduled next read:

...

Starting web socket read loop continue reading...
```

Multiplex-Datenströme und gleichzeitige Verwendung von TCP-Verbindungen in einem sicheren Tunnel

Mithilfe der Multiplexing-Funktion für Secure Tunneling können Sie mehrere Datenströme pro Tunnel verwenden. Mit Multiplexing können Sie Fehler bei Geräten beheben, die mehrere Datenströme verwenden. Sie können auch Ihre Betriebslast reduzieren, indem Sie nicht mehr mehrere lokale Proxys erstellen, bereitstellen und starten oder mehrere Tunnel zum selben Gerät öffnen müssen.

Multiplexing kann beispielsweise für einen Webbrowser verwendet werden, der das Senden mehrerer HTTP- und SSH-Datenströme erfordert.

Für jeden Datenstrom unterstützt AWS IoT Secure Tunneling gleichzeitige TCP-Verbindungen. Durch die Verwendung gleichzeitiger Verbindungen wird das Risiko eines Timeouts bei mehreren Anfragen vom Client verringert. So kann beispielsweise die Ladezeit beim Fernzugriff auf einen Webserver reduziert werden, der sich lokal auf dem Zielgerät befindet.

In den folgenden Abschnitten werden weitere Informationen zum Multiplexing und zur Verwendung gleichzeitiger TCP-Verbindungen sowie deren verschiedenen Anwendungsfälle erläutert.

Themen

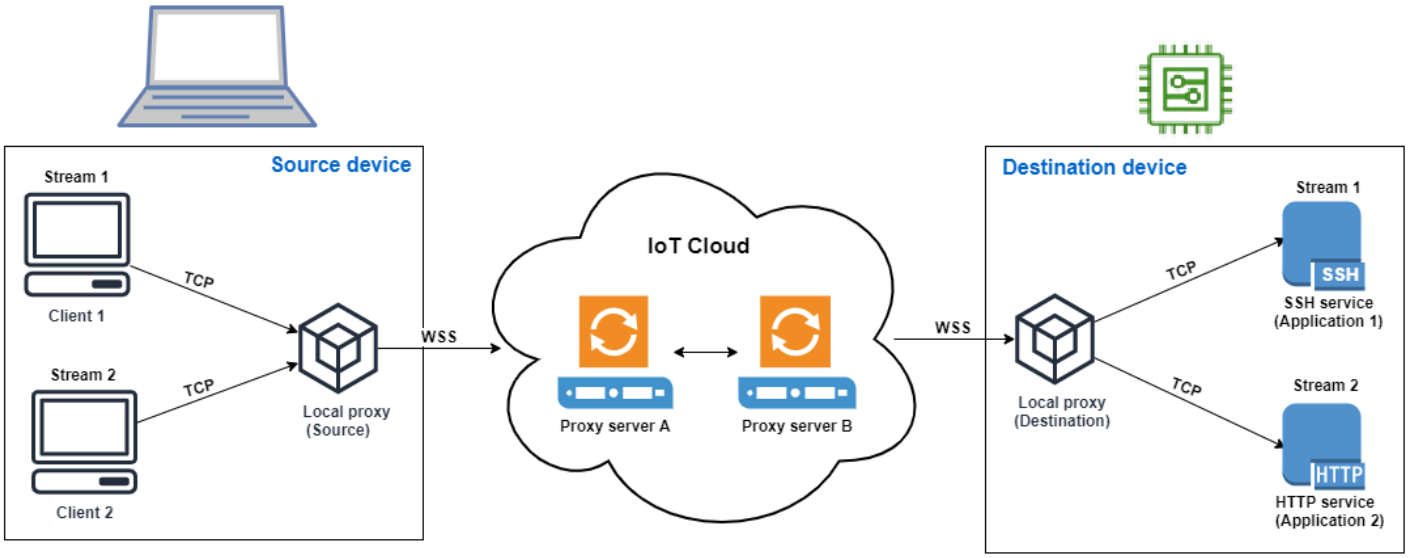
- Multiplexing mehrerer Datenströme in einem sicheren Tunnel
- Gleichzeitige TCP-Verbindungen in einem sicheren Tunnel verwenden

Multiplexing mehrerer Datenströme in einem sicheren Tunnel

Sie können die Multiplexing-Funktion für Geräte verwenden, die mehrere Verbindungen oder Anschlüsse verwenden. Multiplexing kann auch verwendet werden, wenn Sie mehrere Verbindungen zu einem Remote-Gerät zur Behebung von Fehlern benötigen. Es kann beispielsweise für einen Webbrowser verwendet werden, der das Senden mehrerer HTTP- und SSH-Datenströme erfordert. Die Anwendungsdaten aus beiden Streams werden gleichzeitig über den Multiplex-Tunnel an das Gerät gesendet.

Beispielanwendungsfall

Sagen wir, Sie müssen eine Verbindung zu einer geräteinternen Webanwendung zum Ändern einiger Netzwerkparameter herstellen und gleichzeitig Shell-Befehle über das Terminal eingeben, um die ordnungsgemäße Funktion des Geräts mit den neuen Netzwerkparametern zu überprüfen. In diesem Szenario müssen Sie möglicherweise sowohl über HTTP als auch über SSH eine Verbindung zum Gerät herstellen und zwei parallele Datenströme übertragen, um gleichzeitig auf die Webanwendung und das Terminal zuzugreifen. Mit der Multiplexing-Funktion können diese beiden unabhängigen Ströme gleichzeitig über denselben Tunnel übertragen werden.



Wie man einen Multiplex-Tunnel einrichtet

Das folgende Verfahren führt Sie durch die Einrichtung eines Multiplex-Tunnels zur Fehlerbehebung bei Geräten mithilfe von Anwendungen, für die Verbindungen zu mehreren Ports erforderlich sind. Sie werden einen Tunnel mit zwei Multiplex-Streams einrichten: einen HTTP-Stream und einen SSH-Stream.

1. (Optional) Konfigurationsdateien erstellen

   Sie können das Quell- und Zielgerät optional mit Konfigurationsdateien konfigurieren. Verwenden Sie Konfigurationsdateien, wenn sich Ihre Portzuordnungen wahrscheinlich häufig ändern. Sie können diesen Schritt überspringen, wenn Sie die Portzuordnung lieber explizit über die CLI angeben möchten oder wenn Sie den lokalen Proxy nicht auf bestimmten Abhörports starten müssen. Weitere Informationen zur Verwendung von Konfigurationsdateien finden Sie unter Optionen, die über --config festgelegt werden, in GitHub.

   1. Erstellen Sie auf Ihrem Quellgerät in dem Ordner, in dem Ihr lokaler Proxy ausgeführt wird, einen Konfigurationsordner mit dem Namen`Config`. Erstellen Sie in diesem Ordner eine Datei mit dem Namen `SSHSource.ini` und dem folgenden Inhalt:

      ```
      HTTP1 = 5555
      SSH1 = 3333
      ```

2. Erstellen Sie auf Ihrem Zielgerät in dem Ordner, in dem Ihr lokaler Proxy ausgeführt wird, einen Konfigurationsordner mit dem Namen`Config`. Erstellen Sie in diesem Ordner eine Datei mit dem Namen `SSHDestination.ini` und dem folgenden Inhalt:

```
HTTP1 = 80
SSH1 = 22
```

2. Tunnel öffnen

Öffnen Sie einen Tunnel mit der `OpenTunnel` API-Funktion oder dem `open-tunnel` CLI-Befehl. Konfigurieren Sie das Ziel, indem Sie die `HTTP1` Dienste `SSH1` und den Namen der AWS IoT Sache, die Ihrem Remote-Gerät entspricht, als und angeben. Ihre SSH- und HTTP-Anwendungen werden auf diesem Remote-Gerät ausgeführt. Sie müssen das IoT-Ding bereits in der AWS IoT Registrierung erstellt haben. Weitere Informationen finden Sie unter Dinge mit der Registrierung verwalten.

```
aws iotsecuretunneling open-tunnel \
 --destination-config thingName=RemoteDevice1,services=HTTP1,SSH1
```

Wenn Sie diesen Befehl ausführen, werden die Quell- und Zielzugriffstoken generiert, mit denen Sie den lokalen Proxy ausführen.

```
{
 "tunnelId": "b2de92a3-b8ff-46c0-b0f2-afa28b00cecd",
 "tunnelArn": "arn:aws:iot:us-west-2:431600097591:tunnel/b2de92a3-b8ff-46c0-b0f2-
afa28b00cecd",
 "sourceAccessToken": source_client_access_token,
 "destinationAccessToken": destination_client_access_token
}
```

3. Konfigurieren und Starten des lokalen Proxys

Bevor Sie den lokalen Proxy ausführen können, richten Sie entweder den AWS IoT Device Client ein oder laden Sie den lokalen Proxy-Quellcode von herunter GitHub und erstellen Sie ihn für die Plattform Ihrer Wahl. Anschließend können Sie das Ziel und den lokalen Quell-Proxy starten, um eine Verbindung zum sicheren Tunnel herzustellen. Weitere Informationen zum Konfigurieren und Verwenden des lokalen Proxys finden Sie unter Wie man den lokalen Proxy benutzt.

> **Note**
>
> Wenn Sie auf Ihrem Quellgerät keine Konfigurationsdateien verwenden oder die Portzuordnung mit der CLI angeben, können Sie trotzdem denselben Befehl verwenden, um den lokalen Proxy auszuführen. Der lokale Proxy im Quellmodus wählt automatisch die zu verwendenden Ports und deren Zuordnungen für Sie aus.

Start local proxy using configuration files

Führen Sie die folgenden Befehle aus, um den lokalen Proxy mithilfe von Konfigurationsdateien im Quell- und Zielmodus auszuführen.

```
// ----------------- Start the destination local proxy ----------------------
./localproxy -r us-east-1 -m dst -t destination_client_access_token

// ----------------- Start the source local proxy ---------------------------
// You also run the same command below if you want the local proxy to
// choose the mappings for you instead of using configuration files.
./localproxy -r us-east-1 -m src -t source_client_access_token
```

Start local proxy using CLI port mapping

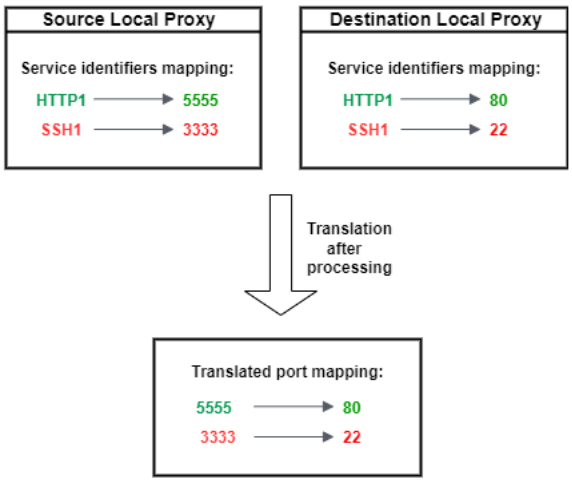
Führen Sie die folgenden Befehle aus, um den lokalen Proxy im Quell- und Zielmodus auszuführen, indem Sie die Portzuordnungen explizit mit der CLI angeben.

```
// ----------------- Start the destination local proxy
 ----------------------------------
./localproxy -r us-east-1 -d HTTP1=80,SSH1=22 -t destination_client_access_token

// ----------------- Start the source local proxy
 ---------------------------------------
./localproxy -r us-east-1 -s HTTP1=5555,SSH1=33 -t source_client_access_token
```

Die Anwendungsdaten aus der SSH- und HTTP-Verbindung können jetzt gleichzeitig über den Multiplex-Tunnel übertragen werden. Wie in der Abbildung unten zu sehen ist, dient die Service-ID als lesbares Format zur Übersetzung der Portzuweisung zwischen dem Quell- und dem Zielgerät. Bei dieser Konfiguration leitet Secure Tunneling jeglichen eingehenden HTTP-Verkehr von Port *5555* auf

dem Quellgerät zu Port *80* auf dem Zielgerät und allen eingehenden SSH-Verkehr von Port *3333* zu Port *22* auf dem Zielgerät weiter.



Gleichzeitige TCP-Verbindungen in einem sicheren Tunnel verwenden

AWS IoT Secure Tunneling unterstützt mehr als eine TCP-Verbindung gleichzeitig für jeden Datenstrom. Sie können diese Funktion verwenden, wenn Sie gleichzeitige Verbindungen zu einem Remote-Gerät benötigen. Durch die Verwendung gleichzeitiger TCP-Verbindungen wird das Risiko eines Timeouts bei mehreren Anfragen vom Client verringert. Wenn Sie beispielsweise auf einen Webserver zugreifen, auf dem mehrere Komponenten ausgeführt werden, können gleichzeitige TCP-Verbindungen die Zeit reduzieren, die zum Laden der Seite benötigt wird.

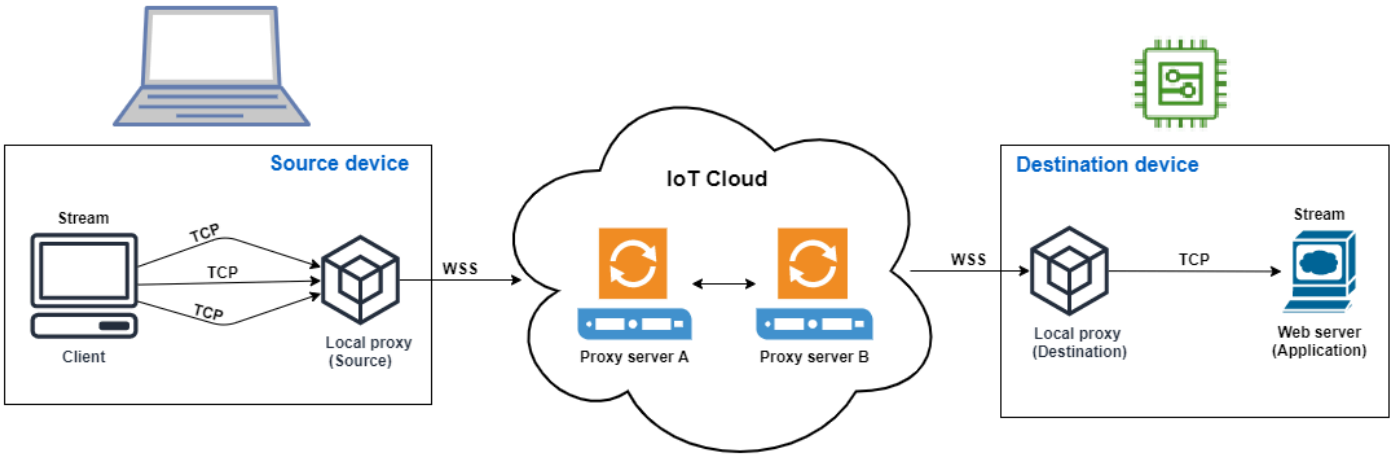
> **Note**
>
> Gleichzeitige TCP-Verbindungen haben ein Bandbreitenlimit von jeweils 800 Kilobyte pro Sekunde. AWS-Konto AWS IoT Secure Tunneling kann dieses Limit je nach Anzahl der eingehenden Anfragen für Sie konfigurieren.

Beispielanwendungsfall

Nehmen wir an, Sie müssen aus der Ferne auf einen Webserver zugreifen, der sich lokal auf dem Zielgerät befindet und auf dem mehrere Komponenten laufen. Bei einer einzigen TCP-Verbindung kann das sequenzielle Laden beim Zugriff auf den Webserver die Zeit verlängern, die zum Laden der Ressourcen auf der Seite benötigt wird. Die gleichzeitigen TCP-Verbindungen können die Ladezeit reduzieren, indem sie die Ressourcenanforderungen der Seite erfüllen, wodurch die Zugriffszeit reduziert wird. Das folgende Diagramm zeigt, wie gleichzeitige TCP-Verbindungen für den

Datenstrom zur Webserver-Anwendung unterstützt werden, die auf dem Remote-Gerät ausgeführt wird.

> **Note**
>
> Wenn Sie über den Tunnel auf mehrere Anwendungen zugreifen möchten, die auf dem Remote-Gerät ausgeführt werden, können Sie Tunnelmultiplexing verwenden. Weitere Informationen finden Sie unter Multiplexing mehrerer Datenströme in einem sicheren Tunnel.



Wie verwendet man gleichzeitige TCP-Verbindungen

Im folgenden Verfahren wird beschrieben, wie Sie gleichzeitige TCP-Verbindungen für den Zugriff auf den Webbrowser auf dem entfernten Gerät verwenden können. Wenn mehrere Anfragen vom Client eingehen, richtet AWS IoT Secure Tunneling automatisch gleichzeitige TCP-Verbindungen ein, um die Anfragen zu bearbeiten, wodurch die Ladezeit reduziert wird.

1. Tunnel öffnen

   Öffnen Sie einen Tunnel mit der `OpenTunnel` API-Funktion oder dem `open-tunnel` CLI-Befehl. Konfigurieren Sie das Ziel, indem Sie `HTTP` als Dienst und den Namen des Objekts AWS IoT angeben, das Ihrem Remote-Gerät entspricht. Ihre Webserver-Anwendung wird auf diesem Remote-Gerät ausgeführt. Sie müssen das IoT-Ding bereits in der AWS IoT Registrierung erstellt haben. Weitere Informationen finden Sie unter Dinge mit der Registrierung verwalten.

   ```
   aws iotsecuretunneling open-tunnel \
   ```

```
 --destination-config thingName=RemoteDevice1,services=HTTP
```

Wenn Sie diesen Befehl ausführen, werden die Quell- und Zielzugriffstoken generiert, mit denen Sie den lokalen Proxy ausführen.

```
{
 "tunnelId": "b2de92a3-b8ff-46c0-b0f2-afa28b00cecd",
 "tunnelArn": "arn:aws:iot:us-west-2:431600097591:tunnel/b2de92a3-b8ff-46c0-b0f2-
afa28b00cecd",
 "sourceAccessToken": source_client_access_token,
 "destinationAccessToken": destination_client_access_token
}
```

2. Konfigurieren und Starten des lokalen Proxys

Bevor Sie den lokalen Proxy ausführen können, laden Sie den lokalen Proxy-Quellcode von herunter GitHub und erstellen Sie ihn für die Plattform Ihrer Wahl. Anschließend können Sie den lokalen Ziel- und Quell-Proxy starten, um eine Verbindung zum sicheren Tunnel herzustellen und die Remote-Web-Server-Anwendung zu verwenden.

> **Note**
>
> Damit AWS IoT sicheres Tunneling gleichzeitige TCP-Verbindungen verwendet, müssen Sie auf die neueste Version des lokalen Proxys aktualisieren. Diese Funktion ist nicht verfügbar, wenn Sie den lokalen Proxy mit dem AWS IoT Device Client konfigurieren.

```
// Start the destination local proxy
./localproxy -r us-east-1 -d HTTP=80 -t destination_client_access_token

// Start the source local proxy
./localproxy -r us-east-1 -s HTTP=5555 -t source_client_access_token
```

Weitere Informationen zum Konfigurieren und Verwenden des lokalen Proxys finden Sie unter Wie man den lokalen Proxy benutzt.

Sie können jetzt den Tunnel verwenden, um auf die Webserver-Anwendung zuzugreifen. AWS IoT Secure Tunneling richtet automatisch die gleichzeitigen TCP-Verbindungen ein und verarbeitet sie, wenn mehrere Anfragen vom Client eingehen.

Ein Remote-Gerät konfigurieren und IoT-Agent verwenden

Der IoT-Agent wird verwendet, um die MQTT-Nachricht zu empfangen, die das Clientzugriffstoken enthält, und einen lokalen Proxy auf dem Remote-Gerät zu starten. Sie müssen den IoT-Agenten auf dem entfernten Gerät installieren und ausführen, wenn Sie das Client-Zugriffstoken über MQTT durch einen sicheren Tunnel übermitteln möchten. Der IoT-Agent muss das folgende reservierte IoT-MQTT-Thema abonnieren:

> **Note**
>
> Wenn Sie dem Remote-Gerät das Ziel-Client-Zugriffstoken über andere Methoden als durch das Abonnieren des reservierten MQTT-Topics zukommen lassen möchten, benötigen Sie möglicherweise einen Ziel-Client-Zugriffstoken-Listener (CAT) und einen lokalen Proxy. Der CAT-Listener muss mit dem von Ihnen gewählten Zustellungsmechanismus für Client-Zugangstoken arbeiten und in einen lokalen Proxy im Zielmodus starten können.

IoT-Agent-Snippet

Der IoT-Agent muss das folgende reservierte IoT-MQTT-Topic abonnieren, damit er die MQTT-Nachricht empfangen und den lokalen Proxy starten kann:

`$aws/things/`*`thing-name`*`/tunnels/notify`

Wo `thing-name` ist der Name der AWS IoT Sache, die mit dem Remote-Gerät verknüpft ist?

Im Folgenden finden Sie ein Beispiel für eine MQTT-Nachrichtennutzlast:

```
{
    "clientAccessToken": "destination-client-access-token",
    "clientMode": "destination",
    "region": "aws-region",
    "services": ["destination-service"]
}
```

Nach dem Empfang einer MQTT-Nachricht muss der IoT-Agent einen lokalen Proxy auf dem Remote-Gerät mit den entsprechenden Parametern starten.

Der folgende Java-Code veranschaulicht, wie Sie mithilfe des AWS IoT Device SDK und ProcessBuilder mithilfe der Java-Bibliothek einen einfachen IoT-Agenten erstellen, der mit sicherem Tunneling arbeitet.

```
// Find the IoT device endpoint for your AWS-Konto
final String endpoint = iotClient.describeEndpoint(new
 DescribeEndpointRequest().withEndpointType("iot:Data-ATS")).getEndpointAddress();

// Instantiate the IoT Agent with your AWS credentials
final String thingName = "RemoteDeviceA";
final String tunnelNotificationTopic = String.format("$aws/things/%s/tunnels/notify",
 thingName);
final AWSIotMqttClient mqttClient = new AWSIotMqttClient(endpoint, thingName,
                 "your_aws_access_key", "your_aws_secret_key");

try {
    mqttClient.connect();
    final TunnelNotificationListener listener = new
 TunnelNotificationListener(tunnelNotificationTopic);
    mqttClient.subscribe(listener, true);
}
finally {
    mqttClient.disconnect();
}

private static class TunnelNotificationListener extends AWSIotTopic {
    public TunnelNotificationListener(String topic) {
        super(topic);
    }

    @Override
    public void onMessage(AWSIotMessage message) {
        try {
            // Deserialize the MQTT message
            final JSONObject json = new JSONObject(message.getStringPayload());

            final String accessToken = json.getString("clientAccessToken");
            final String region = json.getString("region");

            final String clientMode = json.getString("clientMode");
            if (!clientMode.equals("destination")) {
                throw new RuntimeException("Client mode " + clientMode + " in the MQTT
 message is not expected");
```

```
            }

            final JSONArray servicesArray = json.getJSONArray("services");
            if (servicesArray.length() > 1) {
                throw new RuntimeException("Services in the MQTT message has more than
 1 service");
            }
            final String service = servicesArray.get(0).toString();
            if (!service.equals("SSH")) {
                throw new RuntimeException("Service " + service + " is not supported");
            }

            // Start the destination local proxy in a separate process to connect to
 the SSH Daemon listening port 22
            final ProcessBuilder pb = new ProcessBuilder("localproxy",
                        "-t", accessToken,
                        "-r", region,
                        "-d", "localhost:22");
            pb.start();
        }
        catch (Exception e) {
            log.error("Failed to start the local proxy", e);
        }
    }
}
```

Steuern des Zugriffs auf Tunnel

DSecure Tunneling bietet dienstspezifische Aktionen, Ressourcen und Bedingungskontextschlüssel zur Verwendung in IAM-Berechtigungsrichtlinien.

Voraussetzungen für den Tunnelzugriff

- Erfahren Sie, wie Sie AWS Ressourcen mithilfe von IAM-Richtlinien sichern.
- Erfahren Sie, wie Sie IAM-Bedingungen erstellen und bewerten.
- Erfahren Sie, wie Sie AWS Ressourcen mithilfe von Ressourcen-Tags sichern.

Richtlinien für den Tunnelzugriff

Sie müssen die folgenden Richtlinien verwenden, um Berechtigungen zur Verwendung der Secure Tunneling-API zu autorisieren. Weitere Informationen zur AWS IoT Sicherheit finden Sie unterIdentitäts- und Zugriffsmanagement für AWS IoT.

IoT: OpenTunnel

Die `iot:OpenTunnel`-Richtlinienaktion erteilt eine Prinzipalberechtigung zum Aufrufen von OpenTunnel.

Im `Resource` Element der IAM-Richtlinienerklärung:

- Geben Sie den Wildcard-Tunnel-ARN an:

  `arn:aws:iot:`*`aws-region`*`:`*`aws-account-id`*`:tunnel/*`
- Geben Sie einen Objekt-ARN an, um die `OpenTunnel` Berechtigung für bestimmte IoT-Objekte zu verwalten:

  `arn:aws:iot:`*`aws-region`*`:`*`aws-account-id`*`:thing/`*`thing-name`*

Mit der folgenden Richtlinienanweisung können Sie beispielsweise einen Tunnel für das IoT-Objekt mit der Bezeichnung `TestDevice` öffnen.

```
{
    "Effect": "Allow",
    "Action": "iot:OpenTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*",
        "arn:aws:iot:aws-region:aws-account-id:thing/TestDevice"
    ]
}
```

Die `iot:OpenTunnel`-Richtlinienaktion unterstützt die folgenden Bedingungsschlüssel:

- `iot:ThingGroupArn`
- `iot:TunnelDestinationService`
- `aws:RequestTag/`*`tag-key`*
- `aws:SecureTransport`

- `aws:TagKeys`

Mit der folgenden Richtlinienanweisung können Sie einen Tunnel zu dem Objekt öffnen, wenn das Objekt zu einer Objektgruppe mit einem Namen gehört, der mit `TestGroup` beginnt und im Tunnel der Zielservice SSH konfiguriert ist.

```
{
    "Effect": "Allow",
    "Action": "iot:OpenTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "ForAnyValue:StringLike": {
            "iot:ThingGroupArn": [
                "arn:aws:iot:aws-region:aws-account-id:thinggroup/TestGroup*"
            ]
        },
        "ForAllValues:StringEquals": {
            "iot:TunnelDestinationService": [
                "SSH"
            ]
        }
    }
}
```

Sie können auch Ressourcen-Tags verwenden, um die Berechtigung zum Öffnen von Tunneln zu steuern. Mit der folgenden Richtlinienanweisung kann beispielsweise ein Tunnel geöffnet werden, wenn der Tag-Schlüssel `Owner` mit dem Wert `Admin` vorhanden ist und keine anderen Tags angegeben werden. Allgemeine Informationen zur Verwendung von Tags finden Sie unter Taggen Sie Ihre AWS IoT Ressourcen.

```
{
    "Effect": "Allow",
    "Action": "iot:OpenTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:RequestTag/Owner": "Admin"
```

```
        },
        "ForAllValues:StringEquals": {
            "aws:TagKeys": "Owner"
        }
    }
}
```

Internet der Dinge: RotateTunnelAccessToken

Die `iot:RotateTunnelAccessToken`-Richtlinienaktion erteilt eine Prinzipalberechtigung zum Aufrufen von RotateTunnelAccessToken.

Im `Resource` Element der IAM-Richtlinienerklärung:

- Geben Sie einen vollqualifizierten Tunnel-ARN an:

  `arn:aws:iot:`*`aws-region`*`: `*`aws-account-id`*`:tunnel/`*`tunnel-id`*

  Sie können auch den Wildcard-Tunnel-ARN verwenden:

  `arn:aws:iot:`*`aws-region`*`:`*`aws-account-id`*`:tunnel/*`
- Geben Sie einen Objekt-ARN an, um die `RotateTunnelAccessToken` Berechtigung für bestimmte IoT-Objekte zu verwalten:

  `arn:aws:iot:`*`aws-region`*`:`*`aws-account-id`*`:thing/`*`thing-name`*

Mit der folgenden Richtlinienanweisung können Sie beispielsweise entweder das Quellzugriffstoken eines Tunnels oder das Zielzugriffstoken eines Clients für das genannte IoT-Objekt namens`TestDevice` rotieren.

```
{
    "Effect": "Allow",
    "Action": "iot:RotateTunnelAccessToken",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*",
        "arn:aws:iot:aws-region:aws-account-id:thing/TestDevice"
    ]
}
```

Die `iot:RotateTunnelAccessToken`-Richtlinienaktion unterstützt die folgenden Bedingungsschlüssel:

- `iot:ThingGroupArn`
- `iot:TunnelDestinationService`
- `iot:ClientMode`
- `aws:SecureTransport`

Mit der folgenden Richtlinienanweisung können Sie das Zielzugriffstoken auf das Objekt rotieren, wenn das Objekt zu einer Objektgruppe mit einem Namen gehört, der mit `TestGroup` beginnt, der konfigurierte Zieldienst auf dem Tunnel SSH ist, und der Client sich im `DESTINATION`-Modus befindet.

```
{
    "Effect": "Allow",
    "Action": "iot:RotateTunnelAccessToken",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "ForAnyValue:StringLike": {
            "iot:ThingGroupArn": [
                "arn:aws:iot:aws-region:aws-account-id:thinggroup/TestGroup*"
            ]
        },
        "ForAllValues:StringEquals": {
            "iot:TunnelDestinationService": [
                "SSH"
            ],
            "iot:ClientMode": "DESTINATION"
        }
    }
}
```

Internet der Dinge: DescribeTunnel

Die `iot:DescribeTunnel`-Richtlinienaktion erteilt eine Prinzipalberechtigung zum Aufrufen von DescribeTunnel.

Geben Sie im `Resource` Element der IAM-Richtlinienerklärung einen vollqualifizierten Tunnel-ARN an:

`arn:aws:iot:aws-region: aws-account-id:tunnel/tunnel-id`

Sie können auch den Wildcard-ARN verwenden:

`arn:aws:iot:aws-region:aws-account-id:tunnel/*`

Die `iot:DescribeTunnel`-Richtlinienaktion unterstützt die folgenden Bedingungsschlüssel:

- `aws:ResourceTag/tag-key`
- `aws:SecureTransport`

Mit der folgenden Richtlinienanweisung können Sie `DescribeTunnel` aufrufen, wenn der angeforderte Tunnel mit dem Schlüssel `Owner` mit dem Wert `Admin` gekennzeichnet ist.

```
{
    "Effect": "Allow",
    "Action": "iot:DescribeTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:ResourceTag/Owner": "Admin"
        }
    }
}
```

Internet der Dinge: ListTunnels

Die `iot:ListTunnels`-Richtlinienaktion erteilt eine Prinzipalberechtigung zum Aufrufen von ListTunnels.

Im `Resource` Element der IAM-Richtlinienerklärung:

- Geben Sie den Wildcard-Tunnel-ARN an:

  `arn:aws:iot:aws-region:aws-account-id:tunnel/*`
- Geben Sie einen Objekt-ARN an, um die `ListTunnels` Berechtigung für ausgewählte IoT-Objekte zu verwalten:

  `arn:aws:iot:aws-region:aws-account-id:thing/thing-name`

Die `iot:ListTunnels`-Richtlinienaktion unterstützt den Bedingungsschlüssel `aws:SecureTransport`.

Mit der folgenden Richtlinienanweisung können Sie Tunnel für das Objekt mit dem Namen `TestDevice` auflisten.

```
{
    "Effect": "Allow",
    "Action": "iot:ListTunnels",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*",
        "arn:aws:iot:aws-region:aws-account-id:thing/TestDevice"
    ]
}
```

Internet der Dinge: ListTagsForResource

Die `iot:ListTagsForResource`-Richtlinienaktion erteilt eine Prinzipalberechtigung zum Aufrufen von `ListTagsForResource`.

Geben Sie im `Resource` Element der IAM-Richtlinienerklärung einen vollqualifizierten Tunnel-ARN an:

`arn:aws:iot:aws-region: aws-account-id:tunnel/tunnel-id`

Sie können auch den Wildcard-Tunnel-ARN verwenden:

`arn:aws:iot:aws-region:aws-account-id:tunnel/*`

Die `iot:ListTagsForResource`-Richtlinienaktion unterstützt den Bedingungsschlüssel `aws:SecureTransport`.

Internet der Dinge: CloseTunnel

Die `iot:CloseTunnel`-Richtlinienaktion erteilt eine Prinzipalberechtigung zum Aufrufen von CloseTunnel.

Geben Sie im `Resource` Element der IAM-Richtlinienerklärung einen vollqualifizierten Tunnel-ARN an:

`arn:aws:iot:aws-region: aws-account-id:tunnel/tunnel-id`

Sie können auch den Wildcard-Tunnel-ARN verwenden:

`arn:aws:iot:aws-region:aws-account-id:tunnel/*`

Die `iot:CloseTunnel`-Richtlinienaktion unterstützt die folgenden Bedingungsschlüssel:

- `iot:Delete`
- `aws:ResourceTag/tag-key`
- `aws:SecureTransport`

Mit der folgenden Richtlinienanweisung können Sie `CloseTunnel` aufrufen, wenn der `Delete`-Parameter der Anforderung `false` ist und der angeforderte Tunnel mit dem Schlüssel `Owner` mit dem Wert `QATeam` gekennzeichnet ist.

```
{
    "Effect": "Allow",
    "Action": "iot:CloseTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "Bool": {
            "iot:Delete": "false"
        },
        "StringEquals": {
            "aws:ResourceTag/Owner": "QATeam"
        }
    }
}
```

Internet der Dinge: TagResource

Die `iot:TagResource`-Richtlinienaktion erteilt eine Prinzipalberechtigung zum Aufrufen von `TagResource`.

Geben Sie im `Resource` Element der IAM-Richtlinienerklärung einen vollqualifizierten Tunnel-ARN an:

`arn:aws:iot:aws-region: aws-account-id:tunnel/tunnel-id`

Sie können auch den Wildcard-Tunnel-ARN verwenden:

`arn:aws:iot:aws-region:aws-account-id:tunnel/*`

Die `iot:TagResource`-Richtlinienaktion unterstützt den Bedingungsschlüssel `aws:SecureTransport`.

Internet der Dinge: UntagResource

Die `iot:UntagResource`-Richtlinienaktion erteilt eine Prinzipalberechtigung zum Aufrufen von `UntagResource`.

Geben Sie im `Resource` Element der IAM-Richtlinienerklärung einen vollqualifizierten Tunnel-ARN an:

`arn:aws:iot:`*`aws-region`*`: `*`aws-account-id`*`:tunnel/`*`tunnel-id`*

Sie können auch den Wildcard-Tunnel-ARN verwenden:

`arn:aws:iot:`*`aws-region`*`:`*`aws-account-id`*`:tunnel/*`

Die `iot:UntagResource`-Richtlinienaktion unterstützt den Bedingungsschlüssel `aws:SecureTransport`.

Auflösen AWS IoT Probleme mit der Tunneling-Konnektivität durch rotierendes Rotieren der Client-Zugriffstoken

Wenn Sie AWS IoT Secure Tunneling verwenden, können Verbindungsprobleme auftreten, selbst wenn der Tunnel geöffnet ist. Die folgenden Abschnitte zeigen einige mögliche Probleme und wie Sie diese durch Rotation der Client-Zugriffstoken lösen können. Um das Client Access Token (CAT) zu rotieren, verwenden Sie die RotateTunnelAccessToken API oder das Rotate-Tunnel-Access-Token. https://docs.aws.amazon.com/cli/latest/reference/iotsecuretunneling/rotate-tunnel-access-token.html AWS CLI Je nachdem, ob bei der Verwendung des Clients im Quell- oder Zielmodus ein Fehler auftritt, können Sie das CAT entweder im Quell- oder Zielmodus oder in beiden Modi rotieren.

> **Note**
>
> - Wenn Sie sich nicht sicher sind, ob das CAT an der Quelle oder am Ziel rotiert werden muss, können Sie das CAT sowohl an der Quelle als auch am Ziel rotieren, indem Sie `ClientMode` auf ALL setzen bei der Verwendung der `RotateTunnelAccessToken`-API.
> - Durch das Rotieren des CAT wird die Tunneldauer nicht verlängert. Nehmen wir zum Beispiel an, die Tunneldauer beträgt 12 Stunden und der Tunnel ist bereits seit 4 Stunden

geöffnet. Wenn Sie die Zugriffstoken rotieren, können die neu generierten Token nur für die verbleibenden 8 Stunden verwendet werden.

Themen

- Fehler beim ungültigen Client-Zugriffstoken
- Fehler bei Nichtübereinstimmung der Client-Tokens
- Verbindungsprobleme bei Remote-Geräten

Fehler beim ungültigen Client-Zugriffstoken

Wenn Sie AWS IoT Secure Tunneling verwenden, kann ein Verbindungsfehler auftreten, wenn Sie dasselbe Client Access Token (CAT) verwenden, um erneut eine Verbindung zum gleichen Tunnel herzustellen. In diesem Fall kann der lokale Proxy keine Verbindung zum Secure Tunneling-Proxyserver herstellen. Wenn Sie einen Client im Quellmodus verwenden, wird möglicherweise die folgende Fehlermeldung angezeigt:

```
Invalid access token: The access token was previously used and cannot be used again
```

Der Fehler tritt auf, weil das Client-Zugriffstoken (CAT) vom lokalen Proxy nur einmal verwendet werden kann und dann ungültig wird. Um diesen Fehler zu beheben, wechseln Sie das Client-Zugriffstoken in den `SOURCE` Modus, in dem ein neues CAT für die Quelle generiert wird. Ein Beispiel, das zeigt, wie Sie das Quell-CAT rotieren, finden Sie unter Beispiel für „Quell-CAT rotieren“.

Fehler bei Nichtübereinstimmung der Client-Tokens

> **Note**
>
> Die Verwendung von Client-Tokens zur Wiederverwendung des CAT wird nicht empfohlen. Wir empfehlen, stattdessen die `RotateTunnelAccessToken`-API zu verwenden, um die Client-Zugriffstoken zu rotieren, um die Verbindung zum Tunnel wiederherzustellen.

Wenn Sie Client-Token verwenden, können Sie das CAT wiederverwenden, um die Verbindung zum Tunnel wiederherzustellen. Um die CAT wiederzuverwenden, müssen Sie dem Client-Token die CAT zur Verfügung stellen, wenn Sie zum ersten Mal eine Verbindung zu Secure Tunneling herstellen.

Secure Tunneling speichert das Client-Token, sodass für nachfolgende Verbindungsversuche mit demselben Token auch das Client-Token bereitgestellt werden muss. Weitere Informationen zur Verwendung von Client-Token finden Sie in der lokalen Proxy-Referenzimplementierung unter. GitHub

Wenn Sie Client-Token verwenden und einen Client im Quellmodus verwenden, wird möglicherweise der folgende Fehler angezeigt:

```
Invalid client token: The provided client token does not match the client token
    that was previously set.
```

Der Fehler tritt auf, weil das bereitgestellte Client-Token nicht mit dem Client-Token übereinstimmt, das beim Zugriff auf den Tunnel mit dem CAT bereitgestellt wurde. Um diesen Fehler zu beheben, rotieren Sie den CAT in dem `SOURCE` Modus, um ein neues CAT für die Quelle zu erzeugen. Im Folgenden sehen Sie ein Beispiel:

Beispiel für „Quell-CAT rotieren“

Im Folgenden finden Sie ein Beispiel dafür, wie die `RotateTunnelAccessToken` API im `SOURCE`-Modus zum Generieren einer neuen CAT für die Quelle ausgeführt wird:

```
aws iotsecuretunneling rotate-tunnel-access-token \
    --region <region> \
    --tunnel-id <tunnel-id> \
    --client-mode SOURCE
```

Wenn Sie diesen Befehl ausführen, wird ein neues Quellzugriffstoken generiert und der ARN Ihres Tunnels zurückgegeben.

```
{
    "sourceAccessToken": "<source-access-token>",
    "tunnelArn": "arn:aws:iot:<region>:<account-id>:tunnel/<tunnel-id>"
}
```

Sie können jetzt das neue Quell-Token verwenden, um den lokalen Proxy im Quellmodus zu verbinden.

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=<source-access-token>
./localproxy -r <region> -s <port>
```

Im Folgenden finden Sie ein Beispiel für die Ausgabe der Ausführung des lokalen Proxys:

```
...

[info]    Starting proxy in source mode
...
[info]    Successfully established websocket connection with proxy server ...
[info]    Listening for new connection on port <port>
...
```

Verbindungsprobleme bei Remote-Geräten

Bei Verwendung von AWS IoT Secure Tunneling wird das Gerät möglicherweise unerwartet getrennt, auch wenn der Tunnel geöffnet ist. Um festzustellen, ob ein Gerät noch mit dem Tunnel verbunden ist, können Sie die DescribeTunnel API oder den Describe-Tunnel verwenden. https://docs.aws.amazon.com/cli/latest/reference/iotsecuretunneling/describe-tunnel.html AWS CLI

Ein Gerät kann aus mehreren Gründen unterbrochen werden. Um das Verbindungsproblem zu beheben, können Sie das CAT auf das Zielgerät rotieren, falls die Verbindung zum Gerät aus den folgenden möglichen Gründen unterbrochen wurde:

- Das CAT auf dem Ziel wurde ungültig.
- Das Token wurde nicht über das für Secure Tunneling reservierte MQTT-Thema an das Gerät übermittelt:

  `$aws/things/<thing-name>/tunnels/notify`

Das folgende Beispiel veranschaulicht, wie dieses Problem gelöst werden kann:

Beispiel für die Rotation des Ziel-CAT

Man nehme ein Remote-Gerät, *<RemoteThing1>*. Um einen Tunnel für dieses Objekt zu öffnen, können Sie den folgenden Befehl verwenden:

```
aws iotsecuretunneling open-tunnel \
    --region <region> \
    --destination-config thingName=<RemoteThing1>,services=SSH
```

Wenn Sie diesen Befehl ausführen, werden die Tunneldetails und das CAT für Ihre Quelle und Ihr Ziel generiert.

```
{
    "sourceAccessToken": "<source-access-token>",
    "destinationAccessToken": "<destination-access-token>",
    "tunnelId": "<tunnel-id>",
    "tunnelArn": "arn:aws:iot:<region>:<account-id>:tunnel/tunnel-id"
}
```

Wenn Sie die DescribeTunnel API verwenden, zeigt die Ausgabe jedoch an, dass das Gerät getrennt wurde, wie unten dargestellt:

```
aws iotsecuretunneling describe-tunnel \
    --tunnel-id <tunnel-id> \
    --region <region>
```

Wenn Sie diesen Befehl ausführen, wird angezeigt, dass das Gerät immer noch nicht verbunden ist.

```
{
    "tunnel": {
        ...
        "destinationConnectionState": {
            "status": "DISCONNECTED"
        },
        ...
    }
}
```

Um diesen Fehler zu beheben, führen Sie die `RotateTunnelAccessToken` API mit dem Client im `DESTINATION`-Modus und den Konfigurationen für das Ziel aus. Wenn Sie diesen Befehl ausführen, wird das alte Zugriffstoken gesperrt, ein neues Token generiert und dieses Token erneut an das MQTT-Thema gesendet:

`$aws/things/<thing-name>/tunnels/notify`

```
aws iotsecuretunneling rotate-tunnel-access-token \
    --tunnel-id <tunnel-id> \
    --client-mode DESTINATION \
    --destination-config thingName=<RemoteThing1>,services=SSH \
    --region <region>
```

Durch die Ausführung dieses Befehls wird das neue Zugriffstoken wie unten dargestellt generiert. Das Token wird dann an das Gerät übermittelt, um eine Verbindung zum Tunnel herzustellen, sofern der Geräteagent korrekt eingerichtet ist.

```
{
    "destinationAccessToken": "destination-access-token",
    "tunnelArn": "arn:aws:iot:region:account-id:tunnel/tunnel-id"
}
```

Gerätebereitstellung

AWS bietet verschiedene Möglichkeiten, ein Gerät bereitzustellen und eindeutige Client-Zertifikate darauf zu installieren. In diesem Abschnitt werden die einzelnen Möglichkeiten beschrieben und erläutert, wie Sie die beste für Ihre IoT-Lösung auswählen. Diese Optionen werden im Whitepaper mit dem Titel Geräteherstellung und Bereitstellung von X.509 Zertifikaten in ausführlich beschrieben. AWS IoT Core

Wählen Sie die für Sie geeignete Option aus.

- Sie können Zertifikate auf IoT-Geräten installieren, bevor sie geliefert werden.

  Wenn Sie eindeutige Client-Zertifikate sicher auf Ihren IoT-Geräten installieren können, bevor sie an die Endbenutzer geliefert werden, sollten Sie das Just-in-Time-Provisioning (JITP) oder die Just-in-Time-Registrierung (JITR) verwenden.

  Mithilfe von JITP und JITR wird die Zertifizierungsstelle (CA), die zum Signieren des Gerätezertifikats verwendet wird, bei der das Gerät zum ersten Mal eine Verbindung herstellt, registriert AWS IoT und erkannt AWS IoT . Das Gerät wird AWS IoT bei seiner ersten Verbindung mithilfe der Details seiner Bereitstellungsvorlage bereitgestellt.

  Weitere Informationen zur Single-Thing-, JITP-, JITR- und Massenbereitstellung von Geräten mit eindeutigen Zertifikaten finden Sie unter the section called “Bereitstellen von Geräten mit Gerätezertifikaten”.
- Endbenutzer oder Installateure können eine App verwenden, um Zertifikate auf ihren IoT-Geräten zu installieren.

  Wenn Sie eindeutige Client-Zertifikate nicht sicher auf Ihrem IoT-Gerät installieren können, bevor sie an die Endbenutzer geliefert werden, die Endbenutzer oder Installateure jedoch eine App verwenden können, um die Geräte zu registrieren und die eindeutigen Gerätezertifikate zu installieren, sollten Sie die Bereitstellung durch vertrauenswürdige Benutzer verwenden.

  Die Verwendung von vertrauenswürdigen Benutzern, z. B. Endbenutzern oder Installateuren mit bekanntem Konto, kann den Geräteherstellungsprozess vereinfachen. Anstelle eines eindeutigen Client-Zertifikats verfügen Geräte über ein temporäres Zertifikat, mit dem das Gerät nur 5 Minuten AWS IoT lang eine Verbindung herstellen kann. Während dieses 5-minütigen Zeitfensters erhalten vertrauenswürdige Benutzer ein eindeutiges Client-Zertifikat mit einer längeren Lebensdauer, das

sie auf dem Gerät installieren können. Durch die begrenzte Gültigkeitsdauer des Antragszertifikats wird das Risiko kompromittierter Zertifikate minimiert.

Weitere Informationen finden Sie unter the section called “Bereitstellung durch vertrauenswürdigen Benutzer”.

- Endbenutzer können KEINE App verwenden, um Zertifikate auf ihren IoT-Geräten zu installieren.

  Wenn keine der vorherigen Optionen in Ihrer IoT-Lösung funktioniert, ist für Sie die Bereitstellung durch Anspruch eine Option. Mit diesem Prozess verfügen Ihre IoT-Geräte über ein Antragszertifikat, das gemeinsam von anderen Geräten in der Flotte genutzt wird. Wenn ein Gerät zum ersten Mal eine Verbindung mit einem Anspruchszertifikat herstellt, AWS IoT registriert es das Gerät anhand seiner Bereitstellungsvorlage und stellt dem Gerät sein eindeutiges Client-Zertifikat für den späteren Zugriff AWS IoT aus.

  Diese Option ermöglicht die automatische Bereitstellung eines Geräts, wenn es eine Verbindung herstellt AWS IoT, kann jedoch ein größeres Risiko darstellen, wenn das Anspruchszertifikat kompromittiert wird. Wenn ein Antragszertifikat kompromittiert wird, können Sie das Zertifikat deaktivieren. Durch die Deaktivierung des Antragszertifikats wird verhindert, dass in Zukunft Geräte mit diesem Antragszertifikat registriert werden. Durch die Deaktivierung des Antragszertifikats werden jedoch keine Geräte blockiert, die bereits bereitgestellt wurden.

  Weitere Informationen finden Sie unter the section called “Bereitstellung durch Anspruch”.

Bereitstellung von Geräten in AWS IoT

Wenn Sie ein Gerät mit bereitstellen AWS IoT, müssen Sie Ressourcen erstellen, damit Ihre Geräte und ich sicher kommunizieren AWS IoT können. Weitere Ressourcen können erstellt werden, um Ihnen bei der Verwaltung Ihrer Geräteflotte zu helfen. Die folgenden Ressourcen können während des Bereitstellungsprozesses erstellt werden:

- Ein IoT-Objekt

  IoT-Dinge sind Einträge in der AWS IoT Geräteregistrierung. Jedes Objekt hat einen eindeutigen Namen und einen Satz von Attributen und ist mit einem physischen Gerät verbunden. Objekte können mit einem Objekttyp definiert oder in Objektgruppen gruppiert werden. Weitere Informationen finden Sie unter Geräte verwalten mit AWS IoT.

Obwohl nicht erforderlich, ermöglicht es das Erstellen eines Objekts, Ihre Geräteflotte effektiver zu verwalten, indem Sie Geräte nach Objekttyp, Objektgruppe und Objektattributen suchen. Weitere Informationen finden Sie unter -Flottenindizierung.

> **Note**
>
> Um die Verbindungsstatusdaten Ihres Dings zu indizieren, stellen Sie Ihr Ding bereit und konfigurieren Sie es so, dass der Name des Dings mit der Client-ID übereinstimmt, die in der Connect-Anfrage verwendet wurde.

- Ein X.509 Zertifikat.

  Geräte verwenden X.509 Zertifikate, um sich gegenseitig zu authentifizieren AWS IoT. Sie können ein vorhandenes Zertifikat registrieren oder ein neues Zertifikat für Sie AWS IoT generieren und registrieren lassen. Sie ordnen einem Gerät ein Zertifikat zu, indem Sie es an das Objekt anhängen, das für das Gerät steht. Sie müssen auch das Zertifikat und den zugehörigen privaten Schlüssel auf das Gerät kopieren. Geräte legen das Zertifikat vor, wenn sie eine Verbindung zu herstellen AWS IoT. Weitere Informationen finden Sie unter Authentifizierung.

- Eine IoT-Richtlinie.

  IoT-Richtlinien definieren, welche Operationen ein Gerät in AWS IoT ausführen kann. IoT-Richtlinien sind an Gerätezertifikate angehängt. Wenn ein Gerät das Zertifikat vorlegt AWS IoT, werden ihm die in der Richtlinie angegebenen Berechtigungen gewährt. Weitere Informationen finden Sie unter Autorisierung. Jedes Gerät benötigt ein Zertifikat für die Kommunikation mit AWS IoT.

AWS IoT unterstützt die automatische Flottenbereitstellung mithilfe von Bereitstellungsvorlagen. Bereitstellungsvorlagen beschreiben die Ressourcen, die für die Bereitstellung AWS IoT Ihres Geräts erforderlich sind. Vorlagen enthalten Variablen, mit denen Sie eine Vorlage zum Bereitstellen mehrerer Geräte verwenden können. Wenn Sie ein Gerät bereitstellen, geben Sie Werte für die für das Gerät spezifischen Variablen mithilfe eines Wörterbuchs oder einer Karte an. Um ein anderes Gerät bereitzustellen, geben Sie neue Werte im Wörterbuch an.

Sie können die automatisierte Bereitstellung verwenden, unabhängig davon, ob Ihre Geräte über eindeutige Zertifikate (und den zugehörigen privaten Schlüssel) verfügen.

Flottenbereitstellungs-APIs

Es gibt mehrere Kategorien von APIs, die bei der Flottenbereitstellung verwendet werden:

- Diese Funktionen der Steuerungsebene erstellen und verwalten die Flottenbereitstellungsvorlagen und konfigurieren vertrauenswürdige Benutzerrichtlinien.
  - CreateProvisioningTemplate
  - CreateProvisioningTemplateVersion
  - DeleteProvisioningTemplate
  - DeleteProvisioningTemplateVersion
  - DescribeProvisioningTemplate
  - DescribeProvisioningTemplateVersion
  - ListProvisioningTemplates
  - ListProvisioningTemplateVersions
  - UpdateProvisioningTemplate
- Vertrauenswürdige Benutzer können diese Funktion der Steuerungsebene verwenden, um einen temporären Onboarding-Antrag zu generieren. Dieser temporäre Anspruch wird während der Wi-Fi Konfiguration oder einer ähnlichen Methode an das Gerät weitergegeben.
  - CreateProvisioningClaim
- Die MQTT-API, die während des Bereitstellungsprozesses von Geräten verwendet wird, deren Bereitstellungsantragszertifikat in einem Gerät eingebettet ist oder von einem vertrauenswürdigen Benutzer an es übergeben wird.
  - the section called “CreateCertificateFromCsr”
  - the section called “CreateKeysAndCertificate”
  - the section called “RegisterThing”

Bereitstellen von Geräten ohne Gerätezertifikate mithilfe der Flottenbereitstellung

Mithilfe der AWS IoT Flottenbereitstellung AWS IoT können Gerätezertifikate und private Schlüssel generiert und sicher an Ihre Geräte übertragen werden, wenn diese AWS IoT zum ersten Mal eine Verbindung herstellen. AWS IoT stellt Client-Zertifikate bereit, die von der Amazon Root Certificate Authority (CA) signiert sind.

Es gibt zwei Möglichkeiten, die Flottenbereitstellung zu verwenden:

- Bereitstellung durch Anspruch
- Bereitstellung durch vertrauenswürdigen Benutzer

Bereitstellung durch Anspruch

Geräte können mit einem Bereitstellungsantragszertifikat und einem privaten Schlüssel (bei denen es sich um spezielle Anmeldeinformationen handelt) hergestellt werden. Wenn diese Zertifikate registriert sind AWS IoT, kann der Service sie gegen eindeutige Gerätezertifikate eintauschen, die das Gerät für den regulären Betrieb verwenden kann. Dieser Prozess umfasst die folgenden Schritte:

Vor der Geräteauslieferung

1. Rufen Sie `CreateProvisioningTemplate` auf, um eine Bereitstellungsvorlage zu erstellen. Diese API gibt einen Vorlagen-ARN zurück. Weitere Informationen finden Sie unter MQTT-API für die Gerätebereitstellung.

   Sie können in der AWS IoT Konsole auch eine Vorlage für die Flottenbereitstellung erstellen.

   a. Wählen Sie im Navigationsbereich das Drop-down-Menü Viele Geräte verbinden aus. Wählen Sie dann Viele Geräte verbinden aus.
   b. Wählen Sie Bereitstellungsvorlage erstellen aus.
   c. Wählen Sie das Bereitstellungsszenario, das am besten zu Ihren Installationsprozessen passt. Wählen Sie anschließend Weiter.
   d. Vervollständigen Sie den Vorlagen-Workflow.
2. Erstellen Sie Zertifikate und zugehörige private Schlüssel, die als Bereitstellungsantragszertifikate verwendet werden sollen.
3. Registrieren Sie diese Zertifikate AWS IoT und verknüpfen Sie sie mit einer IoT-Richtlinie, die die Verwendung der Zertifikate einschränkt. Die folgende IoT-Beispielrichtlinie beschränkt die Verwendung des Zertifikats, das dieser Richtlinie zugeordnet ist, auf Bereitstellungsgeräte.

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

```
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topic/$aws/certificates/create/
*",
    "arn:aws:iot:us-east-1:123456789012:topic/$aws/provisioning-
templates/templateName/provision/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/certificates/
create/*",
    "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/provisioning-
templates/templateName/provision/*"
            ]
        }
    ]
}
```

4. Erteilen Sie dem AWS IoT Dienst bei der Bereitstellung von Geräten die Erlaubnis, IoT-Ressourcen wie Dinge und Zertifikate in Ihrem Konto zu erstellen oder zu aktualisieren. Hängen Sie dazu die `AWSIoTThingsRegistration` verwaltete Richtlinie an eine IAM-Rolle (die so genannte Bereitstellungsrolle) an, die dem Service Principal vertraut. AWS IoT
5. Stellen Sie das Gerät mit dem sicher darin eingebetteten Bereitstellungsantragszertifikat her.

Das Gerät kann nun an den Ort geliefert werden, an dem es zur Verwendung installiert wird.

> **Important**
>
> Private Schlüssel für Bereitstellungsanträge sollten jederzeit gesichert werden, auch auf dem Gerät. Wir empfehlen, mithilfe von AWS IoT CloudWatch Metriken und Protokollen nach Hinweisen auf Missbrauch zu suchen. Wenn Sie einen Missbrauch feststellen, deaktivieren Sie das Bereitstellungsantragszertifikat, damit es nicht für die Gerätebereitstellung verwendet werden kann.

So initialisieren Sie das Gerät für die Verwendung

1. Das Gerät verwendet dasAWS IoT Geräte-SDKs, SDKs für Mobilgeräte und AWS IoT Geräte-Client, um eine Verbindung AWS IoT mit dem auf dem Gerät installierten Provisioning Claim Certificate herzustellen und sich mit diesem zu authentifizieren.

   > **Note**
   >
   > Aus Sicherheitsgründen wird das `certificateOwnershipToken` von `CreateCertificateFromCsr` zurückgesendet und `CreateKeysAndCertificate` läuft nach einer Stunde ab. `RegisterThing` muss aufgerufen werden, bevor das `certificateOwnershipToken` abläuft. Wenn das von `CreateCertificateFromCsr` oder `CreateKeysAndCertificate` erstellte Zertifikat nicht aktiviert und bis zum Ablauf des Tokens noch nicht an eine Richtlinie oder ein Objekt angehängt wurde, wird das Zertifikat gelöscht. Wenn das Token abläuft, kann das Gerät `CreateCertificateFromCsr` oder `CreateKeysAndCertificate` erneut aufrufen, um ein neues Zertifikat zu generieren.

2. Das Gerät erhält ein permanentes Zertifikat und einen privaten Schlüssel mithilfe einer dieser Optionen. Das Gerät verwendet das Zertifikat und den Schlüssel für alle zukünftigen Authentifizierungen mit. AWS IoT

   a. Rufen Sie `CreateKeysAndCertificate` an, um mithilfe der Zertifizierungsstelle ein neues Zertifikat und einen privaten Schlüssel zu erstellen. AWS

      Oder

   b. Rufen Sie `CreateCertificateFromCsr` auf, um ein Zertifikat aus einer Zertifikatsignieranforderung zu generieren, das seinen privaten Schlüssel schützt.

3. Rufen Sie vom Gerät aus `RegisterThing` auf, um das Gerät in AWS IoT zu registrieren und Cloud-Ressourcen zu erstellen.

   Der Fleet-Provisioning-Service verwendet Bereitstellungsvorlagen, um Cloud-Ressourcen wie IoT-Objekte zu definieren und zu erstellen. Die Vorlage kann Attribute und Gruppen angeben, denen das Objekt angehört. Die Objektgruppen müssen vorhanden sein, damit das neue Objekt hinzugefügt werden kann.
4. Nach dem Speichern des permanenten Zertifikats auf dem Gerät muss das Gerät die Verbindung von der Sitzung trennen, die es mit dem Bereitstellungsantragszertifikat initiiert hat, und die Verbindung mit dem permanenten Zertifikat erneut herstellen.

Das Gerät ist jetzt bereit, normal mit ihm zu kommunizieren AWS IoT.

Bereitstellung durch vertrauenswürdigen Benutzer

In vielen Fällen stellt ein Gerät AWS IoT zum ersten Mal eine Verbindung her, wenn ein vertrauenswürdiger Benutzer, z. B. ein Endbenutzer oder Installationstechniker, eine mobile App verwendet, um das Gerät an seinem Einsatzort zu konfigurieren.

> **Important**
>
> Sie müssen den Zugriff und die Berechtigung des vertrauenswürdigen Benutzers verwalten, um dieses Verfahren auszuführen. Eine Möglichkeit besteht darin, vertrauenswürdigen Benutzern ein Konto bereitzustellen und für sie zu verwalten, mit dem sie authentifiziert werden und Zugriff auf die AWS IoT -Funktionen und API-Operationen erhalten, die für dieses Vorgehen erforderlich sind.

Vor der Geräteauslieferung

1. Rufen Sie `CreateProvisioningTemplate` auf, um eine Bereitstellungsvorlage zu erstellen und deren *`templateArn`* and *`templateName`* zurückzugeben.
2. Erstellen Sie eine IAM-Rolle, die von vertrauenswürdigen Benutzern verwendet wird, um den Bereitstellungsprozess zu initiieren. Mit der Bereitstellungsvorlage kann nur dieser Benutzer ein Gerät bereitstellen. Beispiel:

   ```
   {
   ```

```
    "Effect": "Allow",
    "Action": [
        "iot:CreateProvisioningClaim"
    ],
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:provisioningtemplate/templateName"
    ]
}
```

3. Erteilen Sie dem AWS IoT Dienst die Erlaubnis, IoT-Ressourcen zu erstellen oder zu aktualisieren, z. B. Dinge und Zertifikate in Ihrem Konto, wenn Sie Geräte bereitstellen. Dazu fügen Sie die `AWSIoTThingsRegistration` verwaltete Richtlinie einer IAM-Rolle (der so genannten Bereitstellungsrolle) hinzu, die dem Service Principal vertraut. AWS IoT
4. Stellen Sie die Mittel bereit, um Ihre vertrauenswürdigen Benutzer zu identifizieren, indem Sie ihnen beispielsweise ein Konto zur Verfügung stellen, das sie authentifizieren und ihre Interaktionen mit den AWS API-Vorgängen autorisieren kann, die für die Registrierung ihrer Geräte erforderlich sind.

So initialisieren Sie das Gerät für die Verwendung

1. Der vertrauenswürdige Benutzer meldet sich bei der mobilen App oder dem Webservice für die Bereitstellung an.
2. Die mobile App oder Webanwendung verwendet die IAM-Rolle und ruft `CreateProvisioningClaim` auf, um ein temporäres Bereitstellungsantragszertifikat von AWS IoT abzurufen.

> **Note**
>
> Aus Sicherheitsgründen läuft das von `CreateProvisioningClaim` zurückgegebene temporäre Bereitstellungsantragszertifikat nach fünf Minuten ab. Die folgenden Schritte müssen erfolgreich ein gültiges Zertifikat zurückgeben, bevor das temporäre Bereitstellungsantragszertifikat abläuft. Temporäre Bereitstellungsantragszertifikate werden nicht in der Zertifikatliste Ihres Kontos angezeigt.

3. Die mobile App oder Webanwendung stellt dem Gerät das temporäre Bereitstellungszertifikat zusammen mit allen erforderlichen Konfigurationsinformationen wie Anmeldeinformationen zur Verfügung. Wi-Fi

4. Das Gerät verwendet das temporäre Bereitstellungsantragszertifikat, um eine Verbindung zu AWS IoT mit dem AWS IoT Geräte-SDKs, SDKs für Mobilgeräte und AWS IoT Geräte-Client herzustellen.
5. Das Gerät erhält mithilfe einer dieser Optionen innerhalb von fünf Minuten nach der Verbindung AWS IoT mit dem temporären Bereitstellungsanforderungszertifikat ein permanentes Zertifikat und einen privaten Schlüssel. Das Gerät verwendet das Zertifikat und den Schlüssel, mit dem diese Optionen für alle zukünftigen Authentifizierungen zurückgegeben werden. AWS IoT

   a. Rufen Sie `CreateKeysAndCertificate` auf, um mithilfe der Zertifizierungsstelle ein neues Zertifikat und einen privaten Schlüssel zu erstellen. AWS

      Oder

   b. Rufen Sie `CreateCertificateFromCsr` auf, um ein Zertifikat aus einer Zertifikatsignieranforderung zu generieren, das seinen privaten Schlüssel schützt.

   > **Note**
   >
   > Denken Sie daran, `CreateKeysAndCertificate` oder Sie `CreateCertificateFromCsr` müssen innerhalb von fünf Minuten nach der Verbindung AWS IoT mit dem temporären Bereitstellungszertifikat ein gültiges Zertifikat zurückgeben.

6. Das Gerät ruft `RegisterThing` an, um das Gerät bei Cloud-Ressourcen zu registrieren AWS IoT und diese zu erstellen.

   Der Fleet-Provisioning-Service verwendet Bereitstellungsvorlagen, um Cloud-Ressourcen wie IoT-Objekte zu definieren und zu erstellen. Die Vorlage kann Attribute und Gruppen angeben, denen das Objekt angehört. Die Objektgruppen müssen vorhanden sein, damit das neue Objekt hinzugefügt werden kann.
7. Nach dem Speichern des permanenten Zertifikats auf dem Gerät muss das Gerät die Verbindung von der Sitzung trennen, die es mit dem temporären Bereitstellungsantragszertifikat initiiert hat, und erneut eine Verbindung mit dem permanenten Zertifikat herstellen.

Das Gerät ist jetzt bereit, normal mit ihm zu kommunizieren AWS IoT.

Verwenden von Pre-Provisioning-Hooks mit dem AWS CLI

Im folgenden Verfahren wird eine Bereitstellungsvorlage mit Pre-Provisioning-Hooks erstellt. Die hier verwendete Lambda-Funktion ist ein Beispiel, das geändert werden kann.

So erstellen Sie Pre-Provisioning-Hooks und übernehmen Sie für eine Bereitstellungsvorlage

1. Erstellen Sie eine Lambda-Funktion mit definierter Eingabe und Ausgabe. Lambda-Funktionen sind hochgradig anpassbar. Für die Erstellung von Pre-Provisioning-Hooks sind `allowProvisioning` und `parameterOverrides` erforderlich. Weitere Informationen zum Erstellen von Lambda-Funktionen finden Sie unter Verwenden AWS Lambda mit der AWS Befehlszeilenschnittstelle.

   Im Folgenden sehen Sie ein Beispiel für eine Lambda-Funktionsausgabe:

   ```
   {
     "allowProvisioning": True,
     "parameterOverrides": {
       "incomingKey0": "incomingValue0",
       "incomingKey1": "incomingValue1"
     }
   }
   ```

2. AWS IoT verwendet ressourcenbasierte Richtlinien, um Lambda aufzurufen. Sie müssen also die AWS IoT Erlaubnis erteilen, Ihre Lambda-Funktion aufzurufen.

   > **Important**
   >
   > Achten Sie darauf, das `source-arn` oder `source-account` in die globalen Bedingungskontextschlüssel der Richtlinien aufzunehmen, die mit Ihrer Lambda-Aktion verknüpft sind, damit Berechtigungen nicht manipuliert werden können. Weitere Informationen hierzu finden Sie unter Cross-service verwirrter Stellvertreter, Prävention.

   Es folgt ein Beispiel dafür, wie Sie IoT mit add-permission die Berechtigung zum Aufrufen Ihrer Lambda-Funktion erteilen.

   ```
   aws lambda add-permission \
       --function-name myLambdaFunction \
       --statement-id iot-permission \
   ```

```
    --action lambda:InvokeFunction \
    --principal iot.amazonaws.com
```

3. Fügen Sie einer Vorlage mit dem Befehl create-provisioning-template oder update-provisioning-template einen Pre-Provisioning-Hook hinzu.

   Im folgenden CLI-Beispiel wird create-provisioning-template verwendet, um eine Bereitstellungsvorlage zu erstellen, die Pre-Provisioning-Hooks enthält:

```
aws iot create-provisioning-template \
    --template-name myTemplate \
    --provisioning-role-arn arn:aws:iam:us-east-1:1234564789012:role/myRole \
    --template-body file://template.json \
    --pre-provisioning-hook file://hooks.json
```

   Die Ausgabe dieses Befehls sieht wie folgt aus:

```
{
    "templateArn": "arn:aws:iot:us-east-1:1234564789012:provisioningtemplate/
myTemplate",
    "defaultVersionId": 1,
    "templateName": myTemplate
}
```

   Sie können einen Parameter auch aus einer Datei laden, statt ihn als Befehlszeilen-Parameterwert einzugeben, um Zeit zu sparen. Weitere Informationen finden Sie unter Laden von AWS CLI -Parametern aus einer Datei. Im Folgenden wird der Parameter `template` im erweiterten JSON-Format gezeigt:

```
{
    "Parameters" : {
        "DeviceLocation": {
            "Type": "String"
        }
    },
    "Mappings": {
        "LocationTable": {
            "Seattle": {
                "LocationUrl": "https://example.aws"
            }
        }
```

```
    },
    "Resources" : {
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "AttributePayload" : {
                    "version" : "v1",
                    "serialNumber" : "serialNumber"
                },
                "ThingName" : {"Fn::Join":["",["ThingPrefix_",
{"Ref":"SerialNumber"}]]},
                "ThingTypeName" : {"Fn::Join":["",["ThingTypePrefix_",
{"Ref":"SerialNumber"}]]},
                "ThingGroups" : ["widgets", "WA"],
                "BillingGroup": "BillingGroup"
            },
            "OverrideSettings" : {
                "AttributePayload" : "MERGE",
                "ThingTypeName" : "REPLACE",
                "ThingGroups" : "DO_NOTHING"
            }
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
            "Properties" : {
                "CertificateId": {"Ref": "AWS::IoT::Certificate::Id"},
                "Status" : "Active",
                "ThingPrincipalType" : "EXCLUSIVE_THING"
            }
        },
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : {
                    "Version": "2012-10-17",
                    "Statement": [{
                        "Effect": "Allow",
                        "Action":["iot:Publish"],
                        "Resource": ["arn:aws:iot:us-east-1:504350838278:topic/foo/
bar"]
                    }]
                }
            }
        }
```

```
    },
    "DeviceConfiguration": {
        "FallbackUrl": "https://www.example.com/test-site",
        "LocationUrl": {
            "Fn::FindInMap": ["LocationTable",{"Ref": "DeviceLocation"},
 "LocationUrl"]}
    }
}
```

Im Folgenden wird der Parameter `pre-provisioning-hook` im erweiterten JSON-Format gezeigt:

```
{
     "targetArn" : "arn:aws:lambda:us-
east-1:765219403047:function:pre_provisioning_test",
     "payloadVersion" : "2020-04-01"
}
```

Bereitstellen von Geräten mit Gerätezertifikaten

AWS IoT bietet drei Möglichkeiten, Geräte bereitzustellen, auf denen bereits ein Gerätezertifikat (und der zugehörige private Schlüssel) installiert sind:

- Single-thing Bereitstellung mit einer Bereitstellungsvorlage. Diese Option eignet sich gut, wenn Sie Geräte nur einzeln bereitstellen müssen.
- Just-in-time Bereitstellung (JITP) mit einer Vorlage, die ein Gerät bereitstellt, wenn es zum ersten Mal eine Verbindung herstellt. AWS IoT Diese Option eignet sich gut, wenn Sie eine große Anzahl von Geräten anmelden müssen, jedoch keine Informationen über sie besitzen, die Sie als Massenbereitstellungsliste zusammenstellen können.
- Massenregistrierung. Mit dieser Option können Sie eine Liste von Bereitstellungsvorlagenwerten für einzelne Objekte angeben, die in einer Datei in einem S3-Bucket gespeichert werden. Dieser Ansatz eignet sich gut, wenn eine große Anzahl bekannter Geräte vorhanden ist, deren gewünschten Merkmale in einer Liste zusammengestellt werden können.

Themen

- Bereitstellung eines einzelnen Objekts
- Just-in-time Bereitstellung

- Massenregistrierung

Bereitstellung eines einzelnen Objekts

Um eine Sache bereitzustellen, verwenden Sie die RegisterThing API oder den `register-thing` CLI-Befehl. Der CLI-Befehl `register-thing` verwendet die folgenden Argumente:

--template-body

Die Bereitstellungsvorlage.

--parameters

Eine Liste von Name/Wert-Paaren für die in der Bereitstellungsvorlage verwendeten Parameter im JSON-Format (z. B. `{"ThingName" : "MyProvisionedThing", "CSR" : "csr-text"}`).

Siehe Bereitstellungsvorlagen.

RegisterThingoder `register-thing` gibt die ARNs für die Ressourcen und den Text des von ihr erstellten Zertifikats zurück:

```
{
    "certificatePem": "certificate-text",
    "resourceArns": {
    "PolicyLogicalName": "arn:aws:iot:us-
west-2:123456789012:policy/2A6577675B7CD1823E271C7AAD8184F44630FFD7",
    "certificate": "arn:aws:iot:us-west-2:123456789012:cert/
cd82bb924d4c6ccbb14986dcb4f40f30d892cc6b3ce7ad5008ed6542eea2b049",
    "thing": "arn:aws:iot:us-west-2:123456789012:thing/MyProvisionedThing"
    }
}
```

Wenn ein Parameter im Lexikon weggelassen wird, wird der Standardwert verwendet. Wenn kein Standardwert angegeben ist, wird der Parameter nicht durch einen Wert ersetzt.

Just-in-time Bereitstellung

Sie können Just-in-Time-Provisioning (JITP) verwenden, um Ihre Geräte bereitzustellen, wenn sie zum ersten Mal versuchen, eine Verbindung mit AWS IoT herzustellen. Zur Bereitstellung des Geräts müssen Sie die automatische Registrierung aktivieren und dem CA-Zertifikat, mit dem das

Gerätezertifikat signiert wurde, eine Bereitstellungsvorlage zuordnen. Erfolge und Fehler bei der Bereitstellung werden wie Gerätebereitstellungsmetriken bei Amazon protokolliert. CloudWatch

Themen

- Überblick über JITP
- Registrieren einer CA mithilfe der Bereitstellungsvorlage
- Registrieren einer CA anhand des Bereitstellungsvorlagennamens

Überblick über JITP

Wenn ein Gerät versucht, mithilfe eines AWS IoT von einem registrierten CA-Zertifikat signierten Zertifikats eine Verbindung herzustellen, wird die Vorlage aus dem CA-Zertifikat AWS IoT geladen und für den Aufruf RegisterThing verwendet. Der JITP-Workflow registriert zuerst ein Zertifikat mit dem Statuswert `PENDING_ACTIVATION`. Wenn die Gerätebereitstellung abgeschlossen ist, wird der Status des Zertifikats in `ACTIVE` geändert.

AWS IoT definiert die folgenden Parameter, die Sie in Bereitstellungsvorlagen deklarieren und auf die Sie verweisen können:

- `AWS::IoT::Certificate::Country`
- `AWS::IoT::Certificate::Organization`
- `AWS::IoT::Certificate::OrganizationalUnit`
- `AWS::IoT::Certificate::DistinguishedNameQualifier`
- `AWS::IoT::Certificate::StateName`
- `AWS::IoT::Certificate::CommonName`
- `AWS::IoT::Certificate::SerialNumber`
- `AWS::IoT::Certificate::Id`

Die Werte für diese Bereitstellungsvorlagenparameter werden auf die Angaben beschränkt, die JITP aus dem Betreff-Feld des Zertifikats des bereitzustellenden Geräts extrahieren kann. Das Zertifikat muss Werte für alle Parameter im Vorlagentext enthalten. Der `AWS::IoT::Certificate::Id`-Parameter bezieht sich auf eine intern generierte ID und nicht auf eine ID, die im Zertifikat enthalten ist. Sie können den Wert dieser ID mithilfe der `principal()` Funktion in einer AWS IoT Regel abrufen.

> **Note**
>
> Sie können Geräte mithilfe der AWS IoT Core Just-in-Time-Provisioning (JITP) -Funktion bereitstellen, ohne die gesamte Vertrauenskette bei der ersten Verbindung eines Geräts an senden zu müssen. AWS IoT Core Die Vorlage des CA-Zertifikats ist optional, aber das Gerät muss die SNI-Erweiterung (Server Name Indication) senden, wenn es eine Verbindung mit AWS IoT Core herstellt.

Beispielvorlagentext

Die folgende JSON-Datei ist ein Beispieltext für eine vollständige JITP-Vorlage.

```
{
   "Parameters":{
      "AWS::IoT::Certificate::CommonName":{
         "Type":"String"
      },
      "AWS::IoT::Certificate::SerialNumber":{
         "Type":"String"
      },
      "AWS::IoT::Certificate::Country":{
         "Type":"String"
      },
      "AWS::IoT::Certificate::Id":{
         "Type":"String"
      }
   },
   "Resources":{
      "thing":{
         "Type":"AWS::IoT::Thing",
         "Properties":{
            "ThingName":{
               "Ref":"AWS::IoT::Certificate::CommonName"
            },
            "AttributePayload":{
               "version":"v1",
               "serialNumber":{
                  "Ref":"AWS::IoT::Certificate::SerialNumber"
               }
            },
            "ThingTypeName":"lightBulb-versionA",
```

```
            "ThingGroups":[
               "v1-lightbulbs",
               {
                  "Ref":"AWS::IoT::Certificate::Country"
               }
            ]
         },
         "OverrideSettings":{
            "AttributePayload":"MERGE",
            "ThingTypeName":"REPLACE",
            "ThingGroups":"DO_NOTHING"
         }
      },
      "certificate":{
         "Type":"AWS::IoT::Certificate",
         "Properties":{
            "CertificateId":{
               "Ref":"AWS::IoT::Certificate::Id"
            },
            "Status":"ACTIVE"
         }
      },
      "policy":{
         "Type":"AWS::IoT::Policy",
         "Properties":{
            "PolicyDocument":"{ \"Version\": \"2012-10-17\",        \"Statement
\": [{ \"Effect\": \"Allow\", \"Action\":[\"iot:Publish\"], \"Resource\":
 [\"arn:aws:iot:us-east-1:123456789012:topic/foo/bar\"] }] }"
         }
      }
   }
}
```

Diese Beispielvorlage deklariert Werte für die Bereitstellungsparameter `AWS::IoT::Certificate::CommonName`, `AWS::IoT::Certificate::SerialNumber`, `AWS::IoT::Certificate::Country` und `AWS::IoT::Certificate::Id`, die aus dem Zertifikat extrahiert und im Abschnitt `Resources` verwendet werden. Der JITP-Workflow führt anhand dieser Vorlage die folgenden Aktionen aus:

- Ein Zertifikat registrieren und als seinen Status PENDING_ACTIVE einstellen.
- Eine Objekt-Ressource erstellen.
- Eine Richtlinien-Ressource erstellen.

- Dem Zertifikat die Richtlinie anfügen.
- Dem Objekt das Zertifikat anfügen.
- Den Status des Zertifikats auf ACTIVE aktualisieren.

Die Gerätebereitstellung schlägt fehl, wenn das Zertifikat nicht alle im Abschnitt von genannten Eigenschaften aufweist. `Parameters` `templateBody` Wenn `AWS::IoT::Certificate::Country` es beispielsweise in der Vorlage enthalten ist, das Zertifikat jedoch keine `Country`-Eigenschaft besitzt, schlägt die Gerätebereitstellung fehl.

Sie können es auch verwenden CloudTrail , um Probleme mit Ihrer JITP-Vorlage zu beheben. Informationen zu den Metriken, die bei Amazon protokolliert werden CloudWatch, finden Sie unterGerätebereitstellungsmetriken. Weitere Informationen zur Bereitstellungsvorlagen finden Sie unter Bereitstellungsvorlagen.

> **Note**
>
> Während des Bereitstellungsprozesses ruft die Just-in-Time-Bereitstellung (JITP) andere API-Operationen der Steuerungsebene auf. AWS IoT Diese Aufrufe können die für Ihr Konto festgelegten AWS IoT -Drosselungskontingente überschreiten und zu gedrosselten Aufrufen führen. Wenden Sie sich an den AWS -Kundenservice, um Ihre Drosselungskontingente bei Bedarf zu erhöhen.

Registrieren einer CA mithilfe der Bereitstellungsvorlage

Gehen Sie folgendermaßen vor, um eine CA mithilfe einer vollständigen Bereitstellungsvorlage zu registrieren:

1. Speichern Sie Ihre Bereitstellungsvorlage und die ARN-Informationen der Rolle wie im folgenden Beispiel als JSON-Datei:

```
{
     "templateBody" : "{\r\n    \"Parameters\" : {\r\n
 \"AWS::IoT::Certificate::CommonName\": {\r\n            \"Type\": \"String\"\r
\n        },\r\n        \"AWS::IoT::Certificate::SerialNumber\": {\r\n
  \"Type\": \"String\"\r\n        },\r\n        \"AWS::IoT::Certificate::Country
\": {\r\n            \"Type\": \"String\"\r\n        },\r\n
 \"AWS::IoT::Certificate::Id\": {\r\n            \"Type\": \"String\"\r
```

```
\n        }\r\n    },\r\n    \"Resources\": {\r\n        \"thing\": {\r
\n            \"Type\": \"AWS::IoT::Thing\",\r\n            \"Properties
\": {\r\n                \"ThingName\": {\r\n                    \"Ref\":
 \"AWS::IoT::Certificate::CommonName\"\r\n                },\r\n
     \"AttributePayload\": {\r\n                    \"version\": \"v1\",\r\n
                  \"serialNumber\": {\r\n                        \"Ref\":
 \"AWS::IoT::Certificate::SerialNumber\"\r\n                    }\r\n
     },\r\n                \"ThingTypeName\": \"lightBulb-versionA\",\r\n
        \"ThingGroups\": [\r\n                    \"v1-lightbulbs\",\r\n
           {\r\n                        \"Ref\": \"AWS::IoT::Certificate::Country
\"\r\n                    }\r\n                ]\r\n            },\r\n
  \"OverrideSettings\": {\r\n                \"AttributePayload\": \"MERGE\",\r\n
               \"ThingTypeName\": \"REPLACE\",\r\n                \"ThingGroups
\": \"DO_NOTHING\"\r\n            }\r\n        },\r\n        \"certificate\": {\r
\n            \"Type\": \"AWS::IoT::Certificate\",\r\n            \"Properties
\": {\r\n                \"CertificateId\": {\r\n                    \"Ref\":
 \"AWS::IoT::Certificate::Id\"\r\n                },\r\n                \"Status\":
 \"ACTIVE\"\r\n            },\r\n            \"OverrideSettings\": {\r\n
     \"Status\": \"DO_NOTHING\"\r\n            }\r\n        },\r\n        \"policy
\": {\r\n            \"Type\": \"AWS::IoT::Policy\",\r\n            \"Properties
\": {\r\n                \"PolicyDocument\": \"{ \\\"Version\\\": \\\"2012-10-17
     \\\", \\\"Statement\\\": [{ \\\"Effect\\\": \\\"Allow\\\", \\\"Action\\\":[\\
\"iot:Publish\\\"], \\\"Resource\\\": [\\\"arn:aws:iot:us-east-1:123456789012:topic
\/foo\/bar\\\"] }] }\"\r\n            }\r\n        }\r\n    }\r\n}",
     "roleArn" : "arn:aws:iam::123456789012:role/JITPRole"
}
```

In diesem Beispiel muss der Wert des Feldes `templateBody` ein JSON-Objekt sein, das als Zeichenfolge mit Escapezeichen angegeben wird. Zulässig sind nur die Werte in der obigen Liste. Sie können eine Vielzahl von Tools verwenden, um die erforderliche JSON-Ausgabe zu erstellen, z. B. `json.dumps` (Python) oder `JSON.stringify` (Node). Der Wert von Feld `roleARN` muss der ARN einer Rolle sein, der `AWSIoTThingsRegistration` zugeordnet ist. Außerdem können Sie in Ihrer Vorlage `PolicyName` anstelle des Inline-`PolicyDocument` wie im Beispiel verwenden.

2. Registrieren Sie ein CA-Zertifikat mit der API-Operation RegisterCACertificate oder dem `register-ca-certificate`-CLI-Befehl. Geben Sie das Verzeichnis der Bereitstellungsvorlage und die ARN-Informationen der Rolle an, die Sie im vorherigen Schritt gespeichert haben:

   Im Folgenden finden Sie ein Beispiel für die Registrierung eines CA-Zertifikats im `DEFAULT`-Modus mithilfe von AWS CLI:

```
aws iot register-ca-certificate --ca-certificate file://your-ca-cert --
verification-cert file://your-verification-cert
                --set-as-active --allow-auto-registration --registration-config
 file://your-template
```

Im Folgenden finden Sie ein Beispiel für die Registrierung eines CA-Zertifikats im SNI_ONLY-Modus mithilfe von AWS CLI:

```
aws iot register-ca-certificate --ca-certificate file://your-ca-cert --certificate-
mode SNI_ONLY
                 --set-as-active --allow-auto-registration --registration-config
 file://your-template
```

Weitere Informationen finden Sie unter Registrieren Ihres CA-Zertifikats.

3. (Optional) Aktualisieren Sie die Einstellungen für ein CA-Zertifikat mithilfe der API-Operation UpdateCACertificate oder des `update-ca-certificate`-CLI-Befehls.

   Im Folgenden finden Sie ein Beispiel für die Aktualisierung eines CA-Zertifikats mithilfe von AWS CLI:

```
aws iot update-ca-certificate --certificate-id caCertificateId
                --new-auto-registration-status ENABLE --registration-config
 file://your-template
```

Registrieren einer CA anhand des Bereitstellungsvorlagennamens

Gehen Sie folgendermaßen vor, um eine CA mit einem Bereitstellungsvorlagennamen zu registrieren:

1. Speichern Sie den Text Ihrer Bereitstellungsvorlage als JSON-Datei. Einen Beispielvorlagentext finden Sie unter Beispielvorlagentext.
2. Verwenden Sie die API oder den CLI-Befehl, um eine Bereitstellungsvorlage zu erstellen: CreateProvisioningTemplate `create-provisioning-template`

```
aws iot create-provisioning-template --template-name your-template-name \
       --template-body file://your-template-body.json --type JITP \
       --provisioning-role-arn arn:aws:iam::123456789012:role/test
```

> **Note**
>
> Für eine Just-in-Time-Bereitstellung (JITP) müssen Sie den Vorlagentyp auf `JITP` festlegen, der bei der Erstellung der Bereitstellungsvorlage verwendet werden soll. Weitere Informationen zum Vorlagentyp finden Sie CreateProvisioningTemplate in der AWS API-Referenz.

3. Registrieren Sie ein CA-Zertifikat mit der RegisterCACertificate-API oder dem `register-ca-certificate`-CLI-Befehl:

```
aws iot register-ca-certificate --ca-certificate file://your-ca-cert --
verification-cert file://your-verification-cert \
        --set-as-active --allow-auto-registration --registration-config
 templateName=your-template-name
```

Massenregistrierung

Mit dem Befehl `start-thing-registration-task` können Sie viele Objekte gemeinsam registrieren. Dieser Befehl verwendet eine Bereitstellungsvorlage, den Namen eines S3;-Buckets, einen Schlüsselnamen und einen Rollen-ARN, der den Zugriff auf die Datei in dem S3-Bucket ermöglicht. Die Datei in dem S3-Bucket enthält die Werte, die die Parameter in der Vorlage ersetzen. Dabei muss es sich um eine JSON-Datei mit Trennung durch neue Zeilen handeln. Jede Zeile enthält alle Parameterwerte für die Registrierung eines einzelnen Geräts. Beispiel:

```
{"ThingName": "foo", "SerialNumber": "123", "CSR": "csr1"}
{"ThingName": "bar", "SerialNumber": "456", "CSR": "csr2"}
```

Folgende API-Operationen können im Zusammenhang mit der Massenregistrierung nützlich sein:

- ListThingRegistrationTasks: Listet die aktuellen Aufgaben zur Bereitstellung von Massenobjekten auf.
- DescribeThingRegistrationTask: Stellt Informationen zu einer bestimmten Aufgabe zur Registrierung von Massenobjekten bereit.
- StopThingRegistrationTask: Stoppt eine Aufgabe zur Registrierung großer Dinger.
- ListThingRegistrationTaskReports: Wird verwendet, um die Ergebnisse und Fehler einer Aufgabe zur Registrierung von Massenobjekten zu überprüfen.

> **Note**
>
> - Es kann jeweils nur ein Massenregistrierungsvorgang gleichzeitig durchgeführt werden (pro Konto).
> - Bei der Massenregistrierung werden andere API-Operationen der AWS IoT Steuerungsebene aufgerufen. Diese Abrufe können die AWS IoT -Drosselungskontingente in Ihrem Konto überschreiten und Drosselungsfehler verursachen. Wenden Sie sich bei Bedarf an den AWS Kundensupport, um Ihre AWS IoT Drosselungsquoten zu erhöhen.

Bereitstellungsvorlagen

Eine Bereitstellungsvorlage ist ein JSON-Dokument, das mithilfe von Parametern die Ressourcen beschreibt, mit denen Ihr Gerät interagieren muss. AWS IoT Eine Bereitstellungsvorlage besteht aus zwei Abschnitten: `Parameters` und `Resources`. Es gibt zwei Arten von Bereitstellungsvorlagen in. AWS IoT Eine wird für die Just-in-Time-Bereitstellung (JITP) und die Massenregistrierung verwendet, und die zweite wird für die Flottenbereitstellung verwendet.

Themen

- Bereich "Parameters"
- Bereich „Ressourcen“
- Vorlagenbeispiel für eine Massenregistrierung
- Vorlagenbeispiel für eine Just-in-Time-Bereitstellung (JITP)
- Flottenbereitstellung

Bereich "Parameters"

Der Abschnitt `Parameters` deklariert die im Abschnitt `Resources` verwendeten Parameter. Jeder Parameter deklariert einen Namen, einen Typ und einen optionalen Standardwert. Der Standardwert wird verwendet, wenn das mit der Vorlage eingegebene Lexikon keinen Wert für den Parameter enthält. Der Abschnitt `Parameters` eines Vorlagendokuments sieht wie folgt aus:

```
{
    "Parameters" : {
        "ThingName" : {
            "Type" : "String"
```

```
        },
        "SerialNumber" : {
            "Type" : "String"
        },
        "Location" : {
            "Type" : "String",
            "Default" : "WA"
        },
        "CSR" : {
            "Type" : "String"
        }
    }
}
```

Dieses Vorlagentextfragment deklariert vier Parameter: `ThingName`, `SerialNumber`, `Location` und `CSR`. Alle diese Parameter sind vom Typ `String`. Der Parameter `Location` deklariert den Standardwert `"WA"`.

Bereich „Ressourcen“

Im `Resources` Abschnitt des Vorlagentexts werden die Ressourcen deklariert, mit denen Ihr Gerät kommunizieren muss AWS IoT: ein Ding, ein Zertifikat und eine oder mehrere IoT-Richtlinien. Jede Ressource gibt einen logischen Namen, einen Typ und eine Reihe von Eigenschaften an.

Ein logischer Name ermöglicht den Verweis auf eine Ressource an einer anderen Stelle in der Vorlage.

Der Typ gibt die Art der Ressource an, die Sie deklarieren. Gültige Typen sind:

- `AWS::IoT::Thing`
- `AWS::IoT::Certificate`
- `AWS::IoT::Policy`

Die Eigenschaften, die Sie angeben, hängen vom Typ der Ressource ab, die Sie deklarieren.

Objektressourcen

Objekt-Ressourcen werden mit den folgenden Eigenschaften deklariert:

- `ThingName`: Zeichenfolge.

- `AttributePayload`: Optional. Eine Liste von Name-Wert-Paaren.
- `ThingTypeName`: Optional. Zeichenfolge für einen zugehörigen Objekttyp für das Objekt.
- `ThingGroups`: Optional. Eine Liste der Gruppen, zu der das Objekt gehört.
- `BillingGroup`: Optional. Zeichenfolge für den Namen einer zugehörigen Fakturierungsgruppe.
- `PackageVersions`: Optional. Zeichenfolge für ein zugeordnetes Paket und Versionsnamen.

Zertifikatressourcen

Sie können Zertifikate auf eine der folgenden Arten angeben:

- Eine Zertifikatssignierungsanforderung (Certificate Signing Request, CSR).
- Eine Zertifikat-ID eines vorhandenen Geräte-Zertifikats. (Nur Zertifikats-IDs können mit einer Flottenbereitstellungsvorlage verwendet werden.)
- Ein Geräte-Zertifikat mit einem für AWS IoT registrierten CA-Zertifikat. Wenn Sie über mehr als ein mit demselben Themafeld registriertes CA-Zertifikat verfügen, müssen Sie auch das verwendete CA-Zertifikat eingeben, um das Geräte-Zertifikat zu signieren.

> **Note**
>
> Wenn Sie ein Zertifikat in einer Vorlage deklarieren, verwenden Sie nur eine dieser Methoden. Wenn Sie z. B. eine Zertifikatsignierungsanforderung (CSR) verwenden, können Sie nicht auch eine Zertifikat-ID oder ein Geräte-Zertifikat angeben. Weitere Informationen finden Sie unter X.509 Client-Zertifikate.

Weitere Informationen finden Sie unter X.509 Überblick über das Zertifikat.

Zertifikat-Ressourcen werden mit den folgenden Eigenschaften deklariert:

- `CertificateSigningRequest`: Zeichenfolge.
- `CertificateId`: Zeichenfolge.
- `CertificatePem`: Zeichenfolge.
- `CACertificatePem`: Zeichenfolge.
- `Status`: Optional. Zeichenfolge, die `ACTIVE` oder `INACTIVE` sein kann. Der Standardwert ist ACTIVE.

- `ThingPrincipalType`: Optional. Zeichenfolge, die die Art der Beziehung zwischen dem Ding und dem Prinzipal (dem Zertifikat) angibt.
  - `EXCLUSIVE_THING`: Stellt eine exklusive Beziehung her. Der Schulleiter kann nur dieser bestimmten Sache zugeordnet werden und keiner anderen.
  - `NON_EXCLUSIVE_THING`: Hängt den angegebenen Prinzipal an Dinge an. Sie können mehrere Dinge an den Prinzipal anhängen. Dies ist der Standardwert, wenn er nicht angegeben ist.

> **Note**
>
> Sie können auch Geräte ohne Gerätezertifikate bereitstellen. Weitere Informationen finden Sie unter Bereitstellen von Geräten ohne Gerätezertifikate mithilfe der Flottenbereitstellung.

Beispiele:

- Zertifikat, das mit einer Zertifikatsignierungsanforderung (CSR) angegeben wird:

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate",
        "Properties" : {
            "CertificateSigningRequest": {"Ref" : "CSR"},
            "Status" : "ACTIVE"
        }
    }
}
```

- Zertifikat, das mit einer vorhandenen Zertifikat-ID angegeben wird:

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate",
        "Properties" : {
            "CertificateId": {"Ref" : "CertificateId"}
        }
    }
}
```

- Zertifikat, das mit der PEM-Datei eines vorhandenen Zertifikats und der PEM-Datei einer vorhandenen Zertifizierungsstelle angegeben wird:

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate",
        "Properties" : {
            "CACertificatePem": {"Ref" : "CACertificatePem"},
            "CertificatePem": {"Ref" : "CertificatePem"}
        }
    }
}
```

- Hängen Sie einem Principal ausschließlich eine Sache an:

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate",
        "Properties" : {
            "ThingPrincipalType" : "EXCLUSIVE_THING"
        }
    }
}
```

Richtlinienressourcen

Richtlinien-Ressourcen werden mit einer der folgenden Eigenschaften deklariert:

- `PolicyName`: Optional. Zeichenfolge. Standardmäßig eine Prüfsumme des Richtliniendokuments. `PolicyName` kann nur auf AWS IoT -Richtlinien verweisen, nicht jedoch auf IAM-Richtlinien. Wenn Sie eine vorhandene AWS IoT -Richtlinie verwenden, geben Sie für die Eigenschaft `PolicyName` den Namen der Richtlinie ein. Verwenden Sie nicht die Eigenschaft `PolicyDocument`.
- `PolicyDocument`: Optional. Ein JSON-Objekt, das als Zeichenfolge mit Escapezeichen angegeben wird. Wenn `PolicyDocument` nicht angegeben wird, muss die Richtlinie bereits erstellt worden sein.

> **Note**
>
> Wenn ein `Policy`-Abschnitt vorhanden ist, muss `PolicyName` oder `PolicyDocument`, nicht aber beide, angegeben werden.

Überschreibungseinstellungen

Wenn eine Vorlage eine Ressource angibt, die bereits vorhanden ist, ermöglicht der Abschnitt `OverrideSettings` die Angabe der durchzuführenden Aktion:

`DO_NOTHING`

Die Ressource unverändert lassen

`REPLACE`

Die Ressource durch die in der Vorlage angegebene Ressource ersetzen.

`FAIL`

Die Anfrage mit einer `ResourceConflictsException` fehlschlagen lassen.

`MERGE`

Dies gilt nur für die Eigenschaften `ThingGroups` und `AttributePayload` eines `thing`. Kombinieren der vorhandenen Attribute oder Gruppenmitgliedschaften des Objekts mit denjenigen, die in der Vorlage spezifiziert sind.

Wenn Sie eine Ding-Ressource deklarieren, können Sie `OverrideSettings` für die folgenden Eigenschaften angeben:

- `ATTRIBUTE_PAYLOAD`
- `THING_TYPE_NAME`
- `THING_GROUPS`

Wenn Sie eine Zertifikat-Ressource deklarieren, können Sie `OverrideSettings` für die `Status`-Eigenschaft angeben.

`OverrideSettings` stehen nicht für Richtlinienressourcen zur Verfügung.

Ressourcenbeispiel

Das folgende Vorlagenfragment deklariert ein Objekt, ein Zertifikat und eine Richtlinie:

```
{
    "Resources" : {
```

```
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "ThingName" : {"Ref" : "ThingName"},
                "AttributePayload" : { "version" : "v1", "serialNumber" :  {"Ref" :
 "SerialNumber"}},
                "ThingTypeName" :  "lightBulb-versionA",
                "ThingGroups" : ["v1-lightbulbs", {"Ref" : "Location"}]
            },
            "OverrideSettings" : {
                "AttributePayload" : "MERGE",
                "ThingTypeName" : "REPLACE",
                "ThingGroups" : "DO_NOTHING"
            }
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
            "Properties" : {
                "CertificateSigningRequest": {"Ref" : "CSR"},
                "Status" : "ACTIVE"
            }
        },
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : "{ \"Version\": \"2012-10-17\",
 \"Statement\": [{ \"Effect\": \"Allow\", \"Action\":[\"iot:Publish\"], \"Resource\":
 [\"arn:aws:iot:us-east-1:123456789012:topic/foo/bar\"] }] }"
            }
        }
    }
}
```

Das Objekt wird deklariert mit:

- Dem logischen Namen `"thing"`.
- Dem Typ `AWS::IoT::Thing`.
- Einer Reihe von Objekteigenschaften.

  Zu den Objekteigenschaften gehören der Name des Objekts, eine Reihe von Attributen, ein optionaler Name für den Objekttyp sowie eine optionale Liste von Objektgruppen, zu denen das Objekt gehört.

Parameter werden von {"Ref":"*parameter-name*"} referenziert. Wenn die Vorlage evaluiert wird, werden die Parameter durch den Parameterwert aus dem mit der Vorlage eingegebenen Lexikon ersetzt.

Das Zertifikat wird deklariert mit:

- Dem logischen Namen "certificate".
- Dem Typ AWS::IoT::Certificate.
- Einer Reihe von Eigenschaften.

  Zu den Eigenschaften gehören die CSR für das Zertifikat und die Einstellung des Status auf ACTIVE. Der CSR-Text wird als Parameter in das mit der Vorlage eingegebene Lexikon eingegeben.

Die Richtlinie wird deklariert mit:

- Dem logischen Namen "policy".
- Dem Typ AWS::IoT::Policy.
- Dem Namen einer vorhandenen Richtlinie oder dem Richtliniendokument.

Vorlagenbeispiel für eine Massenregistrierung

Die folgende JSON-Datei ist ein Beispiel für eine vollständige Bereitstellungsvorlage, die das Zertifikat mit einer Zertifikatsignierungsanforderung angibt:

(Der Feldwert PolicyDocument muss ein JSON-Objekt sein, da als Zeichenfolge mit Escapezeichen angegeben wird.)

```
{
    "Parameters" : {
        "ThingName" : {
            "Type" : "String"
        },
        "SerialNumber" : {
            "Type" : "String"
        },
        "Location" : {
            "Type" : "String",
            "Default" : "WA"
```

```
        },
        "CSR" : {
            "Type" : "String"
        }
    },
    "Resources" : {
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "ThingName" : {"Ref" : "ThingName"},
                "AttributePayload" : { "version" : "v1", "serialNumber" :  {"Ref" :
 "SerialNumber"}},
                "ThingTypeName" :  "lightBulb-versionA",
                "ThingGroups" : ["v1-lightbulbs", {"Ref" : "Location"}]
            }
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
            "Properties" : {
                "CertificateSigningRequest": {"Ref" : "CSR"},
                "Status" : "ACTIVE",
                "ThingPrincipalType" : "EXCLUSIVE_THING"
            }
        },
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : "{ \"Version\": \"2012-10-17\",
 \"Statement\": [{ \"Effect\": \"Allow\", \"Action\":[\"iot:Publish\"], \"Resource\":
 [\"arn:aws:iot:us-east-1:123456789012:topic/foo/bar\"] }] }"
            }
        }
    }
}
```

Vorlagenbeispiel für eine Just-in-Time-Bereitstellung (JITP)

Die folgende JSON-Datei ist ein Beispiel für eine vollständige Bereitstellungsvorlage, die ein vorhandenes Zertifikat mit einer Zertifikat-ID angibt:

```
{
   "Parameters":{
      "AWS::IoT::Certificate::CommonName":{
```

```
            "Type":"String"
         },
         "AWS::IoT::Certificate::SerialNumber":{
            "Type":"String"
         },
         "AWS::IoT::Certificate::Country":{
            "Type":"String"
         },
         "AWS::IoT::Certificate::Id":{
            "Type":"String"
         }
      },
      "Resources":{
         "thing":{
            "Type":"AWS::IoT::Thing",
            "Properties":{
               "ThingName":{
                  "Ref":"AWS::IoT::Certificate::CommonName"
               },
               "AttributePayload":{
                  "version":"v1",
                  "serialNumber":{
                     "Ref":"AWS::IoT::Certificate::SerialNumber"
                  }
               },
               "ThingTypeName":"lightBulb-versionA",
               "ThingGroups":[
                  "v1-lightbulbs",
                  {
                     "Ref":"AWS::IoT::Certificate::Country"
                  }
               ]
            },
            "OverrideSettings":{
               "AttributePayload":"MERGE",
               "ThingTypeName":"REPLACE",
               "ThingGroups":"DO_NOTHING"
            }
         },
         "certificate":{
            "Type":"AWS::IoT::Certificate",
            "Properties":{
               "CertificateId":{
                  "Ref":"AWS::IoT::Certificate::Id"
```

```
            },
            "Status":"ACTIVE",
            "ThingPrincipalType" : "EXCLUSIVE_THING"
         }
      },
      "policy":{
         "Type":"AWS::IoT::Policy",
         "Properties":{
            "PolicyDocument":"{ \"Version\": \"2012-10-17\",        \"Statement
\": [{ \"Effect\": \"Allow\", \"Action\":[\"iot:Publish\"], \"Resource\":
 [\"arn:aws:iot:us-east-1:123456789012:topic/foo/bar\"] }] }"
         }
      }
   }
}
```

> **Important**
>
> Sie müssen `CertificateId` in einer Vorlage verwenden, die für die JIT-Bereitstellung verwendet wird.

Weitere Informationen zum Typ einer Bereitstellungsvorlage finden Sie `CreateProvisioningTemplate` in der AWS API-Referenz.

Weitere Informationen zur Verwendung dieser Vorlage für die Just-in-Time-Bereitstellung finden Sie unter Bereitstellung. Just-in-time

Flottenbereitstellung

Vorlagen für die Bereitstellung von Flotten werden von verwendet, AWS IoT um die Cloud- und Gerätekonfiguration einzurichten. Diese Vorlagen verwenden dieselben Parameter und Ressourcen wie die JITP- und Massenregistrierungsvorlagen. Weitere Informationen finden Sie unter Bereitstellungsvorlagen. Flottenbereitstellungsvorlagen können einen `Mapping`-Abschnitt und einen `DeviceConfiguration`-Abschnitt enthalten. Sie können intrinsische Funktionen innerhalb einer Flottenbereitstellungsvorlage verwenden, um die gerätespezifische Konfiguration zu generieren. Flottenbereitstellungsvorlagen sind benannte Ressourcen und werden durch ARNs identifiziert (z. B. `arn:aws:iot:us-west-2:1234568788:provisioningtemplate/`*`templateName`*).

Mappings

Im optionalen Abschnitt `Mappings` werden Schlüssel einem Satz benannter Werte zugewiesen. Wenn Sie beispielsweise Werte auf der Grundlage einer AWS Region festlegen möchten, können Sie eine Zuordnung erstellen, die den AWS-Region Namen als Schlüssel verwendet und die Werte enthält, die Sie für jede bestimmte Region angeben möchten. Zum Abrufen von Werten aus einer Karte nutzen Sie die intrinsische Funktion `Fn::FindInMap`.

Im Abschnitt `Mappings` dürfen keine Parameter, Pseudoparameter verwendet oder intrinsische Funktionen aufgerufen werden.

Gerätekonfiguration

Der Abschnitt „Gerätekonfiguration“ enthält beliebige Daten, die Sie bei der Bereitstellung an Ihre Geräte senden möchten. Beispiel:

```
{
    "DeviceConfiguration": {
        "Foo":"Bar"
    }
}
```

Wenn Sie mithilfe des Nutzdatenformats JavaScript Object Notation (JSON) Nachrichten an Ihre Geräte senden, AWS IoT Core formatieren Sie diese Daten als JSON. Wenn Sie das Payload-Format Concise Binary Object Representation (CBOR) verwenden, AWS IoT Core formatiert Sie diese Daten als CBOR. Der Abschnitt `DeviceConfiguration` unterstützt keine verschachtelten JSON-Objekte.

Intrinsische Funktionen

Intrinsische Funktionen werden in jedem Abschnitt der Bereitstellungsvorlage mit Ausnahme des `Mappings`-Abschnitts verwendet.

`Fn::Join`

> Fügt einem einzelnen Wert eine Gruppe von Werten getrennt durch das angegebene Trennzeichen an. Wenn das Trennzeichen eine leere Zeichenfolge ist, dann werden die Werte ohne Trennzeichen verkettet.

> **Important**
>
> `Fn::Join` wird nicht für the section called “Richtlinienressourcen” unterstützt.

`Fn::Select`

Gibt ein einzelnes Objekt aus einer Liste von Objekten nach Index zurück.

> **Important**
>
> `Fn::Select` überprüft nicht auf `null`-Werte oder ob sich der Index außerhalb des Arrays befindet. Beide Bedingungen führen zu einem Bereitstellungsfehler. Stellen Sie daher sicher, dass Sie einen gültigen Indexwert ausgewählt haben und die Liste Werte enthält, die nicht Null sind.

`Fn::FindInMap`

Gibt die Werte von Schlüsseln in einer Zwei-Ebenen-Karte zurück, die im Abschnitt `Mappings` deklariert ist.

`Fn::Split`

Teilt eine Zeichenfolge in eine Liste von Zeichenfolgenwerten auf, sodass Sie ein Element aus der Liste der Zeichenfolgen auswählen können. Sie geben ein Trennzeichen an, das bestimmt, wo die Zeichenfolge geteilt wird (z. B. ein Komma). Nachdem Sie eine Zeichenfolge geteilt haben, verwenden Sie `Fn::Select`, um ein Element auszuwählen.

Wenn beispielsweise eine durch ein Komma getrennte Zeichenfolge von Subnetz-IDs in Ihre Stack-Vorlage importiert wird, können Sie die Zeichenfolge an jedem Komma teilen. Geben Sie in der Liste der Subnetz-IDs mit `Fn::Select` eine Subnetz-ID für eine Ressource an.

`Fn::Sub`

Sie können Variablen in einer Eingabezeichenfolge mit Werten ersetzen, die Sie angeben. Sie können diese Funktion verwenden, um Befehle oder Ausgaben mit Werten zu erstellen, die erst verfügbar sind, wenn Sie ein Stack erstellen oder aktualisieren.

Beispiel einer Flottenbereitstellungsvorlage

```
{
    "Parameters" : {
        "ThingName" : {
            "Type" : "String"
        },
        "SerialNumber": {
            "Type": "String"
        },
        "DeviceLocation": {
            "Type": "String"
        }
    },
    "Mappings": {
        "LocationTable": {
            "Seattle": {
                "LocationUrl": "https://example.aws"
            }
        }
    },
    "Resources" : {
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "AttributePayload" : {
                    "version" : "v1",
                    "serialNumber" : "serialNumber"
                },
                "ThingName" : {"Ref" : "ThingName"},
                "ThingTypeName" : {"Fn::Join":["",["ThingPrefix_",
{"Ref":"SerialNumber"}]]},
                "ThingGroups" : ["v1-lightbulbs", "WA"],
                "BillingGroup": "LightBulbBillingGroup"
            },
            "OverrideSettings" : {
                "AttributePayload" : "MERGE",
                "ThingTypeName" : "REPLACE",
                "ThingGroups" : "DO_NOTHING"
            }
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
```

```
            "Properties" : {
                "CertificateId": {"Ref": "AWS::IoT::Certificate::Id"},
                "Status" : "Active",
                "ThingPrincipalType" : "EXCLUSIVE_THING"
            }
        },
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : {
                    "Version": "2012-10-17",
                    "Statement": [{
                        "Effect": "Allow",
                        "Action":["iot:Publish"],
                        "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/foo/
bar"]
                    }]
                }
            }
        }
    },
    "DeviceConfiguration": {
        "FallbackUrl": "https://www.example.com/test-site",
        "LocationUrl": {
            "Fn::FindInMap": ["LocationTable",{"Ref": "DeviceLocation"},
 "LocationUrl"]}
        }
}
```

> **Note**
>
> Eine vorhandene Bereitstellungsvorlage kann aktualisiert werden, um einen Pre-Provisioning-Hook hinzuzufügen.

Pre-provisioning Haken

AWS empfiehlt, bei der Erstellung von Bereitstellungsvorlagen Hook-Funktionen vor der Bereitstellung zu verwenden, um besser kontrollieren zu können, welche und wie viele Geräte Ihr Konto nutzt. Pre-provisioning Hooks sind Lambda-Funktionen, die vom Gerät übergebene Parameter validieren, bevor das Gerät bereitgestellt werden kann. Diese Lambda-Funktion muss in Ihrem Konto

vorhanden sein, bevor Sie ein Gerät bereitstellen, da sie jedes Mal aufgerufen wird, wenn ein Gerät eine Anfrage über the section called “RegisterThing” sendet.

> **Important**
>
> Achten Sie darauf, das `source-arn` oder `source-account` in die globalen Bedingungskontextschlüssel der Richtlinien aufzunehmen, die mit Ihrer Lambda-Aktion verknüpft sind, damit Berechtigungen nicht manipuliert werden können. Weitere Informationen hierzu finden Sie unter Cross-service verwirrter Stellvertreter, Prävention.

Damit Geräte bereitgestellt werden können, muss Ihre Lambda-Funktion das Eingabeobjekt akzeptieren und das in diesem Abschnitt beschriebene Ausgabeobjekt zurückgeben.
Die Bereitstellung wird nur fortgesetzt, wenn die Lambda-Funktion ein Objekt mit `"allowProvisioning": True` zurückgibt.

Pre-provision Hook-Eingabe

AWS IoT sendet dieses Objekt an die Lambda-Funktion, wenn sich ein Gerät bei AWS IoT registriert.

```
{
    "claimCertificateId" : "string",
    "certificateId" : "string",
    "certificatePem" : "string",
    "templateArn" : "arn:aws:iot:us-east-1:1234567890:provisioningtemplate/MyTemplate",
    "clientId" : "221a6d10-9c7f-42f1-9153-e52e6fc869c1",
    "parameters" : {
        "string" : "string",
        ...
    }
}
```

Das an die Lambda-Funktion übergebene `parameters`-Objekt enthält die Eigenschaften im `parameters`-Argument, das in der the section called “RegisterThing”-Anforderungsnutzlast übergeben wird.

Pre-provision Hook-Rückgabewert

Die Lambda-Funktion muss eine Antwort zurückgeben, die angibt, ob sie die Bereitstellungsanforderung und die Werte der zu überschreibenden Eigenschaften autorisiert hat.

Im Folgenden finden Sie ein Beispiel für eine erfolgreiche Antwort der Pre-Provisioning-Funktion.

```
{
    "allowProvisioning": true,
    "parameterOverrides" : {
        "Key": "newCustomValue",
        ...
    }
}
```

`"parameterOverrides"`-Werte werden dem Parameter `"parameters"` in der the section called “RegisterThing”-Anforderungsnutzlast hinzugefügt.

> **Note**
>
> - Wenn die Lambda-Funktion fehlschlägt, schlägt die Bereitstellungsanforderung fehl `ACCESS_DENIED` und ein Fehler wird in Logs protokolliert. CloudWatch
> - Wenn die Lambda-Funktion in der Antwort nicht `"allowProvisioning": "true"` zurückgibt, schlägt die Bereitstellungsanforderung mit `ACCESS_DENIED` fehl.
> - Die Lambda-Funktion muss die Ausführung beenden und innerhalb von 5 Sekunden zurückgeben, andernfalls schlägt die Bereitstellungsanforderung fehl.

Pre-provisioning Hook-Lambda-Beispiel

Python

Ein Beispiel für einen Pre-Provisioning-Hook ist Lambda in Python.

```
import json

def pre_provisioning_hook(event, context):
    print(event)

    return {
        'allowProvisioning': True,
        'parameterOverrides': {
            'DeviceLocation': 'Seattle'
        }
```

```
    }
```

Java

Ein Beispiel für einen Pre-Provisioning-Hook ist Lambda in Java.

Handler-Klasse

```
package example;

import java.util.Map;
import java.util.HashMap;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

public class PreProvisioningHook implements
 RequestHandler<PreProvisioningHookRequest, PreProvisioningHookResponse> {

    public PreProvisioningHookResponse handleRequest(PreProvisioningHookRequest
 object, Context context) {
        Map<String, String> parameterOverrides = new HashMap<String, String>();
        parameterOverrides.put("DeviceLocation", "Seattle");

        PreProvisioningHookResponse response = PreProvisioningHookResponse.builder()
                .allowProvisioning(true)
                .parameterOverrides(parameterOverrides)
                .build();

        return response;
    }

}
```

Anforderungsklasse:

```
package example;

import java.util.Map;
import lombok.Builder;
import lombok.Data;
import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;
```

```
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class PreProvisioningHookRequest {
    private String claimCertificateId;
    private String certificateId;
    private String certificatePem;
    private String templateArn;
    private String clientId;
    private Map<String, String> parameters;
}
```

Response-Klasse:

```
package example;

import java.util.Map;
import lombok.Builder;
import lombok.Data;
import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;


@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class PreProvisioningHookResponse {
    private boolean allowProvisioning;
    private Map<String, String> parameterOverrides;
}
```

JavaScript

Ein Beispiel für einen Lambda-Hook vor der Bereitstellung. JavaScript

```
exports.handler = function(event, context, callback) {
    console.log(JSON.stringify(event, null, 2));
    var reply = {
        allowProvisioning: true,
        parameterOverrides: {
            DeviceLocation: 'Seattle'
```

```
        }
    };
    callback(null, reply);
}
```

Self-managed Signieren von Zertifikaten mit AWS IoT Core Zertifikatsanbieter

Sie können einen AWS IoT Core Zertifikatsanbieter erstellen, um Certificate Signing Requests (CSRs) in der AWS IoT Flottenbereitstellung zu signieren. Ein Zertifikatsanbieter verweist auf eine Lambda-Funktion und die `CreateCertificateFromCsr` MQTT-API für die Flottenbereitstellung. Die Lambda-Funktion akzeptiert eine CSR und gibt ein signiertes Client-Zertifikat zurück.

Wenn Sie keinen Zertifikatsanbieter bei sich haben AWS-Konto, wird die CreateCertificateFromCsr MQTT-API in der Flottenbereitstellung aufgerufen, um das Zertifikat aus einem CSR zu generieren. Nachdem Sie einen Zertifikatsanbieter erstellt haben, ändert sich das Verhalten der CreateCertificateFromCsr MQTT-API, und alle Aufrufe dieser MQTT-API rufen den Zertifikatsanbieter auf, das Zertifikat auszustellen.

Mit dem AWS IoT Core Zertifikatsanbieter können Sie Lösungen implementieren, die private Zertifizierungsstellen (CAs) wie AWS Private CA andere öffentlich vertrauenswürdige Zertifizierungsstellen oder Ihre eigene Public Key Infrastructure (PKI) verwenden, um die CSR zu signieren. Darüber hinaus können Sie mithilfe des Zertifikatsanbieters die Felder Ihres Client-Zertifikats wie Gültigkeitszeiträume, Signaturalgorithmen, Aussteller und Erweiterungen anpassen.

> **Important**
>
> Sie können jeweils nur einen Zertifikatsanbieter erstellen. AWS-Konto Die Änderung des Signierungsverhaltens gilt für die gesamte Flotte, die die CreateCertificateFromCsr MQTT-API aufruft, bis Sie den Zertifikatsanbieter aus Ihrem AWS-Konto löschen.

In diesem Thema:

- So funktioniert die selbstverwaltete Zertifikatsignierung bei der Flottenbereitstellung
- Eingabe der Lambda-Funktion des Zertifikatsanbieters
- Rückgabewert der Lambda-Funktion des Zertifikatsanbieters

- Beispiel-Lambda-Funktion
- Self-managed Unterzeichnung eines Zertifikats für die Bereitstellung der Flotte
- AWS CLI Befehle für den Zertifikatsanbieter

So funktioniert die selbstverwaltete Zertifikatsignierung bei der Flottenbereitstellung

Die wichtigsten Konzepte

Die folgenden Konzepte enthalten Einzelheiten, anhand derer Sie besser verstehen, wie die selbstverwaltete Zertifikatsignierung bei der Flottenbereitstellung funktioniert. AWS IoT Weitere Informationen finden Sie unter Bereitstellung von Geräten ohne Gerätezertifikate mithilfe von Fleet Provisioning.

AWS IoT Bereitstellung der Flotte

Mit AWS IoT Fleet Provisioning (kurz für Fleet Provisioning) werden Gerätezertifikate AWS IoT Core generiert und sicher an Ihre Geräte übertragen, wenn diese AWS IoT Core zum ersten Mal eine Verbindung herstellen. Sie können Fleet Provisioning verwenden, um Geräte zu verbinden, zu denen keine Gerätezertifikate vorliegen. AWS IoT Core

Anfrage zur Unterzeichnung eines Zertifikats (CSR)

Während der Flottenbereitstellung sendet ein Gerät eine Anfrage AWS IoT Core über die MQTT-APIs für die Flottenbereitstellung. Diese Anfrage beinhaltet eine Certificate Signing Request (CSR), die signiert wird, um ein Client-Zertifikat zu erstellen.

AWS verwaltete das Signieren von Zertifikaten bei der Flottenbereitstellung

AWS Managed ist die Standardeinstellung für das Signieren von Zertifikaten bei der Flottenbereitstellung. Bei AWS verwalteter Zertifikatsignierung signiert AWS IoT Core das Unternehmen CSRs mithilfe seiner eigenen Zertifizierungsstellen.

Self-managed Unterzeichnung von Zertifikaten bei der Bereitstellung von Flotten

Self-managed ist eine weitere Option für das Signieren von Zertifikaten bei der Flottenbereitstellung. Bei der selbstverwalteten Zertifikatsignierung erstellen Sie einen AWS IoT Core Zertifikatsanbieter zum Signieren von CSRs. Sie können die selbstverwaltete Zertifikatsignierung verwenden, um CSRs mit einer CA zu signieren, die von einer AWS privaten

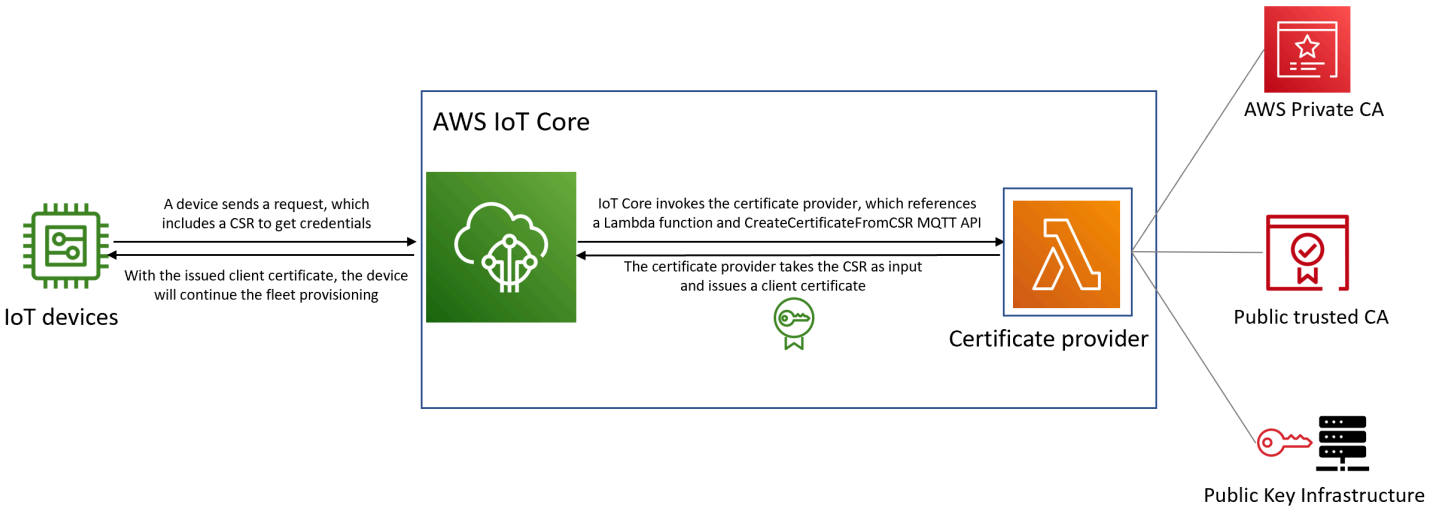
CA, einer anderen öffentlich vertrauenswürdigen CA oder Ihrer eigenen Public Key Infrastructure (PKI) generiert wurde.

AWS IoT Core Zertifikatsanbieter

AWS IoT Core Certificate Provider (kurz für Certificate Provider) ist eine vom Kunden verwaltete Ressource, die für die selbstverwaltete Zertifikatsignierung bei der Flottenbereitstellung verwendet wird.

Diagramm

Das folgende Diagramm veranschaulicht in vereinfachter Form, wie das Signieren von Selbstzertifikaten bei der AWS IoT Flottenbereitstellung funktioniert.



- Wenn ein neues IoT-Gerät hergestellt oder in die Flotte eingeführt wird, benötigt es Client-Zertifikate, mit denen es sich authentifizieren kann. AWS IoT Core
- Im Rahmen des Flottenbereitstellungsprozesses fordert das Gerät über die MQTT-APIs AWS IoT Core für die Flottenbereitstellung Kundenzertifikate an. Diese Anfrage beinhaltet eine Anfrage zur Zertifikatsignierung (CSR).
- AWS IoT Core ruft den Zertifikatsanbieter auf und übergibt die CSR als Eingabe an den Anbieter.
- Der Zertifikatsanbieter verwendet die CSR als Eingabe und stellt ein Client-Zertifikat aus.

  Bei AWS verwalteter Zertifikatsignierung signiert er den CSR mit seiner eigenen Zertifizierungsstelle und stellt ein Client-Zertifikat aus. AWS IoT Core
- Mit dem ausgestellten Client-Zertifikat setzt das Gerät die Bereitstellung der Flotte fort und stellt eine sichere Verbindung zu her. AWS IoT Core

Eingabe der Lambda-Funktion des Zertifikatsanbieters

AWS IoT Core sendet das folgende Objekt an die Lambda-Funktion, wenn sich ein Gerät bei ihr registriert. Der Wert von `certificateSigningRequest` ist der CSR im Privacy-Enhanced Mail-Format (PEM), der in der Anfrage bereitgestellt wird. `CreateCertificateFromCsr` Das `principalId` ist die ID des Prinzipals, mit dem AWS IoT Core bei der Anfrage eine Verbindung hergestellt wurde. `CreateCertificateFromCsr` `clientId`dist die für die MQTT-Verbindung festgelegte Client-ID.

```
{
 "certificateSigningRequest": "string",
 "principalId": "string",
 "clientId": "string"
}
```

Rückgabewert der Lambda-Funktion des Zertifikatsanbieters

Die Lambda-Funktion muss eine Antwort zurückgeben, die den `certificatePem` Wert enthält. Das Folgende ist ein Beispiel für eine erfolgreiche Antwort. AWS IoT Core verwendet den Rückgabewert (`certificatePem`), um das Zertifikat zu erstellen.

```
{
 "certificatePem": "string"
}
```

Wenn die Registrierung erfolgreich ist, `CreateCertificateFromCsr` wird dasselbe `certificatePem` in der `CreateCertificateFromCsr` Antwort zurückgegeben. Weitere Informationen finden Sie im Beispiel für eine Antwort-Nutzlast von CreateCertificateFromCsr.

Beispiel-Lambda-Funktion

Bevor Sie einen Zertifikatsanbieter erstellen, müssen Sie eine Lambda-Funktion erstellen, um eine CSR zu signieren. Das Folgende ist ein Beispiel für eine Lambda-Funktion in Python. Diese Funktion ruft AWS Private CA auf, um den eingegebenen CSR mithilfe einer privaten CA und des `SHA256WITHRSA` Signaturalgorithmus zu signieren. Das zurückgegebene Client-Zertifikat ist ein Jahr lang gültig. Weitere Informationen zu AWS Private CA und wie Sie eine private CA erstellen, finden Sie unter Was ist eine AWS private CA? und Erstellen einer privaten CA.

```
import os
```

```
import time
import uuid
import boto3

def lambda_handler(event, context):
    ca_arn = os.environ['CA_ARN']
    csr = (event['certificateSigningRequest']).encode('utf-8')

    acmpca = boto3.client('acm-pca')
    cert_arn = acmpca.issue_certificate(
        CertificateAuthorityArn=ca_arn,
        Csr=csr,
        Validity={"Type": "DAYS", "Value": 365},
        SigningAlgorithm='SHA256WITHRSA',
        IdempotencyToken=str(uuid.uuid4())
    )['CertificateArn']

    # Wait for certificate to be issued
    time.sleep(1)
    cert_pem = acmpca.get_certificate(
        CertificateAuthorityArn=ca_arn,
        CertificateArn=cert_arn
    )['Certificate']

    return {
        'certificatePem': cert_pem
    }
```

Important

- Von der Lambda-Funktion zurückgegebene Zertifikate müssen denselben Betreffnamen und öffentlichen Schlüssel wie die Certificate Signing Request (CSR) haben.
- Die Lambda-Funktion muss in 5 Sekunden vollständig ausgeführt werden.
- Die Lambda-Funktion muss sich in derselben Region wie die AWS-Konto Ressource des Zertifikatsanbieters befinden.
- Dem AWS IoT Dienstprinzipal muss die Aufrufberechtigung für die Lambda-Funktion erteilt werden. Um verwirrende Stellvertreterprobleme zu vermeiden, empfehlen wir, die `sourceAccount` Aufrufberechtigungen festzulegen`sourceArn`. Weitere Informationen finden Sie unter Cross-service Confused Deputy Prevention.

Das folgende ressourcenbasierte Richtlinienbeispiel für Lambda gewährt die Erlaubnis, AWS IoT die Lambda-Funktion aufzurufen:

```
{
 "Version":"2012-10-17",
 "Id": "InvokePermission",
 "Statement": [
  {
   "Sid": "LambdaAllowIotProvider",
   "Effect": "Allow",
   "Principal": {
    "Service": "iot.amazonaws.com"
   },
   "Action": "lambda:InvokeFunction",
   "Resource": "arn:aws:lambda:us-east-1:123456789012:function:my-function",
   "Condition": {
    "StringEquals": {
     "AWS:SourceAccount": "123456789012"
    },
    "ArnLike": {
    "AWS:SourceArn": "arn:aws:iot:us-east-1:123456789012:certificateprovider/my-
certificate-provider"
    }
   }
  }
 ]
}
```

Self-managed Unterzeichnung eines Zertifikats für die Bereitstellung der Flotte

Sie können die selbstverwaltete Zertifikatsignierung für die Flottenbereitstellung mithilfe von oder wählen. AWS CLI AWS-Managementkonsole

AWS CLI

Um die selbstverwaltete Zertifikatsignierung zu wählen, müssen Sie einen AWS IoT Core Zertifikatsanbieter einrichten, der CSRs bei der Flottenbereitstellung signiert. AWS IoT Core ruft den Zertifikatsanbieter auf, der eine CSR als Eingabe verwendet und ein Client-Zertifikat

zurückgibt. Verwenden Sie die `CreateCertificateProvider` API-Operation oder den `create-certificate-provider` CLI-Befehl, um einen Zertifikatsanbieter zu erstellen.

> **Note**
>
> Nachdem Sie einen Zertifikatsanbieter erstellt haben, ändert sich das Verhalten der `CreateCertificateFromCsr` API für die Flottenbereitstellung, sodass alle Aufrufe von den Zertifikatsanbieter aufrufen, um `CreateCertificateFromCsr` die Zertifikate zu erstellen. Es kann einige Minuten dauern, bis sich dieses Verhalten ändert, nachdem ein Zertifikatsanbieter erstellt wurde.

```
aws iot create-certificate-provider \
               --certificateProviderName my-certificate-provider \
               --lambdaFunctionArn arn:aws:lambda:us-east-1:123456789012:function:my-
function-1 \
               --accountDefaultForOperations CreateCertificateFromCsr
```

Das Folgende zeigt eine Beispielausgabe für diesen Befehl:

```
{
 "certificateProviderName": "my-certificate-provider",
 "certificateProviderArn": "arn:aws:iot:us-east-1:123456789012:certificateprovider:my-
certificate-provider"
}
```

Weitere Informationen finden Sie in `CreateCertificateProvider` der AWS IoT API-Referenz.

AWS-Managementkonsole

Gehen Sie wie folgt vor, um die selbstverwaltete Zertifikatsignierung mithilfe AWS-Managementkonsole der folgenden Schritte auszuwählen:

1. Rufen Sie die AWS IoT -Konsole auf.
2. Wählen Sie im linken Navigationsbereich unter Sicherheit die Option Zertifikatsignierung aus.
3. Wählen Sie auf der Seite zum Signieren von Zertifikaten unter Details zur Zertifikatsignierung die Option Methode zur Zertifikatsignierung bearbeiten aus.
4. Wählen Sie auf der Seite Signaturmethode für das Zertifikat bearbeiten unter Signaturmethode für das Zertifikat die Option aus Self-managed.

5. Geben Sie im Abschnitt „Self-managed Einstellungen“ einen Namen für den Zertifikatsanbieter ein und erstellen Sie dann eine Lambda-Funktion oder wählen Sie sie aus.
6. Wählen Sie Zertifikatsignierung aktualisieren.

AWS CLI Befehle für den Zertifikatsanbieter

Zertifikatsanbieter erstellen

Verwenden Sie den `CreateCertificateProvider` API-Vorgang oder den `create-certificate-provider` CLI-Befehl, um einen Zertifikatsanbieter zu erstellen.

> **Note**
>
> Nachdem Sie einen Zertifikatsanbieter erstellt haben, ändert sich das Verhalten der `CreateCertificateFromCsr` API für die Flottenbereitstellung, sodass alle Aufrufe von den Zertifikatsanbieter aufrufen, um `CreateCertificateFromCsr` die Zertifikate zu erstellen. Es kann einige Minuten dauern, bis sich dieses Verhalten ändert, nachdem ein Zertifikatsanbieter erstellt wurde.

```
aws iot create-certificate-provider \
                --certificateProviderName my-certificate-provider \
                --lambdaFunctionArn arn:aws:lambda:us-east-1:123456789012:function:my-
function-1 \
                --accountDefaultForOperations CreateCertificateFromCsr
```

Das Folgende zeigt eine Beispielausgabe für diesen Befehl:

```
{
 "certificateProviderName": "my-certificate-provider",
 "certificateProviderArn": "arn:aws:iot:us-east-1:123456789012:certificateprovider:my-
certificate-provider"
}
```

Weitere Informationen finden Sie in `CreateCertificateProvider` der AWS IoT API-Referenz.

Aktualisieren Sie den Zertifikatsanbieter

Verwenden Sie den `UpdateCertificateProvider` API-Vorgang oder den `update-certificate-provider` CLI-Befehl, um einen Zertifikatsanbieter zu aktualisieren.

```
aws iot update-certificate-provider \
                --certificateProviderName my-certificate-provider \
                --lambdaFunctionArn arn:aws:lambda:us-east-1:123456789012:function:my-function-2 \
                --accountDefaultForOperations CreateCertificateFromCsr
```

Das Folgende zeigt eine Beispielausgabe für diesen Befehl:

```
{
 "certificateProviderName": "my-certificate-provider",
 "certificateProviderArn": "arn:aws:iot:us-east-1:123456789012:certificateprovider:my-certificate-provider"
}
```

Weitere Informationen finden Sie in `UpdateCertificateProvider` der AWS IoT API-Referenz.

Beschreiben Sie den Zertifikatsanbieter

Verwenden Sie den `DescribeCertificateProvider` API-Vorgang oder den `describe-certificate-provider` CLI-Befehl, um einen Zertifikatsanbieter zu beschreiben.

```
aws iot describe-certificate-provider --certificateProviderName my-certificate-provider
```

Das Folgende zeigt eine Beispielausgabe für diesen Befehl:

```
{
 "certificateProviderName": "my-certificate-provider",
 "lambdaFunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function",
 "accountDefaultForOperations": [
  "CreateCertificateFromCsr"
 ],
 "creationDate": "2022-11-03T00:15",
 "lastModifiedDate": "2022-11-18T00:15"
}
```

Weitere Informationen finden Sie in `DescribeCertificateProvider` der AWS IoT API-Referenz.

Löschen Sie den Zertifikatsanbieter

Verwenden Sie den `DeleteCertificateProvider` API-Vorgang oder den `delete-certificate-provider` CLI-Befehl, um einen Zertifikatsanbieter zu löschen. Wenn Sie die Ressource des Zertifikatsanbieters löschen, `CreateCertificateFromCsr` wird das Verhalten von wieder aufgenommen und AWS IoT es werden Zertifikate erstellt, die AWS IoT von einem CSR signiert wurden.

```
aws iot delete-certificate-provider --certificateProviderName my-certificate-provider
```

Dieser Befehl liefert keine Ausgabe.

Weitere Informationen finden Sie in `DeleteCertificateProvider` der AWS IoT API-Referenz.

Den Zertifikatsanbieter auflisten

Verwenden Sie die `ListCertificateProviders` API-Operation oder den `list-certificate-providers` CLI-Befehl AWS-Konto, um die Zertifikatsanbieter in Ihrem aufzulisten.

```
aws iot list-certificate-providers
```

Das Folgende zeigt eine Beispielausgabe für diesen Befehl:

```
{
 "certificateProviders": [
  {
   "certificateProviderName": "my-certificate-provider",
   "certificateProviderArn": "arn:aws:iot:us-
east-1:123456789012:certificateprovider:my-certificate-provider"
  }
 ]
}
```

Weitere Informationen finden Sie in `ListCertificateProvider` der AWS IoT API-Referenz.

Erstellen von IAM-Richtlinien und -Rollen für Benutzer, die ein Gerät installieren

> **Note**
>
> Diese Verfahren dürfen nur verwendet werden, wenn Sie von der AWS IoT Konsole dazu aufgefordert werden.
> Öffnen Sie Neue Bereitstellungsvorlage erstellen, um von der Konsole aus zu dieser Seite zu gelangen.

Warum kann das nicht in der AWS IoT Konsole?

IAM-Aktionen werden aus Sicherheitsgründen in der IAM-Konsole ausgeführt. Die Verfahren in diesem Abschnitt führen Sie durch die Schritte zum Erstellen der IAM-Rollen und -Richtlinien, die für die Verwendung der Bereitstellungsvorlage erforderlich sind.

Erstellen einer IAM-Richtlinie für Benutzer, die ein Gerät installieren werden

In diesem Verfahren wird beschrieben, wie Sie eine IAM-Richtlinie erstellen, die Benutzer autorisiert, ein Gerät mithilfe einer Bereitstellungsvorlage zu installieren.

Während Sie dieses Verfahren ausführen, wechseln Sie zwischen der IAM-Konsole und der AWS IoT Konsole. Wir empfehlen, während dieses Verfahrens beide Konsolen gleichzeitig zu öffnen.

Erstellen einer IAM-Richtlinie für Benutzer, die ein Gerät installieren werden

1. Öffnen Sie das Richtlinien-Hub in der IAM-Konsole.
2. Wählen Sie Richtlinie erstellen aus.
3. Wählen Sie auf der Seite Richtlinie erstellen die Registerkarte JSON aus.
4. Wechseln Sie in der AWS IoT Konsole zu der Seite, auf der Sie Benutzerrichtlinie und Rolle konfigurieren ausgewählt haben.
5. Wählen Sie in der Beispiel-Bereitstellungsrichtlinie die Option Kopieren.
6. Wechseln Sie zurück zur IAM-Konsole.
7. Fügen Sie im JSON-Editor die Richtlinie ein, die Sie von der AWS IoT Konsole kopiert haben. Diese Richtlinie ist spezifisch für die Vorlage, die Sie in der AWS IoT Konsole erstellen.
8. Wählen Sie Weiter: Tags, um fortzufahren.

9. Wählen Sie auf der Seite Tags hinzufügen (optional) für jedes Tag, das Sie zu dieser Richtlinie hinzufügen möchten, die Option Tag hinzufügen aus. Diesen Schritt können Sie überspringen, wenn Sie keine Tags hinzufügen möchten.
10. Wählen Sie Weiter: Überprüfung, um fortzufahren.
11. Führen Sie auf der Seite Richtlinie überprüfen die folgenden Schritte aus:

    a. Geben Sie unter Name* einen Namen für die Richtlinie ein, der Sie an den Zweck der Richtlinie erinnert.

       Notieren Sie sich den Namen, den Sie dieser Richtlinie geben, da Sie ihn im nächsten Verfahren benötigen werden.

    b. Optional können Sie eine Beschreibung für die Richtlinie eingeben, die Sie erstellen.

    c. Lesen Sie sich den Rest dieser Richtlinie und ihrer Tags durch.
12. Wählen Sie Richtlinie erstellen, um Ihre Richtlinie zu erstellen.

Nachdem Sie Ihre neue Richtlinie erstellt haben, fahren Sie mit the section called “Erstellen einer IAM-Rolle für Benutzer, die ein Gerät installieren werden” fort, um den Rolleneintrag der Benutzer zu erstellen, denen Sie diese Richtlinie anhängen.

Erstellen einer IAM-Rolle für Benutzer, die ein Gerät installieren werden

In diesen Schritten wird beschrieben, wie Sie eine IAM-Rolle erstellen, die die Benutzer authentifiziert, die ein Gerät mithilfe einer Bereitstellungsvorlage installieren.

Erstellen einer IAM-Richtlinie für Benutzer, die ein Gerät installieren werden

1. Öffnen Sie die Seite Rollen-Hub in der IAM-Konsole.
2. Wählen Sie Rolle erstellen aus.
3. Wählen Sie unter Vertrauenswürdige Entität auswählen den Typ der vertrauenswürdigen Entität aus, der Sie Zugriff auf die von Ihnen erstellte Vorlage gewähren möchten.
4. Wählen Sie die ID der vertrauenswürdigen Entität, der Sie Zugriff gewähren möchten, oder geben Sie sie ein, und klicken Sie dann auf Weiter.
5. Geben Sie auf der Seite Berechtigungen hinzufügen unter Berechtigungsrichtlinien im Suchfeld den Namen der Richtlinie ein, die Sie im vorherigen Verfahren erstellt haben.
6. Wählen Sie für die Richtlinienliste die Richtlinie aus, die Sie im vorherigen Verfahren erstellt haben, und klicken Sie anschließend auf Weiter.

7. Gehen Sie im Abschnitt Name, prüfen und erstellen wie folgt vor:

    a. Geben Sie unter Rollenname einen Rollennamen ein, der Sie an den Zweck dieser Rolle erinnert.
    b. Unter Beschreibung können Sie optional eine Beschreibung der Rolle eingeben. Dies ist nicht erforderlich, um fortzufahren.
    c. Überprüfen Sie die Werte in Schritt 1 und Schritt 2.
    d. Unter Tags hinzufügen (optional) können Sie auswählen, ob Sie dieser Rolle Tags hinzufügen möchten. Dies ist nicht erforderlich, um fortzufahren.
    e. Vergewissern Sie sich, dass die Informationen auf dieser Seite vollständig und korrekt sind, und wählen Sie dann Rolle erstellen.

Nachdem Sie die neue Rolle erstellt haben, kehren Sie zur AWS IoT Konsole zurück, um mit der Erstellung der Vorlage fortzufahren.

Aktualisieren einer vorhandenen Richtlinie, um eine neue Vorlage zu autorisieren

In den folgenden Schritten wird beschrieben, wie Sie einer IAM-Richtlinie eine neue Vorlage hinzufügen, die Benutzer autorisiert, ein Gerät mithilfe einer Bereitstellungsvorlage zu installieren.

Hinzufügen einer neuen Vorlage zu einer bestehenden IAM-Richtlinie

1. Öffnen Sie das Richtlinien-Hub in der IAM-Konsole.
2. Geben Sie im Suchfeld den Namen der zu aktualisierenden Richtlinie ein.
3. Suchen Sie in der Liste unter dem Suchfeld die Richtlinie, die Sie aktualisieren möchten, und wählen Sie den Richtliniennamen aus.
4. Wählen Sie für die Richtlinienübersicht die Registerkarte JSON, falls dieser Bereich noch nicht sichtbar ist.
5. Wählen Sie Richtlinie bearbeiten, um das Richtliniendokument zu bearbeiten.
6. Wählen Sie im Editor die Registerkarte JSON, falls dieser Bereich noch nicht sichtbar ist.
7. Suchen Sie im Richtliniendokument nach der Richtlinienerklärung, die die `iot:CreateProvisioningClaim`-Aktion enthält.

    Wenn das Richtliniendokument keine Richtlinienanweisung mit der `iot:CreateProvisioningClaim`-Aktion enthält, kopieren Sie den folgenden

Anweisungsausschnitt und fügen Sie ihn als zusätzlichen Eintrag in das `Statement`-Array im Richtliniendokument ein.

> **Note**
>
> Dieser Ausschnitt muss vor dem letzten `]`-Zeichen im `Statement`-Array stehen. Möglicherweise müssen Sie vor oder nach diesem Ausschnitt ein Komma setzen, um Syntaxfehler zu korrigieren.

```
{
    "Effect": "Allow",
    "Action": [
        "iot:CreateProvisioningClaim"
    ],
    "Resource": [
        "--PUT YOUR NEW TEMPLATE ARN HERE--"
    ]
}
```

8. Wechseln Sie zu der Seite in der AWS IoT Konsole, auf der Sie Benutzerrollenberechtigungen ändern ausgewählt haben.
9. Suchen Sie den Ressourcen-ARN der Vorlage und wählen Sie Kopieren.
10. Wechseln Sie zurück zur IAM-Konsole.
11. Fügen Sie den kopierten Amazon-Ressourcennamen (ARN) als ersten Eintrag oben in die Liste der Vorlagen-ARNs in das `Statement`-Array ein.

    Wenn dies der einzige ARN im Array ist, entfernen Sie das Komma am Ende des Werts, den Sie gerade eingefügt haben.
12. Lesen Sie die aktualisierte Richtlinienanweisung und korrigieren Sie alle vom Editor angegebenen Fehler.
13. Wählen Sie Richtlinie überprüfen, um das aktualisierte Richtliniendokument zu speichern.
14. Überprüfen Sie die Richtlinie und wählen Sie anschließend Änderungen speichern.
15. Kehren Sie zur AWS IoT Konsole zurück.

MQTT-API für die Gerätebereitstellung

Der Fleet Provisioning-Dienst unterstützt die folgenden MQTT-API-Operationen:

- the section called “CreateCertificateFromCsr”
- the section called “CreateKeysAndCertificate”
- the section called “RegisterThing”

Diese API unterstützt Antwortpuffer im Format Concise Binary Object Representation (CBOR) und JavaScript Object Notation (JSON), je nach Thema. *payload-format* Der Übersichtlichkeit halber werden die Antwort- und Anforderungsbeispiele in diesem Abschnitt im JSON-Format angezeigt.

payload-format	Datentyp des Antwortformats
cbor	Concise Binary Object Representation (CBOR)
json	JavaScript Objektnotation (JSON)

> **Important**
>
> Bevor Sie ein Anforderungsnachrichtenthema veröffentlichen, abonnieren Sie die Antwortthemen, um die Antwort zu erhalten. Die von dieser API verwendeten Nachrichten verwenden das publish/subscribe MQTT-Protokoll, um eine Anfrage- und Antwortinteraktion bereitzustellen.
> Wenn Sie die Antwortthemen nicht abonnieren, bevor Sie eine Anfrage veröffentlichen, erhalten Sie möglicherweise nicht die Ergebnisse dieser Anfrage.
> IoT Core Fleet Provisioning gibt die MQTT-API-Ergebnisse für die Gerätebereitstellung über dieselbe MQTT-Verbindung zurück, die für die Veröffentlichung der API-Anfrage verwendet wurde.

CreateCertificateFromCsr

Erstellt ein Zertifikat aus einer Zertifikatsignieranforderung (CSR). AWS IoT stellt Client-Zertifikate bereit, die von der Amazon Root Certificate Authority (CA) signiert sind. Das neue Zertifikat hat den Status `PENDING_ACTIVATION`. Wenn Sie `RegisterThing` aufrufen, um ein Objekt mit diesem Zertifikat bereitzustellen, ändert sich der Zertifikatstatus in `ACTIVE` oder `INACTIVE`, wie in der Vorlage beschrieben.

Weitere Informationen zum Erstellen eines Client-Zertifikats mit Ihrem Zertifizierungsstellenzertifikat und einer Zertifikatsignierungsanforderung finden Sie unter Erstellen eines Clientzertifikats mit Ihrem CA-Zertifikat.

> **Note**
>
> Aus Sicherheitsgründen läuft das von `CreateCertificateFromCsr` ausgegebene `certificateOwnershipToken` zurückgesendet nach einer Stunde ab. `RegisterThing` muss aufgerufen werden, bevor `certificateOwnershipToken` abläuft. Wenn das von erstellte Zertifikat `CreateCertificateFromCsr` bis zum Ablauf des Tokens nicht aktiviert und an eine Richtlinie oder eine Sache angehängt wurde, wird das Zertifikat gelöscht. Wenn das Token abläuft, kann das Gerät `CreateCertificateFromCsr` erneut aufrufen, um ein neues Zertifikat zu generieren.

CreateCertificateFromCsr Anfrage

Veröffentlichen Sie eine Nachricht mit dem Thema `$aws/certificates/create-from-csr/`*`payload-format`*.

`payload-format`

Das Nachrichtennutzlastformat ist `cbor` oder `json`.

CreateCertificateFromCsr Nutzlast anfordern

```
{
    "certificateSigningRequest": "string"
}
```

`certificateSigningRequest`

Die CSR im PEM-Format.

CreateCertificateFromCsr Antwort

Abonnieren Sie `$aws/certificates/create-from-csr/`*`payload-format`*`/accepted`.

`payload-format`

Das Nachrichtennutzlastformat ist `cbor` oder `json`.

CreateCertificateFromCsr Nutzlast der Antwort

```
{
    "certificateOwnershipToken": "string",
    "certificateId": "string",
    "certificatePem": "string"
}
```

`certificateOwnershipToken`

Das Token, um den Besitz des Zertifikats während der Bereitstellung nachzuweisen.

`certificateId`

Die ID des Zertifikats. Zertifikatsverwaltungsoperationen verwenden nur eine certificateId.

`certificatePem`

Die Zertifikatdaten im PEM-Format.

CreateCertificateFromCsr Fehler

Um Fehlerantworten zu empfangen, abonnieren Sie `$aws/certificates/create-from-csr/`*`payload-format`*`/rejected`.

`payload-format`

Das Nachrichtennutzlastformat ist `cbor` oder `json`.

CreateCertificateFromCsr Fehler bei der Nutzlast

```
{
    "statusCode": int,
    "errorCode": "string",
    "errorMessage": "string"
}
```

`statusCode`

Welcher Statuscode gesendet wird

`errorCode`

Der Fehlercode.

`errorMessage`

Die Fehlermeldung.

CreateKeysAndCertificate

Erstellt neue Schlüssel und ein Zertifikat. AWS IoT stellt Client-Zertifikate bereit, die von der Amazon Root Certificate Authority (CA) signiert sind. Das neue Zertifikat hat den Status `PENDING_ACTIVATION`. Wenn Sie `RegisterThing` aufrufen, um ein Objekt mit diesem Zertifikat bereitzustellen, ändert sich der Zertifikatstatus in `ACTIVE` oder `INACTIVE`, wie in der Vorlage beschrieben.

> **Note**
>
> Aus Sicherheitsgründen läuft das von `CreateKeysAndCertificate` ausgegebene `certificateOwnershipToken` zurückgesendet nach einer Stunde ab. `RegisterThing` muss aufgerufen werden, bevor `certificateOwnershipToken` abläuft. Wenn das von erstellte Zertifikat `CreateKeysAndCertificate` bis zum Ablauf des Tokens nicht aktiviert und an eine Richtlinie oder eine Sache angehängt wurde, wird das Zertifikat gelöscht. Wenn das Token abläuft, kann das Gerät `CreateKeysAndCertificate` erneut aufrufen, um ein neues Zertifikat zu generieren.

CreateKeysAndCertificate Anfrage

Veröffentlichen Sie eine Nachricht unter `$aws/certificates/create/`*`payload-format`* mit einer leeren Nachrichtennutzlast.

`payload-format`

Das Nachrichtennutzlastformat ist `cbor` oder `json`.

CreateKeysAndCertificate Antwort

Abonnieren Sie `$aws/certificates/create/`*`payload-format`*`/accepted`.

`payload-format`

Das Nachrichtennutzlastformat ist `cbor` oder `json`.

CreateKeysAndCertificate Antwort

```
{
    "certificateId": "string",
    "certificatePem": "string",
    "privateKey": "string",
    "certificateOwnershipToken": "string"
}
```

`certificateId`

Die ID des Zertifikats.

`certificatePem`

Die Zertifikatdaten im PEM-Format.

`privateKey`

Der private Schlüssel.

`certificateOwnershipToken`

Das Token, um den Besitz des Zertifikats während der Bereitstellung nachzuweisen.

CreateKeysAndCertificate Fehler

Um Fehlerantworten zu empfangen, abonnieren Sie `$aws/certificates/create/`*`payload-format`*`/rejected`.

`payload-format`

Das Nachrichtennutzlastformat ist `cbor` oder `json`.

CreateKeysAndCertificate Fehler bei der Nutzlast

```
{
    "statusCode": int,
    "errorCode": "string",
    "errorMessage": "string"
}
```

`statusCode`

Welcher Statuscode gesendet wird

`errorCode`

Der Fehlercode.

`errorMessage`

Die Fehlermeldung.

RegisterThing

Stellt ein Objekt anhand einer vordefinierten Vorlage bereit.

RegisterThing Anfrage

Veröffentlichen Sie eine Nachricht unter `$aws/provisioning-templates/`*`templateName`*`/provision/`*`payload-format`*.

`payload-format`

Das Nachrichtennutzlastformat ist `cbor` oder `json`.

`templateName`

Der Name der Bereitstellungsvorlage.

RegisterThing Nutzlast anfordern

```
{
    "certificateOwnershipToken": "string",
    "parameters": {
```

```
        "string": "string",
        ...
    }
}
```

`certificateOwnershipToken`

Das Token zum Nachweis der Inhaberschaft des Zertifikats. AWS IoT generiert das Token, wenn Sie ein Zertifikat über MQTT erstellen.

`parameters`

Fakultativ. Key-value Paare vom Gerät, die von den Pre-Provisioning-Hooks zur Auswertung der Registrierungsanforderung verwendet werden.

RegisterThing Antwort

Abonnieren Sie `$aws/provisioning-templates/`*`templateName`*`/provision/`*`payload-format`*`/accepted`.

`payload-format`

Das Nachrichtennutzlastformat ist `cbor` oder `json`.

`templateName`

Der Name der Bereitstellungsvorlage.

RegisterThing Nutzlast der Antwort

```
{
    "deviceConfiguration": {
        "string": "string",
        ...
    },
    "thingName": "string"
}
```

`deviceConfiguration`

Die in der Vorlage definierte Gerätekonfiguration.

`thingName`

Der Name des IoT-Objekts, das während der Bereitstellung erstellt wurde.

RegisterThing Fehlerantwort

Um Fehlerantworten zu empfangen, abonnieren Sie `$aws/provisioning-templates/`*`templateName`*`/provision/`*`payload-format`*`/rejected`.

`payload-format`

Das Nachrichtennutzlastformat ist `cbor` oder `json`.

`templateName`

Der Name der Bereitstellungsvorlage.

RegisterThing Nutzlast für die Fehlerantwort

```
{
    "statusCode": int,
    "errorCode": "string",
    "errorMessage": "string"
}
```

`statusCode`

Welcher Statuscode gesendet wird

`errorCode`

Der Fehlercode.

`errorMessage`

Die Fehlermeldung.

-Flottenindizierung

Sie können die Flottenindizierung verwenden, um die Daten Ihrer Geräte aus den folgenden Quellen zu indizieren, zu durchsuchen und zu aggregieren: AWS IoT Registrierung, AWS IoT Device Shadow, AWS IoT Konnektivität, AWS IoT Geräteverwaltungs-Softwarepaketkatalog und AWS IoT Device DefenderVerstöße. Sie können eine Gruppe von Geräten abfragen und Statistiken zu Gerätedatensätzen aggregieren, die auf verschiedenen Kombinationen von Geräteattributen basieren, darunter Status, Konnektivität und Geräteverstöße. Mit der Flottenindizierung können Sie Ihre Geräteflotte organisieren, untersuchen und Fehler beheben.

Die Flottenindizierung bietet die folgenden Funktionen.

Verwalten von Indexaktualisierungen

Sie können einen Flottenindex einrichten, um Updates für Ihre Objektgruppen, Objektregister, Geräteschatten, Gerätekonnektivität und Geräteverletzungen zu indexieren. Wenn Sie die Flottenindizierung aktivieren, wird AWS IoT ein Index für Ihre Objekte oder Objektgruppen erstellen. `AWS_Things` ist der Index, der für all Ihre Objekte erstellt wurde. `AWS_ThingGroups` ist der Index, der all Ihre Objektgruppen enthält. Wenn dies aktiviert ist, können Sie Abfragen für Ihren Index ausführen. Sie können beispielsweise alle Geräte finden, die tragbar sind und eine Akkulaufzeit von mehr als 70 Prozent haben. AWS IoT aktualisiert den Index kontinuierlich mit Ihren neuesten Daten. Weitere Informationen finden Sie unter Verwalten der Flottenindexierung.

Der Konnektivitätsstatus für ein bestimmtes Gerät wird abgefragt

Diese API ermöglicht den Zugriff auf die neuesten gerätespezifischen Konnektivitätsinformationen mit niedriger Latenz und hohem Durchsatz. Weitere Informationen finden Sie unter Status der Gerätekonnektivität.

Datenquellenübergreifende Suche

Sie können eine Abfragezeichenfolge auf der Grundlage einer Abfragesprache erstellen und diese für die datenquellenübergreifende Suche verwenden. Sie müssen auch Datenquellen in der Flottenindizierungseinstellung so konfigurieren, dass die Indizierungskonfiguration die Datenquellen enthält, aus denen Sie suchen möchten. Die Abfragezeichenfolge beschreibt die Objekte, die Sie suchen möchten. Sie können Abfragen mithilfe AWS verwalteter Felder, benutzerdefinierter Felder

und beliebiger Attribute aus Ihren indizierten Datenquellen erstellen. Weitere Informationen zu Datenquellen, die die Flottenindizierung unterstützen, finden Sie unter Verwaltung der Indexierung von Objekten.

Abfragen von Aggregatdaten

Sie können Ihre Geräte nach aggregierten Daten durchsuchen und Statistiken, Perzentile, Kardinalität oder eine Liste von Objekten mit Suchanfragen zu bestimmten Feldern zurückgeben. Sie können Aggregationen für AWS verwaltete Felder oder alle Attribute ausführen, die Sie in den Flottenindizierungseinstellungen als benutzerdefinierte Felder konfigurieren. Weitere Informationen zur Aggregationsabfrage finden Sie unter Abfragen aggregierter Daten.

Überwachung aggregierter Daten und Erstellung von Alarmen mithilfe von Flottenkennzahlen

Sie können Flottenkennzahlen verwenden, um aggregierte Daten CloudWatch automatisch an diese zu senden, Trends zu analysieren und Alarme zu erstellen, um den Gesamtzustand Ihrer Flotte auf der Grundlage vordefinierter Schwellenwerte zu überwachen. Weitere Informationen zu Metriken erhalten Sie unter Ressourcenmetriken.

Verwalten der Flottenindizierung

Flottenindizierung verwaltet zwei Arten von Indizes für Sie: die Objektindizierung und die Objektgruppenindizierung.

Objektindizierung

`AWS_Things` ist der Index für all Ihre Objekte. Die Objektindizierung unterstützt die folgenden Datenquellen: AWS IoT Registrierungsdaten, AWS IoT Device Schatten-Daten, AWS IoT Konnektivitätsdaten und Daten zu AWS IoT Device DefenderVerstößen. Indem Sie diese Datenquellen zu Ihrer Flottenindexierungskonfiguration hinzufügen, können Sie nach Objekten suchen, aggregierte Daten abfragen und dynamische Objektgruppen und Flottenmetriken auf der Grundlage Ihrer Suchanfragen erstellen.

Registry — AWS IoT bietet eine Registrierung, die Ihnen bei der Verwaltung hilft. Sie können die Registrierungsdaten zu Ihrer Flottenindexierungskonfiguration hinzufügen, um anhand der Objektnamen, Beschreibungen und anderen Registrierungsattributen nach Geräten zu suchen.

Weitere Informationen zur -Registrierung finden Sie unter Verwalten von Aufgaben mit der -Registrierung.

Shadow — Der AWS IoT Device Shadow-Dienst bietet Schatten, mit denen Sie Ihre Gerätestatusdaten speichern können. Die Indizierung von Objekten unterstützt sowohl klassische unbenannte Schatten als auch benannte Schatten. Um benannte Schatten zu indizieren, aktivieren Sie Ihre Einstellungen für benannte Schatten und geben Sie Ihre Schattennamen in der Konfiguration für die Objektindizierung an. Standardmäßig können Sie bis zu 10 Shadow-Namen pro Person hinzufügen AWS-Konto. Informationen darüber, wie Sie die Obergrenze für die Anzahl der Schattennamen erhöhen können, finden Sie in der AWS allgemeinen Referenz unter AWS IoT Device Management Kontingente.

So fügen Sie benannte Schatten für die Indizierung hinzu:

- Wenn Sie die AWS IoT Konsole verwenden, aktivieren Sie die Objektindizierung, wählen Sie Benannte Schatten hinzufügen und fügen Sie Ihre Schattennamen über die Auswahl benannter Schatten hinzu.
- Wenn Sie die AWS Command Line Interface (AWS CLI) verwenden, setzen Sie `namedShadowIndexingMode` sie auf be `ON` und geben Sie Schattennamen in an `IndexingFilter`. Beispiele für CLI-Befehle finden Sie unter Verwaltung der Dingindizierung.

Weitere Informationen zu Schatten finden Sie unter AWS IoT Device Schatten-Service.

Konnektivität — Mithilfe der Daten zur Gerätekonnektivität können Sie den Verbindungsstatus Ihrer Geräte ermitteln. Diese Konnektivitätsdaten werden durch Lebenszyklusereignisse bestimmt. Wenn ein Client eine Verbindung herstellt oder trennt, AWS IoT veröffentlicht er Lebenszyklusereignisse mit Meldungen zu MQTT-Themen. Eine Verbindungs- oder Trennungsnachricht kann eine Liste von JSON-Elementen sein, die Details zum Verbindungsstatus enthalten, einschließlich der Client-ID, der Keep-Alive-Dauer, der Clean-Sitzungskennzeichnung und des Ablaufs der Sitzung. Weitere Informationen zur Gerätekonnektivität finden Sie unter Lebenszyklusereignisse.

Device AWS IoT Device Defender Defender-Verstöße — Daten zu Verstößen helfen dabei, anomales Geräteverhalten im Vergleich zu den normalen Verhaltensweisen zu identifizieren, die Sie in einem Sicherheitsprofil definieren. Ein Sicherheitsprofil enthält eine Reihe von Verhaltensweisen. Für jedes Verhalten wird eine Metrik verwendet, die das normale Verhalten Ihrer Geräte angibt. Weitere Informationen zu Device Defender-Verstößen finden AWS IoT Device Defender Sie unter Erkennen.

Weitere Informationen finden Sie unter Verwalten der Objektindizierung.

Modus für die Objektgruppenindizierung.

`AWS_ThingGroups` ist der Index, die alle Ihre Objektgruppen enthält. Dieser Index ermöglicht Ihnen die Suche nach Gruppen basierend auf Gruppennamen, Beschreibung, Attributen und allen übergeordneten Gruppennamen.

Weitere Informationen finden Sie unter Verwalten der Objektgruppenindizierung.

Verwaltete Felder

Verwaltete Felder enthalten Daten, die Dingen, Dinggruppen, Geräteschatten, Gerätekonnektivität und Device Defender-Verstößen zugeordnet sind. AWS IoT definiert den Datentyp in verwalteten Feldern. Sie geben die Werte jedes verwalteten Felds an, wenn Sie ein Objekt AWS IoT erstellen. Beispielsweise sind Objektnamen, Objektgruppen und Objektbeschreibungen alles verwaltete Felder. Der Flottenindizierungsservice indiziert verwaltete Felder basierend auf dem von Ihnen angegebenen Indizierungsmodus: Verwaltete Felder können nicht geändert oder in `customFields` angezeigt werden. Weitere Informationen finden Sie unter Benutzerdefinierte Felder.

Im Folgenden sind verwaltete Felder für die Indizierung von Objekten aufgeführt:

- Verwaltete Felder für die Registrierung

```
"managedFields" : [
  {name:thingId, type:String},
  {name:thingName, type:String},
  {name:registry.version, type:Number},
  {name:registry.thingTypeName, type:String},
  {name:registry.thingGroupNames, type:String},
]
```

- Verwaltete Felder für klassische unbenannte Schatten

```
"managedFields" : [
  {name:shadow.version, type:Number},
  {name:shadow.hasDelta, type:Boolean}
]
```

- Verwaltete Felder für benannte Schatten

```
"managedFields" : [
  {name:shadow.name.shadowName.version, type:Number},
```

```
  {name:shadow.name.shadowName.hasDelta, type:Boolean}
]
```

- Verwaltete Felder für die Objektkonnektivität

```
"managedFields" : [
  {name:connectivity.timestamp, type:Number},
  {name:connectivity.version, type:Number},
  {name:connectivity.connected, type:Boolean},
  {name:connectivity.disconnectReason, type:String},
  {name:connectivity.clientId, type:String},
  {name:connectivity.keepAliveDuration, type:Number},
  {name:connectivity.cleanSession, type:Boolean},
  {name:connectivity.sessionExpiry, type:Number}
]
```

- Verwaltete Felder für Device Defender

```
"managedFields" : [
  {name:deviceDefender.violationCount, type:Number},
  {name:deviceDefender.securityprofile.behaviorname.metricName, type:String},
  {name:deviceDefender.securityprofile.behaviorname.lastViolationTime, type:Number},
  {name:deviceDefender.securityprofile.behaviorname.lastViolationValue, type:String},
  {name:deviceDefender.securityprofile.behaviorname.inViolation, type:Boolean}
]
```

- Verwaltete Felder für Objektgruppen

```
"managedFields" : [
  {name:description, type:String},
  {name:parentGroupNames, type:String},
  {name:thingGroupId, type:String},
  {name:thingGroupName, type:String},
  {name:version, type:Number},
]
```

In der folgenden Tabelle sind verwaltete Felder aufgeführt, die nicht durchsucht werden können.

Datenquelle	Verwaltetes Feld, das nicht durchsucht werden kann
Registrierung	`registry.version`
Unbenannte Schatten	`shadow.version`
Benannter Schatten	`shadow.name.*.version`
Device Defender	`deviceDefender.version`
Objektgruppen	`version`

Benutzerdefinierte Felder

Sie können Objekt-Attribute, Device Schatten-Daten und Daten zu Device Defender-Verstößen aggregieren, indem Sie benutzerdefinierte Felder erstellen, um sie zu indizieren. Das Attribut `customFields` ist eine Liste von Feld- und Datentyppaaren. Sie können Aggregationsabfragen auf der Grundlage des Datentyps durchführen. Der von Ihnen gewählte Indizierungsmodus wirkt sich auf Felder aus, in denen Sie angeben können. `customFields` Wenn Sie beispielsweise den Indizierungsmodus `REGISTRY` angeben, können Sie kein Feld aus einem Schattenobjekt angeben. Sie können den CLI-Befehl update-indexing-configuration verwenden, um die benutzerdefinierten Felder zu erstellen oder zu aktualisieren (einen Beispielbefehl finden Sie unter Beispiele zur Aktualisierung der Indexierungskonfiguration).

Namen benutzerdefinierter Felder

Benutzerdefinierte Feldnamen für Objekt- und Objektgruppenattribute beginnen mit `attributes.`, gefolgt vom Attributnamen. Wenn die unbenannte Schattenindizierung aktiviert ist, können Objekte benutzerdefinierte Feldnamen haben, die mit `shadow.desired` oder`shadow.reported` beginnen, gefolgt vom Namen des unbenannten Schattendatenwerts. Wenn die benannte Schattenindizierung aktiviert ist, können Objekte benutzerdefinierte Feldnamen haben, die mit `shadow.name.*.desired.` oder `shadow.name.*.reported.` beginnen, gefolgt vom benannten Schattendatenwert. Wenn die Device Defender-Indizierung für Verstöße aktiviert ist, können Objekte benutzerdefinierte Feldnamen haben, die mit `deviceDefender.`, gefolgt vom Datenwert Device Defender-Verletzungen beginnen.

Der Name des Attributs oder Datenwerts, der auf das Präfix folgt, darf nur aus alphanumerischen Zeichen, Bindestrichen und Unterstrichen bestehen. Er darf keine Leerzeichen enthalten.

Wenn zwischen einem benutzerdefinierten Feld in der Konfiguration und dem indizierten Wert eine Typinkonsistenz besteht, ignoriert der Flottenindizierungsservice den inkonsistenten Wert für Aggregationsabfragen. -Protokolle sind hilfreich bei der Behebung von Problemen mit Aggregationsabfragen. CloudWatch Protokolle sind hilfreich bei der Behebung von Problemen mit Aggregationsabfragen. Weitere Informationen finden Sie unter Fehlerbehebung bei Aggregationsabfragen für den Flottenindizierungsservice.

Benutzerdefinierte Feldtypen

Benutzerdefinierte Feldtypen haben die folgenden unterstützten Werte: `Number`, `String` und `Boolean`.

Verwalten der Objektindizierung

`AWS_Things` ist der Index für all Ihre Objekte. Sie können aus den folgenden Datenquellen steuern, was indexiert werden soll: AWS IoT Registrierungsdaten, AWS IoT Device Schatten-Daten, AWS IoT Konnektivitätsdaten und Daten zu AWS IoT Device DefenderVerstößen.

In diesem Thema:

- Aktivieren der Objektindizierung
- Beschreiben eines Objektindex
- Abfragen des Objektindex
- Beschränkungen und Einschränkungen
- Autorisierung

Aktivieren der Objektindizierung

Sie verwenden den CLI-Befehl update-indexing-configuration oder die UpdateIndexingConfiguration API-Operation, um den `AWS_Things` Index zu erstellen und seine Konfiguration zu steuern. Mit dem Parameter `--thing-indexing-configuration` (`thingIndexingConfiguration`) können Sie steuern, welche Art von Daten indiziert werden (z. B. Registrierungs-, Schatten- und Gerätekonnektivitätsdaten).

Der Parameter `--thing-indexing-configuration` nimmt eine Zeichenfolge mit der folgenden Struktur an:

```
{
  "thingIndexingMode": "OFF"|"REGISTRY"|"REGISTRY_AND_SHADOW",
  "thingConnectivityIndexingMode": "OFF"|"STATUS",
  "deviceDefenderIndexingMode": "OFF"|"VIOLATIONS",
  "namedShadowIndexingMode": "OFF"|"ON",
  "managedFields": [
    {
      "name": "string",
      "type": "Number"|"String"|"Boolean"
    },
    ...
  ],
  "customFields": [
    {
      "name": "string",
      "type": "Number"|"String"|"Boolean"
    },
    ...
  ],
  "filter": {
     "namedShadowNames": [ "string" ],
     "geoLocations": [
        {
            "name": "String",
            "order": "LonLat|LatLon"
        }
    ],
     "connectivity": {
        "includeSocketInformation": [
           "GET_THING_CONNECTIVITY_DATA"
        ]
     }
  }
}
```

Objektindizierungsmodus.

Sie können in Ihrer Indizierungskonfiguration verschiedene Indizierungsmodi angeben, je nachdem, aus welchen Datenquellen Sie Geräte indizieren und durchsuchen möchten:

- `thingIndexingMode`: Steuert, ob Registry oder Schatten indexiert ist. Wenn `thingIndexingMode` auf `OFF` eingestellt ist, ist die Indizierung von Objekten deaktiviert.

- `thingConnectivityIndexingMode`: Gibt an, ob die Objektkonnektivitätsdaten indiziert sind. Dies muss aktiviert sein, um Informationen zum Konnektivitätsstatus in SearchIndex Abfragen abzurufen oder die API zu verwenden. GetThingConnectivityData Die GetThingConnectivityData API-Antwort enthält auch Socket-Informationen, wenn dies im Filter aktiviert ist und das `includeSocketInformation` Flag `true` in der GetThingConnectivityData API-Anfrage enthalten ist.

- `deviceDefenderIndexingMode`: Gibt an, ob Device Defender-Daten, die Verstöße verletzen, indexiert werden.
- `namedShadowIndexingMode`: Gibt an, ob benannte Schatten-Daten indexiert werden. Um benannte Schatten auszuwählen, die zu Ihrer Flottenindizierungskonfiguration hinzugefügt werden sollen, legen Sie `namedShadowIndexingMode` als `ON` fest und geben Sie Ihre benannten Schattennamen unter `filter` an.

Die folgende Tabelle zeigt die gültigen Werte für jeden Indizierungsmodus und die Datenquelle, die für jeden Wert indexiert ist.

Attribut	Zulässige Werte	Registrierung	Shadow	Konnektivität	DD-Verstöße	Benannter Schatten
`thingIndexingMode`	OFF					
	REGISTRY	✓				
	REGISTRY_AND_SHADOW	✓	✓			
`thingConnectivityIndexingMode`	Nicht angegeben.					
	OFF					
	STATUS			✓		
`deviceDefenderIndexingMode`	Nicht angegeben.					

Attribut	Zulässige Werte	Registrierung	Shadow	Konnektivität	DD-Verstöße	Benannter Schatten
	OFF					
	Verstöße				✓	
`namedShadowIndexingMode`	Nicht angegeben.					
	OFF					
	ON					✓

Verwaltete Felder und benutzerdefinierte Felder

Verwaltete Felder

Verwaltete Felder enthalten Daten, die mit Dingen, Dinggruppen, Geräteschatten, Gerätekonnektivität und Device Defender-Verstößen verknüpft sind. AWS IoT definiert den Datentyp in verwalteten Feldern. Beim Erstellen eines IoT-Objekts geben Sie die Werte jedes verwalteten Felds an. Beispielsweise sind Objektnamen, Objektgruppen und Objektbeschreibungen alles verwaltete Felder. Der Flottenindizierungsservice indiziert verwaltete Felder basierend auf dem von Ihnen angegebenen Indizierungsmodus: Verwaltete Felder können nicht geändert oder in `customFields` angezeigt werden.

Benutzerdefinierte Felder

Sie können Attribute, Device Schatten-Daten und Daten zu Device Defender-Verstößen aggregieren, indem Sie benutzerdefinierte Felder erstellen, um sie zu indizieren. Das Attribut `customFields` ist eine Liste von Feld- und Datentyppaaren. Sie können Aggregationsabfragen auf der Grundlage des Datentyps durchführen. Der von Ihnen gewählte Indizierungsmodus wirkt sich auf Felder aus, in denen Sie angeben können. `customFields` Wenn Sie beispielsweise den Indizierungsmodus `REGISTRY` angeben, können Sie kein Feld aus einem Schattenobjekt angeben. Sie können den CLI-Befehl update-indexing-configuration verwenden, um die benutzerdefinierten Felder zu erstellen oder zu aktualisieren (einen Beispielbefehl finden Sie unter Beispiele zur Aktualisierung der Indexierungskonfiguration). Weitere Informationen zu RDS CUstom finden Sie unter .

Indizierungsfilter

Der Indexfilter bietet zusätzliche Auswahlmöglichkeiten für benannte Schatten, Geolokalisierungs- und Konnektivitätsdaten.

`namedShadowNames`

Um Ihrer Flottenindizierungskonfiguration benannte Schatten hinzuzufügen, legen Sie `namedShadowIndexingMode` als `ON` fest und geben Sie Ihre benannten Schattennamen im `namedShadowNames`-Filter an.

Beispiel

```
"filter": {
     "namedShadowNames": [ "namedShadow1", "namedShadow2" ]
}
```

`geoLocations`

So fügen Sie Ihrer Flottenindexierungskonfiguration Geolokalisierungsdaten hinzu:

- Wenn Ihre Geolokalisierungsdaten in einem klassischen (unbenannten) Schatten gespeichert sind, legen Sie `thingIndexingMode` auf REGISTRY_AND_SHADOW fest und geben Sie Ihre Geolokalisierungsdaten im `geoLocations`-Filter an.

  Der folgende Beispielfilter spezifiziert ein GeoLocation-Objekt in einem klassischen (unbenannten) Schatten:

  ```
  "filter": {
       "geoLocations": [
          {
              "name": "shadow.reported.location",
              "order": "LonLat"
          }
       ]
    }
  ```

- Wenn Ihre Geolokalisierungsdaten in einem benannten Schatten gespeichert sind, setzen Sie `namedShadowIndexingMode` auf ON, fügen Sie den Schattennamen im `namedShadowNames`-Filter hinzu und geben Sie Ihre Geolokalisierungsdaten im `geoLocations`-Filter an.

Der folgende Beispielfilter spezifiziert ein GeoLocation-Objekt in einem benannten Schatten (nameShadow1):

```
"filter": {
     "namedShadowNames": [ "namedShadow1" ],
     "geoLocations": [
        {
            "name": "shadow.name.namedShadow1.reported.location",
            "order": "LonLat"
        }
     ]
  }
```

`connectivity`

- Um Socket-Informationen (Quell-IP-Adresse, Ziel-IP-Adresse, Quellport, Zielport und VPC-Endpunkt-ID) in die `GetThingConnectivityData` Antwort aufzunehmen, verwenden Sie die `includeSocketInformation` Option im Konnektivitätsfilter, wie im folgenden Beispiel gezeigt. Sie müssen den `includeSocketInformation` Parameter auch auf einstellen, `true` wenn Sie die `GetThingConnectivityData` API aufrufen. Um bestimmte Aufrufer am Zugriff auf Socket-Informationen zu hindern, müssen Sie dies in ihrer IAM-Richtlinie explizit angeben, indem Sie den `iot:IncludeSocketInformation` Bedingungskontextschlüssel auf setzen. `false`

> **Note**
>
> Die Option, Socket-Informationen einzubeziehen, wird nur für die `GetThingConnectivityData` Antwort und nicht für `SearchIndex` andere APIs zur Flottenindizierung unterstützt.

```
"filter": {
  "connectivity": {
    "includeSocketInformation": [
      "GET_THING_CONNECTIVITY_DATA"
    ]
  }
}
```

Um die Indizierung von Socket-Informationen zu deaktivieren, müssen Sie `includeSocketInformation` im Konnektivitätsfilter ein leeres Array festlegen, wie im folgenden Beispiel gezeigt.

> **Note**
>
> Wenn in einer `UpdateIndexingConfiguration` Anforderung die Konnektivitätsindizierung aktiviert ist und ein Konnektivitätsfilter nicht konfiguriert oder nicht angegeben oder Null `includeSocketInformation` ist, wird der vorherige Wert des Konnektivitätsfilters beim Aktualisieren der Indizierungskonfiguration beibehalten.

Beispiel zum Deaktivieren der Indizierung von Socket-Informationen

```
"filter": {
  "connectivity": {
    "includeSocketInformation": [ ]
   }
  }
```

Weitere Informationen finden Sie in IndexingFilter der AWS IoT API-Referenz.

Aktualisieren von Konfigurationsbeispielen für die Indexierung

Verwenden Sie den AWS IoT update-indexing-configuration CLI-Befehl, um Ihre Indexierungskonfiguration zu aktualisieren. Im folgenden Beispiel wird gezeigt, wie update-indexing-configuration verwendet wird.

Kurze Syntax:

```
aws iot update-indexing-configuration --thing-indexing-configuration \
'thingIndexingMode=REGISTRY_AND_SHADOW, deviceDefenderIndexingMode=VIOLATIONS,
namedShadowIndexingMode=ON,thingConnectivityIndexingMode=STATUS,
filter={namedShadowNames=[namedShadow1],geoLocations=[{name=shadow.name.namedShadow1.reported.l
customFields=[{name=attributes.version,type=Number},
{name=shadow.name.namedShadow1.desired.DefaultDesired, type=String},
 {name=shadow.desired.power, type=Boolean},
{name=deviceDefender.securityProfile1.NUMBER_VALUE_BEHAVIOR.lastViolationValue.number,
 type=Number}]'
```

JSON-Syntax:

```
aws iot update-indexing-configuration --cli-input-json \ '{
          "thingIndexingConfiguration": { "thingIndexingMode": "REGISTRY_AND_SHADOW",
          "thingConnectivityIndexingMode": "STATUS",
          "deviceDefenderIndexingMode": "VIOLATIONS",
          "namedShadowIndexingMode": "ON",
          "filter": { "namedShadowNames": ["namedShadow1"], "geoLocations": [{"name":
 "shadow.name.namedShadow1.reported.location", "order": "LonLat"}], "connectivity":
 {"includeSocketInformation": ["GET_THING_CONNECTIVITY_DATA"]}},
          "customFields": [ { "name": "shadow.desired.power", "type": "Boolean" },
          {"name": "attributes.version", "type": "Number"},
          {"name": "shadow.name.namedShadow1.desired.DefaultDesired", "type":
 "String"},
          {"name":
 "deviceDefender.securityProfile1.NUMBER_VALUE_BEHAVIOR.lastViolationValue.number",
 "type": "Number"} ] } }'
```

Dieser Befehl liefert keine Ausgabe.

Führen Sie den describe-index CLI-Befehl aus, um den Status des Dingindexes zu überprüfen:

```
aws iot describe-index --index-name "AWS_Things"
```

Die Ausgabe des Befehls describe-index sieht wie folgt aus:

```
{
    "indexName": "AWS_Things",
    "indexStatus": "ACTIVE",
    "schema": "MULTI_INDEXING_MODE"
}
```

> **Note**
>
> Es kann einen Moment dauern, bis der Flottenindex bei der Flottenindizierung aktualisiert ist. Wir empfehlen, mit der Verwendung zu warten, bis die `indexStatus` Anzeige AKTIV angezeigt wird. Je nachdem, welche Datenquellen Sie konfiguriert haben, können Sie im Schemafeld unterschiedliche Werte angeben. Weitere Informationen finden Sie unter Beschreiben eines Objektindizents.

Führen Sie den get-indexing-configuration CLI-Befehl aus, um die Konfigurationsdetails für die Indexierung Ihres Objekts abzurufen:

```
aws iot get-indexing-configuration
```

Die Ausgabe des Befehls get-indexing-configuration sieht wie folgt aus:

```
{
    "thingIndexingConfiguration": {
        "thingIndexingMode": "REGISTRY_AND_SHADOW",
        "thingConnectivityIndexingMode": "STATUS",
        "deviceDefenderIndexingMode": "VIOLATIONS",
        "namedShadowIndexingMode": "ON",
        "managedFields": [
            {
                "name": "connectivity.keepAliveDuration",
                "type": "Number"
            },
            {
                "name": "connectivity.version",
                "type": "Number"
            },
            {
                "name": "deviceDefender.*.*.metricName",
                "type": "String"
            },
            {
                "name": "connectivity.connected",
                "type": "Boolean"
            },
            {
                "name": "shadow.version",
                "type": "Number"
            },
            {
                "name": "deviceDefender.*.*.lastViolationTime",
                "type": "Number"
            },
            {
                "name": "deviceDefender.*.*.inViolation",
                "type": "Boolean"
            },
            {
```

```
            "name": "thingName",
            "type": "String"
        },
        {
            "name": "deviceDefender.version",
            "type": "Number"
        },
        {
            "name": "shadow.name.*.version",
            "type": "Number"
        },
        {
            "name": "registry.version",
            "type": "Number"
        },
        {
            "name": "connectivity.sessionExpiry",
            "type": "Number"
        },
        {
            "name": "registry.thingTypeName",
            "type": "String"
        },
        {
            "name": "deviceDefender.violationCount",
            "type": "Number"
        },
        {
            "name": "connectivity.timestamp",
            "type": "Number"
        },
        {
            "name": "connectivity.clientId",
            "type": "String"
        },
        {
            "name": "thingId",
            "type": "String"
        },
        {
            "name": "shadow.name.*.hasDelta",
            "type": "Boolean"
        },
        {
```

```
                "name": "registry.thingGroupNames",
                "type": "String"
            },
            {
                "name": "shadow.hasDelta",
                "type": "Boolean"
            },
            {
                "name": "connectivity.cleanSession",
                "type": "Boolean"
            },
            {
                "name": "connectivity.disconnectReason",
                "type": "String"
            }
        ],
        "customFields": [
            {
                "name": "attributes.version",
                "type": "Number"
            },
            {
                "name": "shadow.desired.power",
                "type": "Boolean"
            },
            {
                "name": "shadow.name.namedShadow1.desired.DefaultDesired",
                "type": "String"
            },
            {
                "name":
"deviceDefender.securityProfile1.NUMBER_VALUE_BEHAVIOR.lastViolationValue.number",
                "type": "Number"
            }
        ],
        "filter": {
            "namedShadowNames": [
                "namedShadow1"
            ],
            "geoLocations": [
                {
                    "name": "shadow.name.namedShadow1.reported.location",
                    "order": "LonLat"
                }
```

```
            ],
            "connectivity": {
                "includeSocketInformation": [
                    "GET_THING_CONNECTIVITY_DATA"
                ]
            }
        }
    },
    "thingGroupIndexingConfiguration": {
        "thingGroupIndexingMode": "OFF"
    }
}
```

Um die benutzerdefinierten Felder zu aktualisieren, können Sie den update-indexing-configuration Befehl ausführen. Das Beispiel ist wie folgt:

```
aws iot update-indexing-configuration --thing-indexing-configuration

 'thingIndexingMode=REGISTRY_AND_SHADOW,customFields=[{name=attributes.version,type=Number},
{name=attributes.color,type=String},{name=shadow.desired.power,type=Boolean},
{name=shadow.desired.intensity,type=Number}]'
```

Dieser Befehl hat der Indizierungskonfiguration `shadow.desired.intensity` hinzugefügt.

> **Note**
>
> Beim Aktualisieren der benutzerdefinierten Felder in der Indizierungskonfiguration werden alle vorhandenen benutzerdefinierten Felder überschrieben. Achten Sie darauf, beim Aufrufen von update-indexing-configuration alle benutzerdefinierten Felder anzugeben.

Nachdem der Index neu erstellt wurde, können Sie Aggregationsabfrage für die neu hinzugefügten Felder, Registrierungsdaten, Schattendaten und Statusdaten der Objektkonnektivität verwenden.

Stellen Sie beim Ändern des Indizierungsmodus sicher, dass alle benutzerdefinierten Felder im neuen Indizierungsmodus gültig sind. Wenn Sie beispielsweise im Modus `REGISTRY_AND_SHADOW` mit dem benutzerdefinierten Feld `shadow.desired.temperature` beginnen, müssen Sie das benutzerdefinierte Feld `shadow.desired.temperature` löschen, bevor Sie den Indizierungsmodus in `REGISTRY` ändern. Wenn Ihre Indizierungskonfiguration benutzerdefinierte Felder enthält, die nicht vom Indizierungsmodus indiziert werden, schlägt die Aktualisierung fehl.

Beschreiben eines Objektindex

Der folgende Befehl zeigt, wie Sie den CLI-Befehl describe-index zum Abrufen des aktuellen Status des Objektindizes verwenden.

```
aws iot describe-index --index-name "AWS_Things"
```

Die Ausgabe des Befehls sieht wie folgt aus:

```
{
    "indexName": "AWS_Things",
    "indexStatus": "BUILDING",
    "schema": "REGISTRY_AND_SHADOW_AND_CONNECTIVITY_STATUS"
}
```

Wenn Sie zum ersten Mal eine Flottenindizierung durchführen, wird Ihr Index AWS IoT erstellt. Sie können den Index nicht abfragen, wenn sich `indexStatus` im Status `BUILDING` befindet. Das `schema` für den Objektindex zeigt an, welche Art von Daten (`REGISTRY_AND_SHADOW_AND_CONNECTIVITY_STATUS`) indiziert werden.

Wenn die Konfiguration für Ihren Index geändert wird, wird der Index neu erstellt. Der `indexStatus` während dieses Vorgangs lautet `REBUILDING`. Sie können Abfragen für Daten im Objektindex ausführen, während er erstellt wird. Wenn Sie beispielsweise die Indexkonfiguration von `REGISTRY` in `REGISTRY_AND_SHADOW` ändern, während der Index neu erstellt wird, können Sie Registrierungsdaten abfragen, einschließlich der aktuellen Updates. Sie können die Schattendaten jedoch erst abfragen, wenn die Wiederherstellung abgeschlossen ist. Die benötigte Zeit zum Erstellen oder Neuerstellen des Index hängt von der Menge an Daten ab.

Abhängig von den Datenquellen, die Sie konfiguriert haben, können Sie im Schemafeld unterschiedliche Werte sehen. Die folgende Tabelle zeigt die verschiedenen Schemawerte und die entsprechenden Beschreibungen:

Schema	Description
OFF	Es sind keine Datenquellen konfiguriert oder indexiert.
REGISTRY	(Nur Registrierungsdaten werden indiziert.)

Schema	Description
REGISTRY_AND_SHADOW	(Registrierungsdaten und Schatten-Daten werden indiziert.)
REGISTRY_AND_CONNECTIVITY	Registrierungsdaten und Konnektivitätsdaten sind indexiert.
REGISTRY_AND_SHADOW_AND_CONNECTIVITY_STATUS	Registrierungsdaten, unbenannte (klassische) Schattendaten und Konnektivitätsdaten werden indexiert.
MULTI_INDEXING_MODE	Daten zu Verletzungen durch benannte Schatten- oder Device Defender-Angriffe werden zusätzlich zu Registrierungsdaten, unbenannten (klassischen) Schatten- oder Konnektivitätsdaten indexiert.

Abfragen des Objektindex

Verwenden Sie den CLI-Befehl search-index zum Abfragen von Daten im Index.

```
aws iot search-index --index-name "AWS_Things" --query-string
          "thingName:mything*"
```


```
{
  "things": [
    {
      "thingName": "mything1",
      "thingGroupNames": ["mygroup1"],
      "thingId": "a4b9f759-b0f2-4857-8a4b-967745ed9f4e",
      "attributes": {
        "attribute1": "abc"
      },
      "connectivity": {
        "connected": false,
        "timestamp": 1556649874716,
        "disconnectReason": "CONNECTION_LOST",
        "clientId": "mything1",
        "keepAliveDuration": 60,
```

```
        "cleanSession": false,
        "sessionExpiry": 3600
      }
    },
    {
      "thingName": "mything2",
      "thingTypeName": "MyThingType",
      "thingGroupNames": ["mygroup1", "mygroup2"],
      "thingId": "01014ef9-e97e-44c6-985a-d0b06924f2af",
      "attributes": {
        "model": "1.2",
        "country": "usa"
      },
      "shadow": {
        "desired": {
          "location": "new york",
          "myvalues": [3, 4, 5]
        },
        "reported": {
          "location": "new york",
          "myvalues": [1, 2, 3],
          "stats": {
            "battery": 78
          }
        },
        "metadata": {
          "desired": {
            "location": { "timestamp": 123456789 },
            "myvalues": { "timestamp": 123456789 }
          },
          "reported": {
            "location": { "timestamp": 34535454 },
            "myvalues": { "timestamp": 34535454 },
            "stats": {
              "battery": { "timestamp": 34535454 }
            }
          }
        },
        "version": 10,
        "timestamp": 34535454
      },
      "connectivity": {
        "connected": true,
        "timestamp": 1556649855046,
```

```
        "clientId": "mything2",
        "keepAliveDuration": 60,
        "cleanSession": true
      }
    }
  ],
  "nextToken": "AQFCuvk7zZ3D9pOYMbFCeHbdZ+h=G"
}
```

In der JSON-Antwort `"connectivity"` (wie durch die `thingConnectivityIndexingMode=STATUS` Einstellung aktiviert) werden der Verbindungsstatus, der Zeitstempel, der Grund für die Trennung, die Client-ID, die Keep-Alive-Dauer, das Flag für eine saubere Sitzung und der Ablauf der Sitzung für das Gerät angegeben. Das Gerät `"mything1"` wurde (`false`) zur POSIX-Zeit aufgrund von `CONNECTION_LOST` getrennt: Weitere Informationen zu den Gründen für die Unterbrechung der Verbindung finden Sie unter Lifecycle-Ereignisse.

```
"connectivity": {
  "connected": false,
  "timestamp": 1556649874716,
  "disconnectReason": "CONNECTION_LOST",
  "clientId": "mything1",
  "keepAliveDuration": 60,
  "cleanSession": false,
  "sessionExpiry": 3600
}
```

Das Gerät `"mything2"` wurde (`true`) zur POSIX-Zeit `1556649855046` verbunden:

```
"connectivity": {
  "connected": true,
  "timestamp": 1556649855046,
  "clientId": "mything2",
  "keepAliveDuration": 60,
  "cleanSession": true
}
```

Da Zeitstempel in Millisekunden seit der Epoche angegeben werden, stellt `1556649855046` 6:44:15.046 PM am Dienstag, 30. April 2019 (GMT), dar.

> **Important**
>
> Wenn ein Gerät noch nie eine Verbindung zu IoT Core hergestellt hat oder die Verbindung länger als 1 Stunde unterbrochen wurde, bevor die Flottenindizierung aktiviert `thingConnectivityIndexingMode` wurde, wird das `connected` Feld in der Antwort auf ohne zusätzliche Sitzungsdetails gesetzt. `false`

Beschränkungen und Einschränkungen

Dies sind die Einschränkungen und Begrenzungen für `AWS_Things`.

Schatten-Felder mit komplexen Typen

Ein Schattenfeld wird nur indiziert, wenn der Wert des Felds einfach ist, ein JSON-Objekt, das kein Array enthält, oder ein Array, das vollständig aus einfachen Typen besteht. Mit „einfacher Typ“ ist eine Zeichenfolge, eine Zahl oder eines der Literale `true` oder `false` gemeint. Beispielsweise angesichts des folgenden Schattenzustands wird der Wert des Feldes `"palette"` nicht indiziert, da es sich um ein Array handelt, das Elemente komplexen Typs enthält. Der Wert des Feldes `"colors"` wird indiziert, da jeder Wert im Array eine Zeichenfolge ist.

```
{
    "state": {
        "reported": {
            "switched": "ON",
            "colors": [ "RED", "GREEN", "BLUE" ],
            "palette": [
                {
                    "name": "RED",
                    "intensity": 124
                },
                {
                    "name": "GREEN",
                    "intensity": 68
                },
                {
                    "name": "BLUE",
                    "intensity": 201
                }
            ]
        }
```

```
    }
}
```

Verschachtelte Schattenfeldnamen

Die Namen verschachtelter Schattenfelder werden als Zeichenfolge mit Punkt als Trennzeichen (.) gespeichert. Beispielsweise bei einem Schattendokument:

```
{
  "state": {
    "desired": {
      "one": {
        "two": {
          "three": "v2"
        }
      }
    }
  }
}
```

Der Name des Feldes `three` wird als `desired.one.two.three` gespeichert. Wenn Sie auch ein Schattendokument wie das folgende haben:

```
{
  "state": {
    "desired": {
      "one.two.three": "v2"
    }
  }
}
```

Beide entsprechen einer Abfrage für `shadow.desired.one.two.three:v2`. Eine bewährte Methode besteht darin, keine Punkte in Schattenfeldnamen zu verwenden.

Schatten-Metadaten

Ein Feld in einem Schatten-Metadatenbereich wird indiziert, aber nur dann, wenn das entsprechende Feld in dem `"state"`-Abschnitt des Shadow indiziert ist. (Im vorherigen Beispiel wird das Feld `"palette"` im Metadatenbereich des Schattens auch nicht indiziert.)

Alle nicht registrierten Geräte

Bei der Flottenindizierung werden keine Daten für nicht registrierte Geräte indexiert. Dinge müssen in der AWS IoT Registrierung registriert sein und `clientId` beim Herstellen der Verbindung den Namen des Dings verwenden. AWS IoT Core

Nicht registrierte Schatten

Wenn Sie UpdateThingShadow einen Shadow mit einem Dingnamen erstellen, der nicht in Ihrem AWS IoT Konto registriert ist, werden Felder in diesem Shadow nicht indexiert. Dies gilt sowohl für den klassischen unbenannten Schatten als auch für den benannten Schatten.

Numerische Werte

Wenn Registrierungs- oder Schattendaten von dem Service als numerischer Wert erkannt werden, werden sie als solche indiziert. Sie können Abfragen erstellen, die Bereiche und Vergleichsoperatoren für numerische Werte umfassen (beispielsweise `"attribute.foo<5"` oder `"shadow.reported.foo:[75 TO 80]"`). Um als numerisch erkannt zu werden, muss es sich bei dem Wert der Daten um eine gültige JSON-Nummer vom Literaltyp handeln. Um als numerisch erkannt zu werden, muss der Wert der Daten ein gültiges JSON-Literal des Typs „Zahl“ (eine Ganzzahl von -2^53...2^53-1 oder ein Gleitkomma mit doppelter Genauigkeit und optionaler Exponentialnotation) oder Teil eines Arrays sein, das ausschließlich solche Werte enthält.

Null-Werte

Null-Werte werden nicht indiziert.

Maximale Werte

Maximale Anzahl von benutzerdefinierten Felder für Aggregationsabfragen ist 5.

Maximale Anzahl von angeforderten Perzentilen für Aggregationsabfragen ist 100.

Autorisierung

Sie können den Index der Dinge oder das Ding (für eine GetThingConnectivityData Aktion) wie folgt als Amazon-Ressourcennamen (ARN) in einer AWS IoT Richtlinienaktion angeben.

Action	Ressource
`iot:SearchIndex`	Ein Index-ARN (z. B. `arn:aws:iot:` *your-aws-region* `:` *your-aws-account* `:index/AWS_Things`).
`iot:DescribeIndex`	Ein Index-ARN (z. B. `arn:aws:iot:` *your-aws-region* `:` *your-aws-account* `:index/AWS_Things`).
`iot:GetThingConnectivityData`	Sie können diese Erlaubnis auf bestimmte Dinge beschränken. Verwenden Sie die `iot:IncludeSocketInformation` Bedingungstaste, um den Zugriff auf Socket-Informationen zu kontrollieren.

> **Note**
>
> Wenn Sie über die Berechtigungen verfügen, den Flottenindex abzufragen, können Sie auf Daten zu Objekten über die gesamte Flotte hinweg zugreifen.

Verwalten der Objektgruppenindizierung

`AWS_ThingGroups` ist der Index, die alle Ihre Objektgruppen enthält. Dieser Index ermöglicht Ihnen die Suche nach Gruppen basierend auf Gruppennamen, Beschreibung, Attributen und allen übergeordneten Gruppennamen.

Aktivieren der Objektgruppenindizierung

Sie können die `thing-group-indexing-configuration` Einstellung in der UpdateIndexingConfiguration API verwenden, um den `AWS_ThingGroups` Index zu erstellen und seine Konfiguration zu steuern. Sie können die GetIndexingConfiguration API verwenden, um die aktuelle Indexierungskonfiguration abzurufen.

Mit dem CLI-Befehl update-indexing-configuration können Sie die Konfigurationen der Objektindizierung aktualisieren.

```
aws iot update-indexing-configuration --thing-group-indexing-configuration
 thingGroupIndexingMode=ON
```

Sie können Konfigurationen für sowohl die Objekt- als auch die Objektgruppenindizierung wie folgt auch in einem einzigen Befehl aktualisieren.

```
aws iot update-indexing-configuration --thing-indexing-configuration
 thingIndexingMode=REGISTRY --thing-group-indexing-configuration
 thingGroupIndexingMode=ON
```

Im Folgenden sehen Sie gültige Werte für `thingGroupIndexingMode`.

OFF

Kein indexing/delete Index.

ON

Den `AWS_ThingGroups`-Index erstellen oder konfigurieren.

Verwenden Sie den CLI-Befehl get-indexing-configuration zum Abrufen der aktuellen Indizierungskonfigurationen des Objekts und der Objektgruppe.

```
aws iot get-indexing-configuration
```

Die Ausgabe des Befehls sieht wie folgt aus:

```
{
   "thingGroupIndexingConfiguration": {
        "thingGroupIndexingMode": "ON"
    }
}
```

Beschreiben von Gruppenindizes

Verwenden Sie den CLI-Befehl `AWS_ThingGroups` zum Abrufen des aktuellen Status des describe-index-Index.

```
aws iot describe-index --index-name "AWS_ThingGroups"
```

Die Ausgabe des Befehls sieht wie folgt aus:

```
{
   "indexStatus": "ACTIVE",
   "indexName": "AWS_ThingGroups",
   "schema": "THING_GROUPS"
}
```

AWS IoT erstellt Ihren Index, wenn Sie zum ersten Mal indexieren. Es ist nicht möglich, den Index abzufragen, wenn `indexStatus` auf `BUILDING` eingestellt ist.

Abfragen eines Objektgruppenindex

Verwenden Sie den CLI-Befehl search-index zum Abfragen von Daten im Index:

```
aws iot search-index --index-name "AWS_ThingGroups" --query-string
 "thingGroupName:mythinggroup*"
```

Autorisierung

Sie können den Index der Dinggruppen wie folgt als Ressourcen-ARN in einer AWS IoT Richtlinienaktion angeben.

Action	Ressource
`iot:SearchIndex`	Ein Index-ARN (z. B. `arn:aws:iot:` *`your-aws-region`* `:index/AWS_ThingGroups`).
`iot:DescribeIndex`	Ein Index-ARN (z. B. `arn:aws:iot:` *`your-aws-region`* `:index/AWS_ThingGroups`).

Abfragen zum Status der Gerätekonnektivität

AWS IoT Fleet Indexing unterstützt die Abfrage der Konnektivität einzelner Geräte, sodass Sie den Konnektivitätsstatus und die zugehörigen Metadaten für bestimmte Geräte effizient abrufen können. Diese Funktion ergänzt die bestehenden flottenweiten Indexierungs- und Abfragefunktionen.

Funktionsweise

Die Unterstützung für Gerätekonnektivitätsabfragen kann für den optimierten Abruf des Konnektivitätsstatus einzelner Geräte verwendet werden. Diese API ermöglicht den Zugriff auf die neuesten gerätespezifischen Konnektivitätsinformationen mit niedriger Latenz und hohem Durchsatz. Sobald Sie die Konnektivitätsindizierung aktiviert haben, haben Sie Zugriff auf diese Abfrage-API, die als Standardabfragen berechnet wird. Weitere Informationen finden Sie unter Preise für die AWS IoT Geräteverwaltung

Features

Mit der Unterstützung für Gerätekonnektivitätsabfragen können Sie:

1. Den aktuellen Konnektivitätsstatus (verbunden oder getrennt) für ein bestimmtes Gerät mithilfe von dessen `thingName` abfragen.
2. Ruft zusätzliche Konnektivitätsmetadaten ab, darunter:
   a. Grund für das Trennen der Verbindung
   b. Zeitstempel für das letzte Verbindungs- oder Trennungsereignis.
   c. Sitzungsinformationen, einschließlich der Keep-Alive-Dauer
   d. Sitzungsinformationen auf Socket-Ebene, einschließlich IP-Adresse, Port und VPC-Endpunkt-ID. Diese Informationen sind nur verfügbar, wenn die folgenden Bedingungen erfüllt sind:
      i. Sie aktivieren diese Option in den Konfigurationseinstellungen für die Flottenindizierung.
      ii. Sie haben die entsprechende Berechtigung in der IAM-Richtlinie, wenn Sie die API aufrufen.

> **Note**
>
> Die Flottenindizierung indexiert den Konnektivitätsstatus für ein Gerät, dessen Verbindung `clientId` mit der eines in `thingName` der Registrierung registrierten Geräts übereinstimmt.

Vorteile

1. Niedrige Latenz: Spiegelt den aktuellen Status der Gerätekonnektivität wider und bietet eine niedrige Latenz, um Änderungen des Verbindungsstatus durch IoT Core widerzuspiegeln. IoT Core stellt fest, dass ein Gerät entweder getrennt wird, sobald es eine Trennungsanforderung vom Gerät erhält oder wenn ein Gerät die Verbindung trennt, ohne eine Trennungsanforderung

zu senden. IoT Core wartet das 1,5-fache der konfigurierten Keep-Alive-Zeit ab, bevor festgestellt wird, dass die Verbindung zum Client getrennt wird. Die Connectivity-Status-API spiegelt diese Änderungen in der Regel weniger als eine Sekunde wider, nachdem IoT Core die Änderung des Verbindungszustands eines Geräts ermittelt hat.

2. Hoher Durchsatz: Unterstützt standardmäßig 350 Transaktionen pro Sekunde (TPS) und kann auf Anfrage auf einen höheren Wert eingestellt werden.
3. Datenspeicherung: Speichert Eventdaten auf unbestimmte Zeit, wenn der ConnectivityIndexing Flottenindexing-Modus (FI) aktiviert ist und das Ding nicht gelöscht wird. Wenn Sie die Connectivity Indexing deaktivieren, werden die Aufzeichnungen nicht aufbewahrt.

> **Note**
>
> Wenn die Indexierung des Konnektivitätsstatus vor dem Start dieser API aktiviert war, beginnt die Fleet Indexing nach dem API-Start mit der Verfolgung von Änderungen des Konnektivitätsstatus und zeigt den aktualisierten Status auf der Grundlage dieser Änderungen an.

Voraussetzungen

Um die Unterstützung für Gerätekonnektivitätsabfragen zu nutzen:

1. Richten Sie ein AWS Konto ein
2. Integrieren und registrieren Sie Geräte AWS IoT Core in Ihrer bevorzugten Region
3. Aktivieren Sie die Flottenindizierung mit Connectivity-Indexierung. Optional: Opt-in um Informationen auf Socket-Ebene zu erhalten, indem Sie die `includeSocketInformation` Option aktivieren.

> **Note**
>
> Wenn Sie die Konnektivitätsindizierung bereits aktiviert haben, ist keine zusätzliche Einrichtung erforderlich.

Eine ausführliche Anleitung zur Einrichtung finden Sie im AWS IoT Entwicklerhandbuch

Beispiel

CLI-Befehl:

```
aws iot get-thing-connectivity-data --include-socket-information --thing-name
 myThingName
```

Antwort:

```
{
   "thingName": "myThingName",
   "connected": true,
   "timestamp": "2024-12-19T10:00:00.000000-08:00",
   "disconnectReason": "NONE",
   "sourceIp": "192.0.2.1",
   "sourcePort": 52123,
   "targetIp": "198.51.100.1",
   "targetPort": 8883,
   "vpcEndpointId": "vpce-1234567890abcdef0",
   "keepAliveDuration": 60,
   "cleanSession": true,
   "clientId": "myThingName"
}
```

API-Parameter

- `thingName`: Der Name des in der AWS IoT Registrierung registrierten Geräts. Dieser muss mit dem übereinstimmen, der für die Verbindung mit AWS IoT Core `clientId` verwendet wird.
- `includeSocketInformation`: Der `includeSocketInformation` Parameter steuert, ob Netzwerkinformationen auf Socket-Ebene in der API-Antwort enthalten sind. Wenn der Wert auf „true“ gesetzt ist, enthält die Antwort die folgenden Felder:. `sourceIp, sourcePort, targetIp, targetPort, vpcEndpointId` Wenn nicht angegeben oder auf False gesetzt `includeSocketInformation` ist, werden diese Socket-Felder von der Antwort ausgeschlossen. Um bestimmte IAM-Benutzer am Zugriff auf Socket-Informationen zu hindern, müssen Sie dies in ihrer IAM-Richtlinie angeben, indem Sie den `includeSocketInformation` Bedingungsschlüssel auf falsch setzen

Antwortfelder

- `thingName`: Der Name des in der AWS IoT Registrierung registrierten Geräts. Dieser muss mit dem übereinstimmen, der für die Verbindung mit AWS IoT Core `clientId` verwendet wird.
- `connected`: Der boolesche Wert true gibt an, dass dieses Gerät derzeit verbunden ist.
- `disconnectReason`: Grund für die Trennung. Gilt `NONE` für ein verbundenes Gerät und `UNKNOWN` für ein Gerät, das noch nie angeschlossen wurde. Bei einem getrennten Gerät gibt dies an, ob die Trennung vom Client oder vom Server initiiert wurde, aufgrund von authentication/authorization Problemen oder aufgrund von Netzwerkproblemen. Die Ursachencodes für die Trennung finden Sie unter LifeCycleEvents
- `timestamp`: Der Zeitstempel, der das letzte Verbindungs- oder Trennungsereignis des Geräts darstellt.
- `clientId`: Die ClientID des MQTT-Clients.
- `keepAliveDuration`: Das Keep-Alive-Intervall in Sekunden, das der Client beim Verbindungsaufbau angegeben hat. Dies bestimmt, wie oft der Client Keep-Alive-Nachrichten sendet, um die Verbindung aufrechtzuerhalten.
- `cleanSession`: Gibt an, ob der Client eine saubere Sitzung verwendet.
- `sessionExpiry`: Die Konfiguration für den dauerhaften Sitzungsablauf, die der Client beim Verbindungsaufbau angegeben hat. Dies bestimmt, wie lange eine Sitzung aktiv bleibt, nachdem der Client die Verbindung getrennt hat.
- `sourceIp`: Die IP-Adresse des Clients, der die Verbindung initiiert hat. Wird nur zurückgegeben, wenn der Wert auf „true“ gesetzt `includeSocketInformation` ist und der Benutzer berechtigt ist, diese Informationen abzurufen.
- `sourcePort`: Die Portnummer, die vom Client für die Verbindung verwendet wird. Wird nur zurückgegeben, wenn der Wert auf „true“ gesetzt `includeSocketInformation` ist und der Benutzer berechtigt ist, diese Informationen abzurufen.
- `targetIp`: Die IP-Adresse, an die die Verbindungsanfrage gestellt wurde. Wird nur zurückgegeben, wenn der Wert auf „true“ gesetzt `includeSocketInformation` ist und der Benutzer berechtigt ist, diese Informationen abzurufen
- `targetPort`: Die Portnummer des AWS IoT Core-Endpunkts, mit dem der Client eine Verbindung hergestellt hat. Wird nur zurückgegeben, wenn der Wert auf „true“ gesetzt `includeSocketInformation` ist und der Benutzer berechtigt ist, diese Informationen abzurufen.
- `vpcEndpointId`: Die ID des VPC-Endpunkts, über den der Client eine Verbindung hergestellt hat, falls zutreffend. Wird nur zurückgegeben, wenn der Wert auf „true“ gesetzt `includeSocketInformation` ist und der Benutzer berechtigt ist, diese Informationen abzurufen.

Erforderliche Berechtigungen

Um die GetThingConnectivityData API verwenden zu können, benötigen Sie die folgende IAM-Berechtigung

`iot:GetThingConnectivityData`

Sie können diese Berechtigung mithilfe ressourcenbasierter Richtlinien auf bestimmte Dinge beschränken. Verwenden Sie die `iot:IncludeSocketInformation` Bedingung, um eine granulare Zugriffskontrolle für Socket-Informationen zu implementieren. Das folgende Richtlinienbeispiel zeigt ein Szenario, in dem Sie dem Benutzer den Zugriff auf Socket-Informationen verweigern. Bitte beachten Sie, dass dieses Beispiel funktioniert, wenn es die einzige Richtlinie ist, die Zugriff auf die Aktion gewährt GetThingConnectivityData.

Beispiele für Autorisierungsrichtlinien

```
{
  "Version": "2012-10-17",
  "Statement": [
  {
    "Effect": "Allow",
    "Action": [ "iot:GetThingConnectivityData" ],
    "Resource": [ "arn:aws:iot:us-east-1:123456789012:thing/*"],
    "Condition": {
      "Bool": {
        "iot:IncludeSocketInformation": "false"
      }
    }
  }]
}
```

Abfragen von Aggregatdaten

AWS IoT bietet vier APIs (`GetStatistics`,`GetCardinality`, und`GetBucketsAggregation`)`GetPercentiles`, mit denen Sie Ihre Geräteflotte nach aggregierten Daten durchsuchen können.

> **Note**
>
> Bei Problemen mit fehlenden oder unerwarteten Werten für die Aggregations-APIs lesen Sie den Leitfaden zur Fehlerbehebung bei der Fleet-Indexierung.

GetStatistics

Die GetStatistics API und der get-statistics CLI-Befehl geben die Anzahl, den Durchschnitt, die Summe, das Minimum, das Maximum, die Summe der Quadrate, die Varianz und die Standardabweichung für das angegebene aggregierte Feld zurück.

Der get-statistics-CLI-Befehl nimmt die folgenden Parameter entgegen:

`index-name`

Der Name des zu durchsuchenden Indexes. Der Standardwert ist `AWS_Things`.

`query-string`

Die zur Abfrage des Indexes verwendete Abfrage. Sie können angeben`"*"`, dass die Anzahl aller indizierten Dinge in Ihrem abgerufen werden soll. AWS-Konto

`aggregationField`

(Optional) Das zu aggregierende Feld. Dieses Feld muss ein verwaltetes oder benutzerdefiniertes Feld sein, das beim Aufruf von update-indexing-configuration definiert wird. Wenn Sie kein Aggregationsfeld angeben, wird `registry.version` als Aggregationsfeld verwendet.

`query-version`

Die Version der Abfrage, die verwendet werden soll. Der Standardwert ist `2017-09-30`.

Der Typ des Aggregationsfelds kann sich auf die zurückgegebenen Statistiken auswirken.

GetStatistics mit Zeichenfolgenwerten

Wenn Sie in einem Zeichenfolgenfeld aggregieren, gibt der Aufruf von `GetStatistics` eine Anzahl von Geräten zurück, die Attribute aufweisen, die der Abfrage entsprechen. Beispiel:

```
aws iot get-statistics --aggregation-field 'attributes.stringAttribute'
            --query-string '*'
```

Dieser Befehl gibt die Anzahl der Geräte zurück, die ein Attribut mit dem Namen `stringAttribute` enthalten:

```
{
  "statistics": {
    "count": 3
  }
}
```

GetStatistics mit booleschen Werten

Wenn Sie `GetStatistics` mit einem booleschen Aggregationsfeld aufrufen:

- AVERAGE ist der Prozentsatz der Geräte, die mit der Abfrage übereinstimmen.
- MINIMUM ist 0 oder 1 gemäß den folgenden Regeln:
  - Wenn alle Werte für das Aggregationsfeld `false` lauten, ist MINIMUM 0.
  - Wenn alle Werte für das Aggregationsfeld `true` lauten, ist MINIMUM 1.
  - Wenn die Werte für das Aggregationsfeld eine Mischung aus `false` und sind `true`, ist MINIMUM 0.
- MAXIMUM ist 0 oder 1 gemäß den folgenden Regeln:
  - Wenn alle Werte für das Aggregationsfeld `false` lauten, ist MAXIMUM 0.
  - Wenn alle Werte für das Aggregationsfeld `true` lauten, ist MAXIMUM 1.
  - Wenn die Werte für das Aggregationsfeld eine Mischung aus `false` und `true` sind, ist MAXIMUM 1.
- SUM ist die Summe des ganzzahligen Äquivalents der booleschen Werte.
- COUNT ist die Anzahl der Elemente, die den Kriterien der Abfragezeichenfolge entsprechen und einen gültigen Aggregationsfeldwert enthalten.

GetStatistics mit numerischen Werten

Wenn Sie `GetStatistics` aufrufen und ein Aggregationsfeld vom Typ `Number` angeben, gibt `GetStatistics` die folgenden Werte zurück:

count

Die Anzahl der Elemente, die den Kriterien der Abfragezeichenfolge entsprechen und einen gültigen Aggregationsfeldwert enthalten.

Durchschnitt

Der Durchschnitt der numerischen Werte, die der Abfrage entsprechen.

sum

Die Summe der numerischen Werte, die der Abfrage entsprechen.

Minimum

Der kleinste numerische Wert, der der Abfrage entspricht.

Maximum

Der größte numerische Wert, der der Abfrage entspricht.

Summe OfSquares

Die Summe der Quadrate der numerischen Werte, die der Abfrage entsprechen.

Varianz

Die Varianz der numerischen Werte, die der Abfrage entsprechen. Die Varianz einer Wertemenge ist der Durchschnitt der Quadrate der Differenzen jedes einzelnen Werts vom Durchschnittswert der Menge.

stdDeviation

Die Standardabweichung der numerischen Werte, die der Abfrage entsprechen. Die Standardabweichung einer Wertemenge ist ein Maß für die Verteilung der Werte.

Das folgende Beispiel zeigt, wie get-statistics mit einem numerischen benutzerdefinierten Feld aufgerufen wird.

```
aws iot get-statistics --aggregation-field 'attributes.numericAttribute2'
            --query-string '*'
```

```
{
  "statistics": {
    "count": 3,
    "average": 33.333333333333336,
    "sum": 100.0,
    "minimum": -125.0,
    "maximum": 150.0,
```

```
    "sumOfSquares": 43750.0,
    "variance": 13472.22222222222,
    "stdDeviation": 116.06990230986766
  }
}
```

Wenn die Feldwerte den maximalen doppelten Wert überschreiten, sind bei numerischen Aggregationsfeldern die Statistikwerte leer.

GetCardinality

Die GetCardinality API und der get-cardinality CLI-Befehl geben die ungefähre Anzahl der eindeutigen Werte zurück, die der Abfrage entsprechen. Beispiel: Sie möchten die Anzahl der Geräte mit einem Akkustand von weniger als 50 Prozent ermitteln:

```
aws iot get-cardinality --index-name AWS_Things --query-string "batterylevel
          > 50" --aggregation-field "shadow.reported.batterylevel"
```

Dieser Befehl gibt die Anzahl der Elemente mit einem Akkustand von mehr als 50 Prozent zurück:

```
{
    "cardinality": 100
}
```

`cardinality` wird immer von get-cardinality zurückgegeben, auch wenn keine übereinstimmenden Felder vorhanden sind. Beispiel:

```
aws iot get-cardinality --query-string "thingName:Non-existent*"
          --aggregation-field "attributes.customField_STR"
```

```
{
    "cardinality": 0
}
```

Der get-cardinality-CLI-Befehl nimmt die folgenden Parameter entgegen:

`index-name`

Der Name des zu durchsuchenden Indexes. Der Standardwert ist `AWS_Things`.

`query-string`

Die zur Abfrage des Indexes verwendete Abfrage. Sie können angeben`"*"`, dass die Anzahl aller indizierten Dinge in Ihrem AWS-Konto abgerufen werden soll.

`aggregationField`

Das zu aggregierende Feld.

`query-version`

Die Version der Abfrage, die verwendet werden soll. Der Standardwert ist `2017-09-30`.

GetPercentiles

Die GetPercentiles API und der get-percentiles CLI-Befehl gruppieren die aggregierten Werte, die der Abfrage entsprechen, in Perzentilgruppierungen. Die standardmäßigen Perzentilgruppierungen sind 1,5,25,50,75,95,99, auch wenn Sie beim Aufrufen von `GetPercentiles` Ihre eigenen angeben können. Diese Funktion gibt einen Wert für jede angegebene Perzentilgruppe (oder die standardmäßigen Perzentilgruppierungen) zurück. Die Perzentilgruppe „1“ enthält den aggregierten Feldwert, der in etwa in einem Prozent der Werte auftritt, die der Abfrage entsprechen. Die Perzentilgruppe „5“ enthält den aggregierten Feldwert, der in etwa in fünf Prozent der Werte auftritt, die der Abfrage entsprechen, usw. Das Ergebnis ist eine Annäherung, je mehr Werte der Abfrage entsprechen, desto genauer sind die Perzentilwerte.

Das folgende Beispiel zeigt, wie der get-percentiles-CLI-Befehl aufgerufen wird.

```
aws iot get-percentiles --query-string "thingName:*" --aggregation-field
          "attributes.customField_NUM" --percents 10 20 30 40 50 60 70 80 90 99
```

```
{
    "percentiles": [
        {
            "value": 3.0,
            "percent": 80.0
        },
        {
            "value": 2.5999999999999996,
            "percent": 70.0
        },
        {
            "value": 3.0,
```

```
            "percent": 90.0
        },
        {
            "value": 2.0,
            "percent": 50.0
        },
        {
            "value": 2.0,
            "percent": 60.0
        },
        {
            "value": 1.0,
            "percent": 10.0
        },
        {
            "value": 2.0,
            "percent": 40.0
        },
        {
            "value": 1.0,
            "percent": 20.0
        },
        {
            "value": 1.4,
            "percent": 30.0
        },
        {
            "value": 3.0,
            "percent": 99.0
        }
    ]
}
```

Der folgende Befehl zeigt die Ausgabe von get-percentiles, wenn keine entsprechenden Dokumente vorhanden sind.

```
aws iot get-percentiles --query-string "thingName:Non-existent*"
          --aggregation-field "attributes.customField_NUM"
```

```
{
    "percentiles": []
}
```

Der get-percentile-CLI-Befehl nimmt die folgenden Parameter entgegen:

`index-name`

Der Name des zu durchsuchenden Indexes. Der Standardwert ist `AWS_Things`.

`query-string`

Die zur Abfrage des Indexes verwendete Abfrage. Sie können angeben`"*"`, dass die Anzahl aller indizierten Dinge in Ihrem abgerufen werden soll. AWS-Konto

`aggregationField`

Das zu aggregierende Feld, das den Typ `Number` aufweisen muss.

`query-version`

Die Version der Abfrage, die verwendet werden soll. Der Standardwert ist `2017-09-30`.

`percents`

Mit diesem Parameter können Sie benutzerdefinierte Perzentilgruppierungen angeben.

GetBucketsAggregation

Die GetBucketsAggregation API und der get-buckets-aggregation CLI-Befehl geben eine Liste von Buckets und die Gesamtzahl der Dinge zurück, die den Kriterien der Abfragezeichenfolge entsprechen.

Das folgende Beispiel zeigt, wie der -CLI-Befehl aufgerufen wird.

```
aws iot get-buckets-aggregation --query-string '*' --index-name AWS_Things --
aggregation-field 'shadow.reported.batterylevelpercent' --buckets-aggregation-type
 'termsAggregation={maxBuckets=5}'
```

Dieser Befehl gibt die folgende Ausgabe zurück: .

```
{
    "totalCount": 20,
    "buckets": [
        {
            "keyValue": "100",
            "count": 12
```

```
        },
        {
            "keyValue": "90",
            "count": 5
        },
        {
            "keyValue": "75",
            "count": 3
        }
    ]
}
```

Der -CLI-Befehl nimmt die folgenden Parameter entgegen:

`index-name`

Der Name des zu durchsuchenden Indexes. Der Standardwert ist `AWS_Things`.

`query-string`

Die zur Abfrage des Indexes verwendete Abfrage. Sie können angeben`"*"`, dass die Anzahl aller indizierten Dinge in Ihrem abgerufen werden soll. AWS-Konto

`aggregation-field`

Das zu aggregierende Feld.

`buckets-aggregation-type`

Die grundlegende Steuerung der Antwortform und des auszuführenden Bucket-Aggregationstyps.

Autorisierung

Sie können den Index der Dinggruppen wie folgt als Ressourcen-ARN in einer AWS IoT Richtlinienaktion angeben.

Action	Ressource
`iot:GetStatistics`	Ein Index-ARN (z. B. `arn:aws:iot:` *`your-aws-region`* `:index/AWS_Things` oder `arn:aws:iot:` *`your-aws-region`* `:index/AWS_ThingGroups`).

Abfragesyntax

Bei der Flottenindizierung verwenden Sie eine Abfragesyntax, um Abfragen zu spezifizieren.

Unterstützte Features

Die Abfragesyntax unterstützt die folgenden Funktionen.

- Begriffe und Ausdrücke
- Suchen nach Feldern
- Präfixsuche
- Bereichssuche
- Boolesche Operatoren `AND`, `OR`, `NOT` und `–`. Der Bindestrich wird verwendet, um Suchergebnisse auszuschließen (z. B. `thingName:(tv* AND -plasma)`).
- Gruppierung
- Feldgruppierung
- Escape-Sonderzeichen (wie bei)
- Die Verwendung von führenden Platzhaltern ist auf einen Abfragebegriff pro Abfrage beschränkt. Sie können beispielsweise nicht nach `thingName:*my` und `thingGroupNames:*echo` in derselben Abfrage suchen. Abfragen, die einen Platzhalter am Anfang enthalten, haben eine maximale Abfragelänge von 100 Zeichen.

> **Note**
>
> Konten mit einem API-Limit für Fleet Indexing-Abfragen von mehr als 30 Anfragen pro Sekunde unterliegen einer Drosselung, wenn Abfrageausdrücke mit führenden Platzhaltern verwendet werden. Dieses Limit von 30 Anfragen pro Sekunde gilt auf Kontoebene für alle Fleet Indexing-Abfrage-APIs.

Nicht unterstützte Funktionen

Die Abfragesyntax unterstützt nicht die folgenden Funktionen:

- Reguläre Ausdrücke

- Boosting
- Ranking
- Fuzzysuchen
- Umgebungssuche
- Sortieren
- Aggregation
- Sonderzeichen: `,@,#,%,\,/, ';, und ,. Beachten Sie, dass , dies nur in Geoqueries unterstützt wird.

Hinweise

Hier einige Hinweise zur Abfragesprache:

- Der standardmäßige Operator ist UND. Eine Abfrage für "thingName:abc thingType:xyz" ist äquivalent mit "thingName:abc AND thingType:xyz".
- Wenn kein Feld angegeben ist, wird in allen Registry-, Device Shadow- und Device Defender-Feldern nach dem Begriff AWS IoT gesucht.
- Bei allen Feldnamen muss die Groß- und Kleinschreibung beachtet werden.
- Bei der Suche muss die Groß- und Kleinschreibung nicht beachtet werden. Wörter werden gemäß Definition von Character.isWhitespace(int) von Java durch Leerzeichen getrennt.
- Die Indizierung von Device Schatten enthält die folgenden Abschnitte: gemeldet, gewünscht, Delta und Metadaten.
- Device Schatten- und Registry-Versionen können nicht durchsucht werden, sind jedoch in der Antwort vorhanden.
- Die maximale Anzahl an Begriffen in einer Abfrage ist 5.
- Das Sonderzeichen , wird nur in Geoqueries unterstützt.

Beispiel für Objektabfragen

Geben Sie Abfragen mithilfe einer Abfragesyntax in einer Abfragezeichenfolge an. Die Abfragen werden an die SearchIndexAPI übergeben. Die folgende Tabelle enthält einige Beispiele für Abfragezeichenfolgen.

Abfragezeichenfolge	Ergebnis
`abc`	Abfragen nach „abc“ in beliebigen Registry-, Schatten- (klassischer unbenannter Shadow und benannter Shadow) oder Device Defender-Verstößen.
`thingName:myThingName`	Fragt nach einer Sache mit dem Namen „myThingName“ ab.
`thingName:my*`	Abfragen von Objekten mit Namen, die mit „my” beginnen.
`thingName:*my`	Abfragen nach Dingen, deren Namen auf „mein“ enden. Die Verwendung von häufig verwendeten Platzhaltern ist auf einen Abfragebegriff pro Abfrage beschränkt. Abfragen, die ein führendes Platzhalterzeichen enthalten, haben eine maximale Abfragelänge von 100 Zeichen.
`thingName:*my*`	Fragt Dinge ab, die die Teilzeichenfolge „my“ enthalten . Die Verwendung von häufig verwendeten Platzhalt ern ist auf einen Abfragebegriff pro Abfrage beschränkt. Abfragen, die ein führendes Platzhalterzeichen enthalten, haben eine maximale Abfragelänge von 100 Zeichen.
`thingName:ab?`	Abfragen von Objekten mit Namen, die "ab" sowie ein zusätzliches Zeichen enthalten, zum Beispiel: "aba", "abb", "abc" usw.
`thingTypeName:aa`	Abfragen für Objekte, die dem Typ aa zugeordnet sind.
`thingGroupNames:a`	Abfragen nach Dingen mit dem Namen einer übergeord neten Dinggruppe oder einer Abrechnungsgruppe „a“.
`thingGroupNames:a*`	Fragt nach Dingen ab, bei denen der Name einer übergeordneten Dinggruppe oder einer Abrechnun gsgruppe dem Muster „a*“ entspricht.
`attributes.myAttribute:75`	Abfragen von Objekten mit einem Attribut namens "myAttribute" mit dem Wert 75.

Abfragezeichenfolge	Ergebnis
`attributes.myAttribute: [75 TO 80]`	Abfragen von Objekten mit einem Attribut namens "myAttribute", dessen Wert innerhalb eines numerischen Bereichs liegt (7580, einschließlich).
`attributes.myAttribute: {75 TO 80]`	Abfragen von Objekten mit einem Attribut namens "myAttribute", dessen Wert innerhalb des numerischen Bereichs liegt (>75 und<=80).
`attributes.serialNumber: ["abcd" TO "abcf"]`	Abfragen von Objekten mit einem Attribut namens "serialNumber", dessen Wert innerhalb eines alphanume rischen Zeichenfolgebereichs liegt. Diese Abfrage gibt Objekte mit einem "serialNumber"-Attribut mit den Werten "abcd", "abce" oder "abcf" zurück.
`attributes.myAttri bute:i*t`	Abfragen von Objekten mit einem Attribut namens "myAttribute" mit dem Wert "i", gefolgt von einer beliebigen Anzahl von Zeichen, gefolgt von "t".
`attributes.attr1:abc AND attributes.attr2<5 NOT attributes.attr3>10`	Abfragen mit booleschen Ausdrücken von Objekten, die Begriffe kombinieren. Diese Abfrage gibt Objekte zurück, die ein Attribut mit dem Namen "attr1" mit dem Wert "abc", ein Attribut mit dem Namen "attr2", das kleiner als 5 ist, und ein Attribut mit dem Namen "attr3", das nicht größer als 10 ist, aufweisen.
`shadow.hasDelta:true`	Abfragen nach Objekten mit einem unbenannten Schatten, der ein Deltaelement enthält.
`NOT attributes.model:l egacy`	Abfragen von Objekten, bei denen das Attribut namens "model" nicht "legacy" ist.

Abfragezeichenfolge	Ergebnis
`shadow.reported.stats.battery:{70 TO 100} (v2 OR v3) NOT attributes.model:legacy`	Abfragen von Objekten, für die Folgendes gilt: • Das Schattenattribut `stats.battery` des Objekts enthält einen Wert zwischen 70 und 100. • Der Text „v2” oder „v3” ist im Namen, im Typnamen oder in den Attributwerten eines Objekts enthalten. • Das Attribut `model` des Objekts ist nicht auf „legacy” festgelegt.
`shadow.reported.myvalues:2`	Abfragen von Objekten, bei denen das Array `myvalues` im Abschnitt Gemeldete des Schattens den Wert 2 enthält.
`shadow.reported.location:* NOT shadow.desired.stats.battery:*`	Abfragen von Objekten, für die Folgendes gilt: • Das Attribut `location` ist im Abschnitt `reported` des Schattens vorhanden. • Das Attribut `stats.battery` ist nicht im Abschnitt `desired` des Schattens vorhanden.
`shadow.name.<shadowName>.hasDelta:true`	Abfragen nach Objekten, die einen Schatten mit dem angegebenen Namen und auch ein Delta-Element haben.
`shadow.name.<shadowName>.desired.filament:*`	Fragt nach Objekten ab, die einen Schatten mit dem angegebenen Namen und auch einer gewünschten Filamenteigenschaft haben.
`shadow.name.<shadowName>.reported.location:*`	Fragt nach Objekten, die einen Schatten mit dem angegebenen Namen haben und bei denen das `location` Attribut im Berichtsbereich des benannten Schattens vorhanden ist.
`connectivity.connected:true`	Abfragen für alle angeschlossenen Geräte.
`connectivity.connected:false`	Abfrage für alle nicht verbundenen Geräte.

Abfragezeichenfolge	Ergebnis
`connectivity.connected:true AND connectivity.timestamp : [1557651600000 TO 1557867600000]`	Abfragen für alle verbundenen Geräte mit einem Verbindungszeitstempel >= 1557651600000 und <= 1557867600000. Zeitstempel werden in Millisekunden seit der Epoche angegeben.
`connectivity.connected:false AND connectivity.timestamp : [1557651600000 TO 1557867600000]`	Abfragen für alle getrennten Geräte mit einem Trennungs zeitstempel >= 1557651600000 und <= 1557867600000. Zeitstempel werden in Millisekunden seit der Epoche angegeben.
`connectivity.connected:true AND connectivity.timestamp > 1557651600000`	Abfragen für alle verbundenen Geräte mit einem Verbindungszeitstempel > 1557651600000. Zeitstempel werden in Millisekunden seit der Epoche angegeben.
`connectivity.connected:*`	Abfragen für alle Geräte mit vorhandenen Verbindun gsinformationen.
`connectivity.disconnectReason:*`	Abfragen für alle Geräte mit vorhandenen Verbindun gsinformationen.
`connectivity.disconnectReason:CLIENT_INITIATED_DISCONNECT`	Abfragen für alle Geräte, die aufgrund von CLIENT_IN ITIATED_DISCONNECT getrennt wurden.
`deviceDefender.violationCount:[0 TO 100]`	Abfragen nach Objekten, bei denen Device Defender einen Zählwert verletzt, der innerhalb des numerischen Bereichs (0-100, einschließlich) liegt.
`deviceDefender.<device-SecurityProfile>.disconnectBehavior.inViolation:true`	Abfragen nach Objekten, die gegen das im Sicherhei tsprofil `device-SecurityProfile` definierte Verhalten `disconnectBehavior` verstoßen. Beachten Sie, dass in keine gültige Abfrage Violation:false ist.

Abfragezeichenfolge	Ergebnis
`deviceDefender.<device-SecurityProfile>.disconnectBehavior.lastViolationValue.number>2`	Fragt nach Dingen, die gegen das im Sicherheitsprofil des Geräts definierte Verhalten `disconnectBehavior` verstoßen — SecurityProfile wobei der Wert für das letzte Verstoßereignis größer als 2 ist.
`deviceDefender.<device-SecurityProfile>.disconnectBehavior.lastViolationTime>1634227200000`	Fragt nach Dingen, die gegen das im Sicherheitsprofil definierte Verhalten `disconnectBehavior` verstoßen . Das letzte Verstoßereignis liegt nach einer bestimmten Epoche vor. SecurityProfile
`shadow.name.gps-tracker.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km`	Abfragen nach Objekten, die sich innerhalb einer radialen Entfernung von 15,5 km von den Koordinaten 47.6204, -122.3491 befinden. Diese Abfragezeichenfolge gilt für den Fall, dass Ihre Standortdaten in einem benannten Schatten gespeichert werden.
`shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km`	Abfragen für Objekte, die sich innerhalb einer radialen Entfernung von 15,5 km von den Koordinaten 47,6204, -122,3491 befinden. Diese Abfragezeichenfolge gilt für den Fall, dass Ihre Standortdaten in einem klassischen Schatten gespeichert werden.

Beispiel für Objektgruppenabfragen

Abfragen werden in einer Abfragezeichenfolge mit einer Abfragesyntax angegeben und an die `SearchIndex`-API übergeben. Die folgende Tabelle enthält einige Beispiele für Abfragezeichenfolgen.

Abfragezeichenfolge	Ergebnis
`abc`	Abfragen von "abc" in einem beliebigen Feld.
`thingGroupName:myGroupThingName`	Fragt nach einer Dinggruppe mit dem Namen „myGroupThingName“ ab.

Abfragezeichenfolge	Ergebnis
`thingGroupName:my*`	Abfragen von Objektgruppen mit Namen, die mit "my" beginnen.
`thingGroupName:ab?`	Abfragen von Objektgruppen mit Namen, die "ab" sowie ein zusätzliches Zeichen enthalten, zum Beispiel: "aba", "abb", "abc" usw.
`attributes.myAttribute:75`	Abfragen von Objektgruppen mit einem Attribut namens "myAttribute" mit dem Wert 75.
`attributes.myAttribute:[75 TO 80]`	Abfragen von Objektgruppen mit einem Attribut namens "myAttribute", dessen Wert innerhalb eines numerischen Bereichs liegt (7580, einschließlich).
`attributes.myAttribute:[75 TO 80]`	Abfragen von Objektgruppen mit einem Attribut namens "myAttribute", dessen Wert innerhalb des numerischen Bereichs liegt (>75 und<=80).
`attributes.myAttribute: ["abcd" TO "abcf"]`	Abfragen von Objektgruppen mit einem Attribut namens "myAttribute", dessen Wert innerhalb eines alphanume rischen Zeichenfolgebereichs liegt. Diese Abfrage gibt Objektgruppen mit einem "serialNumber"-Attribut mit den Werten "abcd", "abce" oder "abcf" zurück.
`attributes.myAttribute:i*t`	Abfragen von Objektgruppen mit einem Attribut namens "myAttribute" mit dem Wert "i", gefolgt von einer beliebige n Anzahl von Zeichen, gefolgt von "t".
`attributes.attr1:abc AND attributes.attr2<5 NOT attributes.attr3>10`	Abfragen für Objektgruppen, mit booleschen Ausdrücke n Begriffe kombinieren. Diese Abfrage gibt Objektgru ppen zurück, die ein Attribut mit dem Namen "attr1" mit dem Wert "abc", ein Attribut mit dem Namen "attr2", das kleiner als 5 ist, und ein Attribut mit dem Namen "attr3", das nicht größer als 10 ist, aufweisen.
`NOT attributes.myAttri bute:cde`	Abfragen von Objektgruppen, bei denen das Attribut namens "myAttribute" nicht "cde" ist.

Abfragezeichenfolge	Ergebnis
`parentGroupNames:(myParentThingGroupName)`	Abfragen für Dinggruppen, deren übergeordneter Gruppenname mit „myParentThingGroupName“ übereinstimmt.
`parentGroupNames:(myParentThingGroupName OR myRootThingGroupName)`	Abfragen nach Dinggruppen, deren übergeordneter Gruppenname mit „myParentThingGroupName“ oder „myRootThingGroupName“ übereinstimmt.
`parentGroupNames:(myParentThingGroupNa*)`	Abfragen für Dinggruppen, deren übergeordneter Gruppenname mit „myParentThingGroupNa“ beginnt.

Indexierung von Standortdaten

Sie können die AWS IoT Flottenindizierung verwenden, um die zuletzt gesendeten Standortdaten Ihrer Geräte zu indizieren und mithilfe von Geoabfragen nach Geräten zu suchen. Diese Funktion löst Anwendungsfälle für die Geräteüberwachung und -verwaltung wie Standortverfolgung und Näherungssuche. Die Standortindizierung funktioniert ähnlich wie andere Funktionen zur Flottenindizierung, allerObjekts mit zusätzlichen Konfigurationen, die Sie bei der Indexierung Ihres Objekts angeben müssen.

Zu den häufigsten Anwendungsfällen gehören: Suchen und Aggregieren von Geräten, die sich innerhalb der gewünschten geografischen Grenzen befinden, Abrufen von standortspezifischen Erkenntnissen mithilfe von Abfragebegriffen in Bezug auf Gerätemetadaten und Status aus indizierten Datenquellen, Bereitstellung einer detaillierten Ansicht, z. B. das Filtern von Ergebnissen nach einem bestimmten geografischen Gebiet, um Rendering-Verzögerungen innerhalb Ihrer Flottenüberwachungskarten zu reduzieren und den zuletzt gemeldeten Gerätestandort zu verfolgen und Geräte zu identifizieren, die sich außerhalb der gewünschten Grenzwerte befinden, und anhand von Flottenmetriken Alarme zu generieren. Erste Schritte mit der Standortindizierung und Geoabfragen finden Sie unter ???.

Unterstützte Datumsformate

AWS IoT Die Flottenindizierung unterstützt die folgenden Standortdatenformate:

1. Well-known Textdarstellung von Koordinatenreferenzsystemen

   Eine Zeichenfolge, die dem Format Geographische Informationen — Well-known Textdarstellung von Koordinatenreferenzsystemen folgt. Ein Beispiel kann sein`"POINT(long lat)"`.

2. Eine Zeichenfolge, die die Koordinaten darstellt

   Eine Zeichenfolge im Format `"latitude, longitude"` oder `"longitude, latitude"`. Wenn Sie `"longitude, latitude"` angeben, müssen Sie auch `order` in `geoLocations` angeben. Ein Beispiel kann sein `"41.12,-71.34"`.

3. Ein Objekt mit den Tasten Lat (Breitengrad) und Lon (Längengrad)

   Dieses Format gilt für klassischen Schatten und benannten Schatten. Unterstützte Schlüssel: `lat`, `latitude`, `lon`, `long`, `longitude`. Ein Beispiel kann `{"lat": 41.12, "lon": -71.34}` sein.

4. Ein Array, das die Koordinaten darstellt

   Ein Array mit dem Format `[lat,lon]` oder `[lon,lat]`. Wenn Sie das Format `[lon,lat]` verwenden, das den Koordinaten in GeoJSON entspricht (gilt für klassischen Schatten und benannten Schatten), müssen Sie auch `order` in `geoLocations` angeben.

   Ein Beispiel kann sein:

```
{
  "location": {
    "coordinates": [
      **Longitude**,
      **Latitude**
    ],
    "type": "Point",
    "properties": {
      "country": "United States",
      "city": "New York",
      "postalCode": "*****",
      "horizontalAccuracy": 20,
      "horizontalConfidenceLevel": 0.67,
      "state": "New York",
      "timestamp": "2023-01-04T20:59:13.024Z"
    }
  }
```

```
}
```

Wie indexiert man Standortdaten

Die folgenden Schritte zeigen, wie Sie die Indexierungskonfiguration für Ihre Standortdaten aktualisieren und Geoabfragen verwenden, um nach Geräten zu suchen.

1. Erfahren Sie, wo Ihre Standortdaten gespeichert sind

   Die Flottenindizierung unterstützt derzeit die Indizierung von Standortdaten, die in klassischen Schatten oder benannten Schatten gespeichert sind.
2. Verwenden Sie unterstützte Standortdatenformate

   Stellen Sie sicher, dass Ihr Standortdatenformat einem der unterstützten Datenformate entspricht.
3. Aktualisieren Sie die Indexkonfiguration

   Aktivieren Sie mindestens die Konfiguration für die Indizierung von Objekten (Registrierung). Sie müssen auch die Indizierung für klassische Schatten oder Named Schatten aktivieren, die Ihre Standortdaten enthalten. Wenn Sie Ihre Objekt-Indizierung aktualisieren, sollten Sie Ihre Standortdaten in die Indexierungskonfiguration einbeziehen.
4. Erstellen und Ausführen von -Abfragen

   Erstellen Sie je nach Ihren Anwendungsfällen Geoabfragen und führen Sie sie aus, um nach Geräten zu suchen. Die Geoabfrage, die Sie erstellen, muss der Abfragesyntax entsprechen. Sie finden einige Beispiele in ???.

Konfiguration der Objektindizierung.

Um Standortdaten zu indizieren, müssen Sie die Indexkonfiguration aktualisieren und Ihre Standortdaten einbeziehen. Gehen Sie je nachdem, wo Ihre Standortdaten gespeichert sind, wie folgt vor, um Ihre Indexierungskonfiguration zu aktualisieren:

Standortdaten, die in klassischen Schatten gespeichert werden

Wenn Ihre Standortdaten in einem klassischen Shadow gespeichert sind, müssen Sie `thingIndexingMode` auf `REGISTRY_AND_SHADOW` einstellen und Ihre Standortdaten in den `geoLocations`-Feldern (`name`und`order`) unter `filter` angeben.

Im folgenden Konfigurationsbeispiel für die Indizierung geben Sie den Standortdatenpfad `shadow.reported.coordinates` als `name` und `LonLat` als `order` an.

```
{
 "thingIndexingMode": "REGISTRY_AND_SHADOW",
 "filter": {
  "geoLocations": [
   {
    "name": "shadow.reported.coordinates",
    "order": "LonLat"
   }
  ]
 }
}
```

- `thingIndexingMode`

  Der Indizierungsmodus steuert, ob Registry oder Shadow indexiert wird. Wenn `thingIndexingMode` auf `OFF` eingestellt ist, ist die Indizierung von Objekten deaktiviert.

  Um Standortdaten zu indizieren, die in einem klassischen Schatten gespeichert sind, müssen Sie `thingIndexingMode` den Wert auf `REGISTRY_AND_SHADOW` einstellen. Weitere Informationen finden Sie unter ???.

- `filter`

  Der Indexfilter bietet zusätzliche Auswahlmöglichkeiten für benannte Schatten und Geolokalisierungsdaten. Weitere Informationen finden Sie unter ???.

- `geoLocations`

  Die Liste der Geolocation-Ziele, die Sie für die Indizierung auswählen. Die Standardanzahl von Geolocation-Zielen für die Indizierung ist 1. Informationen zum Erhöhen von Limits finden Sie unter AWS IoT Device Management Kotingente.

- `name`

  Der Name des Geolocation-Zielfelds. Ein Beispielwert von `name` kann der Standortdatenpfad Ihres Schattens sein:`shadow.reported.coordinates`.

- `order`

Die Reihenfolge des Geolocation-Zielfeldes. Gültige Werte: `LatLon` und `LonLat`. `LatLon` bedeutet Breitengrad und Längengrad. `LonLat` bedeutet Längengrad und Breitengrad. Dies ist ein optionales Feld. Der Standardwert ist `LatLon`.

Standortdaten, die in benannten Schatten gespeichert sind

Wenn Ihre Standortdaten in einem benannten Schatten gespeichert sind, setzen Sie `namedShadowIndexingMode` auf den Wert `ON`, fügen Sie Ihre (n) benannten Schattennamen zum `namedShadowNames`-Feld in `filter`hinzu und geben Sie Ihren Standortdatenpfad im `geoLocations` Feld in `filter` an.

Im folgenden Konfigurationsbeispiel für die Indizierung geben Sie den Standortdatenpfad `shadow.name.namedShadow1.reported.coordinates` als `name` und `LonLat` als `order` an.

```
{
 "thingIndexingMode": "REGISTRY",
 "namedShadowIndexingMode": "ON",
 "filter": {
  "namedShadowNames": [
   "namedShadow1"
  ],
  "geoLocations": [
   {
    "name": "shadow.name.namedShadow1.reported.coordinates",
    "order": "LonLat"
   }
  ]
 }
}
```

- `thingIndexingMode`

  Der Indizierungsmodus steuert, ob Registry oder Shadow indexiert wird. Wenn `thingIndexingMode` auf `OFF` eingestellt ist, ist die Indizierung von Objekten deaktiviert.

  Um Standortdaten, die in einem benannten Schatten gespeichert sind, zu indizieren, müssen Sie `thingIndexingMode` den Wert auf `REGISTRY` (oder `REGISTRY_AND_SHADOW`) setzen. Weitere Informationen finden Sie unter ???.

- `filter`

Der Indexfilter bietet zusätzliche Auswahlmöglichkeiten für benannte Schatten und Geolokalisierungsdaten. Weitere Informationen finden Sie unter ???.

- `geoLocations`

  Die Liste der Geolocation-Ziele, die Sie für die Indizierung auswählen. Die Standardanzahl von Geolocation-Zielen für die Indizierung ist 1. Informationen zum Erhöhen von Limits finden Sie unter AWS IoT Device Management Kotingente.

- `name`

  Der Name des Geolocation-Zielfelds. Ein Beispielwert von `name` kann der Standortdatenpfad Ihres Schattens sein:`shadow.name.namedShadow1.reported.coordinates`.

- `order`

  Die Reihenfolge des Geolocation-Zielfeldes. Gültige Werte: `LatLon` und `LonLat`. `LatLon` bedeutet Breitengrad und Längengrad. `LonLat` bedeutet Längengrad und Breitengrad. Dies ist ein optionales Feld. Der Standardwert ist `LatLon`.

Beispiele für Geoqueries

Nachdem Sie die Indizierungskonfiguration für Ihre Standortdaten abgeschlossen haben, führen Sie Geoqueries aus, um nach Geräten zu suchen. Sie können Ihre Geoqueries auch mit anderen Abfragezeichenfolgen kombinieren. Weitere Informationen erhalten Sie unter ??? und ???.

Beispiel: Abfrage

In diesem Beispiel wird davon ausgegangen, dass die Standortdaten in einem benannten Schatten `gps-tracker` gespeichert sind. Die Ausgabe dieses Befehls ist die Liste der Geräte, die sich innerhalb einer radialen Entfernung von 15,5 km vom Mittelpunkt befinden, mit Koordinaten (47.6204, -122.3491).

```
aws iot search-index --query-string \
"shadow.name.gps-tracker.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km"
```

Beispiel: Abfrage

In diesem Beispiel wird davon ausgegangen, dass die Standortdaten in einem klassischen Schatten gespeichert sind. Die Ausgabe dieses Befehls ist die Liste der Geräte, die sich innerhalb einer radialen Entfernung von 15,5 km vom Mittelpunkt befinden, mit Koordinaten (47.6204, -122.3491).

```
aws iot search-index --query-string \
"shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km"
```

Beispiel: Abfrage

In diesem Beispiel wird davon ausgegangen, dass die Standortdaten in einem klassischen Schatten gespeichert sind. Die Ausgabe dieses Befehls ist die Liste der Geräte, die nicht angeschlossen sind und sich außerhalb der radialen Entfernung von 15,5 km vom Mittelpunkt befinden, mit Koordinaten (47.6204, -122.3491).

```
aws iot search-index --query-string \
"connectivity.connected:false AND (NOT
 shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km)"
```

Erste Schritte-Tutorial

In diesem Tutorial wird gezeigt, wie Sie mithilfe der Flottenindizierung Ihre Standortdaten indexieren können. Der Einfachheit halber erstellen Sie ein Objekt, das Ihr Gerät darstellt, und speichern die Standortdaten in einem benannten Schatten, aktualisieren die Konfiguration der Objektindizierung für die Standortindizierung und führen Beispiel-Geoabfragen aus, um nach Geräten innerhalb einer radialen Grenze zu suchen.

Für dieses Tutorial brauchen Sie ungefähr 20 Minuten.

In diesem Thema:

- Voraussetzungen
- Objekte und Schatten erstellen
- Konfiguration der Objektindizierung.
- Ausführen von Geoquery

Voraussetzungen

- Installieren der neuesten Version von AWS CLI
- Machen Sie sich mit der Standortindizierung und Geoabfragen, der Verwaltung der Dingindizierung und der Abfragesyntax vertraut.

Objekte und Schatten erstellen

Sie erstellen ein Objekt, das Ihr Gerät repräsentiert, und einen benannten Schatten, um dessen Standortdaten zu speichern (Koordinaten 47.61564, -122.33584).

1. Führe den folgenden Befehl aus, um dein Ding zu erstellen, das den Namen deines Fahrrads repräsentiert. Bike-1 Weitere Informationen zum Erstellen eines Dings mithilfe AWS CLI von Reference finden Sie unter Create-Thing from AWS CLI Reference.

   ```
   aws iot create-thing --thing-name "Bike-1" \
   --attribute-payload '{"attributes": {"model":"OEM-2302-12", "battery":"35",
    "acqDate":"06/09/23"}}'
   ```

   Die Ausgabe dieses Befehls kann folgendermaßen aussehen:

   ```
   {
       "thingName": "Bike-1",
       "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/Bike-1",
       "thingId": "df9cf01d-b0c8-48fe-a2e2-e16cff6b23df"
   }
   ```

2. Führen Sie den folgenden Befehl aus, um einen benannten Schatten zum Speichern Bike-1 der Standortdaten (Koordinaten 47.61564, -122.33584) zu erstellen. Weitere Informationen zum Erstellen eines benannten Schattens mithilfe von Reference finden Sie unter update-thing-shadow. AWS CLIhttps://docs.aws.amazon.com/cli/latest/reference/iot-data/update-thing-shadow.html AWS CLI

   ```
   aws iot-data update-thing-shadow \
   --thing-name Bike-1 \
   --shadow-name Bike1-shadow \
   --cli-binary-format raw-in-base64-out \
   --payload '{"state":{"reported":{"coordinates":{"lat": 47.6153, "lon": -122.3333}}}}'
    \
   "output.txt" \
   ```

   Dieser Befehl liefert keine Ausgabe. Um den von Ihnen erstellten benannten Schatten anzuzeigen, können Sie den CLI-Befehl list-named-shadows-for-thing ausführen.

   ```
   aws iot-data list-named-shadows-for-thing --thing-name Bike-1
   ```

Die Ausgabe dieses Befehls sieht wie folgt aus:

```
{
    "results": [
        "Bike1-shadow"
    ],
    "timestamp": 1699574309
}
```

Konfiguration der Objektindizierung.

Um Ihre Standortdaten zu indizieren, müssen Sie Ihre Objekt-Indizierungskonfiguration so aktualisieren, dass sie die Standortdaten einbezieht. Da Ihre Standortdaten in diesem Tutorial in einem benannten Schatten gespeichert werden, setzen Sie `thingIndexingMode` auf `REGISTRY` (bei einer Mindestanforderung), setzen Sie `namedShadowIndexingMode` auf `ON` und fügen Sie Ihre Standortdaten zur Konfiguration hinzu. In diesem Beispiel müssen Sie den Namen Ihres benannten Schattens und den Pfad für die Standortdaten des Schattens zu `filter` hinzufügen.

1. Führen Sie den Befehl aus, um Ihre Indizierungskonfiguration für die Standortindizierung zu aktualisieren.

```
aws iot update-indexing-configuration --cli-input-json '{
"thingIndexingConfiguration": { "thingIndexingMode": "REGISTRY",
"thingConnectivityIndexingMode": "OFF",
"deviceDefenderIndexingMode": "OFF",
"namedShadowIndexingMode": "ON",
"filter": {
    "namedShadowNames": ["Bike1-shadow"],
    "geoLocations":[{
        "name":"shadow.name.Bike1-shadow.reported.coordinates"
    }]
},
"customFields": [
{ "name":"attributes.battery",
"type":"Number"}] } }'
```

Der Befehl erzeugt keine Ausgabe. Möglicherweise müssen Sie einen Moment warten, bis das Update abgeschlossen ist. Führen Sie den CLI-Befehl describe-index aus, um den Status zu

überprüfen. Wenn `indexStatus` Folgendes anzeigt:`ACTIVE`, ist das Indizierungsupdate Ihres Objekts abgeschlossen.

2. Führen Sie den Befehl aus, um Ihre Konfiguration zu überprüfen. Dieser Schritt ist optional.

```
aws iot get-indexing-configuration
```

Die Ausgabe sieht wie folgt aus:

```
{
    "thingIndexingConfiguration": {
        "thingIndexingMode": "REGISTRY",
        "thingConnectivityIndexingMode": "OFF",
        "deviceDefenderIndexingMode": "OFF",
        "namedShadowIndexingMode": "ON",
        "managedFields": [
            {
                "name": "shadow.name.*.hasDelta",
                "type": "Boolean"
            },
            {
                "name": "registry.version",
                "type": "Number"
            },
            {
                "name": "registry.thingTypeName",
                "type": "String"
            },
            {
                "name": "registry.thingGroupNames",
                "type": "String"
            },
            {
                "name": "shadow.name.*.version",
                "type": "Number"
            },
            {
                "name": "thingName",
                "type": "String"
            },
            {
                "name": "thingId",
                "type": "String"
```

```
            }
        ],
        "customFields": [
            {
                "name": "attributes.battery",
                "type": "Number"
            }
        ],
        "filter": {
            "namedShadowNames": [
                "Bike1-shadow"
            ],
            "geoLocations": [
                {
                    "name": "shadow.name.Bike1-shadow.reported.coordinates",
                    "order": "LatLon"
                }
            ]
        }
    },
    "thingGroupIndexingConfiguration": {
        "thingGroupIndexingMode": "OFF"
    }
}
```

Ausführen von Geoquery

Jetzt haben Sie Ihre Objekt-Indizierungskonfiguration aktualisiert, sodass sie die Standortdaten enthält. Versuchen Sie, einige Geoqueries zu erstellen und sie auszuführen, um zu sehen, ob Sie die gewünschten Suchergebnisse erhalten können. Eine Geoabfrage muss der Abfragesyntax folgen. Einige nützliche Beispiel-Geoabfragen finden Sie unter ???.

Im folgenden Beispielbefehl verwenden Sie die Geoabfrage, `shadow.name.Bike1-shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km` um mit Koordinaten (47.6204, -122.3491) nach Geräten zu suchen, die sich innerhalb einer radialen Entfernung von 15,5 km vom Mittelpunkt befinden.

```
aws iot search-index --query-string "shadow.name.Bike1-
shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km"
```

Da Sie ein Gerät an den Koordinaten „lat“: 47.6153, „lon“: -122.3333 haben, das in einer Entfernung von 15,5 km vom Mittelpunkt liegt, sollten Sie dieses Gerät () in der Ausgabe sehen können. Bike-1 Die Ausgabe sieht wie folgt aus:


```
{
    "things": [
        {
            "thingName": "Bike-1",
            "thingId": "df9cf01d-b0c8-48fe-a2e2-e16cff6b23df",
            "attributes": {
                "acqDate": "06/09/23",
                "battery": "35",
                "model": "OEM-2302-12"
            },
            "shadow": "{\"reported\":{\"coordinates\":{\"lat\":47.6153,\"lon
\":-122.3333}},\"metadata\":{\"reported\":{\"coordinates\":{\"lat\":{\"timestamp
\":1699572906},\"lon\":{\"timestamp\":1699572906}}}},\"hasDelta\":false,\"version\":1}"
        }
    ]
}
```


Weitere Informationen finden Sie unter ???.

Flottenmetriken

Flottenkennzahlen sind eine Funktion der Flottenindizierung, eines verwalteten Dienstes, mit dem Sie die Daten Ihrer Geräte indexieren, durchsuchen und aggregieren können. AWS IoT Mithilfe von Flottenmetriken können Sie den Gesamtstatus Ihrer Flottengeräte im CloudWatch Laufe der Zeit überwachen, einschließlich der Überprüfung der Verbindungsrate Ihrer Flottengeräte oder der durchschnittlichen Änderungen des Akkustands in einem bestimmten Zeitraum.

Mithilfe von Flottenkennzahlen können Sie Aggregationsabfragen erstellen, deren Ergebnisse kontinuierlich CloudWatch als Metriken zur Analyse von Trends und zur Erstellung von Alarmen ausgegeben werden. Für Ihre Überwachungsaufgaben können Sie die Aggregationsabfragen verschiedener Aggregationstypen (Statistik, Kardinalität und Perzentil) angeben. Sie können all Ihre Aggregationsabfragen speichern, um Flottenmetriken für die zukünftige Wiederverwendung zu erstellen.

Erste Schritte-Tutorial

In diesem Tutorial erstellen Sie eine Flottenmetrik, um die Temperaturen Ihrer Sensoren zu überwachen und potenzielle Anomalien zu erkennen. Bei der Erstellung der Flottenmetrik definieren Sie eine Aggregationsabfrage, die die Anzahl der Sensoren mit Temperaturen über ca. 80 Grad Fahrenheit erkennt. Sie geben an, dass die Abfrage alle 60 Sekunden ausgeführt wird CloudWatch, und die Abfrageergebnisse werden an diese Adresse gesendet. Dort können Sie die Anzahl der Sensoren anzeigen, bei denen potenzielle Hochtemperaturrisiken bestehen, und Alarme einrichten. Zum Abschließen dieses Tutorials wird die AWS CLI verwendet.

In diesem Tutorial lernen Sie Folgendes:

- Einrichten
- Erstellen einer Flottenmetrik
- Messwerte anzeigen in CloudWatch
- Bereinigen von Ressourcen

Für dieses Tutorial brauchen Sie ungefähr 15 Minuten.

Voraussetzungen

- Installieren der neuesten Version von AWS CLI
- Vertrautmachen mit Abfragen von Aggregatdaten
- Machen Sie sich mit der Verwendung von CloudWatch Amazon-Metriken vertraut

Einrichten

Um Flottenmetriken zu verwenden, aktivieren Sie die Flottenindizierung. Um die Flottenindizierung für Ihre Objekte oder Objektgruppen mit bestimmten Datenquellen und zugehörigen Konfigurationen zu aktivieren, folgen Sie den Anweisungen unter Verwaltung der Objektindizierung und Verwalten der Objektgruppenindizierung.

So führen Sie die Einrichtung durch:

1. Führen Sie den folgenden Befehl aus, um die Flottenindizierung zu aktivieren, und geben Sie die Datenquellen an, in denen gesucht werden soll.

```
aws iot update-indexing-configuration \
--thing-indexing-configuration
 "thingIndexingMode=REGISTRY_AND_SHADOW,customFields=[{name=attributes.temperature,type=Num
{name=attributes.rackId,type=String},
{name=attributes.stateNormal,type=Boolean}],thingConnectivityIndexingMode=STATUS" \
```

Der obenstehende CLI-Beispielbefehl ermöglicht die Flottenindizierung, um die Suche nach Registrierungsdaten, Schattendaten und dem Status der Objektkonnektivität mithilfe des `AWS_Things`-Index zu unterstützen.

Die Änderung der Konfiguration kann einige Minuten in Anspruch nehmen. Stellen Sie sicher, dass Ihre Flottenindizierung aktiviert ist, bevor Sie Flottenmetriken erstellen.

Führen Sie zum Überprüfen, ob Ihre Flottenindizierung aktiviert wurde, den folgenden CLI-Befehl aus:

```
aws --region us-east-1 iot describe-index --index-name "AWS_Things"
```

Weitere Informationen finden Sie unter Aktivieren der Objektindizierung.

2. Führen Sie das folgende Bash-Skript aus, um zehn Objekte zu erstellen und zu beschreiben.

```
# Bash script. Type `bash` before running in other shells.

Temperatures=(70 71 72 73 74 75 47 97 98 99)
Racks=(Rack1 Rack1 Rack2 Rack2 Rack3 Rack4 Rack5 Rack6 Rack6 Rack6)
IsNormal=(true true true true true true false false false false)

for ((i=0; i < 10; i++))
do
  thing=$(aws iot create-thing --thing-name "TempSensor$i" --attribute-
payload attributes="{temperature=${Temperatures[@]:$i:1},rackId=${Racks[@]:
$i:1},stateNormal=${IsNormal[@]:$i:1}}")
  aws iot describe-thing --thing-name "TempSensor$i"
done
```

Dieses Skript erstellt zehn Objekte, die zehn Sensoren repräsentieren. Jedes Objekt hat die Attribute `temperature`, `rackId` und `stateNormal`, wie in der folgenden Tabelle beschrieben:

Attribut	Datentyp	Description
`temperature`	Zahl	Temperaturwert in Fahrenheit
`rackId`	Zeichenfolge	ID des Server-Racks, das Sensoren enthält
`stateNormal`	Boolesch	Ob der Temperaturwert des Sensors normal ist oder nicht

Die Ausgabe dieses Skripts enthält zehn JSON-Dateien. Eine der JSON-Dateien sieht wie folgt aus:

```
{
    "version": 1,
    "thingName": "TempSensor0",
    "defaultClientId": "TempSensor0",
    "attributes": {
        "rackId": "Rack1",
        "stateNormal": "true",
        "temperature": "70"
    },
    "thingArn": "arn:aws:iot:region:account:thing/TempSensor0",
    "thingId": "example-thing-id"
}
```

Weitere Informationen finden Sie unter Ein Objekt erstellen.

Erstellen einer Flottenmetrik

So erstellen Sie eine Flottenmetrik:

1. Führen Sie den folgenden Befehl aus, um eine Flottenmetrik mit dem Namen zu erstellen*high_temp_FM*. Sie erstellen die Flottenmetrik, um die Anzahl der Sensoren mit Temperaturen über 80 Grad Fahrenheit zu überwachen. CloudWatch

```
aws iot create-fleet-metric --metric-name "high_temp_FM" --query-string
 "thingName:TempSensor* AND attributes.temperature >80" --period 60 --aggregation-
field "attributes.temperature" --aggregation-type name=Statistics,values=count
```

--metric-name

Datentyp: Zeichenfolge. Der Parameter `--metric-name` gibt den Namen einer Flottenmetrik an. In diesem Beispiel erstellen Sie eine Flottenmetrik mit dem Namen high_temp_FM.

--query-string

Datentyp: Zeichenfolge. Der Parameter `--query-string` gibt die Abfragezeichenfolge an. In diesem Beispiel bedeutet die Abfragezeichenfolge, dass alle Dinge abgefragt werden, deren Namen mit TempSensor und mit Temperaturen über 80 Grad Fahrenheit beginnen. Weitere Informationen finden Sie unter Abfragesyntax.

--period

Datentyp: Ganzzahl. Der Parameter `--period` gibt die Zeit in Sekunden für die Abfrage der aggregierten Daten an. In diesem Beispiel geben Sie an, dass die Flottenmetrik, die Sie erstellen, die aggregierten Daten alle 60 Sekunden abruft.

--aggregation-field

Datentyp: Zeichenfolge. Der Parameter `--aggregation-field` gibt das auszuwertende Attribut an. In diesem Beispiel soll das Temperaturattribut ausgewertet werden.

--aggregation-type

Der Parameter`--aggregation-type` gibt die statistische Zusammenfassung an, die in der Flottenmetrik angezeigt werden soll. Für Ihre Überwachungsaufgaben können Sie die Eigenschaften von Aggregationsabfragen für die verschiedenen Aggregationstypen (Statistik, Kardinalität und Perzentil) anpassen. In diesem Beispiel geben Sie Anzahl für den Aggregationstyp und Statistics an, um die Anzahl der Geräte zurückzugeben, deren Attribute der Abfrage entsprechen, mit anderen Worten, um die Anzahl der Geräte zurückzugeben, deren Namen mit TempSensor und mit Temperaturen über 80 Grad Fahrenheit beginnen. Weitere Informationen finden Sie unter Abfragen von Aggregatdaten.

Die Ausgabe dieses Befehls sieht wie folgt aus:

```
{
    "metricArn": "arn:aws:iot:region:111122223333:fleetmetric/high_temp_FM",
    "metricName": "high_temp_FM"
}
```

> **Note**
>
> Es kann einen Moment dauern, bis die Datenpunkte angezeigt werden. CloudWatch

Weitere Informationen zum Erstellen einer Flottenmetrik finden Sie unter Verwalten von Flottenmetriken.

Wenn Sie keine Flottenmetrik erstellen können, lesen Sie den Artikel Problembehebung bei Flottenmetriken.

2. (Optional) Führen Sie den folgenden Befehl aus, um Ihre Flottenmetrik mit dem Namen high_temp_FM zu beschreiben.

```
aws iot describe-fleet-metric --metric-name "high_temp_FM"
```

Die Ausgabe dieses Befehls sieht wie folgt aus:

```
{
    "queryVersion": "2017-09-30",
    "lastModifiedDate": 1625249775.834,
    "queryString": "*",
    "period": 60,
    "metricArn": "arn:aws:iot:region:111122223333:fleetmetric/high_temp_FM",
    "aggregationField": "registry.version",
    "version": 1,
    "aggregationType": {
        "values": [
            "count"
        ],
        "name": "Statistics"
    },
    "indexName": "AWS_Things",
    "creationDate": 1625249775.834,
    "metricName": "high_temp_FM"
```

```
}
```

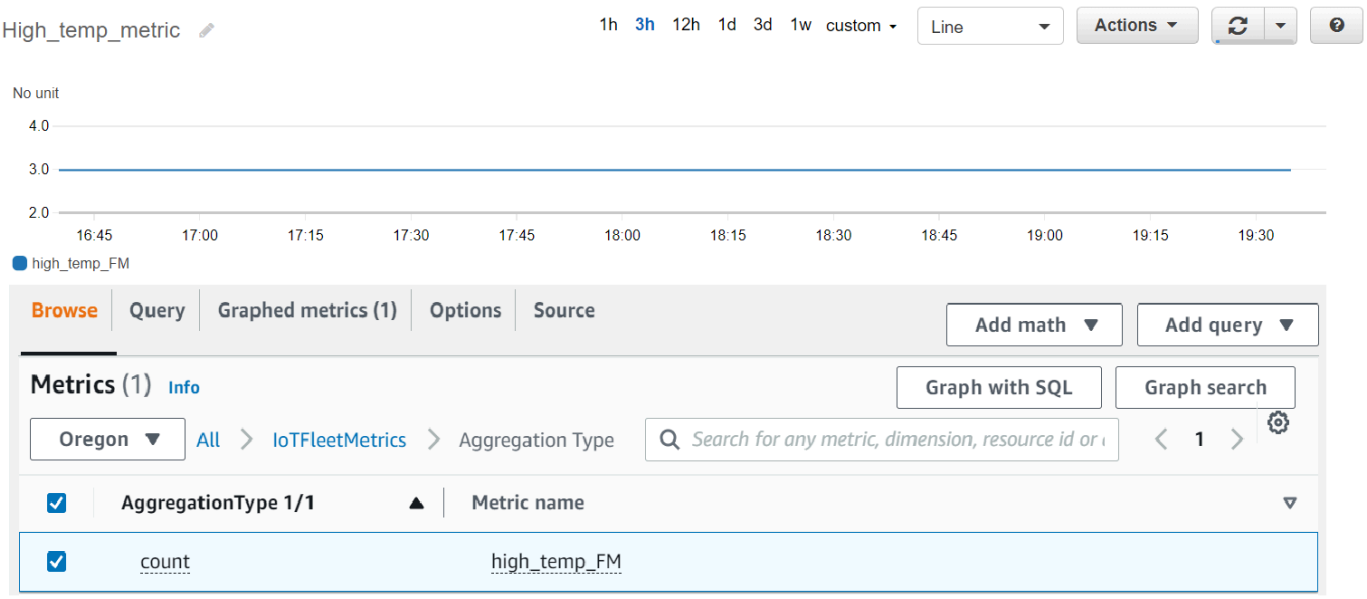
Sehen Sie sich Flottenkennzahlen an unter CloudWatch

Nachdem Sie die Flottenmetrik erstellt haben, können Sie die Metrikdaten in einsehen CloudWatch. In diesem Tutorial sehen Sie die Metrik, die die Anzahl der Sensoren anzeigt, deren Namen mit TempSensor und mit Temperaturen über 80 Grad Fahrenheit beginnen.

Um Datenpunkte anzuzeigen in CloudWatch

1. Öffnen Sie die CloudWatch Konsole unter https://console.aws.amazon.com/cloudwatch/.
2. Wählen Sie im CloudWatch Menü im linken Bereich Metriken aus, um das Untermenü zu erweitern, und wählen Sie dann Alle Metriken aus. Dadurch wird eine Seite geöffnet, bei der die obere Hälfte das Diagramm und die untere Hälfte vier Abschnitte mit Registerkarten enthält.
3. Im ersten Abschnitt „Alle Metriken“ sind alle Metriken aufgeführt, die Sie in Gruppen anzeigen können. Wählen Sie aus. IoTFleetMetrics Diese enthält all Ihre Flottenmetriken.
4. Wählen Sie auf der Registerkarte Alle Metriken im Abschnitt Aggregationstyp die Option Aggregationstyp aus, um alle von Ihnen erstellten Flottenmetriken anzuzeigen.
5. Wählen Sie links im Abschnitt Aggregationstyp die Flottenmetrik aus, für die das Diagramm angezeigt werden soll. Sie sehen den Wert *count* links neben Ihrem Metriknamen. Dies ist der Wert des Aggregationstyps, den Sie in diesem Tutorial im Abschnitt Flottenmetriken erstellen angegeben haben.
6. Wählen Sie die zweite Registerkarte mit dem Namen Grafische Metriken rechts neben der Registerkarte Alle Metriken, um die Flottenmetrik anzuzeigen, die Sie im vorherigen Schritt ausgewählt haben.

   Sie sollten ein Diagramm sehen, das die Anzahl der Sensoren mit Temperaturen über 80 Grad Fahrenheit anzeigt, wie in diesem Beispiel:



> **Note**
>
> Das Periodenattribut CloudWatch ist standardmäßig auf 5 Minuten eingestellt. Es ist das Zeitintervall zwischen Datenpunkten, die in CloudWatch angezeigt werden. Sie können die Einstellung für den Zeitraum je nach Bedarf ändern.

7. (Optional) Sie können einen metrischen Alarm einrichten.

   1. Wählen Sie im CloudWatch Menü auf der linken Seite Alarme, um das Untermenü zu erweitern, und wählen Sie dann Alle Alarme aus.
   2. Wählen Sie auf der Seite Alarme in der oberen rechten Ecke die Option Alarm erstellen aus. Folgen Sie den Anweisungen zum Erstellen eines Alarms in der Konsole, um bei Bedarf einen Alarm zu erstellen. Weitere Informationen finden Sie unter CloudWatch Amazon-Alarme https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/AlarmThatSendsEmail.html verwenden.

Weitere Informationen finden Sie unter Verwenden von CloudWatch Amazon-Metriken.

Wenn Sie darin keine Datenpunkte sehen können CloudWatch, lesen Sie den Artikel zur Fehlerbehebung bei Flottenkennzahlen.

Bereinigen

So löschen Sie Flottenkennzahlen:

Verwenden Sie den delete-fleet-metric-CLI-Befehl, um Flottenmetriken zu löschen.

Führen Sie den folgenden Befehl aus, um die Flottenmetrik mit dem Namen high_temp_FM zu löschen.

```
aws iot delete-fleet-metric --metric-name "high_temp_FM"
```

So löschen Sie Objekte:

Mit dem CLI-Befehl delete-thing können Sie ein Objekt löschen.

Um die zehn Objekte zu löschen, die Sie erstellt haben, führen Sie das folgende Skript aus:

```
# Bash script. Type `bash` before running in other shells.

for ((i=0; i < 10; i++))
do
  thing=$(aws iot delete-thing --thing-name "TempSensor$i")
done
```

Um Metriken zu bereinigen in CloudWatch

CloudWatch unterstützt das Löschen von Metriken nicht. Metriken laufen je nach ihren Aufbewahrungplänen ab. Weitere Informationen finden Sie unter Verwenden von CloudWatch Amazon-Metriken.

Verwalten von Flottenmetriken

In diesem Thema erfahren Sie, wie Sie die AWS IoT Konsole verwenden und AWS CLI Ihre Flottenkennzahlen verwalten.

Themen

- Verwalten von Flottenmetriken (Konsole)
- Verwalten von Flottenmetriken (CLI)
- Autorisieren des Taggings von IoT-Ressourcen

Verwalten von Flottenmetriken (Konsole)

In den folgenden Abschnitten wird gezeigt, wie Sie die AWS IoT Konsole zur Verwaltung Ihrer Flottenkennzahlen verwenden. Stellen Sie sicher, dass Sie die Flottenindizierung mit den zugehörigen Datenquellen und Konfigurationen aktiviert haben, bevor Sie Flottenmetriken erstellen.

Aktivieren der Flottenindizierung

Wenn Sie die Flottenindizierung bereits aktiviert haben, überspringen Sie diesen Abschnitt.

Wenn Sie die Flottenindizierung nicht aktiviert haben, folgen Sie diesen Anweisungen.

1. Öffnen Sie Ihre AWS IoT Konsole unter https://console.aws.amazon.com/iot/.
2. Wählen Sie im AWS IoT Menü Einstellungen aus.
3. Um die detaillierten Einstellungen einzusehen, scrollen Sie auf der Seite Einstellungen nach unten zum Abschnitt Flottenindizierung.
4. Um Ihre Einstellungen für die Flottenindizierung zu aktualisieren, wählen Sie rechts neben dem Abschnitt Flottenindizierung die Option Indizierung verwalten aus.
5. Aktualisieren Sie auf der Seite Einstellungen für die Flottenindizierung verwalten Ihre Einstellungen für die Flottenindizierung nach Ihren Anforderungen.
   - Konfiguration

     Um die Objektindizierung zu aktivieren, aktivieren Sie die Objektindizierung und wählen Sie dann die Datenquellen aus, für die Sie die Indizierung durchführen möchten.

     Um die Indizierung von Objektgruppen zu aktivieren, aktivieren Sie die Indexierung von Thing-Gruppen.
   - Benutzerdefinierte Felder für die Aggregation – optional

     Benutzerdefinierte Felder sind eine Liste von Paaren aus Feldnamen und Feldtypen.

     Um ein benutzerdefiniertes Feldpaar hinzuzufügen, wählen Sie Neues Feld hinzufügen. Geben Sie einen benutzerdefinierten Feldnamen ein, wie z. B. `attributes.temperature`, und wählen Sie dann einen Feldtyp aus dem Menü Feldtyp aus. Beachten Sie, dass ein benutzerdefinierter Feldname mit `attributes.` beginnt und als Attribut gespeichert wird, um Abfragen zur Aggregation von Objekten auszuführen.

     Um die Einstellung zu aktualisieren und zu speichern, wählen Sie Aktualisieren.

Erstellen einer Flottenmetrik

1. Öffnen Sie Ihre AWS IoT Konsole unter https://console.aws.amazon.com/iot/.
2. Wählen Sie im AWS IoT Menü Verwalten und dann Flottenkennzahlen aus.
3. Wählen Sie auf der Seite Flottenmetriken die Option Flottenmetrik erstellen aus und schließen Sie die Schritte zur Erstellung ab.
4. Konfigurieren Sie in Schritt 1 Flottenmetriken.
   - Geben Sie im Abschnitt Abfrage eine Abfragezeichenfolge ein, um die Objekte oder Objektgruppen anzugeben, für die Sie die Aggregatsuche durchführen möchten. Die Abfragezeichenfolge besteht aus einem Attribut und einem Wert. Wählen Sie unter Eigenschaften das gewünschte Attribut aus, oder geben Sie das Attribut in das Feld ein, falls es nicht in der Liste angezeigt wird. Geben Sie den Wert nach `:` ein. Ein Beispiel für eine Abfragezeichenfolge kann `thingName:TempSensor*` sein. Drücken Sie für jede eingegebene Abfragezeichenfolge die Eingabetaste auf Ihrer Tastatur. Wenn Sie mehrere Abfragezeichenfolgen eingeben, geben Sie deren Beziehung an, indem Sie und, oder und nicht oder oder nicht auswählen.
   - Wählen Sie in den Berichtseigenschaften Indexname, Aggregationstyp und Aggregationsfeld aus den jeweiligen Listen aus. Wählen Sie als Nächstes unter Daten auswählen die Daten aus, die Sie aggregieren möchten. Sie können dabei mehrere Datenwerte auswählen.
   - Wählen Sie Weiter aus.
5. Geben Sie in Schritt 2 die Eigenschaften der Flottenmetrik an.
   - Geben Sie im Feld Name der Flottenmetrik einen Namen für die Flottenmetrik ein, die Sie erstellen.
   - Geben Sie im Feld Beschreibung – optional eine Beschreibung für die Flottenmetrik ein, die Sie erstellen. Dies ist ein optionales Feld.
   - Geben Sie in den Feldern „Stunden“ und „Minuten“ die Zeit (wie oft) ein, zu der die Flottenkennzahl Daten ausgeben soll CloudWatch.
   - Wählen Sie Weiter aus.
6. Überprüfen und Erstellen Sie Ihre Metrik in Schritt 5.
   - Überprüfen Sie die Einstellungen aus Schritt 1 und Schritt 2. Zum Bearbeiten der Einstellungen wählen Sie Bearbeiten.
   - Wählen Sie Flottenmetrik erstellen aus.

Nach erfolgreicher Erstellung wird die Flottenmetrik auf der Seite Flottenmetrik aufgeführt.

Aktualisieren einer Flottenmetrik

1. Wählen Sie auf der Seite Flottenmetrik die Flottenmetrik aus, die Sie aktualisieren möchten.
2. Klicken Sie auf der Seite Details auf Bearbeiten. Dadurch werden die Schritte zur Erstellung geöffnet, in denen Sie Ihre Flottenmetrik in einem der drei Schritte aktualisieren können.
3. Nachdem Sie die Aktualisierung der Flottenmetrik abgeschlossen haben, wählen Sie Flottenmetrik aktualisieren aus.

Löschen einer Flottenmetrik

1. Wählen Sie auf der Seite Flottenmetrik die Flottenmetrik aus, die Sie löschen möchten.
2. Wählen Sie auf der nächsten Seite, auf der Details zu Ihrer Flottenmetrik angezeigt werden, Löschen aus.
3. Geben Sie im Dialogfeld den Namen Ihrer Flottenmetrik ein, um das Löschen zu bestätigen.
4. Wählen Sie Löschen aus. In diesem Schritt wird Ihre Flottenmetrik dauerhaft gelöscht.

Verwalten von Flottenmetriken (CLI)

In den folgenden Abschnitten wird gezeigt, wie Sie die AWS CLI zur Verwaltung Ihrer Flottenkennzahlen verwenden. Stellen Sie sicher, dass Sie die Flottenindizierung mit den zugehörigen Datenquellen und Konfigurationen aktiviert haben, bevor Sie Flottenmetriken erstellen. Um die Flottenindizierung für Ihre Objekte oder Objektgruppen zu aktivieren, folgen Sie den Anweisungen unter Verwalten der Objektindizierung oder Verwalten der Objektgruppenindizierung.

Erstellen einer Flottenmetrik

Sie können den CLI-Befehl create-fleet-metric verwenden, um eine Flottenmetrik zu erstellen.

```
aws iot create-fleet-metric --metric-name "YourFleetMetricName" --query-string
 "*" --period 60 --aggregation-field "registry.version" --aggregation-type
 name=Statistics,values=sum
```

Die Ausgabe dieses Befehls enthält den Namen und den Amazon-Ressourcennamen (ARN) Ihrer Flottenmetrik. Die Ausgabe sollte wie folgt aussehen:

```
{
    "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/YourFleetMetricName",
```

```
    "metricName": "YourFleetMetricName"
}
```

Auflisten von Flottenmetriken

Sie können den CLI-Befehl list-fleet-metric verwenden, um alle Flottenmetriken in Ihrem Konto aufzulisten.

```
aws iot list-fleet-metrics
```

Die Ausgabe dieses Befehls enthält alle Flottenmetriken. Die Ausgabe sollte wie folgt aussehen:

```
{
    "fleetMetrics": [
        {
            "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/
YourFleetMetric1",
            "metricName": "YourFleetMetric1"
        },
        {
            "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/
YourFleetMetric2",
            "metricName": "YourFleetMetric2"
        }
    ]
}
```

Beschreiben einer Flottenmetrik

Sie können den CLI-Befehl describe-fleet-metric verwenden, um detailliertere Informationen zu einer Flottenmetrik anzuzeigen.

```
aws iot describe-fleet-metric --metric-name "YourFleetMetricName"
```

Die Ausgabe des Befehls enthält detaillierte Informationen über die angegebene Flottenmetrik. Die Ausgabe sollte wie folgt aussehen:

```
{
    "queryVersion": "2017-09-30",
    "lastModifiedDate": 1625790642.355,
    "queryString": "*",
```

```
    "period": 60,
    "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/YourFleetMetricName",
    "aggregationField": "registry.version",
    "version": 1,
    "aggregationType": {
        "values": [
            "sum"
        ],
        "name": "Statistics"
    },
    "indexName": "AWS_Things",
    "creationDate": 1625790642.355,
    "metricName": "YourFleetMetricName"
}
```

Aktualisieren einer Flottenmetrik

Sie können den CLI-Befehl update-fleet-metric verwenden, um eine Flottenmetrik zu aktualisieren.

```
aws iot update-fleet-metric --metric-name "YourFleetMetricName" --query-string
 "*" --period 120 --aggregation-field "registry.version" --aggregation-type
 name=Statistics,values=sum,count --index-name AWS_Things
```

Der Befehl update-fleet-metric erzeugt keine Ausgabe. Mit dem CLI-Befehl describe-fleet-metric können Sie das Ergebnis anzeigen.

```
{
    "queryVersion": "2017-09-30",
    "lastModifiedDate": 1625792300.881,
    "queryString": "*",
    "period": 120,
    "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/YourFleetMetricName",
    "aggregationField": "registry.version",
    "version": 2,
    "aggregationType": {
        "values": [
            "sum",
            "count"
        ],
        "name": "Statistics"
    },
    "indexName": "AWS_Things",
    "creationDate": 1625792300.881,
```

```
    "metricName": "YourFleetMetricName"
}
```

Löschen einer Flottenmetrik

Verwenden Sie den CLI-Befehl delete-fleet-metric, um eine Flottenmetrik zu löschen.

```
aws iot delete-fleet-metric --metric-name "YourFleetMetricName"
```

Dieser Befehl erzeugt keine Ausgabe, wenn das Löschen erfolgreich ist oder wenn Sie eine Flottenmetrik angeben, die nicht existiert.

Weitere Informationen finden Sie unter Problembehebung bei Flottenmetriken.

Autorisieren des Taggings von IoT-Ressourcen

Für eine bessere Kontrolle über Flottenmetriken, die Sie erstellen, ändern oder verwenden, können Sie den Flottenmetriken Tags hinzufügen.

Um Flottenkennzahlen, die Sie mithilfe von AWS-Managementkonsole oder erstellen AWS CLI, zu kennzeichnen, müssen Sie die `iot:TagResource` Aktion in Ihre IAM-Richtlinie aufnehmen, um den Benutzern Berechtigungen zu gewähren. Wenn Ihre IAM-Richtlinie `iot:TagResource` nicht beinhaltet, wird bei allen Aktionen zur Erstellung einer Flottenmetrik mit einem Tag ein Fehler `AccessDeniedException` zurückgegeben.

Weitere allgemeine Informationen zum Taggen von Ressourcen finden Sie unter Markieren Ihrer AWS IoT -Ressourcen.

Beispiele für IAM-Richtlinien

Sehen Sie sich das folgende Beispiel für eine IAM-Richtlinie an, in dem Tagging-Berechtigungen gewährt werden, wenn Sie eine Flottenmetrik erstellen:

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Action": [
    "iot:TagResource"
   ],
```

```
    "Effect": "Allow",
    "Resource": [
     "arn:aws:iot:*:*:fleetmetric/*"
    ]
   },
   {
    "Action": [
     "iot:CreateFleetMetric"
    ],
    "Effect": "Allow",
    "Resource": [
     "arn:aws:iot:*:*:index/*",
     "arn:aws:iot:*:*:fleetmetric/*"
    ]
   }
  ]
}
```

Weitere Informationen finden Sie unter Aktionen, Ressourcen und Bedingungsschlüssel für AWS IoT.

MQTT-based Dateizustellung

Eine Option, mit der Sie Dateien verwalten und auf AWS IoT Geräte in Ihrer Flotte übertragen können, ist die MQTT-based Dateizustellung. Mit dieser Funktion in der AWS Cloud können Sie einen Stream erstellen, der mehrere Dateien enthält, Sie können Stream-Daten (die Dateiliste und Beschreibungen) aktualisieren, die Stream-Daten abrufen und vieles mehr. AWS IoT MQTT-based Bei der Dateibereitstellung können Daten in kleinen Blöcken auf Ihre IoT-Geräte übertragen werden. Dabei wird das MQTT-Protokoll mit Unterstützung für Anfrage- und Antwortnachrichten in JSON oder CBOR verwendet.

Weitere Informationen zu den Möglichkeiten, Daten von und zu IoT-Geräten mithilfe von IoT-Geräten zu übertragen AWS IoT, finden Sie unter. Geräte verbinden mit AWS IoT

Themen

- Was ist ein Stream?
- Verwaltung eines Streams in der AWS Cloud
- Verwenden AWS IoT MQTT-based Dateizustellung in Geräten
- Ein Beispiel für einen Anwendungsfall in FreeRTOS OTA

Was ist ein Stream?

In AWS IoT ist ein Stream eine öffentlich adressierbare Ressource, die eine Abstraktion für eine Liste von Dateien darstellt, die auf ein IoT-Gerät übertragen werden können. Ein typischer Stream enthält folgende Informationen:

- Ein Amazon-Ressourcenname (ARN), der einen Stream zu einem bestimmten Zeitpunkt eindeutig identifiziert. Dieser ARN hat das Muster `arn:`*`partition`*`:iot:`*`region`*`:`*`account-ID`*`:stream/`*`stream ID`*.
- Eine Stream-ID, die Ihren Stream identifiziert und die in () - oder SDK-Befehlen verwendet AWS Command Line Interface (und normalerweise erforderlich AWS CLI) wird.
- Eine Stream-Beschreibung, die eine Beschreibung der Stream-Ressource enthält.
- Eine Stream-Version, die eine bestimmte Version des Streams identifiziert. Da Stream-Daten unmittelbar geändert werden können, bevor Geräte mit der Datenübertragung beginnen, kann die Stream-Version von den Geräten verwendet werden, um eine Konsistenzprüfung zu erzwingen.

- Eine Liste von Dateien, die auf Geräte übertragen werden können. Für jede Datei in der Liste zeichnet der Stream eine Datei-ID, die Dateigröße und die Adressinformationen der Datei auf, die z. B. aus dem Namen des Amazon S3-Bucket, dem Objektschlüssel und der Objektversion bestehen.
- Eine AWS Identity and Access Management (IAM-) Rolle, die der AWS IoT MQTT-based Dateibereitstellung die Erlaubnis erteilt, im Datenspeicher gespeicherte Stream-Dateien zu lesen.

AWS IoT MQTT-based Die Dateibereitstellung bietet die folgenden Funktionen, damit Geräte Daten aus der AWS Cloud übertragen können:

- Datenübertragung über das MQTT-Protokoll.
- Support für die Formate JSON oder CBOR.
- Die Fähigkeit, einen Stream (`DescribeStream`-API) zu beschreiben, um eine Stream-Dateiliste, eine Stream-Version und verwandte Informationen abzurufen.
- Die Fähigkeit, Daten in kleinen Blöcken (`GetStream`-API) zu senden, sodass Geräte mit Hardwareeinschränkungen die Blöcke empfangen können.
- Support für eine dynamische Blockgröße pro Anforderung, um Geräte mit unterschiedlichen Speicherkapazitäten zu unterstützen.
- Optimierung für gleichzeitige Streaming-Anforderungen, wenn mehrere Geräte Datenblöcke aus derselben Stream-Datei anfordern.
- Amazon S3 als Datenspeicher für Stream-Dateien.
- Unterstützung für die Veröffentlichung von Datenübertragungsprotokollen von der AWS IoT MQTT-based Dateizustellung an CloudWatch.

Informationen zu Kontingenten für die MQTT-based Dateizustellung finden Sie unter AWS IoT Core Dienstkontingente im Allgemeine AWS-Referenz.

Verwaltung eines Streams in der AWS Cloud

AWS IoT bietet AWS SDK und AWS CLI Befehle, mit denen Sie einen Stream in der AWS Cloud verwalten können. Diese Befehle bieten Ihnen folgende Möglichkeiten:

- Erstellen Sie einen Stream. CLI/SDK
- Beschreiben Sie einen Stream, um seine Informationen abzurufen. CLI/SDK

- Listet Streams in Ihrem auf AWS-Konto. CLI/SDK
- Aktualisieren Sie die Dateiliste oder die Stream-Beschreibung in einem Stream. CLI/SDK
- Löschen Sie einen Stream. CLI/SDK

> **Note**
>
> Derzeit sind Streams in der AWS-Managementkonsole nicht sichtbar. Sie müssen das AWS CLI oder AWS SDK verwenden, um einen Stream in zu verwalten AWS IoT. Außerdem ist das Embedded C SDK das einzige SDK, das MQTT-based Dateiübertragungen unterstützt.

Bevor Sie die AWS IoT MQTT-based Dateiübertragung von Ihren Geräten aus verwenden, müssen Sie sicherstellen, dass die folgenden Bedingungen für Ihre Geräte erfüllt sind, wie in den nächsten Abschnitten beschrieben:

- Eine Richtlinie, welche die korrekten Berechtigungen widerspiegelt, die für die Übertragung von Daten über MQTT erforderlich sind.
- Ihr Gerät kann eine Verbindung zum AWS IoT Device Gateway herstellen.
- Eine Richtlinienanweisung, die besagt, dass Sie Ressourcen markieren können. Wenn `CreateStream` mit Markierungen aufgerufen wird, dann ist `iot:TagResource` erforderlich.

Bevor Sie die AWS IoT MQTT-based Dateizustellung von Ihren Geräten aus verwenden, müssen Sie die Schritte in den nächsten Abschnitten befolgen, um sicherzustellen, dass Ihre Geräte ordnungsgemäß autorisiert sind und eine Verbindung zum AWS IoT Device Gateway herstellen können.

Erteilen von Berechtigungen für Ihre Geräte

Sie können den Schritten unter Eine AWS IoT -Richtlinie erstellen folgen, um eine Geräterichtlinie zu erstellen oder eine bestehende Geräterichtlinie zu verwenden. Hängen Sie die Richtlinie an die Zertifikate an, die Ihren Geräten zugeordnet sind, und fügen Sie der Geräterichtlinie folgenden Berechtigungen hinzu.

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive",
                "iot:Publish"
            ],
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
${iot:Connection.Thing.ThingName}/streams/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/streams/*"
            ]
        }
    ]
}
```

Verbinden Sie Ihre Geräte mit AWS IoT

Geräte, die die AWS IoT MQTT-based Dateizustellung verwenden, müssen eine Verbindung herstellen AWS IoT. AWS IoT MQTT-based Die Dateizustellung ist AWS IoT in die AWS Cloud integriert, sodass Ihre Geräte eine direkte Verbindung zum Endpunkt der AWS IoT Datenebene herstellen sollten.

> **Note**
>
> Der Endpunkt der AWS IoT Datenebene ist spezifisch für die AWS-Konto UND-Region. Sie müssen den Endpunkt für die AWS-Konto und die Region verwenden, in der Ihre Geräte registriert sind AWS IoT.

Weitere Informationen finden Sie unter Connect to (Verbinden mit) AWS IoT Core.

TagResource Nutzung

Die API-Aktion `CreateStream` erstellt einen Stream für die Bereitstellung einer oder mehrerer Dateien in Fragmenten über MQTT.

Für einen erfolgreichen API-Aufruf `CreateStream` sind die folgenden Berechtigungen erforderlich:

- `iot:CreateStream`
- `iot:TagResource` (wenn `CreateStream` mit Markierungen vorliegt)

Die Richtlinie, die diese beiden Berechtigungen unterstützt, ist unten dargestellt:

```
{
    "Version":"2012-10-17",
    "Statement": {
        "Action": [
            "iot:CreateStream",
            "iot:TagResource"
        ],
        "Effect": "Allow",
        "Resource": "arn:aws:iot:us-east-1:123456789012:stream/streamId"
    }
}
```

Die Richtlinienanweisungsaktion `iot:TagResource` ist erforderlich, um sicherzustellen, dass ein Benutzer ohne die entsprechenden Berechtigungen keine Markierung für eine Ressource erstellen oder aktualisieren kann. Ohne die spezifische Richtlinienanweisung von `iot:TagResource` gibt der API-Aufruf `CreateStream` ein `AccessDeniedException` zurück, wenn die Anforderung Markierungen enthält.

Weitere Informationen erhalten Sie in folgenden Themen:

- CreateStream
- TagResource
- Markierung

Verwenden AWS IoT MQTT-based Dateizustellung in Geräten

Um den Datenübertragungsprozess einzuleiten, muss ein Gerät einen ersten Datensatz erhalten, der mindestens eine Stream-ID enthält. Sie können einen AWS IoT Jobs verwenden, um Datenübertragungsaufgaben für Ihre Geräte zu planen, indem Sie den ursprünglichen Datensatz in das Auftragsdokument aufnehmen. Wenn ein Gerät den ursprünglichen Datensatz empfängt, sollte es dann mit der Interaktion mit der AWS IoT MQTT-based Dateizustellung beginnen. Um Daten bei der AWS IoT MQTT-based Dateibereitstellung auszutauschen, sollte ein Gerät:

- Verwenden Sie das MQTT-Protokoll, um MQTT-based Themen zur Dateibereitstellung zu abonnieren.
- Senden Sie Anforderungen und warten Sie dann, bis Sie die Antworten mithilfe von MQTT-Nachrichten erhalten.

Sie können optional eine Stream-Datei-ID und eine Stream-Version in den ursprünglichen Datensatz aufnehmen. Das Senden einer Stream-Datei-ID an ein Gerät kann die Programmierung des Geräts vereinfachen firmware/software, da es nicht mehr erforderlich ist, eine `DescribeStream` Anfrage vom Gerät zu stellen, um diese ID abzurufen. Das Gerät kann die Stream-Version in einer `GetStream`-Anforderung angeben, um eine Konsistenzprüfung zu erzwingen, falls der Stream unerwartet aktualisiert wurde.

Wird verwendet DescribeStream , um Stream-Daten abzurufen

AWS IoT MQTT-based Die Dateibereitstellung stellt die `DescribeStream` API zum Senden von Stream-Daten an ein Gerät bereit. Die von dieser API zurückgegebenen Stream-Daten umfassen die Stream-ID, die Stream-Version, die Stream-Beschreibung und eine Liste von Stream-Dateien, von denen jede eine Datei-ID und die Dateigröße in Byte hat. Mit diesen Informationen kann ein Gerät beliebige Dateien auswählen, um den Datenübertragungsprozess einzuleiten.

> **Note**
>
> Sie müssen die `DescribeStream`-API nicht verwenden, wenn Ihr Gerät alle erforderlichen Stream-Datei-IDs im ursprünglichen Datensatz empfängt.

Führen Sie diese Schritte aus, um eine `DescribeStream`-Anforderung zu stellen.

1. Abonnieren Sie den „Akzeptiert“-Themenfilter `$aws/things/`*`ThingName`*`/streams/`*`StreamId`*`/description/json`.
2. Abonnieren Sie den „Abgelehnt“-Themenfilter `$aws/things/`*`ThingName`*`/streams/`*`StreamId`*`/rejected/json`.
3. Veröffentlichen Sie eine `DescribeStream`-Anforderung, indem Sie eine Nachricht an `$aws/things/`*`ThingName`*`/streams/`*`StreamId`*`/describe/json` senden.
4. Wenn die Anforderung akzeptiert wurde, erhält Ihr Gerät eine `DescribeStream`-Antwort auf den Themenfilter „Akzeptiert“.
5. Wenn die Anforderung abgelehnt wurde, erhält Ihr Gerät die Fehlerantwort im Themenfilter „Abgelehnt“.

> **Note**
>
> Wenn Sie in den angezeigten Themen und Themenfiltern `json` durch `cbor` ersetzen, empfängt Ihr Gerät Nachrichten im CBOR-Format, das kompakter als JSON ist.

DescribeStream Anfrage

Eine typische `DescribeStream`-JSON-Anforderung sieht wie im folgenden Beispiel aus.

```
{
    "c": "ec944cfb-1e3c-49ac-97de-9dc4aaad0039"
}
```

- (Optional) „`c`“ ist das Feld für das Client-Token.

  Das Client-Token darf nicht länger als 64 Bytes sein. Ein Client-Token, das länger als 64 Byte ist, verursacht eine Fehlerantwort und eine `InvalidRequest`-Fehlermeldung.

DescribeStream Antwort

Eine `DescribeStream`-Antwort in JSON ähnelt dem folgenden Beispiel.

```
{
    "c": "ec944cfb-1e3c-49ac-97de-9dc4aaad0039",
    "s": 1,
    "d": "This is the description of stream ABC.",
    "r": [
        {
            "f": 0,
            "z": 131072
        },
        {
            "f": 1,
            "z": 51200
        }
    ]
}
```

- „`c`“ ist das Feld für das Client-Token. Es wird zurückgegeben, wenn es in der `DescribeStream`-Anforderung angegeben wurde. Verwenden Sie das Client-Token, um die Antwort der Anforderung zuzuordnen.
- "`s`" ist die Stream-Version als Ganzzahl. Sie können diese Version verwenden, um eine Konsistenzprüfung Ihrer `GetStream` Anfragen durchzuführen.
- „`r`“ enthält eine Liste der Dateien im Stream.
  - „`f`“ ist die Stream-Datei-ID als Ganzzahl.
  - „`z`“ ist die Größe der Stream-Datei in Byte.
- „d“ enthält die Beschreibung des Streams.

Abrufen von Datenblöcken aus einer Stream-Datei

Sie können die `GetStream`-API verwenden, damit ein Gerät Stream-Dateien in kleinen Datenblöcken empfangen kann, sodass sie von Geräten verwendet werden kann, die Einschränkungen bei der Verarbeitung großer Blockgrößen haben. Um eine komplette Datendatei zu empfangen, muss ein Gerät möglicherweise mehrere Anforderungen und Antworten senden oder empfangen, bis alle Datenblöcke empfangen und verarbeitet sind.

GetStream Anfrage

Führen Sie diese Schritte aus, um eine `GetStream`-Anforderung zu stellen.

1. Abonnieren Sie den „Akzeptiert“-Themenfilter `$aws/things/`*`ThingName`*`/streams/`*`StreamId`*`/data/json`.
2. Abonnieren Sie den „Abgelehnt“-Themenfilter `$aws/things/`*`ThingName`*`/streams/`*`StreamId`*`/rejected/json`.
3. Veröffentlichen Sie eine `GetStream`-Anforderung zum Thema `$aws/things/`*`ThingName`*`/streams/`*`StreamId`*`/get/json`.
4. Wenn die Anforderung akzeptiert wurde, erhält Ihr Gerät eine oder mehrere `GetStream` Antworten auf den Themenfilter „Akzeptiert“. Jede Antwortnachricht enthält grundlegende Informationen und Nutzlastdaten für einen einzelnen Block.
5. Wiederholen Sie die Schritte 3 und 4, um alle Datenblöcke zu empfangen. Sie müssen diese Schritte wiederholen, wenn die angeforderte Datenmenge mehr als 128 KB beträgt. Sie müssen Ihr Gerät so programmieren, dass es mehrere `GetStream`-Anforderungen verwendet, damit alle angeforderten Daten empfangen werden.
6. Wenn die Anforderung abgelehnt wurde, erhält Ihr Gerät die Fehlerantwort im Themenfilter „Abgelehnt“.

> **Note**
>
> - Wenn Sie in den angezeigten Themen und Themenfiltern „json“ durch „cbor“ ersetzen, empfängt Ihr Gerät Nachrichten im CBOR-Format, das kompakter als das JSON-Format ist.
> - AWS IoT MQTT-based Die Dateizustellung begrenzt die Größe eines Blocks auf 128 KB. Wenn Sie eine Anforderung für einen Block stellen, der mehr als 128 KB groß ist, schlägt die Anforderung fehl.
> - Sie können eine Anforderung für mehrere Blöcke stellen, deren Gesamtgröße größer als 128 KB ist (wenn Sie beispielsweise eine Anforderung für 5 Blöcke mit jeweils 32 KB für insgesamt 160 KB an Daten stellen). In diesem Fall schlägt die Anfrage nicht fehl, aber Ihr Gerät muss mehrere Anfragen stellen, um alle angeforderten Daten zu erhalten. Der Service sendet zusätzliche Blöcke, wenn Ihr Gerät zusätzliche Anforderungen stellt. Wir empfehlen Ihnen, erst dann mit einer neuen Anforderungen fortzufahren, wenn die vorherige Antwort korrekt empfangen und verarbeitet wurde.

- Unabhängig von der Gesamtgröße der angeforderten Daten sollten Sie Ihr Gerät so programmieren, dass es Wiederholungsversuche einleitet, wenn Blöcke nicht oder nicht korrekt empfangen werden.

Eine typische `GetStream`-JSON-Anforderung sieht wie im folgenden Beispiel aus.

```
{
    "c": "1bb8aaa1-5c18-4d21-80c2-0b44fee10380",
    "s": 1,
    "f": 0,
    "l": 4096,
    "o": 2,
    "n": 100,
    "b": "..."
}
```

- [optional] „`c`“ ist das Feld für das Client-Token.

  Das Client-Token darf nicht länger als 64 Bytes sein. Ein Client-Token, das länger als 64 Byte ist, verursacht eine Fehlerantwort und eine `InvalidRequest`-Fehlermeldung.

- [optional] „`s`“ ist das Feld für die Stream-Version (eine Ganzzahl).

  MQTT-based Bei der Dateibereitstellung wird eine Konsistenzprüfung durchgeführt, die auf dieser angeforderten Version und der neuesten Stream-Version in der Cloud basiert. Wenn die von einem Gerät in einer `GetStream`-Anforderung gesendete Stream-Version nicht mit der neuesten Stream-Version in der Cloud übereinstimmt, sendet der Service eine Fehlerantwort und eine `VersionMismatch`-Fehlermeldung. In der Regel empfängt ein Gerät die erwartete (neueste) Stream-Version im ursprünglichen Datensatz oder in der Antwort auf `DescribeStream`.

- „`f`“ ist die Stream-Datei-ID (eine Ganzzahl im Bereich von 0 bis 255).

  Die Stream-Datei-ID ist erforderlich, wenn Sie einen Stream mithilfe des AWS CLI oder SDK erstellen oder aktualisieren. Wenn ein Gerät eine Stream-Datei mit einer ID anfordert, die nicht existiert, sendet der Service eine Fehlerantwort und eine `ResourceNotFound`-Fehlermeldung.

- „`l`“ ist die Datenblockgröße in Byte (eine Ganzzahl im Bereich von 256 bis 131.072).

  Informationen zu Erstellen Sie eine Bitmap für eine Anfrage GetStream für Anweisungen, wie die Bitmap-Felder verwendet werden, um festzulegen, welcher Teil der Stream-Datei in der

`GetStream`-Antwort zurückgegeben werden soll. Wenn ein Gerät eine Blockgröße angibt, die außerhalb des zulässigen Bereichs liegt, sendet der Service eine Fehlerantwort und eine `BlockSizeOutOfBounds`-Fehlermeldung.

- [optional] „o“ ist der Offset des Blocks in der Stream-Datei (eine Ganzzahl im Bereich von 0 bis 98.304).

  Informationen zu Erstellen Sie eine Bitmap für eine Anfrage GetStream für Anweisungen, wie die Bitmap-Felder verwendet werden, um festzulegen, welcher Teil der Stream-Datei in der `GetStream`-Antwort zurückgegeben werden soll. Der Höchstwert von 98.304 basiert auf einer Größenbeschränkung von 24 MB für Stream-Dateien und 256 Byte für die minimale Blockgröße. Wenn nichts angegeben ist, beträgt der Standardwert 0.

- [optional] „n“ ist die Anzahl der angeforderten Blöcke (eine Ganzzahl im Bereich von 0 bis 98.304).

  Das Feld „n“ gibt entweder (1) die Anzahl der angeforderten Blöcke oder (2), wenn das Bitmap-Feld („b“) verwendet wird, eine Obergrenze für die Anzahl der Blöcke an, die von der Bitmap-Anforderung zurückgegeben werden. Diese zweite Verwendung ist optional. Wenn nicht definiert, ist es standardmäßig *`DataBlockSize`* 131072/.

- [optional] „b“ ist eine Bitmap, welche die angeforderten Blöcke darstellt.

  Mithilfe einer Bitmap kann Ihr Gerät nicht aufeinanderfolgende Blöcke anfordern, was die Handhabung von Wiederholungsversuchen nach einem Fehler komfortabler macht. Informationen zu Erstellen Sie eine Bitmap für eine Anfrage GetStream für Anweisungen, wie die Bitmap-Felder verwendet werden, um festzulegen, welcher Teil der Stream-Datei in der `GetStream`-Antwort zurückgegeben werden soll. Konvertieren Sie für dieses Feld die Bitmap in eine Zeichenfolge, die den Bitmap-Wert in hexadezimaler Schreibweise darstellt. Die Bitmap muss kleiner als 12.288 Byte sein.

> **Important**
>
> Entweder „n“ oder „b“ sollte angegeben werden. Wenn keines von beiden festgelegt wurde, ist die `GetStream` Anforderung möglicherweise nicht gültig, wenn die Dateigröße weniger als 131.072 Byte (128 KB) beträgt.

GetStream Antwort

Eine `GetStream`-Antwort in JSON sieht für jeden angeforderten Datenblock wie in diesem Beispiel aus.

```
{
    "c": "1bb8aaa1-5c18-4d21-80c2-0b44fee10380",
    "f": 0,
    "l": 4096,
    "i": 2,
    "p": "..."
}
```

- „`c`“ ist das Feld für das Client-Token. Es wird zurückgegeben, wenn es in der `GetStream`-Anforderung angegeben wurde. Verwenden Sie das Client-Token, um die Antwort der Anforderung zuzuordnen.
- „`f`“ ist die ID der Stream-Datei, zu der die aktuelle Datenblock-Nutzlast gehört.
- „`l`“ ist die Größe der Datenblock-Nutzlast in Byte.
- „`i`“ ist die ID des Datenblocks, der in der Nutzlast enthalten ist. Datenblöcke werden beginnend mit 0 durchnummeriert.
- „`p`“ enthält die Nutzlast des Datenblocks. Dieses Feld ist eine Zeichenfolge, die den Wert des Datenblocks in Base64-Kodierung darstellt.

Erstellen Sie eine Bitmap für eine Anfrage GetStream

Sie können das Bitmap-Feld (b) in einer `GetStream`-Anforderung verwenden, um nicht aufeinanderfolgende Blöcke aus einer Stream-Datei abzurufen. Dies hilft Geräten mit begrenzter RAM-Kapazität, Probleme mit der Netzwerkbereitstellung zu lösen. Ein Gerät kann nur die Blöcke anfordern, die nicht oder nicht korrekt empfangen wurden. Die Bitmap bestimmt, welche Blöcke der Stream-Datei zurückgegeben werden. Für jedes Bit, das in der Bitmap auf 1 festgelegt ist, wird ein entsprechender Block der Stream-Datei zurückgegeben.

Hier sehen Sie ein Beispiel für die Angabe einer Bitmap und ihrer unterstützenden Felder in einer `GetStream`-Anforderung. Sie möchten beispielsweise eine Stream-Datei in Blöcken von 256 Byte (der Blockgröße) empfangen. Stellen Sie sich vor, dass jeder Block mit 256 Byte eine Zahl hat, die seine Position in der Datei angibt, beginnend bei 0. Block 0 ist also der erste Block mit 256 Byte in der Datei, Block 1 der zweite usw. Sie möchten die Blöcke 20, 21, 24 und 43 aus der Datei anfordern.

Block-Offset

Da der erste Block die Nummer 20 hat, geben Sie als Offset (Feld o) 20 an, um Platz in der Bitmap zu sparen.

Anzahl der Blöcke

Um sicherzustellen, dass Ihr Gerät nicht mehr Blöcke empfängt, als es mit begrenzten Speicherressourcen verarbeiten kann, können Sie die maximale Anzahl von Blöcken angeben, die in jeder von der MQTT-based Dateizustellung gesendeten Nachricht zurückgegeben werden sollen. Beachten Sie, dass dieser Wert ignoriert wird, wenn die Bitmap selbst weniger als diese Anzahl von Blöcken angibt oder wenn dadurch die Gesamtgröße der per MQTT-based Dateizustellung gesendeten Antwortnachrichten das Servicelimit von 128 KB pro `GetStream` Anfrage überschreiten würde.

Block-Bitmap

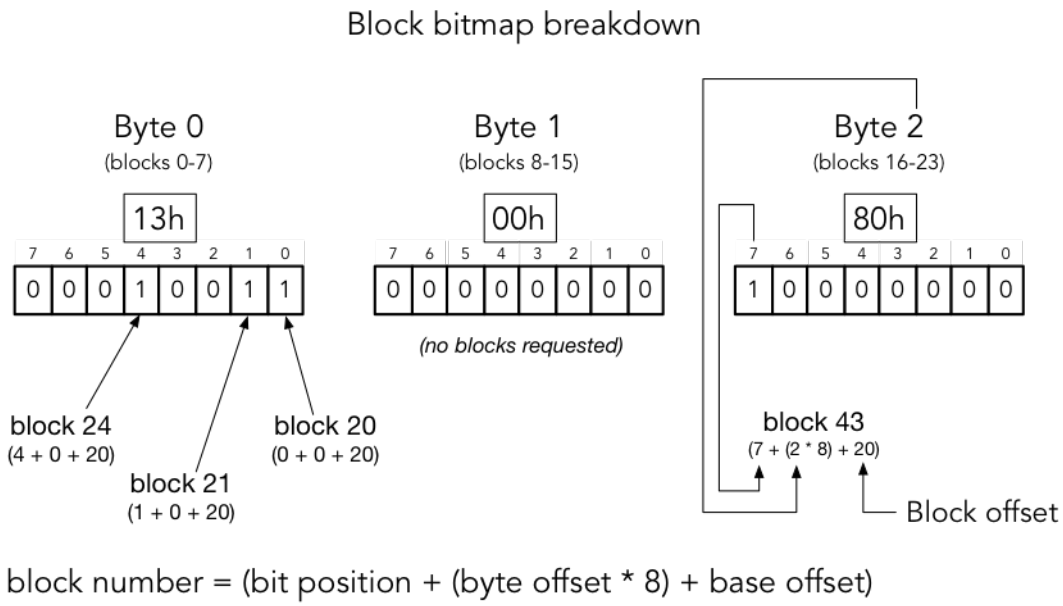
Die Bitmap selbst ist ein Array von Bytes ohne Vorzeichen, die in hexadezimaler Schreibweise ausgedrückt und in der `GetStream`-Anforderung als Zeichenkettendarstellung der Zahl enthalten sind. Aber um diese Zeichenfolge zu konstruieren, stellen wir uns die Bitmap zunächst als eine lange Folge von Bits (eine Binärzahl) vor. Wenn ein Bit in dieser Sequenz auf 1 festgelegt ist, wird der entsprechende Block aus der Stream-Datei an das Gerät zurückgesendet. In unserem Beispiel wollen wir die Blöcke 20, 21, 24 und 43 empfangen, also müssen wir die Bits 20, 21, 24 und 43 in unserer Bitmap setzen. Wir können den Block-Offset verwenden, um Platz zu sparen. Nachdem wir also den Offset von jeder Blocknummer subtrahiert haben, möchten wir die Bits 0, 1, 4 und 23 wie im folgenden Beispiel setzen.

```
1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
```

Wenn wir jeweils ein Byte (8 Bit) nehmen, wird dies üblicherweise wie folgt geschrieben: „0b00010011“, „0b00000000“ und „0b10000000“. Bit 0 erscheint in unserer binären Darstellung am Ende des ersten Bytes und Bit 23 am Anfang des letzten Bytes. Das kann verwirrend sein, es sei denn, Sie kennen die Konventionen. Das erste Byte enthält die Bits 7-0 (in dieser Reihenfolge), das zweite Byte enthält die Bits 15-8, das dritte Byte enthält die Bits 23-16 und so weiter. In hexadezimaler Schreibweise wird dies in „0x130080“ umgewandelt.

> **Tip**
>
> Sie können die standardmäßige Binärschreibweise in die hexadezimale Schreibweise konvertieren. Nehmen Sie jeweils vier Binärziffern und konvertieren Sie diese in ihre

hexadezimale Entsprechung. Zum Beispiel wird aus „0001“ „1“, aus „0011“ wird „3“ und so weiter.



Zusammengenommen sieht das JSON für unsere `GetStream`-Anforderung wie folgt aus.

```
{
    "c" : "1",
    "s" : 1,
    "l" : 256,
    "f" : 1,
    "o" : 20,
    "n" : 32,
    "b" : "130080"
}
```

- „c“ ist das Feld für das Client-Token.
- „s“ ist die erwartete Stream-Version.
- „l“ ist die Größe der Datenblock-Nutzlast in Byte.
- „f“ ist die ID des Quelldatei-Indexes.
- „o“ ist der Block-Offset.
- „n“ ist die Anzahl der Blöcke.

- „b“ ist die fehlende blockId-Bitmap, beginnend mit dem Offset. Dieser Wert muss based64-codiert sein.

Behandlung von Fehlern von AWS IoT MQTT-based Lieferung von Dateien

Eine Fehlerantwort, die sowohl für APIs als auch `DescribeStream` für `GetStream`-APIs an ein Gerät gesendet wird, enthält ein Client-Token, einen Fehlercode und eine Fehlermeldung. Eine typische Fehlerantwort sieht wie im folgenden Beispiel aus.

```
{
    "o": "BlockSizeOutOfBounds",
    "m": "The block size is out of bounds",
    "c": "1bb8aaa1-5c18-4d21-80c2-0b44fee10380"
}
```

- „o“ ist der Fehlercode, der den Grund anzeigt, warum ein Fehler aufgetreten ist. Weitere Informationen finden Sie in den Fehlercodes weiter unten in diesem Abschnitt.
- „m“ ist die Fehlermeldung, die Einzelheiten des Fehlers anzeigt.
- „c“ ist das Feld für das Client-Token. Dies kann zurückgegeben werden, wenn es in der `DescribeStream`-Anforderung angegeben wurde. Sie können das Client-Token verwenden, um die Antwort der Anforderung zuzuordnen.

  Das Client-Token-Feld ist nicht immer in einer Fehlerantwort enthalten. Wenn das in der Anforderung angegebene Client-Token nicht gültig oder falsch formatiert ist, wird es in der Fehlerantwort nicht zurückgegeben.

> **Note**
>
> Aus Gründen der Abwärtskompatibilität können Felder in der Fehlerantwort nicht abgekürzt sein. Beispielsweise kann der Fehlercode entweder durch die Felder „Code“ oder „o“ und das Client-Token-Feld entweder durch die Felder „clientToken“ oder „c“ gekennzeichnet werden. Wir empfehlen Ihnen, die oben abgebildete Abkürzungsform zu verwenden.

InvalidTopic

Das MQTT-Thema der Stream-Nachricht ist ungültig.

InvalidJson

Die Stream-Anforderung ist kein gültiges JSON-Dokument.

InvalidCbor

Die Stream--Anforderung ist kein gültiges CBOR-Dokument.

InvalidRequest

Die Anforderung wird im Allgemeinen als falsch formatiert identifiziert. Weitere Informationen finden Sie in der Fehlermeldung.

Nicht autorisiert

Die Anforderung ist nicht berechtigt, auf die Stream-Daten-Dateien auf dem Speichermedium wie Amazon S3 zuzugreifen. Weitere Informationen finden Sie in der Fehlermeldung.

BlockSizeOutOfBounds

Die Blockgröße liegt außerhalb der Grenzen. Weitere Informationen finden Sie im Abschnitt "MQTT-based Dateizustellung" in AWS IoT Core Servicequotas.

OffsetOutOfBounds

Der Offset liegt außerhalb der Grenzen. Weitere Informationen finden Sie im Abschnitt „MQTT-based Dateizustellung“ in AWS IoT Core Servicequotas.

BlockCountLimitExceeded

Die Anzahl der Anforderungsblöcke ist außerhalb des zulässigen Bereichs. Weitere Informationen finden Sie im Abschnitt „MQTT-based Dateizustellung“ in AWS IoT Core Servicequotas.

BlockBitmapLimitExceeded

Die Größe der Anforderungsbitmap liegt außerhalb des gültigen Bereichs. Weitere Informationen finden Sie im Abschnitt „MQTT-based Dateizustellung“ in AWS IoT Core Servicequotas.

ResourceNotFound

Der angeforderte Stream, die Dateien, Dateiversionen oder Blöcke wurden nicht gefunden. Weitere Details und Informationen finden Sie in der Fehlermeldung.

VersionMismatch

Die Stream-Version in der Anfrage stimmt nicht mit der Stream-Version in der MQTT-based Dateizustellungsfunktion überein. Dies weist darauf hin, dass die Stream-Daten seit dem ersten Empfang der Stream-Version durch das Gerät geändert wurden.

ETagMismatch

Das S3-ETag im Stream stimmt nicht mit dem ETag der neuesten S3-Objektversion überein.

InternalError

Bei der MQTT-based Dateiübermittlung ist ein interner Fehler aufgetreten.

Ein Beispiel für einen Anwendungsfall in FreeRTOS OTA

Der FreeRTOS OTA (Over-the-Air) -Agent verwendet die AWS IoT MQTT-based Dateizustellung, um FreeRTOS-Firmware-Images auf FreeRTOS-Geräte zu übertragen. Um den ursprünglichen Datensatz an ein Gerät zu senden, verwendet er den AWS IoT Job-Service, um einen OTA-Aktualisierungsauftrag für FreeRTOS-Geräte zu planen.

Eine Referenzimplementierung eines MQTT-based Dateizustellungsclients finden Sie in den FreeRTOS OTA-Agent-Codes in der FreeRTOS-Dokumentation.

Device Advisor

Device Advisor ist eine cloudbasierte, vollständig verwaltete Testfunktion zur Validierung von IoT-Geräten während der Entwicklung von Gerätesoftware. Device Advisor bietet vorgefertigte Tests, mit denen Sie IoT-Geräte auf ihre zuverlässige und sichere Konnektivität überprüfen können AWS IoT Core, bevor Sie Geräte in der Produktion einsetzen. Die vorgefertigten Tests von Device Advisor helfen Ihnen dabei, Ihre Gerätesoftware anhand von Best Practices für die Verwendung von TLS-, MQTT-, Device Shadow- und IoT-Aufträgen zu validieren. Sie können auch signierte Qualifikationsberichte herunterladen, um sie an das AWS -Partnernetzwerk zu senden, damit Ihr Gerät für den AWS -Partnergerätekatalog qualifiziert wird, ohne dass Sie Ihr Gerät einschicken und warten müssen, bis es getestet wird.

> **Note**
>
> Device Advisor wird in den Regionen us-east-1, us-west-2, ap-northeast-1 und eu-west-1 unterstützt.
> Device Advisor unterstützt Geräte und Clients, die die Protokolle MQTT und MQTT over WebSocket Secure (WSS) verwenden, um Nachrichten zu veröffentlichen und zu abonnieren.
> Alle Protokolle unterstützen IPv4 und IPv6.
> Device Advisor unterstützt RSA-Serverzertifikate.

Jedes Gerät, mit dem eine Verbindung hergestellt AWS IoT Core werden kann, kann Device Advisor nutzen. Sie können von der AWS IoT Konsole aus oder mithilfe des SDK AWS CLI oder auf Device Advisor zugreifen. Wenn Sie bereit sind, Ihr Gerät zu testen, registrieren Sie es beim Device Advisor-Endpunkt AWS IoT Core und konfigurieren Sie die Gerätesoftware. Wählen Sie dann die vorgefertigten Tests aus, konfigurieren Sie sie und führen Sie die Tests auf Ihrem Gerät aus. Sie erhalten dann die Testergebnisse zusammen mit detaillierten Protokollen oder einem Qualifizierungsbericht.

Device Advisor ist ein Testendpunkt in der AWS Cloud. Sie können Ihre Geräte testen, indem Sie sie so konfigurieren, dass sie eine Verbindung zu dem vom Device Advisor bereitgestellten Testendpunkt herstellen. Nachdem ein Gerät für die Verbindung mit dem Testendpunkt konfiguriert wurde, können Sie die Device Advisor-Konsole aufrufen oder das AWS SDK verwenden, um die Tests auszuwählen, die Sie auf Ihren Geräten ausführen möchten. Device Advisor verwaltet dann den gesamten Lebenszyklus eines Tests, einschließlich der Bereitstellung von Ressourcen, der Planung des Testprozesses, der Verwaltung des Zustandsautomats, der Aufzeichnung des

Geräteverhaltens, der Protokollierung der Ergebnisse und der Bereitstellung der Endergebnisse in Form eines Testberichts.

TLS-Protokolle

Das TLS-Protokoll (Transport Layer Security) wird für die Verschlüsselung vertraulicher Daten über unsichere Netzwerke wie das Internet verwendet. Das TLS-Protokoll ist der Nachfolger des SSL-Protokolls (Secure Sockets Layer).

Device Advisor unterstützt die folgenden TLS-Protokolle:

- TLS 1.3 (empfohlen)
- TLS 1.2

Protokolle, Port-Zuweisungen und Authentifizierung

Das Gerätekommunikationsprotokoll wird von einem Gerät oder einem Client verwendet, um über einen Geräteendpunkt eine Verbindung zum Message Broker herzustellen. In der folgenden Tabelle sind die Protokolle aufgeführt, die von den Device-Advisor-Endpunkten unterstützt werden, sowie die verwendeten Authentifizierungsmethoden und Ports.

Protokolle, Authentifizierung und Port-Zuweisungen

Protocol (Protokoll)	Unterstützte Operationen	Authentifizierung	Port	ALPN-Protokollname
MQTT über WebSocket	Veröffentlichen, Abonnieren	Signaturversion 4	443	N/A
MQTT	Veröffentlichen, Abonnieren	X.509 Client-Zertifikat	8883	`x-amzn-mqtt-ca`
MQTT	Veröffentlichen, Abonnieren	X.509 Client-Zertifikat	443	N/A

Dieses Kapitel enthält die folgenden Abschnitte:

- Einrichtung
- Erste Schritte mit Device Advisor in der Konsole

- Device-Advisor-Workflow
- Ausführlicher Konsolen-Workflow von Device Advisor
- Konsolen-Workflow für Tests mit langer Dauer
- Device Advisor VPC-Endpunkte (AWS PrivateLink)
- Device-Advisor-Testfälle

Einrichtung

Führen Sie die folgenden Schritte aus, bevor Sie Device Advisor zum ersten Mal verwenden:

Erstellen eines IoT-Dings

Erstellen Sie zunächst ein IoT-Ding und fügen Sie diesem Ding ein Zertifikat hinzu. Ein Tutorial zum Erstellen von Dingen finden Sie unter Create a thing object (Erstellen eines Dingobjekts).

Erstellen einer IAM-Rolle, die Sie als Ihre Geräterolle verwenden möchten

> **Note**
>
> Sie können die Geräterolle mit der Device-Advisor-Konsole schnell erstellen. Informationen zum Einrichten Ihrer Geräterolle mit der Device-Advisor-Konsole finden Sie unter Getting started with the Device Advisor in the console.

1. Gehen Sie zur AWS Identity and Access Management Konsole und melden Sie sich bei dem an, das AWS-Konto Sie für Device Advisor-Tests verwenden.
2. Wählen Sie im linken Navigationsbereich die Option Richtlinien aus.
3. Wählen Sie Richtlinie erstellen aus.
4. Gehen Sie unter Richtlinie erstellen wie folgt vor:

    a. Wählen Sie unter Service die Option IoT aus.

    b. Führen Sie unter Aktionen eine der folgenden Aktionen aus.

    - (Empfohlen) Wählen Sie Aktionen auf der Grundlage der Richtlinie aus, die dem IoT-Ding oder dem IoT-Zertifikat zugeordnet ist, das Sie im vorherigen Abschnitt erstellt haben.
    - Suchen Sie im Feld Filteraktion nach den folgenden Aktionen und wählen Sie sie aus:

- `Connect`
- `Publish`
- `Subscribe`
- `Receive`
- `RetainPublish`

c. Schränken Sie unter Ressourcen die Ressourcen für Client und Thema ein. Die Einschränkung dieser Ressourcen ist eine bewährte Sicherheitsmethode. Gehen Sie wie folgt vor, um Ressourcen einzuschränken:

i. Wählen Sie Specify client resource ARN for the Connect action (Client-Ressourcen-ARN für die Connect-Aktion angeben) aus.

ii. Wählen Sie ARN hinzufügen aus und führen Sie dann einen der folgenden Schritte aus:

> **Note**
>
> Die clientId ist die MQTT-Client-ID, die Ihr Gerät für die Interaktion mit Device Advisor verwendet.

- Geben Sie die Region, die accountID und die clientID im visuellen ARN-Editor an.
- Geben Sie manuell die Amazon-Ressourcennamen (ARNs) der IoT-Themen ein, mit denen Sie Ihre Testfälle ausführen möchten.

iii. Wählen Sie Hinzufügen aus.

iv. Wählen Specify topic resource ARN for the Receive and one more action (ARN der Themenressource für den Receive und eine weitere Aktion angeben) aus.

v. Wählen Sie ARN hinzufügen aus und führen Sie dann einen der folgenden Schritte aus:

> **Note**
>
> Der Themenname ist das MQTT-Thema, zu dem Ihr Gerät Nachrichten veröffentlicht.

- Geben Sie die Region, die accountID und den Themennamen im visuellen ARN-Editor an.

- Geben Sie manuell die ARNs der IoT-Themen ein, mit denen Sie Ihre Testfälle ausführen möchten.

vi. Wählen Sie Hinzufügen aus.

vii. Wählen Sie Specify topicFilter resource ARN for the Subscribe action (ARN der topicFilter-Ressource für die Subscribe-Aktion angeben) aus.

viii. Wählen Sie ARN hinzufügen aus und führen Sie dann einen der folgenden Schritte aus:

> **Note**
>
> Der Themenname ist das MQTT-Thema, das Ihr Gerät abonniert.

- Geben Sie die Region, die accountID und den Themennamen im visuellen ARN-Editor an.
- Geben Sie manuell die ARNs der IoT-Themen ein, mit denen Sie Ihre Testfälle ausführen möchten.

ix. Wählen Sie Hinzufügen aus.

5. Wählen Sie Next: Tags (Weiter: Tags) aus.
6. Klicken Sie auf Weiter: Prüfen.
7. Geben Sie unter Richtlinie überprüfen einen Namen für Ihre Richtlinie ein.
8. Wählen Sie Richtlinie erstellen aus.
9. Wählen Sie im linken Navigationsbereich Rollen aus.
10. Wählen Sie Rolle erstellen aus.
11. Wählen Sie unter Vertrauenswürdige Entitäten auswählen die Option Benutzerdefinierte Vertrauensrichtlinie aus.
12. Geben Sie die folgende Vertrauensrichtlinie in das Feld Benutzerdefinierte Vertrauensrichtlinie ein. Verwenden Sie die globalen Konditionsschlüssel `aws:SourceArn` und `aws:SourceAccount` für die Richtlinie, um vor dem Confused-Deputy-Problem zu schützen.

> **Important**
>
> Ihr `aws:SourceArn` muss mit dem `format: arn:aws:iotdeviceadvisor:region:account-id:*.` konform sein. Stellen Sie sicher, dass *`region`* Ihrer AWS IoT -Region entspricht und dass *`account-id`*

Ihrer Kundenkonto-ID entspricht. Weitere Informationen finden Sie unter Cross-service Confused Deputy Prevention.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAwsIoTCoreDeviceAdvisor",
            "Effect": "Allow",
            "Principal": {
                "Service": "iotdeviceadvisor.amazonaws.com"
        },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
            },
                "ArnLike": {
                    "aws:SourceArn":
 "arn:aws:iotdeviceadvisor:*:123456789012:suitedefinition/*"
            }
        }
        }
    ]
}
```

13. Wählen Sie Weiter aus.
14. Wählen Sie die Richtlinie aus, den Sie in Schritt 4 erstellt haben.
15. (Optional) Wählen Sie unter Berechtigungsgrenze festlegen die Option Use a permissions boundary to control the maximum role permissions (Eine Berechtigungsgrenze verwenden, um die maximalen Rollenberechtigungen zu kontrollieren) aus und dann die Richtlinie, die Sie erstellt haben.
16. Wählen Sie Weiter aus.
17. Geben Sie einen Rollennamen und eine Rollenbeschreibung an.
18. Wählen Sie Rolle erstellen aus.

Erstellen einer individuell verwalteten Richtlinie für einen IAM-Benutzer zur Verwendung von Device Advisor

1. Navigieren Sie zur IAM-Konsole unter https://console.aws.amazon.com/iam/. Geben Sie bei Aufforderung Ihre AWS -Anmeldeinformationen ein.
2. Wählen Sie im linken Navigationsbereich Richtlinien aus.
3. Wählen Sie Richtlinie erstellen und anschließend die Registerkarte JSON aus.
4. Fügen Sie die erforderlichen Berechtigungen hinzu, um Device Advisor zu verwenden. Das Richtliniendokument finden Sie im Thema Bewährte Methoden für die Sicherheit.
5. Wählen Sie Review policy (Richtlinie überprüfen) aus.
6. Geben Sie Name (Name) und Description (Beschreibung) ein.
7. Wählen Sie Richtlinie erstellen aus.

Erstellen eines IAM-Benutzers für die Verwendung von Device Advisor

> **Note**
>
> Wir empfehlen, einen IAM-Benutzer für die Ausführung von Device-Advisor-Tests zu erstellen. Das Ausführen von Device-Advisor-Tests als Administratorbenutzer kann Sicherheitsrisiken bergen und wird nicht empfohlen.

1. Navigieren Sie zur IAM-Konsole unter https://console.aws.amazon.com/iam/ Wenn Sie dazu aufgefordert werden, geben Sie Ihre AWS Anmeldeinformationen ein, um sich anzumelden.
2. Wählen Sie im linken Navigationsbereich Benutzer aus.
3. Wählen Sie Benutzer hinzufügen.
4. Geben Sie einen Benutzernamen ein.
5. Benutzer benötigen programmgesteuerten Zugriff, wenn sie AWS außerhalb von interagieren möchten. AWS-Managementkonsole Die Art und Weise, wie programmgesteuerter Zugriff gewährt wird, hängt vom Benutzertyp ab, der zugreift. AWS

   Um Benutzern programmgesteuerten Zugriff zu gewähren, wählen Sie eine der folgenden Optionen.

Welcher Benutzer benötigt programmgesteuerten Zugriff?	Bis	Von
IAM	(Empfohlen) Verwenden Sie Konsolenanmeldedaten als temporäre Anmeldeinformationen, um programmatische Anforderungen an die AWS SDKs AWS CLI oder APIs zu signieren. AWS	Befolgen Sie die Anweisungen für die Schnittstelle, die Sie verwenden möchten. • Informationen dazu finden Sie im AWS Command Line Interface Benutzerhandbuch unter Anmeldung für AWS lokale Entwicklung. AWS CLI • Informationen zu AWS SDKs finden Sie unter Anmeldung für AWS lokale Entwicklung im Referenzhandbuch für AWS SDKs und Tools.

Welcher Benutzer benötigt programmgesteuerten Zugriff?	Bis	Von
Mitarbeiteridentität (Benutzer, die in IAM Identity Center verwaltet werden)	Verwenden Sie temporäre Anmeldeinformationen, um programmatische Anfragen an die AWS SDKs oder AWS CLI APIs zu signieren. AWS	Befolgen Sie die Anweisungen für die Schnittstelle, die Sie verwenden möchten. • Informationen dazu finden Sie AWS IAM Identity Center im AWS Command Line Interface Benutzerhandbuch unter Configuring AWS CLI to use. AWS CLI • Informationen zu AWS SDKs, Tools und AWS APIs finden Sie unter IAM Identity Center-Authentifizierung im Referenzhandbuch für AWS SDKs und Tools.
IAM	Verwenden Sie temporäre Anmeldeinformationen, um programmatische Anfragen an die AWS SDKs oder APIs AWS CLI zu signieren. AWS	Folgen Sie den Anweisungen unter Verwenden temporärer Anmeldeinformationen mit AWS Ressourcen im IAM-Benutzerhandbuch.

Welcher Benutzer benötigt programmgesteuerten Zugriff?	Bis	Von
IAM	(Nicht empfohlen) Verwenden Sie langfristige Anmeldeinformationen, um programmatische Anfragen an die AWS CLI, AWS SDKs oder APIs zu signieren. AWS	Befolgen Sie die Anweisungen für die Schnittstelle, die Sie verwenden möchten. • Informationen dazu finden Sie im AWS CLI Benutzerhandbuch unter Authentifizierung mit IAM-Benutzeranmeldeinformationen. AWS Command Line Interface • Informationen zu AWS SDKs und Tools finden Sie unter Authentifizieren mit langfristigen Anmeldeinformationen im Referenzhandbuch für AWS SDKs und Tools. • Informationen zu AWS APIs finden Sie unter Verwalten von Zugriffsschlüsseln für IAM-Benutzer im IAM-Benutzerhandbuch.

6. Wählen Sie Weiter: Berechtigungen aus.
7. Um Zugriff zu gewähren, fügen Sie Ihren Benutzern, Gruppen oder Rollen Berechtigungen hinzu:

   • Benutzer und Gruppen in: AWS IAM Identity Center

     Erstellen Sie einen Berechtigungssatz. Befolgen Sie die Anweisungen unter Erstellen eines Berechtigungssatzes im AWS IAM Identity Center -Benutzerhandbuch.

   • Benutzer, die in IAM über einen Identitätsanbieter verwaltet werden:

Erstellen Sie eine Rolle für den Identitätsverbund. Befolgen Sie die Anleitung unter Eine Rolle für einen externen Identitätsanbieter (Verbund) erstellen im IAM-Benutzerhandbuch.

- IAM-Benutzer:
  - Erstellen Sie eine Rolle, die Ihr Benutzer annehmen kann. Befolgen Sie die Anleitung unter Eine Rolle für einen IAM-Benutzer erstellen im IAM-Benutzerhandbuch.
  - (Nicht empfohlen) Weisen Sie einem Benutzer eine Richtlinie direkt zu oder fügen Sie einen Benutzer zu einer Benutzergruppe hinzu. Befolgen Sie die Anweisungen unter Hinzufügen von Berechtigungen zu einem Benutzer (Konsole) im IAM-Benutzerhandbuch.

8. Geben Sie in das Suchfeld den Namen der individuell verwalteten Richtlinie ein, die Sie erstellt haben. Aktivieren Sie dann das Kontrollkästchen für Richtliniennamen.
9. Wählen Sie Next: Tags (Weiter: Tags) aus.
10. Klicken Sie auf Next: Review (Weiter: Prüfen).
11. Wählen Sie Create user aus.
12. Klicken Sie auf Schließen.

Device Advisor benötigt in Ihrem Namen Zugriff auf Ihre AWS Ressourcen (Dinge, Zertifikate und Endpunkte). Ihr IAM-Benutzer muss über die nötigen Berechtigungen verfügen. Device Advisor veröffentlicht auch Protokolle bei Amazon, CloudWatch wenn Sie Ihrem IAM-Benutzer die erforderlichen Berechtigungsrichtlinien beifügen.

Konfigurieren Ihres Geräts

Device Advisor nutzt die TLS-Erweiterung für die Servernamenindikation (SNI), um TLS-Konfigurationen anzuwenden. Geräte müssen diese Erweiterung verwenden, wenn sie eine Verbindung herstellen und einen Servernamen übergeben, der mit dem Device-Advisor-Testendpunkt identisch ist.

Device Advisor ermöglicht die TLS-Verbindung, wenn sich ein Test im `Running`-Status befindet. Es verweigert die TLS-Verbindung vor und nach jedem Testlauf. Aus diesem Grund empfehlen wir, den Mechanismus zur Wiederholung der Geräteverbindung zu verwenden, um ein vollautomatisches Testerlebnis mit Device Advisor zu gewährleisten. Sie können Testsuiten ausführen, die mehr als einen Testfall enthalten, z. B. TLS Connect, MQTT Connect und MQTT Publish. Wenn Sie mehrere Testfälle ausführen, empfehlen wir, dass Ihr Gerät versucht, alle fünf Sekunden eine Verbindung zu unserem Testendpunkt herzustellen. Sie können dann die Ausführung mehrerer Testfälle nacheinander automatisieren.

> **Note**
>
> Um Ihre Gerätesoftware für das Testen vorzubereiten, empfehlen wir Ihnen, ein SDK zu verwenden, das eine Verbindung zu AWS IoT Core herstellen kann. Anschließend sollten Sie das SDK mit dem Device-Advisor-Testendpunkt aktualisieren, der für Ihr AWS-Konto bereitgestellt wurde.

Device Advisor unterstützt zwei Arten von Endpunkten: Account-level und Device-level Endpunkte. Wählen Sie den Endpunkt aus, der Ihrem Anwendungsfall am besten entspricht. Verwenden Sie einen Endpunkt, um mehrere Testsuiten für verschiedene Geräte gleichzeitig auszuführen. Device-level

Führen Sie den folgenden Befehl aus, um den Device-level Endpunkt abzurufen:

Für MQTT-Kunden, die X.509 Client-Zertifikate verwenden:

```
aws iotdeviceadvisor get-endpoint --thing-arn your-thing-arn
```

oder

```
aws iotdeviceadvisor get-endpoint --certificate-arn your-certificate-arn
```

Für WebSocket MQTT-over-Kunden, die Signature Version 4 verwenden:

```
aws iotdeviceadvisor get-endpoint --device-role-arn your-device-role-arn --
authentication-method SignatureVersion4
```

Um jeweils eine Testsuite auszuführen, wählen Sie einen Account-level Endpunkt aus. Führen Sie den folgenden Befehl aus, um den Account-level Endpunkt abzurufen:

```
aws iotdeviceadvisor get-endpoint
```

Erste Schritte mit Device Advisor in der Konsole

Dieses Tutorial hilft Ihnen bei den ersten Schritten AWS IoT Core Device Advisor auf der Konsole. Device Advisor bietet Features wie erforderliche Tests und signierte Qualifikationsberichte.

Sie können diese Tests und Berichte verwenden, um Geräte zu qualifizieren und im -Partner-Gerätekatalog aufzulisten, wie im AWS IoT Core -Qualifizierungsprogramm beschrieben.

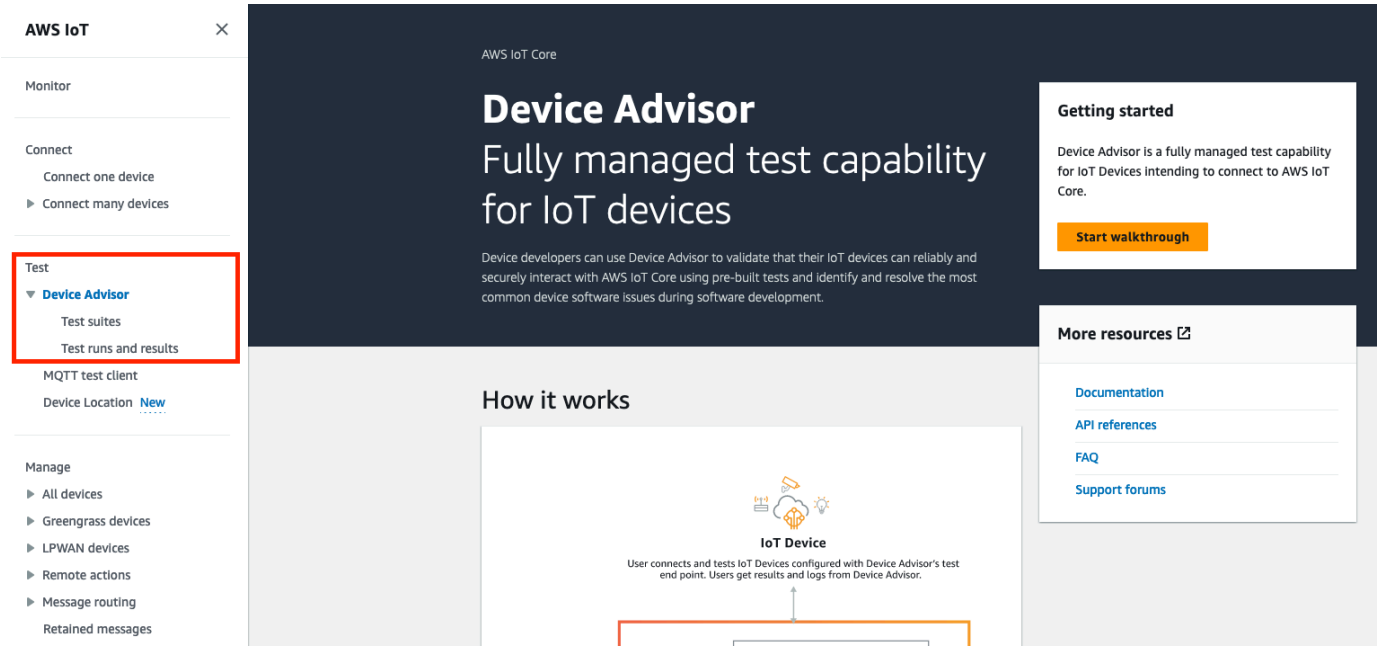
Weitere Informationen zum Verwenden von Device Advisor finden Sie unter Device-Advisor-Workflow und Ausführlicher Konsolen-Workflow von Device Advisor.

Führen Sie die Schritte in Einrichtung aus, um dieses Tutorial abzuschließen.

> **Note**
>
> Device Advisor wird in den folgenden Bereichen unterstützt AWS-Regionen:
>
> - USA Ost (Nord-Virginia)
> - USA West (Oregon)
> - Asien-Pazifik (Tokio)
> - Europa (Irland)

Erste Schritte

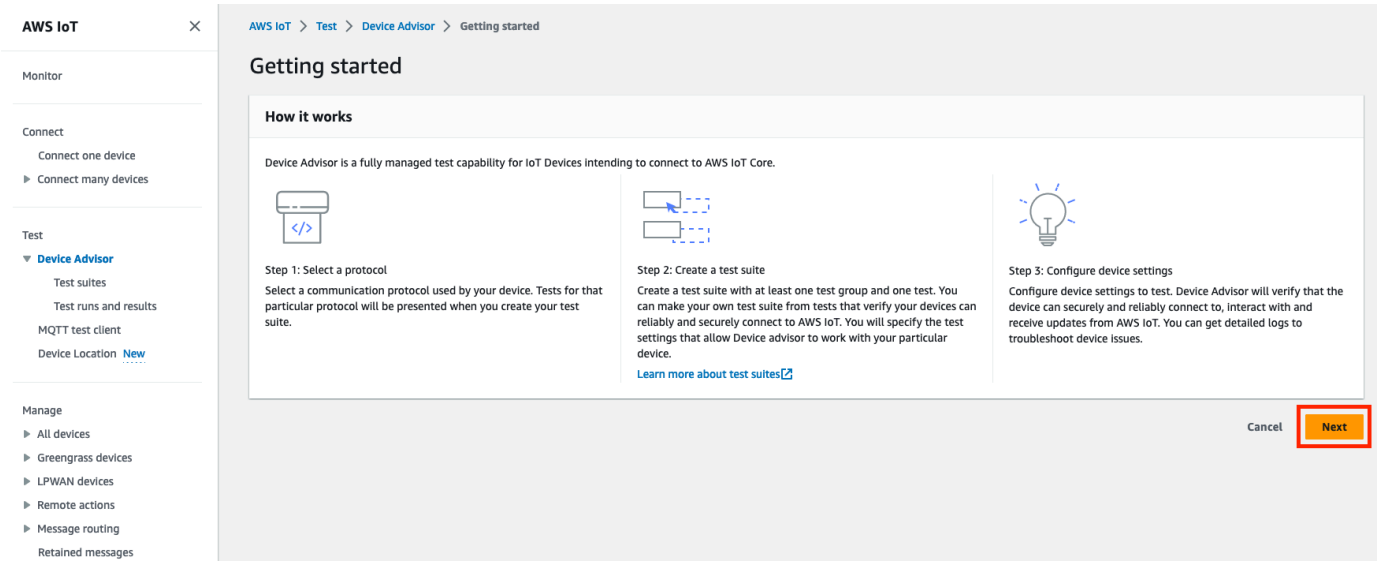
1. Wählen Sie im Navigationsbereich der AWS IoT -Konsole unter Test die Option Device Advisor aus. Wählen Sie dann auf der Konsole die Schaltfläche Walkthrough starten aus.



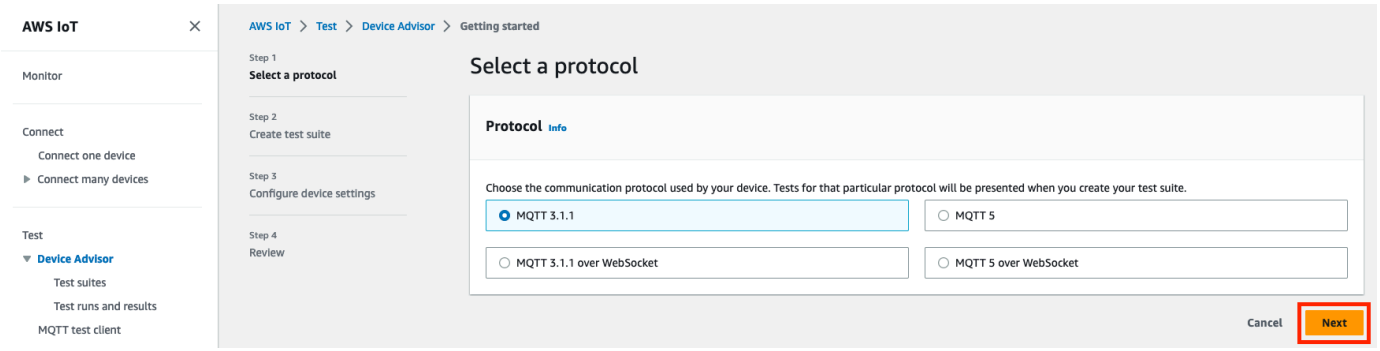
2. Die Seite Erste Schritte mit Device Advisor bietet einen Überblick über die Schritte, die zum Erstellen einer Testsuite und zum Ausführen von Tests auf Ihrem Gerät erforderlich sind. Den

Device-Advisor-Testendpunkt für Ihr Konto finden Sie auch hier. Sie müssen die Firmware oder Software auf dem zum Testen verwendeten Gerät konfigurieren, um eine Verbindung zu diesem Testendpunkt herzustellen.

Um dieses Tutorial abzuschließen, erstellen Sie zunächst ein Objekt und ein Zertifikat. Nachdem Sie sich die Informationen unter Funktionsweise gelesen haben, wählen Sie Weiter aus.

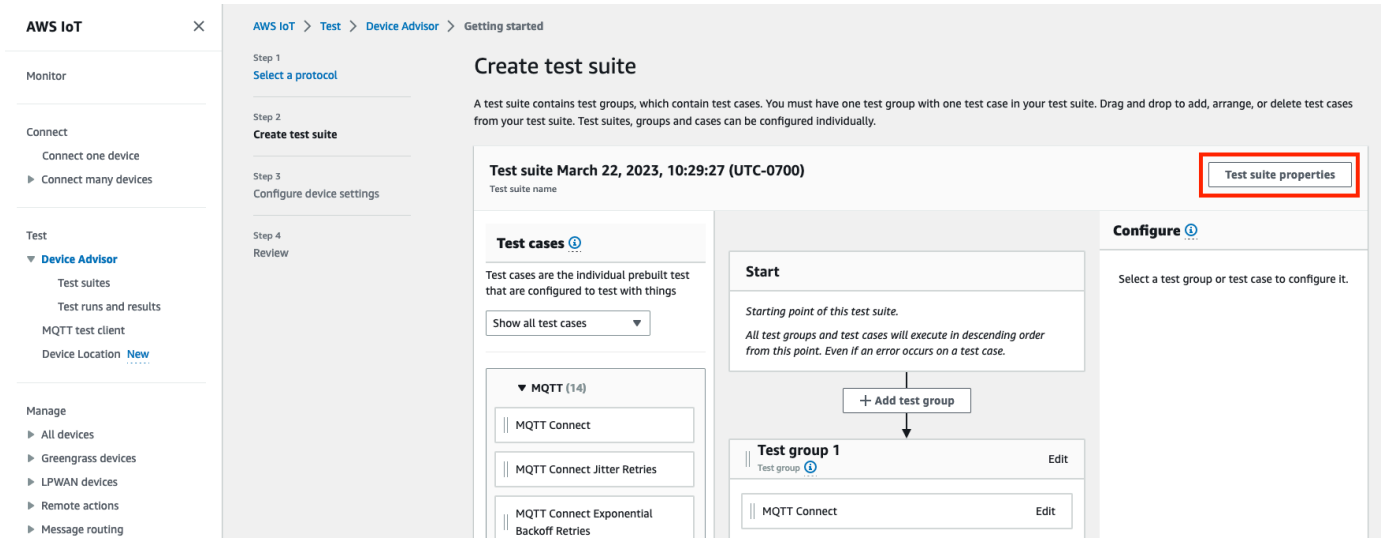


3. Wählen Sie in Schritt 1: Protokoll auswählen ein Protokoll aus den aufgelisteten Optionen aus. Wählen Sie anschließend Weiter.



4. In Schritt 2 erstellen und konfigurieren Sie eine benutzerdefinierte Testsuite. Eine benutzerdefinierte Testsuite muss mindestens eine Testgruppe haben und jede Testgruppe muss mindestens einen Testfall haben. Wir haben den MQTT-Connect-Testfall hinzugefügt, damit Sie erste Schritte unternehmen können.

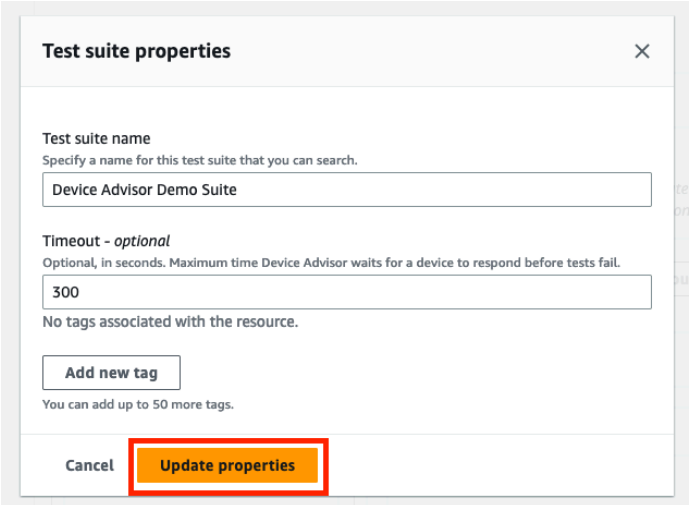
   Wählen Sie Eigenschaften der Testsuite aus.



Geben Sie die Eigenschaften der Testsuite an, wenn Sie Ihre Testsuite erstellen. Sie können die folgenden Eigenschaften auf Suite-Ebene konfigurieren:

- Name der Testsuite: Geben Sie einen Namen für Ihre Testsuite ein.
- Timeout (optional): Das Timeout (in Sekunden) für jeden Testfall in der aktuellen Testsuite. Wenn Sie keinen Timeout-Wert angeben, wird der Standardwert verwendet.
- Tags (optional): Fügen Sie der Testsuite Tags hinzu.

Wenn Sie fertig sind, wählen Sie Eigenschaften aktualisieren aus.

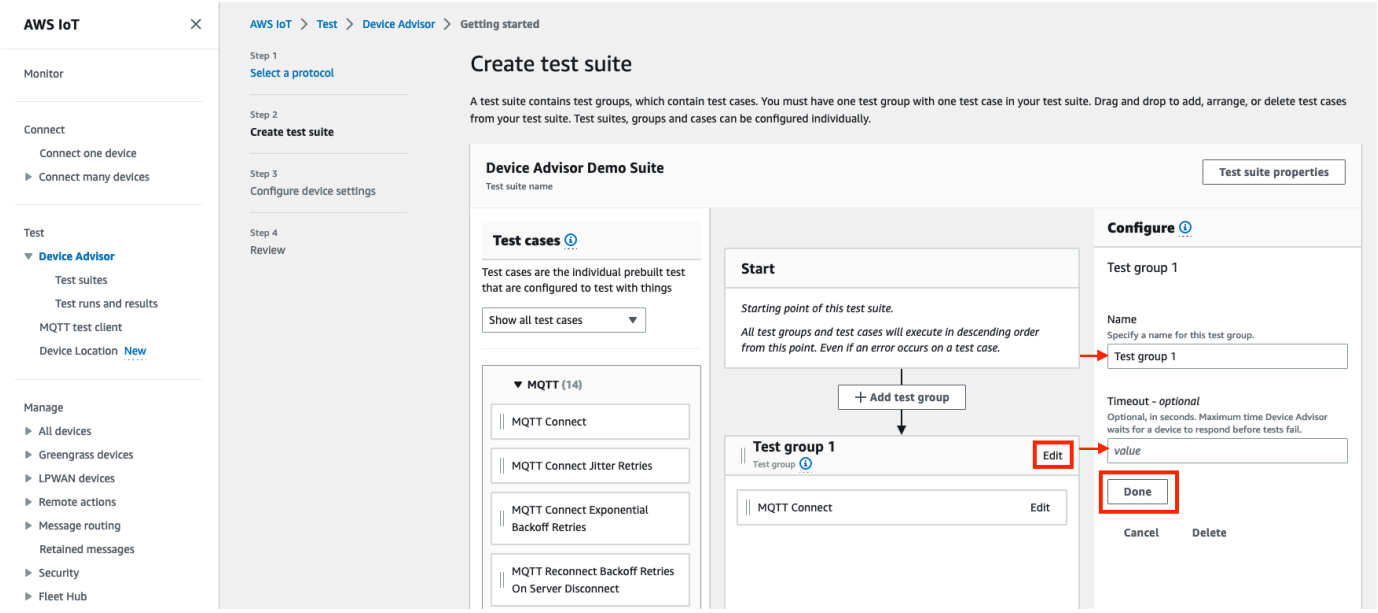


5. (Optional) Um die Konfiguration der Testsuite-Gruppe zu aktualisieren, klicken Sie auf die Schaltfläche Bearbeiten neben dem Namen der Testgruppe.

   - Name: Geben Sie einen Namen für die Testsuite-Gruppe ein.

- Timeout (optional): Das Timeout (in Sekunden) für jeden Testfall in der aktuellen Testsuite. Wenn Sie keinen Timeout-Wert angeben, wird der Standardwert verwendet.

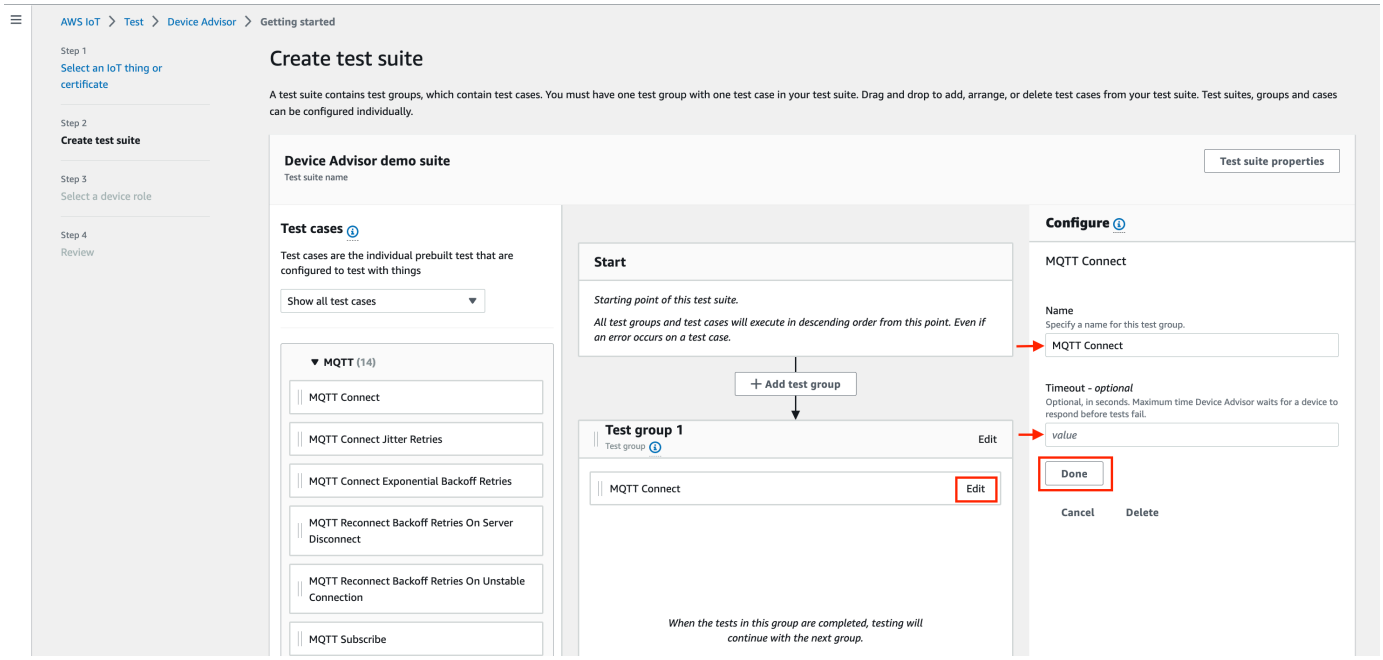
Wenn Sie fertig sind, wählen Sie Fertig aus, um fortzufahren.



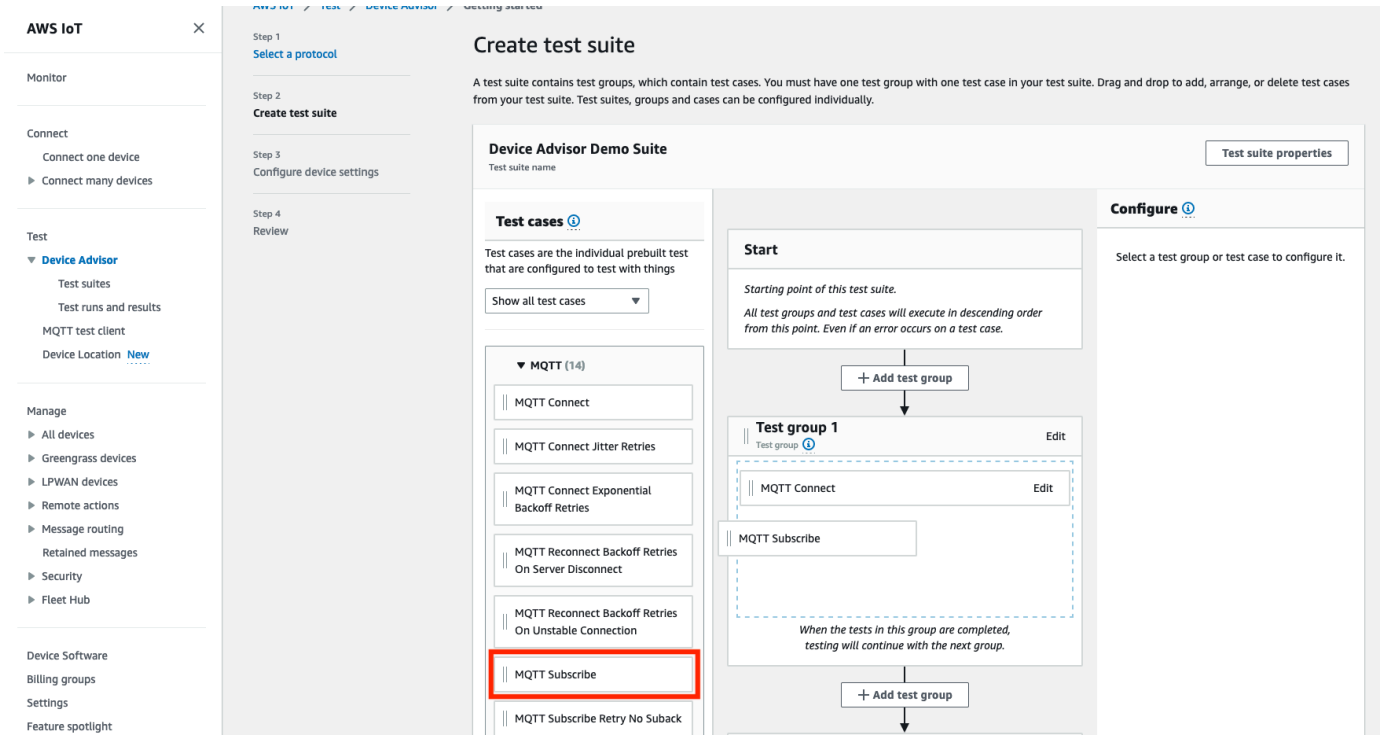
6. (Optional) Um die Testfall-Konfiguration für einen Testfall zu aktualisieren, klicken Sie auf die Schaltfläche Bearbeiten neben dem Namen der Testgruppe.

   - Name: Geben Sie einen Namen für die Testsuite-Gruppe ein.
   - Timeout (optional): Das Timeout (in Sekunden) für den ausgewählten Testfall. Wenn Sie keinen Timeout-Wert angeben, wird der Standardwert verwendet.

   Wenn Sie fertig sind, wählen Sie Fertig aus, um fortzufahren.



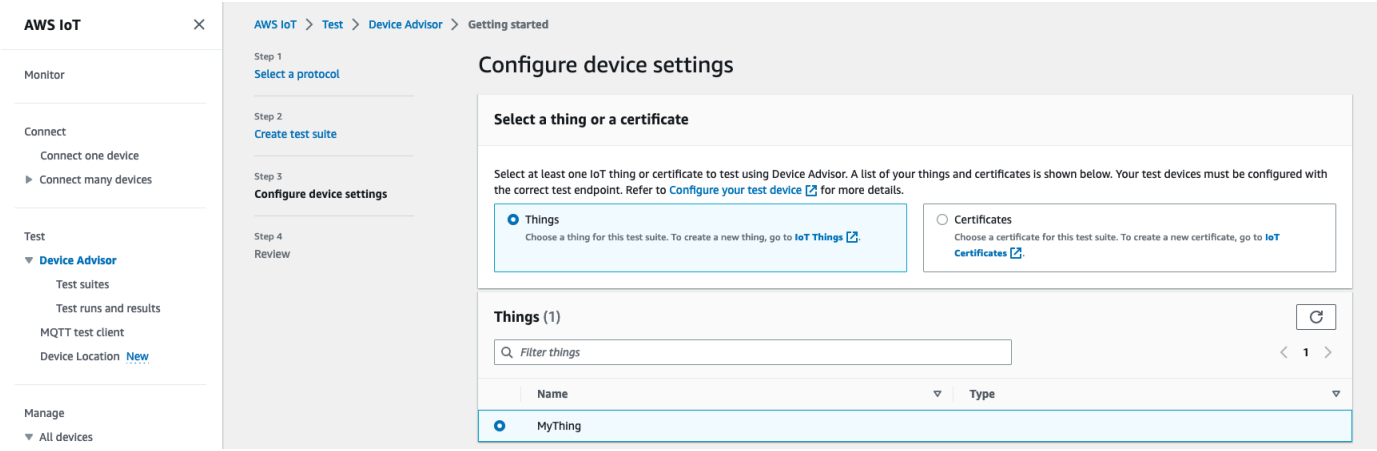
7. (Optional) Um der Testsuite weitere Testgruppen hinzuzufügen, wählen Sie Add test group (Testgruppe hinzufügen) aus und folgen Sie dann den Anweisungen in Schritt 5.
8. (Optional) Um weitere Testfälle hinzuzufügen, ziehen Sie die Testfälle im Abschnitt Testfälle in eine Ihrer Testgruppen.



9. Sie können die Reihenfolge Ihrer Testgruppen und Testfälle ändern. Ziehen Sie die aufgelisteten Testfälle in der Liste nach oben oder unten, um Änderungen vorzunehmen. Device Advisor führt Tests in der Reihenfolge aus, in der Sie sie aufgelistet haben.

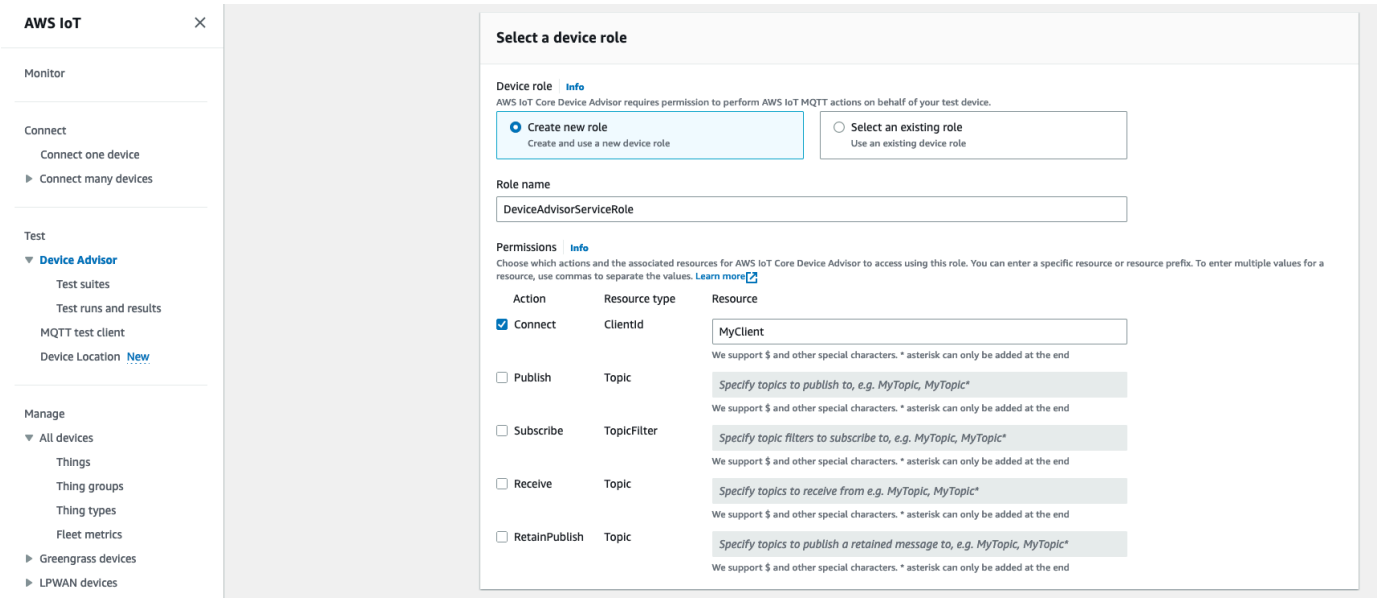
   Nachdem Sie Ihre Testsuite konfiguriert haben, wählen Sie Weiter aus.

10. Wählen Sie in Schritt 3 eine AWS IoT Sache oder ein Zertifikat aus, das Sie mit Device Advisor testen möchten. Wenn Sie noch keine Dinge oder Zertifikate haben, finden Sie weitere Informationen unter Einrichten.

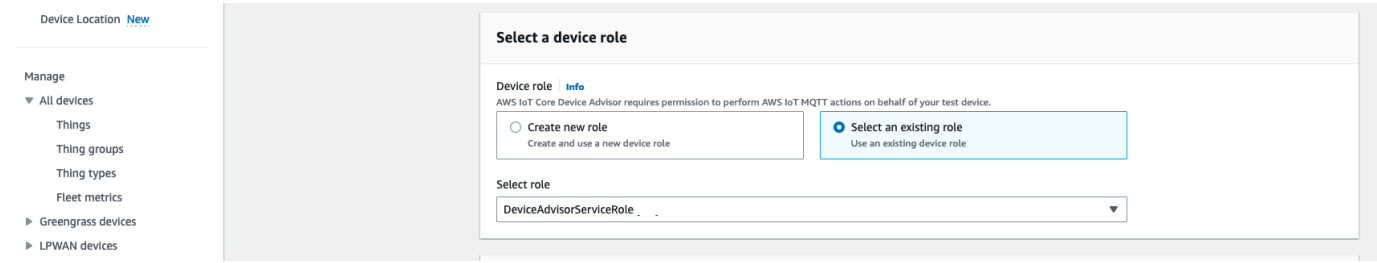


11. Sie können eine Geräterolle konfigurieren, die Device Advisor verwendet, um AWS IoT MQTT-Aktionen im Namen Ihres Testgeräts auszuführen. Die Connect-Aktion wird nur für den MQTT-Connect-Testfall automatisch ausgewählt. Dies liegt daran, dass die Geräterolle diese Berechtigung benötigt, um die Testsuite auszuführen. Für andere Testfälle werden die entsprechenden Aktionen ausgewählt.

    Geben Sie die Ressourcenwerte für jede der ausgewählten Aktionen an. Geben Sie beispielsweise für die Connect-Aktion die Client-ID an, die Ihr Gerät für die Verbindung mit dem Device-Advisor-Endpunkt verwendet. Sie können mehrere Werte mit durch Kommas getrennten Werten und Präfixwerte mit einem Platzhalterzeichen (*) angeben. Um beispielsweise die Erlaubnis zur Veröffentlichung zu einem beliebigen Thema zu erteilen, das mit `MyTopic` beginnt, geben Sie **`MyTopic*`** als Ressourcenwert ein.

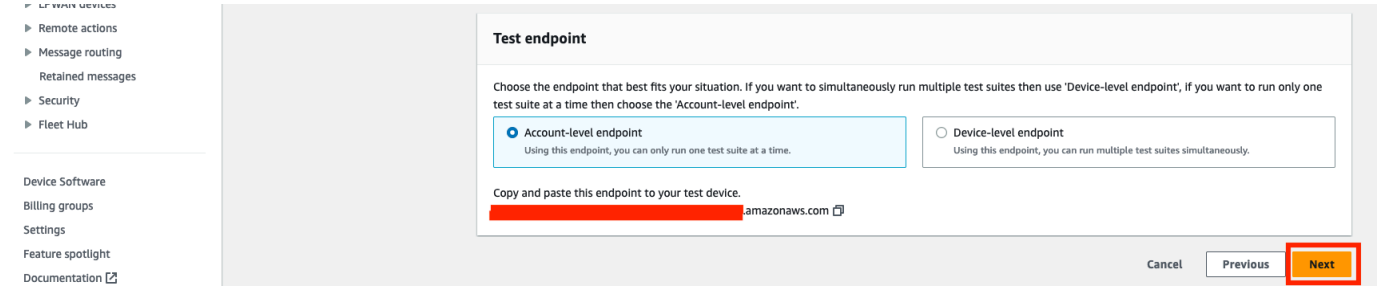


Um eine zuvor erstellte Geräterolle aus Einrichten zu verwenden, wählen Sie Vorhandene Rolle auswählen aus. Wählen Sie dann unter Rolle auswählen Ihre Geräterolle aus.



Konfigurieren Sie Ihre Geräterolle mit einer der beiden verfügbaren Optionen und wählen Sie dann Weiter aus.

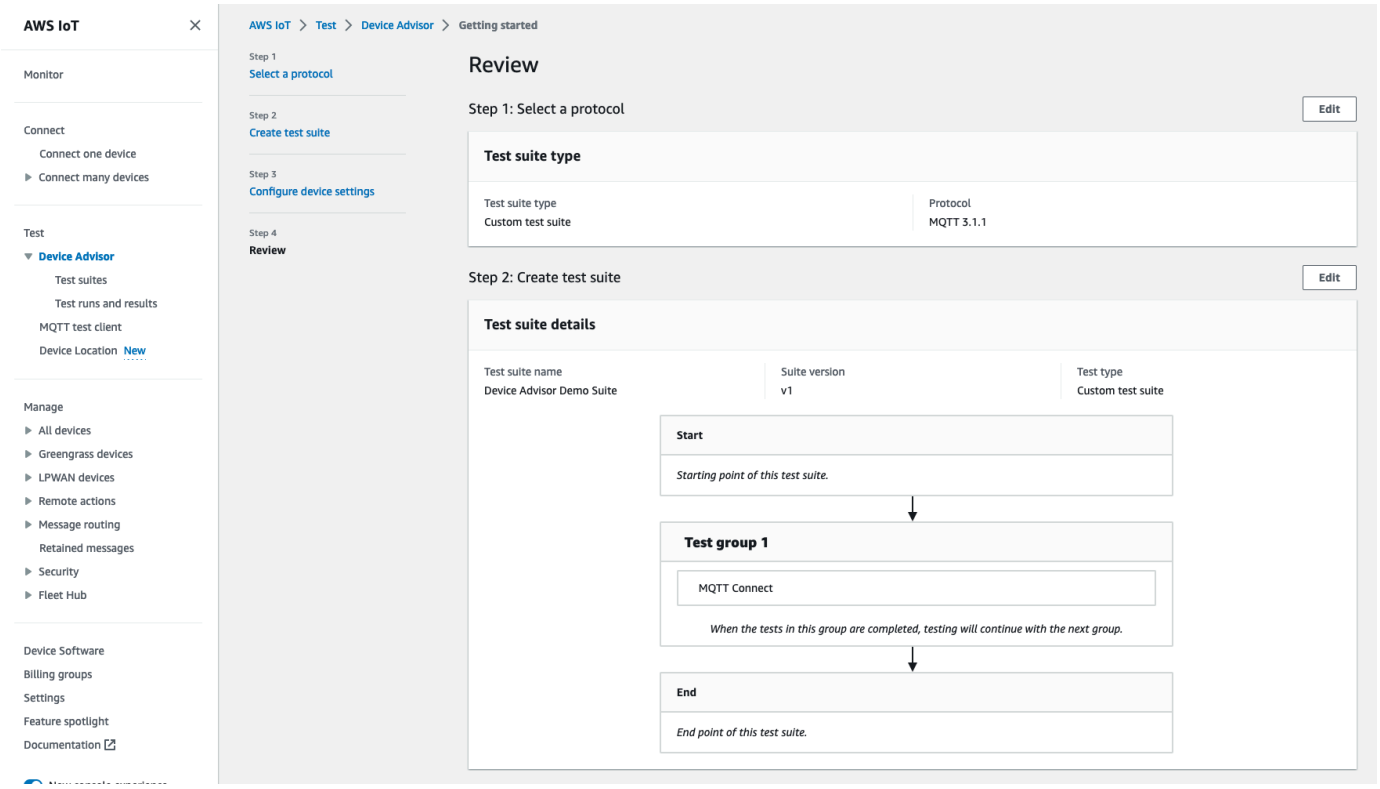
12. Wählen Sie im Abschnitt Testendpunkt den Endpunkt aus, der am besten zu Ihrem Anwendungsfall passt. Um mehrere Testsuiten gleichzeitig gleichzeitig auszuführen AWS-Konto, wählen Sie Device-level Endpunkt aus. Um jeweils eine Testsuite auszuführen, wählen Sie Account-level Endpunkt aus.

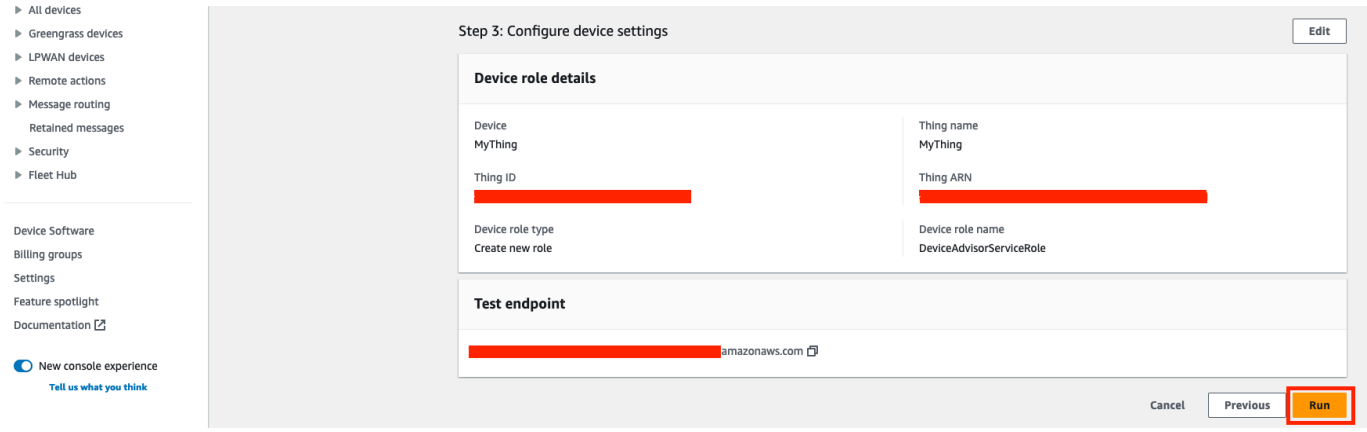


13. Schritt 4 zeigt eine Übersicht über das ausgewählte Testgerät, den Testendpunkt, die Testsuite und die konfigurierte Testgeräterolle. Wählen Sie die Schaltfläche Bearbeiten für jeden

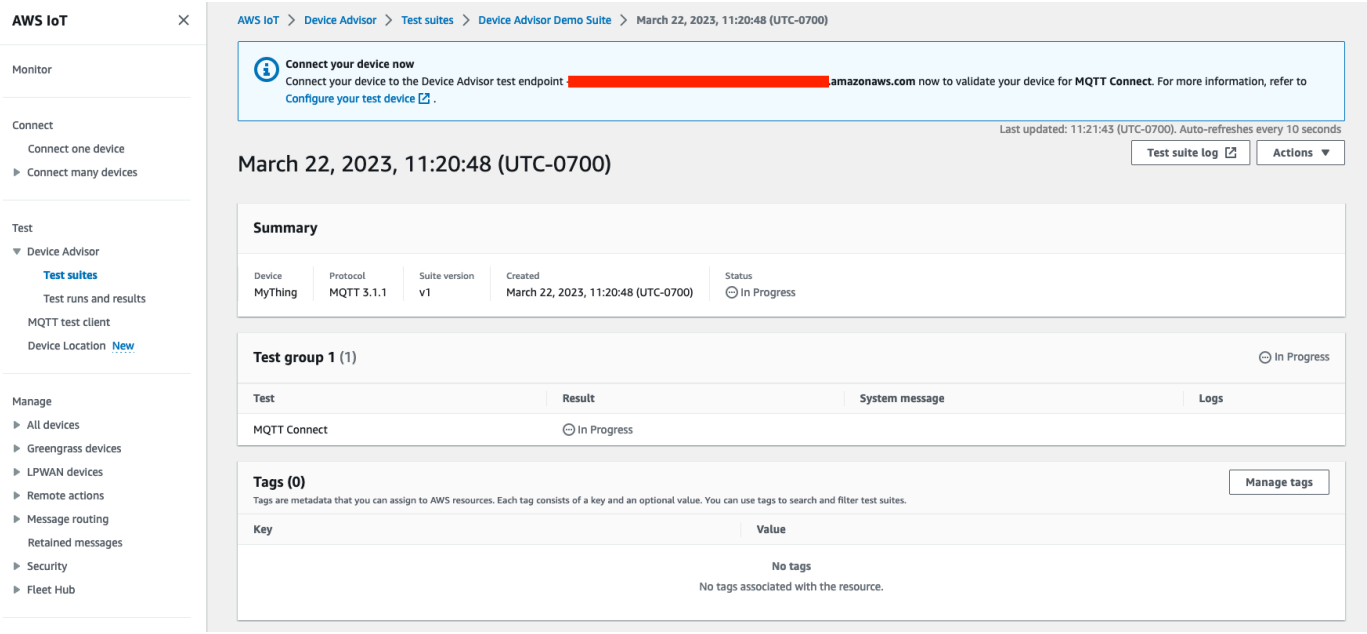
Abschnitt aus, den Sie bearbeiten möchten, um Änderungen vorzunehmen. Nachdem Sie Ihre Testkonfiguration bestätigt haben, wählen Sie Ausführen aus, um die Testsuite zu erstellen und Ihre Tests auszuführen.

> **Note**
>
> Zum Erzielen optimaler Ergebnisse können Sie Ihr ausgewähltes Testgerät mit dem Device-Advisor-Testendpunkt verbinden, bevor Sie mit der Ausführung der Testsuite beginnen. Wir empfehlen, dass Sie für Ihr Gerät einen Mechanismus entwickeln, mit dem Sie alle fünf Sekunden bis zu ein bis zwei Minuten lang versuchen können, eine Verbindung zu unserem Testendpunkt herzustellen.





14. Wählen Sie im Navigationsbereich unter Test die Option Device Advisor und dann Testläufe und Ergebnisse aus. Wählen Sie eine Testsuite-Ausführung aus, um deren Ausführungsdetails und Protokolle anzuzeigen.



15. Gehen Sie wie folgt vor, um auf die CloudWatch Amazon-Logs für die Suite zuzugreifen:

    - Wählen Sie Test Suite Log, um die CloudWatch Protokolle für den Test Suite-Lauf anzuzeigen.
    - Wählen Sie Testfallprotokoll für einen beliebigen Testfall, um testfallspezifische CloudWatch Protokolle anzuzeigen.

16. Führen Sie auf der Grundlage Ihrer Testergebnisse eine Fehlerbehebung auf Ihrem Gerät durch, bis alle Tests bestanden sind.

Device-Advisor-Workflow

In diesem Tutorial wird erklärt, wie Sie eine benutzerdefinierte Testsuite erstellen und Tests für das Gerät ausführen, das Sie in der Konsole testen möchten. Nach Abschluss der Tests können Sie die Testergebnisse und detaillierte Protokolle anzeigen.

Voraussetzungen

Bevor Sie mit diesem Tutorial beginnen, führen Sie die unter Einrichtung beschriebenen Schritte aus.

Erstellen einer Testsuite-Definition

Installieren Sie zunächst ein AWS SDK.

`RootGroup-Syntax`

Eine Root-Gruppe ist eine JSON-Zeichenfolge, die angibt, welche Testfälle in Ihre Testsuite aufgenommen werden sollen. Sie spezifiziert auch alle erforderlichen Konfigurationen für diese Testfälle. Verwenden Sie die Root-Gruppe, um Ihre Testsuite nach Ihren Bedürfnissen zu strukturieren und zu ordnen. Die Hierarchie einer Testsuite ist folgende:

```
test suite # test group(s) # test case(s)
```

Eine Testsuite muss mindestens eine Testgruppe haben und jede Testgruppe muss mindestens einen Testfall haben. Device Advisor führt Tests in der Reihenfolge aus, in der Sie die Testgruppen und Testfälle definieren.

Jede Root-Gruppe folgt dieser Grundstruktur:

```
{
    "configuration": {  // for all tests in the test suite
        "": ""
    }
    "tests": [{
        "name": ""
        "configuration": {  // for all sub-groups in this test group
            "": ""
        },
        "tests": [{
```

```
            "name": ""
            "configuration": {  // for all test cases in this test group
                "": ""
            },
            "test": {
                "id": ""
                "version": ""
            }
        }]
    }]
}
```

In der Root-Gruppe definieren Sie die Testsuite mit einem `name`, der `configuration` und den `tests`, die die Gruppe enthält. Die `tests`-Gruppe enthält die Definitionen der einzelnen Tests. Sie definieren jeden Test mit einem `name`, einer `configuration` und einem `test`-Block, der die Testfälle für diesen Test definiert. Schließlich wird jeder Testfall mit einer `id` und einer `version` definiert.

Hinweise zur Verwendung der Felder `"id"` und `"version"` und für jeden Testfall (`test`-Block) finden Sie unter Device-Advisor-Testfälle. Dieser Abschnitt enthält auch Informationen zu den verfügbaren `configuration`-Einstellungen.

Der folgende Block ist ein Beispiel für eine Root-Gruppenkonfiguration. Diese Konfiguration spezifiziert die Testfälle MQTT Connect Happy Case und MQTT Connect Exponential Backoff Retries zusammen mit Beschreibungen der Konfigurationsfelder.

```
{
    "configuration": {},  // Suite-level configuration
    "tests": [            // Group definitions should be provided here
      {
        "name": "My_MQTT_Connect_Group",  // Group definition name
        "configuration": {}               // Group definition-level configuration,
        "tests": [                        // Test case definitions should be provided
 here
        {
            "name": "My_MQTT_Connect_Happy_Case",  // Test case definition name
            "configuration": {
                "EXECUTION_TIMEOUT": 300        // Test case definition-level
 configuration, in seconds
            },
            "test": {
```

```
                "id": "MQTT_Connect",              // test case id
                "version": "0.0.0"                 // test case version
            }
        },
        {
            "name": "My_MQTT_Connect_Jitter_Backoff_Retries",  // Test case definition
 name
            "configuration": {
                "EXECUTION_TIMEOUT": 600                // Test case definition-level
 configuration,  in seconds
            },
            "test": {
                "id": "MQTT_Connect_Jitter_Backoff_Retries",  // test case id
                "version": "0.0.0"                                  // test case version
            }
        }]
    }]
}
```

Sie müssen die Root-Gruppenkonfiguration angeben, wenn Sie Ihre Testsuite-Definition erstellen. Speichern Sie die `suiteDefinitionId`, die im Antwortobjekt zurückgegeben wird. Sie können diese ID verwenden, um Ihre Testsuite-Definitionsinformationen abzurufen und Ihre Testsuite auszuführen.

Hier ist ein Beispiel für ein Java-SDK:

```
response = iotDeviceAdvisorClient.createSuiteDefinition(
        CreateSuiteDefinitionRequest.builder()
            .suiteDefinitionConfiguration(SuiteDefinitionConfiguration.builder()
                .suiteDefinitionName("your-suite-definition-name")
                .devices(
                    DeviceUnderTest.builder()
                        .thingArn("your-test-device-thing-arn")
                        .certificateArn("your-test-device-certificate-arn")
                        .deviceRoleArn("your-device-role-arn") //if using SigV4 for
 MQTT over WebSocket
                        .build()
                )
                .rootGroup("your-root-group-configuration")
                .devicePermissionRoleArn("your-device-permission-role-arn")
                .protocol("MqttV3_1_1 || MqttV5 || MqttV3_1_1_OverWebSocket ||
 MqttV5_OverWebSocket")
                .build()
```

```
            )
            .build()
)
```

Abrufen einer Testsuite-Definition

Nachdem Sie Ihre Testsuite-Definition erstellt haben, erhalten Sie die `suiteDefinitionId` im Antwortobjekt der `CreateSuiteDefinition`-API-Operation.

Wenn der Vorgang die `suiteDefinitionId` zurückgibt, sehen Sie möglicherweise neue `id`-Felder in jeder Gruppe und eine Testfalldefinition innerhalb der Root-Gruppe. Sie können diese IDs verwenden, um eine Teilmenge Ihrer Testsuite-Definition auszuführen.

Java-SDK-Beispiel:

```
response = iotDeviceAdvisorClient.GetSuiteDefinition(
    GetSuiteDefinitionRequest.builder()
        .suiteDefinitionId("your-suite-definition-id")
        .build()
)
```

Abrufen eines Testendpunkts

Verwenden Sie die `GetEndpoint`-API-Operation, um den von Ihrem Gerät verwendeten Testendpunkt abzurufen. Wählen Sie den Endpunkt aus, der am besten zu Ihrem Test passt. Um mehrere Testsuiten gleichzeitig auszuführen, verwenden Sie den Device-level Endpunkt, indem Sie ein `thing ARN` `certificate ARN`, oder angeben. `device role ARN` Um eine einzelne Testsuite auszuführen, geben Sie keine Argumente für den GetEndpoint Vorgang zur Auswahl des Account-level Endpunkts an.

SDK-Beispiel:

```
response = iotDeviceAdvisorClient.getEndpoint(GetEndpointRequest.builder()
.certificateArn("your-test-device-certificate-arn")
.thingArn("your-test-device-thing-arn")
.deviceRoleArn("your-device-role-arn") //if using SigV4 for MQTT over WebSocket

.build())
```

Ausführen einer Testsuite

Nachdem Sie eine Testsuite-Definition erstellt und Ihr Testgerät so konfiguriert haben, dass es eine Verbindung zu Ihrem Device-Advisor-Testendpunkt herstellt, führen Sie Ihre Testsuite mit der `StartSuiteRun`-API aus.

Für MQTT-Kunden verwenden Sie entweder `certificateArn` oder `thingArn`, um die Testsuite auszuführen. Wenn beide konfiguriert sind, wird das Zertifikat verwendet, wenn es zum Ding gehört.

Verwenden Sie für MQTT statt WebSocket Kunde, `deviceRoleArn` um die Testsuite auszuführen. Wenn sich die angegebene Rolle von der in der Testsuite-Definition angegebenen Rolle unterscheidet, hat die angegebene Rolle Vorrang vor der definierten Rolle.

Verwenden Sie diese Option z. B.`.parallelRun()`, `true` wenn Sie einen Device-level Endpunkt verwenden, um mehrere Testsuiten parallel mit einer AWS-Konto Testsuite auszuführen.

SDK-Beispiel:

```
response = iotDeviceAdvisorClient.startSuiteRun(StartSuiteRunRequest.builder()
.suiteDefinitionId("your-suite-definition-id")
.suiteRunConfiguration(SuiteRunConfiguration.builder()
    .primaryDevice(DeviceUnderTest.builder()
        .certificateArn("your-test-device-certificate-arn")
        .thingArn("your-test-device-thing-arn")
        .deviceRoleArn("your-device-role-arn") //if using SigV4 for MQTT over WebSocket

        .build())
    .parallelRun(true | false)
    .build())
.build())
```

Speichern Sie die `suiteRunId` aus der Antwort. Sie werden diese verwenden, um die Ergebnisse dieser Testsuite-Ausführung abzurufen.

Abrufen einer Testsuite-Ausführung

Nachdem Sie eine Testsuite ausgeführt haben, können Sie ihren Fortschritt und ihre Ergebnisse mit der `GetSuiteRun`-API überprüfen.

SDK-Beispiel:

```
// Using the SDK, call the GetSuiteRun API.

response = iotDeviceAdvisorClient.GetSuiteRun(
GetSuiteRunRequest.builder()
    .suiteDefinitionId("your-suite-definition-id")
    .suiteRunId("your-suite-run-id")
.build())
```

Beenden einer Testsuite-Ausführung

Um eine Testsuite-Ausführung, die noch läuft, zu beenden, können Sie die `StopSuiteRun`-API-Operation aufrufen. Nachdem Sie die `StopSuiteRun`-Operation aufgerufen haben, startet der Service den Bereinigungsvorgang. Während der Service den Bereinigungsvorgang ausführt, führt die Testsuite Statusaktualisierungen bis `Stopping` durch. Der Bereinigungsvorgang kann mehrere Minuten in Anspruch nehmen. Sobald der Vorgang abgeschlossen ist, führt die Testsuite Statusaktualisierungen bis `Stopped` durch. Nachdem ein Testlauf vollständig beendet wurde, starten Sie eine weitere Testsuite-Ausführung. Sie können den Status der Ausführung der Suite regelmäßig mithilfe der `GetSuiteRun`-API-Operation überprüfen, wie im vorherigen Abschnitt gezeigt.

SDK-Beispiel:

```
// Using the SDK, call the StopSuiteRun API.

response = iotDeviceAdvisorClient.StopSuiteRun(
StopSuiteRun.builder()
    .suiteDefinitionId("your-suite-definition-id")
    .suiteRunId("your-suite-run-id")
.build())
```

Abrufen eines Qualifizierungsberichts für eine erfolgreiche Ausführung der Qualifizierungstestsuite

Wenn Sie eine Qualifizierungstestsuite ausführen, die erfolgreich abgeschlossen wurde, können Sie mit der `GetSuiteRunReport`-API-Operation einen Qualifizierungsbericht abrufen. Sie verwenden diesen Qualifizierungsbericht, um Ihr Gerät für das AWS IoT Core -Qualifizierungsprogramm zu qualifizieren. Um festzustellen, ob es sich bei Ihrer Testsuite um eine Qualifizierungstestsuite handelt, überprüfen Sie, ob der `intendedForQualification`-Parameter auf `true` eingestellt ist. Nachdem Sie die `GetSuiteRunReport`-API-Operation aufgerufen haben, können Sie den Bericht bis zu

90 Sekunden lang von der zurückgegebenen URL herunterladen. Wenn seit dem letzten Aufruf der `GetSuiteRunReport`-Operation mehr als 90 Sekunden vergangen sind, rufen Sie die Operation erneut auf, um eine neue, gültige URL abzurufen.

SDK-Beispiel:

```
// Using the SDK, call the getSuiteRunReport API.

response = iotDeviceAdvisorClient.getSuiteRunReport(
    GetSuiteRunReportRequest.builder()
        .suiteDefinitionId("your-suite-definition-id")
        .suiteRunId("your-suite-run-id")
        .build()
)
```

Ausführlicher Konsolen-Workflow von Device Advisor

In diesem Tutorial erstellen Sie eine benutzerdefinierte Testsuite und führen Tests für das Gerät aus, das Sie in der Konsole testen möchten. Nach Abschluss der Tests können Sie die Testergebnisse und detaillierte Protokolle anzeigen.

Lernprogramme

- Voraussetzungen
- Erstellen einer Testsuite-Definition
- Ausführen einer Testsuite
- Beenden einer Testsuite-Ausführung (optional)
- Anzeigen von Details und Protokolle der Testsuite-Ausführung
- Laden Sie eine herunter AWS IoT Qualifizierungsbericht

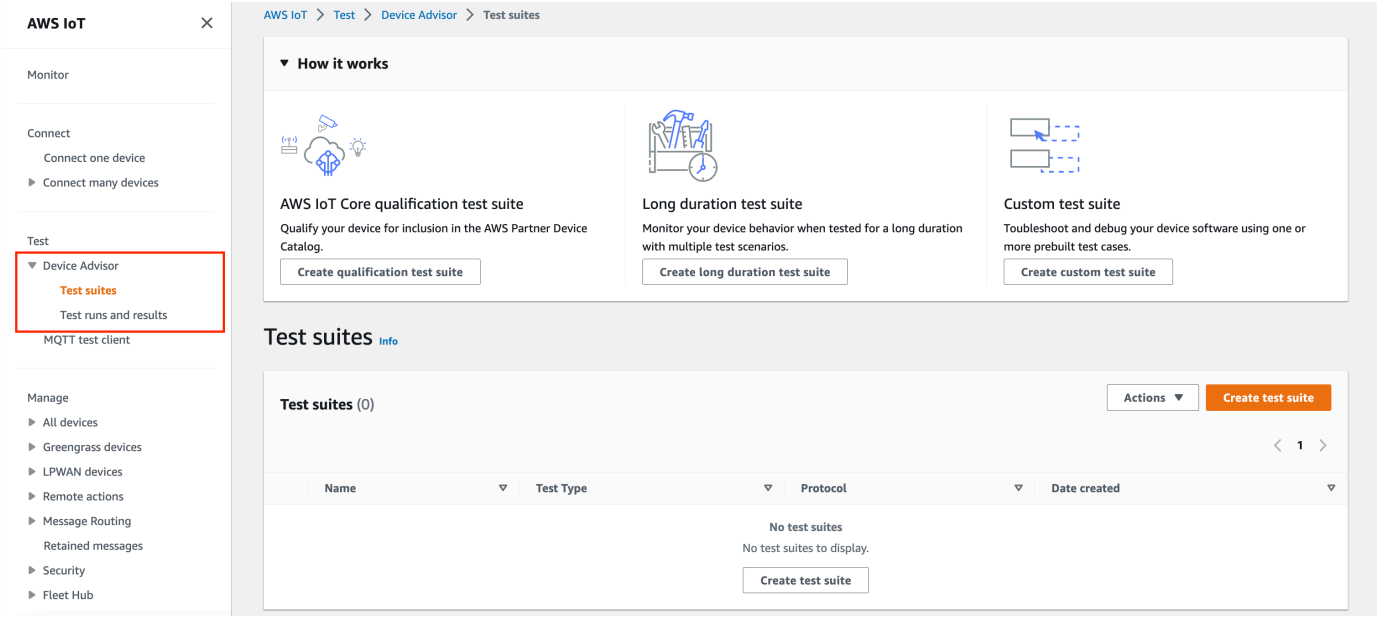
Voraussetzungen

Sie müssen zuerst ein Ding und ein Zertifikat erstellen, bevor Sie dieses Tutorial abschließen können.

Erstellen einer Testsuite-Definition

Erstellen Sie eine Testsuite, damit Sie sie für Ihre Geräte ausführen und die Überprüfung durchführen können.

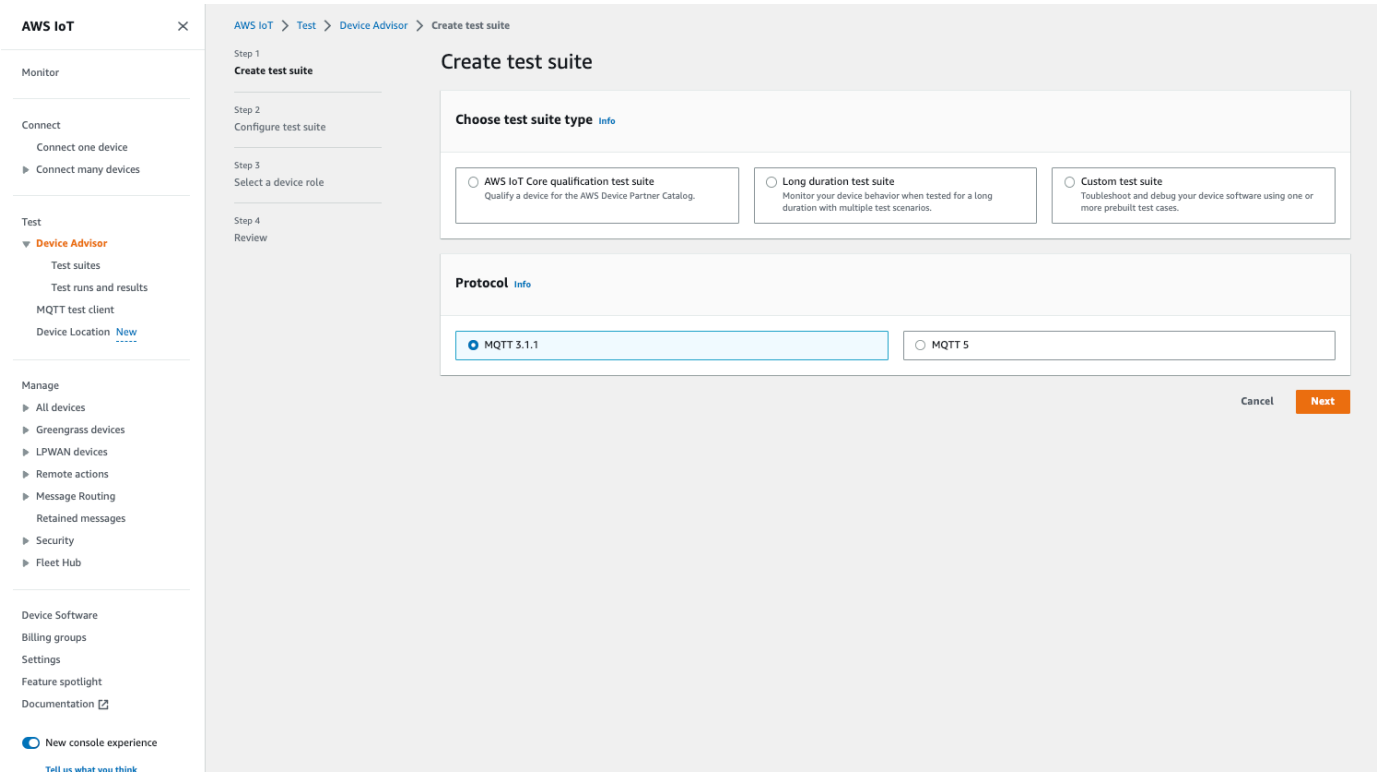
1. Erweitern Sie im Navigationsbereich in der AWS IoT -Konsole die Optionen Test, Device Advisor und wählen Sie dann Testsuites aus.



   Wählen Sie Testsuite erstellen aus.

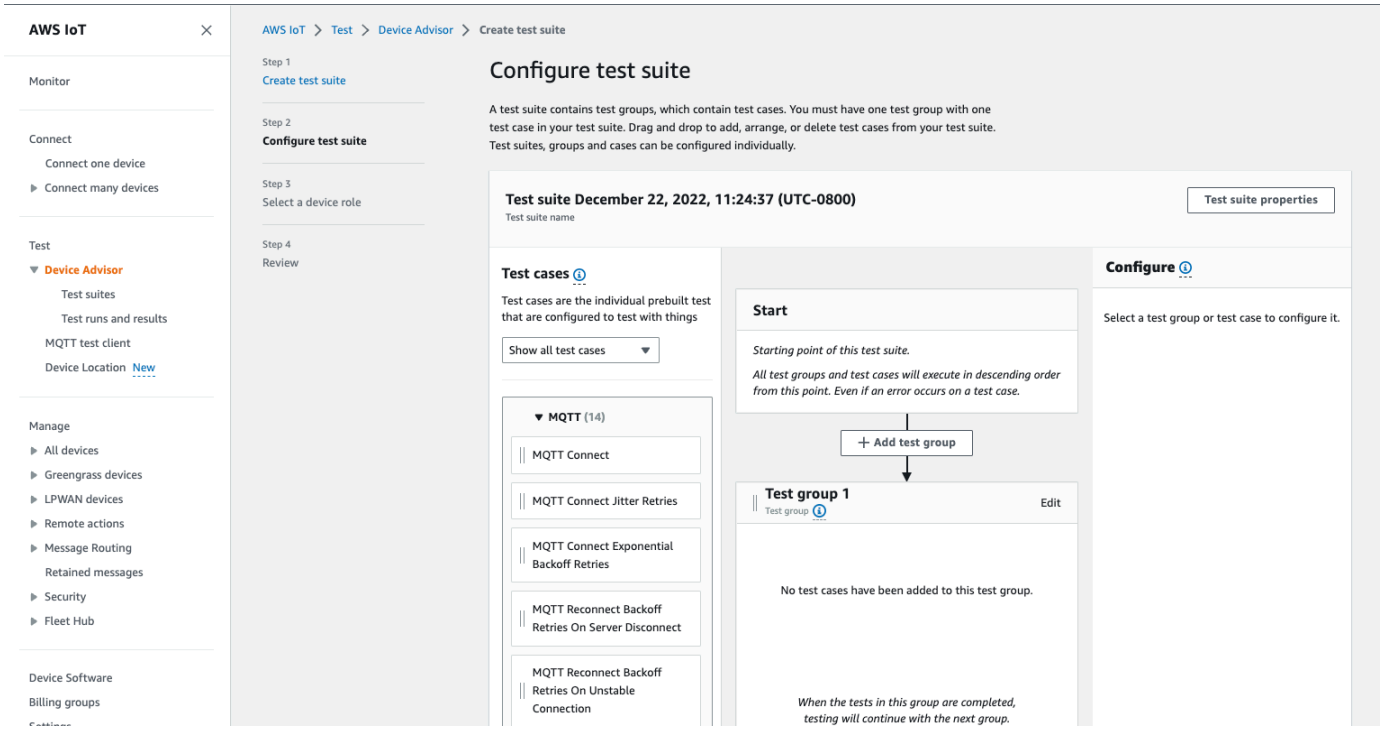
2. Wählen Sie entweder `Use the AWS Qualification test suite` oder `Create a new test suite` aus.

   Wählen Sie als Protokoll entweder MQTT 3.1.1 oder MQTT 5 aus.

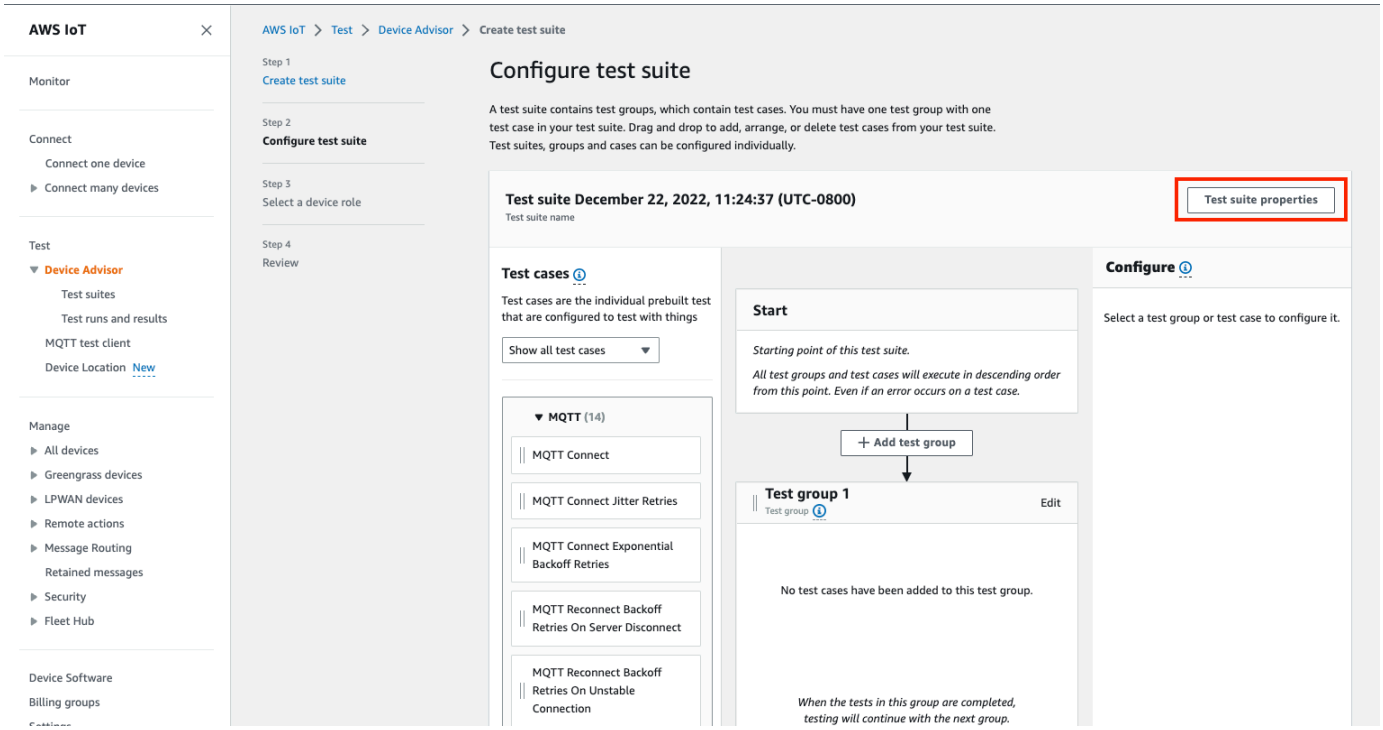


Wählen Sie`Use the AWS Qualification test suite`, ob Sie sich qualifizieren möchten, und nehmen Sie Ihr Gerät im AWS Partner-Gerätekatalog auf. Wenn Sie diese Option wählen, werden die Testfälle, die für die Qualifizierung Ihres Geräts für das AWS IoT Core - Qualifizierungsprogramm erforderlich sind, vorab ausgewählt. Testgruppen und Testfälle können nicht hinzugefügt oder entfernt werden. Sie müssen trotzdem die Eigenschaften der Testsuite konfigurieren.

Wählen Sie `Create a new test suite` aus, um eine benutzerdefinierte Testsuite zu erstellen und zu konfigurieren. Wir empfehlen, für erste Tests und zur Fehlerbehebung mit dieser Option zu beginnen. Eine benutzerdefinierte Testsuite muss mindestens eine Testgruppe haben und jede Testgruppe muss mindestens einen Testfall haben. Für die Zwecke dieses Tutorials wählen wir diese Option und wählen Weiter aus.



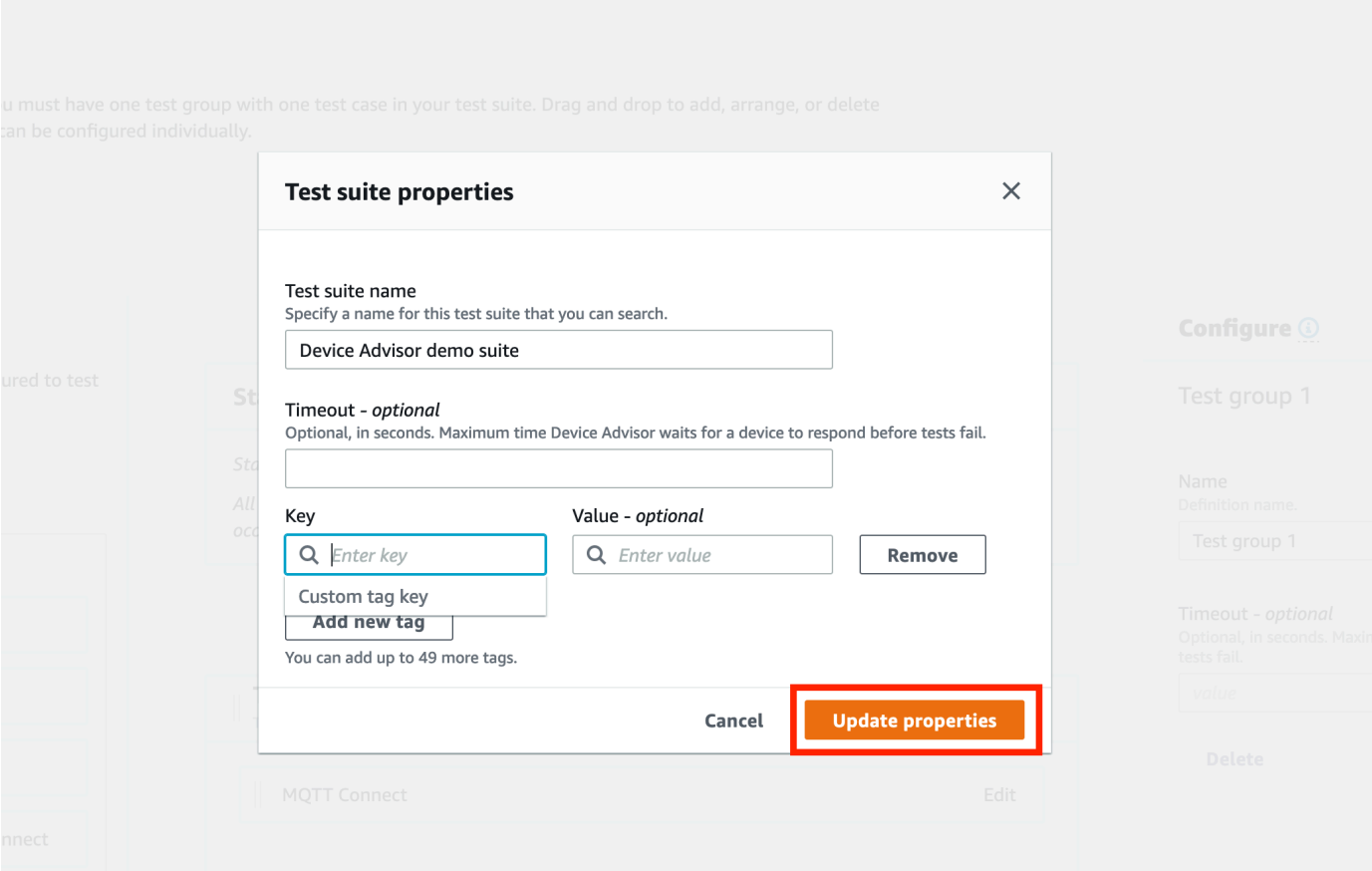
3. Wählen Sie Eigenschaften der Testsuite aus. Sie müssen die Eigenschaften der Testsuite erstellen, wenn Sie Ihre Testsuite erstellen.



Füllen Sie unter Eigenschaften der Testsuite die folgenden Felder aus:

- Name der Testsuite: Sie können die Suite mit einem benutzerdefinierten Namen erstellen.

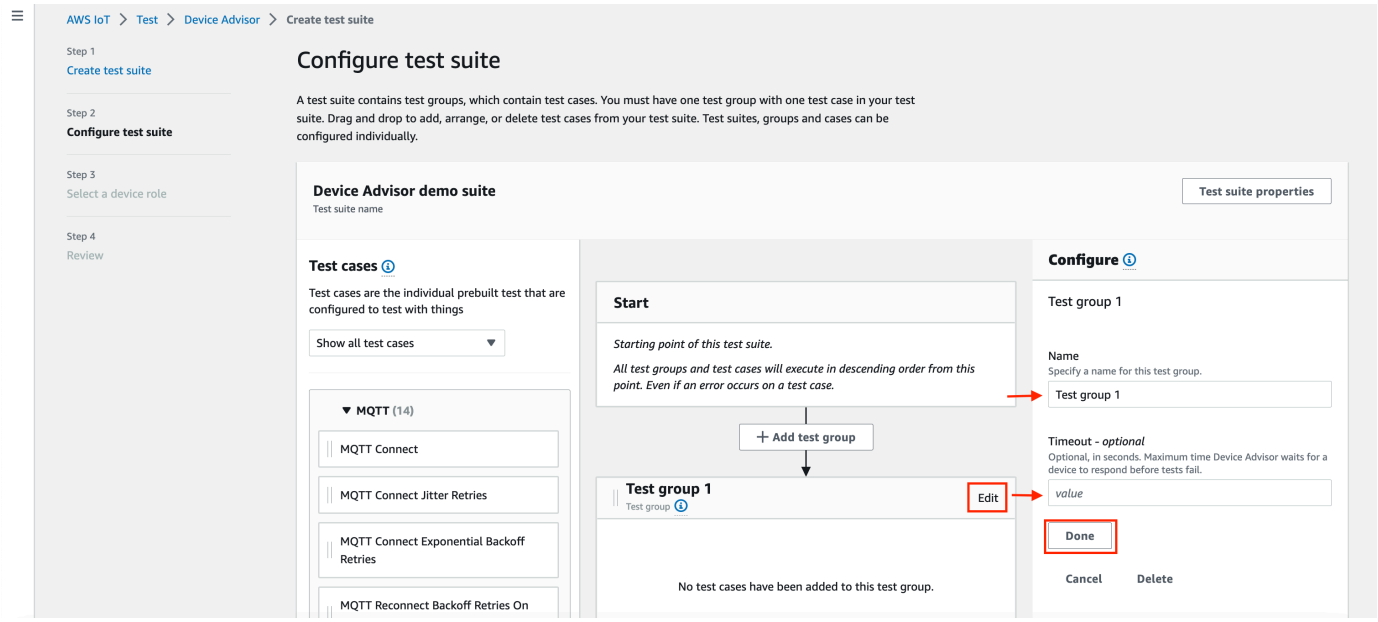
- Timeout (optional): Das Timeout in Sekunden für jeden Testfall in der aktuellen Testsuite. Wenn Sie keinen Timeout-Wert angeben, wird der Standardwert verwendet.
- Tags (optional): Fügen Sie der Testsuite Tags hinzu.



Wenn Sie fertig sind, wählen Sie Eigenschaften aktualisieren aus.

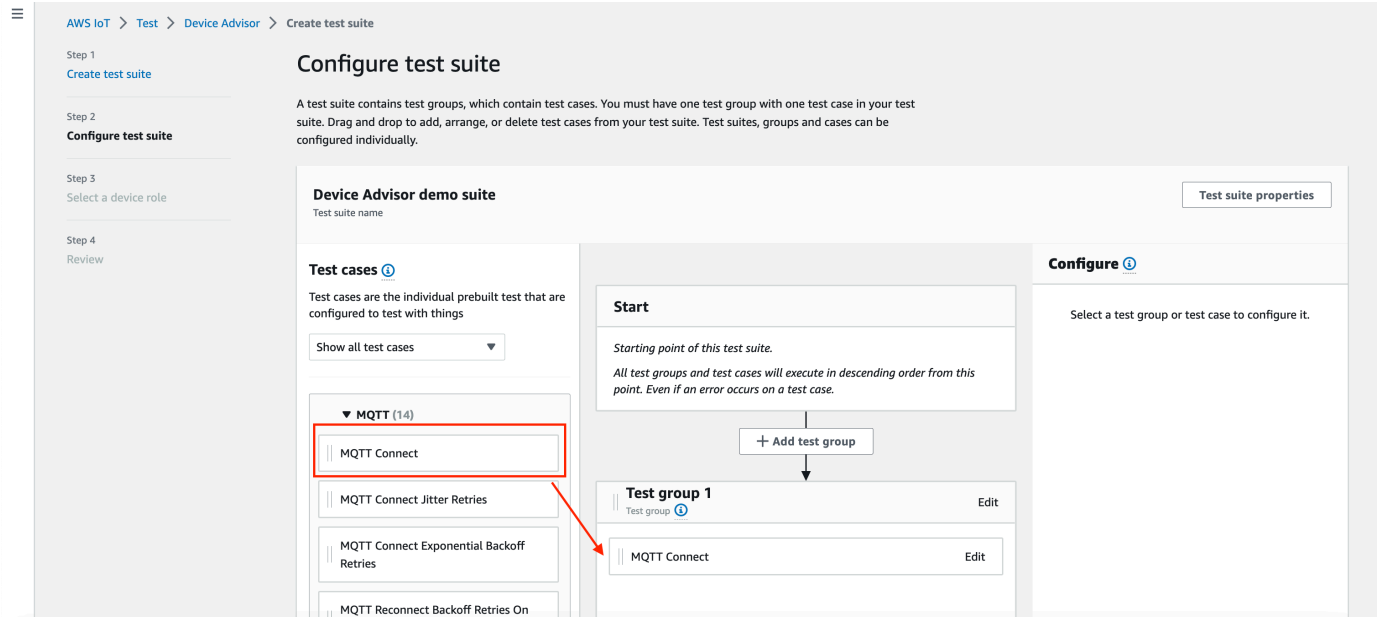
4. Um die Konfiguration auf Gruppenebene zu ändern, wählen Sie unter `Test group 1` Bearbeiten aus. Geben Sie dann einen Namen ein, um der Gruppe einen benutzerdefinierten Namen zu geben.

   Optional können Sie unter der ausgewählten Testgruppe auch einen Timeout-Wert in Sekunden eingeben. Wenn Sie keinen Timeout-Wert angeben, wird der Standardwert verwendet.



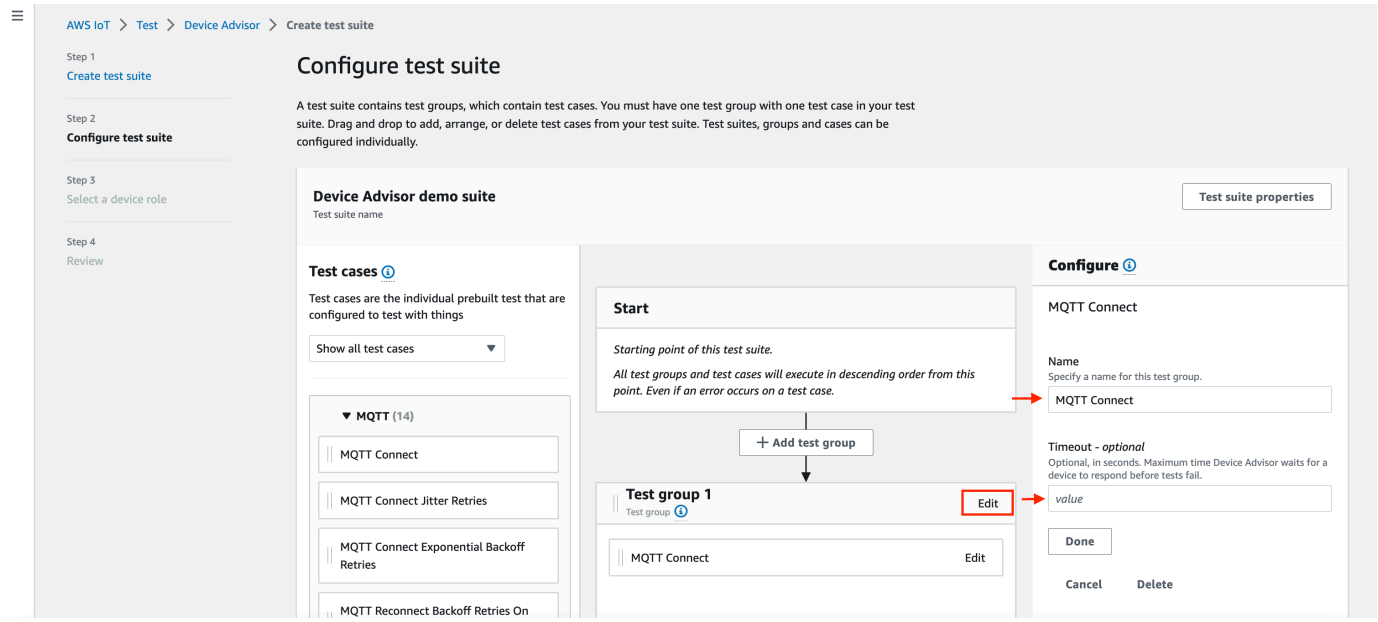
Wählen Sie Fertig aus.

5. Ziehen Sie einen der verfügbaren Testfälle aus Testfälle in die Testgruppe.



6. Um die Konfiguration auf Testfallebene für den Testfall zu ändern, den Sie Ihrer Testgruppe hinzugefügt haben, wählen Sie Bearbeiten aus. Geben Sie dann einen Namen ein, um der Gruppe einen benutzerdefinierten Namen zu geben.

   Optional können Sie unter der ausgewählten Testgruppe auch einen Timeout-Wert in Sekunden eingeben. Wenn Sie keinen Timeout-Wert angeben, wird der Standardwert verwendet.



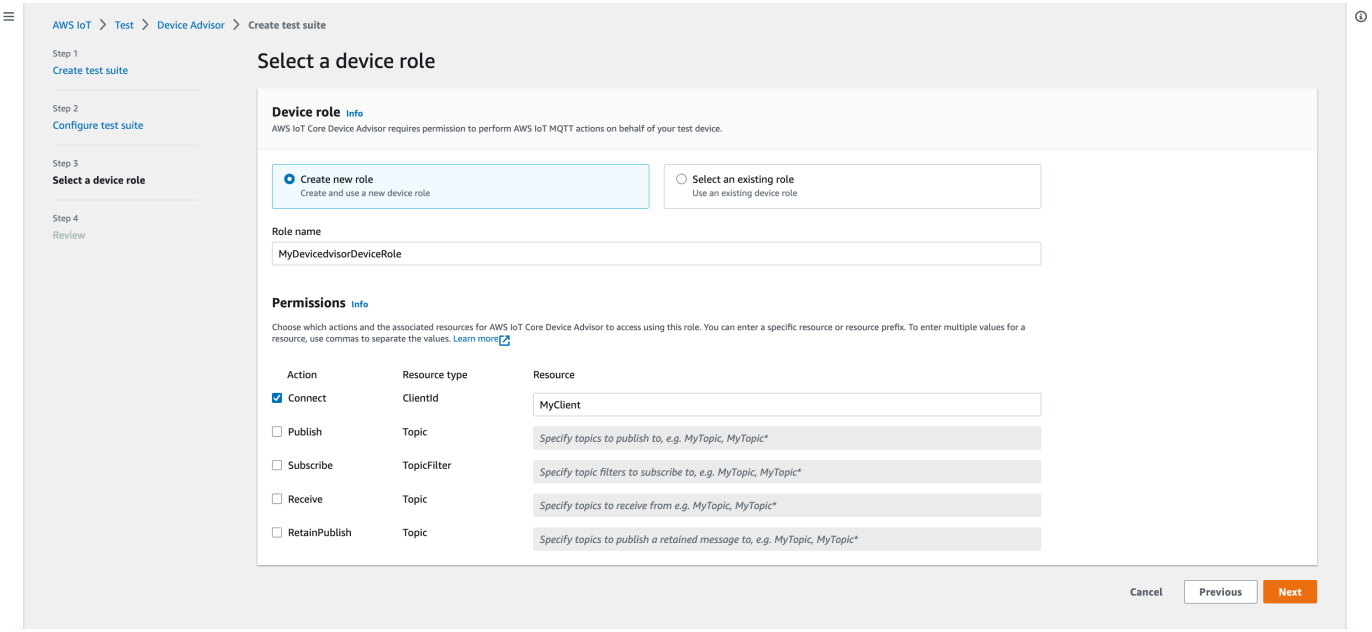
Wählen Sie Fertig aus.

> **Note**
>
> Um der Testsuite weitere Testgruppen hinzuzufügen, wählen Sie Testgruppe hinzufügen aus. Folgen Sie den vorherigen Schritten, um weitere Testgruppen zu erstellen und zu konfigurieren oder um einer oder mehreren Testgruppen weitere Testfälle hinzuzufügen. Testgruppen und Testfälle können neu angeordnet werden, indem Sie einen Testfall auswählen und an die gewünschte Position ziehen. Device Advisor führt Tests in der Reihenfolge aus, in der Sie die Testgruppen und Testfälle definieren.

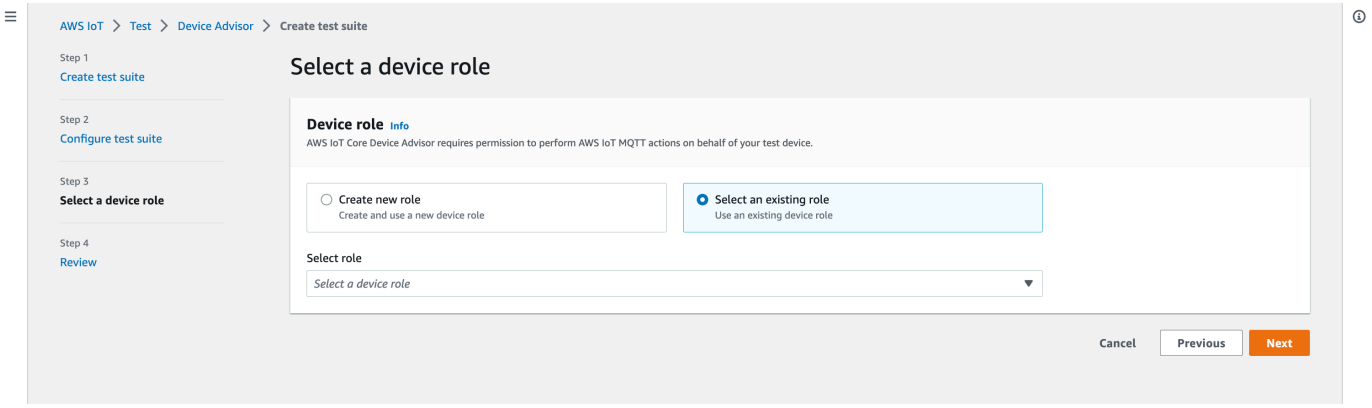
7. Wählen Sie Weiter aus.
8. Konfigurieren Sie in Schritt 3 eine Geräterolle, die Device Advisor verwendet, um AWS IoT MQTT-Aktionen im Namen Ihres Testgeräts auszuführen.

   Wenn Sie in Schritt 2 nur den MQTT-Connect-Testfall ausgewählt haben, wird die Connect-Aktion automatisch überprüft, da diese Berechtigung für die Geräterolle erforderlich ist, um diese Testsuite auszuführen. Wenn Sie andere Testfälle ausgewählt haben, werden die entsprechenden erforderlichen Aktionen überprüft. Stellen Sie sicher, dass die Ressourcenwerte für jede der Aktionen angegeben werden. Geben Sie beispielsweise für die Connect-Aktion die Client-ID an, mit der sich Ihr Gerät mit dem Device-Advisor-Endpunkt verbindet. Sie können mehrere Werte angeben, indem Sie die Werte durch Kommas trennen, und Sie können Präfixwerte auch mit einem Platzhalterzeichen (*) angeben. Um beispielsweise die Erlaubnis zur

Veröffentlichung zu einem beliebigen Thema zu erteilen, das mit `MyTopic` beginnt, können Sie „`MyTopic*`“ als Ressourcenwert eingeben.



Wenn Sie bereits zuvor eine Geräterolle erstellt haben und diese Rolle verwenden möchten, wählen Sie Eine vorhandene Rolle auswählen und anschließend unter Rolle auswählen Ihre Geräterolle aus.



Konfigurieren Sie Ihre Geräterolle mit einer der beiden verfügbaren Optionen und wählen Sie dann Weiter aus.

9. Stellen Sie in Schritt 4 sicher, dass die in jedem der Schritte angegebene Konfiguration korrekt ist. Um die für einen bestimmten Schritt bereitgestellte Konfiguration zu bearbeiten, wählen Sie Bearbeiten für den entsprechenden Schritt aus.

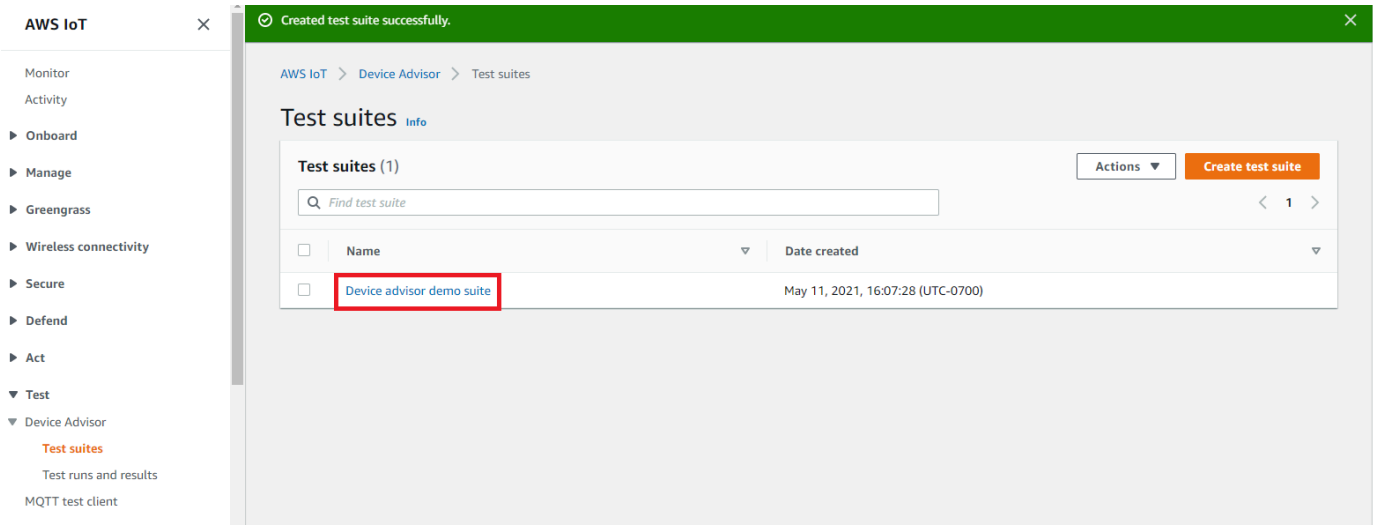
   Nachdem Sie die Konfiguration überprüft haben, wählen Sie Testsuite erstellen aus.

Die Testsuite sollte erfolgreich erstellt worden sein und Sie werden zur Seite Testsuites weitergeleitet, auf der Sie alle erstellten Testsuites anzeigen können.

Wenn die Erstellung der Testsuite fehlgeschlagen ist, stellen Sie sicher, dass die Testsuite, die Testgruppen, die Testfälle und die Geräterolle gemäß den vorherigen Anweisungen konfiguriert wurden.

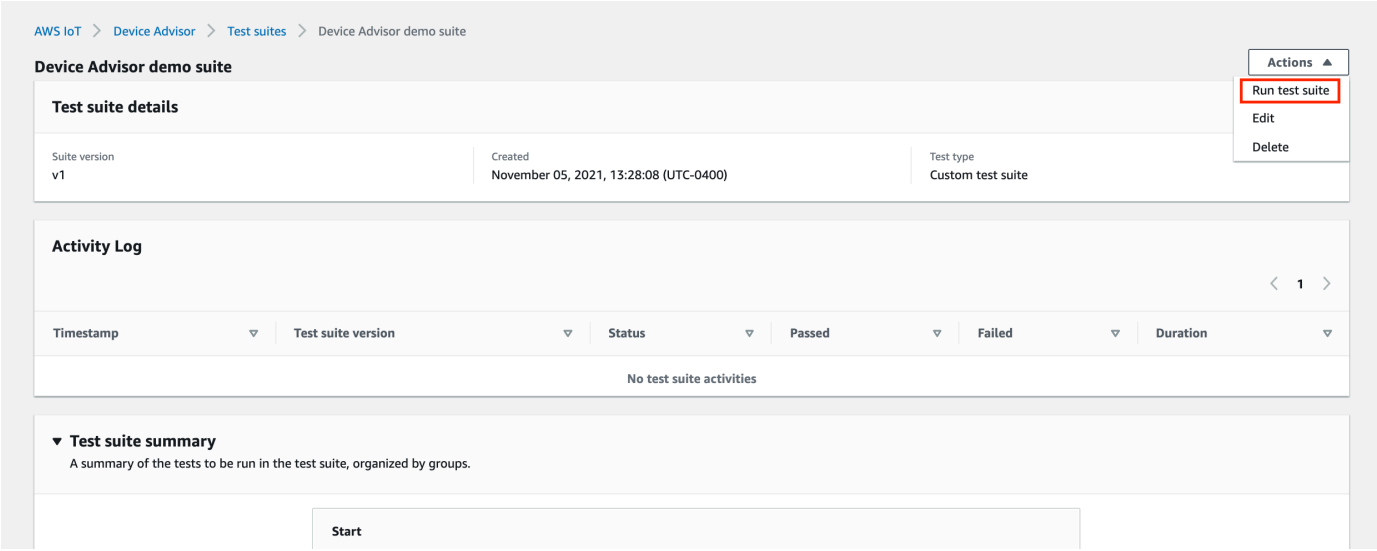
Ausführen einer Testsuite

1. Erweitern Sie im Navigationsbereich in der AWS IoT -Konsole die Optionen Test, Device Advisor und wählen Sie dann Testsuites aus.
2. Wählen Sie die Testsuite aus, für die Sie die Details der Testsuite anzeigen möchten.



Auf der Detailseite der Testsuite werden alle Informationen zur Testsuite angezeigt.

3. Wählen Sie Aktionen und dann Testsuite ausführen aus.

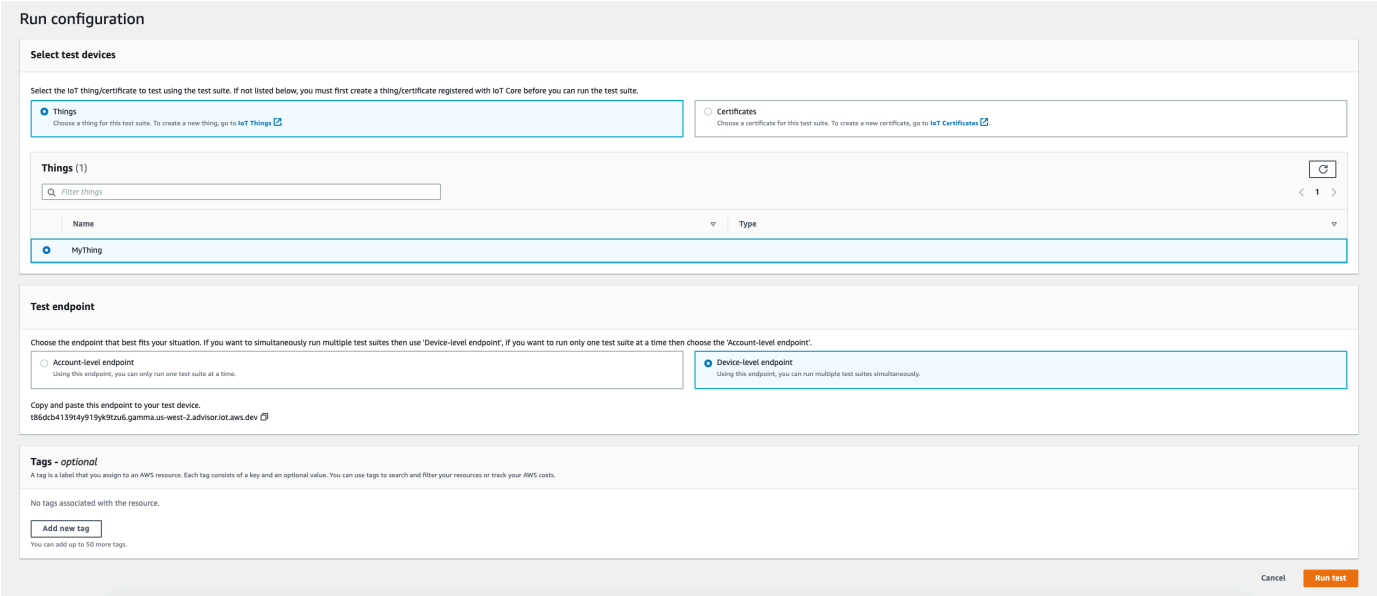


4. Unter Konfiguration ausführen müssen Sie eine AWS IoT Sache oder ein Zertifikat auswählen, das Sie mit Device Advisor testen möchten. Wenn Sie keine vorhandenen Dinge oder Zertifikate haben, erstellen Sie zunächst AWS IoT Core Ressourcen.

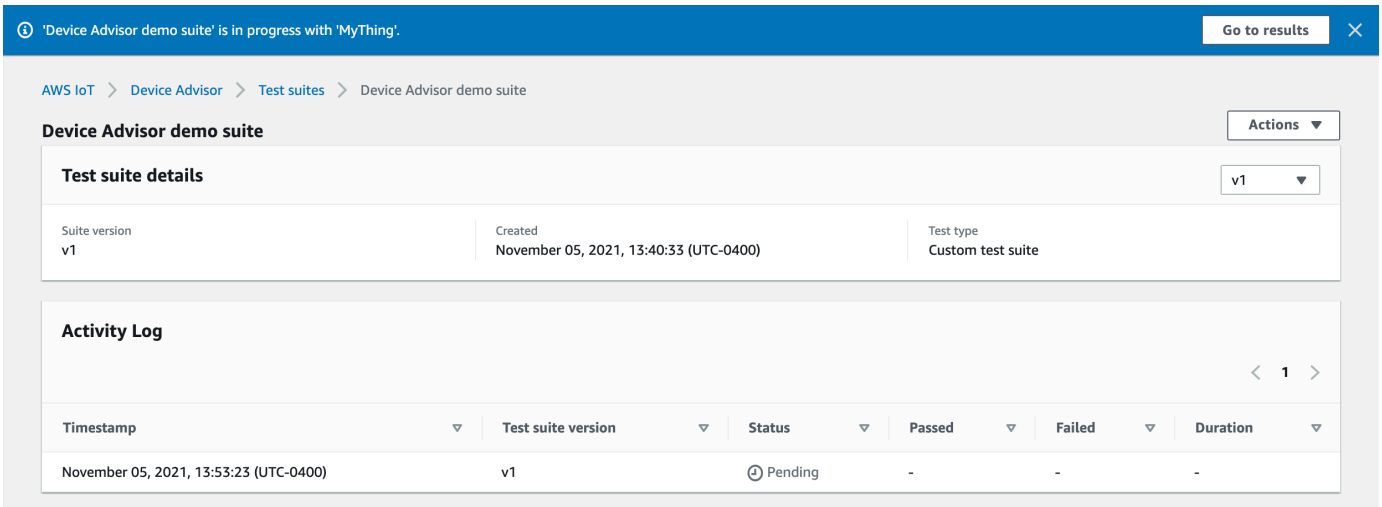
   Wählen Sie im Abschnitt Testendpunkt den Endpunkt aus, der am besten zu Ihrem Fall passt. Wenn Sie in Zukunft mehrere Testsuiten gleichzeitig mit demselben AWS Konto ausführen möchten, wählen Sie Device-level Endpunkt aus. Andernfalls, wenn Sie nur eine Testsuite gleichzeitig ausführen möchten, wählen Sie Account-level Endpunkt aus.

   Konfigurieren Sie Ihr Testgerät mit dem Testendpunkt des ausgewählten Device Advisors.

   Nachdem Sie ein Ding oder ein Zertifikat und einen Device-Advisor-Endpunkt ausgewählt haben, wählen Sie Test ausführen aus.

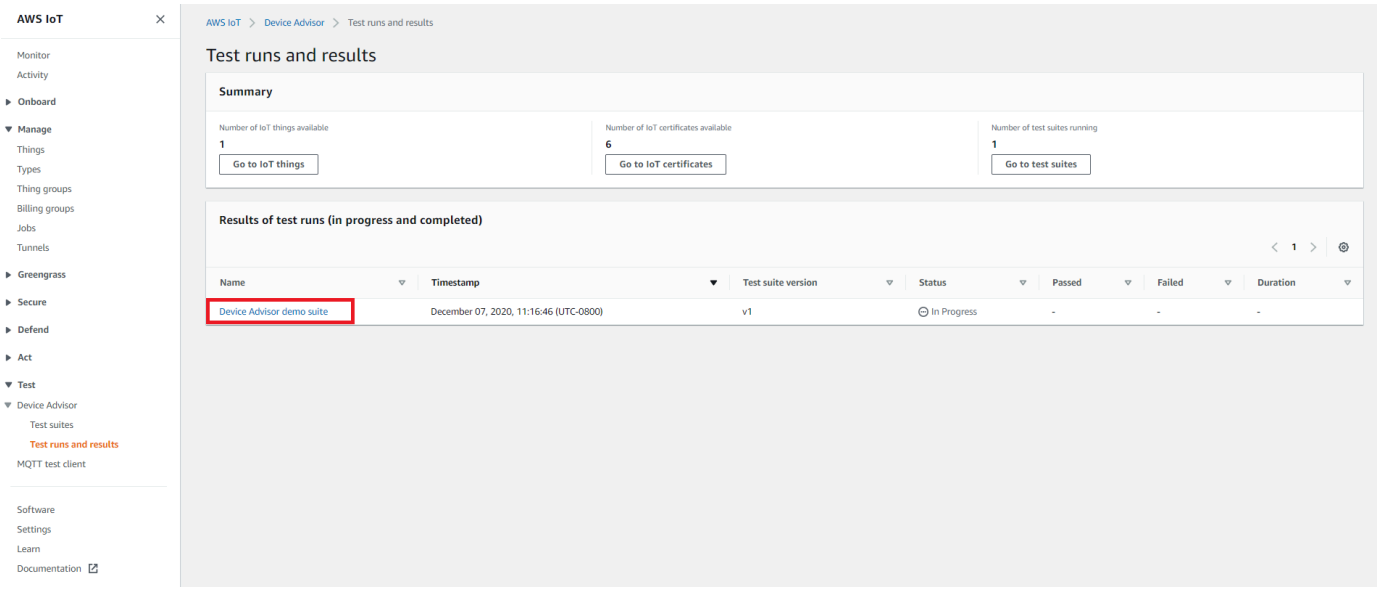


5. Wählen Sie im oberen Banner die Option Gehe zu den Ergebnissen, um die Details des Testlaufs anzuzeigen.

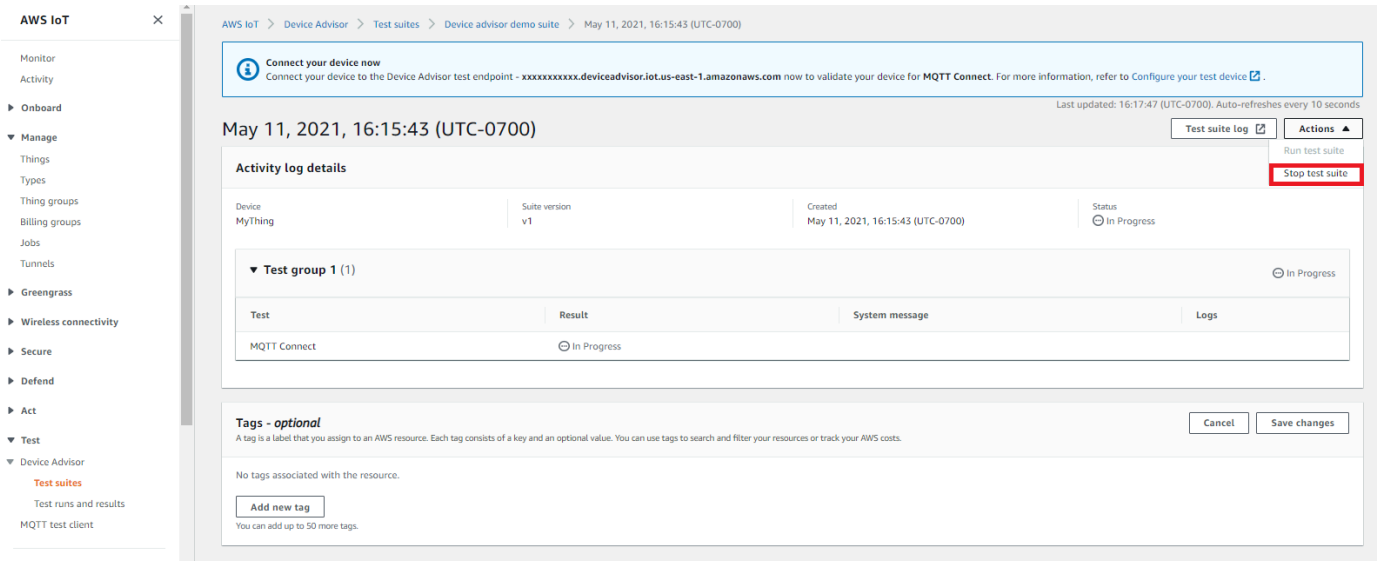


Beenden einer Testsuite-Ausführung (optional)

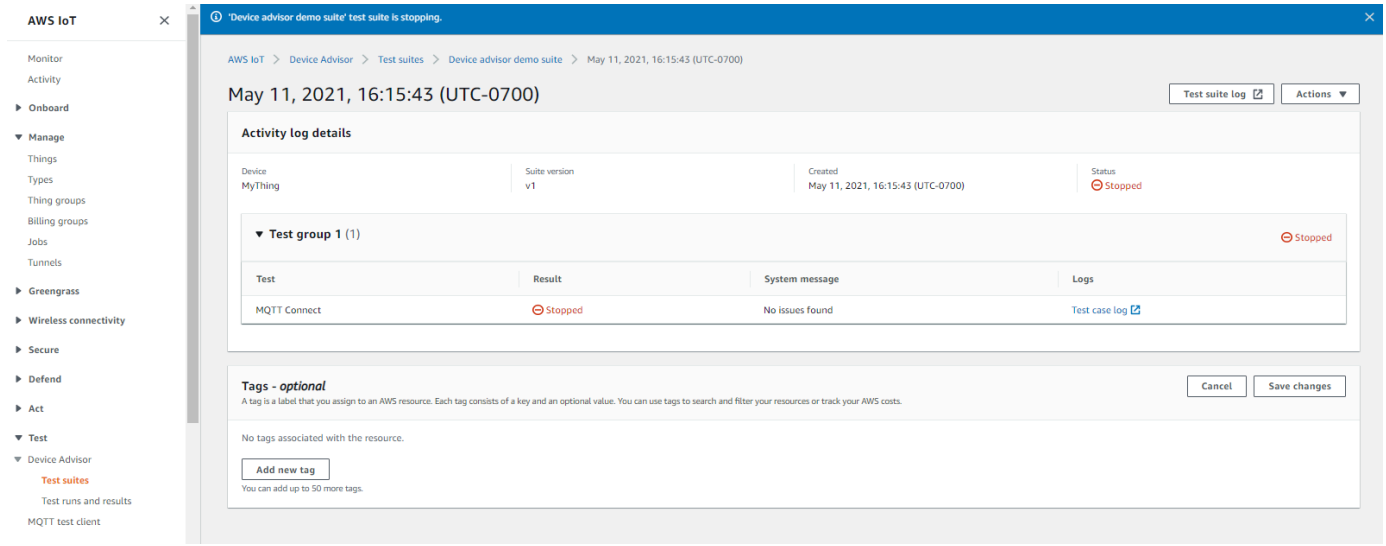
1. Erweitern Sie im Navigationsbereich in der AWS IoT -Konsole die Optionen Test, Device Advisor und wählen Sie dann Testläufe und Ergebnisse aus.
2. Wählen Sie die laufende Testsuite, die Sie beenden möchten.



3. Wählen Sie Aktionen und dann Testsuite beenden aus.



4. Der Bereinigungsvorgang nimmt einige Minuten in Anspruch. Während der Bereinigungsvorgang läuft, ist der Status des Testlaufs `STOPPING`. Warten Sie, bis der Bereinigungsprozess abgeschlossen ist und sich der Status der Testsuite in den `STOPPED`-Status ändert, bevor Sie eine neue Suite ausführen.

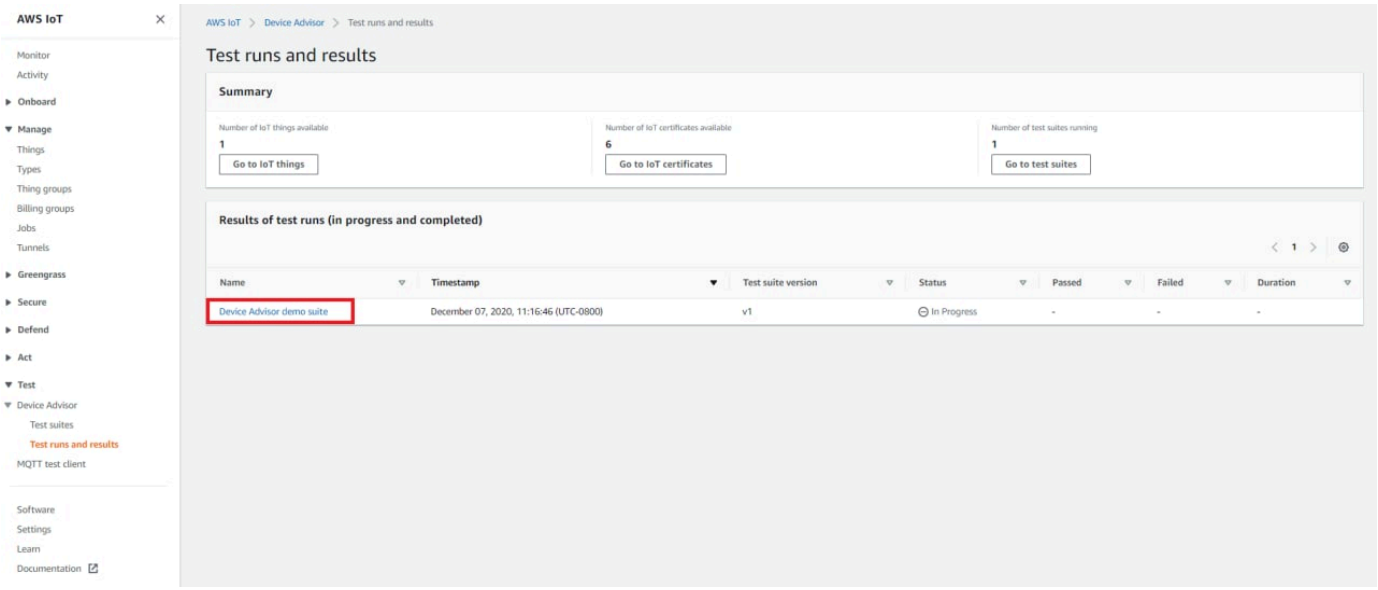


Anzeigen von Details und Protokolle der Testsuite-Ausführung

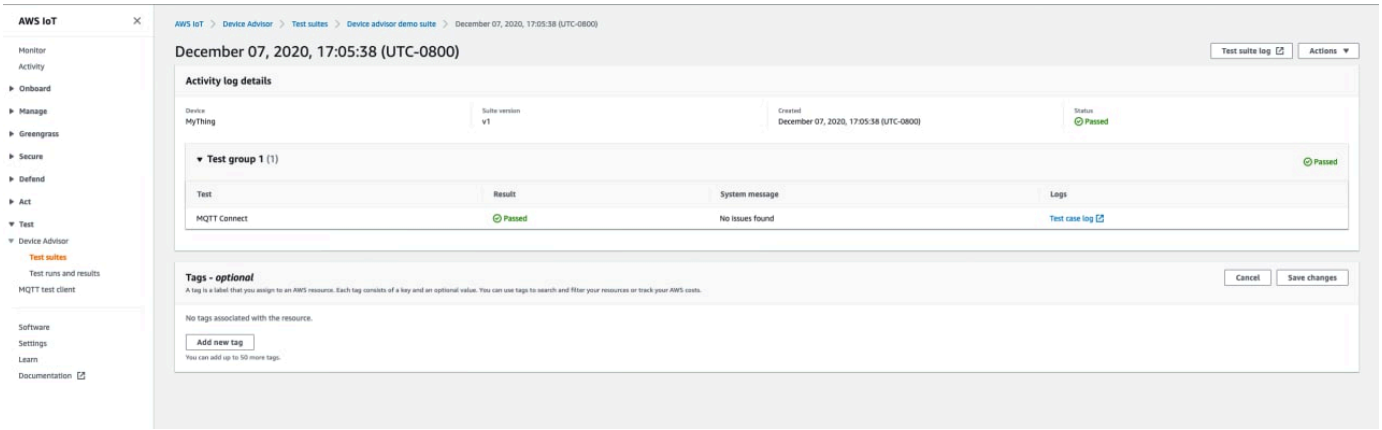
1. Erweitern Sie im Navigationsbereich in der AWS IoT -Konsole die Optionen Test, Device Advisor und wählen Sie dann Testläufe und Ergebnisse aus.

   Diese Seite zeigt Folgendes an:

   - Anzahl der IoT-Dinge
   - Anzahl der IoT-Zertifikate
   - Anzahl der derzeit ausgeführten Testsuites
   - Alle Testsuite-Ausführungen, die erstellt wurden
2. Wählen Sie die Testsuite aus, für die Sie die Details der Ausführung und die Protokolle anzeigen möchten.



Auf der Seite mit der Zusammenfassung der Testsuite wird der Status der aktuellen Testsuite-Ausführung angezeigt. Diese Seite wird automatisch alle 10 Sekunden aktualisiert. Wir empfehlen, dass Sie für Ihr Gerät einen Mechanismus entwickeln, mit dem Sie alle fünf Sekunden ein bis zwei Minuten lang versuchen können, eine Verbindung zu unserem Testendpunkt herzustellen. Sie können dann mehrere Testfälle nacheinander automatisiert ausführen.



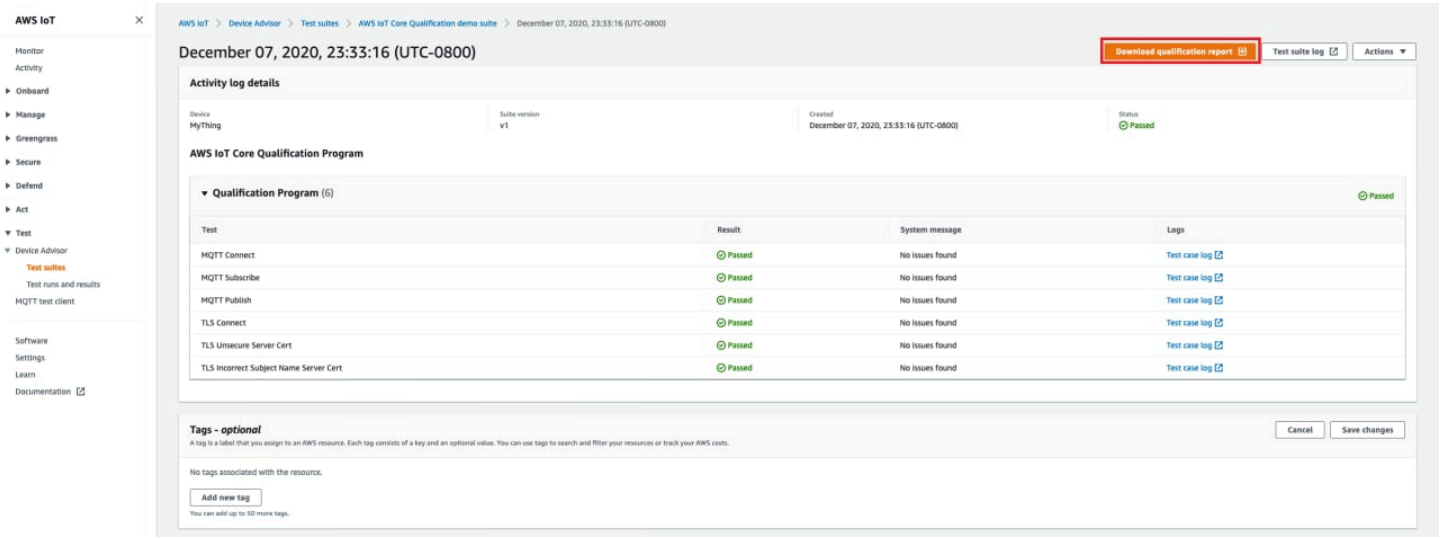
3. Um auf die CloudWatch Protokolle für den Testsuite-Lauf zuzugreifen, wählen Sie Test Suite Log.

   Um auf CloudWatch Protokolle für einen beliebigen Testfall zuzugreifen, wählen Sie Testfallprotokoll.

4. Führen Sie auf der Grundlage Ihrer Testergebnisse eine Fehlerbehebung auf Ihrem Gerät durch, bis alle Tests bestanden sind.

Laden Sie eine herunter AWS IoT Qualifizierungsbericht

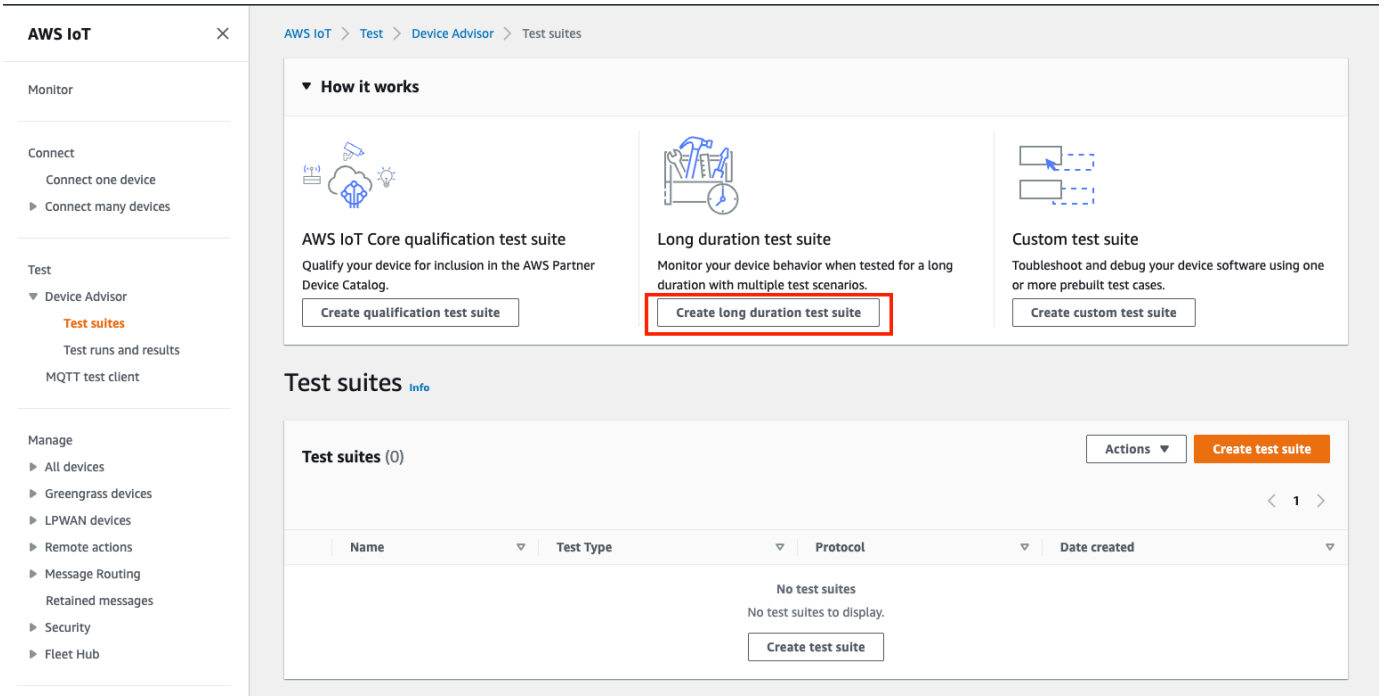
Wenn Sie beim Erstellen einer Testsuite die Option AWS IoT Qualifikationstestsuite verwenden gewählt haben und eine Qualifikationstestsuite ausführen konnten, können Sie einen Qualifikationsbericht herunterladen, indem Sie auf der Übersichtsseite des Testlaufs auf Qualifikationsbericht herunterladen klicken.



Konsolen-Workflow für Tests mit langer Dauer

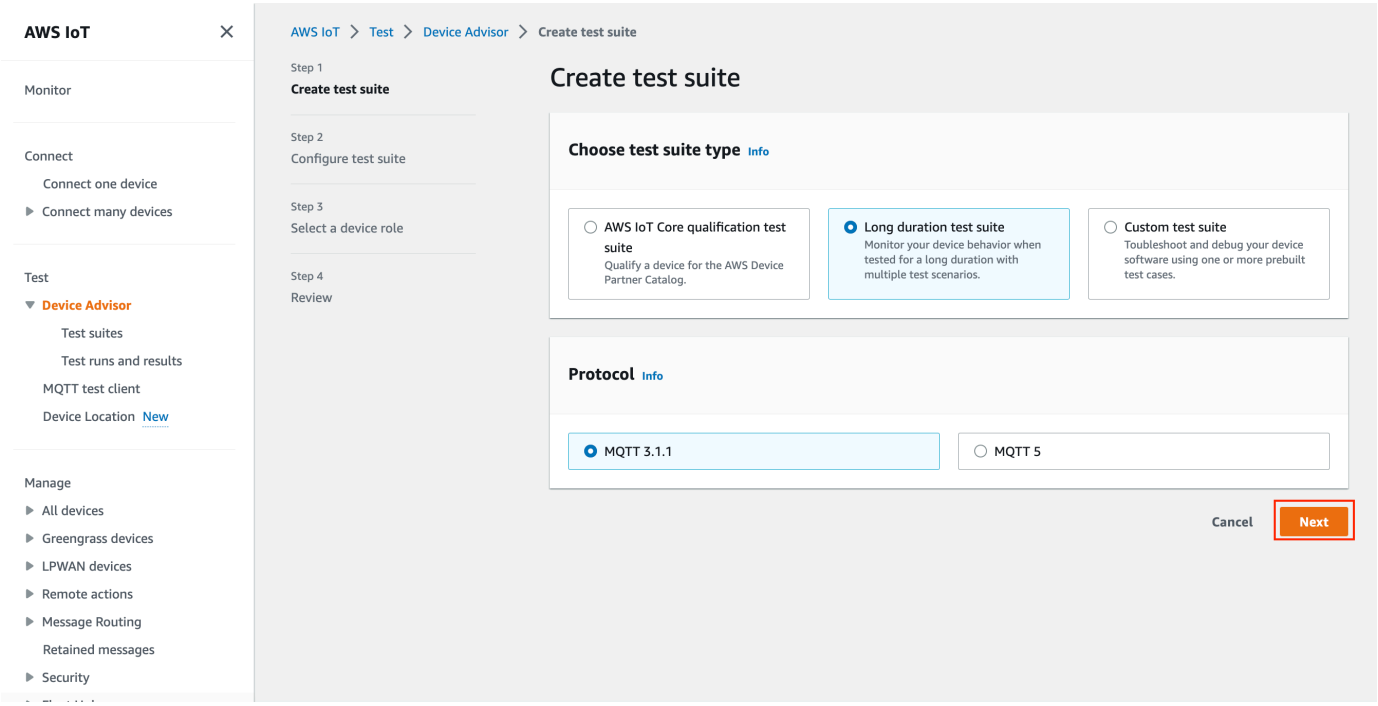
Dieses Tutorial hilft Ihnen bei den ersten Schritten mit der Durchführung von Tests mit langer Dauer auf Device Advisor mithilfe der Konsole. Folgen Sie den Schritten unter Einrichtung, um das Tutorial abzuschließen.

1. Erweitern Sie im Navigationsbereich in der AWS IoT -Konsole die Optionen Test, dann Device Advisor und dann Testsuites. Wählen Sie auf der Seite die Option Create long duration test suite (Testsuite mit langer Dauer erstellen) aus.



2. Wählen Sie auf der Seite Testsuite erstellen die Option Testsuite mit langer Dauer und dann Weiter aus.

   Wählen Sie als Protokoll entweder MQTT 3.1.1 oder MQTT 5 aus.



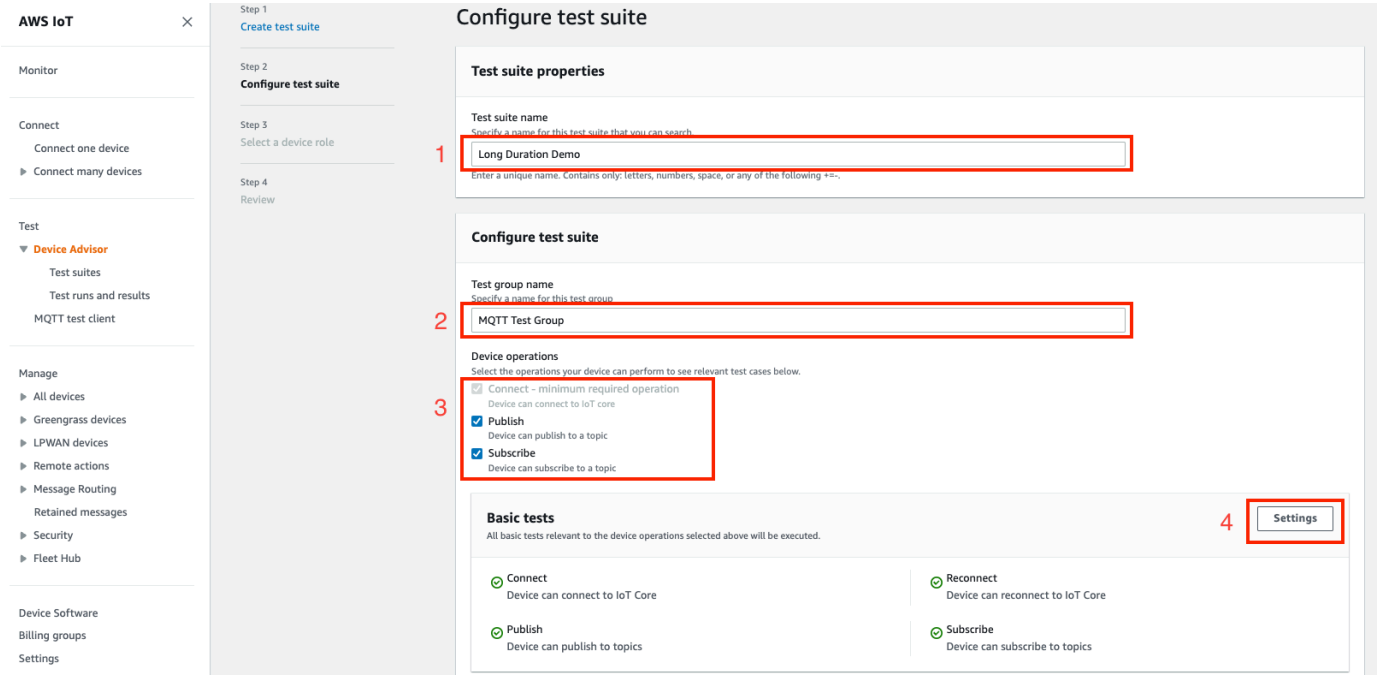
3. Führen Sie auf der Seite Testsuite konfigurieren die folgenden Schritte aus:

   a. Aktualisieren Sie das Feld Name der Testsuite.

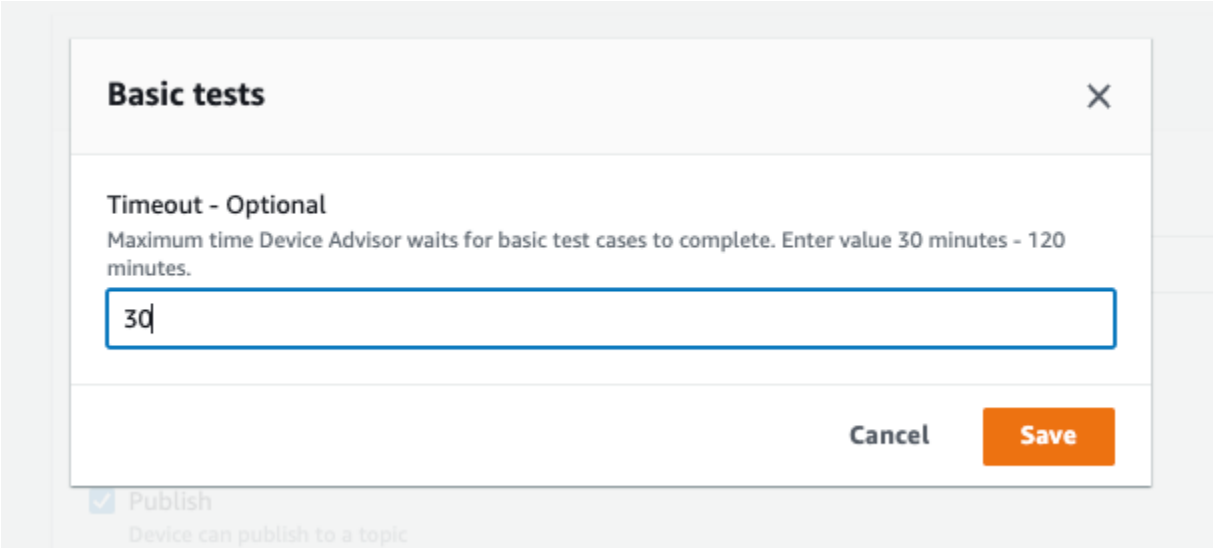
b. Aktualisieren Sie das Feld Name der Testgruppe.

c. Wählen Sie die Geräteoperationen aus, die das Gerät ausführen kann. Dadurch werden die auszuführenden Tests ausgewählt.

d. Wählen Sie die Option Einstellungen aus.



4. (Optional) Geben Sie die maximale Zeit ein, die Device Advisor warten muss, bis die Basistests abgeschlossen sind. Wählen Sie Speichern aus.



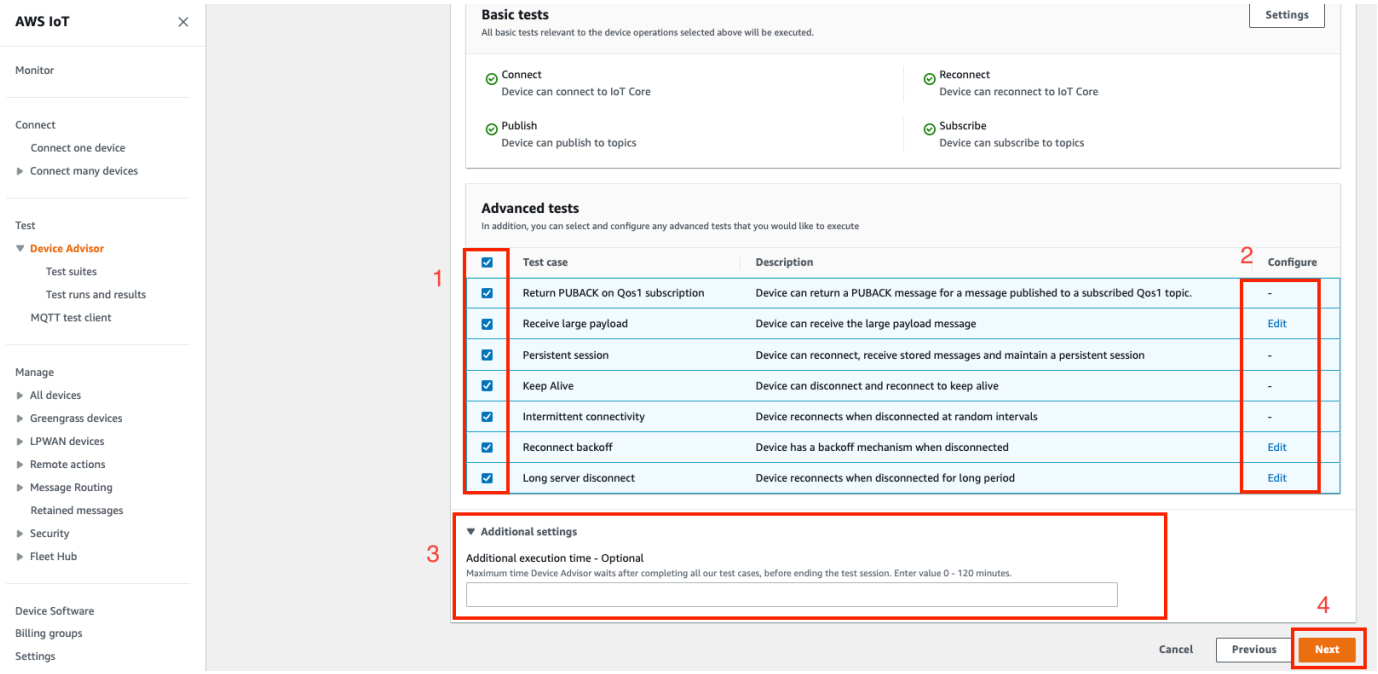
5. Gehen Sie in den Abschnitten Advanced tests (Erweiterte Tests) und Zusätzliche Einstellungen wie folgt vor.

a. Wählen Sie die erweiterten Tests aus, die Sie im Rahmen dieses Tests ausführen möchten, bzw. heben Sie die Auswahl auf.

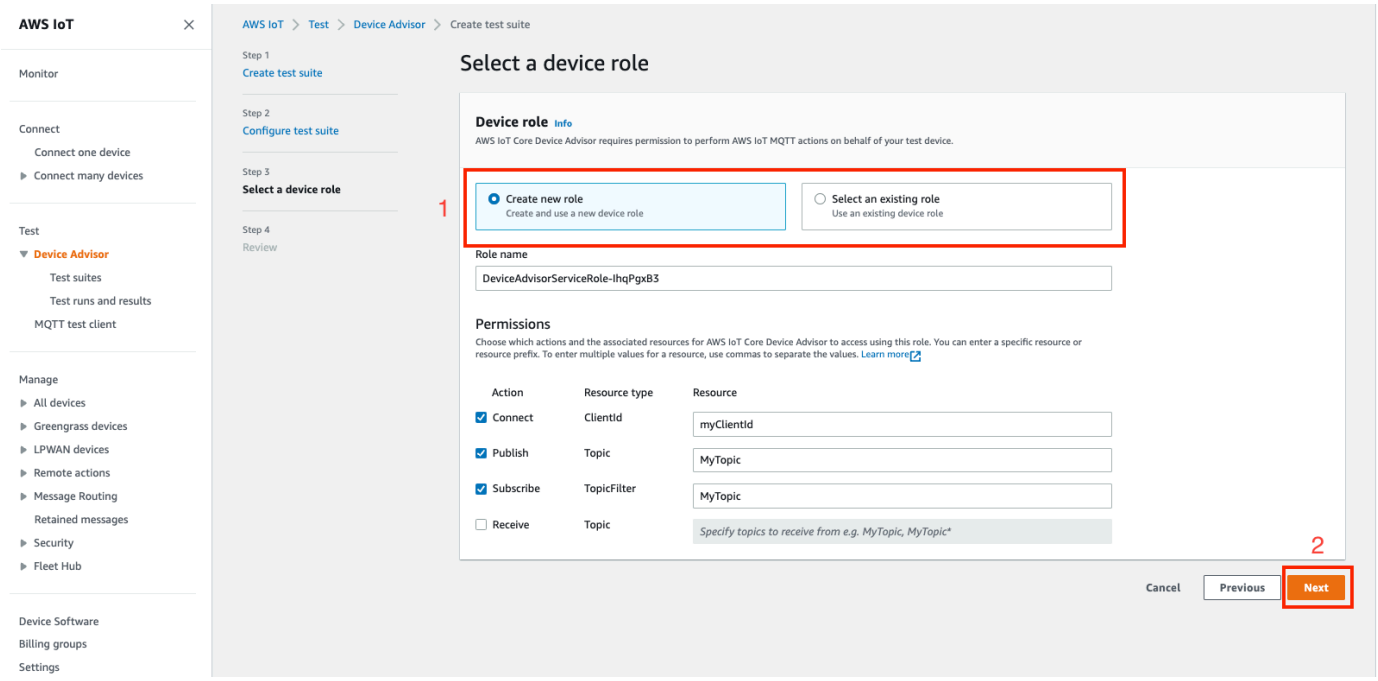
b. Bearbeiten Sie gegebenenfalls die Konfigurationen für die Tests.

c. Konfigurieren Sie die zusätzliche Ausführungszeit im Abschnitt Zusätzliche Einstellungen.

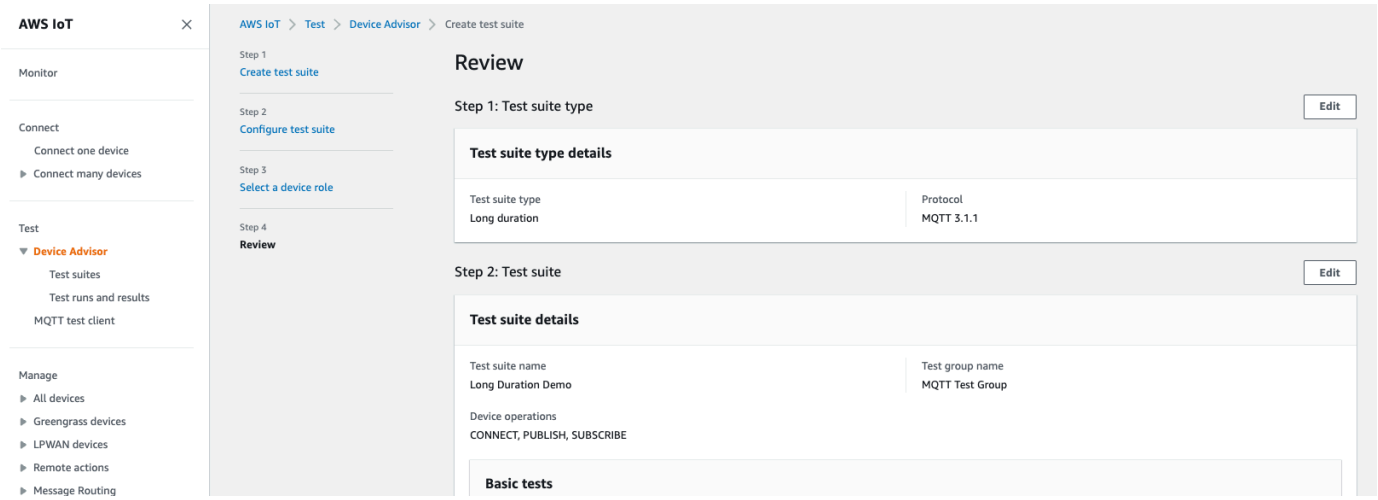
d. Wählen Sie Weiter aus, um mit dem nächsten Schritt fortzufahren.

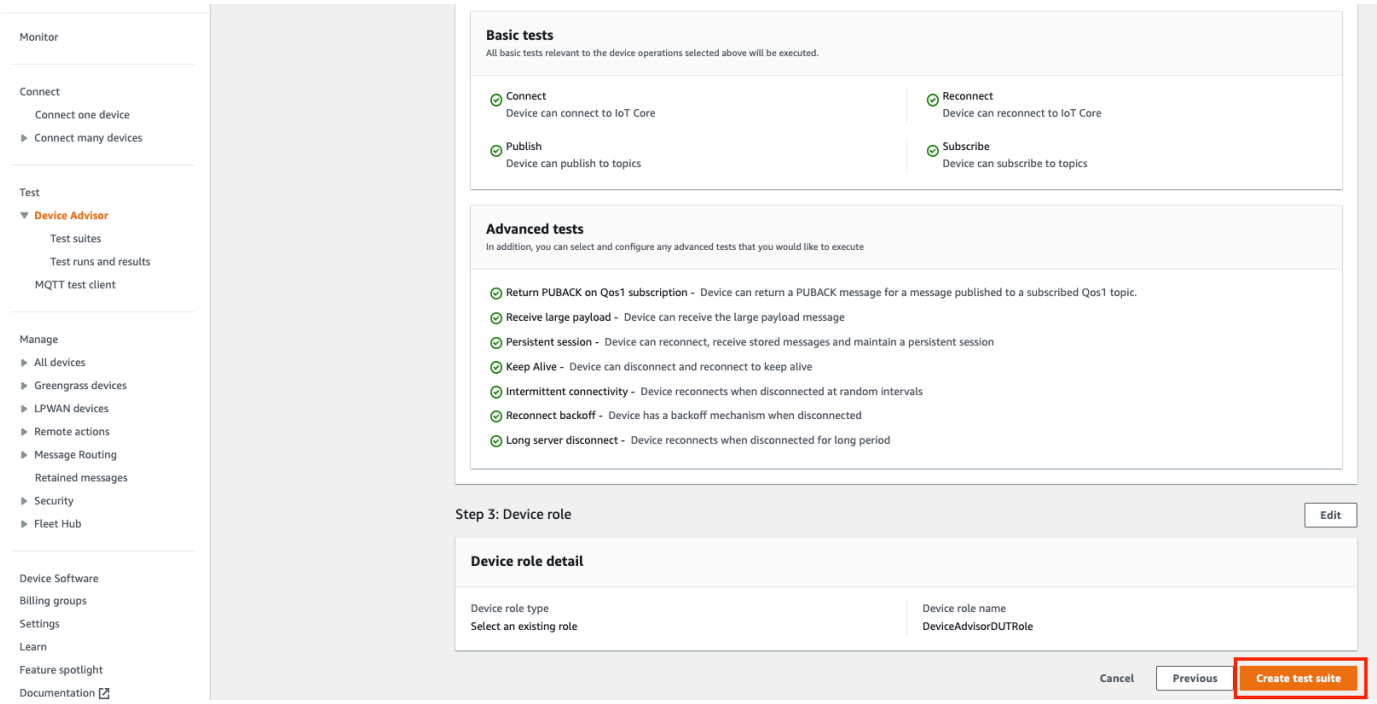


6. Erstellen Sie in diesem Schritt eine neue Rolle oder wählen Sie eine vorhandene Rolle aus. Details dazu finden Sie unter Erstellen einer IAM-Rolle, die Sie als Ihre Geräterolle verwenden möchten.

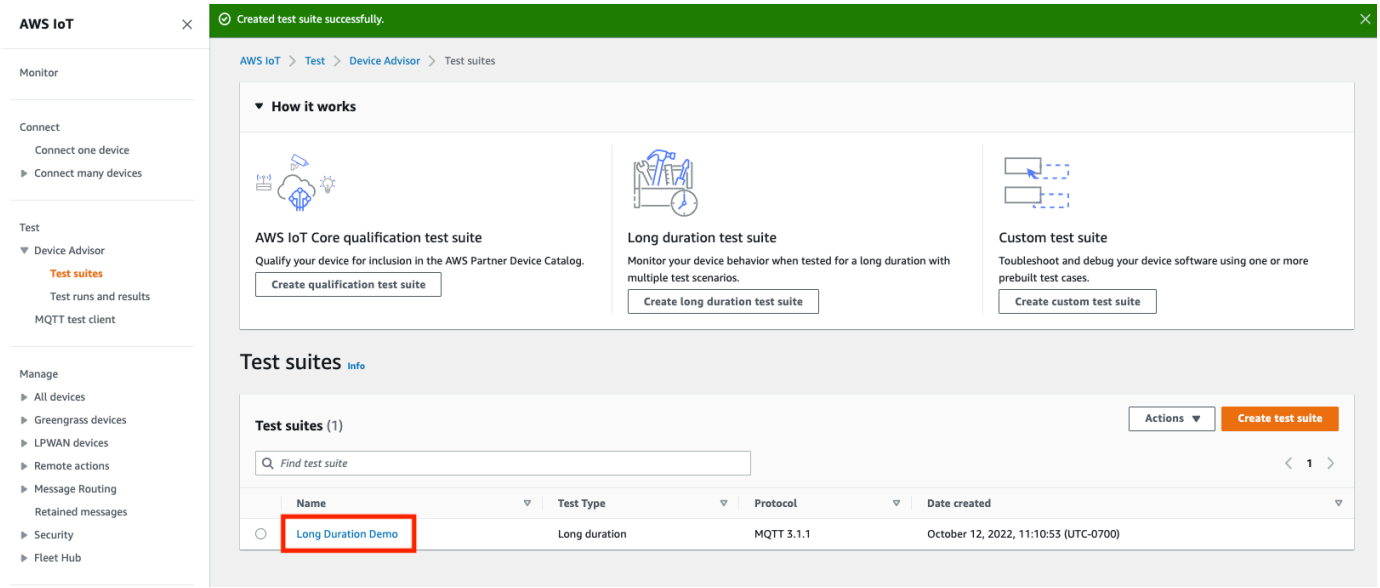


7. Überprüfen Sie alle bis zu diesem Schritt erstellten Konfigurationen und wählen Sie Testsuite erstellen aus.

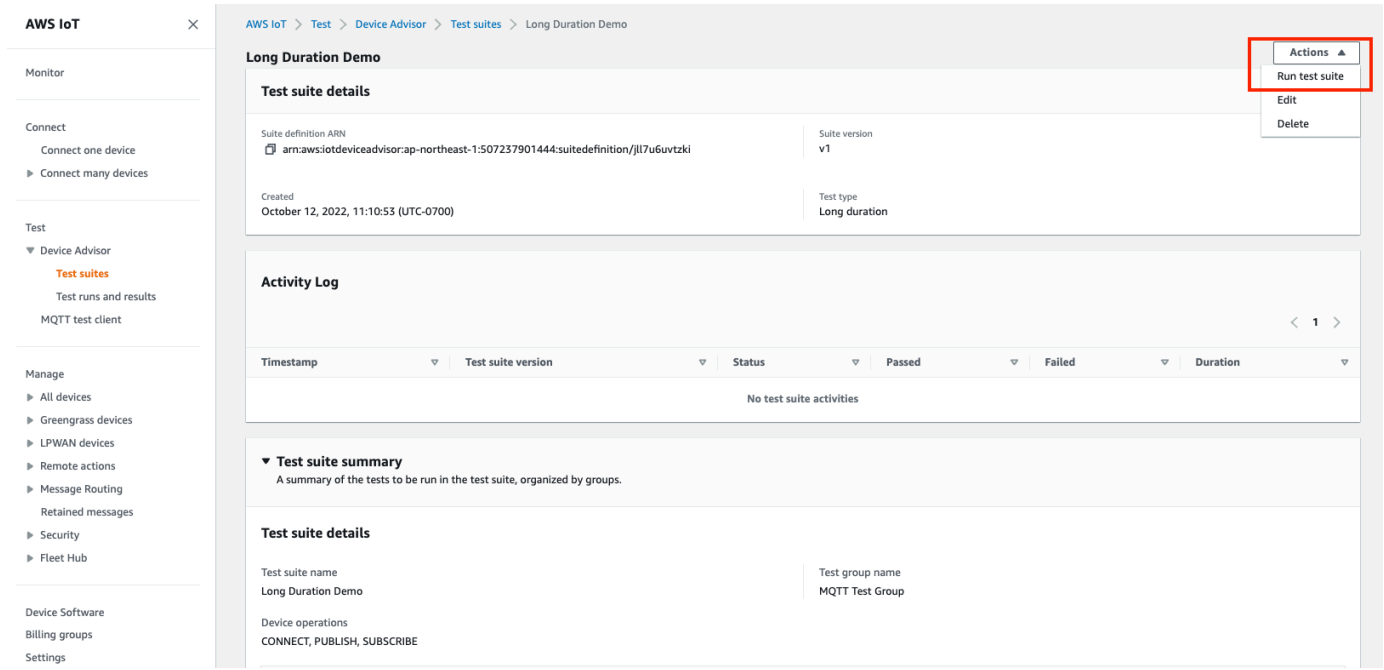




8. Die erstellte Testsuite befindet sich im Abschnitt Testsuites. Wählen Sie die Suite aus, um Details dazu anzuzeigen.



9. Um die erstellte Testsuite auszuführen, wählen Sie Aktionen und dann Testsuite ausführen aus.

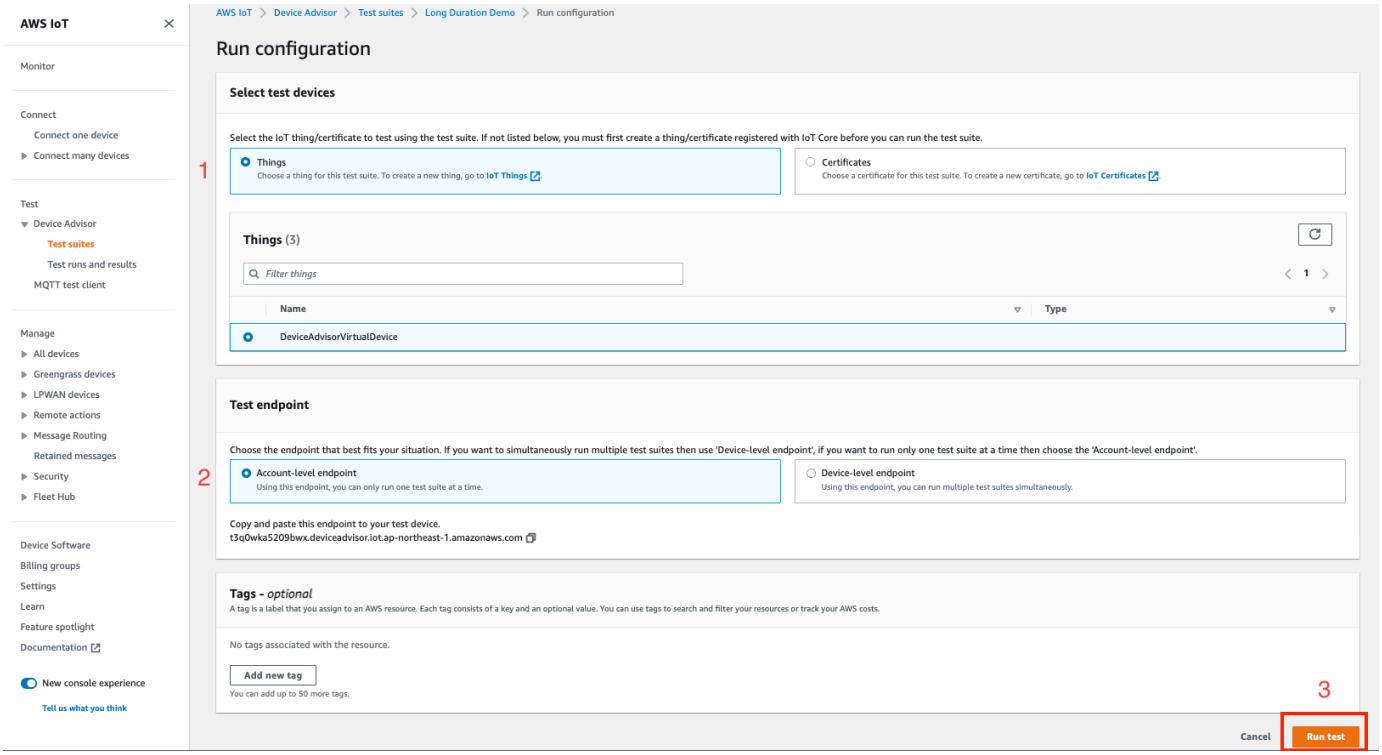


10. Wählen Sie die Konfigurationsoptionen auf der Seite Konfiguration ausführen aus.

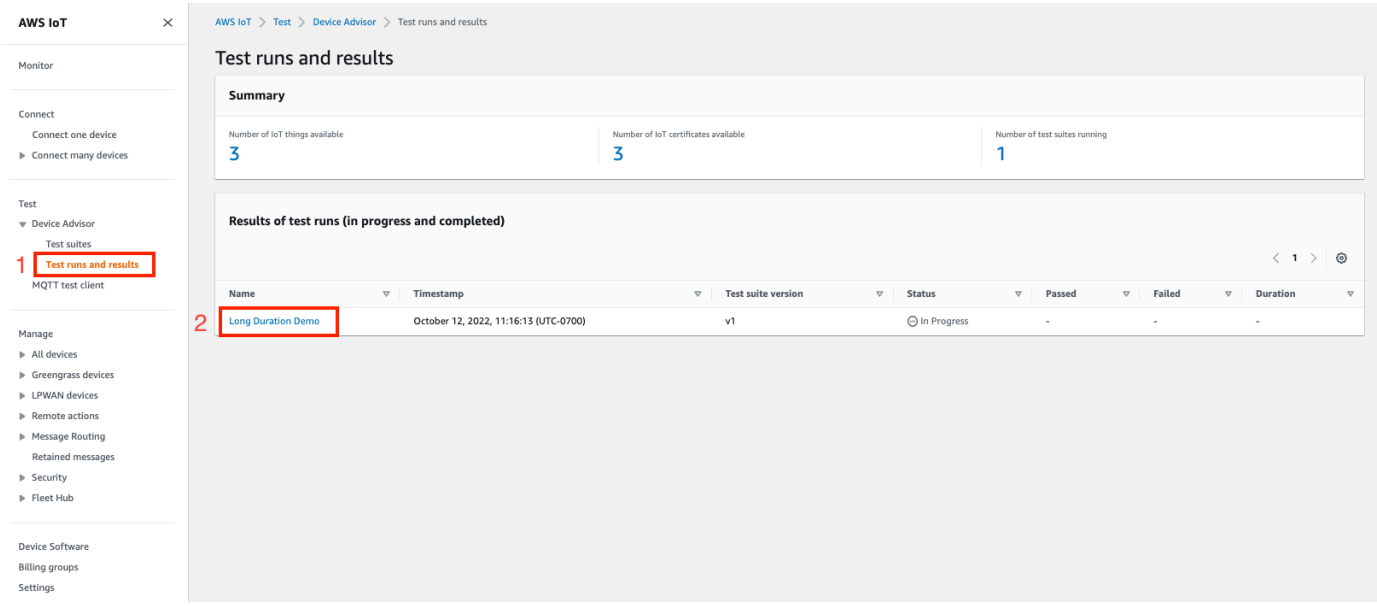
    a. Wählen Sie die Dinge oder das Zertifikat aus, für das der Test ausgeführt werden soll.

    b. Wählen Sie entweder den Account-level Endpunkt oder den Device-level Endpunkt aus.

    c. Wählen Sie zum Ausführen des Tests die Option Test ausführen aus.

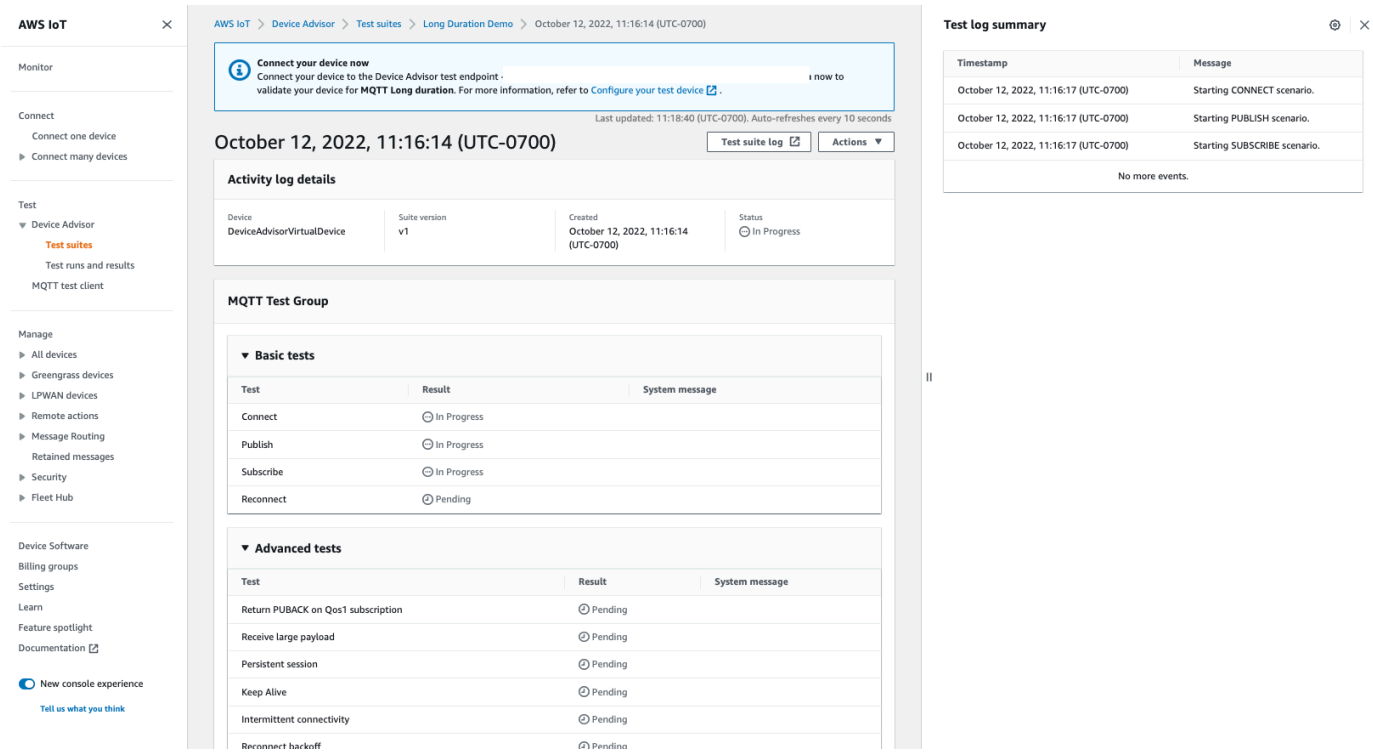


11. Um die Ergebnisse der Testsuite-Ausführung anzuzeigen, wählen Sie im linken Navigationsbereich Testläufe und Ergebnisse aus. Wählen Sie die Testsuite aus, die ausgeführt wurde, um die Details der Ergebnisse anzuzeigen.

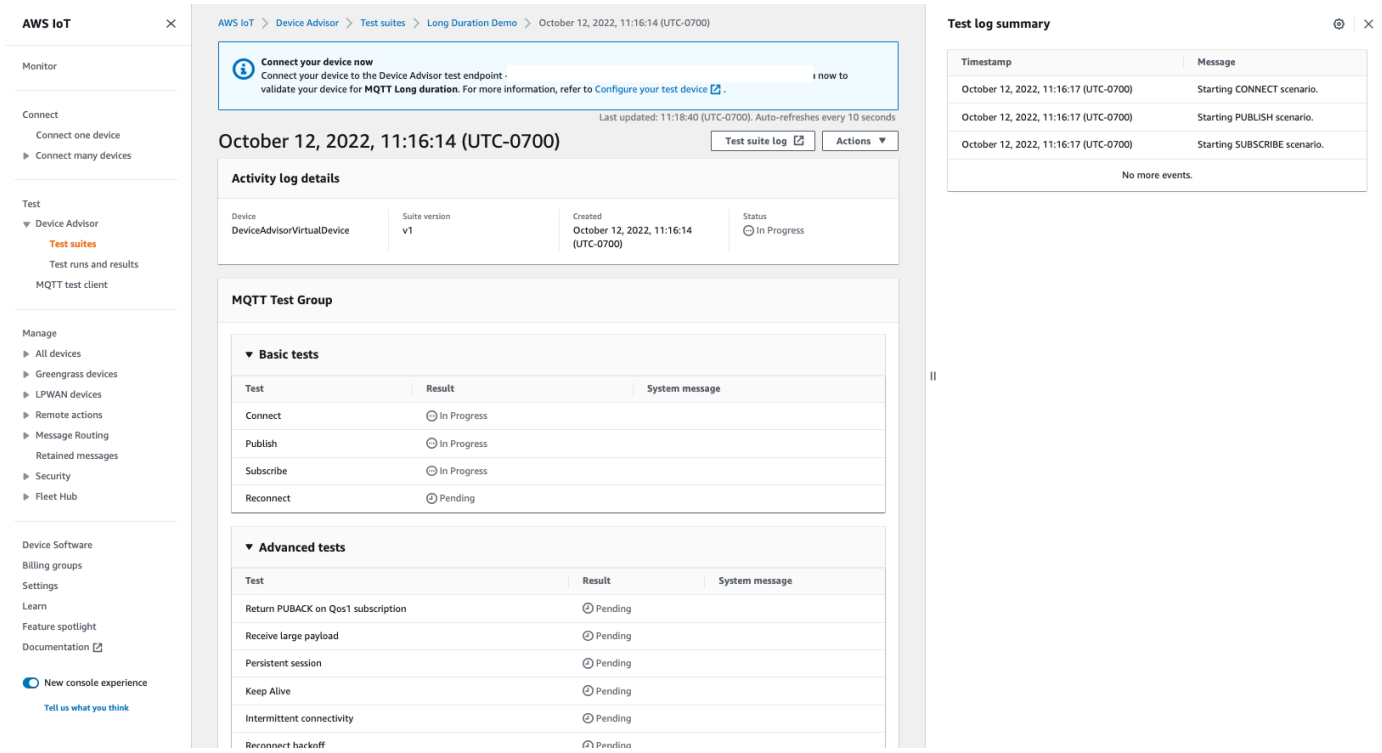


12. Im vorherigen Schritt wird die Seite mit der Testzusammenfassung aufgerufen. Alle Details des Testlaufs werden auf dieser Seite angezeigt. Wenn Sie in der Konsole aufgefordert werden, die

Geräteverbindung herzustellen, verbinden Sie Ihr Gerät mit dem angegebenen Endpunkt. Der Fortschritt der Tests ist auf dieser Seite zu sehen.



13. Der Test mit langer Dauer bietet eine zusätzliche Zusammenfassung des Testprotokolls im Seitenbereich, in der alle wichtigen Ereignisse zwischen dem Gerät und dem Broker nahezu in Echtzeit angezeigt werden. Um detailliertere Protokolle anzuzeigen, klicken Sie auf Testfallprotokoll.



Device Advisor VPC-Endpunkte (AWS PrivateLink)

Sie können eine private Verbindung zwischen Ihrer VPC und dem AWS IoT Core Device Advisor Testendpunkt (Datenebene) herstellen, indem Sie einen Schnittstellen-VPC-Endpunkt erstellen. Sie können diesen Endpunkt verwenden, um AWS IoT Geräte auf zuverlässige und sichere Konnektivität zu überprüfen, AWS IoT Core bevor Sie Geräte in der Produktion einsetzen. Die vorgefertigten Tests von Device Advisor helfen Ihnen dabei, Ihre Gerätesoftware anhand von Best Practices für die Verwendung von TLS-, MQTT-, Device Shadow- und AWS IoT -Aufträgen zu validieren.

AWS PrivateLinkversorgt die Schnittstellen-Endpunkte, die mit Ihren IoT-Geräten verwendet werden. Über diesen Service können Sie privat auf den AWS IoT Core Device Advisor -Testendpunkt zugreifen, ohne Internet-Gateway, NAT-Gerät, VPN-Verbindung oder Direct Connect -Verbindung. Instances in Ihrer VPC, die TCP- und MQTT-Pakete senden, benötigen keine öffentlichen IP-Adressen, um mit AWS IoT Core Device Advisor Testendpunkten zu kommunizieren. Der Verkehr zwischen Ihrer VPC und AWS IoT Core Device Advisor geht nicht weg. AWS Cloud Jegliche TLS- und MQTT-Kommunikation zwischen IoT-Geräten und Device-Advisor-Testfällen bleibt innerhalb der Ressourcen in Ihrem AWS-Konto.

Jeder Schnittstellenendpunkt wird durch eine oder mehrere Elastic Network-Schnittstellen in Ihren Subnetzen dargestellt.

Weitere Informationen zur Verwendung von Schnittstelllen-VPC-Endpunkten finden Sie unter Schnittstellen-VPC-Endpunkte (AWS PrivateLink) im Amazon-VPC-Benutzerhandbuch.

Überlegungen für AWS IoT Core Device Advisor VPC-Endpunkte

Bevor Sie Schnittstellen-VPC-Endpunkte einrichten, lesen Sie über die Eigenschaften und Einschränkungen von Schnittstellenendpunkten im Amazon-VPC-Benutzerhandbuch nach. Beachten Sie Folgendes, bevor Sie fortfahren:

- AWS IoT Core Device Advisor unterstützt derzeit Anrufe an den Device Advisor-Testendpunkt (Datenebene) von Ihrer VPC aus. Ein Message Broker verwendet Kommunikation auf Datenebene, um Daten zu senden und zu empfangen. Dies geschieht mithilfe von TLS- und MQTT-Paketen. VPC-Endpunkte zum AWS IoT Core Device Advisor Verbinden Ihres AWS IoT Geräts mit Device Advisor-Testendpunkten. API-Aktionen der Steuerebene werden von diesem VPC-Endpunkt nicht verwendet. Verwenden Sie die Konsole, ein AWS SDK oder eine AWS Befehlszeilenschnittstelle über das öffentliche Internet, um eine Testsuite oder andere APIs für die Steuerungsebene zu erstellen oder auszuführen.
- Die folgenden AWS-Regionen unterstützen VPC-Endpunkte für: AWS IoT Core Device Advisor
  - USA Ost (Nord-Virginia)
  - USA West (Oregon)
  - Asien-Pazifik (Tokio)
  - Europa (Irland)
- Device Advisor unterstützt MQTT mit X.509 Client-Zertifikaten und RSA-Serverzertifikaten.
- VPC-Endpunktrichtlinien werden derzeit nicht unterstützt.
- Anweisungen zum Erstellen von Ressourcen, die VPC-Endpunkte verbinden, finden Sie unter Voraussetzungen für VPC-Endpunkte. Sie müssen eine VPC und private Subnetze erstellen, um VPC-Endpunkte verwenden zu können. AWS IoT Core Device Advisor
- Es gibt Kontingente für Ihre Ressourcen. AWS PrivateLink Weitere Informationen finden Sie unter AWS PrivateLink -Kontingente.
- VPC-Endpunkte unterstützen nur IPv4-Datenverkehr.

Erstellen Sie einen Schnittstellen-VPC-Endpunkt für AWS IoT Core Device Advisor

Erstellen Sie einen Schnittstellen-VPC-Endpunkt, um erste Schritte mit VPC-Endpunkten zu unternehmen. Wählen Sie AWS IoT Core Device Advisor als Nächstes als. AWS-Service Wenn Sie das verwenden AWS CLI, rufen Sie describe-vpc-endpoint-services auf, um zu bestätigen, dass es sich in einer Availability Zone in Ihrem AWS IoT Core Device Advisor befindet. AWS-Region Vergewissern Sie sich, dass die dem Endpunkt zugeordnete Sicherheitsgruppe die TCP-Protokollkommunikation für MQTT- und TLS-Datenverkehr zulässt. Verwenden Sie zum Beispiel in der Region USA Ost (Nord-Virginia) den folgenden Befehl:

```
aws ec2 describe-vpc-endpoint-services --service-name com.amazonaws.us-
east-1.deviceadvisor.iot
```

Sie können einen VPC-Endpunkt für die Verwendung des folgenden Dienstnamens erstellen: AWS IoT Core

- com.amazonaws.region.deviceadvisor.iot

Standardmäßig ist privates DNS für den Endpunkt aktiviert. Dadurch wird sichergestellt, dass der Standard-Testendpunkt weiterhin in Ihren privaten Subnetzen verwendet wird. Verwenden Sie die Konsole oder ein AWS SDK, um Ihren Endpunkt auf Konto AWS CLI - oder Geräteebene abzurufen. Wenn Sie beispielsweise get-endpoint in einem öffentlichen Subnetz oder im öffentlichen Internet ausführen, können Sie Ihren Endpunkt abrufen und ihn verwenden, um eine Verbindung zu Device Advisor herzustellen. Weitere Informationen finden Sie unter Zugriff auf einen Service über einen Schnittstellenendpunkt im Benutzerhandbuch für Amazon VPC.

Um MQTT-Clients mit den VPC-Endpunktschnittstellen zu verbinden, erstellt der AWS PrivateLink Dienst DNS-Einträge in einer privaten gehosteten Zone, die an Ihre VPC angehängt ist. Diese DNS-Einträge leiten die Anforderungen des AWS IoT -Geräts an den VPC-Endpunkt weiter.

Steuerung des Zugriffs auf AWS IoT Core Device Advisor über VPC-Endpunkte

Sie können den Gerätezugriff auf VPC-Endpunkte einschränken AWS IoT Core Device Advisor und den Zugriff nur über VPC-Endpunkte zulassen, indem Sie VPC-Bedingungskontextschlüssel verwenden. AWS IoT Core unterstützt die folgenden VPC-bezogenen Kontextschlüssel:

- SourceVpc
- SourceVpce
- VPCSourceIp

> **Note**
>
> AWS IoT Core Device Advisor unterstützt derzeit keine VPC-Endpunktrichtlinien.

Die folgende Richtlinie gewährt die Erlaubnis, AWS IoT Core Device Advisor mithilfe einer Client-ID, die dem Namen der Sache entspricht, eine Verbindung herzustellen. Außerdem veröffentlicht sie zu jedem Thema, dem der Dingname vorangestellt ist. Die Richtlinie ist abhängig davon, dass das Gerät eine Verbindung zu einem VPC-Endpunkt mit einer bestimmten VPC-Endpunkt-ID herstellt. Diese Richtlinie verweigert Verbindungsversuche zu Ihrem öffentlichen AWS IoT Core Device Advisor -Testendpunkt.

```
{
"Version":"2012-10-17",
    "Statement": [
        {
"Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ],
            "Condition": {
"StringEquals": {
"aws:SourceVpce": "vpce-1a2b3c4d"
            }
        }

        },
        {
"Effect": "Allow",
            "Action": [
                "iot:Publish"
```

```
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/
${iot:Connection.Thing.ThingName}/*"
            ]
        }
    ]
}
```

Device-Advisor-Testfälle

Device Advisor bietet vorgefertigte Tests in sechs Kategorien.

- TLS
- MQTT
- Shadow
- Auftragsausführung
- Berechtigungen und Richtlinien
- Tests mit langer Dauer

Device Advisor-Testfälle, um sich für das zu qualifizieren AWS Programm zur Gerätequalifizierung.

Ihr Gerät muss die folgenden Tests bestehen, um sich gemäß dem AWS - Gerätequalifizierungsprogramm zu qualifizieren.

> **Note**
>
> Dies ist eine überarbeitete Liste der Qualifizierungstests.

- TLS Connect („TLS Connect“)
- TLS Incorrect Subject Name Server Cert („Incorrect Subject Common Name (CN)/Subject Alternative Name (SAN)“)

- TLS Unsecure Server Cert („Not Signed By Recognized CA“)
- TLS-Geräteunterstützung für AWS IoT Cipher Suites („TLS-Geräteunterstützung für AWS IoT empfohlene Cipher Suites“)
- TLS Receive Maximum Size Fragments(„TLS Receive Maximum Size Fragments“)
- TLS Expired Server Cert(„Expired server certificate“)
- TLS Large Size Server Cert(„TLS large Size Server Certificate“)
- MQTT Connect („Gerät sendet CONNECT an AWS IoT Core (Happy Case)“)
- MQTT Subscribe („Can Subscribe (Happy Case)“)
- MQTT Publish („QoS0 (Happy Case)“)
- MQTT Connect Jitter Retries(„Device connect retries with jitter backoff - No CONNACK response“)

TLS

Verwenden Sie diese Tests, um festzustellen, ob das Transport Layer Security Protocol (TLS) zwischen Ihren Geräten sicher AWS IoT ist.

> **Note**
>
> Device Advisor unterstützt jetzt TLS 1.3.

Happy Path

TLS Connect

Überprüft, ob das zu testende Gerät den TLS-Handshake bis abschließen kann. AWS IoT Dieser Test validiert nicht die MQTT-Implementierung des Client-Geräts.

Example Definition des API-Testfalls:

> **Note**
>
> `EXECUTION_TIMEOUT` hat einen Standardwert von 5 Minuten. Für optimale Ergebnisse empfehlen wir einen Timeout-Wert von 2 Minuten.

```
"tests":[
```

```
   {
      "name":"my_tls_connect_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  //in seconds
      },
      "test":{
         "id":"TLS_Connect",
         "version":"0.0.0"
      }
   }
]
```

Example Ergebnisse des Testfalls:

- Bestanden — Das getestete Gerät hat den TLS-Handshake mit abgeschlossen. AWS IoT
- Mit Warnungen bestanden — Das getestete Gerät hat den TLS-Handshake mit abgeschlossen AWS IoT, aber es gab TLS-Warnmeldungen vom Gerät oder. AWS IoT
- Fehlgeschlagen — Das getestete Gerät konnte den TLS-Handshake AWS IoT aufgrund eines Handshake-Fehlers nicht abschließen.

TLS empfängt Fragmente mit maximaler Größe

Dieser Testfall bestätigt, dass Ihr Gerät TLS-Fragmente mit maximaler Größe empfangen und verarbeiten kann. Ihr Testgerät muss ein vorkonfiguriertes Thema mit QoS 1 abonnieren, um eine große Nutzlast zu empfangen. Sie können die Nutzlast mit der Konfiguration `${payload}` anpassen.

Example Definition des API-Testfalls:

> **Note**
>
> `EXECUTION_TIMEOUT` hat einen Standardwert von 5 Minuten. Für optimale Ergebnisse empfehlen wir einen Timeout-Wert von 2 Minuten.

```
"tests":[
   {
      "name":"TLS Receive Maximum Size Fragments",
```

```
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  //in seconds
         "PAYLOAD_FORMAT":"{"message":"${payload}"}", // A string with a placeholder
 ${payload}, or leave it empty to receive a plain string.
         "TRIGGER_TOPIC": "test_1" // A topic to which a device will subscribe, and
 to which a test case will publish a large payload.
      },
      "test":{
         "id":"TLS_Receive_Maximum_Size_Fragments",
         "version":"0.0.0"
      }
   }
]
```

Cipher Suites

TLS-Geräteunterstützung für AWS IoT empfohlene Cipher Suites

Überprüft, ob die Cipher Suites in der TLS-Client-Hello-Nachricht des getesteten Geräts die empfohlenen AWS IoT -Cipher-Suites enthalten. Es bietet mehr Einblicke in die vom Gerät unterstützten Cipher-Suites.

Example Definition des API-Testfalls:

> **Note**
>
> EXECUTION_TIMEOUT hat einen Standardwert von 5 Minuten. Wir empfehlen einen Timeout-Wert von 2 Minuten.

```
"tests":[
   {
      "name":"my_tls_support_aws_iot_cipher_suites_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
      },
      "test":{
         "id":"TLS_Support_AWS_IoT_Cipher_Suites",
         "version":"0.0.0"
```

```
      }
   }
]
```

Example Ergebnisse des Testfalls:

- Bestanden — Das zu testende Gerät enthält mindestens eine der empfohlenen Verschlüsselungssammlungen und keine AWS IoT Verschlüsselungssammlungen, die nicht unterstützt werden.
- Pass with warnings (Mit Warnungen bestanden): Die Cipher Suites des Geräts enthalten mindestens eine AWS IoT -Cipher-Suite, aber:
  1. sie enthält keine der empfohlenen Cipher Suites.
  2. Es enthält Cipher-Suites, die von nicht unterstützt werden. AWS IoT

  Wir empfehlen Ihnen, zu überprüfen, ob alle nicht unterstützten Cipher Suites sicher sind.
- Fehlgeschlagen — Das Gerät, auf dem die Cipher-Suites getestet werden, enthält keine der AWS IoT unterstützten Cipher-Suites.

Größeres Serverzertifikat

Großes TLS-Serverzertifikat

Wird auf Ihrem Gerät validiert und kann den TLS-Handshake mit AWS IoT abschließen, wenn es ein größeres Serverzertifikat empfängt und verarbeitet. Die Größe des für diesen Test verwendeten Serverzertifikats (in Byte) ist um 20 größer als das, was derzeit im TLS Connect-Testfall und in IoT Core verwendet wird. AWS IoT Testet in diesem Testfall den Pufferspeicher Ihres Geräts auf TLS. Wenn der Pufferspeicher groß genug ist, wird der TLS-Handshake ohne Fehler abgeschlossen. Dieser Test validiert nicht die MQTT-Implementierung des Geräts. Der Testfall tritt auf, nachdem der TLS-Handshake-Vorgang abgeschlossen ist.

Example Definition des API-Testfalls:

> **Note**
>
> `EXECUTION_TIMEOUT` hat einen Standardwert von 5 Minuten. Für optimale Ergebnisse empfehlen wir einen Timeout-Wert von 2 Minuten. Wenn dieser Testfall fehlschlägt, der TLS Connect-Testfall jedoch erfolgreich ist, empfehlen wir Ihnen, das Pufferspeicherlimit Ihres Geräts für TLS zu erhöhen. Durch die Erhöhung des Pufferspeicherlimits wird

sichergestellt, dass Ihr Gerät ein größeres Serverzertifikat verarbeiten kann, falls die Größe zunimmt.

```
"tests":[
   {
      "name":"my_tls_large_size_server_cert_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
      },
      "test":{
         "id":"TLS_Large_Size_Server_Cert",
         "version":"0.0.0"
      }
   }
]
```

Example Ergebnisse des Testfalls:

- Pass (Bestanden): Das getestete Gerät hat den TLS-Handshake mit AWS IoT abgeschlossen.
- Mit Warnungen bestanden — Das getestete Gerät hat den TLS-Handshake mit abgeschlossen AWS IoT, aber es gibt TLS-Warnmeldungen entweder vom Gerät oder. AWS IoT
- Fehlgeschlagen — Das getestete Gerät konnte den TLS-Handshake AWS IoT aufgrund eines Fehlers während des Handshake-Vorgangs nicht abschließen.

Zertifikat für unsicheren TLS-Server

Nicht von einer anerkannten Zertifizierungsstelle signiert

Überprüft, ob das getestete Gerät die Verbindung schließt, wenn ihm ein Serverzertifikat ohne gültige Signatur der ATS-Zertifizierungsstelle vorgelegt wird. Ein Gerät sollte nur eine Verbindung zu einem Endpunkt herstellen, der ein gültiges Zertifikat vorlegt.

Example Definition des API-Testfalls:

> **Note**
>
> `EXECUTION_TIMEOUT` hat einen Standardwert von 5 Minuten. Wir empfehlen einen Timeout-Wert von 2 Minuten.

```
"tests":[
   {
      "name":"my_tls_unsecure_server_cert_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  //in seconds
      },
      "test":{
         "id":"TLS_Unsecure_Server_Cert",
         "version":"0.0.0"
      }
   }
]
```

Example Ergebnisse des Testfalls:

- Pass (Bestanden): Das getestete Gerät hat die Verbindung geschlossen.
- Fehlgeschlagen — Das getestete Gerät hat den TLS-Handshake mit abgeschlossen. AWS IoT

TLS Incorrect Subject Name Server Cert/Incorrect Subject Common Name (CN)/Subject Alternative Name (SAN)

Überprüft, ob das getestete Gerät die Verbindung schließt, wenn ihm ein Serverzertifikat für einen anderen Domainnamen als den angeforderten vorgelegt wird.

Example Definition des API-Testfalls:

> **Note**
>
> `EXECUTION_TIMEOUT` hat einen Standardwert von 5 Minuten. Wir empfehlen einen Timeout-Wert von 2 Minuten.

```
"tests":[
   {
      "name":"my_tls_incorrect_subject_name_cert_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",   // in seconds
      },
      "test":{
         "id":"TLS_Incorrect_Subject_Name_Server_Cert",
         "version":"0.0.0"
      }
   }
]
```

Example Ergebnisse des Testfalls:

- Pass (Bestanden): Das getestete Gerät hat die Verbindung geschlossen.
- Fehlgeschlagen — Das getestete Gerät hat den TLS-Handshake mit abgeschlossen. AWS IoT

TLS-Serverzertifikat ist abgelaufen

Abgelaufenes Serverzertifikat

Überprüft, ob das getestete Gerät die Verbindung schließt, wenn ihm ein abgelaufenes Serverzertifikat vorgelegt wird.

Example Definition des API-Testfalls:

> **Note**
>
> EXECUTION_TIMEOUT hat einen Standardwert von 5 Minuten. Wir empfehlen einen Timeout-Wert von 2 Minuten.

```
"tests":[
   {
      "name":"my_tls_expired_cert_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  //in seconds
```

```
      },
      "test":{
         "id":"TLS_Expired_Server_Cert",
         "version":"0.0.0"
      }
   }
]
```

Example Ergebnisse des Testfalls:

- Bestanden — Das getestete Gerät weigert sich, den TLS-Handshake mit abzuschließen. AWS IoT Das Gerät sendet eine TLS-Warnmeldung, bevor es die Verbindung schließt.
- Pass with warnings (Bestanden mit Warnungen): Das getestete Gerät weigert sich, den TLS-Handshake mit AWS IoT abzuschließen. Es sendet jedoch keine TLS-Warnmeldung, bevor die Verbindung geschlossen wird.
- Fehlgeschlagen — Das getestete Gerät schließt den TLS-Handshake mit ab. AWS IoT

MQTT

CONNECT, DISCONNECT und RECONNECT

„Gerät sendet CONNECT an AWS IoT Core (Happy Case)“

Überprüft, ob das getestete Gerät eine CONNECT-Anfrage sendet.

Definition des API-Testfalls:

> **Note**
>
> `EXECUTION_TIMEOUT` hat einen Standardwert von 5 Minuten. Wir empfehlen einen Timeout-Wert von 2 Minuten.

```
"tests":[
   {
      "name":"my_mqtt_connect_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",   // in seconds
```

```
      },
      "test":{
         "id":"MQTT_Connect",
         "version":"0.0.0"
      }
   }
]
```

„Device can return PUBACK to an arbitrary topic for QoS1“

In diesem Testfall wird überprüft, ob das Gerät (Client) eine PUBACK-Nachricht zurückgeben kann, wenn es nach dem Abonnieren eines Themas mit QoS1 eine Veröffentlichungsnachricht vom Broker erhalten hat.

Der Inhalt der Nutzlast und die Größe der Nutzlast sind für diesen Testfall konfigurierbar. Wenn die Nutzlastgröße konfiguriert ist, überschreibt Device Advisor den Wert für den Nutzlastinhalt und sendet eine vordefinierte Nutzlast mit der gewünschten Größe an das Gerät. Die Nutzlastgröße ist ein Wert zwischen 0 und 128 und darf 128 KB nicht überschreiten. AWS IoT Core lehnt Veröffentlichungs- und Verbindungsanfragen ab, die größer als 128 KB sind, wie auf der Seite Grenzwerte und Kontingente für AWS IoT Core -Message-Broker und -Protokolle beschrieben.

Definition des API-Testfalls:

> **Note**
>
> `EXECUTION_TIMEOUT` hat einen Standardwert von 5 Minuten. Wir empfehlen einen Timeout-Wert von 2 Minuten. `PAYLOAD_SIZE` kann auf einen Wert zwischen 0 und 128 Kilobyte konfiguriert werden. Die Definition einer Nutzlastgröße hat Vorrang vor dem Nutzlastinhalt, da Device Advisor eine vordefinierte Nutzlast mit der angegebenen Größe zurück an das Gerät sendet.

```
"tests":[
{
        "name":"my_mqtt_client_puback_qos1",
        "configuration": {
            // optional:"TRIGGER_TOPIC": "myTopic",
            "EXECUTION_TIMEOUT":"300", // in seconds
            "PAYLOAD_FOR_PUBLISH_VALIDATION":"custom payload",
            "PAYLOAD_SIZE":"100" // in kilobytes
```

```
        },
        "test": {
            "id": "MQTT_Client_Puback_QoS1",
            "version": "0.0.0"
        }
    }
]
```

„Device connect retries with jitter backoff - No CONNACK response“

Überprüft, ob das getestete Gerät den richtigen Jitter-Backoff verwendet, wenn es mindestens fünfmal erneut eine Verbindung zum Broker herstellt. Der Broker protokolliert den Zeitstempel der CONNECT-Anfrage des getesteten Geräts, führt eine Paketvalidierung durch, pausiert, ohne einen CONNACK an das getestete Gerät zu senden, und wartet, bis das getestete Gerät die Anfrage erneut sendet. Der sechste Verbindungsversuch darf übergeben werden und CONNACK darf zurück zum getesteten Gerät fließen.

Der vorherige Vorgang wird erneut ausgeführt. Insgesamt erfordert dieser Testfall, dass das Gerät insgesamt mindestens 12 Mal eine Verbindung herstellt. Die gesammelten Zeitstempel werden verwendet, um zu überprüfen, ob das getestete Gerät den Jitter-Backoff verwendet. Wenn das getestete Gerät eine streng exponentielle Backoff-Verzögerung aufweist, wird dieser Testfall mit Warnungen bestanden.

Wir empfehlen, den Mechanismus Exponential Backoff And Jitter (Exponentielles Backoff und Jitter) auf dem getesteten Gerät zu implementieren, um diesen Testfall zu bestehen.

Definition des API-Testfalls:

> **Note**
>
> `EXECUTION_TIMEOUT` hat einen Standardwert von 5 Minuten. Wir empfehlen einen Timeout-Wert von 4 Minuten.

```
"tests":[
   {
      "name":"my_mqtt_jitter_backoff_retries_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",    // in seconds
```

```
      },
      "test":{
         "id":"MQTT_Connect_Jitter_Backoff_Retries",
         "version":"0.0.0"
      }
   }
]
```

„Device connect retries with exponential backoff - No CONNACK response“

Überprüft, ob das getestete Gerät das richtige exponentielle Backoff verwendet, wenn es mindestens fünfmal erneut eine Verbindung zum Broker herstellt. Der Broker protokolliert den Zeitstempel der CONNECT-Anfrage des getesteten Geräts, führt eine Paketvalidierung durch, pausiert, ohne einen CONNACK an das Client-Gerät zu senden, und wartet, bis das getestete Gerät die Anfrage erneut sendet. Die gesammelten Zeitstempel werden verwendet, um zu überprüfen, ob das getestete Gerät ein exponentielles Backoff verwendet.

Wir empfehlen, den Mechanismus Exponential Backoff And Jitter (Exponentielles Backoff und Jitter) auf dem getesteten Gerät zu implementieren, um diesen Testfall zu bestehen.

Definition des API-Testfalls:

> **Note**
>
> `EXECUTION_TIMEOUT` hat einen Standardwert von 5 Minuten. Wir empfehlen einen Timeout-Wert von 4 Minuten.

```
"tests":[
   {
      "name":"my_mqtt_exponential_backoff_retries_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"600",  // in seconds
      },
      "test":{
         "id":"MQTT_Connect_Exponential_Backoff_Retries",
         "version":"0.0.0"
      }
   }
]
```

„Device re-connect with jitter backoff - After server disconnect“

Überprüft, ob ein getestetes Gerät bei der Wiederherstellung der Verbindung, nachdem es vom Server getrennt wurde, die erforderlichen Jitter- und Backoff-Werte verwendet. Device Advisor trennt das Gerät mindestens fünfmal vom Server und beobachtet das Verhalten des Geräts bei der Wiederherstellung der Verbindung mit MQTT. Der Broker protokolliert den Zeitstempel der CONNECT-Anfrage des getesteten Geräts, führt eine Paketvalidierung durch, pausiert, ohne einen CONNACK an das Client-Gerät zu senden, und wartet, bis das getestete Gerät die Anfrage erneut sendet. Die gesammelten Zeitstempel werden verwendet, um zu überprüfen, ob das getestete Gerät den Jitter-Backoff verwendet. Wenn das getestete Gerät eine streng exponentielle Backoff-Verzögerung aufweist oder keinen ordnungsgemäßen Jitter-Backoff-Mechanismus implementiert, wird dieser Testfall mit Warnungen bestanden. Wenn das getestete Gerät entweder einen linearen Backoff-Mechanismus oder einen konstanten Backoff-Mechanismus implementiert hat, schlägt der Test fehl.

Damit dieser Testfall bestanden wird, empfehlen wir, den Mechanismus Exponential Backoff And Jitter (Exponentielles Backoff und Jitter) auf dem getesteten Gerät zu implementieren.

Definition des API-Testfalls:

> **Note**
>
> `EXECUTION_TIMEOUT` hat einen Standardwert von 5 Minuten. Wir empfehlen einen Timeout-Wert von 4 Minuten.
> Die Anzahl der Wiederverbindungsversuche zur Überprüfung des Backoffs kann durch Angabe der `RECONNECTION_ATTEMPTS` geändert werden. Die Zahl muss zwischen 5 und 10 liegen. Der Standardwert ist 5.

```
"tests":[
   {
      "name":"my_mqtt_reconnect_backoff_retries_on_server_disconnect",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
         "RECONNECTION_ATTEMPTS": 5
      },
      "test":{
         "id":"MQTT_Reconnect_Backoff_Retries_On_Server_Disconnect",
```

```
         "version":"0.0.0"
      }
   }
]
```

„Device re-connect with jitter backoff - On unstable connection“

Überprüft, ob ein getestetes Gerät beim erneuten Herstellen der Verbindung über eine instabile Verbindung die erforderlichen Jitter- und Backoff-Werte verwendet. Device Advisor trennt das Gerät nach fünf erfolgreichen Verbindungen vom Server und beobachtet das Verhalten des Geräts bei der Wiederherstellung der Verbindung mit MQTT. Der Broker protokolliert den Zeitstempel der CONNECT-Anfrage des getesteten Geräts, führt eine Paketvalidierung durch, sendet CONNACK zurück, trennt die Verbindung, protokolliert den Zeitstempel der getrennten Verbindung und wartet, bis das getestete Gerät die Anfrage erneut sendet. Die gesammelten Zeitstempel werden verwendet, um zu überprüfen, ob das getestete Gerät Jitter und Backoff verwendet, während es nach erfolgreichen, aber instabilen Verbindungen erneut eine Verbindung herstellt. Wenn das getestete Gerät eine streng exponentielle Backoff-Verzögerung aufweist oder keinen ordnungsgemäßen Jitter-Backoff-Mechanismus implementiert, wird dieser Testfall mit Warnungen bestanden. Wenn das getestete Gerät entweder einen linearen Backoff-Mechanismus oder einen konstanten Backoff-Mechanismus implementiert hat, schlägt der Test fehl.

Damit dieser Testfall bestanden wird, empfehlen wir, den Mechanismus Exponential Backoff And Jitter (Exponentielles Backoff und Jitter) auf dem getesteten Gerät zu implementieren.

Definition des API-Testfalls:

> **Note**
>
> `EXECUTION_TIMEOUT` hat einen Standardwert von 5 Minuten. Wir empfehlen einen Timeout-Wert von 4 Minuten.
> Die Anzahl der Wiederverbindungsversuche zur Überprüfung des Backoffs kann durch Angabe der `RECONNECTION_ATTEMPTS` geändert werden. Die Zahl muss zwischen 5 und 10 liegen. Der Standardwert ist 5.

```
"tests":[
   {
      "name":"my_mqtt_reconnect_backoff_retries_on_unstable_connection",
      "configuration":{
```

```
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
         "RECONNECTION_ATTEMPTS": 5
      },
      "test":{
         "id":"MQTT_Reconnect_Backoff_Retries_On_Unstable_Connection",
         "version":"0.0.0"
      }
   }
]
```

Veröffentlichen

„QoS0 (Happy Case)“

Überprüft, ob das getestete Gerät eine Nachricht mit QoS0 oder QoS1 veröffentlicht. Sie können auch das Thema der Nachricht und die Nutzlast überprüfen, indem Sie den Themenwert und die Nutzlast in den Testeinstellungen angeben.

> **Note**
>
> EXECUTION_TIMEOUT hat einen Standardwert von 5 Minuten. Wir empfehlen einen Timeout-Wert von 2 Minuten.

```
"tests":[
   {
      "name":"my_mqtt_publish_test",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
         "TOPIC_FOR_PUBLISH_VALIDATION": "my_TOPIC_FOR_PUBLISH_VALIDATION",
         "PAYLOAD_FOR_PUBLISH_VALIDATION": "my_PAYLOAD_FOR_PUBLISH_VALIDATION",
      },
      "test":{
         "id":"MQTT_Publish",
         "version":"0.0.0"
      }
   }
]
```

„QoS1 publish retry - No PUBACK“

Überprüft, ob das getestete Gerät eine mit QoS1 gesendete Nachricht erneut veröffentlicht, falls der Broker PUBACK nicht sendet. Sie können auch das Thema der Nachricht überprüfen, indem Sie dieses Thema in den Testeinstellungen angeben. Das Client-Gerät darf die Verbindung nicht trennen, bevor die Nachricht erneut veröffentlicht wird. Mit diesem Test wird auch überprüft, ob die erneut veröffentlichte Nachricht dieselbe Paket-ID wie das Original hat. Wenn das Gerät während der Testausführung die Verbindung verliert und die Verbindung wiederherstellt, wird der Testfall ohne Fehler zurückgesetzt und das Gerät muss die Testfallschritte erneut ausführen.

Definition des API-Testfalls:

> **Note**
>
> `EXECUTION_TIMEOUT` hat einen Standardwert von 5 Minuten. Es wird eine Dauer von mindestens 4 Minuten empfohlen.

```
"tests":[
   {
      "name":"my_mqtt_publish_retry_test",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
         "TOPIC_FOR_PUBLISH_VALIDATION": "my_TOPIC_FOR_PUBLISH_VALIDATION",
         "PAYLOAD_FOR_PUBLISH_VALIDATION": "my_PAYLOAD_FOR_PUBLISH_VALIDATION",
      },
      "test":{
         "id":"MQTT_Publish_Retry_No_Puback",
         "version":"0.0.0"
      }
   }
]
```

„Publish Retained messages“

Überprüft, ob das getestete Gerät eine Nachricht veröffentlicht, bei der `retainFlag` auf „true“ gesetzt ist. Sie können das Thema der Nachricht und die Nutzlast überprüfen, indem Sie den Themenwert und die Nutzlast in den Testeinstellungen angeben. Wenn die `retainFlag`, die im PUBLISH-Paket gesendet wurde, nicht auf „true“ gesetzt ist, schlägt der Testfall fehl.

Definition des API-Testfalls:

> **Note**
>
> `EXECUTION_TIMEOUT` hat einen Standardwert von 5 Minuten. Wir empfehlen einen Timeout-Wert von 2 Minuten. Um diesen Testfall auszuführen, fügen Sie die `iot:RetainPublish`-Aktion zu Ihrer Geräterolle hinzu.

```
"tests":[
   {
      "name":"my_mqtt_publish_retained_messages_test",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds

 "TOPIC_FOR_PUBLISH_RETAINED_VALIDATION": "my_TOPIC_FOR_PUBLISH_RETAINED_VALIDATION",

 "PAYLOAD_FOR_PUBLISH_RETAINED_VALIDATION": "my_PAYLOAD_FOR_PUBLISH_RETAINED_VALIDATION",
      },
      "test":{
         "id":"MQTT_Publish_Retained_Messages",
         "version":"0.0.0"
      }
   }
]
```

„Publish with User Property“

Überprüft, ob das getestete Gerät eine Nachricht mit der korrekten Benutzereigenschaft veröffentlicht. Sie können die Benutzereigenschaft überprüfen, indem Sie das Name-Wert-Paar in den Testeinstellungen festlegen. Wenn die Benutzereigenschaft nicht bereitgestellt wird oder nicht übereinstimmt, schlägt der Testfall fehl.

Definition des API-Testfalls:

> **Note**
>
> Dies ist ein reiner MQTT5-Testfall.

EXECUTION_TIMEOUT hat einen Standardwert von 5 Minuten. Wir empfehlen einen Timeout-Wert von 2 Minuten.

```
"tests":[
   {
      "name":"my_mqtt_user_property_test",
      "test":{
        "USER_PROPERTIES": [
            {"name": "name1", "value":"value1"},
            {"name": "name2", "value":"value2"}
        ],
        "EXECUTION_TIMEOUT":"300",  // in seconds
      },
      "test":{
         "id":"MQTT_Publish_User_Property",
         "version":"0.0.0"
      }
   }
]
```

Abonnieren

„Can Subscribe (Happy Case)“

Überprüft, ob das getestete Gerät MQTT-Themen abonniert hat. Sie können auch das Thema überprüfen, das das getestete Gerät abonniert, indem Sie dieses Thema in den Testeinstellungen angeben.

Definition des API-Testfalls:

Note

EXECUTION_TIMEOUT hat einen Standardwert von 5 Minuten. Wir empfehlen einen Timeout-Wert von 2 Minuten.

```
"tests":[
   {
```

```
      "name":"my_mqtt_subscribe_test",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
         "TOPIC_LIST_FOR_SUBSCRIPTION_VALIDATION":
["my_TOPIC_FOR_PUBLISH_VALIDATION_a","my_TOPIC_FOR_PUBLISH_VALIDATION_b"]
      },
      "test":{
         "id":"MQTT_Subscribe",
         "version":"0.0.0"
      }
   }
]
```

„Subscribe Retry - No SUBACK“

Überprüft, ob das getestete Gerät ein fehlgeschlagenes Abonnement von MQTT-Themen erneut versucht. Der Server wartet dann und sendet kein SUBACK. Wenn das Client-Gerät das Abonnement nicht erneut versucht, schlägt der Test fehl. Das Client-Gerät muss das fehlgeschlagene Abonnement mit derselben Paket-ID erneut versuchen. Sie können auch das Thema überprüfen, das das getestete Gerät abonniert, indem Sie dieses Thema in den Testeinstellungen angeben. Wenn das Gerät während der Testausführung die Verbindung verliert und die Verbindung wiederherstellt, wird der Testfall ohne Fehler zurückgesetzt und das Gerät muss die Testfallschritte erneut ausführen.

Definition des API-Testfalls:

> **Note**
>
> EXECUTION_TIMEOUT hat einen Standardwert von 5 Minuten. Wir empfehlen einen Timeout-Wert von 4 Minuten.

```
"tests":[
   {
      "name":"my_mqtt_subscribe_retry_test",
      "configuration":{
         "EXECUTION_TIMEOUT":"300",  // in seconds
         // optional:
         "TOPIC_LIST_FOR_SUBSCRIPTION_VALIDATION":
["myTOPIC_FOR_PUBLISH_VALIDATION_a","my_TOPIC_FOR_PUBLISH_VALIDATION_b"]
```

```
      },
      "test":{
         "id":"MQTT_Subscribe_Retry_No_Suback",
         "version":"0.0.0"
      }
   }
]
```

Keep-Alive

„Matt kein Ack“ PingResp

Dieser Testfall überprüft, ob das getestete Gerät die Verbindung trennt, wenn es keine Ping-Antwort erhält. Im Rahmen dieses Testfalls blockiert Device Advisor Antworten, die von AWS IoT Core für Veröffentlichungs-, Abonnement- und Ping-Anfragen gesendet wurden. Es überprüft auch, ob das getestete Gerät die MQTT-Verbindung unterbricht.

Definition des API-Testfalls:

> **Note**
>
> `EXECUTION_TIMEOUT` hat einen Standardwert von 5 Minuten. Wir empfehlen ein Timeout, das mehr als das 1,5-fache des `keepAliveTime`-Werts beträgt.
> Die maximale `keepAliveTime` darf für diesen Test nicht länger als 230 Sekunden betragen.

```
"tests":[
    {
       "name":"Mqtt No Ack PingResp",
       "configuration":
          //optional:
          "EXECUTION_TIMEOUT":"306",   // in seconds
       },
       "test":{
          "id":"MQTT_No_Ack_PingResp",
          "version":"0.0.0"
       }
    }
]
```

Persistente Sitzung

„Persistent Session (Happy Case)“

Dieser Testfall validiert das Geräteverhalten, wenn die Verbindung zu einer persistenten Sitzung getrennt wird. Der Testfall prüft, ob das Gerät erneut eine Verbindung herstellen kann, die Abonnements für seine Trigger-Themen fortsetzen kann, ohne sich explizit erneut anzumelden, die in den Themen gespeicherten Nachrichten empfangen kann und während einer persistenten Sitzung erwartungsgemäß funktionieren kann. Wenn dieser Testfall erfolgreich ist, bedeutet dies, dass das Client-Gerät in der Lage ist, eine dauerhafte Sitzung mit dem AWS IoT Core Broker in erwarteter Weise aufrechtzuerhalten. Weitere Informationen zu AWS IoT persistenten Sitzungen finden Sie unter Persistente MQTT-Sitzungen https://docs.aws.amazon.com/iot/latest/developerguide/mqtt.html#mqtt-persistent-sessions verwenden.

In diesem Testfall wird erwartet, dass das Client-Gerät ein CONNECT mit dem AWS IoT Core mit einem Sitzungs-Flag auf „false“ gesetzt durchführt und dann ein Trigger-Thema abonniert. Nach einem erfolgreichen Abonnement wird das Gerät von AWS IoT Core Device Advisor getrennt. Solange sich das Gerät in einem getrennten Zustand befindet, wird eine QoS 1-Nachrichtennutzlast in diesem Thema gespeichert. Device Advisor ermöglicht dem Client-Gerät dann, erneut eine Verbindung mit dem Testendpunkt herzustellen. Da zu diesem Zeitpunkt eine persistente Sitzung besteht, wird erwartet, dass das Client-Gerät seine Themenabonnements wieder aufnimmt, ohne zusätzliche SUBSCRIBE-Pakete zu senden und die QoS 1-Nachricht vom Broker zu empfangen. Wenn das Client-Gerät nach der erneuten Verbindung sein Trigger-Thema erneut abonniert, indem es ein zusätzliches SUBSCRIBE-Paket and/or sendet und der Client die gespeicherte Nachricht vom Trigger-Thema nicht empfängt, schlägt der Testfall fehl.

Definition des API-Testfalls:

> **Note**
>
> `EXECUTION_TIMEOUT` hat einen Standardwert von 5 Minuten. Wir empfehlen einen Timeout-Wert von mindestens 4 Minuten. Bei der ersten Verbindung muss das Client-Gerät explizit ein `TRIGGER_TOPIC` abonnieren, die zuvor nicht abonniert wurde. Um den Testfall zu bestehen, muss das Client-Gerät erfolgreich `TRIGGER_TOPIC` mit einer QoS 1 abonnieren. Nach dem erneuten Herstellen der Verbindung wird erwartet, dass das Client-Gerät erkennt, dass eine aktive persistente Sitzung besteht. Daher sollte es die vom Trigger-Thema gesendete gespeicherte Nachricht akzeptieren und PUBACK für diese spezifische Nachricht zurückgeben.

```
"tests":[
   {
      "name":"my_mqtt_persistent_session_happy_case",
      "configuration":{
         //required:
         "TRIGGER_TOPIC": "myTrigger/topic",
         // optional:
         // if Payload not provided, a string will be stored in the trigger topic to
 be sent back to the client device
         "PAYLOAD": "The message which should be received from AWS IoT Broker after
 re-connecting to a persistent session from the specified trigger topic.",

         "EXECUTION_TIMEOUT":"300" // in seconds
      },
      "test":{
         "id":"MQTT_Persistent_Session_Happy_Case",
         "version":"0.0.0"
      }
   }
]
```

„Persistent Session - Session Expiry“

Dieser Testfall hilft bei der Überprüfung des Geräteverhaltens, wenn ein getrenntes Gerät erneut eine Verbindung zu einer abgelaufenen persistenten Sitzung herstellt. Nach Ablauf der Sitzung erwarten wir, dass das Gerät die zuvor abonnierten Themen erneut abonniert, indem es explizit ein neues SUBSCRIBE-Paket sendet.

Bei der ersten Verbindung erwarten wir, dass sich das Testgerät mit dem AWS IoT-Broker VERBINDET, da dessen `CleanSession` Flag auf false gesetzt ist, um eine persistente Sitzung zu initiieren. Das Gerät sollte dann ein Trigger-Thema abonnieren. Nach erfolgreichem Abonnement und Initiierung einer dauerhaften Sitzung wird AWS IoT Core das Gerät dann von Device Advisor getrennt. Nach der Trennung ermöglicht AWS IoT Core Device Advisor dem Testgerät, sich wieder mit dem Testendpunkt zu verbinden. Wenn das Testgerät zu diesem Zeitpunkt ein weiteres CONNECT-Paket sendet, sendet AWS IoT Core Device Advisor ein CONNACK-Paket zurück, das darauf hinweist, dass die persistente Sitzung abgelaufen ist. Das Testgerät muss dieses Paket richtig interpretieren und es wird erwartet, dass es dasselbe Trigger-Thema erneut abonniert, wenn die persistente Sitzung beendet wird. Wenn das Testgerät seinen Themen-Trigger nicht erneut abonniert, schlägt der Testfall fehl. Damit der Test erfolgreich

ist, muss das Gerät verstehen, dass die persistente Sitzung beendet ist, und in der zweiten Verbindung ein neues SUBSCRIBE-Paket für dasselbe Trigger-Thema zurücksenden.

Wenn dieser Testfall für ein Testgerät erfolgreich ist, bedeutet dies, dass das Gerät in der Lage ist, die erneute Verbindung nach Ablauf der persistenten Sitzung erwartungsgemäß zu handhaben.

Definition des API-Testfalls:

> **Note**
>
> `EXECUTION_TIMEOUT` hat einen Standardwert von 5 Minuten. Wir empfehlen einen Timeout-Wert von mindestens 4 Minuten. Das Testgerät muss explizit ein `TRIGGER_TOPIC` abonnieren, das es zuvor nicht abonniert hatte. Um den Testfall zu bestehen, muss das Testgerät ein CONNECT-Paket mit einem auf „false“ gesetzten `CleanSession`-Flag senden und erfolgreich ein Trigger-Thema mit QoS 1 abonnieren. Nach einer erfolgreichen Verbindung trennt AWS IoT Core Device Advisor das Gerät. Nach der Trennung ermöglicht AWS IoT Core Device Advisor dem Gerät, wieder eine Verbindung herzustellen, und es wird erwartet, dass das Gerät dieselbe Verbindung erneut abonniert, `TRIGGER_TOPIC` da AWS IoT Core Device Advisor die persistente Sitzung beendet hätte.

```
"tests":[
   {
      "name":"my_expired_persistent_session_test",
      "configuration":{
         //required:
         "TRIGGER_TOPIC": "myTrigger/topic",
         // optional:
         "EXECUTION_TIMEOUT":"300" // in seconds
      },
      "test":{
         "id":"MQTT_Expired_Persistent_Session",
         "version":"0.0.0"
      }
   }
]
```

Shadow

Verwenden Sie diese Tests, um zu überprüfen, ob Ihre getesteten Geräte den AWS IoT Device Shadow-Dienst korrekt verwenden. Weitere Informationen finden Sie unter AWS IoT Geräteschatten-Service. Wenn diese Testfälle in Ihrer Testsuite konfiguriert sind, müssen Sie beim Start der Suite-Ausführung etwas angeben.

WebSocketMQTT-Over wird derzeit nicht unterstützt.

Veröffentlichen

„Device publishes state after it connects (Happy case)“

Überprüft, ob ein Gerät seinen Status veröffentlichen kann, nachdem es eine Verbindung hergestellt hat AWS IoT Core

Definition des API-Testfalls:

> **Note**
>
> `EXECUTION_TIMEOUT` hat einen Standardwert von 5 Minuten. Wir empfehlen einen Timeout-Wert von 2 Minuten.

```
"tests":[
   {
      "name":"my_shadow_publish_reported_state",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300", // in seconds
         "SHADOW_NAME": "SHADOW_NAME",
         "REPORTED_STATE": {
            "STATE_ATTRIBUTE": "STATE_VALUE"
         }
      },
      "test":{
         "id":"Shadow_Publish_Reported_State",
         "version":"0.0.0"
      }
   }
]
```

Der REPORTED_STATE kann für eine zusätzliche Überprüfung des exakten Shadow-Status Ihres Geräts bereitgestellt werden, nachdem es eine Verbindung hergestellt hat. Standardmäßig überprüft dieser Testfall den Veröffentlichungsstatus Ihres Geräts.

Falls *SHADOW_NAME* nicht angegeben, sucht der Testfall standardmäßig nach Nachrichten, die mit Themenpräfixen des Shadow-Typs Unbenannt (klassisch) veröffentlicht wurden. Geben Sie einen Shadow-Namen an, wenn Ihr Gerät den benannten Shadow-Typ verwendet. Weitere Informationen finden Sie unter Verwenden von Shadows in Geräten.

Aktualisierung

„Device updates reported state to desired state (Happy case)“

Überprüft, ob Ihr Gerät alle empfangenen Aktualisierungsnachrichten liest, und synchronisiert den Status des Geräts so, dass er mit den gewünschten Statuseigenschaften übereinstimmt. Ihr Gerät sollte nach der Synchronisierung den zuletzt gemeldeten Status veröffentlichen. Wenn auf Ihrem Gerät bereits ein Shadow vorhanden ist, bevor Sie den Test ausführen, stellen Sie sicher, dass der für den Testfall konfigurierte gewünschte Status und der vorhandene gemeldete Status nicht bereits übereinstimmen. Sie können die vom Device Advisor gesendeten Shadow-Aktualisierungsnachrichten erkennen, indem Sie sich das ClientToken Feld im Shadow-Dokument so ansehen, wie es aussehen wirdDeviceAdvisorShadowTestCaseSetup.

Definition des API-Testfalls:

> **Note**
>
> EXECUTION_TIMEOUT hat einen Standardwert von 5 Minuten. Wir empfehlen einen Timeout-Wert von 2 Minuten.

```
"tests":[
   {
      "name":"my_shadow_update_reported_state",
      "configuration": {
         "DESIRED_STATE": {
            "STATE_ATTRIBUTE": "STATE_VALUE"
         },
         // optional:
         "EXECUTION_TIMEOUT":"300", // in seconds
```

```
            "SHADOW_NAME": "SHADOW_NAME"
        },
        "test":{
            "id":"Shadow_Update_Reported_State",
            "version":"0.0.0"
        }
    }
]
```

Der DESIRED_STATE sollte mindestens ein Attribut und einen zugehörigen Wert haben.

Falls SHADOW_NAME nicht angegeben, sucht der Testfall standardmäßig nach Nachrichten, die mit Themenpräfixen des Shadow-Typs Unbenannt (klassisch) veröffentlicht wurden. Geben Sie einen Shadow-Namen an, wenn Ihr Gerät den benannten Shadow-Typ verwendet. Weitere Informationen finden Sie unter Verwenden von Shadows in Geräten.

Auftragsausführung

„Device can complete a job execution“

Mit diesem Testfall können Sie überprüfen, ob Ihr Gerät mithilfe von AWS IoT Jobs Updates empfangen kann, und den Status erfolgreicher Updates veröffentlichen. Weitere Informationen zu AWS IoT Jobs finden Sie unter Jobs.

Um diesen Testfall erfolgreich ausführen zu können, gibt es zwei reservierte AWS Themen, denen Sie Ihre Geräterolle zuweisen müssen. Verwenden Sie die Themen notify und notify-next, um Nachrichten zu Auftragsaktivitäten zu abonnieren. Ihre Geräterolle muss die Aktion PUBLISH für die folgenden Themen gewähren:

- $aws/things/ThingName /jobs/ jobId /get
- $/ThingName /jobs/ jobId /update aws/things

Es wird empfohlen, die Aktionen SUBSCRIBE und RECEIVE für die folgenden Themen zu gewähren:

- $aws/things/thingNamejobs/get//akzeptiert
- $aws/things/ThingName /jobs/ jobId/get/rejected
- $/ThingName /jobs/ jobId/aws/thingsupdate/accepted
- $/ThingName /jobs/ jobId/aws/thingsupdate/rejected

Es wird empfohlen, die Aktionen SUBSCRIBE für die folgenden Themen zu gewähren:

- $/ThingName/-weiter aws/things jobs/notify

Weitere Informationen zu diesen reservierten Themen finden Sie unter Reservierte Themen für AWS IoT -Aufträge.

WebSocketMQTT-Over wird derzeit nicht unterstützt.

Definition des API-Testfalls:

> **Note**
>
> `EXECUTION_TIMEOUT` hat einen Standardwert von 5 Minuten. Wir empfehlen einen Timeout-Wert von 3 Minuten. Passen Sie je nach dem bereitgestellten AWS IoT Jobdokument oder der angegebenen Quelle den Timeout-Wert an (wenn die Ausführung eines Jobs beispielsweise lange dauert, definieren Sie einen längeren Timeout-Wert für den Testfall). Um den Test auszuführen, ist entweder ein gültiges AWS IoT Jobdokument oder eine bereits vorhandene Job-ID erforderlich. Ein AWS IoT Jobdokument kann als JSON-Dokument oder als S3-Link bereitgestellt werden. Wenn ein Auftragsdokument bereitgestellt wird, ist die Angabe einer Auftrags-ID optional. Wenn eine Job-ID angegeben wird, verwendet Device Advisor diese ID, während der AWS IoT Job in Ihrem Namen erstellt wird. Wenn das Auftragsdokument nicht bereitgestellt wird, können Sie eine vorhandene ID angeben, die sich in derselben Region befindet, in der Sie den Testfall ausführen. In diesem Fall verwendet Device Advisor diesen AWS IoT Job, während der Testfall ausgeführt wird.

```
"tests": [
   {
      "name":"my_job_execution",
      "configuration": {
         // optional:
         // Test case will create a job task by using either JOB_DOCUMENT or
 JOB_DOCUMENT_SOURCE.
         // If you manage the job task on your own, leave it empty and provide the
 JOB_JOBID (self-managed job task).
         // JOB_DOCUMENT is a JSON formatted string
         "JOB_DOCUMENT": "{
            \"operation\":\"reboot\",
            \"files\" : {
```

```
                \"fileName\" : \"install.py\",
                \"url\" : \"${aws:iot:s3-presigned-url:https://s3.amazonaws.com/
bucket-name/key}\"
             }
          }",
          // JOB_DOCUMENT_SOURCE is an S3 link to the job document. It will be used
 only if JOB_DOCUMENT is not provided.
          "JOB_DOCUMENT_SOURCE": "https://s3.amazonaws.com/bucket-name/key",
          // JOB_JOBID is mandatory, only if neither document nor document source is
 provided. (Test case needs to know the self-managed job task id).
          "JOB_JOBID": "String",
          // JOB_PRESIGN_ROLE_ARN is used for the presign Url, which will replace the
 placeholder in the JOB_DOCUMENT field
          "JOB_PRESIGN_ROLE_ARN": "String",
          // Presigned Url expiration time. It must be between 60 and 3600 seconds,
 with the default value being 3600.
          "JOB_PRESIGN_EXPIRES_IN_SEC": "Long"
          "EXECUTION_TIMEOUT": "300", // in seconds
      },
      "test": {
         "id": "Job_Execution",
         "version": "0.0.0"
      }
   }
]
```

Weitere Informationen zum Erstellen und Verwenden von Auftragsdokumenten finden Sie im Auftragsdokument.

Berechtigungen und Richtlinien

Mithilfe der folgenden Tests können Sie feststellen, ob die mit den Zertifikaten Ihrer Geräte verknüpften Richtlinien den bewährten Standardmethoden entsprechen.

WebSocketMQTT-Over wird derzeit nicht unterstützt.

„Device certificate attached policies don’t contain wildcards“

Überprüft, ob die mit einem Gerät verknüpften Berechtigungsrichtlinien bewährten Methoden entsprechen und dem Gerät nicht mehr Berechtigungen als erforderlich gewähren.

Definition des API-Testfalls:

> **Note**
>
> `EXECUTION_TIMEOUT` hat einen Standardwert von 1 Minute. Wir empfehlen, ein Timeout von mindestens 30 Sekunden festzulegen.

```
"tests":[
   {
        "name":"my_security_device_policies",
        "configuration": {
            // optional:
            "EXECUTION_TIMEOUT":"60"    // in seconds
        },
        "test": {
            "id": "Security_Device_Policies",
            "version": "0.0.0"
        }
    }
]
```

Tests mit langer Dauer

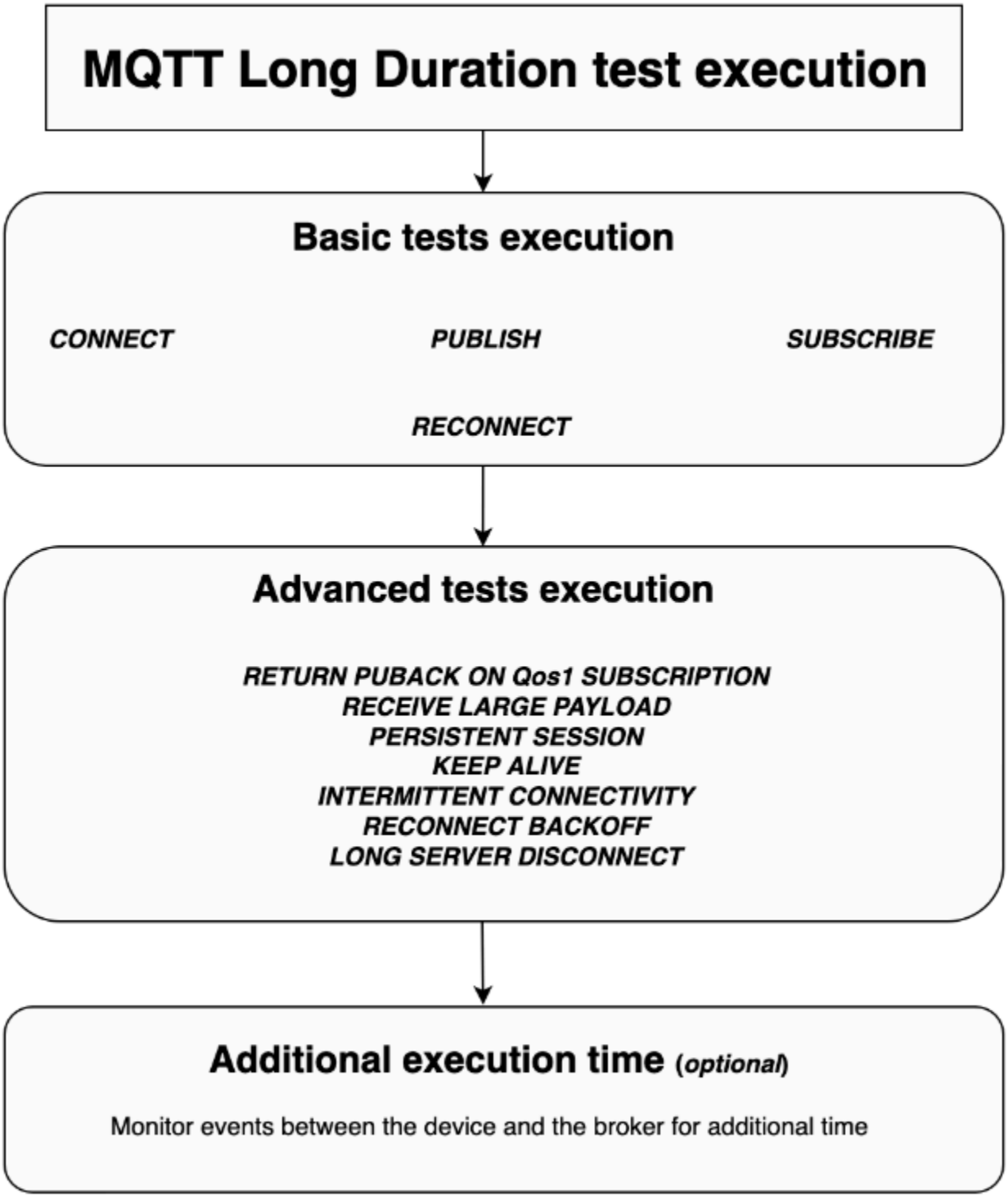
Tests mit langer Dauer sind eine neue Testsuite, die das Verhalten eines Geräts überwacht, wenn es über einen längeren Zeitraum betrieben wird. Im Vergleich zur Durchführung einzelner Tests, die sich auf bestimmte Verhaltensweisen eines Geräts konzentrieren, untersucht der Tests mit langer Dauer das Verhalten des Geräts in einer Vielzahl von realen Szenarien über die gesamte Lebensdauer des Geräts. Device Advisor orchestriert die Tests in der effizientesten Reihenfolge. Der Test generiert Ergebnisse und Protokolle, einschließlich eines Übersichtsprotokolls mit nützlichen Messwerten zur Leistung des Geräts während des Tests.

MQTT-Testfall mit langer Dauer

Im MQTT-Testfall mit langer Dauer wird das Verhalten des Geräts zunächst in Happy Case-Szenarien wie MQTT Connect, Subscribe, Publish und Reconnect beobachtet. Anschließend wird das Gerät in mehreren komplexen Ausfallszenarien wie MQTT Reconnect Backoff, Long Server Disconnect und Intermittent Connectivity beobachtet.

Ablauf der Ausführung von MQTT-Testfällen mit langer Dauer

Die Ausführung eines MQTT-Testfalls mit langer Dauer besteht aus drei Phasen:



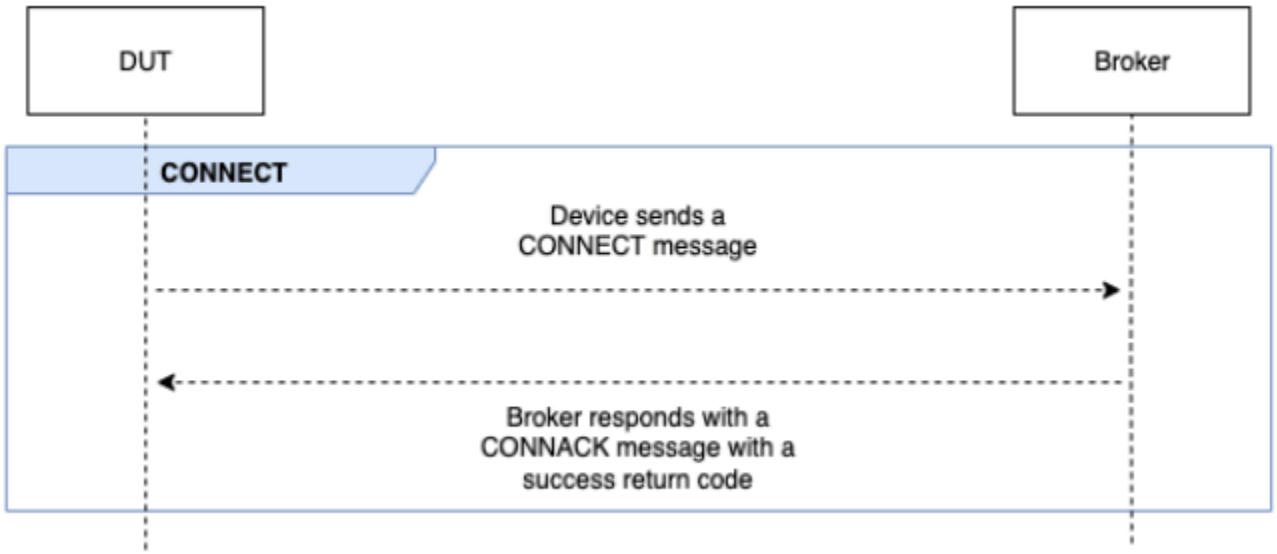
Grundlegende Testausführung

In dieser Phase führt der Testfall einfache Tests parallel durch. Der Test überprüft, ob für das Gerät die in der Konfiguration ausgewählten Operationen ausgewählt wurden.

Die grundlegenden Tests können je nach den ausgewählten Vorgängen Folgendes umfassen:

CONNECT

In diesem Szenario wird überprüft, ob das Gerät eine erfolgreiche Verbindung mit dem Broker herstellen kann.

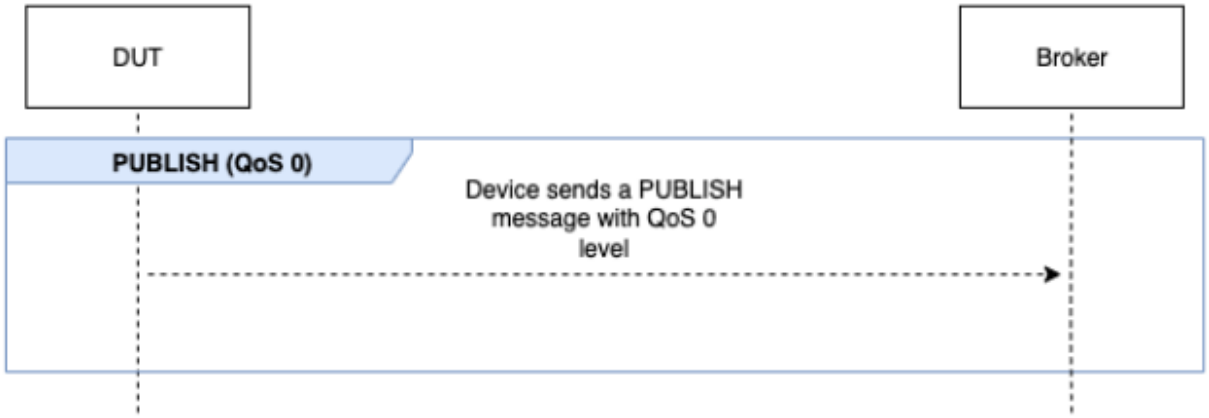


PUBLISH

In diesem Szenario wird überprüft, ob das Gerät erfolgreich beim Broker veröffentlicht.

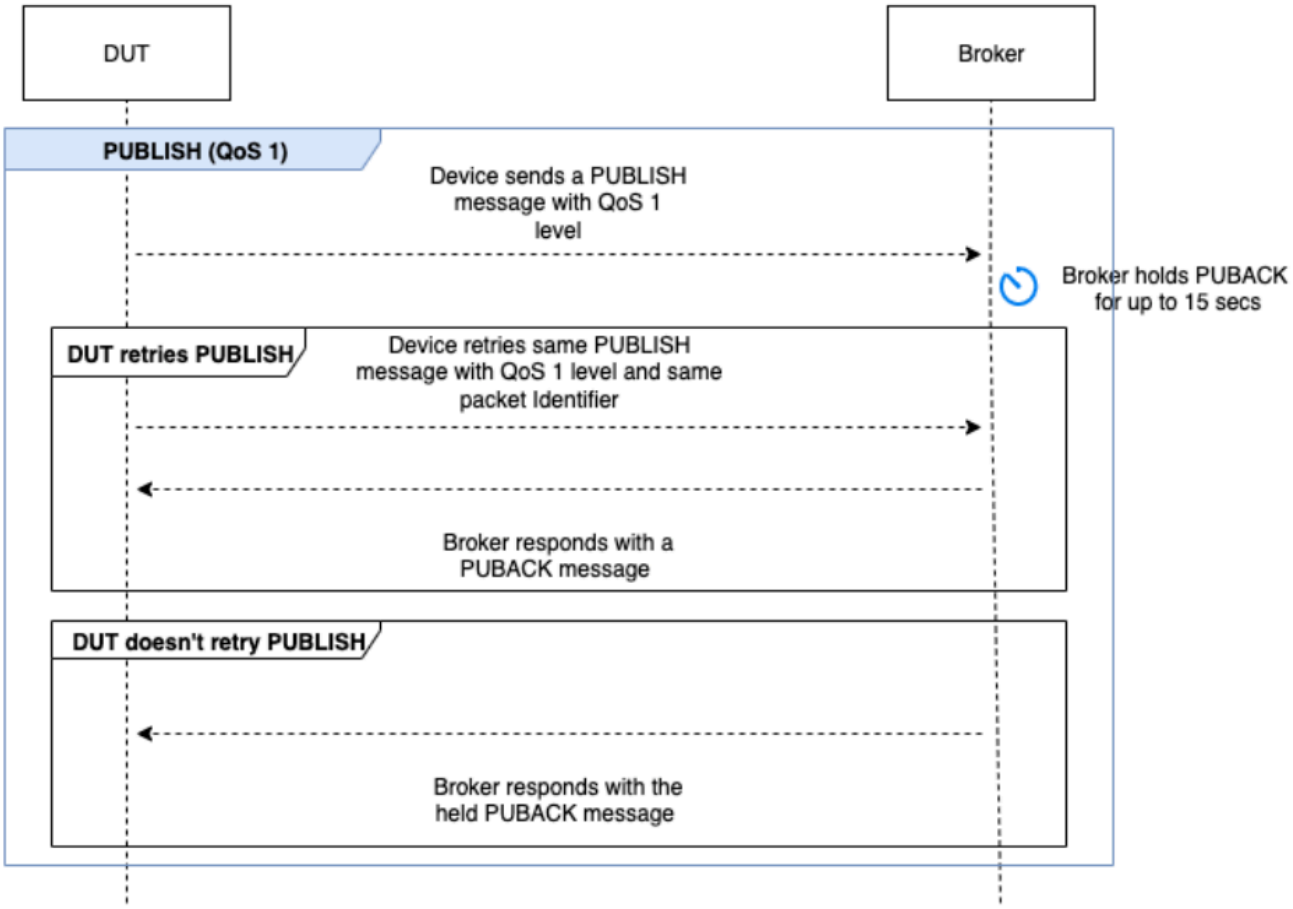
QoS 0

Dieser Testfall überprüft, ob das Gerät während einer Veröffentlichung mit QoS 0 erfolgreich eine `PUBLISH`-Nachricht an den Broker sendet. Der Test wartet nicht darauf, dass die `PUBACK`-Nachricht vom Gerät empfangen wird.



QoS 0

In diesem Testfall wird erwartet, dass das Gerät zwei PUBLISH-Nachrichten mit QoS 1 an den Broker sendet. Nach der ersten PUBLISH-Nachricht wartet der Broker bis zu 15 Sekunden, bevor er antwortet. Das Gerät muss innerhalb des 15-Sekunden-Fensters die ursprüngliche PUBLISH-Nachricht mit derselben Paket-ID erneut versuchen. Ist dies der Fall, antwortet der Broker mit einer PUBACK-Nachricht und der Test wird validiert. Wenn das Gerät den PUBLISH-Versuch nicht wiederholt, wird das Original-PUBACK an das Gerät gesendet und der Test wird als Pass with warnings (Mit Warnungen bestanden) markiert, zusammen mit einer Systemnachricht. Wenn das Gerät während der Testausführung die Verbindung verliert und die Verbindung wiederherstellt, wird der Testszenario ohne Fehler zurückgesetzt und das Gerät muss die Schritte des Testszenarios erneut ausführen.

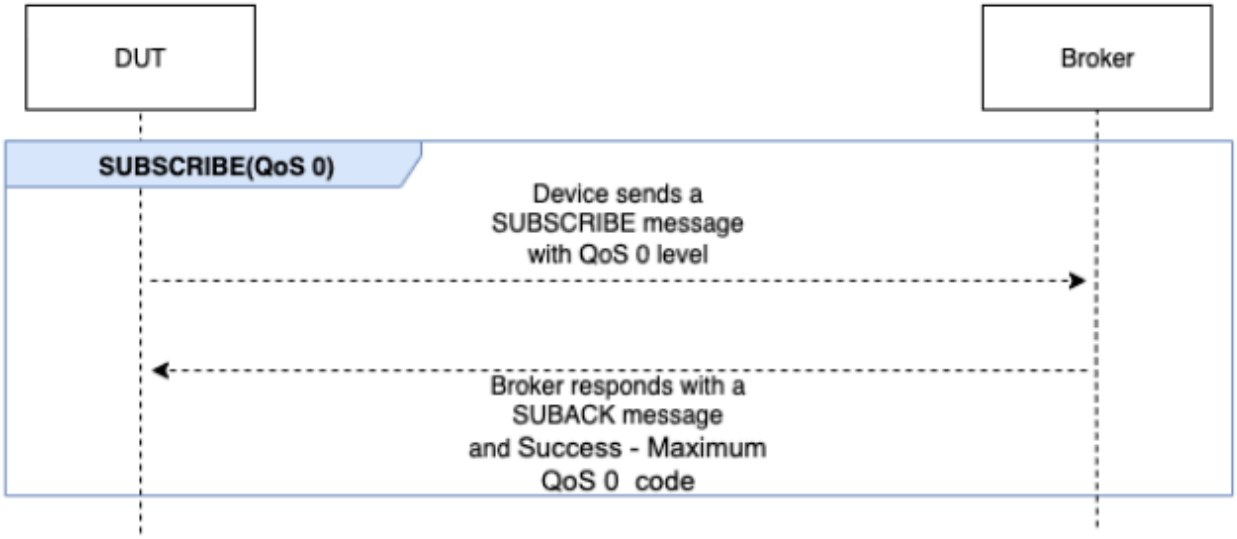


SUBSCRIBE

In diesem Szenario wird überprüft, ob das Gerät erfolgreich beim Broker abonniert.

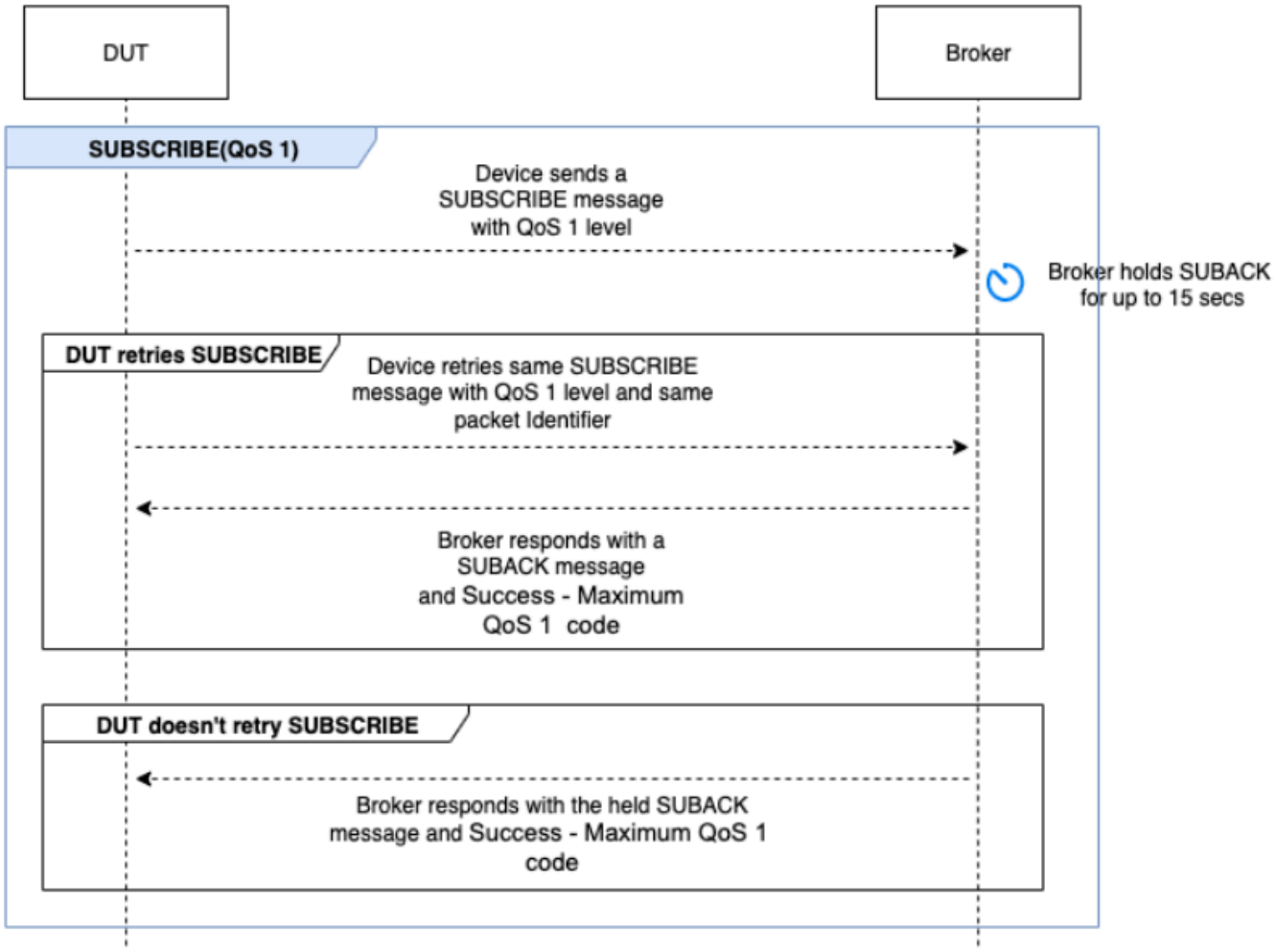
QoS 0

Dieser Testfall überprüft, ob das Gerät während eines Abonnements mit QoS 0 erfolgreich eine `SUBSCRIBE`-Nachricht an den Broker sendet. Der Test wartet nicht, bis das Gerät eine SUBACK-Nachricht erhält.



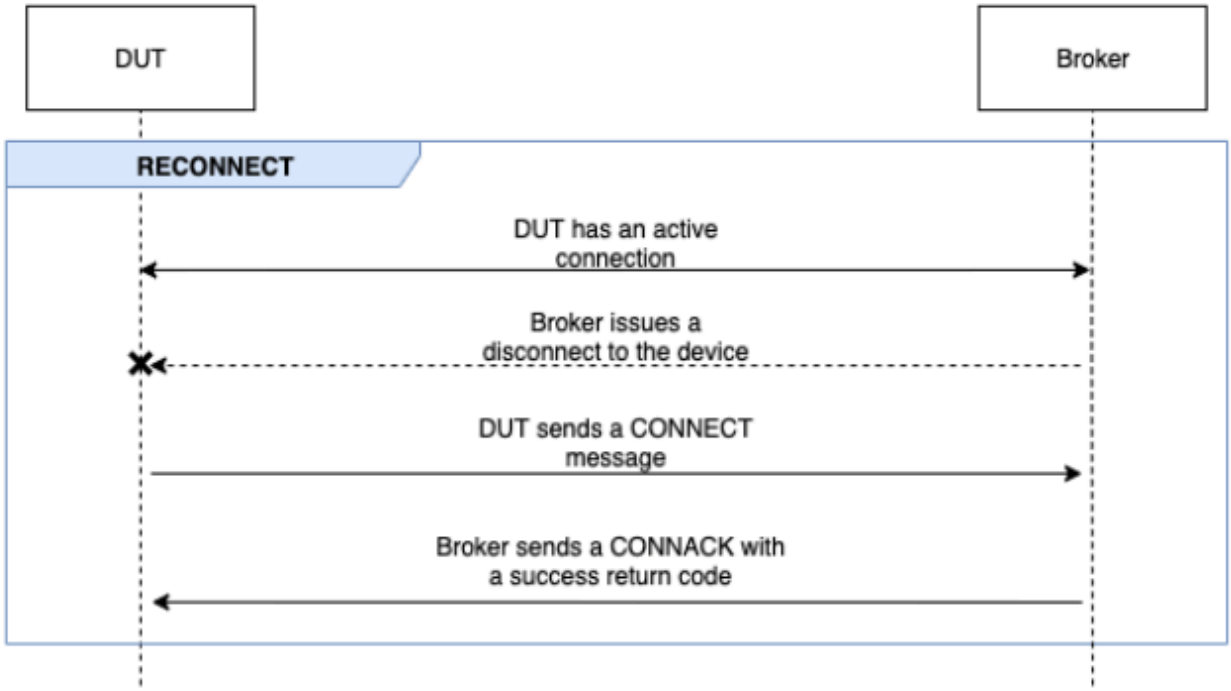
QoS 0

In diesem Testfall wird erwartet, dass das Gerät zwei SUBSCRIBE-Nachrichten mit QoS 1 an den Broker sendet. Nach der ersten SUBSCRIBE-Nachricht wartet der Broker bis zu 15 Sekunden, bevor er antwortet. Das Gerät muss innerhalb des 15-Sekunden-Fensters die ursprüngliche SUBSCRIBE-Nachricht mit derselben Paket-ID erneut versuchen. Ist dies der Fall, antwortet der Broker mit einer SUBACK-Nachricht und der Test wird validiert. Wenn das Gerät den SUBSCRIBE-Versuch nicht wiederholt, wird das Original-SUBACK an das Gerät gesendet und der Test wird als Pass with warnings (Mit Warnungen bestanden) markiert, zusammen mit einer Systemnachricht. Wenn das Gerät während der Testausführung die Verbindung verliert und die Verbindung wiederherstellt, wird der Testszenario ohne Fehler zurückgesetzt und das Gerät muss die Schritte des Testszenarios erneut ausführen.



RECONNECT

In diesem Szenario wird überprüft, ob das Gerät erfolgreich wieder eine Verbindung zum Broker herstellt, nachdem das Gerät von einer erfolgreichen Verbindung getrennt wurde. Device Advisor trennt das Gerät nicht, wenn es zuvor während der Testsuite mehr als einmal eine Verbindung hergestellt hat. Stattdessen wird der Test als Pass (Bestanden) markiert.



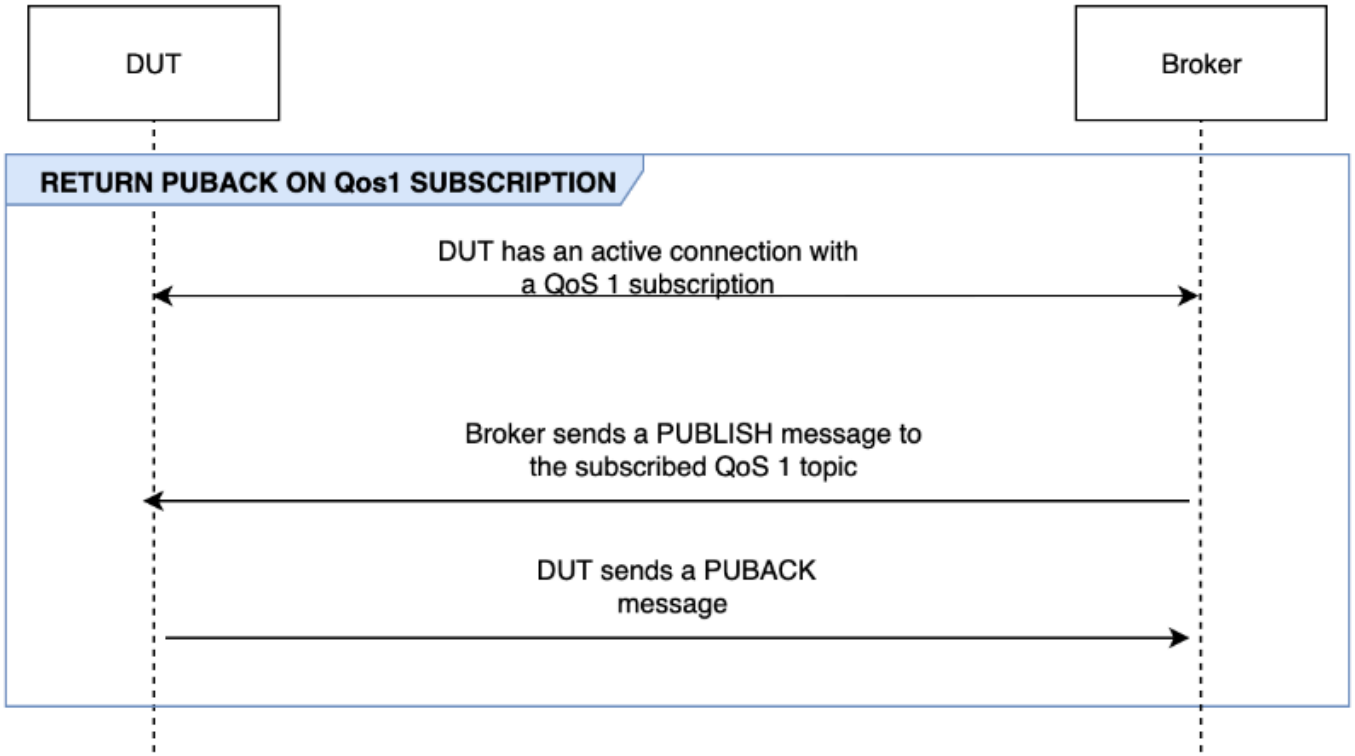
Erweiterte Testausführung

In dieser Phase führt der Testfall komplexere Tests seriell durch, um zu überprüfen, ob das Gerät den bewährten Methoden entspricht. Diese erweiterten Tests stehen zur Auswahl und können deaktiviert werden, falls sie nicht erforderlich sind. Jeder erweiterte Test hat seinen eigenen Timeout-Wert, der auf den Anforderungen des Szenarios basiert.

RETURN PUBACK ON QoS 1 SUBSCRIPTION

> **Note**
>
> Wählen Sie dieses Szenario nur aus, wenn Ihr Gerät QoS 1-Abonnements ausführen kann.

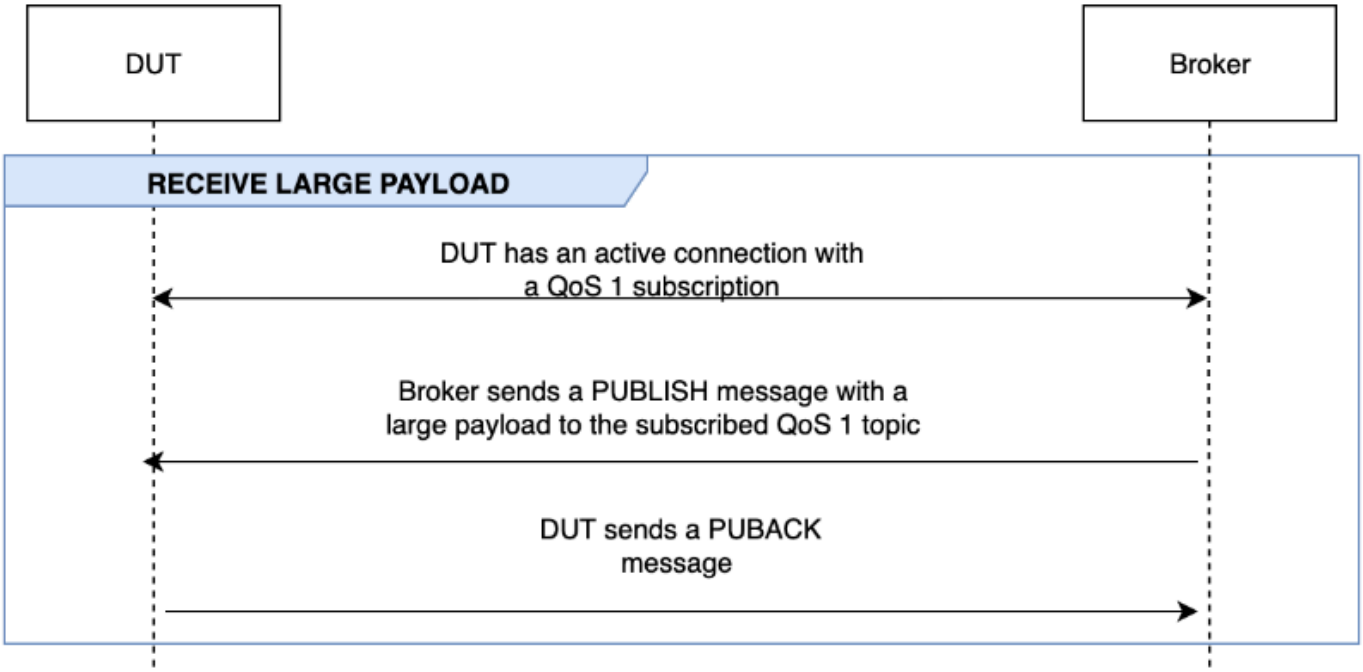
In diesem Szenario wird überprüft, ob das Gerät, nachdem es ein Thema abonniert und eine `PUBLISH`-Nachricht vom Broker erhalten hat, eine `PUBACK`-Nachricht zurückgibt.



RECEIVE LARGE PAYLOAD

> **Note**
>
> Wählen Sie dieses Szenario nur aus, wenn Ihr Gerät QoS 1-Abonnements ausführen kann.

In diesem Szenario wird überprüft, ob das Gerät mit einer `PUBACK`-Nachricht antwortet, nachdem es eine `PUBLISH`-Nachricht vom Broker für ein QoS 1-Thema mit einer großen Nutzlast erhalten hat. Das Format der erwarteten Nutzlast kann mit der `LONG_PAYLOAD_FORMAT`-Option konfiguriert werden.



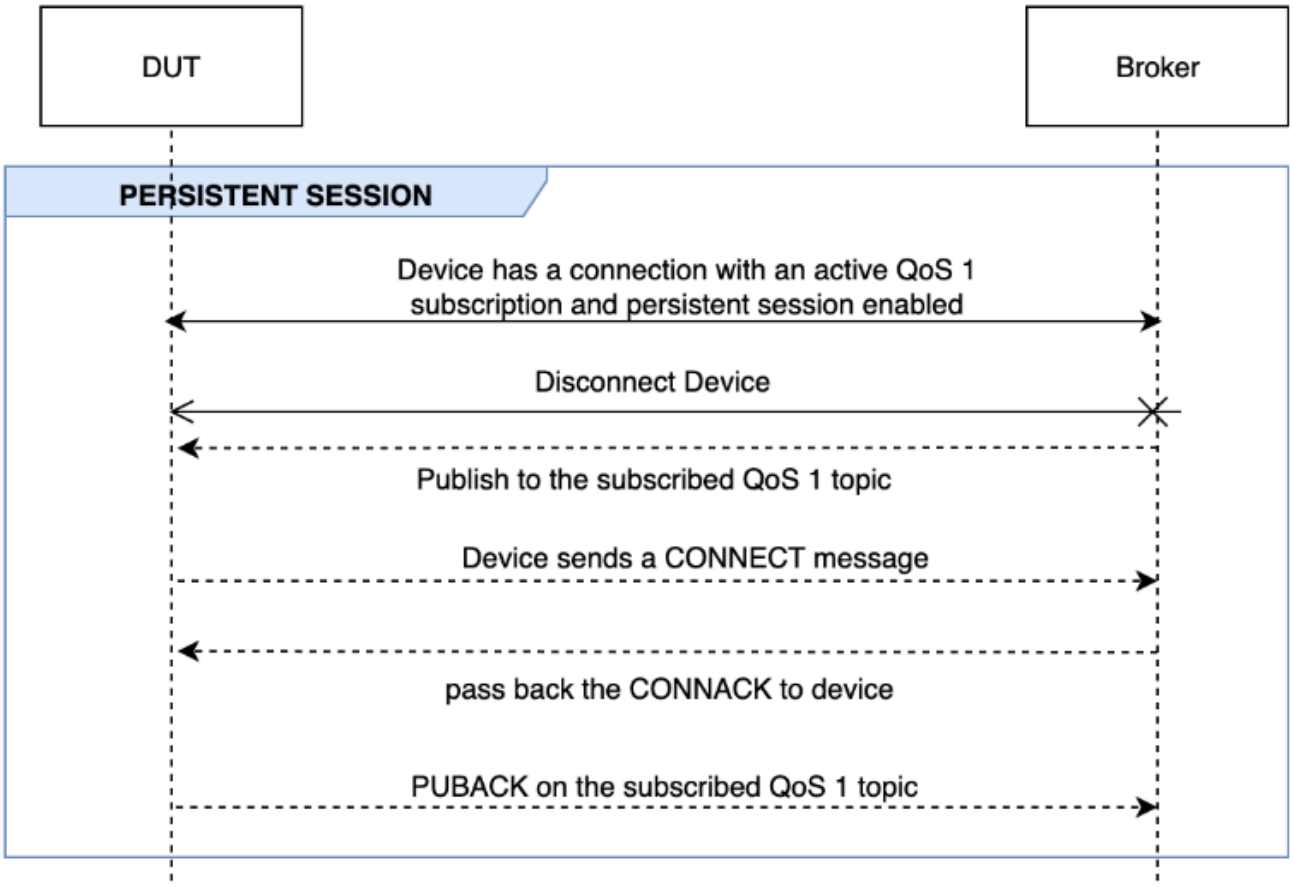
PERSISTENT SESSION

> **Note**
>
> Wählen Sie dieses Szenario nur aus, wenn Ihr Gerät QoS 1-Abonnements ausführen und eine persistente Sitzung aufrechterhalten kann.

In diesem Szenario wird das Geräteverhalten bei der Aufrechterhaltung persistenter Sitzungen validiert. Der Test validiert, wenn die folgenden Bedingungen erfüllt sind:

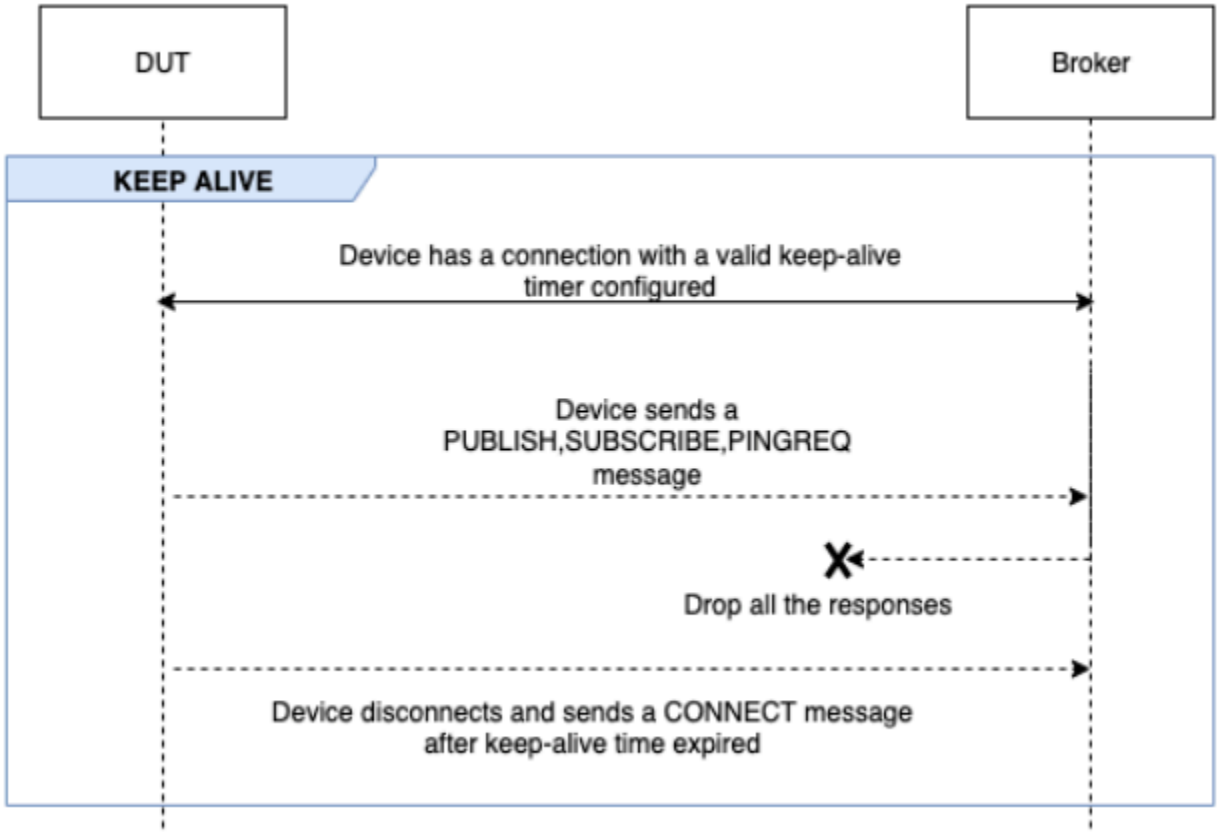
- Das Gerät stellt mit einem aktiven QoS 1-Abonnement und aktivierten persistenten Sitzungen eine Verbindung zum Broker her.
- Das Gerät trennt während der Sitzung erfolgreich die Verbindung zum Broker.
- Das Gerät stellt erneut eine Verbindung zum Broker her und nimmt die Abonnements für seine Trigger-Themen wieder auf, ohne diese Themen explizit erneut zu abonnieren.
- Das Gerät empfängt erfolgreich Nachrichten, die vom Broker für die abonnierten Themen gespeichert wurden, und läuft wie erwartet.

Weitere Informationen zu AWS IoT persistenten Sitzungen finden Sie unter Persistente MQTT-Sitzungen verwenden.



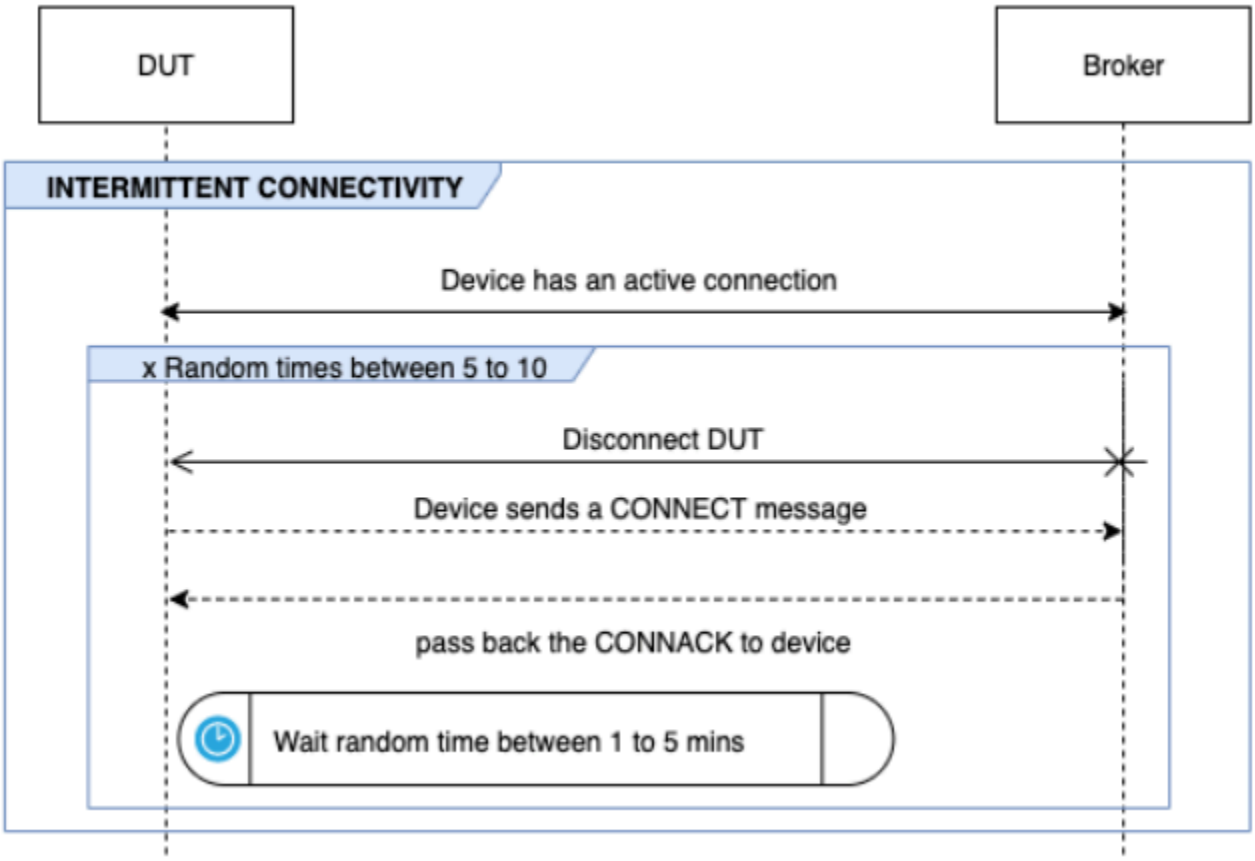
KEEP ALIVE

In diesem Szenario wird überprüft, ob das Gerät erfolgreich die Verbindung trennt, nachdem es keine Ping-Antwort vom Broker erhalten hat. Für die Verbindung muss ein gültiger Keep-Alive-Timer konfiguriert sein. Im Rahmen dieses Tests blockiert der Broker alle Antworten, die für `PUBLISH`-, `SUBSCRIBE`- und `PINGREQ`-Nachrichten gesendet wurden. Es überprüft auch, ob das getestete Gerät die MQTT-Verbindung unterbricht.



INTERMITTENT CONNECTIVITY

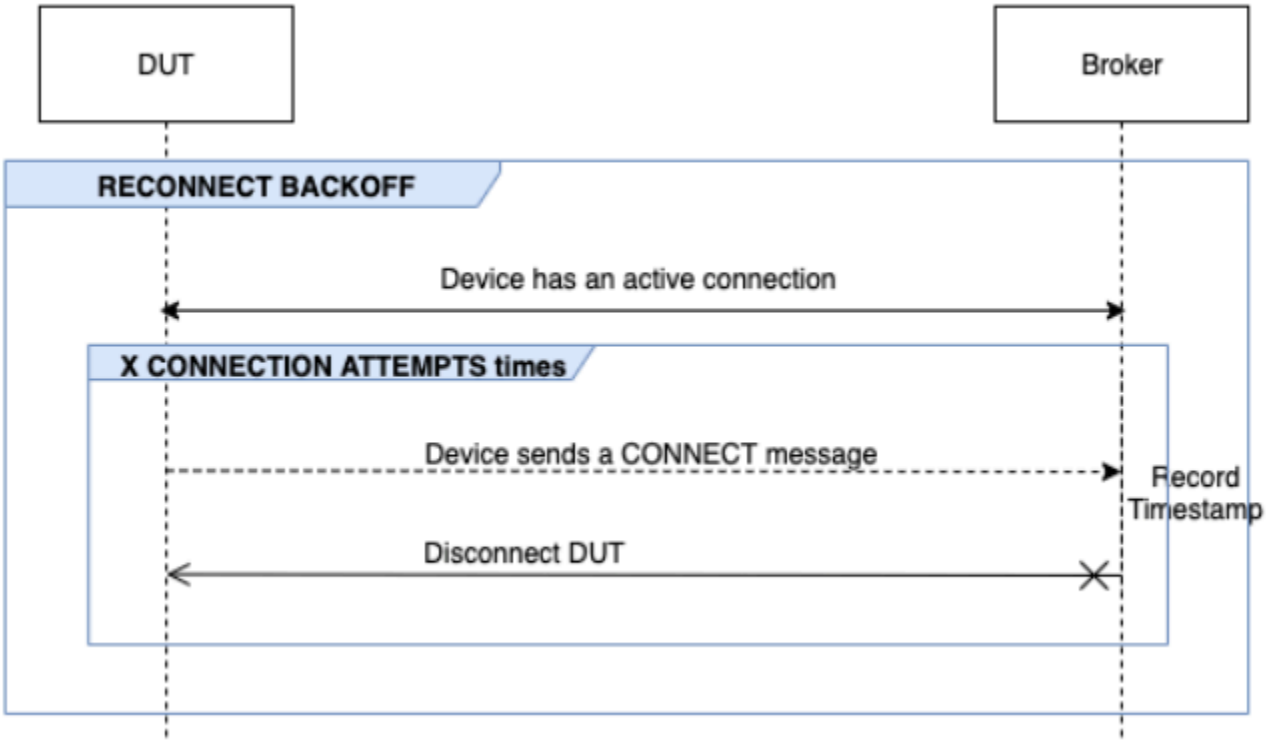
In diesem Szenario wird überprüft, ob das Gerät wieder eine Verbindung zum Broker herstellen kann, nachdem der Broker die Verbindung zum Gerät in zufälligen Intervallen für einen zufälligen Zeitraum getrennt hat.



RECONNECT BACKOFF

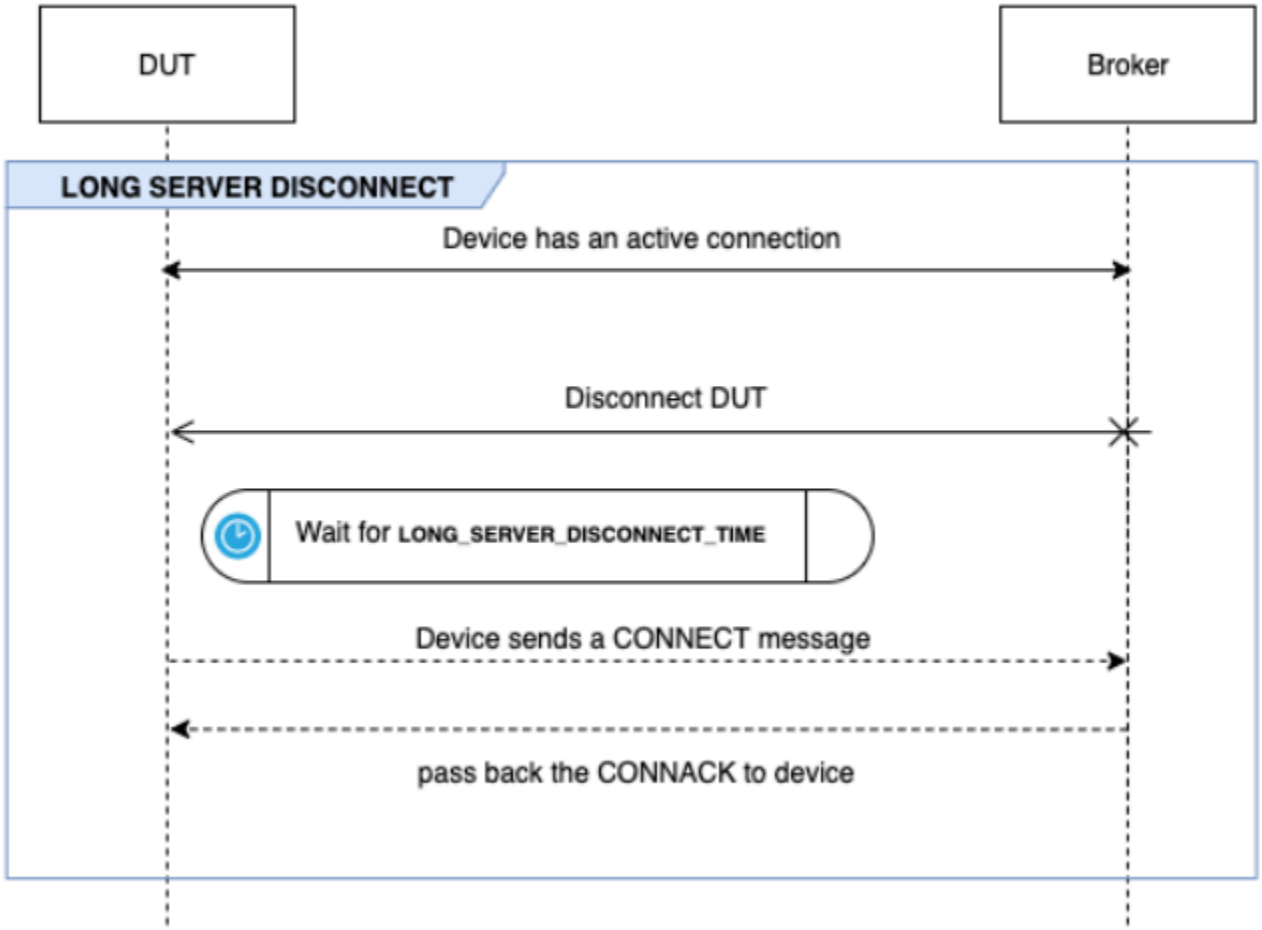
In diesem Szenario wird überprüft, ob auf dem Gerät ein Backoff-Mechanismus implementiert ist, wenn der Broker die Verbindung mehrmals trennt. Device Advisor meldet den Backoff-Typ als exponentiell, Jitter, linear oder konstant. Die Anzahl der Backoff-Versuche ist mit der `BACKOFF_CONNECTION_ATTEMPTS`-Option konfigurierbar. Der Standardwert ist 5. Der Wert ist zwischen 5 und 10 konfigurierbar.

Damit dieser Test bestanden wird, empfehlen wir, den Mechanismus Exponential Backoff And Jitter (Exponentielles Backoff und Jitter) auf dem getesteten Gerät zu implementieren.



LONG SERVER DISCONNECT

In diesem Szenario wird überprüft, ob das Gerät erfolgreich wieder eine Verbindung herstellen kann, nachdem der Broker die Verbindung zum Gerät über einen längeren Zeitraum (bis zu 120 Minuten) unterbrochen hat. Die Zeit für die Servertrennung kann mit der `LONG_SERVER_DISCONNECT_TIME`-Option konfiguriert werden. Der Standardwert beträgt 120 Minuten. Dieser Wert ist zwischen 30 und 120 Minuten konfigurierbar.



Zusätzliche Ausführungszeit

Die zusätzliche Ausführungszeit ist die Zeit, die der Test nach Abschluss aller oben genannten Tests und vor dem Beenden des Testfalls wartet. Kunden nutzen diesen zusätzlichen Zeitraum, um die gesamte Kommunikation zwischen dem Gerät und dem Broker zu überwachen und zu protokollieren. Die zusätzliche Ausführungszeit kann mit der `ADDITIONAL_EXECUTION_TIME`-Option konfiguriert werden. Standardmäßig ist diese Option auf 0 Minuten eingestellt und kann 0 bis 120 Minuten betragen.

Konfigurationsoptionen für MQTT-Tests mit langer Dauer

Alle für den MQTT-Test mit langer Dauer bereitgestellten Konfigurationsoptionen sind optional. Die folgenden Optionen sind verfügbar:

OPERATIONEN

Die Liste der Operationen, die das Gerät ausführt, wie `CONNECT`, `PUBLISH` Uand `SUBSCRIBE`. Im Testfall werden Szenarien ausgeführt, die auf den angegebenen Operationen basieren. Operationen, die nicht angegeben sind, werden als gültig vorausgesetzt.

```
{
"OPERATIONS": ["PUBLISH", "SUBSCRIBE"]
//by default the test assumes device can CONNECT
}
```

SZENARIEN

Basierend auf den ausgewählten Operationen führt der Testfall Szenarien aus, um das Verhalten des Geräts zu überprüfen. Es gibt zwei Arten von Szenarien:

- Bei Basisszenarien handelt es sich um einfache Tests, mit denen überprüft wird, ob das Gerät die oben als Teil der Konfiguration ausgewählten Vorgänge ausführen kann. Diese werden auf der Grundlage der in der Konfiguration angegebenen Operationen vorab ausgewählt. In der Konfiguration sind keine weiteren Eingaben erforderlich.
- Fortgeschrittene Szenarien sind komplexere Szenarien, die für das Gerät ausgeführt werden, um zu überprüfen, ob das Gerät unter realen Bedingungen bewährte Verfahren befolgt. Diese sind optional und können als eine Reihe von Szenarien an die Konfigurationseingabe der Testsuite übergeben werden.

```
{
    "SCENARIOS": [      // list of advanced scenarios
                "PUBACK_QOS_1",
                "RECEIVE_LARGE_PAYLOAD",
                "PERSISTENT_SESSION",
                "KEEP_ALIVE",
                "INTERMITTENT_CONNECTIVITY",
                "RECONNECT_BACK_OFF",
                "LONG_SERVER_DISCONNECT"
    ]
}
```

BASIC_TESTS_EXECUTION_TIME_OUT:

Die maximale Zeit, in der der Testfall auf den Abschluss aller Basistests wartet. Der Standardwert beträgt 60 Minuten. Dieser Wert ist zwischen 30 und 120 Minuten konfigurierbar.

LONG_SERVER_DISCONNECT_TIME:

Die Zeit, die der Testfall benötigt hat, um das Gerät während des Long-Server-Disconnect-Tests zu trennen und wieder zu verbinden. Der Standardwert beträgt 60 Minuten. Dieser Wert ist zwischen 30 und 120 Minuten konfigurierbar.

ADDITIONAL_EXECUTION_TIME:

Durch die Konfiguration dieser Option wird nach Abschluss aller Tests ein Zeitfenster zur Überwachung der Ereignisse zwischen dem Gerät und dem Broker bereitgestellt. Der Standardwert beträgt 0 Minuten. Dieser Wert ist zwischen 0 und 120 Minuten konfigurierbar.

BACKOFF_CONNECTION_ATTEMPTS:

Diese Option konfiguriert, wie oft das Gerät durch den Testfall getrennt wird. Dies wird vom Reconnect-Backoff-Test verwendet. Der Standardwert ist 5 Versuche. Dieser Wert ist von 5 bis 10 konfigurierbar.

LONG_PAYLOAD_FORMAT:

Das Format der Nachrichtennutzlast, die das Gerät erwartet, wenn der Testfall zu einem QoS 1-Thema veröffentlicht wird, das vom Gerät abonniert wurde.

Definition des API-Testfalls:

```
{
"tests":[
   {
      "name":"my_mqtt_long_duration_test",
      "configuration": {
         // optional
         "OPERATIONS": ["PUBLISH", "SUBSCRIBE"],
         "SCENARIOS": [
            "LONG_SERVER_DISCONNECT",
            "RECONNECT_BACK_OFF",
            "KEEP_ALIVE",
            "RECEIVE_LARGE_PAYLOAD",
            "INTERMITTENT_CONNECTIVITY",
            "PERSISTENT_SESSION",
         ],
         "BASIC_TESTS_EXECUTION_TIMEOUT": 60, // in minutes (60 minutes by default)
         "LONG_SERVER_DISCONNECT_TIME": 60,   // in minutes (120 minutes by default)
         "ADDITIONAL_EXECUTION_TIME": 60,     // in minutes (0 minutes by default)
```

```
         "BACKOFF_CONNECTION_ATTEMPTS": "5",
         "LONG_PAYLOAD_FORMAT":"{"message":"${payload}"}"
      },
      "test":{
         "id":"MQTT_Long_Duration",
         "version":"0.0.0"
      }
   }
 ]
}
```

Zusammenfassungsprotokoll des MQTT-Testfalls mit langer Dauer

Der MQTT-Testfall mit langer Dauer wird länger als normale Testfälle ausgeführt. Es wird ein separates Zusammenfassungsprotokoll bereitgestellt, das wichtige Ereignisse wie Geräteverbindungen, Veröffentlichungen und Abonnieren während der Ausführung auflistet. Zu den Details gehört, was getestet wurde, was nicht getestet wurde und was fehlgeschlagen ist. Am Ende des Protokolls enthält der Test eine Zusammenfassung aller Ereignisse, die während der Ausführung des Testfalls aufgetreten sind. Dies umfasst:

- Der Keep-Alive-Timer ist auf dem Gerät konfiguriert.
- Auf dem Gerät ist ein persistentes Sitzungs-Flag konfiguriert.
- Die Anzahl der Geräteverbindungen während des Testlaufs.
- Der Backoff-Typ für die Wiederverbindung des Geräts, sofern er für den Backoff-Test für die Wiederherstellung der Verbindung validiert wurde.
- Die Themen, zu denen das Gerät während der Testfallausführung veröffentlicht hat.
- Die Themen, die das Gerät während der Testfallausführung abonniert hat.

AWS IoT Core Standort des Geräts

Bevor Sie die Funktion „ AWS IoT Core Gerätestandort“ verwenden, lesen Sie sich die Nutzungsbedingungen für diese Funktion durch. Beachten Sie, dass AWS möglicherweise die Parameter Ihrer Geolokalisierungs-Suchanfrage, wie z. B. die für die Durchführung von Suchanfragen verwendeten Standortdaten, und andere Informationen an den von Ihnen ausgewählten Drittanbieter übertragen werden, der möglicherweise nicht dem entspricht AWS-Region , den Sie derzeit verwenden. Der Drittanbieter und der zu verwendende Solver werden auf der Grundlage der empfangenen Eingabe-Payload ausgewählt. Weitere Informationen finden Sie unter AWS -Servicebedingungen.

Verwenden Sie AWS IoT Core Device Location, um den Standort Ihrer IoT-Geräte mithilfe von Solvern von Drittanbietern zu testen. Solver sind von Drittanbietern bereitgestellte Algorithmen, die Messungsdaten auflösen und den Standort Ihres Geräts schätzen. Indem Sie den Standort Ihrer Geräte ermitteln, können Sie sie vor Ort verfolgen und debuggen, um etwaige Probleme zu beheben.

Die aus verschiedenen Quellen gesammelten Messungsdaten werden aufgelöst, und die Geolokalisierungsinformationen werden als GeoJSON-Nutzlast gemeldet. Das GeoJSON-Format ist ein Format, das zur Codierung geografischer Datenstrukturen verwendet wird. Die Nutzlast enthält die Breiten- und Längengradkoordinaten Ihres Gerätestandorts, die auf dem Koordinatensystem des World Geodetic Systems (WGS84) basieren.

Themen

- Messungstypen und Solver
- Wie AWS IoT Core Der Gerätestandort funktioniert
- Wie benutzt man AWS IoT Core Standort des Geräts
- Auflösen des Standorts von IoT-Geräten
- Auflösen des Gerätestandorts mit AWS IoT Core MQTT-Themen zum Gerätestandort
- Location Solver und Geräte-Payload
- Erweiterte Konfiguration für AWS IoT Core Standort des Geräts

Messungstypen und Solver

AWS IoT Core Device Location arbeitet mit Drittanbietern zusammen, um die Messdaten aufzulösen und einen geschätzten Gerätestandort zu ermitteln. In der folgenden Tabelle sind die Messungstypen und die Location Solver von Drittanbietern sowie Informationen zu unterstützten Geräten aufgeführt. Informationen zu LoRa WAN-Geräten und zur Konfiguration des Gerätestandorts für diese Geräte finden Sie unter Konfiguration der Position von LoRa WAN-Ressourcen.

> **Note**
>
> Allgemeine IoT-Geräte und Sidewalk-Geräte können die MQTT-Themen zum Gerätestandort verwenden, um die Standortinformationen abzurufen. Bei Wi-Fi den Messarten Mobilfunk und IP-Adresse kann AWS IoT Core Device Location den Standort des Geräts ermitteln, wenn die Geräte die Messdaten zu den reservierten Themen im definierten GeoJSON-Format veröffentlichen. Für GNSS- und Multi-Frame-GNSS-Messungen muss das Gerät über den LR11xx-Chip verfügen, der die Messdaten scannt, um die aufgelösten Ortsinformationen mit dem GNSS- oder Multi-Frame-GNSS-Solver zu erhalten. Informationen zum Abrufen von Standortinformationen für LoRa WAN-Geräte finden Sie unter Konfiguration der Position für WAN-Ressourcen in der Wireless-Dokumentation. LoRa AWS IoT

Messungstypen und Solver

Messungstyp	Third-party Solver	Unterstützte Geräte
Wi-Fi Zugriffspunkte	Wi-Fi basierter Solver	Allgemeine IoT-Geräte, LoRa WAN und Amazon Sidewalk-Geräte
Mobilfunkmasten: GSM, LTE, CDMA, SCDMA, WCMDA und Daten TD-SCDMA	Solver auf Mobilfunk basis	Allgemeine IoT-Geräte, LoRa WAN und Amazon Sidewalk-Geräte
IP-Adresse	IP-Reverse-Lookup-Solver	Jedes IoT-Gerät, das eine Verbindung herstellt TCP/IP
GNSS-Scandaten (NAV-Nach richten)	GNSS-Solver	Allgemeine IoT-Geräte, LoRa WAN und Amazon Sidewalk-Geräte

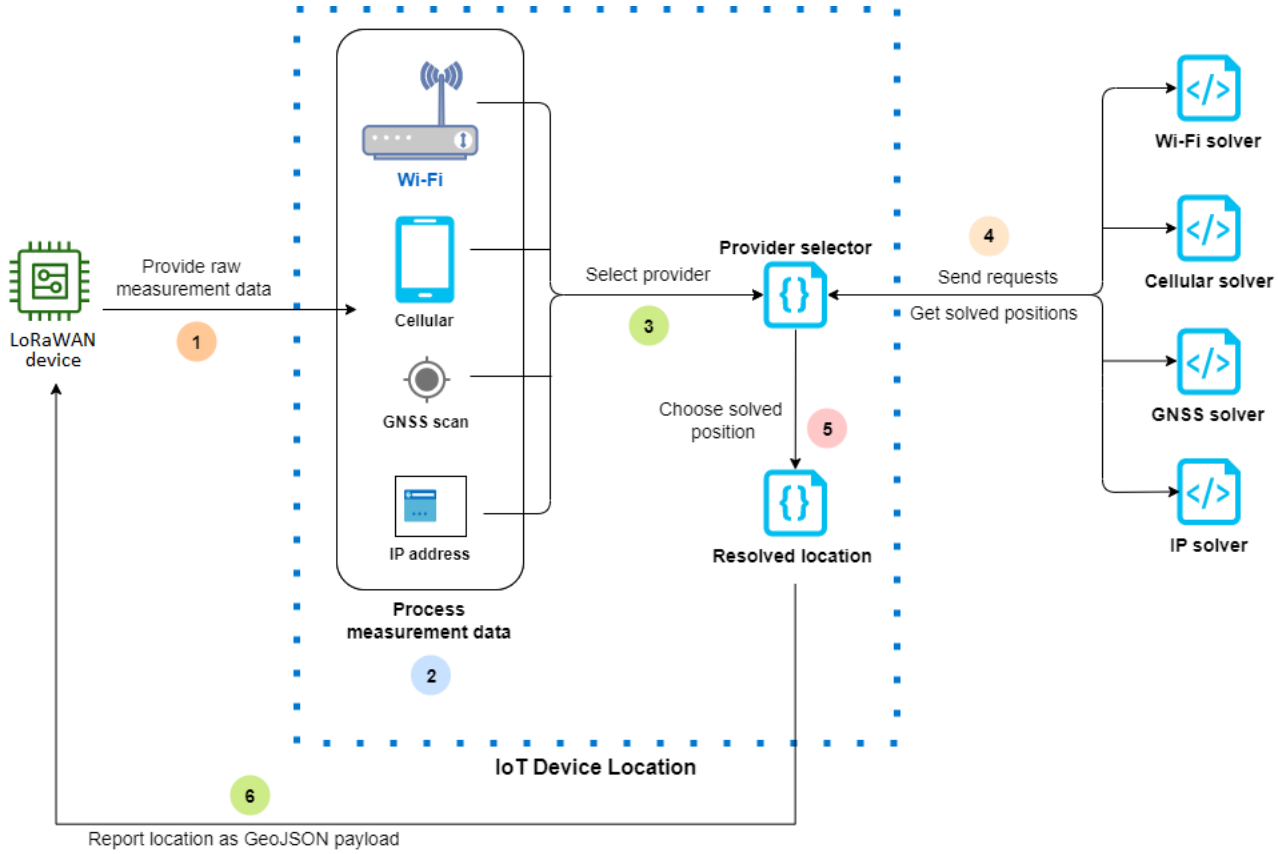
Messungstyp	Third-party Solver	Unterstützte Geräte
Multi-frame GNSS-Scandaten (mehrere NAV-Nachrichten von demselben Gerät)	Multi-frame GNSS-Solver	Allgemeine IoT-Geräte
Bluetooth Low Energy (BLE)	BLE-basierter Solver	Amazon-Sidewalk-Geräte

> **Note**
>
> Die Messtypen GNSS und Multi-Frame-GNSS schließen sich gegenseitig aus. Bei einer einzelnen Anfrage kann der `Gnss` Messtyp verwendet werden, um den Standort anhand einer einzigen NAV-Meldungserfassung zu ermitteln, oder der `GnssMultiFrame` Messtyp, um den Standort anhand mehrerer NAV-Nachrichtenerfassungen von demselben Gerät aufzulösen, aber es kann nicht beides in derselben Anfrage enthalten.

Weitere Informationen zu den Location Solvern und Beispiele, welche die Gerätenutzlast für die verschiedenen Messungstypen zeigen, finden Sie unter Location Solver und Geräte-Payload.

Wie AWS IoT Core Der Gerätestandort funktioniert

Das folgende Diagramm zeigt, wie AWS IoT Core Device Location Messdaten sammelt und die Standortinformationen Ihrer Geräte auflöst.



Die folgenden Schritte zeigen, wie der AWS IoT Core Gerätestandort funktioniert.

1. Empfangen von Messungsdaten

   Die Rohmessungsdaten, die sich auf Ihren Gerätestandort beziehen, werden zuerst vom Gerät gesendet. Die Messungsdaten werden als JSON-Nutzlast angegeben.

2. Prozessmessungsdaten

   Die Messdaten werden verarbeitet, und AWS IoT Core Device Location wählt die zu verwendenden Messdaten aus. Dabei kann es sich um Mobilfunk- Wi-Fi, GNSS-Scan-, Multi-Frame-GNSS-Scan oder IP-Adressinformationen handeln.

3. Auswählen des Solver

   Der Solver eines Drittanbieters wird auf der Grundlage der Messungsdaten ausgewählt. Wenn die Messdaten beispielsweise IP-Adressinformationen enthalten, werden der Wi-Fi Solver Wi-Fi und der IP-Reverse-Lookup-Solver ausgewählt.

4. Abrufen des gelösten Standorts

   Eine API-Anfrage wird an die Solver-Anbieter gesendet, mit der Bitte, den Standort zu ermitteln. AWS IoT Core Der Gerätestandort ruft dann die geschätzten Geolokalisierungsinformationen von den Solvern ab.

5. Auswählen des aufgelösten Standort

   Die aufgelösten Standortinformationen und ihre Genauigkeit werden verglichen, und der AWS IoT Core Gerätestandort wählt die Geolokalisierungsergebnisse mit der höchsten Genauigkeit aus.

6. Ausgabe der Standortinformationen

   Die Geolokalisierungsinformationen werden Ihnen als GeoJSON-Nutzlast gesendet. Die Nutzlast enthält die WGS84-Geokoordinaten, die Genauigkeitsinformationen, den Zuverlässigkeitsgrad und den Zeitstempel, zu dem der gelöste Standort abgerufen wurde.

Wie benutzt man AWS IoT Core Standort des Geräts

Die folgenden Schritte zeigen, wie Sie den AWS IoT Core Gerätestandort verwenden.

1. Bereitstellen von Messungsdaten

   Geben Sie die Rohmessungsdaten, die sich auf den Standort Ihres Geräts beziehen, als JSON-Nutzlast an. Um die Nutzlast-Messdaten abzurufen, gehen Sie zu Ihren Geräteprotokollen oder verwenden Sie CloudWatch Protokolle und kopieren Sie die Nutzdateninformationen. Die JSON-Nutzlast muss mindestens eine Art von Messungsdaten enthalten. Beispiele, in denen das Nutzlastformat für verschiedene Solver gezeigt werden, finden Sie unter Location Solver und Geräte-Payload.

2. Auflösen von Standortinformationen

   Übergeben Sie mithilfe der Seite „Gerätestandort“ in der AWS IoT Konsole oder im GetPositionEstimateAPI-Vorgang die Payload-Messdaten und ermitteln Sie den Gerätestandort. AWS IoT Core Der Gerätestandort wählt dann den Solver mit der höchsten Genauigkeit aus und meldet den Gerätestandort. Weitere Informationen finden Sie unter Auflösen des Standorts von IoT-Geräten.

3. Kopieren von Standortinformationen

   Überprüfen Sie die Geolokalisierungsinformationen, die durch AWS IoT Core Device Location aufgelöst und als GeoJSON-Payload gemeldet wurden. Sie können die Payload kopieren, um sie mit Ihren Anwendungen und anderen zu verwenden. AWS-Service Beispielsweise können Sie Ihre geografischen Standortdaten mithilfe der Speicherort AWS IoT Regelaktion an Amazon Location Service senden.

In den folgenden Themen wird die Verwendung von AWS IoT Core Device Location und Beispiele für Payloads zum Gerätestandort beschrieben.

- Auflösen des Standorts von IoT-Geräten
- Location Solver und Geräte-Payload

Auflösen des Standorts von IoT-Geräten

Verwenden Sie den AWS IoT Core Gerätestandort, um die Messdaten Ihrer Geräte zu dekodieren und den Gerätestandort mithilfe von Solvern von Drittanbietern zu ermitteln. Der aufgelöste Standort wird als GeoJSON-Nutzlast mit Geokoordinaten und Genauigkeitsinformationen generiert. Sie können den Standort Ihres Geräts über die AWS IoT Konsole, die AWS IoT Wireless-API oder ermitteln. AWS CLI

Themen

- Auflösen des Gerätestandorts (Konsole)
- Auflösen des Gerätestandorts (API)
- Fehlerbehebung beim Auflösen des Standorts

Auflösen des Gerätestandorts (Konsole)

So lösen Sie den Gerätestandort auf (Konsole)

1. Rufen Sie in der AWS IoT Konsole die Seite „Gerätestandort“ auf.
2. Rufen Sie die Nutzlast-Messdaten aus Ihren Geräteprotokollen oder aus den CloudWatch Protokollen ab und geben Sie sie im Abschnitt Position über Payload auflösen ein.

Der folgende Code zeigt ein Beispiel für eine JSON-Nutzlast. Die Nutzlast enthält Mobilfunk- und Wi-Fi Messdaten. Wenn Ihre Nutzlast zusätzliche Arten von Messungsdaten enthält, wird der Solver mit der besten Genauigkeit verwendet. Weitere Informationen und Beispiele zur Nutzlast finden Sie unter the section called “Location Solver und Geräte-Payload”.

> **Note**
>
> Die JSON-Nutzlast muss mindestens eine Art von Messungsdaten enthalten.

```
{
    "Timestamp": 1664313161,
    "Ip":{
        "IpAddress": "54.240.198.35"
    },
    "WiFiAccessPoints": [{
        "MacAddress": "A0:EC:F9:1E:32:C1",
        "Rss": -77
    }],
    "CellTowers": {
        "Gsm": [{
            "Mcc": 262,
            "Mnc": 1,
            "Lac": 5126,
            "GeranCid": 16504,
            "GsmLocalId": {
                "Bsic": 6,
                "Bcch": 82
            },
            "GsmTimingAdvance": 1,
            "RxLevel": -110,
            "GsmNmr": [{
                "Bsic": 7,
                "Bcch": 85,
                "RxLevel": -100,
                "GlobalIdentity": {
                    "Lac": 1,
                    "GeranCid": 1
                }
            }]
        }],
```

```
        "Wcdma": [{
            "Mcc": 262,
            "Mnc": 7,
            "Lac": 65535,
            "UtranCid": 14674663,
            "WcdmaNmr": [{
                    "Uarfcndl": 10786,
                    "UtranCid": 14674663,
                    "Psc": 149
                },
                {
                    "Uarfcndl": 10762,
                    "UtranCid": 14674663,
                    "Psc": 211
                }
            ]
        }],
        "Lte": [{
            "Mcc": 262,
            "Mnc": 2,
            "EutranCid": 2898945,
            "Rsrp": -50,
            "Rsrq": -5,
            "LteNmr": [{
                    "Earfcn": 6300,
                    "Pci": 237,
                    "Rsrp": -60,
                    "Rsrq": -6,
                    "EutranCid": 2898945
                },
                {
                    "Earfcn": 6300,
                    "Pci": 442,
                    "Rsrp": -70,
                    "Rsrq": -7,
                    "EutranCid": 2898945
                }
            ]
        }]
    },
    "AdvancedConfiguration": {
        "WiFiCellular": {
            "ConfidencePercent": 68
        }
```

```
    }
}
```

3. Um die Standortinformationen aufzulösen, wählen Sie Auflösen aus.

   Die Standortinformationen sind vom Typ BLOB und werden als Nutzlast zurückgegeben, die das GeoJSON-Format verwendet. Dieses Format wird für die Kodierung geografischer Datenstrukturen verwendet. Die Nutzlast enthält Folgendes:

   - Die WGS84-Geokoordinaten, welche die Breiten- und Längengradinformationen enthalten. Sie kann auch eine Höheninformation enthalten.
   - Den Typ der gemeldeten Standortinformationen, z. B. Punkt. Ein Punkt-Standorttyp stellt den Standort als WGS84-Breitengrad und -Längengrad dar, codiert als GeoJSON-Punkt.
   - Die horizontalen und vertikalen Genauigkeitsinformationen, die den Unterschied in Metern zwischen den von den Solvern geschätzten Standortinformationen und dem tatsächlichen Gerätestandort angeben.
   - Den Zuverlässigkeitsgrad, der die Unsicherheit in der bereitgestellten Standortschätzung angibt. Der Standardwert ist 0,68, was auf eine Wahrscheinlichkeit von 68 % hinweist, dass der tatsächliche Gerätestandort innerhalb des Unsicherheitsradius des geschätzten Standorts liegt. Sie können diesen Wert mithilfe des `AdvancedConfiguration` Parameters anpassen. Weitere Informationen finden Sie unter Erweiterte Konfiguration für AWS IoT Core Standort des Geräts.
   - Die Stadt, das Bundesland, das Land und die Postleitzahl, in der sich das Gerät befindet. Diese Informationen werden nur gemeldet, wenn der IP-Reverse-Lookup-Solver verwendet wird.
   - Die Informationen zum Zeitstempel, die dem Datum und der Uhrzeit entspricht, an denen der Standort aufgelöst wurde. Es wird das UNIX-Zeitstempelformat verwendet.

   Der folgende Code zeigt ein Beispiel für eine GeoJSON-Nutzlast, die bei der Auflösung des Standorts zurückgegeben wurde.

   > **Note**
   >
   > Wenn AWS IoT Core Device Location beim Versuch, den Standort zu ermitteln, Fehler meldet, können Sie die Fehler beheben und den Standort ermitteln. Weitere Informationen finden Sie unter Fehlerbehebung beim Auflösen des Standorts.

```
{
    "coordinates": [
        13.376077,
        52.51823,
        80.1
    ],
    "type": "Point",
    "properties": {
        "verticalAccuracy": 45,
        "verticalConfidenceLevel": 0.68,
        "horizontalAccuracy": 303,
        "horizontalConfidenceLevel": 0.68,
        "timestamp": "2026-04-07T01:16:39.971767716Z"
    }
}
```


4. Gehen Sie zum Abschnitt Ressourcenstandort und überprüfen Sie die vom AWS IoT Core Gerätestandort gemeldeten Geolokalisierungsinformationen. Sie können die Payload kopieren, um sie mit anderen Anwendungen und Apps zu verwenden. AWS-Service Beispielsweise können Sie Ihre geografischen Standortdaten mithilfe der Speicherort-Regelaktion an Amazon Location Service senden.

Auflösen des Gerätestandorts (API)

Verwenden Sie den API-Vorgang oder den CLI-Befehl get-position-estimate, um den Gerätestandort mithilfe der AWS IoTGetPositionEstimateWireless-API aufzulösen. Geben Sie die Nutzlast-Messungsdaten als Eingabe an und führen Sie die API-Operation aus, um den Gerätestandort aufzulösen.

> **Note**
>
> Die API-Operation `GetPositionEstimate` speichert keine Geräte- oder Statusinformationen und kann nicht zum Abrufen historischer Standortdaten verwendet werden. Er führt eine einmalige Operation durch, bei der die Messungsdaten aufgelöst und der geschätzte Standort erstellt werden. Um die Standortinformationen abzurufen, müssen Sie die Daten zur Nutzlast jedes Mal angeben, wenn Sie diese API-Operation ausführen.

Der folgende Befehl zeigt ein Beispiel für die Auflösung des Standorts mithilfe dieser API-Operation.

> **Note**
>
> Wenn Sie den CLI-Befehl `get-position-estimate` ausführen, müssen Sie die JSON-Ausgabedatei als erste Eingabe angeben. Diese JSON-Datei speichert die geschätzten Standortinformationen, die als Antwort von der CLI erhalten wurden, im GeoJSON-Format. Der folgende Befehl speichert beispielsweise die Standortinformationen in der Datei. *`locationout.json`*

> **Note**
>
> Der `GnssMultiFrame` Messtyp und der `AdvancedConfiguration` Parameter sind nur in AWS CLI Version 2 verfügbar.
> AWS CLI Version 1 befindet sich im Wartungsmodus und der Support wird bald eingestellt, und die in Version 2 eingeführten Funktionen werden nicht auf Version 1 zurückportiert. Um sie mit dem zu verwenden AWS CLI, migrieren Sie auf AWS CLI Version 2.

```
aws iotwireless get-position-estimate locationout.json \
    --ip IpAddress="54.240.198.35" \
    --wi-fi-access-points \
        MacAddress="A0:EC:F9:1E:32:C1",Rss=-75 \
        MacAddress="A0:EC:F9:15:72:5E",Rss=-67 \
    --advanced-configuration \
        WiFiCellular={ConfidencePercent=68}
```

In diesem Beispiel sind sowohl Wi-Fi Access Points als auch IP-Adressen als Messtypen enthalten. AWS IoT Core Device Location wählt zwischen dem Wi-Fi Solver und dem IP-Reverse-Lookup-Solver und wählt den Solver mit der höheren Genauigkeit aus.

Der optionale `--advanced-configuration` Parameter enthält das `WiFiCellular` Objekt, zu dem auch das `ConfidencePercent` Feld zur Anpassung des Konfidenzniveaus für Wi-Fi Mobilfunk-Messdaten gehören. In diesem Beispiel `ConfidencePercent` ist der Wert auf gesetzt*68*, was auch der Standardwert ist, wenn dieser Parameter nicht angegeben ist. Weitere Informationen finden Sie unter Erweiterte Konfiguration für AWS IoT Core Standort des Geräts.

Der aufgelöste Standort wird als Nutzlast zurückgegeben, die das GeoJSON-Format verwendet. Dieses Format wird für die Kodierung geografischer Datenstrukturen verwendet. Es wird dann in der *locationout.json* Datei gespeichert. Die Nutzlast enthält die WGS84-Geokoordinaten zu Breiten- und Längengrad, Daten zur Genauigkeit und zum Zuverlässigkeitsgrad, den Datentyp des Standorts und den Zeitstempel, zu dem der Standort gelöst wurde.

```
{
    "coordinates": [
        13.376077,
        52.51823,
        80.1
    ],
    "type": "Point",
    "properties": {
        "verticalAccuracy": 45,
        "verticalConfidenceLevel": 0.68,
        "horizontalAccuracy": 303,
        "horizontalConfidenceLevel": 0.68,
        "timestamp": "2026-04-07T01:16:39.971767716Z"
    }
}
```

Fehlerbehebung beim Auflösen des Standorts

Wenn Sie versuchen, den Standort zu ermitteln, wird möglicherweise einer der folgenden Fehlercodes angezeigt. AWS IoT Core Der Gerätestandort generiert möglicherweise einen Fehler, wenn Sie den `GetPositionEstimate` API-Vorgang verwenden, oder es wird auf die Zeilennummer verwiesen, die dem Fehler in der AWS IoT Konsole entspricht.

- 400-Fehler

  Dieser Fehler weist darauf hin, dass das Format der Geräte-Payload-JSON nicht anhand des AWS IoT Core Gerätestandorts überprüft werden kann. Der Fehler kann aus folgenden Gründen auftreten:

  - Die JSON-Messungsdaten sind falsch formatiert.
  - Die Nutzlast enthält nur die Informationen zum Zeitstempel.
  - Die Messungsdatenparameter, wie z. B. die IP-Adresse, sind ungültig.

  Um diesen Fehler zu beheben, überprüfen Sie, ob Ihre JSON-Datei korrekt formatiert ist und Daten von mindestens einem Messungstyp als Eingabe enthält. Wenn die IP-Adresse ungültig ist, finden

Sie Informationen darüber, wie Sie eine gültige IP-Adresse zur Behebung des Fehlers angeben können, unter IP-Reverse-Lookup-Solver.

- 403-Fehler

  Dieser Fehler weist darauf hin, dass Sie nicht über die erforderlichen Berechtigungen verfügen, um den API-Vorgang auszuführen oder die AWS IoT Konsole zum Abrufen des Gerätestandorts zu verwenden. Um diesen Fehler zu beheben, stellen Sie sicher, dass Sie über die erforderlichen Berechtigungen verfügen, um diese Aktion auszuführen. Dieser Fehler kann auftreten, wenn Ihre AWS-Managementkonsole Sitzung oder Ihr AWS CLI Sitzungstoken abgelaufen sind. Um diesen Fehler zu beheben, aktualisieren Sie das Sitzungstoken, um das zu verwenden AWS CLI, oder melden Sie sich ab AWS-Managementkonsole und melden Sie sich dann mit Ihren Anmeldeinformationen an.

- 404-Fehler

  Dieser Fehler weist darauf hin, dass keine Standortinformationen gefunden oder anhand AWS IoT Core des Gerätestandorts behoben wurden. Der Fehler kann beispielsweise aufgrund unzureichender Daten in der Eingabe von Messungsdaten auftreten. Beispiel:

  - Die Informationen zur MAC-Adresse oder zum Mobilfunkmast reichen nicht aus.
  - Die IP-Adresse ist nicht verfügbar, um den Standort zu suchen und abzurufen.
  - Die GNSS-Nutzlast ist nicht ausreichend.
  - Die GNSS-Erfassungen mit mehreren Frames reichen nicht aus.

  Um den Fehler in solchen Fällen zu beheben, überprüfen Sie, ob Ihre Messungsdaten ausreichende Informationen enthalten, um den Gerätestandort zu ermitteln.

- 500-Fehler

  Dieser Fehler weist darauf hin, dass eine interne Serverausnahme aufgetreten ist, als AWS IoT Core Device Location versucht hat, den Standort aufzulösen. Um diesen Fehler zu beheben, aktualisieren Sie die Sitzung und versuchen Sie erneut, die zu lösenden Messungsdaten zu senden.

Auflösen des Gerätestandorts mit AWS IoT Core MQTT-Themen zum Gerätestandort

Sie können reservierte MQTT-Themen verwenden, um mit der AWS IoT Core Gerätestandortfunktion die neuesten Standortinformationen für Ihre Geräte abzurufen.

MQTT-Themen zum Format des Gerätestandorts

Für reservierte Themen für den AWS IoT Core Gerätestandort wird das folgende Präfix verwendet:

`$aws/device_location/{customer_device_id}/`

Um ein vollständiges Thema zu erstellen, ersetzen Sie zunächst *customer_device_id* durch Ihre eindeutige ID, mit der Sie Ihr Gerät identifizieren. Wir empfehlen, dass Sie das angeben`WirelessDeviceId`, z. B. für LoRa WAN- und Sidewalk-Geräte, und*thingName*, ob Ihr Gerät als AWS IoT Ding registriert ist. Anschließend fügen Sie das Thema mit dem Themenbereich hinzu, z. B. `get_position_estimate` oder `get_position_estimate/accepted`, wie im folgenden Abschnitt gezeigt.

> **Note**
>
> *{customer_device_id}* darf nur Buchstaben, Zahlen und Bindestriche enthalten. Wenn Sie Themen zum Gerätestandort abonnieren, können Sie nur das Pluszeichen (+) als Platzhalterzeichen verwenden. Sie können beispielsweise den Platzhalter + für *{customer_device_id}* verwenden, um die Standortinformationen für Ihre Geräte abzurufen. Wenn Sie das Thema `$aws/device_location/+/get_position_estimate/accepted` abonnieren, wird eine Nachricht mit den Standortinformationen für Geräte veröffentlicht, die mit einer beliebigen Geräte-ID übereinstimmen, sofern das Problem erfolgreich gelöst wurde.

Im Folgenden finden Sie die reservierten Themen, die für die Interaktion mit AWS IoT Core Device Location verwendet werden.

MQTT-Themen zum Gerätestandort

Thema	Zulässige Operationen	Description
$aws/device_location/*customer_device_id* /get_position_estimate	Veröffentlichen	Ein Gerät veröffentlicht zu diesem Thema, um die gescannten Rohmessdaten nach Gerätestandort aufzulösen. AWS IoT Core
$aws/device_location/*customer_*	Abonnieren	AWS IoT Core Der Gerätestandort veröffentlicht die Standortinformationen zu diesem

Thema	Zulässige Operationen	Description
device_id /get_position_estimate/accepted		Thema, wenn der Gerätestandort erfolgreich ermittelt wurde.
$aws/device_location/*customer_device_id* /get_position_estimate/rejected	Abonnieren	AWS IoT Core Der Gerätestandort veröffentlicht die Fehlerinformationen zu diesem Thema, wenn der Gerätestandort nicht aufgelöst werden kann.

Richtlinie für MQTT-Themen zum Gerätestandort

Um Nachrichten zu Themen zum Gerätestandort zu empfangen, muss Ihr Gerät eine Richtlinie verwenden, die es ihm ermöglicht, eine Verbindung zum AWS IoT Gerätegateway herzustellen und die MQTT-Themen zu abonnieren.

Im Folgenden finden Sie ein Beispiel für die Richtlinie, die für den Empfang von Nachrichten zu verschiedenen Themen erforderlich ist.

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/device_location/
device-123/get_position_estimate"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
```

```
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/device_location/
device-123/get_position_estimate/accepted",
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/device_location/
device-123/get_position_estimate/rejected"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
device_location/device-123/get_position_estimate/accepted",
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
device_location/device-123/get_position_estimate/rejected"
            ]
        }
    ]
}
```

Themen zum Gerätestandort und Nutzlast

Im Folgenden werden die Themen AWS IoT Core zum Gerätestandort, das Format ihrer Nachrichtennutzdaten und eine Beispielrichtlinie für jedes Thema beschrieben.

Topics

- /get_position_estimate
- /get_position_estimate/accepted
- /get_position_estimate/rejected

/get_position_estimate

Veröffentlichen Sie eine Nachricht zu diesem Thema, um die Rohmessdaten des Geräts abzurufen und nach AWS IoT Core Gerätestandort aufzulösen.

```
$aws/device_location/customer_device_id/get_position_estimate
```

AWS IoT Core Device Location antwortet mit der Veröffentlichung entweder auf /get_position_estimate/accepted oder/get_position_estimate/rejected.

> **Note**
>
> Bei der zu diesem Thema veröffentlichten Nachricht muss es sich um eine gültige JSON-Nutzlast handeln. Wenn die Eingabenachricht kein gültiges JSON-Format hat, erhalten Sie keine Antwort. Weitere Informationen finden Sie unter Nachrichtennutzlast.

Nachrichtennutzlast

Das Format der Nachrichtennutzdaten folgt einer ähnlichen Struktur wie der Anforderungstext für den AWS IoT Wireless-API-Vorgang, `GetPositionEstimate`. Sie enthält Folgendes:

- Eine optionale `Timestamp`-Zeichenfolge, die dem Datum und der Uhrzeit entspricht, an denen der Standort aufgelöst wurde. Die `Timestamp`-Zeichenfolge kann eine Mindestlänge von 1 und eine Maximallänge von 10 haben.
- Eine optionale `MessageId`-Zeichenfolge, die verwendet werden kann, um die Anforderung der Antwort zuzuordnen. Wenn Sie diese Zeichenfolge angeben, enthält die Nachricht, die in den Themen `get_position_estimate/accepted` oder `get_position_estimate/rejected` veröffentlicht wird, diese `MessageId`. Die `MessageID`-Zeichenfolge kann eine Mindestlänge von 1 und eine Maximallänge von 256 haben.
- Die Messungsdaten des Geräts, das mindestens einen der folgenden Messungstypen enthält:
  - WiFiAccessPoint
  - CellTowers
  - IpAddress
  - Gnss
  - GnssMultiFrame

Im Folgenden finden Sie ein Beispiel für eine Nachrichtennutzlast.

```
{
    "Timestamp": "1664313161",
    "MessageId": "ABCD1",
    "WiFiAccessPoints":  [
        {
```

```
            "MacAddress": "A0:EC:F9:1E:32:C1",
            "Rss": -66
        }
    ],
    "Ip":{
        "IpAddress": "54.192.168.0"
    },
    "Gnss":{
      "Payload":"8295A614A2029517F4F77C0A7823B161A6FC57E25183D96535E3689783F6CA48",
      "CaptureTime":1354393948
    }
}
```

Beispielrichtline

Im Folgenden finden Sie ein Beispiel für die erforderliche Richtlinie:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/device_location/
device-123/get_position_estimate"
            ]
        }
    ]
}
```

/get_position_estimate/accepted

AWS IoT Core Device Location veröffentlicht eine Antwort zu diesem Thema, wenn die aufgelösten Standortinformationen für Ihr Gerät zurückgegeben werden. Die Standortinformationen werden im GeoJSON-Format zurückgegeben.

```
$aws/device_location/customer_device_id/get_position_estimate/accepted
```

Im Folgenden werden die Nachrichtennutzlast und ein Beispiel für eine Richtlinie dargestellt.

Nachrichtennutzlast

Im Folgenden sehen Sie ein Beispiel für die Nachrichtennutzlast im GeoJSON-Format. Wenn Sie `MessageId` in Ihren Rohmessdaten ein angegeben haben und der AWS IoT Core Gerätestandort die Standortinformationen erfolgreich aufgelöst hat, gibt die Nachrichtennutzlast dieselben `MessageId` Informationen zurück.

```
{
    "coordinates": [
        13.37704086303711,
        52.51865005493164
    ],
    "type": "Point",
    "properties": {
        "verticalAccuracy": 707,
        "verticalConfidenceLevel": 0.68,
        "horizontalAccuracy": 389,
        "horizontalConfidenceLevel": 0.68,
        "country": "USA",
        "state": "CA",
        "city": "Sunnyvalue",
        "postalCode": "91234",
        "timestamp": "2022-11-18T14:03:57.391Z",
        "messageId": "ABCD1"
    }
}
```

Beispielrichtline

Im Folgenden finden Sie ein Beispiel für die erforderliche Richtlinie:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
```

```
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/
device_location/device-123/get_position_estimate/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/device_location/
device-123/get_position_estimate/accepted"
            ]
        }
    ]
}
```

/get_position_estimate/rejected

AWS IoT Core Device Location veröffentlicht eine Fehlermeldung zu diesem Thema, wenn der Gerätestandort nicht aufgelöst werden kann.

```
$aws/device_location/customer_device_id/get_position_estimate/rejected
```

Im Folgenden sehen Sie die Nachrichtennutzlast und ein Beispiel für die Richtlinie. Weitere Informationen zu den Fehlern finden Sie unter Fehlerbehebung beim Auflösen des Standorts.

Nachrichtennutzlast

Im Folgenden finden Sie ein Beispiel für die Nachrichten-Payload, die den Fehlercode und die Meldung enthält, aus der hervorgeht, warum AWS IoT Core Device Location die Standortinformationen nicht auflösen konnte. Wenn Sie `MessageId` bei der Bereitstellung Ihrer Rohmessdaten ein angegeben haben und der AWS IoT Core Gerätestandort die Standortinformationen nicht auflösen konnte, werden dieselben `MessageId` Informationen in der Nachrichtennutzlast zurückgegeben.

```
{
    "errorCode": 500,
    "errorMessage":"Internal server error",
    "messageId": "ABCD1"
```

```
}
```

Beispielrichtline

Im Folgenden finden Sie ein Beispiel für die erforderliche Richtlinie:

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/device_location/
device-123/get_position_estimate/rejected"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/$aws/device_location/device-123/
get_position_estimate/rejected"
      ]
    }
  ]
}
```

Location Solver und Geräte-Payload

Location Solver sind Algorithmen, mit denen der Standort Ihrer IoT-Geräte ermittelt werden kann. AWS IoT Core Device Location unterstützt die folgenden Location Solver. Sie sehen Beispiele für das Format der JSON-Nutzlast für diese Messungstypen, die vom Solver unterstützten Geräte und die Art und Weise, wie der Standort aufgelöst wird.

Um den Gerätestandort zu ermitteln, geben Sie mindestens einen dieser Messungsdatentypen an. Für alle Messungsdaten zusammen wird ein einziger, aufgelöster Standort zurückgegeben.

Themen

- Wi-Fi basierter Solver
- Solver auf Mobilfunkbasis
- IP-Reverse-Lookup-Solver
- GNSS-Solver
- Multi-frame GNSS-Solver
- BLE-Solver für Amazon Sidewalk-fähige Geräte

Wi-Fi basierter Solver

Verwenden Sie den Wi-Fi basierten Solver, um den Standort anhand der Scaninformationen von Wi-Fi Access Points zu ermitteln. Der Solver unterstützt die WLAN-Technologie und kann zur Berechnung des Gerätestandorts für allgemeine IoT-Geräte, die eine Verbindung herstellen TCP/IP, LoRa WAN-Wireless-Geräte und Amazon Sidewalk-fähige Geräte verwendet werden.

Die LoRa WAN-Geräte müssen über den LoRa Edge-Chipsatz verfügen, der die eingehenden Scaninformationen dekodieren kann. Wi-Fi LoRa Edge ist eine Plattform mit extrem geringem Stromverbrauch, die einen LoRa Transceiver mit großer Reichweite, einen GNSS-Scanner mit mehreren Konstellationen und einen passiven MAC-Scanner für Geolokalisierungsanwendungen integriert. Wi-Fi Wenn eine Uplink-Nachricht vom Gerät empfangen wird, werden die Wi-Fi Scandaten an den Gerätestandort gesendet, und der Standort wird auf AWS IoT Core der Grundlage der Scanergebnisse geschätzt. Wi-Fi Die dekodierten Informationen werden dann an den Wi-Fi basierten Solver weitergeleitet, um die Standortinformationen abzurufen.

Weitere Informationen zur Auflösung von Standortdaten für Amazon Sidewalk-fähige Geräte finden Sie im AWS IoT Core für Amazon Sidewalk Entwicklerhandbuch.

Wi-Fi Beispiel für eine basierte Solver-Payload

Der folgende Code zeigt ein Beispiel für die JSON-Nutzlast des Geräts, das die Messungsdaten enthält. Wenn AWS IoT Core Device Location diese Daten als Eingabe empfängt, sendet es eine HTTP-Anfrage an den Solver-Anbieter, um die Standortinformationen aufzulösen. Geben Sie zum Abrufen der Informationen Werte für die MAC-Adresse und RSS (Received Signal Strength) an.

Stellen Sie dazu entweder die JSON-Nutzlast in diesem Format bereit oder verwenden Sie den WiFiAccessPointsObjektparameter des GetPositionEstimateAPI-Vorgangs.

```
{
    "Timestamp": 1664313161,    // optional
    "WiFiAccessPoints": [
        {
            "MacAddress": "A0:EC:F9:1E:32:C1",  // required
            "Rss": -75                          // required
        }
    ]
}
```

Solver auf Mobilfunkbasis

Sie können den Solver auf Mobilfunkbasis verwenden, um den Standort anhand von Messungsdaten von Mobilfunkmasten aufzulösen. Der Solver unterstützt die folgenden Technologien. Es wird eine einzige aufgelöste Standortinformation abgerufen, auch wenn Sie Messungsdaten aus einer oder allen dieser Technologien einbeziehen.

- GSM
- CDMA
- WCDMA
- TD-SCDMA
- LTE

Beispiele für Solver-Nutzlasten auf Mobilfunkbasis

Der folgende Code zeigt Beispiele für die JSON-Nutzlast des Geräts, das die Messungsdaten auf Mobilfunkbasis enthält. Wenn AWS IoT Core Device Location diese Daten als Eingabe empfängt, sendet es eine HTTP-Anfrage an den Solver-Anbieter, um die Standortinformationen aufzulösen. Um die Informationen abzurufen, geben Sie entweder die JSON-Nutzlast in diesem Format in der Konsole an oder geben Werte für den CellTowersParameter des GetPositionEstimateAPI-Vorgangs an. Sie können die Messungsdaten bereitstellen, indem Sie Werte für Parameter angeben, die eine oder alle dieser Mobilfunktechnologien verwenden.

LTE (Long-term Weiterentwicklung)

Wenn Sie diese Messungsdaten verwenden, müssen Sie Informationen wie das Netzwerk und die Landesvorwahl des Mobilfunknetzes sowie optionale zusätzliche Parameter, einschließlich Informationen zur lokalen ID, angeben. Der nachfolgende Code zeigt ein Beispiel für das Nutzlastformat. Weitere Informationen zu diesen Parametern finden Sie unter LTE-Objekt.

```
{
    "Timestamp": 1664313161,           // optional
    "CellTowers": {
        "Lte": [
        {
          "Mcc": int,                   // required
          "Mnc": int,                   // required
          "EutranCid": int,             // required. Make sure that you use int for
 EutranCid.
          "Tac": int,                   // optional
          "LteLocalId": {               // optional
              "Pci": int,               // required
              "Earfcn": int,            // required
          },
          "LteTimingAdvance": int,      // optional
          "Rsrp": int,                  // optional
          "Rsrq": float,                // optional
          "NrCapable": boolean,         // optional
          "LteNmr": [                   // optional
                {
                    "Pci": int,         // required
                    "Earfcn": int,      // required
                    "EutranCid": int,   // required
                    "Rsrp": int,        // optional
                    "Rsrq": float       // optional
                }
            ]
         }
      ]
   }
}
```

GSM (Global System for Mobile Communications)

Wenn Sie diese Messungsdaten verwenden, müssen Sie Informationen wie das Netzwerk und die Landesvorwahl des Mobilfunknetzes, Informationen zur Basisstation und optionale weitere Parameter

angeben. Der nachfolgende Code zeigt ein Beispiel für das Nutzlastformat. Weitere Informationen zu diesen Parametern finden Sie unter GSM-Objekt.

```
{
    "Timestamp": 1664313161,           // optional
    "CellTowers": {
        "Gsm": [
        {
          "Mcc": int,                   // required
          "Mnc": int,                   // required
          "Lac": int,                   // required
          "GeranCid": int,              // required
          "GsmLocalId": {               // optional
              "Bsic": int,              // required
              "Bcch": int,              // required
          },
          "GsmTimingAdvance": int,      // optional
          "RxLevel": int,               // optional
          "GsmNmr": [                   // optional
            {
                "Bsic": int,            // required
                "Bcch": int,            // required
                "RxLevel": int,         // optional
                "GlobalIdentity": {
                    "Lac": int,         // required
                    "GeranCid": int     // required
                }
             }
          ]
       }
    ]
}
```

CDMA (Code-division Mehrfachzugriff)

Wenn Sie diese Messungsdaten verwenden, müssen Sie Informationen wie die Signalleistung und Identifikationsinformationen, Informationen zur Basisstation und optionale weitere Parameter angeben. Der nachfolgende Code zeigt ein Beispiel für das Nutzlastformat. Weitere Informationen zu diesen Parametern finden Sie unter CDMA-Objekt.

```
{
    "Timestamp": 1664313161,              // optional
```

```
    "CellTowers": {
        "Cdma": [
        {
            "SystemId": int,                // required
            "NetworkId": int,               // required
            "BaseStationId": int,           // required
            "RegistrationZone": int,        // optional
            "CdmaLocalId": {                // optional
              "PnOffset": int,              // required
              "CdmaChannel": int,           // required
            },
            "PilotPower": int,              // optional
            "BaseLat": float,               // optional
            "BaseLng": float,               // optional
            "CdmaNmr": [                    // optional
                {
                    "PnOffset": int,        // required
                    "CdmaChannel": int,     // required
                    "PilotPower": int,      // optional
                    "BaseStationId": int    // optional
                }
             ]
          }
       ]
    }
}
```

WCDMA (Wideband Code-Division Multiple Access)

Wenn Sie diese Messungsdaten verwenden, müssen Sie Informationen wie das Netzwerk und die Landesvorwahl, die Signalleistung und Identifikationsinformationen, Informationen zur Basisstation und optionale weitere Parameter angeben. Der nachfolgende Code zeigt ein Beispiel für das Nutzlastformat. Weitere Informationen zu diesen Parametern finden Sie unter CDMA-Objekt.

```
{
    "Timestamp": 1664313161,           // optional
    "CellTowers": {
        "Wcdma": [
        {
          "Mcc": int,                   // required
          "Mnc": int,                   // required
          "UtranCid": int,              // required
          "Lac": int,                   // optional
```

```
          "WcdmaLocalId": {             // optional
              "Uarfcndl": int,          // required
              "Psc": int,               // required
          },
          "Rscp": int,                  // optional
          "Pathloss": int,              // optional
          "WcdmaNmr": [                 // optional
                {
                  "Uarfcndl": int,      // required
                  "Psc": int,           // required
                  "UtranCid": int,      // required
                  "Rscp": int,          // optional
                  "Pathloss": int,      // optional
                }
             ]
          }
       ]
    }
}
```

TD-SCDMA (Zeitteilung, synchrone Codeteilung, Mehrfachzugriff)

Wenn Sie diese Messungsdaten verwenden, müssen Sie Informationen wie das Netzwerk und die Landesvorwahl, die Signalleistung und Identifikationsinformationen, Informationen zur Basisstation und optionale weitere Parameter angeben. Der nachfolgende Code zeigt ein Beispiel für das Nutzlastformat. Weitere Informationen zu diesen Parametern finden Sie unter CDMA-Objekt.

```
{
    "Timestamp": 1664313161,           // optional
    "CellTowers": {
        "Tdscdma": [
        {
          "Mcc": int,                  // required
          "Mnc": int,                  // required
          "UtranCid": int,             // required
          "Lac": int,                  // optional
          "TdscdmaLocalId": {          // optional
              "Uarfcn": int,           // required
              "CellParams": int,       // required
          },
          "TdscdmaTimingAdvance": int,  // optional
          "Rscp": int,                 // optional
          "Pathloss": int,             // optional
```

```
          "TdscdmaNmr": [               // optional
                {
                  "Uarfcn": int,        // required
                  "CellParams": int,    // required
                  "UtranCid": int,      // optional
                  "Rscp": int,          // optional
                  "Pathloss": int,      // optional
                }
             ]
          }
       ]
    }
}
```

IP-Reverse-Lookup-Solver

Sie können den IP-Reverse-Lookup-Solver verwenden, um den Standort mithilfe der IP-Adresse als Eingabe aufzulösen. Der Solver kann die Standortinformationen von Geräten abrufen, die mit ausgestattet wurden. AWS IoT Geben Sie die IP-Adressinformationen in einem Format an, das entweder dem IPv4- oder IPv6-Standardmuster oder dem hexadezimalen komprimierten IPv6-Muster entspricht. Anschließend erhalten Sie die aufgelöste Standortschätzung, einschließlich zusätzlicher Informationen wie Stadt und Land, in denen sich das Gerät befindet.

> **Note**
>
> Durch die Verwendung des IP-Reverse-Lookup-Solvers erklären Sie sich damit einverstanden, diesen nicht zum Zweck der Identifizierung oder Ortung eines bestimmten Haushalts oder einer Straßenadresse zu verwenden.

Beispiel für eine Nutzlast des IP-Reverse-Lookup-Solvers

Der folgende Code zeigt ein Beispiel für die JSON-Nutzlast des Geräts, das die Messungsdaten enthält. Wenn AWS IoT Core Device Location die in den Messdaten enthaltenen IP-Adressinformationen empfängt, sucht es diese Informationen in der Datenbank des Solver-Anbieters, die dann zur Auflösung der Standortinformationen verwendet wird. Um die Informationen abzurufen, geben Sie entweder die JSON-Payload in diesem Format an oder geben Sie Werte für den IP-Parameter des GetPositionEstimateAPI-Vorgangs an.

> **Note**
>
> Wenn dieser Solver verwendet wird, werden zusätzlich zu den Koordinaten auch die Stadt, das Bundesland, das Land und die Postleitzahl gemeldet, in denen sich das Gerät befindet. Ein Beispiel finden Sie unter Auflösen des Gerätestandorts (Konsole).

```
{
    "Timestamp": 1664313161,
    "Ip":{
        "IpAddress":"54.240.198.35"
      }
}
```

GNSS-Solver

Verwenden Sie den GNSS-Solver (Global Navigation Satellite System), um den Gerätestandort anhand der in den GNSS-Scanergebnisnachrichten oder NAV-Nachrichten enthaltenen Informationen abzurufen. Sie können optional zusätzliche GNSS-Unterstützungsinformationen angeben, wodurch die Anzahl der Variablen reduziert wird, die der Solver für die Suche nach Signalen verwenden muss. Durch die Bereitstellung dieser unterstützenden Informationen, zu denen Position, Höhe, Erfassungszeit und Genauigkeitsinformationen gehören, kann der Solver die Satelliten im Sichtfeld leicht identifizieren und den Gerätestandort berechnen.

Dieser Solver kann mit LoRa WAN-Geräten und Amazon Sidewalk-fähigen Geräten sowie mit anderen Geräten verwendet werden, die mit AWS IoT-Geräten ausgestattet wurden, mit denen ausgestattet wurde. AWS IoT Bei allgemeinen IoT-Geräten gilt Folgendes: Wenn die Geräte die Standortschätzung mithilfe von GNSS unterstützen, lösen die Transceiver die Standortinformationen auf, wenn die GNSS-Scaninformationen vom Gerät empfangen werden. Bei LoRa WAN-Geräten müssen die Geräte über den Edge-Chipsatz verfügen. LoRa Wenn eine Uplink-Nachricht vom Gerät empfangen wird, werden die GNSS-Scandaten AWS IoT für LoRa WAN oder AWS IoT Amazon Sidewalk gesendet, und der Standort wird auf der Grundlage der Scanergebnisse der Transceiver geschätzt.

> **Note**
>
> Die beiden Messarten schließen sich `Gnss` gegenseitig `GnssMultiFrame` aus. Eine einzelne Anfrage kann nicht beide enthalten.

Beispiel für die Nutzlast eines GNSS-Solvers

Der folgende Code zeigt ein Beispiel für die JSON-Nutzlast des Geräts, das die Messungsdaten enthält. Wenn AWS IoT Core Device Location die GNSS-Scaninformationen empfängt, die die Nutzdaten in den Messdaten enthalten, verwendet es die Transceiver und alle zusätzlichen enthaltenen Hilfsinformationen, um nach Signalen zu suchen und die Standortinformationen aufzulösen. Um die Informationen abzurufen, geben Sie entweder die JSON-Nutzdaten in diesem Format an oder geben Sie Werte für den GNSS-Parameter der API-Operation an. GetPositionEstimate

> **Note**
>
> Bevor AWS IoT Core Device Location den Gerätestandort auflösen kann, müssen Sie das Zielbyte aus der Payload entfernen.

```
{
    "Timestamp": 1664313161,                // optional
    "Gnss": {
        "AssistAltitude": number,           // optional
        "AssistPosition": [ number ],       // optional
        "CaptureTime": number,              // optional
        "CaptureTimeAccuracy": number,      // optional
        "Payload": "string",                // required
        "Use2DSolver": boolean              // optional
   }
}
```

Multi-frame GNSS-Solver

Verwenden Sie den GNSS-Solver (Global Navigation Satellite System) mit mehreren Frames, um den Gerätestandort anhand der Informationen abzurufen, die in mehreren GNSS-Scanergebnismeldungen oder NAV-Meldungen enthalten sind, die von demselben Gerät erfasst wurden. Sie geben die Aufzeichnungen in der Liste an. `Captures` Sie können optional zusätzliche GNSS-Unterstützungsinformationen angeben, wodurch die Anzahl der Variablen reduziert wird, die der Solver für die Suche nach Signalen verwenden muss. Durch die Bereitstellung dieser unterstützenden Informationen, zu denen Position, Höhe, Erfassungszeit und Genauigkeitsinformationen gehören, kann der Solver die Satelliten im Sichtfeld leicht identifizieren und den Gerätestandort berechnen.

> **Note**
>
> Die Messtypen `Gnss` und die `GnssMultiFrame` Messtypen schließen sich gegenseitig aus. Eine einzelne Anfrage kann nicht beide enthalten.

Multi-frame Beispiel für eine GNSS-Solver-Payload

Der folgende Code zeigt ein Beispiel für die JSON-Nutzlast des Geräts, das die Messungsdaten enthält. Wenn AWS IoT Core Device Location die GNSS-Scaninformationen mit mehreren Frames empfängt, die die in den Messdaten erfassten Daten enthalten, verwendet es die Transceiver und alle zusätzlichen enthaltenen Hilfsinformationen, um nach Signalen zu suchen und die Standortinformationen aufzulösen. Um die Informationen abzurufen, geben Sie entweder die JSON-Nutzdaten in diesem Format an oder geben Sie Werte für den GnssMultiFrameParameter des API-Vorgangs an. GetPositionEstimate

> **Note**
>
> 1. Bevor AWS IoT Core Device Location den Gerätestandort auflösen kann, müssen Sie das Zielbyte aus jeder Capture-Payload entfernen.
> 2. Die Anzahl der in der `Captures` Liste angegebenen Captures muss einem der folgenden Werte entsprechen: 2, 4, 8, 16 oder 32.
> 3. Wenn eine Anfrage auf `Use2DSolver` gesetzt ist`true`, müssen Sie auch angeben`AssistAltitude`. Andernfalls schlägt die Anfrage mit einem Validierungsfehler fehl.

```
{
    "Timestamp": 1664313161,                    // optional
    "GnssMultiFrame": {
        "AssistAltitude": number,               // optional
        "AssistPosition": [ number ],           // optional
        "CaptureTimeAccuracy": number,          // optional
        "Use2DSolver": boolean,                 // optional
        "Captures": [                           // required, must be 2, 4, 8, 16, or 32
 captures
            {
                "Payload": "string",            // required
```

```
                "CaptureTime": number          // optional
            },
            {
                "Payload": "string",           // required
                "CaptureTime": number          // optional
            },
                        ...
            {
                "Payload": "string",           // required
                "CaptureTime": number          // optional
            }
        ]
    }
}
```

BLE-Solver für Amazon Sidewalk-fähige Geräte

> **Note**
>
> Gibt für den Standort mit Bluetooth Low Energy die Standortkoordinaten AWS IoT zurück, die auf dem ungefähren Standort von Sidewalk Gateways in der Nähe basieren, die mit Amazon Sidewalk verbunden sind und für die die Community Finding-Funktion aktiviert ist. Gateway-Standortdaten sind AWS Inhalte und werden Ihnen ausschließlich zu dem Zweck zur Verfügung gestellt, Sie beim Auffinden Ihrer mit Amazon Sidewalk verbundenen Geräte zu unterstützen, und Sie dürfen die Daten nur für diesen Zweck verwenden. Sie dürfen Standortdaten nur über die Schnittstelle und die Funktionen, die wir Ihnen allgemein zur Verfügung stellen, verwenden und darauf zugreifen, und Sie dürfen nicht versuchen, Gateway-Standortdaten, die wir Ihnen zur Verfügung gestellt haben, neu zu identifizieren, zurückzuentwickeln oder neu zuzuordnen.

Amazon Sidewalk-Endgeräte, die eine Verbindung herstellen, AWS IoT Core für Amazon Sidewalk können ihre Standortdaten mithilfe von BLE- oder GNSS-Uplink-Nachrichten auflösen Wi-Fi, die vom Amazon Sidewalk-Endgerät veröffentlicht werden. Weitere Informationen finden Sie im Entwicklerhandbuch.AWS IoT Core für Amazon Sidewalk

Erweiterte Konfiguration für AWS IoT Core Standort des Geräts

Sie können den `AdvancedConfiguration` Parameter verwenden, um anzupassen, wie der AWS IoT Core Gerätestandort den Standort Ihrer Geräte ermittelt. Dieser Parameter wird unterstützt, wenn Sie die MQTT-Themen GetPositionEstimateAPI-Operation, AWS IoT Konsole oder Gerätestandort verwenden.

> **Note**
>
> Der `AdvancedConfiguration` Parameter ist optional und wird derzeit für den HTTP-API-Flow und den MQTT-Gerätestandortfluss unterstützt. Er wird für den LoRa WAN-Gerätestandortfluss nicht unterstützt.
> Bestehende Integrationen, die diesen Parameter nicht verwenden, sind davon nicht betroffen und funktionieren weiterhin wie zuvor. Wenn der Parameter nicht angegeben ist, wird das Standard-Konfidenzniveau von 68 Prozent verwendet. Dies war das Standardverhalten vor der Einführung dieses Parameters.

Themen

- WiFiCellular Konfiguration
- Wi-Fi Payload-Beispiel mit AdvancedConfiguration
- Beispiel für eine mobile Nutzlast mit AdvancedConfiguration

WiFiCellular Konfiguration

Die `WiFiCellular` Konfiguration gilt, wenn Ihre Messdaten Mobilfunkdaten oder beide Nutzlastinformationen enthalten Wi-Fi. Verwenden Sie diese Konfiguration, um das Konfidenzniveau für die Solver Wi-Fi und die auf Mobilfunk basierenden Solver individuell anzupassen.

`ConfidencePercent`

Der Konfidenzprozentsatz, der bei der Auflösung des Gerätestandorts anhand von Messdaten Wi-Fi oder Mobilfunkdaten verwendet wird. Dieser Wert bestimmt die Wahrscheinlichkeit, dass der tatsächliche Gerätestandort innerhalb des angegebenen Genauigkeitsradius liegt. Geben Sie einen Ganzzahlwert zwischen 50 und 99 an.

Ein höherer Konfidenzprozentsatz führt zu einem größeren Genauigkeitsradius, während ein niedrigerer Konfidenzprozentsatz zu einem kleineren Genauigkeitsradius führt. Das Standard-Konfidenzniveau beträgt 68 Prozent, wenn dieser Parameter nicht angegeben ist.

> **Note**
>
> Die `WiFiCellular` Konfiguration wird nur verwendet, wenn die Messdaten Mobilfunkinformationen enthalten Wi-Fi . Wenn die Messdaten nur andere Typen wie IP-Adresse oder GNSS enthalten, wird die `WiFiCellular` Konfiguration ignoriert.

Wi-Fi Payload-Beispiel mit AdvancedConfiguration

Das folgende Beispiel zeigt eine Nutzlast für eine Wi-Fi Messung mit dem `AdvancedConfiguration` Parameter, der das Konfidenzniveau auf 95 Prozent festlegt.

```
{
    "Timestamp": 1664313161,
    "WiFiAccessPoints": [
        {
            "MacAddress": "A0:EC:F9:1E:32:C1",
            "Rss": -75
        },
        {
            "MacAddress": "A0:EC:F9:15:72:5E",
            "Rss": -67
        }
    ],
    "AdvancedConfiguration": {
        "WiFiCellular": {
            "ConfidencePercent": 95
        }
    }
}
```

Die folgende GeoJSON-Nutzlast zeigt den aufgelösten Standort mit einem Konfidenzniveau von 0,95, was der angeforderten 95-prozentigen Konfidenz entspricht.

```
{
    "coordinates": [
```

```
        23.864079,
        61.44683
    ],
    "type": "Point",
    "properties": {
        "horizontalAccuracy": 22,
        "horizontalConfidenceLevel": 0.95,
        "timestamp": "2026-04-03T00:13:15.134343033Z"
    }
}
```

Beispiel für eine mobile Nutzlast mit AdvancedConfiguration

Das folgende Beispiel zeigt eine Nutzlast für Mobilfunkmessungen mit dem `AdvancedConfiguration` Parameter, der das Konfidenzniveau auf 55 Prozent festlegt.

```
{
    "Timestamp": 1664313161,
    "CellTowers": {
        "Gsm": [{
            "Mcc": 262,
            "Mnc": 1,
            "Lac": 5126,
            "GeranCid": 16504,
            "GsmLocalId": {
                "Bsic": 6,
                "Bcch": 82
            },
            "GsmTimingAdvance": 1,
            "RxLevel": -110,
            "GsmNmr": [{
                "Bsic": 7,
                "Bcch": 85,
                "RxLevel": -100,
                "GlobalIdentity": {
                    "Lac": 1,
                    "GeranCid": 1
                }
            }]
        }],
        "Wcdma": [{
            "Mcc": 262,
            "Mnc": 7,
```

```
            "Lac": 65535,
            "UtranCid": 14674663,
            "WcdmaNmr": [{
                    "Uarfcndl": 10786,
                    "UtranCid": 14674663,
                    "Psc": 149
                },
                {
                    "Uarfcndl": 10762,
                    "UtranCid": 14674663,
                    "Psc": 211
                }
            ]
        }],
        "Lte": [{
            "Mcc": 262,
            "Mnc": 2,
            "EutranCid": 2898945,
            "Rsrp": -50,
            "Rsrq": -5,
            "LteNmr": [{
                    "Earfcn": 6300,
                    "Pci": 237,
                    "Rsrp": -60,
                    "Rsrq": -6,
                    "EutranCid": 2898945
                },
                {
                    "Earfcn": 6300,
                    "Pci": 442,
                    "Rsrp": -70,
                    "Rsrq": -7,
                    "EutranCid": 2898945
                }
            ]
        }]
    },
    "AdvancedConfiguration": {
        "WiFiCellular": {
            "ConfidencePercent": 55
        }
    }
}
```

Die folgende GeoJSON-Nutzlast zeigt den aufgelösten Standort mit einem Konfidenzniveau von 0,55, was der angeforderten Zuverlässigkeit von 55 Prozent entspricht.


```
{
    "coordinates": [
        13.376077,
        52.51823,
        80.1
    ],
    "type": "Point",
    "properties": {
        "verticalAccuracy": 34,
        "verticalConfidenceLevel": 0.55,
        "horizontalAccuracy": 254,
        "horizontalConfidenceLevel": 0.55,
        "timestamp": "2026-04-06T23:43:45.198088456Z"
    }
}
```

Ereignismeldungen

Dieser Abschnitt enthält Informationen zu Nachrichten, die veröffentlicht werden AWS IoT , wenn Dinge oder Jobs aktualisiert oder geändert werden.

Generieren von Ereignisnachrichten

AWS IoT veröffentlicht Ereignismeldungen, wenn bestimmte Ereignisse eintreten. Beispielsweise werden Ereignisse von der Registry generiert, wenn Geräte hinzugefügt, aktualisiert oder gelöscht werden. Jedes Ereignis bewirkt, dass eine einzelne Ereignismeldung gesendet wird. Ereignismeldungen werden mit einer JSON-Nutzlast über MQTT veröffentlicht. Der Inhalt der Nutzlast hängt von der Art des Ereignisses ab.

> **Note**
>
> Ereignismeldungen werden garantiert einmal veröffentlicht. Es ist möglich, dass sie mehr als einmal veröffentlicht werden. Die Reihenfolge von Ereignismeldungen ist nicht garantiert.

Richtlinie für den Empfang von Ereignisnachrichten

Um Ereignismeldungen zu empfangen, muss Ihr Gerät eine entsprechende Richtlinie verwenden, die es ihm ermöglicht, eine Verbindung zum AWS IoT Geräte-Gateway herzustellen und MQTT-Ereignisthemen zu abonnieren. Sie müssen auch die entsprechenden Themenfilter abonnieren.

Es folgt ein Beispiel der für den Empfang von Ereignissen zum Lebenszyklus erforderlichen Richtlinie:

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe",
                "iot:Receive"
            ],
            "Resource": [
```

```
                "arn:aws:iot:us-east-1:123456789012:/$aws/events/*"
            ]
        }
    ]
}
```

Aktivieren Sie Ereignisse für AWS IoT

Bevor Abonnenten der reservierten Themen Nachrichten empfangen können, müssen Sie mithilfe der API AWS-Managementkonsole oder CLI Ereignisnachrichten von oder aktivieren. Informationen zu den Ereignisnachrichten, die mit den verschiedenen Optionen verwaltet werden, finden Sie in der Tabelle der AWS IoT Veranstaltungskonfigurationseinstellungen.

- Um Ereignismeldungen zu aktivieren, wechseln Sie in der AWS IoT Konsole zur Registerkarte Einstellungen und wählen Sie dann im Abschnitt Event-based Nachrichten die Option Ereignisse verwalten aus. Sie können die Ereignisse angeben, die Sie verwalten möchten.
- Um zu steuern, welche Ereignistypen mithilfe der API oder CLI veröffentlicht werden, rufen Sie die UpdateEventConfigurations API auf oder verwenden Sie den update-event-configurations CLI-Befehl. Beispiel:

```
aws iot update-event-configurations --event-configurations "{\"THING\":{\"Enabled\":
 true}}"
```

> **Note**
>
> Alle Anführungszeichen („“) werden durch Backslashes (\) umgangen.

Sie können die aktuelle Ereigniskonfiguration abrufen, indem Sie die DescribeEventConfigurations API aufrufen oder den describe-event-configurations CLI-Befehl verwenden. Beispiel:

```
aws iot describe-event-configurations
```

Tabelle der Einstellungen für die AWS IoT -Ereigniskonfiguration

Ereigniskategorie (AWS IoT Konsole: Einstellungen: Event-based Nachrichten)	**`eventConfigurations`** - Schlüsselwert (AWS CLI/API)	Topic der Ereignismeldung
(Kann nur mit der AWS CLI/API konfiguriert werden)	`CA_CERTIFICATE`	`$aws/events/certificates/registered/` *`caCertificateId`*
(Kann nur mit der AWS CLI/API konfiguriert werden)	`CERTIFICATE`	`$aws/events/presence/connected/` *`clientId`*
(Kann nur mit der AWS CLI/API konfiguriert werden)	`CERTIFICATE`	`$aws/events/presence/disconnected/` *`clientId`*
(Kann nur mit der AWS CLI/API konfiguriert werden)	`CERTIFICATE`	`$aws/events/subscriptions/subscribed/` *`clientId`*
(Kann nur mit der AWS CLI/API konfiguriert werden)	`CERTIFICATE`	`$aws/events/subscriptions/unsubscribed/` *`clientId`*
Auftrag abgeschlossen, storniert	`JOB`	`$aws/events/job/`*`jobID`*`/canceled`
Auftrag abgeschlossen, storniert	`JOB`	`$aws/events/job/`*`jobID`*`/cancellation_in_progress`
Auftrag abgeschlossen, storniert	`JOB`	`$aws/events/job/`*`jobID`*`/completed`
Auftrag abgeschlossen, storniert	`JOB`	`$aws/events/job/`*`jobID`*`/deleted`

Ereigniskategorie (AWS IoT Konsole: Einstellungen: Event-based Nachrichten)	**`eventConfigurations`** - Schlüsselwert (AWS CLI/API)	Topic der Ereignismeldung
Auftrag abgeschlossen, storniert	`JOB`	`$aws/events/job/`*`jobID`*`/deletion_in_progress`
Auftragsausführung: erfolgreich, abgelehnt, storniert, entfernt	`JOB_EXECUTION`	`$aws/events/jobExecution/ `*`jobID`*`/canceled`
Auftragsausführung: erfolgreich, abgelehnt, storniert, entfernt	`JOB_EXECUTION`	`$aws/events/jobExecution/ `*`jobID`*`/deleted`
Auftragsausführung: erfolgreich, abgelehnt, storniert, entfernt	`JOB_EXECUTION`	`$aws/events/jobExecution/ `*`jobID`*`/failed`
Auftragsausführung: erfolgreich, abgelehnt, storniert, entfernt	`JOB_EXECUTION`	`$aws/events/jobExecution/ `*`jobID`*`/rejected`
Auftragsausführung: erfolgreich, abgelehnt, storniert, entfernt	`JOB_EXECUTION`	`$aws/events/jobExecution/ `*`jobID`*`/removed`
Auftragsausführung: erfolgreich, abgelehnt, storniert, entfernt	`JOB_EXECUTION`	`$aws/events/jobExecution/ `*`jobID`*`/succeeded`
Auftragsausführung: erfolgreich, abgelehnt, storniert, entfernt	`JOB_EXECUTION`	`$aws/events/jobExecution/ `*`jobID`*`/timed_out`

Ereigniskategorie (AWS IoT Konsole: Einstellungen: Event-based Nachrichten)	**`eventConfigurations`** - Schlüsselwert (AWS CLI/API)	Topic der Ereignismeldung
Objekt: erstellt, aktualisiert, gelöscht	`THING`	`$aws/events/thing/` *`thingName`* `/created`
Objekt: erstellt, aktualisiert, gelöscht	`THING`	`$aws/events/thing/` *`thingName`* `/updated`
Objekt: erstellt, aktualisiert, gelöscht	`THING`	`$aws/events/thing/` *`thingName`* `/deleted`
Objektgruppe: hinzugefügt, entfernt	`THING_GROUP`	`$aws/events/thingGroup/` *`thingGroupName`* `/created`
Objektgruppe: hinzugefügt, entfernt	`THING_GROUP`	`$aws/events/thingGroup/` *`thingGroupName`* `/updated`
Objektgruppe: hinzugefügt, entfernt	`THING_GROUP`	`$aws/events/thingGroup/` *`thingGroupName`* `/deleted`
Objektgruppenhierarchie: hinzugefügt, entfernt	`THING_GROUP_HIERARCHY`	`$aws/events/thingGroupHierarchy/thingGroup/` *`parentThingGroupName`* `/childThingGroup/` *`childThingGroupName`* `/added`

Ereigniskategorie (AWS IoT Konsole: Einstellungen: Event-based Nachrichten)	**`eventConfigurations`** - Schlüsselwert (AWS CLI/API)	Topic der Ereignismeldung
Objektgruppenhierarchie: hinzugefügt, entfernt	`THING_GROUP_HIERARCHY`	`$aws/events/thingGroupHierarchy/thingGroup/` *`parentThingGroupName`* `/childThingGroup/` *`childThingGroupName`* `/removed`
Mitgliedschaft in einer Objektgruppe: hinzugefügt, entfernt	`THING_GROUP_MEMBERSHIP`	`$aws/events/thingGroupMembership/thingGroup/` *`thingGroupName`* `/thing/`*`thingName`* `/added`
Mitgliedschaft in einer Objektgruppe: hinzugefügt, entfernt	`THING_GROUP_MEMBERSHIP`	`$aws/events/thingGroupMembership/thingGroup/` *`thingGroupName`* `/thing/`*`thingName`* `/removed`
Objekttyp: erstellt, aktualisiert, gelöscht	`THING_TYPE`	`$aws/events/thingType/` *`thingTypeName`* `/created`
Objekttyp: erstellt, aktualisiert, gelöscht	`THING_TYPE`	`$aws/events/thingType/` *`thingTypeName`* `/updated`
Objekttyp: erstellt, aktualisiert, gelöscht	`THING_TYPE`	`$aws/events/thingType/` *`thingTypeName`* `/deleted`

Ereigniskategorie (AWS IoT Konsole: Einstellungen: Event-based Nachrichten)	**`eventConfigurations`** - Schlüsselwert (AWS CLI/API)	Topic der Ereignismeldung

Zuordnung des Objekttyps: hinzugefügt, entfernt	`THING_TYPE_ASSOCIATION`	`$aws/events/thingTypeAssociation/thing/` *`thingName`* `/thingType/` *`thingTypeName`* `/added`
		
		`$aws/events/thingTypeAssociation/thing/` *`thingName`* `/thingType/` *`thingTypeName`* `/removed`

Registry-Ereignisse

Die Registry kann Ereignismeldungen veröffentlichen, wenn Objekte, Objekttypen und Objektgruppen erstellt, aktualisiert oder gelöscht werden. Diese Ereignisse sind jedoch standardmäßig nicht verfügbar. Weitere Informationen darüber, wie Sie diese Ereignisse aktivieren, finden Sie unter Aktivieren Sie Ereignisse für AWS IoT.

Die Registry kann die folgenden Ereignistypen bereitstellen:

- Objektereignisse
- Objekttypereignisse
- Objektgruppenereignisse

Objektereignisse

Sache Created/Updated/Deleted

Die Registry veröffentlicht die folgenden Ereignismeldungen, wenn Objekte erstellt, aktualisiert oder gelöscht werden:

- $aws/events/thing/*thingName*/created
- $aws/events/thing/*thingName*/updated
- $aws/events/thing/*thingName*/deleted

Die Meldungen enthalten die folgende Beispielnutzlast:

```
{
    "eventType" : "THING_EVENT",
    "eventId" : "f5ae9b94-8b8e-4d8e-8c8f-b3266dd89853",
    "timestamp" : 1234567890123,
    "operation" : "CREATED|UPDATED|DELETED",
    "accountId" : "123456789012",
    "thingId" : "b604f69c-aa9a-4d4a-829e-c480e958a0b5",
    "thingName" : "MyThing",
    "versionNumber" : 1,
    "thingTypeName" : null,
    "attributes": {
                "attribute3": "value3",
                "attribute1": "value1",
                "attribute2": "value2"
    }
}
```

Die Nutzlasten enthalten die folgenden Attribute:

eventType

Auf „THING_EVENT“ festgelegt.

eventId

Eine eindeutige Ereignis-ID (Zeichenfolge).

timestamp

Der UNIX-Zeitstempel für den Zeitpunkt, an dem das Ereignis aufgetreten ist.

operation

Die Operation, die das Ereignis ausgelöst hat. Folgende sind gültige Werte:

- CREATED
- UPDATED

- GELÖSCHT

accountId

Dein AWS-Konto Ausweis.

thingId

Die ID des gerade erstellten, aktualisierten oder gelöschten Objekts.

thingName

Der Name des gerade erstellten, aktualisierten oder gelöschten Objekts.

versionNumber

Die Version des gerade erstellten, aktualisierten oder gelöschten Objekts. Dieser Wert wird bei der Erstellung eines Objekts auf 1 festgelegt. Er wird bei jeder Aktualisierung des Objekts um 1 erhöht.

Sache TypeName

Der mit dem Objekt verknüpfte Objekttyp, sofern vorhanden. Andernfalls `null`.

Attribute

Eine Reihe von mit dem Objekt verknüpften Name-Wert-Paaren.

Objekttypereignisse

Ereignisse im Zusammenhang mit dem Objekttyp:

- Art der Sache Created/Updated/Deprecated/Undeprecated/Deleted
- Objekttyp einem Objekt zugeordnet/nicht zugeordnet

Art der Sache Created/Updated/Deprecated/Undeprecated/Deleted

Die Registrierung veröffentlicht die folgenden Ereignismeldungen, wenn Dingtypen erstellt, aktualisiert, veraltet, nicht mehr unterstützt oder gelöscht werden:

- `$aws/events/thingType/thingTypeName/created`
- `$aws/events/thingType/thingTypeName/updated`
- `$aws/events/thingType/thingTypeName/deleted`

Die Nachricht enthält die folgende Beispielnutzlast:

```
{
    "eventType" : "THING_TYPE_EVENT",
    "eventId" : "8827376c-4b05-49a3-9b3b-733729df7ed5",
    "timestamp" : 1234567890123,
    "operation" : "CREATED|UPDATED|DELETED",
    "accountId" : "123456789012",
    "thingTypeId" : "c530ae83-32aa-4592-94d3-da29879d1aac",
    "thingTypeName" : "MyThingType",
    "isDeprecated" : false|true,
    "deprecationDate" : null,
    "searchableAttributes" : [ "attribute1", "attribute2", "attribute3" ],
    "propagatingAttributes": [
        {
     "userPropertyKey": "key",
     "thingAttribute": "model"
        },
        {
     "userPropertyKey": "key",
     "connectionAttribute": "iot:ClientId"
        }
    ],
    "description" : "My thing type"
}
```

Die Nutzlasten enthalten die folgenden Attribute:

eventType

Auf „THING_TYPE_EVENT“ festgelegt.

eventId

Eine eindeutige Ereignis-ID (Zeichenfolge).

timestamp

Der UNIX-Zeitstempel für den Zeitpunkt, an dem das Ereignis aufgetreten ist.

operation

Die Operation, die das Ereignis ausgelöst hat. Folgende sind gültige Werte:

- CREATED

- UPDATED
- GELÖSCHT

accountId

Ihre ID. AWS-Konto

Sache TypeId

Die ID des Dingtyps, der erstellt, aktualisiert, veraltet oder gelöscht wird.

Sache TypeName

Der Name des Dingtyps, der erstellt, aktualisiert, veraltet oder gelöscht wird.

isDeprecated

`true`, wenn der Objekttyp veraltet ist. Andernfalls `false`.

deprecationDate

Der UNIX-Zeitstempel für den Zeitpunkt, an dem der Objekttyp als veraltet eingestuft wurde.

searchableAttributes

Eine Reihe von mit dem Objekttyp verknüpften Name-Wert-Paaren, die für Suchen verwendet werden können.

Weitergabe von Attributen

Eine Liste von Attributen, die sich verbreiten. Ein Propagierungsattribut kann ein Dingattribut, ein Verbindungsattribut und einen Benutzereigenschaftsschlüssel enthalten. Weitere Informationen finden Sie unter Hinzufügen von Propagierungsattributen zur Nachrichtenanreicherung.

description

Eine Beschreibung des Objekttyps.

Objekttyp einem Objekt zugeordnet/nicht zugeordnet

Die Registry veröffentlicht die folgenden Ereignismeldungen, wenn ein Objekttyp einem Objekt zugeordnet wird oder die Zuordnung entfernt wird.

- `$aws/events/thingTypeAssociation/thing/`*`thingName`*`/thingType/`*`typeName`*`/added`

- `$aws/events/thingTypeAssociation/thing/thingName/thingType/typeName/removed`

Im Folgenden sehen Sie ein Beispiel für eine `added`-Nutzlast. Die Nutzlasten für `removed`-Nachrichten sind ähnlich.

```
{
    "eventId" : "87f8e095-531c-47b3-aab5-5171364d138d",
    "eventType" : "THING_TYPE_ASSOCIATION_EVENT",
    "operation" : "ADDED",
    "thingId" : "b604f69c-aa9a-4d4a-829e-c480e958a0b5",
    "thingName": "myThing",
    "thingTypeName" : "MyThingType",
    "timestamp" : 1234567890123,
}
```

Die Nutzlasten enthalten die folgenden Attribute:

eventId

Eine eindeutige Ereignis-ID (Zeichenfolge).

eventType

Auf "THING_TYPE_ASSOCIATION_EVENT" gesetzt.

operation

Die Operation, die das Ereignis ausgelöst hat. Folgende sind gültige Werte:

- HINZUGEFÜGT
- ENTFERNT

thingId

Die ID des Objekts, dessen Zuordnung geändert wurde.

thingName

Der Name des Objekts, dessen Zuordnung geändert wurde.

Sache TypeName

Der Objekttyp, der dem Objekt zugeordnet oder nicht mehr zugeordnet ist.

timestamp

Der UNIX-Zeitstempel für den Zeitpunkt, an dem das Ereignis aufgetreten ist.

Objektgruppenereignisse

Ereignisse im Zusammenhang mit Objektgruppen:

- Sache (Gruppe) Created/Updated/Deleted
- Objekt, das einer Objektgruppe hinzugefügt/aus dieser entfernt wird
- Objektgruppe, die einer Objektgruppe hinzugefügt/aus dieser gelöscht wird

Sache (Gruppe) Created/Updated/Deleted

Die Registry veröffentlicht die folgenden Ereignismeldungen, wenn eine Objektgruppe erstellt, aktualisiert oder gelöscht wird.

- `$aws/events/thingGroup/`*`groupName`*`/created`
- `$aws/events/thingGroup/`*`groupName`*`/updated`
- `$aws/events/thingGroup/`*`groupName`*`/deleted`

Im Folgenden sehen Sie ein Beispiel für eine `updated`-Nutzlast. Die Nutzlasten für `created`- und `deleted`-Nachrichten sind ähnlich.

```
{
  "eventType": "THING_GROUP_EVENT",
  "eventId": "8b9ea8626aeaa1e42100f3f32b975899",
  "timestamp": 1603995417409,
  "operation": "UPDATED",
  "accountId": "571EXAMPLE833",
  "thingGroupId": "8757eec8-bb37-4cca-a6fa-403b003d139f",
  "thingGroupName": "Tg_level5",
  "versionNumber": 3,
  "parentGroupName": "Tg_level4",
  "parentGroupId": "5fce366a-7875-4c0e-870b-79d8d1dce119",
  "description": "New description for Tg_level5",
  "rootToParentThingGroups": [
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/TgTopLevel",
```

```
      "groupId": "36aa0482-f80d-4e13-9bff-1c0a75c055f6"
    },
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/Tg_level1",
      "groupId": "bc1643e1-5a85-4eac-b45a-92509cbe2a77"
    },
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/Tg_level2",
      "groupId": "0476f3d2-9beb-48bb-ae2c-ea8bd6458158"
    },
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/Tg_level3",
      "groupId": "1d9d4ffe-a6b0-48d6-9de6-2e54d1eae78f"
    },
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/Tg_level4",
      "groupId": "5fce366a-7875-4c0e-870b-79d8d1dce119"
    }
  ],
  "attributes": {
    "attribute1": "value1",
    "attribute3": "value3",
    "attribute2": "value2"
  },
  "dynamicGroupMappingId": null
}
```

Die Nutzlasten enthalten die folgenden Attribute:

eventType

Auf „THING_GROUP_EVENT“ festgelegt.

eventId

Eine eindeutige Ereignis-ID (Zeichenfolge).

timestamp

Der UNIX-Zeitstempel für den Zeitpunkt, an dem das Ereignis aufgetreten ist.

operation

Die Operation, die das Ereignis ausgelöst hat. Folgende sind gültige Werte:

- CREATED

- UPDATED
- GELÖSCHT

accountId

Dein AWS-Konto Ausweis.

Sache GroupId

Die ID der gerade erstellten, aktualisierten oder gelöschten Objektgruppe.

Sache GroupName

Der Name der gerade erstellten, aktualisierten oder gelöschten Objektgruppe.

versionNumber

Die Version der Objektgruppe. Dieser Wert wird bei der Erstellung einer Objektgruppe auf 1 festgelegt. Er wird bei jeder Aktualisierung der Objektgruppe um 1 erhöht.

Elternteil GroupName

Der Name der übergeordneten Objektgruppe (sofern vorhanden).

Elternteil GroupId

Die ID der übergeordneten Objektgruppe (sofern vorhanden).

description

Eine Beschreibung der Objektgruppe.

Wurzel ToParentThingGroups

Ein Array mit Informationen zur übergeordneten Objektgruppe. Es gibt ein Element für jede übergeordnete Objektgruppe. Der erste Eintrag bezieht sich auf die Stamm-Objektgruppe und dies wird fortgesetzt, bis die übergeordnete Objektgruppe erreicht wurde. Jeder Eintrag enthält die Werte `groupArn` und `groupId` der Objektgruppe.

Attribute

Eine Reihe von mit der Objektgruppe verknüpften Name-Wert-Paaren.

Objekt, das einer Objektgruppe hinzugefügt/aus dieser entfernt wird

Die Registry veröffentlicht die folgenden Ereignismeldungen, wenn ein Objekt einer Objektgruppe hinzugefügt oder aus dieser entfernt wird.

- `$aws/events/thingGroupMembership/thingGroup/thingGroupName/thing/thingName/added`
- `$aws/events/thingGroupMembership/thingGroup/thingGroupName/thing/thingName/removed`

Die Meldungen enthalten die folgende Beispielnutzlast:

```
{
    "eventType" : "THING_GROUP_MEMBERSHIP_EVENT",
    "eventId" : "d684bd5f-6f6e-48e1-950c-766ac7f02fd1",
    "timestamp" : 1234567890123,
    "operation" : "ADDED|REMOVED",
    "accountId" : "123456789012",
    "groupArn" : "arn:aws:iot:ap-northeast-2:123456789012:thinggroup/
MyChildThingGroup",
    "groupId" : "06838589-373f-4312-b1f2-53f2192291c4",
    "thingArn" : "arn:aws:iot:ap-northeast-2:123456789012:thing/MyThing",
    "thingId" : "b604f69c-aa9a-4d4a-829e-c480e958a0b5",
    "membershipId" : "8505ebf8-4d32-4286-80e9-c23a4a16bbd8"
}
```

Die Nutzlasten enthalten die folgenden Attribute:

eventType

Auf „THING_GROUP_MEMBERSHIP_EVENT“ festgelegt.

eventId

Die Ereignis-ID.

Zeitstempel

Der UNIX-Zeitstempel für den Zeitpunkt, an dem das Ereignis aufgetreten ist.

operation

`ADDED`, wenn ein Objekt zu einer Objektgruppe hinzugefügt wird, `REMOVED`, wenn ein Objekt aus einer Objektgruppe entfernt wird.

accountId

Dein AWS-Konto Ausweis.

groupArn

Der ARN der Objektgruppe.

groupId

Die ID der Gruppe.

thingArn

Der ARN des Objekts, das der Objektgruppe hinzugefügt bzw. daraus entfernt wurde.

thingId

Die ID des Objekts, das der Objektgruppe hinzugefügt bzw. daraus entfernt wurde.

membershipId

Eine ID, die die Beziehung zwischen dem Objekt und der Objektgruppe angibt. Dieser Wert wird generiert, wenn Sie ein Objekt zu einer Objektgruppe hinzufügen.

Objektgruppe, die einer Objektgruppe hinzugefügt/aus dieser gelöscht wird

Die Registry veröffentlicht die folgenden Ereignismeldungen, wenn eine Objektgruppe einer anderen Objektgruppe hinzugefügt oder aus dieser entfernt wird.

- `$aws/events/thingGroupHierarchy/thingGroup/`*`parentThingGroupName`*`/childThingGroup/`*`childThingGroupName`*`/added`
- `$aws/events/thingGroupHierarchy/thingGroup/`*`parentThingGroupName`*`/childThingGroup/`*`childThingGroupName`*`/removed`

Die Nachricht enthält die folgende Beispielnutzlast:

```
{
    "eventType" : "THING_GROUP_HIERARCHY_EVENT",
    "eventId" : "264192c7-b573-46ef-ab7b-489fcd47da41",
    "timestamp" : 1234567890123,
    "operation" : "ADDED|REMOVED",
    "accountId" : "123456789012",
    "thingGroupId" : "8f82a106-6b1d-4331-8984-a84db5f6f8cb",
    "thingGroupName" : "MyRootThingGroup",
    "childGroupId" : "06838589-373f-4312-b1f2-53f2192291c4",
    "childGroupName" : "MyChildThingGroup"
```

```
}
```

Die Nutzlasten enthalten die folgenden Attribute:

eventType

Auf „THING_GROUP_HIERARCHY_EVENT“ festgelegt.

eventId

Die Ereignis-ID.

Zeitstempel

Der UNIX-Zeitstempel für den Zeitpunkt, an dem das Ereignis aufgetreten ist.

operation

`ADDED`, wenn ein Objekt zu einer Objektgruppe hinzugefügt wird, `REMOVED`, wenn ein Objekt aus einer Objektgruppe entfernt wird.

accountId

Dein AWS-Konto Ausweis.

Sache GroupId

Die ID der übergeordneten Objektgruppe.

Sache GroupName

Der Name der übergeordneten Objektgruppe.

Kind GroupId

Die ID der untergeordneten Objektgruppe.

Kind GroupName

Der Name der untergeordneten Objektgruppe.

Auftragsereignisse

Der AWS IoT Jobs-Dienst veröffentlicht in reservierten Themen im MQTT-Protokoll, wenn Jobs ausstehen, abgeschlossen oder abgebrochen sind und wenn ein Gerät bei der Ausführung eines Jobs Erfolg oder Misserfolg meldet. Geräte oder Management- und Überwachungsanwendungen können den Status der Aufträge nachverfolgen, indem sie diese Topics abonnieren.

Aktivieren von Ereignisbenachrichtigungen

Antwortnachrichten vom AWS IoT Jobs-Dienst passieren nicht den Message Broker und können nicht von anderen Clients oder Regeln abonniert werden. Verwenden Sie die Topics `notify` and `notify-next`, um Nachrichten zu Auftragsaktivitäten zu abonnieren. Weitere Informationen über auftragsbezogene Topics finden Sie unter Auftragsthemen.

Um über Jobaktualisierungen informiert zu werden, aktivieren Sie diese Auftragsereignisse mithilfe der AWS-Managementkonsole oder mithilfe der API oder CLI. Weitere Informationen finden Sie unter Aktivieren Sie Ereignisse für AWS IoT.

Funktionsweise von Auftragsereignissen

Da das Abbrechen oder Löschen eines Auftrags einige Zeit in Anspruch nehmen kann, werden zwei Nachrichten gesendet, um den Start und das Ende einer Anfrage anzuzeigen. Wenn zum Beispiel eine Abbruchanfrage gestartet wird, wird eine Nachricht an das `$aws/events/job/jobID/cancellation_in_progress`-Thema gesendet. Wenn die Abbruchanfrage abgeschlossen ist, wird eine Nachricht an das `$aws/events/job/jobID/canceled`-Thema gesendet.

Ein ähnlicher Prozess findet bei einer Anfrage zur Löschung eines Auftrags statt. Management- und Überwachungsanwendungen können diese Themen abonnieren, um den Status von Aufträgen nachzuverfolgen. Für weitere Informationen zur Veröffentlichung und zum Abonnement von MQTT-Themen vgl. the section called “Gerätekommunikationsprotokolle”.

Auftragsereignistypen

Die folgenden Beispiele zeigen die verschiedenen Arten von Auftragsereignissen:

Job Completed/Canceled/Deleted

Der AWS IoT Jobs-Service veröffentlicht eine Nachricht zu einem MQTT-Thema, wenn ein Job abgeschlossen, storniert oder gelöscht wird oder wenn ein Stornierungs- oder Löschvorgang im Gange ist:

- `$aws/events/job/`*`jobID`*`/completed`
- `$aws/events/job/`*`jobID`*`/canceled`
- `$aws/events/job/`*`jobID`*`/deleted`
- `$aws/events/job/`*`jobID`*`/cancellation_in_progress`
- `$aws/events/job/`*`jobID`*`/deletion_in_progress`

Die `completed`-Nachricht enthält die folgende Beispielnutzlast:

```
{
  "eventType": "JOB",
  "eventId": "7364ffd1-8b65-4824-85d5-6c14686c97c6",
  "timestamp": 1234567890,
  "operation": "completed",
  "jobId": "27450507-bf6f-4012-92af-bb8a1c8c4484",
  "status": "COMPLETED",
  "targetSelection": "SNAPSHOT|CONTINUOUS",
  "targets": [
    "arn:aws:iot:us-east-1:123456789012:thing/a39f6f91-70cf-4bd2-a381-9c66df1a80d0",
    "arn:aws:iot:us-east-1:123456789012:thinggroup/2fc4c0a4-6e45-4525-
a238-0fe8d3dd21bb"
  ],
  "description": "My Job Description",
  "completedAt": 1234567890123,
  "createdAt": 1234567890123,
  "lastUpdatedAt": 1234567890123,
  "jobProcessDetails": {
    "numberOfCanceledThings": 0,
    "numberOfRejectedThings": 0,
    "numberOfFailedThings": 0,
    "numberOfRemovedThings": 0,
    "numberOfSucceededThings": 3
  }
}
```

Die `canceled`-Nachricht enthält die folgende Beispielnutzlast.

```
{
  "eventType": "JOB",
  "eventId": "568d2ade-2e9c-46e6-a115-18afa1286b06",
  "timestamp": 1234567890,
  "operation": "canceled",
  "jobId": "4d2a531a-da2e-47bb-8b9e-ff5adcd53ef0",
  "status": "CANCELED",
  "targetSelection": "SNAPSHOT|CONTINUOUS",
  "targets": [
    "arn:aws:iot:us-east-1:123456789012:thing/Thing0-947b9c0c-ff10-4a80-b4b3-
cd33d0145a0f",
    "arn:aws:iot:us-east-1:123456789012:thinggroup/
ThingGroup1-95c644d5-1621-41a6-9aa5-ad2de581d18f"
  ],
  "description": "My job description",
```

```
  "createdAt": 1234567890123,
  "lastUpdatedAt": 1234567890123
}
```

Die `deleted`-Nachricht enthält die folgende Beispielnutzlast.

```
{
      "eventType": "JOB",
      "eventId": "568d2ade-2e9c-46e6-a115-18afa1286b06",
      "timestamp": 1234567890,
      "operation": "deleted",
      "jobId": "4d2a531a-da2e-47bb-8b9e-ff5adcd53ef0",
      "status": "DELETED",
      "targetSelection": "SNAPSHOT|CONTINUOUS",
      "targets": [
        "arn:aws:iot:us-east-1:123456789012:thing/Thing0-947b9c0c-ff10-4a80-b4b3-
cd33d0145a0f",
        "arn:aws:iot:us-east-1:123456789012:thinggroup/
ThingGroup1-95c644d5-1621-41a6-9aa5-ad2de581d18f"
      ],
      "description": "My job description",
      "createdAt": 1234567890123,
      "lastUpdatedAt": 1234567890123,
      "comment": "Comment for this operation"
    }
```

Die `cancellation_in_progress`-Nachricht enthält die folgende Beispielnutzlast:

```
{
      "eventType": "JOB",
      "eventId": "568d2ade-2e9c-46e6-a115-18afa1286b06",
      "timestamp": 1234567890,
      "operation": "cancellation_in_progress",
      "jobId": "4d2a531a-da2e-47bb-8b9e-ff5adcd53ef0",
      "status": "CANCELLATION_IN_PROGRESS",
      "targetSelection": "SNAPSHOT|CONTINUOUS",
      "targets": [
        "arn:aws:iot:us-east-1:123456789012:thing/Thing0-947b9c0c-ff10-4a80-b4b3-
cd33d0145a0f",
        "arn:aws:iot:us-east-1:123456789012:thinggroup/
ThingGroup1-95c644d5-1621-41a6-9aa5-ad2de581d18f"
      ],
      "description": "My job description",
```

```
      "createdAt": 1234567890123,
      "lastUpdatedAt": 1234567890123,
      "comment": "Comment for this operation"
    }
```

Die `deletion_in_progress`-Nachricht enthält die folgende Beispielnutzlast:

```
{
      "eventType": "JOB",
      "eventId": "568d2ade-2e9c-46e6-a115-18afa1286b06",
      "timestamp": 1234567890,
      "operation": "deletion_in_progress",
      "jobId": "4d2a531a-da2e-47bb-8b9e-ff5adcd53ef0",
      "status": "DELETION_IN_PROGRESS",
      "targetSelection": "SNAPSHOT|CONTINUOUS",
      "targets": [
        "arn:aws:iot:us-east-1:123456789012:thing/Thing0-947b9c0c-ff10-4a80-b4b3-
cd33d0145a0f",
        "arn:aws:iot:us-east-1:123456789012:thinggroup/
ThingGroup1-95c644d5-1621-41a6-9aa5-ad2de581d18f"
      ],
      "description": "My job description",
      "createdAt": 1234567890123,
      "lastUpdatedAt": 1234567890123,
      "comment": "Comment for this operation"
    }
```

Endstatus der Auftragsausführung

Der AWS IoT Jobs-Dienst veröffentlicht eine Nachricht, wenn ein Gerät eine Auftragsausführung auf den Terminalstatus aktualisiert:

- `$aws/events/jobExecution/`*`jobID`*`/succeeded`
- `$aws/events/jobExecution/`*`jobID`*`/failed`
- `$aws/events/jobExecution/`*`jobID`*`/rejected`
- `$aws/events/jobExecution/`*`jobID`*`/canceled`
- `$aws/events/jobExecution/`*`jobID`*`/timed_out`
- `$aws/events/jobExecution/`*`jobID`*`/removed`
- `$aws/events/jobExecution/`*`jobID`*`/deleted`

Die Nachricht enthält die folgende Beispielnutzlast:

```
{
  "eventType": "JOB_EXECUTION",
  "eventId": "cca89fa5-8a7f-4ced-8c20-5e653afb3572",
  "timestamp": 1234567890,
  "operation": "succeeded|failed|rejected|canceled|removed|timed_out",
  "jobId": "154b39e5-60b0-48a4-9b73-f6f8dd032d27",
  "thingArn": "arn:aws:iot:us-east-1:123456789012:myThing/6d639fbc-8f85-4a90-924d-
a2867f8366a7",
  "status": "SUCCEEDED|FAILED|REJECTED|CANCELED|REMOVED|TIMED_OUT",
  "statusDetails": {
    "key": "value"
  }
}
```

Ereignisse bei der Befehlsausführung

AWS IoT veröffentlicht Ereignismeldungen in MQTT-Themen, wenn sich der Status von Befehlsausführungen ändert. Sie können diese Ereignisse verwenden, um den Fortschritt der Befehlsausführung zu überwachen und Anwendungen zu erstellen, die auf Statusänderungen reagieren.

Themen zu Ereignissen bei der Befehlsausführung

Ereignisse zur Befehlsausführung werden unter dem folgenden MQTT-Thema veröffentlicht:

```
$aws/events/commandExecution/commandId/status
```

Wobei Folgendes gilt:

- *commandId*ist der Bezeichner des Befehls.
- *status*ist der Status der Befehlsausführung. Folgende Werte sind zulässig: `CREATED`, `IN_PROGRESS`, `SUCCEEDED`, `FAILED`, `REJECTED`, `TIMED_OUT`.

Abonnieren Sie das folgende Thema und verwenden Sie Platzhalterzeichen, um Benachrichtigungen für alle Befehle und Status zu erhalten:

```
$aws/events/commandExecution/+/#
```

Meldung über ein Ereignis bei der Befehlsausführung

Wenn sich der Ausführungsstatus eines Befehls ändert, AWS IoT veröffentlicht eine Ereignisnachricht im entsprechenden MQTT-Thema. Die Nachricht enthält die folgende Beispielnutzlast:

```
{
    "executionId": "2bd65c51-4cfd-49e4-9310-d5cbfdbc8554",
    "status": "FAILED",
    "statusReason": {
        "reasonCode": "DEVICE_TOO_BUSY",
        "reasonDescription": ""
    },
    "eventType": "COMMAND_EXECUTION",
    "commandArn": "arn:aws:iot:us-east-1:123456789012:command/0b9d9ddf-
e873-43a9-8e2c-9fe004a90086",
    "targetArn": "arn:aws:iot:us-east-1:123456789012:thing/5006c3fc-
de96-4def-8427-7eee36c6f2bd",
    "timestamp": 1717708862107
}
```

Die Nutzlast enthält die folgenden Attribute:

Ausführungs-ID

Ein eindeutiger Bezeichner für die Befehlsausführung (Zeichenfolge).

status

Der Status der Befehlsausführung. Folgende Werte sind zulässig: `CREATED`, `IN_PROGRESS`, `SUCCEEDED`, `FAILED`, `REJECTED`, `TIMED_OUT`.

Status/Grund

Ein Objekt, das zusätzliche Informationen über den Status enthält, falls verfügbar. Enthält `reasonCode` und `reasonDescription` Felder.

eventType

Auf „COMMAND_EXECUTION“ setzen.

Befehl ARN

Der Amazon-Ressourcenname (ARN) des Befehls.

Target ARN

Der ARN des Zielgeräts (Ding oder Client) für die Befehlsausführung.

timestamp

Der UNIX-Zeitstempel für den Zeitpunkt, an dem das Ereignis aufgetreten ist.

Ereignisse im Lebenszyklus

AWS IoT kann Lebenszyklusereignisse zu den MQTT-Themen veröffentlichen. Diese Ereignisse stehen standardmäßig zur Verfügung und können nicht deaktiviert werden.

> **Note**
>
> Nachrichten zum Lebenszyklus können ohne feste Reihenfolge gesendet werden. Unter Umständen erhalten Sie einzelne Nachrichten auch mehrfach.
> `thingName`wird nur aufgenommen, wenn der Client mithilfe der exklusiven Ding-Funktion eine Verbindung herstellt.

In diesem Thema:

- Connect/Disconnect Ereignisse
- Ereignis, dass der Verbindungsversuch fehlgeschlagen ist
- Subscribe/Unsubscribe Ereignisse

Connect/Disconnect Ereignisse

> **Note**
>
> Mit der AWS IoT Gerätemanagement-Flottenindizierung können Sie nach Dingen suchen, Aggregatabfragen ausführen und dynamische Gruppen auf der Grundlage von Connect/Disconnect Dingereignissen erstellen. Weitere Informationen finden Sie unter Flottenindizierung.

AWS IoT veröffentlicht eine Nachricht zu den folgenden MQTT-Themen, wenn ein Client eine Verbindung herstellt oder trennt:

- `$aws/events/presence/connected/`*`clientId`*: Ein Client, der mit dem Message Broker verbunden ist.

- `$aws/events/presence/disconnected/clientId`: Ein Client, dessen Verbindung mit dem Message Broker getrennt wurde

Im Folgenden finden Sie eine Liste der JSON-Elemente, die in den zum Thema veröffentlichten connection/disconnection Nachrichten enthalten sind. `$aws/events/presence/connected/clientId`

clientId

Die Client-ID des Clients, der eine Verbindung herstellt oder trennt

> **Note**
>
> Client-IDs, die „#“ oder „+“ enthalten, empfangen keine Ereignisse zum Lebenszyklus.

thingName

Der Name Ihrer IoT-Sache. `thingName`wird nur aufgenommen, wenn der Client mithilfe der Funktion „Exklusive Dinge“ eine Verbindung herstellt.

Kunde InitiatedDisconnect

„True“, wenn der Client die Verbindungstrennung initiiert hat. Ansonsten „false“. Nur in Trennungsnachrichten.

disconnectReason

Der Grund, warum der Client die Verbindung trennt. Nur in Trennungsnachrichten. Die folgende Tabelle enthält gültige Werte und gibt an, ob der Broker beim Verbindungsabbruch Last Will and Testament (LWT)-Nachrichten sendet.

Grund für das Trennen der Verbindung	Description	Der Broker sendet die LWT-Nachr ichten
`AUTH_ERROR`	Der Client konnte sich nicht authentifizieren, oder die Autorisierung ist fehlgeschlagen.	Ja[1]

Grund für das Trennen der Verbindung	Description	Der Broker sendet die LWT-Nachr ichten
`CLIENT_IN ITIATED_D ISCONNECT`	Der Client gibt an, dass die Verbindung getrennt wird. Der Client kann dies tun, indem er entweder ein `DISCONNECT` MQTT-Steuerpaket sendet oder ein, `Close frame` wenn der Client eine WebSocket Verbindung verwendet.	Nein
`CLIENT_ERROR`	Der Client hat etwas falsch gemacht, das dazu führt, dass die Verbindung getrennt wird. Beispielsweise wird ein Client getrennt, wenn mehr als ein MQTT `CONNECT`-Paket auf derselben Verbindung gesendet wird, oder wenn der Client versucht, mit einer Nutzlast zu veröffent lichen, die das Nutzlast-Limit überschreitet.	Ja
`CONNECTIO N_LOST`	Die Client-Server-Verbindung wird unterbroc hen. Dies kann während eines Zeitraums mit hoher Netzwerklatenz geschehen, oder wenn die Internetverbindung unterbrochen wird.	Ja
`DUPLICATE _CLIENTID`	Der Client verwendet eine Client-ID, die bereits verwendet wird. In diesem Fall wird der Client, der bereits verbunden ist, mit diesem Trenngrund getrennt.	Ja
`FORBIDDEN _ACCESS`	Der Client darf keine Verbindung herstellen. So kann ein Client beispielsweise mit einer abgelehnt en IP-Adresse keine Verbindung herstellen.	Ja[1]
`MQTT_KEEP _ALIVE_TI MEOUT`	Wenn es für das 1,5-fache der Keepalive-Zeit des Clients keine Client-Server-Kommunikation gibt, wird der Client getrennt.	Ja
`SERVER_ERROR`	Trennung der Verbindung aufgrund unerwarteter Serverprobleme.	Ja

Grund für das Trennen der Verbindung	Description	Der Broker sendet die LWT-Nachr ichten
`SERVER_IN ITIATED_D ISCONNECT`	Der Server trennt einen Client aus betrieblichen Gründen absichtlich.	Ja
`API_INITI ATED_DISC ONNECT`	Der Client wurde über die `DeleteConnection` API getrennt.	Ja[2]
`THROTTLED`	Der Client wird getrennt, weil er ein Drosselun gslimit überschritten hat.	Ja
`WEBSOCKET _TTL_EXPI RATION`	Die Verbindung zum Client WebSocket wurde unterbrochen, weil die Verbindung länger als der angegebene Time-to-Live-Wert überschritten wurde.	Ja
`CUSTOMAUT H_TTL_EXP IRATION`	Die Verbindung zum Client wurde getrennt, weil die Verbindung länger als der Time-to-Live-Wert seines benutzerdefinierten Authorizers überschri tten wurde.	Ja

[1] Wenn das Gerät eine aktive Verbindung hatte, bevor dieser Fehler angezeigt wurde.

[2] Um LWT-Nachrichten (Last Will and Testament) `preventWillMessage=true` zu verhindern, legen Sie fest, dass das standardmäßige `DeleteConnection` LWT-Sendeverhalten der API außer Kraft gesetzt wird.

eventType

Der Ereignistyp. Gültige Werte sind `connected` oder `disconnected`.

ipAddress

Die IP-Adresse des verbindenden Clients. Diese kann im IPv4- oder IPv6-Format vorliegen. Nur in Verbindungsnachrichten.

principalIdentifier

Die zur Anmeldung verwendeten Anmeldeinformationen. Für gegenseitige TLS-Authentifizierungszertifikate ist dies die Zertifikat-ID. Für andere Verbindungsarten sind dies die IAM-Anmeldeinformationen.

sessionIdentifier

Ein global eindeutiger Bezeichner AWS IoT , der für die Dauer der Sitzung gültig ist.

Zeitstempel

Eine ungefähre Schätzung, wann das Ereignis eingetreten ist.

versionNumber

Die Versionsnummer für das Lebenszyklusereignis. Dies ist ein monoton ansteigender langer Ganzzahlwert für jede Client-ID-Verbindung. Die Versionsnummer kann von einem Abonnenten verwendet werden, um die Reihenfolge der Lebenszyklusereignisse festzustellen.

> **Note**
>
> Die Verbindungs- und Trennungsnachrichten für eine Client-Verbindung haben dieselbe Versionsnummer.
> Die Versionsnummer kann Werte überspringen und wird nicht zwingend konsistent für jedes Ereignis um 1 erhöht.
> Wenn ein Client etwa eine Stunde lang nicht verbunden ist, wird die Versionsnummer auf 0 zurückgesetzt. Bei persistenten Sitzungen wird die Versionsnummer auf 0 zurückgesetzt, nachdem ein Client für länger als die konfigurierte TTL (Time-to-live) für die persistente Sitzung getrennt war.

Eine Verbindungsnachricht hat die folgende Struktur.

```
{
    "clientId": "186b5",
    "thingName": "exampleThing",
    "timestamp": 1573002230757,
    "eventType": "connected",
    "sessionIdentifier": "00000000-0000-0000-0000-000000000000",
    "principalIdentifier": "12345678901234567890123456789012",
    "ipAddress": "192.0.2.0",
```

```
    "versionNumber": 0
}
```

Eine Trennungsnachricht hat die folgende Struktur.

```
{
    "clientId": "186b5",
    "thingName": "exampleThing",
    "timestamp": 1573002340451,
    "eventType": "disconnected",
    "sessionIdentifier": "00000000-0000-0000-0000-000000000000",
    "principalIdentifier": "12345678901234567890123456789012",
    "clientInitiatedDisconnect": true,
    "disconnectReason": "CLIENT_INITIATED_DISCONNECT",
    "versionNumber": 0
}
```

Umgang mit Client-Verbindungstrennungen

Es ist eine bewährte Methode, für Lebenszyklusereignisse (einschließlich Last Will and Testament (LWT)-Nachrichten) stets einen implementierten Wartestatus zu verwenden. Wenn eine getrennte Verbindung signalisiert wird, sollte Ihr Code eine gewisse Zeit warten und überprüfen, ob ein Gerät noch offline ist, bevor Maßnahmen ergriffen werden. Eine Möglichkeit dazu ist das Verwenden von SQS-Verzögerungswarteschlangen. Wenn ein Client ein LWT- oder Lebenszyklusereignis erhält, können Sie eine Nachricht für (beispielsweise) ca. 5 Sekunden in eine Warteschlange stellen. Wenn die Nachricht verfügbar ist und verarbeitet wird (durch Lambda oder einen anderen Service), können Sie zunächst überprüfen, ob das Gerät immer noch offline ist, bevor Sie zu weiteren Maßnahmen greifen.

Ereignis, dass der Verbindungsversuch fehlgeschlagen ist

AWS IoT veröffentlicht eine Nachricht zum folgenden MQTT-Thema, wenn ein Client nicht autorisiert ist, eine Verbindung herzustellen, oder wenn ein Testament konfiguriert ist und der Client nicht berechtigt ist, zu diesem letzten Testament-Thema zu veröffentlichen.

```
$aws/events/presence/connect_failed/clientId
```

Im Folgenden finden Sie eine Liste der JSON-Elemente, die in den für das `$aws/events/presence/connect_failed/clientId` Thema veröffentlichten Verbindungsautorisierungsnachrichten enthalten sind.

clientId

Die Client-ID des Clients, der versucht hat, eine Verbindung herzustellen, aber fehlgeschlagen ist.

> **Note**
>
> Client-IDs, die „#“ oder „+“ enthalten, empfangen keine Ereignisse zum Lebenszyklus.

thingName

Der Name Ihrer IoT-Sache. `thingName`wird nur aufgenommen, wenn der Client mithilfe der Funktion „Exklusive Dinge“ eine Verbindung herstellt.

Zeitstempel

Eine ungefähre Schätzung, wann das Ereignis eingetreten ist.

eventType

Der Ereignistyp. Gültiger Wert ist`connect_failed`.

verbinden FailureReason

Der Grund, warum die Verbindung fehlschlägt. Gültiger Wert ist`AUTHORIZATION_FAILED`.

principalIdentifier

Die zur Anmeldung verwendeten Anmeldeinformationen. Für gegenseitige TLS-Authentifizierungszertifikate ist dies die Zertifikat-ID. Für andere Verbindungsarten sind dies die IAM-Anmeldeinformationen.

sessionIdentifier

Ein global eindeutiger Bezeichner AWS IoT , der für die Dauer der Sitzung gültig ist.

ipAddress

Die IP-Adresse des verbindenden Clients. Diese kann im IPv4- oder IPv6-Format vorliegen. Nur in Verbindungsnachrichten.

Eine Meldung über einen Verbindungsausfall hat die folgende Struktur.

```
{
```

```
    "clientId": "186b5",
    "thingName": "exampleThing",
    "timestamp": 1460065214626,
    "eventType": "connect_failed",
    "connectFailureReason": "AUTHORIZATION_FAILED",
    "principalIdentifier": "123456789012345678901234567890123456789012",
    "sessionIdentifier": "00000000-0000-0000-0000-000000000000",
    "ipAddress" : "192.0.2.0"
}
```

Subscribe/Unsubscribe Ereignisse

AWS IoT veröffentlicht eine Nachricht zum folgenden MQTT-Thema, wenn ein Client ein MQTT-Thema abonniert oder abbestellt:

```
$aws/events/subscriptions/subscribed/clientId
```

oder

```
$aws/events/subscriptions/unsubscribed/clientId
```

Hierbei ist `clientId` die MQTT-Client-ID, über die eine Verbindung zum AWS IoT Message Broker hergestellt wird.

Die im Topic veröffentlichte Nachricht weist die folgende Struktur auf:

```
{
    "clientId": "186b5",
    "thingName": "exampleThing",
    "timestamp": 1460065214626,
    "eventType": "subscribed" | "unsubscribed",
    "sessionIdentifier": "00000000-0000-0000-0000-000000000000",
    "principalIdentifier": "123456789012345678901234567890123456789012",
    "topics" : ["foo/bar","device/data","dog/cat"]
}
```

Die nachfolgende Liste enthält JSON-Elemente, die in „Abonnieren/Abonnement abbestellen“-Nachrichten enthalten sind, die an die Topics `$aws/events/subscriptions/subscribed/clientId` und `$aws/events/subscriptions/unsubscribed/clientId` gesendet werden.

clientId

Die Client-ID des Clients, der ein Topic abonniert oder das Abonnement abbestellt.

> **Note**
>
> Client-IDs, die „#“ oder „+“ enthalten, empfangen keine Ereignisse zum Lebenszyklus.

thingName

Der Name Ihres IoT-Dings. `thingName`wird nur aufgenommen, wenn der Client mithilfe der Funktion „Exklusive Dinge“ eine Verbindung herstellt.

eventType

Der Ereignistyp. Gültige Werte sind `subscribed` oder `unsubscribed`.

principalIdentifier

Die zur Anmeldung verwendeten Anmeldeinformationen. Für gegenseitige TLS-Authentifizierungszertifikate ist dies die Zertifikat-ID. Für andere Verbindungsarten sind dies die IAM-Anmeldeinformationen.

sessionIdentifier

Ein global eindeutiger Bezeichner AWS IoT , der für die Dauer der Sitzung gültig ist.

Zeitstempel

Eine ungefähre Schätzung, wann das Ereignis eingetreten ist.

topics

Ein Array der vom Client abonnierten MQTT-Topics.

> **Note**
>
> Nachrichten zum Lebenszyklus können ohne feste Reihenfolge gesendet werden. Unter Umständen erhalten Sie einzelne Nachrichten auch mehrfach.

Fehlerbehebung AWS IoT

Helfen Sie uns, dieses Thema zu verbessern

Lassen Sie uns wissen, was dazu beitragen würde, es besser zu machen

Die folgenden Informationen helfen Ihnen möglicherweise bei der Lösung häufiger Probleme in AWS IoT.

Aufgaben

- AWS IoT Core Anleitung zur Fehlerbehebung
- AWS IoT Device Management Anleitung zur Fehlerbehebung
- AWS IoT Anleitung zur Fehlerbehebung bei Device Advisor
- AWS IoT Fehler

AWS IoT Core Anleitung zur Fehlerbehebung

Helfen Sie uns, dieses Thema zu verbessern

Lassen Sie uns wissen, was dazu beitragen würde, es besser zu machen

Dies ist der Abschnitt zur Fehlerbehebung für AWS IoT Core.

Topics

- Diagnostizieren von Verbindungsproblemen
- Fehler bei der Regeldiagnose
- Diagnostizieren von Problemen mit Schatten
- Diagnostizieren von Stream-Limits
- Behebung von Verbindungsabbrüchen bei Geräteflotten

Diagnostizieren von Verbindungsproblemen

Helfen Sie uns, dieses Thema zu verbessern

Lassen Sie uns wissen, was dazu beitragen würde, es besser zu machen

Für eine erfolgreiche Verbindung zu AWS IoT ist Folgendes erforderlich:

- Eine gültige Verbindung
- Ein gültiges und aktives Zertifikat
- Eine Richtlinie, die die gewünschte Verbindung und den gewünschten Vorgang ermöglicht

Connection (Verbindung)

Wie finde ich den richtigen Endpunkt?

- die von `aws iot describe-endpoint --endpoint-type iot:Data-ATS` zurückgegebene `endpointAddress`

  oder

- den von `aws iot describe-domain-configuration --domain-configuration-name "domain_configuration_name"` zurückgegebenen `domainName`

Wie finde ich den richtigen Wert für Server Name Indication (SNI)?

Der richtige SNI-Wert ist `endpointAddress`, der vom describe-endpoint-Befehl zurückgegeben wird, oder `domainName`, der vom describe-domain-configuration-Befehl zurückgegeben wird. Es ist dieselbe Adresse wie der Endpunkt im vorherigen Schritt. Beim Anschließen von Geräten an AWS IoT Core können Clients die Erweiterung Server Name Indication (SNI) senden. Dies ist nicht erforderlich, wird aber dringend empfohlen. Um Funktionen wie die Registrierung mehrerer Konten, benutzerdefinierte Domänen und VPC-Endpunkte zu verwenden, müssen Sie die SNI-Erweiterung verwenden. Weitere Informationen finden Sie unter Transportsicherheit unter AWS IoT.

Wie löse ich ein anhaltendes Verbindungsproblem?

Sie können AWS Device Advisor verwenden, um bei der Problembehandlung zu helfen. Die vorgefertigten Tests von Device Advisor helfen Ihnen dabei, Ihre Gerätesoftware anhand von Best

Practices für die Verwendung von TLS, MQTT, AWS IoT Geräteschatten und AWS IoT -Jobs zu validieren.

Hier ist ein Link zu den vorhandenen Device Advisor-Inhalten.

Authentifizierung

Geräte müssen authentifiziert werden, um eine Verbindung zu AWS IoT Endpunkten herstellen zu können. Für Geräte, die X.509 Client-Zertifikate zur Authentifizierung verwendet werden, müssen die Zertifikate registriert AWS IoT und aktiv sein.

Wie authentifizieren meine Geräte AWS IoT Endpunkte?

Fügen Sie das AWS IoT CA-Zertifikat dem Trust Store Ihres Kunden hinzu. Lesen Sie die Dokumentation zu Serverauthentifizierung in AWS IoT Core und folgen Sie dann den Links, um das entsprechende CA-Zertifikat herunterzuladen.

Was wird überprüft, wenn ein Gerät eine Verbindung herstellt AWS IoT?

Wenn ein Gerät versucht, eine Verbindung mit AWS IoT herzustellen:

1. AWS IoT prüft, ob ein gültiges Zertifikat und ein SNI-Wert (Server Name Indication) gültig sind.
2. AWS IoT überprüft, ob das verwendete Zertifikat für das AWS IoT Konto registriert ist und ob es aktiviert wurde.
3. Wenn ein Gerät versucht, eine Aktion auszuführen AWS IoT, z. B. eine Nachricht zu abonnieren oder zu veröffentlichen, wird die Richtlinie, die dem Zertifikat beigefügt ist, mit dem es eine Verbindung hergestellt hat, überprüft, um zu bestätigen, dass das Gerät zur Ausführung dieser Aktion autorisiert ist.

Wie kann ich ein korrekt konfiguriertes Zertifikat validieren?

Verwenden Sie den OpenSSL `s_client`-Befehl, um die Verbindung zum AWS IoT -Endpunkt zu überprüfen:

```
openssl s_client -connect custom_endpoint.iot.aws-region.amazonaws.com:8443 -
CAfile CA.pem -cert cert.pem -key privateKey.pem
```

Weitere Informationen zur Verwendung von `openssl s_client` finden Sie in der OpenSSL s_client-Dokumentation.

Wie überprüfe ich den Status eines Zertifikats?

- Listen Sie die Zertifikate auf

Wenn Sie die Zertifikat-ID nicht kennen, können Sie den Status all Ihrer Zertifikate mit dem `aws iot list-certificates`-Befehl anzeigen.

- Zertifikatdetails anzeigen

  Wenn Sie die ID des Zertifikats kennen, zeigt Ihnen dieser Befehl detailliertere Informationen zum Zertifikat an.

  ```
  aws iot describe-certificate --certificate-id "certificateId"
  ```

- Überprüfen Sie das Zertifikat in der AWS IoT Konsole

  Wählen Sie in der AWS IoT Konsole im linken Menü die Option Sicher und dann Zertifikate aus.

  Wählen Sie das Zertifikat aus der Liste aus, das Sie für die Verbindung verwenden, um die Detailseite zu öffnen.

  Auf der Detailseite des Zertifikats können Sie seinen aktuellen Status sehen.

  Der Zertifikatstatus kann mit dem Menü Aktionen in der oberen rechten Ecke der Detailseite geändert werden.

Autorisierung

AWS IoT RessourcenAWS IoT Core Richtlinien, die verwendet werden, um diese Ressourcen zum Ausführen von Aktionen zu autorisieren. Damit eine Aktion autorisiert werden kann, muss den angegebenen AWS IoT Ressourcen ein Richtliniendokument beigefügt sein, das die Berechtigung zum Ausführen dieser Aktion erteilt.

Ich habe die Antwort `PUBNACK` oder `SUBNACK` vom Broker erhalten. Was soll ich tun?

Vergewissern Sie sich, dass dem Zertifikat, das Sie für den Anruf verwenden, eine Richtlinie beigefügt ist AWS IoT. Alle publish/subscribe Operationen werden standardmäßig verweigert.

Stellen Sie sicher, dass die beigefügte Richtlinie die Aktionen autorisiert, die Sie ausführen möchten.

Stellen Sie sicher, dass die beigefügte Richtlinie die Ressourcen autorisiert, die versuchen, die autorisierten Aktionen auszuführen.

Ich habe einen AUTHORIZATION_FAILURE-Eintrag in meinen Protokollen.

Stellen Sie sicher, dass dem Zertifikat, das Sie für den Anruf verwenden, eine Richtlinie beigefügt ist AWS IoT. Alle publish/subscribe Operationen werden standardmäßig verweigert.

Stellen Sie sicher, dass die beigefügte Richtlinie die Aktionen autorisiert, die Sie ausführen möchten.

Stellen Sie sicher, dass die beigefügte Richtlinie die Ressourcen autorisiert, die versuchen, die autorisierten Aktionen auszuführen.

Wie überprüfe ich, was die Richtlinie autorisiert?

Wählen Sie in der AWS IoT Konsole im linken Menü Sicherheit und dann Zertifikate aus.

Wählen Sie das Zertifikat aus der Liste aus, das Sie für die Verbindung verwenden, um die Detailseite zu öffnen.

Auf der Detailseite des Zertifikats können Sie seinen aktuellen Status sehen.

Wählen Sie im linken Menü der Detailseite des Zertifikats die Option Richtlinien aus, um die mit dem Zertifikat verknüpften Richtlinien anzuzeigen.

Wählen Sie die gewünschte Richtlinie aus, um ihre Detailseite aufzurufen.

Sehen Sie sich auf der Detailseite der Richtlinie das Richtliniendokument der Richtlinie an, um zu sehen, was damit autorisiert wird.

Wählen Sie Richtliniendokument bearbeiten, um Änderungen am Richtliniendokument vorzunehmen.

Sicherheits- und Identitätsmethoden

Wenn Sie die Serverzertifikate für die AWS IoT benutzerdefinierte Domänenkonfiguration angeben, haben die Zertifikate maximal vier Domainnamen.

Weitere Informationen finden Sie unter AWS IoT Core -Endpunkte und -Kontingente.

Fehler bei der Regeldiagnose

Helfen Sie uns, dieses Thema zu verbessern

Lassen Sie uns wissen, was dazu beitragen würde, es besser zu machen

In diesem Abschnitt werden einige Dinge beschrieben, die Sie überprüfen sollten, wenn Sie auf ein Problem mit einer Regel stoßen.

Konfiguration von CloudWatch Protokollen zur Problembehandlung

Die beste Methode, um Probleme mit Regeln zu debuggen, ist die Verwendung von CloudWatch Logs. Wenn Sie CloudWatch Logs für aktivieren AWS IoT, können Sie sehen, welche Regeln ausgelöst werden und ob sie erfolgreich waren oder nicht. Sie werden ebenfalls informiert, ob die Bedingungen der WHERE-Klausel zutreffen. Weitere Informationen finden Sie unter Überwachen AWS IoT mithilfe von Protokollen CloudWatch.

Der häufigsten Probleme in Zusammenhang mit Regeln betreffen die Autorisierung. In den Protokollen wird angezeigt, ob Ihre Rolle nicht autorisiert ist, AssumeRole die Ressource zu bearbeiten. Hier sehen Sie ein Beispielprotokoll bei einer differenzierten Protokollierung:


```
{
    "timestamp": "2017-12-09 22:49:17.954",
    "logLevel": "ERROR",
    "traceId": "ff563525-6469-506a-e141-78d40375fc4e",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "RuleExecution",
    "clientId": "iotconsole-123456789012-3",
    "topicName": "test-topic",
    "ruleName": "rule1",
    "ruleAction": "DynamoAction",
    "resources": {
        "ItemHashKeyField": "id",
        "Table": "trashbin",
        "Operation": "Insert",
        "ItemHashKeyValue": "id",
        "IsPayloadJSON": "true"
    },
    "principalId": "ABCDEFG1234567ABCD890:outis",
```

```
    "details": "User: arn:aws:sts::123456789012:assumed-role/dynamo-
testbin/5aUMInJH is not authorized to perform: dynamodb:PutItem on
 resource: arn:aws:dynamodb:us-east-1:123456789012:table/testbin (Service:
 AmazonDynamoDBv2; Status Code: 400; Error Code: AccessDeniedException; Request ID:
 AKQJ987654321AKQJ123456789AKQJ987654321AKQJ987654321)"
}
```

Hier sehen Sie ein ähnliches Beispielprotokoll bei einer globalen Protokollierung:

```
2017-12-09 22:49:17.954 TRACEID:ff562535-6964-506a-e141-78d40375fc4e
PRINCIPALID:ABCDEFG1234567ABCD890:outis [ERROR] EVENT:DynamoActionFailure
TOPICNAME:test-topic CLIENTID:iotconsole-123456789012-3
MESSAGE:Dynamo Insert record failed. The error received was User:
 arn:aws:sts::123456789012:assumed-role/dynamo-testbin/5aUMInJI is not authorized to
 perform: dynamodb:PutItem on resource: arn:aws:dynamodb:us-east-1:123456789012:table/
testbin
(Service: AmazonDynamoDBv2; Status Code: 400; Error Code: AccessDeniedException;
 Request ID: AKQJ987654321AKQJ987654321AKQJ987654321AKQJ987654321).
Message arrived on: test-topic, Action: dynamo, Table: trashbin, HashKeyField: id,
 HashKeyValue: id, RangeKeyField: None, RangeKeyValue: 123456789012
No newer events found at the moment. Retry.
```

Weitere Informationen finden Sie unter the section called “Anzeigen AWS IoT loggt sich in der CloudWatch Konsole ein”.

Diagnose externer Services

Externe Services werden vom Endbenutzer reguliert. Stellen Sie vor der Ausführung der Regel sicher, dass die externen Services, die Sie mit Ihrer Regel verknüpft haben, eingerichtet sind und über genügend Durchsatz- und Kapazitätseinheiten für Ihre Anwendung verfügen.

Diagnose von SQL-Problemen

Wenn Ihre SQL-Abfrage nicht die erwarteten Daten zurückgibt:

- Überprüfen Sie die Protokolle auf Fehlermeldungen.
- Vergewissern Sie sich, dass Ihre SQL-Syntax mit dem JSON-Dokument in der Nachricht übereinstimmt.

Vergleichen Sie die in der Abfrage verwendeten Objekt- und Eigenschaftsnamen mit denen, die im JSON-Dokument der Nachrichtennutzlast des Themas verwendet wurden. Weitere Informationen über die JSON-Formatierung in SQL-Abfragen finden Sie unter JSON-Erweiterungen.

- Prüfen Sie, ob die JSON-Objekt- oder Eigenschaftsnamen reservierte oder numerische Zeichen enthalten.

  Weitere Hinweise zu reservierten Zeichen in JSON-Objektreferenzen in SQL-Abfragen finden Sie unter JSON-Erweiterungen.

Diagnostizieren von Problemen mit Schatten

Helfen Sie uns, dieses Thema zu verbessern

Lassen Sie uns wissen, was dazu beitragen würde, es besser zu machen

Diagnostizieren von Schatten

Problem	Richtlinien für die Fehlerbehebung
Das Schatten-Dokument eines Geräts wird mit `Invalid JSON document` abgelehnt.	Wenn Sie sich mit JSON nicht auskennen, passen Sie die hier aufgeführten Beispiele nach Ihrem Bedarf an. Weitere Informationen finden Sie unter Beispiele für Schatten-Dokumente.
Ich habe ein korrektes JSON-Dokument eingereicht, aber nichts oder nur Teile davon sind im Geräteschattendokument gespeichert.	Stellen Sie sicher, dass Sie die JSON-Form atierungsrichtlinien einhalten. Nur JSON-Feld er in den Abschnitten `desired` und `reported` werden gespeichert. JSON-Inhalte außerhalb dieser Abschnitte werden ignoriert (auch wenn sie formal korrekt sind).
Ich habe eine Fehlermeldung erhalten, dass mein Geräteschatten die zulässige Größe überschreitet.	Der Geräteschatten unterstützt nur 8 KB Daten. Versuchen Sie, Feldnamen in Ihrem JSON-Dokument zu kürzen, oder erstellen Sie einfach mehrere Schatten, indem Sie mehrere Objekte erstellen. Einem Gerät kann eine unbegrenzte

Problem	Richtlinien für die Fehlerbehebung
	Anzahl von Geräten things/shadows zugeordnet werden. Voraussetzung ist lediglich, dass der Name des Objekts in Ihrem Konto nur einmal verwendet werden darf.
Wenn ich einen Geräteschatten erhalte, ist er größer als 8 KB. Wie kann das sein?	Nach Erhalt fügt der AWS IoT Dienst dem Shadow des Geräts Metadaten hinzu. Der Service fügt diese Daten bei der Antwort hinzu, aber sie sind nicht in der Obergrenze von 8 KB enthalten. Nur die Daten für den Status `desired` und den Status `reported` des Statusdokuments, die an den Geräteschatten gesandt werden, werden für den Grenzwert berücksichtigt.
Meine Anfrage wurde aufgrund einer inkorrekten Version abgelehnt. Was soll ich tun?	Führen Sie einen GET-Vorgang durch, um auf die letzte Dokumentenversion zu synchronisieren. Wenn Sie MQTT verwenden, abonnieren Sie den. /update/accepted Thema, um über Statusänderungen informiert zu werden und die neueste Version des JSON-Dokuments zu erhalten.
Der Zeitstempel ist um einige Sekunden ungenau.	Der Zeitstempel für einzelne Felder und das gesamte JSON-Dokument wird aktualisiert, wenn das Dokument vom AWS IoT Dienst empfangen wird oder wenn das Statusdokument auf dem veröffentlicht wird. /update/accepted und. /update/delta Nachricht. Nachrichten können im Netzwerk verspätet sein, sodass der Zeitstempel um einige Sekunden abweicht.

Problem	Richtlinien für die Fehlerbehebung
Ich kann mein Gerät in den entsprechenden Schattenthemen veröffentlichen und anmelden, aber wenn ich versuche, das Schattend okument über die HTTP-REST-API zu aktualisi eren, erhalte ich den Fehler HTTP 403.	Stellen Sie sicher, dass Sie in IAM Richtlinien für den Zugriff auf diese Themen und für die entsprechende Aktion (UPDATE/GET/DELETE) für die verwendeten Anmeldeinformationen erstellt haben. IAM-Richtlinien und Zertifika tsrichtlinien sind unabhängig.
Sonstige Probleme	Der Device Shadow-Dienst protokolliert Fehler in den CloudWatch Protokollen. Um Geräte- und Konfigurationsprobleme zu identifizieren, aktivieren Sie CloudWatch Protokolle und sehen Sie sich die Protokolle mit Debug-Inf ormationen an.

Diagnostizieren von Stream-Limits

Fehlerbehebung von „Stream-Begrenzung für Ihr AWS -Konto überschritten“

Wenn `"Error: You have exceeded the limit for the number of streams in your AWS account."` angezeigt wird, können Sie die nicht verwendeten Streams in Ihrem Konto bereinigen, anstatt eine Limiterhöhung anzufordern.

So bereinigen Sie einen unbenutzten Stream, den Sie mit dem AWS CLI oder SDK erstellt haben:

```
aws iot delete-stream –stream-id value
```

Weitere Informationen finden Sie unter delete-stream.

> **Note**
>
> Sie können den `list-streams`-Befehl verwenden, um die Stream-IDs zu finden.

Behebung von Verbindungsabbrüchen bei Geräteflotten

Helfen Sie uns, dieses Thema zu verbessern

Lassen Sie uns wissen, was dazu beitragen würde, es besser zu machen

AWS IoT Unterbrechungen der Geräteflotte können aus mehreren Gründen auftreten. In diesem Artikel wird erklärt, wie ein Grund für die Unterbrechung der Verbindung diagnostiziert wird und wie mit Verbindungsabbrüchen umgegangen wird, die durch eine regelmäßige Wartung des AWS IoT Dienstes oder eine Drosselungsbegrenzung verursacht werden.

Um den Grund für die Unterbrechung zu diagnostizieren

Sie können die AWSIotLogsV2 Protokollgruppe einchecken CloudWatch, um den Grund für die Trennung im `disconnectReason` Feld des Protokolleintrags zu ermitteln.

Sie können auch die Funktion für AWS IoT Lebenszyklusereignisse verwenden, um den Grund für die Trennung zu ermitteln. Wenn Sie das Trennungsereignis (`$aws/events/presence/disconnected/clientId`) des Lebenszyklus abonniert haben, erhalten Sie eine Benachrichtigung, AWS IoT sobald die Verbindung unterbrochen wird. Sie können den Grund für die Unterbrechung der Verbindung im `disconnectReason`-Feld der Benachrichtigung identifizieren.

Weitere Informationen finden Sie unter CloudWatch AWS IoT Logeinträge und Lifecycle-Ereignisse.

Zur Behebung von Verbindungsabbrüchen aufgrund von AWS IoT Wartungsarbeiten

Verbindungsabbrüche, die durch Wartungsarbeiten verursacht wurden AWS IoT, werden als `SERVER_INITIATED_DISCONNECT` Lebenszyklusereignis und protokolliert. AWS IoT CloudWatch Um diese Verbindungsabbrüche zu beheben, passen Sie Ihre clientseitigen Einstellungen an, um sicherzustellen, dass Ihre Geräte automatisch wieder mit der Plattform verbunden werden können. AWS IoT

Zur Behebung von Verbindungsabbrüchen aufgrund einer Drosselungsgrenze

Verbindungsabbrüche, die durch ein Drosselungslimit verursacht werden, werden als Lebenszyklusereignis und protokolliert. `THROTTLED` AWS IoT CloudWatch Um diese Verbindungsabbrüche zu bewältigen, können Sie beantragen, dass das Message-Broker-Limit erhöht wird, wenn die Anzahl der Geräte zunimmt.

Weitere Informationen finden Sie unter AWS IoT Core Message Broker.

AWS IoT Device Management Anleitung zur Fehlerbehebung

> Helfen Sie uns, dieses Thema zu verbessern
>
> Lassen Sie uns wissen, was dazu beitragen würde, es besser zu machen

Dies ist der Abschnitt zur Fehlerbehebung für AWS IoT Device Management.

Topics

- AWS IoT Probleme bei Aufträgen
- AWS IoT Befehle zur Problembehandlung
- Fehlerbehebung bei der Flottenindizierung
- AWS IoT Fehlerbehebung im Softwarepaketkatalog für die Geräteverwaltung

AWS IoT Probleme bei Aufträgen

Dies ist der Abschnitt zur Problembehandlung für AWS IoT Jobs.

Wie finde ich eine AWS IoT Endpunkt „Jobs“?

Wie finde ich den Endpunkt der AWS IoT Jobs-Kontrollebene?

AWS IoT Jobs unterstützt API-Operationen auf Steuerungsebene mithilfe des HTTPS-Protokolls. Stellen Sie sicher, dass Sie über das HTTPS-Protokoll eine Verbindung zum richtigen Endpunkt der Steuerebene hergestellt haben.

Eine Liste der AWS regionsspezifischen Endpunkte finden Sie unter AWS IoT Core — Control Plane-Endpoints.

Eine Liste der FIPS-kompatiblen Endpunkte der AWS IoT -Auftragssteuerebene finden Sie unter FIPS-Endpunkte nach Dienst

> Note
>
> AWS IoT Jobs und AWS IoT Core teilen sich dieselben Endpunkte. AWS Region-specific

Wie finde ich den Endpunkt der AWS IoT Jobs-Datenebene?

AWS IoT Jobs unterstützt API-Operationen auf Datenebene mithilfe der HTTPS- und MQTT-Protokolle. Stellen Sie sicher, dass Sie über das HTTPS- oder MQTT-Protokoll eine Verbindung zum richtigen Endpunkt der Datenebene hergestellt haben.

- Protokoll: HTTPS
  - Verwenden Sie den folgenden describe-endpoint-CLI-Befehl, der unten gezeigt wird, oder die `DescribeEndpoint`-REST-API. Verwenden Sie `iot:Jobs` für den Endpunkttyp.

    ```
    aws iot describe-endpoint --endpoint-type iot:Jobs
    ```

- MQTT-Protokoll
  - Verwenden Sie den folgenden describe-endpoint-CLI-Befehl, der unten gezeigt wird, oder die `DescribeEndpoint`-REST-API. Verwenden Sie `iot:Data-ATS` für den Endpunkttyp.

    ```
    aws iot describe-endpoint --endpoint-type iot:Data-ATS
    ```

Eine Liste der FIPS-kompatiblen AWS IoT Auftrags-Datenebenen-Endpunkte finden Sie unter FIPS-Endpunkte nach Dienst

Wie überwache ich AWS IoT Jobaktivitäten und Kennzahlen bereitstellen?

Die Überwachung der AWS IoT Jobaktivitäten mithilfe von Amazon CloudWatch bietet einen Überblick über den laufenden AWS IoT Auftragsbetrieb in Echtzeit und hilft, die Kosten mithilfe von CloudWatch Alarmen mithilfe von AWS IoT Regeln zu kontrollieren. Sie müssen die Protokollierung konfigurieren, bevor Sie die AWS IoT Jobaktivitäten überwachen und CloudWatch Alarme einrichten können. Weitere Informationen zum Einrichten der Protokollierung finden Sie unter Konfiguration AWS IoT Protokollierung.

Weitere Informationen zu Amazon CloudWatch und dazu, wie Sie mithilfe einer IAM-Benutzerrolle die Berechtigung zur Nutzung von CloudWatch Ressourcen einrichten, finden Sie unter Identitäts- und Zugriffsmanagement für Amazon CloudWatch.

Wie richte ich AWS IoT Job-Metriken und die Überwachung mithilfe von Amazon CloudWatch ein?

Um die AWS IoT Protokollierung einzurichten, folgen Sie den unter AWS IoT Protokollierung konfigurieren beschriebenen Schritten. AWS IoT Die Einrichtung der Protokollierung kann in der

AWS-Managementkonsole AWS CLI, oder API erfolgen. AWS IoT Die Protokollierung, die für bestimmte Dinggruppen eingerichtet wurde, muss nur in der AWS CLI oder API erfolgen.

Der Abschnitt AWS IoT Job-Metriken enthält die AWS IoT Job-Metriken, die zur Überwachung der AWS IoT Jobaktivitäten verwendet werden. Es wird erklärt, wie die Metriken im Feld AWS-Managementkonsole und angezeigt AWS CLI werden.

Darüber hinaus können Sie CloudWatch Alarme einrichten, um Sie auf bestimmte Metriken aufmerksam zu machen, die Sie genau überwachen möchten. Eine Anleitung zur Einrichtung von Alarmen finden Sie unter CloudWatch Amazon-Alarme verwenden.

Problembehebung bei Geräteflotten und einzelnen Geräten

Bei der Ausführung eines Auftrags wird der Status von **`QUEUED`** auf unbestimmte Zeit beibehalten

Wenn eine Auftragsausführung mit einem Status von `QUEUED` nicht zum nächsten logischen Status übergeht, z. B. `IN_PROGRESS`, `FAILED` oder `TIMED_OUT`, kann eines der folgenden Szenarien die Ursache sein:

- Überprüfen Sie Ihre Geräteaktivität in den CloudWatch Protokollen in der CloudWatch Konsole. Weitere Informationen finden Sie unter Überwachen AWS IoT mithilfe von CloudWatch Protokollen.
- Die IAM-Rolle, die dem Job und der nachfolgenden Auftragsausführung zugeordnet ist, verfügt möglicherweise nicht über die richtigen Berechtigungen, die in einer der Richtlinienerklärungen der IAM-Richtlinie aufgeführt sind, die dieser IAM-Rolle zugeordnet ist. Verwenden Sie die `describe-job`-API, um die IAM-Rolle zu identifizieren, die mit diesem Auftrag und der nachfolgenden Auftragsausführung verknüpft ist, und überprüfen Sie die IAM-Richtlinie auf die richtigen Berechtigungen. Sobald die Richtlinienberechtigungserklärungen aktualisiert wurden, sollten Sie in der Lage sein, den `AssumeRole`-API-Befehl für die Ressource auszuführen.

Für mein Objekt oder meine Objektgruppe wurde keine Auftragsausführung erstellt

Wenn der Status eines Auftrags auf `IN_PROGRESS` aktualisiert wird, beginnt der Rollout des Auftragsdokuments auf allen Geräten in Ihrer Zielgruppe. Durch diese Statusaktualisierung wird für jedes Zielgerät eine Auftragsausführung erstellt. Wenn für eines der Zielgeräte keine Auftragsausführung erstellt wurde, lesen Sie die folgenden Anleitungen:

- Wirkt sich der Auftrag direkt auf `thing` aus, hat der Auftrag den Status `IN_PROGRESS`, und läuft der Auftrag gleichzeitig? Wenn alle drei Bedingungen erfüllt sind, sendet der Auftrag immer

noch Auftragsausführungen an alle Geräte in Ihrer Zielgruppe, und für dieses spezielle Gerät `thing` wurde der Auftrag noch nicht ausgeführt.

- Prüfen Sie in der AWS Management Console die Geräte in Ihrer Zielgruppe auf den Job und den Jobstatus, oder verwenden Sie den `describe-job` API-Befehl.
- Verwenden Sie den `describe-job`-API-Befehl, um zu überprüfen, ob die `IsConcurrent`-Eigenschaft für den Auftrag auf „Wahr“ oder „Falsch“ festgelegt ist. Weitere Informationen finden Sie unter Auftragslimits.

- Der Auftrag zielt nicht direkt auf `thing` ab.
  - Wenn `Thing` zu einem `ThingGroup` hinzugefügt wurde und der Auftrag auf `ThingGroup` abzielte, überprüfen Sie, ob `Thing` Teil von `ThingGroup` ist.
  - Handelt es sich bei dem Auftrag um einen Snapshot-Auftrag mit dem Status `IN_PROGRESS` der gleichzeitig läuft, sendet der Auftrag weiterhin Auftragsausführungen an alle Geräte in Ihrer Zielgruppe, und dieser spezifische `Thing` hat seine Auftragsausführung noch nicht erhalten.
  - Wenn es sich bei dem Auftrag um einen kontinuierlichen gleichzeitigen Auftrag mit dem Status `IN_PROGRESS` handelt, sendet der Auftrag immer noch Auftragsausführungen an alle Geräte in Ihrer Zielgruppe, und dieser spezifische `Thing` hat seine Auftragsausführung noch nicht erhalten. Nur für fortlaufende Aufträge können Sie den `Thing` aus dem `ThingGroup` entfernen und dann den `Thing` wieder dem `ThingGroup` hinzufügen.
  - Wenn es sich bei dem Job um einen Snapshot-Job mit dem Statusstatus handelt `IN_PROGRESS` und dieser nicht gleichzeitig ausgeführt wird, ist es wahrscheinlich, dass die Beziehung `Thing` oder die `ThingGroup` Mitgliedschaft von AWS IoT Jobs nicht bestätigt wird. Es wird empfohlen, nach Ihrem `AddThingToThingGroup` Anruf einige Sekunden Wartezeit einzuplanen, bevor Sie Ihren Anruf erstellen. `Job` Alternativ können Sie die Zielauswahl auf auf umstellen`Continuous`, sodass der Dienst das Ereignis „Verspätung“ `Thing` und „Zuweisung einer `ThingGroup` Mitgliedschaft“ automatisch ausfüllt.

Ein neuer Auftrag schlägt aufgrund eines **`LimitedExceededException`**-Fehlers fehl

Wenn Ihre Auftragserstellung mit der Fehlerantwort von `LimitedExceededException` fehlschlägt, rufen Sie die `list-jobs`-API auf und überprüfen Sie alle Aufträge mit `isConcurrent=true`, um festzustellen, ob Sie das Limit für die gleichzeitige Ausführung von Aufträgen erreicht haben. Weitere Informationen zu gleichzeitigen Aufträgen finden Sie unter Auftragslimits. Um Ihre Limits für die gleichzeitige Ausführung von Aufträgen zu sehen und eine

Erhöhung des Limits zu beantragen, gehen Sie zu AWS IoT Device Management Auftragslimits und -kontingente.

Begrenzung der Dokumentgröße

Die Größe des Auftragsdokuments ist durch die Größe der MQTT-Payload begrenzt. Wenn Sie ein Auftragsdokument benötigen, das größer als 32 kB (Kilobyte), 32.000 B (Byte) ist, erstellen und speichern Sie das Auftragsdokument in Amazon S3 und fügen Sie eine Amazon S3-Objekt-URL in das `documentSource`-Feld für die `CreateJob`-API ein oder verwenden Sie AWS CLI. Fügen Sie bei der AWS-Managementkonsole Erstellung eines Jobs eine Amazon S3-Objekt-URL in das Amazon S3-URL-Textfeld ein.

- AWS-Managementkonsole Jobdokumentation erstellen: Erstellen und verwalten Sie Jobs mithilfe der AWS-Managementkonsole
- AWS CLI Jobdokumentation erstellen: Erstellen und verwalten Sie Jobs mithilfe der AWS CLI
- `CreateJob`API-Dokumentation: CreateJob

Geräteseitige MQTT-Nachrichtenanfragen - Drosselungsgrenzwerte

Wenn Sie den Fehlercode 400 `ThrottlingException` erhalten, ist die geräteseitige MQTT-Nachricht fehlgeschlagen, da das Limit für gleichzeitige geräteseitige Anfragen erreicht wurde. Unter AWS IoT Device Management -Limits und Kontingente für Aufträge finden Sie weitere Informationen zu den Limits für Aufträge und darüber, ob diese einstellbar sind.

Verbindungstimeoutfehler

Ein Fehlercode 400 `RequestExpired` weist auf einen Verbindungsfehler hin, der auf eine hohe Latenz oder niedrige clientseitige Timeout-Werte zurückzuführen ist.

- Informationen zum Testen der Verbindung zwischen der Client- und der Serverseite finden Sie unter Testen der Konnektivität mit Ihrem Gerätedatenendpunkt.

Ungültiger API-Befehl

Vergewissern Sie sich, dass der richtige API-Befehl eingegeben wurde, um zu vermeiden, dass die Fehlermeldung angezeigt wird, dass der API-Befehl ungültig ist. In der AWS IoT API-Referenz finden Sie eine umfassende Liste aller AWS IoT -API-Befehle.

Verbindungsfehler auf der Serviceseite

Ein Fehlercode 503 `ServiceUnavailable` gibt an, dass der Fehler auf der Serverseite auftrat.

- Den aktuellen Status aller AWS Dienste finden Sie unter AWS Health Dashboard (alle AWS Dienste).
- Den aktuellen Status Ihrer persönlichen Daten finden Sie unter AWS Health Dashboard (persönlich AWS-Konto) AWS-Konto.

AWS IoT Befehle zur Problembehandlung

Dies ist der Abschnitt zur Problembehandlung bei AWS IoT Device Management Befehlen.

Probleme bei der Befehlsausführung

Die Befehlsausführung bleibt im Status CREATED

Wenn eine Befehlsausführung weiterhin `CREATED` den Status hat und nicht in einen anderen Status übergeht, sollten Sie Folgendes beachten: `IN_PROGRESS`

- Stellen Sie sicher, dass das Gerät mit AWS IoT Core verbunden ist und das Thema für die Befehlsanforderung abonniert hat.
- Stellen Sie sicher, dass die Geräterichtlinie dies zulässt, `iot:Subscribe` und überprüfen Sie, ob das `iot:Receive` Thema für die Befehlsanforderung und das `iot:Publish` Thema für die Befehlsantwort dies zulässt.
- Wenn das Gerät offline ist und persistente MQTT-Sitzungen verwendet, wartet der Befehl bei AWS IoT Core. Wenn das Gerät vor dem dauerhaften Sitzungs-Timeout und dem Ausführungs-Timeout wieder eine Verbindung herstellt, kann es den Befehl verarbeiten. Wenn das Ausführungs-Timeout abläuft, wechselt die Ausführung zu. `TIMED_OUT`

DataConflict Fehler am UpdateCommandExecution

Ein `DataConflict` Fehler tritt auf, wenn mehrere `UpdateCommandExecution` Anfragen parallel oder innerhalb eines kurzen Zeitrahmens an den Dienst gestellt werden (z. B. unmittelbar `IN_PROGRESS` gefolgt von`SUCCEEDED`).

So beheben Sie dieses Problem

- Abonnieren Sie die Themen `/accepted` und `/rejected` antworten Sie, um zu bestätigen, dass jede Statusaktualisierung verarbeitet wurde, bevor Sie die nächste senden.

- Implementieren Sie eine Wiederholungslogik mit exponentiellem Backoff beim Empfang von Fehlern. `DataConflict`

Befehlsausführungen nehmen unerwartet den Terminalstatus TIMED_OUT an

Wenn eine Befehlsausführung in den Zustand übergeht, `TIMED_OUT` bevor das Gerät sie verarbeiten kann:

- Überprüfen Sie den für die Befehlsausführung konfigurierten Timeout-Wert. Das Standard-Timeout ist möglicherweise zu kurz für Ihren Anwendungsfall.
- Wenn das Gerät offline war, als der Befehl gesendet wurde, stellen Sie sicher, dass das Gerät die Verbindung wieder hergestellt und die Ausführungsanforderung erhalten hat, bevor das Timeout abgelaufen ist.
- Ein Cloud-initiierter Vorgang `TIMED_OUT` ist kein Terminal. Das Gerät kann die Ausführung trotzdem auf einen Terminalstatus (`SUCCEEDED`, `FAILEDREJECTED`, oder`TIMED_OUT`) aktualisieren.

> **Note**
>
> Ein Gerät initiiert `TIMED_OUT` ist ein Terminalstatus, und an dieser Befehlsausführung können keine weiteren Aktualisierungen vorgenommen werden.

Wie kann ich Fehler in CloudWatch Protokollen anzeigen?

Fehler aus der `UpdateCommandExecution` MQTT-Anforderung werden in der `AWSIoTLogsV2` Protokollgruppe in Amazon CloudWatch protokolliert. Informationen zum Aktivieren der Protokollierung und zum Anzeigen der Protokolle finden Sie unterKonfiguration AWS IoT Protokollierung.

UpdateCommandExecution Fehlercodes

Wenn die `UpdateCommandExecution` MQTT-Anforderung fehlschlägt, veröffentlicht der Dienst eine Fehlerantwort auf das `/rejected` Thema. Die Fehlerantwort enthält einen Fehlercode und eine Meldung. In der folgenden Tabelle sind die Fehlercodes aufgeführt, die zurückgegeben werden können.

Fehlercode	Wiederholbar	Description
`InvalidStateTransition`	Nein	Der angeforderte Statusübergang ist nicht zulässig. Zum Beispiel der Übergang von `SUCCEEDED` zu`IN_PROGRESS`, das Aktualisieren einer vom Gerät initiierten Ausführung. `TIMED_OUT`
`TerminalStateReached`	Nein	Die Befehlsausführung hat bereits einen Terminalstatus erreicht und kann nicht aktualisiert werden.
`ResourceNotFound`	Nein	Die angegebene Befehlsausführung ist nicht vorhanden.
`DataConflict`	Ja	Die Befehlsausführung wurde gleichzeitig geändert. Dies kann auftreten, wenn Statusaktualisierungen in schneller Folge gesendet werden. Abonnieren Sie die Themen `/accepted` und beantworten `/rejected` Sie sie, um zu bestätigen, dass jede Statusaktualisierung verarbeitet wurde, bevor Sie die nächste senden. Weitere Informationen finden Sie unter DataConflict Fehler am UpdateCommandExecution.
`InternalError`	Ja	Es ist ein interner Serverfehler aufgetreten. Versuchen Sie die Anfrage mit exponentiellem Backoff erneut.

Das Folgende ist ein Beispiel für eine Fehlerantwort, die zum Thema veröffentlicht wurde: `/rejected`

```
{
    "clientToken": "client-token-1",
    "executionId": "2bd65c51-4cfd-49e4-9310-d5cbfdbc8554",
    "error": "DataConflict",
    "errorMessage": "The command execution was modified concurrently"
```

```
}
```

Fehlerbehebung bei der Flottenindizierung

Fehlerbehebung bei Aggregationsabfragen für den Flottenindizierungsservice

Wenn bei Ihnen Typkonflikte auftreten, können Sie CloudWatch Logs verwenden, um das Problem zu beheben. CloudWatch Logs müssen aktiviert werden, bevor Logs vom Fleet Indexing Service geschrieben werden. Weitere Informationen finden Sie unter Überwachen AWS IoT mithilfe von Protokollen CloudWatch.

Zur Durchführung von Aggregationsabfragen für nicht verwaltete Felder müssen Sie ein Feld angeben, das Sie in dem `customFields`-Argument definiert haben, das an `UpdateIndexingConfiguration` oder update-indexing-configuration übergeben wurde. Wenn der Feldwert nicht mit dem konfigurierten Felddatentyp übereinstimmt, wird dieser Wert beim Ausführen einer Aggregationsabfrage ignoriert.

Wenn ein Feld aufgrund eines nicht übereinstimmenden Typs nicht indexiert werden kann, sendet der Fleet Indexing Service ein Fehlerprotokoll an Logs. CloudWatch Das Fehlerprotokoll enthält den Feldnamen, den Wert, der nicht konvertiert werden konnte, und den Namen des Elements für das Gerät. Nachfolgend sehen Sie ein Beispiel für ein Fehlerprotokoll:

```
{
  "timestamp": "2017-02-20 20:31:22.932",
  "logLevel": "ERROR",
  "traceId": "79738924-1025-3a00-a669-7bec69f7f07a",
  "accountId": "000000000000",
  "status": "SucceededWithIssues",
  "eventType": "IndexingCustomFieldFailed",
  "thingName": "thing0",
  "failedCustomFields": [
    {
      "Name": "attributeName1",
      "Value": "apple",
      "ExpectedType": "String"
    },
    {
      "Name": "attributeName2",
      "Value": "2",
      "ExpectedType": "Boolean"
    }
```

```
  ]
}
```

Wenn ein Gerät etwa eine Stunde lang getrennt war, fehlt möglicherweise der `timestamp`-Wert des Konnektivitätsstatus. Bei persistenten Sitzungen fehlt möglicherweise der Wert, nachdem ein Client für länger als die konfigurierte TTL (Time-to-live) für die persistente Sitzung getrennt war. Die Konnektivitätsstatusdaten werden nur für Verbindungen indiziert, bei denen die Client-ID einen übereinstimmenden Objektnamen hat. (Die Client-ID ist der Wert, mit dem ein Gerät verbunden wird.) AWS IoT Core

Fehlerbehebung bei der Konfiguration der Flottenindizierung

Die Konfiguration der Flottenindizierung kann nicht herabgestuft werden

Das Herabstufen der Konfiguration für die Flottenindizierung wird nicht unterstützt, wenn Sie die Datenquellen entfernen möchten, die mit einer Flottenkennzahl oder einer dynamischen Gruppe verknüpft sind.

Wenn Ihre Indizierungskonfiguration beispielsweise Registrierungsdaten, Schattendaten und Konnektivitätsdaten enthält und die Flottenmetrik existiert mit der Abfrage `thingName:TempSensor* AND shadow.desired.temperature>80`, führt die Aktualisierung der Indizierungskonfiguration, sodass sie nur die Registrierungsdaten enthält, zu einem Fehler.

Das Ändern von benutzerdefinierten Feldern, die von vorhandenen Flottenkennzahlen verwendet werden, wird nicht unterstützt.

Ihre Indexkonfiguration kann aufgrund inkompatibler Flottenkennzahlen oder dynamischer Gruppen nicht aktualisiert werden

Wenn Sie Ihre Indexkonfiguration aufgrund inkompatibler Flottenkennzahlen oder dynamischer Gruppen nicht aktualisieren können, löschen Sie die inkompatiblen Flottenkennzahlen oder dynamischen Gruppen, bevor Sie die Indexkonfiguration aktualisieren.

Problembehebung bei der Standortindizierung und bei Geoabfragen

Zur Behebung von Fehlern, die nicht übereinstimmen, bei der Standortindizierung und bei Geoabfragen können Sie Protokolle aktivieren. CloudWatch Weitere Informationen zur Überwachung der AWS IoT Verwendung finden Sie in der CloudWatch schrittweisen Anleitung.

Wenn Sie Standortdaten mithilfe von Geoabfragen indizieren, müssen die Standortfelder, die Sie in `geoLocations` angeben, mit den Standortfeldern übereinstimmen, die Sie an

`UpdateIndexingConfiguration` übergeben. Wenn es eine Nichtübereinstimmung gibt, sendet die Flottenindizierung einen Fehler beim Typ „Nicht übereinstimmender Typ“ an. CloudWatch Das Fehlerprotokoll enthält den Feldnamen, den Wert, der nicht konvertiert werden konnte, und den Namen des Elements für das Gerät.

Nachfolgend sehen Sie ein Beispiel für ein Fehlerprotokoll:

```
{
"timestamp": "2023-11-09 01:39:43.466",
    "logLevel": "ERROR",
    "traceId": "79738924-1025-3a00-a669-7bec69f7f07a",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "IndexingGeoLocationFieldFailed",
    "thingName": "thing0",
    "failedGeolocationFields": [
        {
"Name": "attributeName1",
            "Value": "apple",
            "ExpectedType": "Geopoint"
        }
    ],
    "reason": "failed to index the field because it could not be converted to one of
 the expected geoLocation formats."
}
```

Weitere Informationen finden Sie unter Indexierung von Standortdaten.

Fehlerbehebung bei Flottenmetriken

Ich sehe keine Datenpunkte in CloudWatch

Wenn Sie in der Lage sind, eine Flottenmetrik zu erstellen, aber keine Datenpunkte darin sehen CloudWatch, ist es wahrscheinlich, dass Sie nichts haben, das die Kriterien der Abfragezeichenfolge erfüllt.

Sehen Sie sich diesen Beispielbefehl an, um eine Flottenmetrik zu erstellen:

```
aws iot create-fleet-metric --metric-name "example_FM" --query-string
 "thingName:TempSensor* AND attributes.temperature>80" --period 60 --aggregation-field
 "attributes.temperature" --aggregation-type name=Statistics,values=count
```

Wenn Sie nichts haben, das die Kriterien für die Abfragezeichenfolge `--query-string "thingName:TempSensor* AND attributes.temperature>80"` erfüllt:

- Mit werden Sie in der Lage sein`values=count`, eine Flottenmetrik zu erstellen, und es gibt Datenpunkte, die angezeigt werden können CloudWatch. Die Datenpunkte des Werts `count` sind immer 0.
- Mit `values` anderem als können Sie eine Flottenkennzahl erstellen`count`, aber die Flottenkennzahl wird nicht angezeigt CloudWatch und es gibt keine Datenpunkte, die angezeigt werden könnten CloudWatch.

AWS IoT Fehlerbehebung im Softwarepaketkatalog für die Geräteverwaltung

Dies ist der Abschnitt zur Problembehandlung für den AWS IoT Geräteverwaltungs-Softwarepaketkatalog.

Allgemeine Fehlermeldungen zur Fehlerbehebung

In diesem Abschnitt werden häufige Fehler aufgeführt, die während des gesamten Lebenszyklus der Softwarepaketversion auftreten.

`HeadBucket`-Fehler

Die folgenden Fehlermeldungen werden angezeigt, wenn der `HeadBucket` API-Vorgang oder der `head-bucket` CLI-Befehl aufgerufen wird, um den Amazon S3-Bucket zu überprüfen, der für den Dateiupload während einer Auftragsbereitstellung verwendet wird.

Weitere Informationen zur Verwendung eines Amazon S3-Buckets zum Hochladen von Dateien während einer Auftragsbereitstellung finden Sie unterVorsignierte URL für den Datei-Upload.

`InvalidRoleException`

```
    "Permission denied when attempting to use role %s to access bucket %s."
```

`InvalidRequestException`

```
    "Cross region S3 bucket is not supported for presigned url upload placeholder"
```

`InvalidRequestException`

```
    "S3 bucket in job document presigned url upload placeholder not found"
```

```
InvalidRequestException
    "Given S3 bucket name is invalid."
```

```
InvalidRequestException
    "Provided S3 bucket is not valid: %s. Error: %s"
```

Amazon S3 GetObject

Die folgende Fehlermeldung tritt auf, wenn ein ungültiges Argument angegeben wird, wodurch der Amazon `GetObject` S3-API-Vorgang fehlschlägt.

```
InvalidRequestException
    "Provided argument for presigned url is invalid"
```

Unterstützung der Amazon S3-Versions-ID

Wenn Sie mithilfe der Versionskontrolle den Zugriff auf einen Amazon S3-Bucket anfordern, stellen Sie sicher, dass Sie Ihren `versionId` oder die folgende Fehlermeldung angeben.

Weitere Informationen zu Amazon S3-Buckets, die die Versionskontrolle verwenden, finden Sie unter Verwenden der Versionierung in Amazon S3-Buckets

```
InvalidRequestException
    "VersionId not found when attempting to access s3 url"
```

Platzhalter innerhalb einer vorsignierten URL für den Datei-Upload

Die folgenden Fehlermeldungen werden angezeigt, wenn Probleme mit einem Platzhalter in einer vorsignierten URL auftreten, die während einer Auftragsbereitstellung zum Hochladen von Dateien in einen Amazon S3-Ziel-Bucket verwendet wird. Weitere Informationen zur Verwendung eines Amazon S3-Buckets zum Hochladen von Dateien während einer Auftragsbereitstellung und zur Definition eines lokalen Platzhalters finden Sie unter. Vorsignierte URL für den Datei-Upload

Die folgende Fehlermeldung wird angezeigt, wenn der lokale Platzhalter nicht erkannt wird.

```
InvalidJobDocumentException
```

```
    "Undefined placeholder, ${...}, inside of presign url upload parameter"
```

Die folgende Fehlermeldung wird angezeigt, wenn versucht wird, den lokalen Platzhalter in einer vorsignierten URL zu verwenden, die nicht für den Dateiupload bestimmt ist.

```
InvalidJobDocumentException
    "Local placeholder, ${...}, is only valid inside of presign url upload"
```

Amazon S3-URL ist falsch verschachtelt

Die folgende Fehlermeldung wird angezeigt, wenn die Amazon S3-URL fälschlicherweise in einem anderen Platzhalter verschachtelt ist.

```
InvalidJobDocumentException
    "${aws:%s[...]} should not be the second layer pattern."
```

Paketversion: Artefakt-Verschachtelung

Die folgende Fehlermeldung wird angezeigt, wenn die vorsignierte URL für das Artefakt der Paketversion fälschlicherweise in einem anderen Platzhalter verschachtelt ist.

```
InvalidJobDocumentException
    "${aws:iot:package:[...]:artifact:s3-presigned-url} cannot be nested inside another
 placeholder."
```

Artefakt der Paketversion fehlt

Die folgende Fehlermeldung wird angezeigt, wenn das Artefakt der Paketversion, auf das verwiesen wird, nicht gefunden wird.

```
InvalidJobDocumentException
    "Package %s version %s does not have an associated artifact to generate an S3
 presigned url."
```

Platzhalter für Softwarepaket und Paketversion

Die folgende Fehlermeldung wird angezeigt, wenn der Platzhalter für das Jobdokument für das Softwarepaket und die Paketversion aufgrund mehrerer Softwarepakete und Paketversionen, auf

die im `destinationPackageVersions` Parameter oder auf der Registerkarte Versions-ARN auf der Detailseite der Paketversion verwiesen wird, nicht zu den gewünschten gültigen Werten für die Auftragsbereitstellung aufgelöst werden kann.

```
InvalidJobDocumentException
    "Cannot resolve empty package name and version name given multiple elements in
 destination package versions."
```

Verwenden Sie ein leeres Softwarepaket und eine leere Paketversion

Die folgende Fehlermeldung wird angezeigt, wenn Sie versuchen, ein leeres Paket oder eine Paketversion ohne die andere in einem Jobdokument zu verwenden.

```
InvalidJobDocumentException
    "Empty package name and version name have to be used in pair."
```

Null: Verwendung im Auftragsdokument

Die folgende Fehlermeldung wird angezeigt, wenn Sie versuchen`$null`, eine Paketversion im Jobdokument anzugeben. `$null`kann nur innerhalb des `destinationPackageVersions` Parameters verwendet werden, wenn die `CreateJob` API-Operation verwendet wird.

```
InvalidJobDocumentException
    "$null is not allowed to be referenced as a package version in job documents."
```

Alle Attribute in einer Paketversion

Die folgende Fehlermeldung wird angezeigt, wenn Sie versuchen, alle Attribute in einer Paketversion zu verwenden und sie mit zusätzlichem Text oder Platzhaltern zu umgeben.

Weitere Informationen zur Verwendung aller Attribute in einer Softwarepaketversion finden Sie unter Substitutionsparameter für AWS IoT jobs

```
InvalidJobDocumentException
    "The package version attribute placeholder for all attributes has to be a
 json value by itself and not appended with other strings or nested with other
 placeholders."
```

Limit für lokale Platzhalter in vorsignierten URLs für den Datei-Upload

Die folgende Fehlermeldung wird angezeigt, wenn Sie das Limit für die Anzahl der lokalen Platzhalter überschreiten, die in einer vorsignierten URL für den Dateiupload während einer Jobbereitstellung verwendet werden.

Weitere Informationen zur Verwendung einer vorsignierten URL für den Dateiupload während einer Auftragsbereitstellung finden Sie unter Vorsignierte URL für den Datei-Upload

```
InvalidJobDocumentException
    "The occurrence of local placeholder %s within S3 presigned url upload placeholder
 exceeds limit of %d."
```

Lokale Platzhalter in einem Amazon S3-Bucket

Die folgende Fehlermeldung wird angezeigt, wenn Sie versuchen, im Amazon S3-Bucket-Namen eine lokale Platzhalter-URL für einen vorsignierten URL-Platzhalter zu platzieren, der für den Datei-Upload während einer Auftragsbereitstellung verwendet wird.

Weitere Informationen zur Verwendung einer vorsignierten URL für den Dateiupload während einer Auftragsbereitstellung finden Sie unter Vorsignierte URL für den Datei-Upload

```
InvalidJobDocumentException
    "S3 bucket name in presigned url upload is not allowed to contain any placeholders"
```

Klammern öffnen und schließen

Die folgende Fehlermeldung wird angezeigt, wenn Sie einem Jobdokument ohne die schließende Klammer „}“ einen Parameter oder Platzhalter hinzufügen.

```
InvalidJobDocumentException
    "One or more parameters or placeholders are not terminated."
```

IAM-Rolle mit vorsignierter Amazon S3-URL

Die folgende Fehlermeldung wird angezeigt, wenn Sie versuchen, eine vorsignierte Amazon S3-URL in einem Jobdokument ohne IAM-Rolle zu verwenden.

Weitere Informationen zu vorsignierten Amazon S3-URLs finden Sie unter Arbeiten mit vorsignierten URLs.

```
InvalidRequestException
```

```
    "presignedUrlConfig role ARN is required to generate an S3 presigned url in job
 document."
```

IAM-Rolle mit vorsignierter Amazon S3-URL für das Artefakt der Paketversion

Die folgende Fehlermeldung wird angezeigt, wenn Sie versuchen, eine vorsignierte Amazon S3-URL zu verwenden, die ein Artefakt der Paketversion in einem Jobdokument ohne IAM-Rolle darstellt.

```
InvalidRequestException
    "presignedUrlConfig role ARN is required to generate an S3 presigned url in job
 document for package %s version %s artifact."
```

Fehlermeldungen zur Stückliste der Software

In diesem Abschnitt werden häufig auftretende Fehler im Zusammenhang mit einer Softwareliste (SBOM) aufgeführt, die mit einer Paketversion verknüpft ist.

Eingabeüberprüfung für die Anfrage zur SBOM Association

Die folgende Fehlermeldung wird angezeigt, wenn der `AssociateSbomWithPackageVersion` API-Vorgang verwendet wird und der `s3Location` Parameter Null ist.

```
InvalidRequestException "Associate request needs to include SBOM reference"
```

Weitere Informationen zum `AssociateSbomWithPackageVersion` API-Vorgang finden Sie unter AssociateSbomWithPackageVersion.

Fehler bei der SBOM-Validierung

In diesem Abschnitt werden häufig auftretende Fehler aufgeführt, die bei der ersten Validierung der Softwareliste (SBOM) im Zusammenhang mit einer Softwarepaketversion auftreten.

Die folgende Fehlermeldung wird angezeigt, wenn der `AssociateSbomWithPackageVersion` API-Vorgang verwendet wird und der Parameter `bucket` Null ists`3Location`.

```
InvalidRequestException "S3 bucket name for SBOM cannot be null"
```

Die folgende Fehlermeldung wird angezeigt, wenn die Zeichenfolge `bucket` im `s3Location` Parameter für den `AssociateSbomWithPackageVersion` API-Vorgang zu lang ist.

```
InvalidRequestException "S3 bucket name for SBOM is illegal. String length exceeds
 limit"
```

Die folgende Fehlermeldung wird angezeigt, wenn der `key` Parameter Null ist.

```
InvalidRequestException "S3 key name for SBOM cannot be null"
```

Die folgende Fehlermeldung wird angezeigt, wenn die Zeichenfolge `key` im `s3Location` Parameter für den `AssociateSbomWithPackageVersion` API-Vorgang zu lang ist.

```
InvalidRequestException "S3 key name for SBOM is illegal. String length exceeds limit"
```

Die folgende Fehlermeldung wird angezeigt, wenn die Zeichenfolge `version` im `s3Location` Parameter für den `AssociateSbomWithPackageVersion` API-Vorgang Null ist.

```
InvalidRequestException "S3 object version for SBOM cannot be null"
```

Die folgende Fehlermeldung wird angezeigt, wenn die Zeichenfolge `version` im `s3Location` Parameter für den `AssociateSbomWithPackageVersion` API-Vorgang zu lang ist.

```
InvalidRequestException "S3 object version for SBOM is illegal. String length exceeds
 limit"
```

Die folgende Fehlermeldung wird angezeigt, wenn die Größe der im Amazon S3-Bucket gespeicherten SBOM-ZIP-Archivdatei zu groß ist.

```
InvalidRequestException "S3 object file size exceeds limit"
```

Die folgende Fehlermeldung wird angezeigt, wenn Sie den `AssociateSbomWithPackageVersion` API-Vorgang verwenden und die aktuelle Anzahl der laufenden SBOM-Validierungen bereits das Höchstlimit erreicht hat.

```
LimitExceededException "Too many ongoing SBOM validation workflows. Please wait and
 retry"
```

Probleme beim Zugriff auf die SBOM-Datei im Amazon S3-Bucket

Die folgende Fehlermeldung wird angezeigt, wenn eine andere Entität nicht auf den Amazon S3-Bucket zugreifen kann, weil der Amazon S3-Bucket nicht existiert oder die richtigen Berechtigungen für den Zugriff auf den Amazon S3-Bucket nicht erteilt wurden.

Weitere Informationen zur Richtlinie über die erforderlichen Berechtigungen für den Zugriff auf den Amazon S3-Bucket finden Sie unterAufbewahrung der Stückliste der Software.

```
InvalidRequestException "SBOM not accessible by the service. Please make sure the
 bucket exists and S3 permission is granted."
```

Die folgende Fehlermeldung wird angezeigt, wenn eine andere Entität nicht auf die SBOM-ZIP-Archivdatei im `key` Parameter zugreift, weil der Amazon S3-Bucket nicht existiert oder nicht die richtigen Berechtigungen für den Zugriff auf im Amazon S3-Bucket gespeicherte Inhalte erteilt wurden.

```
InvalidRequestException "SBOM not accessible by the service. Please make sure the key
 exists and S3 permission is granted."
```

Die folgende Fehlermeldung wird angezeigt, wenn eine andere Entität nicht auf den Amazon S3-Bucket zugreifen kann, weil der Bucket, der Schlüssel und die Versions-ID nicht existieren oder die richtigen Berechtigungen für den Zugriff auf den Amazon S3-Bucket nicht erteilt wurden. Darüber hinaus kann diese Fehlermeldung angezeigt werden, wenn die erteilten Berechtigungen für den Zugriff auf die SBOM-Zip-Archivdatei im Amazon S3-Bucket nicht ausreichen.

```
InvalidRequestException "SBOM not accessible by the service. Please make sure the
 bucket/key/version exists and S3 permission is granted."
```

Die folgende Fehlermeldung wird angezeigt, wenn eine andere Entität nicht auf den Amazon S3-Bucket zugreifen kann, weil sich der Bucket in einer anderen Region befindet.

```
InvalidRequestException "Cross-region S3 bucket for %s is not supported."
```

Die folgende Fehlermeldung wird angezeigt, wenn eine andere Entität nicht auf den Amazon S3-Bucket zugreifen kann`bucket`, weil die`key`, oder `version` Parameter bei der Verwendung des `AssociateSbomWithPackageVersion` API-Vorgangs falsch geschrieben wurden.

```
InvalidRequestException "Please make sure SBOM S3 bucket name/key length/version is
 valid"
```

AWS IoT Anleitung zur Fehlerbehebung bei Device Advisor

Helfen Sie uns, dieses Thema zu verbessern

Lassen Sie uns wissen, was dazu beitragen würde, es besser zu machen

General

F: Kann ich mehrere Testsuiten parallel ausführen?

A: Ja. Device Advisor unterstützt jetzt die Ausführung mehrerer Testsuiten auf verschiedenen Geräten mithilfe eines Device-level Endpunkts. Wenn Sie den Account-level Endpunkt verwenden, können Sie jeweils eine Suite ausführen, da pro Konto ein Device Advisor-Endpunkt verfügbar ist. Weitere Informationen finden Sie unter Konfigurieren Ihres Geräts.

F: Ich habe von meinem Gerät aus gesehen, dass die TLS-Verbindung von Device Advisor verweigert wurde. Ist das normal?

A: Ja. Device Advisor verweigert die TLS-Verbindung vor und nach jedem Testlauf. Wir empfehlen Benutzern, einen Mechanismus zur Gerätewiederholung zu implementieren, um ein vollautomatisches Testerlebnis mit Device Advisor zu gewährleisten. Wenn Sie eine Testsuite mit mehr als einem Testfall ausführen, zum Beispiel TLS Connect, MQTT Connect und MQTT Publish, dann empfehlen wir, dass Sie einen Mechanismus für Ihr Gerät entwickeln lassen. Der Mechanismus kann versuchen, alle 5 Sekunden für ein bis zwei Minuten eine Verbindung zu unserem Testendpunkt herzustellen. Auf diese Weise können Sie mehrere Testfälle nacheinander automatisiert ausführen.

Kann ich für Sicherheitsanalysen und zu Fehlerbehebungszwecken einen Verlauf der API-Aufrufe abrufen, die von der Device Advisor API in meinem Konto gesendet wurden?

A: Ja. Um eine Historie der Device Advisor-API-Aufrufe zu erhalten, die für Ihr Konto getätigt wurden, schalten Sie das Gerät einfach CloudTrail in der AWS IoT Management Console ein und filtern Sie die Eventquelle danach`iotdeviceadvisor.amazonaws.com`.

F: Wie sehe ich mir die Device Advisor-Logins an CloudWatch?

A: Protokolle, die während einer Testsuite-Ausführung generiert wurden, werden hochgeladen, CloudWatch wenn Sie die erforderliche Richtlinie (z. B. CloudWatchFullAccess) zu Ihrer Servicerolle hinzufügen (siehe Einrichtung). Wenn die Testsuite mindestens einen Testfall

enthält, wird eine Protokollgruppe "aws/iot/deviceadvisor/ $testSuiteId" mit zwei Logstreams erstellt. Ein Stream ist der „$testRunId“ und enthält Protokolle der Aktionen, die vor und nach der Ausführung der Testfälle in Ihrer Testsuite ergriffen wurden, z. B. die Schritte zur Einrichtung und Bereinigung. Der andere Log-Stream ist „$suite RunId _$test“RunId, der spezifisch für eine Testsuite-Ausführung ist. Ereignisse, die von Geräten gesendet und in diesem Log-Stream protokolliert AWS IoT Core werden.

F: Was ist der Zweck der Geräteberechtigungsrolle?

A: Device Advisor steht zwischen Ihrem Testgerät und AWS IoT Core zur Simulation von Testszenarien. Er akzeptiert Verbindungen und Nachrichten von Ihren Testgeräten und leitet sie weiter an AWS IoT Core , indem er Ihre Geräteberechtigungsrolle übernimmt und in Ihrem Namen eine Verbindung initiiert. Es ist wichtig, sicherzustellen, dass die Geräterollenberechtigungen mit denen auf dem Zertifikat übereinstimmen, das Sie für die Ausführung von Tests verwenden. AWS IoT Zertifikatsrichtlinien werden nicht durchgesetzt, wenn Device Advisor in AWS IoT Core Ihrem Namen mithilfe der Geräteberechtigungsrolle eine Verbindung herstellt. Die Berechtigungen der von Ihnen festgelegten Geräteberechtigungsrolle werden jedoch durchgesetzt.

F: In welchen Regionen wird Device Advisor unterstützt?

A: Device Advisor wird in den Regionen us-east-1, us-west-2, ap-northeast-1 und eu-west-1 unterstützt.

F: Warum sehe ich inkonsistente Ergebnisse?

A: Eine der Hauptursachen für inkonsistente Ergebnisse ist die Einstellung von `EXECUTION_TIMEOUT` eines Tests auf einen zu niedrigen Wert. Weitere Informationen zu empfohlenen und standardmäßigen `EXECUTION_TIMEOUT`-Werten finden Sie unter Device Advisor-Testfälle.

F: Welches MQTT-Protokoll unterstützt Device Advisor?

A: Device Advisor unterstützt MQTT Version 3.1.1 mit X509-Client-Zertifikaten.

F: Was ist, wenn mein Testfall mit einer Meldung über das Ausführungszeitlimit fehlschlägt, obwohl ich versucht habe, mein Gerät mit dem Testendpunkt zu verbinden?

A: Überprüfen Sie alle Schritte unter Erstellen Sie eine IAM-Rolle, die als Ihre Geräterolle verwendet werden soll. Wenn der Test immer noch fehlschlägt, sendet das Gerät möglicherweise nicht die richtige SNI-Erweiterung (Server Name Indication), die erforderlich ist, damit Device Advisor funktioniert. Der richtige SNI-Wert ist die Endpunktadresse, die zurückgegeben wird, wenn Sie den Anweisungen im Abschnitt Gerät konfigurieren folgen. AWS IoT erfordert

außerdem, dass Geräte die SNI-Erweiterung (Server Name Indication) an das Transport Layer Security (TLS) -Protokoll senden. Weitere Informationen finden Sie unter Transportsicherheit unter AWS IoT.

F: Meine MQTT-Verbindung schlägt mit einem „libaws-c-mqtt: AWS_ERROR_MQTT_UNEXPECTED_HANGUP“ -Fehler fehl (oder) die MQTT-Verbindung meines Geräts wird automatisch vom Device Advisor-Endpunkt getrennt. Wie kann dieser Fehler behoben werden?

A: Dieser spezielle Fehlercode und unerwartete Verbindungsabbrüche können viele verschiedene Ursachen haben, hängen aber höchstwahrscheinlich mit der dem Gerät zugewiesenen Geräterolle zusammen. Die folgenden Prüfpunkte (in der Reihenfolge ihrer Priorität) lösen dieses Problem.

- Die dem Gerät zugeordnete Geräterolle muss über die IAM-Mindestrechte verfügen, die für die Ausführung der Tests erforderlich sind. Device Advisor verwendet die angefügte Geräterolle, um AWS IoT -MQTT-Aktionen im Namen des Testgeräts durchzuführen. Wenn die erforderlichen Berechtigungen nicht vorhanden sind, wird der Fehler `AWS_ERROR_MQTT_UNEXPECTED_HANGUP` angezeigt oder es kommt zu unerwarteten Verbindungsabbrüchen, während das Gerät versucht, eine Verbindung zum Device Advisor-Endpunkt herzustellen. Wenn Sie beispielsweise ausgewählt haben, den MQTT Publish-Testfall auszuführen, müssen sowohl die Connect- als auch die Publish-Aktionen in der Rolle mit dem entsprechenden Thema ClientId und enthalten sein (Sie können mehrere Werte angeben, indem Sie die Werte durch Kommas trennen, und Sie können Präfixwerte mit einem Platzhalterzeichen (*) angeben. Beispiel: Um Berechtigungen zur Veröffentlichung für ein beliebiges Thema zu erteilen, das mit `TestTopic` beginnt, können Sie „`TestTopic*`“ als Ressourcenwert angeben. Hier sind einige Beispiele für Richtlinien.
- Nichtübereinstimmung zwischen den in der Geräterolle für Ihre Ressourcentypen definierten Werten und den tatsächlich im Code verwendeten Werten. Beispiel: Eine Diskrepanz zwischen dem in der Rolle ClientId definierten und dem tatsächlich ClientId verwendeten Code in Ihrem Gerätecode. Werte wie ClientId, Topic und TopicFilter müssen in der Geräterolle und im Code identisch sein.
- Das an Ihr Gerät angehängte Gerätezertifikat muss aktiv sein und mit einer Richtlinie versehen sein, die die erforderlichen Aktionsberechtigungen für Ressourcen enthält. Beachten Sie, dass die Gerätezertifikatsrichtlinie den Zugriff auf AWS IoT Ressourcen und Operationen auf AWS IoT Core Datenebene gewährt oder verweigert. Device Advisor setzt voraus, dass Sie ein aktives Gerätezertifikat an Ihr Gerät angeschlossen haben, das die während eines Testfalls verwendeten Aktionsberechtigungen gewährt.

AWS IoT Fehler

Helfen Sie uns, dieses Thema zu verbessern

Lassen Sie uns wissen, was dazu beitragen würde, es besser zu machen

In diesem Abschnitt sind die Fehlercodes aufgeführt, die von AWS IoT gesendet wurden.

Fehlercodes für Message Broker

Fehlercode	Fehlerbeschreibung
400	Inkorrekte Anfrage
401	Nicht autorisiert
403	Verboten.
426	Upgrade erforderlich.
503	Service nicht verfügbar

Identitäts- und Sicherheitsfehlercodes

Fehlercode	Fehlerbeschreibung
401	Nicht autorisiert

Geräteschatten-Fehlercodes

Fehlercode	Fehlerbeschreibung
400	Inkorrekte Anfrage
401	Nicht autorisiert
403	Verboten.
404	Nicht gefunden

Fehlercode	Fehlerbeschreibung
409	Konflikt
413	Anfrage zu lang
422	Anfrage konnte nicht verarbeitet werden
429	Zu viele Anfragen
500	Interner Fehler
503	Service nicht verfügbar

AWS IoT Geräte-SDKs, SDKs für Mobilgeräte und AWS IoT Geräte-Client

Auf dieser Seite sind die AWS IoT Geräte-SDKs, Open-Source-Bibliotheken, Entwicklerhandbücher, Beispiel-Apps und Portierungsanleitungen zusammengefasst, die Ihnen helfen, innovative IoT-Lösungen mit AWS IoT Hardwareplattformen Ihrer Wahl zu entwickeln.

Diese SDKs sind für die Verwendung auf Ihrem IoT-Gerät vorgesehen. Wenn Sie eine IoT-App für die Verwendung auf einem Mobilgerät entwickeln, finden Sie weitere Informationen unter AWS SDKs für Mobilgeräte. Wenn Sie eine IoT-App oder ein serverseitiges Programm entwickeln, finden Sie weitere Informationen unter AWS SDKs.

AWS IoT Geräte-SDKs

Die AWS IoT Geräte-SDKs enthalten Open-Source-Bibliotheken, Entwicklerhandbücher mit Beispielen und Portierungsanleitungen, sodass Sie innovative IoT-Produkte oder -Lösungen auf den Hardwareplattformen Ihrer Wahl entwickeln können.

> **Note**
>
> Die AWS IoT Device SDKs haben einen MQTT 5-Client veröffentlicht. Die AWS IoT Geräte-SDKs unterstützen die Verwendung von TLS 1.3 unter macOS nicht.

Diese SDKs helfen Ihnen dabei, Ihre IoT-Geräte mit AWS IoT mithilfe er Protokolle MQTT und WSS zu verbinden.

C++

AWS IoT C++-Geräte-SDK

Das AWS IoT C++ Device SDK ermöglicht es Entwicklern, verbundene Anwendungen mithilfe AWS der AWS IoT APIs zu erstellen. Dieses SDK wurde speziell für Geräte entwickelt, die nicht ressourcenbeschränkt sind und die erweiterte Funktionen benötigen, wie beispielsweise Nachrichtenwarteschlangen, Multithreading-Support und die aktuellen Sprachfunktionen. Weitere Informationen finden Sie hier:

- AWS IoT Geräte-SDK C++ v2 ein GitHub

- AWS IoT Readme-Datei zum Geräte-SDK C++ v2
- AWS IoT Beispiele für Geräte-SDK C++ v2
- AWS IoT API-Dokumentation zum Geräte-SDK C++ v2

Python

AWS IoT Geräte-SDK für Python

Das AWS IoT Geräte-SDK für Python ermöglicht es Entwicklern, Python-Skripte zu schreiben, um mit ihren Geräten über MQTT oder MQTT über das Protokoll auf die AWS IoT WebSocket Plattform zuzugreifen. Indem sie ihre Geräte mit verbinden AWS IoT, können Benutzer sicher mit dem Message Broker, den Regeln und Shadows arbeiten, die von AWS IoT und mit anderen AWS Diensten wie AWS Lambda Kinesis und Amazon S3 und mehr bereitgestellt werden.

- AWS IoT Geräte-SDK für Python v2 an GitHub
- AWS IoT Readme-Datei zum Geräte-SDK für Python v2
- AWS IoT Beispiele für das Geräte-SDK für Python v2
- AWS IoT API-Dokumentation zum Geräte-SDK für Python v2

JavaScript

AWS IoT Geräte-SDK für JavaScript

Das Paket aws-iot-device-sdk.js ermöglicht es Entwicklern, JavaScript Anwendungen zu schreiben, die AWS IoT mithilfe von MQTT oder MQTT über das WebSocket Protokoll zugreifen. Es kann in Node.js Umgebungen und Browseranwendungen verwendet werden. Weitere Informationen finden Sie hier:

- AWS IoT Geräte-SDK für JavaScript Version 2 an GitHub
- AWS IoT Readme-Datei zum Geräte-SDK JavaScript für Version 2
- AWS IoT Geräte-SDK für JavaScript v2-Beispiele
- AWS IoT Geräte-SDK für die JavaScript v2-API-Dokumentation

Java

AWS IoT Geräte-SDK für Java

Das AWS IoT Geräte-SDK für Java ermöglicht es Java-Entwicklern, über MQTT oder MQTT über das Protokoll auf die AWS IoT WebSocket Plattform zuzugreifen. Das SDK wird mit Support für Schattengeräte angelegt. Sie können über die HTTP-Methoden GET, UPDATE und DELETE auf Schattengeräte zugreifen. Das SDK unterstützt auch ein vereinfachtes Zugangsmodell für Schattengeräte, sodass Entwickler mithilfe der Methoden „Getter” und „Setter” Daten mit den Schattengeräten austauschen können, ohne JSON-Dokumente serialisieren oder deserialisieren zu müssen.

> **Note**
>
> Das AWS IoT Device SDK für Java v2 unterstützt jetzt die Android-Entwicklung. Weitere Informationen finden Sie unter AWS IoT Geräte-SDK für Android.

Weitere Informationen finden Sie hier:

- AWS IoT Geräte-SDK für Java v2 auf GitHub
- AWS IoT Readme-Datei zum Geräte-SDK für Java v2
- AWS IoT Beispiele für Geräte-SDK für Java v2
- AWS IoT API-Dokumentation zum Geräte-SDK für Java v2

Swift

AWS IoT Geräte-SDK für Swift

Das AWS IoT Geräte-SDK für Swift ermöglicht es Swift-Entwicklern, AWS IoT Anwendungen für Linux- und Apple macOS-, iOS- und tvOS-Plattformen mithilfe des MQTT 5-Protokolls zu erstellen.

Weitere Informationen finden Sie hier:

- AWS IoT Geräte-SDK für Swift an GitHub
- AWS IoT Geräte-SDK für Swift Readme
- AWS IoT Geräte-SDK für Swift-Beispiele

AWS IoT Geräte-SDK für Embedded C

> **Note**
>
> Dieses SDK ist für die Verwendung durch erfahrene Entwickler eingebetteter Software vorgesehen.

AWS IoT Device SDK for Embedded C (C-SDK) ist eine Sammlung von C-Quelldateien unter der MIT-Open-Source-Lizenz, die in eingebetteten Anwendungen verwendet werden können, um IoT-Geräte sicher mit ihnen zu verbinden AWS IoT Core. Es umfasst einen MQTT-Client, JSON-Parser und AWS IoT Device Shadow, AWS IoT Jobs, AWS IoT Fleet Provisioning und Bibliotheken. AWS IoT Device Defender Dieses SDK wird im Quellformat verteilt und kann zusammen mit dem Anwendungscode, weiteren Bibliotheken und einem Betriebssystem (BS) Ihrer Wahl in die Kunden-Firmware integriert werden.

Das AWS IoT Device SDK for Embedded C richtet sich in der Regel an Geräte mit eingeschränkten Ressourcen, die eine optimierte C-Sprachlaufzeit benötigen. Sie können das SDK auf jedem Betriebssystem verwenden und es auf jedem Prozessortyp hosten (z. B. MCUs und MPUs).

Weitere Informationen finden Sie hier:

- AWS IoT Geräte-SDK für Embedded C an GitHub
- AWS IoT Readme-Datei zum Geräte-SDK für Embedded C
- AWS IoT Geräte-SDK für Embedded C-Beispiele

AWS SDKs für Mobilgeräte

Die AWS mobilen SDKs bieten Entwicklern mobiler Apps plattformspezifische Unterstützung für die APIs der AWS IoT Core Dienste, die IoT-Gerätekommunikation mithilfe von MQTT und die APIs anderer Dienste. AWS

Android

AWS Mobile SDK für Android

Das AWS Mobile SDK für Android enthält eine Bibliothek, Beispiele und Dokumentation, mit der Entwickler verbundene mobile Anwendungen erstellen können. AWS Dieses SDK unterstützt auch

die Kommunikation mit MQTT-Geräten und das Aufrufen der APIs der AWS IoT Core Dienste. Weitere Informationen finden Sie hier:

- AWS Mobile SDK für Android auf GitHub
- AWS Mobile SDK für Android Readme
- AWS Mobile SDK für Android Beispiele
- AWS Mobile SDK für Android API-Referenz
- AWSIoTClient Dokumentation zur Klassenreferenz

iOS

AWS Mobile SDK for iOS

Das AWS Mobile SDK for iOS ist ein Open-Source-Software-Entwicklungskit, das unter einer Apache Open Source-Lizenz vertrieben wird. Das AWS Mobile SDK for iOS bietet eine Bibliothek, Codebeispiele und Dokumentation, mit denen Entwickler verbundene mobile Anwendungen erstellen AWS können. Dieses SDK beinhaltet auch Unterstützung für die MQTT-Gerätekommunikation und das Aufrufen der APIs der AWS IoT Core -Services. Weitere Informationen finden Sie hier:

- AWS Mobile SDK for iOS auf GitHub
- AWS Mobile SDK for iOS Readme
- AWS Mobile SDK for iOS Beispiele
- Referenzdokumente zur AWSIot-Klasse im AWS Mobile SDK for iOS

AWS IoT Geräte-Client

Der AWS IoT Geräteclient stellt Code bereit, der Ihrem Gerät hilft, eine Verbindung herzustellen AWS IoT, Flottenbereitstellungsaufgaben auszuführen, Gerätesicherheitsrichtlinien zu unterstützen, eine Verbindung über sicheres Tunneling herzustellen und Aufträge auf Ihrem Gerät zu verarbeiten. Sie können diese Software auf Ihrem Gerät installieren, um diese routinemäßigen Geräteaufgaben zu erledigen, sodass Sie sich auf Ihre spezifische Lösung konzentrieren können.

> **Note**
>
> Der AWS IoT Device Client funktioniert mit mikroprozessorbasierten IoT-Geräten mit x86_64- oder ARM-Prozessoren und gängigen Linux-Betriebssystemen.

C++

AWS IoT Geräte-Client

Weitere Informationen zum AWS IoT Device Client in C++ finden Sie unter den folgenden Links:

- AWS IoT Der Geräteclient im C++-Quellcode ist aktiviert GitHub
- AWS IoT Readme-Datei zum Geräteclient in C++

Früher AWS IoT Versionen der Geräte-SDKs

Dies sind frühere Versionen von AWS IoT Geräte-SDKs, die durch die oben aufgeführten neueren Versionen ersetzt wurden. Diese SDKs erhalten nur Wartungs- und Sicherheitsupdates. Sie werden nicht aktualisiert, um neue Funktionen aufzunehmen, und sollten nicht für neue Projekte verwendet werden.

- AWS IoT C++-Geräte-SDK aktiviert GitHub
- AWS IoT Readme-Datei zum C++ Geräte-SDK
- AWS IoT Geräte-SDK für Python v1 an GitHub
- AWS IoT Readme-Datei zum Geräte-SDK für Python v1
- AWS IoT Geräte-SDK für Java an GitHub
- AWS IoT Readme-Datei zum Geräte-SDK für Java
- AWS IoT Geräte-SDK für ein JavaScript GitHub
- AWS IoT Geräte-SDK für JavaScript Readme

Codebeispiele für AWS IoT verwenden AWS SDKs

Die folgenden Codebeispiele zeigen die Verwendung AWS IoT mit einem AWS Software Development Kit (SDK).

Bei Grundlagen handelt es sich um Codebeispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden AWS IoT mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Codebeispiele

- Grundlegende Beispiele für AWS IoT verwenden AWS SDKs
  - Hallo AWS IoT
  - Lernen Sie die Grundlagen von AWS IoT mit einem AWS SDK
  - Aktionen für AWS IoT verwenden AWS SDKs
    - Verwenden Sie mit AttachThingPrincipal einem AWS SDK oder CLI
    - Verwenden Sie CreateKeysAndCertificate mit einem AWS SDK oder CLI
    - Verwenden Sie CreateThing mit einem AWS SDK oder CLI
    - Verwenden Sie CreateTopicRule mit einem AWS SDK oder CLI
    - Verwenden Sie DeleteCertificate mit einem AWS SDK oder CLI
    - Verwenden Sie DeleteThing mit einem AWS SDK oder CLI
    - Verwenden Sie DeleteTopicRule mit einem AWS SDK oder CLI
    - Verwenden Sie DescribeEndpoint mit einem AWS SDK oder CLI
    - Verwenden Sie DescribeThing mit einem AWS SDK oder CLI
    - Verwenden Sie DetachThingPrincipal mit einem AWS SDK oder CLI
    - Verwenden Sie ListCertificates mit einem AWS SDK oder CLI

    - Verwenden Sie ListThings mit einem AWS SDK oder CLI
    - Verwenden Sie SearchIndex mit einem AWS SDK oder CLI
    - Verwenden Sie UpdateIndexingConfiguration mit einem AWS SDK oder CLI
    - Verwenden Sie UpdateThing mit einem AWS SDK oder CLI
- Szenarien für AWS IoT verwenden AWS SDKs
    - Erste Schritte mit dem Geräteschutz im Internet der Dinge
    - Erste Schritte mit Nachrichten im Internet der Dinge

Grundlegende Beispiele für AWS IoT verwenden AWS SDKs

Die folgenden Codebeispiele zeigen, wie Sie die Grundlagen von AWS IoT mit AWS SDKs verwenden.

Beispiele

- Hallo AWS IoT
- Lernen Sie die Grundlagen von AWS IoT mit einem AWS SDK
- Aktionen für AWS IoT verwenden AWS SDKs
    - Verwenden Sie mit AttachThingPrincipal einem AWS SDK oder CLI
    - Verwenden Sie CreateKeysAndCertificate mit einem AWS SDK oder CLI
    - Verwenden Sie CreateThing mit einem AWS SDK oder CLI
    - Verwenden Sie CreateTopicRule mit einem AWS SDK oder CLI
    - Verwenden Sie DeleteCertificate mit einem AWS SDK oder CLI
    - Verwenden Sie DeleteThing mit einem AWS SDK oder CLI
    - Verwenden Sie DeleteTopicRule mit einem AWS SDK oder CLI
    - Verwenden Sie DescribeEndpoint mit einem AWS SDK oder CLI
    - Verwenden Sie DescribeThing mit einem AWS SDK oder CLI
    - Verwenden Sie DetachThingPrincipal mit einem AWS SDK oder CLI
    - Verwenden Sie ListCertificates mit einem AWS SDK oder CLI
    - Verwenden Sie ListThings mit einem AWS SDK oder CLI
    - Verwenden Sie SearchIndex mit einem AWS SDK oder CLI
    - Verwenden Sie UpdateIndexingConfiguration mit einem AWS SDK oder CLI

- Verwenden Sie UpdateThing mit einem AWS SDK oder CLI

Hallo AWS IoT

Die folgenden Codebeispiele zeigen, wie Sie mit der Verwendung von AWS IoT beginnen.

.NET

SDK für .NET (v4)

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
/// <summary>
/// Hello AWS IoT example.
/// </summary>
public class HelloIoT
{
    /// <summary>
    /// Main method to run the Hello IoT example.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        var iotClient = new AmazonIoTClient();

        try
        {
            Console.WriteLine("Hello AWS IoT! Let's list your IoT Things:");
            Console.WriteLine(new string('-', 80));

            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await iotClient.ListThingsAsync(request);
```

```
            // Since there is not a built-in paginator, use the NextMarker to
paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the
next page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await iotClient.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
            }

            if (things is { Count: > 0 })
            {
                Console.WriteLine($"Found {things.Count} IoT Things:");
                foreach (var thing in things)
                {
                    Console.WriteLine($"- Thing Name: {thing.ThingName}");
                    Console.WriteLine($"  Thing ARN: {thing.ThingArn}");
                    Console.WriteLine($"  Thing Type: {thing.ThingTypeName ?? "No
type specified"}");
                    Console.WriteLine($"  Version: {thing.Version}");

                    if (thing.Attributes?.Count > 0)
                    {
                        Console.WriteLine("  Attributes:");
                        foreach (var attr in thing.Attributes)
                        {
                            Console.WriteLine($"    {attr.Key}: {attr.Value}");
                        }
                    }
                    Console.WriteLine();
                }
            }
            else
```

```
            {
                Console.WriteLine("No IoT Things found in your account.");
                Console.WriteLine("You can create IoT Things using the IoT Basics
 scenario example.");
            }

            Console.WriteLine("Hello IoT completed successfully.");
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            Console.WriteLine($"Request throttled, please try again later:
 {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list Things. Here's why: {ex.Message}");
        }
    }
}
```

- Weitere API-Informationen finden Sie unter listThings in der AWS SDK für .NET -API-Referenz.

C++

SDK für C++

Code für die CMakeLists.txt CMake-Datei.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS iot)

# Set this project's name.
project("hello_iot")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)
```

```
# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
    # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
    # and set the proper subdirectory to the executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_iot.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Code für die Quelldatei „hello_iot.cpp“.

```
#include <aws/core/Aws.h>
#include <aws/iot/IoTClient.h>
#include <aws/iot/model/ListThingsRequest.h>
#include <iostream>

/*
 *  A "Hello IoT" starter application which initializes an AWS IoT client and
 *  lists the AWS IoT topics in the current account.
 *
```

```
 *  main function
 *
 *  Usage: 'hello_iot'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    //  Optional: change the log level for debugging.
    //  options.loggingOptions.logLevel = Aws::Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::IoT::IoTClient iotClient(clientConfig);
        // List the things in the current account.
        Aws::IoT::Model::ListThingsRequest listThingsRequest;

        Aws::String nextToken; // Used for pagination.
        Aws::Vector<Aws::IoT::Model::ThingAttribute> allThings;

        do {
            if (!nextToken.empty()) {
                listThingsRequest.SetNextToken(nextToken);
            }

            Aws::IoT::Model::ListThingsOutcome listThingsOutcome =
 iotClient.ListThings(
                    listThingsRequest);
            if (listThingsOutcome.IsSuccess()) {
                const Aws::Vector<Aws::IoT::Model::ThingAttribute> &things =
 listThingsOutcome.GetResult().GetThings();
                allThings.insert(allThings.end(), things.begin(), things.end());
                nextToken = listThingsOutcome.GetResult().GetNextToken();
            }
            else {
                std::cerr << "List things failed"
                          << listThingsOutcome.GetError().GetMessage() <<
 std::endl;
                break;
            }
        } while (!nextToken.empty());
```

```
        std::cout << allThings.size() << " thing(s) found." << std::endl;
        for (auto const &thing: allThings) {
            std::cout << thing.GetThingName() << std::endl;
        }
    }

    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- Weitere API-Informationen finden Sie unter listThings in der AWS SDK für C++ -API-Referenz.

> **Note**
>
> Es läuft noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Java

SDK für Java 2.x

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iot.IotClient;
import software.amazon.awssdk.services.iot.model.ListThingsRequest;
import software.amazon.awssdk.services.iot.model.ListThingsResponse;
import software.amazon.awssdk.services.iot.model.ThingAttribute;
import software.amazon.awssdk.services.iot.paginators.ListThingsIterable;

import java.util.List;

public class HelloIoT {
```

```
    public static void main(String[] args) {
        System.out.println("Hello AWS IoT. Here is a listing of your AWS IoT
 Things:");
        IotClient iotClient = IotClient.builder()
            .region(Region.US_EAST_1)
            .build();

        listAllThings(iotClient);
    }

    public static void listAllThings(IotClient iotClient) {
        iotClient.listThingsPaginator(ListThingsRequest.builder()
                .maxResults(10)
                .build())
            .stream()
            .flatMap(response -> response.things().stream())
            .forEach(attribute -> {
                System.out.println("Thing name: " + attribute.thingName());
                System.out.println("Thing ARN: " + attribute.thingArn());
            });
    }
}
```

- Weitere API-Informationen finden Sie unter listThings in der AWS SDK for Java 2.x -API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import aws.sdk.kotlin.services.iot.IotClient
import aws.sdk.kotlin.services.iot.model.ListThingsRequest

suspend fun main() {
```

```
    println("A listing of your AWS IoT Things:")
    listAllThings()
}

suspend fun listAllThings() {
    val thingsRequest =
        ListThingsRequest {
            maxResults = 10
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val response = iotClient.listThings(thingsRequest)
        val thingList = response.things
        if (thingList != null) {
            for (attribute in thingList) {
                println("Thing name ${attribute.thingName}")
                println("Thing ARN: ${attribute.thingArn}")
            }
        }
    }
}
```

- Weitere API-Informationen finden Sie unter listThings in der API-Referenz zum AWS -SDK für Kotlin.

Python

SDK für Python (Boto3)

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
def hello_iot():
    """
    Use the AWS SDK for Python (Boto3) to create an AWS IoT client and list
    up to 10 things in your AWS IoT account.
    This example uses the default settings specified in your shared credentials
```

```
    and config files.
    """
    try:
        iot_client = boto3.client("iot")
        response = iot_client.list_things(maxResults=10)
        things = response.get("things", [])

        print("Hello, AWS IoT! Here are your things:")
        if things:
            for i, thing in enumerate(things, 1):
                print(f"{i}. {thing['thingName']}")
        else:
            print("No things found in your AWS IoT account.")
    except ClientError as e:
        if e.response["Error"]["Code"] == "UnauthorizedException":
            print("You don't have permission to access AWS IoT.")
        else:
            print(f"Couldn't access AWS IoT. Error: {e}")
    except NoCredentialsError:
        print("No AWS credentials found. Please configure your credentials.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
```

- API-Details finden Sie unter ListThings in der AWS API-Referenz für SDK for Python (Boto3).

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. Verwenden AWS IoT mit einem AWS SDK Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Lernen Sie die Grundlagen von AWS IoT mit einem AWS SDK

Die folgenden Code-Beispiele veranschaulichen Folgendes:

- Erschaffe eine AWS IoT Sache.
- Generieren Sie ein Gerätezertifikat.
- Aktualisiere ein AWS IoT Ding mit Attributen.
- Geben Sie einen eindeutigen Endpunkt zurück.

- Listen Sie Ihre AWS IoT Zertifikate auf.
- Aktualisiere einen AWS IoT Shadow.
- Schreiben Sie Statusinformationen aus.
- Erstellt eine Regel.
- Listen Sie Ihre Regeln auf.
- Suchen Sie Objekte anhand des Objektnamens.
- Lösche eine AWS IoT Sache.

.NET

SDK für .NET (v4)

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario aus, das die AWS IoT Funktionen demonstriert.

```
/// <summary>
/// Scenario class for AWS IoT basics.
/// </summary>
public class IoTBasics
{
    public static bool IsInteractive = true;
    public static IoTWrapper? Wrapper = null;
    public static IAmazonCloudFormation? CloudFormationClient = null;
    public static ILogger<IoTBasics> logger = null!;
    private static IoTWrapper _iotWrapper = null!;
    private static IAmazonCloudFormation _amazonCloudFormation = null!;
    private static ILogger<IoTBasics> _logger = null!;

    private static string _stackName = "IoTBasicsStack";
    private static string _stackResourcePath = "../../../../../../scenarios/
basics/iot/iot_usecase/resources/cfn_template.yaml";

    /// <summary>
    /// Main method for the IoT Basics scenario.
```

```
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonIoT>(new AWSOptions() { Region =
 RegionEndpoint.USEast1 })
                    .AddAWSService<IAmazonCloudFormation>()
                        .AddTransient<IoTWrapper>()
                        .AddLogging(builder => builder.AddConsole())
                        .AddSingleton<IAmazonIotData>(sp =>
                        {
                            var iotService = sp.GetRequiredService<IAmazonIoT>();
                            var request = new DescribeEndpointRequest
                            {
                                EndpointType = "iot:Data-ATS"
                            };
                            var response =
 iotService.DescribeEndpointAsync(request).Result;
                            return new AmazonIotDataClient($"https://
{response.EndpointAddress}/");
                        })
            )
            .Build();

        logger = LoggerFactory.Create(builder => builder.AddConsole())
            .CreateLogger<IoTBasics>();

        Wrapper = host.Services.GetRequiredService<IoTWrapper>();
        CloudFormationClient =
 host.Services.GetRequiredService<IAmazonCloudFormation>();

        // Set the private fields for backwards compatibility
        _logger = logger;
        _iotWrapper = Wrapper;
        _amazonCloudFormation = CloudFormationClient;

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the AWS IoT example scenario.");
        Console.WriteLine("This example program demonstrates various interactions
 with the AWS Internet of Things (IoT) Core service.");
```

```
        Console.WriteLine();
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }
        Console.WriteLine(new string('-', 80));

        try
        {
            await RunScenarioAsync();
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem running the scenario.");
            Console.WriteLine($"\nAn error occurred: {ex.Message}");
        }

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("The AWS IoT scenario has successfully completed.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Run the IoT Basics scenario.
    /// </summary>
    /// <returns>A Task object.</returns>
    public static async Task RunScenarioAsync()
    {
        // Use static properties if available, otherwise use private fields
        var iotWrapper = Wrapper ?? _iotWrapper;
        var cloudFormationClient = CloudFormationClient ?? _amazonCloudFormation;
        var scenarioLogger = logger ?? _logger;

        await RunScenarioInternalAsync(iotWrapper, cloudFormationClient,
scenarioLogger);
    }

    /// <summary>
    /// Internal method to run the IoT Basics scenario with injected
dependencies.
    /// </summary>
    /// <param name="iotWrapper">The IoT wrapper instance.</param>
```

```
    /// <param name="cloudFormationClient">The CloudFormation client instance.</
param>
    /// <param name="scenarioLogger">The logger instance.</param>
    /// <returns>A Task object.</returns>
    private static async Task RunScenarioInternalAsync(IoTWrapper iotWrapper,
 IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        string thingName = $"iot-thing-{Guid.NewGuid():N}";
        string certificateArn = "";
        string certificateId = "";
        string ruleName = $"iotruledefault";
        string snsTopicArn = "";

        try
        {
            // Step 1: Create an AWS IoT Thing
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("1. Create an AWS IoT Thing.");
            Console.WriteLine("An AWS IoT Thing represents a virtual entity in
 the AWS IoT service that can be associated with a physical device.");
            Console.WriteLine();

            if (IsInteractive)
            {
                Console.Write("Enter Thing name: ");
                var userInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(userInput))
                    thingName = userInput;
            }
            else
            {
                Console.WriteLine($"Using default Thing name: {thingName}");
            }

            var thingArn = await iotWrapper.CreateThingAsync(thingName);
            Console.WriteLine($"{thingName} was successfully created. The ARN
 value is {thingArn}");
            Console.WriteLine(new string('-', 80));

            // Step 1.1: List AWS IoT Things
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("2. List AWS IoT Things.");
            Console.WriteLine("Now let's list the IoT Things to see the Thing we
 just created.");
```

```
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var things = await iotWrapper.ListThingsAsync();
            Console.WriteLine($"Found {things.Count} IoT Things:");
            foreach (var thing in things.Take(10)) // Show first 10 things
            {
                Console.WriteLine($"Thing Name: {thing.ThingName}");
                Console.WriteLine($"Thing ARN: {thing.ThingArn}");
                if (thing.Attributes != null && thing.Attributes.Any())
                {
                    Console.WriteLine("Attributes:");
                    foreach (var attr in thing.Attributes)
                    {
                        Console.WriteLine($"  {attr.Key}: {attr.Value}");
                    }
                }
                Console.WriteLine("--------------");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 2: Generate a Device Certificate
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("3. Generate a device certificate.");
            Console.WriteLine("A device certificate performs a role in securing
the communication between devices (Things) and the AWS IoT platform.");
            Console.WriteLine();

            var createCert = "y";
            if (IsInteractive)
            {
                Console.Write($"Do you want to create a certificate for
{thingName}? (y/n)");
                createCert = Console.ReadLine();
            }
            else
            {
                Console.WriteLine($"Creating certificate for {thingName}...");
            }
```

```
            if (createCert?.ToLower() == "y")
            {
                var certificateResult = await
iotWrapper.CreateKeysAndCertificateAsync();
                if (certificateResult.HasValue)
                {
                    var (certArn, certPem, certId) = certificateResult.Value;
                    certificateArn = certArn;
                    certificateId = certId;

                    Console.WriteLine($"\nCertificate:");
                    // Show only first few lines of certificate for brevity
                    var lines = certPem.Split('\n');
                    for (int i = 0; i < Math.Min(lines.Length, 5); i++)
                    {
                        Console.WriteLine(lines[i]);
                    }
                    if (lines.Length > 5)
                    {
                        Console.WriteLine("...");
                    }

                    Console.WriteLine($"\nCertificate ARN:");
                    Console.WriteLine(certificateArn);

                    // Step 3: Attach the Certificate to the AWS IoT Thing
                    Console.WriteLine("Attach the certificate to the AWS IoT
Thing.");
                    var attachResult = await
iotWrapper.AttachThingPrincipalAsync(thingName, certificateArn);
                    if (attachResult)
                    {
                        Console.WriteLine("Certificate attached to Thing
successfully.");
                    }
                    else
                    {
                        Console.WriteLine("Failed to attach certificate to
Thing.");
                    }

                    Console.WriteLine("Thing Details:");
                    Console.WriteLine($"Thing Name: {thingName}");
```

```
                Console.WriteLine($"Thing ARN: {thingArn}");
            }
            else
            {
                Console.WriteLine("Failed to create certificate.");
            }
        }
        Console.WriteLine(new string('-', 80));

        // Step 4: Update an AWS IoT Thing with Attributes
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("4. Update an AWS IoT Thing with Attributes.");
        Console.WriteLine("IoT Thing attributes, represented as key-value
pairs, offer a pivotal advantage in facilitating efficient data");
        Console.WriteLine("management and retrieval within the AWS IoT
ecosystem.");
        Console.WriteLine();
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }

        var attributes = new Dictionary<string, string>
        {
            { "Location", "Seattle" },
            { "DeviceType", "Sensor" },
            { "Firmware", "1.2.3" }
        };

        await iotWrapper.UpdateThingAsync(thingName, attributes);
        Console.WriteLine("Thing attributes updated successfully.");
        Console.WriteLine(new string('-', 80));

        // Step 5: Return a unique endpoint specific to the Amazon Web
Services account
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("5. Return a unique endpoint specific to the Amazon
Web Services account.");
        Console.WriteLine();
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
```

```
            }

            var endpoint = await iotWrapper.DescribeEndpointAsync();
            if (endpoint != null)
            {
                var subdomain = endpoint.Split('.')[0];
                Console.WriteLine($"Extracted subdomain: {subdomain}");
                Console.WriteLine($"Full Endpoint URL: https://{endpoint}");
            }
            else
            {
                Console.WriteLine("Failed to retrieve endpoint.");
            }
            Console.WriteLine(new string('-', 80));

            // Step 6: List your AWS IoT certificates
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("6. List your AWS IoT certificates");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var certificates = await iotWrapper.ListCertificatesAsync();
            foreach (var cert in certificates.Take(5)) // Show first 5
certificates
            {
                Console.WriteLine($"Cert id: {cert.CertificateId}");
                Console.WriteLine($"Cert Arn: {cert.CertificateArn}");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 7: Create an IoT shadow
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("7. Update an IoT shadow that refers to a digital
representation or virtual twin of a physical IoT device");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }
```

```
            var shadowPayload = JsonSerializer.Serialize(new
            {
                state = new
                {
                    desired = new
                    {
                        temperature = 25,
                        humidity = 50
                    }
                }
            });

            await iotWrapper.UpdateThingShadowAsync(thingName, shadowPayload);
            Console.WriteLine("Thing Shadow updated successfully.");
            Console.WriteLine(new string('-', 80));

            // Step 8: Write out the state information, in JSON format
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("8. Write out the state information, in JSON
format.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var shadowData = await iotWrapper.GetThingShadowAsync(thingName);
            Console.WriteLine($"Received Shadow Data: {shadowData}");
            Console.WriteLine(new string('-', 80));

            // Step 9: Set up resources (SNS topic and IAM role) and create a
rule
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("9. Set up resources and create a rule");
            Console.WriteLine();

            // Deploy CloudFormation stack to create SNS topic and IAM role
            Console.WriteLine("Deploying CloudFormation stack to create SNS topic
and IAM role...");

            var deployStack = !IsInteractive || GetYesNoResponse("Would you like
to deploy the CloudFormation stack? (y/n) ");
            if (deployStack)
```

```
            {
                if (IsInteractive)
                {
                    Console.Write(
                        $"Enter stack resource file path (or press Enter for
default '{_stackResourcePath}'): ");
                    var userResourcePath = Console.ReadLine();
                    if (!string.IsNullOrEmpty(userResourcePath))
                        _stackResourcePath = userResourcePath;
                }

                _stackName = PromptUserForStackName();

                var deploySuccess = await DeployCloudFormationStack(_stackName,
cloudFormationClient, scenarioLogger);

                if (deploySuccess)
                {
                    // Get stack outputs
                    var stackOutputs = await GetStackOutputs(_stackName,
cloudFormationClient, scenarioLogger);
                    if (stackOutputs != null)
                    {
                        snsTopicArn = stackOutputs["SNSTopicArn"];
                        string roleArn = stackOutputs["RoleArn"];

                        Console.WriteLine($"Successfully deployed stack. SNS
topic: {snsTopicArn}");
                        Console.WriteLine($"Successfully deployed stack. IAM
role: {roleArn}");

                        if (IsInteractive)
                        {
                            Console.Write($"Enter Rule name (press Enter for
default '{ruleName}'): ");
                            var userRuleName = Console.ReadLine();
                            if (!string.IsNullOrEmpty(userRuleName))
                                ruleName = userRuleName;
                        }
                        else
                        {
                            Console.WriteLine($"Using default rule name:
{ruleName}");
                        }
```

```
                        // Now create the IoT rule with the CloudFormation
outputs
                        var ruleResult = await
iotWrapper.CreateTopicRuleAsync(ruleName, snsTopicArn, roleArn);
                        if (ruleResult)
                        {
                            Console.WriteLine("IoT Rule created successfully.");
                        }
                        else
                        {
                            Console.WriteLine("Failed to create IoT rule.");
                        }
                    }
                    else
                    {
                        Console.WriteLine("Failed to get stack outputs. Skipping
rule creation.");
                    }
                }
                else
                {
                    Console.WriteLine("Failed to deploy CloudFormation stack.
Skipping rule creation.");
                }
            }
            else
            {
                Console.WriteLine("Skipping CloudFormation stack deployment and
rule creation.");
            }
            Console.WriteLine(new string('-', 80));

            // Step 10: List your rules
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("10. List your rules.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var rules = await iotWrapper.ListTopicRulesAsync();
            Console.WriteLine("List of IoT Rules:");
```

```
            foreach (var rule in rules.Take(5)) // Show first 5 rules
            {
                Console.WriteLine($"Rule Name: {rule.RuleName}");
                Console.WriteLine($"Rule ARN: {rule.RuleArn}");
                Console.WriteLine("--------------");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 11: Search things using the Thing name
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("11. Search things using the Thing name.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var searchResults = await iotWrapper.SearchIndexAsync($"thingName:
{thingName}");
            if (searchResults.Any())
            {
                Console.WriteLine($"Thing id found using search is
 {searchResults.First().ThingId}");
            }
            else
            {
                Console.WriteLine($"No search results found for Thing:
 {thingName}");
            }
            Console.WriteLine(new string('-', 80));

            // Step 12: Cleanup - Detach and delete certificate
            if (!string.IsNullOrEmpty(certificateArn))
            {
                Console.WriteLine(new string('-', 80));
                var deleteCert = "y";
                if (IsInteractive)
                {
                    Console.Write($"Do you want to detach and delete the
 certificate for {thingName}? (y/n)");
                    deleteCert = Console.ReadLine();
                }
                else
```

```
                {
                    Console.WriteLine($"Detaching and deleting certificate for
{thingName}...");
                }

                if (deleteCert?.ToLower() == "y")
                {
                    Console.WriteLine("12. You selected to detach and delete the
certificate.");
                    if (IsInteractive)
                    {
                        Console.WriteLine("Press Enter to continue...");
                        Console.ReadLine();
                    }

                    await iotWrapper.DetachThingPrincipalAsync(thingName,
certificateArn);
                    Console.WriteLine($"{certificateArn} was successfully removed
from {thingName}");

                    await iotWrapper.DeleteCertificateAsync(certificateId);
                    Console.WriteLine($"{certificateArn} was successfully
deleted.");
                }
                Console.WriteLine(new string('-', 80));
            }

            // Step 13: Delete the AWS IoT Thing
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("13. Delete the AWS IoT Thing.");
            var deleteThing = "y";
            if (IsInteractive)
            {
                Console.Write($"Do you want to delete the IoT Thing? (y/n)");
                deleteThing = Console.ReadLine();
            }
            else
            {
                Console.WriteLine($"Deleting IoT Thing {thingName}...");
            }

            if (deleteThing?.ToLower() == "y")
            {
                await iotWrapper.DeleteThingAsync(thingName);
```

```
                Console.WriteLine($"Deleted Thing {thingName}");
            }
            Console.WriteLine(new string('-', 80));

            // Step 14: Clean up CloudFormation stack
            if (!string.IsNullOrEmpty(snsTopicArn))
            {
                Console.WriteLine(new string('-', 80));
                Console.WriteLine("14. Clean up CloudFormation stack.");
                Console.WriteLine("Deleting the CloudFormation stack and all
resources...");

                var cleanup = !IsInteractive || GetYesNoResponse("Do you want to
delete the CloudFormation stack and all resources? (y/n) ");
                if (cleanup)
                {
                    var ruleCleanupSuccess = await
iotWrapper.DeleteTopicRuleAsync(ruleName);

                    var stackCleanupSuccess = await
DeleteCloudFormationStack(_stackName, cloudFormationClient, scenarioLogger);
                    if (ruleCleanupSuccess && stackCleanupSuccess)
                    {
                        Console.WriteLine("Successfully cleaned up CloudFormation
stack and all resources.");
                    }
                    else
                    {
                        Console.WriteLine("Some cleanup operations failed. Check
the logs for details.");
                    }
                }
                else
                {
                    Console.WriteLine($"Resources will remain. Stack name:
{_stackName}");
                }
                Console.WriteLine(new string('-', 80));
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, "Error occurred during scenario
execution.");
```

```
            // Cleanup on error
            if (!string.IsNullOrEmpty(certificateArn) && !
string.IsNullOrEmpty(thingName))
            {
                try
                {
                    await iotWrapper.DetachThingPrincipalAsync(thingName,
 certificateArn);
                    await iotWrapper.DeleteCertificateAsync(certificateId);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during cleanup.");
                }
            }

            if (!string.IsNullOrEmpty(thingName))
            {
                try
                {
                    await iotWrapper.DeleteThingAsync(thingName);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during Thing
 cleanup.");
                }
            }

            // Clean up CloudFormation stack on error
            if (!string.IsNullOrEmpty(snsTopicArn))
            {
                try
                {
                    await _iotWrapper.DeleteTopicRuleAsync(ruleName);
                    await DeleteCloudFormationStack(_stackName,
 cloudFormationClient, scenarioLogger);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during
 CloudFormation stack cleanup.");
                }
```

```
            }

            throw;
        }
    }

    /// <summary>
    /// Deploys the CloudFormation stack with the necessary resources.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was deployed successfully.</returns>
    private static async Task<bool> DeployCloudFormationStack(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        Console.WriteLine($"\nDeploying CloudFormation stack: {stackName}");

        try
        {
            var request = new CreateStackRequest
            {
                StackName = stackName,
                TemplateBody = await File.ReadAllTextAsync(_stackResourcePath),
                Capabilities = new List<string>
{ Capability.CAPABILITY_NAMED_IAM }
            };

            var response = await cloudFormationClient.CreateStackAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

                bool stackCreated = await
WaitForStackCompletion(response.StackId, cloudFormationClient, scenarioLogger);

                if (stackCreated)
                {
                    Console.WriteLine("CloudFormation stack created
successfully.");
                    return true;
                }
```

```
                else
                {
                    scenarioLogger.LogError($"CloudFormation stack creation
failed: {stackName}");
                    return false;
                }
            }
            else
            {
                scenarioLogger.LogError($"Failed to create CloudFormation stack:
{stackName}");
                return false;
            }
        }
        catch (AlreadyExistsException)
        {
            scenarioLogger.LogWarning($"CloudFormation stack '{stackName}'
already exists. Please provide a unique name.");
            var newStackName = PromptUserForStackName();
            return await DeployCloudFormationStack(newStackName,
cloudFormationClient, scenarioLogger);
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"An error occurred while deploying the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the CloudFormation stack to be in the CREATE_COMPLETE state.
    /// </summary>
    /// <param name="stackId">The ID of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was created successfully.</returns>
    private static async Task<bool> WaitForStackCompletion(string stackId,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;
```

```
        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackId
            };

            var describeStacksResponse = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

            if (describeStacksResponse.Stacks.Count > 0)
            {
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_COMPLETE)
                {
                    return true;
                }
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_FAILED ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.ROLLBACK_COMPLETE)
                {
                    return false;
                }
            }

            Console.WriteLine("Waiting for CloudFormation stack creation to
complete...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        scenarioLogger.LogError("Timed out waiting for CloudFormation stack
creation to complete.");
        return false;
    }

    /// <summary>
    /// Gets the outputs from the CloudFormation stack.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>A dictionary of stack outputs.</returns>
```

```
   private static async Task<Dictionary<string, string>?> GetStackOutputs(string
stackName, IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics>
scenarioLogger)
   {
       try
       {
           var describeStacksRequest = new DescribeStacksRequest
           {
               StackName = stackName
           };

           var response = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

           if (response.Stacks.Count > 0)
           {
               var outputs = new Dictionary<string, string>();
               foreach (var output in response.Stacks[0].Outputs)
               {
                   outputs[output.OutputKey] = output.OutputValue;
               }
               return outputs;
           }

           return null;
       }
       catch (Exception ex)
       {
           scenarioLogger.LogError(ex, $"Failed to get stack outputs for
{stackName}");
           return null;
       }
   }

   /// <summary>
   /// Deletes the CloudFormation stack and waits for confirmation.
   /// </summary>
   /// <param name="stackName">The name of the CloudFormation stack.</param>
   /// <param name="cloudFormationClient">The CloudFormation client.</param>
   /// <param name="scenarioLogger">The logger.</param>
   /// <returns>True if the stack was deleted successfully.</returns>
   private static async Task<bool> DeleteCloudFormationStack(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
   {
```

```
        try
        {
            var request = new DeleteStackRequest
            {
                StackName = stackName
            };

            await cloudFormationClient.DeleteStackAsync(request);
            Console.WriteLine($"CloudFormation stack '{stackName}' is being
deleted. This may take a few minutes.");

            bool stackDeleted = await WaitForStackDeletion(stackName,
cloudFormationClient, scenarioLogger);

            if (stackDeleted)
            {
                Console.WriteLine($"CloudFormation stack '{stackName}' has been
deleted.");
                return true;
            }
            else
            {
                scenarioLogger.LogError($"Failed to delete CloudFormation stack
'{stackName}'.");
                return false;
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"An error occurred while deleting the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the stack to be deleted.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was deleted successfully.</returns>
    private static async Task<bool> WaitForStackDeletion(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
```

```
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };

            try
            {
                var describeStacksResponse = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

                if (describeStacksResponse.Stacks.Count == 0 ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_COMPLETE)
                {
                    return true;
                }
            }
            catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
"ValidationError")
            {
                return true;
            }

            Console.WriteLine($"Waiting for CloudFormation stack '{stackName}' to
be deleted...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        scenarioLogger.LogError($"Timed out waiting for CloudFormation stack
'{stackName}' to be deleted.");
        return false;
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
```

```
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
 StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Prompts the user for a stack name.
    /// </summary>
    private static string PromptUserForStackName()
    {
        if (IsInteractive)
        {
            Console.Write($"Enter a name for the CloudFormation stack (press
 Enter for default '{_stackName}'): ");
            string? input = Console.ReadLine();
            if (!string.IsNullOrWhiteSpace(input))
            {
                var regex = new System.Text.RegularExpressions.Regex("[a-zA-Z][-
a-zA-Z0-9]*");
                if (!regex.IsMatch(input))
                {
                    Console.WriteLine($"Invalid stack name. Using default:
 {_stackName}");
                    return _stackName;
                }
                return input;
            }
        }
        return _stackName;
    }
}
```

Eine Wrapper-Klasse für AWS IoT SDK-Methoden.

```
/// <summary>
/// Wrapper methods to use Amazon IoT Core with .NET.
/// </summary>
public class IoTWrapper
```

```
{
    private readonly IAmazonIoT _amazonIoT;
    private readonly IAmazonIotData _amazonIotData;
    private readonly ILogger<IoTWrapper> _logger;

    /// <summary>
    /// Constructor for the IoT wrapper.
    /// </summary>
    /// <param name="amazonIoT">The injected IoT client.</param>
    /// <param name="amazonIotData">The injected IoT Data client.</param>
    /// <param name="logger">The injected logger.</param>
    public IoTWrapper(IAmazonIoT amazonIoT, IAmazonIotData amazonIotData,
 ILogger<IoTWrapper> logger)
    {
        _amazonIoT = amazonIoT;
        _amazonIotData = amazonIotData;
        _logger = logger;
    }

    /// <summary>
    /// Creates an AWS IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to create.</param>
    /// <returns>The ARN of the Thing created, or null if creation failed.</
returns>
    public async Task<string?> CreateThingAsync(string thingName)
    {
        try
        {
            var request = new CreateThingRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIoT.CreateThingAsync(request);
            _logger.LogInformation($"Created Thing {thingName} with ARN
 {response.ThingArn}");
            return response.ThingArn;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Thing {thingName} already exists:
 {ex.Message}");
            return null;
```

```
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create Thing {thingName}. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Creates a device certificate for AWS IoT.
    /// </summary>
    /// <returns>The certificate details including ARN and certificate PEM, or
null if creation failed.</returns>
    public async Task<(string CertificateArn, string CertificatePem, string
CertificateId)?> CreateKeysAndCertificateAsync()
    {
        try
        {
            var request = new CreateKeysAndCertificateRequest
            {
                SetAsActive = true
            };

            var response = await
_amazonIoT.CreateKeysAndCertificateAsync(request);
            _logger.LogInformation($"Created certificate with ARN
{response.CertificateArn}");
            return (response.CertificateArn, response.CertificatePem,
response.CertificateId);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create certificate. Here's why:
{ex.Message}");
            return null;
        }
    }
```

```
    /// <summary>
    /// Attaches a certificate to an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to attach.</
param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> AttachThingPrincipalAsync(string thingName, string
 certificateArn)
    {
        try
        {
            var request = new AttachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.AttachThingPrincipalAsync(request);
            _logger.LogInformation($"Attached certificate {certificateArn} to
 Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot attach certificate - resource not found:
 {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't attach certificate to Thing. Here's why:
 {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Updates an IoT Thing with attributes.
    /// </summary>
    /// <param name="thingName">The name of the Thing to update.</param>
    /// <param name="attributes">Dictionary of attributes to add.</param>
    /// <returns>True if successful, false otherwise.</returns>
```

```
    public async Task<bool> UpdateThingAsync(string thingName, Dictionary<string,
string> attributes)
    {
        try
        {
            var request = new UpdateThingRequest
            {
                ThingName = thingName,
                AttributePayload = new AttributePayload
                {
                    Attributes = attributes,
                    Merge = true
                }
            };

            await _amazonIoT.UpdateThingAsync(request);
            _logger.LogInformation($"Updated Thing {thingName} with attributes");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing attributes. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Gets the AWS IoT endpoint URL.
    /// </summary>
    /// <returns>The endpoint URL, or null if retrieval failed.</returns>
    public async Task<string?> DescribeEndpointAsync()
    {
        try
        {
            var request = new DescribeEndpointRequest
            {
                EndpointType = "iot:Data-ATS"
```

```
            };

            var response = await _amazonIoT.DescribeEndpointAsync(request);
            _logger.LogInformation($"Retrieved endpoint:
{response.EndpointAddress}");
            return response.EndpointAddress;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't describe endpoint. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Lists all certificates associated with the account.
    /// </summary>
    /// <returns>List of certificate information, or empty list if listing
failed.</returns>
    public async Task<List<Certificate>> ListCertificatesAsync()
    {
        try
        {
            var request = new ListCertificatesRequest();
            var response = await _amazonIoT.ListCertificatesAsync(request);

            _logger.LogInformation($"Retrieved {response.Certificates.Count}
certificates");
            return response.Certificates;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<Certificate>();
        }
        catch (Exception ex)
```

```
        {
            _logger.LogError($"Couldn't list certificates. Here's why:
{ex.Message}");
            return new List<Certificate>();
        }
    }

    /// <summary>
    /// Updates the Thing's shadow with new state information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="shadowPayload">The shadow payload in JSON format.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingShadowAsync(string thingName, string
shadowPayload)
    {
        try
        {
            var request = new UpdateThingShadowRequest
            {
                ThingName = thingName,
                Payload = new
MemoryStream(System.Text.Encoding.UTF8.GetBytes(shadowPayload))
            };

            await _amazonIotData.UpdateThingShadowAsync(request);
            _logger.LogInformation($"Updated shadow for Thing {thingName}");
            return true;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing shadow - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing shadow. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
```

```
    /// Gets the Thing's shadow information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <returns>The shadow data as a string, or null if retrieval failed.</
returns>
    public async Task<string?> GetThingShadowAsync(string thingName)
    {
        try
        {
            var request = new GetThingShadowRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIotData.GetThingShadowAsync(request);
            using var reader = new StreamReader(response.Payload);
            var shadowData = await reader.ReadToEndAsync();

            _logger.LogInformation($"Retrieved shadow for Thing {thingName}");
            return shadowData;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot get Thing shadow - resource not found:
 {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't get Thing shadow. Here's why:
 {ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Creates an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="snsTopicArn">The ARN of the SNS topic for the action.</
param>
    /// <param name="roleArn">The ARN of the IAM role.</param>
    /// <returns>True if successful, false otherwise.</returns>
```

```
   public async Task<bool> CreateTopicRuleAsync(string ruleName, string
snsTopicArn, string roleArn)
   {
       try
       {
           var request = new CreateTopicRuleRequest
           {
               RuleName = ruleName,
               TopicRulePayload = new TopicRulePayload
               {
                   Sql = "SELECT * FROM 'topic/subtopic'",
                   Description = $"Rule created by .NET example: {ruleName}",
                   Actions = new List<Amazon.IoT.Model.Action>
                   {
                       new Amazon.IoT.Model.Action
                       {
                           Sns = new SnsAction
                           {
                               TargetArn = snsTopicArn,
                               RoleArn = roleArn
                           }
                       },
                   RuleDisabled = false
               }
           };

           await _amazonIoT.CreateTopicRuleAsync(request);
           _logger.LogInformation($"Created IoT rule {ruleName}");
           return true;
       }
       catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
       {
           _logger.LogWarning($"Rule {ruleName} already exists: {ex.Message}");
           return false;
       }
       catch (Exception ex)
       {
           _logger.LogError($"Couldn't create topic rule. Here's why:
{ex.Message}");
           return false;
       }
   }
```

```
    /// <summary>
    /// Deletes an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteTopicRuleAsync(string ruleName)
    {
        try
        {
            var request = new DeleteTopicRuleRequest
            {
                RuleName = ruleName,
            };

            await _amazonIoT.DeleteTopicRuleAsync(request);
            _logger.LogInformation($"Deleted IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogWarning($"Rule {ruleName} not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete topic rule. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Lists all IoT topic rules.
    /// </summary>
    /// <returns>List of topic rules, or empty list if listing failed.</returns>
    public async Task<List<TopicRuleListItem>> ListTopicRulesAsync()
    {
        try
        {
            var request = new ListTopicRulesRequest();
            var response = await _amazonIoT.ListTopicRulesAsync(request);

            _logger.LogInformation($"Retrieved {response.Rules.Count} IoT
rules");
```

```
            return response.Rules;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<TopicRuleListItem>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list topic rules. Here's why:
{ex.Message}");
            return new List<TopicRuleListItem>();
        }
    }

    /// <summary>
    /// Searches for IoT Things using the search index.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
search failed.</returns>
    public async Task<List<ThingDocument>> SearchIndexAsync(string queryString)
    {
        try
        {
            // First, try to perform the search
            var request = new SearchIndexRequest
            {
                QueryString = queryString
            };

            var response = await _amazonIoT.SearchIndexAsync(request);
            _logger.LogInformation($"Search found {response.Things.Count}
Things");
            return response.Things;
        }
        catch (Amazon.IoT.Model.IndexNotReadyException ex)
        {
            _logger.LogWarning($"Search index not ready, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
```

```
        catch (Amazon.IoT.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
        {
            _logger.LogWarning($"Search index not configured, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't search index. Here's why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Sets up the indexing configuration and retries the search after waiting
for the index to be ready.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
setup/search failed.</returns>
    private async Task<List<ThingDocument>> SetupIndexAndRetrySearchAsync(string
queryString)
    {
        try
        {
            // Update indexing configuration to REGISTRY mode
            _logger.LogInformation("Setting up IoT search indexing
configuration...");
            await _amazonIoT.UpdateIndexingConfigurationAsync(
                new UpdateIndexingConfigurationRequest()
                {
                    ThingIndexingConfiguration = new ThingIndexingConfiguration()
                    {
                        ThingIndexingMode = ThingIndexingMode.REGISTRY
                    }
                });
```

```
            _logger.LogInformation("Indexing configuration updated. Waiting for
index to be ready...");

            // Wait for the index to be set up - this can take some time
            const int maxRetries = 10;
            const int retryDelaySeconds = 10;

            for (int attempt = 1; attempt <= maxRetries; attempt++)
            {
                try
                {
                    _logger.LogInformation($"Waiting for index to be ready
(attempt {attempt}/{maxRetries})...");
                    await Task.Delay(TimeSpan.FromSeconds(retryDelaySeconds));

                    // Try to get the current indexing configuration to see if
it's ready
                    var configResponse = await
_amazonIoT.GetIndexingConfigurationAsync(new GetIndexingConfigurationRequest());
                    if
(configResponse.ThingIndexingConfiguration?.ThingIndexingMode ==
ThingIndexingMode.REGISTRY)
                    {
                        // Try the search again
                        var request = new SearchIndexRequest
                        {
                            QueryString = queryString
                        };

                        var response = await
_amazonIoT.SearchIndexAsync(request);
                        _logger.LogInformation($"Search found
{response.Things.Count} Things after index setup");
                        return response.Things;
                    }
                }
                catch (Amazon.IoT.Model.IndexNotReadyException)
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
```

```
                catch (Amazon.IoT.Model.InvalidRequestException ex) when
 (ex.Message.Contains("index") || ex.Message.Contains("Index"))
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
 wait...");
                    continue;
                }
            }

            _logger.LogWarning("Timeout waiting for search index to be ready
 after configuration update");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't set up search index configuration. Here's
 why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Detaches a certificate from an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to detach.</
param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DetachThingPrincipalAsync(string thingName, string
 certificateArn)
    {
        try
        {
            var request = new DetachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.DetachThingPrincipalAsync(request);
            _logger.LogInformation($"Detached certificate {certificateArn} from
 Thing {thingName}");
            return true;
```

```
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot detach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't detach certificate from Thing. Here's
why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT certificate.
    /// </summary>
    /// <param name="certificateId">The ID of the certificate to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteCertificateAsync(string certificateId)
    {
        try
        {
            // First, update the certificate to inactive state
            var updateRequest = new UpdateCertificateRequest
            {
                CertificateId = certificateId,
                NewStatus = CertificateStatus.INACTIVE
            };
            await _amazonIoT.UpdateCertificateAsync(updateRequest);

            // Then delete the certificate
            var deleteRequest = new DeleteCertificateRequest
            {
                CertificateId = certificateId
            };

            await _amazonIoT.DeleteCertificateAsync(deleteRequest);
            _logger.LogInformation($"Deleted certificate {certificateId}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
```

```
            _logger.LogError($"Cannot delete certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete certificate. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteThingAsync(string thingName)
    {
        try
        {
            var request = new DeleteThingRequest
            {
                ThingName = thingName
            };

            await _amazonIoT.DeleteThingAsync(request);
            _logger.LogInformation($"Deleted Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete Thing. Here's why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
```

```
    /// Lists IoT Things with pagination support.
    /// </summary>
    /// <returns>List of Things, or empty list if listing failed.</returns>
    public async Task<List<ThingAttribute>> ListThingsAsync()
    {
        try
        {
            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await _amazonIoT.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the
next page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await _amazonIoT.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
            }

            _logger.LogInformation($"Retrieved {things.Count} Things");
            return things;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingAttribute>();
        }
        catch (Exception ex)
```

```
        {
            _logger.LogError($"Couldn't list Things. Here's why: {ex.Message}");
            return new List<ThingAttribute>();
        }
    }

}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für .NET - API-Referenz.
  - AttachThingPrincipal
  - CreateKeysAndCertificate
  - CreateThing
  - CreateTopicRule
  - DeleteCertificate
  - DeleteThing
  - DeleteTopicRule
  - DescribeEndpoint
  - DescribeThing
  - DetachThingPrincipal
  - ListCertificates
  - ListThings
  - SearchIndex
  - UpdateIndexingConfiguration
  - UpdateThing

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erschaffe AWS IoT irgendwas.

```
    Aws::String thingName = askQuestion("Enter a thing name: ");

    if (!createThing(thingName, clientConfiguration)) {
        std::cerr << "Exiting because createThing failed." << std::endl;
        cleanup("", "", "", "", "", false, clientConfiguration);
        return false;
    }
```

```
//! Create an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::createThing(const Aws::String &thingName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::CreateThingRequest createThingRequest;
    createThingRequest.SetThingName(thingName);

    Aws::IoT::Model::CreateThingOutcome outcome = iotClient.CreateThing(
            createThingRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to create thing " << thingName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

Generieren Sie ein Gerätezertifikat und fügen Sie es an.

```
    Aws::String certificateARN;
    Aws::String certificateID;
```

```
   if (askYesNoQuestion("Would you like to create a certificate for your thing?
 (y/n) ")) {
       Aws::String outputFolder;
       if (askYesNoQuestion(
               "Would you like to save the certificate and keys to file? (y/n)
 ")) {
           outputFolder = std::filesystem::current_path();
           outputFolder += "/device_keys_and_certificates";

           std::filesystem::create_directories(outputFolder);

           std::cout << "The certificate and keys will be saved to the folder: "
                     << outputFolder << std::endl;
       }

       if (!createKeysAndCertificate(outputFolder, certificateARN,
certificateID,
                                     clientConfiguration)) {
           std::cerr << "Exiting because createKeysAndCertificate failed."
                     << std::endl;
           cleanup(thingName, "", "", "", "", false, clientConfiguration);
           return false;
       }

       std::cout << "\nNext, the certificate will be attached to the thing.\n"
                 << std::endl;
       if (!attachThingPrincipal(certificateARN, thingName,
clientConfiguration)) {
           std::cerr << "Exiting because attachThingPrincipal failed." <<
std::endl;
           cleanup(thingName, certificateARN, certificateID, "", "",
                   false,
                   clientConfiguration);
           return false;
       }
   }
```

```
//! Create keys and certificate for an Aws IoT device.
//! This routine will save certificates and keys to an output folder, if
 provided.
/*!
```

```
  \param outputFolder: Location for storing output in files, ignored when string
 is empty.
  \param certificateARNResult: A string to receive the ARN of the created
 certificate.
  \param certificateID: A string to receive the ID of the created certificate.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::createKeysAndCertificate(const Aws::String &outputFolder,
                                           Aws::String &certificateARNResult,
                                           Aws::String &certificateID,
                                           const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient client(clientConfiguration);
    Aws::IoT::Model::CreateKeysAndCertificateRequest
 createKeysAndCertificateRequest;

    Aws::IoT::Model::CreateKeysAndCertificateOutcome outcome =
            client.CreateKeysAndCertificate(createKeysAndCertificateRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created a certificate and keys" << std::endl;
        certificateARNResult = outcome.GetResult().GetCertificateArn();
        certificateID = outcome.GetResult().GetCertificateId();
        std::cout << "Certificate ARN: " << certificateARNResult << ",
 certificate ID: "
                  << certificateID << std::endl;

        if (!outputFolder.empty()) {
            std::cout << "Writing certificate and keys to the folder '" <<
 outputFolder
                      << "'." << std::endl;
            std::cout << "Be sure these files are stored securely." << std::endl;

            Aws::String certificateFilePath = outputFolder + "/
certificate.pem.crt";
            std::ofstream certificateFile(certificateFilePath);
            if (!certificateFile.is_open()) {
                std::cerr << "Error opening certificate file, '" <<
 certificateFilePath
                          << "'."
                          << std::endl;
                return false;
            }
            certificateFile << outcome.GetResult().GetCertificatePem();
```

```
            certificateFile.close();

            const Aws::IoT::Model::KeyPair &keyPair =
 outcome.GetResult().GetKeyPair();

            Aws::String privateKeyFilePath = outputFolder + "/private.pem.key";
            std::ofstream privateKeyFile(privateKeyFilePath);
            if (!privateKeyFile.is_open()) {
                std::cerr << "Error opening private key file, '" <<
 privateKeyFilePath
                          << "'."
                          << std::endl;
                return false;
            }
            privateKeyFile << keyPair.GetPrivateKey();
            privateKeyFile.close();

            Aws::String publicKeyFilePath = outputFolder + "/public.pem.key";
            std::ofstream publicKeyFile(publicKeyFilePath);
            if (!publicKeyFile.is_open()) {
                std::cerr << "Error opening public key file, '" <<
 publicKeyFilePath
                          << "'."
                          << std::endl;
                return false;
            }
            publicKeyFile << keyPair.GetPublicKey();
        }
    }
    else {
        std::cerr << "Error creating keys and certificate: "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

//! Attach a principal to an AWS IoT thing.
/*!
  \param principal: A principal to attach.
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
```

```
bool AwsDoc::IoT::attachThingPrincipal(const Aws::String &principal,
                                       const Aws::String &thingName,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
   Aws::IoT::IoTClient client(clientConfiguration);
   Aws::IoT::Model::AttachThingPrincipalRequest request;
   request.SetPrincipal(principal);
   request.SetThingName(thingName);
   Aws::IoT::Model::AttachThingPrincipalOutcome outcome =
 client.AttachThingPrincipal(
           request);
   if (outcome.IsSuccess()) {
       std::cout << "Successfully attached principal to thing." << std::endl;
   }
   else {
       std::cerr << "Failed to attach principal to thing." <<
                 outcome.GetError().GetMessage() << std::endl;
   }

   return outcome.IsSuccess();
}
```

Führe verschiedene Operationen an dem AWS IoT Ding aus.

```
   if (!updateThing(thingName, { {"location", "Office"}, {"firmwareVersion",
 "v2.0"} }, clientConfiguration)) {
       std::cerr << "Exiting because updateThing failed." << std::endl;
       cleanup(thingName, certificateARN, certificateID, "", "", false,
               clientConfiguration);
       return false;
   }

   printAsterisksLine();

   std::cout << "Now an endpoint will be retrieved for your account.\n" <<
 std::endl;
   std::cout << "An IoT Endpoint refers to a specific URL or Uniform Resource
 Locator that serves as the entry point\n"
   << "for communication between IoT devices and the AWS IoT service." <<
 std::endl;

   askQuestion("Press Enter to continue:", alwaysTrueTest);
```

```
    Aws::String endpoint;
    if (!describeEndpoint(endpoint, clientConfiguration)) {
        std::cerr << "Exiting because getEndpoint failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, "", "", false,
                clientConfiguration);
        return false;
    }
    std::cout <<"Your endpoint is " << endpoint << "." << std::endl;
    printAsterisksLine();

    std::cout << "Now the certificates in your account will be listed." <<
 std::endl;
    askQuestion("Press Enter to continue:", alwaysTrueTest);

    if (!listCertificates(clientConfiguration)) {
        std::cerr << "Exiting because listCertificates failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, "", "", false,
                clientConfiguration);
        return false;
    }

    printAsterisksLine();

    std::cout << "Now the shadow for the thing will be updated.\n" << std::endl;
    std::cout << "A thing shadow refers to a feature that enables you to create a
 virtual representation, or \"shadow,\"\n"
    << "of a physical device or thing. The thing shadow allows you to synchronize
 and control the state of a device between\n"
    << "the cloud and the device itself. and the AWS IoT service. For example,
 you can write and retrieve JSON data from a thing shadow." << std::endl;
    askQuestion("Press Enter to continue:", alwaysTrueTest);

    if (!updateThingShadow(thingName, R"({"state":{"reported":
{"temperature":25,"humidity":50}}})", clientConfiguration)) {
        std::cerr << "Exiting because updateThingShadow failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, "", "", false,
                clientConfiguration);
        return false;
    }

    printAsterisksLine();
```

```
   std::cout << "Now, the state information for the shadow will be retrieved.\n"
<< std::endl;
   askQuestion("Press Enter to continue:", alwaysTrueTest);

   Aws::String shadowState;
   if (!getThingShadow(thingName, shadowState, clientConfiguration)) {
       std::cerr << "Exiting because getThingShadow failed." << std::endl;
       cleanup(thingName, certificateARN, certificateID, "", "", false,
               clientConfiguration);
       return false;
   }
   std::cout << "The retrieved shadow state is: " << shadowState << std::endl;

   printAsterisksLine();

   std::cout << "A rule with now be added to to the thing.\n" << std::endl;
   std::cout << "Any user who has permission to create rules will be able to
access data processed by the rule." << std::endl;
   std::cout << "In this case, the rule will use an Simple Notification Service
(SNS) topic and an IAM rule." << std::endl;
   std::cout << "These resources will be created using a CloudFormation
template." << std::endl;
   std::cout << "Stack creation may take a few minutes." << std::endl;

   askQuestion("Press Enter to continue: ", alwaysTrueTest);
   Aws::Map<Aws::String, Aws::String> outputs
=createCloudFormationStack(STACK_NAME,clientConfiguration);
   if (outputs.empty()) {
       std::cerr << "Exiting because createCloudFormationStack failed." <<
std::endl;
       cleanup(thingName, certificateARN, certificateID, "", "", false,
               clientConfiguration);
       return false;
   }

   // Retrieve the topic ARN and role ARN from the CloudFormation stack outputs.
   auto topicArnIter = outputs.find(SNS_TOPIC_ARN_OUTPUT);
   auto roleArnIter = outputs.find(ROLE_ARN_OUTPUT);
   if ((topicArnIter == outputs.end()) || (roleArnIter == outputs.end())) {
       std::cerr << "Exiting because output '" << SNS_TOPIC_ARN_OUTPUT <<
       "' or '" << ROLE_ARN_OUTPUT << "'not found in the CloudFormation stack."
<< std::endl;
       cleanup(thingName, certificateARN, certificateID, STACK_NAME, "",
               false,
```

```
                clientConfiguration);
        return false;
    }

    Aws::String topicArn = topicArnIter->second;
    Aws::String roleArn = roleArnIter->second;
    Aws::String sqlStatement = "SELECT * FROM '";
    sqlStatement += MQTT_MESSAGE_TOPIC_FILTER;
    sqlStatement += "'";

    printAsterisksLine();

    std::cout << "Now a rule will be created.\n" << std::endl;
    std::cout << "Rules are an administrator-level action. Any user who has
 permission\n"
                 << "to create rules will be able to access data processed by the
 rule." << std::endl;
    std::cout << "In this case, the rule will use an SNS topic" << std::endl;
    std::cout << "and the following SQL statement '" << sqlStatement << "'." <<
 std::endl;
    std::cout << "For more information on IoT SQL, see https://
docs.aws.amazon.com/iot/latest/developerguide/iot-sql-reference.html" <<
 std::endl;
    Aws::String ruleName = askQuestion("Enter a rule name: ");
    if (!createTopicRule(ruleName, topicArn, sqlStatement, roleArn,
 clientConfiguration)) {
        std::cerr << "Exiting because createRule failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, STACK_NAME, "",
                false,
                clientConfiguration);
        return false;
    }

    printAsterisksLine();

    std::cout << "Now your rules will be listed.\n" << std::endl;
    askQuestion("Press Enter to continue: ", alwaysTrueTest);
    if (!listTopicRules(clientConfiguration)) {
        std::cerr << "Exiting because listRules failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, STACK_NAME, ruleName,
                false,
                clientConfiguration);
        return false;
    }
```

```
    printAsterisksLine();
    Aws::String queryString = "thingName:" + thingName;
    std::cout << "Now the AWS IoT fleet index will be queried with the query\n'"
    << queryString << "'.\n" << std::endl;
    std::cout << "For query information, see https://docs.aws.amazon.com/iot/
latest/developerguide/query-syntax.html" << std::endl;

    std::cout << "For this query to work, thing indexing must be enabled in your
 account.\n"
    << "This can be done with the awscli command line by calling 'aws iot update-
indexing-configuration'\n"
       << "or it can be done programmatically." << std::endl;
    std::cout << "For more information, see https://docs.aws.amazon.com/iot/
latest/developerguide/managing-index.html" << std::endl;
    if (askYesNoQuestion("Do you want to enable thing indexing in your account?
 (y/n) "))
    {
        Aws::IoT::Model::ThingIndexingConfiguration thingIndexingConfiguration;

 thingIndexingConfiguration.SetThingIndexingMode(Aws::IoT::Model::ThingIndexingMode::REGI

 thingIndexingConfiguration.SetThingConnectivityIndexingMode(Aws::IoT::Model::ThingConnec
        // The ThingGroupIndexingConfiguration object is ignored if not set.
        Aws::IoT::Model::ThingGroupIndexingConfiguration
 thingGroupIndexingConfiguration;
        if (!updateIndexingConfiguration(thingIndexingConfiguration,
 thingGroupIndexingConfiguration, clientConfiguration)) {
            std::cerr << "Exiting because updateIndexingConfiguration failed." <<
 std::endl;
            cleanup(thingName, certificateARN, certificateID, STACK_NAME,
                    ruleName, false,
                    clientConfiguration);
            return false;
        }
    }

    if (!searchIndex(queryString, clientConfiguration)) {

        std::cerr << "Exiting because searchIndex failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, STACK_NAME, ruleName,
                false,
                clientConfiguration);
        return false;
```

```
    }
```

```
//! Update an AWS IoT thing with attributes.
/*!
  \param thingName: The name for the thing.
  \param attributeMap: A map of key/value attributes/
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::updateThing(const Aws::String &thingName,
                              const std::map<Aws::String, Aws::String>
 &attributeMap,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::UpdateThingRequest request;
    request.SetThingName(thingName);
    Aws::IoT::Model::AttributePayload attributePayload;
    for (const auto &attribute: attributeMap) {
        attributePayload.AddAttributes(attribute.first, attribute.second);
    }
    request.SetAttributePayload(attributePayload);

    Aws::IoT::Model::UpdateThingOutcome outcome = iotClient.UpdateThing(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to update thing " << thingName << ":" <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

//! Describe the endpoint specific to the AWS account making the call.
/*!
  \param endpointResult: String to receive the endpoint result.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::describeEndpoint(Aws::String &endpointResult,
```

```
                                        const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::String endpoint;
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DescribeEndpointRequest describeEndpointRequest;
    describeEndpointRequest.SetEndpointType(
            "iot:Data-ATS"); // Recommended endpoint type.

    Aws::IoT::Model::DescribeEndpointOutcome outcome =
 iotClient.DescribeEndpoint(
            describeEndpointRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully described endpoint." << std::endl;
        endpointResult = outcome.GetResult().GetEndpointAddress();
    }
    else {
        std::cerr << "Error describing endpoint" <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}

//! List certificates registered in the AWS account making the call.
/*!
   \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::listCertificates(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::ListCertificatesRequest request;

    Aws::Vector<Aws::IoT::Model::Certificate> allCertificates;
    Aws::String marker; // Used to paginate results.
    do {
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::IoT::Model::ListCertificatesOutcome outcome =
 iotClient.ListCertificates(
```

```
                request);

        if (outcome.IsSuccess()) {
            const Aws::IoT::Model::ListCertificatesResult &result =
 outcome.GetResult();
            marker = result.GetNextMarker();
            allCertificates.insert(allCertificates.end(),
                                   result.GetCertificates().begin(),
                                   result.GetCertificates().end());
        }
        else {
            std::cerr << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
            return false;
        }
    } while (!marker.empty());

    std::cout << allCertificates.size() << " certificate(s) found." << std::endl;

    for (auto &certificate: allCertificates) {
        std::cout << "Certificate ID: " << certificate.GetCertificateId() <<
 std::endl;
        std::cout << "Certificate ARN: " << certificate.GetCertificateArn()
                  << std::endl;
        std::cout << std::endl;
    }

    return true;
}

//! Update the shadow of an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param document: The state information, in JSON format.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::updateThingShadow(const Aws::String &thingName,
                                    const Aws::String &document,
                                    const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoTDataPlane::IoTDataPlaneClient
 iotDataPlaneClient(clientConfiguration);
    Aws::IoTDataPlane::Model::UpdateThingShadowRequest updateThingShadowRequest;
```

```
    updateThingShadowRequest.SetThingName(thingName);
    std::shared_ptr<std::stringstream> streamBuf =
 std::make_shared<std::stringstream>(
            document);
    updateThingShadowRequest.SetBody(streamBuf);
    Aws::IoTDataPlane::Model::UpdateThingShadowOutcome outcome =
 iotDataPlaneClient.UpdateThingShadow(
            updateThingShadowRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated thing shadow." << std::endl;
    }
    else {
        std::cerr << "Error while updating thing shadow."
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

//! Get the shadow of an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param documentResult: String to receive the state information, in JSON format.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::getThingShadow(const Aws::String &thingName,
                                 Aws::String &documentResult,
                                 const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoTDataPlane::IoTDataPlaneClient iotClient(clientConfiguration);
    Aws::IoTDataPlane::Model::GetThingShadowRequest request;
    request.SetThingName(thingName);
    auto outcome = iotClient.GetThingShadow(request);
    if (outcome.IsSuccess()) {
        std::stringstream ss;
        ss << outcome.GetResult().GetPayload().rdbuf();
        documentResult = ss.str();
    }
    else {
        std::cerr << "Error getting thing shadow: " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
```

```
    return outcome.IsSuccess();
}

//! Create an AWS IoT rule with an SNS topic as the target.
/*!
  \param ruleName: The name for the rule.
  \param snsTopic: The SNS topic ARN for the action.
  \param sql: The SQL statement used to query the topic.
  \param roleARN: The IAM role ARN for the action.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::IoT::createTopicRule(const Aws::String &ruleName,
                             const Aws::String &snsTopicARN, const Aws::String
 &sql,
                             const Aws::String &roleARN,
                             const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::CreateTopicRuleRequest request;
    request.SetRuleName(ruleName);

    Aws::IoT::Model::SnsAction snsAction;
    snsAction.SetTargetArn(snsTopicARN);
    snsAction.SetRoleArn(roleARN);

    Aws::IoT::Model::Action action;
    action.SetSns(snsAction);

    Aws::IoT::Model::TopicRulePayload topicRulePayload;
    topicRulePayload.SetSql(sql);
    topicRulePayload.SetActions({action});

    request.SetTopicRulePayload(topicRulePayload);
    auto outcome = iotClient.CreateTopicRule(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created topic rule " << ruleName << "." <<
 std::endl;
    }
    else {
        std::cerr << "Error creating topic rule " << ruleName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
```

```
    }
    return outcome.IsSuccess();
}

//! Lists the AWS IoT topic rules.
/*!
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::listTopicRules(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::ListTopicRulesRequest request;

    Aws::Vector<Aws::IoT::Model::TopicRuleListItem> allRules;
    Aws::String nextToken; // Used for pagination.
    do {
        if (!nextToken.empty()) {
            request.SetNextToken(nextToken);
        }

        Aws::IoT::Model::ListTopicRulesOutcome outcome =
 iotClient.ListTopicRules(
                request);

        if (outcome.IsSuccess()) {
            const Aws::IoT::Model::ListTopicRulesResult &result =
 outcome.GetResult();
            allRules.insert(allRules.end(),
                            result.GetRules().cbegin(),
                            result.GetRules().cend());

            nextToken = result.GetNextToken();
        }
        else {
            std::cerr << "ListTopicRules error: " <<
                      outcome.GetError().GetMessage() << std::endl;
            return false;
        }

    } while (!nextToken.empty());

    std::cout << "ListTopicRules: " << allRules.size() << " rule(s) found."
              << std::endl;
```

```
    for (auto &rule: allRules) {
        std::cout << "  Rule name: " << rule.GetRuleName() << ", rule ARN: "
                  << rule.GetRuleArn() << "." << std::endl;
    }

    return true;
}

//! Query the AWS IoT fleet index.
//! For query information, see https://docs.aws.amazon.com/iot/latest/
developerguide/query-syntax.html
/*!
  \param: query: The query string.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::searchIndex(const Aws::String &query,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::SearchIndexRequest request;
    request.SetQueryString(query);

    Aws::Vector<Aws::IoT::Model::ThingDocument> allThingDocuments;
    Aws::String nextToken; // Used for pagination.
    do {
        if (!nextToken.empty()) {
            request.SetNextToken(nextToken);
        }

        Aws::IoT::Model::SearchIndexOutcome outcome =
 iotClient.SearchIndex(request);

        if (outcome.IsSuccess()) {
            const Aws::IoT::Model::SearchIndexResult &result =
 outcome.GetResult();
            allThingDocuments.insert(allThingDocuments.end(),
                                     result.GetThings().cbegin(),
                                     result.GetThings().cend());
            nextToken = result.GetNextToken();

        }
        else {
```

```
            std::cerr << "Error in SearchIndex: " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!nextToken.empty());

    std::cout << allThingDocuments.size() << " thing document(s) found." <<
 std::endl;
    for (const auto thingDocument: allThingDocuments) {
        std::cout << "  Thing name: " << thingDocument.GetThingName() << "."
                  << std::endl;
    }
    return true;
}
```

Bereinigen von Ressourcen.

```
bool
AwsDoc::IoT::cleanup(const Aws::String &thingName, const Aws::String
 &certificateARN,
                     const Aws::String &certificateID, const Aws::String
 &stackName,
                     const Aws::String &ruleName, bool askForConfirmation,
                     const Aws::Client::ClientConfiguration &clientConfiguration)
 {
    bool result = true;

    if (!ruleName.empty() && (!askForConfirmation ||
                              askYesNoQuestion("Delete the rule '" + ruleName +
                                               "'? (y/n) "))) {
        result &= deleteTopicRule(ruleName, clientConfiguration);
    }

    Aws::CloudFormation::CloudFormationClient
 cloudFormationClient(clientConfiguration);

    if (!stackName.empty() && (!askForConfirmation ||
                               askYesNoQuestion(
                                       "Delete the CloudFormation stack '" +
 stackName +
                                       "'? (y/n) "))) {
```

```
        result &= deleteStack(stackName, clientConfiguration);
    }

    if (!certificateARN.empty() && (!askForConfirmation ||
                                    askYesNoQuestion("Delete the certificate '" +
                                                     certificateARN + "'? (y/n)
 "))) {
        result &= detachThingPrincipal(certificateARN, thingName,
 clientConfiguration);
        result &= deleteCertificate(certificateID, clientConfiguration);
    }

    if (!thingName.empty() && (!askForConfirmation ||
                               askYesNoQuestion("Delete the thing '" + thingName
 +
                                                "'? (y/n) "))) {
        result &= deleteThing(thingName, clientConfiguration);
    }

    return result;
}
```

```
//! Detach a principal from an AWS IoT thing.
/*!
  \param principal: A principal to detach.
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::detachThingPrincipal(const Aws::String &principal,
                                       const Aws::String &thingName,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::DetachThingPrincipalRequest detachThingPrincipalRequest;
    detachThingPrincipalRequest.SetThingName(thingName);
    detachThingPrincipalRequest.SetPrincipal(principal);

    Aws::IoT::Model::DetachThingPrincipalOutcome outcome =
 iotClient.DetachThingPrincipal(
            detachThingPrincipalRequest);
```

```
    if (outcome.IsSuccess()) {
        std::cout << "Successfully detached principal " << principal << " from
 thing "
                  << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to detach principal " << principal << " from thing "
                  << thingName << ": "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

//! Delete a certificate.
/*!
  \param certificateID: The ID of a certificate.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteCertificate(const Aws::String &certificateID,
                                    const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::DeleteCertificateRequest request;
    request.SetCertificateId(certificateID);

    Aws::IoT::Model::DeleteCertificateOutcome outcome =
 iotClient.DeleteCertificate(
            request);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted certificate " << certificateID <<
 std::endl;
    }
    else {
        std::cerr << "Error deleting certificate " << certificateID << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

```
//! Delete an AWS IoT rule.
/*!
  \param ruleName: The name for the rule.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteTopicRule(const Aws::String &ruleName,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DeleteTopicRuleRequest request;
    request.SetRuleName(ruleName);

    Aws::IoT::Model::DeleteTopicRuleOutcome outcome = iotClient.DeleteTopicRule(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted rule " << ruleName << std::endl;
    }
    else {
        std::cerr << "Failed to delete rule " << ruleName <<
                  ": " << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

//! Delete an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteThing(const Aws::String &thingName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DeleteThingRequest request;
    request.SetThingName(thingName);
    const auto outcome = iotClient.DeleteThing(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted thing " << thingName << std::endl;
    }
    else {
```

```
        std::cerr << "Error deleting thing " << thingName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für C++ -API-Referenz.
  - AttachThingPrincipal
  - CreateKeysAndCertificate
  - CreateThing
  - CreateTopicRule
  - DeleteCertificate
  - DeleteThing
  - DeleteTopicRule
  - DescribeEndpoint
  - DescribeThing
  - DetachThingPrincipal
  - ListCertificates
  - ListThings
  - SearchIndex
  - UpdateIndexingConfiguration
  - UpdateThing

Java

SDK für Java 2.x

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario aus, das die AWS IoT Funktionen demonstriert.

```
import java.util.Scanner;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * This Java example performs these tasks:
 *
 * 1. Creates an AWS IoT Thing.
 * 2. Generate and attach a device certificate.
 * 3. Update an AWS IoT Thing with Attributes.
 * 4. Get an AWS IoT Endpoint.
 * 5. List your certificates.
 * 6. Updates the shadow for the specified thing..
 * 7. Write out the state information, in JSON format
 * 8. Creates a rule
 * 9. List rules
 * 10. Search things
 * 11. Detach amd delete the certificate.
 * 12. Delete Thing.
 */
public class IotScenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) {
        final String usage =
            """
                Usage:
                    <roleARN> <snsAction>

                Where:
                    roleARN - The ARN of an IAM role that has permission to work
 with AWS IOT.
                    snsAction  - An ARN of an SNS topic.
                """;
```

```
        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        IotActions iotActions = new IotActions();
        String thingName;
        String ruleName;
        String roleARN = args[0];
        String snsAction = args[1];
        Scanner scanner = new Scanner(System.in);

        System.out.println(DASHES);
        System.out.println("Welcome to the AWS IoT basics scenario.");
        System.out.println("""
            This example program demonstrates various interactions with the AWS
Internet of Things (IoT) Core service. The program guides you through a series
of steps,
            including creating an IoT Thing, generating a device certificate,
updating the Thing with attributes, and so on.
            It utilizes the AWS SDK for Java V2 and incorporates functionality
for creating and managing IoT Things, certificates, rules,
            shadows, and performing searches. The program aims to showcase AWS
IoT capabilities and provides a comprehensive example for
            developers working with AWS IoT in a Java environment.

            Let's get started...

            """);
        System.out.println(DASHES);

        System.out.println("1. Create an AWS IoT Thing.");
        System.out.println("""
            An AWS IoT Thing represents a virtual entity in the AWS IoT service
that can be associated with
            a physical device.
            """);
        // Prompt the user for input.
        System.out.print("Enter Thing name: ");
        thingName = scanner.nextLine();
        iotActions.createIoTThing(thingName);
        System.out.println(DASHES);

        System.out.println(DASHES);
```

```
        System.out.println("2. Generate a device certificate.");
        System.out.println("""
            A device certificate performs a role in securing the communication
between devices (Things)
            and the AWS IoT platform.
            """);

        System.out.print("Do you want to create a certificate for " +thingName
+"? (y/n)");
        String certAns = scanner.nextLine();
        String certificateArn="" ;
        if (certAns != null && certAns.trim().equalsIgnoreCase("y")) {
            certificateArn = iotActions.createCertificate();
            System.out.println("Attach the certificate to the AWS IoT Thing.");
            iotActions.attachCertificateToThing(thingName, certificateArn);
        } else {
            System.out.println("A device certificate was not created.");
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Update an AWS IoT Thing with Attributes.");
        System.out.println("""
             IoT Thing attributes, represented as key-value pairs, offer a
pivotal advantage in facilitating efficient data
             management and retrieval within the AWS IoT ecosystem.
            """);
        waitForInputToContinue(scanner);
        iotActions.updateShadowThing(thingName);
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Return a unique endpoint specific to the Amazon
Web Services account.");
        System.out.println("""
            An IoT Endpoint refers to a specific URL or Uniform Resource Locator
that serves as the entry point for communication between IoT devices and the AWS
IoT service.
          """);
        waitForInputToContinue(scanner);
        String endpointUrl = iotActions.describeEndpoint();
        System.out.println("The endpoint is "+endpointUrl);
        waitForInputToContinue(scanner);
```

```
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. List your AWS IoT certificates");
        waitForInputToContinue(scanner);
        if (certificateArn.length() > 0) {
            iotActions.listCertificates();
        } else {
            System.out.println("You did not create a certificates. Skipping this
step.");
        }
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Create an IoT shadow that refers to a digital
representation or virtual twin of a physical IoT device");
        System.out.println("""
            A Thing Shadow refers to a feature that enables you to create a
virtual representation, or "shadow,"
            of a physical device or thing. The Thing Shadow allows you to
synchronize and control the state of a device between
            the cloud and the device itself. and the AWS IoT service. For
example, you can write and retrieve JSON data from a Thing Shadow.
          """);
        waitForInputToContinue(scanner);
        iotActions.updateShadowThing(thingName);
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Write out the state information, in JSON
format.");
        waitForInputToContinue(scanner);
        iotActions.getPayload(thingName);
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Creates a rule");
        System.out.println("""
        Creates a rule that is an administrator-level action.
        Any user who has permission to create rules will be able to access data
processed by the rule.
```

```
        """);
        System.out.print("Enter Rule name: ");
        ruleName = scanner.nextLine();
        iotActions.createIoTRule(roleARN, ruleName, snsAction);
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. List your rules.");
        waitForInputToContinue(scanner);
        iotActions.listIoTRules();
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("10. Search things using the Thing name.");
        waitForInputToContinue(scanner);
        String queryString = "thingName:"+thingName ;
        iotActions.searchThings(queryString);
        waitForInputToContinue(scanner);
        System.out.println(DASHES);

        System.out.println(DASHES);
        if (certificateArn.length() > 0) {
            System.out.print("Do you want to detach and delete the certificate
for " +thingName +"? (y/n)");
            String delAns = scanner.nextLine();
            if (delAns != null && delAns.trim().equalsIgnoreCase("y")) {
                System.out.println("11. You selected to detach amd delete the
certificate.");
                waitForInputToContinue(scanner);
                iotActions.detachThingPrincipal(thingName, certificateArn);
                iotActions.deleteCertificate(certificateArn);
                waitForInputToContinue(scanner);
            } else {
                System.out.println("11. You selected not to delete the
certificate.");
            }
        } else {
            System.out.println("11. You did not create a certificate so there is
nothing to delete.");
        }
        System.out.println(DASHES);
```

```
        System.out.println(DASHES);
        System.out.println("12. Delete the AWS IoT Thing.");
        System.out.print("Do you want to delete the IoT Thing? (y/n)");
        String delAns = scanner.nextLine();
        if (delAns != null && delAns.trim().equalsIgnoreCase("y")) {
            iotActions.deleteIoTThing(thingName);
        } else {
            System.out.println("The IoT Thing was not deleted.");
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The AWS IoT workflow has successfully completed.");
        System.out.println(DASHES);
    }


    private static void waitForInputToContinue(Scanner scanner) {
        while (true) {
            System.out.println("");
            System.out.println("Enter 'c' followed by <ENTER> to continue:");
            String input = scanner.nextLine();

            if (input.trim().equalsIgnoreCase("c")) {
                System.out.println("Continuing with the program...");
                System.out.println("");
                break;
            } else {
                // Handle invalid input.
                System.out.println("Invalid input. Please try again.");
            }
        }
    }
}
```

Eine Wrapper-Klasse für AWS IoT SDK-Methoden.

```
import
 software.amazon.awssdk.auth.credentials.EnvironmentVariableCredentialsProvider;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.core.client.config.ClientOverrideConfiguration;
import software.amazon.awssdk.core.retry.RetryPolicy;
```

```
import software.amazon.awssdk.http.async.SdkAsyncHttpClient;
import software.amazon.awssdk.http.nio.netty.NettyNioAsyncHttpClient;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iot.IotAsyncClient;
import software.amazon.awssdk.services.iot.model.Action;
import software.amazon.awssdk.services.iot.model.AttachThingPrincipalRequest;
import software.amazon.awssdk.services.iot.model.AttachThingPrincipalResponse;
import software.amazon.awssdk.services.iot.model.Certificate;
import
 software.amazon.awssdk.services.iot.model.CreateKeysAndCertificateResponse;
import software.amazon.awssdk.services.iot.model.CreateThingRequest;
import software.amazon.awssdk.services.iot.model.CreateThingResponse;
import software.amazon.awssdk.services.iot.model.CreateTopicRuleRequest;
import software.amazon.awssdk.services.iot.model.CreateTopicRuleResponse;
import software.amazon.awssdk.services.iot.model.DeleteCertificateRequest;
import software.amazon.awssdk.services.iot.model.DeleteCertificateResponse;
import software.amazon.awssdk.services.iot.model.DeleteThingRequest;
import software.amazon.awssdk.services.iot.model.DeleteThingResponse;
import software.amazon.awssdk.services.iot.model.DescribeEndpointRequest;
import software.amazon.awssdk.services.iot.model.DescribeEndpointResponse;
import software.amazon.awssdk.services.iot.model.DescribeThingRequest;
import software.amazon.awssdk.services.iot.model.DescribeThingResponse;
import software.amazon.awssdk.services.iot.model.DetachThingPrincipalRequest;
import software.amazon.awssdk.services.iot.model.DetachThingPrincipalResponse;
import software.amazon.awssdk.services.iot.model.IotException;
import software.amazon.awssdk.services.iot.model.ListCertificatesResponse;
import software.amazon.awssdk.services.iot.model.ListTopicRulesRequest;
import software.amazon.awssdk.services.iot.model.ListTopicRulesResponse;
import software.amazon.awssdk.services.iot.model.SearchIndexRequest;
import software.amazon.awssdk.services.iot.model.SearchIndexResponse;
import software.amazon.awssdk.services.iot.model.TopicRuleListItem;
import software.amazon.awssdk.services.iot.model.SnsAction;
import software.amazon.awssdk.services.iot.model.TopicRulePayload;
import software.amazon.awssdk.services.iotdataplane.IotDataPlaneAsyncClient;
import software.amazon.awssdk.services.iotdataplane.model.GetThingShadowRequest;
import software.amazon.awssdk.services.iotdataplane.model.GetThingShadowResponse;
import
 software.amazon.awssdk.services.iotdataplane.model.UpdateThingShadowRequest;
import
 software.amazon.awssdk.services.iotdataplane.model.UpdateThingShadowResponse;
import java.nio.charset.StandardCharsets;
import java.time.Duration;
import java.util.List;
import java.util.concurrent.CompletableFuture;
```

```
import java.util.concurrent.CompletionException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class IotActions {

    private static IotAsyncClient iotAsyncClient;

    private static IotDataPlaneAsyncClient iotAsyncDataPlaneClient;

    private static final String TOPIC = "your-iot-topic";

    private static IotDataPlaneAsyncClient getAsyncDataPlaneClient() {
        SdkAsyncHttpClient httpClient = NettyNioAsyncHttpClient.builder()
            .maxConcurrency(100)
            .connectionTimeout(Duration.ofSeconds(60))
            .readTimeout(Duration.ofSeconds(60))
            .writeTimeout(Duration.ofSeconds(60))
            .build();

        ClientOverrideConfiguration overrideConfig =
 ClientOverrideConfiguration.builder()
            .apiCallTimeout(Duration.ofMinutes(2))
            .apiCallAttemptTimeout(Duration.ofSeconds(90))
            .retryPolicy(RetryPolicy.builder()
                .numRetries(3)
                .build())
            .build();

        if (iotAsyncDataPlaneClient == null) {
            iotAsyncDataPlaneClient = IotDataPlaneAsyncClient.builder()
                .region(Region.US_EAST_1)
                .httpClient(httpClient)
                .overrideConfiguration(overrideConfig)
                .build();
        }
        return iotAsyncDataPlaneClient;
    }


    private static IotAsyncClient getAsyncClient() {
        SdkAsyncHttpClient httpClient = NettyNioAsyncHttpClient.builder()
            .maxConcurrency(100)
            .connectionTimeout(Duration.ofSeconds(60))
```

```
            .readTimeout(Duration.ofSeconds(60))
            .writeTimeout(Duration.ofSeconds(60))
            .build();

        ClientOverrideConfiguration overrideConfig =
ClientOverrideConfiguration.builder()
            .apiCallTimeout(Duration.ofMinutes(2))
            .apiCallAttemptTimeout(Duration.ofSeconds(90))
            .retryPolicy(RetryPolicy.builder()
                .numRetries(3)
                .build())
            .build();

        if (iotAsyncClient == null) {
            iotAsyncClient = IotAsyncClient.builder()
                .region(Region.US_EAST_1)
                .httpClient(httpClient)
                .overrideConfiguration(overrideConfig)
                .build();
        }
        return iotAsyncClient;
    }

    /**
     * Creates an IoT certificate asynchronously.
     *
     * @return The ARN of the created certificate.
     * <p>
     * This method initiates an asynchronous request to create an IoT
certificate.
     * If the request is successful, it prints the certificate details and
returns the certificate ARN.
     * If an exception occurs, it prints the error message.
     */
    public String createCertificate() {
        CompletableFuture<CreateKeysAndCertificateResponse> future =
getAsyncClient().createKeysAndCertificate();
        final String[] certificateArn = {null};
        future.whenComplete((response, ex) -> {
            if (response != null) {
                String certificatePem = response.certificatePem();
                certificateArn[0] = response.certificateArn();

                // Print the details.
```

```
                System.out.println("\nCertificate:");
                System.out.println(certificatePem);
                System.out.println("\nCertificate ARN:");
                System.out.println(certificateArn[0]);

            } else {
                Throwable cause = (ex instanceof CompletionException) ?
ex.getCause() : ex;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                }
            }
        });

        future.join();
        return certificateArn[0];
    }

    /**
     * Creates an IoT Thing with the specified name asynchronously.
     *
     * @param thingName The name of the IoT Thing to create.
     *
     * This method initiates an asynchronous request to create an IoT Thing with
the specified name.
     * If the request is successful, it prints the name of the thing and its ARN
value.
     * If an exception occurs, it prints the error message.
     */
    public void createIoTThing(String thingName) {
        CreateThingRequest createThingRequest = CreateThingRequest.builder()
            .thingName(thingName)
            .build();

        CompletableFuture<CreateThingResponse> future =
getAsyncClient().createThing(createThingRequest);
        future.whenComplete((createThingResponse, ex) -> {
            if (createThingResponse != null) {
                System.out.println(thingName + " was successfully created. The
ARN value is " + createThingResponse.thingArn());
```

```
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                }
            }
        });

        future.join();
    }

    /**
     * Attaches a certificate to an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing.
     * @param certificateArn The ARN of the certificate to attach.
     *
     * This method initiates an asynchronous request to attach a certificate to
an IoT Thing.
     * If the request is successful, it prints a confirmation message and
additional information about the Thing.
     * If an exception occurs, it prints the error message.
     */
    public void attachCertificateToThing(String thingName, String certificateArn)
{
        AttachThingPrincipalRequest principalRequest =
AttachThingPrincipalRequest.builder()
            .thingName(thingName)
            .principal(certificateArn)
            .build();

        CompletableFuture<AttachThingPrincipalResponse> future =
getAsyncClient().attachThingPrincipal(principalRequest);
        future.whenComplete((attachResponse, ex) -> {
            if (attachResponse != null &&
attachResponse.sdkHttpResponse().isSuccessful()) {
                System.out.println("Certificate attached to Thing
successfully.");

                // Print additional information about the Thing.
```

```
                describeThing(thingName);
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to attach certificate to Thing.
HTTP Status Code: " +
                        attachResponse.sdkHttpResponse().statusCode());
                }
            }
        });

        future.join();
    }

    /**
     * Describes an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing.
     *
     * This method initiates an asynchronous request to describe an IoT Thing.
     * If the request is successful, it prints the Thing details.
     * If an exception occurs, it prints the error message.
     */
    private void describeThing(String thingName) {
        DescribeThingRequest thingRequest = DescribeThingRequest.builder()
            .thingName(thingName)
            .build();

        CompletableFuture<DescribeThingResponse> future =
getAsyncClient().describeThing(thingRequest);
        future.whenComplete((describeResponse, ex) -> {
            if (describeResponse != null) {
                System.out.println("Thing Details:");
                System.out.println("Thing Name: " +
describeResponse.thingName());
                System.out.println("Thing ARN: " + describeResponse.thingArn());
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
```

```
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to describe Thing.");
                }
            }
        });

        future.join();
    }

    /**
     * Updates the shadow of an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing.
     *
     * This method initiates an asynchronous request to update the shadow of an
IoT Thing.
     * If the request is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void updateShadowThing(String thingName) {
        // Create Thing Shadow State Document.
        String stateDocument = "{\"state\":{\"reported\":{\"temperature\":25,
\"humidity\":50}}}";
        SdkBytes data = SdkBytes.fromString(stateDocument,
StandardCharsets.UTF_8);
        UpdateThingShadowRequest updateThingShadowRequest =
UpdateThingShadowRequest.builder()
            .thingName(thingName)
            .payload(data)
            .build();

        CompletableFuture<UpdateThingShadowResponse> future =
getAsyncDataPlaneClient().updateThingShadow(updateThingShadowRequest);
        future.whenComplete((updateResponse, ex) -> {
            if (updateResponse != null) {
                System.out.println("Thing Shadow updated successfully.");
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
```

```
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
 cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
 cause.getMessage());
                } else {
                    System.err.println("Failed to update Thing Shadow.");
                }
            }
        });

        future.join();
    }

    /**
     * Describes the endpoint of the IoT service asynchronously.
     *
     * @return A CompletableFuture containing the full endpoint URL.
     *
     * This method initiates an asynchronous request to describe the endpoint of
 the IoT service.
     * If the request is successful, it prints and returns the full endpoint URL.
     * If an exception occurs, it prints the error message.
     */
    public String describeEndpoint() {
        CompletableFuture<DescribeEndpointResponse> future =
 getAsyncClient().describeEndpoint(DescribeEndpointRequest.builder().endpointType("iot:Da
ATS").build());
        final String[] result = {null};

        future.whenComplete((endpointResponse, ex) -> {
            if (endpointResponse != null) {
                String endpointUrl = endpointResponse.endpointAddress();
                String exString = getValue(endpointUrl);
                String fullEndpoint = "https://" + exString + "-ats.iot.us-
east-1.amazonaws.com";

                System.out.println("Full Endpoint URL: " + fullEndpoint);
                result[0] = fullEndpoint;
            } else {
                Throwable cause = (ex instanceof CompletionException) ?
 ex.getCause() : ex;
                if (cause instanceof IotException) {
```

```
                    System.err.println(((IotException)
 cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " +
 cause.getMessage());
                }
            }
        });

        future.join();
        return result[0];
    }

    /**
     * Extracts a specific value from the endpoint URL.
     *
     * @param input The endpoint URL to process.
     * @return The extracted value from the endpoint URL.
     */
    private static String getValue(String input) {
        // Define a regular expression pattern for extracting the subdomain.
        Pattern pattern = Pattern.compile("^(.*?)\\.iot\\.us-east-1\\.amazonaws\
\.com");

        // Match the pattern against the input string.
        Matcher matcher = pattern.matcher(input);

        // Check if a match is found.
        if (matcher.find()) {
            // Extract the subdomain from the first capturing group.
            String subdomain = matcher.group(1);
            System.out.println("Extracted subdomain: " + subdomain);
            return subdomain ;
        } else {
            System.out.println("No match found");
        }
        return "" ;
    }

    /**
     * Lists all certificates asynchronously.
     *
     * This method initiates an asynchronous request to list all certificates.
     * If the request is successful, it prints the certificate IDs and ARNs.
```

```
     * If an exception occurs, it prints the error message.
     */
    public void listCertificates() {
        CompletableFuture<ListCertificatesResponse> future =
getAsyncClient().listCertificates();
        future.whenComplete((response, ex) -> {
            if (response != null) {
                List<Certificate> certList = response.certificates();
                for (Certificate cert : certList) {
                    System.out.println("Cert id: " + cert.certificateId());
                    System.out.println("Cert Arn: " + cert.certificateArn());
                }
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to list certificates.");
                }
            }
        });

        future.join();
    }

    /**
     * Retrieves the payload of a Thing's shadow asynchronously.
     *
     * @param thingName The name of the IoT Thing.
     *
     * This method initiates an asynchronous request to get the payload of a
Thing's shadow.
     * If the request is successful, it prints the shadow data.
     * If an exception occurs, it prints the error message.
     */
    public void getPayload(String thingName) {
        GetThingShadowRequest getThingShadowRequest =
GetThingShadowRequest.builder()
            .thingName(thingName)
            .build();
```

```
        CompletableFuture<GetThingShadowResponse> future =
getAsyncDataPlaneClient().getThingShadow(getThingShadowRequest);
        future.whenComplete((getThingShadowResponse, ex) -> {
            if (getThingShadowResponse != null) {
                // Extracting payload from response.
                SdkBytes payload = getThingShadowResponse.payload();
                String payloadString = payload.asUtf8String();
                System.out.println("Received Shadow Data: " + payloadString);
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to get Thing Shadow payload.");
                }
            }
        });

        future.join();
    }

    /**
     * Creates an IoT rule asynchronously.
     *
     * @param roleARN The ARN of the IAM role that grants access to the rule's
actions.
     * @param ruleName The name of the IoT rule.
     * @param action The ARN of the action to perform when the rule is triggered.
     *
     * This method initiates an asynchronous request to create an IoT rule.
     * If the request is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void createIoTRule(String roleARN, String ruleName, String action) {
        String sql = "SELECT * FROM '" + TOPIC + "'";
        SnsAction action1 = SnsAction.builder()
            .targetArn(action)
            .roleArn(roleARN)
            .build();
```

```
        // Create the action.
        Action myAction = Action.builder()
            .sns(action1)
            .build();

        // Create the topic rule payload.
        TopicRulePayload topicRulePayload = TopicRulePayload.builder()
            .sql(sql)
            .actions(myAction)
            .build();

        // Create the topic rule request.
        CreateTopicRuleRequest topicRuleRequest =
CreateTopicRuleRequest.builder()
            .ruleName(ruleName)
            .topicRulePayload(topicRulePayload)
            .build();

        CompletableFuture<CreateTopicRuleResponse> future =
getAsyncClient().createTopicRule(topicRuleRequest);
        future.whenComplete((response, ex) -> {
            if (response != null) {
                System.out.println("IoT Rule created successfully.");
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to create IoT Rule.");
                }
            }
        });

        future.join();
    }

    /**
     * Lists IoT rules asynchronously.
     *
```

```
    * This method initiates an asynchronous request to list IoT rules.
    * If the request is successful, it prints the names and ARNs of the rules.
    * If an exception occurs, it prints the error message.
    */
   public void listIoTRules() {
       ListTopicRulesRequest listTopicRulesRequest =
ListTopicRulesRequest.builder().build();
       CompletableFuture<ListTopicRulesResponse> future =
getAsyncClient().listTopicRules(listTopicRulesRequest);
       future.whenComplete((listTopicRulesResponse, ex) -> {
           if (listTopicRulesResponse != null) {
               System.out.println("List of IoT Rules:");
               List<TopicRuleListItem> ruleList =
listTopicRulesResponse.rules();
               for (TopicRuleListItem rule : ruleList) {
                   System.out.println("Rule Name: " + rule.ruleName());
                   System.out.println("Rule ARN: " + rule.ruleArn());
                   System.out.println("--------------");
               }
           } else {
               Throwable cause = ex != null ? ex.getCause() : null;
               if (cause instanceof IotException) {
                   System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
               } else if (cause != null) {
                   System.err.println("Unexpected error: " +
cause.getMessage());
               } else {
                   System.err.println("Failed to list IoT Rules.");
               }
           }
       });

       future.join();
   }

   /**
    * Searches for IoT Things asynchronously based on a query string.
    *
    * @param queryString The query string to search for Things.
    *
    * This method initiates an asynchronous request to search for IoT Things.
    * If the request is successful and Things are found, it prints their IDs.
    * If no Things are found, it prints a message indicating so.
```

```
     * If an exception occurs, it prints the error message.
     */
    public void searchThings(String queryString) {
        SearchIndexRequest searchIndexRequest = SearchIndexRequest.builder()
            .queryString(queryString)
            .build();

        CompletableFuture<SearchIndexResponse> future =
getAsyncClient().searchIndex(searchIndexRequest);
        future.whenComplete((searchIndexResponse, ex) -> {
            if (searchIndexResponse != null) {
                // Process the result.
                if (searchIndexResponse.things().isEmpty()) {
                    System.out.println("No things found.");
                } else {
                    searchIndexResponse.things().forEach(thing ->
System.out.println("Thing id found using search is " + thing.thingId()));
                }
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to search for IoT Things.");
                }
            }
        });

        future.join();
    }

    /**
     * Detaches a principal (certificate) from an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing.
     * @param certificateArn The ARN of the certificate to detach.
     *
     * This method initiates an asynchronous request to detach a certificate from
an IoT Thing.
     * If the detachment is successful, it prints a confirmation message.
```

```
     * If an exception occurs, it prints the error message.
     */
    public void detachThingPrincipal(String thingName, String certificateArn) {
        DetachThingPrincipalRequest thingPrincipalRequest =
DetachThingPrincipalRequest.builder()
            .principal(certificateArn)
            .thingName(thingName)
            .build();

        CompletableFuture<DetachThingPrincipalResponse> future =
getAsyncClient().detachThingPrincipal(thingPrincipalRequest);
        future.whenComplete((voidResult, ex) -> {
            if (ex == null) {
                System.out.println(certificateArn + " was successfully removed
from " + thingName);
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " + ex.getMessage());
                }
            }
        });

        future.join();
    }

    /**
     * Deletes a certificate asynchronously.
     *
     * @param certificateArn The ARN of the certificate to delete.
     *
     * This method initiates an asynchronous request to delete a certificate.
     * If the deletion is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void deleteCertificate(String certificateArn) {
        DeleteCertificateRequest certificateProviderRequest =
DeleteCertificateRequest.builder()
            .certificateId(extractCertificateId(certificateArn))
            .build();
```

```
        CompletableFuture<DeleteCertificateResponse> future =
getAsyncClient().deleteCertificate(certificateProviderRequest);
        future.whenComplete((voidResult, ex) -> {
            if (ex == null) {
                System.out.println(certificateArn + " was successfully
deleted.");
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " + ex.getMessage());
                }
            }
        });

        future.join();
    }

    /**
     * Deletes an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing to delete.
     *
     * This method initiates an asynchronous request to delete an IoT Thing.
     * If the deletion is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void deleteIoTThing(String thingName) {
        DeleteThingRequest deleteThingRequest = DeleteThingRequest.builder()
            .thingName(thingName)
            .build();

        CompletableFuture<DeleteThingResponse> future =
getAsyncClient().deleteThing(deleteThingRequest);
        future.whenComplete((voidResult, ex) -> {
            if (ex == null) {
                System.out.println("Deleted Thing " + thingName);
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
```

```
                } else {
                    System.err.println("Unexpected error: " + ex.getMessage());
                }
            }
        });

        future.join();
    }

    // Get the cert Id  from the Cert ARN value.
    private String extractCertificateId(String certificateArn) {
        // Example ARN: arn:aws:iot:region:account-id:cert/certificate-id.
        String[] arnParts = certificateArn.split(":");
        String certificateIdPart = arnParts[arnParts.length - 1];
        return certificateIdPart.substring(certificateIdPart.lastIndexOf("/") +
 1);
    }
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK for Java 2.x - API-Referenz.
  - AttachThingPrincipal
  - CreateKeysAndCertificate
  - CreateThing
  - CreateTopicRule
  - DeleteCertificate
  - DeleteThing
  - DeleteTopicRule
  - DescribeEndpoint
  - DescribeThing
  - DetachThingPrincipal
  - ListCertificates
  - ListThings
  - SearchIndex
  - UpdateIndexingConfiguration
  - UpdateThing

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
import aws.sdk.kotlin.services.iot.IotClient
import aws.sdk.kotlin.services.iot.model.Action
import aws.sdk.kotlin.services.iot.model.AttachThingPrincipalRequest
import aws.sdk.kotlin.services.iot.model.AttributePayload
import aws.sdk.kotlin.services.iot.model.CreateThingRequest
import aws.sdk.kotlin.services.iot.model.CreateTopicRuleRequest
import aws.sdk.kotlin.services.iot.model.DeleteCertificateRequest
import aws.sdk.kotlin.services.iot.model.DeleteThingRequest
import aws.sdk.kotlin.services.iot.model.DescribeEndpointRequest
import aws.sdk.kotlin.services.iot.model.DescribeThingRequest
import aws.sdk.kotlin.services.iot.model.DetachThingPrincipalRequest
import aws.sdk.kotlin.services.iot.model.ListTopicRulesRequest
import aws.sdk.kotlin.services.iot.model.SearchIndexRequest
import aws.sdk.kotlin.services.iot.model.SnsAction
import aws.sdk.kotlin.services.iot.model.TopicRulePayload
import aws.sdk.kotlin.services.iot.model.UpdateThingRequest
import aws.sdk.kotlin.services.iotdataplane.IotDataPlaneClient
import aws.sdk.kotlin.services.iotdataplane.model.GetThingShadowRequest
import aws.sdk.kotlin.services.iotdataplane.model.UpdateThingShadowRequest
import aws.smithy.kotlin.runtime.content.ByteStream
import aws.smithy.kotlin.runtime.content.toByteArray
import java.util.Scanner
import java.util.regex.Pattern
import kotlin.system.exitProcess

/**
 * Before running this Kotlin code example, ensure that your development
 environment
 * is set up, including configuring your credentials.
 *
 * For detailed instructions, refer to the following documentation topic:
```

```
 * [Setting Up Your Development Environment](https://docs.aws.amazon.com/sdk-for-
kotlin/latest/developer-guide/setup.html)
 *
 * This code example requires an SNS topic and an IAM Role.
 * Follow the steps in the documentation to set up these resources:
 *
 * - [Creating an SNS Topic](https://docs.aws.amazon.com/sns/latest/dg/sns-
getting-started.html#step-create-topic)
 * - [Creating an IAM Role](https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_create.html)
 */

val DASHES = String(CharArray(80)).replace("\u0000", "-")
val TOPIC = "your-iot-topic"

suspend fun main(args: Array<String>) {
    val usage =
        """
        Usage:
            <roleARN> <snsAction>

        Where:
            roleARN - The ARN of an IAM role that has permission to work with AWS
 IOT.
            snsAction  - An ARN of an SNS topic.

        """.trimIndent()

    if (args.size != 2) {
        println(usage)
        exitProcess(1)
    }

    var thingName: String
    val roleARN = args[0]
    val snsAction = args[1]
    val scanner = Scanner(System.`in`)

    println(DASHES)
    println("Welcome to the AWS IoT example scenario.")
    println(
        """
        This example program demonstrates various interactions with the AWS
 Internet of Things (IoT) Core service.
```

```
        The program guides you through a series of steps, including creating an
IoT thing, generating a device certificate,
        updating the thing with attributes, and so on.

        It utilizes the AWS SDK for Kotlin and incorporates functionality for
creating and managing IoT things, certificates, rules,
        shadows, and performing searches. The program aims to showcase AWS IoT
capabilities and provides a comprehensive example for
        developers working with AWS IoT in a Kotlin environment.
        """.trimIndent(),
    )

    print("Press Enter to continue...")
    scanner.nextLine()
    println(DASHES)

    println(DASHES)
    println("1. Create an AWS IoT thing.")
    println(
        """
        An AWS IoT thing represents a virtual entity in the AWS IoT service that
can be associated with a physical device.
        """.trimIndent(),
    )
    // Prompt the user for input.
    print("Enter thing name: ")
    thingName = scanner.nextLine()
    createIoTThing(thingName)
    describeThing(thingName)
    println(DASHES)

    println(DASHES)
    println("2. Generate a device certificate.")
    println(
        """
        A device certificate performs a role in securing the communication
between devices (things) and the AWS IoT platform.
        """.trimIndent(),
    )

    print("Do you want to create a certificate for $thingName? (y/n)")
    val certAns = scanner.nextLine()
    var certificateArn: String? = ""
```

```
    if (certAns != null && certAns.trim { it <= ' ' }.equals("y", ignoreCase =
true)) {
        certificateArn = createCertificate()
        println("Attach the certificate to the AWS IoT thing.")
        attachCertificateToThing(thingName, certificateArn)
    } else {
        println("A device certificate was not created.")
    }
    println(DASHES)

    println(DASHES)
    println("3. Update an AWS IoT thing with Attributes.")
    println(
        """
        IoT thing attributes, represented as key-value pairs, offer a pivotal
advantage in facilitating efficient data
        management and retrieval within the AWS IoT ecosystem.
        """.trimIndent(),
    )
    print("Press Enter to continue...")
    scanner.nextLine()
    updateThing(thingName)
    println(DASHES)

    println(DASHES)
    println("4. Return a unique endpoint specific to the Amazon Web Services
account.")
    println(
        """
        An IoT Endpoint refers to a specific URL or Uniform Resource Locator that
serves as the entry point for communication between IoT devices and the AWS IoT
service.
        """.trimIndent(),
    )
    print("Press Enter to continue...")
    scanner.nextLine()
    val endpointUrl = describeEndpoint()
    println(DASHES)

    println(DASHES)
    println("5. List your AWS IoT certificates")
    print("Press Enter to continue...")
    scanner.nextLine()
    if (certificateArn!!.isNotEmpty()) {
```

```
        listCertificates()
    } else {
        println("You did not create a certificates. Skipping this step.")
    }
    println(DASHES)

    println(DASHES)
    println("6. Create an IoT shadow that refers to a digital representation or
virtual twin of a physical IoT device")
    println(
        """
        A thing shadow refers to a feature that enables you to create a virtual
representation, or "shadow,"
        of a physical device or thing. The thing shadow allows you to synchronize
and control the state of a device between
        the cloud and the device itself. and the AWS IoT service. For example,
you can write and retrieve JSON data from a thing shadow.

        """.trimIndent(),
    )
    print("Press Enter to continue...")
    scanner.nextLine()
    updateShawdowThing(thingName)
    println(DASHES)

    println(DASHES)
    println("7. Write out the state information, in JSON format.")
    print("Press Enter to continue...")
    scanner.nextLine()
    getPayload(thingName)
    println(DASHES)

    println(DASHES)
    println("8. Creates a rule")
    println(
        """
        Creates a rule that is an administrator-level action.
        Any user who has permission to create rules will be able to access data
processed by the rule.
        """.trimIndent(),
    )
    print("Enter Rule name: ")
    val ruleName = scanner.nextLine()
    createIoTRule(roleARN, ruleName, snsAction)
```

```
    println(DASHES)

    println(DASHES)
    println("9. List your rules.")
    print("Press Enter to continue...")
    scanner.nextLine()
    listIoTRules()
    println(DASHES)

    println(DASHES)
    println("10. Search things using the name.")
    print("Press Enter to continue...")
    scanner.nextLine()
    val queryString = "thingName:$thingName"
    searchThings(queryString)
    println(DASHES)

    println(DASHES)
    if (certificateArn.length > 0) {
        print("Do you want to detach and delete the certificate for $thingName?
(y/n)")
        val delAns = scanner.nextLine()
        if (delAns != null && delAns.trim { it <= ' ' }.equals("y", ignoreCase =
true)) {
            println("11. You selected to detach amd delete the certificate.")
            print("Press Enter to continue...")
            scanner.nextLine()
            detachThingPrincipal(thingName, certificateArn)
            deleteCertificate(certificateArn)
        } else {
            println("11. You selected not to delete the certificate.")
        }
    } else {
        println("11. You did not create a certificate so there is nothing to
delete.")
    }
    println(DASHES)

    println(DASHES)
    println("12. Delete the AWS IoT thing.")
    print("Do you want to delete the IoT thing? (y/n)")
    val delAns = scanner.nextLine()
    if (delAns != null && delAns.trim { it <= ' ' }.equals("y", ignoreCase =
true)) {
```

```
        deleteIoTThing(thingName)
    } else {
        println("The IoT thing was not deleted.")
    }
    println(DASHES)

    println(DASHES)
    println("The AWS IoT workflow has successfully completed.")
    println(DASHES)
}

suspend fun deleteIoTThing(thingNameVal: String) {
    val deleteThingRequest =
        DeleteThingRequest {
            thingName = thingNameVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.deleteThing(deleteThingRequest)
        println("Deleted $thingNameVal")
    }
}

suspend fun deleteCertificate(certificateArn: String) {
    val certificateProviderRequest =
        DeleteCertificateRequest {
            certificateId = extractCertificateId(certificateArn)
        }
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.deleteCertificate(certificateProviderRequest)
        println("$certificateArn was successfully deleted.")
    }
}

private fun extractCertificateId(certificateArn: String): String? {
    // Example ARN: arn:aws:iot:region:account-id:cert/certificate-id.
    val arnParts = certificateArn.split(":".toRegex()).dropLastWhile
 { it.isEmpty() }.toTypedArray()
    val certificateIdPart = arnParts[arnParts.size - 1]
    return certificateIdPart.substring(certificateIdPart.lastIndexOf("/") + 1)
}

suspend fun detachThingPrincipal(
    thingNameVal: String,
```

```
    certificateArn: String,
) {
    val thingPrincipalRequest =
        DetachThingPrincipalRequest {
            principal = certificateArn
            thingName = thingNameVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.detachThingPrincipal(thingPrincipalRequest)
        println("$certificateArn was successfully removed from $thingNameVal")
    }
}

suspend fun searchThings(queryStringVal: String?) {
    val searchIndexRequest =
        SearchIndexRequest {
            queryString = queryStringVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val searchIndexResponse = iotClient.searchIndex(searchIndexRequest)
        if (searchIndexResponse.things?.isEmpty() == true) {
            println("No things found.")
        } else {
            searchIndexResponse.things
                ?.forEach { thing -> println("Thing id found using search is
 ${thing.thingId}") }
        }
    }
}

suspend fun listIoTRules() {
    val listTopicRulesRequest = ListTopicRulesRequest {}

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val listTopicRulesResponse =
 iotClient.listTopicRules(listTopicRulesRequest)
        println("List of IoT rules:")
        val ruleList = listTopicRulesResponse.rules
        ruleList?.forEach { rule ->
            println("Rule name: ${rule.ruleName}")
            println("Rule ARN: ${rule.ruleArn}")
            println("--------------")
```

```
        }
    }
}

suspend fun createIoTRule(
    roleARNVal: String?,
    ruleNameVal: String?,
    action: String?,
) {
    val sqlVal = "SELECT * FROM '$TOPIC '"
    val action1 =
        SnsAction {
            targetArn = action
            roleArn = roleARNVal
        }

    val myAction =
        Action {
            sns = action1
        }

    val topicRulePayloadVal =
        TopicRulePayload {
            sql = sqlVal
            actions = listOf(myAction)
        }

    val topicRuleRequest =
        CreateTopicRuleRequest {
            ruleName = ruleNameVal
            topicRulePayload = topicRulePayloadVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.createTopicRule(topicRuleRequest)
        println("IoT rule created successfully.")
    }
}

suspend fun getPayload(thingNameVal: String?) {
    val getThingShadowRequest =
        GetThingShadowRequest {
            thingName = thingNameVal
        }
```

```
    IotDataPlaneClient.fromEnvironment { region = "us-east-1" }.use
 { iotPlaneClient ->
        val getThingShadowResponse =
 iotPlaneClient.getThingShadow(getThingShadowRequest)
        val payload = getThingShadowResponse.payload
        val payloadString = payload?.let { java.lang.String(it, Charsets.UTF_8) }
        println("Received shadow data: $payloadString")
    }
}

suspend fun listCertificates() {
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val response = iotClient.listCertificates()
        val certList = response.certificates
        certList?.forEach { cert ->
            println("Cert id: ${cert.certificateId}")
            println("Cert Arn: ${cert.certificateArn}")
        }
    }
}

suspend fun describeEndpoint(): String? {
    val request = DescribeEndpointRequest {}
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val endpointResponse = iotClient.describeEndpoint(request)
        val endpointUrl: String? = endpointResponse.endpointAddress
        val exString: String = getValue(endpointUrl)
        val fullEndpoint = "https://$exString-ats.iot.us-east-1.amazonaws.com"
        println("Full endpoint URL: $fullEndpoint")
        return fullEndpoint
    }
}

private fun getValue(input: String?): String {
    // Define a regular expression pattern for extracting the subdomain.
    val pattern = Pattern.compile("^(.*?)\\.iot\\.us-east-1\\.amazonaws\\.com")

    // Match the pattern against the input string.
    val matcher = pattern.matcher(input)

    // Check if a match is found.
    if (matcher.find()) {
        val subdomain = matcher.group(1)
```

```
        println("Extracted subdomain: $subdomain")
        return subdomain
    } else {
        println("No match found")
    }
    return ""
}

suspend fun updateThing(thingNameVal: String?) {
    val newLocation = "Office"
    val newFirmwareVersion = "v2.0"
    val attMap: MutableMap<String, String> = HashMap()
    attMap["location"] = newLocation
    attMap["firmwareVersion"] = newFirmwareVersion

    val attributePayloadVal =
        AttributePayload {
            attributes = attMap
        }

    val updateThingRequest =
        UpdateThingRequest {
            thingName = thingNameVal
            attributePayload = attributePayloadVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        // Update the IoT thing attributes.
        iotClient.updateThing(updateThingRequest)
        println("$thingNameVal attributes updated successfully.")
    }
}

suspend fun updateShawdowThing(thingNameVal: String?) {
    // Create the thing shadow state document.
    val stateDocument = "{\"state\":{\"reported\":{\"temperature\":25, \"humidity
\":50}}}"
    val byteStream: ByteStream = ByteStream.fromString(stateDocument)
    val byteArray: ByteArray = byteStream.toByteArray()

    val updateThingShadowRequest =
        UpdateThingShadowRequest {
            thingName = thingNameVal
            payload = byteArray
```

```
        }

    IotDataPlaneClient.fromEnvironment { region = "us-east-1" }.use
 { iotPlaneClient ->
        iotPlaneClient.updateThingShadow(updateThingShadowRequest)
        println("The thing shadow was updated successfully.")
    }
}

suspend fun attachCertificateToThing(
    thingNameVal: String?,
    certificateArn: String?,
) {
    val principalRequest =
        AttachThingPrincipalRequest {
            thingName = thingNameVal
            principal = certificateArn
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.attachThingPrincipal(principalRequest)
        println("Certificate attached to $thingNameVal successfully.")
    }
}

suspend fun describeThing(thingNameVal: String) {
    val thingRequest =
        DescribeThingRequest {
            thingName = thingNameVal
        }

    // Print Thing details.
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val describeResponse = iotClient.describeThing(thingRequest)
        println("Thing details:")
        println("Thing name: ${describeResponse.thingName}")
        println("Thing ARN:  ${describeResponse.thingArn}")
    }
}

suspend fun createCertificate(): String? {
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val response = iotClient.createKeysAndCertificate()
        val certificatePem = response.certificatePem
```

```
        val certificateArn = response.certificateArn

        // Print the details.
        println("\nCertificate:")
        println(certificatePem)
        println("\nCertificate ARN:")
        println(certificateArn)
        return certificateArn
    }
}

suspend fun createIoTThing(thingNameVal: String) {
    val createThingRequest =
        CreateThingRequest {
            thingName = thingNameVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.createThing(createThingRequest)
        println("Created $thingNameVal}")
    }
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS -SDK für Kotlin.
  - AttachThingPrincipal
  - CreateKeysAndCertificate
  - CreateThing
  - CreateTopicRule
  - DeleteCertificate
  - DeleteThing
  - DeleteTopicRule
  - DescribeEndpoint
  - DescribeThing
  - DetachThingPrincipal
  - ListCertificates
  - ListThings

- SearchIndex
- UpdateIndexingConfiguration
- UpdateThing

Python

SDK für Python (Boto3)

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen Sie eine IoT-Wrapper-Klasse, um den Betrieb zu verwalten.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)


    def create_thing(self, thing_name):
        """
        Creates an AWS IoT thing.

        :param thing_name: The name of the thing to create.
        :return: The name and ARN of the created thing.
        """
```

```
        try:
            response = self.iot_client.create_thing(thingName=thing_name)
            logger.info("Created thing %s.", thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceAlreadyExistsException":
                logger.info("Thing %s already exists. Skipping creation.",
thing_name)
                return None
            logger.error(
                "Couldn't create thing %s. Here's why: %s: %s",
                thing_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def list_things(self):
        """
        Lists AWS IoT things.

        :return: The list of things.
        """
        try:
            things = []
            paginator = self.iot_client.get_paginator("list_things")
            for page in paginator.paginate():
                things.extend(page["things"])
            logger.info("Retrieved %s things.", len(things))
            return things
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't list things. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

```
    def create_keys_and_certificate(self):
        """
        Creates keys and a certificate for an AWS IoT thing.

        :return: The certificate ID, ARN, and PEM.
        """
        try:
            response =
self.iot_client.create_keys_and_certificate(setAsActive=True)
            logger.info("Created certificate %s.", response["certificateId"])
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't create keys and certificate. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response


    def attach_thing_principal(self, thing_name, principal):
        """
        Attaches a certificate to an AWS IoT thing.

        :param thing_name: The name of the thing.
        :param principal: The ARN of the certificate.
        """
        try:
            self.iot_client.attach_thing_principal(
                thingName=thing_name, principal=principal
            )
            logger.info("Attached principal %s to thing %s.", principal,
thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot attach principal. Resource not found.")
                return
            logger.error(
                "Couldn't attach principal to thing. Here's why: %s: %s",
```

```
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def describe_endpoint(self, endpoint_type="iot:Data-ATS"):
        """
        Gets the AWS IoT endpoint.

        :param endpoint_type: The endpoint type.
        :return: The endpoint.
        """
        try:
            response =
self.iot_client.describe_endpoint(endpointType=endpoint_type)
            logger.info("Retrieved endpoint %s.", response["endpointAddress"])
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't describe endpoint. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response["endpointAddress"]


    def list_certificates(self):
        """
        Lists AWS IoT certificates.

        :return: The list of certificates.
        """
        try:
            certificates = []
            paginator = self.iot_client.get_paginator("list_certificates")
            for page in paginator.paginate():
                certificates.extend(page["certificates"])
            logger.info("Retrieved %s certificates.", len(certificates))
            return certificates
```

```
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't list certificates. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise


    def detach_thing_principal(self, thing_name, principal):
        """
        Detaches a certificate from an AWS IoT thing.

        :param thing_name: The name of the thing.
        :param principal: The ARN of the certificate.
        """
        try:
            self.iot_client.detach_thing_principal(
                thingName=thing_name, principal=principal
            )
            logger.info("Detached principal %s from thing %s.", principal,
thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot detach principal. Resource not found.")
                return
            logger.error(
                "Couldn't detach principal from thing. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_certificate(self, certificate_id):
        """
        Deletes an AWS IoT certificate.

        :param certificate_id: The ID of the certificate to delete.
        """
        try:
```

```
            self.iot_client.update_certificate(
                certificateId=certificate_id, newStatus="INACTIVE"
            )
            self.iot_client.delete_certificate(certificateId=certificate_id)
            logger.info("Deleted certificate %s.", certificate_id)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot delete certificate. Resource not found.")
                return
            logger.error(
                "Couldn't delete certificate. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def create_topic_rule(self, rule_name, topic, sns_action_arn, role_arn):
        """
        Creates an AWS IoT topic rule.

        :param rule_name: The name of the rule.
        :param topic: The MQTT topic to subscribe to.
        :param sns_action_arn: The ARN of the SNS topic to publish to.
        :param role_arn: The ARN of the IAM role.
        """
        try:
            self.iot_client.create_topic_rule(
                ruleName=rule_name,
                topicRulePayload={
                    "sql": f"SELECT * FROM '{topic}'",
                    "actions": [
                        {"sns": {"targetArn": sns_action_arn, "roleArn":
role_arn}}
                    ],
                },
            )
            logger.info("Created topic rule %s.", rule_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceAlreadyExistsException":
                logger.info("Topic rule %s already exists. Skipping creation.",
rule_name)
                return
            logger.error(
```

```
                "Couldn't create topic rule. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def list_topic_rules(self):
        """
        Lists AWS IoT topic rules.

        :return: The list of topic rules.
        """
        try:
            rules = []
            paginator = self.iot_client.get_paginator("list_topic_rules")
            for page in paginator.paginate():
                rules.extend(page["rules"])
            logger.info("Retrieved %s topic rules.", len(rules))
            return rules
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't list topic rules. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise



    def search_index(self, query):
        """
        Searches the AWS IoT index.

        :param query: The search query.
        :return: The list of things found.
        """
        try:
            response = self.iot_client.search_index(queryString=query)
            logger.info("Found %s things.", len(response.get("things", [])))
        except ClientError as err:
```

```
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't search index. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response.get("things", [])


    def update_indexing_configuration(self):
        """
        Updates the AWS IoT indexing configuration to enable thing indexing.
        """
        try:
            self.iot_client.update_indexing_configuration(
                thingIndexingConfiguration={"thingIndexingMode": "REGISTRY"}
            )
            logger.info("Updated indexing configuration.")
        except ClientError as err:
            logger.error(
                "Couldn't update indexing configuration. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_thing(self, thing_name):
        """
        Deletes an AWS IoT thing.

        :param thing_name: The name of the thing to delete.
        """
        try:
            self.iot_client.delete_thing(thingName=thing_name)
            logger.info("Deleted thing %s.", thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot delete thing. Resource not found.")
                return
```

```
            logger.error(
                "Couldn't delete thing. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_topic_rule(self, rule_name):
        """
        Deletes an AWS IoT topic rule.

        :param rule_name: The name of the rule to delete.
        """
        try:
            self.iot_client.delete_topic_rule(ruleName=rule_name)
            logger.info("Deleted topic rule %s.", rule_name)
        except ClientError as err:
            logger.error(
                "Couldn't delete topic rule. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def update_thing_shadow(self, thing_name, shadow_state):
        """
        Updates the shadow for an AWS IoT thing.

        :param thing_name: The name of the thing.
        :param shadow_state: The shadow state as a dictionary.
        """
        import json
        try:
            self.iot_data_client.update_thing_shadow(
                thingName=thing_name, payload=json.dumps(shadow_state)
            )
            logger.info("Updated shadow for thing %s.", thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot update thing shadow. Resource not found.")
                return
            logger.error(
```

```
                "Couldn't update thing shadow. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def get_thing_shadow(self, thing_name):
        """
        Gets the shadow for an AWS IoT thing.

        :param thing_name: The name of the thing.
        :return: The shadow state as a dictionary.
        """
        import json
        try:
            response =
 self.iot_data_client.get_thing_shadow(thingName=thing_name)
            shadow = json.loads(response["payload"].read())
            logger.info("Retrieved shadow for thing %s.", thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot get thing shadow. Resource not found.")
                return None
            logger.error(
                "Couldn't get thing shadow. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return shadow
```

Führen Sie ein interaktives Szenario aus, das die IoT-Grundlagen demonstriert.

```
class IoTScenario:
    """Runs an interactive scenario that shows how to use AWS IoT."""

    is_interactive = True
```

```
    def __init__(self, iot_wrapper, iot_data_client, cfn_client,
 stack_name="IoTBasicsStack", template_path=None):
        """
        :param iot_wrapper: An instance of the IoTWrapper class.
        :param iot_data_client: A Boto3 IoT Data Plane client.
        :param cfn_client: A Boto3 CloudFormation client.
        :param stack_name: Name for the CloudFormation stack.
        :param template_path: Path to the CloudFormation template file.
        """
        self.iot_wrapper = iot_wrapper
        self.iot_data_client = iot_data_client
        self.cfn_client = cfn_client
        self.thing_name = None
        self.certificate_arn = None
        self.certificate_id = None
        self.rule_name = None
        self.stack_name = stack_name
        self.template_path = template_path or "../../../scenarios/basics/iot/
iot_usecase/resources/cfn_template.yaml"

    def _deploy_stack(self):
        """Deploy CloudFormation stack and return outputs."""
        with open(self.template_path, "r") as f:
            template_body = f.read()

        try:
            self.cfn_client.create_stack(
                StackName=self.stack_name,
                TemplateBody=template_body,
                Capabilities=["CAPABILITY_NAMED_IAM"]
            )

            waiter = self.cfn_client.get_waiter("stack_create_complete")
            waiter.wait(StackName=self.stack_name)

            response = self.cfn_client.describe_stacks(StackName=self.stack_name)
            outputs = {output["OutputKey"]: output["OutputValue"]
                       for output in response["Stacks"][0]["Outputs"]}
            return outputs["SNSTopicArn"], outputs["RoleArn"]
        except ClientError as err:
            if err.response["Error"]["Code"] == "AlreadyExistsException":
                response =
 self.cfn_client.describe_stacks(StackName=self.stack_name)
                outputs = {output["OutputKey"]: output["OutputValue"]
```

```
                        for output in response["Stacks"][0]["Outputs"]}
                return outputs["SNSTopicArn"], outputs["RoleArn"]
            raise

    def _cleanup_stack(self):
        """Delete CloudFormation stack."""
        try:
            self.cfn_client.delete_stack(StackName=self.stack_name)
            waiter = self.cfn_client.get_waiter("stack_delete_complete")
            waiter.wait(StackName=self.stack_name)
            print("CloudFormation stack deleted successfully.")
        except ClientError as err:
            logger.error(f"Failed to delete stack: {err}")

    def run_scenario(self, thing_name, rule_name):
        """
        Runs the IoT basics scenario.

        :param thing_name: The name of the thing to create.
        :param rule_name: The name of the topic rule to create.
        """
        print("-" * 88)
        print("Welcome to the AWS IoT basics scenario!")
        print("-" * 88)
        print(
            "This scenario demonstrates how to interact with AWS IoT using the
AWS SDK for Python (Boto3).\n"
            "AWS IoT provides secure, bi-directional communication between
Internet-connected devices\n"
            "and the AWS cloud. You can manage device connections, process device
data, and build IoT applications.\n"
        )

        self.thing_name = thing_name
        self.rule_name = rule_name

        try:
            print("\nDeploying CloudFormation stack...")
            sns_topic_arn, role_arn = self._deploy_stack()
            print(f"Stack deployed. SNS Topic: {sns_topic_arn}")

            input("\nNext, we'll create an AWS IoT thing. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
```

```
            print("1. Create an AWS IoT thing")
            print("-" * 88)
            response = self.iot_wrapper.create_thing(thing_name)
            print(f"Created thing: {response['thingName']}")
            print(f"Thing ARN: {response['thingArn']}")

            input("\nNext, we'll list things. Press Enter to continue...") if
self.is_interactive else None
            print("\n" + "-" * 88)
            print("2. List things")
            print("-" * 88)
            things = self.iot_wrapper.list_things()
            print(f"Found {len(things)} thing(s) in your account")
            for thing in things[:5]:  # Show first 5
                print(f"  Thing name: {thing['thingName']}")

            input("\nNext, we'll generate a device certificate. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("3. Generate a device certificate")
            print("-" * 88)
            cert_response = self.iot_wrapper.create_keys_and_certificate()
            self.certificate_arn = cert_response["certificateArn"]
            self.certificate_id = cert_response["certificateId"]
            print(f"Created certificate: {self.certificate_id}")

            input("\nNext, we'll attach the certificate to the thing. Press Enter
to continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("4. Attach the certificate to the thing")
            print("-" * 88)
            self.iot_wrapper.attach_thing_principal(thing_name,
self.certificate_arn)
            print(f"Attached certificate to thing: {thing_name}")

            input("\nNext, we'll update the thing shadow. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("5. Update the thing shadow")
            print("-" * 88)
            shadow_state = {"state": {"reported": {"temperature": 25, "humidity":
50}}}
            self.iot_wrapper.update_thing_shadow(thing_name, shadow_state)
            print(f"Updated shadow for thing: {thing_name}")
```

```
            input("\nNext, we'll get the thing shadow. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("6. Get the thing shadow")
            print("-" * 88)
            shadow = self.iot_wrapper.get_thing_shadow(thing_name)
            print(f"Shadow state: {json.dumps(shadow['state'], indent=2)}")

            input("\nNext, we'll get the AWS IoT endpoint. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("7. Get the AWS IoT endpoint")
            print("-" * 88)
            endpoint = self.iot_wrapper.describe_endpoint()
            print(f"IoT endpoint: {endpoint}")

            input("\nNext, we'll list certificates. Press Enter to continue...")
if self.is_interactive else None
            print("\n" + "-" * 88)
            print("8. List certificates")
            print("-" * 88)
            certificates = self.iot_wrapper.list_certificates()
            print(f"Found {len(certificates)} certificate(s)")
            for cert in certificates:
                print(f"  Certificate ID: {cert['certificateId']}")
                print(f"  Certificate ARN: {cert['certificateArn']}")
                print(f"  Status: {cert['status']}")

            input("\nNext, we'll create a topic rule. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("9. Create a topic rule")
            print("-" * 88)
            self.iot_wrapper.create_topic_rule(
                rule_name, f"device/{thing_name}/data", sns_topic_arn, role_arn
            )
            print(f"Created topic rule: {rule_name}")

            input("\nNext, we'll list topic rules. Press Enter to continue...")
if self.is_interactive else None
            print("\n" + "-" * 88)
            print("10. List topic rules")
            print("-" * 88)
```

```
            rules = self.iot_wrapper.list_topic_rules()
            print(f"Found {len(rules)} topic rule(s)")
            for rule in rules:
                print(f"  Rule name: {rule['ruleName']}")
                print(f"  Rule ARN: {rule['ruleArn']}")

            input("\nNext, we'll configure thing indexing. Press Enter to
continue...") if self.is_interactive else None
            print("\n" + "-" * 88)
            print("11. Configure thing indexing")
            print("-" * 88)
            self.iot_wrapper.update_indexing_configuration()
            print("Enabled thing indexing")
            print("Waiting for indexing to be ready...")
            time.sleep(10)

            input("\nNext, we'll search for things. Press Enter to continue...")
if self.is_interactive else None
            print("\n" + "-" * 88)
            print("12. Search for things")
            print("-" * 88)
            try:
                things = self.iot_wrapper.search_index(f"thingName:{thing_name}")
                if things:
                    print(f"Found {len(things)} thing(s) matching the query")
                    for thing in things:
                        print(f"  Thing name: {thing.get('thingName', 'N/A')}")
                        print(f"  Thing ID: {thing.get('thingId', 'N/A')}")
                else:
                    print("No things found. Indexing may take a few minutes.")
            except ClientError as err:
                if err.response["Error"]["Code"] in [
                    "IndexNotReadyException",
                    "InvalidRequestException",
                ]:
                    print("Search index not ready yet. This is expected.")
                else:
                    raise

        except ClientError as err:
            logger.error(
                "Scenario failed: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
```

```
            )
            raise
        finally:
            self._cleanup()

    def _cleanup(self):
        """Cleans up resources created during the scenario."""
        if not self.thing_name:
            return

        print("\n" + "-" * 88)
        print("Cleanup")
        print("-" * 88)

        if q.ask("Do you want to delete the resources? (y/n) ", q.is_yesno):
            try:
                if self.certificate_arn:
                    print(f"Detaching certificate from thing: {self.thing_name}")
                    self.iot_wrapper.detach_thing_principal(
                        self.thing_name, self.certificate_arn
                    )

                if self.certificate_id:
                    print(f"Deleting certificate: {self.certificate_id}")
                    self.iot_wrapper.delete_certificate(self.certificate_id)

                if self.thing_name:
                    print(f"Deleting thing: {self.thing_name}")
                    self.iot_wrapper.delete_thing(self.thing_name)

                if self.rule_name:
                    print(f"Deleting topic rule: {self.rule_name}")
                    self.iot_wrapper.delete_topic_rule(self.rule_name)

                self._cleanup_stack()
                print("Resources deleted successfully.")
            except ClientError as err:
                logger.error(
                    "Cleanup failed: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
        else:
            print("Resources will remain in your account.")
```

```
        print("\n" + "-" * 88)
        print("Thanks for using AWS IoT!")
        print("-" * 88)
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS SDK für Python (Boto3).
  - AttachThingPrincipal
  - CreateKeysAndCertificate
  - CreateThing
  - CreateTopicRule
  - DeleteCertificate
  - DeleteThing
  - DeleteTopicRule
  - DescribeEndpoint
  - DescribeThing
  - DetachThingPrincipal
  - ListCertificates
  - ListThings
  - SearchIndex
  - UpdateIndexingConfiguration
  - UpdateThing

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden AWS IoT mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Aktionen für AWS IoT verwenden AWS SDKs

Die folgenden Codebeispiele veranschaulichen, wie einzelne AWS IoT Aktionen mit AWS SDKs ausgeführt werden. Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes finden.

Diese Auszüge rufen die AWS IoT API auf und sind Codeauszüge aus größeren Programmen, die im Kontext ausgeführt werden müssen. Sie können Aktionen im Kontext unter Szenarien für AWS IoT verwenden AWS SDKs anzeigen.

Die folgenden Beispiele enthalten nur die am häufigsten verwendeten Aktionen. Eine vollständige Liste finden Sie in der AWS IoT -API-Referenz.

Beispiele

- Verwenden Sie mit AttachThingPrincipal einem AWS SDK oder CLI
- Verwenden Sie CreateKeysAndCertificate mit einem AWS SDK oder CLI
- Verwenden Sie CreateThing mit einem AWS SDK oder CLI
- Verwenden Sie CreateTopicRule mit einem AWS SDK oder CLI
- Verwenden Sie DeleteCertificate mit einem AWS SDK oder CLI
- Verwenden Sie DeleteThing mit einem AWS SDK oder CLI
- Verwenden Sie DeleteTopicRule mit einem AWS SDK oder CLI
- Verwenden Sie DescribeEndpoint mit einem AWS SDK oder CLI
- Verwenden Sie DescribeThing mit einem AWS SDK oder CLI
- Verwenden Sie DetachThingPrincipal mit einem AWS SDK oder CLI
- Verwenden Sie ListCertificates mit einem AWS SDK oder CLI
- Verwenden Sie ListThings mit einem AWS SDK oder CLI
- Verwenden Sie SearchIndex mit einem AWS SDK oder CLI
- Verwenden Sie UpdateIndexingConfiguration mit einem AWS SDK oder CLI
- Verwenden Sie UpdateThing mit einem AWS SDK oder CLI

Verwenden Sie mit **`AttachThingPrincipal`** einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `AttachThingPrincipal` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Kennenlernen der Grundlagen
- Erste Schritte mit Nachrichten im Internet der Dinge

.NET

SDK für .NET (v4)

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Attaches a certificate to an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to attach.</
param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> AttachThingPrincipalAsync(string thingName, string
 certificateArn)
    {
        try
        {
            var request = new AttachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.AttachThingPrincipalAsync(request);
            _logger.LogInformation($"Attached certificate {certificateArn} to
 Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot attach certificate - resource not found:
 {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
```

```
            _logger.LogError($"Couldn't attach certificate to Thing. Here's why:
 {ex.Message}");
            return false;
        }
    }
```

- API-Details finden Sie AttachThingPrincipal in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Attach a principal to an AWS IoT thing.
/*!
  \param principal: A principal to attach.
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::attachThingPrincipal(const Aws::String &principal,
                                       const Aws::String &thingName,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient client(clientConfiguration);
    Aws::IoT::Model::AttachThingPrincipalRequest request;
    request.SetPrincipal(principal);
    request.SetThingName(thingName);
    Aws::IoT::Model::AttachThingPrincipalOutcome outcome =
 client.AttachThingPrincipal(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully attached principal to thing." << std::endl;
    }
    else {
        std::cerr << "Failed to attach principal to thing." <<
```

```
                    outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API-Details finden Sie AttachThingPrincipal in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So hängen Sie Ihrem Objekt ein Zertifikat an

Das folgende `attach-thing-principal` Beispiel hängt ein Zertifikat an das MyTemperatureSensor Ding an. Das Zertifikat wird durch einen ARN identifiziert. Den ARN für ein Zertifikat finden Sie in der AWS IoT-Konsole.

```
aws iot attach-thing-principal \
    --thing-name MyTemperatureSensor \
    --principal arn:aws:iot:us-
west-2:123456789012:cert/2e1eb273792174ec2b9bf4e9b37e6c6c6923454995060002a35159767055278e8
```

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Weitere Informationen finden Sie unter Objektverwaltung mit der Registry im Entwicklerhandbuch für AWS IoT.

- API-Details finden Sie AttachThingPrincipal in der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   /**
    * Attaches a certificate to an IoT Thing asynchronously.
    *
    * @param thingName The name of the IoT Thing.
    * @param certificateArn The ARN of the certificate to attach.
    *
    * This method initiates an asynchronous request to attach a certificate to
an IoT Thing.
    * If the request is successful, it prints a confirmation message and
additional information about the Thing.
    * If an exception occurs, it prints the error message.
    */
   public void attachCertificateToThing(String thingName, String certificateArn)
{
       AttachThingPrincipalRequest principalRequest =
AttachThingPrincipalRequest.builder()
           .thingName(thingName)
           .principal(certificateArn)
           .build();

       CompletableFuture<AttachThingPrincipalResponse> future =
getAsyncClient().attachThingPrincipal(principalRequest);
       future.whenComplete((attachResponse, ex) -> {
           if (attachResponse != null &&
attachResponse.sdkHttpResponse().isSuccessful()) {
               System.out.println("Certificate attached to Thing
successfully.");

               // Print additional information about the Thing.
               describeThing(thingName);
           } else {
               Throwable cause = ex != null ? ex.getCause() : null;
               if (cause instanceof IotException) {
                   System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
               } else if (cause != null) {
                   System.err.println("Unexpected error: " +
cause.getMessage());
               } else {
                   System.err.println("Failed to attach certificate to Thing.
HTTP Status Code: " +
                           attachResponse.sdkHttpResponse().statusCode());
               }
```

```
            }
        });

        future.join();
    }
```

- API-Details finden Sie AttachThingPrincipal in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun attachCertificateToThing(
    thingNameVal: String?,
    certificateArn: String?,
) {
    val principalRequest =
        AttachThingPrincipalRequest {
            thingName = thingNameVal
            principal = certificateArn
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.attachThingPrincipal(principalRequest)
        println("Certificate attached to $thingNameVal successfully.")
    }
}
```

- API-Details finden Sie AttachThingPrincipal in der AWS SDK für die Kotlin-API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def attach_thing_principal(self, thing_name, principal):
        """
        Attaches a certificate to an AWS IoT thing.

        :param thing_name: The name of the thing.
        :param principal: The ARN of the certificate.
        """
        try:
            self.iot_client.attach_thing_principal(
                thingName=thing_name, principal=principal
            )
            logger.info("Attached principal %s to thing %s.", principal,
 thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
```

```
                logger.error("Cannot attach principal. Resource not found.")
                return
            logger.error(
                "Couldn't attach principal to thing. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API-Details finden Sie AttachThingPrincipal in der AWS API-Referenz zum SDK für Python (Boto3).

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
   DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
   DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
   TRY.
       lo_iot->attachthingprincipal(
         iv_thingname = iv_thing_name
         iv_principal = iv_principal ).
       MESSAGE |Principal attached to IoT thing '{ iv_thing_name }'.| TYPE 'I'.
     CATCH /aws1/cx_iotresourcenotfoundex INTO DATA(lo_ex).
       MESSAGE |Resource not found when attaching principal to
'{ iv_thing_name }'.| TYPE 'I'.
       RAISE EXCEPTION lo_ex.
     CATCH /aws1/cx_rt_service_generic INTO DATA(lo_svc_ex).
       MESSAGE lo_svc_ex->get_text( ) TYPE 'I'.
       RAISE EXCEPTION lo_svc_ex.
   ENDTRY.
```

- API-Details finden Sie AttachThingPrincipal in der AWS SDK for SAP ABAP API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden AWS IoT mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **`CreateKeysAndCertificate`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `CreateKeysAndCertificate` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Kennenlernen der Grundlagen
- Erste Schritte mit Nachrichten im Internet der Dinge

.NET

SDK für .NET (v4)

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   /// <summary>
   /// Creates a device certificate for AWS IoT.
   /// </summary>
   /// <returns>The certificate details including ARN and certificate PEM, or
null if creation failed.</returns>
   public async Task<(string CertificateArn, string CertificatePem, string
CertificateId)?> CreateKeysAndCertificateAsync()
   {
       try
       {
           var request = new CreateKeysAndCertificateRequest
           {
               SetAsActive = true
           };
```

```
            var response = await
_amazonIoT.CreateKeysAndCertificateAsync(request);
            _logger.LogInformation($"Created certificate with ARN
{response.CertificateArn}");
            return (response.CertificateArn, response.CertificatePem,
response.CertificateId);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create certificate. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- API-Details finden Sie CreateKeysAndCertificate in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Create keys and certificate for an Aws IoT device.
//! This routine will save certificates and keys to an output folder, if
 provided.
/*!
  \param outputFolder: Location for storing output in files, ignored when string
 is empty.
```

```
  \param certificateARNResult: A string to receive the ARN of the created
 certificate.
  \param certificateID: A string to receive the ID of the created certificate.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::createKeysAndCertificate(const Aws::String &outputFolder,
                                           Aws::String &certificateARNResult,
                                           Aws::String &certificateID,
                                           const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient client(clientConfiguration);
    Aws::IoT::Model::CreateKeysAndCertificateRequest
 createKeysAndCertificateRequest;

    Aws::IoT::Model::CreateKeysAndCertificateOutcome outcome =
            client.CreateKeysAndCertificate(createKeysAndCertificateRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created a certificate and keys" << std::endl;
        certificateARNResult = outcome.GetResult().GetCertificateArn();
        certificateID = outcome.GetResult().GetCertificateId();
        std::cout << "Certificate ARN: " << certificateARNResult << ",
 certificate ID: "
                  << certificateID << std::endl;

        if (!outputFolder.empty()) {
            std::cout << "Writing certificate and keys to the folder '" <<
 outputFolder
                      << "'." << std::endl;
            std::cout << "Be sure these files are stored securely." << std::endl;

            Aws::String certificateFilePath = outputFolder + "/
certificate.pem.crt";
            std::ofstream certificateFile(certificateFilePath);
            if (!certificateFile.is_open()) {
                std::cerr << "Error opening certificate file, '" <<
 certificateFilePath
                          << "'."
                          << std::endl;
                return false;
            }
            certificateFile << outcome.GetResult().GetCertificatePem();
            certificateFile.close();
```

```
            const Aws::IoT::Model::KeyPair &keyPair =
 outcome.GetResult().GetKeyPair();

            Aws::String privateKeyFilePath = outputFolder + "/private.pem.key";
            std::ofstream privateKeyFile(privateKeyFilePath);
            if (!privateKeyFile.is_open()) {
                std::cerr << "Error opening private key file, '" <<
 privateKeyFilePath
                          << "'."
                          << std::endl;
                return false;
            }
            privateKeyFile << keyPair.GetPrivateKey();
            privateKeyFile.close();

            Aws::String publicKeyFilePath = outputFolder + "/public.pem.key";
            std::ofstream publicKeyFile(publicKeyFilePath);
            if (!publicKeyFile.is_open()) {
                std::cerr << "Error opening public key file, '" <<
 publicKeyFilePath
                          << "'."
                          << std::endl;
                return false;
            }
            publicKeyFile << keyPair.GetPublicKey();
        }
    }
    else {
        std::cerr << "Error creating keys and certificate: "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API-Details finden Sie CreateKeysAndCertificate in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So erstellen Sie ein RSA-Schlüsselpaar und stellen ein X.509 Zertifikat aus

Im Folgenden `create-keys-and-certificate` wird ein 2048-Bit-RSA-Schlüsselpaar erstellt und ein X.509 Zertifikat unter Verwendung des ausgestellten öffentlichen Schlüssels ausgestellt. Da dies das einzige Mal ist, dass AWS IoT den privaten Schlüssel für dieses Zertifikat bereitstellt, sollten Sie es an einem sicheren Ort aufbewahren.

```
aws iot create-keys-and-certificate \
    --certificate-pem-outfile "myTest.cert.pem" \
    --public-key-outfile "myTest.public.key" \
    --private-key-outfile "myTest.private.key"
```

Ausgabe:

```
{
    "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/9894ba17925e663f1d29c23af4582b8e3b7619c31f3fbd93adcb51ae54b83dc2
    "certificateId":
 "9894ba17925e663f1d29c23af4582b8e3b7619c31f3fbd93adcb51ae54b83dc2",
    "certificatePem": "
-----BEGIN CERTIFICATE-----
MIICiTCCEXAMPLE6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMC
VVMxCzAJBgNVBAgEXAMPLEAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6
b24xFDASBgNVBAsTC0lBTSEXAMPLE2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAd
BgkqhkiG9w0BCQEWEG5vb25lQGFtYEXAMPLEb20wHhcNMTEwNDI1MjA0NTIxWhcN
MTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCEXAMPLEJBgNVBAgTAldBMRAwDgYD
VQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDAEXAMPLEsTC0lBTSBDb25z
b2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEXAMPLE25lQGFt
YXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMaK0dn+aEXAMPLE
EXAMPLEfEvySWtC2XADZ4nB+BLYgVIk60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9T
rDHudUZEXAMPLELG5M43q7Wgc/MbQITxOUSQv7c7ugFFDzQGBzZswY6786m86gpE
Ibb3OhjZnzcvQAEXAMPLEWIMm2nrAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4
nUhVVxYUntneD9+h8Mg9qEXAMPLEyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0Fkb
FFBjvSfpJIlJ00zbhNYS5f6GuoEDEXAMPLEBHjJnyp378OD8uTs7fLvjx79LjSTb
NYiytVbZPQUQ5Yaxu2jXnimvw3rrszlaEXAMPLE=
-----END CERTIFICATE-----\n",
    "keyPair": {
        "PublicKey": "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\nMMEXAMPLEuuN/
dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/gHr99VEEXAMPLE5VF13\n59VK7cEXAMPLE67GK+y
+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQ
\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa
```

```
\nhJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\nFQIDAQAB
\n-----END PUBLIC KEY-----\n",
        "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\nkey omittted for security
 reasons\n-----END RSA PRIVATE KEY-----\n"
    }
}
```

Weitere Informationen finden Sie unter Erstellen und Registrieren eines AWS IoT-Gerätezertifikats im AWS IoT Developer Guide.

- API-Details finden Sie CreateKeysAndCertificate in der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /**
     * Creates an IoT certificate asynchronously.
     *
     * @return The ARN of the created certificate.
     * <p>
     * This method initiates an asynchronous request to create an IoT
certificate.
     * If the request is successful, it prints the certificate details and
returns the certificate ARN.
     * If an exception occurs, it prints the error message.
     */
    public String createCertificate() {
        CompletableFuture<CreateKeysAndCertificateResponse> future =
getAsyncClient().createKeysAndCertificate();
        final String[] certificateArn = {null};
        future.whenComplete((response, ex) -> {
            if (response != null) {
                String certificatePem = response.certificatePem();
                certificateArn[0] = response.certificateArn();
```

```
                // Print the details.
                System.out.println("\nCertificate:");
                System.out.println(certificatePem);
                System.out.println("\nCertificate ARN:");
                System.out.println(certificateArn[0]);

            } else {
                Throwable cause = (ex instanceof CompletionException) ?
ex.getCause() : ex;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                }
            }
        });

        future.join();
        return certificateArn[0];
    }
```

- API-Details finden Sie CreateKeysAndCertificate in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun createCertificate(): String? {
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val response = iotClient.createKeysAndCertificate()
        val certificatePem = response.certificatePem
```

```
        val certificateArn = response.certificateArn

        // Print the details.
        println("\nCertificate:")
        println(certificatePem)
        println("\nCertificate ARN:")
        println(certificateArn)
        return certificateArn
    }
}
```

- API-Details finden Sie CreateKeysAndCertificate in der AWS SDK für die Kotlin-API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)
```

```
    def create_keys_and_certificate(self):
        """
        Creates keys and a certificate for an AWS IoT thing.

        :return: The certificate ID, ARN, and PEM.
        """
        try:
            response =
self.iot_client.create_keys_and_certificate(setAsActive=True)
            logger.info("Created certificate %s.", response["certificateId"])
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't create keys and certificate. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response
```

- API-Details finden Sie CreateKeysAndCertificate in der AWS API-Referenz zum SDK für Python (Boto3).

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
    DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
```

```
    DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
    TRY.
        oo_result = lo_iot->createkeysandcertificate( iv_setasactive =
abap_true ).
        MESSAGE |Certificate created: { oo_result->get_certificateid( ) }| TYPE
'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_ex).
        MESSAGE lo_ex->get_text( ) TYPE 'I'.
        RAISE EXCEPTION lo_ex.
    ENDTRY.
```

- API-Details finden Sie CreateKeysAndCertificate in der AWS SDK for SAP ABAP API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden AWS IoT mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **`CreateThing`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `CreateThing` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Kennenlernen der Grundlagen
- Erste Schritte mit Nachrichten im Internet der Dinge

.NET

SDK für .NET (v4)

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
```

```
    /// Creates an AWS IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to create.</param>
    /// <returns>The ARN of the Thing created, or null if creation failed.</
returns>
    public async Task<string?> CreateThingAsync(string thingName)
    {
        try
        {
            var request = new CreateThingRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIoT.CreateThingAsync(request);
            _logger.LogInformation($"Created Thing {thingName} with ARN
 {response.ThingArn}");
            return response.ThingArn;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Thing {thingName} already exists:
 {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create Thing {thingName}. Here's why:
 {ex.Message}");
            return null;
        }
    }
```

- API-Details finden Sie CreateThing in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Create an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::createThing(const Aws::String &thingName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::CreateThingRequest createThingRequest;
    createThingRequest.SetThingName(thingName);

    Aws::IoT::Model::CreateThingOutcome outcome = iotClient.CreateThing(
            createThingRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to create thing " << thingName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API-Details finden Sie CreateThing in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

Beispiel 1: So erstellen Sie einen Objektdatensatz in der Registry

Das folgende `create-thing` Beispiel erstellt einen Eintrag für ein Gerät in der AWS IoT-Dingregistrierung.

```
aws iot create-thing \
    --thing-name SampleIoTThing
```

Ausgabe:

```
{
    "thingName": "SampleIoTThing",
    "thingArn": "arn:aws:iot:us-west-2: 123456789012:thing/SampleIoTThing",
    "thingId": " EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE "
}
```

Beispiel 2: So definieren Sie ein Objekt, das einem Objekttyp zugeordnet ist

Im folgenden Beispiel für `create-thing` wird ein Objekt mit dem angegebenen Objekttyp und seinen Attributen erstellt.

```
aws iot create-thing \
    --thing-name "MyLightBulb" \
    --thing-type-name "LightBulb" \
    --attribute-payload "{"attributes": {"wattage":"75", "model":"123"}}"
```

Ausgabe:

```
{
    "thingName": "MyLightBulb",
    "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyLightBulb",
    "thingId": "40da2e73-c6af-406e-b415-15acae538797"
}
```

Weitere Informationen finden Sie unter Objektverwaltung mit der Registry und Objekttypen im Entwicklerhandbuch für AWS IoT.

- API-Details finden Sie CreateThing in der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   /**
    * Creates an IoT Thing with the specified name asynchronously.
    *
    * @param thingName The name of the IoT Thing to create.
    *
    * This method initiates an asynchronous request to create an IoT Thing with
the specified name.
    * If the request is successful, it prints the name of the thing and its ARN
value.
    * If an exception occurs, it prints the error message.
    */
   public void createIoTThing(String thingName) {
       CreateThingRequest createThingRequest = CreateThingRequest.builder()
           .thingName(thingName)
           .build();

       CompletableFuture<CreateThingResponse> future =
getAsyncClient().createThing(createThingRequest);
       future.whenComplete((createThingResponse, ex) -> {
           if (createThingResponse != null) {
               System.out.println(thingName + " was successfully created. The
ARN value is " + createThingResponse.thingArn());
           } else {
               Throwable cause = ex.getCause();
               if (cause instanceof IotException) {
                   System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
               } else {
                   System.err.println("Unexpected error: " +
cause.getMessage());
               }
           }
```

```
        });

        future.join();
    }
```

- API-Details finden Sie CreateThing in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun createIoTThing(thingNameVal: String) {
    val createThingRequest =
        CreateThingRequest {
            thingName = thingNameVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.createThing(createThingRequest)
        println("Created $thingNameVal}")
    }
}
```

- API-Details finden Sie CreateThing in der AWS SDK für die Kotlin-API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def create_thing(self, thing_name):
        """
        Creates an AWS IoT thing.

        :param thing_name: The name of the thing to create.
        :return: The name and ARN of the created thing.
        """
        try:
            response = self.iot_client.create_thing(thingName=thing_name)
            logger.info("Created thing %s.", thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceAlreadyExistsException":
                logger.info("Thing %s already exists. Skipping creation.",
 thing_name)
                return None
```

```
            logger.error(
                "Couldn't create thing %s. Here's why: %s: %s",
                thing_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- API-Details finden Sie CreateThing in der AWS API-Referenz zum SDK für Python (Boto3).

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
   DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
   DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
   TRY.
       oo_result = lo_iot->creatething(
         iv_thingname = iv_thing_name ).
       MESSAGE |IoT thing created: { oo_result->get_thingname( ) } ARN:
{ oo_result->get_thingarn( ) }| TYPE 'I'.
     CATCH /aws1/cx_iotresrcalrdyexistsex.
       MESSAGE |IoT thing '{ iv_thing_name }' already exists.| TYPE 'I'.
     CATCH /aws1/cx_rt_service_generic INTO DATA(lo_ex).
       MESSAGE lo_ex->get_text( ) TYPE 'I'.
       RAISE EXCEPTION lo_ex.
   ENDTRY.
```

- API-Details finden Sie CreateThing in der AWS SDK for SAP ABAP API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden AWS IoT mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **`CreateTopicRule`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `CreateTopicRule` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Creates an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="snsTopicArn">The ARN of the SNS topic for the action.</
param>
    /// <param name="roleArn">The ARN of the IAM role.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> CreateTopicRuleAsync(string ruleName, string
 snsTopicArn, string roleArn)
    {
        try
        {
            var request = new CreateTopicRuleRequest
            {
                RuleName = ruleName,
                TopicRulePayload = new TopicRulePayload
                {
```

```
                    Sql = "SELECT * FROM 'topic/subtopic'",
                    Description = $"Rule created by .NET example: {ruleName}",
                    Actions = new List<Amazon.IoT.Model.Action>
                    {
                        new Amazon.IoT.Model.Action
                        {
                            Sns = new SnsAction
                            {
                                TargetArn = snsTopicArn,
                                RoleArn = roleArn
                            }
                        }
                    },
                    RuleDisabled = false
                }
            };

            await _amazonIoT.CreateTopicRuleAsync(request);
            _logger.LogInformation($"Created IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Rule {ruleName} already exists: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create topic rule. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- API-Details finden Sie CreateTopicRule in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Create an AWS IoT rule with an SNS topic as the target.
/*!
  \param ruleName: The name for the rule.
  \param snsTopic: The SNS topic ARN for the action.
  \param sql: The SQL statement used to query the topic.
  \param roleARN: The IAM role ARN for the action.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::IoT::createTopicRule(const Aws::String &ruleName,
                             const Aws::String &snsTopicARN, const Aws::String
 &sql,
                             const Aws::String &roleARN,
                             const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::CreateTopicRuleRequest request;
    request.SetRuleName(ruleName);

    Aws::IoT::Model::SnsAction snsAction;
    snsAction.SetTargetArn(snsTopicARN);
    snsAction.SetRoleArn(roleARN);

    Aws::IoT::Model::Action action;
    action.SetSns(snsAction);

    Aws::IoT::Model::TopicRulePayload topicRulePayload;
    topicRulePayload.SetSql(sql);
    topicRulePayload.SetActions({action});
```

```
    request.SetTopicRulePayload(topicRulePayload);
    auto outcome = iotClient.CreateTopicRule(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created topic rule " << ruleName << "." <<
 std::endl;
    }
    else {
        std::cerr << "Error creating topic rule " << ruleName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
    return outcome.IsSuccess();
}
```

- API-Details finden Sie CreateTopicRule in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So erstellen Sie eine Regel, die eine Amazon-SNS-Warnung sendet

Im folgenden Beispiel für `create-topic-rule` wird eine Regel erstellt, die eine Amazon-SNS-Nachricht sendet, wenn die Werte für die Erdfeuchte in einem Geräteschatten niedrig sind.

```
aws iot create-topic-rule \
    --rule-name "LowMoistureRule" \
    --topic-rule-payload file://plant-rule.json
```

Für das Beispiel muss der folgende JSON-Code in einer Datei mit dem Namen `plant-rule.json` gespeichert werden:

```
{
    "sql": "SELECT * FROM '$aws/things/MyRPi/shadow/update/accepted' WHERE
 state.reported.moisture = 'low'\n",
    "description": "Sends an alert whenever soil moisture level readings are too
 low.",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [{
            "sns": {
```

```
                "targetArn": "arn:aws:sns:us-
west-2:123456789012:MyRPiLowMoistureTopic",
                "roleArn": "arn:aws:iam::123456789012:role/service-role/
MyRPiLowMoistureTopicRole",
                "messageFormat": "RAW"
            }
    }]
}
```

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Weitere Informationen finden Sie unter Erstellen einer AWS IoT-Regel im AWS IoT Developers Guide.

- API-Details finden Sie CreateTopicRule in der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /**
     * Creates an IoT rule asynchronously.
     *
     * @param roleARN The ARN of the IAM role that grants access to the rule's
 actions.
     * @param ruleName The name of the IoT rule.
     * @param action The ARN of the action to perform when the rule is triggered.
     *
     * This method initiates an asynchronous request to create an IoT rule.
     * If the request is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void createIoTRule(String roleARN, String ruleName, String action) {
        String sql = "SELECT * FROM '" + TOPIC + "'";
        SnsAction action1 = SnsAction.builder()
            .targetArn(action)
```

```
            .roleArn(roleARN)
            .build();

        // Create the action.
        Action myAction = Action.builder()
            .sns(action1)
            .build();

        // Create the topic rule payload.
        TopicRulePayload topicRulePayload = TopicRulePayload.builder()
            .sql(sql)
            .actions(myAction)
            .build();

        // Create the topic rule request.
        CreateTopicRuleRequest topicRuleRequest =
CreateTopicRuleRequest.builder()
            .ruleName(ruleName)
            .topicRulePayload(topicRulePayload)
            .build();

        CompletableFuture<CreateTopicRuleResponse> future =
getAsyncClient().createTopicRule(topicRuleRequest);
        future.whenComplete((response, ex) -> {
            if (response != null) {
                System.out.println("IoT Rule created successfully.");
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to create IoT Rule.");
                }
            }
        });

        future.join();
    }
```

- API-Details finden Sie CreateTopicRule in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun createIoTRule(
    roleARNVal: String?,
    ruleNameVal: String?,
    action: String?,
) {
    val sqlVal = "SELECT * FROM '$TOPIC '"
    val action1 =
        SnsAction {
            targetArn = action
            roleArn = roleARNVal
        }

    val myAction =
        Action {
            sns = action1
        }

    val topicRulePayloadVal =
        TopicRulePayload {
            sql = sqlVal
            actions = listOf(myAction)
        }

    val topicRuleRequest =
        CreateTopicRuleRequest {
            ruleName = ruleNameVal
            topicRulePayload = topicRulePayloadVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
```

```
        iotClient.createTopicRule(topicRuleRequest)
        println("IoT rule created successfully.")
    }
}
```

- API-Details finden Sie CreateTopicRule in der AWS SDK für die Kotlin-API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def create_topic_rule(self, rule_name, topic, sns_action_arn, role_arn):
        """
        Creates an AWS IoT topic rule.

        :param rule_name: The name of the rule.
        :param topic: The MQTT topic to subscribe to.
        :param sns_action_arn: The ARN of the SNS topic to publish to.
```

```
        :param role_arn: The ARN of the IAM role.
        """
        try:
            self.iot_client.create_topic_rule(
                ruleName=rule_name,
                topicRulePayload={
                    "sql": f"SELECT * FROM '{topic}'",
                    "actions": [
                        {"sns": {"targetArn": sns_action_arn, "roleArn":
role_arn}}
                    ],
                },
            )
            logger.info("Created topic rule %s.", rule_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceAlreadyExistsException":
                logger.info("Topic rule %s already exists. Skipping creation.",
rule_name)
                return
            logger.error(
                "Couldn't create topic rule. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API-Details finden Sie CreateTopicRule in der AWS API-Referenz zum SDK für Python (Boto3).

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
    DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
    DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
    TRY.
        " Build the SNS action that will receive messages matching the rule.
        DATA lo_sns_action TYPE REF TO /aws1/cl_iotsnsaction.
        CREATE OBJECT lo_sns_action
          EXPORTING
            iv_targetarn = iv_sns_action_arn
            iv_rolearn   = iv_role_arn.

        DATA lo_action TYPE REF TO /aws1/cl_iotaction.
        CREATE OBJECT lo_action
          EXPORTING
            io_sns = lo_sns_action.

        DATA lt_actions TYPE /aws1/cl_iotaction=>tt_actionlist.
        APPEND lo_action TO lt_actions.

        " iv_topic = 'my/iot/topic' - The MQTT topic pattern to match
        DATA lo_payload TYPE REF TO /aws1/cl_iottopicrulepayload.
        CREATE OBJECT lo_payload
          EXPORTING
            iv_sql     = |SELECT * FROM '{ iv_topic }'|
            it_actions = lt_actions.

        lo_iot->createtopicrule(
          iv_rulename         = iv_rule_name
          io_topicrulepayload = lo_payload ).
        MESSAGE |IoT topic rule created: { iv_rule_name }| TYPE 'I'.
      CATCH /aws1/cx_iotresrcalrdyexistsex.
        MESSAGE |Topic rule '{ iv_rule_name }' already exists.| TYPE 'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_ex).
        MESSAGE lo_ex->get_text( ) TYPE 'I'.
        RAISE EXCEPTION lo_ex.
    ENDTRY.
```

- API-Details finden Sie CreateTopicRule in der AWS SDK for SAP ABAP API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden AWS IoT mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **`DeleteCertificate`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `DeleteCertificate` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Kennenlernen der Grundlagen
- Erste Schritte mit Nachrichten im Internet der Dinge

.NET

SDK für .NET (v4)

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
/// <summary>
/// Deletes an IoT certificate.
/// </summary>
/// <param name="certificateId">The ID of the certificate to delete.</param>
/// <returns>True if successful, false otherwise.</returns>
public async Task<bool> DeleteCertificateAsync(string certificateId)
{
    try
    {
        // First, update the certificate to inactive state
        var updateRequest = new UpdateCertificateRequest
        {
            CertificateId = certificateId,
            NewStatus = CertificateStatus.INACTIVE
        };
        await _amazonIoT.UpdateCertificateAsync(updateRequest);
```

```
            // Then delete the certificate
            var deleteRequest = new DeleteCertificateRequest
            {
                CertificateId = certificateId
            };

            await _amazonIoT.DeleteCertificateAsync(deleteRequest);
            _logger.LogInformation($"Deleted certificate {certificateId}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete certificate. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- API-Details finden Sie DeleteCertificate in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Delete a certificate.
/*!
  \param certificateID: The ID of a certificate.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
```

```
 */
bool AwsDoc::IoT::deleteCertificate(const Aws::String &certificateID,
                                    const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::DeleteCertificateRequest request;
    request.SetCertificateId(certificateID);

    Aws::IoT::Model::DeleteCertificateOutcome outcome =
 iotClient.DeleteCertificate(
            request);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted certificate " << certificateID <<
 std::endl;
    }
    else {
        std::cerr << "Error deleting certificate " << certificateID << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API-Details finden Sie DeleteCertificate in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So löschen Sie ein Gerätezertifikat

Im folgenden Beispiel für `delete-certificate` wird das Gerätezertifikat mit der angegebenen ID gelöscht.

```
aws iot delete-certificate \
    --certificate-
id c0c57bbc8baaf4631a9a0345c957657f5e710473e3ddbee1428d216d54d53ac9
```

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Weitere Informationen finden Sie DeleteCertificate in der AWS IoT-API-Referenz.

- API-Details finden Sie DeleteCertificate in der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /**
     * Deletes a certificate asynchronously.
     *
     * @param certificateArn The ARN of the certificate to delete.
     *
     * This method initiates an asynchronous request to delete a certificate.
     * If the deletion is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void deleteCertificate(String certificateArn) {
        DeleteCertificateRequest certificateProviderRequest =
DeleteCertificateRequest.builder()
            .certificateId(extractCertificateId(certificateArn))
            .build();

        CompletableFuture<DeleteCertificateResponse> future =
getAsyncClient().deleteCertificate(certificateProviderRequest);
        future.whenComplete((voidResult, ex) -> {
            if (ex == null) {
                System.out.println(certificateArn + " was successfully
deleted.");
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " + ex.getMessage());
```

```
                }
            }
        });

        future.join();
    }
```

- API-Details finden Sie DeleteCertificate in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun deleteCertificate(certificateArn: String) {
    val certificateProviderRequest =
        DeleteCertificateRequest {
            certificateId = extractCertificateId(certificateArn)
        }
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.deleteCertificate(certificateProviderRequest)
        println("$certificateArn was successfully deleted.")
    }
}
```

- API-Details finden Sie DeleteCertificate in der AWS SDK für die Kotlin-API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def delete_certificate(self, certificate_id):
        """
        Deletes an AWS IoT certificate.

        :param certificate_id: The ID of the certificate to delete.
        """
        try:
            self.iot_client.update_certificate(
                certificateId=certificate_id, newStatus="INACTIVE"
            )
            self.iot_client.delete_certificate(certificateId=certificate_id)
            logger.info("Deleted certificate %s.", certificate_id)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot delete certificate. Resource not found.")
```

```
            return
        logger.error(
            "Couldn't delete certificate. Here's why: %s: %s",
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
```

- API-Details finden Sie DeleteCertificate in der AWS API-Referenz zum SDK für Python (Boto3).

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
TRY.
    " Certificates must be deactivated before they can be deleted.
    lo_iot->updatecertificate(
      iv_certificateid = iv_certificate_id
      iv_newstatus     = 'INACTIVE' ).
    lo_iot->deletecertificate( iv_certificateid = iv_certificate_id ).
    MESSAGE |Certificate deleted: { iv_certificate_id }| TYPE 'I'.
  CATCH /aws1/cx_iotresourcenotfoundex INTO DATA(lo_ex).
    MESSAGE |Certificate '{ iv_certificate_id }' not found.| TYPE 'I'.
    RAISE EXCEPTION lo_ex.
  CATCH /aws1/cx_rt_service_generic INTO DATA(lo_svc_ex).
    MESSAGE lo_svc_ex->get_text( ) TYPE 'I'.
    RAISE EXCEPTION lo_svc_ex.
ENDTRY.
```

- API-Details finden Sie DeleteCertificate in der AWS SDK for SAP ABAP API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden AWS IoT mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **`DeleteThing`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `DeleteThing` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Kennenlernen der Grundlagen
- Erste Schritte mit Nachrichten im Internet der Dinge

.NET

SDK für .NET (v4)

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
/// <summary>
/// Deletes an IoT Thing.
/// </summary>
/// <param name="thingName">The name of the Thing to delete.</param>
/// <returns>True if successful, false otherwise.</returns>
public async Task<bool> DeleteThingAsync(string thingName)
{
    try
    {
        var request = new DeleteThingRequest
        {
            ThingName = thingName
        };
```

```
            await _amazonIoT.DeleteThingAsync(request);
            _logger.LogInformation($"Deleted Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete Thing - resource not found:
 {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete Thing. Here's why: {ex.Message}");
            return false;
        }
    }
```

- API-Details finden Sie DeleteThing in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Delete an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteThing(const Aws::String &thingName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DeleteThingRequest request;
    request.SetThingName(thingName);
```

```
    const auto outcome = iotClient.DeleteThing(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Error deleting thing " << thingName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API-Details finden Sie DeleteThing in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So zeigen Sie detaillierte Informationen zu einem Objekt an

Im folgenden `delete-thing` Beispiel wird ein Objekt aus der AWS IoT-Registrierung für Ihr AWS Konto gelöscht.

aws iot delete-thing --thing-name“ FourthBulb

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Weitere Informationen finden Sie unter Objektverwaltung mit der Registry im Entwicklerhandbuch für AWS IoT.

- API-Details finden Sie in der Befehlsreferenz. DeleteThing AWS CLI

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /**
     * Deletes an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing to delete.
     *
     * This method initiates an asynchronous request to delete an IoT Thing.
     * If the deletion is successful, it prints a confirmation message.
     * If an exception occurs, it prints the error message.
     */
    public void deleteIoTThing(String thingName) {
        DeleteThingRequest deleteThingRequest = DeleteThingRequest.builder()
            .thingName(thingName)
            .build();

        CompletableFuture<DeleteThingResponse> future =
getAsyncClient().deleteThing(deleteThingRequest);
        future.whenComplete((voidResult, ex) -> {
            if (ex == null) {
                System.out.println("Deleted Thing " + thingName);
            } else {
                Throwable cause = ex.getCause();
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " + ex.getMessage());
                }
            }
        });

        future.join();
    }
```

- API-Details finden Sie DeleteThing in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun deleteIoTThing(thingNameVal: String) {
    val deleteThingRequest =
        DeleteThingRequest {
            thingName = thingNameVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.deleteThing(deleteThingRequest)
        println("Deleted $thingNameVal")
    }
}
```

- API-Details finden Sie DeleteThing in der AWS SDK für die Kotlin-API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
```

```
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def delete_thing(self, thing_name):
        """
        Deletes an AWS IoT thing.

        :param thing_name: The name of the thing to delete.
        """
        try:
            self.iot_client.delete_thing(thingName=thing_name)
            logger.info("Deleted thing %s.", thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error("Cannot delete thing. Resource not found.")
                return
            logger.error(
                "Couldn't delete thing. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API-Details finden Sie DeleteThing in der AWS API-Referenz zum SDK für Python (Boto3).

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
TRY.
    lo_iot->deletething( iv_thingname = iv_thing_name ).
    MESSAGE |IoT thing deleted: { iv_thing_name }| TYPE 'I'.
  CATCH /aws1/cx_iotresourcenotfoundex INTO DATA(lo_ex).
    MESSAGE |IoT thing '{ iv_thing_name }' not found.| TYPE 'I'.
    RAISE EXCEPTION lo_ex.
  CATCH /aws1/cx_rt_service_generic INTO DATA(lo_svc_ex).
    MESSAGE lo_svc_ex->get_text( ) TYPE 'I'.
    RAISE EXCEPTION lo_svc_ex.
ENDTRY.
```

- API-Details finden Sie DeleteThing in der AWS SDK for SAP ABAP API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden AWS IoT mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **`DeleteTopicRule`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `DeleteTopicRule` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

C++

SDK für C++

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Delete an AWS IoT rule.
/*!
  \param ruleName: The name for the rule.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteTopicRule(const Aws::String &ruleName,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DeleteTopicRuleRequest request;
    request.SetRuleName(ruleName);

    Aws::IoT::Model::DeleteTopicRuleOutcome outcome = iotClient.DeleteTopicRule(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted rule " << ruleName << std::endl;
    }
    else {
        std::cerr << "Failed to delete rule " << ruleName <<
                  ": " << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API-Details finden Sie DeleteTopicRule in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So löschen Sie eine Regel

Im folgenden Beispiel für `delete-topic-rule` wird die angegebene Regel gelöscht.

```
aws iot delete-topic-rule \
    --rule-name "LowMoistureRule"
```

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Weitere Informationen finden Sie unter Löschen einer Regel im Entwicklerhandbuch für AWS IoT.

- API-Details finden Sie DeleteTopicRule in der AWS CLI Befehlsreferenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
```

```
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def delete_topic_rule(self, rule_name):
        """
        Deletes an AWS IoT topic rule.

        :param rule_name: The name of the rule to delete.
        """
        try:
            self.iot_client.delete_topic_rule(ruleName=rule_name)
            logger.info("Deleted topic rule %s.", rule_name)
        except ClientError as err:
            logger.error(
                "Couldn't delete topic rule. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API-Details finden Sie DeleteTopicRule in der AWS API-Referenz zum SDK für Python (Boto3).

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
TRY.
    lo_iot->deletetopicrule( iv_rulename = iv_rule_name ).
```

```
      MESSAGE |IoT topic rule deleted: { iv_rule_name }| TYPE 'I'.
    CATCH /aws1/cx_rt_service_generic INTO DATA(lo_ex).
      MESSAGE lo_ex->get_text( ) TYPE 'I'.
      RAISE EXCEPTION lo_ex.
  ENDTRY.
```

- API-Details finden Sie DeleteTopicRule in der AWS SDK for SAP ABAP API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden AWS IoT mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **`DescribeEndpoint`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `DescribeEndpoint` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Kennenlernen der Grundlagen
- Erste Schritte mit Nachrichten im Internet der Dinge

.NET

SDK für .NET (v4)

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
/// <summary>
/// Gets the AWS IoT endpoint URL.
/// </summary>
/// <returns>The endpoint URL, or null if retrieval failed.</returns>
public async Task<string?> DescribeEndpointAsync()
{
```

```
        try
        {
            var request = new DescribeEndpointRequest
            {
                EndpointType = "iot:Data-ATS"
            };

            var response = await _amazonIoT.DescribeEndpointAsync(request);
            _logger.LogInformation($"Retrieved endpoint:
{response.EndpointAddress}");
            return response.EndpointAddress;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't describe endpoint. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- API-Details finden Sie DescribeEndpoint in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Describe the endpoint specific to the AWS account making the call.
/*!
  \param endpointResult: String to receive the endpoint result.
```

```
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::describeEndpoint(Aws::String &endpointResult,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::String endpoint;
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DescribeEndpointRequest describeEndpointRequest;
    describeEndpointRequest.SetEndpointType(
            "iot:Data-ATS"); // Recommended endpoint type.

    Aws::IoT::Model::DescribeEndpointOutcome outcome =
 iotClient.DescribeEndpoint(
            describeEndpointRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully described endpoint." << std::endl;
        endpointResult = outcome.GetResult().GetEndpointAddress();
    }
    else {
        std::cerr << "Error describing endpoint" <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API-Details finden Sie DescribeEndpoint in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

Beispiel 1: Um Ihren aktuellen AWS Endpunkt abzurufen

Im folgenden `describe-endpoint` Beispiel wird der AWS Standardendpunkt abgerufen, auf den alle Befehle angewendet werden.

```
aws iot describe-endpoint
```

Ausgabe:

```
{
    "endpointAddress": "abc123defghijk.iot.us-west-2.amazonaws.com"
}
```

Weitere Informationen finden Sie DescribeEndpoint im AWS IoT Developer Guide.

Beispiel 2: So rufen Sie Ihren ATS-Endpunkt ab

Im folgenden Beispiel für `describe-endpoint` wird der Amazon Trust Services (ATS)-Endpunkt abgerufen.

```
aws iot describe-endpoint \
    --endpoint-type iot:Data-ATS
```

Ausgabe:

```
{
    "endpointAddress": "abc123defghijk-ats.iot.us-west-2.amazonaws.com"
}
```

Weitere Informationen finden Sie unter X.509 Zertifikate und AWS IoT im AWS IoT Developer Guide.

- API-Details finden Sie DescribeEndpoint in der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /**
```

```
     * Describes the endpoint of the IoT service asynchronously.
     *
     * @return A CompletableFuture containing the full endpoint URL.
     *
     * This method initiates an asynchronous request to describe the endpoint of
 the IoT service.
     * If the request is successful, it prints and returns the full endpoint URL.
     * If an exception occurs, it prints the error message.
     */
    public String describeEndpoint() {
        CompletableFuture<DescribeEndpointResponse> future =
 getAsyncClient().describeEndpoint(DescribeEndpointRequest.builder().endpointType("iot:Da
ATS").build());
        final String[] result = {null};

        future.whenComplete((endpointResponse, ex) -> {
            if (endpointResponse != null) {
                String endpointUrl = endpointResponse.endpointAddress();
                String exString = getValue(endpointUrl);
                String fullEndpoint = "https://" + exString + "-ats.iot.us-
east-1.amazonaws.com";

                System.out.println("Full Endpoint URL: " + fullEndpoint);
                result[0] = fullEndpoint;
            } else {
                Throwable cause = (ex instanceof CompletionException) ?
 ex.getCause() : ex;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
 cause).awsErrorDetails().errorMessage());
                } else {
                    System.err.println("Unexpected error: " +
 cause.getMessage());
                }
            }
        });

        future.join();
        return result[0];
    }
```

- API-Details finden Sie DescribeEndpoint in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun describeEndpoint(): String? {
    val request = DescribeEndpointRequest {}
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val endpointResponse = iotClient.describeEndpoint(request)
        val endpointUrl: String? = endpointResponse.endpointAddress
        val exString: String = getValue(endpointUrl)
        val fullEndpoint = "https://$exString-ats.iot.us-east-1.amazonaws.com"
        println("Full endpoint URL: $fullEndpoint")
        return fullEndpoint
    }
}
```

- API-Details finden Sie DescribeEndpoint in der AWS SDK für die Kotlin-API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
```

```
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def describe_endpoint(self, endpoint_type="iot:Data-ATS"):
        """
        Gets the AWS IoT endpoint.

        :param endpoint_type: The endpoint type.
        :return: The endpoint.
        """
        try:
            response =
self.iot_client.describe_endpoint(endpointType=endpoint_type)
            logger.info("Retrieved endpoint %s.", response["endpointAddress"])
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't describe endpoint. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response["endpointAddress"]
```

- API-Details finden Sie DescribeEndpoint in der AWS API-Referenz zum SDK für Python (Boto3).

Rust

SDK für Rust

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
async fn show_address(client: &Client, endpoint_type: &str) -> Result<(), Error>
 {
    let resp = client
        .describe_endpoint()
        .endpoint_type(endpoint_type)
        .send()
        .await?;

    println!("Endpoint address: {}", resp.endpoint_address.unwrap());

    println!();

    Ok(())
}
```

- API-Details finden Sie DescribeEndpoint in der AWS SDK for Rust API-Referenz.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es läuft noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
    DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
```

```
    DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
    " iv_endpoint_type = 'iot:Data-ATS' - Endpoint type for data operations
    TRY.
        DATA(lo_result) = lo_iot->describeendpoint( iv_endpointtype =
iv_endpoint_type ).
        ov_endpoint_address = lo_result->get_endpointaddress( ).
        MESSAGE |Endpoint address: { ov_endpoint_address }| TYPE 'I'.
      CATCH /aws1/cx_iotthrottlingex INTO DATA(lo_throttle_ex).
        MESSAGE 'Request throttled. Please try again later.' TYPE 'I'.
        RAISE EXCEPTION lo_throttle_ex.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_ex).
        MESSAGE lo_ex->get_text( ) TYPE 'I'.
        RAISE EXCEPTION lo_ex.
    ENDTRY.
```

- API-Details finden Sie DescribeEndpoint in der AWS SDK for SAP ABAP API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden AWS IoT mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **`DescribeThing`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `DescribeThing` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

C++

SDK für C++

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Describe an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::describeThing(const Aws::String &thingName,
                                const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::DescribeThingRequest request;
    request.SetThingName(thingName);

    Aws::IoT::Model::DescribeThingOutcome outcome =
 iotClient.DescribeThing(request);

    if (outcome.IsSuccess()) {
        const Aws::IoT::Model::DescribeThingResult &result = outcome.GetResult();
        std::cout << "Retrieved thing '" << result.GetThingName() << "'" <<
 std::endl;
        std::cout << "thingArn: " << result.GetThingArn() << std::endl;
        std::cout << result.GetAttributes().size() << " attribute(s) retrieved"
                  << std::endl;
        for (const auto &attribute: result.GetAttributes()) {
            std::cout << "  attribute: " << attribute.first << "=" <<
 attribute.second
                      << std::endl;
        }
    }
    else {
        std::cerr << "Error describing thing " << thingName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API-Details finden Sie DescribeThing in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So zeigen Sie detaillierte Informationen zu einem Objekt an

Im folgenden `describe-thing` Beispiel werden Informationen zu einer Sache (Gerät) angezeigt, die in der AWS IoT-Registrierung für Ihr AWS Konto definiert ist.

as iot describe-thing --thing-name MyLightBulb

Ausgabe:

```
{
    "defaultClientId": "MyLightBulb",
    "thingName": "MyLightBulb",
    "thingId": "40da2e73-c6af-406e-b415-15acae538797",
    "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyLightBulb",
    "thingTypeName": "LightBulb",
    "attributes": {
        "model": "123",
        "wattage": "75"
    },
    "version": 1
}
```

Weitere Informationen finden Sie unter Objektverwaltung mit der Registry im Entwicklerhandbuch für AWS IoT.

- API-Details finden Sie in der Befehlsreferenz. DescribeThing AWS CLI

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /**
```

```
     * Describes an IoT Thing asynchronously.
     *
     * @param thingName The name of the IoT Thing.
     *
     * This method initiates an asynchronous request to describe an IoT Thing.
     * If the request is successful, it prints the Thing details.
     * If an exception occurs, it prints the error message.
     */
    private void describeThing(String thingName) {
        DescribeThingRequest thingRequest = DescribeThingRequest.builder()
            .thingName(thingName)
            .build();

        CompletableFuture<DescribeThingResponse> future =
getAsyncClient().describeThing(thingRequest);
        future.whenComplete((describeResponse, ex) -> {
            if (describeResponse != null) {
                System.out.println("Thing Details:");
                System.out.println("Thing Name: " +
describeResponse.thingName());
                System.out.println("Thing ARN: " + describeResponse.thingArn());
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to describe Thing.");
                }
            }
        });

        future.join();
    }
```

- API-Details finden Sie DescribeThing in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun describeThing(thingNameVal: String) {
    val thingRequest =
        DescribeThingRequest {
            thingName = thingNameVal
        }

    // Print Thing details.
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val describeResponse = iotClient.describeThing(thingRequest)
        println("Thing details:")
        println("Thing name: ${describeResponse.thingName}")
        println("Thing ARN:  ${describeResponse.thingArn}")
    }
}
```

- API-Details finden Sie DescribeThing in der AWS SDK für die Kotlin-API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden AWS IoT mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **`DetachThingPrincipal`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `DetachThingPrincipal` verwendet wird.

Aktionsbeispiele sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Sie können diese Aktion in den folgenden Codebeispielen im Kontext sehen:

- Kennenlernen der Grundlagen

- Erste Schritte mit Nachrichten im Internet der Dinge

.NET

SDK für .NET (v4)

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Detaches a certificate from an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to detach.</
param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DetachThingPrincipalAsync(string thingName, string
 certificateArn)
    {
        try
        {
            var request = new DetachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.DetachThingPrincipalAsync(request);
            _logger.LogInformation($"Detached certificate {certificateArn} from
 Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot detach certificate - resource not found:
 {ex.Message}");
            return false;
        }
        catch (Exception ex)
```

```
        {
            _logger.LogError($"Couldn't detach certificate from Thing. Here's
 why: {ex.Message}");
            return false;
        }
    }
```

- API-Details finden Sie DetachThingPrincipal in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Detach a principal from an AWS IoT thing.
/*!
  \param principal: A principal to detach.
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::detachThingPrincipal(const Aws::String &principal,
                                       const Aws::String &thingName,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::DetachThingPrincipalRequest detachThingPrincipalRequest;
    detachThingPrincipalRequest.SetThingName(thingName);
    detachThingPrincipalRequest.SetPrincipal(principal);

    Aws::IoT::Model::DetachThingPrincipalOutcome outcome =
 iotClient.DetachThingPrincipal(
            detachThingPrincipalRequest);

    if (outcome.IsSuccess()) {
```

```
        std::cout << "Successfully detached principal " << principal << " from
 thing "
                  << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to detach principal " << principal << " from thing "
                  << thingName << ": "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API-Details finden Sie DetachThingPrincipal in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

Um eine certificate/principal von einer Sache zu trennen

Im folgenden Beispiel für `detach-thing-principal` wird ein Zertifikat, das einen Prinzipal darstellt, aus dem angegebenen Objekt entfernt.

```
aws iot detach-thing-principal \
    --thing-name "MyLightBulb" \
    --principal "arn:aws:iot:us-
west-2:123456789012:cert/604c48437a57b7d5fc5d137c5be75011c6ee67c9a6943683a1acb4b1626bac36
```

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Weitere Informationen finden Sie unter Objektverwaltung mit der Registry im Entwicklerhandbuch für AWS IoT.

- API-Details finden Sie DetachThingPrincipal in der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   /**
    * Detaches a principal (certificate) from an IoT Thing asynchronously.
    *
    * @param thingName The name of the IoT Thing.
    * @param certificateArn The ARN of the certificate to detach.
    *
    * This method initiates an asynchronous request to detach a certificate from
an IoT Thing.
    * If the detachment is successful, it prints a confirmation message.
    * If an exception occurs, it prints the error message.
    */
   public void detachThingPrincipal(String thingName, String certificateArn) {
       DetachThingPrincipalRequest thingPrincipalRequest =
DetachThingPrincipalRequest.builder()
           .principal(certificateArn)
           .thingName(thingName)
           .build();

       CompletableFuture<DetachThingPrincipalResponse> future =
getAsyncClient().detachThingPrincipal(thingPrincipalRequest);
       future.whenComplete((voidResult, ex) -> {
           if (ex == null) {
               System.out.println(certificateArn + " was successfully removed
from " + thingName);
           } else {
               Throwable cause = ex.getCause();
               if (cause instanceof IotException) {
                   System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
               } else {
                   System.err.println("Unexpected error: " + ex.getMessage());
               }
```

```
            }
        });

        future.join();
    }
```

- API-Details finden Sie DetachThingPrincipal in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun detachThingPrincipal(
    thingNameVal: String,
    certificateArn: String,
) {
    val thingPrincipalRequest =
        DetachThingPrincipalRequest {
            principal = certificateArn
            thingName = thingNameVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        iotClient.detachThingPrincipal(thingPrincipalRequest)
        println("$certificateArn was successfully removed from $thingNameVal")
    }
}
```

- API-Details finden Sie DetachThingPrincipal in der AWS SDK für die Kotlin-API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def detach_thing_principal(self, thing_name, principal):
        """
        Detaches a certificate from an AWS IoT thing.

        :param thing_name: The name of the thing.
        :param principal: The ARN of the certificate.
        """
        try:
            self.iot_client.detach_thing_principal(
                thingName=thing_name, principal=principal
            )
            logger.info("Detached principal %s from thing %s.", principal,
 thing_name)
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
```

```
                logger.error("Cannot detach principal. Resource not found.")
                return
            logger.error(
                "Couldn't detach principal from thing. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API-Details finden Sie DetachThingPrincipal in der AWS API-Referenz zum SDK für Python (Boto3).

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es läuft noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
   DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
   DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
   TRY.
       lo_iot->detachthingprincipal(
         iv_thingname = iv_thing_name
         iv_principal = iv_principal ).
       MESSAGE |Principal detached from IoT thing '{ iv_thing_name }'.| TYPE
'I'.
     CATCH /aws1/cx_iotresourcenotfoundex INTO DATA(lo_ex).
       MESSAGE |Resource not found when detaching principal from
'{ iv_thing_name }'.| TYPE 'I'.
       RAISE EXCEPTION lo_ex.
     CATCH /aws1/cx_rt_service_generic INTO DATA(lo_svc_ex).
       MESSAGE lo_svc_ex->get_text( ) TYPE 'I'.
       RAISE EXCEPTION lo_svc_ex.
   ENDTRY.
```

- API-Details finden Sie DetachThingPrincipal in der AWS SDK for SAP ABAP API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden AWS IoT mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **`ListCertificates`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `ListCertificates` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Lists all certificates associated with the account.
    /// </summary>
    /// <returns>List of certificate information, or empty list if listing
failed.</returns>
    public async Task<List<Certificate>> ListCertificatesAsync()
    {
        try
        {
            var request = new ListCertificatesRequest();
            var response = await _amazonIoT.ListCertificatesAsync(request);
```

```
            _logger.LogInformation($"Retrieved {response.Certificates.Count}
 certificates");
            return response.Certificates;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return new List<Certificate>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list certificates. Here's why:
 {ex.Message}");
            return new List<Certificate>();
        }
    }
```

- API-Details finden Sie ListCertificates in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! List certificates registered in the AWS account making the call.
/*!
   \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::listCertificates(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::ListCertificatesRequest request;

    Aws::Vector<Aws::IoT::Model::Certificate> allCertificates;
```

```
    Aws::String marker; // Used to paginate results.
    do {
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::IoT::Model::ListCertificatesOutcome outcome =
 iotClient.ListCertificates(
                request);

        if (outcome.IsSuccess()) {
            const Aws::IoT::Model::ListCertificatesResult &result =
 outcome.GetResult();
            marker = result.GetNextMarker();
            allCertificates.insert(allCertificates.end(),
                                   result.GetCertificates().begin(),
                                   result.GetCertificates().end());
        }
        else {
            std::cerr << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
            return false;
        }
    } while (!marker.empty());

    std::cout << allCertificates.size() << " certificate(s) found." << std::endl;

    for (auto &certificate: allCertificates) {
        std::cout << "Certificate ID: " << certificate.GetCertificateId() <<
 std::endl;
        std::cout << "Certificate ARN: " << certificate.GetCertificateArn()
                  << std::endl;
        std::cout << std::endl;
    }

    return true;
}
```

- API-Details finden Sie ListCertificates in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

Beispiel 1: Um die in Ihrem AWS Konto registrierten Zertifikate aufzulisten

Im folgenden Beispiel für `list-certificates` werden alle Zertifikate aufgelistet, die in Ihrem Konto registriert sind. Wenn die Anzahl das standardmäßige Paging-Limit von 25 überschreitet, können Sie den Antwortwert `nextMarker` aus diesem Befehl an den nächsten Befehl übergeben, um den nächsten Ergebnisstapel zu erhalten. Wiederholen Sie den Vorgang, bis `nextMarker` keinen Wert mehr zurückgibt.

```
aws iot list-certificates
```

Ausgabe:

```
{
    "certificates": [
        {
            "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/604c48437a57b7d5fc5d137c5be75011c6ee67c9a6943683a1acb4b1626bac36
            "certificateId":
 "604c48437a57b7d5fc5d137c5be75011c6ee67c9a6943683a1acb4b1626bac36",
            "status": "ACTIVE",
            "creationDate": 1556810537.617
        },
        {
            "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/262a1ac8a7d8aa72f6e96e365480f7313aa9db74b8339ec65d34dc3074e1c31e
            "certificateId":
 "262a1ac8a7d8aa72f6e96e365480f7313aa9db74b8339ec65d34dc3074e1c31e",
            "status": "ACTIVE",
            "creationDate": 1546447050.885
        },
        {
            "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
b193ab7162c0fadca83246d24fa090300a1236fe58137e121b011804d8ac1d6b",
            "certificateId":
 "b193ab7162c0fadca83246d24fa090300a1236fe58137e121b011804d8ac1d6b",
            "status": "ACTIVE",
            "creationDate": 1546292258.322
        },
```

```
        {
            "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/7aebeea3845d14a44ec80b06b8b78a89f3f8a706974b8b34d18f5adf0741db42
            "certificateId":
 "7aebeea3845d14a44ec80b06b8b78a89f3f8a706974b8b34d18f5adf0741db42",
            "status": "ACTIVE",
            "creationDate": 1541457693.453
        },
        {
            "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/54458aa39ebb3eb39c91ffbbdcc3a6ca1c7c094d1644b889f735a6fc2cd9a7e3
            "certificateId":
 "54458aa39ebb3eb39c91ffbbdcc3a6ca1c7c094d1644b889f735a6fc2cd9a7e3",
            "status": "ACTIVE",
            "creationDate": 1541113568.611
        },
        {
            "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/4f0ba725787aa94d67d2fca420eca022242532e8b3c58e7465c7778b443fd65e
            "certificateId":
 "4f0ba725787aa94d67d2fca420eca022242532e8b3c58e7465c7778b443fd65e",
            "status": "ACTIVE",
            "creationDate": 1541022751.983
        }
    ]
}
```

- API-Details finden Sie ListCertificates in der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /**
     * Lists all certificates asynchronously.
     *
```

```
     * This method initiates an asynchronous request to list all certificates.
     * If the request is successful, it prints the certificate IDs and ARNs.
     * If an exception occurs, it prints the error message.
     */
    public void listCertificates() {
        CompletableFuture<ListCertificatesResponse> future =
getAsyncClient().listCertificates();
        future.whenComplete((response, ex) -> {
            if (response != null) {
                List<Certificate> certList = response.certificates();
                for (Certificate cert : certList) {
                    System.out.println("Cert id: " + cert.certificateId());
                    System.out.println("Cert Arn: " + cert.certificateArn());
                }
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
                    System.err.println(((IotException)
cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
cause.getMessage());
                } else {
                    System.err.println("Failed to list certificates.");
                }
            }
        });

        future.join();
    }
```

- API-Details finden Sie ListCertificates in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun listCertificates() {
    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val response = iotClient.listCertificates()
        val certList = response.certificates
        certList?.forEach { cert ->
            println("Cert id: ${cert.certificateId}")
            println("Cert Arn: ${cert.certificateArn}")
        }
    }
}
```

- API-Details finden Sie ListCertificates in der AWS SDK für die Kotlin-API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)
```

```
    def list_certificates(self):
        """
        Lists AWS IoT certificates.

        :return: The list of certificates.
        """
        try:
            certificates = []
            paginator = self.iot_client.get_paginator("list_certificates")
            for page in paginator.paginate():
                certificates.extend(page["certificates"])
            logger.info("Retrieved %s certificates.", len(certificates))
            return certificates
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't list certificates. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

- API-Details finden Sie ListCertificates in der AWS API-Referenz zum SDK für Python (Boto3).

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es läuft noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
    DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
```

```
    DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
    TRY.
        " Collect all certificates by following the pagination marker.
        DATA lv_marker TYPE /aws1/iotmarker.
        DATA lv_count  TYPE i.

        DO.
          oo_result = lo_iot->listcertificates( iv_marker = lv_marker ).
          lv_count  = lv_count + lines( oo_result->get_certificates( ) ).
          lv_marker = oo_result->get_nextmarker( ).
          IF lv_marker IS INITIAL.
            EXIT.
          ENDIF.
        ENDDO.

        MESSAGE |Retrieved { lv_count } IoT certificates.| TYPE 'I'.
      CATCH /aws1/cx_iotthrottlingex INTO DATA(lo_throttle_ex).
        MESSAGE 'Request throttled. Please try again later.' TYPE 'I'.
        RAISE EXCEPTION lo_throttle_ex.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_ex).
        MESSAGE lo_ex->get_text( ) TYPE 'I'.
        RAISE EXCEPTION lo_ex.
    ENDTRY.
```

- API-Details finden Sie ListCertificates in der AWS SDK for SAP ABAP API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden AWS IoT mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **`ListThings`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `ListThings` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Lists IoT Things with pagination support.
    /// </summary>
    /// <returns>List of Things, or empty list if listing failed.</returns>
    public async Task<List<ThingAttribute>> ListThingsAsync()
    {
        try
        {
            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await _amazonIoT.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the
next page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await _amazonIoT.ListThingsAsync(request);
                }
                else
```

```
                    hasMoreResults = false;
            }

            _logger.LogInformation($"Retrieved {things.Count} Things");
            return things;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingAttribute>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list Things. Here's why: {ex.Message}");
            return new List<ThingAttribute>();
        }
    }
```

- API-Details finden Sie ListThings in der AWS SDK für .NET API-Referenz.

CLI

AWS CLI

Beispiel 1: So listen Sie alle Objekte in der Registrierung auf

Das folgende `list-things` Beispiel listet die Dinge (Geräte) auf, die in der AWS IoT-Registrierung für Ihr AWS Konto definiert sind.

```
aws iot list-things
```

Ausgabe:

```
{
    "things": [
        {
            "thingName": "ThirdBulb",
            "thingTypeName": "LightBulb",
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/ThirdBulb",
            "attributes": {
```

```
                "model": "123",
                "wattage": "75"
            },
            "version": 2
        },
        {
            "thingName": "MyOtherLightBulb",
            "thingTypeName": "LightBulb",
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/
MyOtherLightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 3
        },
        {
            "thingName": "MyLightBulb",
            "thingTypeName": "LightBulb",
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyLightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1
        },
        {
        "thingName": "SampleIoTThing",
        "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/SampleIoTThing",
        "attributes": {},
        "version": 1
        }
    ]
}
```

Beispiel 2: So listen Sie die definierten Objekte auf, die ein bestimmtes Attribut haben

Im folgenden Beispiel für `list-things` wird eine Liste von Objekten angezeigt, die ein Attribut mit dem Namen `wattage` haben.

```
aws iot list-things \
    --attribute-name wattage
```

Ausgabe:

```
{
    "things": [
        {
            "thingName": "MyLightBulb",
            "thingTypeName": "LightBulb",
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyLightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1
        },
        {
            "thingName": "MyOtherLightBulb",
            "thingTypeName": "LightBulb",
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/
MyOtherLightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 3
        }
    ]
}
```

Weitere Informationen finden Sie unter Objektverwaltung mit der Registry im Entwicklerhandbuch für AWS IoT.

- API-Details finden Sie ListThings in der AWS CLI Befehlsreferenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def list_things(self):
        """
        Lists AWS IoT things.

        :return: The list of things.
        """
        try:
            things = []
            paginator = self.iot_client.get_paginator("list_things")
            for page in paginator.paginate():
                things.extend(page["things"])
            logger.info("Retrieved %s things.", len(things))
            return things
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't list things. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
```

- API-Details finden Sie ListThings in der AWS API-Referenz zum SDK für Python (Boto3).

Rust

SDK für Rust

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
async fn show_things(client: &Client) -> Result<(), Error> {
    let resp = client.list_things().send().await?;

    println!("Things:");

    for thing in resp.things.unwrap() {
        println!(
            "  Name:  {}",
            thing.thing_name.as_deref().unwrap_or_default()
        );
        println!(
            "  Type:  {}",
            thing.thing_type_name.as_deref().unwrap_or_default()
        );
        println!(
            "  ARN:   {}",
            thing.thing_arn.as_deref().unwrap_or_default()
        );
        println!();
    }

    println!();

    Ok(())
}
```

- API-Details finden Sie ListThings in der AWS SDK for Rust API-Referenz.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es läuft noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
TRY.
    " Collect all things by following the pagination token.
    DATA lv_nexttoken TYPE /aws1/iotnexttoken.
    DATA lv_count     TYPE i.

    DO.
      oo_result    = lo_iot->listthings( iv_nexttoken = lv_nexttoken ).
      lv_count     = lv_count + lines( oo_result->get_things( ) ).
      lv_nexttoken = oo_result->get_nexttoken( ).
      IF lv_nexttoken IS INITIAL.
        EXIT.
      ENDIF.
    ENDDO.

    MESSAGE |Retrieved { lv_count } IoT things.| TYPE 'I'.
  CATCH /aws1/cx_iotthrottlingex INTO DATA(lo_throttle_ex).
    MESSAGE 'Request throttled. Please try again later.' TYPE 'I'.
    RAISE EXCEPTION lo_throttle_ex.
  CATCH /aws1/cx_rt_service_generic INTO DATA(lo_ex).
    MESSAGE lo_ex->get_text( ) TYPE 'I'.
    RAISE EXCEPTION lo_ex.
ENDTRY.
```

- API-Details finden Sie ListThings in der AWS SDK for SAP ABAP API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden AWS IoT mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **`SearchIndex`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `SearchIndex` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Searches for IoT Things using the search index.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
search failed.</returns>
    public async Task<List<ThingDocument>> SearchIndexAsync(string queryString)
    {
        try
        {
            // First, try to perform the search
            var request = new SearchIndexRequest
            {
                QueryString = queryString
            };

            var response = await _amazonIoT.SearchIndexAsync(request);
```

```
            _logger.LogInformation($"Search found {response.Things.Count}
Things");
            return response.Things;
        }
        catch (Amazon.IoT.Model.IndexNotReadyException ex)
        {
            _logger.LogWarning($"Search index not ready, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
        {
            _logger.LogWarning($"Search index not configured, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't search index. Here's why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Sets up the indexing configuration and retries the search after waiting
for the index to be ready.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
setup/search failed.</returns>
    private async Task<List<ThingDocument>> SetupIndexAndRetrySearchAsync(string
queryString)
    {
        try
        {
            // Update indexing configuration to REGISTRY mode
```

```
            _logger.LogInformation("Setting up IoT search indexing
configuration...");
            await _amazonIoT.UpdateIndexingConfigurationAsync(
                new UpdateIndexingConfigurationRequest()
                {
                    ThingIndexingConfiguration = new ThingIndexingConfiguration()
                    {
                        ThingIndexingMode = ThingIndexingMode.REGISTRY
                    }
                });

            _logger.LogInformation("Indexing configuration updated. Waiting for
index to be ready...");

            // Wait for the index to be set up - this can take some time
            const int maxRetries = 10;
            const int retryDelaySeconds = 10;

            for (int attempt = 1; attempt <= maxRetries; attempt++)
            {
                try
                {
                    _logger.LogInformation($"Waiting for index to be ready
(attempt {attempt}/{maxRetries})...");
                    await Task.Delay(TimeSpan.FromSeconds(retryDelaySeconds));

                    // Try to get the current indexing configuration to see if
it's ready
                    var configResponse = await
_amazonIoT.GetIndexingConfigurationAsync(new GetIndexingConfigurationRequest());
                    if
(configResponse.ThingIndexingConfiguration?.ThingIndexingMode ==
ThingIndexingMode.REGISTRY)
                    {
                        // Try the search again
                        var request = new SearchIndexRequest
                        {
                            QueryString = queryString
                        };

                        var response = await
_amazonIoT.SearchIndexAsync(request);
                        _logger.LogInformation($"Search found
{response.Things.Count} Things after index setup");
```

```
                    return response.Things;
                }
            }
            catch (Amazon.IoT.Model.IndexNotReadyException)
            {
                // Index still not ready, continue waiting
                _logger.LogInformation("Index still not ready, continuing to
wait...");
                continue;
            }
            catch (Amazon.IoT.Model.InvalidRequestException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
            {
                // Index still not ready, continue waiting
                _logger.LogInformation("Index still not ready, continuing to
wait...");
                continue;
            }
        }

        _logger.LogWarning("Timeout waiting for search index to be ready
after configuration update");
        return new List<ThingDocument>();
    }
    catch (Exception ex)
    {
        _logger.LogError($"Couldn't set up search index configuration. Here's
why: {ex.Message}");
        return new List<ThingDocument>();
    }
}
```

- API-Details finden Sie SearchIndex in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Query the AWS IoT fleet index.
//! For query information, see https://docs.aws.amazon.com/iot/latest/
developerguide/query-syntax.html
/*!
  \param: query: The query string.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::searchIndex(const Aws::String &query,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::SearchIndexRequest request;
    request.SetQueryString(query);

    Aws::Vector<Aws::IoT::Model::ThingDocument> allThingDocuments;
    Aws::String nextToken; // Used for pagination.
    do {
        if (!nextToken.empty()) {
            request.SetNextToken(nextToken);
        }

        Aws::IoT::Model::SearchIndexOutcome outcome =
 iotClient.SearchIndex(request);

        if (outcome.IsSuccess()) {
            const Aws::IoT::Model::SearchIndexResult &result =
 outcome.GetResult();
            allThingDocuments.insert(allThingDocuments.end(),
                                     result.GetThings().cbegin(),
                                     result.GetThings().cend());
```

```
            nextToken = result.GetNextToken();

        }
        else {
            std::cerr << "Error in SearchIndex: " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!nextToken.empty());

    std::cout << allThingDocuments.size() << " thing document(s) found." <<
 std::endl;
    for (const auto thingDocument: allThingDocuments) {
        std::cout << "  Thing name: " << thingDocument.GetThingName() << "."
                  << std::endl;
    }
    return true;
}
```

- API-Details finden Sie SearchIndex in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So fragen Sie den Objektindex ab

Im folgenden Beispiel für `search-index` wird der `AWS_Things`-Index nach Objekten abgefragt, die den Typ `LightBulb` haben.

```
aws iot search-index \
    --index-name "AWS_Things" \
    --query-string "thingTypeName:LightBulb"
```

Ausgabe:

```
{
    "things": [
        {
            "thingName": "MyLightBulb",
```

```
            "thingId": "40da2e73-c6af-406e-b415-15acae538797",
            "thingTypeName": "LightBulb",
            "thingGroupNames": [
                "LightBulbs",
                "DeadBulbs"
            ],
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "connectivity": {
                "connected": false
            }
        },
        {
            "thingName": "ThirdBulb",
            "thingId": "615c8455-33d5-40e8-95fd-3ee8b24490af",
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "connectivity": {
                "connected": false
            }
        },
        {
            "thingName": "MyOtherLightBulb",
            "thingId": "6dae0d3f-40c1-476a-80c4-1ed24ba6aa11",
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "connectivity": {
                "connected": false
            }
        }
    ]
}
```

Weitere Informationen finden Sie unter Verwalten der Objektindizierung im Entwicklerhandbuch für AWS IoT.

- API-Details finden Sie SearchIndex in der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /**
     * Searches for IoT Things asynchronously based on a query string.
     *
     * @param queryString The query string to search for Things.
     *
     * This method initiates an asynchronous request to search for IoT Things.
     * If the request is successful and Things are found, it prints their IDs.
     * If no Things are found, it prints a message indicating so.
     * If an exception occurs, it prints the error message.
     */
    public void searchThings(String queryString) {
        SearchIndexRequest searchIndexRequest = SearchIndexRequest.builder()
            .queryString(queryString)
            .build();

        CompletableFuture<SearchIndexResponse> future =
getAsyncClient().searchIndex(searchIndexRequest);
        future.whenComplete((searchIndexResponse, ex) -> {
            if (searchIndexResponse != null) {
                // Process the result.
                if (searchIndexResponse.things().isEmpty()) {
                    System.out.println("No things found.");
                } else {
                    searchIndexResponse.things().forEach(thing ->
System.out.println("Thing id found using search is " + thing.thingId()));
                }
            } else {
                Throwable cause = ex != null ? ex.getCause() : null;
                if (cause instanceof IotException) {
```

```
                    System.err.println(((IotException)
 cause).awsErrorDetails().errorMessage());
                } else if (cause != null) {
                    System.err.println("Unexpected error: " +
 cause.getMessage());
                } else {
                    System.err.println("Failed to search for IoT Things.");
                }
            }
        });

        future.join();
    }
```

- API-Details finden Sie SearchIndex in der AWS SDK for Java 2.x API-Referenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun searchThings(queryStringVal: String?) {
    val searchIndexRequest =
        SearchIndexRequest {
            queryString = queryStringVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        val searchIndexResponse = iotClient.searchIndex(searchIndexRequest)
        if (searchIndexResponse.things?.isEmpty() == true) {
            println("No things found.")
        } else {
            searchIndexResponse.things
                ?.forEach { thing -> println("Thing id found using search is
 ${thing.thingId}") }
        }
```

```
    }
}
```

- API-Details finden Sie SearchIndex in der AWS SDK für die Kotlin-API-Referenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client

    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def search_index(self, query):
        """
        Searches the AWS IoT index.

        :param query: The search query.
        :return: The list of things found.
        """
        try:
            response = self.iot_client.search_index(queryString=query)
```

```
            logger.info("Found %s things.", len(response.get("things", [])))
        except ClientError as err:
            if err.response["Error"]["Code"] == "ThrottlingException":
                logger.error("Request throttled. Please try again later.")
            else:
                logger.error(
                    "Couldn't search index. Here's why: %s: %s",
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
            raise
        else:
            return response.get("things", [])
```

- API-Details finden Sie SearchIndex in der AWS API-Referenz zum SDK für Python (Boto3).

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es läuft noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
   DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
   DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
   " iv_query_string = 'thingName:MyThing*' - Fleet indexing query string
   TRY.
       oo_result = lo_iot->searchindex( iv_querystring = iv_query_string ).
       MESSAGE |Found { lines( oo_result->get_things( ) ) } IoT things matching
query.| TYPE 'I'.
     CATCH /aws1/cx_iotindexnotreadyex INTO DATA(lo_idx_ex).
       MESSAGE 'Fleet indexing is not ready. Enable indexing with
UpdateIndexingConfiguration first.' TYPE 'I'.
       RAISE EXCEPTION lo_idx_ex.
     CATCH /aws1/cx_iotthrottlingex INTO DATA(lo_throttle_ex).
       MESSAGE 'Request throttled. Please try again later.' TYPE 'I'.
```

```
        RAISE EXCEPTION lo_throttle_ex.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_ex).
        MESSAGE lo_ex->get_text( ) TYPE 'I'.
        RAISE EXCEPTION lo_ex.
    ENDTRY.
```

- API-Details finden Sie SearchIndex in der AWS SDK for SAP ABAP API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden AWS IoT mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **`UpdateIndexingConfiguration`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `UpdateIndexingConfiguration` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

C++

SDK für C++

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Update the indexing configuration.
/*!
  \param thingIndexingConfiguration: A ThingIndexingConfiguration object which is
 ignored if not set.
  \param thingGroupIndexingConfiguration: A ThingGroupIndexingConfiguration
 object which is ignored if not set.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
```

```
 */
bool AwsDoc::IoT::updateIndexingConfiguration(
        const Aws::IoT::Model::ThingIndexingConfiguration
 &thingIndexingConfiguration,
        const Aws::IoT::Model::ThingGroupIndexingConfiguration
 &thingGroupIndexingConfiguration,
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::UpdateIndexingConfigurationRequest request;

    if (thingIndexingConfiguration.ThingIndexingModeHasBeenSet()) {
        request.SetThingIndexingConfiguration(thingIndexingConfiguration);
    }

    if (thingGroupIndexingConfiguration.ThingGroupIndexingModeHasBeenSet()) {

 request.SetThingGroupIndexingConfiguration(thingGroupIndexingConfiguration);
    }

    Aws::IoT::Model::UpdateIndexingConfigurationOutcome outcome =
 iotClient.UpdateIndexingConfiguration(
            request);

    if (outcome.IsSuccess()) {
        std::cout << "UpdateIndexingConfiguration succeeded." << std::endl;
    }
    else {
        std::cerr << "UpdateIndexingConfiguration failed."
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API-Details finden Sie UpdateIndexingConfiguration in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So aktivieren Sie die Indizierung von Objekten

Das folgende `update-indexing-configuration` Beispiel aktiviert die Dingindizierung, um die Suche nach Registrierungsdaten, Schattendaten und dem Konnektivitätsstatus von AWS Dingen mithilfe des _Things-Indexes zu unterstützen.

```
aws iot update-indexing-configuration
    --thing-indexing-
configuration thingIndexingMode=REGISTRY_AND_SHADOW,thingConnectivityIndexingMode=STATUS
```

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Weitere Informationen finden Sie unter Verwalten der Objektindizierung im Entwicklerhandbuch für AWS IoT.

- API-Details finden Sie UpdateIndexingConfiguration in der AWS CLI Befehlsreferenz.

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class IoTWrapper:
    """Encapsulates AWS IoT actions."""

    def __init__(self, iot_client, iot_data_client=None):
        """
        :param iot_client: A Boto3 AWS IoT client.
        :param iot_data_client: A Boto3 AWS IoT Data Plane client.
        """
        self.iot_client = iot_client
        self.iot_data_client = iot_data_client
```

```
    @classmethod
    def from_client(cls):
        iot_client = boto3.client("iot")
        iot_data_client = boto3.client("iot-data")
        return cls(iot_client, iot_data_client)

    def update_indexing_configuration(self):
        """
        Updates the AWS IoT indexing configuration to enable thing indexing.
        """
        try:
            self.iot_client.update_indexing_configuration(
                thingIndexingConfiguration={"thingIndexingMode": "REGISTRY"}
            )
            logger.info("Updated indexing configuration.")
        except ClientError as err:
            logger.error(
                "Couldn't update indexing configuration. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API-Details finden Sie UpdateIndexingConfiguration in der AWS API-Referenz zum SDK für Python (Boto3).

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es läuft noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    CONSTANTS cv_pfl TYPE /aws1/rt_profile_id VALUE 'ZCODE_DEMO'.
    DATA(lo_session) = /aws1/cl_rt_session_aws=>create( cv_pfl ).
    DATA(lo_iot) = /aws1/cl_iot_factory=>create( lo_session ).
```

```
    TRY.
        DATA lo_idx_conf TYPE REF TO /aws1/cl_iotthingindexingconf.
        CREATE OBJECT lo_idx_conf
          EXPORTING
            iv_thingindexingmode = 'REGISTRY'.
        lo_iot->updateindexingconfiguration(
          io_thingindexingconf = lo_idx_conf ).
        MESSAGE 'IoT thing indexing configuration updated to REGISTRY mode.' TYPE
'I'.
      CATCH /aws1/cx_rt_service_generic INTO DATA(lo_ex).
        MESSAGE lo_ex->get_text( ) TYPE 'I'.
        RAISE EXCEPTION lo_ex.
    ENDTRY.
```

- API-Details finden Sie UpdateIndexingConfiguration in der AWS SDK for SAP ABAP API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden AWS IoT mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwenden Sie **`UpdateThing`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `UpdateThing` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET (v4)

> **Note**
>
> Es läuft noch mehr GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <summary>
    /// Updates an IoT Thing with attributes.
    /// </summary>
    /// <param name="thingName">The name of the Thing to update.</param>
    /// <param name="attributes">Dictionary of attributes to add.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingAsync(string thingName, Dictionary<string,
string> attributes)
    {
        try
        {
            var request = new UpdateThingRequest
            {
                ThingName = thingName,
                AttributePayload = new AttributePayload
                {
                    Attributes = attributes,
                    Merge = true
                }
            };

            await _amazonIoT.UpdateThingAsync(request);
            _logger.LogInformation($"Updated Thing {thingName} with attributes");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing attributes. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- API-Details finden Sie UpdateThing in der AWS SDK für .NET API-Referenz.

C++

SDK für C++

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
//! Update an AWS IoT thing with attributes.
/*!
  \param thingName: The name for the thing.
  \param attributeMap: A map of key/value attributes/
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::updateThing(const Aws::String &thingName,
                              const std::map<Aws::String, Aws::String>
 &attributeMap,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::UpdateThingRequest request;
    request.SetThingName(thingName);
    Aws::IoT::Model::AttributePayload attributePayload;
    for (const auto &attribute: attributeMap) {
        attributePayload.AddAttributes(attribute.first, attribute.second);
    }
    request.SetAttributePayload(attributePayload);

    Aws::IoT::Model::UpdateThingOutcome outcome = iotClient.UpdateThing(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to update thing " << thingName << ":" <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- API-Details finden Sie UpdateThing in der AWS SDK für C++ API-Referenz.

CLI

AWS CLI

So weisen Sie einem Objekt einen Objekttyp zu

Das folgende `update-thing` Beispiel ordnet ein Ding in der AWS IoT-Registrierung einem Dingtyp zu. Wenn Sie die Zuordnung vornehmen, geben Sie Werte für die durch den Objekttyp definierten Attribute an.

```
aws iot update-thing \
    --thing-name "MyOtherLightBulb" \
    --thing-type-name "LightBulb" \
    --attribute-payload "{"attributes": {"wattage":"75", "model":"123"}}"
```

Dieser Befehl erzeugt keine Ausgabe. Verwenden Sie den `describe-thing`-Befehl, um das Ergebnis anzuzeigen.

Weitere Informationen finden Sie unter Objekttypen im Entwicklerhandbuch für AWS IoT.

- API-Details finden Sie UpdateThing in der AWS CLI Befehlsreferenz.

Kotlin

SDK für Kotlin

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
suspend fun updateThing(thingNameVal: String?) {
    val newLocation = "Office"
    val newFirmwareVersion = "v2.0"
    val attMap: MutableMap<String, String> = HashMap()
    attMap["location"] = newLocation
```

```
    attMap["firmwareVersion"] = newFirmwareVersion

    val attributePayloadVal =
        AttributePayload {
            attributes = attMap
        }

    val updateThingRequest =
        UpdateThingRequest {
            thingName = thingNameVal
            attributePayload = attributePayloadVal
        }

    IotClient.fromEnvironment { region = "us-east-1" }.use { iotClient ->
        // Update the IoT thing attributes.
        iotClient.updateThing(updateThingRequest)
        println("$thingNameVal attributes updated successfully.")
    }
}
```

- API-Details finden Sie UpdateThing in der AWS SDK für die Kotlin-API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden AWS IoT mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Szenarien für AWS IoT verwenden AWS SDKs

Die folgenden Codebeispiele zeigen Ihnen, wie Sie gängige Szenarien AWS IoT mit AWS SDKs implementieren. Diese Szenarien zeigen Ihnen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb AWS IoT oder in Kombination mit anderen AWS-Services aufrufen. Jedes Szenario enthält einen Link zum vollständigen Quell-Code, wo Sie Anleitungen zum Einrichten und Ausführen des Codes finden.

Szenarien zielen auf eine mittlere Erfahrungsebene ab, um Ihnen zu helfen, Service-Aktionen im Kontext zu verstehen.

Beispiele

- Erste Schritte mit dem Geräteschutz im Internet der Dinge

- Erste Schritte mit Nachrichten im Internet der Dinge

Erste Schritte mit dem Geräteschutz im Internet der Dinge

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie die erforderlichen IAM-Rollen
- Aktivieren Sie die Audit-Checks von IoT Device Defender
- Führen Sie ein On-Demand Audit durch
- Erstellen Sie eine Schadensbegrenzungsmaßnahme
- Wenden Sie Abhilfemaßnahmen auf Ergebnisse an
- Richten Sie SNS-Benachrichtigungen ein (optional)
- Aktivieren Sie die IoT-Protokollierung

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es läuft noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie es im Tutorials und Beispiele-Repository für Entwickler einrichten und ausführen.

```
#!/bin/bash

# AWS IoT Device Defender Getting Started Script
# This script demonstrates how to use AWS IoT Device Defender to enable audit
 checks,
# view audit results, create mitigation actions, and apply them to findings.

set -euo pipefail

# Set up logging
LOG_FILE="iot-device-defender-script-$(date +%Y%m%d%H%M%S).log"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "================================================="
```

```
echo "AWS IoT Device Defender Getting Started Script"
echo "================================================="
echo "Starting script execution at $(date)"
echo ""

# Function to check for errors in command output
check_error() {
    if echo "$1" | grep -iE "An error occurred|Exception|Failed|usage: aws" > /
dev/null; then
        echo "ERROR: Command failed with the following output:"
        echo "$1"
        return 1
    fi
    return 0
}

# Function to safely extract JSON values using jq
extract_json_value() {
    local json="$1"
    local key="$2"
    echo "$json" | jq -r ".${key} // empty" 2>/dev/null || echo ""
}

# Function to validate JSON
validate_json() {
    local json="$1"
    echo "$json" | jq empty 2>/dev/null
}

# Function to check AWS CLI availability
check_aws_cli() {
    if ! command -v aws &> /dev/null; then
        echo "ERROR: AWS CLI is not installed or not in PATH"
        return 1
    fi
    if ! command -v jq &> /dev/null; then
        echo "ERROR: jq is not installed or not in PATH"
        return 1
    fi
    return 0
}

# Function to get AWS account ID
get_account_id() {
```

```
    local account_id
    account_id=$(aws sts get-caller-identity --query 'Account' --output text 2>/
dev/null) || true
    if [ -z "$account_id" ]; then
        echo "ERROR: Could not retrieve AWS account ID"
        return 1
    fi
    echo "$account_id"
    return 0
}

# Function to create IAM roles with retry logic
create_iam_role() {
    local ROLE_NAME=$1
    local TRUST_POLICY=$2
    local MANAGED_POLICY=$3
    local RETRY_COUNT=0
    local MAX_RETRIES=3

    echo "Creating IAM role: $ROLE_NAME"

    # Validate trust policy JSON
    if ! validate_json "$TRUST_POLICY"; then
        echo "ERROR: Invalid trust policy JSON for role $ROLE_NAME"
        return 1
    fi

    # Check if role already exists
    if aws iam get-role --role-name "$ROLE_NAME" >/dev/null 2>&1; then
        echo "Role $ROLE_NAME already exists, skipping creation"
        ROLE_ARN=$(aws iam get-role --role-name "$ROLE_NAME" --query 'Role.Arn'
 --output text 2>/dev/null) || true
        if [ -z "$ROLE_ARN" ]; then
            echo "ERROR: Could not retrieve ARN for existing role $ROLE_NAME"
            return 1
        fi
        echo "Role ARN: $ROLE_ARN"
        return 0
    fi

    # Create the role with trust policy and retry logic
    while [ $RETRY_COUNT -lt $MAX_RETRIES ]; do
        ROLE_RESULT=$(aws iam create-role \
            --role-name "$ROLE_NAME" \
```

```
            --assume-role-policy-document "$TRUST_POLICY" 2>&1) || true

        if check_error "$ROLE_RESULT"; then
            break
        fi

        RETRY_COUNT=$((RETRY_COUNT + 1))
        if [ $RETRY_COUNT -lt $MAX_RETRIES ]; then
            echo "Retrying role creation (attempt $((RETRY_COUNT + 1))/
$MAX_RETRIES)..."
            sleep $((RETRY_COUNT * 2))
        fi
    done

    if ! check_error "$ROLE_RESULT"; then
        echo "Failed to create role $ROLE_NAME after $MAX_RETRIES attempts"
        return 1
    fi

    aws iam tag-role --role-name "$ROLE_NAME" --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-iot-device-defender-gs 2>&1 || true

    # For IoT logging role, create an inline policy instead of using a managed
 policy
    if [[ "$ROLE_NAME" == "AWSIoTLoggingRole" ]]; then
        local LOGGING_POLICY
        LOGGING_POLICY=$(cat <<'EOF'
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:PutMetricFilter",
                "logs:PutRetentionPolicy",
                "logs:GetLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": "arn:aws:logs:*:*:*"
        }
    ]
```

```
}
EOF
)

        if ! validate_json "$LOGGING_POLICY"; then
            echo "ERROR: Invalid logging policy JSON"
            return 1
        fi

        POLICY_RESULT=$(aws iam put-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-name "${ROLE_NAME}Policy" \
            --policy-document "$LOGGING_POLICY" 2>&1) || true

        if ! check_error "$POLICY_RESULT"; then
            echo "Failed to attach inline policy to role $ROLE_NAME"
            return 1
        fi
    elif [[ "$ROLE_NAME" == "IoTMitigationActionErrorLoggingRole" ]]; then
        local MITIGATION_POLICY
        MITIGATION_POLICY=$(cat <<'EOF'
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:UpdateCACertificate",
                "iot:UpdateCertificate",
                "iot:SetV2LoggingOptions",
                "iot:SetLoggingOptions",
                "iot:AddThingToThingGroup"
            ],
            "Resource": "arn:aws:iot:*:*:*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "iot.amazonaws.com"
                }
            }
```

```
        }
    ]
}
EOF
)

        if ! validate_json "$MITIGATION_POLICY"; then
            echo "ERROR: Invalid mitigation policy JSON"
            return 1
        fi

        POLICY_RESULT=$(aws iam put-role-policy \
            --role-name "$ROLE_NAME" \
            --policy-name "${ROLE_NAME}Policy" \
            --policy-document "$MITIGATION_POLICY" 2>&1) || true

        if ! check_error "$POLICY_RESULT"; then
            echo "Failed to attach inline policy to role $ROLE_NAME"
            return 1
        fi
    else
        # Attach managed policy to role if provided
        if [ -n "$MANAGED_POLICY" ]; then
            ATTACH_RESULT=$(aws iam attach-role-policy \
                --role-name "$ROLE_NAME" \
                --policy-arn "$MANAGED_POLICY" 2>&1) || true

            if ! check_error "$ATTACH_RESULT"; then
                echo "Failed to attach policy to role $ROLE_NAME"
                return 1
            fi
        fi
    fi

    echo "Role $ROLE_NAME created successfully"

    # Get the role ARN with error handling
    ROLE_ARN=$(aws iam get-role --role-name "$ROLE_NAME" --query 'Role.Arn' --
output text 2>/dev/null) || true
    if [ -z "$ROLE_ARN" ]; then
        echo "ERROR: Could not retrieve ARN for newly created role $ROLE_NAME"
        return 1
    fi
    echo "Role ARN: $ROLE_ARN"
```

```
    return 0
}

# Array to store created resources for cleanup
declare -a CREATED_RESOURCES

# Validate prerequisites
echo "Validating prerequisites..."
if ! check_aws_cli; then
    echo "ERROR: Prerequisites not met"
    exit 1
fi

ACCOUNT_ID=$(get_account_id) || exit 1
echo "AWS Account ID: $ACCOUNT_ID"
echo ""

# Step 1: Create IAM roles needed for the tutorial
echo "=================================================="
echo "Step 1: Creating required IAM roles"
echo "=================================================="

# Create IoT Device Defender Audit role
IOT_DEFENDER_AUDIT_TRUST_POLICY=$(cat <<'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
)

if ! create_iam_role "AWSIoTDeviceDefenderAuditRole"
 "$IOT_DEFENDER_AUDIT_TRUST_POLICY" "arn:aws:iam::aws:policy/service-role/
AWSIoTDeviceDefenderAudit"; then
    echo "ERROR: Failed to create audit role"
    exit 1
```

```
fi
AUDIT_ROLE_ARN=$ROLE_ARN
CREATED_RESOURCES+=("IAM Role: AWSIoTDeviceDefenderAuditRole")

# Create IoT Logging role
IOT_LOGGING_TRUST_POLICY=$(cat <<'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
)

if ! create_iam_role "AWSIoTLoggingRole" "$IOT_LOGGING_TRUST_POLICY" ""; then
    echo "ERROR: Failed to create logging role"
    exit 1
fi
LOGGING_ROLE_ARN=$ROLE_ARN
CREATED_RESOURCES+=("IAM Role: AWSIoTLoggingRole")

# Create IoT Mitigation Action role
IOT_MITIGATION_TRUST_POLICY=$(cat <<'EOF'
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
)
```

```
if ! create_iam_role "IoTMitigationActionErrorLoggingRole"
 "$IOT_MITIGATION_TRUST_POLICY" ""; then
    echo "ERROR: Failed to create mitigation role"
    exit 1
fi
MITIGATION_ROLE_ARN=$ROLE_ARN
CREATED_RESOURCES+=("IAM Role: IoTMitigationActionErrorLoggingRole")

# Wait for IAM role propagation
echo "Waiting for IAM role propagation..."
sleep 5

# Step 2: Enable audit checks
echo ""
echo "================================================="
echo "Step 2: Enabling AWS IoT Device Defender audit checks"
echo "================================================="

# Get current audit configuration
echo "Getting current audit configuration..."
CURRENT_CONFIG=$(aws iot describe-account-audit-configuration --output json 2>&1)
 || true
if validate_json "$CURRENT_CONFIG"; then
    echo "$CURRENT_CONFIG" | jq '.' 2>/dev/null || echo "Could not parse current
 configuration"
fi

# Enable specific audit checks with proper JSON escaping
echo "Enabling audit checks..."
AUDIT_CONFIG='{"LOGGING_DISABLED_CHECK":{"enabled":true}}'

if ! validate_json "$AUDIT_CONFIG"; then
    echo "ERROR: Invalid audit configuration JSON"
    exit 1
fi

UPDATE_RESULT=$(aws iot update-account-audit-configuration \
  --role-arn "$AUDIT_ROLE_ARN" \
  --audit-check-configurations "$AUDIT_CONFIG" 2>&1) || true

if ! check_error "$UPDATE_RESULT"; then
    echo "Failed to update audit configuration"
    exit 1
```

```
fi

echo "Audit checks enabled successfully"

# Step 3: Run an on-demand audit
echo ""
echo "================================================="
echo "Step 3: Running an on-demand audit"
echo "================================================="

echo "Starting on-demand audit task..."
AUDIT_TASK_RESULT=$(aws iot start-on-demand-audit-task \
  --target-check-names LOGGING_DISABLED_CHECK --output json 2>&1) || true

if ! check_error "$AUDIT_TASK_RESULT"; then
    echo "Failed to start on-demand audit task"
    exit 1
fi

TASK_ID=$(extract_json_value "$AUDIT_TASK_RESULT" "taskId")
if [ -z "$TASK_ID" ]; then
    echo "ERROR: Could not extract task ID from response"
    exit 1
fi

echo "Audit task started with ID: $TASK_ID"
CREATED_RESOURCES+=("Audit Task: $TASK_ID")

# Tag the audit task via IoT service
aws iot tag-resource --resource-arn "arn:aws:iot:$(aws configure get region):
${ACCOUNT_ID}:audittask/${TASK_ID}" --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-iot-device-defender-gs 2>&1 || true

# Wait for the audit task to complete
echo "Waiting for audit task to complete (this may take a few minutes)..."
TASK_STATUS="IN_PROGRESS"
TIMEOUT=0
MAX_TIMEOUT=600
POLL_INTERVAL=15

while [ "$TASK_STATUS" != "COMPLETED" ]; do
    if [ $TIMEOUT -ge $MAX_TIMEOUT ]; then
        echo "WARNING: Audit task did not complete within ${MAX_TIMEOUT} seconds,
 continuing..."
```

```
        break
    fi

    sleep "$POLL_INTERVAL"
    TIMEOUT=$((TIMEOUT + POLL_INTERVAL))

    TASK_DETAILS=$(aws iot describe-audit-task --task-id "$TASK_ID" --output json
 2>&1) || true
    if validate_json "$TASK_DETAILS"; then
        TASK_STATUS=$(extract_json_value "$TASK_DETAILS" "taskStatus")
        echo "Current task status: $TASK_STATUS (elapsed: ${TIMEOUT}s)"

        if [ "$TASK_STATUS" = "FAILED" ]; then
            echo "WARNING: Audit task failed, continuing with script..."
            FAILURE_REASON=$(extract_json_value "$TASK_DETAILS"
 "taskStatistics.failedChecksNotApplicable")
            if [ -n "$FAILURE_REASON" ]; then
                echo "Reason: $FAILURE_REASON"
            fi
            break
        fi
    else
        echo "WARNING: Could not parse task details, retrying..."
    fi
done

echo "Audit task processing completed"

# Get audit findings (non-blocking)
echo "Getting audit findings..."
FINDINGS=$(aws iot list-audit-findings \
  --task-id "$TASK_ID" --output json 2>&1) || true

if validate_json "$FINDINGS"; then
    FINDING_COUNT=$(echo "$FINDINGS" | jq '.findings | length' 2>/dev/null ||
 echo "0")
    echo "Audit findings count: $FINDING_COUNT"
    if [ "$FINDING_COUNT" -gt 0 ]; then
        echo "Sample finding:"
        echo "$FINDINGS" | jq '.findings[0]' 2>/dev/null || echo "Could not parse
 finding"
    fi
else
    echo "WARNING: Could not parse audit findings response"
```

```
    FINDINGS='{"findings":[]}'
fi

# Check if we have any non-compliant findings
FINDING_ID=$(extract_json_value "$FINDINGS" "findings[0].findingId")
if [ -n "$FINDING_ID" ]; then
    echo "Found non-compliant finding with ID: $FINDING_ID"
    HAS_FINDINGS=true
else
    echo "No non-compliant findings detected"
    HAS_FINDINGS=false
fi

# Step 4: Create a mitigation action
echo ""
echo "================================================="
echo "Step 4: Creating a mitigation action"
echo "================================================="

# Check if mitigation action already exists and delete it
if aws iot describe-mitigation-action --action-name "EnableErrorLoggingAction" >/
dev/null 2>&1; then
    echo "Mitigation action 'EnableErrorLoggingAction' already exists, deleting
 it first..."
    aws iot delete-mitigation-action --action-name "EnableErrorLoggingAction"
 2>&1 || true
    sleep 2
fi

echo "Creating mitigation action to enable AWS IoT logging..."

# Build mitigation action parameters JSON
MITIGATION_PARAMS=$(cat <<EOF
{
  "enableIoTLoggingParams": {
    "roleArnForLogging": "$LOGGING_ROLE_ARN",
    "logLevel": "ERROR"
  }
}
EOF
)

if ! validate_json "$MITIGATION_PARAMS"; then
    echo "ERROR: Invalid mitigation parameters JSON"
```

```
    exit 1
fi

MITIGATION_RESULT=$(aws iot create-mitigation-action \
  --action-name "EnableErrorLoggingAction" \
  --role-arn "$MITIGATION_ROLE_ARN" \
  --action-params "$MITIGATION_PARAMS" --output json 2>&1) || true

if ! check_error "$MITIGATION_RESULT"; then
    echo "Failed to create mitigation action"
    exit 1
fi

if validate_json "$MITIGATION_RESULT"; then
    echo "Mitigation action created successfully"
    MITIGATION_ACTION_ARN=$(extract_json_value "$MITIGATION_RESULT" "actionArn")
    if [ -n "$MITIGATION_ACTION_ARN" ]; then
        echo "Mitigation Action ARN: $MITIGATION_ACTION_ARN"
        aws iot tag-resource --resource-arn "$MITIGATION_ACTION_ARN" --tags
 Key=project,Value=doc-smith Key=tutorial,Value=aws-iot-device-defender-gs 2>&1
 || true
    fi
else
    echo "WARNING: Could not validate mitigation action response, but action may
 have been created"
fi

CREATED_RESOURCES+=("Mitigation Action: EnableErrorLoggingAction")

# Step 5: Apply mitigation action to findings (if any)
if [ "$HAS_FINDINGS" = true ]; then
    echo ""
    echo "================================================="
    echo "Step 5: Applying mitigation action to findings"
    echo "================================================="

    MITIGATION_TASK_ID="MitigationTask-$(date +%s)"
    echo "Starting mitigation actions task with ID: $MITIGATION_TASK_ID"

    # Build target JSON
    TARGET_JSON=$(cat <<EOF
{
  "findingIds": ["$FINDING_ID"]
}
```

```
EOF
)

    if ! validate_json "$TARGET_JSON"; then
        echo "ERROR: Invalid target JSON"
        exit 1
    fi

    # Build audit check to actions mapping JSON
    AUDIT_CHECK_MAPPING=$(cat <<EOF
{
  "LOGGING_DISABLED_CHECK": ["EnableErrorLoggingAction"]
}
EOF
)

    if ! validate_json "$AUDIT_CHECK_MAPPING"; then
        echo "ERROR: Invalid audit check mapping JSON"
        exit 1
    fi

    MITIGATION_TASK_RESULT=$(aws iot start-audit-mitigation-actions-task \
      --task-id "$MITIGATION_TASK_ID" \
      --target "$TARGET_JSON" \
      --audit-check-to-actions-mapping "$AUDIT_CHECK_MAPPING" --output json 2>&1)
 || true

    if ! check_error "$MITIGATION_TASK_RESULT"; then
        echo "WARNING: Failed to start mitigation actions task, continuing..."
    else
        echo "Mitigation actions task started successfully"
        CREATED_RESOURCES+=("Mitigation Task: $MITIGATION_TASK_ID")
    fi
else
    echo ""
    echo "=================================================="
    echo "Step 5: Skipping mitigation action application (no findings)"
    echo "=================================================="
fi

# Step 6: Set up SNS notifications (optional)
echo ""
echo "=================================================="
echo "Step 6: Setting up SNS notifications"
```

```
echo "================================================="

# Check if SNS topic already exists
SNS_TOPICS=$(aws sns list-topics --output json 2>&1) || true
TOPIC_ARN=""
if validate_json "$SNS_TOPICS"; then
    TOPIC_ARN=$(echo "$SNS_TOPICS" | jq -r '.Topics[] | select(.TopicArn |
 contains("IoTDDNotifications")) | .TopicArn' 2>/dev/null | head -1 || echo "")
fi

if [ -n "$TOPIC_ARN" ]; then
    echo "SNS topic 'IoTDDNotifications' already exists, using existing topic..."
    echo "Topic ARN: $TOPIC_ARN"
else
    echo "Creating SNS topic for notifications..."
    SNS_RESULT=$(aws sns create-topic --name "IoTDDNotifications" --tags
 Key=project,Value=doc-smith Key=tutorial,Value=aws-iot-device-defender-gs --
output json 2>&1) || true

    if ! check_error "$SNS_RESULT"; then
        echo "WARNING: Failed to create SNS topic, continuing..."
        SNS_RESULT=""
    else
        TOPIC_ARN=$(extract_json_value "$SNS_RESULT" "TopicArn")
        if [ -n "$TOPIC_ARN" ]; then
            echo "SNS topic created with ARN: $TOPIC_ARN"
            CREATED_RESOURCES+=("SNS Topic: IoTDDNotifications")
        fi
    fi
fi

if [ -n "$TOPIC_ARN" ]; then
    echo "Updating audit configuration to enable SNS notifications..."

    # Build SNS notification configuration JSON
    SNS_CONFIG=$(cat <<EOF
{
  "SNS": {
    "targetArn": "$TOPIC_ARN",
    "roleArn": "$AUDIT_ROLE_ARN",
    "enabled": true
  }
}
EOF
```

```
)

    if ! validate_json "$SNS_CONFIG"; then
        echo "ERROR: Invalid SNS configuration JSON"
        exit 1
    fi

    SNS_UPDATE_RESULT=$(aws iot update-account-audit-configuration \
      --audit-notification-target-configurations "$SNS_CONFIG" 2>&1) || true

    if ! check_error "$SNS_UPDATE_RESULT"; then
        echo "WARNING: Failed to update audit configuration for SNS
 notifications"
    else
        echo "SNS notifications enabled successfully"
    fi
else
    echo "Skipping SNS configuration due to topic creation failure"
fi

# Step 7: Enable AWS IoT logging
echo ""
echo "================================================="
echo "Step 7: Enabling AWS IoT logging"
echo "================================================="

echo "Setting up AWS IoT logging options..."

LOGGING_RESULT=$(aws iot set-v2-logging-options \
  --role-arn "$LOGGING_ROLE_ARN" \
  --default-log-level "ERROR" 2>&1) || true

if ! check_error "$LOGGING_RESULT"; then
    echo "V2 logging setup failed, trying v1 logging..."

    V1_LOGGING_CONFIG=$(cat <<EOF
{
  "roleArn": "$LOGGING_ROLE_ARN",
  "logLevel": "ERROR"
}
EOF
)

    if ! validate_json "$V1_LOGGING_CONFIG"; then
```

```
        echo "ERROR: Invalid v1 logging configuration JSON"
        exit 1
    fi

    LOGGING_RESULT_V1=$(aws iot set-logging-options \
      --logging-options-payload "$V1_LOGGING_CONFIG" 2>&1) || true

    if ! check_error "$LOGGING_RESULT_V1"; then
        echo "WARNING: Failed to set up AWS IoT logging with both v1 and v2
 methods, continuing..."
    else
        echo "AWS IoT v1 logging enabled successfully"
    fi
else
    echo "AWS IoT v2 logging enabled successfully"
fi

# Verify logging is enabled
echo "Verifying logging configuration..."
LOGGING_CONFIG=$(aws iot get-v2-logging-options --output json 2>&1) || true
if [ -n "$LOGGING_CONFIG" ] && ! check_error "$LOGGING_CONFIG" && validate_json
 "$LOGGING_CONFIG"; then
    echo "Logging configuration verified"
    echo "$LOGGING_CONFIG" | jq '.' 2>/dev/null || echo "Configuration retrieved
 but could not display details"
else
    echo "Could not verify logging configuration, but setup completed"
fi

# Script completed successfully
echo ""
echo "================================================="
echo "AWS IoT Device Defender setup completed successfully!"
echo "================================================="
echo "The following resources were created:"
for resource in "${CREATED_RESOURCES[@]+"${CREATED_RESOURCES[@]}"}"; do
    echo "- $resource"
done
echo ""

# Cleanup phase
echo "=========================================="
echo "CLEANUP"
echo "=========================================="
```

```
echo "Starting automatic cleanup of resources..."
echo "Waiting 10 seconds before cleanup to allow resource stabilization..."
sleep 10

# Disable AWS IoT logging
echo "Disabling AWS IoT logging..."

DISABLE_V2_RESULT=$(aws iot set-v2-logging-options \
  --default-log-level "DISABLED" 2>&1) || true

if check_error "$DISABLE_V2_RESULT"; then
    echo "V2 logging disabled successfully"
else
    echo "Attempting v1 logging disable..."

    V1_DISABLE_CONFIG=$(cat <<'EOF'
{
  "logLevel": "DISABLED"
}
EOF
)

    DISABLE_V1_RESULT=$(aws iot set-logging-options \
      --logging-options-payload "$V1_DISABLE_CONFIG" 2>&1) || true

    if check_error "$DISABLE_V1_RESULT"; then
        echo "V1 logging disabled successfully"
    else
        echo "WARNING: Could not disable logging"
    fi
fi

# Delete mitigation action
echo "Deleting mitigation action..."
aws iot delete-mitigation-action --action-name "EnableErrorLoggingAction" 2>&1 ||
 true

# Reset audit configuration
echo "Resetting IoT Device Defender audit configuration..."
RESET_AUDIT_CONFIG='{"LOGGING_DISABLED_CHECK":{"enabled":false}}'
aws iot update-account-audit-configuration \
  --audit-check-configurations "$RESET_AUDIT_CONFIG" 2>&1 || true

# Delete SNS topic
```

```
echo "Deleting SNS topic..."
if [ -n "${TOPIC_ARN:-}" ] && [ "$TOPIC_ARN" != "null" ]; then
    aws sns delete-topic --topic-arn "$TOPIC_ARN" 2>&1 || true
fi

# Clean up IAM roles with improved error handling
echo "Cleaning up IAM roles..."

cleanup_role() {
    local role_name=$1
    echo "Cleaning up role: $role_name"

    if aws iam get-role --role-name "$role_name" >/dev/null 2>&1; then
        ROLE_POLICIES=$(aws iam list-role-policies --role-name "$role_name" --
output json 2>&1 || echo '{"PolicyNames":[]}')
        if validate_json "$ROLE_POLICIES"; then
            while IFS= read -r policy_name; do
                if [ -n "$policy_name" ] && [ "$policy_name" != "null" ]; then
                    echo "  Deleting inline policy: $policy_name"
                    aws iam delete-role-policy \
                        --role-name "$role_name" \
                        --policy-name "$policy_name" 2>&1 || true
                fi
            done < <(echo "$ROLE_POLICIES" | jq -r '.PolicyNames[]' 2>/dev/null
 || echo "")
        fi

        ATTACHED_POLICIES=$(aws iam list-attached-role-policies --role-name
 "$role_name" --output json 2>&1 || echo '{"AttachedPolicies":[]}')
        if validate_json "$ATTACHED_POLICIES"; then
            while IFS= read -r policy_arn; do
                if [ -n "$policy_arn" ] && [ "$policy_arn" != "null" ]; then
                    echo "  Detaching managed policy: $policy_arn"
                    aws iam detach-role-policy \
                        --role-name "$role_name" \
                        --policy-arn "$policy_arn" 2>&1 || true
                fi
            done < <(echo "$ATTACHED_POLICIES" | jq -r
 '.AttachedPolicies[].PolicyArn' 2>/dev/null || echo "")
        fi

        echo "  Deleting role: $role_name"
        aws iam delete-role --role-name "$role_name" 2>&1 || true
    else
```

```
        echo "  Role $role_name does not exist or already deleted"
    fi
}

cleanup_role "AWSIoTDeviceDefenderAuditRole"
cleanup_role "AWSIoTLoggingRole"
cleanup_role "IoTMitigationActionErrorLoggingRole"

echo "Cleanup completed successfully"

echo ""
echo "Script execution completed at $(date)"
echo "Log file: $LOG_FILE"
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS CLI - Befehlsreferenz.
  - AttachRolePolicy
  - CreateMitigationAction
  - CreateRole
  - CreateTopic
  - DeleteAccountAuditConfiguration
  - DeleteMitigationAction
  - DeleteRole
  - DeleteRolePolicy
  - DeleteTopic
  - DescribeAccountAuditConfiguration
  - DescribeAuditTask
  - DetachRolePolicy
  - GetLoggingOptions
  - GetRole
  - GetV2LoggingOptions
  - ListAuditFindings
  - ListAuditMitigationActionsTasks
  - ListMitigationActions
  - ListRolePolicies

- ListTopics
- PutRolePolicy
- SetLoggingOptions
- SetV2LoggingOptions
- StartAuditMitigationActionsTask
- StartOnDemandAuditTask
- UpdateAccountAuditConfiguration

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden AWS IoT mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Erste Schritte mit Nachrichten im Internet der Dinge

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- IoT-Ressourcen erstellen
- Konfigurieren Ihres Geräts
- Ausführen der Beispielanwendungen
- Bereinigen von Ressourcen

Bash

AWS CLI mit Bash-Skript

> **Note**
>
> Es läuft noch mehr. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie es im Tutorials und Beispiele-Repository für Entwickler einrichten und ausführen.

```
#!/bin/bash

# AWS IoT Core Getting Started Script
# This script creates AWS IoT resources, configures a device, and runs a sample
 application
```

```
# Set up logging
LOG_FILE="iot-core-setup.log"
echo "Starting AWS IoT Core setup at $(date)" > $LOG_FILE

# Function to log commands and their outputs
log_cmd() {
    echo "$(date): Running command: $1" >> $LOG_FILE
    eval "$1" 2>&1 | tee -a $LOG_FILE
    return ${PIPESTATUS[0]}
}

# Function to check for errors
check_error() {
    if [ $1 -ne 0 ]; then
        echo "ERROR: Command failed with exit code $1" | tee -a $LOG_FILE
        echo "Please check the log file $LOG_FILE for details" | tee -a $LOG_FILE
        cleanup_on_error
        exit $1
    fi
}

# Function to cleanup resources on error
cleanup_on_error() {
    echo "Error encountered. Attempting to clean up resources..." | tee -a
 $LOG_FILE
    echo "Resources created:" | tee -a $LOG_FILE
    if [ ! -z "$CERTIFICATE_ARN" ]; then
        echo "Certificate ARN: $CERTIFICATE_ARN" | tee -a $LOG_FILE
        if [ ! -z "$POLICY_NAME" ]; then
            log_cmd "aws iot detach-policy --policy-name $POLICY_NAME --target
 $CERTIFICATE_ARN"
        fi
        if [ ! -z "$THING_NAME" ]; then
            log_cmd "aws iot detach-thing-principal --thing-name $THING_NAME --
principal $CERTIFICATE_ARN"
        fi
        if [ ! -z "$CERTIFICATE_ID" ]; then
            log_cmd "aws iot update-certificate --certificate-id $CERTIFICATE_ID
 --new-status INACTIVE"
            log_cmd "aws iot delete-certificate --certificate-id $CERTIFICATE_ID"
        fi
    fi
    if [ ! -z "$THING_NAME" ]; then
```

```
        echo "Thing Name: $THING_NAME" | tee -a $LOG_FILE
        log_cmd "aws iot delete-thing --thing-name $THING_NAME"
    fi
    if [ ! -z "$POLICY_NAME" ]; then
        echo "Policy Name: $POLICY_NAME" | tee -a $LOG_FILE
        log_cmd "aws iot delete-policy --policy-name $POLICY_NAME"
    fi
    if [ ! -z "$SHARED_POLICY_NAME" ]; then
        echo "Shared Policy Name: $SHARED_POLICY_NAME" | tee -a $LOG_FILE
        log_cmd "aws iot delete-policy --policy-name $SHARED_POLICY_NAME"
    fi
}

# Generate unique identifiers
RANDOM_SUFFIX=$(openssl rand -hex 4)
THING_NAME="MyIoTThing-${RANDOM_SUFFIX}"
POLICY_NAME="MyIoTPolicy-${RANDOM_SUFFIX}"
SHARED_POLICY_NAME="SharedSubPolicy-${RANDOM_SUFFIX}"
CERTS_DIR="$HOME/certs"

echo "================================================" | tee -a $LOG_FILE
echo "AWS IoT Core Getting Started" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE
echo "This script will:" | tee -a $LOG_FILE
echo "1. Create AWS IoT resources (policy, thing, certificate)" | tee -a
 $LOG_FILE
echo "2. Configure your device" | tee -a $LOG_FILE
echo "3. Set up for running the sample application" | tee -a $LOG_FILE
echo "" | tee -a $LOG_FILE
echo "Thing Name: $THING_NAME" | tee -a $LOG_FILE
echo "Policy Name: $POLICY_NAME" | tee -a $LOG_FILE
echo "Certificates Directory: $CERTS_DIR" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE
echo "" | tee -a $LOG_FILE

# Get AWS account ID
echo "Getting AWS account ID..." | tee -a $LOG_FILE
ACCOUNT_ID=$(log_cmd "aws sts get-caller-identity --query Account --output text")
check_error $?

# Get AWS region
echo "Getting AWS region..." | tee -a $LOG_FILE
REGION=$(log_cmd "aws configure get region")
check_error $?
```

```
if [ -z "$REGION" ]; then
    echo "AWS region not configured. Please run 'aws configure' to set your
 region." | tee -a $LOG_FILE
    exit 1
fi

echo "Using AWS Account ID: $ACCOUNT_ID and Region: $REGION" | tee -a $LOG_FILE

# Step 1: Create AWS IoT Resources
echo "" | tee -a $LOG_FILE
echo "Step 1: Creating AWS IoT Resources..." | tee -a $LOG_FILE

# Create IoT policy
echo "Creating IoT policy document..." | tee -a $LOG_FILE
cat > iot-policy.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:$REGION:$ACCOUNT_ID:client/test-*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:$REGION:$ACCOUNT_ID:topic/test/topic"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:$REGION:$ACCOUNT_ID:topicfilter/test/topic"
```

```
      ]
    }
  ]
}
EOF

echo "Creating IoT policy: $POLICY_NAME..." | tee -a $LOG_FILE
log_cmd "aws iot create-policy --policy-name $POLICY_NAME --policy-
document file://iot-policy.json --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-iot-core-gs"
check_error $?

# Create IoT thing
echo "Creating IoT thing: $THING_NAME..." | tee -a $LOG_FILE
log_cmd "aws iot create-thing --thing-name $THING_NAME"
check_error $?

THING_ARN=$(aws iot describe-thing --thing-name "$THING_NAME" --query 'thingArn'
 --output text)
aws iot tag-resource --resource-arn "$THING_ARN" --tags Key=project,Value=doc-
smith Key=tutorial,Value=aws-iot-core-gs

# Create directory for certificates
echo "Creating certificates directory..." | tee -a $LOG_FILE
log_cmd "mkdir -p $CERTS_DIR"
check_error $?

# Create keys and certificate
echo "Creating keys and certificate..." | tee -a $LOG_FILE
CERT_OUTPUT=$(log_cmd "aws iot create-keys-and-certificate --set-as-active
 --certificate-pem-outfile $CERTS_DIR/device.pem.crt --public-key-outfile
 $CERTS_DIR/public.pem.key --private-key-outfile $CERTS_DIR/private.pem.key")
check_error $?

# Extract certificate ARN and ID
CERTIFICATE_ARN=$(echo "$CERT_OUTPUT" | grep "certificateArn" | cut -d'"' -f4)
CERTIFICATE_ID=$(echo "$CERTIFICATE_ARN" | cut -d/ -f2)

if [ -z "$CERTIFICATE_ARN" ] || [ -z "$CERTIFICATE_ID" ]; then
    echo "Failed to extract certificate ARN or ID" | tee -a $LOG_FILE
    cleanup_on_error
    exit 1
fi
```

```
echo "Certificate ARN: $CERTIFICATE_ARN" | tee -a $LOG_FILE
echo "Certificate ID: $CERTIFICATE_ID" | tee -a $LOG_FILE

# Attach policy to certificate
echo "Attaching policy to certificate..." | tee -a $LOG_FILE
log_cmd "aws iot attach-policy --policy-name $POLICY_NAME --target
 $CERTIFICATE_ARN"
check_error $?

# Attach certificate to thing
echo "Attaching certificate to thing..." | tee -a $LOG_FILE
log_cmd "aws iot attach-thing-principal --thing-name $THING_NAME --principal
 $CERTIFICATE_ARN"
check_error $?

# Download Amazon Root CA certificate
echo "Downloading Amazon Root CA certificate..." | tee -a $LOG_FILE
log_cmd "curl -s -o $CERTS_DIR/Amazon-root-CA-1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem"
check_error $?

# Step 2: Configure Your Device
echo "" | tee -a $LOG_FILE
echo "Step 2: Configuring Your Device..." | tee -a $LOG_FILE

# Check if Git is installed
echo "Checking if Git is installed..." | tee -a $LOG_FILE
if ! command -v git &> /dev/null; then
    echo "Git is not installed. Please install Git and run this script again." |
 tee -a $LOG_FILE
    cleanup_on_error
    exit 1
fi

# Check if Python is installed
echo "Checking if Python is installed..." | tee -a $LOG_FILE
if ! command -v python3 &> /dev/null; then
    echo "Python 3 is not installed. Please install Python 3 and run this script
 again." | tee -a $LOG_FILE
    cleanup_on_error
    exit 1
fi

# Install AWS IoT Device SDK for Python
```

```
echo "Installing AWS IoT Device SDK for Python..." | tee -a $LOG_FILE
log_cmd "python3 -m pip install awsiotsdk"
check_error $?

# Clone the AWS IoT Device SDK for Python repository
echo "Cloning AWS IoT Device SDK for Python repository..." | tee -a $LOG_FILE
if [ ! -d "$HOME/aws-iot-device-sdk-python-v2" ]; then
    log_cmd "cd $HOME && git clone https://github.com/aws/aws-iot-device-sdk-
python-v2.git"
    check_error $?
else
    echo "AWS IoT Device SDK for Python repository already exists." | tee -a
 $LOG_FILE
fi

# Step 3: Get AWS IoT Endpoint
echo "" | tee -a $LOG_FILE
echo "Step 3: Getting AWS IoT Endpoint..." | tee -a $LOG_FILE

IOT_ENDPOINT=$(log_cmd "aws iot describe-endpoint --endpoint-type iot:Data-ATS --
query endpointAddress --output text")
check_error $?

echo "AWS IoT Endpoint: $IOT_ENDPOINT" | tee -a $LOG_FILE

# Create a shared subscription policy (optional)
echo "" | tee -a $LOG_FILE
echo "Creating shared subscription policy (optional)..." | tee -a $LOG_FILE

cat > shared-sub-policy.json << EOF
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:$REGION:$ACCOUNT_ID:client/*"
      ]
    },
    {
      "Effect": "Allow",
```

```
      "Action": [
        "iot:Publish",
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:$REGION:$ACCOUNT_ID:topic/test/topic"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:$REGION:$ACCOUNT_ID:topicfilter/test/topic",
        "arn:aws:iot:$REGION:$ACCOUNT_ID:topicfilter/\$share/*/test/topic"
      ]
    }
  ]
}
EOF

log_cmd "aws iot create-policy --policy-name $SHARED_POLICY_NAME --policy-
document file://shared-sub-policy.json --tags Key=project,Value=doc-smith
 Key=tutorial,Value=aws-iot-core-gs"
check_error $?

log_cmd "aws iot attach-policy --policy-name $SHARED_POLICY_NAME --target
 $CERTIFICATE_ARN"
check_error $?

# Summary of created resources
echo "" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE
echo "Setup Complete! Resources Created:" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE
echo "Thing Name: $THING_NAME" | tee -a $LOG_FILE
echo "Policy Name: $POLICY_NAME" | tee -a $LOG_FILE
echo "Shared Subscription Policy Name: $SHARED_POLICY_NAME" | tee -a $LOG_FILE
echo "Certificate ID: $CERTIFICATE_ID" | tee -a $LOG_FILE
echo "Certificate ARN: $CERTIFICATE_ARN" | tee -a $LOG_FILE
echo "Certificate Files Location: $CERTS_DIR" | tee -a $LOG_FILE
echo "AWS IoT Endpoint: $IOT_ENDPOINT" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE
```

```
# Instructions for running the sample application
echo "" | tee -a $LOG_FILE
echo "To run the sample application, execute:" | tee -a $LOG_FILE
echo "cd $HOME/aws-iot-device-sdk-python-v2/samples" | tee -a $LOG_FILE
echo "python3 pubsub.py \\" | tee -a $LOG_FILE
echo "  --endpoint $IOT_ENDPOINT \\" | tee -a $LOG_FILE
echo "  --ca_file $CERTS_DIR/Amazon-root-CA-1.pem \\" | tee -a $LOG_FILE
echo "  --cert $CERTS_DIR/device.pem.crt \\" | tee -a $LOG_FILE
echo "  --key $CERTS_DIR/private.pem.key" | tee -a $LOG_FILE
echo "" | tee -a $LOG_FILE
echo "To run the shared subscription example, execute:" | tee -a $LOG_FILE
echo "cd $HOME/aws-iot-device-sdk-python-v2/samples" | tee -a $LOG_FILE
echo "python3 mqtt5_shared_subscription.py \\" | tee -a $LOG_FILE
echo "  --endpoint $IOT_ENDPOINT \\" | tee -a $LOG_FILE
echo "  --ca_file $CERTS_DIR/Amazon-root-CA-1.pem \\" | tee -a $LOG_FILE
echo "  --cert $CERTS_DIR/device.pem.crt \\" | tee -a $LOG_FILE
echo "  --key $CERTS_DIR/private.pem.key \\" | tee -a $LOG_FILE
echo "  --group_identifier consumer" | tee -a $LOG_FILE

# Ask if user wants to clean up resources
echo "" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE
echo "CLEANUP CONFIRMATION" | tee -a $LOG_FILE
echo "================================================" | tee -a $LOG_FILE
echo "Do you want to clean up all created resources? (y/n): " | tee -a $LOG_FILE
read -r CLEANUP_CHOICE

if [[ $CLEANUP_CHOICE =~ ^[Yy]$ ]]; then
    echo "Cleaning up resources..." | tee -a $LOG_FILE

    # Detach policies from certificate
    echo "Detaching policies from certificate..." | tee -a $LOG_FILE
    log_cmd "aws iot detach-policy --policy-name $POLICY_NAME --target
 $CERTIFICATE_ARN"
    log_cmd "aws iot detach-policy --policy-name $SHARED_POLICY_NAME --target
 $CERTIFICATE_ARN"

    # Detach certificate from thing
    echo "Detaching certificate from thing..." | tee -a $LOG_FILE
    log_cmd "aws iot detach-thing-principal --thing-name $THING_NAME --principal
 $CERTIFICATE_ARN"

    # Update certificate status to INACTIVE
```

```
    echo "Setting certificate to inactive..." | tee -a $LOG_FILE
    log_cmd "aws iot update-certificate --certificate-id $CERTIFICATE_ID --new-
status INACTIVE"

    # Delete certificate
    echo "Deleting certificate..." | tee -a $LOG_FILE
    log_cmd "aws iot delete-certificate --certificate-id $CERTIFICATE_ID"

    # Delete thing
    echo "Deleting thing..." | tee -a $LOG_FILE
    log_cmd "aws iot delete-thing --thing-name $THING_NAME"

    # Delete policies
    echo "Deleting policies..." | tee -a $LOG_FILE
    log_cmd "aws iot delete-policy --policy-name $POLICY_NAME"
    log_cmd "aws iot delete-policy --policy-name $SHARED_POLICY_NAME"

    echo "Cleanup complete!" | tee -a $LOG_FILE
else
    echo "Resources were not cleaned up. You can manually clean them up later." |
 tee -a $LOG_FILE
    echo "To clean up resources, run the following commands:" | tee -a $LOG_FILE
    echo "aws iot detach-policy --policy-name $POLICY_NAME --target
 $CERTIFICATE_ARN" | tee -a $LOG_FILE
    echo "aws iot detach-policy --policy-name $SHARED_POLICY_NAME --target
 $CERTIFICATE_ARN" | tee -a $LOG_FILE
    echo "aws iot detach-thing-principal --thing-name $THING_NAME --principal
 $CERTIFICATE_ARN" | tee -a $LOG_FILE
    echo "aws iot update-certificate --certificate-id $CERTIFICATE_ID --new-
status INACTIVE" | tee -a $LOG_FILE
    echo "aws iot delete-certificate --certificate-id $CERTIFICATE_ID" | tee -a
 $LOG_FILE
    echo "aws iot delete-thing --thing-name $THING_NAME" | tee -a $LOG_FILE
    echo "aws iot delete-policy --policy-name $POLICY_NAME" | tee -a $LOG_FILE
    echo "aws iot delete-policy --policy-name $SHARED_POLICY_NAME" | tee -a
 $LOG_FILE
fi

echo "" | tee -a $LOG_FILE
echo "Script execution completed. See $LOG_FILE for details." | tee -a $LOG_FILE
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS CLI - Befehlsreferenz.
  - AttachPolicy
  - AttachThingPrincipal
  - CreateKeysAndCertificate
  - CreatePolicy
  - CreateThing
  - DeleteCertificate
  - DeletePolicy
  - DeleteThing
  - DescribeEndpoint
  - DetachPolicy
  - DetachThingPrincipal
  - GetCallerIdentity
  - UpdateCertificate

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden AWS IoT mit einem AWS SDK. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

AWS IoT Kontingente

Informationen zu AWS IoT Kontingenten finden Sie in der AWS Allgemeinen Referenz.

- Informationen zu AWS IoT Core Kontingenten finden Sie unter AWS IoT Core Endpunkte und Kontingente.
- Informationen zu AWS IoT Device Management Kontingenten finden Sie unter AWS IoT Device Management Endpunkte und Kontingente.
- Informationen zu AWS IoT Device Defender Kontingenten finden Sie unter AWS IoT Device Defender Endpunkte und Kontingente.

AWS IoT Core Preise

Informationen zur AWS IoT Core Preisgestaltung finden Sie auf der AWS Marketing-Seite und im AWS Preisrechner .

- AWS IoT Core Preisinformationen finden Sie unter AWS IoT Core Preisinformationen.
- Eine Kostenschätzung für Ihre Architektenlösung erhalten Sie unter AWS -Preisrechner.

Häufig gestellte Fragen zu unterstützten Protokollen

Der AWS IoT Secure Tunneling-Dienst fügt zusätzlich zum bestehenden JSON-Protokoll SDK-Unterstützung für das Smithy RPC v2 CBOR-Protokoll hinzu. AWS Das Smithy RPC v2 CBOR-Protokoll ist für die meisten Workloads leistungsfähiger als JSON. AWS AWS SDKs priorisieren das Protokoll, das für jede Sprache am leistungsstärksten ist.

Was sind die AWS JSON- und Smithy RPC v2 CBOR-Protokolle, und wie unterscheiden sie sich?

Ein Übertragungsprotokoll fungiert als Vermittler zwischen einem Kunden und dem Service. Das AWS SDK wickelt die Serialisierung (Konvertierung eines Objekts in das Wire-Format) und die Deserialisierung (Konvertierung des Wire-Formats zurück in ein Objekt) zwischen dem Client und dem Server für beide Protokolle transparent ab.

AWS JSON — JSON ist eines der am häufigsten verwendeten und akzeptierten Datenformate für die Kommunikation zwischen Systemen. Dieser Dienst verwendet AWS JSON als Medium für die Kommunikation zwischen einem AWS SDK-Client und dem Server. Eine HTTP-Anfrage für einen Dienst-API-Vorgang akzeptiert Eingaben in Form von JSON, der Vorgang wird ausgeführt und die Antwort wird in Form von JSON an den SDK-Client zurückgegeben.

Smithy RPC v2 CBOR - Smithy RPC v2 CBOR (Concise Binary Object Representation) ist ein von Amazon entwickeltes Protokoll für eine effizientere Datenübertragung. Es ist ein Datenformat, das entwickelt wurde, um die Nachrichtengrößen kompakt zu halten und eine Weiterentwicklung des Formats zu ermöglichen, ohne dass Versionsverhandlungen zwischen den Systemen erforderlich sind. Der wesentliche Unterschied zu AWS JSON besteht darin, dass AWS JSON für Menschen lesbaren Text (wie`{"name": "John"}`) verwendet, während Smithy RPC v2 CBOR ein Binärformat verwendet (Sequenzen von Einsen und Nullen). Dieser binäre Ansatz erzeugt kleinere Datenpakete, die schneller über das Netzwerk übertragen werden, und reduziert die Serialisierungszeit, da Computer Binärdaten direkt verarbeiten.

Wie fange ich mit dem Smithy RPC v2 CBOR-Protokoll an?

Verwenden Sie zunächst die neueste AWS SDK/CLI Version, um das Smithy RPC v2 CBOR-Protokoll für diesen Dienst zu verwenden. Das SDK verwendet automatisch das Smithy RPC v2

CBOR-Protokoll, sofern verfügbar. Aktualisieren Sie Ihr AWS SDK auf die angegebene Version oder eine nachfolgende Version.

Ab dem 23.07.2020 verwenden die folgenden AWS SDKs das Smithy RPC v2 CBOR-Protokoll, während die anderen SDKs weiterhin das JSON-Protokoll verwenden werden: AWS

- AWS SDK für C++
- AWS SDK für Go V2
- AWS SDK für Java 2.x
- AWS SDK für.NET v4
- AWS Tools für PowerShell v5
- AWS SDK für Swift
- AWS SDK für Kotlin

Was sind die Risiken der Aktivierung des Smithy RPC v2 CBOR-Protokolls?

Die zusätzliche Unterstützung von Smithy RPC v2 CBOR in den SDKs wird für Kunden voraussichtlich vollständig transparent und mühelos sein. Wenn Sie jedoch eine benutzerdefinierte Implementierung des SDK oder eine Kombination aus benutzerdefinierten Clients und dem AWS SDK verwenden, hängt dies vom AWS JSON-specific Serialisierungsverhalten ab und ist möglicherweise nicht mit dem AWS Smithy RPC v2 CBOR-Protokoll kompatibel.

Was ist, wenn meine Lösung Smithy RPC v2 CBOR nicht unterstützt?

Um das AWS JSON-Protokoll zu verwenden, führen Sie ein Downgrade Ihres SDK auf eine Version durch, die vor 2026-07-23 veröffentlicht wurde.

Was AWS Unterstützen Regionen das Smithy RPC v2 CBOR-Protokoll?

Das Smithy RPC v2 CBOR-Protokoll wird in allen AWS Regionen unterstützt, in denen dieser Dienst verfügbar ist.

Welche Latenzverbesserungen kann ich beim Upgrade auf SDK-Versionen erwarten, die Smithy RPC v2 CBOR unterstützen?

Smithy RPC v2 CBOR bietet im Vergleich zu JSON eine um bis zu 13% geringere Latenz und eine Reduzierung der Nutzlastgröße um bis zu 15%. AWS Durch die binäre Kodierung entfällt der Aufwand für das Parsen von JSON-Text und das Escaping von Zeichenketten, wodurch sowohl die Netzwerkübertragungszeit als auch die Serialisierungskosten reduziert werden.

Wird der AWS Ist das JSON-Protokoll veraltet?

AWS Das JSON-Protokoll wird weiterhin durch ältere Versionen von SDKs unterstützt. Außerdem werden einige sprachspezifische SDKs weiterhin AWS JSON verwenden, da es leistungsfähiger ist als Smithy RPC v2 CBOR, da die verfügbare JSON-Implementierung für die Sprache schneller ist als ihre CBOR-Implementierung.

Wo finde ich weitere Informationen zu Smithy RPC v2 CBOR?

Weitere Informationen zum AWS JSON-Protokoll unter JSON 1.0-Protokoll und zum AWS JSON 1.1-Protokoll finden Sie in der Smithy-Dokumentation und AWS zum Smithy RPC v2 CBOR at Smithy RPC v2 CBOR-Protokoll in der Smithy-Dokumentation.

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.