aws

用户指南

AWS CodeBuild



API 版本 2016-10-06

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

AWS CodeBuild: 用户指南

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

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Table of Contents

什么是 AWS CodeBuild？ 1
注册获取 AWS 账户 1
怎么跑 CodeBuild 1
的定价 CodeBuild 3
我该如何开始 CodeBuild？ 3
概念 3
如何 CodeBuild 运作 3
后续步骤 5
开始使用 6
通过控制台开始使用 6
步骤 1：创建源代码 7
步骤 2：创建 buildspec 文件 10
步骤 3：创建两个 S3 存储桶 12
步骤 4：上传源代码和 buildspec 文件 12
步骤 5：创建构建项目 14
步骤 6：运行构建 15
步骤 7：查看汇总的构建信息 15
步骤 8：查看详细的构建信息 16
步骤 9：获取构建输出构件 17
步骤 10：删除 S3 存储桶 18
总结 19
开始使用 AWS CLI 19
步骤 1：创建源代码 20
步骤 2：创建 buildspec 文件 23
步骤 3：创建两个 S3 存储桶 25
步骤 4：上传源代码和 buildspec 文件 25
步骤 5：创建构建项目 26
步骤 6：运行构建 30
步骤 7：查看汇总的构建信息 32
步骤 8：查看详细的构建信息 34
步骤 9：获取构建输出构件 37
步骤 10：删除 S3 存储桶 37
总结 38
基于使用案例的示例 39

Cross-service 样本 40
Amazon ECR 示例 40
Amazon EFS 示例 47
AWS CodePipeline 样本 52
AWS Config 样本 62
构建通知示例 64
构建徽章示例 78
创建具有构建徽章的构建项目 78
访问 AWS CodeBuild 制作徽章 81
发布 CodeBuild 编译徽章 81
CodeBuild 徽章状态 81
测试报告示例 82
运行测试报告示例 82
的 Docker 示例 CodeBuild 88
自定义映像示例中的 Docker 89
Docker 映像编译服务器示例 92
Windows Docker 构建示例 94
“将 Docker 映像发布到 Amazon ECR”示例 97
带有 AWS Secrets Manager 示例的私人注册表 105
将构建输出托管在 S3 存储桶中 109
多个输入和输出示例 112
创建包含多个输入和输出的构建项目 112
创建没有源的项目 115
buildspec 文件示例中的运行时版本 116
更新 buildspec 文件中的运行时版本 116
指定两个运行时 120
源版本示例 124
使用提交 ID 指定 GitHub存储库版本 125
使用引用和提交 ID 指定 GitHub存储库版本 127
Third-party 源存储库示例 128
运行 Bitbucket 示例 128
运行 GitHub 企业服务器示例 133
运行 GitHub 拉取请求和 webhook 过滤器示例 138
教程： CodeBuild 使用 S3 进行证书存储时使用 Fastlane 进行 Apple 代码签名 142
教程：在 “使用证书存储” 中 CodeBuild 使用 GitHub Fastlane 进行 Apple 代码签名 148
在构建时设置构件名称 153

运行 Windows 示例 156
运行 Windows 示例 156
目录结构 157
F# 和 .NET Framework 157
Visual Basic 和 .NET Framework 158
文件 158
F# 和 .NET Framework 158
Visual Basic 和 .NET Framework 163
计划构建 177
Buildspec 参考 179
buildspec 文件名称和存储位置 179
buildspec 语法 180
version 183
run-as 183
env 183
proxy 188
phases 188
reports 192
artifacts 194
cache 199
buildspec 示例 201
buildspec 版本 204
批量 buildspec 参考 204
批处理 204
batch/build-## 205
batch/build-## 207
batch/build-## 210
batch/build-fanout 211
构建环境参考 213
Docker 镜像由 CodeBuild 213
获取当前 Docker 映像的列表 214
EC2 计算映像 214
Lambda 计算映像 216
过时的图片 CodeBuild 222
可用的运行时 223
运行时版本 248

构建环境计算模式和类型 252
关于计算 252
关于预留容量环境类型 252
关于按需环境类型 315
构建环境中的 Shell 和命令 327
构建环境中的环境变量 328
构建环境中的后台任务 333
构建项目 335
创建构建项目 335
先决条件 336
创建构建项目（控制台） 336
创建构建项目 (AWS CLI) 354
创建构建项目 (AWS 软件开发工具包) 372
创建构建项目 (CloudFormation) 372
创建通知规则 372
更改构建项目设置 375
更改构建项目的设置（控制台） 375
更改构建项目的设置 (AWS CLI) 395
更改构建项目的设置 (AWS 软件开发工具包) 396
多个访问令牌 396
第 1 步：创建 Secrets Manager 密钥或 AWS CodeConnections 连接 397
第 2 步：向 CodeBuild 项目 IAM 角色授予对 Secrets Manager 密钥的访问权限 397
第 3 步：配置 Secrets Manager 或 AWS CodeConnections 令牌 399
其他设置选项 403
删除构建项目 406
删除构建项目（控制台） 406
删除构建项目 (AWS CLI) 407
删除构建项目 (AWS SDK) 407
获取公共构建项目 URL 407
共享构建项目 408
共享项目 409
相关服务 412
访问共享项目 412
取消共享已共享的项目 412
标识已共享的项目 413
共享项目权限 413

标记构建项目 413
为项目添加标签 414
查看项目的标签 415
编辑项目的标签 416
从项目中移除标签 417
使用运行器 418
GitHub 行动 418
GitLab 跑步者 440
Buildkite 运行程序 453
使用 Webhook 473
使用 Webhook 的最佳实操 473
Bitbucket Webhook 事件 474
GitHub 全球和组织网络挂钩 487
GitHub 手动网络挂钩 493
GitHub webhook 事件 494
GitLab 群组 webhook 509
GitLab 手动网络挂钩 515
GitLab webhook 事件 516
Buildkite 手动 webhook 530
拉取请求评论批准 531
查看构建项目详细信息 536
查看构建项目的详细信息（控制台） 537
查看构建项目的详细信息 (AWS CLI) 537
查看构建项目的详细信息 (AWS SDK) 539
查看构建项目名称 539
查看构建项目名称的列表（控制台） 540
查看构建项目名称的列表 (AWS CLI) 540
查看构建项目名称 (AWS SDK) 列表 541
Builds 542
手动运行构建 543
在本地运行构建 543
运行构建（控制台） 547
运行构建 (AWS CLI) 548
运行批量构建 (AWS CLI) 553
开始自动运行构建（AWS CLI） 554
停止自动运行构建（AWS CLI） 555

运行构建 (AWS SDK) 556
在 Lambda 计算上运行构建 556
精选的运行时环境 docker 镜像中将包含哪些工具和运行时 AWS Lambda? 557
如果精选映像未包括我需要的工具，该怎么办？ 557
哪些地区支持 AWS Lambda 计算进去 CodeBuild？ 558
的局限性 AWS Lambda 计算 558
使用 AWS SAM Lambda Java 部署 L CodeBuild ambda 函数 558
使用 CodeBuild Lambda 创建单页的 React 应用程序 Node.js 562
使用 Lambda Python 更新 L CodeBuild ambda 函数配置 564
在预留容量实例集上运行构建 568
创建预留容量实例集 569
最佳实践 571
我可以在多个 CodeBuild 项目之间共享预留容量队列吗？ 571
基于属性的计算是如何工作的？ 572
我能否为我的实例集手动指定 Amazon EC2 实例？ 572
哪些区域支持预留容量实例集？ 572
如何配置 macOS 预留容量实例集？ 573
如何为预留容量实例集配置自定义亚马逊机器映像（AMI）？ 574
预留容量实例集的局限性 575
预留容量实例集属性 575
预留容量示例 580
运行批量构建 582
安全角色 582
批量构建类型 582
批量报告模式 586
更多信息 586
执行并行测试 587
中的支持 AWS CodeBuild 588
在批量构建中启用并行测试执行 590
使用 codebuild-tests- run CLI 命令 591
使用 codebuild-glob- search CLI 命令 593
关于测试拆分 594
自动合并各个构建报告 595
并行测试执行示例 597
缓存构建 607
Amazon S3 缓存 607

本地缓存 614
指定本地缓存 615
调试构建 617
使用 CodeBuild 沙盒调试构建 617
使用会话管理器调试构建 618
使用 CodeBuild 沙箱调试构建 618
使用会话管理器调试构建 647
删除构建 652
删除构建 (AWS CLI) 652
删除构建 (AWS SDK) 653
手动重试构建 653
手动重试构建（控制台） 653
手动重试构建 (AWS CLI) 654
手动重试构建 (AWS 软件开发工具包） 654
自动重试构建 654
自动重试构建（控制台） 655
自动重试构建（AWS CLI） 655
自动重试构建 (AWS SDK) 656
停止构建 656
停止构建（控制台） 656
停止构建 (AWS CLI) 657
停止构建 (AWS 软件开发工具包） 657
停止批量构建 657
停止批量构建（控制台） 658
停止批量构建 (AWS CLI) 658
停止批量构建 (AWS 软件开发工具包） 658
自动触发构建 659
创建构建触发器 659
编辑构建触发器 662
查看构建详细信息 664
查看构建详细信息（控制台） 664
查看构建详细信息（AWS CLI） 664
查看版本详情 (AWS SDK) 665
构建阶段过渡 665
查看构建 ID 666
查看构建 ID 的列表（控制台） 666

查看构建 ID 的列表（AWS CLI） 666
查看批量构建 ID 的列表 (AWS CLI) 668
查看构建 ID (AWS SDK) 列表 669
查看构建项目的构建 ID 669
查看构建项目的构建 ID 列表（控制台） 669
查看构建项目的构建 ID 列表 (AWS CLI) 670
查看构建项目的批量构建 ID 列表 (AWS CLI) 671
查看构建项目的构建 ID 列表 (AWS 软件开发工具包） 672
测试报告 673
创建测试报告 674
创建代码覆盖率报告 675
创建代码覆盖率报告 676
自动发现报告 677
使用控制台配置报告自动发现 678
使用项目环境变量配置报告自动发现 678
报告组 678
创建报告组 679
报告组命名 684
共享报告组 685
指定测试文件 690
指定测试命令 691
标记报告组 691
更新报告组 697
测试框架 699
设置 Jasmine 700
设置 Jest 702
设置 pytest 703
设置 RSpec 704
查看测试报告 705
查看构建的测试报告 705
查看报告组的测试报告 706
查看您的测试报告 AWS 账户 706
测试报告权限 706
测试报告的 IAM 角色 706
测试报告操作的权限 708
测试报告权限示例 709

测试报告状态 709
VPC 支持 711
使用案例 711
VPC 的最佳实操 712
VPC 的限制 712
在您的 CodeBuild 项目中允许 Amazon VPC 访问权限 712
排查 VPC 设置的问题 714
使用 VPC 端点 714
在您创建 VPC 端点前 714
为创建 VPC 终端节点 CodeBuild 715
为创建 VPC 终端节点策略 CodeBuild 716
使用托 CodeBuild 管代理服务器 716
为预留容量实例集配置托管代理配置 717
运行 CodeBuild 预留容量队伍 718
使用代理服务器 718
设置在代理服务器 CodeBuild 中运行所需的组件 719
CodeBuild 在显式代理服务器中运行 722
CodeBuild 在透明的代理服务器上运行 726
在代理服务器中运行程序包管理器和其他工具 727
CloudFormation VPC 模板 729
日志记录和监控 736
记录 CodeBuild API 调用 736
关于我们 AWS CodeBuild 中的信息 CloudTrail 736
关于我们 AWS CodeBuild 日志文件条目 737
监控构建 739
CloudWatch 指标 740
CloudWatch 资源利用率指标 742
CloudWatch 维度 744
CloudWatch 警报 744
查看 CodeBuild 指标 745
查看 CodeBuild 资源利用率指标 747
在中创建 CodeBuild 警报 CloudWatch 750
安全性 752
数据保护 752
数据加密 754
密钥管理 755

流量隐私 755
Identity and access management 755
有关管理访问的概述 755
使用基于身份的策略 759
AWS CodeBuild 权限参考 785
使用标签控制对以下内容的访问 AWS CodeBuild 资源 792
在控制台中查看资源 796
合规性验证 797
恢复能力 797
基础设施安全性 797
源提供商访问权限 798
在 Secrets Manager 密钥中创建和存储令牌 798
GitHub 和 GitHub 企业服务器访问权限 801
Bitbucket 访问权限 811
GitLab 访问 821
Cross-service 混乱的副手预防 826
高级主题 828
允许用户与之互动 CodeBuild 828
CodeBuild 允许与其他人互动 AWS 务 835
加密构建输出 842
CodeBuild 使用进行交互 AWS CLI 844
命令行参考 845
AWS SDK 和工具参考 846
支持 AWS 软件开发工具包和工具 AWS CodeBuild 846
使用 AWS SDK 847
指定 CodeBuild 端点 848
指定 AWS CodeBuild 端点（AWS CLI) 849
指定 AWS CodeBuild 端点（AWS SDK) 849
CodeBuild 搭配使用 CodePipeline 851
先决条件 852
创建管道（控制台） 853
创建管线（AWS CLI） 857
添加构建操作 862
添加测试操作 865
CodeBuild 与 Codecov 一起使用 868
将 Codecov 集成到构建项目中 868

CodeBuild 与 Jenkins 一起使用 871
设置 Jenkins 872
安装插件 872
使用插件 872
CodeBuild 与无服务器应用程序一起使用 874
相关资源 875
第三方通知 875
1) 基本 Docker 映像 — windowsservercore 875
2) 基于 Windows 的 Docker 映像 – choco 876
3) 基于 Windows 的 Docker 映像 – git -- 版本 2.16.2 877
4) 基于 Windows 的 Docker 映像 – microsoft-build-tools -- 版本 15.0.26320.2 877
5) 基于 Windows 的 Docker 映像 – nuget.commandline -- 版本 4.5.1 880
7) 基于 Windows 的 Docker 映像 – netfx-4.6.2-devpack 880
8) 基于 Windows 的 Docker 映像 – visualfsharptools，v 4.0 881
9) 基于 Windows 的 Docker 映像 – netfx-pcl-reference-assemblies-4.6 882
10) 基于 Windows 的 Docker 映像 – visualcppbuildtools v 14.0.25420.1 884
11) 基于 Windows 的 Docker 映像 – microsoft-windows-netfx3-ondemand-package.cab 887
12) 基于 Windows 的 Docker 映像 – dotnet-sdk 888
使用 CodeBuild 条件键作为 IAM 服务角色变量 888
AWS CodeBuild 条件键 889
对您的项目和实例集强制执行 VPC 连接设置 889
防止对项目 buildspec 进行未经授权的修改 890
限制构建的计算类型 891
控制环境变量设置 892
在条件键名称中使用变量 893
检查 API 请求中是否存在属性 893
代码示例 895
基本功能 895
操作 895
公共构建 API 912
公共构建操作 912
DescribeBuildBatchesForPublicProject 913
DescribeBuildsForPublicProject 916
GetCloudWatchLogsForPublicBuild 921
GetPresignedUrlsForPublicBuild 924
GetPublicBuild 927

GetPublicBuildBatch 931
GetPublicProject 936
公共版本数据类型 939
BuildBatchForDescribeBuildBatchesPublic 941
PublicBuild 943
PublicBuildArtifacts 947
PublicBuildBatch 948
PublicBuildGroup 952
PublicBuildSummary 954
PublicLogsStatus 956
PublicProject 957
PublicProjectArtifacts 960
PublicProjectBuildBatchConfig 961
PublicProjectEnvironment 962
PublicProjectSource 964
PublicWebhook 966
S3可下载 968
故障排除 970
来自错误存储库的 Apache Maven 构建参考构件 971
默认情况下，以根用户身份运行构建命令 972
如果文件名不是-，则生成可能会失败U.S。构建可能会失败 973
当从 Amazon EC2 Parameter Store 获取参数时，构建可能失败 973
无法在 CodeBuild 控制台中访问分支筛选器 974
无法查看构建是成功还是失败 975
未向源提供商报告构建状态 975
无法找到并选择 Windows Server Core 2019 平台的基本映像 975
构建规范文件中的前期命令无法被后续命令识别 975
尝试下载缓存时出现错误：“访问被拒绝” 976
使用自定义构建映像时出现错误“BUILD_CONTAINER_UNABLE_TO_PULL_IMAGE” 976
错误：“构建容器在完成构建之前发现已失效。构建容器因内存不足而死亡，或者不支持 Docker 镜像。 ErrorCode: 500 英寸 977
错误：运行构建时出现“无法连接到 Docker 进程守护程序” 978
创建或更新构建项目时出现错误：AssumeRole“无权执行：st CodeBuild s:” 979
错误：“调用时出错 GetBucketAcl：要么存储桶所有者已更改，要么服务角色不再有权调用 s3:GetBucketAcl” 979
运行构建时收到错误：“无法上传构件：arn 无效” 980

错误：“Git 克隆失败：无法访问 '#######': SSL 证书问题：自签名证书” 980
运行构建时收到错误：“必须使用指定的终端节点来寻址当前尝试访问的存储桶” 980
错误：“此构建映像需要至少选择一个运行时版本。” 981
构建队列中的构建失败时出现错误“QUEUED: INSUFFICIENT_SUBNET” 982
错误：“无法下载缓存： RequestError：发送请求失败原因是：x509：无法加载系统根目录且未提供根目录” 982
错误：“无法从 S3 下载证书。 AccessDenied” 982
错误：“找不到凭证” 983
RequestError CodeBuild 在代理服务器上运行时出现超时错误 984
bourne shell（sh）必须存在于构建映像中 985
警告：“跳过运行时安装。此构建映像不支持运行时版本选择”（在运行构建时出现） 985
错误：“无法验证 JobWorker 身份” 986
构建启动失败 986
访问本地缓存版本中的 GitHub 元数据 986
AccessDenied：报告组的存储桶所有者与 S3 存储桶的所有者不匹配... 987
使用以下命令创建 CodeBuild 项目时出现错误：“您的凭证缺少一个或多个必需的权限范围” AWS CodeConnections 987
错误：使用 Ubuntu 安装命令构建时出现“Sorry, no terminal at all requested - can't get input” ... 988
使用 Secrets Manager OAuth 后，Bitbucket 构建因身份验证错误而失败 989
配额 990
服务配额 990
其他限制 994
构建项目 994
Builds 995
计算实例集 995
Reports 996
标签 996
文档历史记录 998
早期更新 1016
..... mxxv

什么是 AWS CodeBuild?

AWS CodeBuild 是云端完全托管的生成服务。 CodeBuild 编译您的源代码，运行单元测试，并生成随时可以部署的工件。 CodeBuild 无需预置、管理和扩展自己的构建服务器。它提供了适用于常用编程语言的预先打包的构建环境以及 Apache Maven 和 Gradle 等构建工具。您还可以在中自定义构建环境 CodeBuild 以使用自己的构建工具。 CodeBuild 自动扩展以满足高峰构建请求。

CodeBuild 提供以下好处：

- 完全托管 — CodeBuild 无需设置、修补、更新和管理自己的构建服务器。
- 按需 CodeBuild 扩展 — 按需扩展以满足您的构建需求。您只需为使用的构建分钟数付费。
- 开箱即用 — CodeBuild 为最流行的编程语言提供预配置的构建环境。您只需指向您的构建脚本，即可开始首次构建。

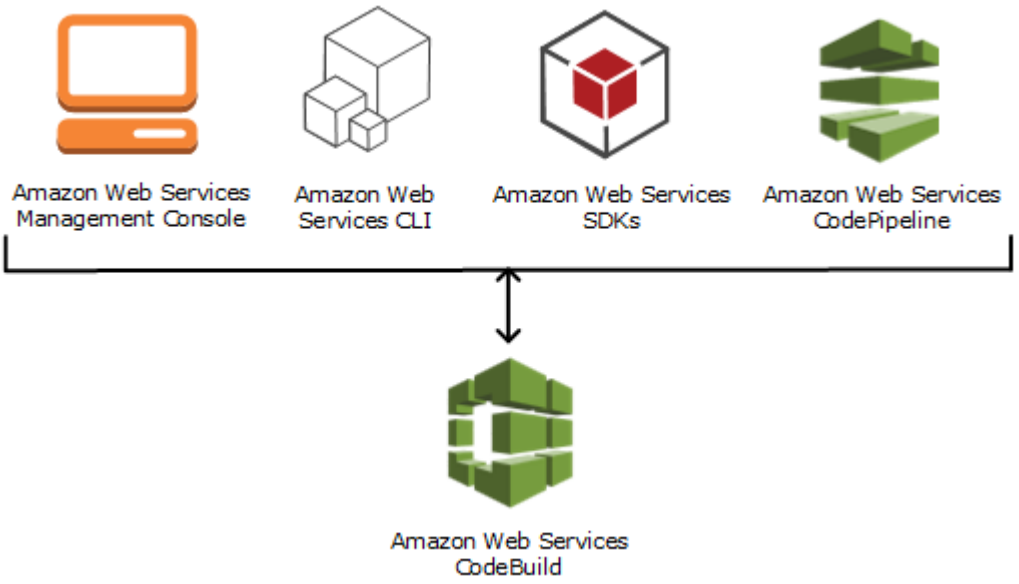
有关更多信息，请参阅 AWS CodeBuild。

注册获取 AWS 账户

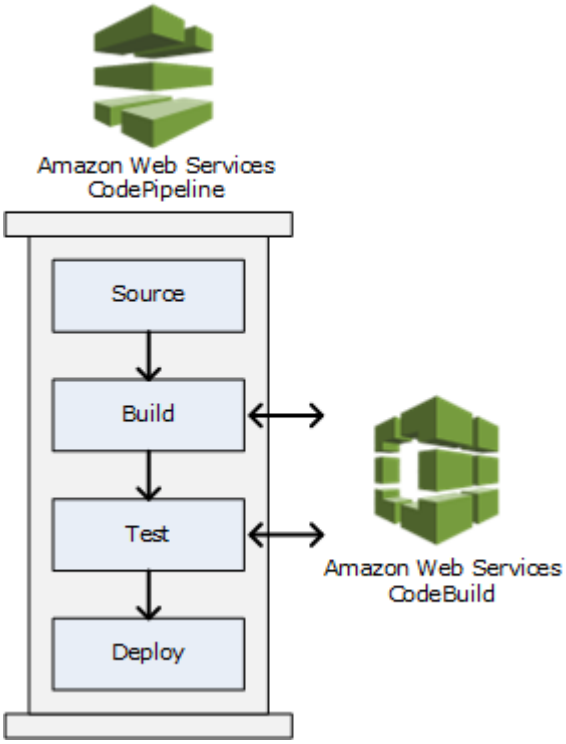
要开始使用 AWS，你需要一个 AWS 账户。有两种注册方式 AWS。如果您想完全控制账户配置和内容存储位置，或者您有受监管的工作负载（例如 HIPAA 或 FedRAMP），请使用注册（高级）。 AWS 如果您是新用户，想要使用您已经拥有的登录名登录 AWS 并创建具有预配置默认值的 select AWS 环境 AWS 区域，请使用 S ign up fo AWS r。

怎么跑 CodeBuild

您可以使用 AWS CodeBuild 或 AWS CodePipeline 控制台来运行 CodeBuild。您也可以使用 AWS Command Line Interface (AWS CLI) 或 AWS SDK 自动运行。 CodeBuild



如下图所示，您可以 CodeBuild 作为生成或测试操作添加到中管道的生成或测试阶段 AWS CodePipeline。 AWS CodePipeline 是一项持续交付服务，可用于对发布代码所需的步骤进行建模、可视化和自动化。其中包括构建您的代码。管道是一个描述发布流程中代码更改情况的工作流程结构。



CodePipeline 要用于创建管道然后添加 CodeBuild 生成或测试操作，请参阅 CodeBuild 搭配使用 CodePipeline。有关的更多信息 CodePipeline，请参阅《AWS CodePipeline 用户指南》。

CodeBuild 控制台还提供了一种快速搜索资源的方法，例如存储库、生成项目、部署应用程序和管道。选择转到资源或按下 / 键，然后键入资源的名称。任何匹配结果都会显示在列表中。搜索不区分大小写。您只能看到您有权查看的资源。有关更多信息，请参阅 在控制台中查看资源。

的定价 CodeBuild

有关更多信息，请参阅 CodeBuild 定价。

我该如何开始 CodeBuild？

我们建议您完成以下步骤：

1. 要了解更多信息 CodeBuild ，请阅读中的信息概念。
2. 按照 CodeBuild 中的说明在示例场景中进行@@ 实验通过控制台开始使用。
3. 按照 CodeBuild 中的说明在您自己的场景中@@ 使用计划构建。

AWS CodeBuild 概念

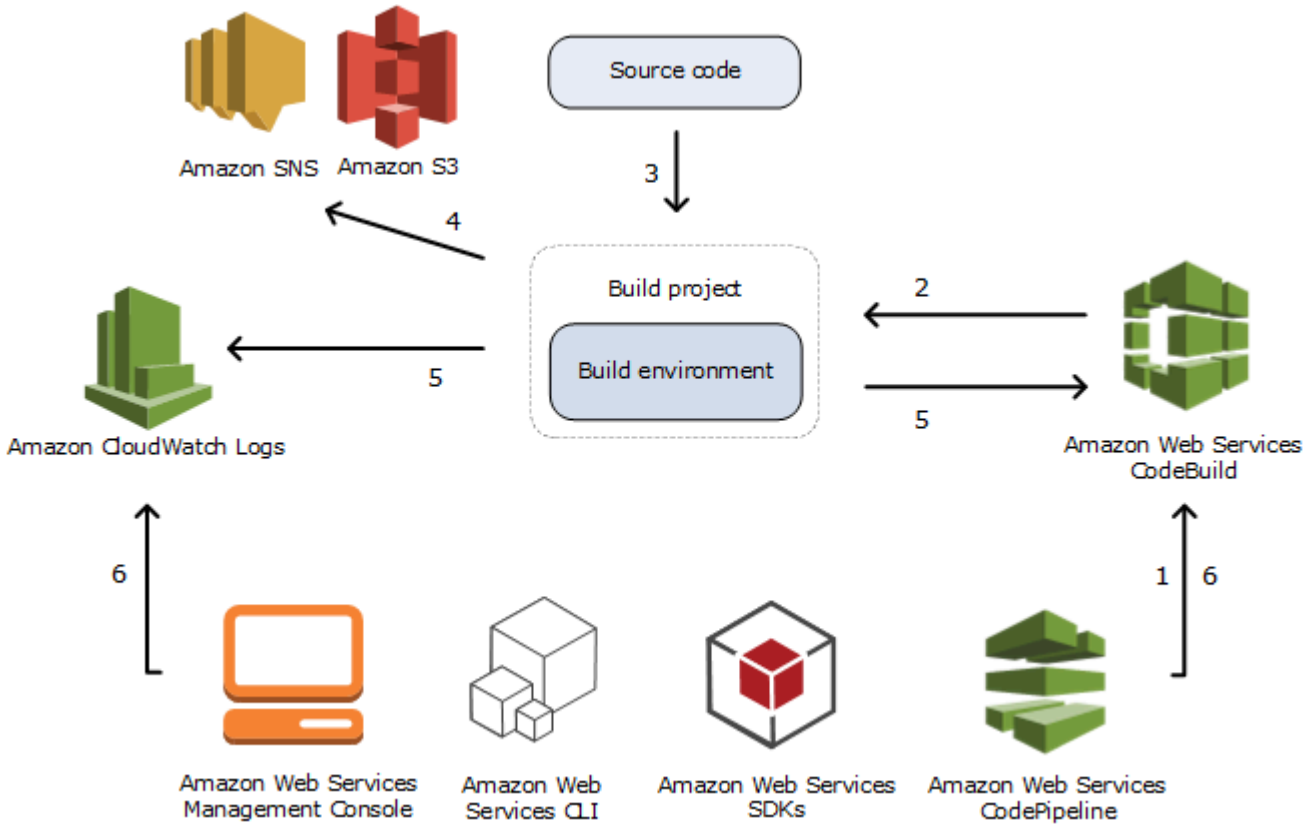
以下概念对于理解 CodeBuild 工作原理非常重要。

主题

- 如何 CodeBuild 运作
- 后续步骤

如何 CodeBuild 运作

下图显示了使用以下命令运行构建时会发生什么 CodeBuild：



1. 作为输入，您必须 CodeBuild 提供一个构建项目。构建项目包含有关如何运行构建的信息，包括从何处获取源代码、要使用的构建环境、要运行的构建命令以及将构建输出存储在何处。构建环境代表操作系统、编程语言运行时和用于运行构建的 CodeBuild 工具的组合。有关更多信息，请参阅:

   - 创建构建项目
   - 构建环境参考

2. CodeBuild 使用构建项目来创建构建环境。
3. CodeBuild 将源代码下载到构建环境中，然后使用构建规范 (buildspec)，该规范在构建项目中定义或直接包含在源代码中。buildspec 是一组生成命令和相关设置，采 CodeBuild 用 YAML 格式，用于运行构建。有关更多信息，请参阅Buildspec 参考。
4. 如果存在任何构建输出，则该构建环境会将其输出上传到 S3 存储桶。构建环境也可以执行您在 buildspec 中指定的任务（例如，将构建通知发送到 Amazon SNS 主题）。有关示例，请参阅构建通知示例。
5. 在构建运行时，构建环境会向 Amazon Logs CodeBuild 和 Amazon CloudWatch Logs 发送信息。
6. 在构建运行期间，您可以使用 AWS CodeBuild 控制台 AWS CLI、或 AWS 软件开发工具包从 Amazon Logs 中获取构建摘要信息， CodeBuild 并从 Amazon L CloudWatch ogs 中获取详

细的构建信息。如果您以前运行构建，则可以从中获取有限的构建信息 CodePipeline。 AWS CodePipeline

后续步骤

现在您已经了解了更多 AWS CodeBuild，我们建议您执行以下步骤：

1. 按照 CodeBuild 中的说明在示例场景中进行@@ 实验通过控制台开始使用。
2. 按照 CodeBuild 中的说明在您自己的场景中@@ 使用计划构建。

入门 CodeBuild

在以下教程中，您将使用 AWS CodeBuild 将示例源代码输入文件集合构建为源代码的可部署版本。

两个教程都有相同的输入和结果，但一个教程使用 AWS CodeBuild 控制台，另一个使用控制台 AWS CLI。

> **Important**
>
> 我们不建议您使用 AWS 根账户来完成本教程。

主题

- 开始使用 AWS CodeBuild 使用控制台
- 开始使用 AWS CodeBuild 使用 AWS CLI

开始使用 AWS CodeBuild 使用控制台

在本教程中，您使用 AWS CodeBuild 将示例源代码输入文件（编译输入构件或编译输入）的集合构建为源代码的可部署版本（编译输出构件或编译输出）。具体而言，你指示 CodeBuild 使用常见的编译工具 Apache Maven 将一组 Java 类文件构建到 Java 存档 (JAR) 文件中。您无需熟悉 Apache Maven 或 Java 即可完成本教程。

您可以 CodeBuild 通过 CodeBuild 控制台 AWS CodePipeline AWS CLI、或 AWS SDK 进行使用。本教程演示如何使用 CodeBuild 控制台。有关使用 CodePipeline 的信息，请参阅 CodeBuild 搭配使用 CodePipeline。

> **Important**
>
> 本教程中的步骤要求您创建资源（例如，S3 存储桶），这可能会导致您的 AWS 账户收费。其中包括与 Amazon S3 和 CloudWatch 日志相关的 AWS 资源和操作可能产生的费用。CodeBuild AWS KMS有关更多信息，请参阅AWS CodeBuild 定价、 Amazon S3 定价、AWS Key Management Service 定价和亚马逊 CloudWatch 定价。

主题

- 步骤 1：创建源代码

- 步骤 2：创建 buildspec 文件
- 步骤 3：创建两个 S3 存储桶
- 步骤 4：上传源代码和 buildspec 文件
- 步骤 5：创建构建项目
- 步骤 6：运行构建
- 步骤 7：查看汇总的构建信息
- 步骤 8：查看详细的构建信息
- 步骤 9：获取构建输出构件
- 步骤 10：删除 S3 存储桶
- 总结

步骤 1：创建源代码

（一部分：开始使用 AWS CodeBuild 使用控制台）

在此步骤中，您将创建要构建 CodeBuild 到输出存储桶的源代码。此源代码包含两个 Java 类文件和一个 Apache Maven 项目对象模型 (POM) 文件。

1. 在您的本地计算机或实例上的空目录中，创建此目录结构。

```
(root directory name)
    `-- src
         |-- main
         |     `-- java
         `-- test
               `-- java
```

2. 使用您选择的文本编辑器创建此文件，将其命名为 `MessageUtil.java`，然后保存在 `src/main/java` 目录中。

```
public class MessageUtil {
  private String message;

  public MessageUtil(String message) {
    this.message = message;
  }

  public String printMessage() {
```

```
    System.out.println(message);
    return message;
  }

  public String salutationMessage() {
    message = "Hi!" + message;
    System.out.println(message);
    return message;
  }
}
```

创建此类文件是用来输出传入的字符串。MessageUtil 构造函数用于设置字符串。printMessage 方法用于创建输出。salutationMessage 方法用于输出 Hi! 后跟字符串。

3. 创建此文件，将其命名为 TestMessageUtil.java，然后将它保存在 /src/test/java 目录中。

```
import org.junit.Test;
import org.junit.Ignore;
import static org.junit.Assert.assertEquals;

public class TestMessageUtil {

  String message = "Robert";
  MessageUtil messageUtil = new MessageUtil(message);

  @Test
  public void testPrintMessage() {
    System.out.println("Inside testPrintMessage()");
    assertEquals(message,messageUtil.printMessage());
  }

  @Test
  public void testSalutationMessage() {
    System.out.println("Inside testSalutationMessage()");
    message = "Hi!" + "Robert";
    assertEquals(message,messageUtil.salutationMessage());
  }
}
```

此类文件用于将 MessageUtil 类中的 message 变量设置为 Robert。然后，通过检查输出中是否出现字符串 Robert 和 Hi!Robert 来测试是否成功设置 message 变量。

4. 创建此文件，将其命名为 pom.xml，然后保存在根（顶级）目录中。

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.example</groupId>
  <artifactId>messageUtil</artifactId>
  <version>1.0</version>
  <packaging>jar</packaging>
  <name>Message Utility Java Sample App</name>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
      </plugin>
    </plugins>
  </build>
</project>
```

Apache Maven 使用此文件中的指令将 MessageUtil.java 和 TestMessageUtil.java 文件转换为名为 messageUtil-1.0.jar 的文件，然后运行指定测试。

此时，您的目录结构应如下所示。

```
(root directory name)
    |-- pom.xml
```

```
    `-- src
        |-- main
        |     `-- java
        |           `-- MessageUtil.java
        `-- test
              `-- java
                    `-- TestMessageUtil.java
```

步骤 2：创建 buildspec 文件

（上一步：步骤 1：创建源代码）

在此步骤中，您将创建一个构建规范 (build spec) 文件。buildspec 是 YAML 格式的编译命令和相关设置的集合， CodeBuild 用于运行构建。如果没有编译规范， CodeBuild 则无法成功地将您的构建输入转换为构建输出，也无法在构建环境中找到要上传到输出存储桶的构建输出对象。

创建此文件，将其命名为 `buildspec.yml`，然后保存在根（顶级）目录中。

```
version: 0.2

phases:
  install:
    runtime-versions:
      java: corretto11
  pre_build:
    commands:
      - echo Nothing to do in the pre_build phase...
  build:
    commands:
      - echo Build started on `date`
      - mvn install
  post_build:
    commands:
      - echo Build completed on `date`
artifacts:
  files:
    - target/messageUtil-1.0.jar
```

> **Important**
>
> 因为 buildspec 声明必须为有效的 YAML，所以 buildspec 声明中的空格至关重要。如果 buildspec 声明中的空格数与此不匹配，则构建可能会立即失败。您可以使用 YAML 验证程序测试 buildspec 声明是否为有效的 YAML。

> **Note**
>
> 您可以在创建构建项目时单独声明构建命令，而不是将 buildspec 文件包含在源代码中。如果您需要使用其他构建命令来构建源代码，而不是每次更新源代码存储库，这个方法很有用。有关更多信息，请参阅 buildspec 语法。

在此 buildspec 声明中：

- `version` 表示正在使用的 buildspec 标准的版本。此 buildspec 声明使用最新版本 `0.2`。
- `phases` 表示您可以指示 CodeBuild 运行命令的生成阶段。这些构建阶段包括 `install`、`pre_build`、`build` 和 `post_build`。您无法更改这些构建阶段名称的拼写，也无法创建更多构建阶段名称。

  在本示例中，在该`build`阶段 CodeBuild 运行`mvn install`命令。此命令指示 Apache Maven 编译和测试 Java 类文件，然后将编译完的文件打包为构建输出构件。出于完整性考虑，本示例的每个构建阶段中都放了几条 `echo` 命令。您稍后查看本教程中详细的构建信息时，这些 `echo` 命令的输出可以帮助您更好地理解 CodeBuild 运行命令的方式以及顺序。（尽管本示例中包含了所有构建阶段，但如果您不打算在某个构建阶段运行任何命令，则无需包含该构建阶段。）对于每个构建阶段，按所列顺序从头到尾逐一 CodeBuild 运行每个指定的命令。
- `artifacts`代表 CodeBuild上传到输出存储桶的一组生成输出项目。`files`表示要包含在编译输出中的文件。 CodeBuild 上传在构建环境的`target`相对目录中找到的单个`messageUtil-1.0.jar`文件。文件 `messageUtil-1.0.jar` 和目录 `target` 只是根据本示例中 Apache Maven 创建和存储构建输出构件的方式来命名的。在您自己的构建项目中，这些文件和目录名称会有所不同。

有关更多信息，请参阅Buildspec 参考。

此时，您的目录结构应如下所示。

```
(root directory name)
    |-- pom.xml
    |-- buildspec.yml
    `-- src
         |-- main
         |     `-- java
         |           `-- MessageUtil.java
         `-- test
               `-- java
                     `-- TestMessageUtil.java
```

步骤 3：创建两个 S3 存储桶

（上一步：步骤 2：创建 buildspec 文件）

尽管您可以为本教程使用单个存储桶，但使用两个存储桶使您可以更容易地查看构建输入的来源以及构建输出的去向。

- 其中一个存储桶（输入存储桶）用于存储构建输入。在本教程中，此输入存储桶的名称是codebuild-*region-ID*-*account-ID*-input-bucket，其中*region-ID*是存储桶的 AWS 区域，*account-ID*是您的 AWS 账户 ID。
- 另一个存储桶（输出存储桶）用于存储构建输出。在本教程中，此输出存储桶的名称为codebuild-*region-ID*-*account-ID*-output-bucket。

如果您为这些存储桶选择了不同的名称，请务必在本教程中使用它们。

这两个存储桶必须与您的版本位于同一个 AWS 区域。例如，如果您指示 CodeBuild 在美国东部（俄亥俄州）区域运行构建，则这些存储桶也必须位于美国东部（俄亥俄州）区域。

有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的创建存储桶。

> **Note**
>
> 尽管 CodeBuild 也支持存储在 CodeCommit GitHub、和 Bitbucket 存储库中的构建输入，但本教程并未向您展示如何使用它们。有关更多信息，请参阅 计划构建。

步骤 4：上传源代码和 buildspec 文件

（上一步：步骤 3：创建两个 S3 存储桶）

在此步骤中，您将源代码和 buildspec 文件添加到输入存储桶中。

使用操作系统的 ZIP 实用工具，创建一个名为 `MessageUtil.zip` 的文件，其中包含 `MessageUtil.java`、`TestMessageUtil.java`、`pom.xml` 和 `buildspec.yml`。

`MessageUtil.zip` 文件的目录结构必须如下所示。

```
MessageUtil.zip
    |-- pom.xml
    |-- buildspec.yml
    `-- src
         |-- main
         |     `-- java
         |           `-- MessageUtil.java
         `-- test
               `-- java
                     `-- TestMessageUtil.java
```

Important

请不要包含 *(root directory name)* 目录，而只包含 *(root directory name)* 目录中的目录和文件。

将 `MessageUtil.zip` 文件上传至名为 `codebuild-`*region-ID*`-`*account-ID*`-input-bucket` 的输入存储桶中。

Important

对于 CodeCommit GitHub、和 Bitbucket 存储库，按照惯例，您必须`buildspec.yml`在每个存储库的根（顶层）中存储一个名为的构建规范文件，或者将构建规范声明作为构建项目定义的一部分。请勿创建包含存储库源代码和 buildspec 文件的 ZIP 文件。

仅对于存储在 S3 存储桶中的构建输入，您必须创建一个包含源代码的 ZIP 文件和一个（按照惯例）位于根（顶级）位置的名为 `buildspec.yml` 的 buildspec 文件，或者将 buildspec 声明作为构建项目定义的一部分包含。

如果您要为 buildspec 文件使用其他名称，或者要在根位置之外的位置引用 buildspec，则可指定 buildspec 覆盖作为构建项目定义的一部分。有关更多信息，请参阅 buildspec 文件名称和存储位置。

步骤 5：创建构建项目

（上一步：步骤 4：上传源代码和 buildspec 文件）

在此步骤中，您将创建一个 AWS CodeBuild 用于运行构建的构建项目。构建项目包含有关如何运行构建的信息，包括从何处获取源代码、要使用的构建环境、要运行的构建命令以及将构建输出存储在何处。构建环境代表操作系统、编程语言运行时和用于运行构建的 CodeBuild 工具的组合。构建环境以 Docker 映像的形式表示。有关更多信息，请参阅 Docker 文档网站上的 Docker 概述。

在这个编译环境中，你指示 CodeBuild 使用包含 Java 开发套件 (JDK) 和 Apache Maven 版本的 Docker 镜像。

创建构建项目

1. 登录 AWS 管理控制台 并打开 AWS CodeBuild 控制台，地址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 使用 AWS 区域选择器选择支持 CodeBuild 的地 AWS 区。有关更多信息，请参阅 Amazon Web Services 一般参考 中的 AWS CodeBuild 端点和配额。
3. 如果显示 CodeBuild 信息页面，请选择创建构建项目。否则，请在导航窗格中，展开构建，选择构建项目，然后选择创建构建项目。
4. 在创建构建项目页面上的项目配置中，对于项目名称，输入此构建项目的名称（在此示例中为 `codebuild-demo-project`）。每个 AWS 账户的构建项目名称必须是唯一的。如果您使用其他名称，请确保在本教程中通篇使用它。

> **Note**
>
> 在创建构建项目页面上，您可能会看到类似于以下内容的错误消息：您没有权限执行此操作。这很可能是因为您以没有创建构建项目权限的用户 AWS 管理控制台 身份登录了。要解决此问题，请注销 AWS 管理控制台，然后使用属于以下 IAM 实体之一的证书重新登录：
>
> - 您 AWS 账户中的管理员用户。有关更多信息，请参阅用户指南中的创建您的第一个 AWS 账户 root 用户和群组。
> - 您的 AWS 账户中将`AWSCodeBuildAdminAccess`、`AmazonS3ReadOnlyAccess`、和`IAMFullAccess`托管策略附加到该用户或该用户所属的 IAM 群组的用户。如果您的 AWS 账户中没有具有这些权限的用户或群组，并且您无法将这些权限添加到您的用户或群组，请联系您的 AWS 账户管理员寻求帮助。有关更多信息，请参阅 AWS 的托管（预定义）策略 AWS CodeBuild。

> 这两个选项都可让您获得创建构建项目所需的管理员权限，以便您能够完成本教程。建议您始终使用完成任务所需的最低权限。有关更多信息，请参阅 AWS CodeBuild 权限参考。

5. 在源中，对于源提供商，选择 Amazon S3。
6. 对于 Bucket，选择 codebuild-***region-ID*-*account-ID***-input- bucket。
7. 对于 S3 对象键，输入 **`MessageUtil.zip`**。
8. 在环境中，对于环境映像，让托管映像处于选中状态。
9. 对于操作系统，选择 Amazon Linux。
10. 对于运行时，选择标准。
11. 对于图片，选择 aws/codebuild /amazon linux-x86_64-standard: corretto11。
12. 在服务角色中，将新建服务角色保持选中状态，并将角色名称保持不变。
13. 对于 buildspec，将使用 buildspec 文件保留为选中状态。
14. 在构件中，对于类型，选择 Amazon S3。
15. 对于存储桶名称，选择 codebuild-***region-ID*-*account-ID***-output- bucket。
16. 将名称和路径留空。
17. 选择创建构建项目。

步骤 6：运行构建

（上一步：步骤 5：创建构建项目）

在此步骤中，您将指示 AWS CodeBuild 使用构建项目中的设置运行构建。

运行构建

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择构建项目。
3. 在构建项目列表中，选择 codebuild-demo-project，然后选择启动构建。构建会立即开始。

步骤 7：查看汇总的构建信息

（上一步：步骤 6：运行构建）

在此步骤中，您将查看有关构建状态的汇总信息。

查看汇总的构建信息

1. 如果未显示 codebuild-demo-project: ***<build-ID>*** 页面，请在导航栏中选择 “生成历史记录”。接下来，在构建项目列表中，对于项目，选择与 codebuild-demo-project 对应的构建运行链接。应该只有一个匹配的链接。（如果您之前已完成本教程，请在已完成列中选择具有最新值的链接。）
2. 在构建状态页面上，在阶段详细信息中，应显示以下构建阶段，并且状态列中为已成功：

   - SUBMITTED
   - QUEUED
   - PROVISIONING
   - DOWNLOAD_SOURCE
   - INSTALL
   - PRE_BUILD
   - BUILD
   - POST_BUILD
   - UPLOAD_ARTIFACTS
   - FINALIZING
   - COMPLETED

   在构建状态中，应显示已成功。

   如果您看到的是正在进行，请选择刷新按钮。
3. 在每个构建阶段的旁边，持续时间值表示构建阶段持续的时间。结束时间值表示构建阶段的结束时间。

步骤 8：查看详细的构建信息

（上一步：步骤 7：查看汇总的构建信息）

在此步骤中，您可以在 L CloudWatch ogs 中查看有关您的构建的详细信息。

> **Note**
>
> 为了保护敏感信息，以下内容隐藏在 CodeBuild 日志中：
>
> - AWS 访问密钥 ID。有关更多信息，请参阅《AWS Identity and Access Management 用户指南》中的管理 IAM 用户的访问密钥。
> - 使用参数存储指定的字符串。有关更多信息，请参阅《Amazon EC2 Systems Manager 用户指南》中的 Systems Manager Parameter Store 和 Systems Manager Parameter Store 控制台演练。
> - 使用指定的字符串 AWS Secrets Manager。有关更多信息，请参阅 密钥管理。

查看详细的构建信息

1. 上一步完成后，构建详细页面继续显示，Build logs 中显示了构建日志的最后 10000 行内容。要在“日志”中查看整个编译 CloudWatch 日志，请选择“查看整个日志”链接。
2. 在 CloudWatch 日志日志流中，您可以浏览日志事件。默认情况下，只显示最近的一组日志事件。要查看以前的日志事件，请滚动到列表开头。
3. 在本教程中，大多数日志事件包含的是关于 CodeBuild 下载构建相关文件并将其安装到构建环境中的详细信息，您可能并不关心这些信息。您可以使用筛选事件框来减少显示的信息。例如，如果您在筛选事件中输入 `"[INFO]"`，则仅显示那些包含 `[INFO]` 的事件。有关更多信息，请参阅《亚马逊 CloudWatch 用户指南》中的筛选条件和模式语法。

步骤 9：获取构建输出构件

（上一步：步骤 8：查看详细的构建信息）

在此步骤中，您将获得 CodeBuild 生成并上传到输出存储桶的`messageUtil-1.0.jar`文件。

您可以使用 CodeBuild 控制台或 Amazon S3 控制台来完成此步骤。

获取编译输出工件 (AWS CodeBuild 控制台）

1. 在 CodeBuild 控制台仍处于打开状态且仍显示上一步中的编译详细信息页面的情况下，选择编译详细信息选项卡并向下滚动到 Artifacts 部分。

> **Note**
>
> 如果未显示构建详细信息页面，请在导航栏中选择构建历史记录，然后选择构建运行链接。

2. 指向 Amazon S3 文件夹的链接位于构件上传位置下方。该链接会打开 Amazon S3 文件夹，您可以在这里找到 `messageUtil-1.0.jar` 构建输出构件文件。

获取构建输出构件（Amazon S3 控制台）

1. 打开 Amazon S3 控制台，网址为 https://console.aws.amazon.com/s3/。
2. 打开 `codebuild-`*`region-ID`*`-`*`account-ID`*`-output-bucket`。
3. 打开 `codebuild-demo-project` 文件夹。
4. 打开 `target` 文件夹，您可以在此处找到 `messageUtil-1.0.jar` 构建输出构件文件。

步骤 10：删除 S3 存储桶

（上一步：步骤 9：获取构建输出构件）

为防止持续向您的 AWS 账户收费，您可以删除本教程中使用的输入和输出存储桶。有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的删除或清空存储桶。

如果您使用 IAM 用户或管理员 IAM 用户删除这些存储桶，则该用户必须具有更多访问权限。在标记（*`### BEGIN ADDING STATEMENT HERE ###`*和*`### END ADDING STATEMENTS HERE ###`*）之间添加以下语句到用户的现有访问策略中。

此语句中的省略号 (...) 旨在力求简洁。请勿删除现有访问策略中的任何语句。请勿在策略中输入这些省略号。

JSON

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

```
        "s3:DeleteBucket",
        "s3:DeleteObject"
      ],
      "Resource": "*"
    }
  ]
}
```

总结

在本教程中，您曾经 AWS CodeBuild 将一组 Java 类文件构建到 JAR 文件中。然后查看了构建的结果。

您现在可以尝试 CodeBuild 在自己的场景中使用。按照计划构建中的说明进行操作。如果您觉得自己还没准备好，可以尝试构建一些示例。有关更多信息，请参阅 使用基于案例的示例 CodeBuild。

开始使用 AWS CodeBuild 使用 AWS CLI

在本教程中，您将使用 AWS CodeBuild 将示例源代码输入文件（称为编译输入构件或编译输入）的集合构建为源代码的可部署版本（称为编译输出构件或编译输出）。具体而言，你指示 CodeBuild 使用常见的编译工具 Apache Maven 将一组 Java 类文件构建到 Java 存档 (JAR) 文件中。您无需熟悉 Apache Maven 或 Java 即可完成本教程。

您可以 CodeBuild 通过 CodeBuild 控制台 AWS CodePipeline AWS CLI、或 AWS SDK 进行使用。本教程演示了如何 CodeBuild 与 AWS CLI. 有关使用的信息 CodePipeline，请参阅 CodeBuild 搭配使用 CodePipeline。

> **Important**
>
> 本教程中的步骤要求您创建资源（例如，S3 存储桶），这可能会导致您的 AWS 账户收费。其中包括与 Amazon S3 和 CloudWatch 日志相关的 AWS 资源和操作可能产生的费用。CodeBuild AWS KMS有关更多信息，请参阅CodeBuild定价、 Amazon S3 定价、AWS Key Management Service 定价和亚马逊 CloudWatch 定价。

主题

- 步骤 1：创建源代码
- 步骤 2：创建 buildspec 文件

- 步骤 3：创建两个 S3 存储桶
- 步骤 4：上传源代码和 buildspec 文件
- 步骤 5：创建构建项目
- 步骤 6：运行构建
- 步骤 7：查看汇总的构建信息
- 步骤 8：查看详细的构建信息
- 步骤 9：获取构建输出构件
- 步骤 10：删除 S3 存储桶
- 总结

步骤 1：创建源代码

（一部分：开始使用 AWS CodeBuild 使用 AWS CLI）

在此步骤中，您将创建要构建 CodeBuild 到输出存储桶的源代码。此源代码包含两个 Java 类文件和一个 Apache Maven 项目对象模型 (POM) 文件。

1. 在您的本地计算机或实例上的空目录中，创建此目录结构。

```
(root directory name)
    `-- src
         |-- main
         |     `-- java
         `-- test
               `-- java
```

2. 使用您选择的文本编辑器创建此文件，将其命名为 `MessageUtil.java`，然后保存在 `src/main/java` 目录中。

```
public class MessageUtil {
  private String message;

  public MessageUtil(String message) {
    this.message = message;
  }

  public String printMessage() {
    System.out.println(message);
```

```
    return message;
  }

  public String salutationMessage() {
    message = "Hi!" + message;
    System.out.println(message);
    return message;
  }
}
```

创建此类文件是用来输出传入的字符串。MessageUtil 构造函数用于设置字符串。printMessage 方法用于创建输出。salutationMessage 方法用于输出 Hi! 后跟字符串。

3. 创建此文件，将其命名为 TestMessageUtil.java，然后将它保存在 /src/test/java 目录中。

```
import org.junit.Test;
import org.junit.Ignore;
import static org.junit.Assert.assertEquals;

public class TestMessageUtil {

  String message = "Robert";
  MessageUtil messageUtil = new MessageUtil(message);

  @Test
  public void testPrintMessage() {
    System.out.println("Inside testPrintMessage()");
    assertEquals(message,messageUtil.printMessage());
  }

  @Test
  public void testSalutationMessage() {
    System.out.println("Inside testSalutationMessage()");
    message = "Hi!" + "Robert";
    assertEquals(message,messageUtil.salutationMessage());
  }
}
```

此类文件用于将 MessageUtil 类中的 message 变量设置为 Robert。然后，通过检查输出中是否出现字符串 Robert 和 Hi!Robert 来测试是否成功设置 message 变量。

4. 创建此文件，将其命名为 pom.xml，然后保存在根（顶级）目录中。

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.example</groupId>
  <artifactId>messageUtil</artifactId>
  <version>1.0</version>
  <packaging>jar</packaging>
  <name>Message Utility Java Sample App</name>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
      </plugin>
    </plugins>
  </build>
</project>
```

Apache Maven 使用此文件中的指令将 MessageUtil.java 和 TestMessageUtil.java 文件转换为名为 messageUtil-1.0.jar 的文件，然后运行指定测试。

此时，您的目录结构应如下所示。

```
(root directory name)
    |-- pom.xml
    `-- src
         |-- main
         |     `-- java
         |           `-- MessageUtil.java
```

```
        `-- test
              `-- java
                    `-- TestMessageUtil.java
```

步骤 2：创建 buildspec 文件

（上一步：步骤 1：创建源代码）

在此步骤中，您将创建一个构建规范 (build spec) 文件。buildspec 是 YAML 格式的编译命令和相关设置的集合，CodeBuild 用于运行构建。如果没有编译规范，CodeBuild 则无法成功地将您的构建输入转换为构建输出，也无法在构建环境中找到要上传到输出存储桶的构建输出对象。

创建此文件，将其命名为 `buildspec.yml`，然后保存在根（顶级）目录中。

```
version: 0.2

phases:
  install:
    runtime-versions:
      java: corretto11
  pre_build:
    commands:
      - echo Nothing to do in the pre_build phase...
  build:
    commands:
      - echo Build started on `date`
      - mvn install
  post_build:
    commands:
      - echo Build completed on `date`
artifacts:
  files:
    - target/messageUtil-1.0.jar
```

> **Important**
>
> 因为 buildspec 声明必须为有效的 YAML，所以 buildspec 声明中的空格至关重要。如果 buildspec 声明中的空格数与此不匹配，则构建可能会立即失败。您可以使用 YAML 验证程序测试 buildspec 声明是否为有效的 YAML。

> **Note**
>
> 您可以在创建构建项目时单独声明构建命令，而不是将 buildspec 文件包含在源代码中。如果您需要使用其他构建命令来构建源代码，而不是每次更新源代码存储库，这个方法很有用。有关更多信息，请参阅 buildspec 语法。

在此 buildspec 声明中：

- `version` 表示正在使用的 buildspec 标准的版本。此 buildspec 声明使用最新版本 `0.2`。
- `phases` 表示您可以指示 CodeBuild 运行命令的生成阶段。这些构建阶段包括 `install`、`pre_build`、`build` 和 `post_build`。您无法更改这些构建阶段名称的拼写，也无法创建更多构建阶段名称。

  在本示例中，在该`build`阶段 CodeBuild 运行`mvn install`命令。此命令指示 Apache Maven 编译和测试 Java 类文件，然后将编译完的文件打包为构建输出构件。出于完整性考虑，本示例的每个构建阶段中都放了几条 `echo` 命令。您稍后查看本教程中详细的构建信息时，这些 `echo` 命令的输出可以帮助您更好地理解 CodeBuild 运行命令的方式以及顺序。（尽管本示例中包含了所有构建阶段，但如果您不打算在某个构建阶段运行任何命令，则无需包含该构建阶段。）对于每个构建阶段，按所列顺序从头到尾逐一 CodeBuild 运行每个指定的命令。
- `artifacts`代表 CodeBuild上传到输出存储桶的一组生成输出项目。`files`表示要包含在编译输出中的文件。 CodeBuild 上传在构建环境的`target`相对目录中找到的单个`messageUtil-1.0.jar`文件。文件 `messageUtil-1.0.jar` 和目录 `target` 只是根据本示例中 Apache Maven 创建和存储构建输出构件的方式来命名的。在您自己的构建项目中，这些文件和目录名称会有所不同。

有关更多信息，请参阅Buildspec 参考。

此时，您的目录结构应如下所示。

```
(root directory name)
    |-- pom.xml
    |-- buildspec.yml
    `-- src
         |-- main
         |     `-- java
         |           `-- MessageUtil.java
         `-- test
               `-- java
```

```
                    `-- TestMessageUtil.java
```

步骤 3：创建两个 S3 存储桶

（上一步：步骤 2：创建 buildspec 文件）

尽管您可以为本教程使用单个存储桶，但使用两个存储桶使您可以更容易地查看构建输入的来源以及构建输出的去向。

- 其中一个存储桶（输入存储桶）用于存储构建输入。在本教程中，此输入存储桶的名称是`codebuild-region-ID-account-ID-input-bucket`，其中*`region-ID`*是存储桶的 AWS 区域，*`account-ID`*是您的 AWS 账户 ID。
- 另一个存储桶（输出存储桶）用于存储构建输出。在本教程中，此输出存储桶的名称为`codebuild-region-ID-account-ID-output-bucket`。

如果您为这些存储桶选择了不同的名称，请务必在本教程中使用它们。

这两个存储桶必须与您的版本位于同一个 AWS 区域。例如，如果您指示 CodeBuild 在美国东部（俄亥俄州）区域运行构建，则这些存储桶也必须位于美国东部（俄亥俄州）区域。

有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的创建存储桶。

> **Note**
>
> 尽管 CodeBuild 也支持存储在 CodeCommit GitHub、和 Bitbucket 存储库中的构建输入，但本教程并未向您展示如何使用它们。有关更多信息，请参阅 计划构建。

步骤 4：上传源代码和 buildspec 文件

（上一步：步骤 3：创建两个 S3 存储桶）

在此步骤中，您将源代码和 buildspec 文件添加到输入存储桶中。

使用操作系统的 ZIP 实用工具，创建一个名为 `MessageUtil.zip` 的文件，其中包含 `MessageUtil.java`、`TestMessageUtil.java`、`pom.xml` 和 `buildspec.yml`。

`MessageUtil.zip` 文件的目录结构必须如下所示。

```
MessageUtil.zip
```

```
|-- pom.xml
|-- buildspec.yml
`-- src
     |-- main
     |     `-- java
     |           `-- MessageUtil.java
     `-- test
           `-- java
                 `-- TestMessageUtil.java
```

> **Important**
>
> 请不要包含 *(root directory name)* 目录，而只包含 *(root directory name)* 目录中的目录和文件。

将 MessageUtil.zip 文件上传至名为 codebuild-*region-ID*-*account-ID*-input-bucket 的输入存储桶中。

> **Important**
>
> 对于 CodeCommit GitHub、和 Bitbucket 存储库，按照惯例，您必须buildspec.yml在每个存储库的根（顶层）中存储一个名为的构建规范文件，或者将构建规范声明作为构建项目定义的一部分。请勿创建包含存储库源代码和 buildspec 文件的 ZIP 文件。
>
> 仅对于存储在 S3 存储桶中的构建输入，您必须创建一个包含源代码的 ZIP 文件和一个（按照惯例）位于根（顶级）位置的名为 buildspec.yml 的 buildspec 文件，或者将 buildspec 声明作为构建项目定义的一部分包含。
>
> 如果您要为 buildspec 文件使用其他名称，或者要在根位置之外的位置引用 buildspec，则可指定 buildspec 覆盖作为构建项目定义的一部分。有关更多信息，请参阅 buildspec 文件名称和存储位置。

步骤 5：创建构建项目

（上一步：步骤 4：上传源代码和 buildspec 文件）

在此步骤中，您将创建一个 AWS CodeBuild 用于运行构建的构建项目。构建项目包含有关如何运行构建的信息，包括从何处获取源代码、要使用的构建环境、要运行的构建命令以及将构建输出存储在何

处。构建环境代表操作系统、编程语言运行时和用于运行构建的 CodeBuild 工具的组合。构建环境以 Docker 映像的形式表示。有关更多信息，请参阅 Docker 文档网站上的 Docker 概述。

在这个编译环境中，你指示 CodeBuild 使用包含 Java 开发套件 (JDK) 和 Apache Maven 版本的 Docker 镜像。

创建构建项目

1. AWS CLI 使用运行create-project命令：

```
aws codebuild create-project --generate-cli-skeleton
```

JSON-formatted 数据出现在输出中。将数据复制到安装`create-project.json`在本地计算机或实例上的某个位置命名的文件中。 AWS CLI 如果您选择使用其他文件名，请务必在本教程中使用该名称。

按照以下格式修改所复制的数据，然后保存结果：

```
{
  "name": "codebuild-demo-project",
  "source": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-bucket/MessageUtil.zip"
  },
  "artifacts": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-output-bucket"
  },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "aws/codebuild/standard:5.0",
    "computeType": "BUILD_GENERAL1_SMALL"
  },
  "serviceRole": "serviceIAMRole"
}
```

*serviceIAMRole*替换为 CodeBuild 服务角色的亚马逊资源名称 (ARN)（例如，`arn:aws:iam::`*account-ID*`:role/`*role-name*）。要创建该文件，请参阅 CodeBuild 允许与其他人互动 AWS 务。

在此数据中：

- name 表示此构建项目的必需标识符（在本示例中为 codebuild-demo-project）。构建项目名称在您账户的所有构建项目中必须是唯一的。
- 对于 source，type 是一个必需值，表示源代码的存储库类型（在本示例中，S3 表示 Amazon S3 存储桶)。
- 对于 source，location 表示源代码的路径（在本示例中，为输入存储桶名称后跟 ZIP 文件名称）。
- 对于 artifacts，type 是一个必需值，表示构建输出构件的存储库类型（在本示例中，S3 表示 Amazon S3 存储桶）。
- 对于 artifacts，location 表示您先前创建或识别的输出存储桶的名称（在本示例中为 codebuild-*region-ID*-*account-ID*-output-bucket）。
- 对于 environment，type 是表示构建环境类型的必填值（在本例中为 LINUX_CONTAINER）。
- For image 是一个必填值environment，表示该构建项目使用的 Docker 镜像名称和标签组合，由 Docker 镜像存储库类型（在本示例中，aws/codebuild/standard:5.0对于 Docker 镜像存储库中的 D CodeBuild ocker 镜像）指定。aws/codebuild/standard是 Docker 镜像的名称。5.0是 Docker 镜像的标签。

  要查找您可以在自己方案中使用的更多 Docker 映像，请参阅构建环境参考。
- 对于environment，computeType是表示所 CodeBuild 用计算资源的必填值（在本示例中为BUILD_GENERAL1_SMALL）。

> **Note**
>
> 原始 JSON-formatted 数据中的其他可用值，例如、descriptionbuildspec、auth（包括type和resource）、、pathnamespaceType、name（用于artifacts）、packaging、environmentVariables（包括name和value）、timeoutInMinutesencryptionKey、和tags（包括key和value）是可选的。本教程中未使用这些值，因此它们没有在这里显示。有关更多信息，请参阅 创建构建项目 (AWS CLI)。

2. 切换到您刚才保存的文件所在的目录，然后再次运行 create-project 命令。

```
aws codebuild create-project --cli-input-json file://create-project.json
```

如果成功，输出中将显示与此类似的数据。

```
{
  "project": {
    "name": "codebuild-demo-project",
    "serviceRole": "serviceIAMRole",
    "tags": [],
    "artifacts": {
      "packaging": "NONE",
      "type": "S3",
      "location": "codebuild-region-ID-account-ID-output-bucket",
      "name": "message-util.zip"
    },
    "lastModified": 1472661575.244,
    "timeoutInMinutes": 60,
    "created": 1472661575.244,
    "environment": {
      "computeType": "BUILD_GENERAL1_SMALL",
      "image": "aws/codebuild/standard:5.0",
      "type": "LINUX_CONTAINER",
      "environmentVariables": []
    },
    "source": {
      "type": "S3",
      "location": "codebuild-region-ID-account-ID-input-bucket/MessageUtil.zip"
    },
    "encryptionKey": "arn:aws:kms:region-ID:account-ID:alias/aws/s3",
    "arn": "arn:aws:codebuild:region-ID:account-ID:project/codebuild-demo-project"
  }
}
```

- `project` 表示有关此构建项目的信息。
  - `tags` 表示已经声明的所有标签。
  - `packaging` 表示构建输出构件将如何存储在输出存储桶中。NONE 表示在输出存储桶中创建文件夹。构建输出构件存储在该文件夹中。
  - `lastModified` 表示构建项目最后一次更改的时间，采用 Unix 时间格式。
  - `timeoutInMinutes`表示生成尚未完成时 CodeBuild 停止构建的分钟数。（默认为 60 分钟。）
  - `created` 表示构建项目的创建时间，采用 Unix 时间格式。

- `environmentVariables`表示已声明并在构建期间 CodeBuild 可供使用的任何环境变量。
- `encryptionKey`代表 CodeBuild 用于加密构建输出构件的客户管理密钥的 ARN。
- `arn` 表示构建项目的 ARN。

> **Note**
>
> 运行create-project命令后，可能会输出类似于以下内容的错误消息：用户：无权执行：***user-ARN***codebuild: CreateProject 。这很可能是因为您使用没有足够权限 AWS CLI 的用户的证书来配置的，无法使用该用户的证书 CodeBuild 来创建构建项目。要修复此问题，请使用属于以下任一 IAM 实体的凭证配置 AWS CLI ：
>
> - 您 AWS 账户中的管理员用户。有关更多信息，请参阅用户指南中的创建您的第一个 AWS 账户 root 用户和群组。
> - 您的 AWS 账户中将`AWSCodeBuildAdminAccess`、`AmazonS3ReadOnlyAccess`、和`IAMFullAccess`托管策略附加到该用户或该用户所属的 IAM 群组的用户。如果您的 AWS 账户中没有具有这些权限的用户或群组，并且您无法将这些权限添加到您的用户或群组，请联系您的 AWS 账户管理员寻求帮助。有关更多信息，请参阅 AWS 的托管（预定义）策略 AWS CodeBuild。

步骤 6：运行构建

（上一步：步骤 5：创建构建项目）

在此步骤中，您将指示 AWS CodeBuild 使用构建项目中的设置运行构建。

运行构建

1. AWS CLI 使用运行start-build命令：

```
aws codebuild start-build --project-name project-name
```

   *project-name*替换为上一步中的构建项目名称（例如，`codebuild-demo-project`）。

2. 如果成功，输出中将显示与以下内容类似的数据：

```
{
  "build": {
```

```
    "buildComplete": false,
    "initiator": "user-name",
    "artifacts": {
      "location": "arn:aws:s3:::codebuild-region-ID-account-ID-output-bucket/
message-util.zip"
    },
    "projectName": "codebuild-demo-project",
    "timeoutInMinutes": 60,
    "buildStatus": "IN_PROGRESS",
    "environment": {
      "computeType": "BUILD_GENERAL1_SMALL",
      "image": "aws/codebuild/standard:5.0",
      "type": "LINUX_CONTAINER",
      "environmentVariables": []
    },
    "source": {
      "type": "S3",
      "location": "codebuild-region-ID-account-ID-input-bucket/MessageUtil.zip"
    },
    "currentPhase": "SUBMITTED",
    "startTime": 1472848787.882,
    "id": "codebuild-demo-project:0cfbb6ec-3db9-4e8c-992b-1ab28EXAMPLE",
    "arn": "arn:aws:codebuild:region-ID:account-ID:build/codebuild-demo-
project:0cfbb6ec-3db9-4e8c-992b-1ab28EXAMPLE"
  }
}
```

- build 表示有关此构建的信息。
  - buildComplete 表示构建是否完成 (true)。否则为 false。
  - initiator 表示启动构建的实体。
  - artifacts 表示有关构建输出的信息，包括其位置。
  - projectName 表示构建项目的名称。
  - buildStatus 表示运行 start-build 命令时当前构建的状态。
  - currentPhase 表示运行 start-build 命令时的当前构建阶段。
  - startTime 表示构建过程开始的时间，采用 Unix 时间格式。
  - id 表示构建的 ID。
  - arn 表示构建的 ARN。

记下此 id 值。您在下一个步骤中需要用到它。

步骤 7：查看汇总的构建信息

（上一步：步骤 6：运行构建）

在此步骤中，您将查看有关构建状态的汇总信息。

查看汇总的构建信息

- 使用运行 AWS CLI 该batch-get-builds命令。

```
aws codebuild batch-get-builds --ids id
```

*id*替换为上一步输出中出现的id值。

如果成功，输出中将显示与此类似的数据。

```
{
  "buildsNotFound": [],
  "builds": [
    {
      "buildComplete": true,
      "phases": [
        {
          "phaseStatus": "SUCCEEDED",
          "endTime": 1472848788.525,
          "phaseType": "SUBMITTED",
          "durationInSeconds": 0,
          "startTime": 1472848787.882
        },
        ... The full list of build phases has been omitted for brevity ...
        {
          "phaseType": "COMPLETED",
          "startTime": 1472848878.079
        }
      ],
      "logs": {
        "groupName": "/aws/codebuild/codebuild-demo-project",
```

```
        "deepLink": "https://console.aws.amazon.com/cloudwatch/home?region=region-
ID#logEvent:group=/aws/codebuild/codebuild-demo-project;stream=38ca1c4a-e9ca-4dbc-
bef1-d52bfEXAMPLE",
        "streamName": "38ca1c4a-e9ca-4dbc-bef1-d52bfEXAMPLE"
      },
      "artifacts": {
        "md5sum": "MD5-hash",
        "location": "arn:aws:s3:::codebuild-region-ID-account-ID-output-bucket/
message-util.zip",
        "sha256sum": "SHA-256-hash"
      },
      "projectName": "codebuild-demo-project",
      "timeoutInMinutes": 60,
      "initiator": "user-name",
      "buildStatus": "SUCCEEDED",
      "environment": {
        "computeType": "BUILD_GENERAL1_SMALL",
        "image": "aws/codebuild/standard:5.0",
        "type": "LINUX_CONTAINER",
        "environmentVariables": []
      },
      "source": {
        "type": "S3",
        "location": "codebuild-region-ID-account-ID-input-bucket/MessageUtil.zip"
      },
      "currentPhase": "COMPLETED",
      "startTime": 1472848787.882,
      "endTime": 1472848878.079,
      "id": "codebuild-demo-project:38ca1c4a-e9ca-4dbc-bef1-d52bfEXAMPLE",
      "arn": "arn:aws:codebuild:region-ID:account-ID:build/codebuild-demo-
project:38ca1c4a-e9ca-4dbc-bef1-d52bfEXAMPLE"
    }
  ]
}
```

- buildsNotFound 表示所有不具备信息的构建的构建 ID。在本示例中，其应该为空。
- builds 表示有关每个具备信息的构建项目的信息。在本示例中，输出中只显示了有关一个构建项目的信息。
  - phases 表示 CodeBuild 在生成过程中运行的一组生成阶段。有关每个构建阶段的信息将分别列出，其中包含：startTime、endTime 和 durationInSeconds (采用 Unix 时间格式的构建阶段开始时间和结束时间，以及构建阶段的持续时间，以秒为单位)，以及 phaseType (如

SUBMITTED、PROVISIONING、DOWNLOAD_SOURCE、INSTALL、PRE_BUILD、BUILD、POST_BU FINALIZING 或 COMPLETED)，还有 phaseStatus (如 SUCCEEDED、FAILED、FAULT、TIMED_OUT、IN_PROGRESS 或 STOPPED)。首次运行 batch-get-builds 命令时，可能不会有很多（或没有）阶段。使用相同构建 ID 再次运行 batch-get-builds 命令后，输出中应当会出现更多构建阶段。

- logs代表亚马逊 CloudWatch 日志中有关构建日志的信息。
- md5sum并sha256sum表示版本输出工件的 MD5 和 SHA-256 哈希值。只有在构建项目的 packaging 值设置为 ZIP 时，这些内容才会显示在输出中。（在本教程中您未设置此值。）您可以将这些哈希值和校验和工具一起使用，确认文件的完整性和真实性。

> **Note**
>
> 您还可以使用 Amazon S3 控制台查看这些哈希值。选中构建输出构件旁边的框，然后依次选择操作和属性。在属性窗格中，展开元数据，然后查看 x-amz-meta-codebuild-content-md5 和 x-amz-meta-codebuild-content-sha256 的值。（在 Amazon S3 控制台中，不应将构建输出项目的 ETag 值解释为 MD5 或 SHA-256哈希。）
>
> 如果您使用 AWS SDK 获取这些哈希值，则这些值的名称为codebuild-content-md5和。codebuild-content-sha256

- endTime 表示构建过程结束的时间，采用 Unix 时间格式。

> **Note**
>
> Amazon S3 元数据有一个名为buildArn的 CodeBuild 标头，x-amz-meta-codebuild-buildarn其中包含将项目发布到 Amazon S3 的 CodeBuild 版本的。添加 buildArn 是为了允许对通知进行源跟踪并引用生成构件的构建。

步骤 8：查看详细的构建信息

（上一步：步骤 7：查看汇总的构建信息）

在此步骤中，您可以在 L CloudWatch ogs 中查看有关您的构建的详细信息。

> **Note**
>
> 为了保护敏感信息，以下内容隐藏在 CodeBuild 日志中：
>
> - AWS 访问密钥 ID。有关更多信息，请参阅《AWS Identity and Access Management 用户指南》中的管理 IAM 用户的访问密钥。
> - 使用参数存储指定的字符串。有关更多信息，请参阅《Amazon EC2 Systems Manager 用户指南》中的 Systems Manager Parameter Store 和 Systems Manager Parameter Store 控制台演练。
> - 使用指定的字符串 AWS Secrets Manager。有关更多信息，请参阅 密钥管理。

查看详细的构建信息

1. 使用您的 Web 浏览器，转到上一步的输出中显示的 deepLink 位置（如 `https://console.aws.amazon.com/cloudwatch/home?region=region-ID#logEvent:group=/aws/codebuild/codebuild-demo-project;stream=38ca1c4a-e9ca-4dbc-bef1-d52bfEXAMPLE`）。
2. 在 CloudWatch 日志日志流中，您可以浏览日志事件。默认情况下，只显示最近的一组日志事件。要查看以前的日志事件，请滚动到列表开头。
3. 在本教程中，大多数日志事件包含的是关于 CodeBuild 下载构建相关文件并将其安装到构建环境中的详细信息，您可能并不关心这些信息。您可以使用筛选事件框来减少显示的信息。例如，如果您在筛选事件中输入 `"[INFO]"`，则仅显示那些包含 `[INFO]` 的事件。有关更多信息，请参阅亚马逊 CloudWatch 用户指南中的筛选条件和模式语法。

CloudWatch 日志日志流的这些部分与本教程有关。

```
...
[Container] 2016/04/15 17:49:42 Entering phase PRE_BUILD
[Container] 2016/04/15 17:49:42 Running command echo Entering pre_build phase...
[Container] 2016/04/15 17:49:42 Entering pre_build phase...
[Container] 2016/04/15 17:49:42 Phase complete: PRE_BUILD Success: true
[Container] 2016/04/15 17:49:42 Entering phase BUILD
[Container] 2016/04/15 17:49:42 Running command echo Entering build phase...
[Container] 2016/04/15 17:49:42 Entering build phase...
[Container] 2016/04/15 17:49:42 Running command mvn install
[Container] 2016/04/15 17:49:44 [INFO] Scanning for projects...
[Container] 2016/04/15 17:49:44 [INFO]
```

```
[Container] 2016/04/15 17:49:44 [INFO]
 ------------------------------------------------------------------------
[Container] 2016/04/15 17:49:44 [INFO] Building Message Utility Java Sample App 1.0
[Container] 2016/04/15 17:49:44 [INFO]
 ------------------------------------------------------------------------
...
[Container] 2016/04/15 17:49:55
 -------------------------------------------------------
[Container] 2016/04/15 17:49:55  T E S T S
[Container] 2016/04/15 17:49:55
 -------------------------------------------------------
[Container] 2016/04/15 17:49:55 Running TestMessageUtil
[Container] 2016/04/15 17:49:55 Inside testSalutationMessage()
[Container] 2016/04/15 17:49:55 Hi!Robert
[Container] 2016/04/15 17:49:55 Inside testPrintMessage()
[Container] 2016/04/15 17:49:55 Robert
[Container] 2016/04/15 17:49:55 Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time
 elapsed: 0.018 sec
[Container] 2016/04/15 17:49:55
[Container] 2016/04/15 17:49:55 Results :
[Container] 2016/04/15 17:49:55
[Container] 2016/04/15 17:49:55 Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
...
[Container] 2016/04/15 17:49:56 [INFO]
 ------------------------------------------------------------------------
[Container] 2016/04/15 17:49:56 [INFO] BUILD SUCCESS
[Container] 2016/04/15 17:49:56 [INFO]
 ------------------------------------------------------------------------
[Container] 2016/04/15 17:49:56 [INFO] Total time: 11.845 s
[Container] 2016/04/15 17:49:56 [INFO] Finished at: 2016-04-15T17:49:56+00:00
[Container] 2016/04/15 17:49:56 [INFO] Final Memory: 18M/216M
[Container] 2016/04/15 17:49:56 [INFO]
 ------------------------------------------------------------------------
[Container] 2016/04/15 17:49:56 Phase complete: BUILD Success: true
[Container] 2016/04/15 17:49:56 Entering phase POST_BUILD
[Container] 2016/04/15 17:49:56 Running command echo Entering post_build phase...
[Container] 2016/04/15 17:49:56 Entering post_build phase...
[Container] 2016/04/15 17:49:56 Phase complete: POST_BUILD Success: true
[Container] 2016/04/15 17:49:57 Preparing to copy artifacts
[Container] 2016/04/15 17:49:57 Assembling file list
[Container] 2016/04/15 17:49:57 Expanding target/messageUtil-1.0.jar
[Container] 2016/04/15 17:49:57 Found target/messageUtil-1.0.jar
[Container] 2016/04/15 17:49:57 Creating zip artifact
```

在此示例中，CodeBuild 成功完成了预构建、构建和生成后构建阶段。它运行单元测试并成功构建 `messageUtil-1.0.jar` 文件。

步骤 9：获取构建输出构件

（上一步：步骤 8：查看详细的构建信息）

在此步骤中，您将获得 CodeBuild 生成并上传到输出存储桶的`messageUtil-1.0.jar`文件。

您可以使用 CodeBuild 控制台或 Amazon S3 控制台来完成此步骤。

获取编译输出工件 (AWS CodeBuild 控制台）

1. 在 CodeBuild 控制台仍处于打开状态且仍显示上一步中的编译详细信息页面的情况下，选择编译详细信息选项卡并向下滚动到 Artifacts 部分。

   > **Note**
   >
   > 如果未显示构建详细信息页面，请在导航栏中选择构建历史记录，然后选择构建运行链接。

2. 指向 Amazon S3 文件夹的链接位于构件上传位置下方。该链接会打开 Amazon S3 文件夹，您可以在这里找到 `messageUtil-1.0.jar` 构建输出构件文件。

获取构建输出构件（Amazon S3 控制台）

1. 打开 Amazon S3 控制台，网址为 https://console.aws.amazon.com/s3/。
2. 打开 `codebuild-`*`region-ID`*`-`*`account-ID`*`-output-bucket`。
3. 打开 `codebuild-demo-project` 文件夹。
4. 打开 `target` 文件夹，您可以在此处找到 `messageUtil-1.0.jar` 构建输出构件文件。

步骤 10：删除 S3 存储桶

（上一步：步骤 9：获取构建输出构件）

为防止持续向您的 AWS 账户收费，您可以删除本教程中使用的输入和输出存储桶。有关更多信息，请参阅《Amazon Simple Storage Service 用户指南》中的删除或清空存储桶。

如果您使用 IAM 用户或管理员 IAM 用户删除这些存储桶，则该用户必须具有更多访问权限。在标记（*### BEGIN ADDING STATEMENT HERE ###*和*### END ADDING STATEMENTS HERE ###*）之间添加以下语句到用户的现有访问策略中。

此语句中的省略号 (...) 旨在力求简洁。请勿删除现有访问策略中的任何语句。请勿在策略中输入这些省略号。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:DeleteBucket",
        "s3:DeleteObject"
      ],
      "Resource": "*"
    }
  ]
}
```

总结

在本教程中，您曾经 AWS CodeBuild 将一组 Java 类文件构建到 JAR 文件中。然后查看了构建的结果。

您现在可以尝试 CodeBuild 在自己的场景中使用。按照计划构建中的说明进行操作。如果您觉得自己还没准备好，可以尝试构建一些示例。有关更多信息，请参阅 使用基于案例的示例 CodeBuild。

使用基于案例的示例 CodeBuild

您可以使用这些基于用例的示例进行 AWS CodeBuild实验：

Cross-service 样本

可供实验的跨服务示例列表。 AWS CodeBuild

构建徽章示例

演示如何 CodeBuild 使用构建徽章进行设置。

测试报告示例

AWS CLI 使用创建、运行和查看测试报告的结果。

的 Docker 示例 CodeBuild

演示如何使用自定义 Docker 映像、将 Docker 映像发布到 Amazon ECR 中的存储库，以及在私有注册表中使用 Docker 映像。

将构建输出托管在 S3 存储桶中

说明如何使用未加密的构建构件在 S3 存储桶中创建静态网站。

多个输入和输出示例

演示如何在构建项目中使用多个输入源和多个输出构件。

并行测试执行示例

演示如何使用 `codebuild-tests-run` CLI 命令跨并行执行环境拆分和运行测试。

buildspec 文件示例中的运行时版本

说明如何在 buildspec 文件中指定运行时及其版本。

源版本示例

演示如何在 CodeBuild 构建项目中使用源代码的特定版本。

Third-party 的源存储库示例 CodeBuild

演示如何使用 CodeBuild webhook 创建 BitBucket、 GitHub 企业服务器和 GitHub 拉取请求。

使用语义版本控制在构建时设置构件名称

说明如何使用语义版本控制在构建时创建构件名称。

Cross-service 的样本 CodeBuild

您可以使用这些跨服务示例进行 AWS CodeBuild试验：

Amazon ECR 示例

使用 Amazon ECR 存储库中的 Docker 映像，以使用 Apache Maven 生成单个 JAR 文件。示例说明将向您展示如何创建 Docker 镜像并将其推送到 Amazon ECR、创建 Go 项目、构建项目、运行项目以及设置允许连接 CodeBuild 到 Amazon ECR 的权限。

Amazon EFS 示例

演示如何配置构建规范文件，以便 CodeBuild 项目在 Amazon EFS 文件系统上挂载和构建。示例说明将向您展示如何创建 Amazon VPC、在 Amazon VPC 中创建文件系统、创建和构建使用 Amazon VPC 的项目，然后演示如何查看生成的项目文件和变量。

AWS CodePipeline 样本

演示 AWS CodePipeline 如何使用批量编译以及多个输入源和多个输出构件创建构建。本节中包括的示例 JSON 文件展示了使用单独构件和合并构件创建批量构建的管线结构。另外还提供了一个 JSON 示例，该示例显示了具有多个输入源和多个输出构件的管道结构。

AWS Config 样本

演示如何设置 AWS Config。列出跟踪哪些 CodeBuild 资源并描述如何在中查找 CodeBuild 项目 AWS Config。示例说明将向您展示与 AWS Config集成的先决条件 AWS Config、设置步骤以及在其中查找 CodeBuild 项目和数据的步骤 AWS Config。

构建通知示例

使用 Apache Maven 生成单个 JAR 文件。给 Amazon SNS 主题的订阅者发送构建通知。示例说明向您展示了如何设置权限以便与 Amazon SNS 通信 CloudWatch，以及如何在 Amazon SNS 中创建和识别 CodeBuild 主题、如何让收件人订阅该主题以及如何在中设置规则。 CodeBuild CloudWatch

的亚马逊 ECR 示例 CodeBuild

此示例使用 Amazon Elastic Container Registry（Amazon ECR）映像存储库中的 Docker 映像生成示例 Go 项目。

> **Important**
>
> 运行此示例可能会导致您的 AWS 账户被扣款。其中包括与 Amazon S3、 AWS KMS、CloudWatch 日志和 Amazon ECR 相关的 AWS 资源和操作可能产生的费用。 AWS CodeBuild 有关更多信息，请参阅CodeBuild 定价、Amazon S3 定价、AWS Key Management Service 定价、亚马逊 CloudWatch 定价和亚马逊弹性容器注册表定价。

主题

- 运行 Amazon ECR 示例

运行 Amazon ECR 示例

按照以下说明运行 Amazon ECR 示例。 CodeBuild

要运行此示例，请执行以下操作：

1. 要创建 Docker 映像并将其推送到 Amazon ECR 中的映像存储库，请完成 “将 Docker 映像发布到 Amazon ECR”示例 的 运行“将 Docker 映像发布到 Amazon ECR”示例 部分中的步骤。
2. 创建 Go 项目：

   a. 按照本主题Go 项目结构和Go 项目文件部分所述创建文件，然后将其上传到 S3 输入存储桶或 AWS CodeCommit GitHub、或 Bitbucket 存储库。

   > **Important**
   >
   > 请不要上传 *(root directory name)*，而只上传 *(root directory name)* 中的文件。
   > 如果您使用的是 S3 输入存储桶，请务必创建一个包含这些文件的 ZIP 文件，然后将其上传到输入存储桶。请不要将 *(root directory name)* 添加到 ZIP 文件中，而只添加 *(root directory name)* 中的文件。

   b. 创建构建项目、运行构建和查看相关构建信息。

   如果您使用 AWS CLI 来创建构建项目，则`create-project`命令的 JSON-formatted输入可能与此类似。(请将占位符替换为您自己的值。)

   ```
   {
   ```

```
  "name": "sample-go-project",
  "source": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-bucket/GoSample.zip"
  },
  "artifacts": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-output-bucket",
    "packaging": "ZIP",
    "name": "GoOutputArtifact.zip"
  },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "aws/codebuild/standard:5.0",
    "computeType": "BUILD_GENERAL1_SMALL"
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name",
  "encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

c. 要获取构建输出构件，请打开您的 S3 输出存储桶。

d. 将 *GoOutputArtifact*.zip 文件下载到您的本地计算机或实例，然后提取该文件的内容。在提取出来的内容中，获取 hello 文件。

3. 如果满足以下条件之一，则必须在 Amazon ECR 中为镜像存储库添加权限，这样 AWS CodeBuild 才能将其 Docker 映像拉入构建环境。

   - 您的项目使用 CodeBuild 凭证来提取 Amazon ECR 映像。这是由 ProjectEnvironment 的 imagePullCredentialsType 属性中的 CODEBUILD 值指示的。
   - 您的项目使用了跨账户 Amazon ECR 映像。在这种情况下，您的项目必须使用其服务角色拉取 Amazon ECR 映像。要启用此行为，请将您的 ProjectEnvironment 的 imagePullCredentialsType 属性设置为 SERVICE_ROLE。

   1. 打开 Amazon ECR 控制台，网址为https://console.aws.amazon.com/ecr/。
   2. 在存储库名称列表中，选择您创建或选择的存储库的名称。
   3. 在导航窗格中，依次选择权限、编辑和添加语句。
   4. 对于声明名称，输入标识符（例如 **CodeBuildAccess**）。
   5. 对于效果，选择允许。这表示您希望允许访问另一个 AWS 账户。
   6. 对于主体，执行以下操作之一：

- 如果您的项目使用 CodeBuild 凭证提取 Amazon ECR 映像，请在服务主体中输入**codebuild.amazonaws.com**。
- 如果您的项目使用了跨账户 Amazon ECR 映像，请在 AWS 账户 ID 中输入您要为其授予访问权限的 AWS 账户的 ID。

7. 跳过所有 IAM 实体列表。
8. 在 “操作” 中，选择仅限拉取的操作：ecr: GetDownloadUrlForLayer、ecr: 和 ecr: BatchGetImage。BatchCheckLayerAvailability
9. 对于条件，请添加以下内容：

```
{
   "StringEquals":{
      "aws:SourceAccount":"<AWS-account-ID>",
      "aws:SourceArn":"arn:aws:codebuild:<region>:<AWS-account-
ID>:project/<project-name>"
   }
}
```

10选择保存。

此策略显示在权限中。主体是您在此过程的步骤 3 中为主体输入的内容：

- 如果您的项目使用 CodeBuild 凭证提取 Amazon ECR 映像，则"codebuild.amazonaws.com"会显示在 “服务主体” 下。
- 如果您的项目使用跨账户 Amazon ECR 图片，则您要授予访问权限的 AWS 账户 ID 会显示在AWS 账户 ID 下。

以下示例策略同时使用 CodeBuild 凭证和跨账户 Amazon ECR 映像。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CodeBuildAccessPrincipal",
            "Effect": "Allow",
            "Action": [
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "ecr:BatchCheckLayerAvailability"
            ],
```

```
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceArn": "arn:aws:codebuild:us-
east-1:111122223333:project/MyProject",
                    "aws:SourceAccount": "111122223333"
                }
            }
        },
        {
            "Sid": "CodeBuildAccessCrossAccount",
            "Effect": "Allow",
            "Action": [
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "ecr:BatchCheckLayerAvailability"
            ],
            "Resource": "*"
        }
    ]
}
```

- 如果您的项目使用 CodeBuild 证书，并且您希望您的 CodeBuild 项目拥有对 Amazon ECR 存储库的开放访问权限，则可以省略`Condition`密钥并添加以下示例策略。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CodeBuildAccessPrincipal",
            "Effect": "Allow",
            "Resource": [
                "arn:aws:codecommit:us-
east-2:111122223333:MySharedDemoRepo"
            ],
            "Action": [
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "ecr:BatchCheckLayerAvailability"
            ]
        },
```

```
        {
            "Sid": "CodeBuildAccessCrossAccount",
            "Effect": "Allow",
            "Resource": [
                "arn:aws:codecommit:us-
east-2:111122223333:MySharedDemoRepo"
            ],
            "Action": [
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "ecr:BatchCheckLayerAvailability"
            ]
        }
    ]
}
```

4. 创建构建项目、运行构建和查看构建信息。

   如果您使用 AWS CLI 来创建构建项目，则`create-project`命令的 JSON-formatted 输入可能与此类似。(请将占位符替换为您自己的值。)

```
{
  "name": "amazon-ecr-sample-project",
  "source": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-bucket/GoSample.zip"
  },
  "artifacts": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-output-bucket",
    "packaging": "ZIP",
    "name": "GoOutputArtifact.zip"
  },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "account-ID.dkr.ecr.region-ID.amazonaws.com/your-Amazon-ECR-repo-
name:tag",
    "computeType": "BUILD_GENERAL1_SMALL"
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name",
  "encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

5. 要获取构建输出构件，请打开您的 S3 输出存储桶。
6. 将 *GoOutputArtifact*.zip 文件下载到您的本地计算机或实例，然后提取 *GoOutputArtifact*.zip 文件的内容。在提取出来的内容中，获取 hello 文件。

Go 项目结构

此示例采用以下目录结构。

```
(root directory name)
### buildspec.yml
### hello.go
```

Go 项目文件

此示例将使用这些文件。

buildspec.yml（在 *(root directory name)*）

```
version: 0.2

phases:
  install:
   runtime-versions:
     golang: 1.13
  build:
    commands:
      - echo Build started on `date`
      - echo Compiling the Go code
      - go build hello.go
  post_build:
    commands:
      - echo Build completed on `date`
artifacts:
  files:
    - hello
```

hello.go（在 *(root directory name)*）

```
package main
import "fmt"
```

```
func main() {
  fmt.Println("hello world")
  fmt.Println("1+1 =", 1+1)
  fmt.Println("7.0/3.0 =", 7.0/3.0)
  fmt.Println(true && false)
  fmt.Println(true || false)
  fmt.Println(!true)
}
```

的亚马逊弹性文件系统示例 AWS CodeBuild

您可能需要在 Amazon Elastic File System 上创建自己的 AWS CodeBuild 版本，这是一种适用于 Amazon EC2 实例的可扩展共享文件服务。Amazon EFS 中的存储容量是弹性的，因此会随着文件的添加和删除而增长或收缩。它具有简单的 Web 服务界面，可用于创建和配置文件系统。它还为您管理所有文件存储基础设施，因此您无需担心部署、修补或维护文件系统配置。有关更多信息，请参阅 Amazon Elastic File System 用户指南中的什么是 Amazon Elastic File System？。

此示例向您展示如何配置 CodeBuild 项目，使其挂载并构建 Java 应用程序到 Amazon EFS 文件系统。在开始之前，必须准备好要构建的 Java 应用程序，该应用程序已上传到 S3 输入存储桶或 AWS CodeCommit、 GitHub、 GitHub 企业服务器或 Bitbucket 存储库。

系统会加密文件系统的传输中数据。要使用其他映像来加密传输中的数据，请参阅加密传输中的数据。

主题

- 使用 AWS CodeBuild 使用亚马逊弹性文件系统
- 排查 Amazon EFS 集成问题

使用 AWS CodeBuild 使用亚马逊弹性文件系统

该示例涵盖了使用 Amazon EFS 所需的四个高级步骤 AWS CodeBuild。它们是：

1. 在您的 AWS 账户中创建虚拟私有云 (VPC)。
2. 创建使用此 VPC 的文件系统。
3. 创建和构建使用 VPC 的 CodeBuild 项目。该 CodeBuild 项目使用以下方法来识别文件系统：
   - 文件系统的唯一标识符。当您在构建项目中指定文件系统时，可以选择该标识符。
   - 文件系统 ID。当您在 Amazon EFS 控制台中查看文件系统时，系统会显示该 ID。
   - 挂载点。这是 Docker 容器中用于挂载文件系统的目录。

- 挂载选项。这些选项包含了有关如何挂载文件系统的详细信息。

4. 查看构建项目，确保生成了正确的项目文件和变量。

> **Note**
>
> 只有 Linux 平台支持在 Amazon EFS 中创建的文件系统。

主题

- 步骤 1：使用创建 VPC CloudFormation
- 步骤 2：使用 VPC 创建 Amazon Elastic File System 文件系统
- 第 3 步：创建与 Amazon EFS 一起使用的 CodeBuild 项目
- 步骤 4：检查构建项目

步骤 1：使用创建 VPC CloudFormation

使用 CloudFormation 模板创建您的 VPC。

1. 按照中的说明CloudFormation VPC 模板使用 CloudFormation 创建 VPC。

   > **Note**
   >
   > 此 CloudFormation 模板创建的 VPC 有两个私有子网和两个公有子网。当您使用 AWS CodeBuild 挂载在 Amazon EFS 中创建的文件系统时，只能使用私有子网。如果您使用其中一个公有子网，则构建会失败。

2. 登录 AWS 管理控制台 并打开亚马逊 VPC 控制台，网址为https://console.aws.amazon.com/vpc/。
3. 选择您创建时使用的 VPC CloudFormation。
4. 在描述选项卡上，记下 VPC 的名称及其 ID。在本示例的后面部分中创建您的 AWS CodeBuild 项目时，需要这两者。

步骤 2：使用 VPC 创建 Amazon Elastic File System 文件系统

使用您之前创建的 VPC 为本示例创建简单的 Amazon EFS 文件系统。

1. 登录 AWS 管理控制台 并打开亚马逊 EFS 控制台，网址为https://console.aws.amazon.com/efs/。
2. 选择创建文件系统。
3. 从 VPC，选择您之前在本示例中记录的 VPC 名称。
4. 保持可用区与您选定子网的关联。
5. 选择下一步。
6. 在添加标签中，对于默认的名称键，在值中，输入 Amazon EFS 文件系统的名称。
7. 保留突增和通用型选定为您的默认性能和吞吐量模式，然后选择下一步。
8. 对于配置客户端访问，请选择下一步。
9. 选择创建文件系统。
10. （可选）我们建议您为您的 Amazon EFS 文件系统添加策略，以强制对传输中的数据进行加密。在 Amazon EFS 控制台中，选择文件系统策略，选择编辑，选中标有针对所有客户端强制执行传输中加密的复选框，然后选择保存。

第 3 步：创建与 Amazon EFS 一起使用的 CodeBuild 项目

使用您之前在本示例中创建的 VPC 创建一个 AWS CodeBuild 项目。运行构建时，它会挂载之前创建的 Amazon EFS 文件系统。接下来，它会将 Java 应用程序创建的 .jar 文件存储在文件系统的挂载点目录中。

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 从导航窗格中选择构建项目，然后选择创建构建项目。
3. 在项目名称中输入项目名称。
4. 从源提供商中，选择包含要构建的 Java 应用程序的存储库。
5. 输入 CodeBuild 用于查找应用程序的信息，例如存储库 URL。每个源提供商的选项有所不同。有关更多信息，请参阅 Choose source provider。
6. 从环境映像中，选择托管映像。
7. 从操作系统中，选择 Amazon Linux 2。
8. 从运行时中，选择标准。
9. 从 “图片” 中选择 aws/codebuild /amazon l inux-x86_64-standard: 4.0。
10. 从环境类型中，选择 Linux。
11. 在服务角色下，选择新建服务角色。在角色名称中，输入为您 CodeBuild 创建的角色的名称。
12. 展开其他配置。

13. 选择如果要构建 Docker 映像或希望您的构建获得提升的特权，请启用此标志。

> **Note**
>
> 默认情况下，为非 VPC 构建启用 Docker 进程守护程序。如果您想使用 Docker 容器进行 VPC 构建，请参阅 Docker 文档网站上的运行时权限和 Linux 功能并启用特权模式。此外，Windows 不支持特权模式。

14. 从 VPC 中，选择 VPC ID。
15. 从子网中，选择一个或多个与您的 VPC 关联的私有子网。您必须在挂载 Amazon EFS 文件系统的构建项目中使用私有子网。如果您使用公有子网，则构建会失败。
16. 从安全组中，选择默认安全组。
17. 在文件系统中，输入以下信息：
    - 针对标识符，输入文件系统的唯一标识符。该标识符必须少于 129 个字符，并且只能包含字母数字字符和下划线。 CodeBuild 使用此标识符来创建用于标识弹性文件系统的环境变量。该环境变量的格式为采用大写字母的 `CODEBUILD_<file_system_identifier>`。例如，如果输入 `my_efs`，则环境变量为 `CODEBUILD_MY_EFS`。
    - 对于 ID，请选择文件系统 ID。
    - （可选）在文件系统中输入一个目录。 CodeBuild 挂载此目录。如果将 Directory path (目录路径) 留为空白，则 CodeBuild 会挂载整个文件系统。该路径相对于文件系统的根目录指定。
    - 对于挂载点，输入构建容器中用于挂载文件系统的目录的绝对路径。如果此目录不存在，则在构建期间 CodeBuild 创建它。
    - （可选）输入挂载选项。如果将装载选项留空，则 CodeBuild 使用其默认装载选项：

      ```
      nfsvers=4.1
      rsize=1048576
      wsize=1048576
      hard
      timeo=600
      retrans=2
      ```

      有关更多信息，请参阅《Amazon Elastic File System 用户指南》中的建议的 NFS 挂载选项。
18. 对于构建规范，选择插入构建命令，然后选择切换到编辑器。
19. 在编辑器中输入以下构建规范命令。将 *<file_system_identifier>* 替换为您在步骤 17 中输入的标识符。使用大写字母（例如 `CODEBUILD_MY_EFS`）。

```
version: 0.2
phases:
  install:
    runtime-versions:
      java: corretto11
  build:
    commands:
      - mvn compile -Dgpg.skip=true -Dmaven.repo.local=
$CODEBUILD_<file_system_identifier>
```

20. 对所有其他设置使用默认值，然后选择创建构建项目。构建完成后，系统会显示项目的控制台页面。
21. 选择开始构建。

步骤 4：检查构建项目

AWS CodeBuild 项目构建完成后：

- 您会拥有一个由 Java 应用程序创建的 .jar 文件，该文件已被构建到您的挂载点目录下的 Amazon EFS 文件系统中。
- 系统会使用您在创建项目时输入的文件系统标识符，创建标识文件系统的环境变量。

有关更多信息，请参阅《Amazon Elastic File System 用户指南》中的装载文件系统。

排查 Amazon EFS 集成问题

以下是您在设置 Amazon EFS 时可能遇到的错误 CodeBuild。

主题

- CLIENT_ERROR：挂载 127.0.0.1:/ 失败。权限被拒绝
- CLIENT_ERROR：挂载 127.0.0.1:/ 失败。连接被对等方重置
- VPC_CLIENT_ERROR：意外的 EC2 错误： UnauthorizedOperation

CLIENT_ERROR：挂载 127.0.0.1:/ 失败。权限被拒绝

使用安装亚马逊 EFS 不支持 IAM 授权 CodeBuild。如果您使用的是自定义 Amazon EFS 文件系统策略，则需要向所有 IAM 主体授予读写权限。例如：

```
"Principal": {
  "AWS": "*"
}
```

CLIENT_ERROR：挂载 127.0.0.1:/ 失败。连接被对等方重置

有两种可能的原因会导致此错误：

- CodeBuild VPC 子网与 Amazon EFS 挂载目标位于不同的可用区域中。您可以通过在 Amazon EFS 挂载目标所在的同一可用区中添加 VPC 子网来解决此问题。
- 安全组不具备与 Amazon EFS 通信的权限。您可以通过添加入站规则来允许来自 VPC（为您的 VPC 添加主要 CIDR 块）或安全组本身的所有流量，从而解决此问题。

VPC_CLIENT_ERROR：意外的 EC2 错误： UnauthorizedOperation

当您的 CodeBuild项目 VPC 配置中的所有子网均为公有子网时，就会出现此错误。您在 VPC 中必须至少有一个私有子网才能确保网络连接正常。

AWS CodePipeline 的样本 CodeBuild

本节介绍 CodePipeline 和 CodeBuild之间的集成示例。

样本	说明
CodePipeline/CodeBuild 集成和批量构建示例	这些示例演示了 AWS CodePipeline 如何使用批量生成创建构建项目。
包含多个输入源和输出工件的 CodePipeline/CodeBuild 集成示例	此示例演示 AWS CodePipeline 如何使用创建使用多个输入源创建多个输出构件的构建项目。

CodePipeline/CodeBuild 集成和批量构建示例

AWS CodeBuild 支持批量构建。以下示例演示 AWS CodePipeline 如何使用创建使用批量生成的构建项目。

您可以使用定义管道结构的 JSON-formatted 文件，然后将其与一起使用 AWS CLI 来创建管道。有关更多信息，请参阅《AWS CodePipeline 用户指南》中的 AWS CodePipeline 管道结构参考。

使用单个构件进行批量构建

使用以下 JSON 文件作为管道结构的示例，该结构使用单个构件创建批量构建。要启用批量构建 CodePipeline，请将`configuration`对象的`BatchEnabled`参数设置为`true`。

```
{
  "pipeline": {
    "roleArn": "arn:aws:iam::account-id:role/my-AWS-CodePipeline-service-role-name",
    "stages": [
      {
        "name": "Source",
        "actions": [
          {
            "inputArtifacts": [],
            "name": "Source1",
            "actionTypeId": {
              "category": "Source",
              "owner": "AWS",
              "version": "1",
              "provider": "S3"
            },
            "outputArtifacts": [
              {
                "name": "source1"
              }
            ],
            "configuration": {
              "S3Bucket": "<my-input-bucket-name>",
              "S3ObjectKey": "my-source-code-file-name.zip"
            },
            "runOrder": 1
          },
          {
            "inputArtifacts": [],
            "name": "Source2",
            "actionTypeId": {
              "category": "Source",
              "owner": "AWS",
              "version": "1",
              "provider": "S3"
            },
            "outputArtifacts": [
              {
                "name": "source2"
```

```
            }
          ],
          "configuration": {
            "S3Bucket": "<my-other-input-bucket-name>",
            "S3ObjectKey": "my-other-source-code-file-name.zip"
          },
          "runOrder": 1
        }
      ]
    },
    {
      "name": "Build",
      "actions": [
        {
          "inputArtifacts": [
            {
              "name": "source1"
            },
            {
              "name": "source2"
            }
          ],
          "name": "Build",
          "actionTypeId": {
            "category": "Build",
            "owner": "AWS",
            "version": "1",
            "provider": "CodeBuild"
          },
          "outputArtifacts": [
            {
              "name": "build1"
            },
            {
              "name": "build1_artifact1"
            },
            {
              "name": "build1_artifact2"
            },
            {
              "name": "build2_artifact1"
            },
            {
              "name": "build2_artifact2"
```

```
              }
            ],
            "configuration": {
              "ProjectName": "my-build-project-name",
              "PrimarySource": "source1",
              "BatchEnabled": "true"
            },
            "runOrder": 1
          }
        ]
      }
    ],
    "artifactStore": {
      "type": "S3",
      "location": "<AWS-CodePipeline-internal-bucket-name>"
    },
    "name": "my-pipeline-name",
    "version": 1
  }
}
```

以下是适用于此管道配置的 CodeBuild buildspec 文件示例。

```
version: 0.2
batch:
  build-list:
    - identifier: build1
      env:
        compute-type: BUILD_GENERAL1_SMALL
    - identifier: build2
      env:
        compute-type: BUILD_GENERAL1_MEDIUM

phases:
  build:
    commands:
      - echo 'file' > output_file

artifacts:
  files:
    - output_file
  secondary-artifacts:
    artifact1:
```

```
      files:
        - output_file
    artifact2:
      files:
        - output_file
```

管道的 JSON 文件中指定的输出构件的名称必须与 buildspec 文件中定义的构建和构件的标识符相匹配。语法*buildIdentifier*用于主要工件，*buildIdentifier_ artifactIdentifier* 用于次要构件。

例如，对于输出对象名称build1， CodeBuild 会将的主要对象上传build1到的位置build1。对于输出名称build1_artifact1， CodeBuild 会将artifact1的辅助对象上传build1到的位置build1_artifact1，依此类推。如果仅指定了一个输出位置，则名称应*buildIdentifier*仅为。

创建 JSON 文件后，可以创建管道。使用运行创建管道命令并将文件传递给参数。 AWS CLI `--cli-input-json`有关更多信息，请参阅《AWS CodePipeline 用户指南》中的创建管道（CLI）。

使用合并的构件进行批量构建

使用以下 JSON 文件作为管道结构的示例，该结构使用合并的构件创建批量构建。要启用批量构建 CodePipeline，请将configuration对象的BatchEnabled参数设置为true。要将构建构件合并到同一位置，请将 configuration 对象的 CombineArtifacts 参数设置为 true。

```
{
 "pipeline": {
  "roleArn": "arn:aws:iam::account-id:role/my-AWS-CodePipeline-service-role-name",
  "stages": [
    {
      "name": "Source",
      "actions": [
        {
          "inputArtifacts": [],
          "name": "Source1",
          "actionTypeId": {
            "category": "Source",
            "owner": "AWS",
            "version": "1",
            "provider": "S3"
          },
          "outputArtifacts": [
            {
```

```
            "name": "source1"
          }
        ],
        "configuration": {
          "S3Bucket": "<my-input-bucket-name>",
          "S3ObjectKey": "my-source-code-file-name.zip"
        },
        "runOrder": 1
      },
      {
        "inputArtifacts": [],
        "name": "Source2",
        "actionTypeId": {
          "category": "Source",
          "owner": "AWS",
          "version": "1",
          "provider": "S3"
        },
        "outputArtifacts": [
          {
            "name": "source2"
          }
        ],
        "configuration": {
          "S3Bucket": "<my-other-input-bucket-name>",
          "S3ObjectKey": "my-other-source-code-file-name.zip"
        },
        "runOrder": 1
      }
    ]
  },
  {
    "name": "Build",
    "actions": [
      {
        "inputArtifacts": [
          {
            "name": "source1"
          },
          {
            "name": "source2"
          }
        ],
        "name": "Build",
```

```
          "actionTypeId": {
            "category": "Build",
            "owner": "AWS",
            "version": "1",
            "provider": "CodeBuild"
          },
          "outputArtifacts": [
            {
              "name": "output1 "
            }
          ],
          "configuration": {
            "ProjectName": "my-build-project-name",
            "PrimarySource": "source1",
             "BatchEnabled": "true",
             "CombineArtifacts": "true"
          },
          "runOrder": 1
        }
      ]
    }
  ],
  "artifactStore": {
    "type": "S3",
    "location": "<AWS-CodePipeline-internal-bucket-name>"
  },
  "name": "my-pipeline-name",
  "version": 1
 }
}
```

以下是适用于此管道配置的 CodeBuild buildspec 文件示例。

```
version: 0.2
batch:
  build-list:
    - identifier: build1
      env:
        compute-type: BUILD_GENERAL1_SMALL
    - identifier: build2
      env:
        compute-type: BUILD_GENERAL1_MEDIUM
```

```
phases:
  build:
    commands:
      - echo 'file' > output_file

artifacts:
  files:
    - output_file
```

如果为批量生成启用了组合构件，则只允许输出一个内容。 CodeBuild 会将所有版本的主要构件合并到一个 ZIP 文件中。

创建 JSON 文件后，可以创建管道。使用运行创建管道命令并将文件传递给参数。 AWS CLI `--cli-input-json`有关更多信息，请参阅《AWS CodePipeline 用户指南》中的创建管道（CLI）。

包含多个输入源和输出工件的 CodePipeline/CodeBuild 集成示例

一个 AWS CodeBuild 项目可以采用多个输入源。也可以创建多个输出构件。此示例演示 AWS CodePipeline 如何使用创建使用多个输入源创建多个输出构件的构建项目。有关更多信息，请参阅 多输入源和输出构件示例。

您可以使用定义管道结构的 JSON-formatted 文件，然后将其与一起使用 AWS CLI 来创建管道。使用以下 JSON 文件作为管道结构的示例，此管道结构可以创建一个具有多输入源和多输出构件的构建。稍后，此示例会介绍该文件如何指定多个输入和输出。有关更多信息，请参阅《AWS CodePipeline 用户指南》中的CodePipeline 管道结构参考。

```
{
 "pipeline": {
  "roleArn": "arn:aws:iam::account-id:role/my-AWS-CodePipeline-service-role-name",
  "stages": [
    {
      "name": "Source",
      "actions": [
        {
          "inputArtifacts": [],
          "name": "Source1",
          "actionTypeId": {
            "category": "Source",
            "owner": "AWS",
            "version": "1",
            "provider": "S3"
          },
```

```
        "outputArtifacts": [
          {
            "name": "source1"
          }
        ],
        "configuration": {
          "S3Bucket": "my-input-bucket-name",
          "S3ObjectKey": "my-source-code-file-name.zip"
        },
        "runOrder": 1
      },
      {
        "inputArtifacts": [],
        "name": "Source2",
        "actionTypeId": {
          "category": "Source",
          "owner": "AWS",
          "version": "1",
          "provider": "S3"
        },
        "outputArtifacts": [
          {
            "name": "source2"
          }
        ],
        "configuration": {
          "S3Bucket": "my-other-input-bucket-name",
          "S3ObjectKey": "my-other-source-code-file-name.zip"
        },
        "runOrder": 1
      }
    ]
  },
  {
    "name": "Build",
    "actions": [
      {
        "inputArtifacts": [
          {
            "name": "source1"
          },
          {
            "name": "source2"
          }
```

```
          ],
          "name": "Build",
          "actionTypeId": {
            "category": "Build",
            "owner": "AWS",
            "version": "1",
            "provider": "AWS CodeBuild"
          },
          "outputArtifacts": [
            {
              "name": "artifact1"
            },
            {
              "name": "artifact2"
            }
          ],
          "configuration": {
            "ProjectName": "my-build-project-name",
            "PrimarySource": "source1"
          },
          "runOrder": 1
        }
      ]
    }
  ],
  "artifactStore": {
    "type": "S3",
    "location": "AWS-CodePipeline-internal-bucket-name"
  },
  "name": "my-pipeline-name",
  "version": 1
 }
}
```

在此 JSON 文件中：

- 必须将输入源之一指定为 PrimarySource。此源代码是 CodeBuild 查找和运行 buildspec 文件的目录。PrimarySource该关键字用于指定 JSON 文件中 CodeBuild 阶段configuration部分的主要来源。
- 每个输入源都安装在各自的目录中。此目录存储在内置环境变量 $CODEBUILD_SRC_DIR（对于主要源）和 $CODEBUILD_SRC_DIR_yourInputArtifactName（对于所有其他源）中。对于此示

例中的管道，两个输入源目录为 $CODEBUILD_SRC_DIR 和 $CODEBUILD_SRC_DIR_source2。有关更多信息，请参阅 构建环境中的环境变量。

- 管道的 JSON 文件中指定的输出构件的名称必须与 buildspec 文件中定义的辅助构件的名称相匹配。此管道使用以下 buildspec 文件。有关更多信息，请参阅 buildspec 语法。

```
version: 0.2

phases:
  build:
    commands:
      - touch source1_file
      - cd $CODEBUILD_SRC_DIR_source2
      - touch source2_file

artifacts:
  files:
    - '**/*'
  secondary-artifacts:
    artifact1:
      base-directory: $CODEBUILD_SRC_DIR
      files:
        - source1_file
    artifact2:
      base-directory: $CODEBUILD_SRC_DIR_source2
      files:
        - source2_file
```

创建 JSON 文件后，可以创建管道。使用运行创建管道命令并将文件传递给参数。 AWS CLI `--cli-input-json`有关更多信息，请参阅《AWS CodePipeline 用户指南》中的创建管道（CLI）。

AWS Config 样品与 CodeBuild

AWS Config 提供了您的 AWS 资源清单以及这些资源的配置更改历史记录。 AWS Config 现在支持 AWS CodeBuild 作为 AWS 资源，这意味着该服务可以跟踪您的 CodeBuild 项目。有关的更多信息 AWS Config，请参阅什么是 AWS Config？ 在《AWS Config 开发人员指南》中。

您可以在 AWS Config 控制台的 “ CodeBuild 资源清单” 页面上看到以下有关资源的信息：

- 您的 CodeBuild 配置更改的时间表。
- 每个 CodeBuild 项目的配置详细信息。

- 与其他 AWS 资源的关系。
- CodeBuild 项目变更清单。

主题

- CodeBuild 搭配使用 AWS Config
- 步骤 3：在 AWS Config 控制台中查看 AWS CodeBuild 数据

CodeBuild 搭配使用 AWS Config

本主题中的过程向您展示如何设置 AWS Config 和查找 CodeBuild项目。

主题

- 先决条件
- 步骤 1：设置 AWS Config
- 第 2 步：查找 AWS CodeBuild 项目

先决条件

创建您的 AWS CodeBuild 项目。有关说明，请参阅创建构建项目。

步骤 1：设置 AWS Config

- 设置 AWS Config（控制台）
- 设置 AWS Config (AWS CLI)

> **Note**
>
> 完成设置后，最多可能需要 10 分钟才能在 AWS Config 控制台中看到 AWS CodeBuild 项目。

第 2 步：查找 AWS CodeBuild 项目

1. 登录 AWS 管理控制台并在上打开 AWS Config 控制台https://console.aws.amazon.com/config。
2. 在资源清单页面上，在资源类型下选择AWS CodeBuild 项目。向下滚动并选中CodeBuild项目复选框。
3. 选择查找。

4. 添加 CodeBuild 项目列表后，在 Config 时间轴列中选择 CodeBuild 项目名称链接。

步骤 3：在 AWS Config 控制台中查看 AWS CodeBuild 数据

在资源清单页面上查找资源时，您可以选择 AWS Config 时间表来查看有关您的 CodeBuild 项目的详细信息。资源的详细信息页面提供了有关该资源的配置、关系和更改次数的信息。

页面顶部的块统称为时间线。时间线显示了记录的创建日期和时间。

有关更多信息，请参阅《AWS Config 开发人员指南》中的在 AWS Config 控制台中查看配置详细信息。

为生成通知示例 CodeBuild

亚马逊 CloudWatch 活动内置了对以下内容的支持 AWS CodeBuild。 CloudWatch 事件是描述 AWS 资源变化的系统事件流。使用 CloudWatch 事件，您可以编写声明性规则，将感兴趣的事件与要采取的自动操作关联起来。此示例使用亚马逊 CloudWatch 事件和亚马逊简单通知服务 (Amazon SNS) 在构建成功、失败、从一个构建阶段进入另一个构建阶段或这些事件的任意组合时向订阅者发送构建通知。

> **Important**
>
> 运行此示例可能会向您的 AWS 账户收费。其中包括与亚马逊 CodeBuild 和亚马逊 SNS 相关的 AWS 资源 CloudWatch 和操作可能产生的费用。有关更多信息，请参阅CodeBuild 定价、亚马逊定 CloudWatch价和亚马逊 SNS 定价。

主题

- 运行构建通知示例
- 构建通知输入格式参考

运行构建通知示例

按照以下过程运行构建通知示例。

要运行此示例，请执行以下操作：

1. 如果您已在 Amazon SNS 中设置并订阅用于此示例的主题，请跳至第 4 步。否则，如果您使用 IAM 用户而不是 AWS 根账户或管理员用户来使用 Amazon SNS，请在用户（或用户关联的 IAM 群组*### END ADDING STATEMENT HERE ###*）中添加以下语句（介于*### BEGIN ADDING*

*STATEMENT HERE ###*和之间）。不建议使用 AWS 根账户。此语句可用于查看、创建、订阅和测试向 Amazon SNS 中的主题发送通知的情况。为了简洁起见，也为了帮您查找添加语句的位置，此处使用了省略号 (`...`)。请勿删除任何语句，也不要将这些省略号键入现有策略中。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sns:CreateTopic",
                "sns:GetTopicAttributes",
                "sns:List*",
                "sns:Publish",
                "sns:SetTopicAttributes",
                "sns:Subscribe"
            ],
            "Resource": "*"
        }
    ]
}
```

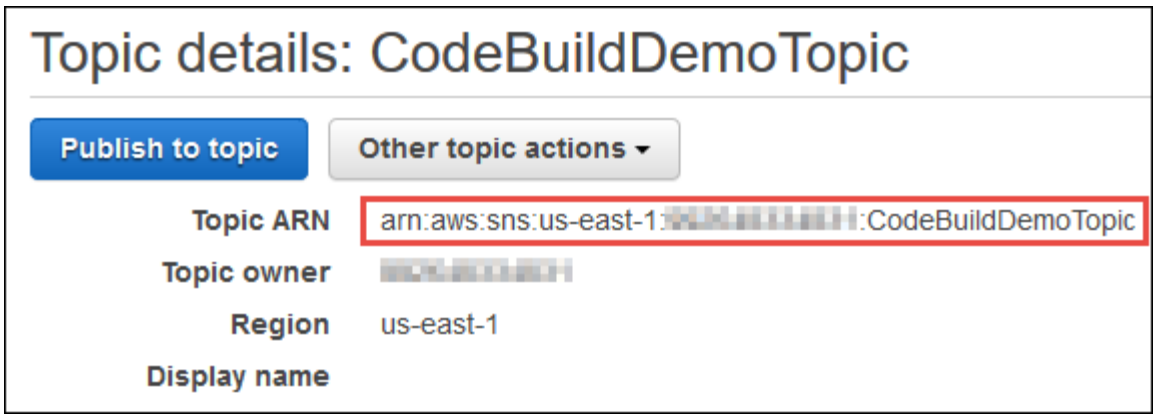
> **Note**
>
> 修改该策略的 IAM 实体必须拥有在 IAM 中修改策略的权限。
> 有关更多信息，请参阅《IAM 用户指南》中的编辑客户管理型策略或使用内联策略（控制台）中的“编辑或删除组、用户或角色的内联策略”部分。

2. 在 Amazon SNS 中创建或确定一个主题。 AWS CodeBuild 使用 CloudWatch 事件通过 Amazon SNS 向该主题发送构建通知。

   要创建主题，请执行以下操作:

   1. 从 https://console.aws.amazon.com/sns 打开 Amazon SNS 控制台。
   2. 选择创建主题。
   3. 在创建新主题对话框中，为主题名称输入主题的名称（例如 **`CodeBuildDemoTopic`**）。（如果您选择了其他名称，请用该名称替换掉本示例中对应的名称。）

4. 选择创建主题。
5. 在 “主题详细信息: CodeBuildDemoTopic” 页面上，复制主题 ARN 值。在下一个步骤中，您需要用到此值。

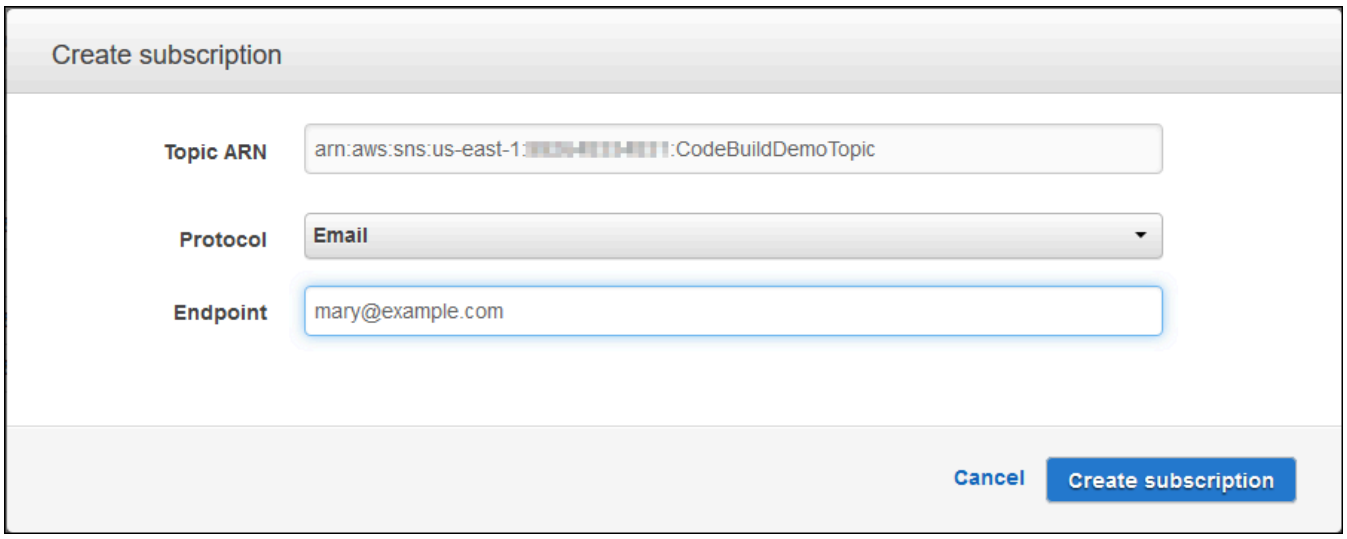


有关更多信息，请参阅《Amazon SNS 开发人员指南》中的创建主题。

3. 为一个或多个收件人订阅主题以接收电子邮件通知。

为收件人订阅主题：

1. 使用上一步中打开的 Amazon SNS 控制台，在导航窗格中，选择订阅，然后选择创建订阅。
2. 在创建订阅中，对于主题 ARN，粘贴您在上一步中复制的主题 ARN。
3. 对于协议，选择电子邮件。
4. 对于端点，输入收件人的完整电子邮件地址。



5. 选择创建订阅。
6. Amazon SNS 向收件人发送订阅确认电子邮件。要开始接收电子邮件通知，收件人必须在订阅确认电子邮件中选择确认订阅链接。在收件人单击该链接后，如果成功订阅，Amazon SNS 将在收件人的 Web 浏览器中显示一条确认消息。

有关更多信息，请参阅《Amazon SNS 开发人员指南》中的订阅主题。

4. 如果您使用用户而不是 AWS 根账户或管理员用户来处理 CloudWatch 事件，请向该用户（或用户关联的 IAM 群组*### END ADDING STATEMENT HERE ###*）添加以下语句（介于*### BEGIN ADDING STATEMENT HERE ###*和之间）。不建议使用 AWS 根账户。此语句用于允许用户处理 CloudWatch 事件。为了简洁起见，也为了帮您查找添加语句的位置，此处使用了省略号 (`...`)。请勿删除任何语句，也不要将这些省略号键入现有策略中。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "events:*",
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::*:role/Service*"
        }
    ]
}
```

> **Note**
>
> 修改该策略的 IAM 实体必须拥有在 IAM 中修改策略的权限。
> 有关更多信息，请参阅《IAM 用户指南》中的编辑客户管理型策略或使用内联策略（控制台）中的“编辑或删除组、用户或角色的内联策略”部分。

5. 在 “ CloudWatch 事件” 中创建规则。为此，请打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch。

6. 在导航窗格中的事件下，选择规则，然后选择创建规则。
7. 在步骤 1：创建规则页面上，事件模式和构建事件模式以按服务匹配事件应已选中。
8. 对于服务名称，选择 CodeBuild。对于事件类型，所有事件应已选中。
9. 事件模式预览中应显示以下代码：

```
{
  "source": [
    "aws.codebuild"
  ]
}
```

10. 选择编辑并将事件模式预览中的代码替换为以下两个规则模式之一。

    每当一个构建开始或完成时，第一个规则模式就会为 AWS CodeBuild中的指定构建项目触发一个事件。

```
{
  "source": [
    "aws.codebuild"
  ],
  "detail-type": [
    "CodeBuild Build State Change"
  ],
  "detail": {
    "build-status": [
      "IN_PROGRESS",
      "SUCCEEDED",
      "FAILED",
      "STOPPED"
    ],
    "project-name": [
      "my-demo-project-1",
      "my-demo-project-2"
    ]
  }
}
```

    在前面的规则中，根据需要更改以下代码。

    - 要在每次构建开始或完成时触发事件，请保留 `build-status` 数组中显示的所有值，或删除整个 `build-status` 数组。

- 要仅在构建完成时触发事件，请从 build-status 阵列中删除 IN_PROGRESS。
- 要仅在构建开始时触发事件，请从 build-status 阵列中删除除 IN_PROGRESS 以外的所有值。
- 要为所有构建项目触发事件，请删除整个 project-name 阵列。
- 要仅为单个构建项目触发事件，请在 project-name 阵列中指定每个构建项目的名称。

每当构建从一个构建阶段转到另一个构建阶段时，第二个规则模式将为 AWS CodeBuild中的指定构建项目触发一个事件。

```
{
  "source": [
    "aws.codebuild"
  ],
  "detail-type": [
    "CodeBuild Build Phase Change"
  ],
  "detail": {
    "completed-phase": [
      "SUBMITTED",
      "PROVISIONING",
      "DOWNLOAD_SOURCE",
      "INSTALL",
      "PRE_BUILD",
      "BUILD",
      "POST_BUILD",
      "UPLOAD_ARTIFACTS",
      "FINALIZING"
    ],
    "completed-phase-status": [
      "TIMED_OUT",
      "STOPPED",
      "FAILED",
      "SUCCEEDED",
      "FAULT",
      "CLIENT_ERROR"
    ],
    "project-name": [
      "my-demo-project-1",
      "my-demo-project-2"
    ]
  }
```

```
}
```

在前面的规则中，根据需要更改以下代码。

- 要为每个构建阶段更改触发一个事件（这可以为每个构建发送最多 9 条通知），请保留 `completed-phase` 数组中显示的所有值，或删除整个 `completed-phase` 数组。
- 要仅针对单个构建阶段更改触发事件，请删除 `completed-phase` 阵列中您不希望为其触发事件的每个构建阶段的名称。
- 要针对所有构建阶段状态更改触发事件，请保留 `completed-phase-status` 阵列中显示的所有值，或删除整个 `completed-phase-status` 阵列。
- 要仅针对单个构建阶段状态更改触发事件，请删除 `completed-phase-status` 阵列中您不希望对其触发事件的每个构建阶段状态的名称。
- 要为所有构建项目触发事件，请删除 `project-name` 阵列。
- 要为单个构建项目触发事件，请在 `project-name` 阵列中指定每个构建项目的名称。

有关事件模式的更多信息，请参阅亚马逊 EventBridge 用户指南中的事件模式。

有关使用事件模式进行筛选的更多信息，请参阅《亚马逊 EventBridge 用户指南》中的使用事件模式进行Content-based筛选。

> **Note**
>
> 如果要同时为构建状态更改和构建阶段更改触发事件，则必须创建两个单独的规则：一个针对构建状态更改，另一个针对构建阶段更改。如果您尝试将两个规则合并为一个规则，则合并后的规则可能产生意外结果或停止协作。

替换完代码后，选择保存。

11. 对于目标，选择添加目标。
12. 在目标列表中，选择 SNS 主题。
13. 对于话题，选择您之前标识或创建的主题。
14. 展开配置输入，然后选择输入转换器。
15. 在输入路径框中，输入以下输入路径之一。

对于 `detail-type` 值为 `CodeBuild Build State Change` 的规则，输入以下内容。

```
{"build-id":"$.detail.build-id","project-name":"$.detail.project-name","build-
status":"$.detail.build-status"}
```

对于 detail-type 值为 CodeBuild Build Phase Change 的规则，输入以下内容。

```
{"build-id":"$.detail.build-id","project-name":"$.detail.project-name","completed-
phase":"$.detail.completed-phase","completed-phase-status":"$.detail.completed-
phase-status"}
```

要获取其他类型的信息，请参阅构建通知输入格式参考。

16. 在输入模板框中，输入以下输入模板之一。

    对于 detail-type 值为 CodeBuild Build State Change 的规则，输入以下内容。

    ```
    "Build '<build-id>' for build project '<project-name>' has reached the build status
     of '<build-status>'."
    ```

    对于 detail-type 值为 CodeBuild Build Phase Change 的规则，输入以下内容。

    ```
    "Build '<build-id>' for build project '<project-name>' has completed the build
     phase of '<completed-phase>' with a status of '<completed-phase-status>'."
    ```

17. 选择配置详细信息。
18. 在步骤 2：配置规则详细信息页面上，输入名称和可选描述。对于状态，将已启用保持选中状态。
19. 选择创建规则。
20. 创建构建项目、运行构建和查看构建信息。
21. 确认 CodeBuild 现在已成功发送构建通知。例如，检查您的收件箱中现在是否有构建通知电子邮件。

要更改规则的行为，请在 CloudWatch 控制台中选择要更改的规则，选择操作，然后选择编辑。对该规则进行更改，选择配置详细信息，然后选择更新规则。

要停止使用规则发送生成通知，请在 CloudWatch 控制台中选择要停止使用的规则，选择操作，然后选择禁用。

要完全删除规则，请在 CloudWatch 控制台中选择要删除的规则，选择操作，然后选择删除。

构建通知输入格式参考

CloudWatch 以 JSON 格式提供通知。

构建状态更改通知使用以下格式：

```
{
  "version": "0",
  "id": "c030038d-8c4d-6141-9545-00ff7b7153EX",
  "detail-type": "CodeBuild Build State Change",
  "source": "aws.codebuild",
  "account": "123456789012",
  "time": "2017-09-01T16:14:28Z",
  "region": "us-west-2",
  "resources":[
    "arn:aws:codebuild:us-west-2:123456789012:build/my-sample-project:8745a7a9-
c340-456a-9166-edf953571bEX"
  ],
  "detail":{
    "build-status": "SUCCEEDED",
    "project-name": "my-sample-project",
    "build-id": "arn:aws:codebuild:us-west-2:123456789012:build/my-sample-
project:8745a7a9-c340-456a-9166-edf953571bEX",
    "additional-information": {
      "artifact": {
        "md5sum": "da9c44c8a9a3cd4b443126e823168fEX",
        "sha256sum":
 "6ccc2ae1df9d155ba83c597051611c42d60e09c6329dcb14a312cecc0a8e39EX",
        "location": "arn:aws:s3:::codebuild-123456789012-output-bucket/my-output-
artifact.zip"
      },
      "environment": {
        "image": "aws/codebuild/standard:5.0",
        "privileged-mode": false,
        "compute-type": "BUILD_GENERAL1_SMALL",
        "type": "LINUX_CONTAINER",
        "environment-variables": []
      },
      "timeout-in-minutes": 60,
      "build-complete": true,
      "initiator": "MyCodeBuildDemoUser",
      "build-start-time": "Sep 1, 2017 4:12:29 PM",
      "source": {
        "location": "codebuild-123456789012-input-bucket/my-input-artifact.zip",
```

```
        "type": "S3"
      },
      "logs": {
        "group-name": "/aws/codebuild/my-sample-project",
        "stream-name": "8745a7a9-c340-456a-9166-edf953571bEX",
        "deep-link": "https://console.aws.amazon.com/cloudwatch/home?region=us-
west-2#logEvent:group=/aws/codebuild/my-sample-project;stream=8745a7a9-c340-456a-9166-
edf953571bEX"
      },
      "phases": [
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:12:29 PM",
          "end-time": "Sep 1, 2017 4:12:29 PM",
          "duration-in-seconds": 0,
          "phase-type": "SUBMITTED",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:12:29 PM",
          "end-time": "Sep 1, 2017 4:13:05 PM",
          "duration-in-seconds": 36,
          "phase-type": "PROVISIONING",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:13:05 PM",
          "end-time": "Sep 1, 2017 4:13:10 PM",
          "duration-in-seconds": 4,
          "phase-type": "DOWNLOAD_SOURCE",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:13:10 PM",
          "end-time": "Sep 1, 2017 4:13:10 PM",
          "duration-in-seconds": 0,
          "phase-type": "INSTALL",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
```

```
          "start-time": "Sep 1, 2017 4:13:10 PM",
          "end-time": "Sep 1, 2017 4:13:10 PM",
          "duration-in-seconds": 0,
          "phase-type": "PRE_BUILD",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:13:10 PM",
          "end-time": "Sep 1, 2017 4:14:21 PM",
          "duration-in-seconds": 70,
          "phase-type": "BUILD",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:14:21 PM",
          "end-time": "Sep 1, 2017 4:14:21 PM",
          "duration-in-seconds": 0,
          "phase-type": "POST_BUILD",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:14:21 PM",
          "end-time": "Sep 1, 2017 4:14:21 PM",
          "duration-in-seconds": 0,
          "phase-type": "UPLOAD_ARTIFACTS",
          "phase-status": "SUCCEEDED"
        },
         {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:14:21 PM",
          "end-time": "Sep 1, 2017 4:14:26 PM",
          "duration-in-seconds": 4,
          "phase-type": "FINALIZING",
          "phase-status": "SUCCEEDED"
        },
        {
          "start-time": "Sep 1, 2017 4:14:26 PM",
          "phase-type": "COMPLETED"
        }
      ]
    },
```

```
    "current-phase": "COMPLETED",
    "current-phase-context": "[]",
    "version": "1"
  }
}
```

构建阶段更改通知使用以下格式：

```
{
  "version": "0",
  "id": "43ddc2bd-af76-9ca5-2dc7-b695e15adeEX",
  "detail-type": "CodeBuild Build Phase Change",
  "source": "aws.codebuild",
  "account": "123456789012",
  "time": "2017-09-01T16:14:21Z",
  "region": "us-west-2",
  "resources":[
    "arn:aws:codebuild:us-west-2:123456789012:build/my-sample-project:8745a7a9-
c340-456a-9166-edf953571bEX"
  ],
  "detail":{
    "completed-phase": "COMPLETED",
    "project-name": "my-sample-project",
    "build-id": "arn:aws:codebuild:us-west-2:123456789012:build/my-sample-
project:8745a7a9-c340-456a-9166-edf953571bEX",
    "completed-phase-context": "[]",
    "additional-information": {
      "artifact": {
        "md5sum": "da9c44c8a9a3cd4b443126e823168fEX",
        "sha256sum":
 "6ccc2ae1df9d155ba83c597051611c42d60e09c6329dcb14a312cecc0a8e39EX",
        "location": "arn:aws:s3:::codebuild-123456789012-output-bucket/my-output-
artifact.zip"
      },
      "environment": {
        "image": "aws/codebuild/standard:5.0",
        "privileged-mode": false,
        "compute-type": "BUILD_GENERAL1_SMALL",
        "type": "LINUX_CONTAINER",
        "environment-variables": []
      },
      "timeout-in-minutes": 60,
      "build-complete": true,
```

```
      "initiator": "MyCodeBuildDemoUser",
      "build-start-time": "Sep 1, 2017 4:12:29 PM",
      "source": {
        "location": "codebuild-123456789012-input-bucket/my-input-artifact.zip",
        "type": "S3"
      },
      "logs": {
        "group-name": "/aws/codebuild/my-sample-project",
        "stream-name": "8745a7a9-c340-456a-9166-edf953571bEX",
        "deep-link": "https://console.aws.amazon.com/cloudwatch/home?region=us-
west-2#logEvent:group=/aws/codebuild/my-sample-project;stream=8745a7a9-c340-456a-9166-
edf953571bEX"
      },
      "phases": [
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:12:29 PM",
          "end-time": "Sep 1, 2017 4:12:29 PM",
          "duration-in-seconds": 0,
          "phase-type": "SUBMITTED",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:12:29 PM",
          "end-time": "Sep 1, 2017 4:13:05 PM",
          "duration-in-seconds": 36,
          "phase-type": "PROVISIONING",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:13:05 PM",
          "end-time": "Sep 1, 2017 4:13:10 PM",
          "duration-in-seconds": 4,
          "phase-type": "DOWNLOAD_SOURCE",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:13:10 PM",
          "end-time": "Sep 1, 2017 4:13:10 PM",
          "duration-in-seconds": 0,
          "phase-type": "INSTALL",
```

```
      "phase-status": "SUCCEEDED"
    },
    {
      "phase-context": [],
      "start-time": "Sep 1, 2017 4:13:10 PM",
      "end-time": "Sep 1, 2017 4:13:10 PM",
      "duration-in-seconds": 0,
      "phase-type": "PRE_BUILD",
      "phase-status": "SUCCEEDED"
    },
    {
      "phase-context": [],
      "start-time": "Sep 1, 2017 4:13:10 PM",
      "end-time": "Sep 1, 2017 4:14:21 PM",
      "duration-in-seconds": 70,
      "phase-type": "BUILD",
      "phase-status": "SUCCEEDED"
    },
    {
      "phase-context": [],
      "start-time": "Sep 1, 2017 4:14:21 PM",
      "end-time": "Sep 1, 2017 4:14:21 PM",
      "duration-in-seconds": 0,
      "phase-type": "POST_BUILD",
      "phase-status": "SUCCEEDED"
    },
    {
      "phase-context": [],
      "start-time": "Sep 1, 2017 4:14:21 PM",
      "end-time": "Sep 1, 2017 4:14:21 PM",
      "duration-in-seconds": 0,
      "phase-type": "UPLOAD_ARTIFACTS",
      "phase-status": "SUCCEEDED"
    },
    {
      "phase-context": [],
      "start-time": "Sep 1, 2017 4:14:21 PM",
      "end-time": "Sep 1, 2017 4:14:26 PM",
      "duration-in-seconds": 4,
      "phase-type": "FINALIZING",
      "phase-status": "SUCCEEDED"
    },
    {
      "start-time": "Sep 1, 2017 4:14:26 PM",
```

```
          "phase-type": "COMPLETED"
        }
      ]
    },
    "completed-phase-status": "SUCCEEDED",
    "completed-phase-duration-seconds": 4,
    "version": "1",
    "completed-phase-start": "Sep 1, 2017 4:14:21 PM",
    "completed-phase-end": "Sep 1, 2017 4:14:26 PM"
  }
}
```

使用以下方法生成徽章示例 CodeBuild

AWS CodeBuild 现在支持使用编译徽章，它提供可嵌入的、动态生成的图像（徽章），用于显示项目最新版本的状态。可通过为您的 CodeBuild 项目生成的公开网址访问此图片。这允许任何人查看 CodeBuild 项目的状态。构建徽章不包含任何安全信息，因此它们无需身份验证。

主题

- 创建具有构建徽章的构建项目
- 访问 AWS CodeBuild 制作徽章
- 发布 CodeBuild 编译徽章
- CodeBuild 徽章状态

创建具有构建徽章的构建项目

使用以下过程之一创建启用了生成徽章的构建项目。你可以使用 AWS CLI 或 AWS 管理控制台.

要创建启用构建徽章的构建项目 (AWS CLI)

- 有关创建构建项目的信息，请参阅创建构建项目 (AWS CLI)。要在 AWS CodeBuild 项目中包含编译徽章，必须指定值*badgeEnabled*为。`true`

创建已启用构建徽章的构建项目（控制台）

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 如果显示 CodeBuild 信息页面，请选择创建构建项目。否则，请在导航窗格中，展开构建，选择构建项目，然后选择创建构建项目。

3. 在项目名称中，输入此构建项目的名称。每个 AWS 账户的构建项目名称必须是唯一的。您还可以包含构建项目的可选描述，以帮助其他用户了解此项目的用途。
4. 在源中，对于源提供商，选择源代码提供商类型，然后执行以下操作之一：

> **Note**
>
> CodeBuild 不支持使用 Amazon S3 源提供商构建徽章。由于 AWS CodePipeline 使用 Amazon S3 进行构件传输，因此在中创建的管道中的 CodePipeline构建项目不支持构建徽章。

- 如果您选择了 CodeCommit，那么对于 Repository (存储库)，请选择存储库的名称。选择启用构建徽章，以使您的项目的构建状态可见且可嵌入。
- 如果选择 GitHub，请按照说明进行连接（或重新连接）。 GitHub在 GitHub授权应用程序页面上，对于组织访问权限，选择您希望 AWS CodeBuild 能够访问的每个存储库旁边的请求访问权限。选择授权应用程序后，返回 AWS CodeBuild 控制台，对于存储库，选择包含源代码的存储库的名称。选择启用构建徽章，以使您的项目的构建状态可见且可嵌入。
- 如果您选择了 Bitbucket，请按照说明连接（或重新连接）Bitbucket。在 Bitbucket 确认对账户的访问页面上，对于组织访问权限，选择授予访问权限。选择 “授予访问权限” 后，回到 AWS CodeBuild 控制台中，为存储库选择包含源代码的存储库的名称。选择启用构建徽章，以使您的项目的构建状态可见且可嵌入。

> **Important**
>
> 更新项目源可能会影响项目构建徽章的准确性。

5. 在环境中：

   对于环境映像，执行下列操作之一：

- 要使用由管理的 Docker 镜像 AWS CodeBuild，请选择 “托管映像”，然后从操作系统、运行时、映像和映像版本中进行选择。从环境类型中进行选择（如果可用）。
- 要使用其他 Docker 映像，请选择自定义映像。对于环境类型，请选择 ARM、Linux、Linux GPU 或 Windows。如果您针对外部注册表 URL 选择其他注册表，请使用 *docker repository*/*docker image name* 格式在 Docker Hub 中输入 Docker 映像的名称和标签。如果您选择亚马逊 ECR，请使用亚马逊 ECR 存储库和亚马逊 ECR 镜像在您的账户中选择 Docker 镜像。 AWS

- 要使用私有 Docker 映像，请选择自定义映像。对于环境类型，请选择 ARM、Linux、Linux GPU 或 Windows。对于映像注册表，选择其他注册表，然后输入您的私有 Docker 映像的凭证的 ARN。凭证必须由 Secrets Manager 创建。有关更多信息，请参阅《AWS Secrets Manager 用户指南》中的什么是 AWS Secrets Manager？。

6. 在服务角色中，执行下列操作之一：

   - 如果您没有 CodeBuild 服务角色，请选择新建服务角色。在角色名称中，为新角色输入名称。
   - 如果您有 CodeBuild 服务角色，请选择现有服务角色。在角色 ARN 中，选择服务角色。

   > **Note**
   >
   > 当您使用控制台创建或更新构建项目时，可以同时创建 CodeBuild 服务角色。默认情况下，这个角色仅能与该构建项目配合使用。如果您使用控制台将此服务角色与另一个构建项目关联，则此角色将更新以便与关联的构建项目结合使用。一个服务角色最多可与 10 个构建项目结合使用。

7. 在 Buildspec 中，执行以下操作之一：

   - 选择使用 buildspec 文件，以在源代码根目录中使用 buildspec.yml 文件。
   - 选择插入构建命令，以使用控制台插入构建命令。

   有关更多信息，请参阅Buildspec 参考。

8. 在构件中，对于类型，执行以下操作之一：

   - 如果您不想创建构建输出构件，请选择无构件。
   - 要将构建输出存储在 S3 存储桶中，请选择 Amazon S3，然后执行以下操作：
     - 如果要将项目名称用于构建输出 ZIP 文件或文件夹，请将名称留空。否则，请输入名称。默认情况下，构件名称是项目名称。如果您要使用其他名称，请在构件名称框中输入该名称。如果您要输出 ZIP 文件，请包含 zip 扩展名。
     - 对于存储桶名称，请选择输出存储桶的名称。
     - 如果您在此过程的前面部分选择了插入构建命令，对于输出文件，请输入构建（该构建要放到构建输出 ZIP 文件或文件夹中）中的文件位置。对于多个位置，使用逗号将各个位置隔开（例如，`appspec.yml, target/my-app.jar`）。有关更多信息，请参阅buildspec 语法中 `files` 的描述。

9. 展开其他配置并根据需要选择选项。

10. 选择 Create build project（创建构建项目）。在审核页面上，选择开始构建以运行构建。

访问 AWS CodeBuild 制作徽章

您可以使用 AWS CodeBuild 控制台或 AWS CLI 访问版本徽章。

- 在 CodeBuild 控制台的构建项目列表的名称列中，选择与构建项目对应的链接。在 “生成项目:***project-name***” 页面的 “配置” 中，选择 “复制徽章 URL” 。有关更多信息，请参阅 查看构建项目的详细信息（控制台）。
- 在中 AWS CLI，运行`batch-get-projects`命令。构建徽章 URL 包含在输出的项目环境详细信息部分中。有关更多信息，请参阅 查看构建项目的详细信息 (AWS CLI)。

构建徽章请求 URL 生成时会归入常见默认分支，但您可以指定源存储库中已用于运行构建的任何分支。例如：

```
https://codebuild.us-east-1.amazon.com/badges?uuid=...&branch=<branch>
```

您也可以通过将徽章 URL 中的 `branch` 参数替换为 `tag` 参数来在源存储库中指定标签。例如：

```
https://codebuild.us-east-1.amazon.com/badges?uuid=...&tag=<tag>
```

发布 CodeBuild 编译徽章

您可以使用 markdown 映像中的构建徽章 URL 显示 markdown 文件中最新构建的状态。这对于在源存储库（例如或）的 readme.md 文件中显示最新版本的状态很有用。 GitHub CodeCommit例如：

```
![](<build badge URL>)
```

CodeBuild 徽章状态

CodeBuild 构建徽章可以具有以下状态之一。

- PASSING 给定分支上的最新构建已传递。
- FAILING 给定分支上的最新构建已超时、失败、出现故障或停止。
- IN_PROGRESS 给定分支上的最新构建正在进行中。
- UNKNOWN 项目尚未为给定分支运行构建或根本未运行。此外，构建徽章功能可能已禁用。

'使用测试报告 AWS CLI'样本

您在 buildspec 文件中指定的测试将在构建期间运行。此示例向您展示了如何使用将测试合并 AWS CLI 到内部版本中 CodeBuild。您可以使用 JUnit 创建单元测试，也可以使用其他工具创建配置测试。然后，您可以评估测试结果以修复问题或优化您的应用程序。

您可以使用 CodeBuild API 或 AWS CodeBuild 控制台来访问测试结果。此示例演示如何配置报告以便将其测试结果导出到 S3 存储桶。

主题

- 运行测试报告示例

运行测试报告示例

使用以下步骤运行测试报告示例。

主题

- 先决条件
- 步骤 1：创建报告组
- 步骤 2：使用报告组配置项目
- 步骤 3：运行和查看报告结果

先决条件

- 创建您的测试用例。编写此示例时假设您有要包含在示例测试报告中的测试用例。您可以在 buildspec 文件中指定测试文件的位置。

  支持以下测试报告文件格式：

  - Cucumber JSON (.json)
  - JUnit XML (.xml)
  - NUnit XML (.xml)
  - NUnit3 XML (.xml)
  - TestNG XML (.xml)
  - Visual Studio TRX (.trx)
  - Visual Studio TRX XML (.xml)

使用任何测试框架创建测试用例，这些测试框架可以采用任何一种格式创建报告文件（例如 Surefire JUnit 插件，TestNG 或 Cucumber）。

- 创建 S3 存储桶并记下其名称。有关更多信息，请参阅《Amazon S3 用户指南》中的如何创建 S3 存储桶？
- 创建一个 IAM 角色并记下其 ARN。创建构建项目时，您需要 ARN。
- 如果您的角色没有以下权限，请添加它们。

```
{
    "Effect": "Allow",
    "Resource": [
        "*"
    ],
    "Action": [
        "codebuild:CreateReportGroup",
        "codebuild:CreateReport",
        "codebuild:UpdateReport",
        "codebuild:BatchPutTestCases"
    ]
}
```

有关更多信息，请参阅 测试报告操作的权限。

步骤 1：创建报告组

1. 创建一个名为 CreateReportGroupInput.json的文件。
2. 在 S3 存储桶中创建要将测试结果导出到的文件夹。
3. 将以下内容复制到 CreateReportGroupInput.json。对于 *<bucket-name>*，使用 S3 存储桶的名称。对于 *<path-to-folder>*，请输入 S3 存储桶中文件夹的路径。

```
{
  "name": "<report-name>",
  "type": "TEST",
  "exportConfig": {
    "exportConfigType": "S3",
    "s3Destination": {
      "bucket": "<bucket-name>",
      "path": "<path-to-folder>",
      "packaging": "NONE"
```

```
    }
  }
}
```

4. 在包含 CreateReportGroupInput.json 的目录中，运行以下命令：

```
aws codebuild create-report-group --cli-input-json file://
CreateReportGroupInput.json
```

输出如下所示。记下 reportGroup 的 ARN。您可以在创建使用此报告组的项目时使用该 ARN。

```
{
  "reportGroup": {
    "arn": "arn:aws:codebuild:us-west-2:123456789012:report-group/<report-name>",
    "name": "<report-name>",
    "type": "TEST",
    "exportConfig": {
      "exportConfigType": "S3",
      "s3Destination": {
        "bucket": "<s3-bucket-name>",
        "path": "<folder-path>",
        "packaging": "NONE",
        "encryptionKey": "arn:aws:kms:us-west-2:123456789012:alias/aws/s3"
      }
    },
    "created": 1570837165.885,
    "lastModified": 1570837165.885
  }
}
```

步骤 2：使用报告组配置项目

要运行报告，首先要创建一个使用您的报告组配置的 CodeBuild 生成项目。为报告组指定的测试用例将在您运行构建时运行。

1. 创建一个名为 buildspec.yml 的 buildspec 文件。
2. 使用以下 YAML 作为 buildspec.yml 文件的模板。请务必包含运行测试的命令。在 reports 部分中，指定包含测试用例结果的文件。这些文件存储您可以访问的测试结果 CodeBuild。它们在创建后 30 天过期。这些文件与您导出到 S3 存储桶的原始测试用例结果文件不同。

```
version: 0.2
    phases:
    install:
        runtime-versions:
            java: openjdk8
    build:
      commands:
        - echo Running tests
        - <enter commands to run your tests>

    reports:
      <report-name-or-arn>: #test file information
      files:
        - '<test-result-files>'
      base-directory: '<optional-base-directory>'
      discard-paths: false #do not remove file paths from test result files
```

> **Note**
>
> 您还可以为尚未创建的报告组指定名称，而不是使用现有报告组的 ARN。如果您指定名称而不是 ARN，则会在运行生成时 CodeBuild 创建报告组。其名称包含您的项目名称和您在 buildspec 文件中指定的名称，格式如下：`project-name-report-group-name`。有关更多信息，请参阅创建测试报告和报告组命名。

3. 创建一个名为 `project.json`的文件。此文件包含 create-project 命令的输入。
4. 将以下 JSON 复制到 `project.json`。对于 `source`，请输入包含源文件的存储库的类型和位置。对于 `serviceRole`，请指定您正在使用的角色的 ARN。

```
{
  "name": "test-report-project",
  "description": "sample-test-report-project",
  "source": {
    "type": "CODECOMMIT|CODEPIPELINE|GITHUB|S3|BITBUCKET|GITHUB_ENTERPRISE|
NO_SOURCE",
    "location": "<your-source-url>"
  },
  "artifacts": {
    "type": "NO_ARTIFACTS"
  },
  "cache": {
```

```
    "type": "NO_CACHE"
  },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "aws/codebuild/standard:5.0",
    "computeType": "small"
  },
  "serviceRole": "arn:aws:iam::<your-aws-account-id>:role/service-role/<your-role-name>"
}
```

5. 在包含 project.json 的目录中，运行以下命令：这将创建一个名为 test-project 的项目。

```
aws codebuild create-project --cli-input-json file://project.json
```

步骤 3：运行和查看报告结果

在此部分中，您将运行之前创建的项目的构建。在生成过程中， CodeBuild 创建包含测试用例结果的报告。该报告包含在您指定的报告组中。

1. 要开始构建，请运行以下命令。test-report-project 是上面创建的构建项目的名称。记下输出中显示的构建 ID。

```
aws codebuild start-build --project-name test-report-project
```

2. 运行以下命令以获取有关您的构建的信息，包括报告的 ARN。对于 *<build-id>*，请指定您的构建 ID。在输出的 reportArns 属性中记下报告 ARN。

```
aws codebuild batch-get-builds --ids <build-id>
```

3. 运行以下命令以获取有关报告的详细信息。对于 *<report-arn>*，请指定您的报告 ARN。

```
aws codebuild batch-get-reports --report-arns <report-arn>
```

输出如下所示。此示例输出显示了成功、失败、跳过、导致错误或返回未知状态的测试数量。

```
{
  "reports": [
    {
      "status": "FAILED",
```

```
      "reportGroupArn": "<report-group-arn>",
      "name": "<report-group-name>",
      "created": 1573324770.154,
      "exportConfig": {
        "exportConfigType": "S3",
        "s3Destination": {
          "bucket": "<amzn-s3-demo-bucket>",
          "path": "<path-to-your-report-results>",
          "packaging": "NONE",
          "encryptionKey": "<encryption-key>"
        }
      },
      "expired": 1575916770.0,
      "truncated": false,
      "executionId": "arn:aws:codebuild:us-west-2:123456789012:build/<name-of-
build-project>:2c254862-ddf6-4831-a53f-6839a73829c1",
      "type": "TEST",
      "arn": "<report-arn>",
      "testSummary": {
        "durationInNanoSeconds": 6657770,
        "total": 11,
        "statusCounts": {
          "FAILED": 3,
          "SKIPPED": 7,
          "ERROR": 0,
          "SUCCEEDED": 1,
          "UNKNOWN": 0
        }
      }
    }
  ],
  "reportsNotFound": []
}
```

4. 运行以下命令列出有关报告的测试用例的信息。对于 *<report-arn>*，请指定报告的 ARN。对于可选 `--filter` 参数，您可以指定一个状态结果（SUCCEEDED、FAILED、SKIPPED、ERROR 或 UNKNOWN）。

```
aws codebuild describe-test-cases \
    --report-arn <report-arn> \
    --filter status=SUCCEEDED|FAILED|SKIPPED|ERROR|UNKNOWN
```

输出如下所示。

```
{
  "testCases": [
    {
      "status": "FAILED",
      "name": "Test case 1",
      "expired": 1575916770.0,
      "reportArn": "<report-arn>",
      "prefix": "Cucumber tests for agent",
      "message": "A test message",
      "durationInNanoSeconds": 1540540,
      "testRawDataPath": "<path-to-output-report-files>"
    },
    {
      "status": "SUCCEEDED",
      "name": "Test case 2",
      "expired": 1575916770.0,
      "reportArn": "<report-arn>",
      "prefix": "Cucumber tests for agent",
      "message": "A test message",
      "durationInNanoSeconds": 1540540,
      "testRawDataPath": "<path-to-output-report-files>"
    }
  ]
}
```

的 Docker 示例 CodeBuild

本节介绍了 Docker 和之间的示例集成。 AWS CodeBuild

样本	说明
自定义镜像示例中的 Docker CodeBuild	此示例通过使用和自定义 Docker 构建映像（在 Docker Hub `docker:dind` 中）来构建 CodeBuild 和运行 Docker 镜像。
的 Docker 镜像构建服务器示例 CodeBuild	此示例将您的 Docker 构建卸载到托管式映像编译服务器。

样本	说明
Windows Docker 为以下版本构建示例 CodeBuild	此示例使用 CodeBuild构建并运行 Windows Docker 镜像。
“将 Docker 镜像发布到亚马逊 ECR 镜像存储库”示例 CodeBuild	该示例会生成一个 Docker 映像作为构建输出，然后将该 Docker 映像推送到 Amazon Elastic Container Registry (Amazon ECR) 映像存储库。
带有 AWS Secrets Manager 示例的私有注册表 CodeBuild	此示例向您展示如何使用存储在私有注册表中的 Docker 镜像作为 CodeBuild 运行时环境。

自定义镜像示例中的 Docker CodeBuild

以下示例通过使用和自定义 Docker 构建映像（在 Docker Hub `docker:dind` 中）来构建 AWS CodeBuild 和运行 Docker 镜像。

要了解如何改用由 Docker 支持的构建镜像来 CodeBuild 构建 Docker 镜像，请参阅我们的。“将 Docker 映像发布到 Amazon ECR”示例

> **Important**
>
> 运行此示例可能会导致您的 AWS 账户被扣款。其中包括与 Amazon S3 和 CloudWatch 日志相关的 AWS 资源和操作可能产生的费用。 CodeBuild AWS KMS有关更多信息，请参阅CodeBuild 定价、Amazon S3 定价、AWS Key Management Service 定价和亚马逊CloudWatch定价。

主题

- 在自定义映像示例中运行 Docker

在自定义映像示例中运行 Docker

使用以下过程在自定义映像示例中运行 Docker。有关此示例的更多信息，请参阅自定义镜像示例中的 Docker CodeBuild。

在自定义映像示例中运行 Docker

1. 按照本主题目录结构和文件部分所述创建文件，然后将其上传到 S3 输入存储桶或 AWS CodeCommit GitHub、或 Bitbucket 存储库。

   > **Important**
   >
   > 请不要上传 *(root directory name)*，而只上传 *(root directory name)* 中的文件。
   > 如果您使用的是 S3 输入存储桶，请务必创建一个包含这些文件的 ZIP 文件，然后将其上传到输入存储桶。请不要将 *(root directory name)* 添加到 ZIP 文件中，而只添加 *(root directory name)* 中的文件。

2. 创建构建项目、运行构建和查看相关构建信息。

   如果您使用 AWS CLI 来创建构建项目，则`create-project`命令的 JSON-formatted 输入可能与此类似。(请将占位符替换为您自己的值。)

```
{
  "name": "sample-docker-custom-image-project",
  "source": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-
bucket/DockerCustomImageSample.zip"
  },
  "artifacts": {
    "type": "NO_ARTIFACTS"
  },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "docker:dind",
    "computeType": "BUILD_GENERAL1_SMALL",
    "privilegedMode": false
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name",
  "encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

> **Note**
>
> 默认情况下，为非 VPC 构建启用 Docker 进程守护程序。如果您想使用 Docker 容器进行 VPC 构建，请参阅 Docker 文档网站上的运行时权限和 Linux 功能并启用特权模式。此外，Windows 不支持特权模式。

3. 要查看构建结果，请在构建的日志中查找字符串 `Hello, World!`。有关更多信息，请参阅 查看构建详细信息。

目录结构

此示例采用以下目录结构。

```
(root directory name)
### buildspec.yml
### Dockerfile
```

文件

在此示例中使用的操作系统的基本映像是 Ubuntu。此示例将使用这些文件。

`buildspec.yml` (在 *`(root directory name)`*)

```
version: 0.2

phases:
  pre_build:
    commands:
      - docker build -t helloworld .
  build:
    commands:
      - docker images
      - docker run helloworld echo "Hello, World!"
```

`Dockerfile` (在 *`(root directory name)`*)

```
FROM maven:3.3.9-jdk-8

RUN echo "Hello World"
```

的 Docker 镜像构建服务器示例 CodeBuild

以下示例将您的 Docker 构建卸载到托管式映像编译服务器。您可以调整此示例，在 CodeBuild 项目配置中配置专用的托管式 Docker 镜像构建服务器。请注意，当项目的构建正在运行时，预调配的实例处于活动状态，当构建未在运行时，实例将停止。预调配的实例最多可存储一个月，然后被回收利用。有关更多信息，请参阅 CodeBuild Docker 服务器功能。

> **Important**
>
> 运行此示例可能会向您的 AWS 账户收费。其中包括与 Amazon S3 和 CloudWatch 日志相关的 AWS 资源和操作可能产生的费用。 CodeBuild AWS KMS有关更多信息，请参阅CodeBuild 定价、 Amazon S3 定价、AWS Key Management Service 定价和亚马逊 CloudWatch定价。

主题

- 配置 Docker 服务器

配置 Docker 服务器

使用以下步骤为管理 Docker 工作负载和存储 Docker 镜像层的 CodeBuild 项目预置专用计算环境。

配置 Docker 服务器

1. 按照本主题目录结构和文件部分的描述创建文件，然后将它们上传到 S3 输入存储桶或 AWS CodeCommit GitHub、或 Bitbucket 存储库。

   > **Important**
   >
   > 请不要上传 *(root directory name)*，而只上传 *(root directory name)* 中的文件。
   > 如果您使用的是 S3 输入存储桶，请务必创建一个包含这些文件的 ZIP 文件，然后将其上传到输入存储桶。请不要将 *(root directory name)* 添加到 ZIP 文件中，而只添加 *(root directory name)* 中的文件。

2. 创建构建项目、运行构建和查看相关构建信息：

   a. 在控制台的环境部分，选择其他配置，导航到 Docker 服务器配置，然后选择为此项目启用 Docker 服务器。然后，您可以选择 Docker 服务器计算类型并提供注册表凭证。

b. 如果您使用创建构建项目，则`create-project`命令的 JSON-formatted 输入可能与此相似。 AWS CLI (请将占位符替换为您自己的值。)

```
{
  "name": "sample-docker-custom-image-project",
  "source": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-
bucket/DockerServerSample.zip"
  },
  "artifacts": {
    "type": "NO_ARTIFACTS"
  },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "aws/codebuild/amazonlinux-x86_64-standard:5.0",
    "computeType": "BUILD_GENERAL1_LARGE",
    "dockerServer": [
         {
            "computeType": "BUILD_GENERAL1_LARGE",
            "securityGroupIds": [ "security-groups-ID" ]
         }
      ]
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name"
}
```

> **Note**
>
> 为 Docker 服务器配置的安全组应支持来自在项目中配置的 VPC 的传入网络流量。它们应该支持在端口 9876 上传入。

3. 要查看构建结果，请在构建的日志中查找字符串 `Hello, World!`。有关更多信息，请参阅 查看构建详细信息。

目录结构

此示例采用以下目录结构。

```
(root directory name)
### buildspec.yml
```

```
### Dockerfile
```

文件

在此示例中使用的操作系统的基本映像是 Ubuntu。此示例将使用这些文件。

buildspec.yml（在 *(root directory name)*）

```
version: 0.2

phases:
  build:
    commands:
      - docker buildx build -t helloworld --load .
      - docker run helloworld echo "Hello, World!"
```

Dockerfile（在 *(root directory name)*）

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest

RUN echo "Hello World"
```

Windows Docker 为以下版本构建示例 CodeBuild

以下示例使用 CodeBuild构建和运行 Windows Docker 镜像。

主题

- 运行 Windows Docker 构建示例

运行 Windows Docker 构建示例

使用以下过程运行 Windows Docker 构建。

运行 Windows Docker 构建示例

1. 按照本主题目录结构和文件部分所述创建文件，然后将其上传到 S3 输入存储桶或 AWS CodeCommit GitHub、或 Bitbucket 存储库。

> **Important**
>
> 请不要上传 *(root directory name)*，而只上传 *(root directory name)* 中的文件。
> 如果您使用的是 S3 输入存储桶，请务必创建一个包含这些文件的 ZIP 文件，然后将其上传到输入存储桶。请不要将 *(root directory name)* 添加到 ZIP 文件中，而只添加 *(root directory name)* 中的文件。

2. 创建 WINDOWS_EC2 实例集。

   如果您使用创建舰队，则`create-fleet`命令的 JSON-formatted 输入可能与此类似。 AWS CLI (请将占位符替换为您自己的值。)

```
{
  "name": "fleet-name",
  "baseCapacity": 1,
  "environmentType": "WINDOWS_EC2",
  "computeType": "BUILD_GENERAL1_MEDIUM"
}
```

3. 创建构建项目、运行构建和查看相关构建信息。

   如果您使用 AWS CLI 来创建构建项目，则`create-project`命令的 JSON-formatted 输入可能与此类似。(请将占位符替换为您自己的值。)

```
{
  "name": "project-name",
  "source": {
    "type": "S3",
    "location": "bucket-name/DockerImageSample.zip"
  },
  "artifacts": {
    "type": "NO_ARTIFACTS"
  },
  "environment": {
    "type": "WINDOWS_EC2",
    "image": "Windows",
    "computeType": "BUILD_GENERAL1_MEDIUM",
    "fleet": {
       "fleetArn": "fleet-arn"
    }
```

```
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name"
}
```

4. 要查看构建结果，请在构建的日志中查找字符串 Hello, World!。有关更多信息，请参阅 查看构建详细信息。

目录结构

此示例采用以下目录结构。

```
(root directory name)
### buildspec.yml
### Dockerfile
```

文件

在此示例中使用的操作系统的基本映像是 mcr.microsoft.com/windows/servercore:ltsc2022。此示例将使用这些文件。

buildspec.yml（在 *(root directory name)*）

```
version: 0.2

phases:
  pre_build:
    commands:
      - docker build -t helloworld .
  build:
    commands:
      - docker images
      - docker run helloworld powershell -Command "Write-Host 'Hello World!'"
```

Dockerfile（在 *(root directory name)*）

```
FROM mcr.microsoft.com/windows/servercore:ltsc2022

RUN powershell -Command "Write-Host 'Hello World'"
```

“将 Docker 镜像发布到亚马逊 ECR 镜像存储库” 示例 CodeBuild

该示例会生成一个 Docker 映像作为构建输出，然后将该 Docker 映像推送到 Amazon Elastic Container Registry (Amazon ECR) 映像存储库。您可以调整该示例以将 Docker 映像推送到 Docker Hub。有关更多信息，请参阅 调整“将 Docker 映像发布到 Amazon ECR”示例，将其推送到 Docker Hub。

要了解如何使用自定义 Docker 构建映像来构建 Docker 映像（Docker Hub 中的 `docker:dind`），请参阅我们的自定义映像示例中的 Docker。

此示例参考 `golang:1.12` 进行了测试。

此示例使用新的多阶段 Docker 构建功能，该功能将生成一个 Docker 映像作为构建输出。然后，它将 Docker 镜像推送到亚马逊 ECR 镜像存储库。 Multi-stageDocker 镜像构建有助于减小最终的 Docker 镜像的大小。有关更多信息，请参阅将多阶段构建和 Docker 结合使用。

> **Important**
>
> 运行此示例可能会向您的 AWS 账户收费。其中包括与 Amazon S3 AWS KMS、 CloudWatch 日志和 Amazon ECR 相关的 AWS 资源和操作可能产生的费用。 AWS CodeBuild 有关更多信息，请参阅CodeBuild 定价、 Amazon S3 定价、AWS Key Management Service 定价、亚马逊 CloudWatch 定价和亚马逊弹性容器注册定价。

主题

- 运行“将 Docker 映像发布到 Amazon ECR”示例
- 调整“将 Docker 映像发布到 Amazon ECR”示例，将其推送到 Docker Hub

运行“将 Docker 映像发布到 Amazon ECR”示例

使用以下过程运行示例，将 Docker 映像发布到 Amazon ECR。有关此示例的更多信息，请参阅“将 Docker 镜像发布到亚马逊 ECR 镜像存储库” 示例 CodeBuild。

要运行此示例，请执行以下操作：

1. 如果 Amazon ECR 中已存在要使用的映像存储库，请跳至第 3 步。否则，如果您使用用户而不是 AWS 根账户或管理员用户来使用 Amazon ECR，请将此语句（介于*### BEGIN ADDING STATEMENT HERE ###*和之间*### END ADDING STATEMENT HERE ###*）添加到该用户（或与该用户关联的 IAM 群组）。不建议使用 AWS 根账户。此声明允许创建用于存储 Docker 镜

像的亚马逊 ECR 存储库。为了简洁起见，也为了帮您查找添加语句的位置，此处使用了省略号 (`...`)。请勿删除任何语句，也不要将这些省略号键入策略中。有关更多信息，请参阅《用户指南》中的通过 AWS 管理控制台使用内联策略。

JSON

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

> **Note**
>
> 修改该策略的 IAM 实体必须拥有在 IAM 中修改策略的权限。

2. 在 Amazon ECR 中创建映像存储库。请务必在创建构建环境和运行构建的同一 AWS 区域中创建存储库。有关更多信息，请参阅《Amazon ECR 用户指南》中的创建存储库。该存储库的名称必须与您将在此过程的稍后部分指定的存储库名称相匹配，并以 `IMAGE_REPO_NAME` 环境变量的形式表示。确保 Amazon ECR 存储库政策为您的 CodeBuild服务 IAM 角色授予图像推送访问权限。
3. 将此声明（介于*`### BEGIN ADDING STATEMENT HERE ###`*和之间*`### END ADDING STATEMENT HERE ###`*）添加到您附加到 AWS CodeBuild 服务角色的策略中。此声明允许将 Docker 镜像上传 CodeBuild 到亚马逊 ECR 存储库。为了简洁起见，也为了帮您查找添加语句的位置，此处使用了省略号 (`...`)。请勿删除任何语句，也不要将这些省略号键入策略中。

   JSON

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

```
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:CompleteLayerUpload",
                "ecr:GetAuthorizationToken",
                "ecr:InitiateLayerUpload",
                "ecr:PutImage",
                "ecr:UploadLayerPart"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> 修改该策略的 IAM 实体必须拥有在 IAM 中修改策略的权限。

4. 按照本主题目录结构和文件部分的描述创建文件，然后将它们上传到 S3 输入存储桶或 AWS CodeCommit GitHub、或 Bitbucket 存储库。有关更多信息，请参阅《AWS CodePipeline 用户指南》中的映像定义文件参考。

> **Important**
>
> 请不要上传 *(root directory name)*，而只上传 *(root directory name)* 中的文件。
> 如果您使用的是 S3 输入存储桶，请务必创建一个包含这些文件的 ZIP 文件，然后将其上传到输入存储桶。请不要将 *(root directory name)* 添加到 ZIP 文件中，而只添加 *(root directory name)* 中的文件。

5. 创建构建项目、运行构建和查看构建信息。

   如果您使用控制台创建项目：

   a. 对于操作系统，选择 Ubuntu。

   b. 对于运行时，选择标准。

   c. 对于图像，选择 aws/codebuild /standar d: 5.0。

   d. 添加以下环境变量：

- AWS_DEFAULT_REGION 值为 *region-ID*
- AWS_ACCOUNT_ID 值为 *account-ID*
- 具有最新值的 IMAGE_TAG
- 值为 IMAGE_REPO_NAME *Amazon-ECR-repo-name*

如果您使用创建构建项目，则`create-project`命令的 JSON-formatted 输入可能与此相似。AWS CLI (请将占位符替换为您自己的值。)

```
{
  "name": "sample-docker-project",
  "source": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-bucket/DockerSample.zip"
  },
  "artifacts": {
    "type": "NO_ARTIFACTS"
  },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "aws/codebuild/standard:5.0",
    "computeType": "BUILD_GENERAL1_SMALL",
    "environmentVariables": [
      {
        "name": "AWS_DEFAULT_REGION",
        "value": "region-ID"
      },
      {
        "name": "AWS_ACCOUNT_ID",
        "value": "account-ID"
      },
      {
        "name": "IMAGE_REPO_NAME",
        "value": "Amazon-ECR-repo-name"
      },
      {
        "name": "IMAGE_TAG",
        "value": "latest"
      }
    ],
  },
```

```
  "serviceRole": "arn:aws:iam::account-ID:role/role-name",
  "encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

6. 确认 CodeBuild 已成功将 Docker 镜像推送到存储库：

   1. 在以下位置打开亚马逊 ECR 控制台。 https://console.aws.amazon.com/ecr/
   2. 选择存储库名称。映像应在映像标签列中列出。

目录结构

此示例采用以下目录结构。

```
(root directory name)
### buildspec.yml
### Dockerfile
```

文件

此示例将使用这些文件。

buildspec.yml（在 *(root directory name)*）

```
version: 0.2

phases:
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - aws ecr get-login-password --region $AWS_DEFAULT_REGION | docker login --
username AWS --password-stdin $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $IMAGE_REPO_NAME:$IMAGE_TAG .
      - docker tag $IMAGE_REPO_NAME:$IMAGE_TAG $AWS_ACCOUNT_ID.dkr.ecr.
$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO_NAME:$IMAGE_TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker image...
```

```
      - docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/
$IMAGE_REPO_NAME:$IMAGE_TAG
```

`Dockerfile`（在 *(root directory name)*）

```
FROM golang:1.12-alpine AS build
#Install git
RUN apk add --no-cache git
#Get the hello world package from a GitHub repository
RUN go get github.com/golang/example/hello
WORKDIR /go/src/github.com/golang/example/hello
# Build the project and send the output to /bin/HelloWorld
RUN go build -o /bin/HelloWorld

FROM golang:1.12-alpine
#Copy the build's output binary from the previous build container
COPY --from=build /bin/HelloWorld /bin/HelloWorld
ENTRYPOINT ["/bin/HelloWorld"]
```

> **Note**
>
> CodeBuild 替换自定义 Docker 镜像的。`ENTRYPOINT`

调整“将 Docker 映像发布到 Amazon ECR”示例，将其推送到 Docker Hub

要调整“将 Docker 映像发布到 Amazon ECR”，以便将 Docker 映像推送到 Docker Hub 而不是推送到 Amazon ECR，请编辑示例的代码。有关示例的更多信息，请参阅 “将 Docker 镜像发布到亚马逊 ECR 镜像存储库” 示例 CodeBuild 和 运行“将 Docker 映像发布到 Amazon ECR”示例。

> **Note**
>
> 如果您使用的是 17.06 版本之前的 Docker 版本，请删除 `--no-include-email` 选项。

1. 替换`buildspec.yml`文件中的以下亚马逊代码 ECR-specific 行：

```
...
  pre_build:
    commands:
```

```
      - echo Logging in to Amazon ECR...
      - aws ecr get-login-password --region $AWS_DEFAULT_REGION |
 docker login --username AWS --password-stdin $AWS_ACCOUNT_ID.dkr.ecr.
$AWS_DEFAULT_REGION.amazonaws.com
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $IMAGE_REPO_NAME:$IMAGE_TAG .
      - docker tag $IMAGE_REPO_NAME:$IMAGE_TAG $AWS_ACCOUNT_ID.dkr.ecr.
$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO_NAME:$IMAGE_TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker image...
      - docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/
$IMAGE_REPO_NAME:$IMAGE_TAG
...
```

使用这些 Docker 代码 Hub-specific 行：

```
...
  pre_build:
    commands:
      - echo Logging in to Docker Hub...
      # Type the command to log in to your Docker Hub account here.
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $IMAGE_REPO_NAME:$IMAGE_TAG .
      - docker tag $IMAGE_REPO_NAME:$IMAGE_TAG $IMAGE_REPO_NAME:$IMAGE_TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker image...
      - docker push $IMAGE_REPO_NAME:$IMAGE_TAG
...
```

2. 将编辑后的代码上传到 S3 输入存储桶或 AWS CodeCommit GitHub、或 Bitbucket 存储库。

> **Important**
>
> 请不要上传 *(root directory name)*，而只上传 *(root directory name)* 中的文件。
> 如果您使用的是 S3 输入存储桶，请务必创建一个包含这些文件的 ZIP 文件，然后将其上传到输入存储桶。请不要将 *(root directory name)* 添加到 ZIP 文件中，而只添加 *(root directory name)* 中的文件。

3. 替换`create-project`命令 JSON-formatted 输入中的以下几行代码：

```
...
    "environmentVariables": [
      {
        "name": "AWS_DEFAULT_REGION",
        "value": "region-ID"
      },
      {
        "name": "AWS_ACCOUNT_ID",
        "value": "account-ID"
      },
      {
        "name": "IMAGE_REPO_NAME",
        "value": "Amazon-ECR-repo-name"
      },
      {
        "name": "IMAGE_TAG",
        "value": "latest"
      }
    ]
...
```

利用这些代码行，可以：

```
...
    "environmentVariables": [
      {
        "name": "IMAGE_REPO_NAME",
        "value": "your-Docker-Hub-repo-name"
      },
      {
```

```
        "name": "IMAGE_TAG",
        "value": "latest"
      }
    ]
...
```

4. 创建构建环境、运行构建和查看相关构建信息。
5. 确认 AWS CodeBuild 已成功将 Docker 镜像推送到存储库。登录 Docker Hub，再转至存储库，然后选择标签选项卡。`latest` 标签应包含最新的上次更新值。

带有 AWS Secrets Manager 示例的私有注册表 CodeBuild

此示例向您展示如何使用存储在私有注册表中的 Docker 镜像作为 AWS CodeBuild 运行时环境。私有注册表的凭证存储在 AWS Secrets Manager中。任何私有注册表都可使用 CodeBuild。此示例使用 Docker Hub。

> **Note**
>
> 密钥对操作可见，在写入文件时不会被屏蔽。

主题

- 私有注册表示例要求
- 使用私有注册表创建 CodeBuild 项目
- 为自托管运行程序配置私有注册表凭证

私有注册表示例要求

要将私有注册表与一起使用 AWS CodeBuild，您必须具备以下条件：

- 一个 Secrets Manager 密钥，用于储存您的 Docker Hub 凭证。该凭证可用于访问您的私有存储库。

  > **Note**
  >
  > 您将需要为您创建的密钥付费。

- 一个私有存储库或账户。

- 一项 CodeBuild 服务角色 IAM 策略，用于授予对您的 Secrets Manager 密钥的访问权限。

按照以下步骤创建这些资源，然后使用存储在私有注册表中的 Docker 镜像创建 CodeBuild 构建项目。

使用私有注册表创建 CodeBuild 项目

1. 有关如何创建免费的私有存储库的更多信息，请参阅 Docker Hub 上的存储库。您还可以在终端中运行以下命令来提取映像、获取其ID，并将其推送到新的存储库。

```
docker pull amazonlinux
docker images amazonlinux --format {{.ID}}
docker tag image-id your-username/repository-name:tag
docker login
docker push your-username/repository-name
```

2. 按照《AWS Secrets Manager 用户指南》中创建 AWS Secrets Manager 密钥中的步骤进行操作。

   a. 在步骤 3 中，在选择密钥类型，选择其他密钥类型。

   b. Key/value 成对地为您的 Docker Hub 用户名创建一个键值对，为你的 Docker Hub 密码创建一个键值对。

   c. 继续按照创建 AWS Secrets Manager 密钥中的步骤进行操作。

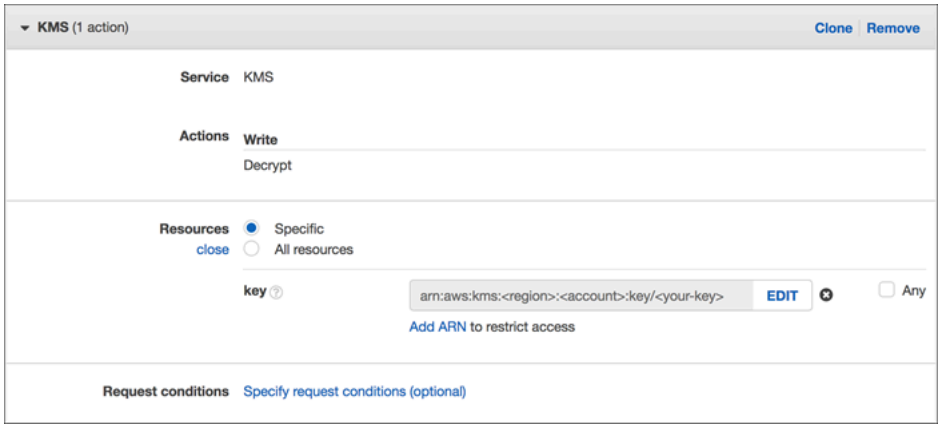
   d. 在步骤 5 中，在配置自动轮换页面上，将其关闭，因为密钥对应于您的 Docker Hub 凭证。

   e. 按照创建 AWS Secrets Manager 密钥中的步骤完成操作。

   有关更多信息，请参阅什么是 AWS Secrets Manager？

3. 在控制台中创建 AWS CodeBuild 项目时，会 CodeBuild 附上所需的权限。如果您使用的 AWS KMS 密钥除外`DefaultEncryptionKey`，则必须将其添加到服务角色中。有关更多信息，请参阅《IAM 用户指南》中的修改角色（控制台）。

   要使您的服务角色与 Secrets Manager 配合使用，它必须至少具有 `secretsmanager:GetSecretValue` 权限。



4. 要使用控制台创建一个具有在私有注册表中存储的环境的项目，请在创建项目时执行以下操作。有关信息，请参阅创建构建项目（控制台）。

> **Note**
>
> 如果您的私有注册表位于您的 VPC 中，则它必须具有公共互联网访问权限。 CodeBuild 无法从 VPC 中的私有 IP 地址提取镜像。

   a. 对于环境映像，选择自定义映像。
   b. 对于环境类型，选择 Linux 或 Windows。
   c. 对于映像注册表，请选择其他注册表。
   d. 在外部注册表 URL 中，输入映像位置，在注册表凭证 - 可选中输入您的 Secrets Manager 凭证的 ARN 或名称。

> **Note**
>
> 如果您的凭证在当前区域中不存在，则必须使用 ARN。如果凭证存在于其他区域中，则无法使用凭证名称。

为自托管运行程序配置私有注册表凭证

按照以下说明为自托管运行程序配置注册表凭证。

> **Note**
>
> 请注意，只有当映像被私有注册表中的映像覆盖时，才会使用这些凭证。

AWS Management Console

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 创建构建项目或选择现有项目。有关更多信息，请参阅 创建构建项目（控制台） 和 更改构建项目的设置（控制台）。
3. 在环境中，选择其他配置。
4. 在其他配置中，输入注册凭据的密钥名称或 ARN（ AWS Secrets Manager 可选）。



AWS CLI

1. 如果要创建一个新项目，请运行 create-project 命令。

```
aws codebuild create-project \
    --name project-name \
    --source type=source-type,location=source-location \
    --environment "type=environment-type,image=image,computeType=compute-
type,registryCredential={credentialProvider=SECRETS_MANAGER,credential=secret-
name-or-arn},imagePullCredentialsType=CODEBUILD|SERVICE_ROLE" \
    --artifacts type=artifacts-type \
    --service-role arn:aws:iam::account-ID:role/service-role/service-role-name
```

2. 如果要更新现有项目，请运行 update-project 命令。

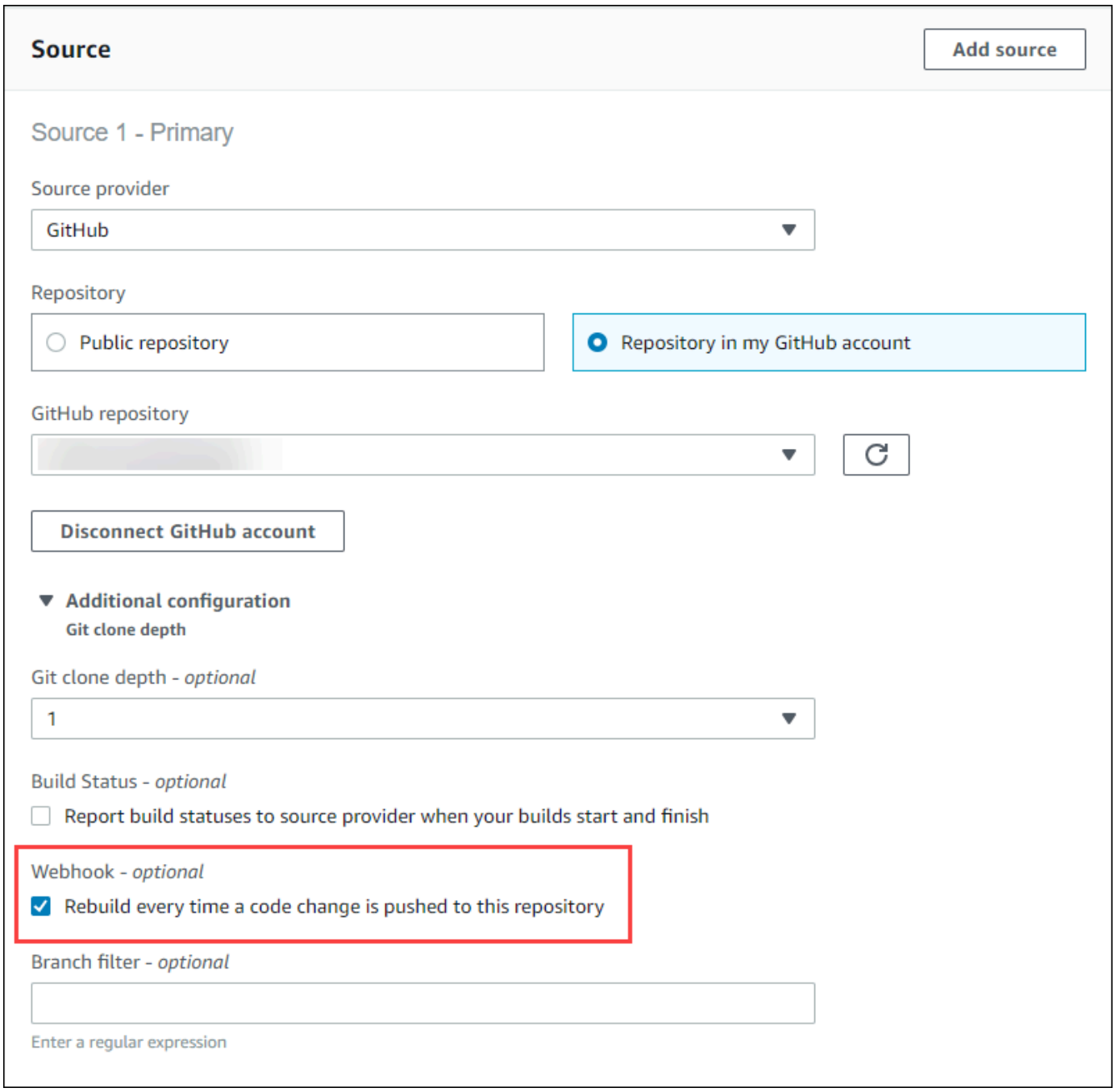
```
aws codebuild update-project \
    --name project-name \
    --environment "type=environment-type,image=image,computeType=compute-
type,registryCredential={credentialProvider=SECRETS_MANAGER,credential=secret-
name-or-arn}"
```

创建将构建输出托管在 S3 存储桶中的静态网站

您可以在构建中禁用构件加密。您可能需要禁用构件加密，以便将构件发布到配置为托管网站的位置。（您不能发布加密的构件。）此示例演示如何使用 Webhook 触发构建并将其构件发布到配置为网站的 S3 存储桶。

1. 按照设置静态网站中的说明操作，以配置一个以网站方式运行的 S3 存储桶。
2. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
3. 如果显示 CodeBuild 信息页面，请选择 “创建构建项目”。否则，请在导航窗格中，展开构建，选择构建项目，然后选择创建构建项目。
4. 在项目名称中，输入此构建项目的名称。每个 AWS 账户中的构建项目名称必须是唯一的。您还可以包含构建项目的可选描述，以帮助其他用户了解此项目的用途。
5. 在 Source (源) 中，对于 Source provider (源提供商)，选择 GitHub。按照说明进行连接（或重新连接）GitHub，然后选择 “授权”。

   对于 Webhook，选择每次将代码更改推送到此存储库时都会重新构建。仅当您已选中在我的账户中使用存储库时才选中此复选框。



6. 在环境中：

   对于环境映像，执行下列操作之一：

   - 要使用由管理的 Docker 映像 AWS CodeBuild，请选择托管映像，然后从 “操作系统”、“运行时”、“映像” 和 “映像版本” 中进行选择。从环境类型中进行选择（如果可用）。
   - 要使用其他 Docker 映像，请选择自定义映像。对于环境类型，请选择 ARM、Linux、Linux GPU 或 Windows。如果您针对外部注册表 URL 选择其他注册表，请使用 *docker*

repository/*docker image name* 格式在 Docker Hub 中输入 Docker 映像的名称和标签。如果您选择 Amazon ECR，请使用亚马逊 ECR 存储库和 A mazon ECR 镜像在您的账户中选择 Docker 镜像。 AWS

- 要使用私有 Docker 映像，请选择自定义映像。对于环境类型，请选择 ARM、Linux、Linux GPU 或 Windows。对于映像注册表，选择其他注册表，然后输入您的私有 Docker 映像的凭证的 ARN。凭证必须由 Secrets Manager 创建。有关更多信息，请参阅《AWS Secrets Manager 用户指南》中的什么是 AWS Secrets Manager？。

7. 在服务角色中，执行下列操作之一：

   - 如果您没有 CodeBuild 服务角色，请选择 “新建服务角色”。在角色名称中，为新角色输入名称。
   - 如果您有 CodeBuild 服务角色，请选择现有服务角色。在角色 ARN 中，选择服务角色。

   > **Note**
   >
   > 使用控制台创建或更新构建项目时，可以同时创建 CodeBuild 服务角色。默认情况下，这个角色仅能与该构建项目配合使用。如果您使用控制台将此服务角色与另一个构建项目关联，则此角色将更新以便与关联的构建项目结合使用。一个服务角色最多可与 10 个构建项目结合使用。

8. 在 Buildspec 中，执行以下操作之一：

   - 选择使用 buildspec 文件，以在源代码根目录中使用 buildspec.yml 文件。
   - 选择插入构建命令，以使用控制台插入构建命令。

   有关更多信息，请参阅Buildspec 参考。

9. 在构件中，对于类型，选择 Amazon S3 以将构建输出存储在 S3 存储桶中。
10. 对于存储桶名称，选择您在步骤 1 中配置以用作网站的 S3 存储桶的名称。
11. 如果您在环境中选择了插入构建命令，那么对于输出文件，请输入构建（该构建要放到输出存储桶中）中的文件位置。如果您有多个位置，请使用逗号分隔每个位置（例如，**`appspec.yml, target/my-app.jar`**）。有关更多信息，请参阅 Artifacts reference-key in the buildspec file。
12. 选择禁用构件加密。
13. 展开其他配置并根据需要选择选项。
14. 选择 Create build project（创建构建项目）。在构建项目页面上的构建历史记录中，选择开始构建以运行构建。

15. （可选）按照《Amazon S3 开发者指南》中的“示例：使用亚马逊加速您的网站” CloudFront 中的说明进行操作。

多输入源和输出构件示例

您可以使用多个输入源和多组输出构件创建 AWS CodeBuild 构建项目。此示例演示如何设置具有以下特点的构建项目：

- 使用多个不同类型的源和存储库。
- 将构建构件发布到单个构建中的多个 S3 存储桶。

在以下示例中，创建一个构建项目并使用它来运行构建。示例使用构建项目的 buildspec 文件演示如何合并多个源和创建多个构件集。

要了解如何创建使用多个源输入 CodeBuild 来创建多个输出构件的管道，请参阅包含多个输入源和输出工件的 CodePipeline/CodeBuild 集成示例。

主题

- 创建包含多个输入和输出的构建项目
- 创建没有源的构建项目

创建包含多个输入和输出的构建项目

按照以下过程来创建包含多个输入和输出的构建项目。

创建包含多个输入和输出的构建项目

1. 将您的源代码上传到一个或多个 S3 存储桶、 CodeCommit GitHub、 GitHub 企业服务器或 Bitbucket 存储库。
2. 选择一个源作为主要源。这是 CodeBuild 查找和运行构建规范文件的源代码。
3. 创建构建项目。有关更多信息，请参阅 在中创建构建项目 AWS CodeBuild。
4. 创建构建项目、运行构建和获取有关构建的信息。
5. 如果您使用创建构建项目，则该`create-project`命令的 JSON-formatted 输入可能与以下内容类似： AWS CLI

```
{
```

```
  "name": "sample-project",
  "source": {
    "type": "S3",
    "location": "<bucket/sample.zip>"
  },
  "secondarySources": [
    {
      "type": "CODECOMMIT",
      "location": "https://git-codecommit.us-west-2.amazonaws.com/v1/repos/repo",
      "sourceIdentifier": "source1"
    },
    {
      "type": "GITHUB",
      "location": "https://github.com/awslabs/aws-codebuild-jenkins-plugin",
      "sourceIdentifier": "source2"
    }
  ],
  "secondaryArtifacts": [ss
    {
      "type": "S3",
      "location": "<output-bucket>",
      "artifactIdentifier": "artifact1"
    },
    {
      "type": "S3",
      "location": "<other-output-bucket>",
      "artifactIdentifier": "artifact2"
    }
  ],
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "aws/codebuild/standard:5.0",
    "computeType": "BUILD_GENERAL1_SMALL"
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name",
  "encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

主要源在 source 属性下定义。所有其他源都称为辅助源，出现在 secondarySources 下方。所有辅助源都安装在各自的目录中。目录存储在内置环境变量 CODEBUILD_SRC_DIR_*sourceIdentifer* 中。有关更多信息，请参阅 构建环境中的环境变量。

`secondaryArtifacts` 属性包含构件定义列表。这些构件使用 buildspec 文件的 `secondary-artifacts` 块（嵌套在 `artifacts` 块内）。

buildspec 文件中的辅助构件与构件具有相同的结构，以其构件标识符分隔。

> **Note**
>
> 在 CodeBuild API 中，辅助项目的 `artifactIdentifier` 是 `CreateProject` 和 `UpdateProject` 中的必需属性。必须使用它引用辅助构件。

使用前面的 JSON-formatted 输入，该项目的构建规范文件可能如下所示：

```
version: 0.2

phases:
  install:
    runtime-versions:
      java: openjdk11
  build:
    commands:
      - cd $CODEBUILD_SRC_DIR_source1
      - touch file1
      - cd $CODEBUILD_SRC_DIR_source2
      - touch file2

artifacts:
  files:
    - '**.*'
  secondary-artifacts:
    artifact1:
      base-directory: $CODEBUILD_SRC_DIR_source1
      files:
        - file1
    artifact2:
      base-directory: $CODEBUILD_SRC_DIR_source2
      files:
        - file2
```

可以在 `StartBuild` 中使用具有 `sourceVersion` 属性的 API 覆盖主要源的版本。要覆盖一个或多个辅助源版本，请使用 `secondarySourceVersionOverride` 属性。

中start-build命令的 JSON-formatted 输入 AWS CLI 可能如下所示：

```
{
   "projectName": "sample-project",
   "secondarySourcesVersionOverride": [
      {
        "sourceIdentifier": "source1",
        "sourceVersion": "codecommit-branch"
      },
      {
        "sourceIdentifier": "source2",
        "sourceVersion": "github-branch"
      },
   ]
}
```

创建没有源的构建项目

在配置源代码时，您可以通过选择**NO_SOURCE**源类型来配置 CodeBuild 项目。当您的源类型为 **NO_SOURCE** 时，您不能指定 buildspec 文件，因为您的项目没有源。相反，您必须在 create-project CLI 命令 JSON-formatted 输入的buildspec属性中指定 YAML-formatted 构建规范字符串。它可能如下所示：

```
{
  "name": "project-name",
  "source": {
    "type": "NO_SOURCE",
    "buildspec": "version: 0.2\n\nphases:\n  build:\n    commands:\n      - command"
   },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "aws/codebuild/standard:5.0",
    "computeType": "BUILD_GENERAL1_SMALL",
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name",
  "encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

有关更多信息，请参阅 创建构建项目 (AWS CLI)。

buildspec 文件示例中的运行时版本 CodeBuild

如果您使用的是 Amazon Linux 2（AL2）标准映像版本 1.0 或更高版本或者 Ubuntu 标准映像版本 2.0 或更高版本，则可以在 buildspec 文件的 `runtime-versions` 部分中指定一个或多个运行时。以下示例向您展示了如何更改项目运行时、指定多个运行时以及指定依赖于另一个运行时的运行时。有关支持的运行时的信息，请参阅 Docker 镜像由 CodeBuild。

> **Note**
>
> 如果您在构建容器中使用 Docker，则您的构建必须在特权模式下运行。有关更多信息，请参阅手动运行 AWS CodeBuild 构建和在中创建构建项目 AWS CodeBuild。

主题

- 更新 buildspec 文件中的运行时版本
- 指定两个运行时

更新 buildspec 文件中的运行时版本

您可以通过更新 buildspec 文件的 `runtime-versions` 部分，将项目使用的运行时修改到新版本。以下示例说明如何指定 Java 版本 8 和 11。

- 一个 `runtime-versions` 部分，指定 Java 版本 8：

```
phases:
  install:
    runtime-versions:
      java: corretto8
```

- 一个 `runtime-versions` 部分，指定 Java 版本 11：

```
phases:
  install:
    runtime-versions:
      java: corretto11
```

以下示例说明如何使用 Ubuntu 标准映像 5.0 或 Amazon Linux 2 标准映像 3.0 指定不同版本的 Python：

- 一个 `runtime-versions` 部分，指定 Python 版本 3.7：

```
phases:
  install:
    runtime-versions:
      python: 3.7
```

- 一个 `runtime-versions` 部分，指定 Python 版本 3.8：

```
phases:
  install:
    runtime-versions:
      python: 3.8
```

本示例演示一个项目，该项目从 Java 版本 8 运行时开始，然后更新到 Java 版本 10 运行时。

1. 下载并安装 Maven。有关信息，请参阅 Apache Maven 网站上的下载 Apache Maven 和安装 Apache Maven。
2. 切换到您的本地计算机或实例上的空目录，然后运行此 Maven 命令。

```
mvn archetype:generate "-DgroupId=com.mycompany.app" "-DartifactId=ROOT" "-
DarchetypeArtifactId=maven-archetype-webapp" "-DinteractiveMode=false"
```

   如果成功，将创建此目录结构和文件。

```
.
### ROOT
    ### pom.xml
    ### src
        ### main
            ### resources
            ### webapp
                ### WEB-INF
                #   ### web.xml
                ### index.jsp
```

3. 使用以下内容创建名为 `buildspec.yml` 的文件。将此文件存储到 *(root directory name)*`/my-web-app` 目录。

```
version: 0.2
```

```
phases:
  install:
    runtime-versions:
      java: corretto8
  build:
    commands:
      - java -version
      - mvn package
artifacts:
  files:
    - '**/*'
  base-directory: 'target/my-web-app'
```

在 buildspec 文件中：

- `runtime-versions` 部分指定项目使用 Java 运行时版本 8。
- `- java -version` 命令显示您的项目在执行构建时所使用的 Java 版本。

您的文件结构现在应如下所示。

```
(root directory name)
### my-web-app
    ### src
    #   ### main
    #   ### resources
    #   ### webapp
    #       ### WEB-INF
    #           ### web.xml
    #               ### index.jsp
    ### buildspec.yml
    ### pom.xml
```

4. 将`my-web-app`目录的内容上传到 S3 输入存储桶或 CodeCommit GitHub、或 Bitbucket 存储库。

> **Important**
>
> 请不要上传 *(root directory name)* 或 *(root directory name)*/my-web-app，而只上传 *(root directory name)*/my-web-app 中的目录和文件。
> 如果您使用的是 S3 输入存储桶，请确保创建一个包含目录结构和文件的 ZIP 文件，然后将其上传至输入存储桶。请不要将 *(root directory name)* 或 *(root directory*

name)/my-web-app 添加到 ZIP 文件中，而只添加 *(root directory name)*/my-web-app 中的目录和文件。

5. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
6. 创建构建项目。有关更多信息，请参阅创建构建项目（控制台）和运行构建（控制台）。除这些设置以外，将所有设置保留为默认值。

   - 对于环境：
     - 对于环境映像，选择托管映像。
     - 对于操作系统，选择 Amazon Linux 2。
     - 对于运行时，选择标准。
     - 对于图片，请选择 aws/codebuild/amazon linux-x86_64-standard: 4.0。
7. 选择开始构建。
8. 在构建配置上，接受默认值，然后选择开始构建。
9. 当构建完成后，在构建日志选项卡上查看构建输出。您应该可以看到类似于如下所示的输出内容：

```
[Container] Date Time Phase is DOWNLOAD_SOURCE
[Container] Date Time CODEBUILD_SRC_DIR=/codebuild/output/src460614277/src
[Container] Date Time YAML location is /codebuild/output/src460614277/src/
buildspec.yml
[Container] Date Time Processing environment variables
[Container] Date Time Selecting 'java' runtime version 'corretto8' based on manual
 selections...
[Container] Date Time Running command echo "Installing Java version 8 ..."
Installing Java version 8 ...

[Container] Date Time Running command export JAVA_HOME="$JAVA_8_HOME"

[Container] Date Time Running command export JRE_HOME="$JRE_8_HOME"

[Container] Date Time Running command export JDK_HOME="$JDK_8_HOME"

[Container] Date Time Running command for tool_path in "$JAVA_8_HOME"/bin/*
 "$JRE_8_HOME"/bin/*;
```

10. 使用 Java 版本 11 更新 `runtime-versions` 部分：

```
install:
```

```
    runtime-versions:
      java: corretto11
```

11. 保存更改后，再次运行您的构建并查看构建输出。您应看到已安装的 Java 版本为 11。您应该可以看到类似于如下所示的输出内容：

```
[Container] Date Time Phase is DOWNLOAD_SOURCE
[Container] Date Time CODEBUILD_SRC_DIR=/codebuild/output/src460614277/src
[Container] Date Time YAML location is /codebuild/output/src460614277/src/
buildspec.yml
[Container] Date Time Processing environment variables
[Container] Date Time Selecting 'java' runtime version 'corretto11' based on manual
 selections...
Installing Java version 11 ...

[Container] Date Time Running command export JAVA_HOME="$JAVA_11_HOME"

[Container] Date Time Running command export JRE_HOME="$JRE_11_HOME"

[Container] Date Time Running command export JDK_HOME="$JDK_11_HOME"

[Container] Date Time Running command for tool_path in "$JAVA_11_HOME"/bin/*
 "$JRE_11_HOME"/bin/*;
```

指定两个运行时

您可以在同一个 CodeBuild 构建项目中指定多个运行时。此示例项目使用两个源文件：一个使用 Go 运行时，另一个使用 Node.js 运行时。

1. 创建一个名为 `my-source` 的目录。
2. 在 `my-source` 目录中，创建一个名为 `golang-app` 的目录。
3. 使用以下内容创建名为 `hello.go` 的文件。将此文件存储到 `golang-app` 目录。

```
package main
import "fmt"

func main() {
  fmt.Println("hello world from golang")
  fmt.Println("1+1 =", 1+1)
  fmt.Println("7.0/3.0 =", 7.0/3.0)
  fmt.Println(true && false)
```

```
  fmt.Println(true || false)
  fmt.Println(!true)
  fmt.Println("good bye from golang")
}
```

4. 在 my-source 目录中，创建一个名为 nodejs-app 的目录。它应该与 golang-app 目录同级。

5. 使用以下内容创建名为 index.js 的文件。将此文件存储到 nodejs-app 目录。

```
console.log("hello world from nodejs");
console.log("1+1 =" + (1+1));
console.log("7.0/3.0 =" + 7.0/3.0);
console.log(true && false);
console.log(true || false);
console.log(!true);
console.log("good bye from nodejs");
```

6. 使用以下内容创建名为 package.json 的文件。将此文件存储到 nodejs-app 目录。

```
{
  "name": "mycompany-app",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"run some tests here\""
  },
  "author": "",
  "license": "ISC"
}
```

7. 使用以下内容创建名为 buildspec.yml 的文件。将文件存储在 my-source 目录中，该目录与 nodejs-app 以及 golang-app 目录同级。该runtime-versions部分指定了 Node.js 版本 12 和 Go 版本 1.13 的运行时。

```
version: 0.2

phases:
  install:
    runtime-versions:
      golang: 1.13
      nodejs: 12
```

```
  build:
    commands:
      - echo Building the Go code...
      - cd $CODEBUILD_SRC_DIR/golang-app
      - go build hello.go
      - echo Building the Node code...
      - cd $CODEBUILD_SRC_DIR/nodejs-app
      - npm run test
artifacts:
  secondary-artifacts:
    golang_artifacts:
      base-directory: golang-app
      files:
        - hello
    nodejs_artifacts:
      base-directory: nodejs-app
      files:
        - index.js
        - package.json
```

8. 您的文件结构现在应如下所示。

```
my-source
### golang-app
#   ### hello.go
### nodejs.app
#   ### index.js
#   ### package.json
### buildspec.yml
```

9. 将`my-source`目录的内容上传到 S3 输入存储桶或 CodeCommit GitHub、或 Bitbucket 存储库。

> **Important**
>
> 如果您使用的是 S3 输入存储桶，请确保创建一个包含目录结构和文件的 ZIP 文件，然后将其上传至输入存储桶。请不要将 `my-source` 添加到 ZIP 文件中，而只添加 `my-source` 中的目录和文件。

10. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
11. 创建构建项目。有关更多信息，请参阅创建构建项目（控制台）和运行构建（控制台）。除这些设置以外，将所有设置保留为默认值。

- 对于环境：
  - 对于环境映像，选择托管映像。
  - 对于操作系统，选择 Amazon Linux 2。
  - 对于运行时，选择标准。
  - 对于图片，请选择 aws/codebuild/amazon linux-x86_64-standard: 4.0。

12. 选择 Create build project（创建构建项目）。
13. 选择开始构建。
14. 在构建配置上，接受默认值，然后选择开始构建。
15. 当构建完成后，在构建日志选项卡上查看构建输出。您应该可以看到类似于如下所示的输出内容。它显示了 Go 和 Node.js 运行时的输出。它还显示来自 Go 和 Node.js 应用程序的输出。

```
[Container] Date Time Processing environment variables
[Container] Date Time Selecting 'golang' runtime version '1.13' based on manual
 selections...
[Container] Date Time Selecting 'nodejs' runtime version '12' based on manual
 selections...
[Container] Date Time Running command echo "Installing Go version 1.13 ..."
Installing Go version 1.13 ...

[Container] Date Time Running command echo "Installing Node.js version 12 ..."
Installing Node.js version 12 ...

[Container] Date Time Running command n $NODE_12_VERSION
   installed : v12.20.1 (with npm 6.14.10)

[Container] Date Time Moving to directory /codebuild/output/src819694850/src
[Container] Date Time Registering with agent
[Container] Date Time Phases found in YAML: 2
[Container] Date Time  INSTALL: 0 commands
[Container] Date Time  BUILD: 1 commands
[Container] Date Time Phase complete: DOWNLOAD_SOURCE State: SUCCEEDED
[Container] Date Time Phase context status code:  Message:
[Container] Date Time Entering phase INSTALL
[Container] Date Time Phase complete: INSTALL State: SUCCEEDED
[Container] Date Time Phase context status code:  Message:
[Container] Date Time Entering phase PRE_BUILD
[Container] Date Time Phase complete: PRE_BUILD State: SUCCEEDED
[Container] Date Time Phase context status code:  Message:
[Container] Date Time Entering phase BUILD
```

```
[Container] Date Time Running command echo Building the Go code...
Building the Go code...

[Container] Date Time Running command cd $CODEBUILD_SRC_DIR/golang-app

[Container] Date Time Running command go build hello.go

[Container] Date Time Running command echo Building the Node code...
Building the Node code...

[Container] Date Time Running command cd $CODEBUILD_SRC_DIR/nodejs-app

[Container] Date Time Running command npm run test

> mycompany-app@1.0.0 test /codebuild/output/src924084119/src/nodejs-app
> echo "run some tests here"

run some tests here
```

源版本示例，包含 AWS CodeBuild

此示例演示如何使用提交 ID 以外的格式（也称为提交 SHA）指定源的版本。您可以通过以下方式指定源的版本：

- 对于 Amazon S3 源提供商，请使用表示构建输入 ZIP 文件的对象的版本 ID。
- 对于 CodeCommit Bitbucket 和 E GitHub nterprise Server，请使用以下任一选项： GitHub
  - 拉取请求作为拉取请求参考（例如，`refs/pull/1/head`）。
  - 分支作为分支名称。
  - 提交 ID。
  - 标签。
  - 引用和提交 ID。参考文献可以是以下之一：
    - 标签（例如，`refs/tags/mytagv1.0^{full-commit-SHA}`）。
    - 分支（例如，`refs/heads/mydevbranch^{full-commit-SHA}`）。
    - 拉取请求（例如，`refs/pull/1/head^{full-commit-SHA}`）。
- 对于 GitLab 和 GitLab 自我管理，请使用以下选项之一：
  - 分支作为分支名称。
  - 提交 ID。

- 标签。

> **Note**
>
> 只有当您的存储库是 GitHub 或 E GitHub nterprise Server 时，您才能指定拉取请求源的版本。

如果使用参考和提交 ID 指定版本，则构建的 DOWNLOAD_SOURCE 阶段比仅提供版本时更快。这是因为添加引用时， CodeBuild 无需下载整个存储库即可找到提交。

- 可以仅使用提交 ID 指定源版本，例如 12345678901234567890123467890123456789。如果这样做，则 CodeBuild 必须下载整个存储库才能找到版本。
- 您可以按此格式使用参考和提交 ID 指定源版本：*refs/heads/branchname*^{*full-commit-SHA*}（例如 refs/heads/main^{12345678901234567890123467890123456789}）。如果执行此操作，则仅 CodeBuild 下载指定的分支以查找版本。

> **Note**
>
> 为了加快构建DOWNLOAD_SOURCE阶段，您还可以将 Git 克隆深度设置为较低的数字。CodeBuild 下载存储库的版本更少。

主题

- 使用提交 ID 指定 GitHub 存储库版本
- 使用引用和提交 ID 指定 GitHub 存储库版本

使用提交 ID 指定 GitHub 存储库版本

可以仅使用提交 ID 指定源版本，例如 12345678901234567890123467890123456789。如果这样做，则 CodeBuild 必须下载整个存储库才能找到版本。

使用提交 ID 指定 GitHub 存储库版本

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。

2. 创建构建项目。有关信息，请参阅创建构建项目（控制台）和运行构建（控制台）。除这些设置以外，将所有设置保留为默认值：

   - 在源中：
     - 对于源提供商，选择GitHub。如果您未连接到 GitHub，请按照说明进行连接。
     - 对于存储库，选择公共存储库。
     - 对于存储库 URL，输入 **https://github.com/aws/aws-sdk-ruby.git**。
   - 在环境中：
     - 对于环境映像，选择托管映像。
     - 对于操作系统，选择 Amazon Linux 2。
     - 对于运行时，选择标准。
     - 对于图片，请选择 aws/codebuild/amazon linux-x86_64-standard: 4.0。
3. 对于构建规范，选择插入构建命令，然后选择切换到编辑器。
4. 在构建命令中，将占位符文本替换为以下内容：

```
version: 0.2

phases:
  install:
    runtime-versions:
      ruby: 2.6
  build:
    commands:
       - echo $CODEBUILD_RESOLVED_SOURCE_VERSION
```

   在使用 Ubuntu 标准映像 2.0 时需要 `runtime-versions` 部分。这里指定了 Ruby 版本 2.6 运行时，但您可以使用任何运行时。`echo` 命令显示存储在 `CODEBUILD_RESOLVED_SOURCE_VERSION` 环境变量中的源代码的版本。
5. 在构建配置上，接受默认值，然后选择开始构建。
6. 对于源版本，请输入 **046e8b67481d53bdc86c3f6affdd5d1afae6d369**。这是 `https://github.com/aws/aws-sdk-ruby.git` 存储库中提交的 SHA。
7. 选择开始构建。
8. 在构建完成后，您应该看到以下内容：

   - 在构建日志选项卡上，使用了哪个版本的项目源。见下列。

```
[Container] Date Time Running command echo $CODEBUILD_RESOLVED_SOURCE_VERSION
046e8b67481d53bdc86c3f6affdd5d1afae6d369

[Container] Date Time Phase complete: BUILD State: SUCCEEDED
```

- 在环境变量选项卡上，解析的源版本与用于创建构建的提交 ID 匹配。
- 在阶段详细信息选项卡上，显示 DOWNLOAD_SOURCE 阶段的持续时间。

使用引用和提交 ID 指定 GitHub 存储库版本

您可以按此格式使用参考和提交 ID 指定源版本：*refs*/*heads*/*branchname*^{*full-commit-SHA*}（例如 refs/heads/main^{12345678901234567890123467890123456789}）。如果执行此操作，则仅 CodeBuild 下载指定的分支以查找版本。

使用引用和提交 ID 指定 GitHub 存储库版本。

1. 完成 使用提交 ID 指定 GitHub 存储库版本 中的步骤。
2. 在左侧导航窗格中，选择构建项目，然后选择您之前创建的项目。
3. 选择开始构建。
4. 在源版本中，输入 **refs/heads/main^{046e8b67481d53bdc86c3f6affdd5d1afae6d369}**。这是相同的提交 ID 以及格式为 *refs*/*heads*/*branchname*^{*full-commit-SHA*} 的分支参考。
5. 选择开始构建。
6. 在构建完成后，您应该看到以下内容：

   - 在构建日志选项卡上，使用了哪个版本的项目源。见下列。

   ```
   [Container] Date Time Running command echo $CODEBUILD_RESOLVED_SOURCE_VERSION
   046e8b67481d53bdc86c3f6affdd5d1afae6d369

   [Container] Date Time Phase complete: BUILD State: SUCCEEDED
   ```

   - 在环境变量选项卡上，解析的源版本与用于创建构建的提交 ID 匹配。
   - 在阶段详细信息选项卡上，DOWNLOAD_SOURCE 阶段的持续时间应短于仅使用提交 ID 指定源版本时的持续时间。

Third-party 的源存储库示例 CodeBuild

本节介绍第三方源代码库与之间的集成示例。 CodeBuild

样本	说明
BitBucket 拉取请求和 webhook 过滤器示例 — 参见 运行 “Bitbucket 拉取请求和 webhook 过滤器” 示例 CodeBuild	此示例向您演示如何使用 Bitbucket 存储库创建拉取请求。它还向您演示如何使用 Bitbucket Webhook 来触发 CodeBuild 创建一个项目的生成。
GitHub 企业服务器示例 — 参见 运行以下内容的 GitHub 企业服务器示例 CodeBuild	此示例向您展示在 GitHub 企业服务器存储库安装了证书时如何设置 CodeBuild 项目。它还展示了如何启用 webhook，以便每次将代码更改推送到 GitHub 企业服务器存储库时都能 CodeBuild 重新生成源代码。
GitHub 拉取请求和 webhook 过滤器示例 — 参见 运行 GitHub 拉取请求和 webhook 过滤器示例 CodeBuild	此示例向您展示如何使用 GitHub企业服务器存储库创建拉取请求。它还展示了如何启用 webhook，以便每次将代码更改推送到 GitHub 企业服务器存储库时都能 CodeBuild 重新生成源代码。

运行 “Bitbucket 拉取请求和 webhook 过滤器” 示例 CodeBuild

AWS CodeBuild 当源存储库为 Bitbucket 时，支持 webhook。这意味着，对于源代码存储在 Bitbucket 存储库中的 CodeBuild 构建项目，每次将代码更改推送到存储库时，都可以使用 webhook 来重建源代码。有关更多信息，请参阅 Bitbucket Webhook 事件。

此示例向您演示如何使用 Bitbucket 存储库创建拉取请求。它还向你展示了如何使用 Bitbucket webhook CodeBuild 来触发创建项目的构建。

> **Note**
>
> 使用 webhook 时，用户可能会触发意外构建。要降低这种风险，请参阅使用 Webhook 的最佳实操。

主题

- 先决条件
- 步骤 1：使用 Bitbucket 创建构建项目并启用 webhook
- 步骤 2：使用 Bitbucket webhook 触发构建

先决条件

要运行此示例，您必须将您的 AWS CodeBuild 项目与 Bitbucket 账户关联起来。

> **Note**
>
> CodeBuild 已更新其使用 Bitbucket 的权限。如果您之前将项目连接到 Bitbucket，但现在收到 Bitbucket 连接错误，则必须重新连接才能授予管理您的 webh CodeBuild ook 的权限。

步骤 1：使用 Bitbucket 创建构建项目并启用 webhook

以下步骤介绍如何使用 Bitbucket 作为源存储库创建 AWS CodeBuild 项目并启用 webhook。

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 如果显示 CodeBuild 信息页面，请选择 “创建构建项目”。否则，请在导航窗格中，展开构建，选择构建项目，然后选择创建构建项目。
3. 选择创建构建项目。
4. 在项目配置中：

   项目名称

   输入此构建项目的名称。每个 AWS 账户中的构建项目名称必须是唯一的。您还可以包含构建项目的可选描述，以帮助其他用户了解此项目的用途。
5. 在源中：

   源提供商

   选择 Bitbucket。按照说明连接（或重新连接）Bitbucket，然后选择授权。

   存储库

   选择我的 Bitbucket 账户中的存储库。

如果您之前未连接过 Bitbucket 账户，请输入您的 Bitbucket 用户名和应用程序密码，然后选择保存 Bitbucket 凭证。要获取 Bitbucket API 令牌，请改为输入您的 Atlassian 账户电子邮件地址和 API 令牌。

Bitbucket 存储库

输入 Bitbucket 存储库的 URL。

6. 在主要源 webhook 事件中，选择以下内容。

> **Note**
>
> 只有在上一步中选择了我的 Bitbucket 账户中的存储库，主要源 Webhook 事件部分才可见。

1. 创建项目时，选择每次将代码更改推送到此存储库时都会重新构建。
2. 从事件类型中，选择一个或多个事件。
3. 要在事件触发构建时进行筛选，请在在这些条件下开始构建下，添加一个或多个可选筛选条件。
4. 要在未触发事件时进行筛选，请在在这些条件下不开始构建下，添加一个或多个可选筛选条件。
5. 选择添加筛选条件组，以添加另一个筛选条件组（如果需要）。

有关 Bitbucket webhook 事件类型和筛选的更多信息，请参阅 Bitbucket Webhook 事件。

7. 在环境中：

环境映像

选择下列选项之一：

要使用由 AWS CodeBuild以下管理的 Docker 镜像，请执行以下操作：

选择托管映像，然后选择操作系统、运行时、映像和映像版本。从环境类型中进行选择（如果可用）。

要使用其他 Docker 映像：

选择自定义映像。对于环境类型，请选择 ARM、Linux、Linux GPU 或 Windows。如果您针对外部注册表 URL 选择其他注册表，请使用 *docker repository*/*docker image*

name 格式在 Docker Hub 中输入 Docker 映像的名称和标签。如果您选择 Amazon ECR，请使用亚马逊 ECR 存储库和 A mazon ECR 镜像在您的账户中选择 Docker 镜像。 AWS

要使用私有 Docker 映像，请执行以下操作：

选择自定义映像。对于环境类型，请选择 ARM、Linux、Linux GPU 或 Windows。对于映像注册表，选择其他注册表，然后输入您的私有 Docker 映像的凭证的 ARN。凭证必须由 Secrets Manager 创建。有关更多信息，请参阅什么是 AWS Secrets Manager？ 在《AWS Secrets Manager 用户指南》中。

服务角色

选择下列选项之一：

- 如果您没有 CodeBuild 服务角色，请选择 “新建服务角色”。在角色名称中，为新角色输入名称。
- 如果您有 CodeBuild 服务角色，请选择现有服务角色。在角色 ARN 中，选择服务角色。

> **Note**
>
> 使用控制台创建或更新构建项目时，可以同时创建 CodeBuild 服务角色。默认情况下，这个角色仅能与该构建项目配合使用。如果您使用控制台将此服务角色与另一个构建项目关联，则此角色将更新以便与关联的构建项目结合使用。一个服务角色最多可与 10 个构建项目结合使用。

8. 在 Buildspec 中，执行以下操作之一：

   - 选择使用 buildspec 文件，以在源代码根目录中使用 buildspec.yml 文件。
   - 选择插入构建命令，以使用控制台插入构建命令。

   有关更多信息，请参阅Buildspec 参考。

9. 在构件中：

   类型

   选择下列选项之一：

   - 如果您不想创建构建输出构件，请选择无构件。
   - 要将构建输出存储在 S3 存储桶中，请选择 Amazon S3，然后执行以下操作：

- 如果要将项目名称用于构建输出 ZIP 文件或文件夹，请将名称留空。否则，请输入名称。默认情况下，构件名称是项目名称。如果您要使用其他名称，请在构件名称框中输入该名称。如果您要输出 ZIP 文件，请包含 zip 扩展名。
- 对于存储桶名称，请选择输出存储桶的名称。
- 如果您在此过程的前面部分选择了插入构建命令，对于输出文件，请输入构建（该构建要放到构建输出 ZIP 文件或文件夹中）中的文件位置。对于多个位置，使用逗号将各个位置隔开（例如，`appspec.yml, target/my-app.jar`）。有关更多信息，请参阅 buildspec 语法中 `files` 的描述。

其他配置

展开其他配置并根据需要设置选项。

10. 选择 Create build project（创建构建项目）。在审核页面上，选择开始构建以运行构建。

步骤 2：使用 Bitbucket webhook 触发构建

对于使用 Bitbucket webhook 的项目，当 Bitbucket 存储库检测到您的源代码发生变化时，会 AWS CodeBuild 创建一个构建。

1. 打开 AWS CodeBuild 控制台，网址为 https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格上，选择构建项目，然后选择项目以与 Bitbucket 存储库和 Webhook 关联。有关创建 Bitbucket webhook 项目的信息，请参阅 the section called “步骤 1：使用 Bitbucket 创建构建项目并启用 webhook”。
3. 在您项目的 Bitbucket 存储库中更改一些代码。
4. 在 Bitbucket 存储库上创建拉取请求。有关更多信息，请参阅发出拉取请求。
5. 在 Bitbucket Webhook 页面上，选择查看请求，以查看最新事件的列表。
6. 选择查看详细信息以查看返回的响应的详细信息 CodeBuild。其内容如下所示：

```
"response":"Webhook received and build started: https://us-
east-1.console.aws.amazon.com/codebuild/home..."
"statusCode":200
```

7. 导航到 Bitbucket 拉取请求页面以查看构建的状态。

运行以下内容的 GitHub 企业服务器示例 CodeBuild

AWS CodeBuild 支持 GitHub 企业服务器作为源存储库。此示例显示了在 GitHub 企业服务器存储库安装了证书后如何设置 CodeBuild 项目。它还展示了如何启用 webhook，以便每次将代码更改推送到 GitHub 企业服务器存储库时都 CodeBuild 重建源代码。

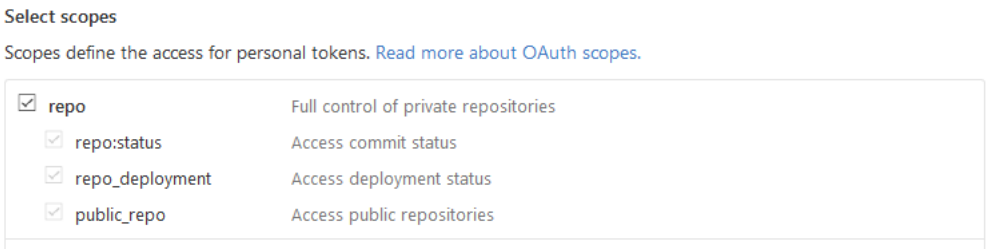
主题

- 先决条件
- 第 1 步：使用 E GitHub nterprise Server 创建构建项目并启用 webhook

先决条件

1. 为您的 CodeBuild 项目生成个人访问令牌。我们建议您创建 GitHub 企业用户并为该用户生成个人访问令牌。将其复制到剪贴板，以便在创建 CodeBuild 项目时使用。有关更多信息，请参阅 GitHub 帮助网站上的为命令行创建个人访问令牌。

   在创建个人访问令牌时，请在定义中包含存储库范围。



2. 从 GitHub 企业服务器下载您的证书。 CodeBuild 使用证书与存储库建立可信的 SSL 连接。

   Linux/macOS 客户：

   从终端窗口中运行以下命令：

```
echo -n | openssl s_client -connect HOST:PORTNUMBER \
    | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > /folder/filename.pem
```

   将命令中的占位符替换为以下值：

   HOST。您的 GitHub 企业服务器存储库的 IP 地址。

   PORTNUMBER。您用于连接的端口号（例如，443）。

   folder。您下载证书的文件夹。

filename。您的证书文件的文件名。

> **Important**
>
> 将证书另存为 .pem 文件。

Windows 客户端：

使用浏览器从 GitHub 企业服务器下载证书。要查看站点的证书详细信息，请选择挂锁图标。有关如何导出证书的信息，请参阅浏览器文档。

> **Important**
>
> 将证书另存为 .pem 文件。

3. 将您的证书文件上传到 S3 存储桶。有关如何创建 S3 存储桶的信息，请参阅如何创建 S3 存储桶？有关如何将对象上传到 S3 存储桶的信息，请参阅如何将文件和文件夹上传至存储桶？

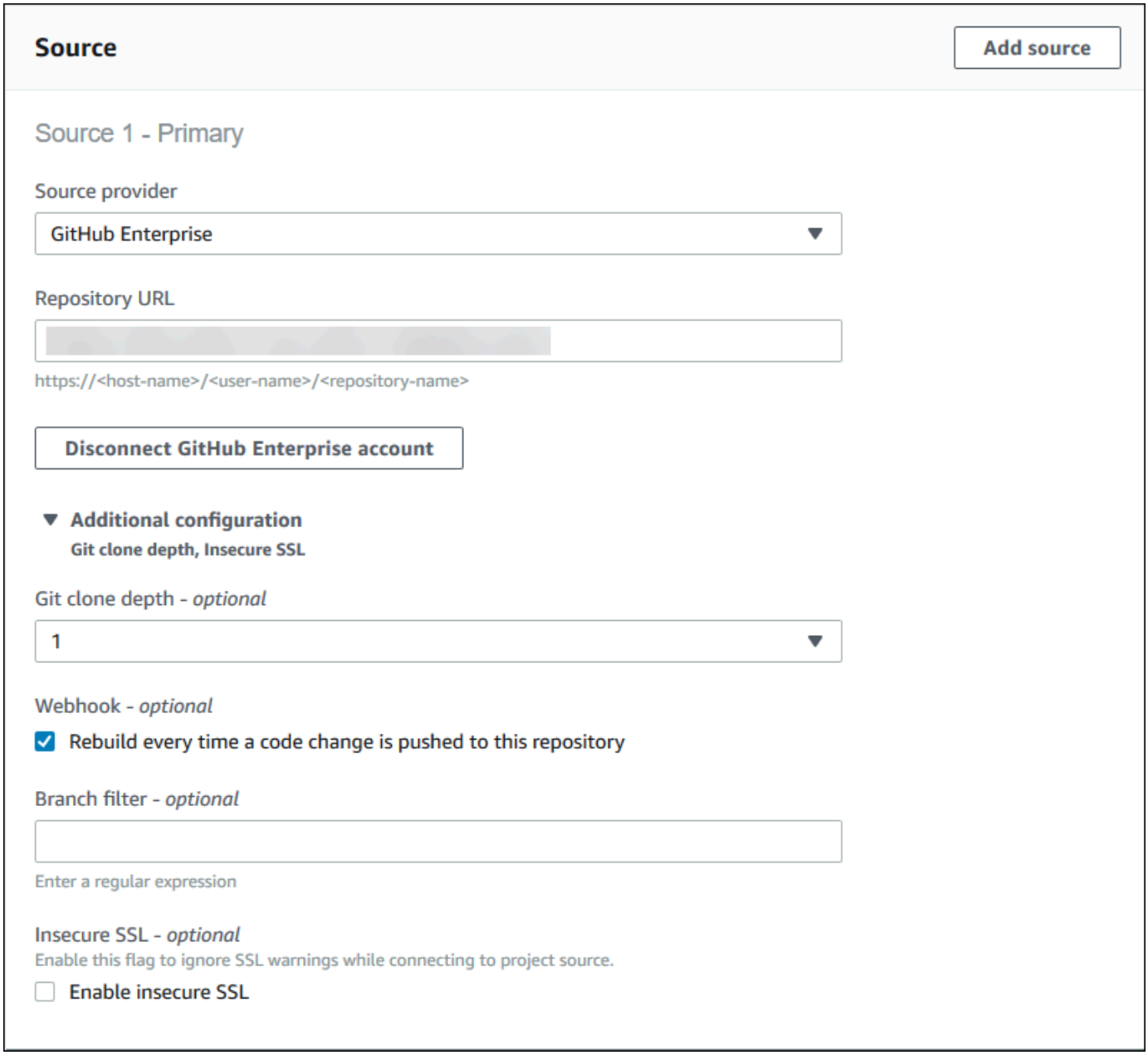
> **Note**
>
> 此存储桶必须与您的版本位于同一 AWS 区域。例如，如果您指示 CodeBuild 在美国东部（俄亥俄州）区域运行构建，则存储桶必须位于美国东部（俄亥俄州）区域。

第 1 步：使用 E GitHub nterprise Server 创建构建项目并启用 webhook

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 如果显示 CodeBuild 信息页面，请选择创建构建项目。否则，请在导航窗格中，展开构建，选择构建项目，然后选择创建构建项目。
3. 在项目名称中，输入此构建项目的名称。每个 AWS 账户的构建项目名称必须是唯一的。您还可以包含构建项目的可选描述，以帮助其他用户了解此项目的用途。
4. 在源代码中，在源提供商中，选择GitHub 企业服务器。
   - 选择管理账户凭证，然后选择个人访问令牌。对于服务，请选择 Secrets Manager（推荐），然后配置您的密钥。然后在GitHub 企业个人访问令牌中，输入您的个人访问令牌并选择保存。
   - 在存储库 URL 中，输入您的存储库的路径，包括存储库的名称。

- 展开其他配置。
- 选择每次将代码推送到此存储库时都会重建以便每次将代码推送到此存储库时进行重建。
- 在连接到 GitHub 企业服务器项目存储库时，选择 “启用不安全 SSL” 以忽略 SSL 警告。

> **Note**
>
> 建议您仅将启用不安全的 SSL 用于测试。它不应在生产环境中使用。



5. 在环境中：

   对于环境映像，执行下列操作之一：

   - 要使用由管理的 Docker 镜像 AWS CodeBuild，请选择“托管映像”，然后从操作系统、运行时、映像和映像版本中进行选择。从环境类型中进行选择（如果可用）。
   - 要使用其他 Docker 映像，请选择自定义映像。对于环境类型，请选择 ARM、Linux、Linux GPU 或 Windows。如果您针对外部注册表 URL 选择其他注册表，请使用 *docker repository*/*docker image name* 格式在 Docker Hub 中输入 Docker 映像的名称和标签。如果您选择亚马逊 ECR，请使用亚马逊 ECR 存储库和亚马逊 ECR 镜像在您的账户中选择 Docker 镜像。 AWS
   - 要使用私有 Docker 映像，请选择自定义映像。对于环境类型，请选择 ARM、Linux、Linux GPU 或 Windows。对于映像注册表，选择其他注册表，然后输入您的私有 Docker 映像的凭证的 ARN。凭证必须由 Secrets Manager 创建。有关更多信息，请参阅《AWS Secrets Manager 用户指南》中的什么是 AWS Secrets Manager？。

6. 在服务角色中，执行下列操作之一：

   - 如果您没有 CodeBuild 服务角色，请选择新建服务角色。在角色名称中，为新角色输入名称。
   - 如果您有 CodeBuild 服务角色，请选择现有服务角色。在角色 ARN 中，选择服务角色。

   > **Note**
   >
   > 当您使用控制台创建或更新构建项目时，可以同时创建 CodeBuild 服务角色。默认情况下，这个角色仅能与该构建项目配合使用。如果您使用控制台将此服务角色与另一个构建项目关联，则此角色将更新以便与关联的构建项目结合使用。一个服务角色最多可与 10 个构建项目结合使用。

7. 展开其他配置。

   如果你 CodeBuild 想使用你的 VPC：

   - 对于 VPC，选择 CodeBuild 使用的 VPC ID。
   - 对于 VPC 子网，选择包含使用的 CodeBuild 资源的子网。
   - 对于 VPC 安全组，选择 CodeBuild 用于允许访问 VPC 中资源的安全组。

   有关更多信息，请参阅 AWS CodeBuild 与亚马逊 Virtual Private Cloud 配合使用。

8. 在 Buildspec 中，执行以下操作之一：

   - 选择使用 buildspec 文件，以在源代码根目录中使用 buildspec.yml 文件。
   - 选择插入构建命令，以使用控制台插入构建命令。

   有关更多信息，请参阅Buildspec 参考。

9. 在构件中，对于类型，执行以下操作之一：

   - 如果您不想创建构建输出构件，请选择无构件。
   - 要将构建输出存储在 S3 存储桶中，请选择 Amazon S3，然后执行以下操作：
     - 如果要将项目名称用于构建输出 ZIP 文件或文件夹，请将名称留空。否则，请输入名称。默认情况下，构件名称是项目名称。如果您要使用其他名称，请在构件名称框中输入该名称。如果您要输出 ZIP 文件，请包含 zip 扩展名。
     - 对于存储桶名称，请选择输出存储桶的名称。
     - 如果您在此过程的前面部分选择了插入构建命令，对于输出文件，请输入构建（该构建要放到构建输出 ZIP 文件或文件夹中）中的文件位置。对于多个位置，使用逗号将各个位置隔开（例如，`appspec.yml, target/my-app.jar`）。有关更多信息，请参阅buildspec 语法中 `files` 的描述。

10. 对于缓存类型，请选择下列选项之一：

    - 如果您不想使用缓存，请选择无缓存。
    - 如果要使用 Amazon S3 缓存，请选择 Amazon S3，然后执行以下操作：
      - 对于存储桶，选择存储缓存的 S3 存储桶的名称。
      - （可选）对于缓存路径前缀，输入 Amazon S3 路径前缀。缓存路径前缀值类似于目录名称。它使您能够在存储桶的同一目录下存储缓存。

        > **Important**
        >
        > 请勿将尾部斜杠 (/) 附加到路径前缀后面。

    - 如果想要使用本地缓存，请选择本地，然后选择一个或多个本地缓存模式。

> **Note**
>
> Docker 层缓存模式仅适用于 Linux。如果您选择该模式，您的项目必须在特权模式下运行。

使用缓存可节省大量构建时间，因为构建环境的可重用部分被存储在缓存中，并且可跨构建使用。有关在 buildspec 文件中指定缓存的信息，请参阅buildspec 语法。有关缓存的更多信息，请参阅缓存构建以提高性能。

11. 选择 Create build project（创建构建项目）。在构建项目页面上，选择开始构建。

运行 GitHub 拉取请求和 webhook 过滤器示例 CodeBuild

AWS CodeBuild 当源存储库为时，支持 webhook。 GitHub这意味着，对于源代码存储在存储 GitHub 库中的 CodeBuild 构建项目，每次将代码更改推送到存储库时，都可以使用 webhook 来重建源代码。有关 CodeBuild 示例，请参阅AWS CodeBuild 示例。

> **Note**
>
> 使用 webhook 时，用户可能会触发意外构建。要降低这种风险，请参阅使用 Webhook 的最佳实操。

主题

- 第 1 步：使用 GitHub 并启用 webhook 创建构建项目
- 步骤 2：确认已启用 webhook

第 1 步：使用 GitHub 并启用 webhook 创建构建项目

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 如果显示 CodeBuild 信息页面，请选择 “创建构建项目”。否则，请在导航窗格中，展开构建，选择构建项目，然后选择创建构建项目。
3. 选择创建构建项目。
4. 在项目配置中：

项目名称

输入此构建项目的名称。每个 AWS 账户中的构建项目名称必须是唯一的。您还可以包含构建项目的可选描述，以帮助其他用户了解此项目的用途。

5. 在源中：

源提供商

选择 GitHub。按照说明进行连接（或重新连接），GitHub 然后选择“授权”。

存储库

在我的 GitHub账户中选择“存储库”。

GitHub repository

输入 GitHub 存储库的 URL。

6. 在主要源 webhook 事件中，选择以下内容。

> **Note**
>
> 只有在上一步中选择了我的 GitHub 账户中的“存储库”时，“主要来源 webhook 事件”部分才可见。

1. 创建项目时，选择每次将代码更改推送到此存储库时都会重新构建。
2. 从事件类型中，选择一个或多个事件。
3. 要在事件触发构建时进行筛选，请在在这些条件下开始构建下，添加一个或多个可选筛选条件。
4. 要在未触发事件时进行筛选，请在在这些条件下不开始构建下，添加一个或多个可选筛选条件。
5. 选择添加筛选条件组，以添加另一个筛选条件组（如果需要）。

有关 GitHub webhook 事件类型和过滤器的更多信息，请参阅GitHub webhook 事件。

7. 在环境中：

环境映像

选择下列选项之一：

要使用由 AWS CodeBuild以下人员管理的 Docker 镜像，请执行以下操作：

选择托管映像，然后选择操作系统、运行时、映像和映像版本。从环境类型中进行选择（如果可用）。

要使用其他 Docker 映像：

选择自定义映像。对于环境类型，请选择 ARM、Linux、Linux GPU 或 Windows。如果您针对外部注册表 URL 选择其他注册表，请使用 *docker repository*/*docker image name* 格式在 Docker Hub 中输入 Docker 映像的名称和标签。如果您选择 Amazon ECR，请使用 Amazon ECR 存储库和 Amazon ECR 映像在您的 AWS 账户中选择 Docker 映像。

要使用私有 Docker 映像，请执行以下操作：

选择自定义映像。对于环境类型，请选择 ARM、Linux、Linux GPU 或 Windows。对于映像注册表，选择其他注册表，然后输入您的私有 Docker 映像的凭证的 ARN。凭证必须由 Secrets Manager 创建。有关更多信息，请参阅什么是 AWS Secrets Manager？ 在《AWS Secrets Manager 用户指南》中。

服务角色

选择下列选项之一：

- 如果您没有 CodeBuild 服务角色，请选择 “新建服务角色”。在角色名称中，为新角色输入名称。
- 如果您有 CodeBuild 服务角色，请选择现有服务角色。在角色 ARN 中，选择服务角色。

> **Note**
>
> 使用控制台创建或更新构建项目时，可以同时创建 CodeBuild 服务角色。默认情况下，这个角色仅能与该构建项目配合使用。如果您使用控制台将此服务角色与另一个构建项目关联，则此角色将更新以便与关联的构建项目结合使用。一个服务角色最多可与 10 个构建项目结合使用。

8. 在 Buildspec 中，执行以下操作之一：

- 选择使用 buildspec 文件，以在源代码根目录中使用 buildspec.yml 文件。
- 选择插入构建命令，以使用控制台插入构建命令。

有关更多信息，请参阅Buildspec 参考。

9. 在构件中：

   类型

   选择下列选项之一：

   - 如果您不想创建构建输出构件，请选择无构件。
   - 要将构建输出存储在 S3 存储桶中，请选择 Amazon S3，然后执行以下操作：
     - 如果要将项目名称用于构建输出 ZIP 文件或文件夹，请将名称留空。否则，请输入名称。默认情况下，构件名称是项目名称。如果您要使用其他名称，请在构件名称框中输入该名称。如果您要输出 ZIP 文件，请包含 zip 扩展名。
     - 对于存储桶名称，请选择输出存储桶的名称。
     - 如果您在此过程的前面部分选择了插入构建命令，对于输出文件，请输入构建（该构建要放到构建输出 ZIP 文件或文件夹中）中的文件位置。对于多个位置，使用逗号将各个位置隔开（例如，`appspec.yml, target/my-app.jar`）。有关更多信息，请参阅 buildspec 语法中 `files` 的描述。

   其他配置

   展开其他配置并根据需要设置选项。

10. 选择 Create build project（创建构建项目）。在审核页面上，选择开始构建以运行构建。

步骤 2：确认已启用 webhook

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择构建项目。
3. 请执行以下操作之一：

   - 选择带有要验证的 Webhook 的构建项目的链接，然后选择构建详细信息。
   - 选择带有要验证的 Webhook 的构建项目旁边的按钮，选择查看详细信息，然后选择构建详细信息选项卡。

4. 在主要源 Webhook 事件中，选择 Webhook URL 链接。
5. 在存储 GitHub 库中，在 “设置” 页面的 “Webhooks” 下，确认已选中 “拉取请求和推送”。

6. 在您的个人 GitHub 资料设置中，在 “个人设置”、“应用程序”、“已授权的 OAuth 应用程序” 下，您应该会看到您的应用程序已被授权访问您选择的 AWS 区域。

教程： CodeBuild 使用 S3 进行证书存储时使用 Fastlane 进行 Apple 代码签名

fastlane 是一款流行的开源自动化工具，可自动部署和发布 iOS 和 Android 应用程序的测试版。它可以处理所有繁琐的任务，例如生成屏幕截图、处理代码签名和发布应用程序等。

先决条件

要完成本教程，您首先必须设置以下条件：

- 一个 AWS 账户
- Apple 开发人员账户
- 用于存储证书的 S3 存储桶
- 在您的项目中安装的 fastlane - fastlane 安装指南

步骤 1：在本地计算机上使用 S3 设置 Fastlane Match

Fastlane Match 是 Fastlane 工具之一，它允许在本地开发环境和其他环境中无缝配置代码签名。CodeBuildFastlane Match 将您的所有代码签名证书和配置文件存储在 Git repository/S3 Bucket/ Google 云存储中，并在需要时下载和安装必要的证书和配置文件。

在此示例配置中，您将设置并使用 Amazon S3 存储桶进行存储。

1. 在项目中初始化匹配：

```
fastlane match init
```

2. 出现提示时，选择 S3 作为存储模式。
3. 更新“Matchfile”以使用 S3：

```
storage_mode("s3")
   s3_bucket("your-s3-bucket-name")
   s3_region("your-aws-region")
```

```
  type("appstore") # The default type, can be: appstore, adhoc, enterprise or
development
```

步骤 2：设置 Fastfile

使用以下通道创建或更新您的“Fastfile”。

开启 CodeBuild，每次构建和签署应用程序时都需要运行 Fastlane Match。执行此操作的最简单方法是将 `match` 操作添加到构建应用程序的通道中。

```
default_platform(:ios)

platform :ios do
  before_all do
    setup_ci
  end

  desc "Build and sign the app"
  lane :build do
    match(type: "appstore", readonly: true)
    gym(
      scheme: "YourScheme",
      export_method: "app-store"
    )
  end
end
```

> **Note**
>
> 请务必将 `setup_ci` 添加至 `Fastfile` 中的 `before_all` 部分，以使匹配操作正常运行。这样可以确保使用具有适当权限的临时 Fastlane 密钥链。如果不使用它，您可能会看到构建失败或结果不一致。

第 3 步：运行 **`fastlane`** match 命令以生成相应的证书和配置文件

给定类型（即开发、应用商店、临时、企业）的 fastlane match 命令将生成证书和配置文件（如果远程存储中未提供）。fastlane 将证书和配置文件存储在 S3 中。

```
bundle exec fastlane match appstore
```

命令执行将是交互式的，fastlane 将要求设置密码短语来解密证书。

步骤 4：为项目创建应用程序文件

根据项目的需要创建或添加应用程序文件。

1. 根据项目构建要求创建或添加 Gymfile、Appfile、Snapfile、Deliverfile。
2. 将更改提交到远程存储库

步骤 5：在 Secrets Manager 中创建环境变量

创建两个用于存储 fastlane 会话 cookie 和匹配密码短语的密钥。有关在 Secrets Manager 中创建密钥的更多信息，请参阅创建 AWS Secrets Manager 密钥。

1. 按如下方式访问 fastlane 会话 cookie。
   a. 秘密密钥：FASTLANE_SESSION
   b. 密钥值：在本地计算机上运行以下命令时生成的会话 cookie。

   > **Note**
   > 完成身份验证后，此值会保存在本地文件 ~/.fastlane/spaceship/my_appleid_username/cookie 中。

   ```
   fastlane spaceauth -u <apple account>
   ```

2. Fastlane 匹配密码-要使 Fastlane Match 能够解密存储在 S3 存储桶中的证书和配置文件，必须将您在匹配设置步骤中配置的加密密码添加到项目的环境变量中。 CodeBuild
   a. 秘密密钥：MATCH_PASSWORD
   b. 密钥值：*<match passphrase to decrypt certificates>*。密码短语是在步骤 3 中生成证书时设置的。

> **Note**
>
> 在 Secrets Manager 中创建上述密钥时，请记得提供一个带有以下前缀的密钥名称：`/CodeBuild/`

步骤 6：创建计算实例集

为您的项目创建计算实例集。

1. 在控制台中，转到 CodeBuild 并创建新的计算队列。
2. 选择“macOS”作为操作系统，然后选择适当的计算类型和映像。

步骤 7：在中创建项目 CodeBuild

在中创建您的项目 CodeBuild。

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 创建构建项目。有关信息，请参阅创建构建项目（控制台）和运行构建（控制台）。
3. 设置您的源提供商（例如 GitHub、 CodeCommit）。这是 iOS 项目源存储库，而不是证书存储库。
4. 在环境中：
   - 选择预留容量。
   - 对于实例集，选择上面创建的实例集。
   - 提供 CodeBuild 将为您创建的服务角色的名称。
   - 提供以下环境变量。
     - 名称:MATCH_PASSWORD，值：*<secrets arn>*，类型：密钥管理器（在步骤 5 中为 MATCH_PASSWORD 创建的密钥 ARN）
     - 名称:FASTLANE_SESSION，值：*<secrets arn>*，类型：密钥管理器（在步骤 5 中为 FASTLANE_SESSION 创建的密钥 ARN）
5. 在 Buildspec 中，添加以下内容：

```
version: 0.2
```

```
phases:
  install:
    commands:
      - gem install bundler
      - bundle install
  build:
    commands:
      - echo "Building and signing the app..."
      - bundle exec fastlane build
  post_build:
    commands:
      - echo "Build completed on date"

artifacts:
  files:
    - '*/.ipa'
  name: app-$(date +%Y-%m-%d)
```

步骤 8：配置 IAM 角色

创建项目后，请确保 CodeBuild 项目的服务角色有权访问包含证书的 S3 存储桶。将下面的策略附加到该角色：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::your-s3-bucket-name"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
```

```
                "s3:PutObject",
                "s3:DeleteObject"
            ],
            "Resource": "arn:aws:s3:::your-s3-bucket-name/*"
        }
    ]
}
```

第 9 步：运行构建

运行构建。您可以查看构建状态并登录 CodeBuild。

作业完成后，您将能够查看该作业的日志。

问题排查

- 如果您在获取证书时遇到问题，请确保您的 IAM 权限设置正确，以便能够访问 S3。
- 如果您在证书解密时遇到问题，请确保在 MATCH_PASSWORD 环境变量中设置了正确的密码短语。
- 对于代码签名问题，请验证您的 Apple 开发人员账户是否具有必要的证书和配置文件，并且 Xcode 项目中的捆绑包标识符是否与预调配配置文件中的捆绑包标识符匹配。

安全注意事项

以下是本教程的安全注意事项。

- 确保您的 S3 存储桶具有适当的安全设置，包括静态加密。特别是，请确保存储分区没有公共访问权限， CodeBuild 并仅限需要访问权限的系统。
- 考虑使用来 AWS Secrets Manager 存储 MATCH_PASSWORD 和 FASTLANE_SESSION 等敏感信息。

此示例提供了 CodeBuild 使用 Amazon S3 进行证书存储时使用 Fastlane 进行 iOS 代码签名的设置。您可能需要根据您的特定项目要求和 CodeBuild 环境调整一些步骤。这种方法利用 AWS 服务来增强安全性和 AWS 生态系统内的集成。

教程： CodeBuild 使用 Fastlane 进行 Apple 代码签名 GitHub 用于证书存储

fastlane 是一款流行的开源自动化工具，可自动部署和发布 iOS 和 Android 应用程序的测试版。它可以处理所有繁琐的任务，例如生成屏幕截图、处理代码签名和发布应用程序等。

此示例演示如何在 Mac 机群上运行的 CodeBuild项目中使用 Fastlane 设置 Apple 代码签名，并用 GitHub 作证书和配置文件的存储空间。

先决条件

要完成本教程，您首先必须设置以下条件：

- 一个 AWS 账户
- Apple 开发人员账户
- 用于 GitHub 存储证书的私有存储库
- 在您的项目中安装的 fastlane - fastlane 安装指南

第 1 步：在本地计算机 GitHub 上设置 Fastlane Match with

Fastlane Match 是 Fastlane 工具之一，它允许在本地开发环境和其他环境中无缝配置代码签名。CodeBuildFastlane Match 将您的所有代码签名证书和配置文件存储在 Git repository/S3 Bucket/ Google 云存储中，并在需要时下载和安装必要的证书和配置文件。

在此示例配置中，我们设置并使用 Git 存储库进行存储。

1. 在项目中初始化匹配：

```
fastlane match init
```

2. 出现提示时，选择 GitHub 作为存储模式。
3. 更新你的 `Match  file`以使用 GitHub：

```
git_url("https://github.com/your-username/your-certificate-repo.git")
storage_mode("git")
type("development") # The default type, can be: appstore, adhoc, enterprise or
 development
```

> **Note**
>
> 请务必输入 Git 存储库的 HTTPS URL，以便 fastlane 成功进行身份验证和克隆。否则，当您尝试使用匹配时，可能会看到身份验证错误。

步骤 2：设置 Fastfile

使用以下通道创建或更新您的“Fastfile”。

开启 CodeBuild，每次构建和签署应用程序时都需要运行 Fastlane Match。执行此操作的最简单方法是将 `match` 操作添加到构建应用程序的通道中。

```
default_platform(:ios)

platform :ios do
  before_all do
    setup_ci
  end

  desc "Build and sign the app"
  lane :build do
    match(type: "appstore", readonly: true)
    gym(
      scheme: "YourScheme",
      export_method: "app-store"
    )
  end
end
```

> **Note**
>
> 请务必将 `setup_ci` 添加至 `Fastfile` 中的 `before_all` 部分，以使匹配操作正常运行。这样可以确保使用具有适当权限的临时 Fastlane 密钥链。如果不使用它，您可能会看到构建失败或结果不一致。

第 3 步：运行 **`fastlane`** match 命令以生成相应的证书和配置文件

给定类型（即开发、应用商店、临时、企业）的 fastlane match 命令将生成证书和配置文件（如果远程存储中未提供）。证书和配置文件将 GitHub 由 fastlane 存储在。

```
bundle exec fastlane match appstore
```

命令执行将是交互式的，fastlane 将要求设置密码短语来解密证书。

步骤 4：为项目创建应用程序文件

根据项目的需要创建或添加应用程序文件。

1. 根据项目构建要求创建或添加 Gymfile、Appfile、Snapfile、Deliverfile。
2. 将更改提交到远程存储库。

步骤 5：在 Secrets Manager 中创建环境变量

创建三个用于存储 fastlane 会话 cookie 和匹配密码短语的密钥。有关在 Secrets Manager 中创建密钥的更多信息，请参阅创建 AWS Secrets Manager 密钥。

1. 按如下方式访问 fastlane 会话 cookie。
   a. 秘密密钥：FASTLANE_SESSION
   b. 密钥值：在本地计算机上运行以下命令时生成的会话 cookie。

   > **Note**
   > 完成身份验证后，此值会保存在本地文件 ~/.fastlane/spaceship/my_appleid_username/cookie 中。

   ```
   fastlane spaceauth -u <Apple_account>
   ```

2. Fastlane 匹配密码-要使 Fastlane Match 能够解密存储在 Git 存储库中的证书和配置文件，必须将您在匹配设置步骤中配置的加密密码添加到项目的环境变量中。 CodeBuild
   a. 秘密密钥：MATCH_PASSWORD
   b. 密钥值：<match passphrase to decrypt certificates>。密码短语是在步骤 3 中生成证书时设置的。
3. Fastlane MATCH_GIT_BASIC_AUTHORIZATION：为匹配设置基本授权：

a. 秘密密钥：

   `MATCH_GIT_BASIC_AUTHORIZATION`

b. 密钥值：该值应是您的用户名和个人访问令牌（PAT）的 base64 编码字符串，格式为 `username:password`。您可以使用以下命令生成它：

```
echo -n your_github_username:your_personal_access_token | base64
```

   您可以在 GitHub 控制台上的 “我的个人资料” > “设置” > “开发者设置” > “个人访问令牌” 中生成 PAT 。有关更多信息，请参阅以下指南：https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens。

> **Note**
>
> 在 Secrets Manager 中创建上述密钥时，请记得提供一个带有以下前缀的密钥名称：`/CodeBuild/`

步骤 6：创建计算实例集

为您的项目创建计算实例集。

1. 在控制台中，转到 CodeBuild 并创建新的计算队列。
2. 选择 `macOS` 作为操作系统，然后选择适当的计算类型和映像。

步骤 7：在中创建项目 CodeBuild

在中创建您的项目 CodeBuild。

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 创建构建项目。有关信息，请参阅创建构建项目（控制台）和运行构建（控制台）。
3. 设置您的源提供商（例如 GitHub、 CodeCommit）。这是 iOS 项目源存储库，而不是证书存储库。
4. 在环境中：

- 选择预留容量。
- 对于实例集，选择上面创建的实例集。
- 提供 CodeBuild 将为您创建的服务角色的名称。
- 提供以下环境变量。
  - 名称:MATCH_PASSWORD，值：*<secrets arn>*，类型：密钥管理器（在步骤 5 中为 MATCH_PASSWORD 创建的密钥 ARN）
  - 名称:FASTLANE_SESSION，值：*<secrets arn>*，类型：密钥管理器（在步骤 5 中为 FASTLANE_SESSION 创建的密钥 ARN）
  - 名称:MATCH_GIT_BASIC_AUTHORIZATION，值：*<secrets ARN>*，类型：密钥管理器密钥 ARN（在步骤 5 中创建）MATCH_GIT_BASIC_AUTHORIZATION

5. 在 Buildspec 中，添加以下内容：

```
version: 0.2

phases:
  install:
    commands:
      - gem install bundler
      - bundle install
  build:
    commands:
      - echo "Building and signing the app..."
      - bundle exec fastlane build
  post_build:
    commands:
      - echo "Build completed on date"

artifacts:
  files:
    - '*/.ipa'
  name: app-$(date +%Y-%m-%d)
```

第 8 步：运行构建

运行构建。您可以查看构建状态并登录 CodeBuild。

作业完成后，您将能够查看该作业的日志。

问题排查

- 如果您在访问 GitHub 存储库时遇到问题，请仔细检查您的个人访问令牌和 MATCH_GIT_BASIC_AUTHORIZATION 环境变量。
- 如果您在证书解密时遇到问题，请确保在 MATCH_PASSWORD 环境变量中设置了正确的密码短语。
- 对于代码签名问题，请验证您的 Apple 开发人员账户是否具有必要的证书和配置文件，并且 Xcode 项目中的捆绑包标识符是否与预调配配置文件中的捆绑包标识符匹配。

安全注意事项

以下是本教程的安全注意事项。

- 将您的证书 GitHub 存储库保密，并定期审核访问权限。
- 考虑使用来 AWS Secrets Manager 存储 MATCH_PASSWORD 和 FASTLANE_SESSION 等敏感信息。

此示例提供了 CodeBuild 使用 GitHub Fastlane 进行证书存储的 iOS 代码签名的设置。您可能需要根据您的特定项目要求和 CodeBuild 环境调整一些步骤。这种方法利用 AWS 服务来增强安全性和 AWS 生态系统内的集成。

使用语义版本控制在构建时设置构件名称

此示例包含示例 buildspec 文件，演示如何指定在构建时创建的构件名称。在 buildspec 文件中指定的名称可以包含 Shell 命令和环境变量，以使其保持唯一。在 buildspec 文件中指定的名称将覆盖创建项目时在控制台中输入的名称。

如果构建多次，则使用在 buildspec 文件中指定的构件名称可以确保输出构件文件名的唯一性。例如，您可以使用在构建时插入构件名称的日期和时间戳。

要使用 buildspec 文件中的构件名称覆盖在控制台中输入的构件名称，请执行以下操作：

1. 设置构建项目以使用 buildspec 文件中的构件名称覆盖相应的构件名称。

   - 如果您使用控制台创建您的构建项目，请选择启用语义版本控制。有关更多信息，请参阅 创建构建项目（控制台）。
   - 如果使用 AWS CLI，请在传递`overrideArtifactName`给 JSON-formatted 的文件中将设置为 true `create-project`。有关更多信息，请参阅 创建构建项目 (AWS CLI)。

- 如果您使用 AWS CodeBuild API，请在创建或更新项目或开始构建时在`ProjectArtifacts`对象上设置`overrideArtifactName`标志。

2. 在 buildspec 文件中指定名称。使用以下示例 buildspec 文件作为指南。

此 Linux 示例演示如何指定包含构建创建日期的构件名称：

```
version: 0.2
phases:
  build:
    commands:
      - rspec HelloWorld_spec.rb
artifacts:
  files:
    - '**/*'
  name: myname-$(date +%Y-%m-%d)
```

此 Linux 示例演示如何指定使用 CodeBuild 环境变量的构件名称。有关更多信息，请参阅 [构建环境中的环境变量](#)。

```
version: 0.2
phases:
  build:
    commands:
      - rspec HelloWorld_spec.rb
artifacts:
  files:
    - '**/*'
  name: myname-$AWS_REGION
```

此 Windows 示例演示如何指定包含构建创建日期和时间的构件名称：

```
version: 0.2
env:
  variables:
    TEST_ENV_VARIABLE: myArtifactName
phases:
  build:
    commands:
      - cd samples/helloworld
      - dotnet restore
```

```
      - dotnet run
artifacts:
  files:
    - '**/*'
  name: $Env:TEST_ENV_VARIABLE-$(Get-Date -UFormat "%Y%m%d-%H%M%S")
```

此 Windows 示例向您展示了如何使用在 buildspec 文件中声明的变量和 CodeBuild 环境变量来指定构件名称。有关更多信息，请参阅 构建环境中的环境变量。

```
version: 0.2
env:
  variables:
    TEST_ENV_VARIABLE: myArtifactName
phases:
  build:
    commands:
      - cd samples/helloworld
      - dotnet restore
      - dotnet run
artifacts:
  files:
    - '**/*'
  name: $Env:TEST_ENV_VARIABLE-$Env:AWS_REGION
```

有关更多信息，请参阅 的构建规范参考 CodeBuild。

运行微软 Windows 示例 CodeBuild

这些示例使用运行微软 Windows Server 2019、.NET Framework 和.NET Core SDK 的 AWS CodeBuild 构建环境，使用用 F# 和 Visual Basic 编写的代码构建运行时文件。

> **Important**
>
> 运行这些样本可能会导致您的 AWS 账户被扣款。其中包括与 Amazon S3 和 CloudWatch 日志相关的 AWS 资源和操作可能产生的费用。 CodeBuild AWS KMS有关更多信息，请参阅CodeBuild定价、Amazon S3 定价、AWS Key Management Service 定价和亚马逊 CloudWatch 定价。

运行 Windows 示例

按照以下过程运行 Windows 示例。

运行 Windows 示例

1. 按照本主题目录结构和文件部分所述创建文件，然后将其上传到 S3 输入存储桶或 CodeCommit 或 GitHub 存储库。

   > **Important**
   >
   > 请不要上传 *(root directory name)*，而只上传 *(root directory name)* 中的文件。
   > 如果您使用的是 S3 输入存储桶，请务必创建一个包含这些文件的 ZIP 文件，然后将其上传到输入存储桶。请不要将 *(root directory name)* 添加到 ZIP 文件中，而只添加 *(root directory name)* 中的文件。

2. 创建构建项目。构建项目必须使用该 `mcr.microsoft.com/dotnet/framework/sdk:4.8` 映像来构建 .NET Framework 项目。

   如果您使用 AWS CLI 来创建构建项目，则`create-project`命令的 JSON-formatted 输入可能与此类似。(请将占位符替换为您自己的值。)

   ```
   {
     "name": "sample-windows-build-project",
   ```

```
  "source": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-bucket/windows-build-input-
artifact.zip"
  },
  "artifacts": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-output-bucket",
    "packaging": "ZIP",
    "name": "windows-build-output-artifact.zip"
  },
  "environment": {
    "type": "WINDOWS_SERVER_2019_CONTAINER",
    "image": "mcr.microsoft.com/dotnet/framework/sdk:4.8",
    "computeType": "BUILD_GENERAL1_MEDIUM"
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name",
  "encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

3. 运行构建，然后按照 手动运行构建 中的步骤操作。
4. 要获取构建输出构件，请在您的 S3 输出存储桶中，将 *windows-build-output-artifact*.zip 文件下载到您的本地计算机或实例。提取内容以获取运行时和其他文件。

   - 在 FSharpHelloWorld\bin\Debug 目录中可以找到使用 .NET Framework 的 F# 示例的运行时文件 FSharpHelloWorld.exe。
   - 在 VBHelloWorld\bin\Debug 目录中可以找到使用 .NET Framework 的 Visual Basic 示例的运行时文件 VBHelloWorld.exe。

目录结构

这些示例采用以下目录结构。

F# 和 .NET Framework

```
(root directory name)
### buildspec.yml
### FSharpHelloWorld.sln
### FSharpHelloWorld
    ### App.config
    ### AssemblyInfo.fs
```

```
    ### FSharpHelloWorld.fsproj
    ### Program.fs
```

Visual Basic 和 .NET Framework

```
(root directory name)
### buildspec.yml
### VBHelloWorld.sln
### VBHelloWorld
    ### App.config
    ### HelloWorld.vb
    ### VBHelloWorld.vbproj
    ### My Project
        ### Application.Designer.vb
        ### Application.myapp
        ### AssemblyInfo.vb
        ### Resources.Designer.vb
        ### Resources.resx
        ### Settings.Designer.vb
        ### Settings.settings
```

文件

这些示例使用以下文件。

F# 和 .NET Framework

`buildspec.yml`（在 *`(root directory name)`*）：

```
version: 0.2

env:
  variables:
    SOLUTION: .\FSharpHelloWorld.sln
    PACKAGE_DIRECTORY: .\packages
    DOTNET_FRAMEWORK: 4.8

phases:
  build:
    commands:
```

```
      - '& nuget restore $env:SOLUTION -PackagesDirectory $env:PACKAGE_DIRECTORY'
      - '& msbuild -p:FrameworkPathOverride="C:\Program Files (x86)\Reference
 Assemblies\Microsoft\Framework\.NETFramework\v$env:DOTNET_FRAMEWORK" $env:SOLUTION'
artifacts:
  files:
    - .\FSharpHelloWorld\bin\Debug\*
```

FSharpHelloWorld.sln（在 *(root directory name)*）：

```
Microsoft Visual Studio Solution File, Format Version 12.00
# Visual Studio 14
VisualStudioVersion = 14.0.25420.1
MinimumVisualStudioVersion = 10.0.40219.1
Project("{F2A71F9B-5D33-465A-A702-920D77279786}") = "FSharpHelloWorld",
 "FSharpHelloWorld\FSharpHelloWorld.fsproj", "{D60939B6-526D-43F4-9A89-577B2980DF62}"
EndProject
Global
  GlobalSection(SolutionConfigurationPlatforms) = preSolution
    Debug|Any CPU = Debug|Any CPU
    Release|Any CPU = Release|Any CPU
  EndGlobalSection
  GlobalSection(ProjectConfigurationPlatforms) = postSolution
    {D60939B6-526D-43F4-9A89-577B2980DF62}.Debug|Any CPU.ActiveCfg = Debug|Any CPU
    {D60939B6-526D-43F4-9A89-577B2980DF62}.Debug|Any CPU.Build.0 = Debug|Any CPU
    {D60939B6-526D-43F4-9A89-577B2980DF62}.Release|Any CPU.ActiveCfg = Release|Any CPU
    {D60939B6-526D-43F4-9A89-577B2980DF62}.Release|Any CPU.Build.0 = Release|Any CPU
  EndGlobalSection
  GlobalSection(SolutionProperties) = preSolution
    HideSolutionNode = FALSE
  EndGlobalSection
EndGlobal
```

App.config（在 *(root directory name)*\FSharpHelloWorld）：

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.8" />
  </startup>
</configuration>
```

AssemblyInfo.fs（在 *(root directory name)*\FSharpHelloWorld）：

```
namespace FSharpHelloWorld.AssemblyInfo

open System.Reflection
open System.Runtime.CompilerServices
open System.Runtime.InteropServices

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[<assembly: AssemblyTitle("FSharpHelloWorld")>]
[<assembly: AssemblyDescription("")>]
[<assembly: AssemblyConfiguration("")>]
[<assembly: AssemblyCompany("")>]
[<assembly: AssemblyProduct("FSharpHelloWorld")>]
[<assembly: AssemblyCopyright("Copyright ©  2017")>]
[<assembly: AssemblyTrademark("")>]
[<assembly: AssemblyCulture("")>]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[<assembly: ComVisible(false)>]

// The following GUID is for the ID of the typelib if this project is exposed to COM
[<assembly: Guid("d60939b6-526d-43f4-9a89-577b2980df62")>]

// Version information for an assembly consists of the following four values:
//
// Major Version
// Minor Version
// Build Number
// Revision
//
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:
// [<assembly: AssemblyVersion("1.0.*")>]
[<assembly: AssemblyVersion("1.0.0.0")>]
[<assembly: AssemblyFileVersion("1.0.0.0")>]

do
  ()
```

FSharpHelloWorld.fsproj（在 *(root directory name)*\FSharpHelloWorld）：

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="14.0" DefaultTargets="Build" xmlns="http://
schemas.microsoft.com/developer/msbuild/2003">
  <Import Project="$(MSBuildExtensionsPath)\
$(MSBuildToolsVersion)\Microsoft.Common.props"
 Condition="Exists('$(MSBuildExtensionsPath)\
$(MSBuildToolsVersion)\Microsoft.Common.props')" />
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <SchemaVersion>2.0</SchemaVersion>
    <ProjectGuid>d60939b6-526d-43f4-9a89-577b2980df62</ProjectGuid>
    <OutputType>Exe</OutputType>
    <RootNamespace>FSharpHelloWorld</RootNamespace>
    <AssemblyName>FSharpHelloWorld</AssemblyName>
    <TargetFrameworkVersion>v4.8</TargetFrameworkVersion>
    <AutoGenerateBindingRedirects>true</AutoGenerateBindingRedirects>
    <TargetFSharpCoreVersion>4.4.0.0</TargetFSharpCoreVersion>
    <Name>FSharpHelloWorld</Name>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>false</Optimize>
    <Tailcalls>false</Tailcalls>
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <WarningLevel>3</WarningLevel>
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DocumentationFile>bin\Debug\FSharpHelloWorld.XML</DocumentationFile>
    <Prefer32Bit>true</Prefer32Bit>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <Tailcalls>true</Tailcalls>
    <OutputPath>bin\Release\</OutputPath>
    <DefineConstants>TRACE</DefineConstants>
    <WarningLevel>3</WarningLevel>
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DocumentationFile>bin\Release\FSharpHelloWorld.XML</DocumentationFile>
    <Prefer32Bit>true</Prefer32Bit>
  </PropertyGroup>
```

```
  <ItemGroup>
    <Reference Include="mscorlib" />
    <Reference Include="FSharp.Core, Version=$(TargetFSharpCoreVersion),
 Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a">
      <Private>True</Private>
    </Reference>
    <Reference Include="System" />
    <Reference Include="System.Core" />
    <Reference Include="System.Numerics" />
  </ItemGroup>
  <ItemGroup>
    <Compile Include="AssemblyInfo.fs" />
    <Compile Include="Program.fs" />
    <None Include="App.config" />
  </ItemGroup>
  <PropertyGroup>
    <MinimumVisualStudioVersion Condition="'$(MinimumVisualStudioVersion)' == ''">11</
MinimumVisualStudioVersion>
  </PropertyGroup>
  <Choose>
    <When Condition="'$(VisualStudioVersion)' == '11.0'">
      <PropertyGroup Condition="Exists('$(MSBuildExtensionsPath32)\..\Microsoft SDKs\F#
\3.0\Framework\v4.0\Microsoft.FSharp.Targets')">
        <FSharpTargetsPath>$(MSBuildExtensionsPath32)\..\Microsoft SDKs\F#
\3.0\Framework\v4.0\Microsoft.FSharp.Targets</FSharpTargetsPath>
      </PropertyGroup>
    </When>
    <Otherwise>
      <PropertyGroup Condition="Exists('$(MSBuildExtensionsPath32)\Microsoft
\VisualStudio\v$(VisualStudioVersion)\FSharp\Microsoft.FSharp.Targets')">
        <FSharpTargetsPath>$(MSBuildExtensionsPath32)\Microsoft\VisualStudio\v
$(VisualStudioVersion)\FSharp\Microsoft.FSharp.Targets</FSharpTargetsPath>
      </PropertyGroup>
    </Otherwise>
  </Choose>
  <Import Project="$(FSharpTargetsPath)" />
  <!-- To modify your build process, add your task inside one of the targets below and
 uncomment it.
       Other similar extension points exist, see Microsoft.Common.targets.
  <Target Name="BeforeBuild">
  </Target>
  <Target Name="AfterBuild">
  </Target>
  -->
```

```
</Project>
```

Program.fs（在 *(root directory name)*\FSharpHelloWorld）：

```
// Learn more about F# at http://fsharp.org
// See the 'F# Tutorial' project for more help.

[<EntryPoint>]
let main argv =
  printfn "Hello World"
  0 // return an integer exit code
```

Visual Basic 和 .NET Framework

buildspec.yml（在 *(root directory name)*）：

```
version: 0.2

env:
  variables:
    SOLUTION: .\VBHelloWorld.sln
    PACKAGE_DIRECTORY: .\packages
    DOTNET_FRAMEWORK: 4.8

phases:
  build:
    commands:
      - '& "C:\ProgramData\chocolatey\bin\NuGet.exe" restore $env:SOLUTION -
PackagesDirectory $env:PACKAGE_DIRECTORY'
      - '& "C:\Program Files (x86)\MSBuild\14.0\Bin\MSBuild.exe" -
p:FrameworkPathOverride="C:\Program Files (x86)\Reference Assemblies\Microsoft
\Framework\.NETFramework\v$env:DOTNET_FRAMEWORK" $env:SOLUTION'
artifacts:
  files:
    - .\VBHelloWorld\bin\Debug\*
```

VBHelloWorld.sln（在 *(root directory name)*）：

```
Microsoft Visual Studio Solution File, Format Version 12.00
# Visual Studio 14
VisualStudioVersion = 14.0.25420.1
```

```
MinimumVisualStudioVersion = 10.0.40219.1
Project("{F184B08F-C81C-45F6-A57F-5ABD9991F28F}") = "VBHelloWorld", "VBHelloWorld
\VBHelloWorld.vbproj", "{4DCEC446-7156-4FE6-8CCC-219E34DD409D}"
EndProject
Global
  GlobalSection(SolutionConfigurationPlatforms) = preSolution
    Debug|Any CPU = Debug|Any CPU
    Release|Any CPU = Release|Any CPU
  EndGlobalSection
  GlobalSection(ProjectConfigurationPlatforms) = postSolution
    {4DCEC446-7156-4FE6-8CCC-219E34DD409D}.Debug|Any CPU.ActiveCfg = Debug|Any CPU
    {4DCEC446-7156-4FE6-8CCC-219E34DD409D}.Debug|Any CPU.Build.0 = Debug|Any CPU
    {4DCEC446-7156-4FE6-8CCC-219E34DD409D}.Release|Any CPU.ActiveCfg = Release|Any CPU
    {4DCEC446-7156-4FE6-8CCC-219E34DD409D}.Release|Any CPU.Build.0 = Release|Any CPU
  EndGlobalSection
  GlobalSection(SolutionProperties) = preSolution
    HideSolutionNode = FALSE
  EndGlobalSection
EndGlobal
```

App.config（在 *(root directory name)*\VBHelloWorld）：

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.8" />
  </startup>
</configuration>
```

HelloWorld.vb（在 *(root directory name)*\VBHelloWorld）：

```
Module HelloWorld

  Sub Main()
    MsgBox("Hello World")
  End Sub

End Module
```

VBHelloWorld.vbproj（在 *(root directory name)*\VBHelloWorld）：

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<Project ToolsVersion="14.0" DefaultTargets="Build" xmlns="http://
schemas.microsoft.com/developer/msbuild/2003">
  <Import Project="$(MSBuildExtensionsPath)\
$(MSBuildToolsVersion)\Microsoft.Common.props"
 Condition="Exists('$(MSBuildExtensionsPath)\
$(MSBuildToolsVersion)\Microsoft.Common.props')" />
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProjectGuid>{4DCEC446-7156-4FE6-8CCC-219E34DD409D}</ProjectGuid>
    <OutputType>Exe</OutputType>
    <StartupObject>VBHelloWorld.HelloWorld</StartupObject>
    <RootNamespace>VBHelloWorld</RootNamespace>
    <AssemblyName>VBHelloWorld</AssemblyName>
    <FileAlignment>512</FileAlignment>
    <MyType>Console</MyType>
    <TargetFrameworkVersion>v4.8</TargetFrameworkVersion>
    <AutoGenerateBindingRedirects>true</AutoGenerateBindingRedirects>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <DefineDebug>true</DefineDebug>
    <DefineTrace>true</DefineTrace>
    <OutputPath>bin\Debug\</OutputPath>
    <DocumentationFile>VBHelloWorld.xml</DocumentationFile>
    <NoWarn>42016,41999,42017,42018,42019,42032,42036,42020,42021,42022</NoWarn>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugType>pdbonly</DebugType>
    <DefineDebug>false</DefineDebug>
    <DefineTrace>true</DefineTrace>
    <Optimize>true</Optimize>
    <OutputPath>bin\Release\</OutputPath>
    <DocumentationFile>VBHelloWorld.xml</DocumentationFile>
    <NoWarn>42016,41999,42017,42018,42019,42032,42036,42020,42021,42022</NoWarn>
  </PropertyGroup>
  <PropertyGroup>
    <OptionExplicit>On</OptionExplicit>
  </PropertyGroup>
  <PropertyGroup>
    <OptionCompare>Binary</OptionCompare>
```

```
  </PropertyGroup>
  <PropertyGroup>
    <OptionStrict>Off</OptionStrict>
  </PropertyGroup>
  <PropertyGroup>
    <OptionInfer>On</OptionInfer>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="System" />
    <Reference Include="System.Data" />
    <Reference Include="System.Deployment" />
    <Reference Include="System.Xml" />
    <Reference Include="System.Core" />
    <Reference Include="System.Xml.Linq" />
    <Reference Include="System.Data.DataSetExtensions" />
    <Reference Include="System.Net.Http" />
  </ItemGroup>
  <ItemGroup>
    <Import Include="Microsoft.VisualBasic" />
    <Import Include="System" />
    <Import Include="System.Collections" />
    <Import Include="System.Collections.Generic" />
    <Import Include="System.Data" />
    <Import Include="System.Diagnostics" />
    <Import Include="System.Linq" />
    <Import Include="System.Xml.Linq" />
    <Import Include="System.Threading.Tasks" />
  </ItemGroup>
  <ItemGroup>
    <Compile Include="HelloWorld.vb" />
    <Compile Include="My Project\AssemblyInfo.vb" />
    <Compile Include="My Project\Application.Designer.vb">
      <AutoGen>True</AutoGen>
      <DependentUpon>Application.myapp</DependentUpon>
    </Compile>
    <Compile Include="My Project\Resources.Designer.vb">
      <AutoGen>True</AutoGen>
      <DesignTime>True</DesignTime>
      <DependentUpon>Resources.resx</DependentUpon>
    </Compile>
    <Compile Include="My Project\Settings.Designer.vb">
      <AutoGen>True</AutoGen>
      <DependentUpon>Settings.settings</DependentUpon>
      <DesignTimeSharedInput>True</DesignTimeSharedInput>
```

```
    </Compile>
  </ItemGroup>
  <ItemGroup>
    <EmbeddedResource Include="My Project\Resources.resx">
      <Generator>VbMyResourcesResXFileCodeGenerator</Generator>
      <LastGenOutput>Resources.Designer.vb</LastGenOutput>
      <CustomToolNamespace>My.Resources</CustomToolNamespace>
      <SubType>Designer</SubType>
    </EmbeddedResource>
  </ItemGroup>
  <ItemGroup>
    <None Include="My Project\Application.myapp">
      <Generator>MyApplicationCodeGenerator</Generator>
      <LastGenOutput>Application.Designer.vb</LastGenOutput>
    </None>
    <None Include="My Project\Settings.settings">
      <Generator>SettingsSingleFileGenerator</Generator>
      <CustomToolNamespace>My</CustomToolNamespace>
      <LastGenOutput>Settings.Designer.vb</LastGenOutput>
    </None>
    <None Include="App.config" />
  </ItemGroup>
  <Import Project="$(MSBuildToolsPath)\Microsoft.VisualBasic.targets" />
  <!-- To modify your build process, add your task inside one of the targets below and
 uncomment it.
       Other similar extension points exist, see Microsoft.Common.targets.
  <Target Name="BeforeBuild">
  </Target>
  <Target Name="AfterBuild">
  </Target>
  -->
</Project>
```

Application.Designer.vb（在 *(root directory name)*\VBHelloWorld\My Project）：

```
'------------------------------------------------------------------------------
' <auto-generated>
'   This code was generated by a tool.
'   Runtime Version:4.0.30319.42000
'
'   Changes to this file may cause incorrect behavior and will be lost if
'   the code is regenerated.
' </auto-generated>
```

```
'------------------------------------------------------------------------------

Option Strict On
Option Explicit On
```

Application.myapp（在 *(root directory name)*\VBHelloWorld\My Project）：

```
<?xml version="1.0" encoding="utf-8"?>
<MyApplicationData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <MySubMain>false</MySubMain>
  <SingleInstance>false</SingleInstance>
  <ShutdownMode>0</ShutdownMode>
  <EnableVisualStyles>true</EnableVisualStyles>
  <AuthenticationMode>0</AuthenticationMode>
  <ApplicationType>2</ApplicationType>
  <SaveMySettingsOnExit>true</SaveMySettingsOnExit>
</MyApplicationData>
```

AssemblyInfo.vb（在 *(root directory name)*\VBHelloWorld\My Project）：

```
Imports System
Imports System.Reflection
Imports System.Runtime.InteropServices

' General Information about an assembly is controlled through the following
' set of attributes. Change these attribute values to modify the information
' associated with an assembly.

' Review the values of the assembly attributes

<Assembly: AssemblyTitle("VBHelloWorld")>
<Assembly: AssemblyDescription("")>
<Assembly: AssemblyCompany("")>
<Assembly: AssemblyProduct("VBHelloWorld")>
<Assembly: AssemblyCopyright("Copyright ©  2017")>
<Assembly: AssemblyTrademark("")>

<Assembly: ComVisible(False)>

'The following GUID is for the ID of the typelib if this project is exposed to COM
<Assembly: Guid("137c362b-36ef-4c3e-84ab-f95082487a5a")>
```

```
' Version information for an assembly consists of the following four values:
'
' Major Version
' Minor Version
' Build Number
' Revision
'
' You can specify all the values or you can default the Build and Revision Numbers
' by using the '*' as shown below:
' <Assembly: AssemblyVersion("1.0.*")>

<Assembly: AssemblyVersion("1.0.0.0")>
<Assembly: AssemblyFileVersion("1.0.0.0")>
```

Resources.Designer.vb（在 *(root directory name)*\VBHelloWorld\My Project）：

```
'------------------------------------------------------------------------------
' <auto-generated>
'   This code was generated by a tool.
'   Runtime Version:4.0.30319.42000
'
'   Changes to this file may cause incorrect behavior and will be lost if
'   the code is regenerated.
' </auto-generated>
'------------------------------------------------------------------------------

Option Strict On
Option Explicit On

Namespace My.Resources

  'This class was auto-generated by the StronglyTypedResourceBuilder
  'class via a tool like ResGen or Visual Studio.
  'To add or remove a member, edit your .ResX file then rerun ResGen
  'with the /str option, or rebuild your VS project.
  '''<summary>
  '''  A strongly-typed resource class, for looking up localized strings, etc.
  '''</summary>

 <Global.System.CodeDom.Compiler.GeneratedCodeAttribute("System.Resources.Tools.StronglyTypedRe
 "4.0.0.0"), _
  Global.System.Diagnostics.DebuggerNonUserCodeAttribute(), _
  Global.System.Runtime.CompilerServices.CompilerGeneratedAttribute(), _
```

```
  Global.Microsoft.VisualBasic.HideModuleNameAttribute()> _
  Friend Module Resources

    Private resourceMan As Global.System.Resources.ResourceManager

    Private resourceCulture As Global.System.Globalization.CultureInfo

    '''<summary>
    '''  Returns the cached ResourceManager instance used by this class.
    '''</summary>

 <Global.System.ComponentModel.EditorBrowsableAttribute(Global.System.ComponentModel.EditorBrow
 _
    Friend ReadOnly Property ResourceManager() As
 Global.System.Resources.ResourceManager
      Get
        If Object.ReferenceEquals(resourceMan, Nothing) Then
          Dim temp As Global.System.Resources.ResourceManager = New
 Global.System.Resources.ResourceManager("VBHelloWorld.Resources",
 GetType(Resources).Assembly)
          resourceMan = temp
        End If
        Return resourceMan
      End Get
    End Property

    '''<summary>
    '''  Overrides the current thread's CurrentUICulture property for all
    '''  resource lookups using this strongly typed resource class.
    '''</summary>

 <Global.System.ComponentModel.EditorBrowsableAttribute(Global.System.ComponentModel.EditorBrow
 _
    Friend Property Culture() As Global.System.Globalization.CultureInfo
      Get
        Return resourceCulture
      End Get
      Set(ByVal value As Global.System.Globalization.CultureInfo)
        resourceCulture = value
      End Set
    End Property
  End Module
End Namespace
```

Resources.resx（在 *(root directory name)*\VBHelloWorld\My Project）：

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <!--
    Microsoft ResX Schema

    Version 2.0

    The primary goals of this format is to allow a simple XML format
    that is mostly human readable. The generation and parsing of the
    various data types are done through the TypeConverter classes
    associated with the data types.

    Example:

    ... ado.net/XML headers & schema ...
    <resheader name="resmimetype">text/microsoft-resx</resheader>
    <resheader name="version">2.0</resheader>
    <resheader name="reader">System.Resources.ResXResourceReader,
System.Windows.Forms, ...</resheader>
    <resheader name="writer">System.Resources.ResXResourceWriter,
System.Windows.Forms, ...</resheader>
    <data name="Name1"><value>this is my long string</value><comment>this is a
comment</comment></data>
    <data name="Color1" type="System.Drawing.Color, System.Drawing">Blue</data>
    <data name="Bitmap1" mimetype="application/x-microsoft.net.object.binary.base64">
      <value>[base64 mime encoded serialized .NET Framework object]</value>
    </data>
    <data name="Icon1" type="System.Drawing.Icon, System.Drawing"
mimetype="application/x-microsoft.net.object.bytearray.base64">
      <value>[base64 mime encoded string representing a byte array form of the .NET
Framework object]</value>
      <comment>This is a comment</comment>
    </data>

    There are any number of "resheader" rows that contain simple
    name/value pairs.

    Each data row contains a name, and value. The row also contains a
    type or mimetype. Type corresponds to a .NET class that support
    text/value conversion through the TypeConverter architecture.
    Classes that don't support this are serialized and stored with the
    mimetype set.
```

```
    The mimetype is used for serialized objects, and tells the
    ResXResourceReader how to depersist the object. This is currently not
    extensible. For a given mimetype the value must be set accordingly:

    Note - application/x-microsoft.net.object.binary.base64 is the format
    that the ResXResourceWriter will generate, however the reader can
    read any of the formats listed below.

    mimetype: application/x-microsoft.net.object.binary.base64
    value   : The object must be serialized with
            : System.Serialization.Formatters.Binary.BinaryFormatter
            : and then encoded with base64 encoding.

    mimetype: application/x-microsoft.net.object.soap.base64
    value   : The object must be serialized with
            : System.Runtime.Serialization.Formatters.Soap.SoapFormatter
            : and then encoded with base64 encoding.

    mimetype: application/x-microsoft.net.object.bytearray.base64
    value   : The object must be serialized into a byte array
            : using a System.ComponentModel.TypeConverter
            : and then encoded with base64 encoding.
 -->
 <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
   <xsd:element name="root" msdata:IsDataSet="true">
     <xsd:complexType>
       <xsd:choice maxOccurs="unbounded">
         <xsd:element name="metadata">
           <xsd:complexType>
             <xsd:sequence>
               <xsd:element name="value" type="xsd:string" minOccurs="0" />
             </xsd:sequence>
             <xsd:attribute name="name" type="xsd:string" />
             <xsd:attribute name="type" type="xsd:string" />
             <xsd:attribute name="mimetype" type="xsd:string" />
           </xsd:complexType>
         </xsd:element>
         <xsd:element name="assembly">
           <xsd:complexType>
             <xsd:attribute name="alias" type="xsd:string" />
             <xsd:attribute name="name" type="xsd:string" />
           </xsd:complexType>
```

```
          </xsd:element>
          <xsd:element name="data">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0"
 msdata:Ordinal="1" />
                <xsd:element name="comment" type="xsd:string" minOccurs="0"
 msdata:Ordinal="2" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" msdata:Ordinal="1" />
              <xsd:attribute name="type" type="xsd:string" msdata:Ordinal="3" />
              <xsd:attribute name="mimetype" type="xsd:string" msdata:Ordinal="4" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="resheader">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0"
 msdata:Ordinal="1" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required" />
            </xsd:complexType>
          </xsd:element>
        </xsd:choice>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
  <resheader name="resmimetype">
    <value>text/microsoft-resx</value>
  </resheader>
  <resheader name="version">
    <value>2.0</value>
  </resheader>
  <resheader name="reader">
    <value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=2.0.0.0,
 Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
  </resheader>
  <resheader name="writer">
    <value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=2.0.0.0,
 Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
  </resheader>
</root>
```

Settings.Designer.vb（在 *(root directory name)*\VBHelloWorld\My Project）：

```
'------------------------------------------------------------------------------
' <auto-generated>
'     This code was generated by a tool.
'     Runtime Version:4.0.30319.42000
'
'     Changes to this file may cause incorrect behavior and will be lost if
'     the code is regenerated.
' </auto-generated>
'------------------------------------------------------------------------------

Option Strict On
Option Explicit On

Namespace My

  <Global.System.Runtime.CompilerServices.CompilerGeneratedAttribute(), _

 Global.System.CodeDom.Compiler.GeneratedCodeAttribute("Microsoft.VisualStudio.Editors.Settings
 "11.0.0.0"), _

 Global.System.ComponentModel.EditorBrowsableAttribute(Global.System.ComponentModel.EditorBrows
 _
  Partial Friend NotInheritable Class MySettings
    Inherits Global.System.Configuration.ApplicationSettingsBase

    Private Shared defaultInstance As MySettings =
 CType(Global.System.Configuration.ApplicationSettingsBase.Synchronized(New
 MySettings), MySettings)

    #Region "My.Settings Auto-Save Functionality"
      #If _MyType = "WindowsForms" Then
        Private Shared addedHandler As Boolean

        Private Shared addedHandlerLockObject As New Object

        <Global.System.Diagnostics.DebuggerNonUserCodeAttribute(),
 Global.System.ComponentModel.EditorBrowsableAttribute(Global.System.ComponentModel.EditorBrows
 _
        Private Shared Sub AutoSaveSettings(ByVal sender As Global.System.Object, ByVal
 e As Global.System.EventArgs)
          If My.Application.SaveMySettingsOnExit Then
            My.Settings.Save()
```

```
          End If
        End Sub
      #End If
    #End Region

    Public Shared ReadOnly Property [Default]() As MySettings
      Get

        #If _MyType = "WindowsForms" Then
          If Not addedHandler Then
            SyncLock addedHandlerLockObject
              If Not addedHandler Then
                AddHandler My.Application.Shutdown, AddressOf AutoSaveSettings
                addedHandler = True
              End If
            End SyncLock
          End If
        #End If
        Return defaultInstance
      End Get
    End Property
  End Class
End Namespace

Namespace My

  <Global.Microsoft.VisualBasic.HideModuleNameAttribute(), _
  Global.System.Diagnostics.DebuggerNonUserCodeAttribute(), _
  Global.System.Runtime.CompilerServices.CompilerGeneratedAttribute()> _
  Friend Module MySettingsProperty

    <Global.System.ComponentModel.Design.HelpKeywordAttribute("My.Settings")> _
    Friend ReadOnly Property Settings() As Global.VBHelloWorld.My.MySettings
      Get
        Return Global.VBHelloWorld.My.MySettings.Default
      End Get
    End Property
  End Module
End Namespace
```

Settings.settings（在 *(root directory name)*\VBHelloWorld\My Project）：

```
<?xml version='1.0' encoding='utf-8'?>
```

```
<SettingsFile xmlns="http://schemas.microsoft.com/VisualStudio/2004/01/settings"
 CurrentProfile="(Default)" UseMySettingsClassName="true">
  <Profiles>
    <Profile Name="(Default)" />
  </Profiles>
  <Settings />
</SettingsFile>
```

计划构建 AWS CodeBuild

在使用之前 AWS CodeBuild，必须回答以下问题：

1. 源代码存储在哪里？ CodeBuild 目前支持通过以下源代码存储库提供程序进行构建。源代码必须包含构建规范 (buildspec) 文件。buildspec 是一组生成命令和相关设置，采 CodeBuild 用 YAML 格式，用于运行构建。您可以在构建项目定义中声明 buildspec。

存储库提供商	必需	文档
CodeCommit	存储库名称。 （可选）与源代码关联的提交 ID。	请参阅《AWS CodeCommit 用户指南》中的以下主题： 创建存储 CodeCommit 库 在中创建提交 CodeCommit
Amazon S3	输入存储桶名称。 与包含源代码的构建输入 ZIP 文件对应的对象名称。 （可选）与构建输入 ZIP 文件相关联的版本 ID。	请参阅《Amazon S3 入门指南》中的以下主题： 创建存储桶 向存储桶添加对象
GitHub	存储库名称。 （可选）与源代码关联的提交 ID。	在 GitHub 帮助网站上查看此主题： 创建存储库
Bitbucket	存储库名称。	请参阅 Bitbucket 云文档网站上的以下主题： 创建存储库

存储库提供商	必需	文档
	（可选）与源代码关联的提交 ID。	

2. 您需要运行哪些构建命令以及按照什么顺序运行？ 默认情况下，从您指定的提供商那里 CodeBuild 下载构建输入，并将构建输出上传到您指定的存储桶。您可以使用构建规范来指示如何将下载的构建输入转换为预期的构建输出。有关更多信息，请参阅Buildspec 参考。
3. 运行构建需要哪些运行时和工具？ 例如，你是在为 Java、Ruby、Python 构建还是 Node.js？ 构建是否需要 Maven 或 Ant？或者是否需要适用于 Java、Ruby 或 Python 的编译器？ 构建是否需要 Git AWS CLI、或其他工具？

   CodeBuild 在使用 Docker 镜像的构建环境中运行构建。这些 Docker 映像必须存储在 CodeBuild 支持的存储库类型中。其中包括 CodeBuild Docker 镜像存储库、Docker Hub 和亚马逊弹性容器注册表 (Amazon ECR) Container Registry。有关 CodeBuild Docker 镜像存储库的更多信息，请参阅Docker 镜像由 CodeBuild。
4. 您是否需要不是由自动提供的 AWS 资源 CodeBuild？ 如果需要，这些资源又需要哪些安全策略呢？ 例如，您可能需要修改 CodeBuild 服务角色 CodeBuild 以允许使用这些资源。
5. 您 CodeBuild 想使用您的 VPC 吗？ 如果是，您需要您的 VPC 配置的 VPC ID、子网 ID 和安全组 ID。有关更多信息，请参阅 AWS CodeBuild 与亚马逊 Virtual Private Cloud 配合使用。

回答完这些问题后，您应该已经具备了成功运行构建所需的设置和资源。要运行构建，您可以：

- 使用 AWS CodeBuild 控制台 AWS CLI、或 AWS SDK。有关更多信息，请参阅 手动运行构建。
- 在中创建或标识管道 AWS CodePipeline，然后添加生成或测试操作，指示 CodeBuild 自动测试您的代码、运行您的构建，或者两者兼而有之。有关更多信息，请参阅 CodeBuild 搭配使用 CodePipeline。

的构建规范参考 CodeBuild

此主题提供有关构建规范 (buildspec) 文件的重要参考信息。buildspec 是 YAML 格式的编译命令和相关设置的集合， CodeBuild 用于运行构建。您可以将 buildspec 作为源代码的一部分，也可以在创建构建项目时定义 buildspec。有关 buildspec 工作原理的信息，请参阅 如何 CodeBuild 运作。

主题

- buildspec 文件名称和存储位置
- buildspec 语法
- buildspec 示例
- buildspec 版本
- 批量构建 buildspec 参考

buildspec 文件名称和存储位置

如果您在源代码中包含 buildspec，则默认情况下，buildspec 文件必须命名为 `buildspec.yml` 且放置在源目录的根目录中。

可以覆盖默认 buildspec 文件名和位置。例如，您可以：

- 对同一存储库中的不同构建使用不同的 buildspec 文件，如 `buildspec_debug.yml` 和 `buildspec_release.yml`。
- 将 buildspec 文件存储在源目录的根目录之外的位置，如 `config/buildspec.yml` 或 S3 存储桶。S3 存储桶必须与您的构建项目位于同一 AWS 区域。使用其 ARN 指定 buildspec 文件（例如，`arn:aws:s3:::<my-codebuild-sample2>/buildspec.yml`）。

您可以只为构建项目指定一个 buildspec，而不管 buildspec 文件的名称如何。

要覆盖默认 buildspec 文件名、默认位置或这两者，执行下列操作之一：

- 运行 AWS CLI `create-project`或`update-project`命令，将`buildspec`值设置为相对于内置环境变量`CODEBUILD_SRC_DIR`值的备用 buildspec 文件路径。您也可以对 AWS 软件开发工具包中的`create project`操作执行等效操作。有关更多信息，请参阅创建构建项目或更改构建项目设置。

- 运行该 AWS CLI `start-build`命令，将`buildspecOverride`值设置为相对于内置环境变量`CODEBUILD_SRC_DIR`值的备用 buildspec 文件路径。您也可以对 AWS 软件开发工具包中的`start build`操作执行等效操作。有关更多信息，请参阅 手动运行构建。
- 在 AWS CloudFormation 模板中，将类型为的资源`Source`中的`BuildSpec`属性设置`AWS::CodeBuild::Project`为相对于内置环境变量`CODEBUILD_SRC_DIR`值的备用 buildspec 文件的路径。有关更多信息，请参阅《AWS CloudFormation 用户指南》中AWS CodeBuild 项目源中的 BuildSpec属性。

buildspec 语法

buildspec 文件必须以 YAML 格式表示。

如果命令包含 YAML 不支持的字符或字符串，则必须用引号 ("") 将命令括起来。以下命令用引号括起来，因为在 YAML 中不允许使用冒号 (:) 后跟空格。命令中的引号会被转义 (\")。

```
"export PACKAGE_NAME=$(cat package.json | grep name | head -1 | awk -F: '{ print $2 }'
 | sed 's/[\",]//g')"
```

buildspec 的语法如下：

```
version: 0.2

run-as: Linux-user-name

env:
  shell: shell-tag
  variables:
    key: "value"
    key: "value"
  parameter-store:
    key: "value"
    key: "value"
  exported-variables:
    - variable
    - variable
  secrets-manager:
    key: secret-id:json-key:version-stage:version-id
  git-credential-helper: no | yes

proxy:
```

```
  upload-artifacts: no | yes
  logs: no | yes

batch:
  fast-fail: false | true
  # build-list:
  # build-matrix:
  # build-graph:
  # build-fanout:

phases:
  install:
    run-as: Linux-user-name
    on-failure: ABORT | CONTINUE | RETRY | RETRY-count | RETRY-regex |
 RETRY-count-regex
    runtime-versions:
      runtime: version
      runtime: version
    commands:
      - command
      - command
    finally:
      - command
      - command

  pre_build:
    run-as: Linux-user-name
    on-failure: ABORT | CONTINUE | RETRY | RETRY-count | RETRY-regex |
 RETRY-count-regex
    commands:
      - command
      - command
    finally:
      - command
      - command

  build:
    run-as: Linux-user-name
    on-failure: ABORT | CONTINUE | RETRY | RETRY-count | RETRY-regex |
 RETRY-count-regex
    commands:
      - command
      - command
    finally:
```

```
      - command
      - command

  post_build:
    run-as: Linux-user-name
    on-failure: ABORT | CONTINUE | RETRY | RETRY-count | RETRY-regex |
 RETRY-count-regex
    commands:
      - command
      - command
    finally:
      - command
      - command

reports:
  report-group-name-or-arn:
    files:
      - location
      - location
    base-directory: location
    discard-paths: no | yes
    file-format: report-format
artifacts:
  files:
    - location
    - location
  name: artifact-name
  discard-paths: no | yes
  base-directory: location
  exclude-paths: excluded paths
  enable-symlinks: no | yes
  s3-prefix: prefix
  secondary-artifacts:
    artifactIdentifier:
      files:
        - location
        - location
      name: secondary-artifact-name
      discard-paths: no | yes
      base-directory: location
    artifactIdentifier:
      files:
        - location
        - location
```

```
      discard-paths: no | yes
      base-directory: location
cache:
  key: key
  fallback-keys:
    - fallback-key
    - fallback-key
  action: restore | save
  paths:
    - path
    - path
```

buildspec 包含以下内容：

version

必需的映射。表示 buildspec 版本。建议使用 `0.2`。

> **Note**
>
> 虽然仍支持版本 0.1，但建议尽可能使用版本 0.2。有关更多信息，请参阅 buildspec 版本。

run-as

可选的序列。仅适用于 Linux 用户。指定用于运行此 buildspec 文件中的命令的 Linux 用户。`run-as` 向指定的用户授予读取和运行权限。当您在 buildspec 文件的顶部指定 `run-as` 时，它将全局应用于所有命令。如果您不希望为所有 buildspec 文件命令指定一个用户，可以通过在其中一个 `phases` 语句块中使用 `run-as`，为一个阶段中的命令指定一个用户。如果未指定 `run-as`，则所有命令都将以根用户身份运行。

env

可选的序列。表示一个或多个自定义环境变量的信息。

> **Note**
>
> 为了保护敏感信息，以下内容隐藏在 CodeBuild 日志中：
>
> - AWS 访问密钥 ID。有关更多信息，请参阅《AWS Identity and Access Management 用户指南》中的管理 IAM 用户的访问密钥。

- 使用参数存储指定的字符串。有关更多信息，请参阅《Amazon EC2 Systems Manager 用户指南》中的 Systems Manager Parameter Store 和 Systems Manager Parameter Store 控制台演练。
- 使用指定的字符串 AWS Secrets Manager。有关更多信息，请参阅 密钥管理。

> **Note**
>
> 掩码与存储在 Amazon EC2 系统管理器参数存储区或 AWS Secrets Manager中的确切值相匹配。如果编译命令在密钥出现在日志中之前对其进行了转换，则结果是未屏蔽的另一个字符串。例如，使用 Base64 对其进行编码、反向编译、更改大小写或将其与其他文本连接会创建不同的值，该值可能出现在构建日志中。
> 要掩盖某个值，请将其存储在构建版本使用的表单中。例如，如果您的构建需要 Base64-encoded 值，则将 Base64-encoded 字符串直接存储在 AWS Secrets Manager 或 Amazon EC2 系统管理器参数存储区中。这样，记录的值与存储的值相匹配。

env/shell

可选的序列。指定 Linux 或 Windows 操作系统支持的 shell。

对于 Linux 操作系统，支持的 shell 标签有：

- `bash`
- `/bin/sh`

对于 Windows 操作系统，支持的 shell 标签有：

- `powershell.exe`
- `cmd.exe`

env/variables

在指定了 env 并且您希望定义纯文本格式的自定义环境变量时必需。包含*key*/*value*标量的映射，其中每个映射代表一个纯文本形式的自定义环境变量。*key*是自定义环境变量的名称，*value*也是该变量的值。

> **Important**
>
> 我们强烈建议不要将敏感值存储在环境变量中。可以使用 CodeBuild 控制台和等工具以纯文本显示环境变量 AWS CLI。对于敏感值，建议改用 `parameter-store` 或 `secrets-manager` 映射，如本节后面所述。
> 您设置的任何环境变量都将替换现有的环境变量。例如，如果 Docker 映像已经包含一个名为 `MY_VAR` 的环境变量（值为 `my_value`），并且您设置了一个名为 `MY_VAR` 的环境变量（值为 `other_value`），那么 `my_value` 将被替换为 `other_value`。同样，如果 Docker 映像已经包含一个名为 `PATH` 的环境变量（值为 `/usr/local/sbin:/usr/local/bin`），并且您设置了一个名为 `PATH` 的环境变量（值为 `$PATH:/usr/share/ant/bin`），那么`/usr/local/sbin:/usr/local/bin` 将被替换为文本值 `$PATH:/usr/share/ant/bin`。
> 请勿设置名称以 `CODEBUILD_` 开头的任何环境变量。此前缀是专为内部使用预留的。
> 如果具有相同名称的环境变量在多处都有定义，则应按照如下方式确定其值：
>
> - 构建操作调用开始时的值优先级最高。您可以在创建构建时添加或覆盖环境变量。有关更多信息，请参阅 手动运行 AWS CodeBuild 构建。
> - 构建项目定义中的值优先级次之。您可以在创建或编辑项目时在项目级别添加环境变量。有关更多信息，请参阅在中创建构建项目 AWS CodeBuild和在中更改构建项目设置 AWS CodeBuild：
> - buildspec 声明中的值优先级最低。

env/parameter-store

在指定了 `env` 并且您要检索存储在 Amazon EC2 Systems Manager Parameter Store 中的自定义环境变量时必需。包含*`key`*/*`value`*标量的映射，其中每个映射代表存储在 Amazon EC2 系统管理器参数存储库中的单个自定义环境变量。*`key`*是您稍后在构建命令中用来引用此自定义环境变量的名称，*`value`*也是存储在 Amazon EC2 Systems Manager 参数存储中的自定义环境变量的名称。要存储敏感值，请参阅《Amazon EC2 Systems Manager 用户指南》中的 Systems Manager Parameter Store 和演练：创建和测试参数（控制台）。

> **Important**
>
> CodeBuild 要允许检索存储在 Amazon EC2 系统管理器参数存储中的自定义环境变量，您必须将`ssm:GetParameters`操作添加到您的 CodeBuild 服务角色中。有关更多信息，请参阅 CodeBuild 允许与其他人互动 AWS 务。

从 Amazon EC2 Systems Manager Parameter Store 检索到的任何环境变量都将替换现有环境变量。例如，如果 Docker 映像已经包含一个名为 MY_VAR 的环境变量，其值为 my_value，且您检索到一个名为 MY_VAR 的环境变量，其值为 other_value，那么，my_value 将替换为 other_value。同样，如果 Docker 映像已经包含一个名为 PATH 的环境变量，其值为 /usr/local/sbin:/usr/local/bin，且您检索到一个名为 PATH 的环境变量，其值为 $PATH:/usr/share/ant/bin，那么，/usr/local/sbin:/usr/local/bin 将替换为文本值 $PATH:/usr/share/ant/bin。
请勿存储名称以 CODEBUILD_ 开头的任何环境变量。此前缀是专为内部使用预留的。
如果具有相同名称的环境变量在多处都有定义，则应按照如下方式确定其值：

- 构建操作调用开始时的值优先级最高。您可以在创建构建时添加或覆盖环境变量。有关更多信息，请参阅 手动运行 AWS CodeBuild 构建。
- 构建项目定义中的值优先级次之。您可以在创建或编辑项目时在项目级别添加环境变量。有关更多信息，请参阅在中创建构建项目 AWS CodeBuild和在中更改构建项目设置 AWS CodeBuild：
- buildspec 声明中的值优先级最低。

env/secrets-manager

如果要检索存储在中的自定义环境变量，则为必填项 AWS Secrets Manager。使用以下模式指定 Secrets Manager reference-key：

<key>: *<secret-id>*:*<json-key>*:*<version-stage>*:*<version-id>*

<key>

（必需）本地环境变量名称。在构建过程中使用此名称访问变量。

<secret-id>

（必需）用作密钥的唯一标识符的名称或 Amazon 资源名称 (ARN)。要访问您的 AWS 账户中的密钥，只需指定密钥名称。要访问其他 AWS 账户中的密钥，请指定密钥 ARN。

<json-key>

（可选）指定要检索其值的 Secrets Manager 键值对的键名称。如果未指定json-key，则 CodeBuild 检索整个机密文本。

<version-stage>

（可选）指定要按附加到版本的暂存标签检索的密钥版本。暂存标签用于在轮换过程中跟踪不同版本。如果您使用 version-stage，则不要指定 version-id。如果您不指定版本阶段或版本 ID，则默认设置是检索版本阶段值为 AWSCURRENT 的版本。

<version-id>

（可选）指定要使用的密钥版本的唯一标识符。如果您指定 version-id，请不要指定 version-stage。如果您不指定版本阶段或版本 ID，则默认设置是检索版本阶段值为 AWSCURRENT 的版本。

在以下示例中，TestSecret 是存储在 Secrets Manager 中的键值对的名称。TestSecret 的密钥是 MY_SECRET_VAR。在构建过程中，您可以使用 LOCAL_SECRET_VAR 名称访问该变量。

```
env:
  secrets-manager:
    LOCAL_SECRET_VAR: "TestSecret:MY_SECRET_VAR"
```

有关更多信息，请参阅《AWS Secrets Manager 用户指南》中的什么是 AWS Secrets Manager。

env/exported-variables

可选的映射。用于列出您要导出的环境变量。在 exported-variables 下的单独行上指定要导出的每个变量的名称。在构建过程中，要导出的变量必须在容器中可用。导出的变量可以是环境变量。

导出的环境变量与一起使用 AWS CodePipeline ，将环境变量从当前构建阶段导出到管道的后续阶段。有关更多信息，请参阅《AWS CodePipeline 用户指南》中的使用变量。

在构建期间，变量的值从 install 阶段开始可用。可以在 install 阶段开始和 post_build 阶段结束之间更新变量的值。在 post_build 阶段结束后，无法更改导出的变量的值。

> **Note**
>
> 无法导出以下项：
>
> - 构建项目中指定的 Amazon EC2 Systems Manager Parameter Store 密钥。
> - 构建项目中指定的 Secrets Manager 密钥
> - 以 AWS_ 开头的环境变量。

env/git-credential-helper

可选的映射。用于表示是否 CodeBuild 使用其 Git 凭据助手提供 Git 凭据。yes如果使用。否则为 no 或未指定。有关更多信息，请参阅 Git 网站上的 gitcredentials。

> **Note**
>
> 由 Webhook 触发的公有 Git 存储库的构建不支持 git-credential-helper。

proxy

可选的序列。如果是在显式代理服务器中运行构建，则用于表示设置。有关更多信息，请参阅 CodeBuild 在显式代理服务器中运行。

proxy/upload-artifacts

可选的映射。如果您希望显式代理服务器中的构建来上传构件，请设置为 yes。默认值为 no。

proxy/logs

可选的映射。将yes您在显式代理服务器中构建时设置为，以创建 CloudWatch 日志。默认值为 no。

phases

必需的序列。表示在构建的每个阶段 CodeBuild 运行的命令。

> **Note**
>
> 在 buildspec 版本 0.1 中，在构建环境中默认外壳的单独实例中 CodeBuild 运行每个命令。这表示各个命令独立于其他所有命令而运行。因此，在默认情况下，您无法运行依赖所有上一个命令状态的单个命令（如更改目录或设置环境变量）。要绕开此限制，建议使用版本 0.2 来解决此问题。如果必须使用 buildspec 版本 0.1，建议使用构建环境中的 Shell 和命令中的方法。

phases/*/run-as

可选的序列。在构建阶段中使用，以指定运行命令的 Linux 用户。如果还在 buildspec 文件的顶部为所有命令全局指定了 run-as，则阶段级别用户优先。例如，如果全局run-as指定 User-1，并

且该`install`阶段只有`run-as`语句指定 User-2，则 buildspec 文件中的所有命令都将作为 User-1 除该*`install`*阶段中的命令以外的所有命令运行，这些命令的运行方式为 User-2。

phases/*/on-failure

可选的序列。指定该阶段发生故障时要采取的操作。它可以是以下值之一：

- ABORT - 中止构建。
- CONTINUE - 继续到下一阶段。
- RETRY：最多可重试构建 3 次，并显示与正则表达式 `.*` 匹配的错误消息。
- RETRY-*`count`*-按指定的次数重试构建，如显示一条*`count`*与正则表达式`.*`匹配的错误消息所示。请注意，*`count`*必须介于 0 和 100 之间。例如，有效值包括 RETRY-4 和 RETRY-8。
- RETRY-*`regex`*-重试构建 3 次，并使用*`regex`*添加正则表达式来匹配指定的错误消息。例如，有效值包括 `Retry-.*Error: Unable to connect to database.*` 和 `RETRY-invalid+`。
- RETRY-*`count`*-*`regex`*-按指定的次数重试构建，如所*`count`*示。请注意，*`count`*必须介于 0 和 100 之间。您也可以使用*`regex`*添加正则表达式来匹配错误消息。例如，有效值包括 `Retry-3-.*connection timed out.*` 和 `RETRY-8-invalid+`。

如果未指定此属性，则发生故障后会进入转换阶段，如构建阶段过渡所述。

> **Important**
>
> 使用 Lambda 计算或预留容量时，不支持 `on-failure` 属性。此属性仅适用于提供的 EC2 计算映像 CodeBuild。

phases/*/finally

可选的数据块。在 `finally` 语句块中指定的命令在 `commands` 语句块命令之后运行。即便 `commands` 语句块命令失败，仍会运行 `finally` 语句块命令。例如，如果`commands`模块包含三个命令而第一个命令失败，则 CodeBuild 跳过剩下的两个命令并运行该`finally`模块中的任何命令。当 `commands` 和 `finally` 语句块中的所有命令都成功运行后，此阶段才算成功。如果某个阶段有任何命令失败，则该阶段失败。

允许的构建阶段名称是：

phases/install

可选的序列。代表安装期间 CodeBuild 运行的命令（如果有）。建议使用仅适用于在构建环境中安装软件包的 `install` 阶段。例如，您可以使用此阶段来安装代码测试框架 (如 Mocha 或 RSpec)。

phases/install/运行时版本

可选的序列。Ubuntu 标准映像 5.0 或更高版本以及 Amazon Linux 2 标准映像 4.0 或更高版本支持运行时版本。如果指定，则本节中必须包含至少一个运行时。使用特定版本指定运行时，然后指定主要版本，指定 CodeBuild使用该主要版本及其最新的次要版本，或者`latest`使用最新的主要和次要版本（例如、`ruby: 3.2`、`nodejs: 18.x`、或`java: latest`）。`.x`您可以使用一个数字或一个环境变量来指定运行时。例如，如果您使用 Amazon Linux 2 标准映像 4.0，则下面指定安装版本 17 的 Java、python 版本 3 的最新次要版本以及 Ruby 的环境变量中包含的版本。有关更多信息，请参阅 Docker 镜像由 CodeBuild。

```
phases:
  install:
    runtime-versions:
      java: corretto8
      python: 3.x
      ruby: "$MY_RUBY_VAR"
```

您可以在 buildspec 文件的 `runtime-versions` 部分中指定一个或多个运行时。如果您的运行时依赖于另一个运行时，您还可以在 buildspec 文件中指定其依赖运行时。如果您未在 buildspec 文件中指定任何运行时，请 CodeBuild 选择所用镜像中可用的默认运行时。如果您指定一个或多个运行时，则仅 CodeBuild 使用这些运行时。如果未指定依赖运行时，则会 CodeBuild 尝试为您选择依赖运行时。

> **Note**
>
> 如果未指定运行时版本，则 CodeBuild 使用默认版本。当先前的默认版本达到生命周期终止 (EOL) 时，默认版本可能会发生变化。为避免对构建环境进行意外更改，我们建议在 buildspec 文件中指定运行时版本。

如果两个指定运行时冲突，构建将失败。例如，`android: 29` 和 `java: openjdk11` 冲突，因此，如果同时指定了这两项，构建将失败。

有关可用运行时的更多信息，请参阅可用的运行时。

> **Note**
>
> 如果您指定 `runtime-versions` 部分且使用的映像不是 Ubuntu 标准映像 2.0 或更高版本或 Amazon Linux 2 (AL2) 标准映像 1.0 或更高版本，则构建会发出警告“`Skipping install of runtimes. Runtime version selection is not supported by this build image`”。

phases/install/命令

可选的序列。包含一系列标量，其中每个标量代表安装期间 CodeBuild 运行的单个命令。CodeBuild 按所列顺序从头到尾逐一运行每条命令。

phases/pre_build

可选的序列。代表在构建之前 CodeBuild 运行的命令（如果有）。例如，可以使用此阶段登录 Amazon ECR，也可以安装 npm 依赖项。

phases/pre_build/命令

如果指定 `pre_build`，则为必需的序列。包含一系列标量，其中每个标量代表在构建之前 CodeBuild 运行的单个命令。 CodeBuild按所列顺序从头到尾逐一运行每条命令。

phases/build

可选的序列。代表在构建期间 CodeBuild 运行的命令（如果有）。例如，您可以使用此阶段运行 Mocha、RSpec 或 sbt。

phases/build/命令

在指定 `build` 时是必需的。包含一系列标量，其中每个标量代表在构建期间 CodeBuild 运行的单个命令。 CodeBuild 按所列顺序从头到尾逐一运行每条命令。

phases/post_build

可选的序列。代表构建后 CodeBuild 运行的命令（如果有）。例如，您可以使用 Maven 将构建构件打包成 JAR 或 WAR 文件，还可以将 Docker 映像推入到 Amazon ECR 中。然后，您可以通过 Amazon SNS 发送构建通知。

phases/post_build/命令

在指定 `post_build` 时是必需的。包含一系列标量，其中每个标量代表在构建后 CodeBuild运行的单个命令。 CodeBuild 按所列顺序从头到尾逐一运行每条命令。

reports

report-group-name-or-arn

可选的序列。指定报告将发送到的报告组。一个项目最多可具有五个报告组。指定现有报告组的 ARN 或新报告组的名称。如果您指定名称，则使用您的项目名称和您在该格式中指定的名称 CodeBuild 创建报告组<project-name>-<report-group-name>。也可以使用 buildspec 中的环境变量（例如 $REPORT_GROUP_NAME）来设置报告组名称。有关更多信息，请参阅 报告组命名。

reports/<report-group>/files

必需的序列。表示由报告生成的测试结果的原始数据所在的位置。包含一系列标量，每个标量代表一个单独的位置， CodeBuild 可以在其中找到测试文件，相对于原始构建位置或（如果已设置）。base-directory位置可包含以下内容：

- 单个文件 (如 my-test-report-file.json)。
- 子目录中的单个文件 (例如，*my-subdirectory*/my-test-report-file.json 或 *my-parent-subdirectory*/*my-subdirectory*/my-test-report-file.json)。
- '**/*' 表示所有文件均采用递归方式。
- *my-subdirectory*/*代表名*my-subdirectory*为的子目录中的所有文件。
- *my-subdirectory*/**/*以递归方式表示所有从名为的子目录开始的文件。*my-subdirectory*

reports/<report-group>/file-format

可选的映射。表示报告文件格式。如果未指定，则使用 JUNITXML。此值不区分大小写。可能的值有：

测试报告

CUCUMBERJSON

Cucumber JSON

JUNITXML

JUnit XML

NUNITXML

NUnit XML

`NUNIT3XML`

NUnit 3 XML

`TESTNGXML`

TestNG XML

`VISUALSTUDIOTRX`

Visual Studio TRX

代码覆盖率报告

`CLOVERXML`

Clover XML

`COBERTURAXML`

Cobertura XML

`JACOCOXML`

JaCoCo XML

`SIMPLECOV`

SimpleCov JSON

> **Note**
>
> CodeBuild 接受 simplecov 生成的 JSON 代码覆盖率报告，而不是 s implecov-json。

reports/<report-group>/base-directory

可选的映射。代表相对于原始构建位置的一个或多个顶级目录， CodeBuild 用于确定在哪里可以找到原始测试文件。

reports/<report-group>/discard-paths

可选。指定是否在输出中展平报告文件目录。如果未指定此项或包含 no，则将输出报告文件，并且其目录结构保持不变。如果此项包含 `yes`，则将所有测试文件放置在同一个输出目录中。例如，如

果测试结果的路径为 com/myapp/mytests/TestResult.xml，指定 yes 会将此文件放置在 / TestResult.xml 中。

artifacts

可选的序列。表示有关在哪里 CodeBuild 可以找到构建输出以及如何 CodeBuild 准备将其上传到 S3 输出存储桶的信息。如果出现以下情况，则不需要此序列，例如，您正在构建或将 Docker 映像推送到 Amazon ECR，或者正在运行源代码上的单元测试，但并未构建源代码。

> **Note**
>
> Amazon S3 元数据有一个名为buildArn的 CodeBuild 标头，x-amz-meta-codebuild-buildarn其中包含将项目发布到 Amazon S3 的 CodeBuild 版本的。添加 buildArn 是为了允许对通知进行源跟踪并引用生成构件的构建。

artifacts/files

必需的序列。表示包含构建环境中的输出项目的位置。包含一系列标量，其中每个标量表示一个相对于原始构建位置或基本目录 (如果已设置) 的单独位置， CodeBuild 可在此处查找构建输出项目。位置可包含以下内容：

- 单个文件 (如 my-file.jar)。
- 子目录中的单个文件 (例如，*my-subdirectory*/my-file.jar 或 *my-parent-subdirectory*/*my-subdirectory*/my-file.jar)。
- '**/*' 表示所有文件均采用递归方式。
- *my-subdirectory*/*代表名*my-subdirectory*为的子目录中的所有文件。
- *my-subdirectory*/**/*以递归方式表示所有从名为的子目录开始的文件。*my-subdirectory*

当您指定生成输出对象位置时， CodeBuild 可以在构建环境中找到原始生成位置。您无需将包含路径的构建项目输出位置放在原始构建位置的前面，或指定 ./ 或类似。如果您需要了解此位置的路径，则可以在构建过程中运行命令 echo $CODEBUILD_SRC_DIR。各个构建环境的位置可能略有不同。

artifacts/name

可选的名称。为构建构件指定名称。将在以下任一条件为 true 时使用此名称。

- 您可以使用 CodeBuild API 来创建构建，并且会在更新项目、创建项目或启动构建时在`ProjectArtifacts`对象上设置该`overrideArtifactName`标志。
- 您可以使用 CodeBuild 控制台创建构建，在 buildspec 文件中指定名称，并在创建或更新项目时选择 “启用语义版本控制”。有关更多信息，请参阅 创建构建项目（控制台）。

可以在 buildspec 文件中指定在构建时计算得出的名称。buildspec 文件中指定的名称使用 Shell 命令语言。例如，您可以将日期和时间附加到您的构件名称后面，以便确保其唯一性。为构件提供唯一名称可防止其被覆盖。有关更多信息，请参阅 Shell 命令语言。

- 这是追加有构件创建日期的构件名称的示例。

```
version: 0.2
phases:
  build:
    commands:
      - rspec HelloWorld_spec.rb
artifacts:
  files:
    - '**/*'
  name: myname-$(date +%Y-%m-%d)
```

- 这是使用 CodeBuild 环境变量的工件名称的示例。有关更多信息，请参阅 构建环境中的环境变量。

```
version: 0.2
phases:
  build:
    commands:
      - rspec HelloWorld_spec.rb
artifacts:
  files:
    - '**/*'
  name: myname-$AWS_REGION
```

- 这是一个对象名称的示例，该对象使用 CodeBuild 环境变量并附加了工件的创建日期。

```
version: 0.2
phases:
  build:
    commands:
      - rspec HelloWorld_spec.rb
artifacts:
```

```
  files:
    - '**/*'
  name: $AWS_REGION-$(date +%Y-%m-%d)
```

您可以在名称中添加路径信息，这样可以根据名称中的路径将命名的构件放置在目录中。在此示例中，构建构件放在 builds/<build number>/my-artifacts 下的输出中。

```
version: 0.2
phases:
  build:
    commands:
      - rspec HelloWorld_spec.rb
artifacts:
  files:
    - '**/*'
  name: builds/$CODEBUILD_BUILD_NUMBER/my-artifacts
```

artifacts/discard-paths

可选。指定是否在输出中展平构建构件目录。如果未指定此项或包含 no，则将输出构建构件，并且其目录结构保持不变。如果此项包含 yes，则将所有构建构件放置在同一个输出目录中。例如，如果构建输出构件中某个文件的路径为 com/mycompany/app/HelloWorld.java，则指定 yes 会将此文件放置在 /HelloWorld.java 中。

artifacts/base-directory

可选的映射。表示相对于原始构建位置的一个或多个顶级目录， CodeBuild 用于确定要在构建输出对象中包含哪些文件和子目录。有效值包括：

- 单个顶级目录（如 my-directory）。
- 'my-directory*' 表示名称以 my-directory 为开头的所有顶级目录。

构建输出项目中不包含映射顶级目录，只包含其文件和子目录。

您可以使用 files 和 discard-paths 进一步限制包含哪些文件和子目录。例如，对于以下目录结构：

```
.
### my-build-1
#   ### my-file-1.txt
### my-build-2
    ### my-file-2.txt
```

```
    ### my-subdirectory
        ### my-file-3.txt
```

对于以下 artifacts 序列：

```
artifacts:
  files:
    - '*/my-file-3.txt'
  base-directory: my-build-2
```

以下子目录和文件应包含在构建输出项目中：

```
.
### my-subdirectory
    ### my-file-3.txt
```

对于以下 artifacts 序列：

```
artifacts:
  files:
    - '**/*'
  base-directory: 'my-build*'
  discard-paths: yes
```

以下文件应包含在构建输出项目中：

```
.
### my-file-1.txt
### my-file-2.txt
### my-file-3.txt
```

artifacts/exclude-paths

可选的映射。表示相对于的一条或多条路径base-directory，这些路径 CodeBuild 将从构建工件中排除。星号 (*) 字符与不跨越文件夹边界的名称组分的零个或多个字符匹配。双星号 (**) 与所有目录中名称组分的零个或多个字符匹配。

exclude-paths 示例包括以下内容：

- 要从所有目录中排除文件："**/*file-name*/**/*"

- 要排除所有点文件夹："**/.*/**/*"
- 要排除所有点文件："**/.*"

artifacts/enable-symlinks

可选。如果输出类型为 ZIP，则指定 ZIP 文件中是否保留内部符号链接。如果为 yes，则来源中的所有内部符号链接都将保留在构件 ZIP 文件中。

artifacts/s3-prefix

可选。指定将构件输出到 Amazon S3 存储桶时使用的前缀，命名空间类型为 BUILD_ID。如果使用，存储桶中的输出路径为 <s3-prefix>/<build-id>/<name>.zip。

artifacts/secondary-artifacts

可选的序列。将一个或多个构件定义表示为构件标识符与构件定义之间的映射。此区块中的每个构件标识符都必须与项目secondaryArtifacts属性中定义的构件相匹配。每个单独的定义与上面的 artifacts 块具有相同的语法。

> **Note**
>
> 即使只定义了辅助构件，该 artifacts/files 序列也始终是必需的。

例如，如果项目具有以下结构：

```
{
  "name": "sample-project",
  "secondaryArtifacts": [
    {
      "type": "S3",
      "location": "<output-bucket1>",
      "artifactIdentifier": "artifact1",
      "name": "secondary-artifact-name-1"
    },
    {
      "type": "S3",
      "location": "<output-bucket2>",
      "artifactIdentifier": "artifact2",
      "name": "secondary-artifact-name-2"
    }
  ]
}
```

则 buildspec 如下所示：

```
version: 0.2

phases:
build:
  commands:
    - echo Building...
artifacts:
  files:
    - '**/*'
  secondary-artifacts:
    artifact1:
      files:
        - directory/file1
      name: secondary-artifact-name-1
    artifact2:
      files:
        - directory/file2
      name: secondary-artifact-name-2
```

cache

可选的序列。表示有关在哪里 CodeBuild 可以准备文件以将缓存上传到 S3 缓存存储桶的信息。如果项目的缓存类型为 `No Cache`，则不需要此序列。

缓存/键

可选的序列。表示搜索或恢复缓存时使用的主键。 CodeBuild 与主键完全匹配。

以下是键的示例：

```
key: npm-key-$(codebuild-hash-files package-lock.json) }
```

缓存/回退键

可选的序列。表示当使用主键找不到缓存时按顺序使用的回退键的列表。最多支持五个回退键，每个回退键都使用前缀搜索进行匹配。如果未提供键，则将忽略此序列。

以下是回退键的示例：

```
fallback-keys:
    - npm-key-$(codebuild-hash-files package-lock.json) }
    - npm-key-
    - npm-
```

缓存/操作

可选的序列。指定要对缓存执行的操作。有效值包括：

- `restore`，只还原缓存而不保存更新。
- `save`，只保存缓存，而不还原以前的版本。

如果未提供任何值，则 CodeBuild 默认为同时执行还原和保存。

cache/paths

必需的序列。表示缓存的位置。包含一系列标量，每个标量代表一个单独的位置，CodeBuild 可以在其中找到相对于原始构建位置或基本目录（如果已设置）的构建输出构件的位置。位置可包含以下内容：

- 单个文件 (如 `my-file.jar`)。
- 子目录中的单个文件 (例如，*my-subdirectory*`/my-file.jar` 或 *my-parent-subdirectory*`/`*my-subdirectory*`/my-file.jar`)。
- `'**/*'` 表示所有文件均采用递归方式。
- *my-subdirectory*`/*`代表名*my-subdirectory*为的子目录中的所有文件。
- *my-subdirectory*`/**/*`以递归方式表示所有从名为的子目录开始的文件。*my-subdirectory*

Important

因为 buildspec 声明必须为有效的 YAML，所以 buildspec 声明中的空格至关重要。如果 buildspec 声明中的空格数量无效，则构建可能会立即失败。您可以使用 YAML 验证程序测试 buildspec 声明是否为有效的 YAML。

如果您在创建或更新构建项目时使用 AWS CLI、或 AWS SDK 声明构建规范，则构建规范必须是以 YAML 格式表示的单个字符串，以及所需的空格和换行符转义字符。将在下一部分中提供示例。

如果您使用 CodeBuild 或 AWS CodePipeline 控制台而不是 buildspec.yml 文件，则只能插入该阶段的命令。`build`不要使用前一语法，改为单行列出要在构建阶段运行的所有命令。对于多个命令，使用 `&&` 分开各个命令（例如 `mvn test && mvn package`）。

您可以使用 CodeBuild 或 CodePipeline 控制台代替 buildspec.yml 文件来指定生成输出工件在构建环境中的位置。您不要使用前一语法，而是要使用单行列出所有位置。对于多个位置，使用逗号将各个位置隔开（例如，`buildspec.yml, target/my-app.jar`）。

buildspec 示例

以下是 buildspec.yml 文件的示例。

```
version: 0.2

env:
  variables:
    JAVA_HOME: "/usr/lib/jvm/java-8-openjdk-amd64"
  parameter-store:
    LOGIN_PASSWORD: /CodeBuild/dockerLoginPassword

phases:
  install:
    commands:
      - echo Entered the install phase...
      - apt-get update -y
      - apt-get install -y maven
    finally:
      - echo This always runs even if the update or install command fails
  pre_build:
    commands:
      - echo Entered the pre_build phase...
      - docker login -u User -p $LOGIN_PASSWORD
    finally:
      - echo This always runs even if the login command fails
  build:
    commands:
      - echo Entered the build phase...
      - echo Build started on `date`
      - mvn install
    finally:
      - echo This always runs even if the install command fails
  post_build:
    commands:
      - echo Entered the post_build phase...
      - echo Build completed on `date`
```

```
reports:
  arn:aws:codebuild:your-region:your-aws-account-id:report-group/report-group-name-1:
    files:
      - "**/*"
    base-directory: 'target/tests/reports'
    discard-paths: no
  reportGroupCucumberJson:
    files:
      - 'cucumber/target/cucumber-tests.xml'
    discard-paths: yes
    file-format: CUCUMBERJSON # default is JUNITXML
artifacts:
  files:
    - target/messageUtil-1.0.jar
  discard-paths: yes
  secondary-artifacts:
    artifact1:
      files:
        - target/artifact-1.0.jar
      discard-paths: yes
    artifact2:
      files:
        - target/artifact-2.0.jar
      discard-paths: yes
cache:
  paths:
    - '/root/.m2/**/*'
```

以下是前述 buildspec 的示例，以单个字符串表示，可用于 AWS CLI、或 AWS SDK。

```
"version: 0.2\n\nenv:\n  variables:\n    JAVA_HOME: \"/usr/lib/jvm/java-8-openjdk-
amd64\\"\n  parameter-store:\n    LOGIN_PASSWORD: /CodeBuild/dockerLoginPassword\n
 phases:\n\n  install:\n    commands:\n      - echo Entered the install phase...\n
 - apt-get update -y\n      - apt-get install -y maven\n    finally:\n      - echo This
 always runs even if the update or install command fails \n  pre_build:\n    commands:
\n      - echo Entered the pre_build phase...\n      - docker login -u User -p
 $LOGIN_PASSWORD\n    finally:\n      - echo This always runs even if the login command
 fails \n  build:\n    commands:\n      - echo Entered the build phase...\n      - echo
 Build started on `date`\n      - mvn install\n    finally:\n      - echo This always
 runs even if the install command fails\n  post_build:\n    commands:\n      - echo
 Entered the post_build phase...\n      - echo Build completed on `date`\n\n reports:
\n  reportGroupJunitXml:\n    files:\n      - \"**/*\"\n    base-directory: 'target/
```

```
tests/reports'\n    discard-paths: false\n  reportGroupCucumberJson:\n    files:\n
 - 'cucumber/target/cucumber-tests.xml'\n    file-format: CUCUMBERJSON\n\nartifacts:\n
 files:\n    - target/messageUtil-1.0.jar\n  discard-paths: yes\n  secondary-artifacts:
\n    artifact1:\n      files:\n       - target/messageUtil-1.0.jar\n      discard-
paths: yes\n    artifact2:\n      files:\n       - target/messageUtil-1.0.jar\n
 discard-paths: yes\n cache:\n  paths:\n    - '/root/.m2/**/*'"
```

以下是该build阶段的命令示例，可与 CodeBuild 或 CodePipeline 控制台一起使用。

```
echo Build started on `date` && mvn install
```

在这些示例中：

- 将设置密钥为 JAVA_HOME，值为 /usr/lib/jvm/java-8-openjdk-amd64 的纯文本格式的自定义环境变量。
- 之后，将通过使用密钥 LOGIN_PASSWORD 在构建命令中引用您在 Amazon EC2 Systems Manager Parameter Store 中存储的一个名为 dockerLoginPassword 的自定义环境变量。
- 您无法更改这些构建阶段名称。将在此示例中运行的命令有 apt-get update -y、apt-get install -y maven（用于安装 Apache Maven）、mvn install（用于编译和测试源代码并将源代码打包到构建输出构件中，以及在构建输出构件的内部存储库中安装该构件）、docker login（用于使用与您在 Amazon EC2 Systems Manager Parameter Store 中设置的自定义环境变量 dockerLoginPassword 的值对应的密码登录 Docker）以及若干 echo 命令。此处包含这些echo命令以显示命令的 CodeBuild 运行方式及其运行顺序。
- files 表示要上传到构建输出位置的文件。在此示例中， CodeBuild 上传单个文件messageUtil-1.0.jar。在构建环境中名为 messageUtil-1.0.jar 的相对目录中，可找到 target 文件。由于指定了 discard-paths: yes，因此将直接上传 messageUtil-1.0.jar（而不上传到中间 target 目录）。文件名称 messageUtil-1.0.jar 和 target 的相对目录名称均基于 Apache Maven 如何创建并存储构建输出项目（仅针对此示例）。在您自己的方案中，这些文件名称和目录会有所不同。
- reports 表示在构建过程中生成报告的两个报告组：
  - arn:aws:codebuild:your-region:your-aws-account-id:report-group/report-group-name-1 指定报告组的 ARN。测试框架生成的测试结果位于 target/tests/reports 目录中。文件格式为 JunitXml，路径不会从包含测试结果的文件中删除。
  - reportGroupCucumberJson 指定一个新的报告组。如果项目名称为 my-project，则在运行构建时创建一个名为 my-project-reportGroupCucumberJson 的报告组。测试框架生成的测试结果位于 cucumber/target/cucumber-tests.xml 中。测试文件格式为 CucumberJson，路径将从包含测试结果的文件中删除。

buildspec 版本

下表列出了 buildspec 版本以及版本间的变化。

版本	更改
0.2	• `environment_variables` 已重命名为 `env`。 • `plaintext` 已重命名为 `variables`。 • `artifacts` 的 `type` 属性已被弃用。 • 在版本 0.1 中，在构建环境中默认 shell 的单独实例中 AWS CodeBuild 运行每个编译命令。在版本 0.2 中，在构建环境中默认 shell 的同一个实例中 CodeBuild 运行所有编译命令。
0.1	这是 buildspec 格式的初始定义。

批量构建 buildspec 参考

本主题包含批量构建属性的 buildspec 参考。

批处理

可选的映射。项目的批量构建设置。

batch/fast-fail

可选。指定当一个或多个构建任务失败时的批量构建行为。

`false`

默认值。所有正在运行的构建都会完成。

`true`

当其中一个构建任务失败时，所有正在运行的构建都将停止。

默认情况下，所有批量构建任务运行时，都使用 buildspec 文件中指定的构建设置（例如 env 和 phases）。您可以通过在 batch/*<batch-type>*/buildspec 参数中指定不同的 env 值或其他 buildspec 文件来覆盖默认的构建设置。

该 batch 属性的内容因所指定的批量构建类型而异。可能的批量构建类型如下：

- batch/build-##
- batch/build-##
- batch/build-##
- batch/build-fanout

batch/build-##

定义构建图。构建图定义了一组任务，这些任务依赖于批量处理中的其他任务。有关更多信息，请参阅构建图。

该元素包含一组构建任务。每个构建任务都包含以下属性。

identifier

必需。任务的标识符。

buildspec

可选。要用于此任务的 buildspec 文件的路径和文件名。如果未指定此参数，将使用当前的 buildspec 文件。

debug-session

可选。布尔值，指示是否为此批量构建启用会话调试。有关会话调试的更多信息，请参阅使用会话管理器调试构建。

`false`

会话调试已禁用。

`true`

会话调试已启用。

depend-on

可选。此任务所依赖的一组任务标识符。在这些任务完成之前，此任务不会运行。

env

可选。此任务的构建环境覆盖。它可以包含以下属性：

compute-type

用于该任务的计算类型的标识符。请参见the section called “构建环境计算模式和类型”中的 computeType，了解可能使用的值。

舰队

要用于任务的实例集的标识符。请参阅the section called “在预留容量实例集上运行构建”了解更多信息。

image

用于该任务的映像的标识符。请参阅 the section called “Docker 镜像由 CodeBuild”中的映像标识符，了解可能使用的值。

privileged-mode

布尔值，指示是否在 Docker 容器内运行 Docker 守护程序。仅在构建项目用于构建 Docker 映像时设置为 `true`。否则，尝试与 Docker 守护程序进行交互的构建将失败。默认设置为 `false`。

type

用于该任务的环境类型的标识符。请参阅the section called “构建环境计算模式和类型”中的环境类型，了解可能使用的值。

variables

构建环境中将存在的环境变量。请参阅env/variables了解更多信息。

> **Note**
>
> 请注意，不能在单个版本的相同标识符中提供计算类型和队列。

ignore-failure

可选。布尔值，用于指示是否可以忽略此构建任务的故障。

`false`

默认值。如果此构建任务失败，则批量构建将失败。

`true`

如果此构建任务失败，则批量构建仍然可以成功。

以下是构建图 buildspec 条目的示例：

```
batch:
  fast-fail: false
  build-graph:
    - identifier: build1
      env:
        variables:
          BUILD_ID: build1
      ignore-failure: false
    - identifier: build2
      buildspec: build2.yml
      env:
        variables:
          BUILD_ID: build2
      depend-on:
        - build1
    - identifier: build3
      env:
        variables:
          BUILD_ID: build3
      depend-on:
        - build2
    - identifier: build4
      env:
        compute-type: ARM_LAMBDA_1GB
    - identifier: build5
      env:
        fleet: fleet_name
```

batch/build-##

定义构建列表。构建列表用于定义多个并行运行的任务。有关更多信息，请参阅 构建列表。

该元素包含一组构建任务。每个构建任务都包含以下属性。

identifier

必需。任务的标识符。

buildspec

可选。要用于此任务的 buildspec 文件的路径和文件名。如果未指定此参数，将使用当前的 buildspec 文件。

debug-session

可选。布尔值，指示是否为此批量构建启用会话调试。有关会话调试的更多信息，请参阅使用会话管理器调试构建。

`false`

会话调试已禁用。

`true`

会话调试已启用。

env

可选。此任务的构建环境覆盖。它可以包含以下属性：

compute-type

用于该任务的计算类型的标识符。请参见the section called “构建环境计算模式和类型”中的 computeType，了解可能使用的值。

舰队

要用于任务的实例集的标识符。请参阅the section called “在预留容量实例集上运行构建”了解更多信息。

image

用于该任务的映像的标识符。请参阅 the section called “Docker 镜像由 CodeBuild”中的映像标识符，了解可能使用的值。

privileged-mode

布尔值，指示是否在 Docker 容器内运行 Docker 守护程序。仅在构建项目用于构建 Docker 映像时设置为 `true`。否则，尝试与 Docker 守护程序进行交互的构建将失败。默认设置为 `false`。

type

用于该任务的环境类型的标识符。请参阅the section called “构建环境计算模式和类型”中的环境类型，了解可能使用的值。

variables

构建环境中将存在的环境变量。请参阅env/variables了解更多信息。

> **Note**
>
> 请注意，不能在单个版本的相同标识符中提供计算类型和队列。

ignore-failure

可选。布尔值，用于指示是否可以忽略此构建任务的故障。

`false`

默认值。如果此构建任务失败，则批量构建将失败。

`true`

如果此构建任务失败，则批量构建仍然可以成功。

以下是构建列表 buildspec 条目的示例：

```
batch:
  fast-fail: false
  build-list:
    - identifier: build1
      env:
        variables:
          BUILD_ID: build1
      ignore-failure: false
    - identifier: build2
      buildspec: build2.yml
      env:
        variables:
          BUILD_ID: build2
      ignore-failure: true
    - identifier: build3
      env:
        compute-type: ARM_LAMBDA_1GB
    - identifier: build4
      env:
        fleet: fleet_name
```

```
    - identifier: build5
      env:
        compute-type: GENERAL_LINUX_XLAGRE
```

batch/build-##

定义构建矩阵。编译矩阵定义了并行运行的具有不同配置的任务。 CodeBuild 为每种可能的配置组合创建单独的构建。有关更多信息，请参阅 构建矩阵。

static

静态属性适用于所有构建任务。

ignore-failure

可选。布尔值，用于指示是否可以忽略此构建任务的故障。

`false`

默认值。如果此构建任务失败，则批量构建将失败。

`true`

如果此构建任务失败，则批量构建仍然可以成功。

env

可选。所有任务的构建环境覆盖。

privileged-mode

布尔值，指示是否在 Docker 容器内运行 Docker 守护程序。仅在构建项目用于构建 Docker 映像时设置为 `true`。否则，尝试与 Docker 守护程序进行交互的构建将失败。默认设置为 `false`。

type

用于该任务的环境类型的标识符。请参阅the section called “构建环境计算模式和类型”中的环境类型，了解可能使用的值。

dynamic

动态属性定义构建矩阵。

buildspec

可选。数组，其中包含用于这些任务的 buildspec 文件的路径和文件名。如果未指定此参数，将使用当前的 buildspec 文件。

env

可选。这些任务的构建环境覆盖。

compute-type

数组，包含用于这些任务的计算类型的标识符。请参见the section called “构建环境计算模式和类型”中的 computeType，了解可能使用的值。

image

数组，包含用于这些任务的映像的标识符。请参阅 the section called “Docker 镜像由 CodeBuild”中的映像标识符，了解可能使用的值。

variables

数组，其中包含这些任务的构建环境中将存在的环境变量。请参阅env/variables了解更多信息。

以下是构建矩阵 buildspec 条目的示例：

```
batch:
  build-matrix:
    static:
      ignore-failure: false
    dynamic:
      buildspec:
        - matrix1.yml
        - matrix2.yml
      env:
        variables:
          MY_VAR:
            - VALUE1
            - VALUE2
            - VALUE3
```

有关更多信息，请参阅 构建矩阵。

batch/build-fanout

定义构建扇出。构建 fanout 用于定义一项任务，该任务分为多个并行运行的构建。有关更多信息，请参阅 在批量构建中执行并行测试。

此元素包含一个可以拆分为多个构建的生成任务。build-fanout 部分包含以下属性。

平行性

必需。将并行运行测试的构建数量。

ignore-failure

可选。一个布尔值，指示是否可以忽略任何扇出构建任务中的失败。ignore-failure 的这个值将应用于所有扇出构建。

false

默认值。如果任何扇出构建任务失败，则批量构建将失败。

true

如果任何扇出构建任务失败，则批量构建仍然可以成功。

以下是构建扇出 buildspec 条目的示例：

```
version: 0.2

batch:
   fast-fail: false
   build-fanout:
     parallelism: 5
     ignore-failure: false

phases:
  install:
    commands:
      - npm install
   build:
    commands:
      - mkdir -p test-results
      - cd test-results
      - |
        codebuild-tests-run \
         --test-command 'npx jest --runInBand --coverage' \
         --files-search "codebuild-glob-search '**/test/**/*.test.js'" \
         --sharding-strategy 'equal-distribution'
```

有关更多信息，请参阅构建扇出和使用 codebuild-tests- run CLI 命令。

的构建环境参考 AWS CodeBuild

当您调 AWS CodeBuild 用运行构建时，必须提供有关构建环境的信息。构建环境代表操作系统、编程语言运行时和用于运行构建的 CodeBuild 工具的组合。有关构建环境工作方式的信息，请参阅 如何 CodeBuild 运作。

构建环境包含 Docker 映像。有关信息，请参阅 Docker 文档网站上的 Docker 词汇表。

当您向其提供 CodeBuild 有关构建环境的信息时，您需要在支持的存储库类型中指定 Docker 镜像的标识符。其中包括 CodeBuild Docker镜像存储库、Docker Hub中的公开镜像以及您的 AWS 账户有权访问的亚马逊弹性容器注册表（Amazon ECR）存储库。

- 我们建议您使用存储在 Docker 镜像存储库中的 CodeBuild Docker 镜像，因为它们已针对该服务进行了优化。有关更多信息，请参阅 Docker 镜像由 CodeBuild。
- 要获取 Docker Hub 中存储的公开可用的 Docker 映像的标识符，请参阅 Docker 文档网站上的搜索存储库。
- 要了解如何在您的 AWS 账户中使用 Amazon ECR 存储库中存储的 Docker 映像，请参阅 Amazon ECR 示例。

除了 Docker 映像标识符，您还可指定生成环境将使用的一组计算资源。有关更多信息，请参阅 构建环境计算模式和类型。

主题

- Docker 镜像由 CodeBuild
- 构建环境计算模式和类型
- 构建环境中的 Shell 和命令
- 构建环境中的环境变量
- 构建环境中的后台任务

Docker 镜像由 CodeBuild

支持的映像是中提供的映像的最新主要版本， CodeBuild 并通过次要和补丁版本更新进行了更新。CodeBuild 通过将支持的映像缓存到计算机的亚马逊系统映像 (AMI) 中，优化使用支持的映像的编译期限。如果您想从缓存中获益并最大限度地缩短构建的配置时间，请在 CodeBuild 控制台的 “映像版

本”部分选择“始终使用此运行时版本的最新映像”，而不是更精细的版本，例如`aws/codebuild/amazonlinux-x86_64-standard:4.0-1.0.0`。

主题

- 获取当前 Docker 映像的列表
- EC2 计算映像
- Lambda 计算映像
- 过时的图片 CodeBuild
- 可用的运行时
- 运行时版本

获取当前 Docker 映像的列表

CodeBuild 经常更新 Docker 镜像列表以添加最新镜像并弃用旧镜像。要获取最新列表，执行下列操作之一：

- 在 CodeBuild 控制台的“创建构建项目”向导或“编辑构建项目”页面中，对于“环境映像”，选择“托管映像”。从操作系统、运行时和运行时版本下拉列表中进行选择。有关更多信息，请参阅创建构建项目（控制台）或更改构建项目的设置（控制台）。
- 对于 AWS CLI，运行以下`list-curated-environment-images`命令：

```
aws codebuild list-curated-environment-images
```

- 对于 AWS SDK，调用目标编程语言的`ListCuratedEnvironmentImages`操作。有关更多信息，请参阅AWS SDK 和工具参考。

EC2 计算映像

AWS CodeBuild 支持以下 Docker 镜像，这些镜像可用于 EC2 计算 CodeBuild。

> **Note**
>
> Windows Server Core 2019 平台的基本映像仅在以下区域可用：
>
> - 美国东部（弗吉尼亚州北部）
> - 美国东部（俄亥俄州）

- 美国西部（俄勒冈州）
- 欧洲地区（爱尔兰）

平台	映像标识符	定义
Amazon Linux 2	aws/codebuild/amazonlinux-x86_64-standard:4.0	al/standard/4.0
Amazon Linux 2023	aws/codebuild/amazonlinux-x86_64-standard:5.0	al/standard/5.0
Amazon Linux 2023	aws/codebuild/amazonlinux-x86_64-standard:6.0	al/standard/6.0
Amazon Linux 2	aws/codebuild/amazonlinux-x86_64-standard:corretto8	al/standard/corretto8
Amazon Linux 2	aws/codebuild/amazonlinux-x86_64-standard:corretto11	al/standard/corretto11
Amazon Linux 2	aws/codebuild/amazonlinux-aarch64-standard:2.0	al/aarch64/standard/2.0
Amazon Linux 2023	aws/codebuild/amazonlinux-aarch64-standard:3.0	al/aarch64/standard/3.0
Ubuntu 20.04	aws/codebuild/standard:5.0	ubuntu/standard/5.0

平台	映像标识符	定义
Ubuntu 22.04	aws/codebuild/standard:6.0	ubuntu/standard/6.0
Ubuntu 22.04	aws/codebuild/standard:7.0	ubuntu/standard/7.0
Ubuntu 24.04	aws/codebuild/standard:8.0	ubuntu/standard/8.0
Windows Server Core 2019	aws/codebuild/windows-base:2019-1.0	N/A
Windows Server Core 2019	aws/codebuild/windows-base:2019-2.0	N/A
Windows Server Core 2019	aws/codebuild/windows-base:2019-3.0	N/A
Windows Server Core 2022	aws/codebuild/windows-base:2022-1.0	N/A

> **Note**
>
> 2024 年 11 月 22 日，Linux-based 标准运行时映像的别名从更新`amazonlinux2`为`amazonlinux`。无需手动更新，因为之前的别名仍然有效。

Lambda 计算映像

AWS CodeBuild 支持以下可用于 AWS Lambda 计算的 Docker 镜像 CodeBuild。

> **Important**
>
> 使用 Lambda 计算或预留容量时，不支持 `on-failure` 属性。此属性仅适用于提供的 EC2 计算映像 CodeBuild。

aarch64 ##

平台	映像标识符	定义
Amazon Linux 2	aws/codebuild/amazonlinux-aarch64-lambda-standard:dotnet6	al-lambda/aarch64 /dotnet6
Amazon Linux 2023	aws/codebuild/amazonlinux-aarch64-lambda-standard:dotnet8	al-lambda/aarch64 /dotnet8
Amazon Linux 2023	aws/codebuild/amazonlinux-aarch64-lambda-standard:dotnet10	al-lambda/aarch64 /dotnet10
Amazon Linux 2	aws/codebuild/amazonlinux-aarch64-lambda-standard:go1.21	al-lambda/aarch64 /go1.21
Amazon Linux 2023	aws/codebuild/amazonlinux-aarch64-lambda-standard:go1.24	al-lambda/aarch64 /go1.24
Amazon Linux 2023	aws/codebuild/amazonlinux-aarch64-lambda-standard:go1.25	al-lambda/aarch64 /go1.25
Amazon Linux 2	aws/codebuild/amazonlinux-aarch64-lambda-standard:corretto11	al-lambda/aarch64 /corretto11

平台	映像标识符	定义
Amazon Linux 2	`aws/codebuild/amazonlinux-aarch64-lambda-standard:corretto17`	al-lambda/aarch64 /corretto17
Amazon Linux 2023	`aws/codebuild/amazonlinux-aarch64-lambda-standard:corretto21`	al-lambda/aarch64 /corretto21
Amazon Linux 2023	`aws/codebuild/amazonlinux-aarch64-lambda-standard:corretto25`	al-lambda/aarch64 /corretto25
Amazon Linux 2	`aws/codebuild/amazonlinux-aarch64-lambda-standard:nodejs18`	al-lambda/aarch64 /nodejs18
Amazon Linux 2023	`aws/codebuild/amazonlinux-aarch64-lambda-standard:nodejs20`	al-lambda/aarch64 /nodejs20
Amazon Linux 2023	`aws/codebuild/amazonlinux-aarch64-lambda-standard:nodejs22`	al-lambda/aarch64 /nodejs22
Amazon Linux 2023	`aws/codebuild/amazonlinux-aarch64-lambda-standard:nodejs24`	al-lambda/aarch64 /nodejs24

平台	映像标识符	定义
Amazon Linux 2	aws/codebuild/amazonlinux-aarch64-lambda-standard:python3.11	al-lambda/aarch64 /python3.11
Amazon Linux 2023	aws/codebuild/amazonlinux-aarch64-lambda-standard:python3.12	al-lambda/aarch64 /python3.12
Amazon Linux 2023	aws/codebuild/amazonlinux-aarch64-lambda-standard:python3.13	al-lambda/aarch64 /python3.13
Amazon Linux 2023	aws/codebuild/amazonlinux-aarch64-lambda-standard:python3.14	al-lambda/aarch64 /python3.14
Amazon Linux 2	aws/codebuild/amazonlinux-aarch64-lambda-standard:ruby3.2	al-lambda/aarch64 /ruby3.2
Amazon Linux 2023	aws/codebuild/amazonlinux-aarch64-lambda-standard:ruby3.4	al-lambda/aarch64 /ruby3.4

x86_64 ##

平台	映像标识符	定义
Amazon Linux 2	aws/codebuild/amazonlinux-x86_64-lam	al-lambda/x86_64 /dotnet6

平台	映像标识符	定义
	bda-standard:dotnet6	
Amazon Linux 2023	aws/codebuild/amazonlinux-x86_64-lambda-standard:dotnet8	al-lambda/x86_64 /dotnet8
Amazon Linux 2023	aws/codebuild/amazonlinux-x86_64-lambda-standard:dotnet10	al-lambda/x86_64 /dotnet10
Amazon Linux 2	aws/codebuild/amazonlinux-x86_64-lambda-standard:go1.21	al-lambda/x86_64 /go1.21
Amazon Linux 2023	aws/codebuild/amazonlinux-x86_64-lambda-standard:go1.24	al-lambda/x86_64 /go1.24
Amazon Linux 2023	aws/codebuild/amazonlinux-x86_64-lambda-standard:go1.25	al-lambda/x86_64 /go1.25
Amazon Linux 2	aws/codebuild/amazonlinux-x86_64-lambda-standard:corretto11	al-lambda/x86_64 /corretto11
Amazon Linux 2	aws/codebuild/amazonlinux-x86_64-lambda-standard:corretto17	al-lambda/x86_64 /corretto17

平台	映像标识符	定义
Amazon Linux 2023	aws/codebuild/amazonlinux-x86_64-lambda-standard:corretto21	al-lambda/x86_64 /corretto21
Amazon Linux 2023	aws/codebuild/amazonlinux-x86_64-lambda-standard:corretto25	al-lambda/x86_64 /corretto25
Amazon Linux 2	aws/codebuild/amazonlinux-x86_64-lambda-standard:nodejs18	al-lambda/x86_64 /nodejs18
Amazon Linux 2023	aws/codebuild/amazonlinux-x86_64-lambda-standard:nodejs20	al-lambda/x86_64 /nodejs20
Amazon Linux 2023	aws/codebuild/amazonlinux-x86_64-lambda-standard:nodejs22	al-lambda/x86_64 /nodejs22
Amazon Linux 2023	aws/codebuild/amazonlinux-x86_64-lambda-standard:nodejs24	al-lambda/x86_64 /nodejs24
Amazon Linux 2	aws/codebuild/amazonlinux-x86_64-lambda-standard:python3.11	al-lambda/x86_64 /python3.11

平台	映像标识符	定义
Amazon Linux 2023	`aws/codebuild/amazonlinux-x86_64-lambda-standard:python3.12`	al-lambda/x86_64 /python3.12
Amazon Linux 2023	`aws/codebuild/amazonlinux-x86_64-lambda-standard:python3.13`	al-lambda/x86_64 /python3.13
Amazon Linux 2023	`aws/codebuild/amazonlinux-x86_64-lambda-standard:python3.14`	al-lambda/x86_64 /python3.14
Amazon Linux 2	`aws/codebuild/amazonlinux-x86_64-lambda-standard:ruby3.2`	al-lambda/x86_64 /ruby3.2
Amazon Linux 2023	`aws/codebuild/amazonlinux-x86_64-lambda-standard:ruby3.4`	al-lambda/x86_64 /ruby3.4

过时的图片 CodeBuild

过时的图像是指不再由 CodeBuild缓存或更新的图像。弃用的映像不再接收次要版本更新或补丁版本更新，而且由于它们不再更新，因此使用它们可能不安全。如果您的 CodeBuild 项目配置为使用较旧的镜像版本，则配置过程将下载此 docker 镜像并使用它来创建容器化运行时环境，这可能会延长配置持续时间和总体构建持续时间。

CodeBuild 已弃用以下 Docker 镜像。您仍然可以使用这些映像，但它们不会缓存在构建主机上，因此会导致预置时间更长。

平台	映像标识符	定义	弃用日期
Amazon Linux 2	`aws/codebuild/amazonlinux2-x86_64-standard:3.0`	al2/standard/3.0	2023 年 5 月 9 日
Ubuntu 18.04	`aws/codebuild/standard:4.0`	ubuntu/standard/4.0	2023 年 3 月 31 日
Amazon Linux 2	`aws/codebuild/amazonlinux2-aarch64-standard:1.0`	al2/aarch64/standard/1.0	2023 年 3 月 31 日
Ubuntu 18.04	`aws/codebuild/standard:3.0`	ubuntu/standard/3.0	2022 年 6 月 30 日
Amazon Linux 2	`aws/codebuild/amazonlinux2-x86_64-standard:2.0`	al2/standard/2.0	2022 年 6 月 30 日

主题

- 可用的运行时
- 运行时版本

可用的运行时

您可以在 buildspec 文件的 `runtime-versions` 部分中指定一个或多个运行时。如果您的运行时依赖于另一个运行时，您还可以在 buildspec 文件中指定其依赖运行时。如果您未在 buildspec 文件中指定任何运行时，请 CodeBuild 选择所用镜像中可用的默认运行时。如果您指定一个或多个运行时，则仅 CodeBuild 使用这些运行时。如果未指定依赖运行时，则会 CodeBuild 尝试为您选择依赖运行时。

> **Note**
>
> 如果未指定运行时版本，则 CodeBuild 使用默认版本。当先前的默认版本达到生命周期终止 (EOL) 时，默认版本可能会发生变化。为避免对构建环境进行意外更改，我们建议在 buildspec 文件中指定运行时版本。

有关更多信息，请参阅 Specify runtime versions in the buildspec file。

主题

- Linux 映像运行时
- macOS 映像运行时
- Windows 映像运行时

Linux 映像运行时

下表包含可用的运行时和支持这些运行时的标准 Linux 映像。

Ubuntu 和 Amazon Linux 平台运行时系统

运行时名称	版本	图片
dotnet	6.0	Amazon Linux 2 x86_64 Lambda 标准：dotnet6 Amazon Linux 2 AArch64 Lambda 标准：dotnet6 Amazon Linux 2023 x86_64 标准：5.0 Amazon Linux 2023 AArch64 标准：3.0 Ubuntu 22.04 标准：7.0
	8.0	Amazon Linux 2023 x86_64 标准：5.0

运行时名称	版本	图片
		亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0
	10.0	Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0
golang	1.20	Amazon Linux 2023 x86_64 标准：5.0 Amazon Linux 2023 AArch64 标准：3.0 Ubuntu 22.04 标准：7.0

运行时名称	版本	图片
	1.21	Amazon Linux 2 x86_64 Lambda 标准：go1.21 Amazon Linux 2 AArch64 Lambda 标准：go1.21 Amazon Linux 2023 x86_64 标准：5.0 Amazon Linux 2023 AArch64 标准：3.0 Ubuntu 22.04 标准：7.0
	1.22	Amazon Linux 2023 x86_64 标准：5.0 Amazon Linux 2023 AArch64 标准：3.0 Ubuntu 22.04 标准：7.0
	1.23	Amazon Linux 2023 x86_64 标准：5.0 Amazon Linux 2023 AArch64 标准：3.0 Ubuntu 22.04 标准：7.0

运行时名称	版本	图片
	1.24	亚马逊 Linux 2023 x86_64 Lambda 标准:go1.24 亚马逊 Linux 2023 Aarch64 Lambda 标准:go1.24 Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0
	1.25	Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0

运行时名称	版本	图片
	1.26	Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0
java	corretto8	Amazon Linux 2 x86_64 标准：corretto8 Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0

运行时名称	版本	图片
	corretto11	Amazon Linux 2 x86_64 标准：corretto11 Amazon Linux 2 x86_64 Lambda 标准：corretto11 Amazon Linux 2 AArch64 Lambda 标准：corretto11 Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0

运行时名称	版本	图片
	corretto17	Amazon Linux 2 x86_64 Lambda 标准：corretto17 Amazon Linux 2 AArch64 Lambda 标准：corretto17 Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0

运行时名称	版本	图片
	corretto21	Amazon Linux 2 x86_64 Lambda 标准：corretto21 Amazon Linux 2 AArch64 Lambda 标准：corretto21 Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0
	corretto25	Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0

运行时名称	版本	图片
nodejs	18	Amazon Linux 2 x86_64 Lambda 标准：nodejs18 Amazon Linux 2 AArch64 Lambda 标准：nodejs18 Amazon Linux 2023 x86_64 标准：5.0 Amazon Linux 2023 AArch64 标准：3.0 Ubuntu 22.04 标准：7.0
	20	Amazon Linux 2 x86_64 Lambda 标准：nodejs20 Amazon Linux 2 AArch64 Lambda 标准：nodejs20 Amazon Linux 2023 x86_64 标准：5.0 Amazon Linux 2023 AArch64 标准：3.0 Ubuntu 22.04 标准：7.0

运行时名称	版本	图片
	22	亚马逊 Linux 2023 x86_64 Lambda 标准:nodejs22 亚马逊 Linux 2023 aarch64 Lambda 标准：nodejs22 Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0
	24	Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0

运行时名称	版本	图片
	26	Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0
php	8.1	Amazon Linux 2023 AArch64 标准：3.0
	8.2	Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0

运行时名称	版本	图片
	8.3	Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0
	8.4	Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0

运行时名称	版本	图片
	8.5	Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0
python	3.9	Amazon Linux 2023 x86_64 标准：5.0 Amazon Linux 2023 AArch64 标准：3.0 Ubuntu 22.04 标准：7.0
	3.10	Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0

运行时名称	版本	图片
	3.11	Amazon Linux 2 x86_64 Lambda 标准：python3.11 Amazon Linux 2 AArch64 Lambda 标准：python3.11 Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0

运行时名称	版本	图片
	3.12	Amazon Linux 2 x86_64 Lambda 标准：python3.12 Amazon Linux 2 AArch64 Lambda 标准：python3.12 Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0

运行时名称	版本	图片
	3.13	亚马逊 Linux 2023 x86_64 Lambda 标准：python3.13 亚马逊 Linux 2023 Aarch64 Lambda 标准：python3.13 Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0
	3.14	Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0

运行时名称	版本	图片
ruby	3.1	Amazon Linux 2023 x86_64 标准：5.0 Amazon Linux 2023 AArch64 标准：3.0 Ubuntu 22.04 标准：7.0
	3.2	Amazon Linux 2 x86_64 Lambda 标准：ruby3.2 Amazon Linux 2 AArch64 Lambda 标准：ruby3.2 Amazon Linux 2023 x86_64 标准：5.0 Amazon Linux 2023 AArch64 标准：3.0 Ubuntu 22.04 标准：7.0
	3.3	Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0

运行时名称	版本	图片
	3.4	亚马逊 Linux 2023 x86_64 Lambda 标准：ruby3.4 亚马逊 Linux 2023 AArch64 Lambda 标准：ruby3.4 Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0
	4.0	Amazon Linux 2023 x86_64 标准：5.0 亚马逊 Linux 2023 x86_64 标准:6.0 Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0 Ubuntu 22.04 标准：7.0 Ubuntu 24.04 标准：8.0

运行时名称	版本	图片
rust	1.94	Amazon Linux 2023 AArch64 标准：3.0 亚马逊 Linux 2023 AArch64 标准：4.0
	1.97	亚马逊 Linux 2023 x86_64 标准:6.0 Ubuntu 24.04 标准：8.0

macOS 映像运行时

> **Important**
>
> Mac 版本的 CodeBuild 精选映像包含预安装的 macOS 和 Xcode。使用 Xcode 软件即表示您确认、理解并同意 Xcode 和 Apple SDK 协议。如果您不接受协议的条款和条件，请不要使用 Xcode 软件。请改为提供您自己的亚马逊机器映像（AMI）。有关更多信息，请参阅 如何配置 macOS 预留容量实例集？。

下表包含 macOS 支持的可用运行时。

macOS 平台运行时

运行时名称	版本	图片	附加说明
bash	3.2.57	macos-arm-base:15 macos-arm-base: 26	
clang	17.0.0	macos-arm-base:15 macos-arm-base: 26	
dotnet sdk	8.0.416	macos-arm-base:15	
	8.0.417	macos-arm-base: 26	

运行时名称	版本	图片	附加说明
	10.0.101	macos-arm-base:15	
	10.0.102	macos-arm-base: 26	
gcc	11.5.0	macos-arm-base:15 macos-arm-base: 26	可通过使用 gcc-11 别名获得
	12.4.0	macos-arm-base:15 macos-arm-base: 26	可通过使用 gcc-12 别名获得
	13.4.0	macos-arm-base:15 macos-arm-base: 26	可通过使用 gcc-13 别名获得
	14.3.0	macos-arm-base:15 macos-arm-base: 26	可通过使用 gcc-14 别名获得
gnu	11.5.0	macos-arm-base:15 macos-arm-base: 26	可通过使用 gfortran-11 别名获得
	12.4.0	macos-arm-base:15 macos-arm-base: 26	可通过使用 gfortran-12 别名获得
	13.4.0	macos-arm-base:15 macos-arm-base: 26	可通过使用 gfortran-13 别名获得
	14.3.0	macos-arm-base:15 macos-arm-base: 26	可通过使用 gfortran-14 别名获得
golang	1.24.11	macos-arm-base:15 macos-arm-base: 26	

运行时名称	版本	图片	附加说明
	1.25.4	macos-arm-base:15 macos-arm-base: 26	
java	Corretto8	macos-arm-base:15 macos-arm-base: 26	
	Corretto11	macos-arm-base:15 macos-arm-base: 26	
	Corretto17	macos-arm-base:15 macos-arm-base: 26	
	Corretto21	macos-arm-base:15 macos-arm-base: 26	
	Corretto25	macos-arm-base:15 macos-arm-base: 26	
kotlin	2.2.21	macos-arm-base:15	
	2.3.0	macos-arm-base: 26	
mono	6.14.1	macos-arm-base:15 macos-arm-base: 26	
nodejs	20.19.6	macos-arm-base:15 macos-arm-base: 26	
	22.21.1	macos-arm-base:15 macos-arm-base: 26	

运行时名称	版本	图片	附加说明
	24.11.1	macos-arm-base:15 macos-arm-base: 26	
perl	5.34.1	macos-arm-base:15 macos-arm-base: 26	
php	8.2.29	macos-arm-base:15	
	8.2.30	macos-arm-base: 26	
	8.3.28	macos-arm-base:15	
	8.3.30	macos-arm-base: 26	
	8.4.15	macos-arm-base:15	
	8.4.17	macos-arm-base: 26	
	8.5.0	macos-arm-base:15	
	8.5.2	macos-arm-base: 26	
python	3.10.19	macos-arm-base:15 macos-arm-base: 26	
	3.11.14	macos-arm-base:15 macos-arm-base: 26	
	3.12.12	macos-arm-base:15 macos-arm-base: 26	
	3.13.10	macos-arm-base:15 macos-arm-base: 26	

运行时名称	版本	图片	附加说明
	3.14.1	macos-arm-base:15 macos-arm-base: 26	
ruby	3.2.9	macos-arm-base:15 macos-arm-base: 26	
	3.3.10	macos-arm-base:15 macos-arm-base: 26	
	3.4.7	macos-arm-base:15 macos-arm-base: 26	
rust	1.91.1	macos-arm-base:15	
	1.92.0	macos-arm-base: 26	
swift	6.2.1	macos-arm-base:15	
	6.2.3	macos-arm-base: 26	
XCode	26.1.1	macos-arm-base:15	
	26.2	macos-arm-base: 26	

Windows 映像运行时

Windows Server Core 2019 的基本映像包含以下运行时。

Windows 平台运行时

运行时名称	Windows Server Core 2019 标准版：1.0 版本	Windows Server Core 2019 标准版：2.0 版本	Windows Server Core 2019 标准版：3.0 版本
dotnet	3.1 5.0	3.1 6.0	8.0

运行时名称	Windows Server Core 2019 标准版：1.0 版本	Windows Server Core 2019 标准版：2.0 版本	Windows Server Core 2019 标准版：3.0 版本
		7.0	
dotnet sdk	3.1 5.0	3.1 6.0 7.0	8.0
golang	1.14	1.18	1.21 1.22 1.23
gradle	6.7	7.6	8.12
java	Corretto11	Corretto11 Corretto17	Corretto8 Corretto11 Corretto17 Corretto21
maven	3.6	3.8	3.9
nodejs	14.15	16.19	20.18 22.13
php	7.4	8.1	8.3 8.4
PowerShell	7.1	7.2	7.4

运行时名称	Windows Server Core 2019 标准版：1.0 版本	Windows Server Core 2019 标准版：2.0 版本	Windows Server Core 2019 标准版：3.0 版本
python	3.8	3.10	3.10 3.11 3.12 3.13
ruby	2.7	3.1	3.2 3.3 3.4

运行时版本

在 buildspec 文件的 `runtime-versions` 部分中指定运行时期间，可以指定特定版本、特定主要版本和最新次要版本或最新版本。下表列出了可用的运行时及其指定方式。并非所有运行时版本都适用于所有映像。自定义镜像也不支持运行时版本选择。有关更多信息，请参阅 可用的运行时。如果您想安装和使用自定义运行时版本，而不是预安装的运行时版本，请参阅自定义运行时版本。

Ubuntu 和 Amazon Linux 2 平台运行时版本

运行时名称	版本	特定版本	特定主要和最新次要版本	最新版本
dotnet	6.0	`dotnet: 6.0`	`dotnet: 6.x`	`dotnet: latest`
	8.0	`dotnet: 8.0`	`dotnet: 8.x`	
	10.0	`dotnet: 10.0`	`dotnet: 10.x`	
golang	1.20	`golang: 1.20`	`golang: 1.x`	`golang: latest`
	1.21	`golang: 1.21`		
	1.22	`golang: 1.22`		

运行时名称	版本	特定版本	特定主要和 最新次要版本	最新版本
	1.23	golang: 1.23		
	1.24	golang: 1.24		
	1.25	golang: 1.25		
	1.26	golang: 1.26		
java	corretto8	java: corretto	java: corretto .x	java: latest
	corretto11	java: corretto 1	java: corretto 1.x	
	corretto17	java: corretto 7	java: corretto 7.x	
	corretto21	java: corretto 1	java: corretto 1.x	
	corretto25	java: corretto 5	java: corretto 5.x	
nodejs	18	nodejs: 18	nodejs: 18.x	nodejs: latest
	20	nodejs: 20	nodejs: 20.x	
	22	nodejs: 22	nodejs: 22.x	
	24	nodejs: 24	nodejs: 24.x	
php	8.1	php: 8.1	php: 8.x	php: latest
	8.2	php: 8.2		
	8.3	php: 8.3		
	8.4	php: 8.4		

运行时名称	版本	特定版本	特定主要和最新次要版本	最新版本
	8.5	php: 8.5		
python	3.9	python: 3.9	python: 3.x	python: latest
	3.10	python: 3.10		
	3.11	python: 3.11		
	3.12	python: 3.12		
	3.13	python: 3.13		
	3.14	python: 3.14		
ruby	3.1	ruby: 3.1	ruby: 3.x	ruby: latest
	3.2	ruby: 3.2		
	3.3	ruby: 3.3		
	3.4	ruby: 3.4		
	4.0	ruby: 4.0	ruby: 4.x	
rust	1.94	rust: 1.94	rust: 1.x	rust: latest

在构建阶段，您可以使用编译规范来安装其他组件（例如，Apache Maven AWS CLI、Apache Ant、Mocha、RSpec 或类似组件）。`install`有关更多信息，请参阅 buildspec 示例。

自定义运行时版本

您可以安装和使用您选择的自定义版本，而不必在 CodeBuild-managed 映像中使用预先安装的运行时版本。下表列出了可用的自定义运行时及其指定方式。

> **Note**
>
> 只有 Ubuntu 和 Amazon Linux 镜像才支持自定义运行时版本选择。

自定义运行时版本

运行时名称	语法	示例
dotnet	*<major>*.*<minor>*.*<patch>*	5.0.408
golang	*<major>*.*<minor>*	1.19
	<major>.*<minor>*.*<patch>*	1.19.1
java	corretto*<major>*	corretto15
nodejs	*<major>*	14
	<major>.*<minor>*	14.21
	<major>.*<minor>*.*<patch>*	14.21.3
php	*<major>*.*<minor>*.*<patch>*	8.0.30
python	*<major>*	3
	<major>.*<minor>*	3.7
	<major>.*<minor>*.*<patch>*	3.7.16
ruby	*<major>*.*<minor>*.*<patch>*	3.0.6

自定义运行时 buildspec 示例

以下是指定自定义运行时版本的 buildspec 示例。

```
version: 0.2
phases:
  install:
    runtime-versions:
      java: corretto15
      php: 8.0.30
      ruby: 3.0.6
      golang: 1.19
      python: 3.7
      nodejs: 14
```

```
      dotnet: 5.0.408
```

构建环境计算模式和类型

在中 CodeBuild，您可以指定用于运行构建的 CodeBuild 计算和运行时环境映像。计算是指由管理和维护的计算引擎（CPU、内存和操作系统） CodeBuild。运行时环境映像 是在您选择的计算平台上运行的容器映像，包含您的构建可能需要的额外工具，例如 AWS CLI。

主题

- 关于计算
- 关于预留容量环境类型
- 关于按需环境类型

关于计算

CodeBuild 提供 EC2 和 AWS Lambda 计算模式。EC2 可在构建过程中提供优化的灵活性，并 AWS Lambda 提供优化的启动速度。 AWS Lambda 由于启动延迟较低，支持更快的构建。 AWS Lambda 还能自动扩展，因此编译版本无需排队等候运行。有关更多信息，请参阅 在 build 上运行 AWS Lambda 计算。

借助 EC2 计算模式，您可以使用按需或预留容量实例集运行构建。对于按需队列，您可以选择预定义的计算类型，例如`BUILD_GENERAL1_SMALL`或。`BUILD_GENERAL1_LARGE`有关更多信息，请参阅关于按需环境类型。对于预留容量实例集，您可以选择自己的计算配置，包括 vCPU、内存和磁盘空间。指定配置后， CodeBuild 将选择符合您要求的支持的计算类型。有关更多信息，请参阅 关于预留容量环境类型。

关于预留容量环境类型

AWS CodeBuild 为预留容量队列提供 Linux x86、Arm、GPU、Windows 和 macOS 环境类型。下表显示了按区域排序的可用计算机类型、内存、vCPU 和磁盘空间：

US East (N. Virginia)

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib
ARM	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
ARM	16	32 GiB	256GB	GENERAL	reserved.arm.16cpu.32gib
ARM	32	64 GiB	256GB	GENERAL	reserved.arm.32cpu.64gib
ARM	48	96 GiB	512GB	GENERAL	reserved.arm.48cpu.96gib
ARM	64	128 GiB	824 GB	GENERAL	reserved.arm.64cpu.128gib
ARM EC2	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM EC2	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM EC2	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
Linux	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Linux	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Linux	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib
Linux	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Linux	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux	72	144 GiB	824 GB	GENERAL	reserved.x86-64.72cpu.144gib
Linux	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib
Linux	48	96 GiB	824 GB (SSD)	NVME	reserved.x86-64.48cpu.96gib.nvme
Linux	72	144 GiB	824 GB (SSD)	NVME	reserved.x86-64.72cpu.144gib.nvme
Linux EC2	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Linux EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux GPU	4	16 GiB	235 GB（固态硬盘）	NVME	`reserved.gpu.4cpu.16gib.nvme`
Linux GPU	8	32 GiB	435 GB（固态硬盘）	NVME	`reserved.gpu.8cpu.32gib.nvme`
Linux GPU	16	64 GiB	585 GB（固态硬盘）	NVME	`reserved.gpu.16cpu.64gib.nvme`
Linux GPU	32	128 GiB	885 GB（固态硬盘）	NVME	`reserved.gpu.32cpu.128gib.nvme`
Linux GPU	48	192 GiB	3785 GB（固态硬盘）	NVME	`reserved.gpu.48cpu.192gib.nvme`
Linux GPU	64	256 GiB	1885 GB（固态硬盘）	NVME	`reserved.gpu.64cpu.256gib.nvme`
Linux GPU	96	384 GiB	3785 GB（固态硬盘）	NVME	`reserved.gpu.96cpu.384gib.nvme`

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
macOS	8	24 GiB	128 GB	GENERAL	reserved.arm.m2.8cpu.24gib
macOS	12	32 GiB	256GB	GENERAL	reserved.arm.m2.12cpu.32gib
Windows	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Windows	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Windows	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Windows	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib
Windows	32	64 GiB	256GB	GENERAL	reserved.x86-64.32cpu.64gib
Windows	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Windows	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib
Windows	64	128 GiB	512GB	GENERAL	reserved.x86-64.64cpu.128gib
Windows	72	144 GiB	824 GB	GENERAL	reserved.x86-64.72cpu.144gib
Windows	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib
Windows EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Windows EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib

有关定价标识符的更多信息，请参阅https://aws.amazon.com/codebuild/pricing/。

US East (Ohio)

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib
ARM	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
ARM	16	32 GiB	256GB	GENERAL	reserved.arm.16cpu.32gib
ARM	32	64 GiB	256GB	GENERAL	reserved.arm.32cpu.64gib
ARM	48	96 GiB	512GB	GENERAL	reserved.arm.48cpu.96gib
ARM	64	128 GiB	824 GB	GENERAL	reserved.arm.64cpu.128gib
ARM EC2	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM EC2	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM EC2	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
Linux	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Linux	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Linux	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib
Linux	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Linux	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib
Linux	48	96 GiB	824 GB (SSD)	NVME	reserved.x86-64.48cpu.96gib.nvme
Linux	72	144 GiB	824 GB (SSD)	NVME	reserved.x86-64.72cpu.144gib.nvme
Linux EC2	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Linux EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Linux GPU	4	16 GiB	235 GB（固态硬盘）	NVME	reserved.gpu.4cpu.16gib.nvme

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux GPU	8	32 GiB	435 GB（固态硬盘）	NVME	reserved.gpu.8cpu.32gib.nvme
Linux GPU	16	64 GiB	585 GB（固态硬盘）	NVME	reserved.gpu.16cpu.64gib.nvme
Linux GPU	32	128 GiB	885 GB（固态硬盘）	NVME	reserved.gpu.32cpu.128gib.nvme
Linux GPU	48	192 GiB	3785 GB（固态硬盘）	NVME	reserved.gpu.48cpu.192gib.nvme
macOS	8	24 GiB	128 GB	GENERAL	reserved.arm.m2.8cpu.24gib
macOS	12	32 GiB	256GB	GENERAL	reserved.arm.m2.12cpu.32gib
Windows	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Windows	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Windows	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Windows	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib
Windows	32	64 GiB	256GB	GENERAL	reserved.x86-64.32cpu.64gib
Windows	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Windows	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib
Windows	64	128 GiB	512GB	GENERAL	reserved.x86-64.64cpu.128gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Windows	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib
Windows EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Windows EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib

有关定价标识符的更多信息，请参阅https://aws.amazon.com/codebuild/pricing/。

US West (N. California)

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib
ARM	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
ARM	16	32 GiB	256GB	GENERAL	reserved.arm.16cpu.32gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM	32	64 GiB	256GB	GENERAL	reserved.arm.32cpu.64gib
ARM	48	96 GiB	824 GB	GENERAL	reserved.arm.48cpu.96gib
ARM	64	128 GiB	824 GB	GENERAL	reserved.arm.64cpu.128gib
ARM EC2	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib
ARM EC2	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM EC2	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
Linux	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Linux	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib
Linux	32	64 GiB	256GB	GENERAL	reserved.x86-64.32cpu.64gib
Linux	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Linux	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib
Linux	64	128 GiB	512GB	GENERAL	reserved.x86-64.64cpu.128gib
Linux	72	144 GiB	824 GB	GENERAL	reserved.x86-64.72cpu.144gib
Linux	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux	72	144 GiB	824 GB (SSD)	NVME	reserved.x86-64.72cpu.144gib.nvme
Linux EC2	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Linux EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Windows	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Windows	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Windows	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Windows	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Windows	32	64 GiB	256GB	GENERAL	reserved.x86-64.32cpu.64gib
Windows	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Windows	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib
Windows	64	128 GiB	512GB	GENERAL	reserved.x86-64.64cpu.128gib
Windows	72	144 GiB	824 GB	GENERAL	reserved.x86-64.72cpu.144gib
Windows	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib
Windows EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Windows EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib

有关定价标识符的更多信息，请参阅https://aws.amazon.com/codebuild/pricing/。

US West (Oregon)

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib
ARM	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
ARM	16	32 GiB	256GB	GENERAL	reserved.arm.16cpu.32gib
ARM	32	64 GiB	256GB	GENERAL	reserved.arm.32cpu.64gib
ARM	48	96 GiB	512GB	GENERAL	reserved.arm.48cpu.96gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM	64	128 GiB	824 GB	GENERAL	reserved.arm.64cpu.128gib
ARM EC2	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib
ARM EC2	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM EC2	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
Linux	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Linux	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Linux	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Linux	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib
Linux	72	144 GiB	824 GB	GENERAL	reserved.x86-64.72cpu.144gib
Linux	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib
Linux	48	96 GiB	824 GB (SSD)	NVME	reserved.x86-64.48cpu.96gib.nvme
Linux	72	144 GiB	824 GB (SSD)	NVME	reserved.x86-64.72cpu.144gib.nvme
Linux EC2	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Linux GPU	4	16 GiB	235 GB（固态硬盘）	NVME	reserved.gpu.4cpu.16gib.nvme
Linux GPU	8	32 GiB	435 GB（固态硬盘）	NVME	reserved.gpu.8cpu.32gib.nvme
Linux GPU	16	64 GiB	585 GB（固态硬盘）	NVME	reserved.gpu.16cpu.64gib.nvme
Linux GPU	32	128 GiB	885 GB（固态硬盘）	NVME	reserved.gpu.32cpu.128gib.nvme
Linux GPU	48	192 GiB	3785 GB（固态硬盘）	NVME	reserved.gpu.48cpu.192gib.nvme

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux GPU	64	256 GiB	1885 GB（固态硬盘）	NVME	reserved.gpu.64cpu.256gib.nvme
macOS	8	24 GiB	128 GB	GENERAL	reserved.arm.m2.8cpu.24gib
macOS	12	32 GiB	256GB	GENERAL	reserved.arm.m2.12cpu.32gib
Windows	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Windows	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Windows	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Windows	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib
Windows	32	64 GiB	256GB	GENERAL	reserved.x86-64.32cpu.64gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Windows	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Windows	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib
Windows	64	128 GiB	512GB	GENERAL	reserved.x86-64.64cpu.128gib
Windows	72	144 GiB	824 GB	GENERAL	reserved.x86-64.72cpu.144gib
Windows	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib
Windows EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Windows EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib

有关定价标识符的更多信息，请参阅https://aws.amazon.com/codebuild/pricing/。

Asia Pacific (Tokyo)

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib
ARM	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
ARM	16	32 GiB	256GB	GENERAL	reserved.arm.16cpu.32gib
ARM	32	64 GiB	256GB	GENERAL	reserved.arm.32cpu.64gib
ARM	48	96 GiB	512GB	GENERAL	reserved.arm.48cpu.96gib
ARM	64	128 GiB	824 GB	GENERAL	reserved.arm.64cpu.128gib
ARM EC2	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM EC2	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM EC2	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
Linux	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Linux	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Linux	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib
Linux	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Linux	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux	72	144 GiB	824 GB	GENERAL	reserved.x86-64.72cpu.144gib
Linux	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib
Linux	48	96 GiB	824 GB (SSD)	NVME	reserved.x86-64.48cpu.96gib.nvme
Linux	72	144 GiB	824 GB (SSD)	NVME	reserved.x86-64.72cpu.144gib.nvme
Linux EC2	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Linux EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux GPU	4	16 GiB	235 GB（固态硬盘）	NVME	reserved.gpu.4cpu.16gib.nvme
Linux GPU	8	32 GiB	435 GB（固态硬盘）	NVME	reserved.gpu.8cpu.32gib.nvme
Linux GPU	48	192 GiB	3785 GB（固态硬盘）	NVME	reserved.gpu.48cpu.192gib.nvme
Windows	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Windows	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Windows	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Windows	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Windows	32	64 GiB	256GB	GENERAL	reserved.x86-64.32cpu.64gib
Windows	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Windows	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib
Windows	64	128 GiB	512GB	GENERAL	reserved.x86-64.64cpu.128gib
Windows	72	144 GiB	824 GB	GENERAL	reserved.x86-64.72cpu.144gib
Windows	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib
Windows EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Windows EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib

有关定价标识符的更多信息，请参阅https://aws.amazon.com/codebuild/pricing/。

Asia Pacific (Malaysia)

环境类型	vCPU	内存	计算机类型	计算实例类型
ARM	2	4 GiB	GENERAL	reserved:arm.g1.small
ARM	4	8 GiB	GENERAL	reserved:arm.g1.medium
ARM	8	16 GiB	GENERAL	reserved:arm.g1.large
ARM	32	64 GiB	GENERAL	reserved:arm.g1.xlarge
ARM	48	96 GiB	GENERAL	reserved:arm.g1.2xlarge
Linux	2	4 GiB	GENERAL	reserved:x86-64.g1.small

环境类型	vCPU	内存	计算机类型	计算实例类型
Linux	4	8 GiB	GENERAL	reserved:x86-64.g1.medium
Linux	8	16 GiB	GENERAL	reserved:x86-64.g1.large

有关定价标识符的更多信息，请参阅https://aws.amazon.com/codebuild/pricing/。

Asia Pacific (New Zealand)

环境类型	vCPU	内存	计算机类型	计算实例类型
ARM	2	4 GiB	GENERAL	reserved:arm.g1.small
ARM	4	8 GiB	GENERAL	reserved:arm.g1.medium
ARM	8	16 GiB	GENERAL	reserved:arm.g1.large
ARM	32	64 GiB	GENERAL	reserved:arm.g1.xlarge
ARM	48	96 GiB	GENERAL	reserved:arm.g1.2xlarge

环境类型	vCPU	内存	计算机类型	计算实例类型
Linux	2	4 GiB	GENERAL	reserved:x86-64.g1.small
Linux	4	8 GiB	GENERAL	reserved:x86-64.g1.medium
Linux	8	16 GiB	GENERAL	reserved:x86-64.g1.large

有关定价标识符的更多信息，请参阅https://aws.amazon.com/codebuild/pricing/。

Asia Pacific (Thailand)

环境类型	vCPU	内存	计算机类型	计算实例类型
ARM	2	4 GiB	GENERAL	reserved:arm.g1.small
ARM	4	8 GiB	GENERAL	reserved:arm.g1.medium
ARM	8	16 GiB	GENERAL	reserved:arm.g1.large
ARM	32	64 GiB	GENERAL	reserved:arm.g1.xlarge

环境类型	vCPU	内存	计算机类型	计算实例类型
ARM	48	96 GiB	GENERAL	reserved:arm.g1.2xlarge
Linux	2	4 GiB	GENERAL	reserved:x86-64.g1.small
Linux	4	8 GiB	GENERAL	reserved:x86-64.g1.medium
Linux	8	16 GiB	GENERAL	reserved:x86-64.g1.large

有关定价标识符的更多信息，请参阅https://aws.amazon.com/codebuild/pricing/。

Asia Pacific (Mumbai)

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib
ARM	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM	16	32 GiB	256GB	GENERAL	reserved.arm.16cpu.32gib
ARM	32	64 GiB	256GB	GENERAL	reserved.arm.32cpu.64gib
ARM	48	96 GiB	512GB	GENERAL	reserved.arm.48cpu.96gib
ARM	64	128 GiB	824 GB	GENERAL	reserved.arm.64cpu.128gib
ARM EC2	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib
ARM EC2	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM EC2	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
Linux	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Linux	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Linux	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib
Linux	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Linux	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib
Linux	72	144 GiB	824 GB	GENERAL	reserved.x86-64.72cpu.144gib
Linux	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib
Linux	72	144 GiB	824 GB (SSD)	NVME	reserved.x86-64.72cpu.144gib.nvme

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux EC2	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Linux EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Linux GPU	4	16 GiB	235 GB（固态硬盘）	NVME	reserved.gpu.4cpu.16gib.nvme
Linux GPU	8	32 GiB	435 GB（固态硬盘）	NVME	reserved.gpu.8cpu.32gib.nvme
Linux GPU	16	64 GiB	585 GB（固态硬盘）	NVME	reserved.gpu.16cpu.64gib.nvme
Windows	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Windows	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Windows	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Windows	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib
Windows	32	64 GiB	256GB	GENERAL	reserved.x86-64.32cpu.64gib
Windows	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Windows	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib
Windows	64	128 GiB	512GB	GENERAL	reserved.x86-64.64cpu.128gib
Windows	72	144 GiB	824 GB	GENERAL	reserved.x86-64.72cpu.144gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Windows	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib
Windows EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Windows EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib

有关定价标识符的更多信息，请参阅https://aws.amazon.com/codebuild/pricing/。

Asia Pacific (Singapore)

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib
ARM	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
ARM	16	32 GiB	256GB	GENERAL	reserved.arm.16cpu.32gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM	32	64 GiB	256GB	GENERAL	reserved.arm.32cpu.64gib
ARM	48	96 GiB	512GB	GENERAL	reserved.arm.48cpu.96gib
ARM	64	128 GiB	824 GB	GENERAL	reserved.arm.64cpu.128gib
ARM EC2	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib
ARM EC2	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM EC2	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
Linux	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Linux	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib
Linux	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Linux	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib
Linux	72	144 GiB	824 GB	GENERAL	reserved.x86-64.72cpu.144gib
Linux	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib
Linux	48	96 GiB	824 GB (SSD)	NVME	reserved.x86-64.48cpu.96gib.nvme
Linux	72	144 GiB	824 GB (SSD)	NVME	reserved.x86-64.72cpu.144gib.nvme

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux EC2	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Linux EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Windows	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Windows	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Windows	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Windows	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib
Windows	32	64 GiB	256GB	GENERAL	reserved.x86-64.32cpu.64gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Windows	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Windows	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib
Windows	64	128 GiB	512GB	GENERAL	reserved.x86-64.64cpu.128gib
Windows	72	144 GiB	824 GB	GENERAL	reserved.x86-64.72cpu.144gib
Windows	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib
Windows EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Windows EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib

有关定价标识符的更多信息，请参阅https://aws.amazon.com/codebuild/pricing/。

Asia Pacific (Sydney)

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib
ARM	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
ARM	16	32 GiB	256GB	GENERAL	reserved.arm.16cpu.32gib
ARM	32	64 GiB	256GB	GENERAL	reserved.arm.32cpu.64gib
ARM	48	96 GiB	512GB	GENERAL	reserved.arm.48cpu.96gib
ARM	64	128 GiB	824 GB	GENERAL	reserved.arm.64cpu.128gib
ARM EC2	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM EC2	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM EC2	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
Linux	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Linux	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Linux	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib
Linux	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Linux	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux	72	144 GiB	824 GB	GENERAL	reserved.x86-64.72cpu.144gib
Linux	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib
Linux	48	96 GiB	824 GB (SSD)	NVME	reserved.x86-64.48cpu.96gib.nvme
Linux	72	144 GiB	824 GB (SSD)	NVME	reserved.x86-64.72cpu.144gib.nvme
Linux EC2	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Linux EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux GPU	4	16 GiB	235 GB（固态硬盘）	NVME	reserved.gpu.4cpu.16gib.nvme
Linux GPU	8	32 GiB	435 GB（固态硬盘）	NVME	reserved.gpu.8cpu.32gib.nvme
Linux GPU	16	64 GiB	585 GB（固态硬盘）	NVME	reserved.gpu.16cpu.64gib.nvme
Linux GPU	48	192 GiB	3785 GB（固态硬盘）	NVME	reserved.gpu.48cpu.192gib.nvme
macOS	8	24 GiB	128 GB	GENERAL	reserved.arm.m2.8cpu.24gib
macOS	12	32 GiB	256GB	GENERAL	reserved.arm.m2.12cpu.32gib
Windows	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Windows	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Windows	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Windows	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib
Windows	32	64 GiB	256GB	GENERAL	reserved.x86-64.32cpu.64gib
Windows	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Windows	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib
Windows	64	128 GiB	512GB	GENERAL	reserved.x86-64.64cpu.128gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Windows	72	144 GiB	824 GB	GENERAL	reserved.x86-64.72cpu.144gib
Windows	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib
Windows EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Windows EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib

有关定价标识符的更多信息，请参阅https://aws.amazon.com/codebuild/pricing/。

Europe (Frankfurt)

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib
ARM	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
ARM	16	32 GiB	256GB	GENERAL	reserved.arm.16cpu.32gib
ARM	32	64 GiB	256GB	GENERAL	reserved.arm.32cpu.64gib
ARM	48	96 GiB	512GB	GENERAL	reserved.arm.48cpu.96gib
ARM	64	128 GiB	824 GB	GENERAL	reserved.arm.64cpu.128gib
ARM EC2	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib
ARM EC2	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM EC2	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
Linux	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Linux	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib
Linux	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Linux	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib
Linux	72	144 GiB	824 GB	GENERAL	reserved.x86-64.72cpu.144gib
Linux	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux	48	96 GiB	824 GB (SSD)	NVME	reserved.x86-64.48cpu.96gib.nvme
Linux	72	144 GiB	824 GB (SSD)	NVME	reserved.x86-64.72cpu.144gib.nvme
Linux EC2	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Linux EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Linux GPU	4	16 GiB	235 GB（固态硬盘）	NVME	reserved.gpu.4cpu.16gib.nvme
Linux GPU	8	32 GiB	435 GB（固态硬盘）	NVME	reserved.gpu.8cpu.32gib.nvme

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux GPU	16	64 GiB	585 GB（固态硬盘）	NVME	reserved.gpu.16cpu.64gib.nvme
Linux GPU	32	128 GiB	885 GB（固态硬盘）	NVME	reserved.gpu.32cpu.128gib.nvme
Linux GPU	48	192 GiB	3785 GB（固态硬盘）	NVME	reserved.gpu.48cpu.192gib.nvme
macOS	8	24 GiB	128 GB	GENERAL	reserved.arm.m2.8cpu.24gib
Windows	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Windows	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Windows	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Windows	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Windows	32	64 GiB	256GB	GENERAL	reserved.x86-64.32cpu.64gib
Windows	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Windows	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib
Windows	64	128 GiB	512GB	GENERAL	reserved.x86-64.64cpu.128gib
Windows	72	144 GiB	824 GB	GENERAL	reserved.x86-64.72cpu.144gib
Windows	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib
Windows EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Windows EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib

有关定价标识符的更多信息，请参阅https://aws.amazon.com/codebuild/pricing/。

Europe (Ireland)

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib
ARM	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
ARM	16	32 GiB	256GB	GENERAL	reserved.arm.16cpu.32gib
ARM	32	64 GiB	256GB	GENERAL	reserved.arm.32cpu.64gib
ARM	48	96 GiB	512GB	GENERAL	reserved.arm.48cpu.96gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM	64	128 GiB	824 GB	GENERAL	reserved.arm.64cpu.128gib
ARM EC2	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib
ARM EC2	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM EC2	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
Linux	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Linux	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Linux	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Linux	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib
Linux	72	144 GiB	824 GB	GENERAL	reserved.x86-64.72cpu.144gib
Linux	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib
Linux	48	96 GiB	824 GB (SSD)	NVME	reserved.x86-64.48cpu.96gib.nvme
Linux	72	144 GiB	824 GB (SSD)	NVME	reserved.x86-64.72cpu.144gib.nvme
Linux EC2	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Linux GPU	4	16 GiB	235 GB（固态硬盘）	NVME	reserved.gpu.4cpu.16gib.nvme
Linux GPU	8	32 GiB	435 GB（固态硬盘）	NVME	reserved.gpu.8cpu.32gib.nvme
Linux GPU	16	64 GiB	585 GB（固态硬盘）	NVME	reserved.gpu.16cpu.64gib.nvme
Linux GPU	32	128 GiB	885 GB（固态硬盘）	NVME	reserved.gpu.32cpu.128gib.nvme
Linux GPU	48	192 GiB	3785 GB（固态硬盘）	NVME	reserved.gpu.48cpu.192gib.nvme

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Windows	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Windows	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Windows	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Windows	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib
Windows	32	64 GiB	256GB	GENERAL	reserved.x86-64.32cpu.64gib
Windows	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Windows	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Windows	64	128 GiB	512GB	GENERAL	reserved.x86-64.64cpu.128gib
Windows	72	144 GiB	824 GB	GENERAL	reserved.x86-64.72cpu.144gib
Windows	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib
Windows EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Windows EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib

有关定价标识符的更多信息，请参阅https://aws.amazon.com/codebuild/pricing/。

South America (São Paulo)

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
ARM	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
ARM	16	32 GiB	256GB	GENERAL	reserved.arm.16cpu.32gib
ARM	32	64 GiB	256GB	GENERAL	reserved.arm.32cpu.64gib
ARM	48	96 GiB	512GB	GENERAL	reserved.arm.48cpu.96gib
ARM EC2	2	4 GiB	64 GB	GENERAL	reserved.arm.2cpu.4gib
ARM EC2	4	8 GiB	128 GB	GENERAL	reserved.arm.4cpu.8gib
ARM EC2	8	16 GiB	128 GB	GENERAL	reserved.arm.8cpu.16gib
Linux	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Linux	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib
Linux	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Linux	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib
Linux	72	144 GiB	824 GB	GENERAL	reserved.x86-64.72cpu.144gib
Linux	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Linux	48	96 GiB	824 GB (SSD)	NVME	reserved.x86-64.48cpu.96gib.nvme
Linux	72	144 GiB	824 GB (SSD)	NVME	reserved.x86-64.72cpu.144gib.nvme
Linux EC2	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Linux EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Linux EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib
Windows	2	4 GiB	64 GB	GENERAL	reserved.x86-64.2cpu.4gib
Windows	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Windows	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Windows	16	32 GiB	256GB	GENERAL	reserved.x86-64.16cpu.32gib
Windows	32	64 GiB	256GB	GENERAL	reserved.x86-64.32cpu.64gib
Windows	36	72 GiB	256GB	GENERAL	reserved.x86-64.36cpu.72gib
Windows	48	96 GiB	512GB	GENERAL	reserved.x86-64.48cpu.96gib
Windows	64	128 GiB	512GB	GENERAL	reserved.x86-64.64cpu.128gib
Windows	72	144 GiB	824 GB	GENERAL	reserved.x86-64.72cpu.144gib
Windows	96	192 GiB	824 GB	GENERAL	reserved.x86-64.96cpu.192gib

环境类型	vCPU	内存	磁盘空间	计算机类型	计算实例类型
Windows EC2	4	8 GiB	128 GB	GENERAL	reserved.x86-64.4cpu.8gib
Windows EC2	8	16 GiB	128 GB	GENERAL	reserved.x86-64.8cpu.16gib

有关定价标识符的更多信息，请参阅https://aws.amazon.com/codebuild/pricing/。

选择计算类型：

- 在 CodeBuild 控制台的计算队列配置页面中，从 vCPU 、内存和磁盘中选择一个选项。有关更多信息，请参阅 创建预留容量实例集。
- 对于 AWS CLI，运行`create-fleet`或`update-fleet`命令，指定 to `computeType` 的值`ATTRIBUTE_BASED_COMPUTE`。有关更多信息，请参阅 create-fleet 或 update-fleet。
- 对于 AWS SDK，调用目标编程语言的`CreateFleet`或等效`UpdateFleet`运算，将值指定`computeType`为`ATTRIBUTE_BASED_COMPUTE`。有关更多信息，请参阅AWS SDK 和工具参考。

> **Note**
>
> 对于 AWS CLI 和 AWS SDK，您仍然可以使用诸如的`computeType`输入来选择计算类型`BUILD_GENERAL1_SMALL`，而不是。`ATTRIBUTE_BASED_COMPUTE`有关更多信息，请参阅 关于按需环境类型。

支持的实例系列

AWS CodeBuild 支持预留容量队列的以下实例：

- 一般用途：M5 | m5a | m5ad | m5d | m5dn | m5zn | m6a | m6g | m6g | m6gd | m6i | m6id | m6id | m6iDn | m6in | m7a | m7g | m7gd | m7i | | m8g | T3 | t3a | t4g M7i-flex

- 计算优化：C5 | c5a | c5ad | c5d | c5n | c6a | c6g | c6gd | c6gd | c6gn | c6i | c6id | c6id | c6in | c7a | c7g | c7gd | c7gn | c7i | | c8g C7i-flex
- 内存优化：R5 | r5a | r5ad | r5b | r5d | r5n | r5n | r6a | r6g | r6gd | r6gd | r6i | r6iDn | r6in | r6id | r7a | r7g | r7g | r7i | r7iz | r8g | | |

计算类型	环境 computeType 值	环境类型值	内存	vCPU	磁盘空间
ARM Small [1]	BUILD_GENERAL1_SMALL	ARM_CONTAINER ARM_EC2	4 GiB	2	64 GB
ARM Medium [1]	BUILD_GENERAL1_MEDIUM	ARM_CONTAINER ARM_EC2	8 GiB	4	128 GB
ARM Large [1]	BUILD_GENERAL1_LARGE	ARM_CONTAINER ARM_EC2	16 GiB	8	128 GB
ARM XLarge [1]	BUILD_GENERAL1_XLARGE	ARM_CONTAINER	64 GiB	32	256GB
ARM 2XLarge [1]	BUILD_GENERAL1_2XLARGE	ARM_CONTAINER	96 GiB	48	824 GB
小型 Linux[1]	BUILD_GENERAL1_SMALL	LINUX_CONTAINER LINUX_EC2	4 GiB	2	64 GB
中型 Linux[1]	BUILD_GENERAL1_MEDIUM	LINUX_CONTAINER LINUX_EC2	8 GiB	4	128 GB
大型 Linux[1]	BUILD_GENERAL1_LARGE	LINUX_CONTAINER	16 GiB	8	128 GB

计算类型	环境 computeType 值	环境类型值	内存	vCPU	磁盘空间
		LINUX_EC2			
Linux XLarge [1]	BUILD_GENERAL1_XLARGE	LINUX_CONTAINER	72 GiB	36	256GB
Linux 2XLarge	BUILD_GENERAL1_2XLARGE	LINUX_CONTAINER	144 GiB	72	824 GB (SSD)
小型 Linux GPU	BUILD_GENERAL1_SMALL	LINUX_GPU_CONTAINER	16 GiB	4	235 GB（固态硬盘）
大型 Linux GPU	BUILD_GENERAL1_LARGE	LINUX_GPU_CONTAINER	255 GiB	32	50 GB
Windows Mediu [1]	BUILD_GENERAL1_MEDIUM	WINDOWS_SERVER_2019_CONTAINER	8 GiB	4	128 GB
		WINDOWS_SERVER_2022_CONTAINER			
		WINDOWS_EC2			

计算类型	环境 computeType 值	环境类型值	内存	vCPU	磁盘空间
Windows Large [1]	BUILD_GENERAL1_LARGE	WINDOWS_SERVER_2019_CONTAINER WINDOWS_SERVER_2022_CONTAINER WINDOWS_EC2	16 GiB	8	128 GB
Windows XLarg [1]	BUILD_GENERAL1_XLARGE	WINDOWS_SERVER_2022_CONTAINER	72 GiB	36	256GB
Windows 2XLar [1]	BUILD_GENERAL1_2XLARGE	WINDOWS_SERVER_2022_CONTAINER	144 GiB	72	824 GB

[1]缓存该映像类型的最新版本。如果您指定更具体的版本，则预 CodeBuild 置该版本而不是缓存的版本。这可能会导致构建时间更长。例如，要受益于缓存，请指定 `aws/codebuild/amazonlinux-x86_64-standard:5.0` 而不是更精细的版本，例如 `aws/codebuild/amazonlinux-x86_64-standard:5.0-1.0.0`。

AWS CodeBuild 为构建环境提供以下 AWS Lambda 计算模式的可用内存和磁盘空间：

计算类型	环境 computeType 值	环境类型值	内存	磁盘空间
ARM Lambda 1GB	BUILD_LAMBDA_1GB	ARM_LAMBDA_CONTAINER	1 GiB	10 GB
ARM Lambda 2GB	BUILD_LAMBDA_2GB	ARM_LAMBDA_CONTAINER	2 GiB	10 GB
ARM Lambda 4GB	BUILD_LAMBDA_4GB	ARM_LAMBDA_CONTAINER	4 GiB	10 GB
ARM Lambda 8GB	BUILD_LAMBDA_8GB	ARM_LAMBDA_CONTAINER	8 GiB	10 GB
ARM Lambda 10GI	BUILD_LAMBDA_10GB	ARM_LAMBDA_CONTAINER	10 GiB	10 GB
Linux Lambda 1GE	BUILD_LAMBDA_1GB	LINUX_LAMBDA_CONTAINER	1 GiB	10 GB
Linux Lambda 2GE	BUILD_LAMBDA_2GB	LINUX_LAMBDA_CONTAINER	2 GiB	10 GB
Linux Lambda 4GE	BUILD_LAMBDA_4GB	LINUX_LAMBDA_CONTAINER	4 GiB	10 GB
Linux Lambda 8GE	BUILD_LAMBDA_8GB	LINUX_LAMBDA_CONTAINER	8 GiB	10 GB

计算类型	环境 computeType 值	环境类型值	内存	磁盘空间
Linux Lambda 10G	BUILD_LAMBDA_10GB	LINUX_LAMBDA_CONTAINER	10 GiB	10 GB

使用其他环境类型时，建议您使用缓存的映像来缩短构建时间。

为每个构建环境列出的磁盘空间仅在 CODEBUILD_SRC_DIR 环境变量指定的目录中可用。

选择计算类型：

- 在 CodeBuild 控制台的 “创建构建项目” 向导或 “编辑构建项目” 页面中，在 “环境” 中展开 “其他配置”，然后从 “计算类型” 中选择一个选项。有关更多信息，请参阅 创建构建项目（控制台）或 更改构建项目的设置（控制台）。
- 对于 AWS CLI，运行create-project或update-project命令，指定environment对象的computeType值。有关更多信息，请参阅 创建构建项目 (AWS CLI)或 更改构建项目的设置 (AWS CLI)。
- 对于 AWS SDK，调用目标编程语言的CreateProject或UpdateProject运算的等效项，指定environment对象computeType值的等效值。有关更多信息，请参阅AWS SDK 和工具参考。

某些环境和计算类型存在区域可用性限制：

- 计算类型 Linux GPU 小型 (LINUX_GPU_CONTAINER) 仅在以下区域可用：
  - 美国东部（弗吉尼亚州北部）
  - 美国西部（俄勒冈）
  - 亚太地区（东京）
  - 加拿大（中部）
  - 欧洲地区（法兰克福）
  - 欧洲地区（爱尔兰）
  - 欧洲地区（伦敦）
- 计算类型 Linux GPU 大型 (LINUX_GPU_CONTAINER) 仅在以下区域可用：
  - 美国东部（俄亥俄州）
  - 美国东部（弗吉尼亚州北部）

- 美国西部（俄勒冈州）
- 亚太地区（首尔）
- 亚太地区（悉尼）
- 亚太地区（东京）
- 加拿大（中部）
- 中国（北京）
- 中国（宁夏）
- 欧洲地区（法兰克福）
- 欧洲地区（爱尔兰）
- 欧洲地区（伦敦）

- 计算类型 BUILD_GENERAL1_2XLARGE 仅在以下区域可用：
  - 美国东部（俄亥俄州）
  - 美国东部（弗吉尼亚州北部）
  - 美国西部（北加利福尼亚）
  - 美国西部（俄勒冈州）
  - 亚太地区（海得拉巴）
  - 亚太地区（香港）
  - 亚太地区（雅加达）
  - 亚太地区（墨尔本）
  - 亚太地区（孟买）
  - 亚太地区（首尔）
  - 亚太地区（新加坡）
  - 亚太地区（悉尼）
  - 亚太地区（东京）
  - 加拿大（中部）
  - 中国（北京）
  - 中国（宁夏）
  - 欧洲地区（法兰克福）
  - 欧洲地区（爱尔兰）
  - 欧洲地区（伦敦）

- Europe (Paris)
- 欧洲（西班牙）
- 欧洲地区（斯德哥尔摩）
- 欧洲（苏黎世）
- 以色列（特拉维夫）
- Middle East (Bahrain)
- 中东（阿联酋）：
- 南美洲（圣保罗）

- 环境类型 ARM_CONTAINER 仅在以下区域可用：
  - 美国东部（俄亥俄州）
  - 美国东部（弗吉尼亚州北部）
  - 美国西部（北加利福尼亚）
  - 美国西部（俄勒冈州）
  - 亚太地区（香港）
  - 亚太地区（雅加达）
  - 亚太地区（海得拉巴）
  - 亚太地区（马来西亚）
  - 亚太地区（孟买）
  - 亚太地区（新西兰）
  - 亚太地区（大阪）
  - 亚太地区（首尔）
  - 亚太地区（新加坡）
  - 亚太地区（悉尼）
  - 亚太地区（泰国）
  - 亚太地区（东京）
  - 加拿大（中部）
  - 中国（北京）
  - 中国（宁夏）
  - 欧洲地区（法兰克福）
  - 欧洲地区（爱尔兰）

- 欧洲地区（伦敦）
- 欧洲地区（米兰）
- 欧洲地区（巴黎）
- 欧洲（西班牙）
- 欧洲地区（斯德哥尔摩）
- 以色列（特拉维夫）
- Middle East (Bahrain)
- 中东（阿联酋）：
- 南美洲（圣保罗）

- 环境类型 WINDOWS_SERVER_2022_CONTAINER 仅在以下区域可用：
  - 美国东部（俄亥俄州）
  - 美国东部（弗吉尼亚州北部）
  - 美国西部（俄勒冈州）
  - 亚太地区（悉尼）
  - 亚太地区（东京）
  - 欧洲地区（法兰克福）
  - 欧洲地区（爱尔兰）
  - 南美洲（圣保罗）
- 环境类型 LINUX_EC2（BUILD_GENERAL1_SMALL、BUILD_GENERAL1_MEDIUM、BUILD_GENERAL1_LARGE）仅在以下区域可用：
  - 美国东部（俄亥俄州）
  - 美国东部（弗吉尼亚州北部）
  - 美国西部（北加利福尼亚）
  - 美国西部（俄勒冈州）
  - 非洲（开普敦）
  - 亚太地区（香港）
  - 亚太地区（雅加达）
  - 亚太地区（墨尔本）
  - 亚太地区（海得拉巴）
  - 亚太地区（马来西亚）

- 亚太地区（孟买）
- 亚太地区（新西兰）
- 亚太地区（大阪）
- 亚太地区（首尔）
- 亚太地区（新加坡）
- 亚太地区（悉尼）
- 亚太地区（泰国）
- 亚太地区（东京）
- 加拿大（中部）
- 中国（北京）
- 中国（宁夏）
- 欧洲地区（法兰克福）
- 欧洲地区（爱尔兰）
- 欧洲地区（伦敦）
- 欧洲地区（米兰）
- 欧洲地区（巴黎）
- 欧洲（西班牙）
- 欧洲地区（斯德哥尔摩）
- 欧洲（苏黎世）
- 以色列（特拉维夫）
- Middle East (Bahrain)
- 中东（阿联酋）：
- 南美洲（圣保罗）
- AWS GovCloud (US-West)
- AWS GovCloud (US-East)

- 环境类型
  `ARM_EC2`（`BUILD_GENERAL1_SMALL`、`BUILD_GENERAL1_MEDIUM`、`BUILD_GENERAL1_LARGE`）
  仅在以下区域可用：
  - 美国东部（俄亥俄州）
  - 美国东部（弗吉尼亚州北部）
  - 美国西部（北加利福尼亚）

- 美国西部（俄勒冈州）
- 亚太地区（香港）
- 亚太地区（雅加达）
- 亚太地区（海得拉巴）
- 亚太地区（马来西亚）
- 亚太地区（孟买）
- 亚太地区（新西兰）
- 亚太地区（大阪）
- 亚太地区（首尔）
- 亚太地区（泰国）
- 亚太地区（新加坡）
- 亚太地区（悉尼）
- 亚太地区（东京）
- 加拿大（中部）
- 中国（北京）
- 中国（宁夏）
- 欧洲地区（法兰克福）
- 欧洲地区（爱尔兰）
- 欧洲地区（伦敦）
- 欧洲地区（米兰）
- 欧洲地区（巴黎）
- 欧洲（西班牙）
- 欧洲地区（斯德哥尔摩）
- 欧洲（苏黎世）
- 以色列（特拉维夫）
- 中东（巴林）
- 南美洲（圣保罗）
- AWS GovCloud (US-West)
- AWS GovCloud (US-East)

- 环境类型 WINDOWS_EC2（BUILD_GENERAL1_MEDIUM、BUILD_GENERAL1_LARGE）仅在以下区域可用：
  - 美国东部（俄亥俄州）
  - 美国东部（弗吉尼亚州北部）
  - 美国西部（俄勒冈州）
  - 亚太地区（悉尼）
  - 亚太地区（东京）
  - 欧洲地区（法兰克福）
  - 欧洲地区（爱尔兰）
  - 南美洲（圣保罗）
- 计算模式 AWS Lambda （ARM_LAMBDA_CONTAINER和LINUX_LAMBDA_CONTAINER）仅在以下区域可用：
  - 美国东部（弗吉尼亚州北部）
  - 美国东部（俄亥俄州）
  - 美国西部（北加利福尼亚）
  - 美国西部（俄勒冈州）
  - 亚太地区（孟买）
  - 亚太地区（新加坡）
  - 亚太地区（悉尼）
  - 亚太地区（东京）
  - 欧洲地区（法兰克福）
  - 欧洲地区（爱尔兰）
  - 南美洲（圣保罗）
- 计算模式 MAC_ARM 仅在以下区域可用：
  - 美国东部（弗吉尼亚州北部）
  - 美国东部（俄亥俄州）
  - 美国西部（俄勒冈州）
  - 亚太地区（悉尼）
  - 欧洲地区（法兰克福）

对于计算类型 BUILD_GENERAL1_2XLARGE，支持高达 100 GB 的未压缩 Docker 映像。

> **Note**
>
> 对于自定义编译环境镜像，无论计算类型如何，在 Linux 和 Windows 中 CodeBuild 支持最大 50 GB 未压缩的 Docker 镜像。要检查构建映像的大小，请使用 Docker 运行 `docker images` *`REPOSITORY`*`:`*`TAG`* 命令。

您可以使用 Amazon EFS 在构建容器中访问更多空间。有关更多信息，请参阅的亚马逊弹性文件系统示例 AWS CodeBuild。如果您希望在构建期间操作容器磁盘空间，则构建必须运行在特权模式下。

> **Note**
>
> 默认情况下，为非 VPC 构建启用 Docker 进程守护程序。如果您想使用 Docker 容器进行 VPC 构建，请参阅 Docker 文档网站上的运行时权限和 Linux 功能并启用特权模式。此外，Windows 不支持特权模式。

构建环境中的 Shell 和命令

您可以提供一组命令， AWS CodeBuild 以便在构建的生命周期中在构建环境中运行（例如，安装构建依赖关系以及测试和编译源代码）。可通过多种方式指定这些命令：

- 创建构建规范文件并将其包含在您的源代码中。在此文件中，指定您要在构建生命周期的每个阶段运行的命令。有关更多信息，请参阅的构建规范参考 CodeBuild。
- 使用 CodeBuild 控制台创建构建项目。在插入构建命令中，对于构建命令，输入您要在 `build` 阶段运行的命令。有关更多信息，请参阅 创建构建项目（控制台）。
- 使用 CodeBuild 控制台更改构建项目的设置。在插入构建命令中，对于构建命令，输入您要在 `build` 阶段运行的命令。有关更多信息，请参阅 更改构建项目的设置（控制台）。
- 使用 AWS CLI 或 AWS 软件开发工具包创建构建项目或更改构建项目的设置。参考包含构建规范文件以及您的命令的源代码，或指定一个包含同等构建规范文件的内容的字符串。有关更多信息，请参阅 创建构建项目或 更改构建项目设置。
- 使用 AWS CLI 或 AWS 软件开发工具包开始构建，指定一个 buildspec 文件或包含等效 buildspec 文件内容的单个字符串。有关更多信息，请参阅 手动运行构建 中 `buildspecOverride` 值的描述。

您可以指定任何 Shell 命令语言 (sh) 命令。在 buildspec 版本 0.1 中，在编译环境的单独实例中 CodeBuild 运行每个 Shell 命令。这表示各个命令独立于其他所有命令而运行。因此，在默认情况下，您无法运行依赖所有上一个命令状态的单个命令（如更改目录或设置环境变量）。要绕开此限制，建议使用版本 0.2 来解决此问题。如果您必须使用版本 0.1，我们建议使用以下方法：

- 在包含您要在默认 Shell 的单个实例中运行的命令的源代码中包含一个 Shell 脚本。例如，您可以在包含 `cd MyDir; mkdir -p mySubDir; cd mySubDir; pwd;` 等命令的源代码中包含一个名为 `my-script.sh` 的文件。然后，在您的 buildspec 文件中，指定命令 `./my-script.sh`。
- 在您的 buildspec 文件中，或对于仅针对 `build` 阶段的构建命令设置，输入包含您想在默认 Shell 的单个实例中运行的所有命令的单个命令（例如，`cd MyDir && mkdir -p mySubDir && cd mySubDir && pwd`）。

如果 CodeBuild 遇到错误，与在自己的默认 shell 实例中运行单个命令相比，可能更难排除错误。

在 Windows 服务器核心映像中运行的命令使用 PowerShell 外壳。

构建环境中的环境变量

AWS CodeBuild 提供了几个可以在构建命令中使用的环境变量：

AWS_DEFAULT_REGION

运行构建的 AWS 区域（例如，`us-east-1`）。此环境变量主要由 AWS CLI使用。

AWS_REGION

运行构建的 AWS 区域（例如，`us-east-1`）。此环境变量主要由 AWS SDK 使用。

CODEBUILD_BATCH_BUILD_IDENTIFIER

批量构建中构建的标识符。这是在批处理 buildspec 中指定的。有关更多信息，请参阅the section called “批量 buildspec 参考”。

CODEBUILD_BUILD_ARN

构建的 Amazon 资源名称 (ARN) (例如，`arn:aws:codebuild:region-ID:account-ID:build/codebuild-demo-project:b1e6661e-e4f2-4156-9ab9-82a19EXAMPLE`)。

CODEBUILD_BUILD_ID

版本的 CodeBuild ID（例如，`codebuild-demo-project:b1e6661e-e4f2-4156-9ab9-82a19EXAMPLE`）。

CODEBUILD_BUILD_IMAGE

CodeBuild 构建映像标识符（例如，`aws/codebuild/standard:2.0`）。

CODEBUILD_BUILD_NUMBER

项目的当前构建编号。

CODEBUILD_BUILD_SUCCEEDING

无论当前构建是否成功。如果构建失败，设置为 `0`；如果构建成功，设置为 `1`。

CODEBUILD_INITIATOR

启动构建的实体。如果 CodePipeline 开始构建，则这是管道的名称（例如，`codepipeline/my-demo-pipeline`）。如果用户开始构建，则这是该用户的名称（例如，`MyUserName`）。如果的 Jenkins 插件 CodeBuild 开始构建，则这是字符串`CodeBuild-Jenkins-Plugin`。

CODEBUILD_KMS_KEY_ID

用于加密编译输出构 CodeBuild 件的 AWS KMS 密钥的标识符（例如，`arn:aws:kms:`*`region-ID`*`:`*`account-ID`*`:key/`*`key-ID`*或`alias/`*`key-alias`*）。

CODEBUILD_PROJECT_ARN

项目的 Amazon 资源名称（ARN）（例如 `arn:aws:codebuild:`*`region-ID`*`:`*`account-ID`*`:project/`*`project-name`*）。

CODEBUILD_PUBLIC_BUILD_URL

此构建在公共构建网站上的构建结果的 URL。仅当构建项目启用了公共构建时，才会设置此变量。有关更多信息，请参阅获取公共构建项目 URL。

CODEBUILD_RESOLVED_SOURCE_VERSION

构建的源代码的版本标识符。内容取决于源代码存储库：

CodeCommit、 GitHub、 GitHub 企业服务器和 Bitbucket

此变量包含提交 ID。

CodePipeline

此变量包含提供的源版本 CodePipeline。

如果 CodePipeline 无法解析源版本，例如当源是未启用版本控制的 Amazon S3 存储桶时，则不设置此环境变量。

Amazon S3

此变量未设置。

如果适用，该 CODEBUILD_RESOLVED_SOURCE_VERSION 变量仅在 DOWNLOAD_SOURCE 阶段之后才可用。

CODEBUILD_SOURCE_REPO_URL

输入构件或源代码存储库的 URL。对于 Amazon S3，这是 s3://，后跟存储桶名称和输入构件的路径。对于 CodeCommit 和 GitHub，这是存储库的克隆 URL。如果编译源自 CodePipeline，则此环境变量可能为空。

对于辅助源，辅助源存储库 URL 的环境变量是 CODEBUILD_SOURCE_REPO_URL_*<sourceIdentifier>*，其中 *<sourceIdentifier>* 是您创建的源标识符。

CODEBUILD_SOURCE_VERSION

值的格式取决于源存储库。

- 对于 Amazon S3，这是与输入构件关联的版本 ID。
- 对于 CodeCommit，它是与要构建的源代码版本相关的提交 ID 或分支名称。
- 对于 GitHub E GitHub nterprise Server 和 Bitbucket 来说，它是与要构建的源代码版本相关的提交 ID、分支名称或标签名称。

> **Note**
>
> 对于由 webhook 拉取请求事件触发的 GitHub 或 GitHub 企业服务器版本来说pr/*pull-request-number*，确实如此。

对于辅助源，辅助源版本的环境变量是 CODEBUILD_SOURCE_VERSION_*<sourceIdentifier>*，其中 *<sourceIdentifier>* 是您创建的源标识符。有关更多信息，请参阅多输入源和输出构件示例。

CODEBUILD_SRC_DIR

CodeBuild 用于构建的目录路径（例如，/tmp/src123456789/src）。

对于辅助源，辅助源目录的环境变量是 CODEBUILD_SRC_DIR_*<sourceIdentifier>*，其中 *<sourceIdentifier>* 是您创建的源标识符。有关更多信息，请参阅 多输入源和输出构件示例。

CODEBUILD_START_TIME

指定为 Unix 时间戳的构建开始时间（以毫秒为单位）。

CODEBUILD_WEBHOOK_ACTOR_ACCOUNT_ID

触发 Webhook 事件的用户的账户 ID。

CODEBUILD_WEBHOOK_BASE_REF

触发当前构建的 Webhook 事件的基本引用名称。对于拉取请求，这是分支引用。

CODEBUILD_WEBHOOK_EVENT

触发当前构建的 Webhook 事件。

CODEBUILD_WEBHOOK_MERGE_COMMIT

用于构建的合并提交的标识符。将 Bitbucket 拉取请求与压缩策略合并且拉取请求分支关闭时，会设置该变量。在这种情况下，原始拉取请求提交不再存在，该环境变量将包含压缩后的合并提交的标识符。

CODEBUILD_WEBHOOK_PREV_COMMIT

在触发当前构建的 Webhook 推送事件之前最新提交的 ID。

CODEBUILD_WEBHOOK_HEAD_REF

触发当前构建的 Webhook 事件的头部引用名称。它可以是分支引用或标签引用。

CODEBUILD_WEBHOOK_TRIGGER

显示触发构建的 Webhook 事件。此变量仅适用于 Webhook 触发的构建。该值是从发送给 GitHub 企业服务器或 Bitbucket CodeBuild 的 GitHub负载中解析出来的。该值的格式取决于触发构建的事件类型。

- 对于拉取请求触发的构建，这是 `pr/pull-request-number`。
- 对于通过创建新分支或将提交操作推送到分支而触发的构建，这是 `branch/branch-name`。
- 对于通过将标签推送到存储库而触发的构建，这是 `tag/tag-name`。

HOME

此环境变量始终设置为 `/root`。

AWS CodeBuild 还支持一组用于自托管运行器构建的环境变量。要了解有关 CodeBuild 自托管运行器的更多信息，请参阅教程：配置 CodeBuild-hosted GitHub 操作运行器。

CODEBUILD_RUNNER_OWNER

触发自托管运行器构建的存储库的拥有者。

CODEBUILD_RUNNER_REPO

触发自托管运行器构建的存储库的名称。

CODEBUILD_RUNNER_REPO_DOMAIN

触发自托管运行器构建的存储库的域。仅限指定的 GitHub 企业版本。

CODEBUILD_WEBHOOK_LABEL

用于在构建期间配置构建覆盖和自托管运行器的标签。

CODEBUILD_WEBHOOK_RUN_ID

与构建关联的工作流的运行 ID。

CODEBUILD_WEBHOOK_JOB_ID

与构建关联的作业的作业 ID。

CODEBUILD_WEBHOOK_WORKFLOW_NAME

与构建关联的工作流的名称（如果存在于 webhook 请求有效载荷中）。

CODEBUILD_RUNNER_WITH_BUILDSPEC

如果在自托管运行器请求标签中配置了 buildspec 覆盖，则将其设置为 `true`。

您也可以为构建环境提供您自己的环境变量。有关更多信息，请参阅以下主题：

- CodeBuild 搭配使用 CodePipeline
- 创建构建项目
- 更改构建项目设置
- 手动运行构建
- Buildspec 参考

要列出构建环境中的所有可用环境变量，可以在构建期间运行该`printenv`命令（用于 Linux-based 构建环境）或`"Get-ChildItem Env:"`（对于 Windows-based 构建环境）。除了前面列出的环境变量外，以开头的`CODEBUILD_`环境变量供 CodeBuild 内部使用。它们不应用于您的构建命令。

> **Important**
>
> 我们强烈建议使用环境变量来存储敏感值，尤其是 AWS 访问密钥 ID。可以使用 CodeBuild 控制台和等工具以纯文本显示环境变量 AWS CLI。
> 我们建议您将敏感值存储在 Amazon EC2 Systems Manager Parameter Store 中，然后从您的 buildspec 中检索它们。要存储敏感值，请参阅《Amazon EC2 Systems Manager 用户指南》中的 Systems Manager Parameter Store 和演练：创建和测试参数（控制台）。要检索它们，请参阅buildspec 语法中的 `parameter-store` 映射。

CODEBUILD_BUILD_URL

此构建的构建结果的 URL。

构建环境中的后台任务

您可以在构建环境中运行后台任务。要执行此操作，请在您的 buildspec 中，使用 nohup 命令将命令作为后台中的任务运行，即使构建过程已退出 Shell 也是如此。使用 disown 命令强制停止正在运行的后台任务。

示例：

- 启动后台进程并等待其稍后完成：

```
|
nohup sleep 30 & echo $! > pidfile
…
wait $(cat pidfile)
```

- 启动后台进程，但不等待其完成：

```
|
nohup sleep 30 & disown $!
```

- 启动后台进程并稍后将其终止：

```
|
nohup sleep 30 & echo $! > pidfile
…
```

```
kill $(cat pidfile)
```

构建项目

构建项目包含有关如何运行构建的信息，包括从何处获取源代码、要使用的构建环境、要运行的构建命令以及将构建输出存储在何处。

在使用构建项目时，您可以执行以下任务：

主题

- 在中创建构建项目 AWS CodeBuild
- 创建通知规则
- 在中更改构建项目设置 AWS CodeBuild
- 里面有多个访问令牌 CodeBuild
- 删除中的构建项目 AWS CodeBuild
- 获取公共构建项目 URL
- 共享构建项目
- 标记构建项目
- 将跑步器与 AWS CodeBuild
- 将 webhook 与 AWS CodeBuild
- 在中查看构建项目的详细信息 AWS CodeBuild
- 在中查看构建项目名称 AWS CodeBuild

在中创建构建项目 AWS CodeBuild

您可以使用 AWS CodeBuild 控制台或 AWS 软件开发工具包来创建构建项目。 AWS CLI

主题

- 先决条件
- 创建构建项目（控制台）
- 创建构建项目 (AWS CLI)
- 创建构建项目 (AWS 软件开发工具包)
- 创建构建项目 (CloudFormation)

先决条件

在创建构建项目之前，请回答计划构建中的问题。

创建构建项目（控制台）

打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。

如果显示 CodeBuild 信息页面，请选择“创建构建项目”。否则，请在导航窗格中，展开构建，选择构建项目，然后选择创建构建项目。

选择创建构建项目。

填写以下部分：完成后，选择页面底部的创建构建项目。

部分：

- 项目配置
- 源
- 环境
- BuildSpec
- 批量配置
- 构件
- 日志

项目配置

项目名称

输入此构建项目的名称。每个 AWS 账户中的构建项目名称必须是唯一的。

描述

输入构建项目的可选描述，以帮助其他用户了解此项目的用途。

构建徽章

（可选）选择启用构建徽章，以使您的项目的构建状态可见且可嵌入。有关更多信息，请参阅 构建徽章示例。

> **Note**
>
> 如果您的源提供商是 Amazon S3，则构建徽章不适用。

启用并发构建限制

（可选）如果要限制此项目的并发构建数量，请执行以下步骤：

1. 选择限制此项目可以启动的并发构建数量。
2. 在并发构建限制中，输入此项目允许的并发构建的最大数量。此限制不得大于为该账户设置的并发构建限制。如果您尝试输入大于账户限制的数字，则会显示错误消息。

仅当当前构建数量小于或等于此限值时，才会启动新构建。如果当前构建计数达到此限值，则新构建将受到限制且不会运行。

其他信息

（可选）在“标签”中，输入您希望支持 AWS 服务使用的任何标签的名称和值。使用添加行添加标签。最多可以添加 50 个标签。

源

源提供商

选择源代码提供商类型。使用以下列表为您的源提供商选择适当的选项：

> **Note**
>
> CodeBuild 不支持 Bitbucket 服务器。

Amazon S3

存储桶

选择包含源代码的输入存储桶的名称。

S3 对象密钥或 S3 文件夹

输入 ZIP 文件的名称或包含源代码的文件夹的路径。输入正斜杠 (/) 以下载 S3 存储桶中的所有内容。

源版本

输入表示输入文件版本的对象的版本 ID。有关更多信息，请参阅 源版本示例，包含 AWS CodeBuild。

CodeCommit

存储库

选择要使用的存储库。

参考类型

选择分支、Git 标签或提交 ID，以指定源代码的版本。有关更多信息，请参阅 源版本示例，包含 AWS CodeBuild。

> **Note**
>
> 我们建议您选择看起来不像提交 ID 的 Git 分支名称，例如 `811dd1ba1aba14473856cee38308caed7190c0d` 或 `5392f7`。这可以帮助您避免 Git 签出与实际提交发生冲突。

Git 克隆深度

选择该选项，以创建一个浅克隆，其历史记录会截断至指定数量的提交。如果您需要完整克隆，请选择完整。

Git 子模块

如果您希望在存储库中包含 Git 子模块，请选择使用 Git 子模块。

Bitbucket

凭证

选择默认来源凭证或自定义来源凭证，然后按照说明管理默认来源凭证或自定义源凭证。

连接类型

选择要AWS CodeConnections连接的 OAuth、应用程序密码或个人访问令牌。 CodeBuild

Connection

选择 Bitbucket 连接或 Secrets Manager 密钥，通过指定的连接类型进行连接。

存储库

选择我的 Bitbucket 账户中的存储库或公共存储库，然后输入存储库 URL。

源版本

输入分支、提交 ID、标签，或引用以及提交 ID。有关更多信息，请参阅 源版本示例，包含 AWS CodeBuild。

> **Note**
>
> 我们建议您选择看起来不像提交 ID 的 Git 分支名称，例如 `811dd1ba1aba14473856cee38308caed7190c0d` 或 `5392f7`。这可以帮助您避免 Git 签出与实际提交发生冲突。

Git 克隆深度

选择Git 克隆深度以创建一个浅克隆，其历史记录会截断至指定数量的提交。如果您需要完整克隆，请选择完整。

Git 子模块

如果您希望在存储库中包含 Git 子模块，请选择使用 Git 子模块。

构建状态

如果您希望向源提供商报告构建的开始和完成状态，请选择在您的构建开始和完成时向源提供商报告构建状态。

为了能够向源提供商报告构建状态，与源提供商关联的用户必须拥有对存储库的写入权限。如果用户没有写入权限，则无法更新构建状态。有关更多信息，请参阅源提供商访问权限。

在状态上下文中，输入要在 Bitbucket 提交状态中用于 `name` 参数的值。有关更多信息，请参阅 Bitbucket API 文档中的构建。

在目标 URL 中，输入要在 Bitbucket 提交状态中用于 `url` 参数的值。有关更多信息，请参阅 Bitbucket API 文档中的构建。

由 Webhook 触发的构建的状态将始终报告给源提供商。要将从控制台或 API 调用启动的构建状态报告给源提供商，您必须选择此设置。

如果项目的构建通过 webhook 触发，则必须将新的提交推送到存储库，此设置才能生效。

如果要在每次将代码更改推送到此存储库时生成源代码，请在 “主源 webhook 事件” 中，选择 “每次将代码更改推送到此存储库时都重建”。 CodeBuild 有关 webhook 和筛选条件组的更多信息，请参阅 Bitbucket Webhook 事件。

GitHub

凭证

选择默认来源凭证或自定义来源凭证，然后按照说明管理默认来源凭证或自定义源凭证。

连接类型

选择要连接的GitHub 应用程序、OAuth 或个人访问令牌。 CodeBuild

Connection

选择要通过您指定的 GitHub 连接类型进行连接或 Secrets Manager 密钥进行连接。

存储库

在我的 GitHub 账户中选择 “存储库”、“公共存储库” 或 “GitHub 限定范围 webhook”，然后输入存储库 URL。

源版本

输入分支、提交 ID、标签，或引用以及提交 ID。有关更多信息，请参阅 源版本示例，包含 AWS CodeBuild。

> **Note**
>
> 我们建议您选择看起来不像提交 ID 的 Git 分支名称，例如 `811dd1ba1aba14473856cee38308caed7190c0d` 或 `5392f7`。这可以帮助您避免 Git 签出与实际提交发生冲突。

Git 克隆深度

选择Git 克隆深度以创建一个浅克隆，其历史记录会截断至指定数量的提交。如果您需要完整克隆，请选择完整。

Git 子模块

如果您希望在存储库中包含 Git 子模块，请选择使用 Git 子模块。

构建状态

如果您希望向源提供商报告构建的开始和完成状态，请选择在您的构建开始和完成时向源提供商报告构建状态。

为了能够向源提供商报告构建状态，与源提供商关联的用户必须拥有对存储库的写入权限。如果用户没有写入权限，则无法更新构建状态。有关更多信息，请参阅 源提供商访问权限。

在状态上下文中，输入要用于 GitHub 提交状态的`context`参数的值。有关更多信息，请参阅 GitHub 开发者指南中的创建提交状态。

在 “目标 URL” 中，输入要用于 GitHub 提交状态的`target_url`参数的值。有关更多信息，请参阅 GitHub 开发者指南中的创建提交状态。

由 Webhook 触发的构建的状态将始终报告给源提供商。要将从控制台或 API 调用启动的构建状态报告给源提供商，您必须选择此设置。

如果项目的构建通过 webhook 触发，则必须将新的提交推送到存储库，此设置才能生效。

如果要在每次将代码更改推送到此存储库时生成源代码，请在 “主源 webhook 事件” 中，选择 “每次将代码更改推送到此存储库时都重建”。 CodeBuild 有关 webhook 和筛选条件组的更多信息，请参阅 GitHub webhook 事件。

GitHub Enterprise Server

凭证

选择默认来源凭证或自定义来源凭证，然后按照说明管理默认来源凭证或自定义源凭证。

连接类型

选择AWS CodeConnections要连接的个人访问令牌 CodeBuild。

Connection

选择 GitHub 企业连接或 Secrets Manager 密钥通过您指定的连接类型进行连接。

存储库

在我的 GitHub 企业帐户中选择存储库或GitHub 企业级范围的 webhook，然后输入存储库 URL。

源版本

输入拉取请求、分支、提交 ID、标签，或引用以及提交 ID。有关更多信息，请参阅 源版本示例，包含 AWS CodeBuild。

> **Note**
>
> 我们建议您选择看起来不像提交 ID 的 Git 分支名称，例如 `811dd1ba1aba14473856cee38308caed7190c0d` 或 `5392f7`。这可以帮助您避免 Git 签出与实际提交发生冲突。

Git 克隆深度

选择Git 克隆深度以创建一个浅克隆，其历史记录会截断至指定数量的提交。如果您需要完整克隆，请选择完整。

Git 子模块

如果您希望在存储库中包含 Git 子模块，请选择使用 Git 子模块。

构建状态

如果您希望向源提供商报告构建的开始和完成状态，请选择在您的构建开始和完成时向源提供商报告构建状态。

为了能够向源提供商报告构建状态，与源提供商关联的用户必须拥有对存储库的写入权限。如果用户没有写入权限，则无法更新构建状态。有关更多信息，请参阅 源提供商访问权限。

在状态上下文中，输入要用于 GitHub 提交状态的`context`参数的值。有关更多信息，请参阅 GitHub 开发者指南中的创建提交状态。

在 “目标 URL” 中，输入要用于 GitHub 提交状态的`target_url`参数的值。有关更多信息，请参阅 GitHub 开发者指南中的创建提交状态。

由 Webhook 触发的构建的状态将始终报告给源提供商。要将从控制台或 API 调用启动的构建状态报告给源提供商，您必须选择此设置。

如果项目的构建通过 webhook 触发，则必须将新的提交推送到存储库，此设置才能生效。

不安全的 SSL

选择 “启用不安全 SSL”，以便在连接到 GitHub 企业项目存储库时忽略 SSL 警告。

如果要在每次将代码更改推送到此存储库时生成源代码，请在 “主源 webhook 事件” 中，选择 “每次将代码更改推送到此存储库时都重建”。 CodeBuild 有关 webhook 和筛选条件组的更多信息，请参阅 GitHub webhook 事件。

GitLab

凭证

选择默认来源凭证或自定义来源凭证，然后按照说明管理默认来源凭证或自定义源凭证。

连接类型

AWS CodeConnections用于连接 GitLab 到 CodeBuild。

Connection

选择要 GitLab 连接的连接 AWS CodeConnections。

存储库

选择要使用的存储库。

源版本

输入拉取请求 ID、分支、提交 ID、标签，或引用以及提交 ID。有关更多信息，请参阅 源版本示例，包含 AWS CodeBuild。

> **Note**
>
> 我们建议您选择看起来不像提交 ID 的 Git 分支名称，例如 `811dd1ba1aba14473856cee38308caed7190c0d` 或 `5392f7`。这可以帮助您避免 Git 签出与实际提交发生冲突。

Git 克隆深度

选择Git 克隆深度以创建一个浅克隆，其历史记录会截断至指定数量的提交。如果您需要完整克隆，请选择完整。

构建状态

如果您希望向源提供商报告构建的开始和完成状态，请选择在您的构建开始和完成时向源提供商报告构建状态。

为了能够向源提供商报告构建状态，与源提供商关联的用户必须拥有对存储库的写入权限。如果用户没有写入权限，则无法更新构建状态。有关更多信息，请参阅 源提供商访问权限。

GitLab Self Managed

凭证

选择默认来源凭证或自定义来源凭证，然后按照说明管理默认来源凭证或自定义源凭证。

连接类型

AWS CodeConnections用于将 GitLab 自助管理连接到 CodeBuild。

Connection

选择要连接的 GitLab 自管理连接 AWS CodeConnections。

存储库

选择要使用的存储库。

源版本

输入拉取请求 ID、分支、提交 ID、标签，或引用以及提交 ID。有关更多信息，请参阅 源版本示例，包含 AWS CodeBuild。

> **Note**
> 我们建议您选择看起来不像提交 ID 的 Git 分支名称，例如 `811dd1ba1aba14473856cee38308caed7190c0d` 或 `5392f7`。这可以帮助您避免 Git 签出与实际提交发生冲突。

Git 克隆深度

选择Git 克隆深度以创建一个浅克隆，其历史记录会截断至指定数量的提交。如果您需要完整克隆，请选择完整。

构建状态

如果您希望向源提供商报告构建的开始和完成状态，请选择在您的构建开始和完成时向源提供商报告构建状态。

为了能够向源提供商报告构建状态，与源提供商关联的用户必须拥有对存储库的写入权限。如果用户没有写入权限，则无法更新构建状态。有关更多信息，请参阅 源提供商访问权限。

环境

预置模型

请执行以下操作之一：

- 要使用由管理的按需队列 AWS CodeBuild，请选择On-demand。使用按需队列，为您的构建 CodeBuild 提供计算。构建完成后，机器就会被销毁。 On-demand 舰队是完全托管的，并包括自动扩展功能，以应对需求激增。
- 要使用由管理的预留容量队列 AWS CodeBuild，请选择预留容量，然后选择队列名称。使用预留容量实例集，您可以为构建环境配置一组专用实例。这些计算机保持闲置状态，可以立即处理生成或测试，并缩短构建持续时间。使用预留容量队列，您的计算机将始终处于运行状态，并且只要配置完毕，它们就会继续产生成本。

有关信息，请参阅在预留容量实例集上运行构建。

环境映像

请执行以下操作之一：

- 要使用由管理的 Docker 映像 AWS CodeBuild，请选择托管映像，然后从 “操作系统”、“运行时”、“映像” 和 “映像版本” 中进行选择。从环境类型中进行选择（如果可用）。
- 要使用其他 Docker 映像，请选择自定义映像。对于环境类型，请选择 ARM、Linux、Linux GPU 或 Windows。如果您针对外部注册表 URL 选择其他注册表，请使用 *docker repository*/*docker image name* 格式在 Docker Hub 中输入 Docker 映像的名称和标签。如果您选择 Amazon ECR，请使用亚马逊 ECR 存储库和 A mazon ECR 镜像在您的账户中选择 Docker 镜像。 AWS
- 要使用私有 Docker 映像，请选择自定义映像。对于环境类型，请选择 ARM、Linux、Linux GPU 或 Windows。对于映像注册表，选择其他注册表，然后输入您的私有 Docker 映像的凭证的 ARN。凭证必须由 Secrets Manager 创建。有关更多信息，请参阅《AWS Secrets Manager 用户指南》中的什么是 AWS Secrets Manager？。

> **Note**
>
> CodeBuild 会替换自定义 Docker 镜像的。`ENTRYPOINT`

计算

请执行以下操作之一：

- 要使用 EC2 计算，请选择 EC2。EC2 计算在操作运行期间提供了优化的灵活性。
- 要使用 Lambda 计算，请选择 Lambda。Lambda 计算为构建提供优化的启动速度。由于启动延迟较短，Lambda 支持更快的构建。Lambda 还会自动扩展，因此构建无需在队列中等待运行。有关更多信息，请参阅 在 build 上运行 AWS Lambda 计算。

服务角色

请执行以下操作之一：

- 如果您没有 CodeBuild 服务角色，请选择 “新建服务角色”。在角色名称中，为新角色输入名称。
- 如果您有 CodeBuild 服务角色，请选择现有服务角色。在角色 ARN 中，选择服务角色。

> **Note**
>
> 使用控制台创建构建项目时，可以同时创建 CodeBuild 服务角色。默认情况下，这个角色仅能与该构建项目配合使用。如果您使用控制台将此服务角色与另一个构建项目关联，则此角色将更新以便与关联的构建项目结合使用。一个服务角色最多可与 10 个构建项目结合使用。

其他配置

Auto-retry limit

指定构建失败后额外的自动重试次数。例如，如果自动重试限制设置为 2，则 CodeBuild 会调用 `RetryBuild` API 自动再重试构建 2 次。

超时

指定一个介于 5 分钟到 36 小时之间的值，如果构建未完成，则在该值之后 CodeBuild 停止构建。如果小时和分钟都留空，则将使用 60 分钟的默认值。

特权

（可选）仅当您打算使用此构建项目来构建 Docker 映像时，才应选择如果要构建 Docker 映像或希望您的构建获得提升的特权，请启用此标志。否则，尝试与 Docker 守护程序交互的所有关联的构建都将失败。您还必须启动 Docker 守护程序，以便您的构建与其交互。一种方法是在构建规范`install`阶段通过运行以下构建命令来初始化 Docker 守护程序。如果您选择了由 CodeBuild Docker 支持的构建环境镜像，请不要运行这些命令。

> **Note**
>
> 默认情况下，为非 VPC 构建启用 Docker 进程守护程序。如果您想使用 Docker 容器进行 VPC 构建，请参阅 Docker 文档网站上的运行时权限和 Linux 功能并启用特权模式。此外，Windows 不支持特权模式。

```
- nohup /usr/local/bin/dockerd --host=unix:///var/run/docker.sock --
host=tcp://127.0.0.1:2375 --storage-driver=overlay2 &
- timeout 15 sh -c "until docker info; do echo .; sleep 1; done"
```

VPC

如果您 CodeBuild 想使用您的 VPC，请执行以下操作：

- 对于 VPC，请选择 CodeBuild 使用的 VPC ID。
- 对于 VPC 子网，请选择包含使用的 CodeBuild 资源的子网。
- 对于 VPC 安全组，请选择 CodeBuild 用于允许访问 VPC 中资源的安全组。

有关更多信息，请参阅 AWS CodeBuild 与亚马逊 Virtual Private Cloud 配合使用。

计算

请选择可用选项之一。

注册表凭证

使用非私有注册表映像配置项目时，请指定注册表凭证。

> **Note**
>
> 仅当映像被私有注册表中的映像覆盖时，才会使用此凭证。

环境变量

请输入每个环境变量的名称和值，然后选择类型，以供构建使用。

> **Note**
>
> CodeBuild 自动为您的 AWS 地区设置环境变量。如果您尚未将以下环境变量添加到 buildspec.yml 中，则必须设置这些变量：

- AWS_ACCOUNT_ID
- IMAGE_REPO_NAME
- IMAGE_TAG

控制台和 AWS CLI 用户可以看到环境变量。如果您不担心环境变量的可见性，请设置名称和值字段，然后将类型设置为明文。

我们建议您将具有敏感值的环境变量（例如访问密钥 ID、私有 AWS 访问 AWS 密钥或密码）作为参数存储在 Amazon EC2 Systems Manager Parameter Store 或 AWS Secrets Manager。

如果您使用的是 Amazon EC2 Systems Manager Parameter Store，则对于类型，请选择参数。在名称中，输入 CodeBuild 要引用的标识符。对于值，请按照 Amazon EC2 Systems Manager Parameter Store 中存储的参数名称输入参数名称。使用名为 `/CodeBuild/dockerLoginPassword` 的参数作为示例，对于类型，选择参数。对于名称，请输入 `LOGIN_PASSWORD`。对于值，请输入 `/CodeBuild/dockerLoginPassword`。

Important

如果您使用 Amazon EC2 Systems Manager Parameter Store，我们建议您使用以 `/CodeBuild/` 开头的参数名称（例如，`/CodeBuild/dockerLoginPassword`）来存储参数。您可以使用 CodeBuild 控制台在 Amazon EC2 Systems Manager 中创建参数。选择创建参数，然后按照对话框中的说明操作。（在该对话框中，对于 KMS 密钥，您可以指定账户中 AWS KMS 密钥的 ARN。 Amazon EC2 Systems Manager 使用此密钥在存储期间加密参数的值，并在检索期间对其进行解密。）如果您使用 CodeBuild 控制台创建参数，则控制台会将参数名称的存储方式`/CodeBuild/`作为参数名称的开头。有关更多信息，请参阅《Amazon EC2 Systems Manager 用户指南》中的 Systems Manager Parameter Store 和 Systems Manager Parameter Store 控制台演练。

如果您的构建项目引用了 Amazon EC2 Systems Manager Parameter Store 中存储的参数，则构建项目的服务角色必须允许 `ssm:GetParameters` 操作。如果您之前选择了“新建服务角色”，请将此操作 CodeBuild 包含在构建项目的默认服务角色中。但是，如果您选择了现有服务角色，必须单独将此操作添加到您的服务角色中。

如果您的构建项目引用了 Amazon EC2 Systems Manager Parameter Store 中存储的但参数名称不以 `/CodeBuild/` 开头的参数，且您选择了新建服务角色，您必须更新该服务角色以允许访问不以 `/CodeBuild/` 开头的参数名称。这是因为该服务角色仅允许访问以 `/CodeBuild/` 开头的参数名称。

如果您选择新建服务角色，服务角色将拥有解密 Amazon EC2 Systems Manager Parameter Store 中 /CodeBuild/ 命名空间下的所有参数的权限。
您设置的环境变量将替换现有的环境变量。例如，如果 Docker 映像已经包含一个名为 MY_VAR 的环境变量（值为 my_value），并且您设置了一个名为 MY_VAR 的环境变量（值为 other_value），那么 my_value 将被替换为 other_value。同样，如果 Docker 映像已经包含一个名为 PATH 的环境变量（值为 /usr/local/sbin:/usr/local/bin），并且您设置了一个名为 PATH 的环境变量（值为 $PATH:/usr/share/ant/bin），那么/usr/local/sbin:/usr/local/bin 将被替换为文本值 $PATH:/usr/share/ant/bin。
请勿使用以 CODEBUILD_ 打头的名称设置任何环境变量。此前缀是专为内部使用预留的。
如果具有相同名称的环境变量在多处都有定义，则应按照如下方式确定其值：

- 构建操作调用开始时的值优先级最高。
- 构建项目定义中的值优先级次之。
- buildspec 声明中的值优先级最低。

如果您使用 Secrets Manager，对于类型，请选择 Secrets Manager。在名称中，输入 CodeBuild 要引用的标识符。对于值，请使用模式 *secret-id*:*json-key*:*version-stage*:*version-id* 输入 reference-key。有关信息，请参阅 Secrets Manager reference-key in the buildspec file。

Important

如果您使用 Secrets Manager，我们建议您存储名称以 /CodeBuild/（例如 /CodeBuild/dockerLoginPassword）开头的密钥。有关更多信息，请参阅《AWS Secrets Manager 用户指南》中的什么是 AWS Secrets Manager？。
如果您的构建项目引用了 Secrets Manager 中存储的密钥，则构建项目的服务角色必须允许 secretsmanager:GetSecretValue 操作。如果您之前选择了“新建服务角色”，请将此操作 CodeBuild 包含在构建项目的默认服务角色中。但是，如果您选择了现有服务角色，必须单独将此操作添加到您的服务角色中。
如果您的构建项目引用了 Secrets Manager 中存储的但密钥名称不以 /CodeBuild/ 开头的密钥，且您选择了新建服务角色，您必须更新该服务角色以允许访问不以 /CodeBuild/ 开头的密钥名称。这是因为该服务角色仅允许访问以 /CodeBuild/ 开头的密钥名称。

如果您选择新建服务角色，该服务角色将拥有解密 Secrets Manager 中 `/CodeBuild/` 命名空间下的所有密钥的权限。

BuildSpec

构建规范

请执行以下操作之一：

- 如果您的源代码包含 buildspec 文件，请选择使用 buildspec 文件。默认情况下， CodeBuild 在源代码根目录中查找名为 `buildspec.yml` 的文件。如果您的 buildspec 文件使用其他名称或位置，请在 Buildspec 名称中输入其从源根目录开始的路径（例如，`buildspec-two.yml` 或 `configuration/buildspec.yml`。如果 buildspec 文件位于 S3 存储桶中，则该存储桶必须位于您的构建项目所在的同一 AWS 区域中。使用 ARN（例如 `arn:aws:s3:::<my-codebuild-sample2>/buildspec.yml`）指定该 buildspec 文件。
- 如果您的源代码不包括 buildspec 文件，或者如果您要运行的构建命令不是在源代码根目录的 `buildspec.yml` 文件中为 `build` 阶段指定的构建命令，则选择插入构建命令。对于构建命令，请输入您要在 `build` 阶段运行的命令。对于多个命令，使用 `&&` 分开各个命令（例如 `mvn test && mvn package`）。要在其他阶段运行命令，或者，如果 `build` 阶段对应的命令列表特别长，请将 `buildspec.yml` 文件添加到源代码根目录，将命令添加到该文件中，然后选择在源代码根目录中使用 buildspec.yml。

有关更多信息，请参阅Buildspec 参考。

批量配置

您可以将一组构建作为单个操作来运行。有关更多信息，请参阅 批量运行构建。

定义批量配置

选择该选项会允许在此项目中进行批量构建。

批量服务角色

为批量构建提供服务角色。

选择下列选项之一：

- 如果您没有批量服务角色，请选择新建服务角色。在服务角色中，为新角色输入名称。
- 如果您拥有批量服务角色，请选择现有服务角色。在服务角色中，选择对应的服务角色。

批量构建为批量配置引入了全新的安全角色。这个新角色是必需的，因为 CodeBuild 必须能够代表你调用StartBuildStopBuild、和RetryBuild操作才能将生成作为批处理的一部分运行。客户应该使用新角色，而不是他们在构建中使用的角色，原因有两个：

- 向构建角色授予 StartBuild、StopBuild 和 RetryBuild 权限后，将允许单个构建通过 buildspec 启动多个构建。
- CodeBuild 批处理生成提供了限制，限制了可用于批次构建的生成数量和计算类型。如果构建角色拥有这些权限，则构建本身就有可能绕过这些限制。

批量支持的计算类型

选择批处理允许的计算类型。选择所有适用的选项。

批量支持的实例集

选择批量支持的实例集。选择所有适用的选项。

批处理允许的最大构建数量

输入批处理允许的最大构建数量。如果批处理超过此限制，则会失败。

批处理超时

输入完成批量构建能够使用的最长时间。

合并构件

选择将批处理中的所有构件合并到一个位置，将批处理中的所有构件合并到一个位置。

批量报告模式

为批量构建选择所需的构建状态报告模式。

> **Note**
>
> 仅当项目源为 Bitbucket 或 E GitHub nterprise 时，此字段才可用，并且在 “来源” 下选择了生成开始和完成时向源提供商报告构建状态。 GitHub

聚合构建

选择该选项，可将批处理中所有构建的状态合并到一个状态报告中。

单个构建

选择该选项，可分别报告批处理中所有构建的构建状态。

构件

类型

请执行以下操作之一：

- 如果您不想创建任何构建输出构件，请选择无构件。如果您只运行构建测试，或者您要将 Docker 映像推送到 Amazon ECR 存储库，建议执行此操作。
- 要将构建输出存储在 S3 存储桶中，请选择 Amazon S3，然后执行以下操作：
  - 如果要将项目名称用于构建输出 ZIP 文件或文件夹，请将名称留空。否则，请输入名称。（如果您要输出 ZIP 文件，并且要让 ZIP 文件包含文件扩展名，请务必在 ZIP 文件名之后添加扩展名。）
  - 如果希望构建规范文件中指定的名称覆盖控制台中指定的任何名称，请选择启用语义版本控制。buildspec 文件中的名称是构建时计算得出的，使用 Shell 命令语言。例如，您可以将日期和时间附加到您的构件名称后面，以便确保其唯一性。为构件提供唯一名称可防止其被覆盖。有关更多信息，请参阅buildspec 语法。
  - 对于存储桶名称，请选择输出存储桶的名称。
  - 如果您在此过程的前面部分选择了插入构建命令，那么对于输出文件，请输入构建（该构建要放到构建输出 ZIP 文件或文件夹中）中的文件位置。对于多个位置，使用逗号将各个位置隔开（例如，`appspec.yml, target/my-app.jar`）。有关更多信息，请参阅buildspec 语法中 `files` 的描述。
  - 如果不想加密构建构件，请选择删除构件加密。

对于所需的每个辅助构件集：

1. 对于构件标识符，输入少于 128 个字符且仅包含字母数字字符和下划线的值。
2. 选择添加构件。
3. 按照前面步骤的说明配置辅助构件。
4. 选择保存构件。

其他配置

加密密钥

（可选）执行以下操作之一：

- 要使用您的账户中的 AWS 托管式密钥 Amazon S3 加密构建输出构件，请将加密密钥留空。这是默认值。

- 要使用客户托管密钥加密构建输出构件，请在加密密钥中输入 KMS 密钥的 ARN。采用格式 `arn:aws:kms:region-ID:account-ID:key/key-ID`。

缓存类型

对于缓存类型，请选择下列选项之一：

- 如果您不想使用缓存，请选择无缓存。
- 如果要使用 Amazon S3 缓存，请选择 Amazon S3，然后执行以下操作：
  - 对于存储桶，选择存储缓存的 S3 存储桶的名称。
  - （可选）对于缓存路径前缀，输入 Amazon S3 路径前缀。缓存路径前缀值类似于目录名称。它使您能够在存储桶的同一目录下存储缓存。

> **Important**
>
> 请勿将尾部斜杠 (/) 附加到路径前缀后面。

- 如果想要使用本地缓存，请选择本地，然后选择一个或多个本地缓存模式。

> **Note**
>
> Docker 层缓存模式仅适用于 Linux。如果您选择该模式，您的项目必须在特权模式下运行。

使用缓存可节省大量构建时间，因为构建环境的可重用部分被存储在缓存中，并且可跨构建使用。有关在 buildspec 文件中指定缓存的信息，请参阅buildspec 语法。有关缓存的更多信息，请参阅 缓存构建以提高性能。

日志

选择要创建的日志。您可以创建 Amazon CloudWatch 日志、Amazon S3 日志或两者兼而有之。

CloudWatch

如果你想要 Amazon CloudWatch Logs 日志：

CloudWatch logs

选择 CloudWatch logs (CloudWatch 日志)。

组名

输入您的 Amazon CloudWatch 日志组的名称。

流名称

输入您的 Amazon CloudWatch 日志流名称。

S3

如果要创建 Amazon S3 日志：

S3 日志

选择 S3 日志。

存储桶

选择您的日志的 S3 存储桶的名称。

路径前缀

输入日志的前缀。

禁用 S3 日志加密

如果您不希望加密您的 S3 日志，请选择此选项。

创建构建项目 (AWS CLI)

有关 AWS CLI 搭配使用的更多信息 CodeBuild，请参阅命令行参考。

要使用创建 CodeBuild 构建项目 AWS CLI，请创建 JSON-formatted 项目结构，填写结构，然后调用`create-project`命令来创建项目。

创建 JSON 文件

利用 `create-project` 命令和 `--generate-cli-skeleton` 选项创建骨架 JSON 文件：

```
aws codebuild create-project --generate-cli-skeleton > <json-file>
```

这将创建一个 JSON 文件，其路径和文件名由指定*<json-file>*。

填写 JSON 文件

按照下面所示修改 JSON 数据，并保存您的结果。

```
{
  "name": "<project-name>",
  "description": "<description>",
  "source": {
    "type": "CODECOMMIT" | "CODEPIPELINE" | "GITHUB" | "GITHUB_ENTERPRISE" | "GITLAB" |
 "GITLAB_SELF_MANAGED" | "BITBUCKET" | "S3" | "NO_SOURCE",
    "location": "<source-location>",
    "gitCloneDepth": "<git-clone-depth>",
    "buildspec": "<buildspec>",
    "InsecureSsl": "<insecure-ssl>",
    "reportBuildStatus": "<report-build-status>",
    "buildStatusConfig": {
      "context": "<context>",
      "targetUrl": "<target-url>"
    },
    "gitSubmodulesConfig": {
      "fetchSubmodules": "<fetch-submodules>"
    },
    "auth": {
      "type": "<auth-type>",
      "resource": "<auth-resource>"
    },
    "sourceIdentifier": "<source-identifier>"
  },
  "secondarySources": [
    {
        "type": "CODECOMMIT" | "CODEPIPELINE" | "GITHUB" | "GITHUB_ENTERPRISE" |
 "GITLAB" | "GITLAB_SELF_MANAGED" | "BITBUCKET" | "S3" | "NO_SOURCE",
        "location": "<source-location>",
        "gitCloneDepth": "<git-clone-depth>",
        "buildspec": "<buildspec>",
        "InsecureSsl": "<insecure-ssl>",
        "reportBuildStatus": "<report-build-status>",
        "auth": {
          "type": "<auth-type>",
          "resource": "<auth-resource>"
        },
        "sourceIdentifier": "<source-identifier>"
    }
  ],
  "secondarySourceVersions": [
    {
      "sourceIdentifier": "<secondary-source-identifier>",
```

```
      "sourceVersion": "<secondary-source-version>"
    }
  ],
  "sourceVersion": "<source-version>",
  "artifacts": {
    "type": "CODEPIPELINE" | "S3" | "NO_ARTIFACTS",
    "location": "<artifacts-location>",
    "path": "<artifacts-path>",
    "namespaceType": "<artifacts-namespacetype>",
    "name": "<artifacts-name>",
    "overrideArtifactName": "<override-artifact-name>",
    "packaging": "<artifacts-packaging>"
  },
  "secondaryArtifacts": [
    {
      "type": "CODEPIPELINE" | "S3" | "NO_ARTIFACTS",
      "location": "<secondary-artifact-location>",
      "path": "<secondary-artifact-path>",
      "namespaceType": "<secondary-artifact-namespaceType>",
      "name": "<secondary-artifact-name>",
      "packaging": "<secondary-artifact-packaging>",
      "artifactIdentifier": "<secondary-artifact-identifier>"
    }
  ],
  "cache": {
    "type": "<cache-type>",
    "location": "<cache-location>",
    "mode": [
      "<cache-mode>"
    ]
  },
  "environment": {
    "type": "LINUX_CONTAINER" | "LINUX_GPU_CONTAINER" | "ARM_CONTAINER" |
"WINDOWS_SERVER_2019_CONTAINER" | "WINDOWS_SERVER_2022_CONTAINER",
    "image": "<image>",
    "computeType": "BUILD_GENERAL1_SMALL" | "BUILD_GENERAL1_MEDIUM" |
"BUILD_GENERAL1_LARGE" | "BUILD_GENERAL1_2XLARGE",
    "certificate": "<certificate>",
    "environmentVariables": [
      {
        "name": "<environmentVariable-name>",
        "value": "<environmentVariable-value>",
        "type": "<environmentVariable-type>"
      }
```

```
    ],
    "registryCredential": [
      {
        "credential": "<credential-arn-or-name>",
        "credentialProvider": "<credential-provider>"
      }
    ],
    "imagePullCredentialsType": "CODEBUILD" | "SERVICE_ROLE",
    "privilegedMode": "<privileged-mode>"
  },
  "serviceRole": "<service-role>",
  "autoRetryLimit": <auto-retry-limit>,
  "timeoutInMinutes": <timeout>,
  "queuedTimeoutInMinutes": <queued-timeout>,
  "encryptionKey": "<encryption-key>",
  "tags": [
    {
      "key": "<tag-key>",
      "value": "<tag-value>"
    }
  ],
  "vpcConfig": {
    "securityGroupIds": [
        "<security-group-id>"
    ],
    "subnets": [
        "<subnet-id>"
    ],
    "vpcId": "<vpc-id>"
  },
  "badgeEnabled": "<badge-enabled>",
  "logsConfig": {
    "cloudWatchLogs": {
      "status": "<cloudwatch-logs-status>",
      "groupName": "<group-name>",
      "streamName": "<stream-name>"
    },
    "s3Logs": {
      "status": "<s3-logs-status>",
      "location": "<s3-logs-location>",
      "encryptionDisabled": "<s3-logs-encryption-disabled>"
    }
  },
  "fileSystemLocations": [
```

```
    {
      "type": "EFS",
      "location": "<EFS-DNS-name-1>:/<directory-path>",
      "mountPoint": "<mount-point>",
      "identifier": "<efs-identifier>",
      "mountOptions": "<efs-mount-options>"
    }
  ],
  "buildBatchConfig": {
    "serviceRole": "<batch-service-role>",
    "combineArtifacts": <combine-artifacts>,
    "restrictions": {
      "maximumBuildsAllowed": <max-builds>,
      "computeTypesAllowed": [
        "<compute-type>"
      ],
      "fleetsAllowed": [
        "<fleet-name>"
      ]
    },
    "timeoutInMins": <batch-timeout>,
    "batchReportMode": "REPORT_AGGREGATED_BATCH" | "REPORT_INDIVIDUAL_BUILDS"
  },
  "concurrentBuildLimit": <concurrent-build-limit>
}
```

替换以下内容：

name

必需。此构建项目的名称。此名称在您 AWS 账户中的所有构建项目中必须是唯一的。

描述

可选。此构建项目的描述。

源

必需。一个ProjectSource对象，其中包含有关此构建项目的源代码设置的信息。添加 source 对象后，可以使用 额外添加多达 12 个源。这些设置包括：

source/type

必需。包含要构建的源代码的存储库的类型。有效值包括：

- CODECOMMIT
- CODEPIPELINE
- GITHUB
- GITHUB_ENTERPRISE
- GITLAB
- GITLAB_SELF_MANAGED
- BITBUCKET
- S3
- NO_SOURCE

如果您使用 NO_SOURCE，则 buildspec 不能是一个文件，因为项目没有源。相反，你必须使用buildspec属性为你的构建规范指定一个 YAML-formatted 字符串。有关更多信息，请参阅 创建没有源的构建项目。

source/location

除非您设置为*<source-type>*，否则为必填项CODEPIPELINE。指定存储库类型的源代码的位置。

- 对于 CodeCommit，指向包含源代码和 buildspec 文件的存储库的 HTTPS 克隆 URL（例如，https://git-codecommit.*<region-id>*.amazonaws.com/v1/repos/*<repo-name>*）。
- 对于 Amazon S3，是构建输入存储桶的名称，后附包含源代码和 buildspec 的 ZIP 文件的路径和名称。例如：
  - 对于位于输入存储桶根目录的 ZIP 文件：*<bucket-name>*/*<object-name>*.zip。
  - 对于位于输入存储桶子文件夹中的 ZIP 文件：*<bucket-name>*/*<subfoler-path>*/*<object-name>*.zip。
- 对于 GitHub，指向包含源代码和 buildspec 文件的存储库的 HTTPS 克隆 URL。该 URL 必须包含 github.com。您必须将您的 AWS 账户关联到您的 GitHub 账户。为此，请使用 CodeBuild 控制台创建构建项目。
  - 选择授权应用程序。（连接到您的 GitHub 帐户后，您无需完成构建项目的创建。 您可以关闭 CodeBuild 控制台。）
- 对于 GitHub 企业服务器，指向包含源代码和 buildspec 文件的存储库的 HTTP 或 HTTPS 克隆 URL。您还必须将您的 AWS 帐户与您的 GitHub企业服务器帐户关联。为此，请使用 CodeBuild 控制台创建构建项目。

1. 在 GitHub 企业服务器中创建个人访问令牌。
2. 将此令牌复制到剪贴板，以便在创建 CodeBuild 项目时使用。有关更多信息，请参阅 GitHub 帮助网站上的为命令行创建个人访问令牌。
3. 使用控制台创建 CodeBuild 项目时，在源代码中，对于源提供商，选择GitHub企业。
4. 对于个人访问令牌，请粘贴已复制到剪贴板中的令牌。选择保存令牌。现在，您的 CodeBuild 帐户已连接到您的 GitHub 企业服务器帐户。

- 对于 GitLab 和 GitLab 自我管理，指向包含源代码和 buildspec 文件的存储库的 HTTPS 克隆 URL。请注意，如果您使用 GitLab，则网址必须包含 gitlab.com。如果您使用 GitLab 自我管理，则网址不必包含 gitlab.com。您必须将您的 AWS 账户与您的账户 GitLab或 GitLab 自行管理的账户关联起来。为此，请使用 CodeBuild 控制台创建构建项目。
  - 在开发人员工具导航窗格中，依次选择设置、连接，然后选择创建连接。在此页面上，创建 GitLab 或 GitLab 自行管理的连接，然后选择 Connect to。 GitLab
- 对于 Bitbucket，则为指向包含源代码和 buildspec 文件的存储库的 HTTPS 克隆 URL。该 URL 必须包含 bitbucket.org。您还必须将您的 AWS 账户与 Bitbucket 账户关联起来。为此，请使用 CodeBuild 控制台创建构建项目。
  1. 当您使用控制台与 Bitbucket 连接（或重新连接）时，在 Bitbucket 确认对账户的访问页面上，选择授予访问权限。（连接到 Bitbucket 账户后，无需完成构建项目的创建。 您可以关闭 CodeBuild 控制台。）
- 对于 AWS CodePipeline，请不要为指定`location`值`source`。 CodePipeline 忽略此值，因为在中创建管道时 CodePipeline，需要在管道的 “源” 阶段指定源代码位置。

来源/ git CloneDepth

可选。要下载的历史记录深度。最小值为 0。如果此值为 0、大于 25 或未提供，则会下载每个构建项目的完整历史记录。如果您的源类型是 Amazon S3，则不支持此值。

source/buildspec

可选。要使用的构建规范定义或文件。如果此值未提供或设置为空字符串，源代码必须在其根目录中包含 `buildspec.yml` 文件。如果设置了该值，则它可以是内联 buildspec 定义，也可以是指向相对于主要源根目录的替代 buildspec 文件的路径，或者是指向 S3 存储桶的路径。存储桶必须与构建项目位于同一个 AWS 区域。使用其 ARN 指定 buildspec 文件（例如，`arn:aws:s3:::`*`<my-codebuild-sample2>`*`/buildspec.yml`）。有关更多信息，请参阅 buildspec 文件名称和存储位置。

source/auth

包含有关访问 CodeBuild 待编译源代码的授权设置的信息。

source/auth/类型

必需。要使用的授权类型。有效值为：

- OAUTH
- CODECONNECTIONS
- SECRETS_MANAGER

source/auth/资源

可选。适用于指定授权类型的资源值。这可以是 Secrets Manager ARN 或 AR AWS CodeConnections N。

来源/ 报告 BuildStatus

指定是否向源提供商发送构建的开始和完成状态。如果您使用除 GitHub 企业服务器或 Bitbucket 之外的 GitHub源提供商进行此设置，invalidInputException则会抛出。

为了能够向源提供商报告构建状态，与源提供商关联的用户必须拥有对存储库的写入权限。如果用户没有写入权限，则无法更新构建状态。有关更多信息，请参阅 源提供商访问权限。

源代码/ 构建 StatusConfig

包含定义 CodeBuild 构建项目如何向源提供者报告构建状态的信息。此选项仅在源提供商为 GITHUB、GITHUB_ENTERPRISE 或 BITBUCKET 时使用。

source/buildStatusConfig/上下文

对于 Bitbucket 源，此参数用于处于 Bitbucket 提交状态的 name 参数。对于 GitHub 源，此参数用于处于 GitHub 提交状态的context参数。

例如，你可以使用 CodeBuild环境变量让 c context ontain 内部版本号和 webhook 触发器：

```
AWS CodeBuild sample-project Build #$CODEBUILD_BUILD_NUMBER -
 $CODEBUILD_WEBHOOK_TRIGGER
```

这会导致由 webhook 拉取请求事件触发的 build #24 的上下文显示如下：

```
AWS CodeBuild sample-project Build #24 - pr/8
```

source/buildStatusConfig/targetURL

对于 Bitbucket 源，此参数用于处于 Bitbucket 提交状态的 url 参数。对于 GitHub 源，此参数用于处于 GitHub 提交状态的target_url参数。

例如，您可以将 targetUrl 设置为 https://aws.amazon.com/codebuild/*<path to build>*，而提交状态将链接到此 URL。

您还可以在中包含 CodeBuild 环境变量，targetUrl以便向 URL 添加其他信息。例如，要将构建区域添加到此 URL，请将 targetUrl 设置为：

```
"targetUrl": "https://aws.amazon.com/codebuild/<path to build>?region=
$AWS_REGION"
```

如果构建区域为 us-east-2，则会扩展为：

```
https://aws.amazon.com/codebuild/<path to build>?region=us-east-2
```

来源/ git SubmodulesConfig

可选。有关 Git 子模块配置的信息。仅与 CodeCommit、 GitHub、 GitHub 企业服务器和 Bitbucket 一起使用。

source/gitSubmodulesConfig/fetch SubModules

如果您希望将 Git 子模块包含到存储库中，请将 fetchSubmodules 设置为 true。包含的 Git 子模块必须配置为 HTTPS。

来源/ InsecureSsl

可选。仅与 GitHub 企业服务器一起使用。将此值设置为，true以便在连接到 GitHub 企业服务器项目存储库时忽略 TLS 警告。默认值为 false。只应将 InsecureSsl 用于测试目的。它不应在生产环境中使用。

source/sourceIdentifier

用户定义的项目源标识符。对于主源来说，为可选项。对于辅助源，则为必选项。

secondarySources

可选。包含有关构建项目辅助源信息的ProjectSource对象数组。最多可以添加 12 个辅助源。这些 secondarySources 对象使用的属性与对象使用的属性相同。在辅助源对象中，sourceIdentifier 是必需项。

次要的 SourceVersions

可选。ProjectSourceVersion 数据元数组。如果在构建级别指定 secondarySourceVersions，则它们优先于此对象。

sourceVersion

可选。要为此项目构建的构建输入的版本。如果未指定，则使用最新版本。如果已指定，则它必须是下列项之一：

- 对于 CodeCommit，要使用的提交 ID、分支或 Git 标签。
- 对于 GitHub，与您要构建的源代码版本相对应的提交 ID、拉取请求 ID、分支名称或标签名称。如果指定了拉取请求 ID，则必须使用格式 `pr/pull-request-ID`（例如，`pr/25`）。如果指定了分支名称，则将使用分支的 HEAD 提交 ID。如果未指定，则使用默认分支的 HEAD 提交 ID。
- 对于 GitLab，提交 ID、拉取请求 ID、分支名称、标签名称或引用以及提交 ID。有关更多信息，请参阅 源版本示例，包含 AWS CodeBuild。
- 对于 Bitbucket，为提交 ID、分支名称或与您要构建的源代码版本相对应的标签名称。如果指定了分支名称，则将使用分支的 HEAD 提交 ID。如果未指定，则使用默认分支的 HEAD 提交 ID。
- 对于 Amazon S3，为表示要使用的构建输入 ZIP 文件的对象的版本 ID。

如果在构建级别指定 `sourceVersion`，则该版本将优先于此 `sourceVersion`（在项目级别）。有关更多信息，请参阅 源版本示例，包含 AWS CodeBuild。

构件

必需。一个ProjectArtifacts对象，其中包含有关此构建项目的输出构件设置的信息。添加 `artifacts` 对象后，可以使用 额外添加最多 12 个构件。这些设置包括：

artifacts/type

必需。构建输出构件的类型。有效值为：

- `CODEPIPELINE`
- `NO_ARTIFACTS`
- `S3`

artifacts/location

仅与 `S3` 构件类型一起使用。不用于其他构件类型。

您在先决条件中创建或标识的输出存储桶的名称。

artifacts/path

仅与 `S3` 构件类型一起使用。不用于其他构件类型。

放置 ZIP 文件或文件夹的输出存储桶的路径。如果未为指定值path，则 CodeBuild 使用namespaceType（如果已指定）和name来确定生成输出 ZIP 文件或文件夹的路径和名称。例如，如果您为 path 指定 MyPath，并为 name 指定 MyArtifact.zip，那么路径和名称将为 MyPath/MyArtifact.zip。

artifacts/namespaceType

仅与 S3 构件类型一起使用。不用于其他构件类型。

构建输出 ZIP 文件或文件夹的命名空间。有效值包括 BUILD_ID 和 NONE。使用 BUILD_ID 将构建 ID 插入到构建输出 ZIP 文件或文件夹的路径中。否则，请使用 NONE。如果未为指定值namespaceType，则 CodeBuild 使用path（如果已指定）和name来确定生成输出 ZIP 文件或文件夹的路径和名称。例如，如果您为 path 指定 MyPath，为 namespaceType 指定 BUILD_ID，并为 name 指定 MyArtifact.zip，那么路径和名称将为 MyPath/*build-ID*/MyArtifact.zip。

artifacts/name

仅与 S3 构件类型一起使用。不用于其他构件类型。

location 中构建输出 ZIP 文件或文件夹的名称。例如，如果您为 path 指定 MyPath，并为 name 指定 MyArtifact.zip，那么路径和名称将为 MyPath/MyArtifact.zip。

构件/ 覆盖 ArtifactName

仅与 S3 构件类型一起使用。不用于其他构件类型。

可选。如果设置为 true，在 buildspec 文件的 artifacts 块中指定的名称将覆盖 name。有关更多信息，请参阅 的构建规范参考 CodeBuild。

artifacts/packaging

仅与 S3 构件类型一起使用。不用于其他构件类型。

可选。指定如何打包构件。允许的值包括：

NONE

创建包含构建构件的文件夹。这是默认值。

ZIP

创建包含构建构件的 ZIP 文件。

secondaryArtifacts

可选。一组ProjectArtifacts对象，其中包含有关构建项目的次要构件设置的信息。最多可以添加 12 个辅助构件。`secondaryArtifacts` 使用的许多设置与 对象相同。

cache

必需。一个ProjectCache对象，其中包含有关此构建项目的缓存设置的信息。有关更多信息，请参阅缓存构建。

environment

必需。一个ProjectEnvironment对象，其中包含有关此项目的构建环境设置的信息。这些设置包括：

environment/type

必需。构建环境的类型。有关更多信息，请参阅 CodeBuild API 参考中的键入。

environment/image

必需。此构建环境使用的 Docker 映像标识符。通常，此标识符表示为*image-name*：*tag*。例如，在 CodeBuild 用于管理其 Docker 镜像的 Docker 存储库中，可能就是这样。`aws/codebuild/standard:5.0`在 Docker Hub 中，为 `maven:3.3.9-jdk-8`。在 Amazon ECR 中，为 *account-id*`.dkr.ecr.`*region-id*`.amazonaws.com/`*your-Amazon-ECR-repo-name*`:`*tag*。有关更多信息，请参阅 Docker 镜像由 CodeBuild。

environment/computeType

必需。指定此构建环境使用的计算资源。有关更多信息，请参阅《API 参考》中的 “计算类型”。CodeBuild

environment/certificate

可选。Amazon S3 存储桶的 ARN、路径前缀和包含证书的对象密钥。 PEM-encoded 对象密钥可以是.pem 文件，也可以是包含证书的.zip 文件。 PEM-encoded 例如，如果 Amazon S3 存储桶名称为 *<my-bucket>*，路径前缀为 *<cert>*，且对象键名称为 *<certificate.pem>*，则 `certificate` 可接受的格式为 *<my-bucket/cert/certificate.pem>* 或 `arn:aws:s3:::`*<my-bucket/cert/certificate.pem>*。

environment/environmentVariables

可选。包含要为此构建环境指定的环境变量的EnvironmentVariable对象数组。每个环境变量都表示为一个对象，其中包含 `name`、`value`，以及 `name` 和 `value` 的 `type`，还有 `type`。

控制台和 AWS CLI 用户可以看到所有环境变量。如果您不担心环境变量的可见性，请设置 name 和 value，并将 type 设置为 PLAINTEXT。

我们建议您将带有敏感值的环境变量（例如访问密钥 ID、私有 AWS 访问 AWS 密钥或密码）作为参数存储在 Amazon EC2 Systems Manager Parameter Store 或 AWS Secrets Manager。对于name，为存储的参数设置标识符 CodeBuild 以供参考。

如果您使用 Amazon EC2 Systems Manager Parameter Store，针对 value，将参数名称设置为存储在 Parameter Store 中。将 type 设置为 PARAMETER_STORE。以名为 /CodeBuild/dockerLoginPassword 的参数为例，将 name 设置为 LOGIN_PASSWORD。将 value 设置为 /CodeBuild/dockerLoginPassword。将 type 设置为 PARAMETER_STORE。

> **Important**
>
> 如果您使用 Amazon EC2 Systems Manager Parameter Store，我们建议您使用以 /CodeBuild/ 开头的参数名称（例如，/CodeBuild/dockerLoginPassword）来存储参数。您可以使用 CodeBuild 控制台在 Amazon EC2 Systems Manager 中创建参数。选择创建参数，然后按照对话框中的说明操作。（在该对话框中，对于 KMS 密钥，您可以指定账户中 AWS KMS 密钥的 ARN。 Amazon EC2 Systems Manager 使用此密钥在存储期间加密参数的值，并在检索期间对其进行解密。）如果您使用 CodeBuild 控制台创建参数，则控制台会将参数名称的存储方式/CodeBuild/作为参数名称的开头。有关更多信息，请参阅《Amazon EC2 Systems Manager 用户指南》中的 Systems Manager Parameter Store 和 Systems Manager Parameter Store 控制台演练。
>
> 如果您的构建项目引用了 Amazon EC2 Systems Manager Parameter Store 中存储的参数，则构建项目的服务角色必须允许 ssm:GetParameters 操作。如果您之前选择了“新建服务角色”，请将此操作 CodeBuild 包含在构建项目的默认服务角色中。但是，如果您选择了现有服务角色，必须单独将此操作添加到您的服务角色中。
>
> 如果您的构建项目引用了 Amazon EC2 Systems Manager Parameter Store 中存储的但参数名称不以 /CodeBuild/ 开头的参数，且您选择了新建服务角色，您必须更新该服务角色以允许访问不以 /CodeBuild/ 开头的参数名称。这是因为该服务角色仅允许访问以 /CodeBuild/ 开头的参数名称。
>
> 如果您选择新建服务角色，服务角色将拥有解密 Amazon EC2 Systems Manager Parameter Store 中 /CodeBuild/ 命名空间下的所有参数的权限。
>
> 您设置的环境变量将替换现有的环境变量。例如，如果 Docker 映像已经包含一个名为 MY_VAR 的环境变量（值为 my_value），并且您设置了一个名为 MY_VAR 的环境变量（值为 other_value），那么 my_value 将被替换为 other_value。同样，如果 Docker 映像已经包含一个名为 PATH 的环境变量（值为 /usr/local/sbin:/usr/local/bin），并且您设置了一个名为 PATH 的环境变量（值为 $PATH:/usr/share/

ant/bin），那么/usr/local/sbin:/usr/local/bin 将被替换为文本值 $PATH:/usr/share/ant/bin。
请勿使用以 CODEBUILD_ 打头的名称设置任何环境变量。此前缀是专为内部使用预留的。
如果具有相同名称的环境变量在多处都有定义，则应按照如下方式确定其值：

- 构建操作调用开始时的值优先级最高。
- 构建项目定义中的值优先级次之。
- buildspec 声明中的值优先级最低。

如果您使用 Secrets Manager，针对 value，请将参数的名称设置为存储在 Secrets Manager 中。将 type 设置为 SECRETS_MANAGER。以名为 /CodeBuild/dockerLoginPassword 的密钥为例，将 name 设置为 LOGIN_PASSWORD。将 value 设置为 /CodeBuild/dockerLoginPassword。将 type 设置为 SECRETS_MANAGER。

Important

如果您使用 Secrets Manager，我们建议您存储名称以 /CodeBuild/（例如 /CodeBuild/dockerLoginPassword）开头的密钥。有关更多信息，请参阅《AWS Secrets Manager 用户指南》中的什么是 AWS Secrets Manager？。
如果您的构建项目引用了 Secrets Manager 中存储的密钥，则构建项目的服务角色必须允许 secretsmanager:GetSecretValue 操作。如果您之前选择了“新建服务角色”，请将此操作 CodeBuild 包含在构建项目的默认服务角色中。但是，如果您选择了现有服务角色，必须单独将此操作添加到您的服务角色中。
如果您的构建项目引用了 Secrets Manager 中存储的但密钥名称不以 /CodeBuild/ 开头的密钥，且您选择了新建服务角色，您必须更新该服务角色以允许访问不以 /CodeBuild/ 开头的密钥名称。这是因为该服务角色仅允许访问以 /CodeBuild/ 开头的密钥名称。
如果您选择新建服务角色，该服务角色将拥有解密 Secrets Manager 中 /CodeBuild/ 命名空间下的所有密钥的权限。

environment/registryCredential

可选。一个RegistryCredential对象，它指定提供私有 Docker 注册表访问权限的凭据。

environment/registryCredential/凭证

指定使用 AWS Managed Services创建的凭证的 ARN 或名称。仅当凭证存在于您当前的区域中时，您才能使用凭证的名称。

environment/registryCredential/凭证提供者

唯一有效值为 SECRETS_MANAGER。

当设置此属性时：

- imagePullCredentials 必须设置为 SERVICE_ROLE。
- 映像不能为辅助映像或 Amazon ECR 映像。

environment/imagePullCredentialsType

可选。 CodeBuild 用于在构建版本中提取图像的凭据类型。有两个有效值：

CODEBUILD

CODEBUILD指定 CodeBuild 使用自己的凭证。您必须编辑您的 Amazon ECR 存储库策略才能信任 CodeBuild 服务委托人。

SERVICE_ROLE

指定 CodeBuild 使用您的构建项目的服务角色。

当您使用跨账户或私有注册表映像时，必须使用 SERVICE_ROLE 凭证。使用 CodeBuild 精选图片时，必须使用CODEBUILD凭据。

environment/privilegedMode

仅在使用此构建项目来构建 Docker 映像时，才设置为 true。否则，尝试与 Docker 守护程序交互的所有关联的构建都将失败。您还必须启动 Docker 守护程序，以便您的构建与其交互。执行此操作的一种方法是通过运行以下构建命令在您的 buildspec 文件的 install 阶段初始化 Docker 进程守护程序。如果您指定了由具有 Docker 支持的 CodeBuild 提供的构建环境映像，请不要运行这些命令。

> **Note**
>
> 默认情况下，为非 VPC 构建启用 Docker 进程守护程序。如果您想使用 Docker 容器进行 VPC 构建，请参阅 Docker 文档网站上的运行时权限和 Linux 功能并启用特权模式。此外，Windows 不支持特权模式。

```
- nohup /usr/local/bin/dockerd --host=unix:///var/run/docker.sock --
host=tcp://127.0.0.1:2375 --storage-driver=overlay2 &
```

```
- timeout 15 sh -c "until docker info; do echo .; sleep 1; done"
```

serviceRole

必需。服务角色的 ARN CodeBuild 用于代表用户与服务进行交互（例如，`arn:aws:iam::account-id:role/role-name`）。

auto RetryLimit

可选。构建失败后额外的自动重试次数。例如，如果自动重试限制设置为 2，则 CodeBuild 会调用 `RetryBuild` API 自动再重试构建 2 次。

超时 InMinutes

可选。分钟数，介于 5 到 2160（36 小时）之间，如果构建未完成，则 CodeBuild停止构建。如果未指定，则使用默认值 60。要确定是否以及何时由于超时而 CodeBuild 停止构建，请运行该`batch-get-builds`命令。要确定构建是否已停止，请在输出中查看 `buildStatus` 的值是否为 `FAILED`。要确定构建何时超时，请在输出中查看与 `TIMED_OUT` 的 `phaseStatus` 值关联的 `endTime` 值。

排队 TimeoutInMinutes

可选。分钟数，介于 5 到 480（8 小时）之间，如果构建仍在排队，则在该时间后 CodeBuild停止构建。如果未指定，则使用默认值 60。

encryptionKey

可选。 AWS KMS key 用于加密生成输出的 CodeBuild 别名或 ARN。如果您指定别名，请使用格式 `arn:aws:kms:region-ID:account-ID:key/key-ID`，或者，如果存在别名，请使用格式 `alias/key-alias`。如果未指定，则使用适用于 Amazon S3 的 AWS托管 KMS 密钥。

标签

可选。一组 Tag 对象，提供您要与此构建项目关联的标签。您最多可指定 50 个标签。任何支持 CodeBuild 构建项目标签的 AWS 服务都可以使用这些标签。每个标签都表示为带有 `key` 和 `value` 的对象。

vpcConfig

可选。一个VpcConfig对象，其中包含有关您的项目 VPC 配置的信息。有关更多信息，请参阅 AWS CodeBuild 与亚马逊 Virtual Private Cloud 配合使用。

这些属性包括：

vpcId

必需。 CodeBuild 使用的 VPC ID。运行此命令获取您的区域中的所有 VPC ID 的列表：

```
aws ec2 describe-vpcs --region <region-ID>
```

subnets

必需。子网 ID 数组，其中包括使用的资源 CodeBuild。运行此命令，以获取这些 ID：

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=<vpc-id>" --region <region-ID>
```

安全 GroupIds

必需。用于允许访问 VPC 中的资源的一 CodeBuild 组安全组 ID。运行此命令，以获取这些 ID：

```
aws ec2 describe-security-groups --filters "Name=vpc-id,Values=<vpc-id>" --<region-ID>
```

badgeEnabled

可选。指定是否在 CodeBuild 项目中包含构建徽章。设置为 `true` 可启用构建徽章；设置为 `false` 可将其禁用。有关更多信息，请参阅 使用以下方法生成徽章示例 CodeBuild。

logsConfig

一个LogsConfig对象，其中包含有关此版本日志所在位置的信息。

logsConfig/ 云 WatchLogs

包含有关将日志推送到 Logs 的 CloudWatch 信息的CloudWatchLogsConfig对象。

logsConfig/s3Logs

一个S3LogsConfig对象，其中包含有关将日志推送到 Amazon S3 的信息。

文件 SystemLocations

可选。包含有关您的 Amazon EFS 配置信息的ProjectFileSystemsLocation对象数组。

建造 BatchConfig

可选。该buildBatchConfig对象是一个包含项目的批量生成配置信息的ProjectBuildBatchConfig结构。

buildBatchConfig/s erviceRol e

批量构建项目的服务角色 ARN。

buil BatchConfig d/ 组合神器

布尔值，用于指定是否将批量构建的构建构件合并到单个构件位置。

buildBatchConfig/restrictions/最大 BuildsAllowed

允许的最大构建数。

buildBatchConfig/restrictions/计算 TypesAllowed

一组字符串，用于指定批量构建允许的计算类型。请参阅构建环境计算类型以了解这些值。

buildBatchConfig/restrictions/允许舰队

一组字符串，用于指定批量构建支持的实例集。有关更多信息，请参阅在预留容量实例集上运行构建。

buildBatchConfig/超时 InMinutes

必须完成批量构建的最长时间（以分钟为单位）。

构建BatchConfig/批处理 ReportMode

指定如何将构建状态报告发送到源提供商以进行批量构建。有效值包括：

`REPORT_AGGREGATED_BATCH`

（默认）将所有构建状态聚合到单个状态报告中。

`REPORT_INDIVIDUAL_BUILDS`

为每个单独的构建发送单独的状态报告。

并发 BuildLimit

此项目允许的并发构建的最大数量。

仅当当前构建数量小于或等于此限值时，才会启动新构建。如果当前构建计数达到此限值，则新构建将受到限制且不会运行。

创建项目

要创建项目，请再次运行 **create-project** 命令，传递您的 JSON 文件：

```
aws codebuild create-project --cli-input-json file://<json-file>
```

如果成功，项目对象的 JSON 表示形式将显示在控制台输出中。有关此数据的示例，请参阅CreateProject 响应语法。

您稍后可以更改构建项目的任何设置，但构建项目名称除外。有关更多信息，请参阅更改构建项目的设置 (AWS CLI)。

要开始运行构建，请参阅运行构建 (AWS CLI)。

如果您的源代码存储在存储 GitHub 库中，并且您希望在每次将代码更改推送 CodeBuild 到存储库时都重新生成源代码，请参阅开始自动运行构建（AWS CLI）。

创建构建项目 (AWS 软件开发工具包)

有关与 AWS 软件开发工具包 AWS CodeBuild 一起使用的信息，请参阅。AWS SDK 和工具参考

创建构建项目 (CloudFormation)

有关 AWS CodeBuild 与一起使用的信息 CloudFormation，请参阅《AWS CloudFormation 用户指南》 CodeBuild中的 CloudFormation 模板。

创建通知规则

您可以使用通知规则在发生重要更改（例如，构建成功和失败）时通知用户。通知规则指定用于发送通知的事件和 Amazon SNS 主题。有关更多信息，请参阅什么是通知？

您可以使用控制台或 AWS CLI 为创建通知规则 AWS CodeBuild。

创建通知规则（控制台）

1. 登录 AWS 管理控制台 并打开 CodeBuild 控制台，网址为https://console.aws.amazon.com/codebuild/。

2. 选择构建，再选择构建项目，然后选择要在其中添加通知的构建项目。
3. 在构建项目页面上，选择通知，然后选择创建通知规则。您也可以转到构建项目的设置页面，然后选择创建通知规则。
4. 在通知名称中，输入规则的名称。
5. 如果您只想在通知中 EventBridge 包含提供给 Amazon 的信息，请在 “详情类型” 中选择 “基本”。如果您想包括提供给 Amazon 的信息 EventBridge 以及可能由 CodeBuild 或通知管理器提供的信息，请选择 “全部”。

   有关更多信息，请参阅了解通知内容和安全性。
6. 在触发通知的事件中，选择要为其发送通知的事件。有关详细信息，请参阅构建项目的通知规则的事件。
7. 在目标中，执行下列操作之一：

   - 如果您已将资源配置为与通知一起使用，请在选择目标类型中，选择聊天应用程序中的 Amazon Q 开发者版（Slack）或 SNS 主题。在选择目标中，选择客户端的名称（对于在聊天应用程序的 Amazon Q 开发者版中配置的 Slack 客户端）或 Amazon SNS 主题的 Amazon 资源名称（ARN）（对于已使用通知所需策略配置的 Amazon SNS 主题）。
   - 如果您尚未将资源配置为与通知一起使用，请选择创建目标，然后选择 SNS 主题。在 codestar-notifications- 之后提供主题的名称，然后选择创建。

   > **Note**
   >
   > - 如果您在创建通知规则的过程中创建 Amazon SNS 主题，则为您应用允许通知功能将事件发布到主题的策略。使用为通知规则创建的主题有助于确保您仅订阅要接收有关此资源的通知的那些用户。
   > - 您不能在创建通知规则的过程中在聊天应用程序中创建 Amazon Q 开发者版。如果您在聊天应用程序（Slack）中选择 Amazon Q 开发者版，您会看到一个按钮，引导您在聊天应用程序的 Amazon Q 开发者版中配置客户端。选择该选项后，会在聊天应用程序控制台中打开 Amazon Q 开发者版。有关更多信息，请参阅在聊天应用程序中配置通知与 Amazon Q 开发者版之间的集成。
   > - 如果要使用现有 Amazon SNS 主题作为目标，则在该主题可能存在的任何其他策略之外，您还必须为 AWS CodeStar 通知添加所需的策略。有关更多信息，请参阅为通知配置 Amazon SNS 主题以及了解通知内容和安全性。
8. 要完成规则创建，请选择提交。

9. 您必须为用户订阅规则的 Amazon SNS 主题，然后他们才能接收通知。有关更多信息，请参阅为用户订阅作为目标的 Amazon SNS 主题。您还可以在聊天应用程序中设置通知与 Amazon Q 开发者版之间的集成，以将通知发送到 Amazon Chime 聊天室。有关更多信息，请参阅在聊天应用程序中配置通知与 Amazon Q 开发者版之间的集成。

创建通知规则（AWS CLI）

1. 在终端或命令提示符处，运行 create-notification rule 命令以生成 JSON 骨架：

```
aws codestarnotifications create-notification-rule --generate-cli-skeleton
 > rule.json
```

   您可以将此文件命名为所需的任意名称。在本示例中，文件命名为 *rule.json*。

2. 在纯文本编辑器中打开 JSON 文件，然后对其进行编辑，以包括该规则所需的资源、事件类型和目标。以下示例显示了一条**MyNotificationRule**为编译项目命名的通知规则，该项目*MyBuildProject*在 AWS 帐号中名为 ID *123456789012*。*codestar-notifications-MyNotificationTopic*当构建成功时，系统会向名为 Amazon SNS 主题的完整详细信息类型发送通知：

```
{
    "Name": "MyNotificationRule",
    "EventTypeIds": [
        "codebuild-project-build-state-succeeded"
    ],
    "Resource": "arn:aws:codebuild:us-east-2:123456789012:MyBuildProject",
    "Targets": [
        {
            "TargetType": "SNS",
            "TargetAddress": "arn:aws:sns:us-east-2:123456789012:codestar-
notifications-MyNotificationTopic"
        }
    ],
    "Status": "ENABLED",
    "DetailType": "FULL"
}
```

   保存该文件。

3. 通过使用您刚编辑的文件，在终端或命令行上，再次运行 create-notification-rule 命令以创建通知规则：

```
aws codestarnotifications create-notification-rule --cli-input-json
 file://rule.json
```

4. 如果成功，该命令将返回通知规则的 ARN，类似于以下内容：

```
{
    "Arn": "arn:aws:codestar-notifications:us-east-1:123456789012:notificationrule/
dc82df7a-EXAMPLE"
}
```

在中更改构建项目设置 AWS CodeBuild

您可以使用 AWS CodeBuild 控制台或 AWS 软件开发工具包来更改构建项目的设置。 AWS CLI

如果您将测试报告添加到构建项目，请确保您的 IAM 角色具有测试报告权限中介绍的权限。

主题

- 更改构建项目的设置（控制台）
- 更改构建项目的设置 (AWS CLI)
- 更改构建项目的设置 (AWS 软件开发工具包)

更改构建项目的设置（控制台）

要更改构建项目的设置，请执行以下过程：

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择构建项目。
3. 请执行以下操作之一：
   - 选择要更改的构建项目的链接，然后选择构建详细信息。
   - 选择要更改的构建项目旁边的按钮，选择查看详细信息，然后选择构建详细信息。

您可以修改以下部分：

各个部分

- 项目配置
- 源
- 环境
- Buildspec
- 批量配置
- 构件
- 日志

项目配置

在项目配置部分，选择编辑。完成更改后，请选择更新配置，以保存新的配置。

您可以修改以下属性。

描述

输入构建项目的可选描述，以帮助其他用户了解此项目的用途。

构建徽章

选择启用构建徽章，以使您的项目的构建状态可见且可嵌入。有关更多信息，请参阅 构建徽章示例。

> **Note**
>
> 如果您的源提供商是 Amazon S3，则构建徽章不适用。

启用并发构建限制

如果要限制此项目的并发构建数量，请执行以下步骤：

1. 选择限制此项目可以启动的并发构建数量。
2. 在并发构建限制中，输入此项目允许的并发构建的最大数量。此限制不得大于为该账户设置的并发构建限制。如果您尝试输入大于账户限制的数字，则会显示错误消息。

仅当当前构建数量小于或等于此限值时，才会启动新构建。如果当前构建计数达到此限值，则新构建将受到限制且不会运行。

启用公共构建访问权限

要向公众（包括无法访问 AWS 账户的用户）公开项目的生成结果，请选择“启用公共生成访问权限”，然后确认要公开生成结果。以下属性用于公共构建项目：

公共构建服务角色

如果您想为您 CodeBuild 创建新的服务角色，请选择新服务角色；如果要使用现有的服务角色，请选择现有服务角色。

公共生成服务角色 CodeBuild 允许读取 CloudWatch 日志并下载项目构建的 Amazon S3 工件。您必须执行此操作，才能向公众提供项目的构建日志和构件。

服务角色

输入新的服务角色名称或现有服务角色名称。

要将项目的构建结果设为私有，请清除启用公共构建访问权限。

有关更多信息，请参阅 获取公共构建项目 URL。

> **Warning**
>
> 在公开项目的构建结果时，应记住以下几点：
>
> - 项目的所有构建结果、日志和构件，包括项目为私有状态时运行的构建，都可向公众开放。
> - 所有构建日志和构件都向公众开放。环境变量、源代码和其他敏感信息可能已输出到构建日志和构件中。您必须谨慎筛选将哪些信息输出到构建日志。以下是一些最佳实操：
>   - 请勿在环境变量中存储敏感值，尤其是 AWS 访问密钥 ID 和私有访问密钥。我们建议您使用 Amazon EC2 Systems Manager 参数存储或 AWS Secrets Manager 存储敏感值。
>   - 请按照使用 Webhook 的最佳实操对哪些实体可以触发构建进行限制且不要将 buildspec 存储在项目本身中，以尽可能确保您的 webhook 安全无虞。
> - 恶意用户可以使用公共构建分发恶意构件。我们建议项目管理员查看所有拉取请求，验证拉取请求是否为合法更改。我们还建议您使用校验和验证所有构件，确保下载的构件正确无误。

其他信息

在标签中，输入您希望支持 AWS 服务使用的任何标签的名称和值。使用添加行添加标签。最多可以添加 50 个标签。

源

在源部分中，请选择编辑。完成更改后，请选择更新配置，以保存新的配置。

您可以修改以下属性：

源提供商

选择源代码提供商类型。使用以下列表为您的源提供商选择适当的选项：

> **Note**
>
> CodeBuild 不支持 Bitbucket 服务器。

Amazon S3

存储桶

选择包含源代码的输入存储桶的名称。

S3 对象密钥或 S3 文件夹

输入 ZIP 文件的名称或包含源代码的文件夹的路径。输入正斜杠 (/) 以下载 S3 存储桶中的所有内容。

源版本

输入表示输入文件版本的对象的版本 ID。有关更多信息，请参阅 源版本示例，包含 AWS CodeBuild。

CodeCommit

存储库

选择要使用的存储库。

参考类型

选择分支、Git 标签或提交 ID，以指定源代码的版本。有关更多信息，请参阅 源版本示例，包含 AWS CodeBuild。

> **Note**
>
> 我们建议您选择看起来不像提交 ID 的 Git 分支名称，例如 `811dd1ba1aba14473856cee38308caed7190c0d` 或 `5392f7`。这可以帮助您避免 Git 签出与实际提交发生冲突。

Git 克隆深度

选择该选项，以创建一个浅克隆，其历史记录会截断至指定数量的提交。如果您需要完整克隆，请选择完整。

Git 子模块

如果您希望在存储库中包含 Git 子模块，请选择使用 Git 子模块。

Bitbucket

凭证

选择默认来源凭证或自定义来源凭证，然后按照说明管理默认来源凭证或自定义源凭证。

连接类型

选择要AWS CodeConnections连接的 OAuth、应用程序密码或个人访问令牌。 CodeBuild

Connection

选择 Bitbucket 连接或 Secrets Manager 密钥，通过指定的连接类型进行连接。

存储库

选择我的 Bitbucket 账户中的存储库或公共存储库，然后输入存储库 URL。

源版本

输入分支、提交 ID、标签，或引用以及提交 ID。有关更多信息，请参阅 源版本示例，包含 AWS CodeBuild。

> **Note**
>
> 我们建议您选择看起来不像提交 ID 的 Git 分支名称，例如 `811dd1ba1aba14473856cee38308caed7190c0d` 或 `5392f7`。这可以帮助您避免 Git 签出与实际提交发生冲突。

Git 克隆深度

选择Git 克隆深度以创建一个浅克隆，其历史记录会截断至指定数量的提交。如果您需要完整克隆，请选择完整。

Git 子模块

如果您希望在存储库中包含 Git 子模块，请选择使用 Git 子模块。

构建状态

如果您希望向源提供商报告构建的开始和完成状态，请选择在您的构建开始和完成时向源提供商报告构建状态。

为了能够向源提供商报告构建状态，与源提供商关联的用户必须拥有对存储库的写入权限。如果用户没有写入权限，则无法更新构建状态。有关更多信息，请参阅源提供商访问权限。

在状态上下文中，输入要在 Bitbucket 提交状态中用于 `name` 参数的值。有关更多信息，请参阅 Bitbucket API 文档中的构建。

在目标 URL 中，输入要在 Bitbucket 提交状态中用于 `url` 参数的值。有关更多信息，请参阅 Bitbucket API 文档中的构建。

由 Webhook 触发的构建的状态将始终报告给源提供商。要将从控制台或 API 调用启动的构建状态报告给源提供商，您必须选择此设置。

如果项目的构建通过 webhook 触发，则必须将新的提交推送到存储库，此设置才能生效。

如果要在每次将代码更改推送到此存储库时生成源代码，请在 “主源 webhook 事件” 中，选择 “每次将代码更改推送到此存储库时都重建”。 CodeBuild 有关 webhook 和筛选条件组的更多信息，请参阅 Bitbucket Webhook 事件。

GitHub

凭证

选择默认来源凭证或自定义来源凭证，然后按照说明管理默认来源凭证或自定义源凭证。

连接类型

选择要连接的GitHub 应用程序、OAuth 或个人访问令牌。 CodeBuild

Connection

选择要通过您指定的 GitHub 连接类型进行连接或 Secrets Manager 密钥进行连接。

存储库

在我的 GitHub 账户中选择 “存储库”、“公共存储库” 或 “GitHub 限定范围 webhook”，然后输入存储库 URL。

源版本

输入分支、提交 ID、标签，或引用以及提交 ID。有关更多信息，请参阅 源版本示例，包含 AWS CodeBuild。

> **Note**
>
> 我们建议您选择看起来不像提交 ID 的 Git 分支名称，例如 `811dd1ba1aba14473856cee38308caed7190c0d` 或 `5392f7`。这可以帮助您避免 Git 签出与实际提交发生冲突。

Git 克隆深度

选择Git 克隆深度以创建一个浅克隆，其历史记录会截断至指定数量的提交。如果您需要完整克隆，请选择完整。

Git 子模块

如果您希望在存储库中包含 Git 子模块，请选择使用 Git 子模块。

构建状态

如果您希望向源提供商报告构建的开始和完成状态，请选择在您的构建开始和完成时向源提供商报告构建状态。

为了能够向源提供商报告构建状态，与源提供商关联的用户必须拥有对存储库的写入权限。如果用户没有写入权限，则无法更新构建状态。有关更多信息，请参阅 源提供商访问权限。

在状态上下文中，输入要用于 GitHub 提交状态的`context`参数的值。有关更多信息，请参阅 GitHub 开发者指南中的创建提交状态。

在“目标 URL”中，输入要用于 GitHub 提交状态的`target_url`参数的值。有关更多信息，请参阅 GitHub 开发者指南中的创建提交状态。

由 Webhook 触发的构建的状态将始终报告给源提供商。要将从控制台或 API 调用启动的构建状态报告给源提供商，您必须选择此设置。

如果项目的构建通过 webhook 触发，则必须将新的提交推送到存储库，此设置才能生效。

如果要在每次将代码更改推送到此存储库时生成源代码，请在“主源 webhook 事件”中，选择“每次将代码更改推送到此存储库时都重建”。 CodeBuild 有关 webhook 和筛选条件组的更多信息，请参阅 GitHub webhook 事件。

GitHub Enterprise Server

凭证

选择默认来源凭证或自定义来源凭证，然后按照说明管理默认来源凭证或自定义源凭证。

连接类型

选择AWS CodeConnections要连接的个人访问令牌 CodeBuild。

Connection

选择 GitHub 企业连接或 Secrets Manager 密钥通过您指定的连接类型进行连接。

存储库

在我的 GitHub 企业帐户中选择存储库或GitHub 企业级范围的 webhook，然后输入存储库 URL。

源版本

输入拉取请求、分支、提交 ID、标签，或引用以及提交 ID。有关更多信息，请参阅 源版本示例，包含 AWS CodeBuild。

> **Note**
>
> 我们建议您选择看起来不像提交 ID 的 Git 分支名称，例如 `811dd1ba1aba14473856cee38308caed7190c0d` 或 `5392f7`。这可以帮助您避免 Git 签出与实际提交发生冲突。

Git 克隆深度

选择Git 克隆深度以创建一个浅克隆，其历史记录会截断至指定数量的提交。如果您需要完整克隆，请选择完整。

Git 子模块

如果您希望在存储库中包含 Git 子模块，请选择使用 Git 子模块。

构建状态

如果您希望向源提供商报告构建的开始和完成状态，请选择在您的构建开始和完成时向源提供商报告构建状态。

为了能够向源提供商报告构建状态，与源提供商关联的用户必须拥有对存储库的写入权限。如果用户没有写入权限，则无法更新构建状态。有关更多信息，请参阅 源提供商访问权限。

在状态上下文中，输入要用于 GitHub 提交状态的`context`参数的值。有关更多信息，请参阅 GitHub 开发者指南中的创建提交状态。

在 “目标 URL” 中，输入要用于 GitHub 提交状态的`target_url`参数的值。有关更多信息，请参阅 GitHub 开发者指南中的创建提交状态。

由 Webhook 触发的构建的状态将始终报告给源提供商。要将从控制台或 API 调用启动的构建状态报告给源提供商，您必须选择此设置。

如果项目的构建通过 webhook 触发，则必须将新的提交推送到存储库，此设置才能生效。

不安全的 SSL

选择 “启用不安全 SSL”，以便在连接到 GitHub 企业项目存储库时忽略 SSL 警告。

如果要在每次将代码更改推送到此存储库时生成源代码，请在 “主源 webhook 事件” 中，选择 “每次将代码更改推送到此存储库时都重建”。 CodeBuild 有关 webhook 和筛选条件组的更多信息，请参阅 GitHub webhook 事件。

GitLab

凭证

选择默认来源凭证或自定义来源凭证，然后按照说明管理默认来源凭证或自定义源凭证。

连接类型

AWS CodeConnections用于连接 GitLab 到 CodeBuild。

Connection

选择要 GitLab 连接的连接 AWS CodeConnections。

存储库

选择要使用的存储库。

源版本

输入拉取请求 ID、分支、提交 ID、标签，或引用以及提交 ID。有关更多信息，请参阅 源版本示例，包含 AWS CodeBuild。

> **Note**
>
> 我们建议您选择看起来不像提交 ID 的 Git 分支名称，例如 `811dd1ba1aba14473856cee38308caed7190c0d` 或 `5392f7`。这可以帮助您避免 Git 签出与实际提交发生冲突。

Git 克隆深度

选择Git 克隆深度以创建一个浅克隆，其历史记录会截断至指定数量的提交。如果您需要完整克隆，请选择完整。

构建状态

如果您希望向源提供商报告构建的开始和完成状态，请选择在您的构建开始和完成时向源提供商报告构建状态。

为了能够向源提供商报告构建状态，与源提供商关联的用户必须拥有对存储库的写入权限。如果用户没有写入权限，则无法更新构建状态。有关更多信息，请参阅 源提供商访问权限。

GitLab Self Managed

凭证

选择默认来源凭证或自定义来源凭证，然后按照说明管理默认来源凭证或自定义源凭证。

连接类型

AWS CodeConnections用于将 “ GitLab 自助管理” 连接到 CodeBuild。

Connection

选择要连接的 GitLab 自管理连接 AWS CodeConnections。

存储库

选择要使用的存储库。

源版本

输入拉取请求 ID、分支、提交 ID、标签，或引用以及提交 ID。有关更多信息，请参阅 源版本示例，包含 AWS CodeBuild。

> **Note**
>
> 我们建议您选择看起来不像提交 ID 的 Git 分支名称，例如 `811dd1ba1aba14473856cee38308caed7190c0d` 或 `5392f7`。这可以帮助您避免 Git 签出与实际提交发生冲突。

Git 克隆深度

选择Git 克隆深度以创建一个浅克隆，其历史记录会截断至指定数量的提交。如果您需要完整克隆，请选择完整。

构建状态

如果您希望向源提供商报告构建的开始和完成状态，请选择在您的构建开始和完成时向源提供商报告构建状态。

为了能够向源提供商报告构建状态，与源提供商关联的用户必须拥有对存储库的写入权限。如果用户没有写入权限，则无法更新构建状态。有关更多信息，请参阅 源提供商访问权限。

环境

在环境部分中，选择编辑。完成更改后，请选择更新配置，以保存新的配置。

您可以修改以下属性：

预置模型

要更改预置模型，请选择更改预置模型并执行下列操作之一：

- 要使用由管理的按需队列 AWS CodeBuild，请选择On-demand。使用按需队列，为您的构建 CodeBuild 提供计算能力。构建完成后，机器就会被销毁。 On-demand 舰队是完全托管的，并包括自动扩展功能，以应对需求激增。
- 要使用由管理的预留容量队列 AWS CodeBuild，请选择预留容量，然后选择队列名称。使用预留容量实例集，您可以为构建环境配置一组专用实例。这些计算机保持闲置状态，可以立即处理生成或测试，并缩短构建持续时间。使用预留容量队列，您的计算机将始终处于运行状态，并且只要配置完毕，它们就会继续产生成本。

有关信息，请参阅在预留容量实例集上运行构建。

环境映像

要更改构建映像，请选择覆盖映像，然后执行以下操作之一：

- 要使用由管理的 Docker 映像 AWS CodeBuild，请选择托管映像，然后从 “操作系统”、“运行时”、“映像” 和 “映像版本” 中进行选择。从环境类型中进行选择（如果可用）。
- 要使用其他 Docker 映像，请选择自定义映像。对于环境类型，请选择 ARM、Linux、Linux GPU 或 Windows。如果您针对外部注册表 URL 选择其他注册表，请使用 *`docker repository/docker image name`* 格式在 Docker Hub 中输入 Docker 映像的名称和标签。如果您选择 Amazon ECR，请使用亚马逊 ECR 存储库和 A mazon ECR 镜像在您的账户中选择 Docker 镜像。 AWS
- 要使用私有 Docker 映像，请选择自定义映像。对于环境类型，请选择 ARM、Linux、Linux GPU 或 Windows。对于映像注册表，选择其他注册表，然后输入您的私有 Docker 映像的凭证的 ARN。凭证必须由 Secrets Manager 创建。有关更多信息，请参阅《AWS Secrets Manager 用户指南》中的什么是 AWS Secrets Manager？。

> **Note**
>
> CodeBuild 会替换自定义 Docker 镜像的。`ENTRYPOINT`

服务角色

请执行以下操作之一：

- 如果您没有 CodeBuild 服务角色，请选择 “新建服务角色”。在角色名称中，为新角色输入名称。
- 如果您有 CodeBuild 服务角色，请选择现有服务角色。在角色 ARN 中，选择服务角色。

> **Note**
>
> 使用控制台创建构建项目时，可以同时创建 CodeBuild 服务角色。默认情况下，这个角色仅能与该构建项目配合使用。如果您使用控制台将此服务角色与另一个构建项目关联，则此角色将更新以便与关联的构建项目结合使用。一个服务角色最多可与 10 个构建项目结合使用。

其他配置

超时

指定一个介于 5 分钟到 36 小时之间的值，如果构建未完成，则在该值之后 CodeBuild 停止构建。如果小时和分钟都留空，则将使用 60 分钟的默认值。

特权

仅当您打算使用此构建项目来构建 Docker 映像时，才应选择如果要构建 Docker 映像或希望您的构建获得提升的特权，请启用此标志。否则，尝试与 Docker 守护程序交互的所有关联的构建都将失败。您还必须启动 Docker 守护程序，以便您的构建与其交互。一种方法是在构建规范`install`阶段通过运行以下构建命令来初始化 Docker 守护程序。如果您选择了由 CodeBuild Docker 支持的构建环境镜像，请不要运行这些命令。

> **Note**
>
> 默认情况下，为非 VPC 构建启用 Docker 进程守护程序。如果您想使用 Docker 容器进行 VPC 构建，请参阅 Docker 文档网站上的运行时权限和 Linux 功能并启用特权模式。此外，Windows 不支持特权模式。

```
- nohup /usr/local/bin/dockerd --host=unix:///var/run/docker.sock --
host=tcp://127.0.0.1:2375 --storage-driver=overlay2 &
- timeout 15 sh -c "until docker info; do echo .; sleep 1; done"
```

VPC

如果您 CodeBuild 想使用您的 VPC，请执行以下操作：

- 对于 VPC，请选择 CodeBuild 使用的 VPC ID。
- 对于 VPC 子网，请选择包含使用的 CodeBuild 资源的子网。
- 对于 VPC 安全组，请选择 CodeBuild 用于允许访问 VPC 中资源的安全组。

有关更多信息，请参阅 AWS CodeBuild 与亚马逊 Virtual Private Cloud 配合使用。

计算

请选择可用选项之一。

注册表凭证

使用非私有注册表映像配置项目时，请指定注册表凭证。

> **Note**
> 仅当映像被私有注册表中的映像覆盖时，才会使用此凭证。

环境变量

请输入每个环境变量的名称和值，然后选择类型，以供构建使用。

> **Note**
> CodeBuild 自动为您的 AWS 地区设置环境变量。如果您尚未将以下环境变量添加到 buildspec.yml 中，则必须设置这些变量：
> - AWS_ACCOUNT_ID
> - IMAGE_REPO_NAME
> - IMAGE_TAG

控制台和 AWS CLI 用户可以看到环境变量。如果您不担心环境变量的可见性，请设置名称和值字段，然后将类型设置为明文。

我们建议您将具有敏感值的环境变量（例如访问密钥 ID、私有 AWS 访问 AWS 密钥或密码）作为参数存储在 Amazon EC2 Systems Manager Parameter Store 或 AWS Secrets Manager。

如果您使用的是 Amazon EC2 Systems Manager Parameter Store，则对于类型，请选择参数。在名称中，输入 CodeBuild 要引用的标识符。对于值，请按照 Amazon EC2 Systems Manager Parameter Store 中存储的参数名称输入参数名称。使用名为 `/CodeBuild/dockerLoginPassword` 的参数作为示例，对于类型，选择参数。对于名称，请输入 `LOGIN_PASSWORD`。对于值，请输入 `/CodeBuild/dockerLoginPassword`。

> **Important**
>
> 如果您使用 Amazon EC2 Systems Manager Parameter Store，我们建议您使用以 `/CodeBuild/` 开头的参数名称（例如，`/CodeBuild/dockerLoginPassword`）来存储参数。您可以使用 CodeBuild 控制台在 Amazon EC2 Systems Manager 中创建参数。选择创建参数，然后按照对话框中的说明操作。（在该对话框中，对于 KMS 密钥，您可以指定账户中 AWS KMS 密钥的 ARN。 Amazon EC2 Systems Manager 使用此密钥在存储期间加密参数的值，并在检索期间对其进行解密。）如果您使用 CodeBuild 控制台创建参数，则控制台会将参数名称的存储方式`/CodeBuild/`作为参数名称的开头。有关更多信息，请参阅《Amazon EC2 Systems Manager 用户指南》中的 Systems Manager Parameter Store 和 Systems Manager Parameter Store 控制台演练。
>
> 如果您的构建项目引用了 Amazon EC2 Systems Manager Parameter Store 中存储的参数，则构建项目的服务角色必须允许 `ssm:GetParameters` 操作。如果您之前选择了“新建服务角色”，请将此操作 CodeBuild 包含在构建项目的默认服务角色中。但是，如果您选择了现有服务角色，必须单独将此操作添加到您的服务角色中。
>
> 如果您的构建项目引用了 Amazon EC2 Systems Manager Parameter Store 中存储的但参数名称不以 `/CodeBuild/` 开头的参数，且您选择了新建服务角色，您必须更新该服务角色以允许访问不以 `/CodeBuild/` 开头的参数名称。这是因为该服务角色仅允许访问以 `/CodeBuild/` 开头的参数名称。
>
> 如果您选择新建服务角色，服务角色将拥有解密 Amazon EC2 Systems Manager Parameter Store 中 `/CodeBuild/` 命名空间下的所有参数的权限。
>
> 您设置的环境变量将替换现有的环境变量。例如，如果 Docker 映像已经包含一个名为 `MY_VAR` 的环境变量（值为 `my_value`），并且您设置了一个名为 `MY_VAR` 的环境变量（值为 `other_value`），那么 `my_value` 将被替换为 `other_value`。同样，如果 Docker 映像已经包含一个名为 `PATH` 的环境变量（值为 `/usr/local/sbin:/usr/local/bin`），并且您设置了一个名为 `PATH` 的环境变量（值为 `$PATH:/usr/share/ant/bin`），那么`/usr/local/sbin:/usr/local/bin` 将被替换为文本值 `$PATH:/usr/share/ant/bin`。
>
> 请勿使用以 `CODEBUILD_` 打头的名称设置任何环境变量。此前缀是专为内部使用预留的。

如果具有相同名称的环境变量在多处都有定义，则应按照如下方式确定其值：

- 构建操作调用开始时的值优先级最高。
- 构建项目定义中的值优先级次之。
- buildspec 声明中的值优先级最低。

如果您使用 Secrets Manager，对于类型，请选择 Secrets Manager。在名称中，输入 CodeBuild 要引用的标识符。对于值，请使用模式 *secret-id*:*json-key*:*version-stage*:*version-id* 输入 reference-key。有关信息，请参阅 Secrets Manager reference-key in the buildspec file。

Important

如果您使用 Secrets Manager，我们建议您存储名称以 /CodeBuild/（例如 /CodeBuild/dockerLoginPassword）开头的密钥。有关更多信息，请参阅《AWS Secrets Manager 用户指南》中的什么是 AWS Secrets Manager？。
如果您的构建项目引用了 Secrets Manager 中存储的密钥，则构建项目的服务角色必须允许 secretsmanager:GetSecretValue 操作。如果您之前选择了“新建服务角色”，请将此操作 CodeBuild 包含在构建项目的默认服务角色中。但是，如果您选择了现有服务角色，必须单独将此操作添加到您的服务角色中。
如果您的构建项目引用了 Secrets Manager 中存储的但密钥名称不以 /CodeBuild/ 开头的密钥，且您选择了新建服务角色，您必须更新该服务角色以允许访问不以 /CodeBuild/ 开头的密钥名称。这是因为该服务角色仅允许访问以 /CodeBuild/ 开头的密钥名称。
如果您选择新建服务角色，该服务角色将拥有解密 Secrets Manager 中 /CodeBuild/ 命名空间下的所有密钥的权限。

Buildspec

在 Buildspec 部分，选择编辑。完成更改后，请选择更新配置，以保存新的配置。

您可以修改以下属性：

构建规范

请执行以下操作之一：

- 如果您的源代码包含 buildspec 文件，请选择使用 buildspec 文件。默认情况下， CodeBuild 在源代码根目录中查找名为 `buildspec.yml` 的文件。如果您的 buildspec 文件使用其他名称或位置，请在 Buildspec 名称中输入其从源根目录开始的路径（例如，`buildspec-two.yml` 或 `configuration/buildspec.yml`。如果 buildspec 文件位于 S3 存储桶中，则该存储桶必须位于您的构建项目所在的同一 AWS 区域中。使用 ARN（例如 `arn:aws:s3:::<my-codebuild-sample2>/buildspec.yml`）指定该 buildspec 文件。
- 如果您的源代码不包括 buildspec 文件，或者如果您要运行的构建命令不是在源代码根目录的 `buildspec.yml` 文件中为 `build` 阶段指定的构建命令，则选择插入构建命令。对于构建命令，请输入您要在 `build` 阶段运行的命令。对于多个命令，使用 `&&` 分开各个命令（例如 `mvn test && mvn package`）。要在其他阶段运行命令，或者，如果 `build` 阶段对应的命令列表特别长，请将 `buildspec.yml` 文件添加到源代码根目录，将命令添加到该文件中，然后选择在源代码根目录中使用 buildspec.yml。

有关更多信息，请参阅Buildspec 参考。

批量配置

在批量配置部分，选择编辑。完成更改后，请选择更新配置，以保存新的配置。有关更多信息，请参阅批量运行构建。

您可以修改以下属性：

批量服务角色

为批量构建提供服务角色。

选择下列选项之一：

- 如果您没有批量服务角色，请选择新建服务角色。在服务角色中，为新角色输入名称。
- 如果您拥有批量服务角色，请选择现有服务角色。在服务角色中，选择对应的服务角色。

批量构建为批量配置引入了全新的安全角色。这个新角色是必需的，因为 CodeBuild 必须能够代表你调用`StartBuild`、`StopBuild`、和`RetryBuild`操作才能将生成作为批处理的一部分运行。客户应该使用新角色，而不是他们在构建中使用的角色，原因有两个：

- 向构建角色授予 `StartBuild`、`StopBuild` 和 `RetryBuild` 权限后，将允许单个构建通过 buildspec 启动多个构建。
- CodeBuild 批处理生成提供了限制，限制了可用于批次构建的生成数量和计算类型。如果构建角色拥有这些权限，则构建本身就有可能绕过这些限制。

批量支持的计算类型

选择批处理允许的计算类型。选择所有适用的选项。

批量支持的实例集

选择批量支持的实例集。选择所有适用的选项。

批处理允许的最大构建数量

输入批处理允许的最大构建数量。如果批处理超过此限制，则会失败。

批处理超时

输入完成批量构建能够使用的最长时间。

合并构件

选择将批处理中的所有构件合并到一个位置，将批处理中的所有构件合并到一个位置。

批量报告模式

为批量构建选择所需的构建状态报告模式。

> **Note**
>
> 仅当项目源为 Bitbucket 或 E GitHub nterprise 时，此字段才可用，并且在 “来源” 下选择了生成开始和完成时向源提供商报告构建状态。 GitHub

聚合构建

选择该选项，可将批处理中所有构建的状态合并到一个状态报告中。

单个构建

选择该选项，可分别报告批处理中所有构建的构建状态。

构件

在构件部分中，选择编辑。完成更改后，请选择更新配置，以保存新的配置。

您可以修改以下属性：

类型

请执行以下操作之一：

- 如果您不想创建任何构建输出构件，请选择无构件。如果您只运行构建测试，或者您要将 Docker 映像推送到 Amazon ECR 存储库，建议执行此操作。
- 要将构建输出存储在 S3 存储桶中，请选择 Amazon S3，然后执行以下操作：
  - 如果要将项目名称用于构建输出 ZIP 文件或文件夹，请将名称留空。否则，请输入名称。（如果您要输出 ZIP 文件，并且要让 ZIP 文件包含文件扩展名，请务必在 ZIP 文件名之后添加扩展名。）
  - 如果希望构建规范文件中指定的名称覆盖控制台中指定的任何名称，请选择启用语义版本控制。buildspec 文件中的名称是构建时计算得出的，使用 Shell 命令语言。例如，您可以将日期和时间附加到您的构件名称后面，以便确保其唯一性。为构件提供唯一名称可防止其被覆盖。有关更多信息，请参阅buildspec 语法。
  - 对于存储桶名称，请选择输出存储桶的名称。
  - 如果您在此过程的前面部分选择了插入构建命令，那么对于输出文件，请输入构建（该构建要放到构建输出 ZIP 文件或文件夹中）中的文件位置。对于多个位置，使用逗号将各个位置隔开（例如，`appspec.yml, target/my-app.jar`）。有关更多信息，请参阅buildspec 语法中 `files` 的描述。
  - 如果不想加密构建构件，请选择删除构件加密。

对于所需的每个辅助构件集：

1. 对于构件标识符，输入少于 128 个字符且仅包含字母数字字符和下划线的值。
2. 选择添加构件。
3. 按照前面步骤的说明配置辅助构件。
4. 选择保存构件。

其他配置

加密密钥

请执行以下操作之一：

- 要使用您账户中的 AWS 托管式密钥 Amazon S3 对构建输出项目进行加密，请将加密密钥留空。这是默认值。
- 要使用客户托管密钥加密构建输出构件，请在加密密钥中输入客户托管密钥的 ARN。采用格式 `arn:aws:kms:`*`region-ID`*`:`*`account-ID`*`:key/`*`key-ID`*。

缓存类型

对于缓存类型，请选择下列选项之一：

- 如果您不想使用缓存，请选择无缓存。
- 如果要使用 Amazon S3 缓存，请选择 Amazon S3，然后执行以下操作：
  - 对于存储桶，选择存储缓存的 S3 存储桶的名称。
  - （可选）对于缓存路径前缀，输入 Amazon S3 路径前缀。缓存路径前缀值类似于目录名称。它使您能够在存储桶的同一目录下存储缓存。

> **Important**
>
> 请勿将尾部斜杠 (/) 附加到路径前缀后面。

- 如果想要使用本地缓存，请选择本地，然后选择一个或多个本地缓存模式。

> **Note**
>
> Docker 层缓存模式仅适用于 Linux。如果您选择该模式，您的项目必须在特权模式下运行。

使用缓存可节省大量构建时间，因为构建环境的可重用部分被存储在缓存中，并且可跨构建使用。有关在 buildspec 文件中指定缓存的信息，请参阅buildspec 语法。有关缓存的更多信息，请参阅 缓存构建以提高性能。

日志

在标签部分中，选择编辑。完成更改后，请选择更新配置，以保存新的配置。

您可以修改以下属性：

选择要创建的日志。您可以创建 Amazon CloudWatch 日志、Amazon S3 日志或两者兼而有之。

CloudWatch

如果你想要 Amazon CloudWatch Logs 日志：

CloudWatch logs

选择 CloudWatch logs (CloudWatch 日志)。

组名

输入您的 Amazon CloudWatch 日志组的名称。

流名称

输入您的 Amazon CloudWatch 日志流名称。

S3

如果要创建 Amazon S3 日志：

S3 日志

选择 S3 日志。

存储桶

选择您的日志的 S3 存储桶的名称。

路径前缀

输入日志的前缀。

禁用 S3 日志加密

如果您不希望加密您的 S3 日志，请选择此选项。

更改构建项目的设置 (AWS CLI)

有关 AWS CLI 搭配使用的信息 AWS CodeBuild，请参阅命令行参考。

要使用更新 CodeBuild 项目 AWS CLI，请使用更新后的属性创建一个 JSON 文件并将该文件传递给update-project命令。更新文件中未包含的所有属性保持不变。

在更新 JSON 文件中，只需要 name 属性和修改的属性。name 属性用于标识要修改的项目。对于任何修改的结构，还必须包括这些结构所需的参数。例如，要修改项目的环境，需要 environment/type 和 environment/computeType 属性。以下是更新环境映像的示例：

```
{
  "name": "<project-name>",
  "environment": {
    "type": "LINUX_CONTAINER",
    "computeType": "BUILD_GENERAL1_SMALL",
    "image": "aws/codebuild/amazonlinux-x86_64-standard:4.0"
  }
```

```
}
```

如果需要获取项目的当前属性值，请使用 batch-get-projects 命令获取正在修改的项目的当前属性，然后将输出写入到文件。

```
aws codebuild batch-get-projects --names "<project-name>" > project-info.json
```

该*project-info.json*文件包含一组项目，因此不能直接用于更新项目。但是，您可以从*project-info.json*文件中复制要修改的属性，然后将其粘贴到更新文件中，作为要修改的属性的基准。有关更多信息，请参阅 查看构建项目的详细信息 (AWS CLI)。

按照 创建构建项目 (AWS CLI) 中所述修改更新 JSON 文件，然后保存结果。修改更新 JSON 文件后，请运行 update-project 命令，传递更新 JSON 文件。

```
aws codebuild update-project --cli-input-json file://<update-project-file>
```

如果成功，则更新后的项目 JSON 将显示在输出中。如果缺少任何必需的参数，则会在输出中显示一条错误消息，标识缺少的参数。例如，如果缺少 environment/type 参数，则会显示以下错误消息：

```
aws codebuild update-project --cli-input-json file://update-project.json

Parameter validation failed:
Missing required parameter in environment: "type"
```

更改构建项目的设置 (AWS 软件开发工具包)

有关与 AWS 软件开发工具包 AWS CodeBuild 一起使用的信息，请参阅。AWS SDK 和工具参考

里面有多个访问令牌 CodeBuild

CodeBuild 支持从您的密钥中 AWS Secrets Manager 或通过 AWS CodeConnections 连接向第三方提供商获取访问令牌。您可以将您的密钥或连接设置为与指定第三方提供商（例如 En GitHub terprise 或 Bitbucket）进行交互的默认凭据。 GitHub

您可以将您的源凭证设置为三个不同级别：

1. 适合所有项目的账户级别凭证：这些是 AWS 账户中所有项目的默认凭证。如果未指定项目或源级别凭证，则凭证将用于项目。

2. 特定存储库的源代码级别凭据：这是在项目源上定义 Secrets Manager 密钥或 AWS CodeConnections 连接的时候。这些凭证将仅用于指定源存储库上的操作。这样您就可以在同一个项目中设置具有不同权限范围的多个访问令牌，而不使用默认的账户级别凭证。
3. 项目级别的备用凭证：您可以使用 NO_SOURCE 作为主源类型来设置项目级别的备用凭证，并在其上定义一个密钥或连接。当你在一个项目上有多个来源但想对它们使用相同的凭证，或者你不想为项目使用默认的账户级别凭证时，请使用此选项。

主题

- 第 1 步：创建 Secrets Manager 密钥或 AWS CodeConnections 连接
- 第 2 步：向 CodeBuild 项目 IAM 角色授予对 Secrets Manager 密钥的访问权限
- 第 3 步：配置 Secrets Manager 或 AWS CodeConnections 令牌
- 其他设置选项

第 1 步：创建 Secrets Manager 密钥或 AWS CodeConnections 连接

按照以下说明创建 Secrets Manager 密钥或 AWS CodeConnections 连接：

- 在 Secrets Manager 密钥中创建和存储令牌.
- 创建连接到 GitHub
- 创建与 GitHub 企业服务器的连接
- 创建到 Bitbucket 的连接

第 2 步：向 CodeBuild 项目 IAM 角色授予对 Secrets Manager 密钥的访问权限

> **Note**
>
> 在继续操作之前，你必须有权访问在 Secrets Manager 中创建的令牌或者 AWS CodeConnections。

要向 CodeBuild 项目 IAM 角色授予对 Secrets Manager 或的访问权限 AWS CodeConnections，您必须添加以下 IAM 策略。

授予 CodeBuild 项目 IAM 角色访问权限

1. 按照项目的说明为您的 CodeBuild 项目创建 IAM 角色。CodeBuild 允许与其他人互动 AWS 务 CodeBuild
2. 请执行以下操作之一：
   - 将以下 IAM 策略添加到您的 CodeBuild 项目角色以授予对您的密钥的访问权限。

     JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [
                "arn:aws:iam::*:role/Service*"
            ]
        }
    ]
}
```

     （可选）如果您使用 AWS KMS 客户管理的密钥来加密 Secrets Manager 密钥，则可以添加以下政策声明来授予访问权限。

     JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:iam::*:role/Service*",
            "Condition": {
```

```
                "StringEquals": {
                    "kms:EncryptionContext:SecretARN":
 "arn:aws:iam::*:role/Service*"
                }
            }
        }
    ]
}
```

- 将以下 IAM 策略添加到您的 CodeBuild 项目角色以授予对连接的访问权限。

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codeconnections:GetConnectionToken",
                "codeconnections:GetConnection"
            ],
            "Resource": [
                "arn:aws:iam::*:role/Service*"
            ]
        }
    ]
}
```

第 3 步：配置 Secrets Manager 或 AWS CodeConnections 令牌

您可以使用 Secrets Manager 或 AWS CodeConnections 令牌将源证书设置为三个不同的级别。

配置 Secrets Manager 或 AWS CodeConnections 代币作为账户级别凭证

您可以将 Secrets Manager 密钥或 AWS CodeConnections 连接配置为账户级凭证，并在项目中使用它。

AWS 管理控制台

要将连接配置为账户级别的凭证，请参阅 AWS 管理控制台

1. 对于源提供商，请选择 Bitbucket 或 E GitHub nter prise。GitHub
2. 对于凭证，执行以下操作之一：
   - 选择默认来源凭证，使用您账户的默认来源凭证应用于所有项目。
     a. 如果未连接到源提供商，请选择管理默认来源凭证。
     b. 在凭证类型中，选择一个凭证类型。
     c. 如果您选择 AWS CodeConnections，请选择使用现有连接或创建新连接。

     如果您选择了另一个凭证类型，请在服务中选择要用于存储令牌的服务，然后执行以下操作：
     - 如果您选择使用 Secrets Manager，则可以选择使用现有密钥连接或创建新密钥并选择保存。有关如何创建新密钥的更多信息，请参阅在 Secrets Manager 密钥中创建和存储令牌。
     - 如果您选择使用 CodeBuild，请输入您的令牌或用户名和应用程序密码，然后选择保存。要获取 Bitbucket API 令牌，请改为输入您的 Atlassian 账户电子邮件地址和 API 令牌。
   - 选择自定义来源凭证，以便使用自定义来源凭证来覆盖您账户的默认设置。
     a. 在凭证类型中，选择一个凭证类型。
     b. 在连接中，选择使用现有连接或创建新连接。

AWS CLI

要将连接配置为账户级别的凭证，请参阅 AWS CLI

- 打开终端（Linux、macOS 或 Unix）或命令提示符（Windows）。 AWS CLI 使用运行import-source-credentials命令。

  使用以下命令配置 Secrets Manager 密钥：

```
aws codebuild import-source-credentials \
    --token "<secret-arn>" \
    --server-type <source-provider> \
    --auth-type SECRETS_MANAGER \
```

```
    --region <aws-region>
```

使用以下命令配置 AWS CodeConnections 连接：

```
aws codebuild import-source-credentials \
    --token "<connection-arn>" \
    --server-type <source-provider> \
    --auth-type CODECONNECTIONS \
    --region <aws-region>
```

使用此命令可以将令牌作为账户级别的默认来源凭证导入。使用 ImportSourceCredentialsAPI 导入凭证时，除非在项目中配置了一组更具体的凭证，否则 CodeBuild 将使用该令牌进行与源提供商的所有交互，例如 webhook、构建状态报告和 git clone 操作。

现在，您可以在构建项目中使用该令牌并运行它。有关更多信息，请参阅在中创建构建项目 AWS CodeBuild和手动运行 AWS CodeBuild 构建。

将多个令牌配置为源级别凭证

要使用 Secrets Manager 密钥或 AWS CodeConnections 连接作为源代码级凭据，请直接在 CodeBuild 项目中引用该令牌，然后开始构建。

AWS 管理控制台

要在中将多个令牌配置为源级凭证 AWS 管理控制台

1. 对于源提供商，请选择GitHub。
2. 对于凭证，执行以下操作之一：
   - 选择默认来源凭证，使用您账户的默认来源凭证应用于所有项目。
     a. 如果您未连接到 GitHub，请选择 “管理默认来源凭据”。
     b. 对于 “凭据类型”，选择 “GitHub 应用程序”。
     c. 在连接中，选择使用现有连接或创建新连接。
   - 选择自定义来源凭证，以便使用自定义来源凭证来覆盖您账户的默认设置。
     a. 对于 “凭据类型”，选择 “GitHub 应用程序”。
     b. 在连接中，选择使用现有连接或创建新连接。
3. 选择添加源，然后重复选择源提供商和凭证的过程。

AWS CLI

要在中将多个令牌配置为源级凭证 AWS CLI

- 打开终端（Linux、macOS 或 Unix）或命令提示符（Windows）。 AWS CLI 使用运行create-project命令。

  使用以下命令：

```
aws codebuild create-project --region <aws-region> \
    --name <project-name> \
    --artifacts type=NO_ARTIFACTS \
    --environment "type=LINUX_CONTAINER,
                   computeType=BUILD_GENERAL1_SMALL,
                   image=aws/codebuild/amazonlinux-x86_64-standard:5.0" \
    --service-role <service-role-name> \
    --source "type=GITHUB,
              location=<github-repository-1>,
              auth={type=SECRETS_MANAGER,resource=<secret-or-connection-arn-1>}"
 \
    --secondary-sources "type=GITHUB,
              location=<github-repository-2>,
              auth={type=SECRETS_MANAGER,resource=<secret-or-connection-arn-2>},
              sourceIdentifier=secondary"

aws codebuild start-build --region <aws-region> --project-name <project-name>
```

设置项目级别源凭证回退

要设置项目级别源凭证回退，请使用项目主源的 NO_SOURCE 并引用令牌。

```
aws codebuild create-project \
    --name <project-name> \
    --service-role <service-role-name> \
    --artifacts type=NO_ARTIFACTS \
    --environment "type=LINUX_CONTAINER,
                   computeType=BUILD_GENERAL1_SMALL,
                   image=aws/codebuild/amazonlinux-x86_64-standard:5.0" \
    --service-role <service-role-name> \
    --source "type=NO_SOURCE,
              auth={type=SECRETS_MANAGER,resource=<secret-or-connection-arn>},
              buildspec=<buildspec>"
```

```
    --secondary-sources "type=GITHUB,
                         location=<github-repository>,
                         sourceIdentifier=secondary"

aws codebuild start-build --region <aws-region> --project-name <project_name>
```

使用 NO_SOURCE 时，因为源模型没有直接配置为使用外部源来获取 buildspec，所以通常在源模型中提供一个 buildspec。通常，NO_SOURCE 源将负责从 buildspec 中克隆所有相关存储库。为确保配置的凭证可用于这些操作，您可以在 buildspec 中启用 `git-credential-helper` 选项。

```
env:
  git-credential-helper: yes
```

在构建过程中， CodeBuild 将从配置的令牌中读取该`AuthServer`字段，并将令牌凭据用于向该特定第三方源提供商发出的所有 git 请求。

其他设置选项

您可以使用 CloudFormation 模板配置 Secrets Manager 账户级别的证书。您可以使用以下 CloudFormation 模板来设置账户级别证书：

```
Parameters:
  GitHubToken:
    Type: String
    NoEcho: true
    Default: placeholder
Resources:
  CodeBuildAuthTokenSecret:
    Type: AWS::SecretsManager::Secret
    Properties:
      Description: CodeBuild auth token
      Name: codebuild-auth-token
      SecretString:
        !Join
          - ''
          - - '{"ServerType":"GITHUB","AuthType":"PERSONAL_ACCESS_TOKEN","Token":"'
            - !Ref GitHubToken
            - '"}'
      Tags:
        - Key: codebuild:source:provider
          Value: github
        - Key: codebuild:source:type
```

```
          Value: personal_access_token
  CodeBuildSecretsManagerAccountCredential:
    Type: AWS::CodeBuild::SourceCredential
    Properties:
      ServerType: GITHUB
      AuthType: SECRETS_MANAGER
      Token: !Ref CodeBuildAuthTokenSecret
```

> **Note**
>
> 如果您还在同一个堆栈中创建项目，请使用 CloudFormation 属性DependsOn来确保在项目之前创建AccountCredential的。

您还可以使用 CloudFormation 模板配置 Secrets Manager 多个源代码级别的凭据。您可以使用以下 CloudFormation 模板使用多个令牌来提取多个来源：

```
Parameters:
  GitHubTokenOne:
    Type: String
    NoEcho: true
    Default: placeholder
  GitHubTokenTwo:
    Type: String
    NoEcho: true
    Default: placeholder

Resources:
  CodeBuildSecretsManagerProject:
    Type: AWS::CodeBuild::Project
    Properties:
      Name: codebuild-multitoken-example
      ServiceRole: <service-role>
      Environment:
        Type: LINUX_CONTAINER
        ComputeType: BUILD_GENERAL1_SMALL
        Image: aws/codebuild/amazonlinux-x86_64-standard:5.0
      Source:
        Type: GITHUB
        Location: <github-repository-one>
        Auth:
          Type: SECRETS_MANAGER
```

```
          Resource: !Ref CodeBuildAuthTokenSecretOne
      SecondarySources:
        - Type: GITHUB
          Location: <github-repository-two>
          Auth:
            Type: SECRETS_MANAGER
            Resource: !Ref CodeBuildAuthTokenSecretTwo
          SourceIdentifier: secondary
      Artifacts:
        Type: NO_ARTIFACTS
      LogsConfig:
        CloudWatchLogs:
          Status: ENABLED
  CodeBuildProjectIAMRoleSecretAccess:
    Type: AWS::IAM::RolePolicy
    Properties:
      RoleName: <role-name>
      PolicyName: CodeBuildProjectIAMRoleSecretAccessPolicy
      PolicyDocument:
        Version: '2012-10-17       '
        Statement:
          - Effect: Allow
            Action:
              - secretsmanager:GetSecretValue
            Resource:
              - !Ref CodeBuildAuthTokenSecretOne
              - !Ref CodeBuildAuthTokenSecretTwo
  CodeBuildAuthTokenSecretOne:
    Type: AWS::SecretsManager::Secret
    Properties:
      Description: CodeBuild auth token one
      Name: codebuild-auth-token-one
      SecretString:
        !Join
          - ''
          - - '{"ServerType":"GITHUB","AuthType":"PERSONAL_ACCESS_TOKEN","Token":"'
            - !Ref GitHubTokenOne
            - '"}'
      Tags:
        - Key: codebuild:source:provider
          Value: github
        - Key: codebuild:source:type
          Value: personal_access_token
  CodeBuildAuthTokenSecretTwo:
```

```
    Type: AWS::SecretsManager::Secret
    Properties:
      Description: CodeBuild auth token two
      Name: codebuild-auth-token-two
      SecretString:
        !Join
          - ''
          - - '{"ServerType":"GITHUB","AuthType":"PERSONAL_ACCESS_TOKEN","Token":"'
            - !Ref GitHubTokenTwo
            - '"}'
      Tags:
        - Key: codebuild:source:provider
          Value: github
        - Key: codebuild:source:type
          Value: personal_access_token
```

删除中的构建项目 AWS CodeBuild

您可以使用 CodeBuild 控制台或 AWS 软件开发工具包删除中的构建项目。 AWS CLI CodeBuild如果删除项目，将不会删除其构建。

> **Warning**
>
> 您不能删除具有构建和资源策略的项目。要删除具有资源策略和构建的项目，您必须先删除资源策略及其构建。

主题

- 删除构建项目（控制台）
- 删除构建项目 (AWS CLI)
- 删除构建项目 (AWS SDK)

删除构建项目（控制台）

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择构建项目。
3. 请执行以下操作之一：

- 选择您要删除的生成项目旁边的单选按钮，然后选择删除。
- 选择您要删除的构建项目的链接，然后选择删除。

> **Note**
>
> 默认情况下，仅显示 10 个最新的构建项目。要查看更多构建项目，请为每页项目数选择其他值，或者使用向后和向前箭头查看项目。

删除构建项目 (AWS CLI)

1. 运行 `delete-project` 命令：

```
aws codebuild delete-project --name name
```

替换以下占位符：

- *name*: 必填字符串。要删除的构建项目的名称。要获取可用构建项目的列表，请运行 `list-projects` 命令。有关更多信息，请参阅 查看构建项目名称的列表 (AWS CLI)。

2. 如果成功，则输出中不会出现任何数据和错误。

有关 AWS CLI 搭配使用的更多信息 AWS CodeBuild，请参阅命令行参考。

删除构建项目 (AWS SDK)

有关使用 AWS 软件开发工具包 AWS CodeBuild 的更多信息，请参阅。AWS SDK 和工具参考

获取公共构建项目 URL

AWS CodeBuild 允许您将生成项目的生成结果、日志和构件公之于众。这样，您的源存储库的贡献者就可以查看结果并下载构建的构件，而无需他们访问 AWS 帐户。

您将自己项目的构建公之于众后，项目的所有构建结果、日志和构件，包括项目为私有状态时运行的构建，都可向公众开放。同样，当您将公共构建项目设为私有时，该项目的构建结果将不再向公众公开。

有关如何更改项目构建结果的公开可见性的信息，请参阅启用公共构建访问权限。

CodeBuild 为您的项目提供一个公共版本的 URL，该网址是您的项目所独有的。

要获取构建项目的公共 URL，请按照以下过程操作。

获取公共构建项目的 URL

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择构建项目。
3. 选择要获取其公共 URL 的构建项目的链接。
4. 公共 URL 显示在配置部分的公共项目 URL 字段中。您可以选择该链接来打开 URL，也可以使用复制按钮复制 URL。

> **Warning**
>
> 在公开项目的构建结果时，应记住以下几点：
>
> - 项目的所有构建结果、日志和构件，包括项目为私有状态时运行的构建，都可向公众开放。
> - 所有构建日志和构件都向公众开放。环境变量、源代码和其他敏感信息可能已输出到构建日志和构件中。您必须谨慎筛选将哪些信息输出到构建日志。以下是一些最佳实操：
>   - 请勿在环境变量中存储敏感值，尤其是 AWS 访问密钥 ID 和私有访问密钥。我们建议您使用 Amazon EC2 Systems Manager 参数存储或 AWS Secrets Manager 存储敏感值。
>   - 请按照使用 Webhook 的最佳实操对哪些实体可以触发构建进行限制且不要将 buildspec 存储在项目本身中，以尽可能确保您的 webhook 安全无虞。
> - 恶意用户可以使用公共构建分发恶意构件。我们建议项目管理员查看所有拉取请求，验证拉取请求是否为合法更改。我们还建议您使用校验和验证所有构件，确保下载的构件正确无误。

共享构建项目

项目共享允许项目所有者与其他 AWS 账户或用户共享他们的 AWS CodeBuild 项目。在此模型中，拥有项目的账户（拥有者）将与其他账户（使用者）共享项目。使用者无法编辑或运行项目。

主题

- 共享项目

- 相关服务
- 访问与您共享的 CodeBuild 项目
- 取消共享已共享的项目
- 标识已共享的项目
- 共享项目权限

共享项目

消费者可以使用 AWS CLI 和 AWS CodeBuild 控制台来查看您共享的项目和版本。使用者无法编辑或运行项目。

您可以将项目添加到现有资源共享，也可以在 AWS RAM 控制台中创建资源共享。

> **Note**
>
> 您不能删除其构建已添加到资源共享的项目。

要与组织单位或整个组织共享项目，您必须启用与 AWS Organizations的共享。有关更多信息，请参阅《AWS RAM 用户指南》中的允许与 AWS Organizations共享。

您可以使用 AWS CodeBuild 控制台、 AWS RAM 控制台或 AWS CLI 来共享您拥有的项目。

共享项目的先决条件

在开始共享项目之前，请确保您的 AWS 账户拥有该项目。无法共享已与您共享的项目。

共享您拥有的项目（CodeBuild 控制台）

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择构建项目。

   > **Note**
   >
   > 默认情况下，仅显示 10 个最新的构建项目。要查看更多构建项目，请选择齿轮图标，然后为每页项目数选择不同值，或使用向后和向前箭头。

3. 选择要共享的项目，然后选择共享。有关更多信息，请参阅《AWS RAM 用户指南》中的创建资源共享。

分享你拥有的项目 (AWS RAM 控制台）

请参阅《AWS RAM 用户指南》中的创建资源共享。

分享你拥有的项目 (AWS RAM 命令）

使用 create-resource-share 命令。

共享您拥有的项目（CodeBuild 命令）

使用 put-resource-policy 命令：

1. 创建一个名为 policy.json 的文件，并将以下内容复制到该文件中。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement":[{
    "Effect":"Allow",
    "Action":[
      "codebuild:BatchGetProjects",
      "codebuild:BatchGetBuilds",
      "codebuild:ListBuildsForProject"],
    "Resource":"arn:aws:iam::*:role/Service*"
  }]
}
```

2. 使用项目 ARN 和标识符更新 policy.json 以便共享项目。以下示例向 123456789012 标识的 AWS 账户的根用户授予只读访问权限。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement":[{
    "Effect":"Allow",
    "Principal":{
      "AWS": [
        "123456789012"
      ]
```

```
    },
    "Action":[
      "codebuild:BatchGetProjects",
      "codebuild:BatchGetBuilds",
      "codebuild:ListBuildsForProject"],
    "Resource":"arn:aws:codebuild:us-west-2:123456789012:project/my-project"
  }]
}
```

3. 运行 put-resource-policy 命令：

```
aws codebuild put-resource-policy --resource-arn <project-arn> --policy file://
policy.json
```

4. 获取 AWS RAM 资源共享 ARN。

```
aws ram list-resources --resource-owner SELF --resource-arns <project-arn>
```

这将返回与以下内容类似的响应：

```
{
  "resources": [
    {
      "arn": "<project-arn>",
      "type": "<type>",
      "resourceShareArn": "<resource-share-arn>",
      "creationTime": "<creation-time>",
      "lastUpdatedTime": "<last-update-time>"
    }
  ]
}
```

从响应中复制该*<resource-share-arn>*值以供下一步使用。

5. 运行 p AWS RAM romote-resource-share-create-from-policy 命令

```
aws ram promote-resource-share-created-from-policy --resource-share-arn <resource-
share-arn>
```

相关服务

项目共享与 AWS Resource Access Manager (AWS RAM) 集成，该服务使您可以与任何 AWS 帐户或通过任何帐户共享 AWS 资源 AWS Organizations。通过使用 AWS RAM，您可以通过创建资源共享 来共享资源，该共享指定要共享的资源和要与其共享资源的使用者。消费者可以是个人 AWS 账户 AWS Organizations、中的组织单位或中的整个组织 AWS Organizations。

有关更多信息，请参阅 AWS RAM 《用户指南》。

访问与您共享的 CodeBuild 项目

要访问共享项目，使用者的 IAM 角色需要 `BatchGetProjects` 权限。您可以将以下策略附加到其 IAM 角色：

```
{
    "Effect": "Allow",
    "Resource": [
        "*"
    ],
    "Action": [
        "codebuild:BatchGetProjects"
    ]
}
```

有关更多信息，请参阅 将基于身份的策略用于 AWS CodeBuild。

取消共享已共享的项目

取消共享的项目（包括其构建）只能由其拥有者访问。如果您取消共享项目，则之前与之共享的任何 AWS 账户或用户都无法访问该项目或其构建。

要取消共享您拥有的已共享项目，必须从资源共享中将其删除。您可以使用 AWS CodeBuild 控制台、控制 AWS RAM 台或执行 AWS CLI 此操作。

取消共享您拥有的共享项目 (AWS RAM 控制台）

请参阅《AWS RAM 用户指南》中的更新资源共享。

取消共享您拥有的共享项目 (AWS CLI)

使用 disassociate-resource-share 命令。

取消共享您拥有的项目（CodeBuild 命令）

运行 delete-resource-policy 命令，并指定要取消共享的项目的 ARN：

```
aws codebuild delete-resource-policy --resource-arn project-arn
```

标识已共享的项目

所有者和消费者可以使用 AWS CLI 来识别共享项目。

识别与您共享的项目 AWS 账户或用户 (AWS CLI)

使用 list-shared-projects 命令返回与您共享的项目。

共享项目权限

拥有者的权限

项目拥有者可以编辑项目并使用它来运行构建。

使用者的权限

项目使用者可以查看项目及其构建，但不能编辑项目或使用项目运行构建。

标记构建项目

标签是您或 AWS 分配给 AWS 资源的自定义属性标签。每个 AWS 标签分为两部分：

- 标签键（例如，CostCenter、Environment、Project 或 Secret）。标签密钥区分大小写。
- 一个称为标签值的可选字段（例如，111122223333、Production 或团队名称）。省略标签值与使用空字符串效果相同。与标签键一样，标签值区分大小写。

这些被统称为键-值对。有关项目可拥有的标签数量以及标签键和值的限制，请参阅标签。

标签可帮助您识别和整理 AWS 资源。许多 AWS 服务都支持标记，因此您可以为来自不同服务的资源分配相同的标签，以表明这些资源是相关的。例如，您可以为 CodeBuild 项目分配与分配给 S3 存储桶相同的标签。有关使用标签的更多信息，请参阅标记最佳实操。

在中 CodeBuild，主要资源是项目和报告组。您可以使用 CodeBuild 控制台、 AWS CLI、 CodeBuild API 或 AWS 软件开发工具包为项目添加、管理和移除标签。除了通过标签标识、组织和跟踪项目之外，您可以在 IAM 策略中使用标签，帮助控制哪些人可以查看并与您的项目交互。有关基于标签的访问策略示例，请参阅使用标签控制对以下内容的访问 AWS CodeBuild 资源。

> **Important**
>
> 使用预留容量特征时，同一账户内的其他项目可以访问实例集实例中缓存的数据，包括源文件、Docker 层和 buildspec 中指定的缓存目录。这是设计使然，让同一账户内的项目可以共享实例集实例。

主题

- 为项目添加标签
- 查看项目的标签
- 编辑项目的标签
- 从项目中移除标签

为项目添加标签

为项目添加标签可以帮助您识别和组织 AWS 资源并管理对资源的访问权限。首先，为项目添加一个或多个标签（键值对）。请记住，项目可以拥有的标签数量有限制。键和值字段中可以使用的字符有限制。有关更多信息，请参阅 标签。有了标签后，您可以创建 IAM 策略以根据这些标签管理对项目的访问。您可以使用 CodeBuild 控制台或 AWS CLI 向项目添加标签。

> **Important**
>
> 使用预留容量特征时，同一账户内的其他项目可以访问实例集实例中缓存的数据，包括源文件、Docker 层和 buildspec 中指定的缓存目录。这是设计使然，让同一账户内的项目可以共享实例集实例。

有关在创建项目时为其添加标签的更多信息，请参阅为项目添加标签（控制台）。

> **Important**
>
> 为项目添加标签之前，请务必查看是否存在任何 IAM 策略可能使用标签来控制对资源（如构建项目）的访问。有关基于标签的访问策略示例，请参阅使用标签控制对以下内容的访问 AWS CodeBuild 资源。

主题

- 为项目添加标签（控制台）
- 为项目添加标签（AWS CLI）

为项目添加标签（控制台）

您可以使用 CodeBuild 控制台向 CodeBuild 项目添加一个或多个标签。

1. 打开 CodeBuild 控制台，网址为https://console.aws.amazon.com/codebuild/。
2. 在构建项目中，选择要在其中添加标签的项目的名称。
3. 在导航窗格中，选择设置。选择构建项目标签。
4. 如果尚未向项目添加任何标签，请选择添加标签。反之，请选择编辑，然后选择添加标签。
5. 在键中，输入标签的名称。您可以在值中添加可选的标签值。
6. （可选）要添加其他标签，请再次选择添加标签。
7. 添加完标签后，选择提交。

为项目添加标签（AWS CLI）

要在创建项目时为其添加标签，请参阅创建构建项目 (AWS CLI)。在 `create-project.json` 中，添加您的标签。

在这些步骤中，我们假设您已安装最新版本的 AWS CLI 或已更新到当前版本。有关更多信息，请参阅安装 AWS Command Line Interface。

如果成功，该命令不返回任何内容。

查看项目的标签

标签可以帮助您识别和整理 AWS 资源并管理对资源的访问权限。有关使用标签的更多信息，请参阅标记最佳实践白皮书。有关基于标签的访问策略示例，请参阅使用标签控制对以下内容的访问 AWS CodeBuild 资源。

查看项目的标签（控制台）

您可以使用 CodeBuild 控制台查看与 CodeBuild 项目关联的标签。

1. 打开 CodeBuild 控制台，网址为https://console.aws.amazon.com/codebuild/。

2. 在构建项目中，选择要在其中查看标签的项目的名称。
3. 在导航窗格中，选择设置。选择构建项目标签。

查看项目的标签（AWS CLI）

要查看构建项目的标签，请运行以下命令。使用项目名称作为 `--names` 参数。

```
aws codebuild batch-get-projects --names your-project-name
```

如果成功，此命令将返回有关您的构建项目 JSON-formatted 的信息，其中包括以下内容：

```
{
    "tags": {
        "Status": "Secret",
        "Team": "JanesProject"
    }
}
```

如果项目没有标签，则 `tags` 部分为空：

```
"tags": []
```

编辑项目的标签

您可以更改与项目关联的标签值。您也可以更改键的名称，这相当于移除当前的标签并使用新名称和与另一个键相同的值添加一个不同的标签。请记住，键和值字段中可以使用的字符有限制。有关更多信息，请参阅 标签。

> **Important**
>
> 编辑项目的标签会影响对该项目的访问。编辑项目的标签名称（键）或值之前，请务必查看是否存在任何 IAM 策略可能使用标签的键或值来控制对资源（如构建项目）的访问。有关基于标签的访问策略示例，请参阅使用标签控制对以下内容的访问 AWS CodeBuild 资源。

编辑项目的标签（控制台）

您可以使用 CodeBuild 控制台编辑与 CodeBuild 项目关联的标签。

1. 打开 CodeBuild 控制台，网址为https://console.aws.amazon.com/codebuild/。
2. 在构建项目中，选择要在其中编辑标签的项目的名称。
3. 在导航窗格中，选择设置。选择构建项目标签。
4. 选择编辑。
5. 请执行以下操作之一：
   - 要更改标签，则在键中输入新名称。更改标签的名称相当于删除标签并使用新的键名添加新标签。
   - 要更改标签的值，则输入新值。如果您想将标签值清空，请删除当前的值并将字段保留为空白。
6. 编辑完标签后，选择提交。

编辑项目的标签（AWS CLI）

要添加、更改或移除构建项目中的标签，请参阅更改构建项目的设置 (AWS CLI)。更新 JSON-formatted 数据中用于更新项目的`tags`部分。

从项目中移除标签

您可以移除与项目关联的一个或多个标签。移除标签不会从与该标签关联的其他 AWS 资源中删除该标签。

> **Important**
>
> 移除项目的标签会影响对该项目的访问。从项目中移除标签之前，请务必查看是否存在任何 IAM 策略可能使用标签的键或值来控制对资源（如构建项目）的访问。有关基于标签的访问策略示例，请参阅使用标签控制对以下内容的访问 AWS CodeBuild 资源。

从项目中移除标签（控制台）

您可以使用 CodeBuild 控制台删除标签和 CodeBuild项目之间的关联。

1. 打开 CodeBuild 控制台，网址为https://console.aws.amazon.com/codebuild/。
2. 在构建项目中，选择要在其中移除标签的项目的名称。
3. 在导航窗格中，选择设置。选择构建项目标签。

4. 选择编辑。
5. 找到要移除的标签，然后选择移除标签。
6. 移除标签之后，选择提交。

从项目中移除标签（AWS CLI）

要从构建项目中删除一个或多个标签，请参阅更改构建项目的设置 (AWS CLI)。使用不包含要删除的标签的更新列表来更新 JSON-formatted 数据中的tags部分。如果要删除所有标签，请将 tags 部分更新为：

```
"tags: []"
```

> **Note**
>
> 如果删除 CodeBuild 构建项目，则会从已删除的构建项目中移除所有标签关联。您无需在删除构建项目之前移除标签。

将跑步器与 AWS CodeBuild

AWS CodeBuild 支持与 Action GitHub s 运行器、自我管理的 GitLab 运行器和 Buildkite 运行器集成。

主题

- Self-hosted GitHub 动作跑步器进来 AWS CodeBuild
- Self-managed GitLab 跑步者进来 AWS CodeBuild
- Self-managed Buildkite runner AWS CodeBuild

Self-hosted GitHub 动作跑步器进来 AWS CodeBuild

您可以将项目配置为在 CodeBuild容器中设置自托管的 Action GitHub s 运行器来处理您的 Actions 工作流程 GitHub 作业。这可以通过使用您的 CodeBuild 项目设置 webhook，然后更新 GitHub 操作工作流程 YAML 以使用托管在计算机上的自托管运行器来完成。 CodeBuild

配置 CodeBuild 项目以运行 GitHub 操作作业的高级步骤如下：

1. 如果您还没有这样做，请创建个人访问令牌或连接 OAuth 应用程序以连接您的项目。 GitHub

2. 导航到 CodeBuild 控制台并使用 webhook 创建 CodeBuild 项目，然后设置 webhook 过滤器。

3. 更新您的 GitHub 操作工作流程 YAML GitHub 以配置您的构建环境。

有关更详细的过程，请参阅教程：配置 CodeBuild-hosted GitHub 操作运行器。

此功能允许您的 A GitHub ctions 工作流程任务与之进行原生集成 AWS，从而通过 IAM AWS CloudTrail、 AWS Secrets Manager 集成和 Amazon VPC 等功能提供安全性和便利性。您可以访问最新的实例类型，包括 ARM-based实例。

主题

- 关于 CodeBuild-hosted GitHub 动作运行器
- 教程：配置 CodeBuild-hosted GitHub 操作运行器
- 排查 webhook 的问题
- CodeBuild-hosted GitHub 操作运行器支持的标签覆盖
- 计算 CodeBuild-hosted GitHub 动作运行器支持的图像

关于 CodeBuild-hosted GitHub 动作运行器

以下是有关 CodeBuild-hosted GitHub 操作运行器的一些常见问题。

我应该何时在标签中包括映像和实例覆盖？

您可以在标签中包含图像和实例覆盖，以便为每个 Actions 工作流程任务指定不同的构建环境。GitHub 无需创建多个 CodeBuild 项目或 webhook 即可完成此操作。例如，当您需要为工作流作业使用矩阵时，这很有用。

```
name: Hello World
on: [push]
jobs:
  Hello-World-Job:
    runs-on:
      - codebuild-myProject-${{ github.run_id }}-${{ github.run_attempt }}
        image:${{ matrix.os }}
        instance-size:${{ matrix.size }}
    strategy:
      matrix:
        include:
```

```
          - os: arm-3.0
            size: small
          - os: linux-5.0
            size: large
    steps:
      - run: echo "Hello World!"
```

> **Note**
>
> 如果runs-on有多个包含 GitHub 操作上下文的标签，则可能需要使用引号。

我能用吗 CloudFormation 为了这个功能？

是的，您可以在 CloudFormation 模板中包含一个筛选器组，用于在项目 webhook 中指定 GitHub操作工作流程作业事件过滤器。

```
Triggers:
  Webhook: true
  FilterGroups:
    - - Type: EVENT
        Pattern: WORKFLOW_JOB_QUEUED
```

有关更多信息，请参阅 过滤 GitHub webhook 事件 (CloudFormation)。

如果您在 CloudFormation 模板中设置项目凭证时需要帮助，请参阅《AWS CloudFormation 用户指南》SourceCredential中的AWS::CodeBuild:: 以了解更多信息。

使用此特征时如何屏蔽密钥？

默认情况下，系统不会屏蔽日志中显示的密钥。如果您想屏蔽密钥，可以使用以下语法：`::add-mask::`*`value`*。以下是如何在 YAML 中使用此语法的示例：

```
name: Secret Job
on: [push]
jobs:
  Secret-Job:
    runs-on: codebuild-myProject-${{ github.run_id }}-${{ github.run_attempt }}
    env:
      SECRET_NAME: "secret-name"
```

```
    steps:
      - run: echo "::add-mask::$SECRET_NAME"
```

有关更多信息，请参阅屏蔽登录 GitHub中的值。

我能否在一个项目中接收 GitHub 来自多个存储库的 Actions webhook 事件？

CodeBuild 支持组织和全局级 webhook，它们接收来自指定组织或企业的事件。有关更多信息，请参阅 GitHub 全球和组织网络挂钩。

哪些地区支持使用 CodeBuild-hosted GitHub 操作运行器？

CodeBuild-hosted GitHub 所有 CodeBuild 区域都支持操作运行器。有关 AWS 区域 何处 CodeBuild 可用的更多信息，请参阅按地区划分的AWS 服务。

哪些平台支持使用 CodeBuild-hosted GitHub 动作运行器？

CodeBuild-hosted GitHub Amazon EC2 和AWS Lambda计算均支持操作运行器。您可以使用以下平台：Amazon Linux 2、Amazon Linux 2023、Ubuntu 和 Windows Server Core 2019。有关更多信息，请参阅EC2 计算映像和Lambda 计算映像。

教程：配置 CodeBuild-hosted GitHub 操作运行器

本教程向您展示如何配置 CodeBuild 项目以运行 Action GitHub s 作业。有关将 GitHub 操作与配合使用的更多信息， CodeBuild 请参阅教程：配置 CodeBuild-hosted GitHub 操作运行器。

要完成本教程，您首先必须：

- 使用个人访问令牌、Secrets Manager 密钥、OAuth 应用程序或 GitHub 应用程序进行连接。如果您想连接 OAuth 应用程序，则必须使用 CodeBuild 控制台进行连接。如果您想创建个人访问令牌，则可以使用 CodeBuild 控制台或使用 ImportSourceCredentials API。有关更多说明，请参阅 GitHub 和 GitHub 企业服务器访问权限 CodeBuild。
- Connect CodeBuild 到您的 GitHub 账户。为此，您可以执行以下操作之一：
  - 您可以在控制台中添加 GitHub 为源提供商。您可以使用个人访问令牌、Secrets Manager 密钥、OAuth 应用程序或 GitHub 应用程序进行连接。有关说明，请参阅GitHub 和 GitHub 企业服务器访问权限 CodeBuild。
  - 您可以通过 ImportSourceCredentials API 导入您的 GitHub 证书。只有使用个人访问令牌才能执行此操作。如果您使用 OAuth 应用程序来连接，则必须改用控制台进行连接。有关说明，请参阅GitHub 使用访问令牌 (CLI) 连接。

> **Note**
>
> 只有当你的账户还没有连接时， GitHub 才需要这样做。

第 1 步：使用 webhook 创建 CodeBuild 项目

在此步骤中，您将创建一个带有 webhook 的 CodeBuild 项目，并在 GitHub 控制台中对其进行审核。您也可以选择 E GitHub nterprise 作为您的源提供商。要了解有关在 GitHub 企业版中创建 webhook 的更多信息，请参阅GitHub 手动网络挂钩。

使用 webhook 创建 CodeBuild 项目

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 创建构建项目。有关信息，请参阅创建构建项目（控制台）和运行构建（控制台）。
3. 在项目类型中，选择运行程序项目。

   在运行程序中：

   a. 对于跑步者提供商，请选择GitHub。

   b. 对于运行程序位置，请选择存储库。

   c. 在 “存储库” 下的 “存储库 URL” 中，选择https://github.com/user-name/repository-name。

   > **Note**
   >
   > 默认情况下，您的项目将仅接收单个存储库的 `WORKFLOW_JOB_QUEUED` 事件。如果您想接收组织或企业内所有存储库的事件，请参阅GitHub 全球和组织网络挂钩。

4. • 在环境中：
   - 选择支持的环境映像和计算。请注意，您可以选择在 GitHub 操作工作流程 YAML 中使用标签来覆盖图像和实例设置。有关更多信息，请参阅 第 2 步：更新您的 GitHub 操作工作流程 YAML。
   - 在 Buildspec (构建规范) 中：
     - 请注意，除非将 `buildspec-override:true` 作为标签添加，否则系统会忽略 buildspec。相反， CodeBuild 将覆盖它以使用设置自托管运行器的命令。

5. 继续使用默认值，然后选择创建构建项目。
6. 打开 GitHub 控制台，验证是否已创建一个 webhook 并已启用 webhook 来传送工作流作业事件。https://github.com/*user-name*/*repository-name*/settings/hooks

第 2 步：更新您的 GitHub 操作工作流程 YAML

在此步骤中，您将更新 GitHub 操作工作流程 YAML 文件GitHub以配置您的构建环境并在中使用 GitHub Actions 自托管运行器。 CodeBuild有关更多信息，请参阅在自托管运行器中使用标签和CodeBuild-hosted GitHub 操作运行器支持的标签覆盖。

更新你的 GitHub 操作工作流程 YAML

导航到 GitHub 操作工作流程 YAML 中的runs-on设置GitHub并进行更新，以配置您的构建环境。为此，您可以执行以下操作之一：

- 您可以指定项目名称和运行 ID，在这种情况下，构建将使用计算、映像、映像版本和实例大小的现有项目配置。需要项目名称才能将 Actions 作业的 AWS相关设置链接到特定 CodeBuild项目。GitHub 通过在 YAML 中包含项目名称 CodeBuild ，可以调用具有正确项目设置的作业。通过提供运行 ID，可以将您的构建 CodeBuild 映射到特定的工作流程运行，并在取消工作流程运行时停止构建。有关更多信息，请参阅 github 上下文。

```
runs-on: codebuild-<project-name>-${{ github.run_id }}-${{ github.run_attempt }}
```

> **Note**
>
> 请确保您的名称与您在上一步中创建的项目名称*<project-name>*相匹配。如果不匹配，CodeBuild 则不会处理 webhook， GitHub操作工作流程可能会挂起。

以下是 GitHub 操作工作流程 YAML 的示例：

```
name: Hello World
on: [push]
jobs:
  Hello-World-Job:
    runs-on:
      - codebuild-myProject-${{ github.run_id }}-${{ github.run_attempt }}
    steps:
      - run: echo "Hello World!"
```

- 您也可以在标签中覆盖映像和计算类型。有关精选映像的列表，请参阅计算 CodeBuild-hosted GitHub 动作运行器支持的图像。有关使用自定义映像，请参阅CodeBuild-hosted GitHub 操作运行器支持的标签覆盖。标签中的计算类型和映像将覆盖项目的环境设置。要替换 CodeBuild EC2 或 Lambda 计算版本的环境设置，请使用以下语法：

```
runs-on:
  - codebuild-<project-name>-${{ github.run_id }}-${{ github.run_attempt }}
    image:<environment-type>-<image-identifier>
    instance-size:<instance-size>
```

以下是 GitHub 操作工作流程 YAML 的示例：

```
name: Hello World
on: [push]
jobs:
  Hello-World-Job:
    runs-on:
      - codebuild-myProject-${{ github.run_id }}-${{ github.run_attempt }}
        image:arm-3.0
        instance-size:small
    steps:
      - run: echo "Hello World!"
```

- 您可以在标签中覆盖构建所用的实例集。这将覆盖在您的项目中配置的实例集设置，以便使用指定的实例集。有关更多信息，请参阅 在预留容量实例集上运行构建。要覆盖 Amazon EC2 计算构建的实例集设置，请使用以下语法：

```
runs-on:
  - codebuild-<project-name>-${{ github.run_id }}-${{ github.run_attempt }}
    fleet:<fleet-name>
```

要同时覆盖构建所用的实例集和映像，请使用以下语法：

```
runs-on:
  - codebuild-<project-name>-${{ github.run_id }}-${{ github.run_attempt }}
    fleet:<fleet-name>
    image:<environment-type>-<image-identifier>
```

以下是 GitHub 操作工作流程 YAML 的示例：

```
name: Hello World
on: [push]
jobs:
  Hello-World-Job:
    runs-on:
      - codebuild-myProject-${{ github.run_id }}-${{ github.run_attempt }}
        fleet:myFleet
        image:arm-3.0
    steps:
      - run: echo "Hello World!"
```

- 要在自定义图像上运行 Actions GitHub 作业，您可以在 CodeBuild 项目中配置自定义图像，避免提供图像覆盖标签。 CodeBuild 如果未提供图像覆盖标签，则将使用项目中配置的图像。
- 或者，您可以在 CodeBuild 支持的标签之外提供标签。在覆盖构建的属性时会忽略这些标签，但不会导致 webhook 请求失败。例如，添加 `testLabel` 作为标签不会阻止构建运行。

> **Note**
>
> 如果 GitHub-hosted 运行器提供的依赖项在 CodeBuild环境中不可用，则可以在工作流程运行中使用 Act GitHub ions 安装依赖项。例如，您可以使用 `setup-python` 操作为构建环境安装 Python。

在 INSTALL、PRE_BUILD 和 POST_BUILD 阶段运行 buildspec 命令

默认情况下，在运行自托管 GitHub 的 Actions 版本时 CodeBuild 会忽略所有 buildspec 命令。要在构建期间运行 buildspec 命令，可以将 `buildspec-override:true` 作为后缀添加到标签中：

```
runs-on:
  - codebuild-<project-name>-${{ github.run_id }}-${{ github.run_attempt }}
    buildspec-override:true
```

通过使用此命令， CodeBuild 将在容器的主源文件夹`actions-runner`中创建一个名为的文件夹。当 Ac GitHub tions 运行器在该`BUILD`阶段启动时，运行器将在`actions-runner`目录中运行。

在自托管的 Actions 版本中使用 buildspec 覆盖有几个限制 GitHub ：

- CodeBuild 在此`BUILD`阶段不会运行 buildspec 命令，因为自托管运行器将在该`BUILD`阶段运行。

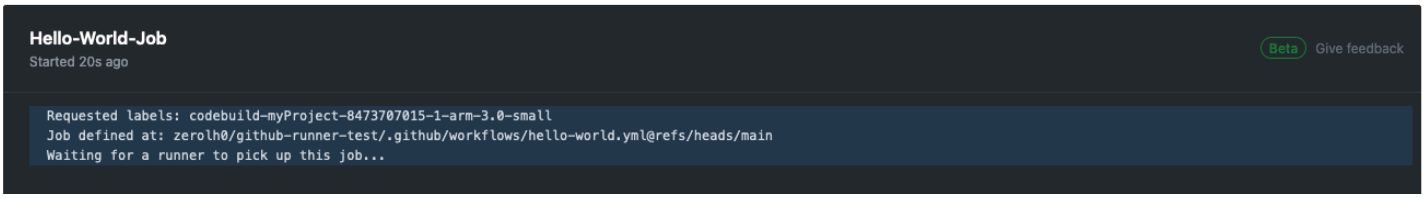
- CodeBuild 在此DOWNLOAD_SOURCE阶段不会下载任何主要或次要来源。如果您配置了 buildspec 文件，则只会从项目的主源下载该文件。
- 如果构建命令在PRE_BUILD或INSTALL阶段失败， CodeBuild 则无法启动自托管运行器，并且需要手动取消 GitHub 操作工作流程作业。
- CodeBuild 在该阶段获取跑步者令牌，该DOWNLOAD_SOURCE阶段的到期时间为一小时。如果您的PRE_BUILD或INSTALL阶段超过一小时，则运行器令牌可能会在 GitHub 自托管运行器启动之前过期。

步骤 3：检查您的结果

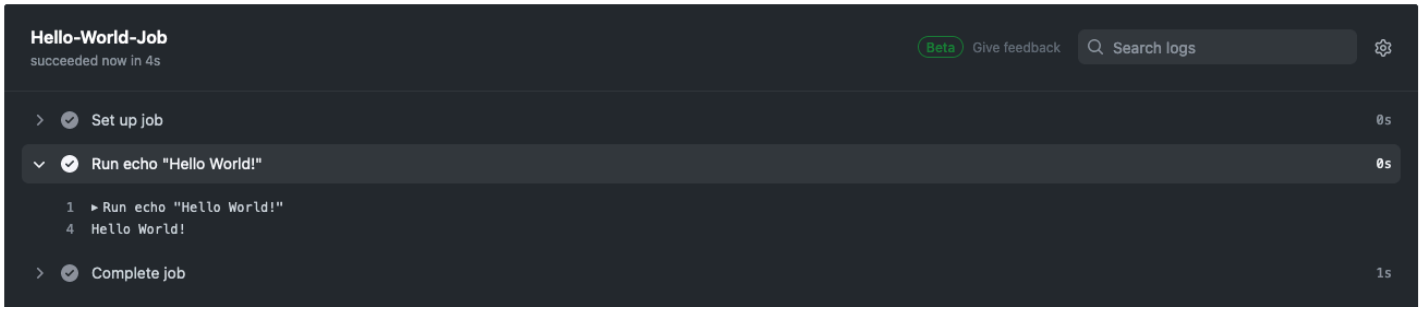
每当 GitHub 操作工作流程运行时， CodeBuild 都会通过 webhook 接收工作流程作业事件。对于工作流程中的每个作业， CodeBuild 启动构建以运行临时的 Actions GitHub 运行器。该运行器负责执行单个工作流作业。作业完成后，运行器和关联的构建过程会立即终止。

要查看您的工作流程作业日志，请导航到中的 GitHub存储库，选择操作，选择所需的工作流程，然后选择要查看日志的特定作业。

当任务等待中的自托管运行器接管时，您可以在日志中 CodeBuild查看请求的标签。



作业完成后，您将能够查看该作业的日志。



GitHub 操作运行器配置选项

您可以在项目配置中指定以下环境变量，以修改自托管运行程序的设置配置。

`CODEBUILD_CONFIG_GITHUB_ACTIONS_ORG_REGISTRATION_NAME`

CodeBuild 会将自托管的运行器注册到指定为该环境变量值的组织名称。有关在组织级别注册运行程序和必要权限的更多信息，请参阅 Create configuration for a just-in-time runner for an organization。

`CODEBUILD_CONFIG_GITHUB_ACTIONS_ENTERPRISE_REGISTRATION_NAME`

CodeBuild 会将自托管的运行器注册到指定为该环境变量值的企业名称。有关在企业级别注册运行程序和必要权限的更多信息，请参阅 Create configuration for a just-in-time runner for an Enterprise。

> **Note**
>
> 默认情况下，企业运行程序不可用于组织存储库。要让自托管运行程序承担工作流程作业，您可能需要配置运行程序组访问权限设置。有关更多信息，请参阅 Making enterprise runners available to repositories。

`CODEBUILD_CONFIG_GITHUB_ACTIONS_RUNNER_GROUP_ID`

CodeBuild 会将自托管的运行器注册到存储为该环境变量值的整数运行器组 ID。默认情况下，该值为 1。有关自托管运行程序组的更多信息，请参阅 Managing access to self-hosted runners using groups。

`CODEBUILD_CONFIG_GITHUB_ACTIONS_ORG_REGISTRATION_NAME`

要使用 Actions 工作流程 YAML 文件配置组织级别的运行器注册，可以使用以下语法： GitHub

```
name: Hello World
on: [push]
jobs:
  Hello-World-Job:
    runs-on:
      - codebuild-myProject-${{ github.run_id }}-${{ github.run_attempt }}
        organization-registration-name:myOrganization
    steps:
      - run: echo "Hello World!"
```

`CODEBUILD_CONFIG_GITHUB_ACTIONS_ENTERPRISE_REGISTRATION_NAME`

要使用 GitHub操作工作流程 YAML 文件配置企业级运行器注册，可以使用以下语法：

```
name: Hello World
on: [push]
jobs:
  Hello-World-Job:
    runs-on:
      - codebuild-myProject-${{ github.run_id }}-${{ github.run_attempt }}
        enterprise-registration-name:myEnterprise
    steps:
      - run: echo "Hello World!"
```

CODEBUILD_CONFIG_GITHUB_ACTIONS_RUNNER_GROUP_ID

要使用 GitHub 操作工作流程 YAML 文件配置将运行器注册到特定的运行器组 ID，可以使用以下语法：

```
name: Hello World
on: [push]
jobs:
  Hello-World-Job:
    runs-on:
      - codebuild-myProject-${{ github.run_id }}-${{ github.run_attempt }}
        registration-group-id:3
    steps:
      - run: echo "Hello World!"
```

筛选 GitHub 操作 webhook 事件 (CloudFormation)

CloudFormation 模板的以下 YAML-formatted 部分创建了一个筛选器组，当计算结果为 true 时，该筛选器组会触发构建。以下筛选器组指定 Acti GitHub ons 工作流任务请求，其工作流程名称与正则表达式匹配\[CI-CodeBuild\]。

```
CodeBuildProject:
  Type: AWS::CodeBuild::Project
  Properties:
    Name: MyProject
    ServiceRole: service-role
    Artifacts:
      Type: NO_ARTIFACTS
    Environment:
      Type: LINUX_CONTAINER
      ComputeType: BUILD_GENERAL1_SMALL
```

```
      Image: aws/codebuild/standard:5.0
    Source:
      Type: GITHUB
      Location: CODEBUILD_DEFAULT_WEBHOOK_SOURCE_LOCATION
    Triggers:
      Webhook: true
      ScopeConfiguration:
        Name: organization-name
        Scope: GITHUB_ORGANIZATION
      FilterGroups:
        - - Type: EVENT
            Pattern: WORKFLOW_JOB_QUEUED
          - Type: WORKFLOW_NAME
            Pattern: \[CI-CodeBuild\]
```

筛选 GitHub 操作 webhook 事件 (AWS CDK)

以下 AWS CDK 模板创建了一个筛选器组，当生成结果为 true 时会触发构建。以下筛选器组指定了 GitHub 操作工作流程任务请求。

```
import { aws_codebuild as codebuild } from 'aws-cdk-lib';
import {EventAction, FilterGroup} from "aws-cdk-lib/aws-codebuild";

const source = codebuild.Source.gitHub({
      owner: 'owner',
      repo: 'repo',
      webhook: true,
      webhookFilters: [FilterGroup.inEventOf(EventAction.WORKFLOW_JOB_QUEUED)],
    })
```

筛选 GitHub 操作 webhook 事件 (Terraform)

以下 Terraform 模板创建一个筛选条件组，该组在计算结果为 true 时会触发构建。以下筛选器组指定了 GitHub 操作工作流程任务请求。

```
resource "aws_codebuild_webhook" "example" {
  project_name = aws_codebuild_project.example.name
  build_type   = "BUILD"
  filter_group {
    filter {
      type    = "EVENT"
      pattern = "WORKFLOW_JOB_QUEUED"
    }
```

```
  }
}
```

筛选 GitHub 操作 webhook 事件 (AWS CLI)

以下 AWS CLI 命令创建一个自托管 GitHub 的 Actions 运行器项目，其中包含一个 Acti GitHub ons 工作流任务请求筛选器组，该筛选器组在评估结果为 true 时触发构建。

```
aws codebuild create-project \
--name <project name> \
--source "{\"type\":\"GITHUB\",\"location\":\"<repository location>\",\"buildspec\":
\"\"}" \
--artifacts {"\"type\":\"NO_ARTIFACTS\""} \
--environment "{\"type\": \"LINUX_CONTAINER\",\"image\": \"aws/codebuild/amazonlinux-
x86_64-standard:5.0\",\"computeType\": \"BUILD_GENERAL1_MEDIUM\"}" \
--service-role "<service role ARN>"
```

```
aws codebuild create-webhook \
--project-name <project name> \
--filter-groups "[[{\"type\":\"EVENT\",\"pattern\":\"WORKFLOW_JOB_QUEUED\"}]]"
```

排查 webhook 的问题

问题：您在中设置的 webhook 教程：配置 CodeBuild-hosted GitHub 操作运行器 无法正常工作或您的工作流程任务暂停。 GitHub

可能的原因：

- 您的 webhook 工作流程作业事件可能无法触发构建。检查响应日志以查看响应或错误消息。
- 由于标签配置，您的作业分配给了不正确的运行程序代理。当单个工作流程运行中一个作业的标签少于另一个作业时，就会出现此问题。例如，如果您在同一个工作流程运行中具有两个带有以下标签的作业：
  - 作业 1：`codebuild-myProject-${{ github.run_id }}-${{ github.run_attempt }}`
  - 作业 2：`codebuild-myProject-${{ github.run_id }}-${{ github.run_attempt }}`、`instance-size:medium`

  路由自托管的 Ac GitHub tions 作业时， GitHub 会将该任务路由到具有所有作业指定标签的任何运行器。此行为意味着为作业 1 或作业 2 创建的运行程序可以承担作业 1，但是由于作业 2 有附加标

签，因此只能由为作业 2 创建的运行程序承担该作业。如果为作业 2 创建的运行程序承担了作业 1，则作业 2 将卡住，因为作业 1 运行程序没有 `instance-size:medium` 标签。

推荐的解决方案：

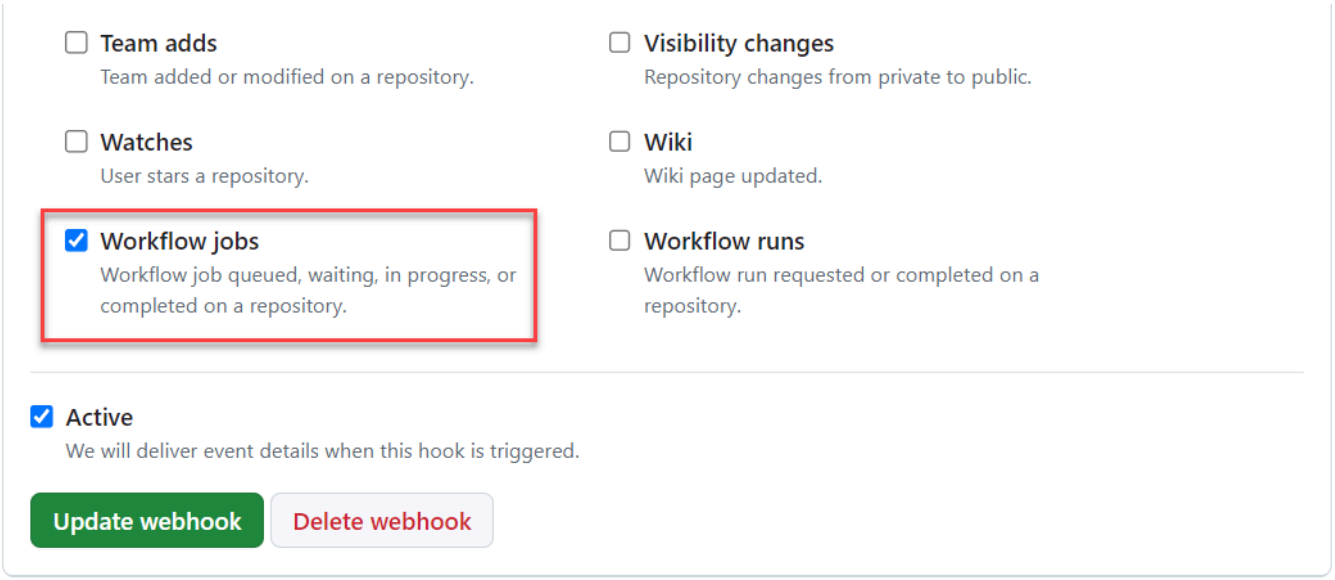
在同一个工作流程运行中创建多个作业时，请为每个作业使用相同数量的标签覆盖，或者为每个作业分配一个自定义标签，例如 `job1` 或 `job2`。

为每项任务分配自己的唯一自定义标签是最可靠的选择。它可以确保任何作业的标签集都不是另一个作业的标签集的子集。当每个作业都带有其他作业没有的标签时，无论每个作业使用多少其他替换，都只将每个作业与为其创建的运行器进行 GitHub 匹配。在以下工作流程中，每个作业都有一个唯一的标签（`job1`和`job2`）。尽管 Job 2 有额外的`instance-size:medium`替换，但作业 1 无法再与为作业 2 创建的运行器匹配，因此两个作业都不会卡住：

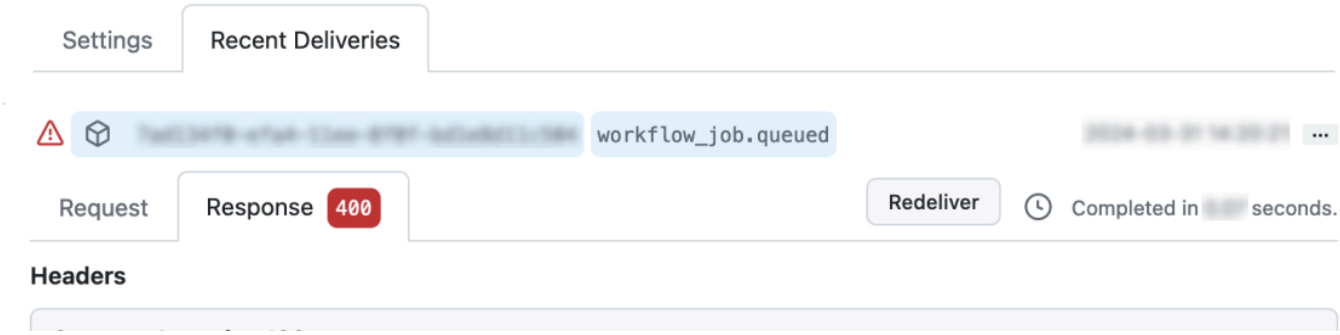
```
name: Hello World
on: [push]
jobs:
  job1:
    runs-on:
      - codebuild-myProject-${{ github.run_id }}-${{ github.run_attempt }}
      - job1
    steps:
      - run: echo "Hello from Job 1!"
  job2:
    runs-on:
      - codebuild-myProject-${{ github.run_id }}-${{ github.run_attempt }}
      - instance-size:medium
      - job2
    steps:
      - run: echo "Hello from Job 2!"
```

如果错误仍然存在，请按照以下说明调试问题。

1. 打开 GitHub 控制台，查看存储库的 webhook 设置。`https://github.com/`*`user-name`*`/`*`repository-name`*`/settings/hooks`在此页面上，您将看到为您的存储库创建的 webhook。
2. 选择编辑并确认已启用 webhook 来传送工作流作业事件。



3. 导航至最近传输选项卡，找到相应的 `workflow_job.queued` 事件，然后展开该事件。
4. 查看有效负载中的标签字段，并确保其符合预期。
5. 最后，查看 “响应” 选项卡，因为其中包含返回的响应或错误消息 CodeBuild。



6. 或者，您可以使用的 API 调试 webhook 故障。 GitHub您可以使用列出存储库 webhook 的传输 API 来查看 webhook 的近期传输：

```
gh api \
  -H "Accept: application/vnd.github+json" \
  -H "X-GitHub-Api-Version: 2022-11-28" \
  /repos/owner/repo/hooks/hook-id/deliveries
```

找到要调试的 webhook 交付并记下交付 ID 后，您可以使用获取存储库 webhook 的交付 API。CodeBuild对 webhook 交付有效负载的响应可以在以下`response`部分中找到：

```
gh api \
  -H "Accept: application/vnd.github+json" \
```

```
  -H "X-GitHub-Api-Version: 2022-11-28" \
  /repos/owner/repo/hooks/hook-id/deliveries/delivery-id
```

问题：启用部署保护规则的 GitHub 操作会在部署获得批准 CodeBuild 之前在内部生成触发器。

可能的原因： CodeBuild 获取与 Actions 作业关联的 GitHub 部署和环境（如果存在），以验证部署是否获得批准。如果 CodeBuild 无法获取部署或环境，则可能会过早触发 CodeBuild 构建。

推荐的解决方案：验证与您的 CodeBuild 项目关联的凭证是否具有内部部署和操作的读取权限 GitHub。

CodeBuild-hosted GitHub 操作运行器支持的标签覆盖

在你的 A GitHub ctions 工作流程 YAML 中，你可以提供各种标签覆盖来修改你的自托管运行器构建。任何未被识别的版本都 CodeBuild 将被忽略，但不会使您的 webhook 请求失败。以下工作流程 YAML 定义了两个任务：对图像、实例大小、队列和 buildspec `job1` 使用替代，并且仅`job2`使用图像覆盖。为每个作业指定一个唯一的标签（`job1`和`job2`），以便将每个作业 GitHub 路由到为其创建的运行器：

```
name: Hello World
on: [push]
jobs:
  job1:
    runs-on:
      - codebuild-myProject-${{ github.run_id }}-${{ github.run_attempt }}
      - image:arm-3.0
      - instance-size:small
      - fleet:myFleet
      - buildspec-override:true
      - job1
    steps:
      - run: echo "Hello from Job 1!"
  job2:
    runs-on:
      - codebuild-myProject-${{ github.run_id }}-${{ github.run_attempt }}
      - image:linux-5.0
      - job2
    steps:
      - run: echo "Hello from Job 2!"
```

您也可以使用矩阵策略在多个配置中运行单个作业。单个矩阵作业的扩展总是有相同数量的标签，因此运行器匹配问题不会影响它们。但是，如果工作流程运行包含多个矩阵作业，请为每个矩阵作业指定其唯一的自定义标签（例如matrix-job-1和matrix-job-2），这样作业就 GitHub 不会与为其他矩阵作业创建的运行器相匹配：

```
name: Hello World
on: [push]
jobs:
  Hello-World-Job:
    runs-on:
      - codebuild-myProject-${{ github.run_id }}-${{ github.run_attempt }}
      - image:${{ matrix.os }}
      - instance-size:${{ matrix.size }}
      - fleet:myFleet
      - buildspec-override:true
      - matrix-job-1
    strategy:
      matrix:
        include:
          - os: arm-3.0
            size: small
          - os: linux-5.0
            size: large
    steps:
      - run: echo "Hello World!"
```

> **Note**
>
> 如果您的工作流程任务处于暂停 GitHub状态，请参阅排查 webhook 的问题和使用自定义标签路由作业。

`codebuild-<project-name>-${{github.run_id}}-${{github.run_attempt}}`（必需）

- 示例：`codebuild-fake-project-${{ github.run_id }}-${{ github.run_attempt }}`
- 所有 GitHub 操作工作流程 YAML 均为必填项。*<project name>*应等于为其配置自托管运行器 webhook 的项目的名称。

`image:<environment-type>-<image-identifier>`

- 示例：image:arm-3.0
- 覆盖在通过精选映像启动自托管运行程序构建时使用的映像和环境类型。要了解支持的值，请参阅计算 CodeBuild-hosted GitHub 动作运行器支持的图像。
  - 要覆盖与自定义映像结合使用的映像和环境类型，请使用 image:custom-*<environment-type>*-*<custom-image-identifier>*
  - 示例：image:custom-arm-public.ecr.aws/codebuild/amazonlinux-aarch64-standard:3.0

> **Note**
>
> 如果自定义映像位于私有注册表中，请参阅为自托管运行程序配置私有注册表凭证。

instance-size:*<instance-size>*

- 示例：instance-size:medium
- 覆盖在启动自托管运行器构建时使用的实例类型。要了解支持的值，请参阅计算 CodeBuild-hosted GitHub 动作运行器支持的图像。

fleet:*<fleet-name>*

- 示例：fleet:myFleet
- 覆盖在您的项目中配置的实例集设置，以便使用指定的实例集。有关更多信息，请参阅 在预留容量实例集上运行构建。

buildspec-override:*<boolean>*

- 示例：buildspec-override:true
- 如果设置为 true，则允许构建以在 INSTALL、PRE_BUILD 和 POST_BUILD 阶段运行 buildspec 命令。

<unique-label>（当工作流程运行有多个作业时，建议使用）

- 示例：job1
- 您定义的自定义标签，并且在同一工作流程中运行的其他作业都不使用该标签。为每个作业分配一个唯一的标签可确保仅将每个作业 GitHub 路由到为其创建的运行器。如果没有它，当同一工作流程运

行中的作业具有不同数量的标签覆盖时，为标签较多的作业创建的运行器可以选择标签较少的作业，从而使第二个作业没有运行器。有关更多信息，请参阅 排查 webhook 的问题。

单标签覆盖

CodeBuild 允许您使用以下方法在单个标签中提供多个覆盖：

> **Note**
>
> 由于所有重写都组合成一个标签，因此每个作业的运行器只注册一个标签。这意味着，即使每个作业都没有唯一的标签，这种格式也排查 webhook 的问题不会出现中所述的多任务运行器匹配问题。如果您更喜欢这种行为，并且您的作业只需要它支持的替换，请使用单标签格式。当您需要它不支持的替代项时，请勿使用它。例如，只能使用`image:custom-<environment-type>-<custom-image-identifier>`标签格式指定自定义映像（包括私有注册表中的图像）。对于这些替代，请使用本主题前面所述的分隔标签格式。

- 要替换 Amazon EC2/Lambda 计算版本的环境设置，请使用以下语法：

```
runs-on: codebuild-<project-name>-${{ github.run_id }}-
${{ github.run_attempt }}-<environment-type>-<image-identifier>-<instance-size>
```

- 要覆盖 Amazon EC2 计算构建的实例集设置，请使用以下语法：

```
runs-on: codebuild-<project-name>-${{ github.run_id }}-${{ github.run_attempt }}-
fleet-<fleet-name>
```

- 要同时覆盖构建所用的实例集和映像，请使用以下语法：

```
runs-on: codebuild-<project-name>-${{ github.run_id }}-
${{ github.run_attempt }}-image-<image-version>-fleet-<fleet-name>
```

- 要在构建期间运行 buildspec 命令，可以将 `-with-buildspec` 作为后缀添加到标签中：

```
runs-on: codebuild-<project-name>-${{ github.run_id }}-
${{ github.run_attempt }}-<image>-<image-version>-<instance-size>-with-buildspec
```

- 或者，您也可以提供实例大小覆盖，而不覆盖映像。对于 Amazon EC2 构建，您可以排除环境类型和映像标识符。对于 Lambda 构建，您可以排除映像标识符。

计算 CodeBuild-hosted GitHub 动作运行器支持的图像

在中配置的标签中教程：配置 CodeBuild-hosted GitHub 操作运行器，您可以使用前三列中的值来覆盖您的 Amazon EC2 环境设置。 CodeBuild 提供了以下 Amazon EC2 计算映像。有关

环境类型	映像标识符	实例大小	平台	已解析映像	定义
linux	4.0	small medium large xlarge	Amazon Linux 2	aws/codebuild/amazonlinux-x86_64-standard:4.0	al/standard/4.0
linux	5.0	2xlarge gpu_small gpu_large	Amazon Linux 2023	aws/codebuild/amazonlinux-x86_64-standard:5.0	al/standard/5.0
linux-ec2	latest	small medium large	Amazon Linux 2023	aws/codebuild/ami/amazonlinux-x86_64-base:latest	无
arm	2.0	small medium large xlarge	Amazon Linux 2	aws/codebuild/amazonlinux-aarch64-standard:2.0	al/aarch64/standard/2.0
arm	3.0	2xlarge	Amazon Linux 2023	aws/codebuild/amazonlinux-a	al/aarch64/standard/3.0

环境类型	映像标识符	实例大小	平台	已解析映像	定义
				arch64-standard:3.0	
arm-ec2	latest	small medium large	Amazon Linux 2023	aws/codebuild/ami/amazonlinux-arm-base:latest	无
ubuntu	5.0	small medium large xlarge 2xlarge gpu_small gpu_large	Ubuntu 20.04	aws/codebuild/standard:5.0	ubuntu/standard/5.0
ubuntu	6.0		Ubuntu 22.04	aws/codebuild/standard:6.0	ubuntu/standard/6.0
ubuntu	7.0		Ubuntu 22.04	aws/codebuild/standard:7.0	ubuntu/standard/7.0
windows	1.0	medium large	Windows Server Core 2019	aws/codebuild/windows-base:2019-1.0	N/A
			Windows Server Core 2022	aws/codebuild/windows-base:2022-1.0	N/A

环境类型	映像标识符	实例大小	平台	已解析映像	定义
windows	2.0		Windows Server Core 2019	aws/codebuild/windows-base:2019-2.0	N/A
windows	3.0		Windows Server Core 2019	aws/codebuild/windows-base:2019-3.0	N/A
windows-ec2	2022	medium large	Windows Server Core 2022	aws/codebuild/ami/windows-base:2022	无

此外，您还可以使用以下值来覆盖 Lambda 环境设置。有关 CodeBuild Lambda 计算的更多信息，请参阅。[在 build 上运行 AWS Lambda 计算](#) CodeBuild 支持以下 Lambda 计算映像：

环境类型	映像标识符	实例大小			
linux-lambda	dotnet6	1GB			
arm-lambda	go1.21	2GB			
	corretto11	4GB			
		8GB			
	corretto17	10GB			
	corretto21				
	nodejs18				
	nodejs20				

环境类型	映像标识符	实例大小			
	python3.11 python3.12 ruby3.2				

有关更多信息，请参阅构建环境计算模式和类型和Docker 镜像由 CodeBuild。

Self-managed GitLab 跑步者进来 AWS CodeBuild

GitLab 提供了两种执行模式来运行 CI/CD 管道中的 GitLab 作业。一种模式是 GitLab-hosted 运行器，它由运行器管理 GitLab 并与之完全集成 GitLab。另一种模式是自我管理的运行器，它允许您使用自己的自定义环境来运行 GitLab CI/CD 管道中的作业。

配置 CodeBuild 项目以运行 GitLab CI/CD 管道作业的高级步骤如下：

1. 如果您还没有这样做，请连接 OAuth 应用程序以连接您的项目。 GitLab
2. 导航到 CodeBuild 控制台并使用 webhook 创建 CodeBuild 项目，然后设置 webhook 过滤器。
3. 更新您的 GitLab CI/CD 管道 YAML GitLab 以配置您的构建环境。

有关更详细的过程，请参阅教程：配置 CodeBuild-hosted GitLab 运行器。

此功能允许您的 GitLab CI/CD 管道任务与之进行原生集成 AWS，从而通过 IAM 和 Amazon VPC 等功能提供安全性和便利性。 AWS CloudTrail您可以访问最新的实例类型，包括 ARM-based实例。

主题

- 关于 CodeBuild-hosted GitLab 跑步者
- 教程：配置 CodeBuild-hosted GitLab 运行器
- 运行器支持的标签覆盖 CodeBuild-hosted GitLab
- 计算 CodeBuild-hosted GitLab 运行器支持的图像

关于 CodeBuild-hosted GitLab 跑步者

以下是有关 CodeBuild-hosted GitLab跑步者的一些常见问题。

CodeBuild-hosted GitLab 运行器支持哪些源类型？

CodeBuild-hosted GitLab `GITLAB`和`GITLAB_SELF_MANAGED`源类型支持运行器。

我应该何时在标签中包括映像和实例覆盖？

您可以在标签中包含图像和实例覆盖，以便为每个 GitLab CI/CD 管道任务指定不同的构建环境。无需创建多个 CodeBuild 项目或 webhook 即可完成此操作。

我能用吗 CloudFormation 为了这个功能？

是的，您可以在 CloudFormation 模板中包含一个筛选器组，用于在项目 webhook 中指定 GitLab 工作流程作业事件过滤器。

```
Triggers:
  Webhook: true
  FilterGroups:
    - - Type: EVENT
        Pattern: WORKFLOW_JOB_QUEUED
```

有关更多信息，请参阅 过滤 GitLab webhook 事件 (CloudFormation)。

如果您在 CloudFormation 模板中设置项目凭证时需要帮助，请参阅《AWS CloudFormation 用户指南》SourceCredential中的AWS::CodeBuild:: 以了解更多信息。

使用此特征时如何屏蔽密钥？

默认情况下，日志中打印的机密不会被屏蔽。如果你想掩盖你的秘密，你可以通过更新你的 CI/CD 环境变量设置来实现：

掩盖秘密 GitLab

1. 在 GitLab “设置” 中，选择CI/CD。
2. 在变量中，为要屏蔽的密钥选择编辑。
3. 在可见性中，选择屏蔽变量，然后选择更新变量来保存更改。

我能否在一个群组中接收来自多个项目的 GitLab webhook 事件？

CodeBuild 支持群组 webhook，用于接收来自指定 GitLab 群组的事件。有关更多信息，请参阅 GitLab 群组 webhook。

我能否在 Docker 执行器中为自行管理的运行器执行作业？ 例如，我想在特定映像上运行管线作业，以便在单独和隔离的容器中维护相同的构建环境。

您可以通过 CodeBuild 使用自定义映像创建项目或覆盖文件中的镜像来运行带有特定映像的 GitLab 自管理运行.gitlab-ci.yml器。

自我管理的跑步者与哪个执行者一起 CodeBuild 运行？

中的自管理运行器与 shell 执行器一起 CodeBuild 运行，其中构建与在 docker 容 GitLab 器内运行的运行器一起在本地运行。

我能否在使用自行管理的运行器时提供 buildspec 命令？

是的，可以将 buildspec 命令与自行管理的运行器一起添加。你可以在 GitLab 仓库中提供 buildspec.yml 文件，然后在作业的 “buildspec-override:true标签” 部分使用标签。有关更多信息，请参阅 buildspec 文件名称和存储位置。

哪些地区支持使用 CodeBuild-hosted GitLab 跑步器？

CodeBuild-hosted GitLab 所有 CodeBuild 地区都支持跑步器。有关 AWS 区域 何处 CodeBuild 可用的更多信息，请参阅按地区划分的AWS 服务。

哪些平台支持使用 CodeBuild-hosted GitLab 运行器？

CodeBuild-hosted GitLab Amazon EC2 和AWS Lambda计算均支持运行器。您可以使用以下平台：Amazon Linux 2、Amazon Linux 2023、Ubuntu 和 Windows Server Core 2019。有关更多信息，请参阅EC2 计算映像和Lambda 计算映像。

教程：配置 CodeBuild-hosted GitLab 运行器

本教程向您展示如何配置 CodeBuild 项目以运行 GitLab CI/CD 管道作业。有关使用 GitLab 或使用 GitLab 自助管理的更多信息 CodeBuild，请参阅Self-managed GitLab 跑步者进来 AWS CodeBuild。

要完成本教程，您首先必须：

- 使用与 OAuth 应用程序连接。 AWS CodeConnections请注意，在连接 OAuth 应用程序时，必须使用 CodeBuild 控制台进行连接。有关更多说明，请参阅 GitLab 进入 CodeBuild。

- Connect CodeBuild 到您的 GitLab 账户。为此，您可以在控制台中添加 GitLab 为源提供商。有关说明，请参阅GitLab 进入 CodeBuild。

> **Note**
>
> 只有当你的账户还没有连接时， GitLab 才需要这样做。
> 使用此功能， CodeBuild 需要其他权限，例如`create_runner`来`manage_runner`自 GitLab OAuth 应用程序的权限。如果特定 GitLab 账户已 AWS CodeConnections 有权限，则它不会自动请求权限更新。为此，您可以访问 AWS CodeConnections 控制台并创建与同一 GitLab 账户的虚拟连接，以触发重新授权以获得额外权限。完成此步骤后，所有现有连接都可以使用运行器功能。完成后，您可以删除虚拟连接。

第 1 步：使用 webhook 创建 CodeBuild 项目

在此步骤中，您将创建一个带有 webhook 的 CodeBuild 项目，并在 GitLab 控制台中对其进行审核。

使用 webhook 创建 CodeBuild 项目

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 创建构建项目。有关信息，请参阅创建构建项目（控制台）和运行构建（控制台）。

   在项目类型中，选择运行程序项目。

   - 在运行程序中：
     - 对于跑步者提供商，请选择GitLab。
     - 对于凭证，选择以下选项之一：
       - 选择默认来源凭证。默认连接将默认 GitLab 连接应用于所有项目。
       - 选择自定义来源凭证。自定义连接应用的自定义 GitLab 连接会覆盖您账户的默认设置。

     > **Note**
     >
     > 如果您尚未创建与提供商的连接，则必须创建一个新的 GitLab 连接。有关说明，请参阅Connect t CodeBuild o GitLab。

     - 对于运行程序位置，请选择存储库。
     - 对于 Reposit ory， GitLab 通过使用命名空间指定项目路径来选择项目名称。

- 在环境中：
  - 选择支持的环境映像和计算。请注意，您可以选择通过在 GitLab CI/CD 管道 YAML 中使用标签来覆盖图像和实例设置。有关更多信息，请参阅 步骤 2：在存储库中创建 .gitlab-ci.yml 文件。
- 在 Buildspec (构建规范) 中：
  - 请注意，除非将 `buildspec-override:true` 作为标签添加，否则系统会忽略 buildspec。取而代之的是， CodeBuild 将覆盖它以使用设置自我管理的运行器的命令。
-

3. 继续使用默认值，然后选择创建构建项目。
4. 打开 GitLab 控制台，验证是否已创建一个 webhook 并已启用 webhook 来传送工作流作业事件。`https://gitlab.com/`*`user-name`*`/`*`repository-name`*`/-/hooks`

步骤 2：在存储库中创建 .gitlab-ci.yml 文件

在此步骤中，您将在中创建一个`.gitlab-ci.yml`文件GitLab来配置您的构建环境并在中 CodeBuild 使用 GitLab 自我管理的运行器。有关更多信息，请参阅使用自行管理的运行器。

更新你的 GitLab CI/CD 管道 YAML

导航到 `https://gitlab.com/`*`user-name`*`/`*`project-name`*`/-/tree/`*`branch-name`* 并在您的存储库中创建 `.gitlab-ci.yml` 文件。您可以执行下列操作之一来配置构建环境：

- 您可以指定 CodeBuild 项目名称，在这种情况下，构建版本将使用您现有的项目配置来计算计算、映像、映像版本和实例大小。需要项目名称才能将 GitLab 作业的 AWS相关设置链接到特定 CodeBuild项目。通过在 YAML 中包含项目名称 CodeBuild ，可以调用具有正确项目设置的作业。

```
tags:
    - codebuild-<codebuild-project-name>-$CI_PROJECT_ID-$CI_PIPELINE_IID-$CI_JOB_NAME
```

  `$CI_PROJECT_ID-$CI_PIPELINE_IID-$CI_JOB_NAME`需要将生成映射到特定的管道作业运行，并在取消管道运行时停止构建。

> **Note**
>
> 请确保您的名称与您在中创建的项目的名称*<project-name>*相匹配 CodeBuild。如果不匹配，则 CodeBuild 不会处理 webhook，并且 GitLab CI/CD 管道可能会挂起。

以下是 GitLab CI/CD 管道 YAML 的示例：

```
workflow:
  name: HelloWorld
stages:          # List of stages for jobs, and their order of execution
  - build

build-job:       # This job runs in the build stage, which runs first.
  stage: build
  script:
    - echo "Hello World!"
  tags:
    - codebuild-myProject-$CI_PROJECT_ID-$CI_PIPELINE_IID-$CI_JOB_NAME
```

- 您也可以在标签中覆盖映像和计算类型。有关精选映像的列表，请参阅计算 CodeBuild-hosted GitLab 运行器支持的图像。有关使用自定义映像，请参阅运行器支持的标签覆盖 CodeBuild-hosted GitLab。标签中的计算类型和映像将覆盖项目上的环境设置。要覆盖 Amazon EC2 计算构建的环境设置，请使用以下语法：

```
tags:
    - codebuild-<codebuild-project-name>-$CI_PROJECT_ID-$CI_PIPELINE_IID-$CI_JOB_NAME
    - image:<environment-type>-<image-identifier>
    - instance-size:<instance-size>
```

以下是 GitLab CI/CD 管道 YAML 的示例：

```
stages:
  - build

build-job:
  stage: build
  script:
    - echo "Hello World!"
  tags:
```

```
    - codebuild-myProject-$CI_PROJECT_ID-$CI_PIPELINE_IID-$CI_JOB_NAME
    - image:arm-3.0
    - instance-size:small
```

- 您可以在标签中覆盖构建所用的实例集。这将覆盖在您的项目中配置的实例集设置，以便使用指定的实例集。有关更多信息，请参阅 在预留容量实例集上运行构建。要覆盖 Amazon EC2 计算构建的实例集设置，请使用以下语法：

  ```
  tags:
      - codebuild-<codebuild-project-name>-$CI_PROJECT_ID-$CI_PIPELINE_IID-$CI_JOB_NAME
      - fleet:<fleet-name>
  ```

  要同时覆盖构建所用的实例集和映像，请使用以下语法：

  ```
  tags:
      - codebuild-<codebuild-project-name>-$CI_PROJECT_ID-$CI_PIPELINE_IID-$CI_JOB_NAME
      - fleet:<fleet-name>
      - image:<environment-type>-<image-identifier>
  ```

  以下是 GitLab CI/CD 管道 YAML 的示例：

  ```
  stages:
    - build

  build-job:
    stage: build
    script:
      - echo "Hello World!"
    tags:
      - codebuild-myProject-$CI_PROJECT_ID-$CI_PIPELINE_IID-$CI_JOB_NAME
      - fleet:myFleet
      - image:arm-3.0
  ```

- 要在自定义映像上运行 GitLab CI/CD 管道作业，您可以在 CodeBuild 项目中配置自定义映像，避免提供图像覆盖标签。 CodeBuild 如果未提供图像覆盖标签，则将使用项目中配置的图像。

在您向提交更改后`.gitlab-ci.yml`，将触发 GitLab 管道，管道`build-job`将发送一个 webhook 通知，启动您的构建。 CodeBuild

在 INSTALL、PRE_BUILD 和 POST_BUILD 阶段运行 buildspec 命令

默认情况下，在运行自 GitLab 管理版本时 CodeBuild 会忽略任何 buildspec 命令。要在构建期间运行 buildspec 命令，可以将 `buildspec-override:true` 作为后缀添加到 `tags`：

```
tags:
    - codebuild-<codebuild-project-name>-$CI_PROJECT_ID-$CI_PIPELINE_IID-$CI_JOB_NAME
    - buildspec-override:true
```

通过使用此命令， CodeBuild 将在容器的主源文件夹`gitlab-runner`中创建一个名为的文件夹。当 GitLab 运行器在该`BUILD`阶段启动时，运行器将在`gitlab-runner`目录中运行。

在自管理 GitLab 版本中使用 buildspec 覆盖时有几个限制：

- CodeBuild 在此`BUILD`阶段不会运行 buildspec 命令，因为自我管理的运行器将在该`BUILD`阶段运行。
- CodeBuild 在此`DOWNLOAD_SOURCE`阶段不会下载任何主要或次要来源。如果您配置了 buildspec 文件，则只会从项目的主源下载该文件。
- 如果构建命令在`PRE_BUILD`或`INSTALL`阶段失败， CodeBuild 则无法启动自管理的运行器，并且需要手动取消 GitLab CI/CD 管道作业。
- CodeBuild 在该阶段获取跑步者令牌，该`DOWNLOAD_SOURCE`阶段的到期时间为一小时。如果您的`PRE_BUILD`或`INSTALL`阶段超过一小时，则运行器令牌可能会在 GitLab 自我管理的运行器启动之前过期。

步骤 3：检查您的结果

每当 GitLab CI/CD 管道运行时， CodeBuild 都会通过 webhook 接收 CI/CD 管道作业事件。对于 CI/CD 管道中的每项作业，都 CodeBuild 启动构建以运行临时 GitLab 运行器。运行器负责执行单个 CI/CD 管道作业。作业完成后，运行器和关联的构建过程会立即终止。

要查看您的 CI/CD 管道作业日志，请导航到中的存储库 GitLab，选择 B uild、J ob s，然后选择要查看其日志的特定作业。

当作业等待中自我管理的运行者接管时，您可以在日志中 CodeBuild查看请求的标签。

筛选 GitLab webhook 事件 (CloudFormation)

CloudFormation 模板的以下 YAML-formatted 部分创建了一个筛选器组，当计算结果为 true 时，该筛选器组会触发构建。以下筛选器组指定 GitLab CI/CD 管道作业请求，其 CI/CD 管道名称与正则表达式匹配`\[CI-CodeBuild\]`。

```
CodeBuildProject:
  Type: AWS::CodeBuild::Project
  Properties:
    Name: MyProject
    ServiceRole: service-role
    Artifacts:
      Type: NO_ARTIFACTS
    Environment:
      Type: LINUX_CONTAINER
      ComputeType: BUILD_GENERAL1_SMALL
      Image: aws/codebuild/standard:5.0
    Source:
      Type: GITLAB
      Location: CODEBUILD_DEFAULT_WEBHOOK_SOURCE_LOCATION
    Triggers:
      Webhook: true
      ScopeConfiguration:
        Name: group-name
        Scope: GITLAB_GROUP
      FilterGroups:
        - - Type: EVENT
            Pattern: WORKFLOW_JOB_QUEUED
          - Type: WORKFLOW_NAME
            Pattern: \[CI-CodeBuild\]
```

运行器支持的标签覆盖 CodeBuild-hosted GitLab

在您的 GitLab CI/CD 管道 YAML 中，您可以提供各种标签覆盖来修改您的自管理运行器构建。任何未被识别的版本都 CodeBuild 将被忽略，但不会使您的 webhook 请求失败。例如，以下 YAML 包括映像、实例大小、实例集和 buildspec 的覆盖：

```
workflow:
  name: HelloWorld
stages:
  - build

build-job:
  stage: build
  script:
    - echo "Hello World!"
  tags:
    - codebuild-myProject-$CI_PROJECT_ID-$CI_PIPELINE_IID-$CI_JOB_NAME
```

```
    - image:arm-3.0
    - instance-size:small
    - fleet:myFleet
    - buildspec-override:true
```

codebuild-*<project-name>*-$CI_PROJECT_ID-$CI_PIPELINE_IID-$CI_JOB_NAME（必需）

- 示例：codebuild-myProject-$CI_PROJECT_ID-$CI_PIPELINE_IID-$CI_JOB_NAME
- 对于所有 GitLab CI/CD 管道 YAML 来说都是必需的。 *<project name>*应等于为其配置自管理运行器 webhook 的项目的名称。

image:*<environment-type>*-*<image-identifier>*

- 示例：image:arm-3.0
- 覆盖在启动自行管理运行器构建时使用的映像和环境类型。要了解支持的值，请参阅计算 CodeBuild-hosted GitLab 运行器支持的图像。
  - 要覆盖与自定义映像结合使用的映像和环境类型，请使用 image:custom-*<environment-type>*-*<custom-image-identifier>*
  - 示例：image:custom-arm-public.ecr.aws/codebuild/amazonlinux-aarch64-standard:3.0

> **Note**
>
> 如果自定义映像位于私有注册表中，请参阅为自托管运行程序配置私有注册表凭证。

instance-size:*<instance-size>*

- 示例：instance-size:small
- 覆盖在启动自行管理运行器构建时使用的实例类型。要了解支持的值，请参阅计算 CodeBuild-hosted GitLab 运行器支持的图像。

fleet:*<fleet-name>*

- 示例：fleet:myFleet

- 覆盖在您的项目中配置的实例集设置，以便使用指定的实例集。有关更多信息，请参阅 在预留容量实例集上运行构建。

`buildspec-override:<boolean>`

- 示例：`buildspec-override:true`
- 如果设置为 `true`，则允许构建以在 INSTALL、PRE_BUILD 和 POST_BUILD 阶段运行 buildspec 命令。

计算 CodeBuild-hosted GitLab 运行器支持的图像

在中配置的标签中教程：配置 CodeBuild-hosted GitLab 运行器，您可以使用前三列中的值来覆盖您的 Amazon EC2 环境设置。 CodeBuild 提供了以下 Amazon EC2 计算映像。有关

环境类型	映像标识符	实例大小	平台	Image	定义

`linux`	`4.0`	`small`
		`medium`
		`large`
		`xlarge`
			Amazon Linux 2	`aws/codebuild/amazonlinux-x86_64-standard:4.0`	al/standard/4.0

`linux`	`5.0`	`2xlarge`
		`gpu_small`
		`gpu_large`
			Amazon Linux 2023	`aws/codebuild/amazonlinux-x86_64-standard:5.0`	al/standard/5.0

`linux-ec2`	`latest`	`small`
		`medium`
		`large`
			Amazon Linux 2023	`aws/codebuild/ami/amazonlinux-x86_64-base:latest`	无

环境类型	映像标识符	实例大小	平台	Image	定义
arm	2.0	small medium large xlarge	Amazon Linux 2	aws/codebuild/amazonlinux-aarch64-standard:2.0	al/aarch64/standard/2.0
arm	3.0	2xlarge	Amazon Linux 2023	aws/codebuild/amazonlinux-aarch64-standard:3.0	al/aarch64/standard/3.0
arm-ec2	latest	small medium large	Amazon Linux 2023	aws/codebuild/ami/amazonlinux-arm-base:latest	无
ubuntu	5.0	small medium large xlarge 2xlarge gpu_small gpu_large	Ubuntu 20.04	aws/codebuild/standard:5.0	ubuntu/standard/5.0
ubuntu	6.0		Ubuntu 22.04	aws/codebuild/standard:6.0	ubuntu/standard/6.0
ubuntu	7.0		Ubuntu 22.04	aws/codebuild/standard:7.0	ubuntu/standard/7.0

环境类型	映像标识符	实例大小	平台	Image	定义
windows	1.0	medium large	Windows Server Core 2019	aws/codebuild/windows-base:2019-1.0	N/A
			Windows Server Core 2022	aws/codebuild/windows-base:2022-1.0	N/A
windows	2.0		Windows Server Core 2019	aws/codebuild/windows-base:2019-2.0	N/A
windows	3.0		Windows Server Core 2019	aws/codebuild/windows-base:2019-3.0	N/A
windows-ec2	2022	medium large	Windows Server Core 2022	aws/codebuild/ami/windows-base:2022	无

此外，您还可以使用以下值来覆盖 Lambda 环境设置。有关 CodeBuild Lambda 计算的更多信息，请参阅。[在 build 上运行 AWS Lambda 计算](#) CodeBuild 支持以下 Lambda 计算映像：

环境类型	运行时版本	实例大小			
linux-lambda	dotnet6 go1.21	1GB 2GB			
arm-lambda	corretto11	4GB			

环境类型	运行时版本	实例大小			
	corretto17 corretto21 nodejs18 nodejs20 python3.11 python3.12 ruby3.2	8GB 10GB			

有关更多信息，请参阅构建环境计算模式和类型和Docker 镜像由 CodeBuild。

Self-managed Buildkite runner AWS CodeBuild

你可以将项目配置为在 CodeBuild 容器中设置自托管的 Buildkite 运行器来处理你的 Buildkite 作业。这可以通过使用您的 CodeBuild项目设置 webhook，然后更新 Buildkite 管道 YAML 步骤以使用托管在计算机上的自托管运行器来完成。 CodeBuild

配置 CodeBuild 项目以运行 Buildkite 作业的高级步骤如下：

- 导航到 CodeBuild 控制台并使用 Buildkite 运行器项目运行器类型配置创建项目 CodeBuild
- 向 Buildkite 组织添加 `job.scheduled` webhook。
- 在 Buildkite 中更新 Buildkite 管道 YAML 步骤，以配置构建环境。

有关更详细的过程，请参阅教程：配置 CodeBuild-hosted Buildkite 运行器。此功能允许您的 Buildkite 任务与之进行原生集成 AWS，从而通过 IAM、、和 Amazon VPC 等功能提供安全性和便利性。 AWS Secrets Manager AWS CloudTrail您可以访问最新的实例类型，包括 ARM-based 实例。

关于 CodeBuild-hosted Buildkite 跑步者

以下是有关 CodeBuild-hosted Buildkite 运行器的一些常见问题。

我应该何时在标签中包括映像和实例覆盖？

您可以在标签中包括映像和实例覆盖，以便为每个 Buildkite 作业指定不同的构建环境。无需创建多个 CodeBuild 项目或 webhook 即可完成此操作。例如，当您需要为 Buildkite 作业使用矩阵时，这很有用。

```
agents:
  queue: "myQueue"
steps:
  - command: "echo \"Hello World\""
    agents:
      project: "codebuild-myProject"
      image: "{{matrix.os}}"
      instance-size: "{{matrix.size}}"
    matrix:
      setup:
        os:
          - "arm-3.0"
          - "al2-5.0"
        size:
          - "small"
          - "large"
```

能否在 Buildkite 中自动 CodeBuild 创建 web 挂钩？

目前，Buildkite 要求所有 webhook 都使用其控制台手动创建。您可以按照教程：配置 CodeBuild-hosted Buildkite 运行器中的教程在 Buildkite 控制台中手动创建 Buildkite webhook。

我能用吗 CloudFormation 创建 Buildkite 网络挂钩？

CloudFormation Buildkite 运行器 webhook 目前不支持，因为 Buildkite 要求使用其控制台手动创建 webhook。

哪些地区支持使用 CodeBuild-hosted Buildkite 运行器？

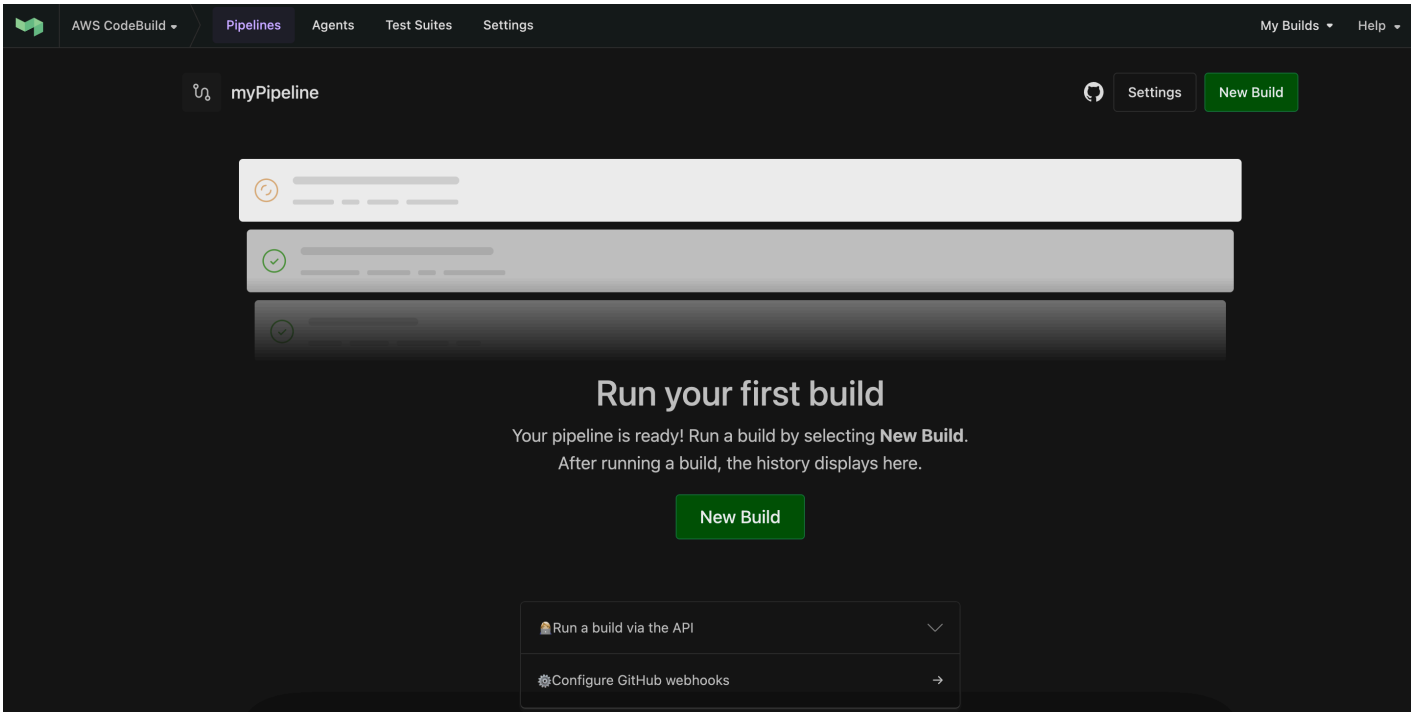
CodeBuild-hosted 所有 CodeBuild 区域都支持 Buildkite 运行器。有关可用 AWS 区域的更多信息，请参阅按地区划分的AWS 服务。 CodeBuild

教程：配置 CodeBuild-hosted Buildkite 运行器

本教程向您展示如何配置 CodeBuild 项目以运行 Buildkite 作业。有关将 Buildkite 与配合 CodeBuild 使用的更多信息，请参阅。Self-managed Buildkite runner AWS CodeBuild

要完成本教程，您首先必须：

- 可以访问 Buildkite 组织。有关设置 Buildkite 账户和组织的更多信息，您可以关注此 Getting Started Tutorial。
- 创建配置为使用自托管运行程序的 Buildkite 管道、集群和队列。有关设置这些资源的更多信息，可以参考 Buildkite Pipeline Setup Tutorial。

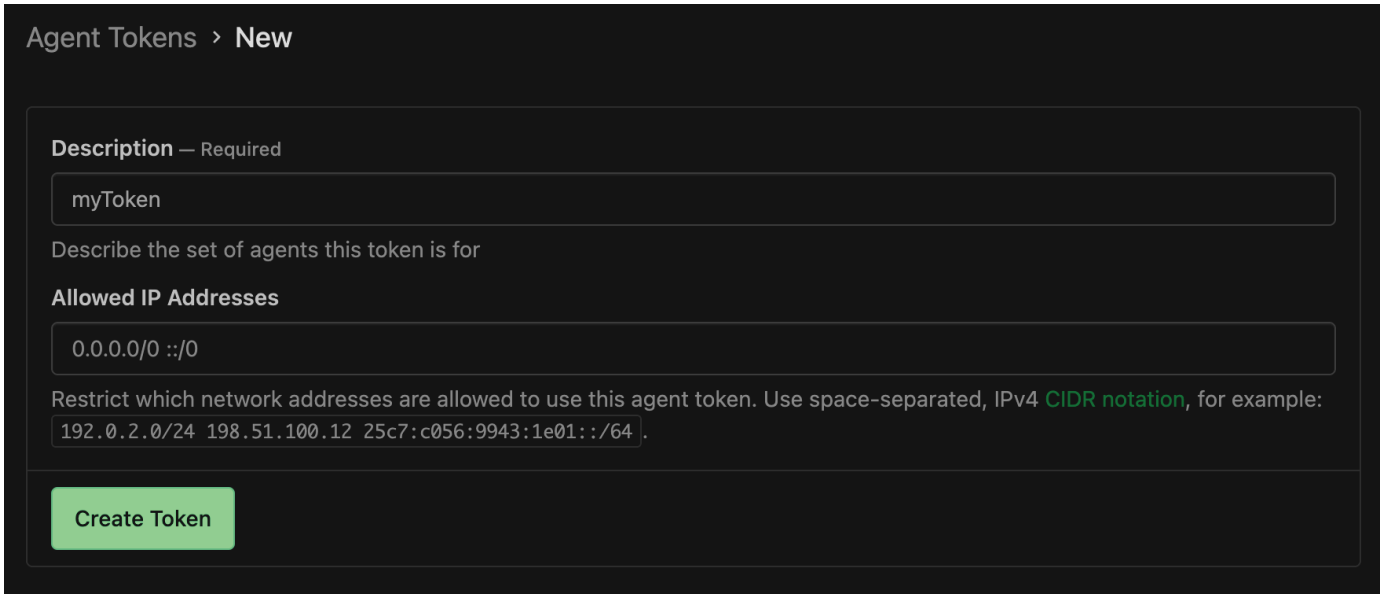


步骤 1：生成 Buildkite 代理令牌

在此步骤中，您将在 Buildkite 中生成代理令牌，该令牌将用于对 CodeBuild 自托管的运行器进行身份验证。有关此资源的更多信息，请参阅 Buildkite Agent Tokens。

生成 Buildkite 代理令牌

1. 在 Buildkite 集群中，选择代理令牌，然后选择新建令牌。
2. 向令牌添加描述，然后单击创建令牌。
3. 保存代理令牌值，以便稍后在 CodeBuild 项目设置过程中使用该值。



第 2 步：使用 webhook 创建 CodeBuild 项目

使用 webhook 创建 CodeBuild 项目

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 创建自托管构建项目。有关更多信息，请参阅 创建构建项目（控制台） 和 运行构建（控制台）。

   - 在项目配置中，选择运行程序项目。在运行程序中：
     - 对于运行程序提供商，选择 Buildkite。
     - 对于 Buildkite 代理令牌，选择使用创建密钥页面创建新的代理令牌。系统将提示您在中 AWS Secrets Manager 创建一个新的密钥，其密钥值等于您在上面生成的 Buildkite 代理令牌。
     - （可选）如果您想为作业使用 CodeBuild 托管凭据，请在 Buildkite 源凭证选项下选择作业的源存储库提供商，并确认已为您的账户配置了凭据。此外，请验证 Buildkite 管道是否采用使用 HTTPS 签出。

   > **Note**
   >
   > Buildkite 需要编译环境中的源证书才能为您的任务提取源代码。有关可用的源凭证选项，请参阅针对私有存储库对 Buildkite 进行身份验证。

   - （可选）在环境中：
     - 选择支持的环境映像和计算。

请注意，您可以选择在 Buildkite YAML 步骤中使用标签来覆盖映像和实例设置。有关更多信息，请参阅 步骤 4：更新 Buildkite 管道步骤。

- （可选）在 Buildspec 中：
  - 除非将 `buildspec-override: "true"` 添加为标签，否则系统默认情况下会忽略 buildspec。相反， CodeBuild 将覆盖它以使用设置自托管运行器的命令。

> **Note**
>
> CodeBuild 不支持 Buildkite 自托管运行器版本的 buildspec 文件。对于内联构建规范，如果您配置了托管源凭据，则需要在构建规范中启用 git-credential-hel per CodeBuild

3. 继续使用默认值，然后选择创建构建项目。
4. 保存创建 webhook 弹出窗口中的有效载荷 URL 和密钥值。要么按照弹出窗口中的说明创建新的 Buildkite 组织 webhook，要么继续下一节。

第 3 步：在 Buildkite 中创建一个 CodeBuild webhook

在此步骤中，您将使用 CodeBuild webhook 中的有效负载 URL 和 S ecret 值在 Buildkite 中创建一个新的 webhook。此 webhook 将用于在有效的 Buildkite 作业启动 CodeBuild 时触发构建。

在 Buildkite 中创建新的 webhook

1. 打开你的 Buildkite 组织的 “设置” 页面。
2. 在集成下，选择通知服务。
3. 选择 Webhook 框旁边的添加。在添加 Webhook 通知页面中，使用以下配置：
   a. 在 Webhook URL 下，添加保存的有效载荷 URL 值。
   b. 在 “令牌” 下，确认已选中 “按原样 X-Buildkite-Token发送令牌”。将 webhook 密钥值添加到令牌字段。
   c. 在下方，确认选中了按原样 X-Buildkite-Token发送令牌。将 webhook 密钥值添加到令牌字段。
   d. 在事件下，选择 `job.scheduled` webhook 事件。
   e. （可选）在管道下，您可以选择仅触发特定管道的构建。
4. 选择添加 Webhook 通知。

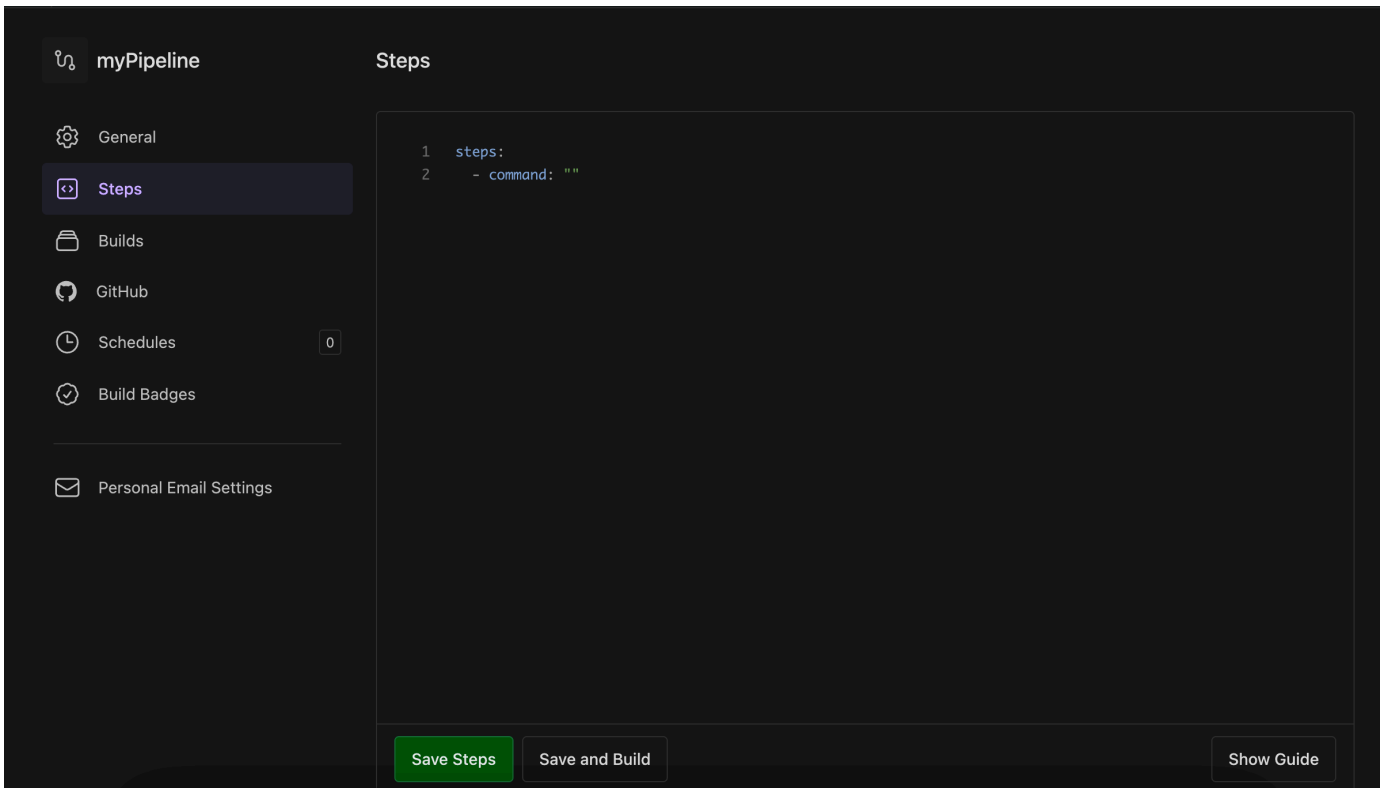
步骤 4：更新 Buildkite 管道步骤

在此步骤中，更新 Buildkite 管道的步骤，以添加必要的标签和可选的替代项。有关支持的标签覆盖的完整列表，请参阅B CodeBuild-hosted uildkite 运行器支持的标签覆盖。

更新管道步骤

1. 打开 Buildkite 工作流步骤页面，方法是选择你的 Buildkite 管道，选择 “设置”，然后选择 “步骤”。

   如果您尚未选择 “转换为 YAML 步骤”，请选择 “转换为 YAML 步骤”。



2. 您至少需要指定一个引用管道名称的 Buildkite 代理标签。 CodeBuild 需要项目名称才能将 Buildkite 作业的 AWS相关设置链接到特定 CodeBuild 项目。通过在 YAML 中包含项目名称 CodeBuild ，可以调用具有正确项目设置的作业。

```
agents:
  project: "codebuild-<project name>"
```

   以下是仅具有项目标注标签的 Buildkite 管道步骤的示例：

```
agents:
  project: "codebuild-myProject"
steps:
```

```
  - command: "echo \"Hello World\""
```

您也可以在标签中覆盖映像和计算类型。有关可用映像的列表，请参阅计算 CodeBuild-hosted Buildkite 运行器支持的图像。标签中的计算类型和映像将覆盖项目的环境设置。要替换 CodeBuild EC2 或 Lambda 计算版本的环境设置，请使用以下语法：

```
agents:
  project: "codebuild-<project name>"
  image: "<environment-type>-<image-identifier>"
  instance-size: "<instance-size>"
```

以下是具有映像和实例大小覆盖的 Buildkite 管道步骤的示例：

```
agents:
  project: "codebuild-myProject"
  image: "arm-3.0"
  instance-size: "small"
steps:
  - command: "echo \"Hello World\""
```

您可以在标签中覆盖构建所用的实例集。这将覆盖在您的项目中配置的实例集设置，以便使用指定的实例集。有关更多信息，请参阅在预留容量实例集上运行构建。

要覆盖 Amazon EC2 计算构建的实例集设置，请使用以下语法：

```
agents:
  project: "codebuild-<project name>"
  fleet: "<fleet-name>"
```

要同时覆盖构建所用的实例集和映像，请使用以下语法：

```
agents:
  project: "codebuild-<project name>"
  fleet: "<fleet-name>"
  image: "<environment-type>-<image-identifier>"
```

以下是具有实例集和映像覆盖的 Buildkite 管道步骤的示例：

```
agents:
```

```
  project: "codebuild-myProject"
  fleet: "myFleet"
  image: "arm-3.0"
steps:
  - command: "echo \"Hello World\""
```

3. 您可以选择在自托管 Buildkite 运行程序构建期间运行内联 buildspec 命令（有关更多详细信息，请参阅对于 INSTALL、PRE_BUILD 和 POST_BUILD 阶段运行 buildspec 命令）。要指定该 CodeBuild 版本应在 Buildkite 自托管运行器构建期间运行 buildspec 命令，请使用以下语法：

```
agents:
  project: "codebuild-<project name>"
  buildspec-override: "true"
```

以下是具有 buildspec 覆盖的 Buildkite 管道的示例：

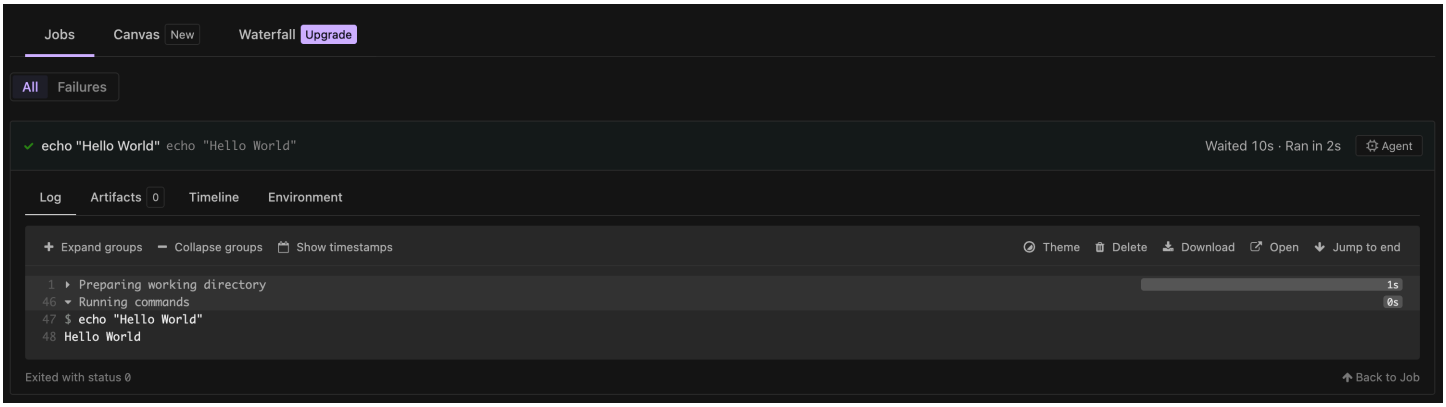
```
agents:
  project: "codebuild-myProject"
  buildspec-override: "true"
steps:
  - command: "echo \"Hello World\""
```

4. 或者，您可以在 CodeBuild 支持的标签之外提供标签。在覆盖构建的属性时会忽略这些标签，但不会导致 webhook 请求失败。例如，添加 `myLabel: "testLabel"` 作为标签不会阻止构建运行。

第 5 步：查看您的结果

每当你的管道中启动 Buildkite 作业时， CodeBuild 都会通过 Buildkite `job.scheduled` webhook 收到一个 webhook 事件。对于你的 Buildkite 版本中的每项任务，都 CodeBuild 将启动一个构建来运行一个临时的 Buildkite 运行器。该运行程序负责执行单个 Buildkite 作业。作业完成后，运行器和关联的构建过程会立即终止。

要查看工作流程作业日志，请导航到 Buildkite 管道并选择最新的构建（您可以通过选择新构建来触发新的构建。每项任务的关联 CodeBuild 构建启动并启动任务后，您应该可以在 Buildkite 控制台中看到该任务的日志



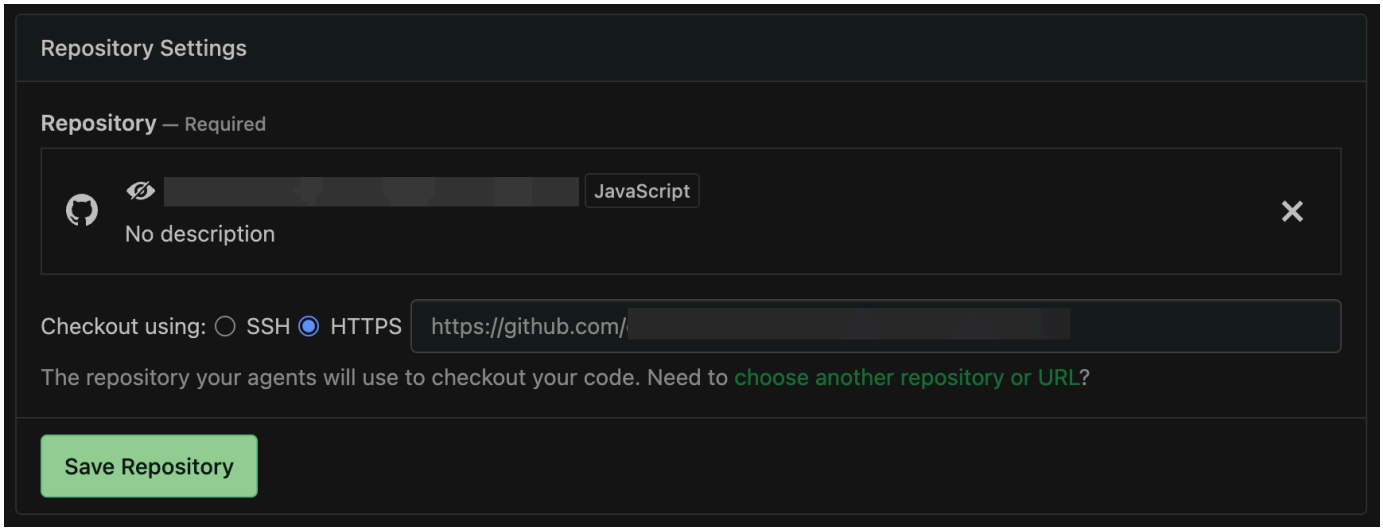
针对私有存储库对 Buildkite 进行身份验证

如果您在 Buildkite 管道中配置了私有存储库，则 Buildkite 需要构建环境中的额外权限才能提取存储库，因为 Buildkite 不会向自托管运行程序提供凭证以从私有存储库中进行提取。要针对外部私有源存储库对 Buildkite 自托管运行程序代理进行身份验证，可以使用以下选项之一。

要使用进行身份验证 CodeBuild

CodeBuild 为支持的源类型提供托管凭据处理。要使用 CodeBuild 源证书提取作业的源存储库，您可以使用以下步骤：

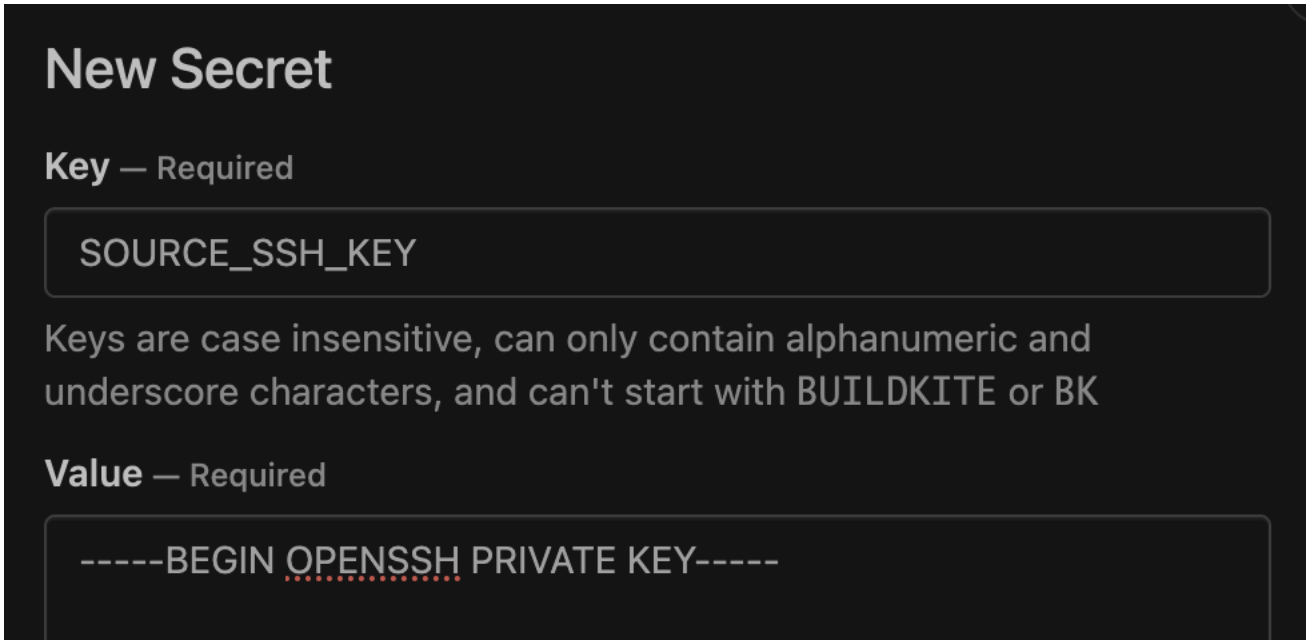
1. 在 CodeBuild 控制台中，导航到编辑项目或使用中的步骤创建新 CodeBuild 项目第 2 步：使用 webhook 创建 CodeBuild 项目。
2. 在 Buildkite 源证书选项下，选择作业的源存储库提供商。
    1. 如果您想使用账户级 CodeBuild 凭证，请验证其配置是否正确。此外，如果您的项目配置了内联 buildspec，请验证是否启用了 git-credential-helper。
    2. 如果您想使用项目级别的 CodeBuild 凭据，请选择 “仅为此项目使用覆盖凭据”，然后为您的项目设置凭据。
3. 在 Buildkite 管道设置中，导航到存储库设置。将源存储库签出设置设为使用 HTTPS 签出



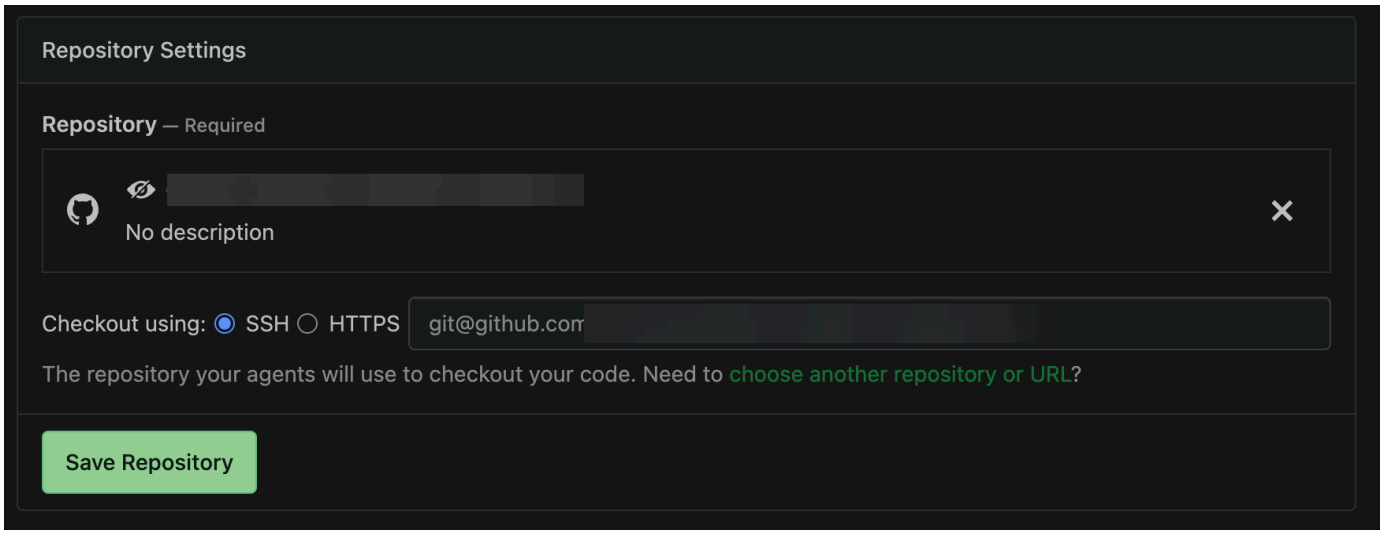
使用 Buildkite 密钥进行身份验证

Buildkite 维护一个 ssh-checkout 插件，该插件可用于使用 ssh 密钥针对外部源存储库对自托管运行程序进行身份验证。密钥值存储为 Buildkite 密钥，并在尝试提取私有存储库时由 Buildkite 自托管运行程序代理自动获取。要为 Buildkite 管道配置 ssh-checkout 插件，可以使用以下步骤：

1. 使用您的电子邮件地址生成私有和公有 ssh 密钥，例如 `ssh-keygen -t rsa -b 4096 -C "myEmail@address.com"`
2. 将公有密钥添加到私有源存储库。例如，您可以按照本指南为 GitHub 账户添加密钥。
3. 向 Buildkite 集群添加一个新的 SSH 密钥。在 Buildkite 集群中，选择密钥 → 新密钥。在密钥字段中为您的密钥添加名称，然后将私有 SSH 密钥添加到值字段中：



4. 在 Buildkite 管道中，导航到存储库设置并将签出设置为使用 SSH。



5. 更新管道 YAML 步骤以使用 `git-ssh-checkout` 插件。例如，以下管道 YAML 文件使用带有上述 Buildkite 秘密密钥的签出操作：

```
agents:
  project: "codebuild-myProject"
steps:
  - command: "npm run build"
    plugins:
      - git-ssh-checkout#v0.4.1:
          ssh-secret-key-name: 'SOURCE_SSH_KEY'
```

6. 在其中运行 Buildkite 自托管运行器作业时 CodeBuild，Buildkite 现在将在提取私有存储库时自动使用你配置的密钥值

运行程序配置选项

您可以在项目配置中指定以下环境变量，以修改自托管运行程序的设置配置：

- CODEBUILD_CONFIG_BUILDKITE_AGENT_TOKEN: CodeBuild 将从 AWS Secrets Manager 中获取配置为该环境变量值的秘密值，以便注册 Buildkite 自托管的运行器代理。此环境变量的类型必须为 SECRETS_MANAGER，其值应是您在 Secrets Manager 中的密钥名称。所有 Buildkite 运行程序项目都需要一个 Buildkite 代理令牌环境变量。
- CODEBUILD_CONFIG_BUILDKITE_CREDENTIAL_DISABLE: 默认情况下， CodeBuild会将账户或项目级别的源凭据加载到构建环境中，因为 Buildkite 代理使用这些凭据来提取作业的源存储库。要禁用此行为，您可以将此环境变量添加到项目中，值设置为 true，这将防止源凭证加载到构建环境中。

对于 INSTALL、PRE_BUILD 和 POST_BUILD 阶段运行 buildspec 命令

默认情况下，在运行自托管 Buildkite 运行器版本时会 CodeBuild 忽略所有 buildspec 命令。要在构建期间中运行 buildspec 命令，

```
buildspec-override: "true"
```

可作为后缀添加到标签：

```
agents:
  project: "codebuild-<project name>"
  buildspec-override: "true"
```

通过使用此命令， CodeBuild 将在容器的主源文件夹buildkite-runner中创建一个名为的文件夹。当 Buildkite 运行程序在 BUILD 阶段启动时，运行程序将在 buildkite-runner 目录中运行。

在自托管 Buildkite 构建中使用 buildspec 覆盖具有若干限制：

- Buildkite 代理要求生成环境中存在源凭据才能提取作业的源存储库。如果您使用 CodeBuild 源凭据进行身份验证，则需要在 buildspec git-credential-helper 中启用。例如，可以使用以下 buildspec 为 Buildkite 构建启用 git-credential-helper：

```
version: 0.2
env:
  git-credential-helper: yes
phases:
  pre_build:
    commands:
       - echo "Hello World"
```

- CodeBuild 在此BUILD阶段不会运行 buildspec 命令，因为自托管运行器将在该BUILD阶段运行。
- CodeBuild 不支持 Buildkite 运行器版本的 buildspec 文件。Buildkite 自托管运行器仅支持内联构建规范
- 如果构建命令在PRE_BUILD或INSTALL阶段失败， CodeBuild 则无法启动自托管运行器，并且需要手动取消 Buildkite 作业。

以编程方式设置 Buildkite 运行程序

要以编程方式配置 Buildkite 运行器项目，您需要配置以下资源：

以编程方式创建 Buildkite 运行程序

1. 创建 Buildkite 代理令牌并将该令牌以纯文本形式保存在 AWS Secrets Manager中。
2. 使用您的首选配置设置 CodeBuild 项目。您需要配置以下额外的属性：

    1. 一个环境值，其名称为 CODEBUILD_CONFIG_BUILDKITE_AGENT_TOKEN、类型为 SECRETS_MANAGER，而值等于与 Buildkite 集群关联的 Buildkite 代理令牌。
    2. 等于 NO_SOURCE 的源类型
    3. 以项目的服务角色访问步骤 1 中创建的密钥的权限

    例如，您可以通过 CLI 使用以下命令创建有效的 Buildkite 运行程序项目：

```
aws codebuild create-project \
--name buildkite-runner-project \
--source "{\"type\": \"NO_SOURCE\",\"buildspec\":\"\"}" \
--environment "{\"image\":\"aws/codebuild/amazonlinux-x86_64-standard:5.0\",
\"type\":\"LINUX_CONTAINER\",\"computeType\":\"BUILD_GENERAL1_MEDIUM\",
\"environmentVariables\":[{\"name\":\"CODEBUILD_CONFIG_BUILDKITE_AGENT_TOKEN\",
\"type\":\"SECRETS_MANAGER\",\"value\":\"<buildkite-secret-name>\"}]}" \
--artifacts "{\"type\": \"NO_ARTIFACTS\"}" \
```

```
--service-role <service-role>
```

3. 在步骤 2 中创建的项目上创建 Buildkite 运行程序 webhook。创建 webhook 时，您需要使用以下配置选项：

   1. build-type 应等于 RUNNER_BUILDKITE_BUILD
   2. 类型为 EVENT 且模式等于 WORKFLOW_JOB_QUEUED 的筛选条件

   例如，您可以通过 CLI 使用以下命令创建有效的 Buildkite 运行程序 webhook：

```
aws codebuild create-webhook \
--project-name buildkite-runner-project \
--filter-groups "[[{\"type\":\"EVENT\",\"pattern\":\"WORKFLOW_JOB_QUEUED\"}]]" \
--build-type RUNNER_BUILDKITE_BUILD
```

4. 保存由 `create-webhook` 调用返回的有效载荷 URL 和密钥值，然后使用凭证在 Buildkite 控制台中创建 webhook。您可以参考中的步骤 3：在 Buildkite 中创建 CodeBuild webhook，以教程：配置 CodeBuild-hosted Buildkite 运行器获取有关如何设置此资源的指南。

针对失败的构建或挂起的作业对 webhook 进行故障排除

问题:

您在教程：配置 CodeBuild-hosted Buildkite 运行器中设置的 webhook 无法正常工作，或工作流程作业在 Buildkite 中挂起。

可能的原因：

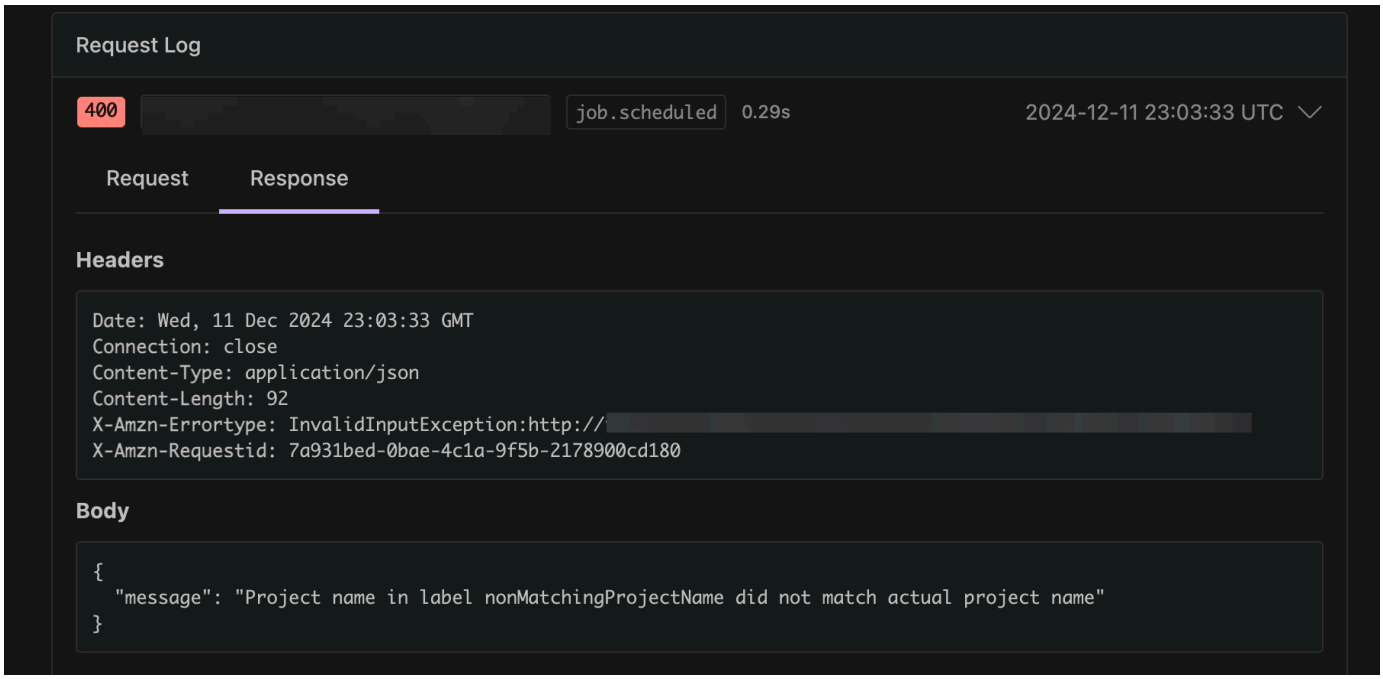
- 您的 job.scheduled 事件可能无法触发构建。检查响应日志以查看响应或错误消息。
- 在启动 Buildkite 自托管运行器代理来处理你的 CodeBuild 工作之前，你的构建失败了。

推荐的解决方案：

要调试失败的 Buildkite webhook 事件：

1. 在你的 Buildkite 组织设置中，导航到通知服务，选择你的 CodeBuild webhook，然后找到请求日志。
2. 查找与卡住的 Buildkite 作业关联的 `job.scheduled` webhook 事件。可以使用 webhook 有效载荷中的作业 ID 字段将 webhook 事件与 Buildkite 作业相关联。

3. 选择响应选项卡并检查响应正文。确认响应状态码为，200且响应正文不包含任何意外消息。



排查 webhook 权限问题

问题:

Buildkite 作业由于权限问题而无法签出作业的源存储库。

可能的原因：

- CodeBuild 没有足够的权限来查看任务的源存储库。
- 管道的存储库设置设置为使用 SSH 签出 CodeBuild托管凭据。

推荐的解决方案：

- 验证是否配置 CodeBuild 了足够的权限来查看作业的源存储库。此外，请验证您的 CodeBuild 项目的服务角色是否有足够的权限来访问配置的源权限选项。
- 如果您使用的是 CodeBuild 托管源存储库凭据，请确认您的 Buildkite 管道已配置为使用 HTTPS 签出。

B CodeBuild-hosted uildkite 运行器支持的标签覆盖

在 Buildkite 管道步骤代理标签标注中，您可以提供各种标注覆盖来修改自托管运行程序构建。任何未被识别的版本都 CodeBuild将被忽略，但不会使您的 webhook 请求失败。例如，以下工作流程 YAML 包括映像、实例大小、实例集和 buildspec 的覆盖：

```
agents:
  queue: "myQueue"
steps:
  - command: "echo \"Hello World\""
    agents:
      project: "codebuild-myProject"
      image: "{{matrix.os}}"
      instance-size: "{{matrix.size}}"
      buildspec-override: "true"
    matrix:
      setup:
        os:
          - "arm-3.0"
          - "al2-5.0"
        size:
          - "small"
          - "large"
```

`project:codebuild-<project-name>`（必需）

- 示例：`project: "codebuild-myProject"`
- 所有 Buildkite 工作流步骤配置都是必需的。*<project name>*应等于为其配置自托管运行器 webhook 的项目的名称。

`queue: "<queue-name>"`

- 示例：`queue: "<queue-name>"`
- 用于将 Buildkite 作业路由到特定队列。有关更多信息，请参阅 Buildkite Agent Queue Tag。

`image: "<environment-type>-<image-identifier>"`

- 示例：`image: "arm-3.0"`

- 覆盖在通过精选映像启动自托管运行程序构建时使用的映像和环境类型。要了解支持的值，请参阅[计算 CodeBuild-hosted Buildkite 运行器支持的图像]。
  1. 要覆盖与自定义映像结合使用的映像和环境类型，请使用 `image: "custom-<environment-type>-<custom-image-identifier>"`
  2. 示例：

```
image:
      "custom-arm-public.ecr.aws/codebuild/amazonlinux-aarch64-standard:3.0"
```

> **Note**
>
> 如果自定义映像位于私有注册表中，则必须在 CodeBuild 项目中配置相应的注册表凭据。

`instance-size: "<instance-size>"`

- 示例：`instance-size: "medium"`
- 覆盖在启动自托管运行器构建时使用的实例类型。要了解支持的值，请参阅[计算 CodeBuild-hosted Buildkite 运行器支持的图像]。

`fleet: "<fleet-name>"`

- 示例：`fleet: "myFleet"`
- 覆盖在您的项目中配置的实例集设置，以便使用指定的实例集。有关更多信息，请参阅[在预留容量实例集上运行构建]。

`buildspec-override: "<boolean>"`

- 示例：`buildspec-override: "true"`
- 如果设置为 `true`，则允许构建以在 `INSTALL`、`PRE_BUILD` 和 `POST_BUILD` 阶段运行 buildspec 命令。

计算 CodeBuild-hosted Buildkite 运行器支持的图像

在中配置的标签中[Self-managed Buildkite runner AWS CodeBuild]，您可以使用前三列中的值来覆盖您的 Amazon EC2 环境设置。 CodeBuild 提供了以下 Amazon EC2 计算映像。有关

环境类型	映像标识符	实例大小	平台	已解析映像	定义
linux	4.0	small medium large xlarge	Amazon Linux 2	aws/codebuild/amazonlinux-x86_64-standard:4.0	al/standard/4.0
linux	5.0	2xlarge gpu_small gpu_large	Amazon Linux 2023	aws/codebuild/amazonlinux-x86_64-standard:5.0	al/standard/5.0
linux-ec2	latest	small medium large	Amazon Linux 2023	aws/codebuild/ami/amazonlinux-x86_64-base:latest	无
arm	2.0	small medium large xlarge	Amazon Linux 2	aws/codebuild/amazonlinux-aarch64-standard:2.0	al/aarch64/standard/2.0
arm	3.0	2xlarge	Amazon Linux 2023	aws/codebuild/amazonlinux-aarch64-standard:3.0	al/aarch64/standard/3.0

环境类型	映像标识符	实例大小	平台	已解析映像	定义
arm-ec2	latest	small medium large	Amazon Linux 2023	aws/codebuild/ami/amazonlinux-arm-base:latest	无
ubuntu	5.0	small medium large xlarge 2xlarge gpu_small gpu_large	Ubuntu 20.04	aws/codebuild/standard:5.0	ubuntu/standard/5.0
ubuntu	6.0		Ubuntu 22.04	aws/codebuild/standard:6.0	ubuntu/standard/6.0
ubuntu	7.0		Ubuntu 22.04	aws/codebuild/standard:7.0	ubuntu/standard/7.0
windows	1.0	medium large	Windows Server Core 2019	aws/codebuild/windows-base:2019-1.0	N/A
			Windows Server Core 2022	aws/codebuild/windows-base:2022-1.0	N/A
windows	2.0		Windows Server Core 2019	aws/codebuild/windows-base:2019-2.0	N/A

环境类型	映像标识符	实例大小	平台	已解析映像	定义
windows	3.0		Windows Server Core 2019	aws/codebuild/windows-base:2019-3.0	N/A
windows-ec2	2022	medium large	Windows Server Core 2022	aws/codebuild/ami/windows-base:2022	无

此外，您还可以使用以下值来覆盖 Lambda 环境设置。有关 CodeBuild Lambda 计算的更多信息，请参阅。[在 build 上运行 AWS Lambda 计算](#) CodeBuild 支持以下 Lambda 计算映像：

环境类型	映像标识符	实例大小			
linux-lambda arm-lambda	dotnet6 go1.21 corretto11 corretto17 corretto21 nodejs18 nodejs20 python3.11 python3.12	1GB 2GB 4GB 8GB 10GB			

环境类型	映像标识符	实例大小			
	ruby3.2				

有关更多信息，请参阅构建环境计算模式和类型和Docker 镜像由 CodeBuild。

将 webhook 与 AWS CodeBuild

AWS CodeBuild 支持与 GitHub、 GitHub 企业服务器、 GitLab 自助管理和 Bitbuck GitLab et 集成 webhook。

主题

- 使用 webhook 的最佳实践 AWS CodeBuild
- Bitbucket Webhook 事件
- GitHub 全球和组织网络挂钩
- GitHub 手动网络挂钩
- GitHub webhook 事件
- GitLab 群组 webhook
- GitLab 手动网络挂钩
- GitLab webhook 事件
- Buildkite 手动 webhook
- 拉取请求评论批准

使用 webhook 的最佳实践 AWS CodeBuild

对于使用公共存储库设置 webhook 的项目，我们建议使用以下选项：

设置ACTOR_ACCOUNT_ID过滤器

> 向项目的 webhook 过滤器组中添加ACTOR_ACCOUNT_ID过滤器，以指定哪些用户可以触发构建。传送到的每个 webhook 事件都 CodeBuild 附带了指定参与者标识符的发送者信息。 CodeBuild 将根据过滤器中提供的正则表达式模式过滤 webhook。您可以使用此筛选器指定允许触发构建的特定用户。有关更多信息，请参阅GitHub webhook 事件和Bitbucket Webhook 事件。

设置FILE_PATH过滤器

向项目的 webhook 过滤器组中添加FILE_PATH过滤器，以包括或排除更改时可能触发构件的文件。例如，您可以使用正则表达式模式（例如 ^buildspec.yml$）和 excludeMatchedPattern 属性来拒绝对 buildspec.yml 文件进行更改的构建请求。有关更多信息，请参阅GitHub webhook 事件和Bitbucket Webhook 事件。

缩小构建 IAM 角色的权限

由 Webhook 触发的构建使用项目中指定的 IAM 服务角色。我们建议将服务角色中的权限设置为运行构建所需的最低权限集。例如，在测试和部署场景中，创建一个用于测试的项目和另一个用于部署的项目。测试项目接受存储库中的 Webhook 构建，但不提供对您的资源的写入权限。部署项目提供对您的资源的写入权限，并且 Webhook 筛选条件配置为仅允许受信任的用户触发构建。

使用内联或 Amazon S3 存储的 buildspec

如果您自行在项目内定义内联 buildspec，或者将 buildspec 文件存储在 Amazon S3 存储桶中，则该 buildspec 文件仅对项目所有者可见。这样可以防止拉取请求对 buildspec 文件进行代码更改并触发不必要的构建。有关更多信息，请参阅《CodeBuild API Reference》中的 ProjectSource.buildspec。

Bitbucket Webhook 事件

您可以使用 Webhook 筛选条件组来指定哪个 Bitbucket Webhook 事件触发构建。例如，您可以指定仅在对特定分支做出更改时触发构建。

您可以创建一个或多个 Webhook 筛选条件组，来指定哪些 Webhook 事件触发构建。如果任何筛选条件组评估为 true（即组中的所有筛选条件都评估为 true），则会触发构建。创建筛选条件组时，应指定：

事件

对于 Bitbucket，您可以选择以下一个或多个事件：

- PUSH
- PULL_REQUEST_CREATED
- PULL_REQUEST_UPDATED
- PULL_REQUEST_MERGED
- PULL_REQUEST_CLOSED

webhook 的事件类型位于其在 X-Event-Key 字段中的标头中。下表显示了 X-Event-Key 标头值如何映射到事件类型。

> **Note**
>
> 如果您创建使用 PULL_REQUEST_MERGED 事件类型的 Webhook 筛选条件组，则必须在 Bitbucket Webhook 设置中启用 merged 事件。如果您创建使用 PULL_REQUEST_CLOSED 事件类型的 webhook 筛选条件组，则还必须在 Bitbucket webhook 设置中启用 declined 事件。

X-Event-Key 标头值	事件类型
repo:push	PUSH
pullrequest:created	PULL_REQUEST_CREATED
pullrequest:updated	PULL_REQUEST_UPDATED
pullrequest:fulfilled	PULL_REQUEST_MERGED
pullrequest:rejected	PULL_REQUEST_CLOSED

对于 PULL_REQUEST_MERGED，如果拉取请求与压缩策略合并且拉取请求分支已关闭，则原始的拉取请求提交将不再存在。在这种情况下，CODEBUILD_WEBHOOK_MERGE_COMMIT 环境变量包含压缩后的合并提交的标识符。

一个或多个可选筛选条件

使用正则表达式来指定筛选条件。对于触发构建的事件，组内与其关联的每个筛选条件都必须评估为 True。

CodeBuild 使用RE2正则表达式语法评估过滤器模式，该语法不支持向前看、向后看、反向引用或原子组。有关更多信息，请参阅 GitHub 网站上的RE2语法页面。

ACTOR_ACCOUNT_ID（控制台中的 ACTOR_ID）

当 Bitbucket 账户 ID 与正则表达式模式匹配时，Webhook 事件会触发构建。此值显示在 Webhook 筛选条件负载中的 actor 对象的 account_id 属性中。

`HEAD_REF`

当头部引用与正则表达式模式（例如 `refs/heads/branch-name` 和 `refs/tags/tag-name`）匹配时，Webhook 事件会触发构建。`HEAD_REF` 筛选条件将评估分支或标签的 Git 引用名称。分支或标签名称显示在 Webhook 负载的 `push` 对象中的 `new` 对象的 `name` 字段中。对于拉取请求事件，分支名称显示在 Webhook 负载中的 `source` 对象的 `branch` 中的 `name` 字段中。

`BASE_REF`

当基础引用与正则表达式模式匹配时，Webhook 事件会触发构建。`BASE_REF` 筛选条件仅处理拉取请求事件（例如，`refs/heads/branch-name`）。`BASE_REF` 筛选条件评估分支的 Git 引用名称。分支名称显示在 `branch` 对象的 `name` 字段中，该对象位于 Webhook 负载的 `destination` 对象中。

`FILE_PATH`

当更改的文件的路径与正则表达式模式匹配时，Webhook 会触发构建。

> **Note**
>
> 根据推送或拉取请求中前 3,000 个更改的文件对`FILE_PATH`过滤器进行评估。为确保按预期应用过滤器，请将更改的文件数量保持在该限制范围内。对于拉取请求事件，我们还建议使用拉取请求构建策略来控制哪些拉取请求可以启动构建。

`COMMIT_MESSAGE`

当 HEAD 提交消息与正则表达式模式匹配时，Webhook 会触发构建操作。

`WORKFLOW_NAME`

当工作流名称与正则表达式模式匹配时，Webhook 会触发构建操作。

> **Note**
>
> 您可以在 Bitbucket 存储库的 webhook 设置中找到 webhook 负载。

主题

- 筛选 Bitbucket Webhook 事件（控制台）

- 筛选 Bitbucket Webhook 事件（开发工具包）
- 过滤 Bitbucket 网络挂钩事件 (CloudFormation)

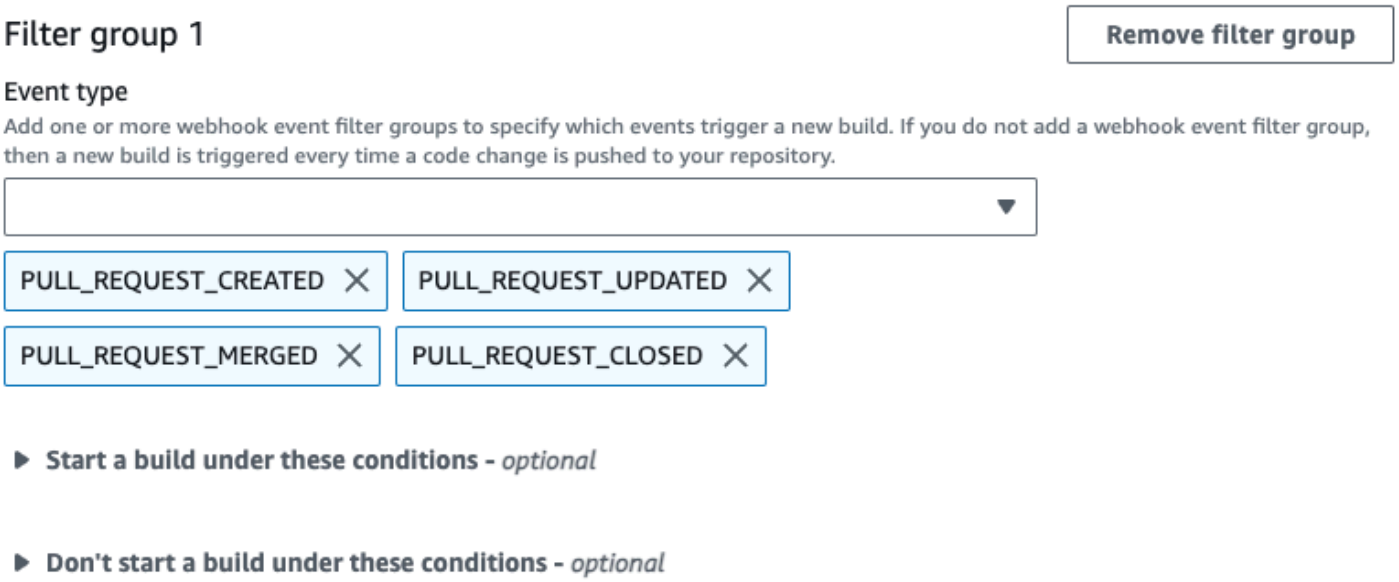
筛选 Bitbucket Webhook 事件（控制台）

要使用过滤 webhook 事件 AWS 管理控制台 ，请执行以下操作：

1. 创建项目时，选择每次将代码更改推送到此存储库时都会重新构建。
2. 从事件类型中，选择一个或多个事件。
3. 要在事件触发构建时进行筛选，请在在这些条件下开始构建下，添加一个或多个可选筛选条件。
4. 要在未触发事件时进行筛选，请在在这些条件下不开始构建下，添加一个或多个可选筛选条件。
5. 选择添加筛选条件组以添加另一个筛选条件组。

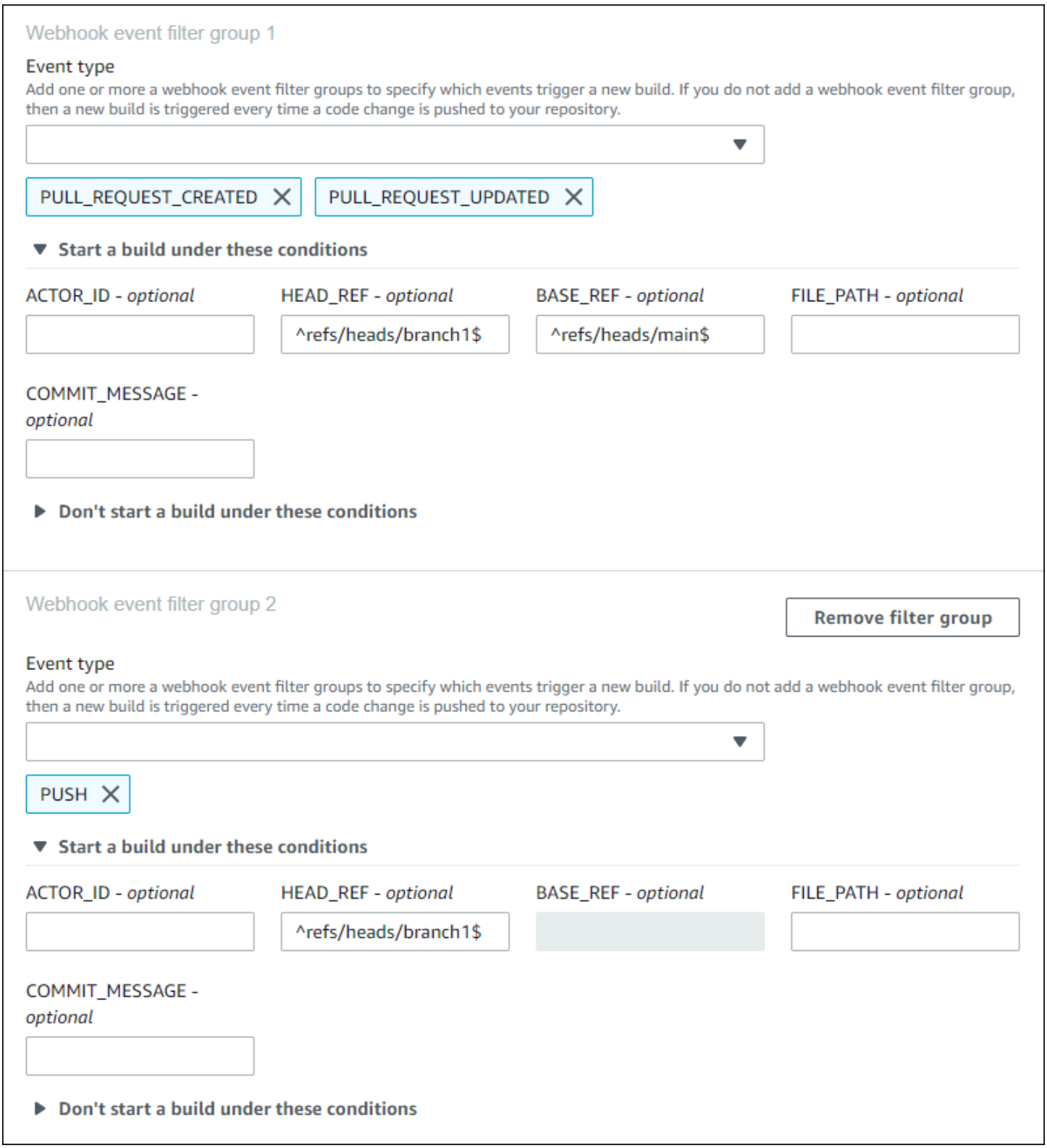
有关更多信息，请参阅《AWS CodeBuild API 参考》中的创建构建项目（控制台）和WebhookFilter。

在此示例中，Webhook 筛选条件组仅针对拉取请求触发构建：

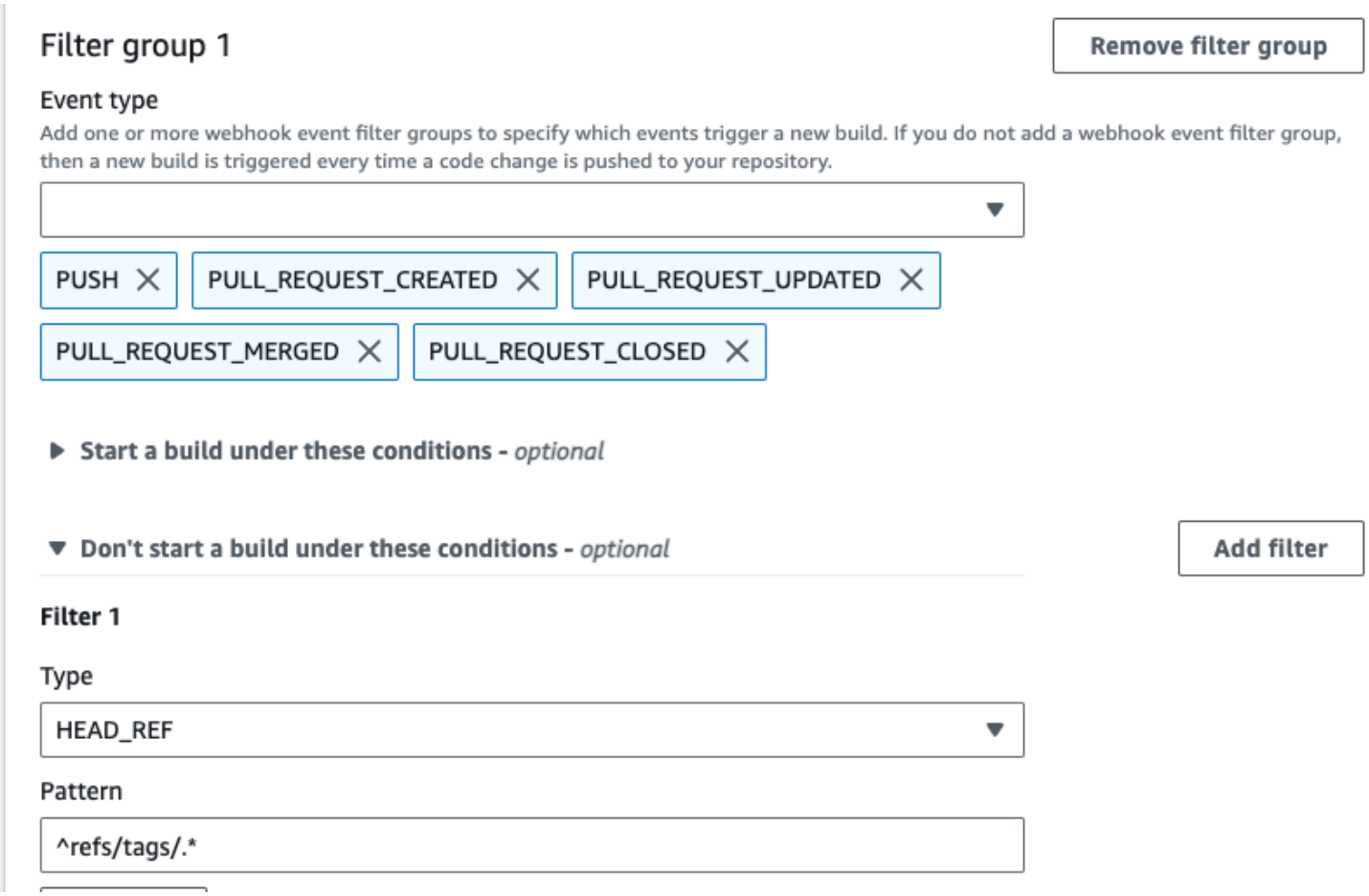


以两个筛选条件组为例，当一个或两个筛选条件评估为 True 时触发构建：

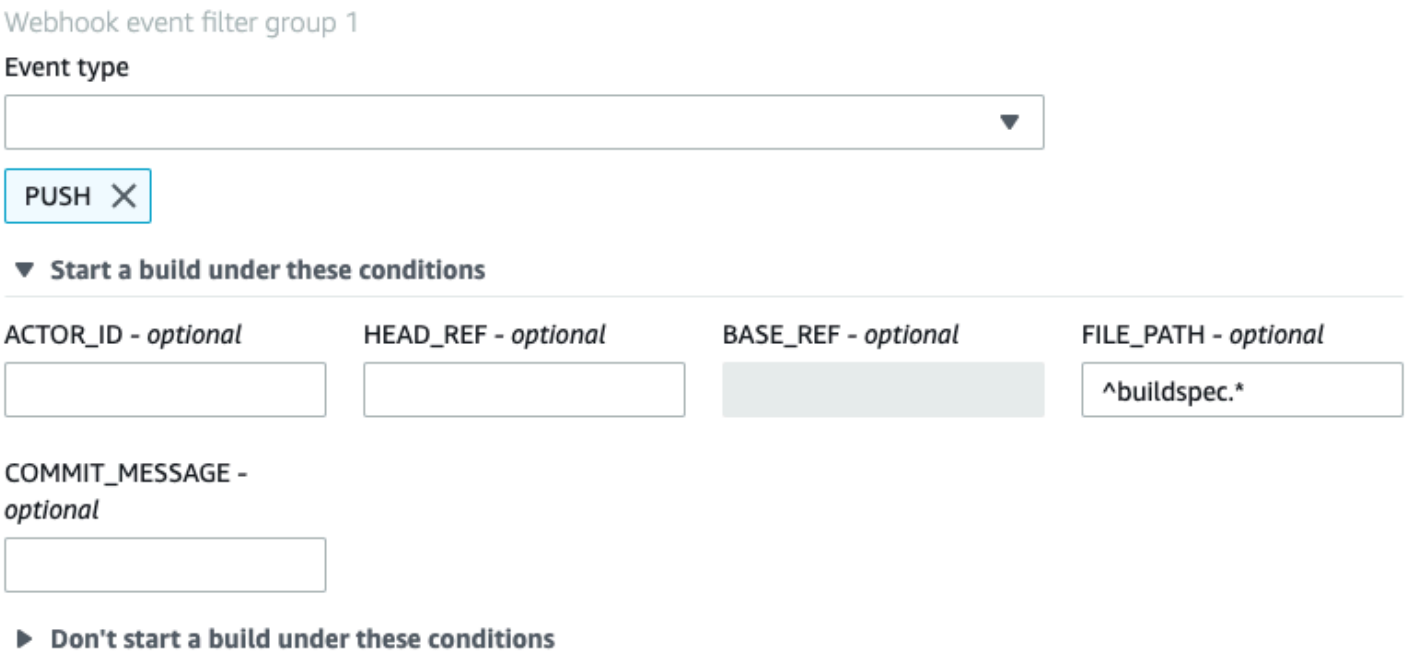
- 第一个筛选条件组使用与正则表达式 `^refs/heads/main$` 匹配的 Git 引用名称以及与 `^refs/heads/branch1!` 匹配的 HEAD 引用，指定在分支上创建或更新的拉取请求。
- 第二个筛选条件组使用与正则表达式 `^refs/heads/branch1$` 匹配的 Git 引用名称，指定分支上的推送请求。



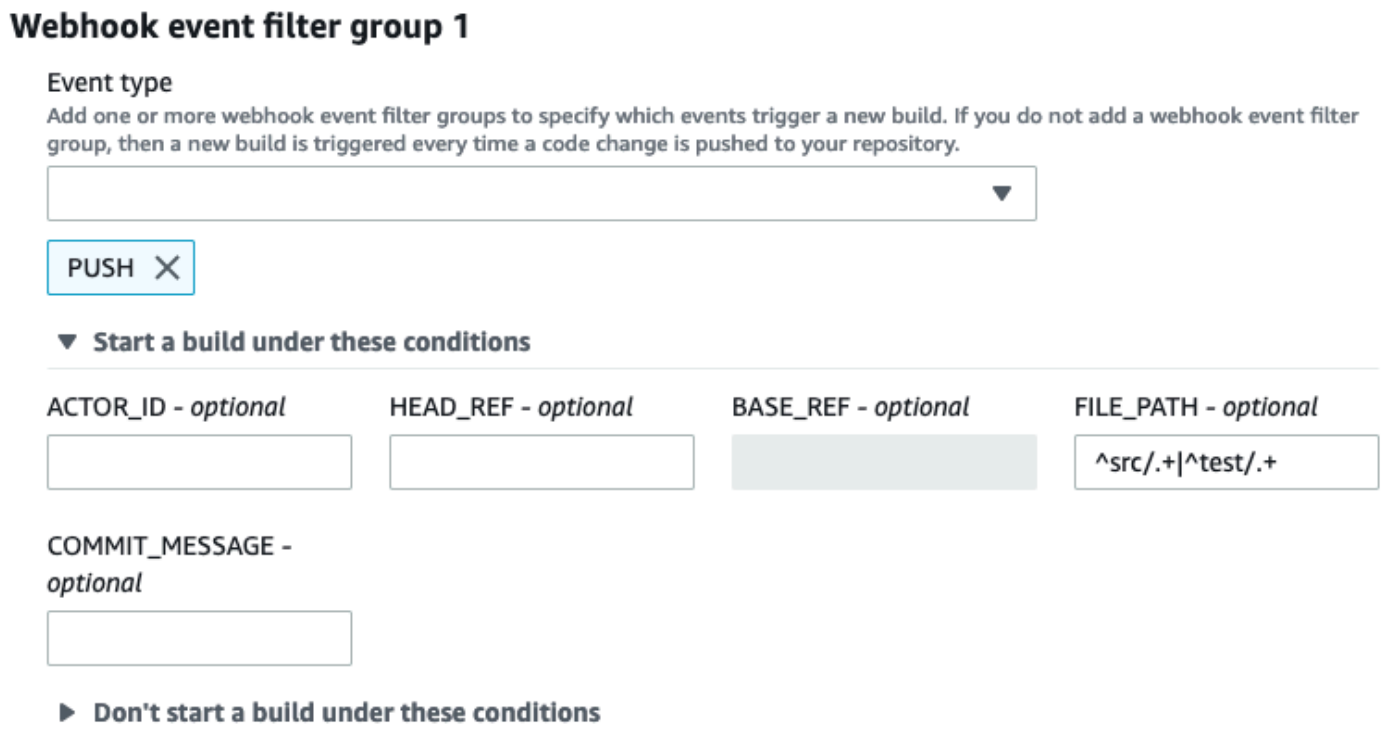
在此示例中，Webhook 筛选条件组会针对除标记事件之外的所有请求触发构建。



在此示例中，仅当名称与正则表达式 `^buildspec.*` 匹配的文件发生更改时，Webhook 筛选条件组才会触发构建。

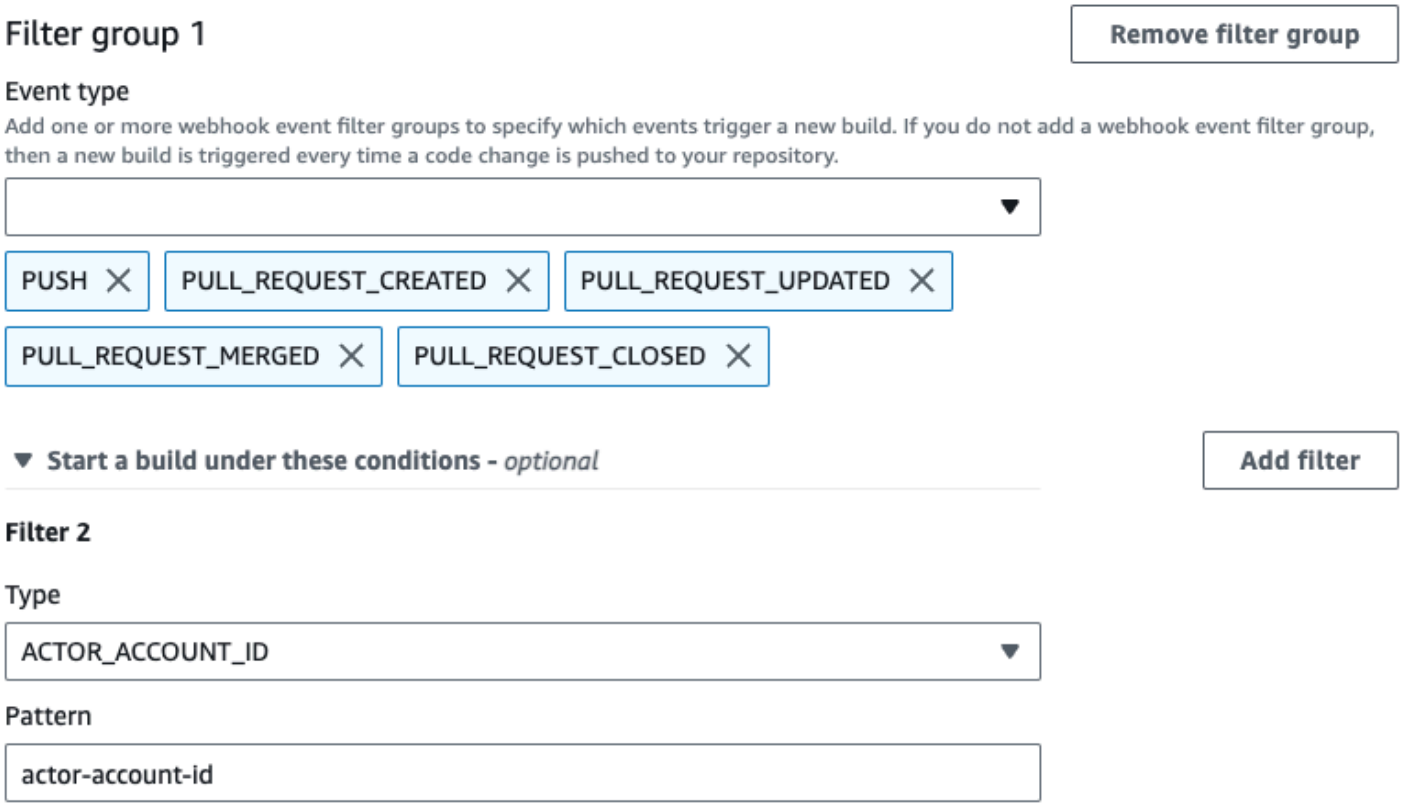


在此示例中，仅当 `src` 或 `test` 文件夹中的文件发生更改时，Webhook 筛选条件组才会触发构建。

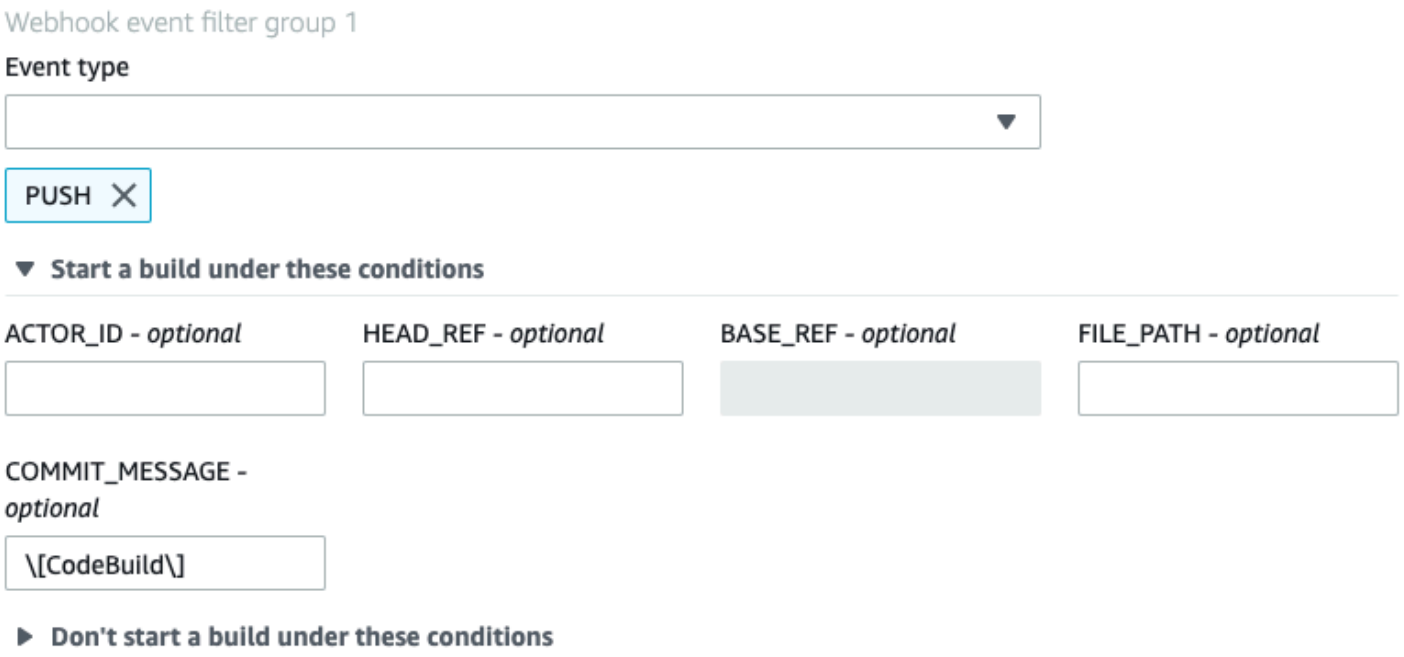


在此示例中，只有当其账户 ID 不与正则表达式 `actor-account-id` 匹配的 Bitbucket 用户进行更改时，Webhook 筛选条件组才会触发构建。

> **Note**
>
> 有关如何查找您的 Bitbucket 账户 ID 的信息 https://api.bitbucket.org/2.0/users/*user-name*，请参阅您的 Bitbucket 用户名在*user-name*哪里。



在本示例中，当 HEAD 提交消息与正则表达式 `\[CodeBuild\]` 匹配时，Webhook 筛选条件组会触发推送事件的构建。



筛选 Bitbucket Webhook 事件（开发工具包）

要使用 AWS CodeBuild 软件开发工具包筛选 webhook 事件，请使用`CreateWebhook`或 `UpdateWebhook` API 方法的请求语法中的`filterGroups`字段。有关更多信息，请参阅《CodeBuild API Reference》中的 WebhookFilter。

要创建仅针对拉取请求触发构建的 Webhook 筛选条件，请在请求语法中插入以下内容：

```
"filterGroups": [
  [
    {
      "type": "EVENT",
      "pattern": "PULL_REQUEST_CREATED, PULL_REQUEST_UPDATED, PULL_REQUEST_MERGED,
 PULL_REQUEST_CLOSED"
    }
  ]
]
```

要创建仅针对指定分支触发构建的 Webhook 筛选条件，请使用 `pattern` 参数指定用于筛选分支名称的正则表达式。以两个筛选条件组为例，当一个或两个筛选条件评估为 True 时触发构建：

- 第一个筛选条件组使用与正则表达式 `^refs/heads/main$` 匹配的 Git 引用名称以及与 `^refs/heads/myBranch$` 匹配的 HEAD 引用，指定在分支上创建或更新的拉取请求。

- 第二个筛选条件组使用与正则表达式 ^refs/heads/myBranch$ 匹配的 Git 引用名称，指定分支上的推送请求。

```
"filterGroups": [
  [
    {
      "type": "EVENT",
      "pattern": "PULL_REQUEST_CREATED, PULL_REQUEST_UPDATED, PULL_REQUEST_CLOSED"
    },
    {
      "type": "HEAD_REF",
      "pattern": "^refs/heads/myBranch$"
    },
    {
      "type": "BASE_REF",
      "pattern": "^refs/heads/main$"
    }
  ],
  [
    {
      "type": "EVENT",
      "pattern": "PUSH"
    },
    {
      "type": "HEAD_REF",
      "pattern": "^refs/heads/myBranch$"
    }
  ]
]
```

您可以使用 excludeMatchedPattern 参数指定不触发构建的事件。在此示例中，将针对除标记事件之外的所有请求触发构建。

```
"filterGroups": [
  [
    {
      "type": "EVENT",
      "pattern": "PUSH, PULL_REQUEST_CREATED, PULL_REQUEST_UPDATED,
 PULL_REQUEST_MERGED, PULL_REQUEST_CLOSED"
    },
    {
      "type": "HEAD_REF",
```

```
      "pattern": "^refs/tags/.*",
      "excludeMatchedPattern": true
    }
  ]
]
```

您可以创建仅在帐户 ID 为 `actor-account-id` 的 Bitbucket 用户进行更改时触发构建的筛选条件。

> **Note**
>
> 有关如何查找您的 Bitbucket 账户 ID 的信息 https://api.bitbucket.org/2.0/users/*user-name*，请参阅您的 Bitbucket 用户名在*user-name*哪里。

```
"filterGroups": [
  [
    {
      "type": "EVENT",
      "pattern": "PUSH, PULL_REQUEST_CREATED, PULL_REQUEST_UPDATED,
 PULL_REQUEST_MERGED, PULL_REQUEST_CLOSED"
    },
    {
      "type": "ACTOR_ACCOUNT_ID",
      "pattern": "actor-account-id"
    }
  ]
]
```

您可以创建只有当名称与 `pattern` 参数中的正则表达式匹配的文件发生更改时，才触发构建的筛选条件。在此示例中，筛选条件组指定仅当名称与正则表达式 `^buildspec.*` 匹配的文件更改时才触发构建。

```
"filterGroups": [
  [
    {
      "type": "EVENT",
      "pattern": "PUSH"
    },
    {
      "type": "FILE_PATH",
      "pattern": "^buildspec.*"
```

```
    }
  ]
]
```

在此示例中，筛选条件组指定仅当 `src` 或 `test` 文件夹中的文件发生更改时，才会触发构建。

```
"filterGroups": [
    [
        {
            "type": "EVENT",
            "pattern": "PUSH"
        },
        {
            "type": "FILE_PATH",
            "pattern": "^src/.+|^test/.+"
        }
    ]
]
```

您可以创建一个筛选条件，仅当 HEAD 提交消息与模式参数中的正则表达式匹配时才触发构建操作。在本示例中，筛选条件组指定仅当推送事件的 HEAD 提交消息与正则表达式 `\[CodeBuild\]` 匹配时，才触发构建操作。

```
  "filterGroups": [
    [
      {
        "type": "EVENT",
        "pattern": "PUSH"
      },
      {
        "type": "COMMIT_MESSAGE",
        "pattern": "\[CodeBuild\]"
      }
    ]
  ]
```

过滤 Bitbucket 网络挂钩事件 (CloudFormation)

要使用 CloudFormation 模板过滤 webhook 事件，请使用 AWS CodeBuild 项目的`FilterGroups`属性。 CloudFormation 模板的以下 YAML-formatted 部分创建了两个过滤器组。当这两个筛选条件的其中一个或两个评估为 True 时触发构建：

- 第一个筛选条件组使用与正则表达式 ^refs/heads/main$ 匹配的 Git 引用名称，指定由账户 ID 不为 12345 的 Bitbucket 用户在分支上创建或更新的拉取请求。
- 第二个筛选条件组使用与正则表达式 ^refs/heads/.* 匹配的 Git 引用名称，指定在分支上创建的推送请求。
- 第三个筛选条件组指定一个推送请求，其中包含与正则表达式 \[CodeBuild\] 匹配的 HEAD 提交消息。

```
CodeBuildProject:
  Type: AWS::CodeBuild::Project
  Properties:
    Name: MyProject
    ServiceRole: service-role
    Artifacts:
      Type: NO_ARTIFACTS
    Environment:
      Type: LINUX_CONTAINER
      ComputeType: BUILD_GENERAL1_SMALL
      Image: aws/codebuild/standard:5.0
    Source:
      Type: BITBUCKET
      Location: source-location
    Triggers:
      Webhook: true
      FilterGroups:
        - - Type: EVENT
            Pattern: PULL_REQUEST_CREATED,PULL_REQUEST_UPDATED
          - Type: BASE_REF
            Pattern: ^refs/heads/main$
            ExcludeMatchedPattern: false
          - Type: ACTOR_ACCOUNT_ID
            Pattern: 12345
            ExcludeMatchedPattern: true
        - - Type: EVENT
            Pattern: PUSH
          - Type: HEAD_REF
            Pattern: ^refs/heads/.*
          - Type: FILE_PATH
            Pattern: READ_ME
            ExcludeMatchedPattern: true
        - - Type: EVENT
            Pattern: PUSH
```

```
          - Type: COMMIT_MESSAGE
            Pattern: \[CodeBuild\]
          - Type: FILE_PATH
            Pattern: ^src/.+|^test/.+
```

GitHub 全球和组织网络挂钩

您可以使用 CodeBuild GitHub 全局或组织 webhook 在 GitHub 组织或企业内任何存储库的 webhook 事件上开始构建。全局和组织 webhook 可与任何现有的 GitHub webhook 事件类型一起使用，可以通过在创建 webhook 时添加范围配置来进行配置。 CodeBuild 您还可以使用全局和组织 webhook 在其中设置自托管的 Act GitHub ion 运行器，以便 CodeBuild在单个项目中接收来自多个存储库WORKFLOW_JOB_QUEUED的事件。

主题

- 设置全球或组织 GitHub webhook
- 筛选 GitHub 全球或组织 webhook 事件（控制台）
- 筛选 GitHub 组织 webhook 事件 (CloudFormation)

设置全球或组织 GitHub webhook

设置全球或组织 GitHub webhook 的高级步骤如下所示。有关全球和组织 GitHub webhook 的更多信息，请参阅GitHub 全球和组织网络挂钩。

1. 将项目的源位置设置为 CODEBUILD_DEFAULT_WEBHOOK_SOURCE_LOCATION。
2. 在 webhook 的范围配置中，将范围设置为 GITHUB_ORGANIZATION 或 GITHUB_GLOBAL，具体取决于范围应该是组织还是全局 webhook。有关更多信息，请参阅 webhook 的类型。
3. 在 webhook 的范围配置过程中指定一个名称。对于组织 webhook，这是组织名称，对于全局 webhook，这是企业名称。

   > **Note**
   >
   > 如果项目的源类型为 GITHUB_ENTERPRISE，则还需要在 webhook 范围配置过程中指定一个域。

4. （可选）如果您只想接收组织或企业内特定存储库的 webhook 事件，则可以在创建 webhook 时将 REPOSITORY_NAME 指定为筛选条件。

5. 如果您正在创建组织 webhook，请确保该用户 CodeBuild 有权在其中创建组织级 webhook。GitHub您可以创建具有组织 webhook 权限的 GitHub个人访问令牌，也可以使用 CodeBuild OAuth。有关更多信息，请参阅 GitHub 和 GitHub 企业服务器访问令牌。

   请注意，组织 webhook 可与任何现有的 GitHub webhook 事件类型一起使用。

6. 如果您要创建全局 webhook，则需要手动创建 webhook。有关如何在中手动创建 Webhook 的更多信息 GitHub，请参阅GitHub 手动网络挂钩。

   请注意，全局 webhook 仅支持 `WORKFLOW_JOB_QUEUED` 事件类型。有关更多信息，请参阅 教程：配置 CodeBuild-hosted GitHub 操作运行器。

筛选 GitHub 全球或组织 webhook 事件（控制台）

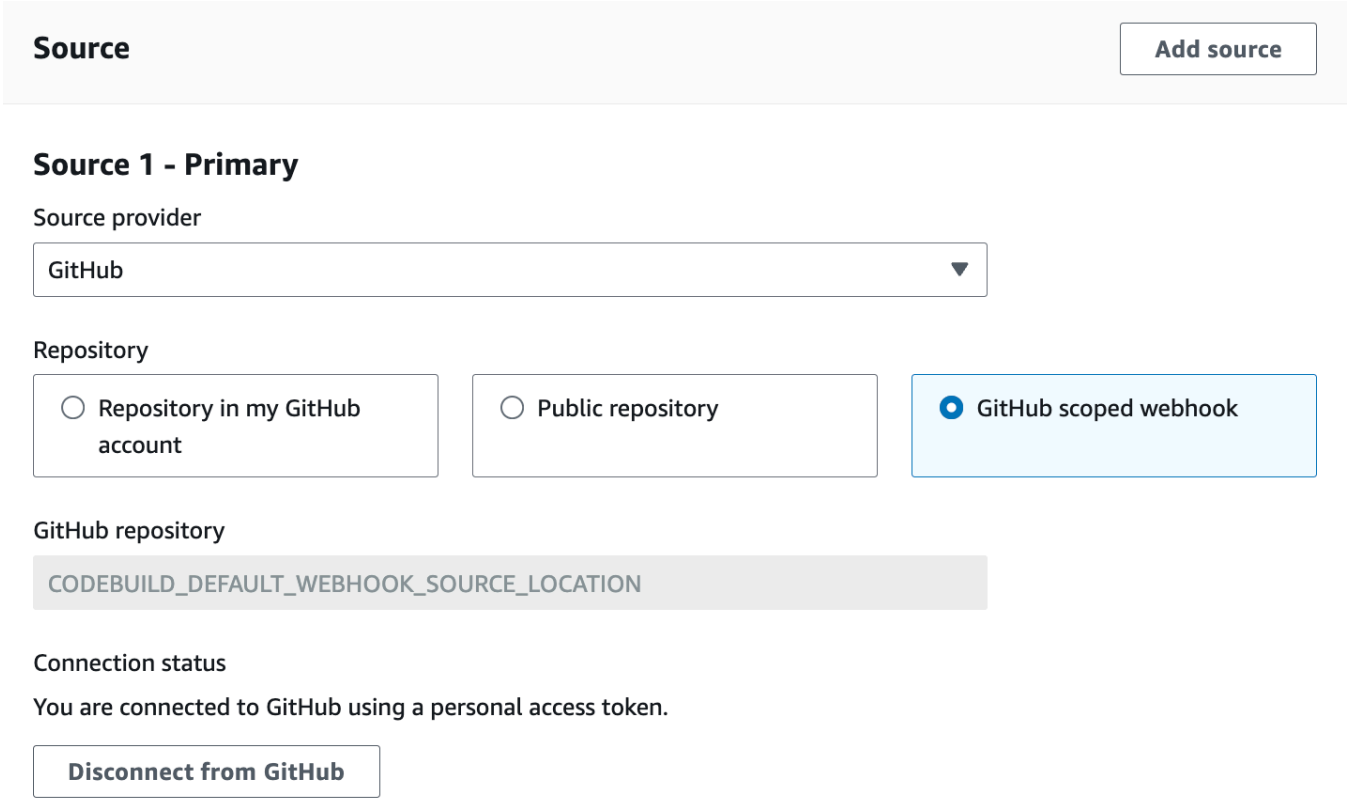
通过控制台创建 GitHub 项目时，选择以下选项在项目内创建 GitHub 全局或组织 webhook。有关全球和组织 GitHub webhook 的更多信息，请参阅GitHub 全球和组织网络挂钩。

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 创建构建项目。有关信息，请参阅创建构建项目（控制台）和运行构建（控制台）。

   - 在源中：
     - 对于来源提供商，选择GitHub或GitHub企业。
     - 对于存储库，选择 GitHub scoped webhook 。

       GitHub 存储库将自动设置为`CODEBUILD_DEFAULT_WEBHOOK_SOURCE_LOCATION`，这是全局和组织 webhook 所需的源位置。

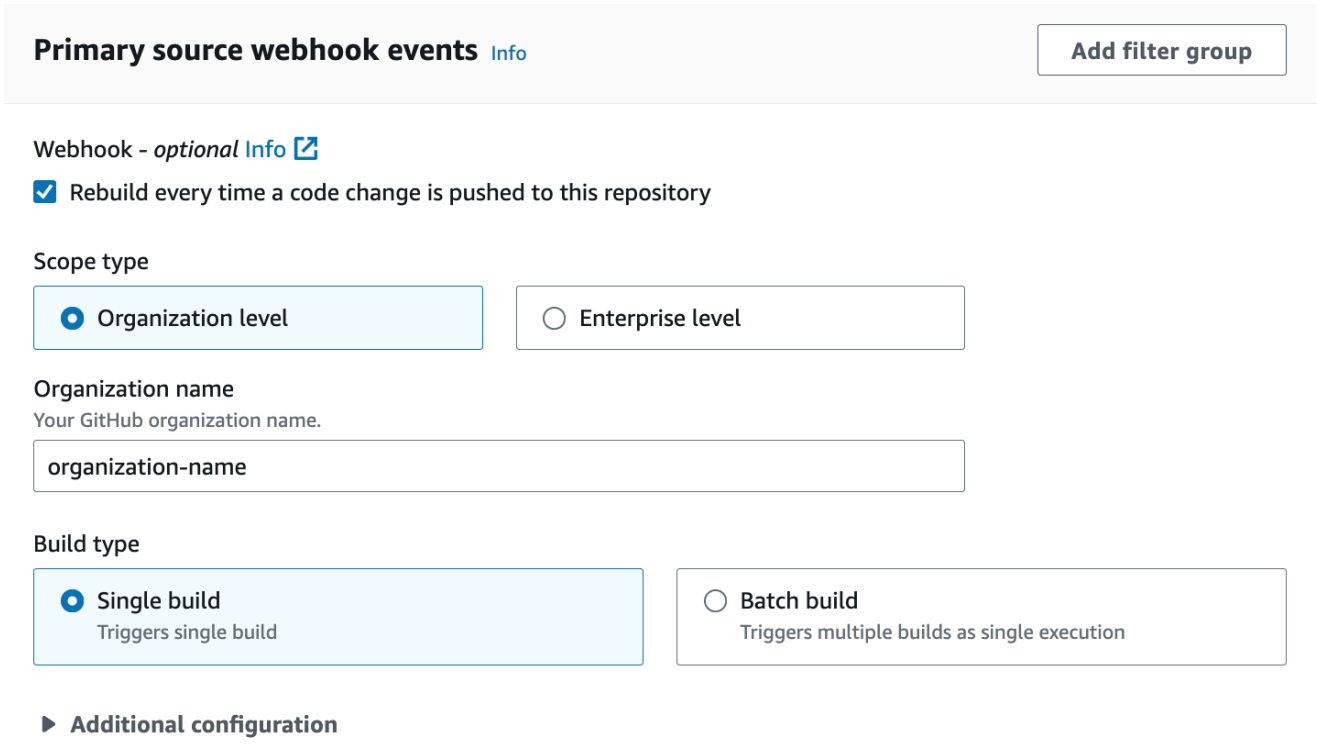
       > **Note**
       >
       > 如果您使用组织网络挂钩，请确保其 CodeBuild 有权在其中创建组织级网络挂钩。 GitHub如果您使用现有的 OAuth 连接，则可能需要重新生成该连接才能授予 CodeBuild 此权限。或者，您可以使用CodeBuild 手动 webhook 功能手动创建 webhook。请注意，如果您已有 GitHub OAuth 令牌并想添加其他组织权限，则可以撤消 OAuth 令牌的权限并通过控制台重新连接该令牌。 CodeBuild



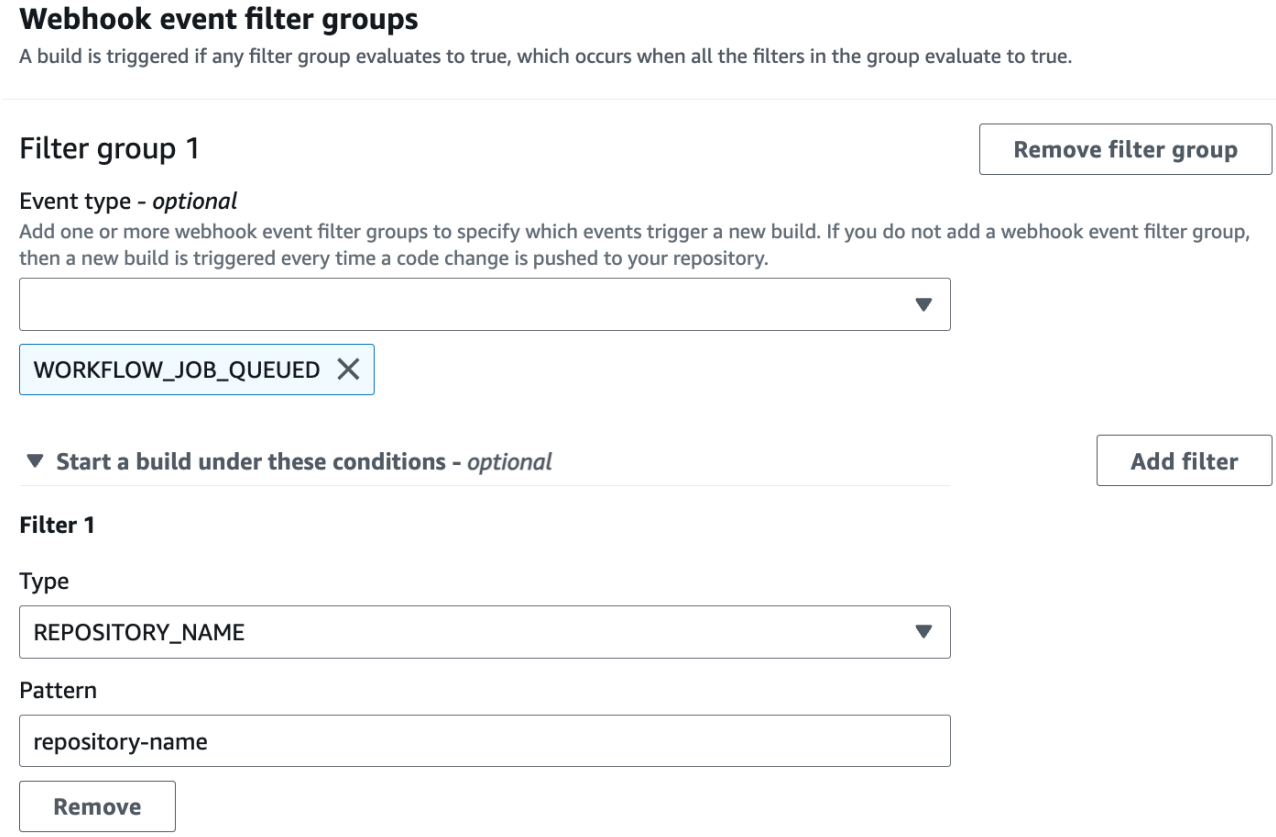
- 在主要源 Webhook 事件中：
  - 在范围类型中，如果您要创建组织 webhook，请选择组织级；如果要创建全局 webhook，请选择企业级。
  - 在名称中，输入企业或组织名称，具体取决于该 webhook 是全局 webhook 还是组织 webhook。

    如果项目的源类型为 `GITHUB_ENTERPRISE`，则还需要在 webhook 组织配置过程中指定一个域。例如，如果您组织的 URL 是 **`https://domain.com/orgs/org-name`**，则域是 **`https://domain.com`**。

    > **Note**
    >
    > 创建 webhook 后不能更改此名称。要更改名称，您可以删除并重新创建 webhook。如果您想完全删除 webhook，也可以将项目源位置更新为 GitHub存储库。



- （可选）在 webhook 事件筛选条件组中，您可以指定要触发新构建的事件。您也可以指定 REPOSITORY_NAME 作为筛选条件，仅根据来自特定存储库的 webhook 事件触发构建。



您也可以将事件类型设置为，`WORKFLOW_JOB_QUEUED`以设置自托管的 GitHub 操作运行器。有关更多信息，请参阅 教程：配置 CodeBuild-hosted GitHub 操作运行器。

3. 继续使用默认值，然后选择创建构建项目。

筛选 GitHub 组织 webhook 事件 (CloudFormation)

要使用 CloudFormation 模板筛选组织 webhook 事件，请使用 AWS CodeBuild 项目的`ScopeConfiguration`属性。有关全球和组织 GitHub webhook 的更多信息，请参阅GitHub 全球和组织网络挂钩。

> **Note**
>
> 不支持全局网络挂钩和 GitHub 企业网络挂钩。 CloudFormation

CloudFormation 模板的以下 YAML-formatted 部分创建了四个过滤器组。当这些筛选条件组的其中一个或全部评估为 True 时触发构建：

- 第一个过滤器组指定拉取请求是由没有账户 ID 的 GitHub 用户在分支上创建或更新的 Git 引用名称与正则表达式`^refs/heads/main$`相匹配`12345`。
- 第二个筛选条件组使用与正则表达式 `^refs/heads/.*` 匹配的 Git 引用名称，指定在名称与正则表达式 `READ_ME` 匹配的文件上创建的推送请求。
- 第三个筛选条件组指定一个推送请求，其中包含与正则表达式 `\[CodeBuild\]` 匹配的 HEAD 提交消息。
- 第四个筛选组使用与正则表达式匹配的工作流名称来指定 GitHub 操作工作流任务请求`\[CI-CodeBuild\]`。

```
CodeBuildProject:
  Type: AWS::CodeBuild::Project
  Properties:
    Name: MyProject
    ServiceRole: service-role
    Artifacts:
      Type: NO_ARTIFACTS
    Environment:
      Type: LINUX_CONTAINER
      ComputeType: BUILD_GENERAL1_SMALL
      Image: aws/codebuild/standard:5.0
    Source:
      Type: GITHUB
      Location: source-location
    Triggers:
      Webhook: true
      ScopeConfiguration:
        Name: organization-name
        Scope: GITHUB_ORGANIZATION
      FilterGroups:
        - - Type: EVENT
            Pattern: PULL_REQUEST_CREATED,PULL_REQUEST_UPDATED
          - Type: BASE_REF
            Pattern: ^refs/heads/main$
            ExcludeMatchedPattern: false
          - Type: ACTOR_ACCOUNT_ID
            Pattern: 12345
            ExcludeMatchedPattern: true
        - - Type: EVENT
            Pattern: PUSH
          - Type: HEAD_REF
```

```
          Pattern: ^refs/heads/.*
        - Type: FILE_PATH
          Pattern: READ_ME
          ExcludeMatchedPattern: true
      - - Type: EVENT
          Pattern: PUSH
        - Type: COMMIT_MESSAGE
          Pattern: \[CodeBuild\]
        - Type: FILE_PATH
          Pattern: ^src/.+|^test/.+
      - - Type: EVENT
          Pattern: WORKFLOW_JOB_QUEUED
        - Type: WORKFLOW_NAME
          Pattern: \[CI-CodeBuild\]
```

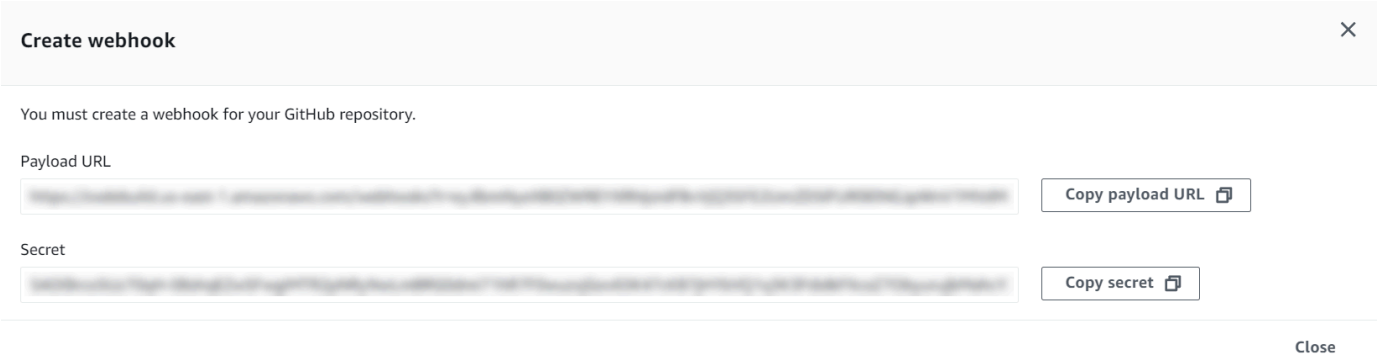
GitHub 手动网络挂钩

您可以配置手动 GitHub webhook 以 CodeBuild 防止自动尝试在其中创建 webhook。 GitHub CodeBuild 返回负载网址作为创建 webhook 调用的一部分，该负载网址可用于在其中手动创建 webhook。 GitHub即使未 CodeBuild 被列入在您的 GitHub 账户中创建 webhook 的许可名单，您仍然可以为构建项目手动创建 webhook。

使用以下步骤创建 GitHub 手动 webhook。

创建手 GitHub 动 webhook

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 创建构建项目。有关信息，请参阅创建构建项目（控制台）和运行构建（控制台）。
   - 在源中：
     - 对于来源提供商，选择GitHub。
     - 对于 “存储库”，选择 “我的 GitHub 账户中的存储库” 。
     - 对于存储库 URL，输入 **https://github.com/*user-name*/*repository-name***。
   - 在主要源 Webhook 事件中：
     - 对于 webhook - 可选，选择每次将代码更改推送到此存储库时都会重新构建。
     - 选择其他配置，对于手动创建-可选，选择在 GitHub 控制台中手动为该存储库创建 Webhook。。
3. 继续使用默认值，然后选择创建构建项目。请记下有效载荷 URL 和密钥值，因为稍后要用到它们。



4. 在处打开 GitHub 控制台，`https://github.com/user-name/repository-name/settings/hooks`然后选择添加 webhook 。

   - 在有效载荷 URL 中，输入之前记下的有效载荷 URL 值。
   - 对于内容类型，选择application/json。
   - 在密钥中，输入之前记下的密钥值。
   - 配置将向发送 webhook 负载的各个事件。 CodeBuild在您希望哪些事件可触发这个 webhook？中，选择让我选择单个事件，然后从以下事件中选择：推送、拉取请求和发布。如果要为 `WORKFLOW_JOB_QUEUED` 事件启动构建，请选择工作流作业。要了解有关 Act GitHub ions 运行器的更多信息，请参阅教程：配置 CodeBuild-hosted GitHub 操作运行器。要了解有关支持的事件类型的更多信息 CodeBuild，请参阅GitHub webhook 事件。

5. 选择添加 webhook。

GitHub webhook 事件

您可以使用 webhook 过滤器组来指定哪些 GitHub webhook 事件触发构建。例如，您可以指定仅在对特定分支做出更改时触发构建。

您可以创建一个或多个 Webhook 筛选条件组，来指定哪些 Webhook 事件触发构建。如果任何筛选条件组评估为 true（即组中的所有筛选条件都评估为 true），则会触发构建。创建筛选条件组时，应指定：

事件

对于 GitHub，您可以选择以下一个或多个事件：`PUSH`、`PULL_REQUEST_CREATED`、`PULL_REQUEST_UPDATED`、`PULL_REQUEST_REOPENED`、`PUL` 和`WORKFLOW_JOB_QUEUED`。webhook 事件类型在 webhook 负载中的 `X-GitHub-Event` 标头中。在 `X-GitHub-Event` 标头中，您可能会看到 `pull_request` 或 `push`。对于拉取请求事件，

类型在 webhook 事件负载的 action 字段中。下表显示了 X-GitHub-Event 标头值和 webhook 拉取请求负载 action 字段值如何映射到可用的事件类型。

X-GitHub-Event 标头值	Webhook 事件负载 **action** 值	事件类型
pull_request	opened	PULL_REQUEST_CREATED
pull_request	reopened	PULL_REQUEST_REOPENED
pull_request	synchronize	PULL_REQUEST_UPDATED
pull_request	closed，并且 merged 字段为 true	PULL_REQUEST_MERGED
pull_request	closed，并且 merged 字段为 false	PULL_REQUEST_CLOSED
push	n/a	PUSH
release	released	RELEASED
release	prereleased	PRERELEASED
workflow_job	queued	WORKFLOW_JOB_QUEUED

> **Note**
>
> 该PULL_REQUEST_REOPENED事件类型只能 GitHub与 GitHub 企业服务器一起使用。RELEASED和PRERELEASED事件类型 GitHub 只能与一起使用。有关 WORKFLOW_JOB_QUEUED 的更多信息，请参阅教程：配置 CodeBuild-hosted GitHub 操作运行器。

一个或多个可选筛选条件

使用正则表达式来指定筛选条件。对于触发构建的事件，组内与其关联的每个筛选条件都必须评估为 True。

CodeBuild 使用RE2正则表达式语法评估过滤器模式，该语法不支持向前看、向后看、反向引用或原子组。有关更多信息，请参阅 GitHub 网站上的RE2语法页面。

`ACTOR_ACCOUNT_ID`（控制台中的 `ACTOR_ID`）

当 GitHub 或 GitHub企业服务器账户 ID 匹配正则表达式模式时，webhook 事件会触发构建。此值在 webhook 负载中的 `sender` 对象的 `id` 属性中。

`HEAD_REF`

当头部引用与正则表达式模式（例如 `refs/heads/branch-name` 和 `refs/tags/tag-name`）匹配时，Webhook 事件会触发构建。对于推送事件，引用名称在 Webhook 负载中的 `ref` 属性中。对于拉取请求事件，分支名称在 Webhook 负载中的 `head` 对象的 `ref` 属性中。

`BASE_REF`

当基本引用与正则表达式模式（例如 `refs/heads/branch-name`）匹配时，Webhook 事件会触发构建。`BASE_REF` 筛选器只能与拉取请求事件一起使用。分支名称在 webhook 负载中的 `base` 对象的 `ref` 属性中。

`FILE_PATH`

当更改的文件的路径与正则表达式模式匹配时，Webhook 会触发构建。`FILE_PATH`过滤器可用于 GitHub 推送和拉取请求事件以及 GitHub企业服务器推送事件。它不能与 GitHub企业服务器拉取请求事件一起使用。

> **Note**
>
> 根据推送或拉取请求中的前 100 个更改的文件对`FILE_PATH`过滤器进行评估。为确保按预期应用过滤器，请将更改的文件数量保持在该限制范围内。对于拉取请求事件，我们还建议使用拉取请求构建策略来控制哪些拉取请求可以启动构建。

`COMMIT_MESSAGE`

当 HEAD 提交消息与正则表达式模式匹配时，Webhook 会触发构建操作。`COMMIT_MESSAGE`过滤器可用于 GitHub 推送和拉取请求事件以及 GitHub企业服务器推送事件。它不能与 GitHub企业服务器拉取请求事件一起使用。

`TAG_NAME`

当发布的标签名称与正则表达式模式匹配时，webhook 会触发构建操作。`TAG_NAME`过滤器可用于 GitHub 已发布和预发布的请求事件。

`RELEASE_NAME`

当发布名称与正则表达式模式匹配时，webhook 会触发构建操作。`RELEASE_NAME`过滤器可用于 GitHub 已发布和预发布的请求事件。

`REPOSITORY_NAME`

当存储库名称与正则表达式模式匹配时，webhook 会触发构建操作。`REPOSITORY_NAME`过滤器只能用于 GitHub 全球或组织 webhook。

`ORGANIZATION_NAME`

当组织名称与正则表达式模式匹配时，webhook 会触发构建。`ORGANIZATION_NAME`过滤器只能用于 GitHub 全局网络挂钩。

`WORKFLOW_NAME`

当工作流名称与正则表达式模式匹配时，Webhook 会触发构建操作。`WORKFLOW_NAME`过滤器可用于 Act GitHub ions 工作流任务队列请求事件。

> **Note**
>
> 你可以在仓库的 webhook 设置中找到 webhook 负载。 GitHub

主题

- 过滤 GitHub webhook 事件（控制台）
- 过滤 GitHub webhook 事件 (SDK)
- 过滤 GitHub webhook 事件 (CloudFormation)

过滤 GitHub webhook 事件（控制台）

使用以下说明使用过滤 GitHub webhook 事件。 AWS 管理控制台有关 GitHub webhook 事件的更多信息，请参阅GitHub webhook 事件。

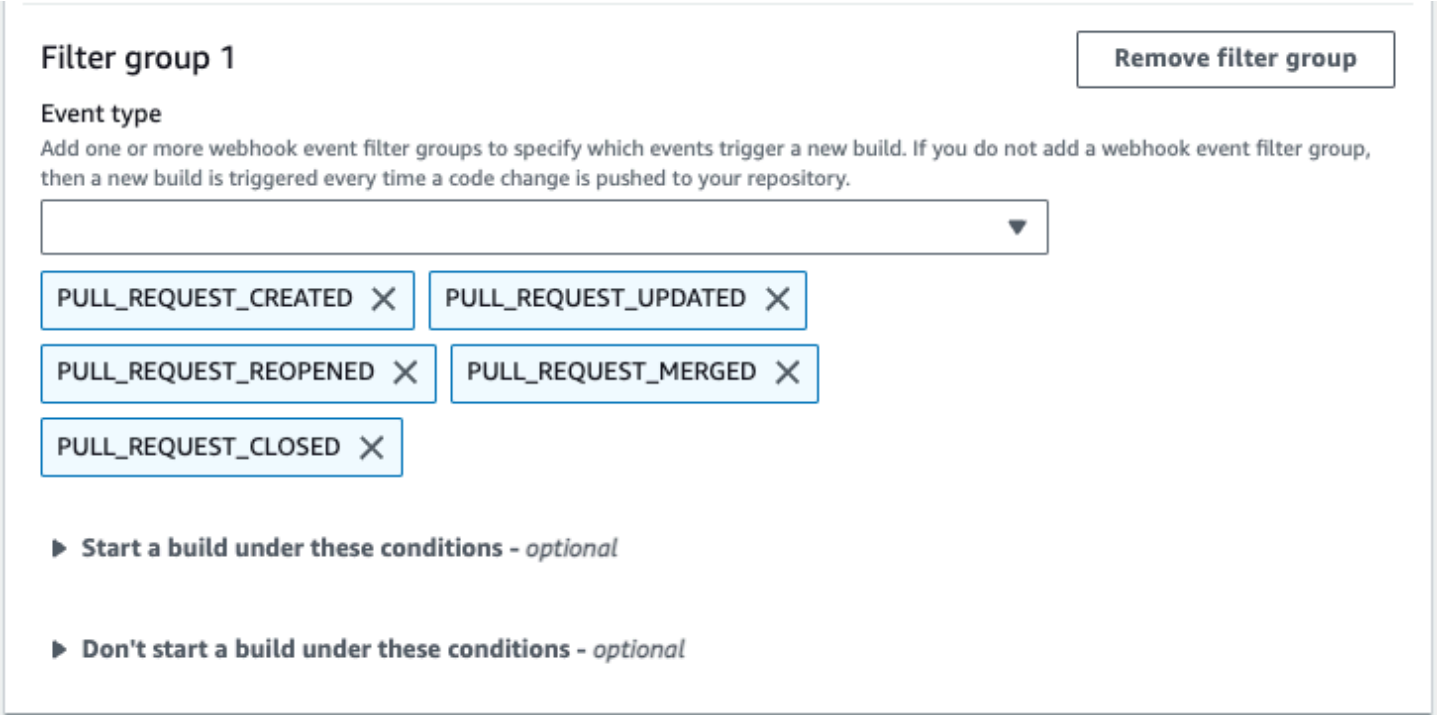
在主要源 webhook 事件中，选择以下内容。仅当您在我的 GitHub帐户中为源存储库选择存储库时，此部分才可用。

1. 创建项目时，选择每次将代码更改推送到此存储库时都会重新构建。
2. 从事件类型中，选择一个或多个事件。

3. 要在事件触发构建时进行筛选，请在在这些条件下开始构建下，添加一个或多个可选筛选条件。
4. 要在未触发事件时进行筛选，请在在这些条件下不开始构建下，添加一个或多个可选筛选条件。
5. 选择添加筛选条件组，以添加另一个筛选条件组（如果需要）。

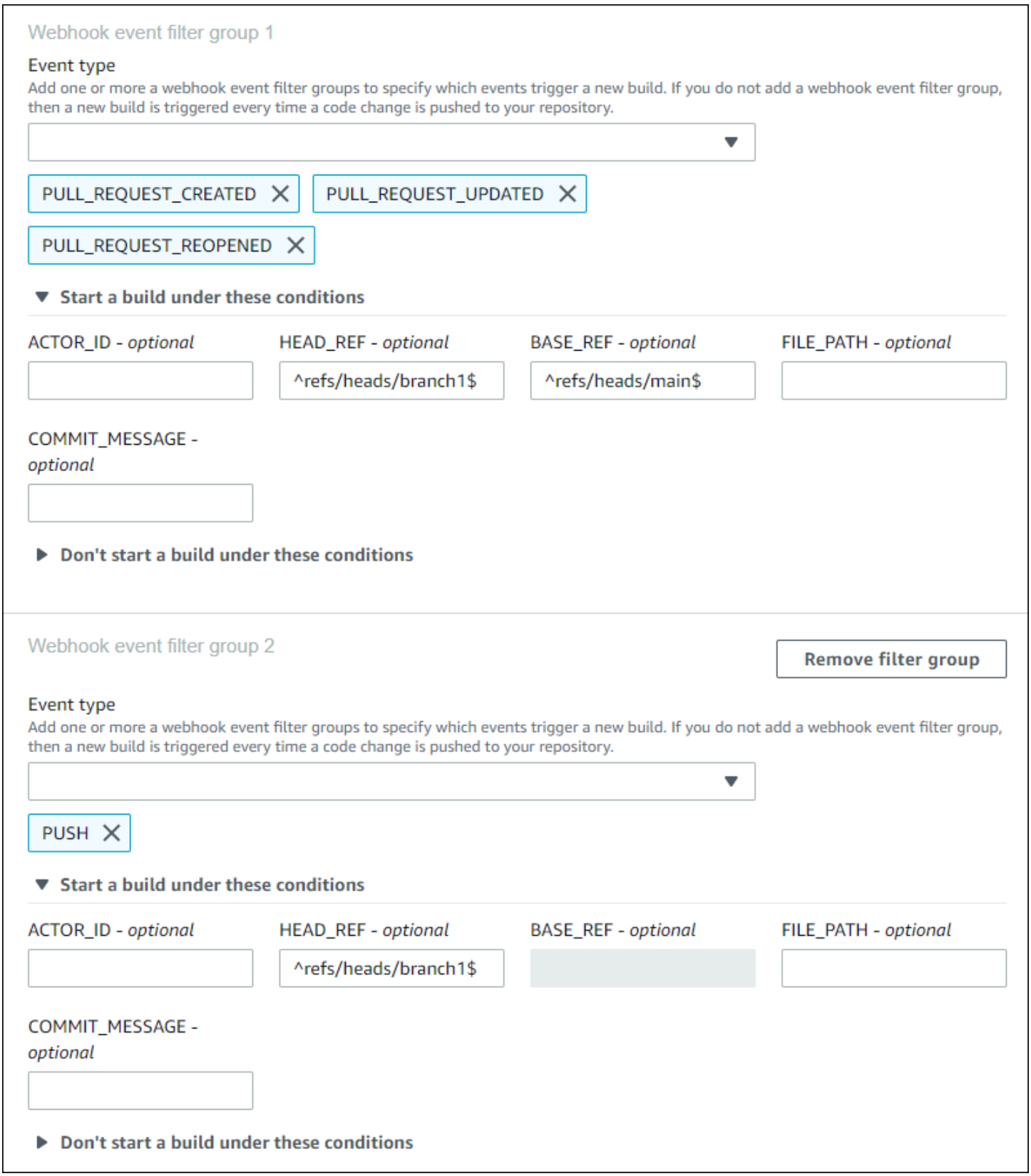
有关更多信息，请参阅《AWS CodeBuild API 参考》中的创建构建项目（控制台）和WebhookFilter。

在此示例中，Webhook 筛选条件组仅针对拉取请求触发构建：

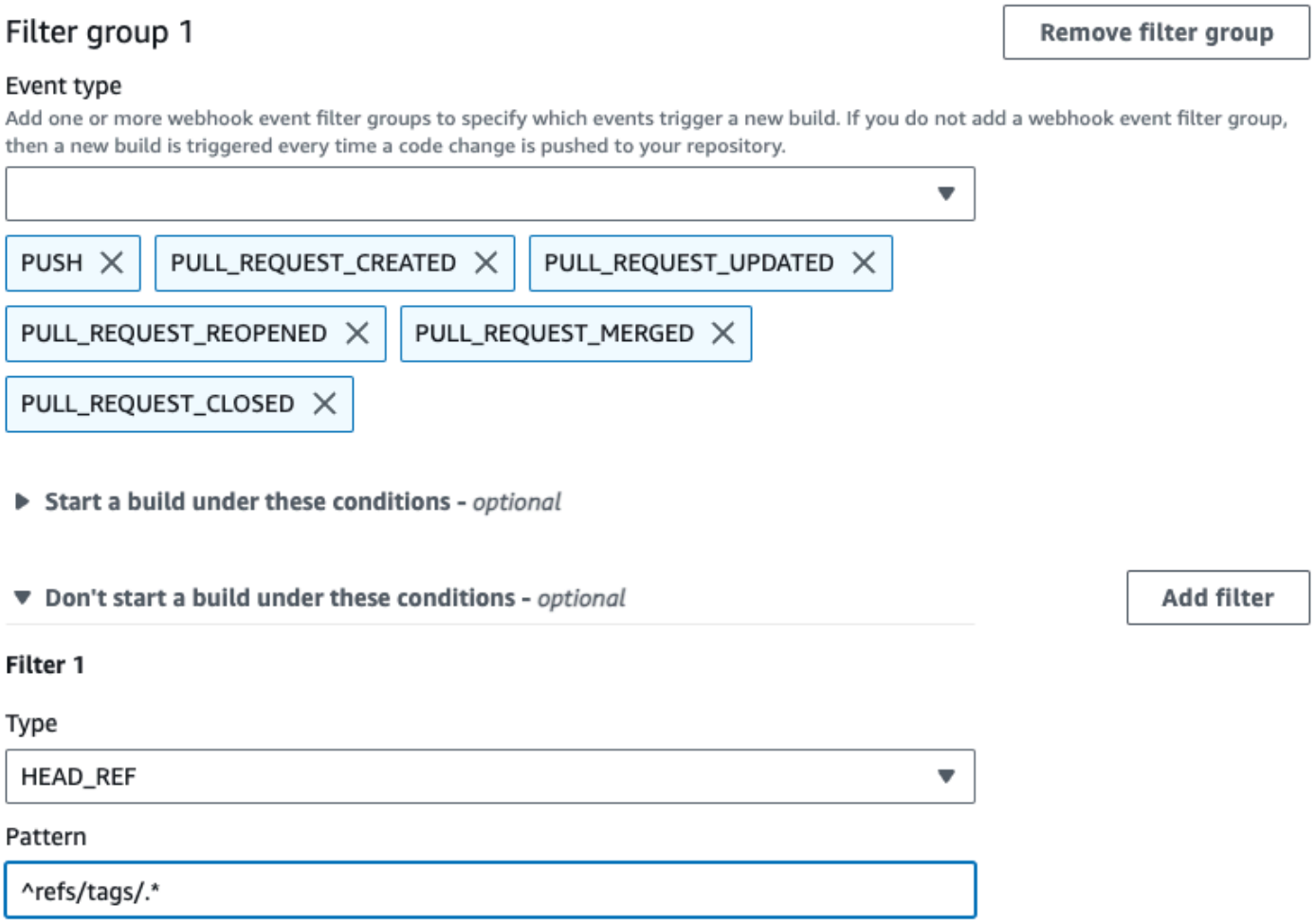


以两个 Webhook 筛选条件组为例，当一个或两个筛选条件评估为 True 时触发构建：

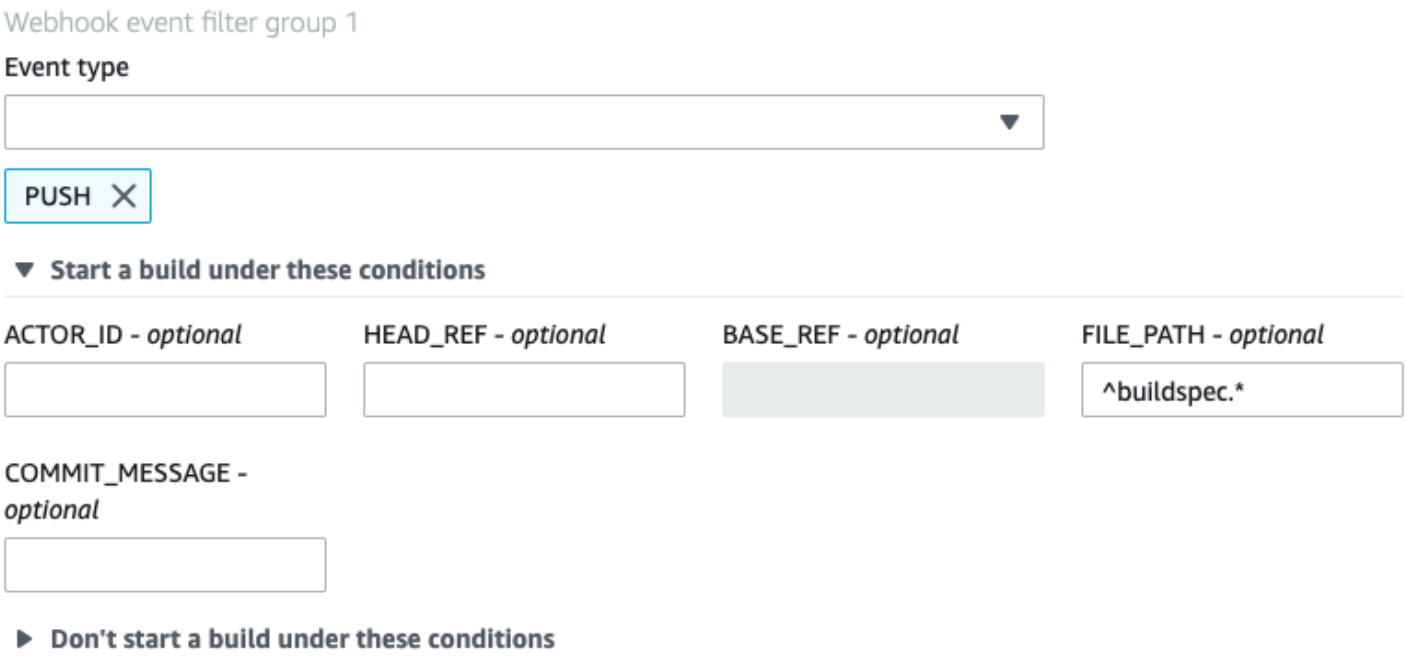
- 第一个筛选条件组使用与正则表达式 `^refs/heads/main$` 匹配的 Git 引用名称以及与 `^refs/heads/branch1$` 匹配的头部引用，指定在分支上创建、更新或重新打开的拉取请求。
- 第二个筛选条件组使用与正则表达式 `^refs/heads/branch1$` 匹配的 Git 引用名称，指定分支上的推送请求。



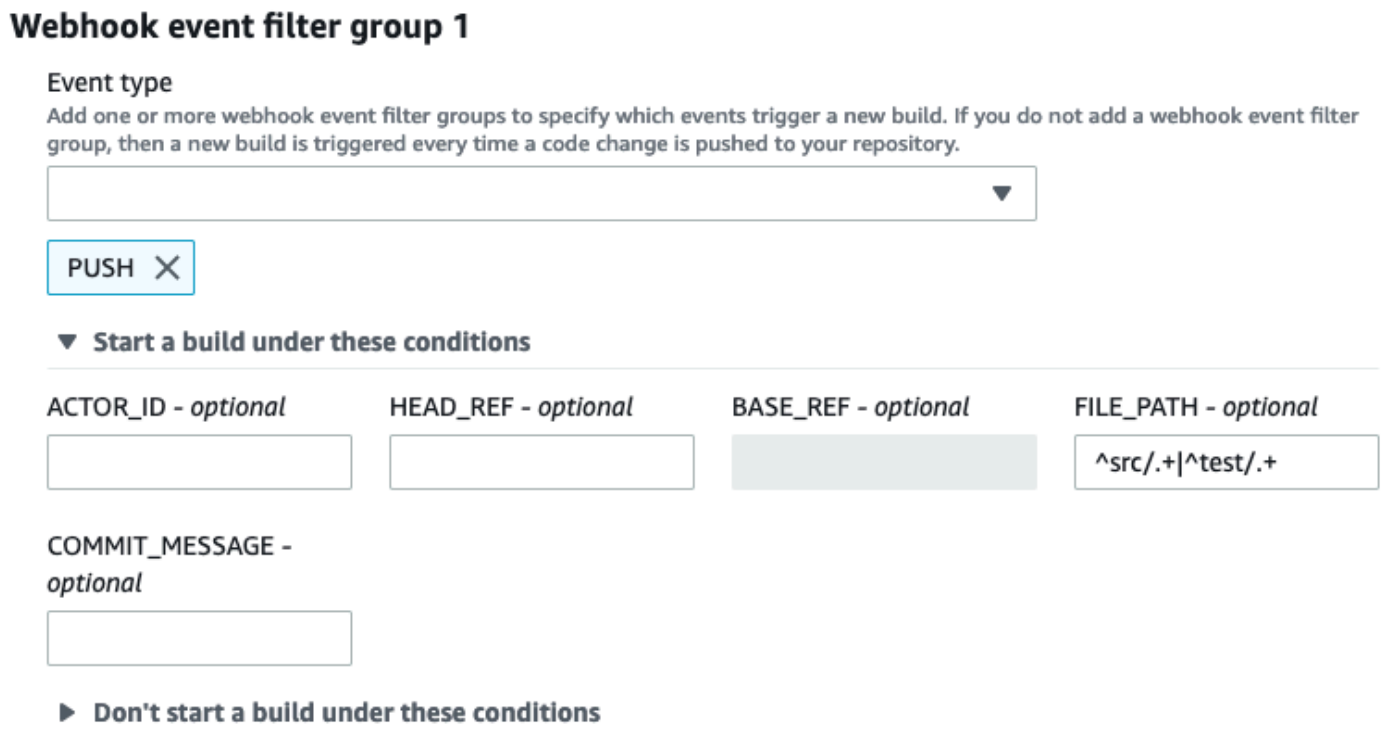
在此示例中，Webhook 筛选条件组会针对除标记事件之外的所有请求触发构建。



在此示例中，仅当名称与正则表达式 `^buildspec.*` 匹配的文件发生更改时，Webhook 筛选条件组才会触发构建。

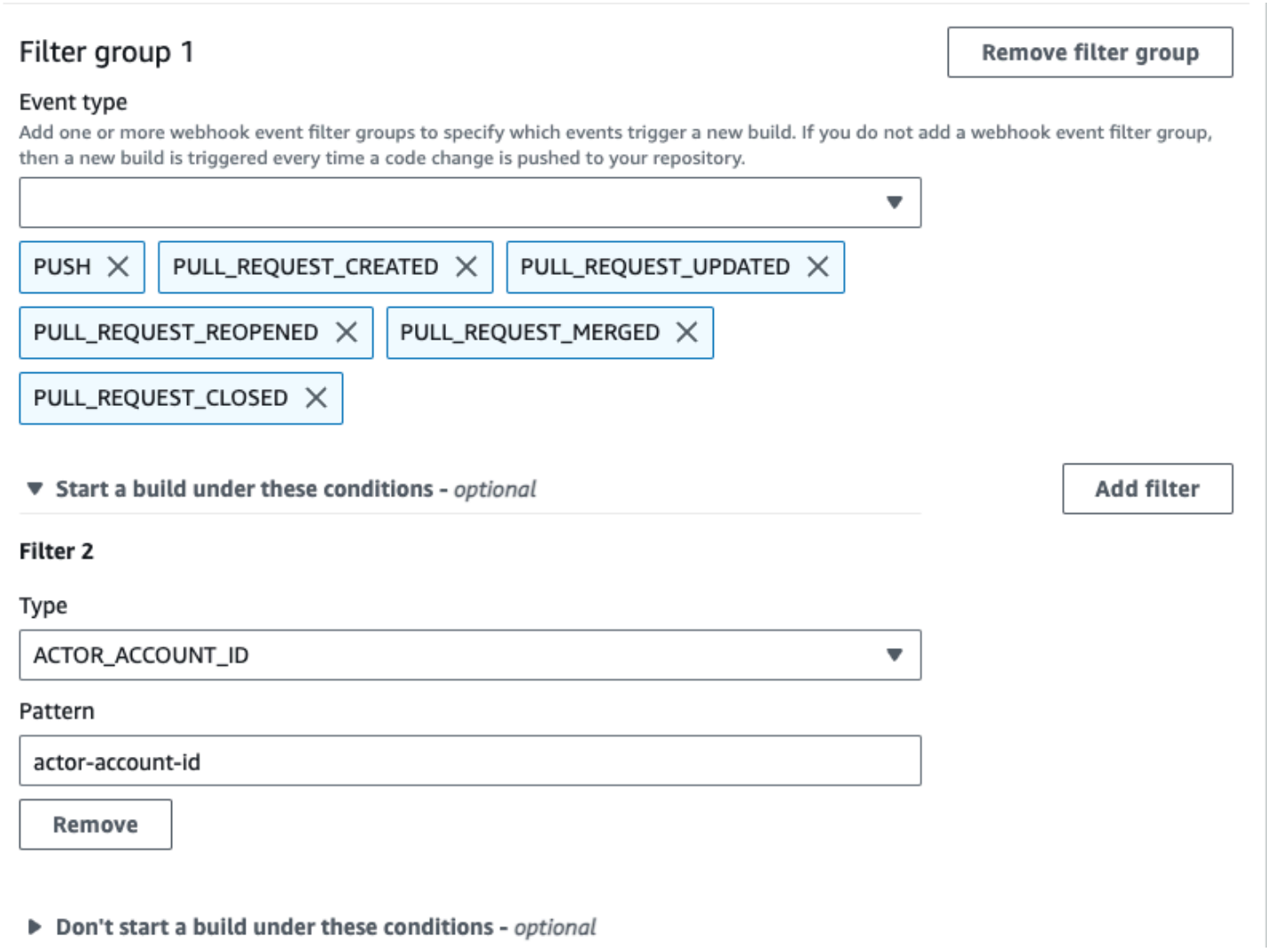


在此示例中，仅当 `src` 或 `test` 文件夹中的文件发生更改时，Webhook 筛选条件组才会触发构建。

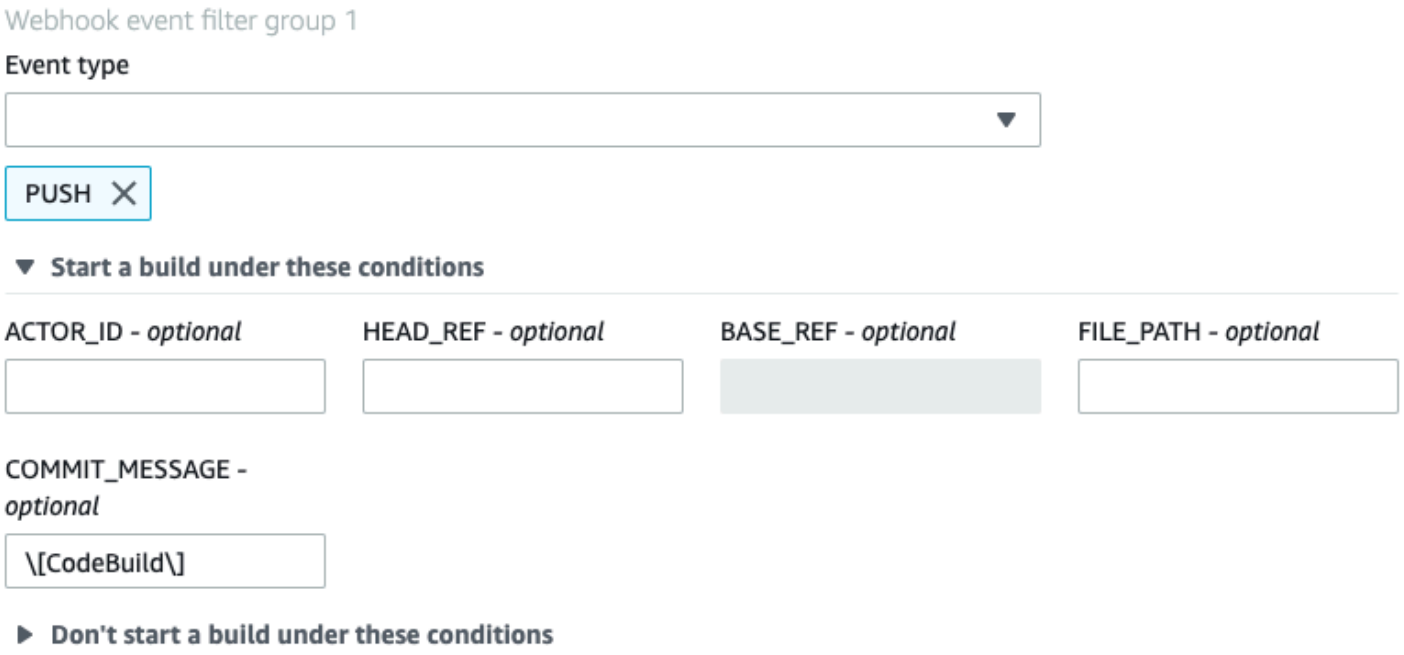


在此示例中，只有当账户 ID 与正则表达式`actor-account-id`匹配的指定用户 GitHub 或 GitHub 企业服务器用户进行更改时，webhook 过滤器组才会触发构建。

> **Note**
>
> 有关如何查找您的 GitHub 账户 ID 的信息 https://api.github.com/users/*user-name*，请参阅您的 GitHub 用户名在*user-name*哪里。

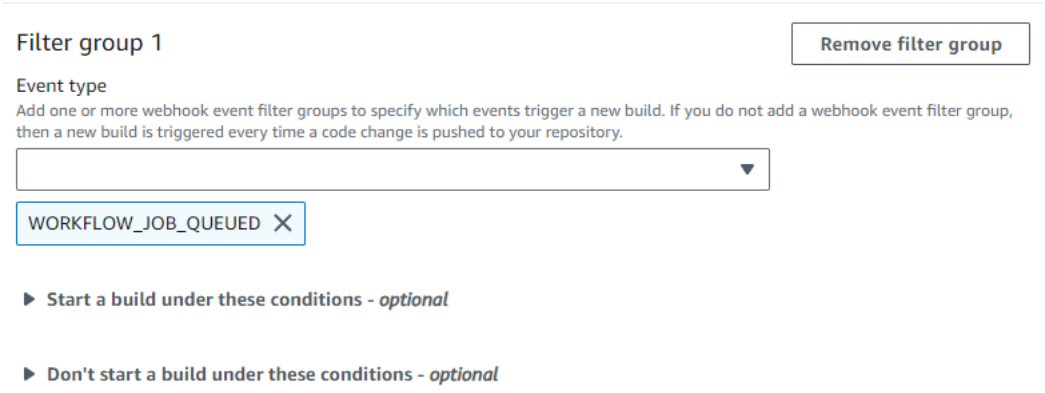


在本示例中，当 HEAD 提交消息与正则表达式 `\[CodeBuild\]` 匹配时，Webhook 筛选条件组会触发推送事件的构建。

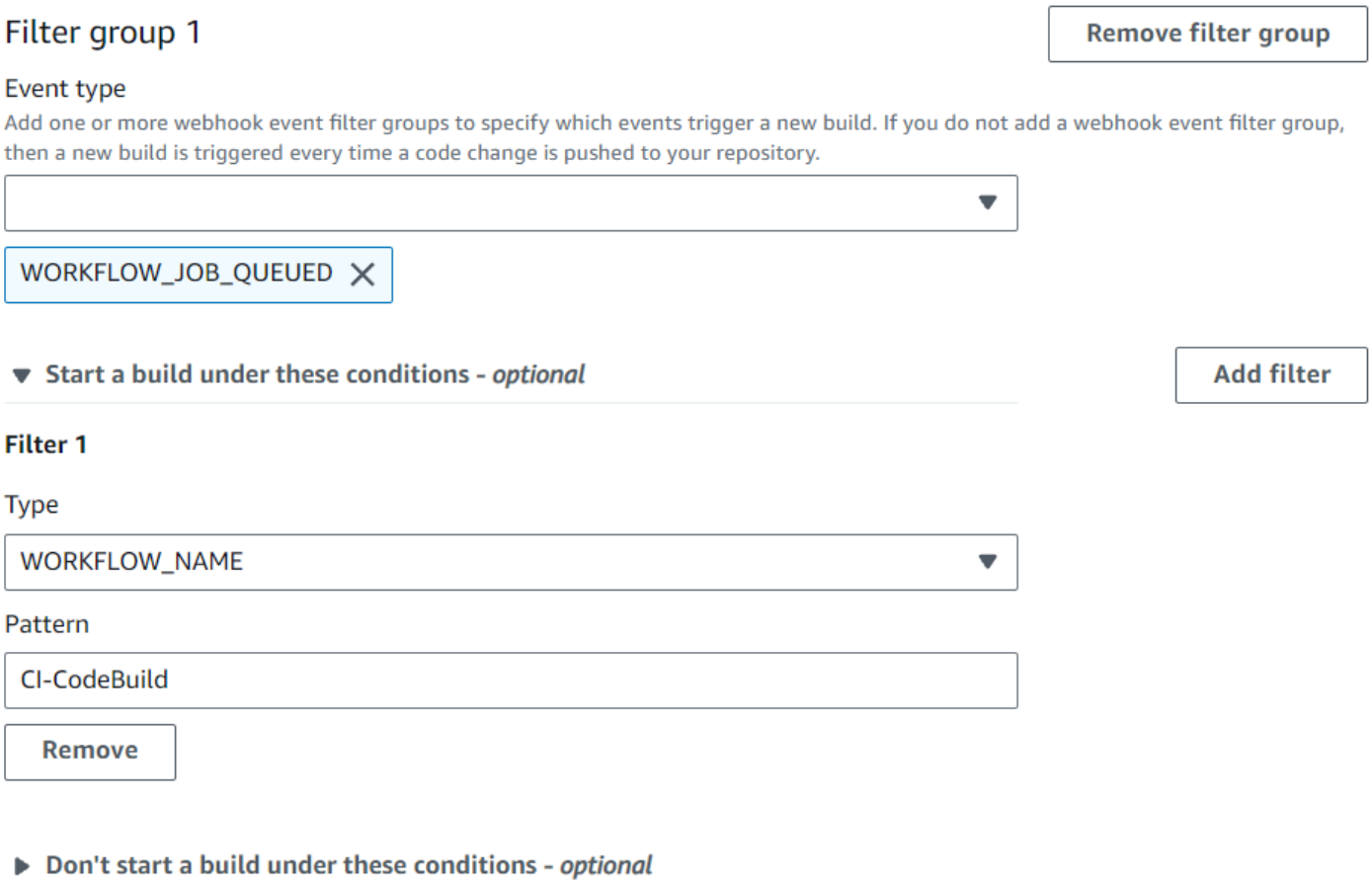


在此示例中，webhook 筛选器组仅触发构建 Act GitHub ions 工作流程作业事件。

> **Note**
>
> CodeBuild 仅当网络挂钩的过滤器组包含 WORKFLOW_JOB _QUEUED 事件过滤器时，才会处理 GitHub 操作工作流作业。



在此示例中，当工作流名称与正则表达式 `CI-CodeBuild` 匹配时，webhook 筛选条件组才会触发构建。



过滤 GitHub webhook 事件 (SDK)

要使用 AWS CodeBuild 软件开发工具包筛选 webhook 事件，请使用`CreateWebhook`或 `UpdateWebhook` API 方法的请求语法中的`filterGroups`字段。有关更多信息，请参阅《CodeBuild API Reference》中的 WebhookFilter。

有关 GitHub webhook 事件的更多信息，请参阅GitHub webhook 事件。

要创建仅针对拉取请求触发构建的 Webhook 筛选条件，请在请求语法中插入以下内容：

```
"filterGroups": [
   [
        {
            "type": "EVENT",
            "pattern": "PULL_REQUEST_CREATED, PULL_REQUEST_UPDATED,
 PULL_REQUEST_REOPENED, PULL_REQUEST_MERGED, PULL_REQUEST_CLOSED"
        }
    ]
```

```
]
```

要创建仅针对指定分支触发构建的 Webhook 筛选条件，请使用 pattern 参数指定用于筛选分支名称的正则表达式。以两个筛选条件组为例，当一个或两个筛选条件评估为 True 时触发构建：

- 第一个筛选条件组使用与正则表达式 ^refs/heads/main$ 匹配的 Git 引用名称以及与 ^refs/heads/myBranch$ 匹配的头部引用，指定在分支上创建、更新或重新打开的拉取请求。
- 第二个筛选条件组使用与正则表达式 ^refs/heads/myBranch$ 匹配的 Git 引用名称，指定分支上的推送请求。

```
"filterGroups": [
    [
        {
            "type": "EVENT",
            "pattern": "PULL_REQUEST_CREATED, PULL_REQUEST_UPDATED,
 PULL_REQUEST_REOPENED"
        },
        {
            "type": "HEAD_REF",
            "pattern": "^refs/heads/myBranch$"
        },
        {
            "type": "BASE_REF",
            "pattern": "^refs/heads/main$"
        }
    ],
    [
        {
            "type": "EVENT",
            "pattern": "PUSH"
        },
        {
            "type": "HEAD_REF",
            "pattern": "^refs/heads/myBranch$"
        }
    ]
]
```

您可以使用 excludeMatchedPattern 参数指定不触发构建的事件。例如，在此示例中，将针对除标记事件之外的所有请求触发构建。

```
"filterGroups": [
    [
        {
            "type": "EVENT",
            "pattern": "PUSH, PULL_REQUEST_CREATED, PULL_REQUEST_UPDATED,
 PULL_REQUEST_REOPENED, PULL_REQUEST_MERGED, PULL_REQUEST_CLOSED"
        },
        {
            "type": "HEAD_REF",
            "pattern": "^refs/tags/.*",
            "excludeMatchedPattern": true
        }
    ]
]
```

您可以创建只有当名称与 pattern 参数中的正则表达式匹配的文件发生更改时，才触发构建的筛选条件。在此示例中，筛选条件组指定仅当名称与正则表达式 ^buildspec.* 匹配的文件更改时才触发构建。

```
"filterGroups": [
    [
        {
            "type": "EVENT",
            "pattern": "PUSH"
        },
        {
            "type": "FILE_PATH",
            "pattern": "^buildspec.*"
        }
    ]
]
```

在此示例中，筛选条件组指定仅当 src 或 test 文件夹中的文件发生更改时，才会触发构建。

```
"filterGroups": [
    [
        {
            "type": "EVENT",
            "pattern": "PUSH"
        },
        {
            "type": "FILE_PATH",
```

```
            "pattern": "^src/.+|^test/.+"
        }
    ]
]
```

您可以创建过滤器，该筛选器仅在具有帐户 ID 的指定用户 GitHub 或 GitHub 企业服务器用户进行更改时才触发构建`actor-account-id`。

> **Note**
>
> 有关如何查找您的 GitHub 账户 ID 的信息 https://api.github.com/users/*user-name*，请参阅您的 GitHub 用户名在*user-name*哪里。

```
"filterGroups": [
    [
        {
            "type": "EVENT",
            "pattern": "PUSH, PULL_REQUEST_CREATED, PULL_REQUEST_UPDATED,
 PULL_REQUEST_REOPENED, PULL_REQUEST_MERGED, PULL_REQUEST_CLOSED"
        },
        {
            "type": "ACTOR_ACCOUNT_ID",
            "pattern": "actor-account-id"
        }
    ]
]
```

您可以创建一个筛选条件，仅当 HEAD 提交消息与模式参数中的正则表达式匹配时才触发构建操作。在本示例中，筛选条件组指定仅当推送事件的 HEAD 提交消息与正则表达式 `\[CodeBuild\]` 匹配时，才触发构建操作。

```
"filterGroups": [
    [
        {
            "type": "EVENT",
            "pattern": "PUSH"
        },
        {
            "type": "COMMIT_MESSAGE",
            "pattern": "\[CodeBuild\]"
```

```
        }
    ]
]
```

要创建仅触发构建 Act GitHub ions 工作流程任务的 webhook 筛选器，请在请求语法中插入以下内容：

```
"filterGroups": [
   [
        {
            "type": "EVENT",
            "pattern": "WORKFLOW_JOB_QUEUED"
        }
    ]
]
```

过滤 GitHub webhook 事件 (CloudFormation)

要使用 CloudFormation 模板过滤 webhook 事件，请使用 AWS CodeBuild 项目的`FilterGroups`属性。

有关 GitHub webhook 事件的更多信息，请参阅GitHub webhook 事件。

CloudFormation 模板的以下 YAML-formatted 部分创建了两个过滤器组。当这两个筛选条件的其中一个或两个评估为 True 时触发构建：

- 第一个过滤器组指定拉取请求是由没有账户 ID 的 GitHub 用户在具有与正则表达式`^refs/heads/main$`相匹配的 Git 引用名称的分支上创建或更新的`12345`。
- 第二个筛选条件组使用与正则表达式 `^refs/heads/.*` 匹配的 Git 引用名称，指定在名称与正则表达式 `READ_ME` 匹配的文件上创建的推送请求。
- 第三个筛选条件组指定一个推送请求，其中包含与正则表达式 `\[CodeBuild\]` 匹配的 HEAD 提交消息。
- 第四个筛选组使用与正则表达式匹配的工作流名称来指定 GitHub 操作工作流任务请求`\[CI-CodeBuild\]`。

```
CodeBuildProject:
  Type: AWS::CodeBuild::Project
  Properties:
    Name: MyProject
```

```
    ServiceRole: service-role
    Artifacts:
      Type: NO_ARTIFACTS
    Environment:
      Type: LINUX_CONTAINER
      ComputeType: BUILD_GENERAL1_SMALL
      Image: aws/codebuild/standard:5.0
    Source:
      Type: GITHUB
      Location: source-location
    Triggers:
      Webhook: true
      FilterGroups:
        - - Type: EVENT
            Pattern: PULL_REQUEST_CREATED,PULL_REQUEST_UPDATED
          - Type: BASE_REF
            Pattern: ^refs/heads/main$
            ExcludeMatchedPattern: false
          - Type: ACTOR_ACCOUNT_ID
            Pattern: 12345
            ExcludeMatchedPattern: true
        - - Type: EVENT
            Pattern: PUSH
          - Type: HEAD_REF
            Pattern: ^refs/heads/.*
          - Type: FILE_PATH
            Pattern: READ_ME
            ExcludeMatchedPattern: true
        - - Type: EVENT
            Pattern: PUSH
          - Type: COMMIT_MESSAGE
            Pattern: \[CodeBuild\]
          - Type: FILE_PATH
            Pattern: ^src/.+|^test/.+
        - - Type: EVENT
            Pattern: WORKFLOW_JOB_QUEUED
          - Type: WORKFLOW_NAME
            Pattern: \[CI-CodeBuild\]
```

GitLab 群组 webhook

您可以使用 CodeBuild GitLab 群组 webhook 在群组内任何存储库的 webhook 事件上启动构建。GitLab 群组 webhook 适用于任何现有的 GitLab webhook 事件类型，并且可以通过在创建 webhook

时添加范围配置来进行配置。 CodeBuild 您还可以使用群组 webhook 在其中设置自托管的 GitLab运行器，以便 CodeBuild在单个项目中接收来自多个存储库WORKFLOW_JOB_QUEUED的事件。

主题

- 设置群组 GitLab webhook
- 筛选 GitLab 群组 webhook 事件（控制台）
- 筛选 GitLab 群组 webhook 事件 (CloudFormation)

设置群组 GitLab webhook

设置群组 GitLab webhook 的高级步骤如下。有关群组 GitLab webhook 的更多信息，请参阅GitLab 群组 webhook。

1. 将项目的源位置设置为 CODEBUILD_DEFAULT_WEBHOOK_SOURCE_LOCATION。
2. 在 webhook 的范围配置中，将范围设置为 GITLAB_GROUP。
3. 在 webhook 的范围配置过程中指定一个名称。对于组 webhook，这是组名称。

> **Note**
>
> 如果项目的源类型为 GITLAB_SELF_MANAGED，则还需要在 webhook 范围配置过程中指定一个域。

4. （可选）如果您只想接收组织或企业内特定存储库的 webhook 事件，则可以在创建 webhook 时将 REPOSITORY_NAME 指定为筛选条件。
5. 创建群组 webhook 时，请确保该群组 CodeBuild 有权在群组内创建群组 Webhook。 GitLab但要做到这一点，你可以使用 CodeBuild OAuth。 AWS CodeConnections有关更多信息，请参阅 GitLab 进入 CodeBuild。

   请注意，群组 webhook 适用于任何现有的 GitLab webhook 事件类型。

筛选 GitLab 群组 webhook 事件（控制台）

通过控制台创建 GitLab 项目时，请选择以下选项在项目中创建 GitLab 群组 webhook。有关群组 GitLab webhook 的更多信息，请参阅GitLab 群组 webhook。

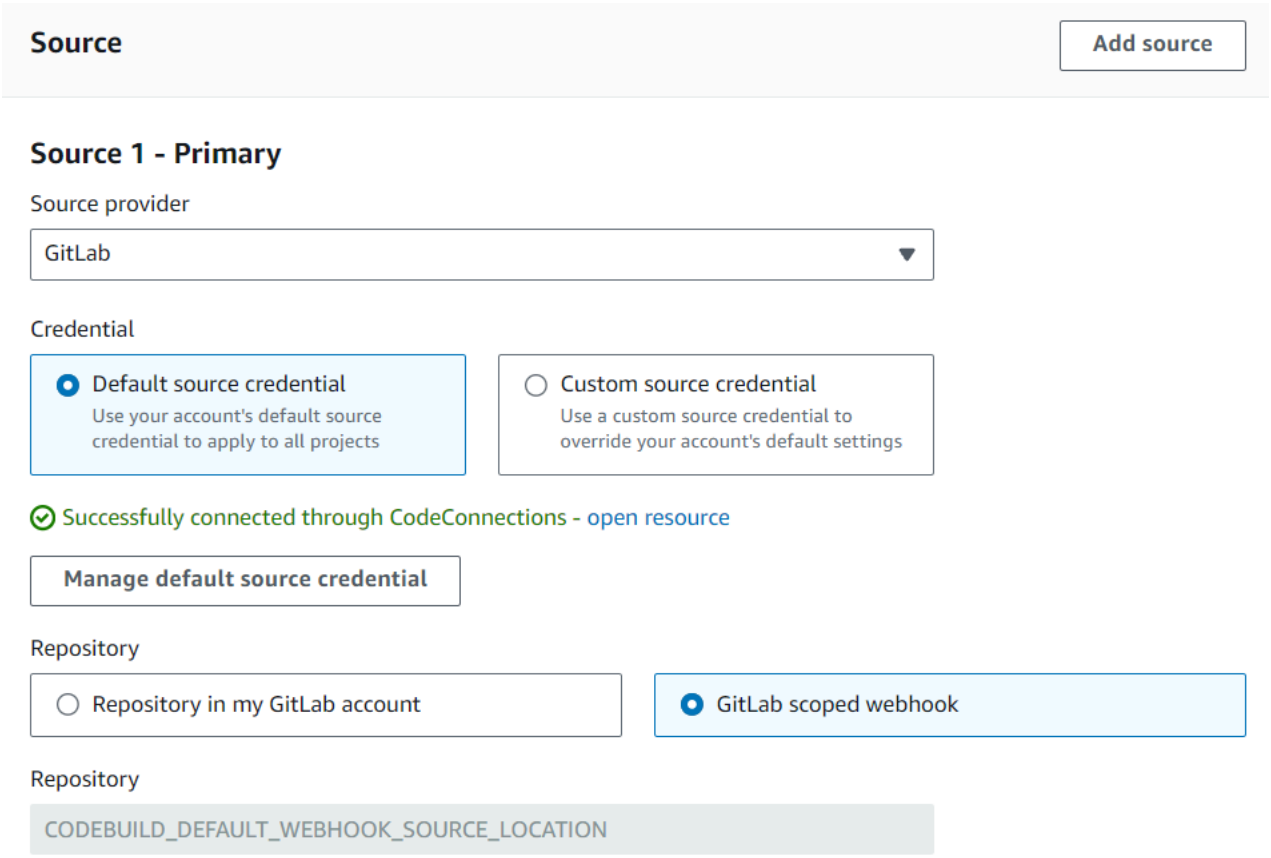
1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。

2. 创建构建项目。有关信息，请参阅创建构建项目（控制台）和运行构建（控制台）。

   - 在源中：
     - 对于源提供商，请选择GitLab或GitLab自行管理。
     - 对于存储库，选择GitLab作用域化的 webh ook。

       GitLab 存储库将自动设置为`CODEBUILD_DEFAULT_WEBHOOK_SOURCE_LOCATION`，这是群组 webhook 所需的源位置。

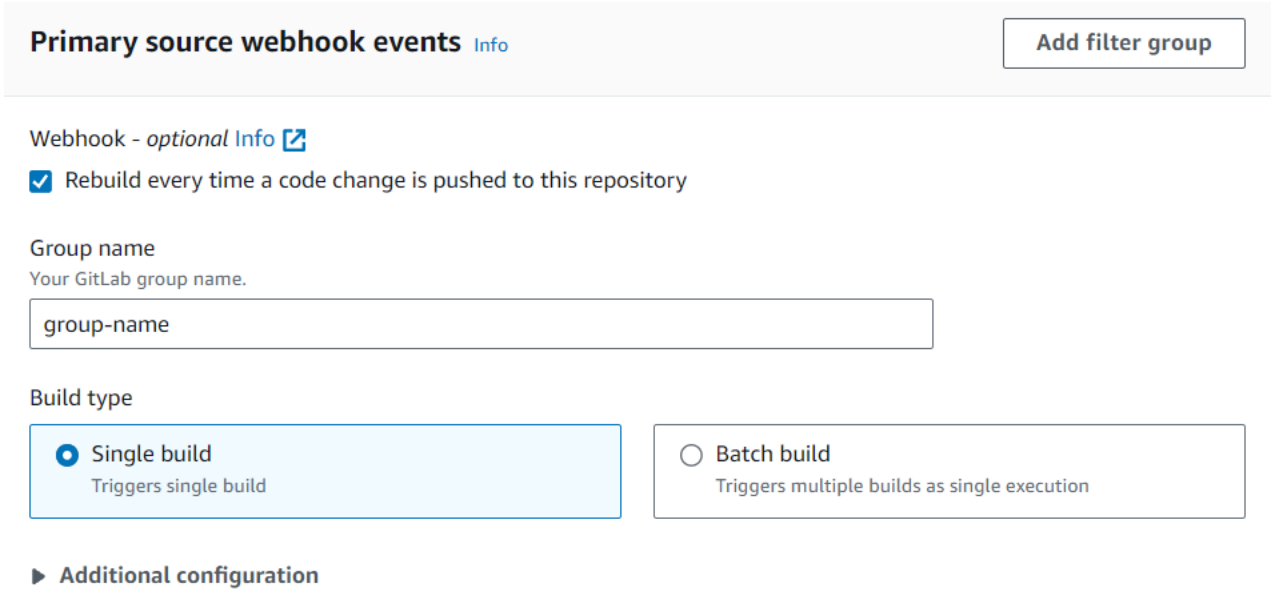
       > **Note**
       >
       > 使用群组 webhook 时，请确保该群组 CodeBuild 有权在群组内创建群组级 webhook。 GitLab如果您使用的是现有的 OAuth 连接，则可能需要重新生成连接才能授予此权限 CodeBuild 。



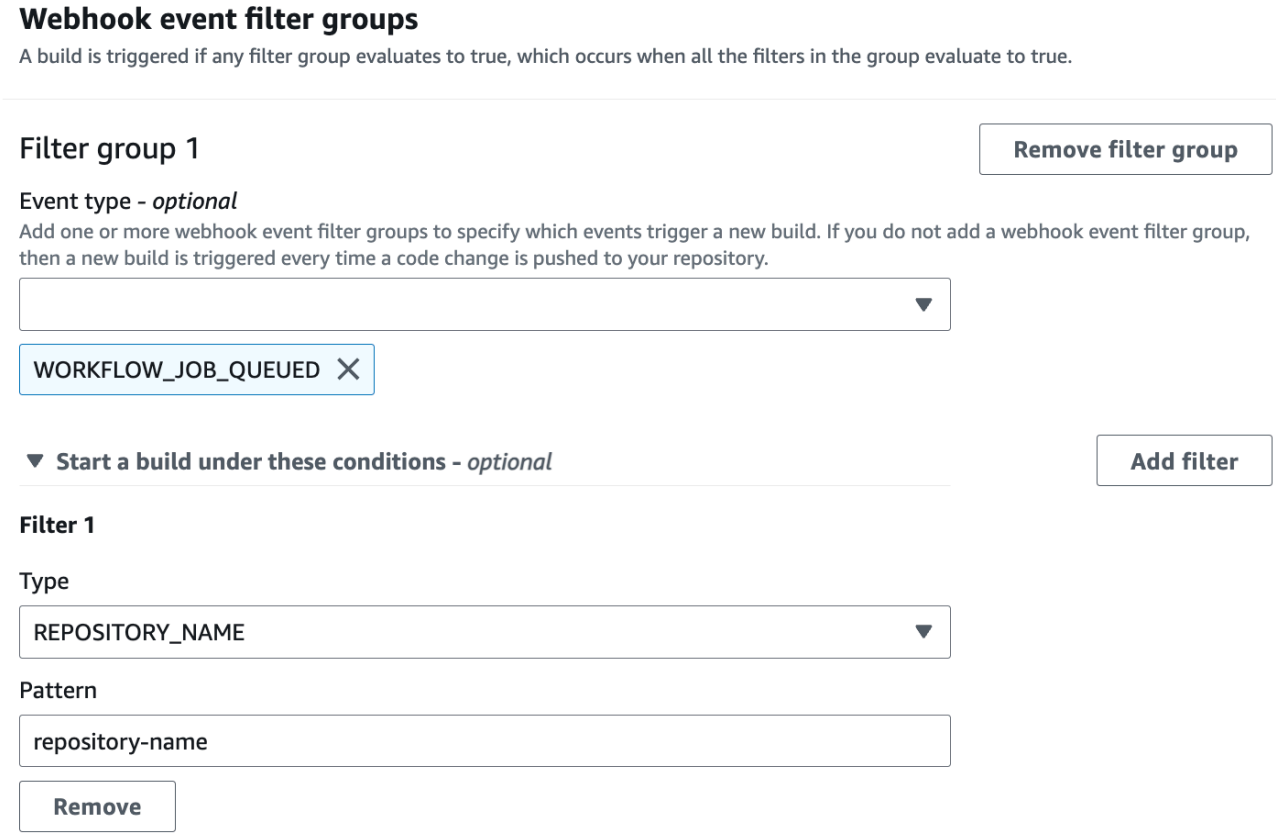
   - 在主要源 Webhook 事件中：
     - 在组名称中，输入组名称。

如果项目的源类型为 GITLAB_SELF_MANAGED，则还需要在 webhook 组配置过程中指定一个域。例如，如果组的 URL 是 **https://domain.com/group/group-name**，则域是 **https://domain.com**。

> **Note**
>
> 创建 webhook 后不能更改此名称。要更改名称，您可以删除并重新创建 webhook。如果要完全移除 webhook，也可以将项目源位置更新为 GitLab存储库。



- （可选）在 webhook 事件筛选条件组中，您可以指定要触发新构建的事件。您也可以指定 REPOSITORY_NAME 作为筛选条件，仅根据来自特定存储库的 webhook 事件触发构建。



您也可以将事件类型设置为`WORKFLOW_JOB_QUEUED`以设置自托管的 GitLab 运行器。有关更多信息，请参阅 Self-managed GitLab 跑步者进来 AWS CodeBuild。

3. 继续使用默认值，然后选择创建构建项目。

筛选 GitLab 群组 webhook 事件 (CloudFormation)

要使用 CloudFormation 模板筛选群组 webhook 事件，请使用 AWS CodeBuild 项目的`ScopeConfiguration`属性。有关群组 GitLab webhook 的更多信息，请参阅GitLab 群组 webhook。

CloudFormation 模板的以下 YAML-formatted 部分创建了四个筛选器组。当这些筛选条件组的其中一个或全部评估为 True 时触发构建：

- 第一个筛选器组指定`^refs/heads/main$`由没有账户 ID 的 GitLab 用户在具有与正则表达式匹配的 Git 引用名称的分支上创建或更新拉取请求`12345`。
- 第二个筛选条件组使用与正则表达式 `^refs/heads/.*` 匹配的 Git 引用名称，指定在名称与正则表达式 `READ_ME` 匹配的文件上创建的推送请求。

- 第三个筛选条件组指定一个推送请求，其中包含与正则表达式 \[CodeBuild\] 匹配的 HEAD 提交消息。
- 第四个筛选器组指定 GitLab CI/CD 管道作业请求，其 CI/CD 管道名称与正则表达式匹配\[CI-CodeBuild\]。

```
CodeBuildProject:
  Type: AWS::CodeBuild::Project
  Properties:
    Name: MyProject
    ServiceRole: service-role
    Artifacts:
      Type: NO_ARTIFACTS
    Environment:
      Type: LINUX_CONTAINER
      ComputeType: BUILD_GENERAL1_SMALL
      Image: aws/codebuild/standard:5.0
    Source:
      Type: GITLAB
      Location: source-location
    Triggers:
      Webhook: true
      ScopeConfiguration:
        Name: group-name
        Scope: GITLAB_GROUP
      FilterGroups:
        - - Type: EVENT
            Pattern: PULL_REQUEST_CREATED,PULL_REQUEST_UPDATED
          - Type: BASE_REF
            Pattern: ^refs/heads/main$
            ExcludeMatchedPattern: false
          - Type: ACTOR_ACCOUNT_ID
            Pattern: 12345
            ExcludeMatchedPattern: true
        - - Type: EVENT
            Pattern: PUSH
          - Type: HEAD_REF
            Pattern: ^refs/heads/.*
          - Type: FILE_PATH
            Pattern: READ_ME
            ExcludeMatchedPattern: true
        - - Type: EVENT
            Pattern: PUSH
```

```
          - Type: COMMIT_MESSAGE
            Pattern: \[CodeBuild\]
          - Type: FILE_PATH
            Pattern: ^src/.+|^test/.+
        - - Type: EVENT
            Pattern: WORKFLOW_JOB_QUEUED
          - Type: WORKFLOW_NAME
            Pattern: \[CI-CodeBuild\]
```

GitLab 手动网络挂钩

您可以配置手动 GitLab webhook 以 CodeBuild 防止自动尝试在其中创建 webhook。 GitLab CodeBuild 返回负载网址作为创建 webhook 调用的一部分，该负载网址可用于在其中手动创建 webhook。 GitLab即使未 CodeBuild 被列入在您的 GitLab 账户中创建 webhook 的许可名单，您仍然可以为构建项目手动创建 webhook。

使用以下步骤创建 GitLab 手动 webhook。

创建手 GitLab 动 webhook

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 创建构建项目。有关信息，请参阅创建构建项目（控制台）和运行构建（控制台）。
   - 在源中：
     - 对于来源提供商，选择GitLab。
     - 对于 “存储库”，选择 “我的 GitLab 账户中的存储库” 。
     - 对于存储库 URL，输入 **https://gitlab.com/*user-name*/*repository-name***。
   - 在主要源 Webhook 事件中：
     - 对于 webhook - 可选，选择每次将代码更改推送到此存储库时都会重新构建。
     - 选择其他配置，对于手动创建-可选，选择在 GitLab 控制台中手动为该存储库创建 Webhook。。
3. 继续使用默认值，然后选择创建构建项目。请记下有效载荷 URL 和密钥值，因为稍后要用到它们。
4. 在处打开 GitLab 控制台，https://gitlab.com/*user-name*/*repository-name*/-/hooks然后选择添加新 webhook 。
   - 对于 URL，输入之前记下的有效载荷 URL 值。
   - 在密钥令牌中，输入之前记下的密钥值。

- 配置将向发送 webhook 负载的各个事件。 CodeBuild对于触发器，请从以下事件中进行选择：推送事件、合并请求事件、发布事件和作业事件。要了解有关支持的事件类型的更多信息 CodeBuild，请参阅GitLab webhook 事件。

5. 选择添加 webhook。

GitLab webhook 事件

您可以使用 webhook 过滤器组来指定哪些 GitLab webhook 事件触发构建。例如，您可以指定仅在对特定分支做出更改时触发构建。

您可以创建一个或多个 Webhook 筛选条件组，来指定哪些 Webhook 事件触发构建。如果任何筛选条件组评估为 true（即组中的所有筛选条件都评估为 true），则会触发构建。创建筛选条件组时，应指定：

事件

对于 GitLab，您可以选择以下一个或多个事件：PUSH、PULL_REQUEST_CREATED、PULL_REQUEST_UPDATED、PULL_REQUEST_MERGED、PULL_和WORKFLOW_JOB_QUEUED。

webhook 的事件类型位于其在 X-GitLab-Event 字段中的标头中。下表显示了 X-GitLab-Event 标头值如何映射到事件类型。对于 Merge Request Hook webhook 事件，有效载荷的 object_atttributes.action 将包含有关合并请求类型的更多信息。

X-GitLab-Event 标头值	**object_atttributes .action**	事件类型
Push Hook	N/A	PUSH
Merge Request Hook	打开	PULL_REQUEST_CREATED
Merge Request Hook	更新	PULL_REQUEST_UPDATED
Merge Request Hook	merge	PULL_REQUEST_MERGED
Merge Request Hook	重新打开	PULL_REQUEST_REOPENED
Merge Request Hook	关闭	PULL_REQUEST_CLOSED

X-GitLab-Event 标头值	object_atttributes .action	事件类型
Release Hook	创建、更新	RELEASED
Job Hook	N/A	WORKFLOW_JOB_QUEUED

对于 PULL_REQUEST_MERGED，如果拉取请求与压缩策略合并且拉取请求分支已关闭，则原始的拉取请求提交将不再存在。在这种情况下，CODEBUILD_WEBHOOK_MERGE_COMMIT 环境变量包含压缩后的合并提交的标识符。

一个或多个可选筛选条件

使用正则表达式来指定筛选条件。对于触发构建的事件，组内与其关联的每个筛选条件都必须评估为 True。

CodeBuild 使用RE2正则表达式语法评估过滤器模式，该语法不支持向前看、向后看、反向引用或原子组。有关更多信息，请参阅 GitHub 网站上的RE2语法页面。

ACTOR_ACCOUNT_ID（控制台中的 ACTOR_ID）

当 GitLab 账户 ID 与正则表达式模式匹配时，webhook 事件会触发构建。此值显示在 Webhook 筛选条件负载中的 actor 对象的 account_id 属性中。

HEAD_REF

当头部引用与正则表达式模式（例如 refs/heads/branch-name 和 refs/tags/tag-name）匹配时，Webhook 事件会触发构建。HEAD_REF 筛选条件将评估分支或标签的 Git 引用名称。分支或标签名称显示在 Webhook 负载的 push 对象中的 new 对象的 name 字段中。对于拉取请求事件，分支名称显示在 Webhook 负载中的 source 对象的 branch 中的 name 字段中。

BASE_REF

当基础引用与正则表达式模式匹配时，Webhook 事件会触发构建。BASE_REF 筛选条件仅处理拉取请求事件（例如，refs/heads/branch-name）。BASE_REF 筛选条件评估分支的 Git 引用名称。分支名称显示在 branch 对象的 name 字段中，该对象位于 Webhook 负载的 destination 对象中。

FILE_PATH

当更改的文件的路径与正则表达式模式匹配时，Webhook 会触发构建。

> **Note**
>
> 根据推送或拉取请求中的前 100 个更改的文件对FILE_PATH过滤器进行评估。为确保按预期应用过滤器，请将更改的文件数量保持在该限制范围内。对于拉取请求事件，我们还建议使用拉取请求构建策略来控制哪些拉取请求可以启动构建。

`COMMIT_MESSAGE`

当 HEAD 提交消息与正则表达式模式匹配时，Webhook 会触发构建操作。

`WORKFLOW_NAME`

当工作流名称与正则表达式模式匹配时，Webhook 会触发构建操作。

> **Note**
>
> 你可以在仓库的 webhook 设置中找到 webhook 负载。 GitLab

主题

- 过滤 GitLab webhook 事件（控制台）
- 过滤 GitLab webhook 事件 (SDK)
- 过滤 GitLab webhook 事件 (CloudFormation)

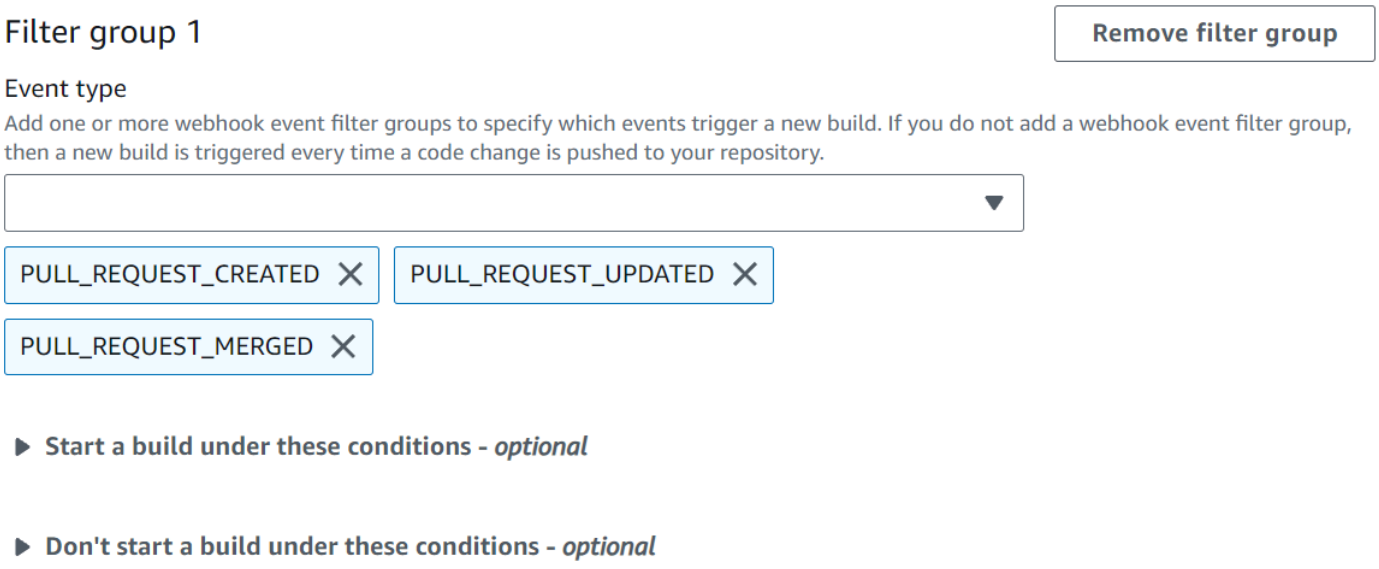
过滤 GitLab webhook 事件（控制台）

使用以下说明使用 AWS 管理控制台 来过滤 webhook 事件。有关 GitLab webhook 事件的更多信息，请参阅GitLab webhook 事件。

1. 创建项目时，选择每次将代码更改推送到此存储库时都会重新构建。
2. 从事件类型中，选择一个或多个事件。
3. 要在事件触发构建时进行筛选，请在在这些条件下开始构建下，添加一个或多个可选筛选条件。
4. 要在未触发事件时进行筛选，请在在这些条件下不开始构建下，添加一个或多个可选筛选条件。
5. 选择添加筛选条件组以添加另一个筛选条件组。

有关更多信息，请参阅《AWS CodeBuild API 参考》中的创建构建项目（控制台）和WebhookFilter。

在此示例中，Webhook 筛选条件组仅针对拉取请求触发构建：



以两个筛选条件组为例，当一个或两个筛选条件评估为 True 时触发构建：

- 第一个筛选条件组使用与正则表达式 `^refs/heads/main$` 匹配的 Git 引用名称以及与 `^refs/heads/branch1!` 匹配的 HEAD 引用，指定在分支上创建或更新的拉取请求。
- 第二个筛选条件组使用与正则表达式 `^refs/heads/branch1$` 匹配的 Git 引用名称，指定分支上的推送请求。

Webhook event filter groups

A build is triggered if any filter group evaluates to true, which occurs when all the filters in the group evaluate to true.

Filter group 1

Remove filter group

Event type

Add one or more webhook event filter groups to specify which events trigger a new build. If you do not add a webhook event filter group, then a new build is triggered every time a code change is pushed to your repository.

PULL_REQUEST_CREATED ✕ PULL_REQUEST_UPDATED ✕

▼ **Start a build under these conditions** - *optional*

Add filter

Filter 1

Type

HEAD_REF

Pattern

^refs/heads/branch1$

Remove

Filter 2

Type

BASE_REF

Pattern

^refs/heads/main

Remove

▶ **Don't start a build under these conditions** - *optional*

Filter group 2

Remove filter group

Event type

Add one or more webhook event filter groups to specify which events trigger a new build. If you do not add a webhook event filter group, then a new build is triggered every time a code change is pushed to your repository.

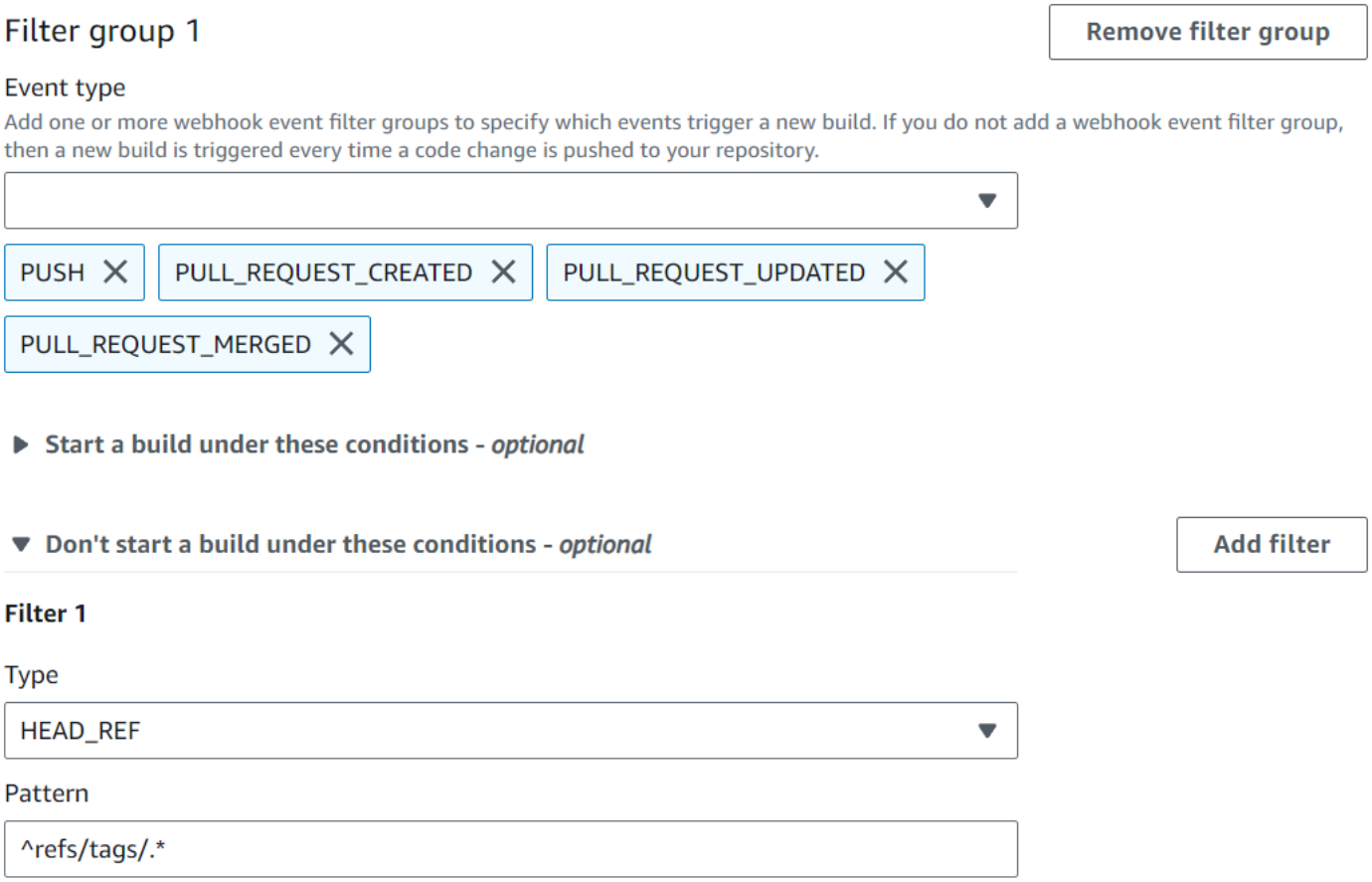
PUSH ✕

▼ **Start a build under these conditions** - *optional*

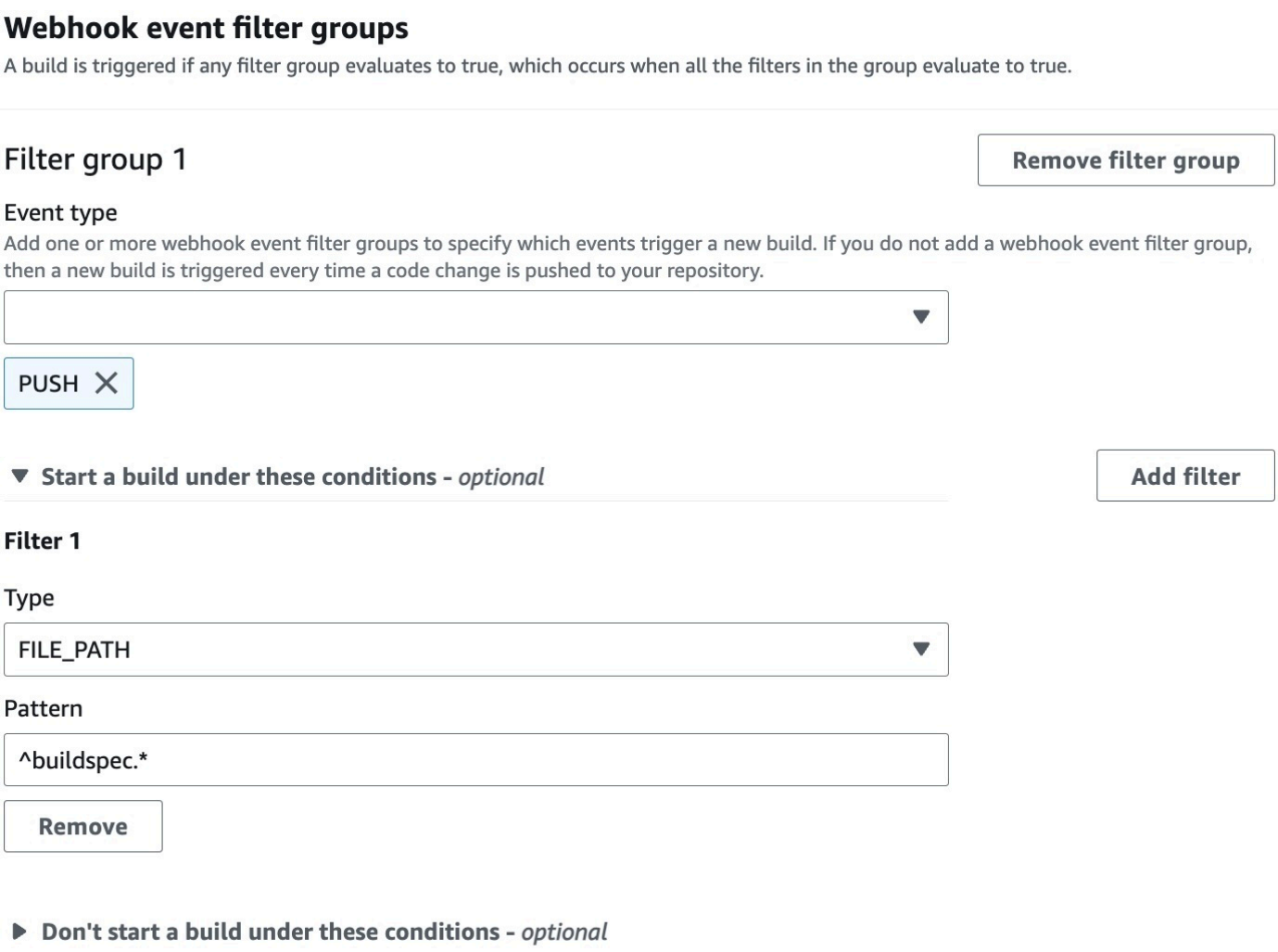
Add filter

Filter 1

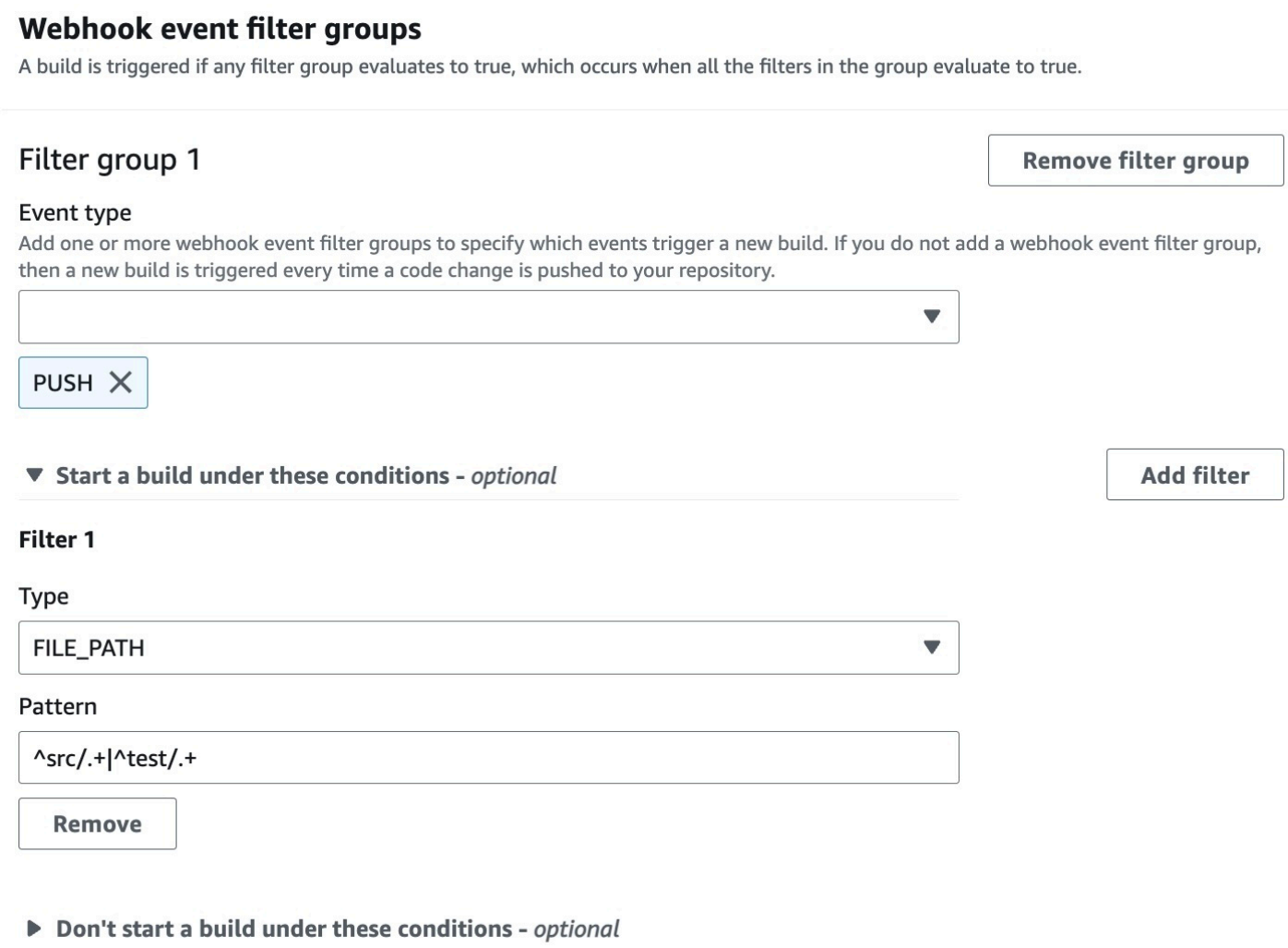
在此示例中，Webhook 筛选条件组会针对除标记事件之外的所有请求触发构建。



在此示例中，仅当名称与正则表达式 `^buildspec.*` 匹配的文件发生更改时，Webhook 筛选条件组才会触发构建。

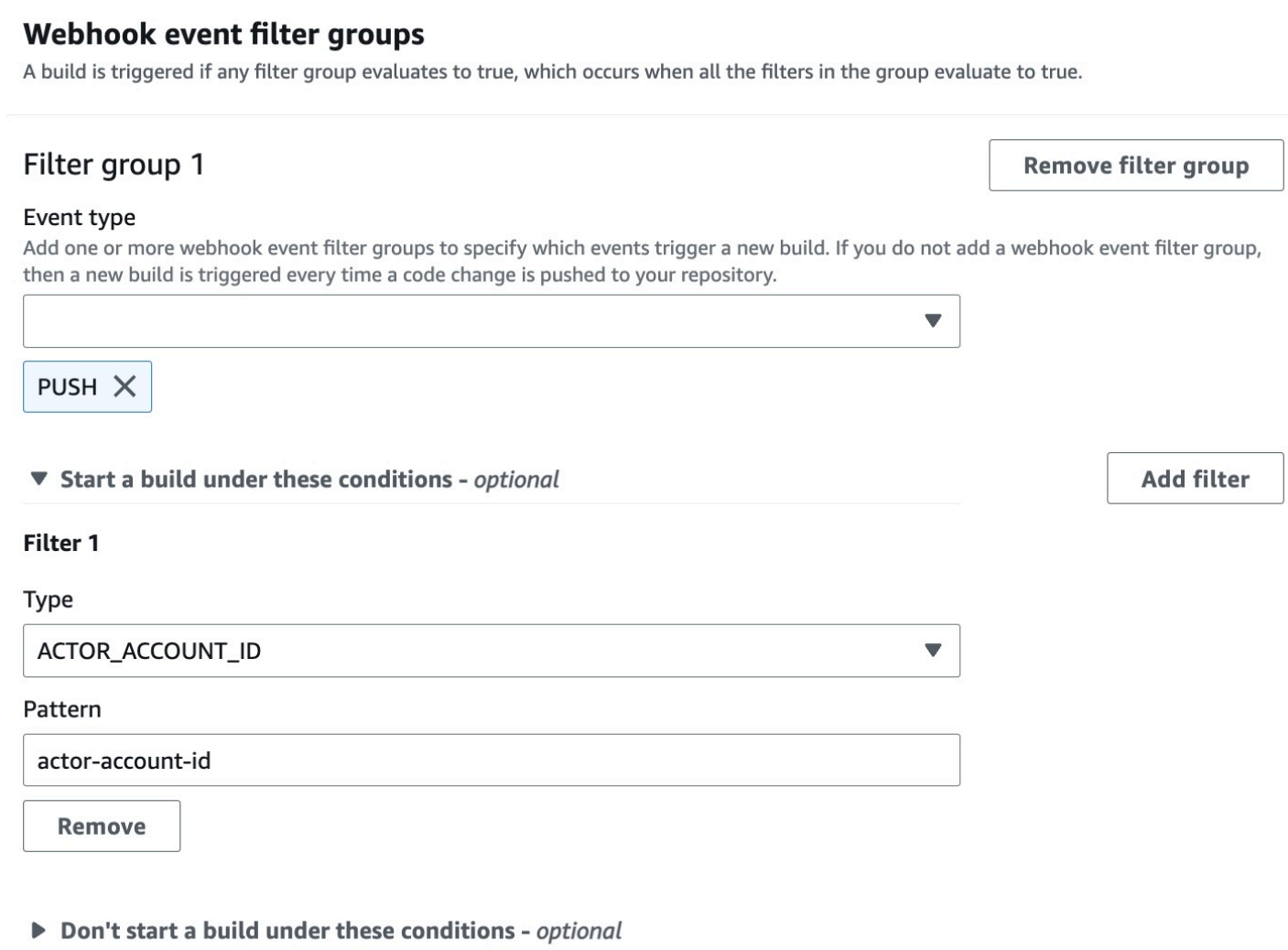


在此示例中，仅当 `src` 或 `test` 文件夹中的文件发生更改时，Webhook 筛选条件组才会触发构建。

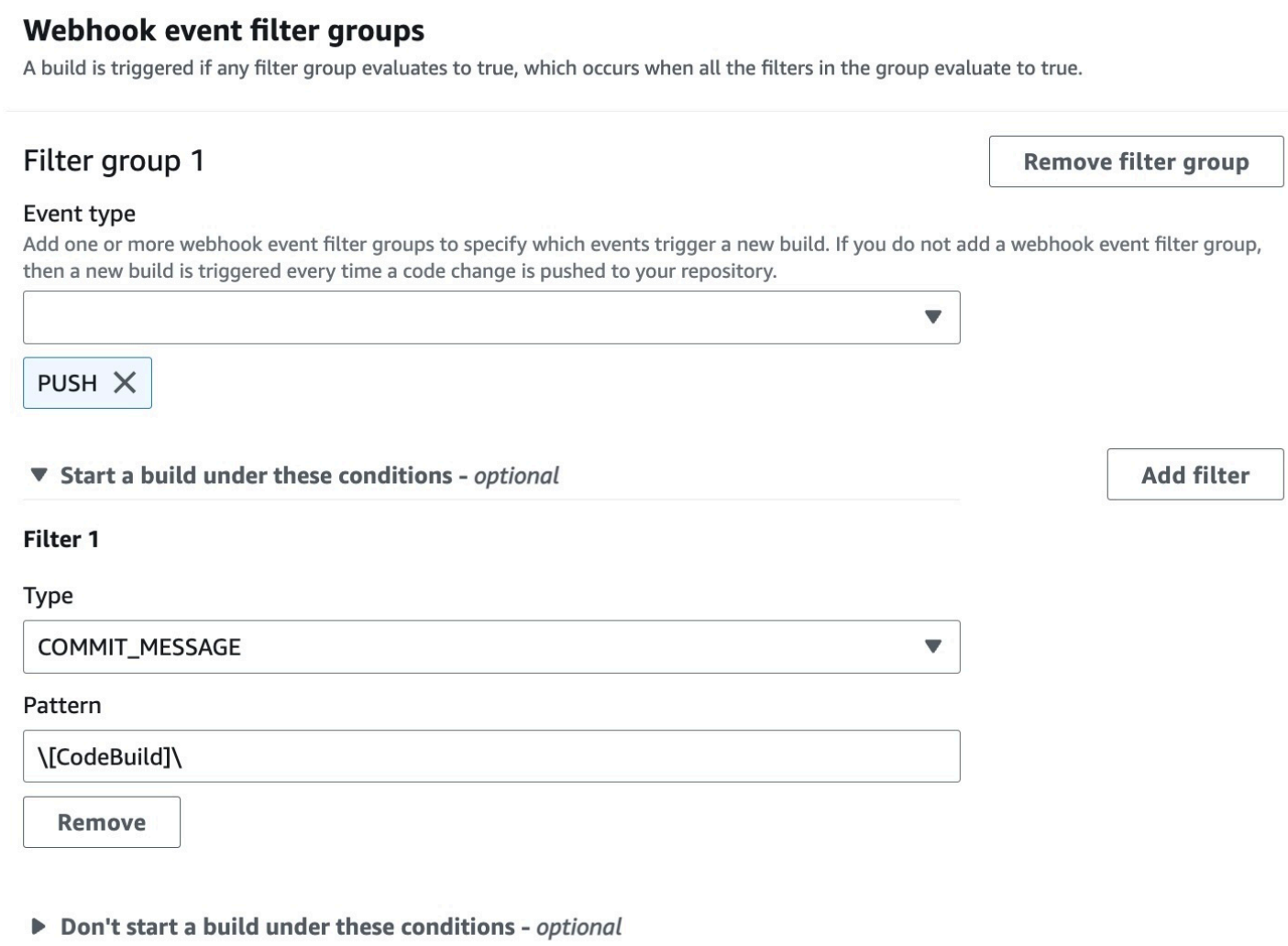


在此示例中，只有当账户 ID 与正则表达式`actor-account-id`不匹配的 GitLab 用户进行更改时，webhook 过滤器组才会触发构建。

> **Note**
>
> 有关如何查找您的 GitLab 账户 ID 的信息，请参阅 https://api.github.com/users/*`user-name`*，您的 GitLab 用户名在*`user-name`*哪里。



在本示例中，当 HEAD 提交消息与正则表达式 `\[CodeBuild\]` 匹配时，Webhook 筛选条件组会触发推送事件的构建。



过滤 GitLab webhook 事件 (SDK)

要使用 AWS CodeBuild 软件开发工具包筛选 webhook 事件，请使用`CreateWebhook`或 `UpdateWebhook` API 方法的请求语法中的`filterGroups`字段。有关更多信息，请参阅《CodeBuild API Reference》中的 WebhookFilter。

有关 GitLab webhook 事件的更多信息，请参阅GitLab webhook 事件。

要创建仅针对拉取请求触发构建的 Webhook 筛选条件，请在请求语法中插入以下内容：

```
"filterGroups": [
  [
    {
      "type": "EVENT",
      "pattern": "PULL_REQUEST_CREATED, PULL_REQUEST_UPDATED, PULL_REQUEST_MERGED"
```

```
    }
  ]
]
```

要创建仅针对指定分支触发构建的 Webhook 筛选条件，请使用 pattern 参数指定用于筛选分支名称的正则表达式。以两个筛选条件组为例，当一个或两个筛选条件评估为 True 时触发构建：

- 第一个筛选条件组使用与正则表达式 ^refs/heads/main$ 匹配的 Git 引用名称以及与 ^refs/heads/myBranch$ 匹配的 HEAD 引用，指定在分支上创建或更新的拉取请求。
- 第二个筛选条件组使用与正则表达式 ^refs/heads/myBranch$ 匹配的 Git 引用名称，指定分支上的推送请求。

```
"filterGroups": [
  [
    {
      "type": "EVENT",
      "pattern": "PULL_REQUEST_CREATED, PULL_REQUEST_UPDATED"
    },
    {
      "type": "HEAD_REF",
      "pattern": "^refs/heads/myBranch$"
    },
    {
      "type": "BASE_REF",
      "pattern": "^refs/heads/main$"
    }
  ],
  [
    {
      "type": "EVENT",
      "pattern": "PUSH"
    },
    {
      "type": "HEAD_REF",
      "pattern": "^refs/heads/myBranch$"
    }
  ]
]
```

您可以使用 excludeMatchedPattern 参数指定不触发构建的事件。在此示例中，将针对除标记事件之外的所有请求触发构建。

```
"filterGroups": [
  [
    {
      "type": "EVENT",
      "pattern": "PUSH, PULL_REQUEST_CREATED, PULL_REQUEST_UPDATED,
 PULL_REQUEST_MERGED"
    },
    {
      "type": "HEAD_REF",
      "pattern": "^refs/tags/.*",
      "excludeMatchedPattern": true
    }
  ]
]
```

您可以创建过滤器，该过滤器仅在拥有账户 ID 的 GitLab用户进行更改时才触发构建`actor-account-id`。

> **Note**
>
> 有关如何查找您的 GitLab 账户 ID 的信息，请参阅 https://api.github.com/users/*user-name*，您的 GitLab 用户名在*user-name*哪里。

```
"filterGroups": [
  [
    {
      "type": "EVENT",
      "pattern": "PUSH, PULL_REQUEST_CREATED, PULL_REQUEST_UPDATED,
 PULL_REQUEST_MERGED"
    },
    {
      "type": "ACTOR_ACCOUNT_ID",
      "pattern": "actor-account-id"
    }
  ]
]
```

您可以创建只有当名称与 `pattern` 参数中的正则表达式匹配的文件发生更改时，才触发构建的筛选条件。在此示例中，筛选条件组指定仅当名称与正则表达式 `^buildspec.*` 匹配的文件更改时才触发构建。

```
"filterGroups": [
  [
    {
      "type": "EVENT",
      "pattern": "PUSH"
    },
    {
      "type": "FILE_PATH",
      "pattern": "^buildspec.*"
    }
  ]
]
```

在此示例中，筛选条件组指定仅当 src 或 test 文件夹中的文件发生更改时，才会触发构建。

```
"filterGroups": [
    [
        {
            "type": "EVENT",
            "pattern": "PUSH"
        },
        {
            "type": "FILE_PATH",
            "pattern": "^src/.+|^test/.+"
        }
    ]
]
```

您可以创建一个筛选条件，仅当 HEAD 提交消息与模式参数中的正则表达式匹配时才触发构建操作。在本示例中，筛选条件组指定仅当推送事件的 HEAD 提交消息与正则表达式 \[CodeBuild\] 匹配时，才触发构建操作。

```
  "filterGroups": [
    [
      {
        "type": "EVENT",
        "pattern": "PUSH"
      },
      {
        "type": "COMMIT_MESSAGE",
        "pattern": "\[CodeBuild\]"
      }
```

```
    ]
  ]
```

过滤 GitLab webhook 事件 (CloudFormation)

要使用 CloudFormation 模板过滤 webhook 事件，请使用 AWS CodeBuild 项目的`FilterGroups`属性。有关 GitLab webhook 事件的更多信息，请参阅GitLab webhook 事件。

CloudFormation 模板的以下 YAML-formatted 部分创建了两个过滤器组。当这两个筛选条件的其中一个或两个评估为 True 时触发构建：

- 第一个过滤器组指定拉取请求是由没有账户 ID 的 GitLab 用户在具有与正则表达式`^refs/heads/main$`相匹配的 Git 引用名称的分支上创建或更新的`12345`。
- 第二个筛选条件组使用与正则表达式 `^refs/heads/.*` 匹配的 Git 引用名称，指定在分支上创建的推送请求。
- 第三个筛选条件组指定一个推送请求，其中包含与正则表达式 `\[CodeBuild\]` 匹配的 HEAD 提交消息。
- 第四个筛选组使用与正则表达式匹配的工作流名称来指定 GitHub 操作工作流任务请求`\[CI-CodeBuild\]`。

```
CodeBuildProject:
  Type: AWS::CodeBuild::Project
  Properties:
    Name: MyProject
    ServiceRole: service-role
    Artifacts:
      Type: NO_ARTIFACTS
    Environment:
      Type: LINUX_CONTAINER
      ComputeType: BUILD_GENERAL1_SMALL
      Image: aws/codebuild/standard:5.0
    Source:
      Type: GITLAB
      Location: source-location
    Triggers:
      Webhook: true
      FilterGroups:
        - - Type: EVENT
            Pattern: PULL_REQUEST_CREATED,PULL_REQUEST_UPDATED
          - Type: BASE_REF
```

```
            Pattern: ^refs/heads/main$
            ExcludeMatchedPattern: false
          - Type: ACTOR_ACCOUNT_ID
            Pattern: 12345
            ExcludeMatchedPattern: true
        - - Type: EVENT
            Pattern: PUSH
          - Type: HEAD_REF
            Pattern: ^refs/heads/.*
        - - Type: EVENT
            Pattern: PUSH
          - Type: COMMIT_MESSAGE
            Pattern: \[CodeBuild\]
        - - Type: EVENT
            Pattern: WORKFLOW_JOB_QUEUED
          - Type: WORKFLOW_NAME
            Pattern: \[CI-CodeBuild\]
```

Buildkite 手动 webhook

目前，CodeBuild 需要手动创建所有 Buildkite 网络挂钩。 CodeBuild返回负载网址作为创建网络挂钩调用的一部分，该网址可用于在 Buildkite 中手动创建 webhook。

使用以下过程创建 Buildkite 手动 webhook。

使用 webhook 创建 CodeBuild 项目

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 创建构建项目。有关信息，请参阅创建构建项目（控制台）和运行构建（控制台）。
3. 在项目配置中，选择运行程序项目。

   在运行程序中：

   - 对于运行程序提供商，选择 Buildkite。
   - 对于 Buildkite 代理令牌，选择使用创建密钥页面创建新的代理令牌。系统将提示您在密钥管理器中 AWS 创建一个新密钥，其密钥值等于您在上面生成的 Buildkite 代理令牌。
   - （可选）如果您想为作业使用 CodeBuild 托管证书，请在 Buildkite 源凭据选项下选择作业的源存储库提供商，并验证是否已为您的账户配置证书。此外，请验证 Buildkite 管道是否采用使用 HTTPS 签出。
4. • 在环境中：

- 选择支持的环境映像和计算。请注意，您可以选择在 Act GitHub ions 工作流程 YAML 中使用标签来覆盖图像和实例设置。有关更多信息，请参阅 第 2 步：更新您的 GitHub 操作工作流程 YAML。
- 在 Buildspec (构建规范) 中：
  - 请注意，除非将 `buildspec-override:true` 作为标签添加，否则系统会忽略 buildspec。相反， CodeBuild 将覆盖它以使用用于设置自托管运行器的命令。

5. 继续使用默认值，然后选择创建构建项目。
6. 保存创建 webhook 弹出窗口中的有效载荷 URL 和密钥值。按照弹出窗口中的说明创建新的 Buildkite 组织 webhook。

拉取请求评论批准

CodeBuild 支持拉取请求构建策略，为拉取请求触发的构建提供额外控制。您可能不想自动构建来自未知用户的拉取请求，直到可以审查其更改。此功能可让您要求一名团队成员先审查代码，然后运行管道。在构建未知贡献者提交的代码时，这通常用作安全措施。

拉取请求构建政策允许您根据贡献者的权限和批准状态控制何时为拉取请求生成 CodeBuild 触发器。这对于公共存储库或接受外部协作者贡献的存储库尤为重要。

启用后，此功能可确保只有在以下任一情况下，才会为拉取请求触发构建：

- 拉取请求由可信贡献者创建。
- 可信贡献者通过发布特定评论来批准拉取请求。

工作原理

可信贡献者

可信贡献者是指其当前在源代码控制系统中的角色在基于拉取请求的策略中设置为批准者角色的用户。当可信贡献者创建拉取请求时，会自动 CodeBuild 触发构建，保持当前行为。

不可信贡献者

不可信贡献者是指其角色未在批准者角色列表中设置的用户。当不可信的贡献者创建拉取请求时：

1. CodeBuild 将构建状态标记为 “失败”，并显示消息 “启动构建需要拉取请求批准”。

2. 可信贡献者必须审核更改并使用 /codebuild_run(*<SHA_OF_THE_LATEST_COMMIT>*) 发布评论才能触发构建。例如 /codebuild_run(*046e8b67481d53bdc86c3f6affdd5d1afae6d369*)。
3. CodeBuild 验证评论者的权限，如果获得批准，则触发构建。
4. 构建结果将在拉取请求页面上报告。

评论批准语法

可信贡献者可以使用以下评论格式批准构建：

- /codebuild_run(*046e8b67481d53bdc86c3f6affdd5d1afae6d369*)：基于指定的提交 SHA 触发构建。

配置

默认 行为

默认情况下，所有新创建的 CodeBuild 项目均启用拉取请求生成策略。

API 参数

拉取请求构建策略是在以下操作中使用 PullRequestBuildPolicy 参数配置的：

- CreateWebhook
- UpdateWebhook

PullRequestBuildPolicy 结构

```
{
    "requiresCommentApproval": "string",
    "approverRoles": ["string", ...]
}
```

requiresCommentApproval

指定在对拉取请求触发构建之前，何时需要基于评论的批准。此设置决定构建是自动运行，还是需要通过评论进行明确批准。

类型：字符串

有效值：

- DISABLED：无需评论批准即可自动触发构建。
- FORK_PULL_REQUESTS：只有来自分叉存储库的拉取请求才需要评论批准（除非贡献者是批准者角色之一）。
- ALL_PULL_REQUESTS：所有拉取请求在构建执行之前都需要评论批准（除非贡献者是批准者角色之一）。这是默认值。

approverRoles

当需要评论批准时，对拉取请求构建具有批准权限的存储库角色列表。只有拥有这些角色的用户才能提供有效的评论批准。如果拉取请求贡献者是这些角色之一，则其拉取请求构建将自动触发。

类型：字符串数组

GitHub 项目的有效值（值映射到 GitHub 角色）：

- GITHUB_ADMIN：存储库管理员
- GITHUB_MAINTAIN：存储库维护者
- GITHUB_WRITE：具有写入权限的用户
- GITHUB_TRIAGE：具有分类权限的用户
- GITHUB_READ：具有读取权限的用户
- 默认值：["GITHUB_ADMIN", "GITHUB_MAINTAIN", "GITHUB_WRITE"]

GitLab 项目的有效值（值映射到 GitLab 角色）：

- GITLAB_OWNER：存储库所有者
- GITLAB_MAINTAINER：存储库维护者
- GITLAB_DEVELOPER：拥有开发人员权限的用户
- GITLAB_REPORTER：具有报告者权限的用户
- GITLAB_PLANNER：拥有计划者权限的用户
- GITLAB_GUEST ：拥有访客权限的用户
- 默认值：["GITLAB_OWNER", "GITLAB_MAINTAINER", "GITLAB_DEVELOPER"]

Bitbucket 项目的有效值（这些值映射到 Bitbucket 角色）：

- BITBUCKET_ADMIN ：存储库管理员
- BITBUCKET_WRITE：具有写入权限的用户
- BITBUCKET_READ ：具有读取权限的用户

- 默认值：["BITBUCKET_ADMIN", "BITBUCKET_WRITE"]

自定义 GitHub 企业角色

CodeBuild 根据授予自定义角色的最高权限级别将自定义 GitHub 企业 (GHE) 存储库角色映射到标准GITHUB_*值。 CodeBuild 从最高到最低权限评估权限，第一个启用的权限决定映射的角色。

- admin → GITHUB_ADMIN
- maintain → GITHUB_MAINTAIN
- push（写作）→ GITHUB_WRITE
- triage → GITHUB_TRIAGE
- pull（阅读）→ GITHUB_READ

例如，同时授予push和triage权限的自定义角色映射到，GITHUB_WRITE因为push其权限高于triage。

如果自定义角色没有任何可识别的权限（管理员、维护、推送、分类、全部禁用），则该用户与任何批准者角色都不匹配，被视为不可信的贡献者。

标准 GitHub 角色（管理员、维护、写入、分类、读取）直接映射到其相应的GITHUB_*值，无需基于权限的解析。

示例

为所有拉取请求启用评论批准

要使用 AWS CodeBuild 软件开发工具包启用或禁用 Webhook 的拉取请求生成策略，请使用CreateWebhook或 UpdateWebhook API 方法的请求语法中的pullRequestBuildPolicy字段。有关更多信息，请参阅《CodeBuild API Reference》中的 WebhookFilter。

GitHub 角色为 “管理员”、“维护” 和 “写入” 的用户将被视为可信贡献者。

```
"pullRequestBuildPolicy": {
    "requiresCommentApproval": "ALL_PULL_REQUESTS",
    "approverRoles": ["GITHUB_ADMIN", "GITHUB_MAINTAIN", "GITHUB_WRITE"]
}
```

仅为存储库管理员和维护者启用评论批准

具有 “管理员” 和 “维护” GitHub 角色的用户将被视为可信贡献者。

```
"pullRequestBuildPolicy": {
    "requiresCommentApproval": "FORK_PULL_REQUESTS",
    "approverRoles": ["GITHUB_ADMIN", "GITHUB_MAINTAIN"]
}
```

禁用评论批准

```
"pullRequestBuildPolicy": {
    "requiresCommentApproval": "DISABLED"
}
```

AWS CloudFormation

要使用 AWS CloudFormation 模板为 Webhook 启用或禁用 Pull Request Build 策略，请使用该 PullRequestBuildPolicy 属性。 AWS CloudFormation 模板的以下 YAML-formatted 部分使用 webhook 创建项目，该项目为所有拉取请求启用了拉取请求生成策略。维护和管理员角色被指定为批准者。

```
CodeBuildProject:
  Type: AWS::CodeBuild::Project
  Properties:
    Name: MyProject
    ServiceRole: service-role
    Artifacts:
      Type: NO_ARTIFACTS
    Environment:
      Type: LINUX_CONTAINER
      ComputeType: BUILD_GENERAL1_SMALL
      Image: aws/codebuild/standard:5.0
    Source:
      Type: BITBUCKET
      Location: source-location
    Triggers:
      Webhook: true
      FilterGroups:
        - - Type: EVENT
            Pattern: PULL_REQUEST_CREATED,PULL_REQUEST_UPDATED
          - Type: BASE_REF
            Pattern: ^refs/heads/main$
            ExcludeMatchedPattern: false
      PullRequestBuildPolicy:
        RequiresCommentApproval: ALL_PULL_REQUESTS
```

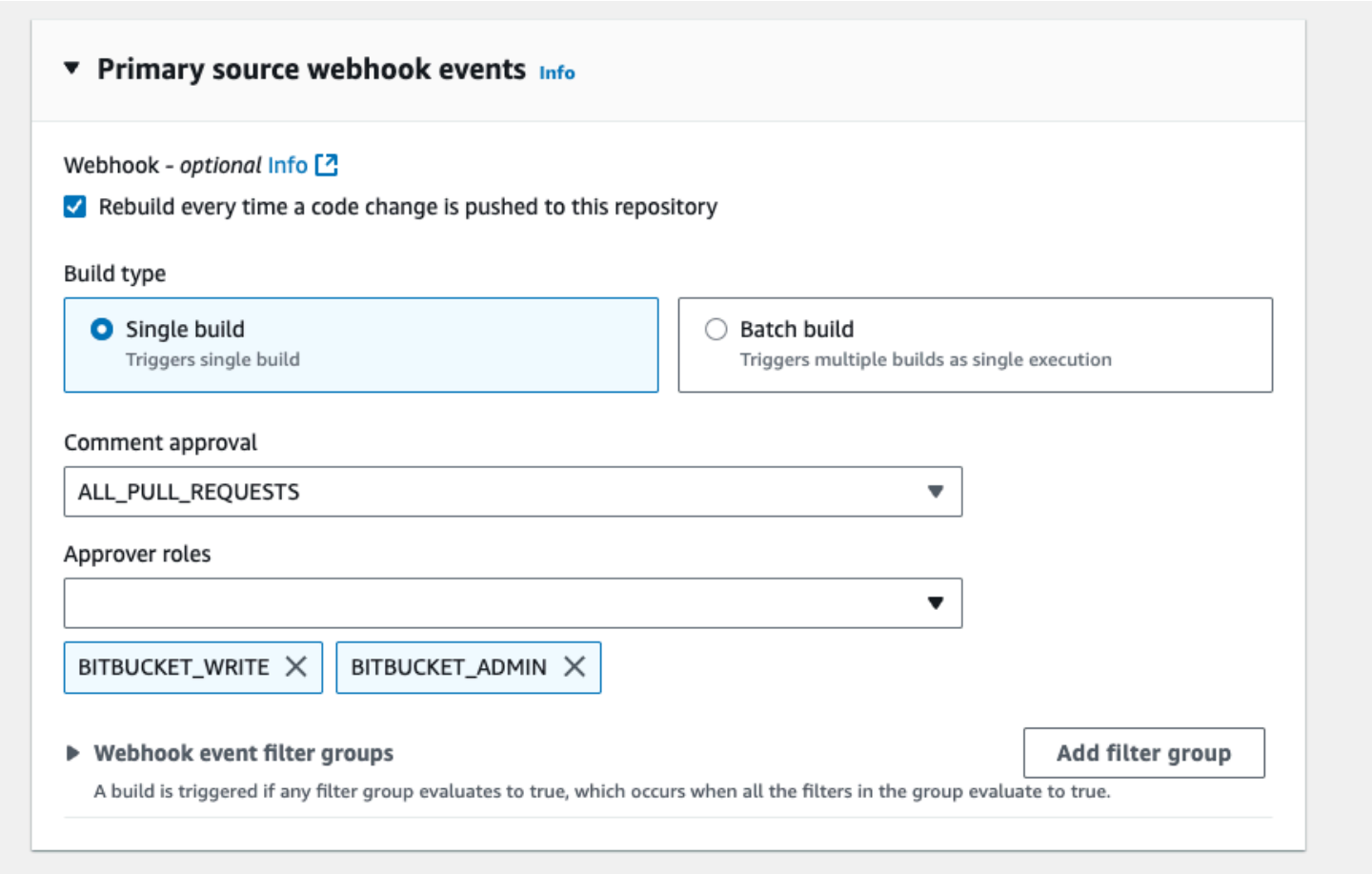
```
        ApproverRoles:
          - GITHUB_MAINTAIN
          - GITHUB_ADMIN
```

控制台配置

要使用 AWS 管理控制台筛选 webhook 事件，请执行以下操作：

1. 对于评论批准，请针对所有拉取请求 (ALL_PULL_REQUEST) 或仅针对来自分叉的拉取请求 (FORK_PULL_REQUEST) 选择已禁用或已启用。
2. 对于批准者角色，选择当需要评论批准时对拉取请求构建具有批准权限的存储库角色列表。

有关更多信息，请参阅《CodeBuild API 参考》中的创建构建项目（控制台）和WebhookFilter。



在中查看构建项目的详细信息 AWS CodeBuild

您可以使用 AWS CodeBuild 控制台或 AWS 软件开发工具包在中 CodeBuild查看构建项目的详细信息。 AWS CLI

主题

- 查看构建项目的详细信息（控制台）
- 查看构建项目的详细信息 (AWS CLI)
- 查看构建项目的详细信息 (AWS SDK)

查看构建项目的详细信息（控制台）

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择构建项目。

> **Note**
>
> 默认情况下，仅显示 10 个最新的构建项目。要查看更多构建项目，请选择齿轮图标，然后为每页项目数选择不同值，或使用向后和向前箭头。

3. 在构建项目列表中的名称列，选择构建项目的链接。
4. 在“生成项目：***project-name***”页面上，选择“生成详细信息”。

查看构建项目的详细信息 (AWS CLI)

运行 batch-get-projects 命令：

```
aws codebuild batch-get-projects --names names
```

在上述命令中，替换以下占位符：

- *names*：必填字符串，用于表示要查看其详细信息的一个或多个构建项目名称。要指定多个构建项目，请用空格分隔各个构建项目的名称。您最多可以指定 100 个构建项目名称。要获取构建项目的列表，请参阅查看构建项目名称的列表 (AWS CLI)。

例如，如果您运行此命令：

```
aws codebuild batch-get-projects --names codebuild-demo-project codebuild-demo-project2
 my-other-demo-project
```

与以下内容类似的结果可能会出现在输出中。为简洁起见，使用省略号 (...) 表示省略的数据。

```
{
  "projectsNotFound": [
    "my-other-demo-project"
  ],
  "projects": [
    {
      ...
      "name": codebuild-demo-project,
      ...
    },
    {
      ...
      "name": codebuild-demo-project2",
      ...
    }
  ]
}
```

在前面的输出中，projectsNotFound 数组列出了已指定但未找到的所有构建项目名称。projects 数组列出了可找到相关信息的所有构建项目的详细信息。为简洁起见，前面的输出中省略了构建项目的详细信息。有关更多信息，请参阅 创建构建项目 (AWS CLI) 的输出。

batch-get-projects 命令不支持筛选某些属性值，但您可以编写一个枚举项目属性的脚本。例如，以下 Linux shell 脚本枚举了当前账户在当前区域中的项目，并打印出每个项目使用的映像。

```
#!/usr/bin/sh

# This script enumerates all of the projects for the current account
# in the current region and prints out the image that each project is using.

imageName=""

function getImageName(){
  local environmentValues=(${1//$'\t'/ })
  imageName=${environmentValues[1]}
}

function processProjectInfo() {
  local projectInfo=$1
```

```
  while IFS=$'\t' read -r section value; do
    if [[ "$section" == *"ENVIRONMENT"* ]]; then
      getImageName "$value"
    fi
  done <<< "$projectInfo"
}

# Get the list of projects.
projectList=$(aws codebuild list-projects --output=text)

for projectName in $projectList
do
  if [[ "$projectName" != *"PROJECTS"* ]]; then
    echo "============================================="

    # Get the detailed information for the project.
    projectInfo=$(aws codebuild batch-get-projects --output=text --names
 "$projectName")

    processProjectInfo "$projectInfo"

    printf 'Project "%s" has image "%s"\n' "$projectName" "$imageName"
  fi
done
```

有关 AWS CLI 搭配使用的更多信息 AWS CodeBuild，请参阅命令行参考。

查看构建项目的详细信息 (AWS SDK)

有关与 AWS 软件开发工具包 AWS CodeBuild 一起使用的更多信息，请参阅。AWS SDK 和工具参考

在中查看构建项目名称 AWS CodeBuild

您可以使用 AWS CodeBuild 控制台或 AWS 软件开发工具包来查看中的 CodeBuild构建项目列表。AWS CLI

主题

- 查看构建项目名称的列表（控制台）
- 查看构建项目名称的列表 (AWS CLI)
- 查看构建项目名称 (AWS SDK) 列表

查看构建项目名称的列表（控制台）

您可以在控制台中查看某个 AWS 区域中的构建项目列表。信息包括名称、源提供程序、存储库、最新构建状态以及描述（如果有）。

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择构建项目。

> **Note**
>
> 默认情况下，仅显示 10 个最新的构建项目。要查看更多构建项目，请选择齿轮图标，然后为每页项目数选择不同值，或使用向后和向前箭头。

查看构建项目名称的列表 (AWS CLI)

运行 list-projects 命令：

```
aws codebuild list-projects --sort-by sort-by --sort-order sort-order --next-
token next-token
```

替换上一命令中的以下占位符：

- *sort-by*：用于指示用于列出构建项目名称的标准的可选字符串。有效值包括：
  - CREATED_TIME：根据每个构建项目的创建时间列出构建项目名称。
  - LAST_MODIFIED_TIME：根据每个构建项目信息上次更改的时间列出构建项目名称。
  - NAME：根据每个构建项目的名称列出构建项目名称。
- *sort-order*：可选字符串，用于指示列出生成项目的顺序*sort-by*。有效值包括 ASCENDING 和 DESCENDING。
- *next-token*: 可选字符串。在上次运行时，如果列表中有 100 个以上的项目，则只能返回前 100 个项目，以及名为下一个令牌的唯一字符串。要获取列表中的下一批项目，请再次运行此命令，将下一个令牌添加到调用中。要获取列表中的所有项目，请利用每个后续的下一个令牌运行此命令，直到不再返回下一个令牌。

例如，如果您运行此命令：

```
aws codebuild list-projects --sort-by NAME --sort-order ASCENDING
```

与以下内容类似的结果可能会出现在输出中：

```
{
  "nextToken": "Ci33ACF6...The full token has been omitted for brevity...U+AkMx8=",
  "projects": [
    "codebuild-demo-project",
    "codebuild-demo-project2",
    ... The full list of build project names has been omitted for brevity ...
    "codebuild-demo-project99"
  ]
}
```

如果您再次运行此命令：

```
aws codebuild list-projects  --sort-by NAME --sort-order ASCENDING --next-token
 Ci33ACF6...The full token has been omitted for brevity...U+AkMx8=
```

与以下内容类似的结果可能会出现在输出中：

```
{
  "projects": [
    "codebuild-demo-project100",
    "codebuild-demo-project101",
    ... The full list of build project names has been omitted for brevity ...
    "codebuild-demo-project122"
  ]
}
```

查看构建项目名称 (AWS SDK) 列表

有关与 AWS 软件开发工具包 AWS CodeBuild 一起使用的更多信息，请参阅。AWS SDK 和工具参考

内置 AWS CodeBuild

生成表示由执行的一组操作，这些操作 AWS CodeBuild 旨在基于一组输入构件（例如 Java 类文件集合）创建输出对象（例如 JAR 文件）。

运行多个构建时，以下规则适用：

- 如果可能，构建会同时运行。最大并发运行构建数会发生变化。有关更多信息，请参阅 的配额 AWS CodeBuild。
- 如果构建项目设置了并发构建限制，则正在运行的构建数量达到该项目的并发构建限制时，构建将返回错误。有关更多信息，请参阅启用并发构建限制。
- 如果构建项目未设置并发构建限制，则正在运行的构建数量达到该平台和计算类型的并发构建限制时，构建将排队。队列中的最大构建数为并发构建限制的 5 倍。有关更多信息，请参阅 的配额 AWS CodeBuild。

  从队列中删除在超时值中指定的分钟数后，不会启动队列中的构建。默认超时值为 8 小时。运行构建时，可以使用介于 5 分钟到 8 小时之间的值覆盖构建队列超时。有关更多信息，请参阅手动运行 AWS CodeBuild 构建。

  无法预测排队的构建的开始顺序。

> **Note**
>
> 您可以访问生成包一年的历史记录。

在使用构建时，您可以执行以下任务：

主题

- 手动运行 AWS CodeBuild 构建
- 在 build 上运行 AWS Lambda 计算
- 在预留容量实例集上运行构建
- 批量运行构建
- 在批量构建中执行并行测试
- 缓存构建以提高性能

- 调试内置 AWS CodeBuild
- 删除内部版本 AWS CodeBuild
- 重试手动构建 AWS CodeBuild
- 在中自动重试构建 AWS CodeBuild
- 停止构建 AWS CodeBuild
- 停止批量构建 AWS CodeBuild
- 触发器会自动 AWS CodeBuild 构建
- 在中查看版本详情 AWS CodeBuild
- 在中查看构建 ID 列表 AWS CodeBuild
- 在中查看构建项目的构建 ID 列表 AWS CodeBuild

手动运行 AWS CodeBuild 构建

您可以使用 AWS CodeBuild 控制台或 AWS 软件开发工具包来运行内部版本。 AWS CLI CodeBuild

主题

- 使用 AWS CodeBuild 代理在本地运行构建
- 运行构建（控制台）
- 运行构建 (AWS CLI)
- 运行批量构建 (AWS CLI)
- 开始自动运行构建（AWS CLI）
- 停止自动运行构建（AWS CLI）
- 运行构建 (AWS SDK)

使用 AWS CodeBuild 代理在本地运行构建

您可以使用 AWS CodeBuild 代理在本地计算机上运行 CodeBuild 构建。有适用于 x86_64 和 ARM 平台的代理。

您还可以进行订阅，这样便能在发布代理的新版本时收到通知。

先决条件

在开始之前，您需要执行以下操作：

- 在本地计算机上安装 Git。
- 在本地计算机上安装和设置 Docker。

设置构建映像

您只需要在首次运行代理时或映像发生更改时设置构建映像。

设置构建映像

1. 如果您想使用精心策划的 Amazon Linux 2 映像，则可以使用以下命令从 CodeBuild公共的 Amazon ECR 存储库中提取该映像：https://gallery.ecr.aws/codebuild/amazonlinux-x86_64-standard

   ```
   $ docker pull public.ecr.aws/codebuild/amazonlinux-x86_64-standard:4.0
   ```

   或者，如果要使用另一个 Linux 映像，请执行以下步骤：

   a. 克隆 CodeBuild 镜像存储库：

      ```
      $ git clone https://github.com/aws/aws-codebuild-docker-images.git
      ```

   b. 切换到该映像目录。本示例使用 `aws/codebuild/standard:5.0` 映像：

      ```
      $ cd aws-codebuild-docker-images/ubuntu/standard/5.0
      ```

   c. 构建映像。这将需要花几分钟的时间。

      ```
      $ docker build -t aws/codebuild/standard:5.0 .
      ```

2. 下载代 CodeBuild 理。

   要下载 x86_64 版本代理，请运行以下命令：

   ```
   $ docker pull public.ecr.aws/codebuild/local-builds:latest
   ```

   要下载 ARM 版本代理，请运行以下命令：

   ```
   $ docker pull public.ecr.aws/codebuild/local-builds:aarch64
   ```

3. CodeBuild 代理可从中获得https://gallery.ecr.aws/codebuild/local-builds。

x86_64 版本代理的安全哈希算法（SHA）签名为：

```
sha256:ccb19bdd7af94e4dc761e4c58c267e9455c28ec68d938086b4dc1cf8fe6b0940
```

ARM 版本代理的 SHA 签名为：

```
sha256:7d7b5d35d2ac4e062ae7ba8c662ffed15229a52d09bd0d664a7816c439679192
```

您可以通过此 SHA 识别代理的版本。要查看代理的 SHA 签名，请运行以下命令并在 `RepoDigests` 下查找 SHA：

```
$ docker inspect public.ecr.aws/codebuild/local-builds:latest
```

运行代 CodeBuild 理

运行代 CodeBuild 理

1. 请切换到包含构建项目源的目录。
2. 下载 codebuild_build.sh 脚本：

```
$ curl -O  https://raw.githubusercontent.com/aws/aws-codebuild-docker-images/
master/local_builds/codebuild_build.sh
$ chmod +x codebuild_build.sh
```

3. 运行 `codebuild_build.sh` 脚本并指定容器映像和输出目录。

   要运行 x86_64 构建，请运行以下命令：

   ```
   $ ./codebuild_build.sh -i <container-image> -a <output directory>
   ```

   要运行 ARM 构建，请运行以下命令：

   ```
   $ ./codebuild_build.sh -i <container-image> -a <output directory> -l
    public.ecr.aws/codebuild/local-builds:aarch64
   ```

   *<container-image>*替换为容器镜像的名称，例如`aws/codebuild/standard:5.0`或`public.ecr.aws/codebuild/amazonlinux-x86_64-standard:4.0`。

该脚本启动构建映像，并在当前目录中的项目上运行构建。要指定构建项目的位置，请在脚本命令中添加 `-s <build project directory>` 选项。

接收有关新 CodeBuild 代理版本的通知

您可以订阅 Amazon SNS 通知，以便在 AWS CodeBuild 代理发布新版本时收到通知。

订阅 CodeBuild 代理通知

1. 从 https://console.aws.amazon.com/sns/v3/home 打开 Amazon SNS 控制台。
2. 在导航栏中，如果尚未选择 AWS 该区域，则将其更改为美国东部（弗吉尼亚北部）。您必须选择此 AWS 区域，因为您订阅的 Amazon SNS 通知是在该区域创建的。
3. 在导航窗格中，选择订阅。
4. 选择创建订阅。
5. 在创建订阅中，请执行以下操作：

   a. 对于主题 ARN，请使用以下 Amazon 资源名称（ARN）：

   ```
   arn:aws:sns:us-east-1:850632864840:AWS-CodeBuild-Local-Agent-Updates
   ```

   b. 对于协议，选择电子邮件或 SMS。

   c. 对于端点，选择要接收通知的位置（电子邮件或 SMS）。输入电子邮件、地址或电话号码，包括区号。

   d. 选择创建订阅。

   e. 选择电子邮件，可接收要求确认订阅的电子邮件。按照电子邮件中的指示完成订阅。

   如果您不希望再收到这些通知，请通过以下步骤取消订阅。

取消订阅 CodeBuild 代理通知

1. 从 https://console.aws.amazon.com/sns/v3/home 打开 Amazon SNS 控制台。
2. 在导航窗格中，选择订阅。
3. 选择订阅，并从操作中，选择删除订阅。请在提示您进行确认时选择删除。

运行构建（控制台）

AWS CodePipeline 要使用运行构建 CodeBuild，请跳过这些步骤并按照中的说明进行操作 CodeBuild 搭配使用 CodePipeline。

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择构建项目。
3. 在构建项目列表中，选择构建项目。
4. 您可以使用默认的构建项目设置运行构建，也可以仅覆盖此构建的构建设置。

   a. 如果要使用默认的构建项目设置运行构建，请选择启动构建。构建会立即开始。

   b. 如果要覆盖默认构建项目设置，请选择使用覆盖启动构建。在启动构建页面中，您可以覆盖以下内容：

      - 构建配置
      - 源
      - 环境变量覆盖

      如果您需要选择更为高级的覆盖，请选择高级构建覆盖。在该页面中，您可以覆盖以下内容：

      - 构建配置
      - 源
      - 环境
      - Buildspec
      - 构件
      - 日志

      做出覆盖选择后，选择启动构建。

有关此构建的详细信息，请参阅查看构建详细信息（控制台）。

运行构建 (AWS CLI)

> **Note**
>
> CodePipeline 要使用运行构建 AWS CodeBuild，请跳过这些步骤并按照中的说明进行操作创建使用 CodeBuild (AWS CLI) 的管道。
> 有关 AWS CLI 与配合使用的更多信息 CodeBuild，请参阅命令行参考。

1. 使用以下方法之一运行 `start-build` 命令：

   ```
   aws codebuild start-build --project-name <project-name>
   ```

   如果您要运行的构建项目使用的是最新版本的构建输入项目和构建项目现有设置，请使用此方法。

   ```
   aws codebuild start-build --generate-cli-skeleton
   ```

   如果您要运行的构建具有早期版本的构建输入项目，或者如果您要覆盖构建输出项目、环境变量、构建规范或默认构建超时期限的设置，请使用此方法。
2. 如果您使用`--project-name`选项运行start-build命令，请*<project-name>*替换为构建项目的名称，然后跳到此过程的步骤 6。要获取构建项目的列表，请参阅查看构建项目名称。
3. 如果您运行带 `--idempotency-token` 选项的 start-build 命令，则 `start-build` 请求将附带唯一的区分大小写的标识符或令牌。令牌在发出请求后的 5 分钟内有效。如果您使用相同的令牌重复`start-build`请求，但更改了参数，则 CodeBuild 会返回参数不匹配错误。
4. 如果您使用`--generate-cli-skeleton`选项运行该start-build命令，则 JSON-formatted 数据会显示在输出中。将数据复制到安装在本地计算机或实例上的某个位置的 AWS CLI 文件（例如*start-build.json*）。修改所复制的数据，使其符合以下格式，然后保存结果：

   ```
   {
     "projectName": "projectName",
     "sourceVersion": "sourceVersion",
     "artifactsOverride": {
       "type": "type",
       "location": "location",
       "path": "path",
       "namespaceType": "namespaceType",
       "name": "artifactsOverride-name",
       "packaging": "packaging"
   ```

```
  },
  "buildspecOverride": "buildspecOverride",
  "cacheOverride": {
    "location": "cacheOverride-location",
    "type": "cacheOverride-type"
  },
  "certificateOverride": "certificateOverride",
  "computeTypeOverride": "computeTypeOverride",
  "environmentTypeOverride": "environmentTypeOverride",
  "environmentVariablesOverride": {
    "name": "environmentVariablesOverride-name",
    "value": "environmentVariablesValue",
    "type": "environmentVariablesOverride-type"
  },
  "gitCloneDepthOverride": "gitCloneDepthOverride",
  "imageOverride": "imageOverride",
  "idempotencyToken": "idempotencyToken",
  "insecureSslOverride": "insecureSslOverride",
  "privilegedModeOverride": "privilegedModeOverride",
  "queuedTimeoutInMinutesOverride": "queuedTimeoutInMinutesOverride",
  "reportBuildStatusOverride": "reportBuildStatusOverride",
  "timeoutInMinutesOverride": "timeoutInMinutesOverride",
  "sourceAuthOverride": "sourceAuthOverride",
  "sourceLocationOverride": "sourceLocationOverride",
  "serviceRoleOverride": "serviceRoleOverride",
  "sourceTypeOverride": "sourceTypeOverride"
}
```

替换以下占位符：

- *projectName*: 必填字符串。用于此构建项目的构建项目名称。
- *sourceVersion*：可选字符串。要构建的源代码版本，如下所示：
  - 对于 Amazon S3，与您需要构建的输入 ZIP 文件的版本相对应的版本 ID。如果*sourceVersion*未指定，则使用最新版本。
  - 对于 CodeCommit，与您要构建的源代码版本相对应的提交 ID。如果未指定，*sourceVersion*则使用默认分支的 HEAD 提交 ID。（您无法为指定标签名称*sourceVersion*，但可以指定标签的提交 ID。）
  - 对于 GitHub与要构建的源代码版本相对应的提交 ID、拉取请求 ID、分支名称或标签名称。如果指定了拉取请求 ID，则必须使用格式 pr/*pull-request-ID*（例如，pr/25）。如果指

定了分支名称，则将使用分支的 HEAD 提交 ID。如果未指定，*sourceVersion*则使用默认分支的 HEAD 提交 ID。

- 对于 Bitbucket，为提交 ID、分支名称或与您要构建的源代码版本相对应的标签名称。如果指定了分支名称，则将使用分支的 HEAD 提交 ID。如果未指定，*sourceVersion*则使用默认分支的 HEAD 提交 ID。

- 以下占位符适用于 artifactsOverride。
  - *type*：可选。构建项目中定义覆盖此构建项目的构建输出项目类型。
  - *location*：可选。构建项目中定义覆盖此构建项目的构建输出项目位置。
  - *path*：可选。构建项目中定义覆盖此构建项目的构建输出项目路径。
  - *namespaceType*：可选。构建项目中定义覆盖此构建项目的构建输出项目路径类型。
  - *name*：可选。构建项目中定义覆盖此构建项目的构建输出项目名称。
  - *packaging*：可选。构建项目中定义覆盖此构建项目的构建输出项目打包类型。
- *buildspecOverride*：可选。构建项目中定义覆盖此构建项目的构建规范声明。如果设置了该值，则它可以是内联构建规范定义，也可以是指向相对于内置 CODEBUILD_SRC_DIR 环境变量的值的替代构建规范文件的路径，或者是指向 S3 存储桶的路径。S3 存储桶必须与构建项目位于同一 AWS 区域中。使用其 ARN 指定 buildspec 文件（例如，arn:aws:s3:::*<my-codebuild-sample2>*/buildspec.yml）。如果此值未提供或设置为空字符串，源代码必须在其根目录中包含 buildspec.yml 文件。有关更多信息，请参阅 buildspec 文件名称和存储位置。
- 以下占位符适用于 cacheOverride。
  - *cacheOverride-location*：可选。此构建的 ProjectCache 对象的位置，该对象将覆盖构建项目中指定的 ProjectCache 对象。cacheOverride 是可选的，它采用 ProjectCache 对象。location 在 ProjectCache 对象中是必需的。
  - *cacheOverride-type*：可选。此构建的 ProjectCache 对象的类型，该对象将覆盖构建项目中指定的 ProjectCache 对象。cacheOverride 是可选的，它采用 ProjectCache 对象。type 在 ProjectCache 对象中是必需的。
- *certificateOverride*：可选。此构建的证书的名称，该证书将覆盖构建项目中指定的证书。
- *environmentTypeOverride*：可选。此构建的容器类型，该容器类型将覆盖构建项目中指定的容器类型。当前的有效字符串为 LINUX_CONTAINER。
- 以下占位符适用于 environmentVariablesOverride。
  - *environmentVariablesOverride-name*：可选。构建项目中的环境变量名称，其值将会覆盖此构建项目中的相应值。

  - *environmentVariablesOverride-type*：可选。构建项目中的环境变量类型，其值将会覆盖此构建项目中的相应值。
  - *environmentVariablesValue*：可选。构建项目中定义的环境变量值，其值将会覆盖此构建项目中的相应值。
- *gitCloneDepthOverride*：可选。构建项目中 Git 克隆深度的值，您希望此构建项目会覆盖其值。如果您的源类型是 Amazon S3，则不支持此值。
- *imageOverride*：可选。此构建的映像的名称，该映像将覆盖构建项目中指定的映像。
- *idempotencyToken*：可选。一个字符串，该字符串用作令牌来指定构建请求是幂等的。您可以选择任何包含 64 个或更少字符的字符串。令牌在发出 start-build 请求后的 5 分钟内有效。如果您使用相同的令牌重复启动构建请求，但更改了参数，则会 CodeBuild 返回参数不匹配错误。
- *insecureSslOverride*：可选布尔值，用于指定是否覆盖构建项目中指定的不安全 TLS 设置。不安全的 TLS 设置确定是否忽略 TLS 警告，并连接到项目源代码。仅当编译版本的源为 GitHub 企业服务器时，此替代才适用。
- *privilegedModeOverride*：可选布尔值。如果设置为 true，则构建将覆盖构建项目中的特权模式。
- *queuedTimeoutInMinutesOverride*：可选整数，用于指定版本在超时之前允许排队的分钟数。最小值为 5 分钟，最大值为 480 分钟（8 个小时）。
- *reportBuildStatusOverride*：可选布尔值，用于指定是否向源提供商发送构建开始和完成的状态。如果您使用 GitHub 企业服务器或 Bitbucket 以外的 GitHub源提供商进行此设置，则会引发无效InputException 消息。
- *sourceAuthOverride*：可选字符串。此构建的授权类型，该授权类型将覆盖构建项目中定义的授权类型。仅当构建项目的源为 Bitbucket 或 GitHub，此替代才适用。
- *sourceLocationOverride*：可选字符串。此构建的源位置，该源位置将覆盖构建项目中定义的源位置。
- *serviceRoleOverride*：可选字符串。此构建的服务角色的名称，该角色将覆盖构建项目中指定的角色。
- *sourceTypeOverride*：可选字符串。此构建的源输入类型，该源输入将覆盖构建项目中定义的源输入。有效字符串包括：NO_SOURCE、CODECOMMIT、CODEPIPELINE、GITHUB、S3、BITBUCKET 和 GITHUB_ENTERPRISE。
- *timeoutInMinutesOverride*：可选号码。构建项目中定义覆盖此构建项目的构建超时分钟数。

我们建议您将具有敏感值的环境变量，例如访问密钥 ID、私有 AWS 访问 AWS 密钥或密码作为参数存储在 Amazon EC2 Systems Manager 参数存储区中。 CodeBuild 只有当参数的名称以/CodeBuild/（例如，/CodeBuild/dockerLoginPassword）开头时，才能使用存储在 Amazon EC2 系统管理器参数存储中的参数。您可以使用 CodeBuild 控制台在 Amazon EC2 系统管理器中创建参数。选择创建参数，然后按照说明操作。（在该对话框中，对于 KMS 密钥，您可以选择性指定您账户中的 AWS KMS 密钥的 ARN。Amazon EC2 Systems Manager 使用此密钥在存储过程中加密参数的值并在检索过程中解密参数的值。）如果您使用 CodeBuild 控制台创建参数，则控制台会在存储参数时启动该参数。/CodeBuild/但是，如果您使用 Amazon EC2 Systems Manager Parameter Store 控制台创建参数，则必须使用以 /CodeBuild/ 开头的参数名称，且必须将类型设置为安全字符串。有关更多信息，请参阅《Amazon EC2 Systems Manager 用户指南》中的 AWS Systems Manager Parameter Store 和演练：创建和测试参数（控制台）。

如果您的构建项目引用了 Amazon EC2 Systems Manager Parameter Store 中存储的参数，则构建项目的服务角色必须允许 ssm:GetParameters 操作。如果您之前选择了在账户中创建新服务角色，则会自动将此操作 CodeBuild 包含在构建项目的默认服务角色中。但是，如果您选择了 Choose an existing service role from your account，则必须将此操作单独包含在您的服务角色中。

您设置的环境变量将替换现有的环境变量。例如，如果 Docker 映像已经包含一个名为 MY_VAR 的环境变量（值为 my_value），并且您设置了一个名为 MY_VAR 的环境变量（值为 other_value），那么 my_value 将被替换为 other_value。同样，如果 Docker 映像已经包含一个名为 PATH 的环境变量（值为 /usr/local/sbin:/usr/local/bin），并且您设置了一个名为 PATH 的环境变量（值为 $PATH:/usr/share/ant/bin），那么/usr/local/sbin:/usr/local/bin 将被替换为文本值 $PATH:/usr/share/ant/bin。

请勿使用以 CODEBUILD_ 打头的名称设置任何环境变量。此前缀是专为内部使用预留的。

如果具有相同名称的环境变量在多处都有定义，则将按照如下方式确定环境变量的值：

- 构建操作调用开始时的值优先级最高。
- 构建项目定义中的值优先级次之。
- 构建规范文件声明中的值优先级最低。

有关这些占位符的有效值的信息，请参阅创建构建项目 (AWS CLI)。有关构建项目的最新设置列表，请参阅查看构建项目详细信息。

5. 切换到包含您刚才保存的文件的目录，然后再次运行 start-build 命令。

```
aws codebuild start-build --cli-input-json file://start-build.json
```

6. 如果成功，与运行构建 过程中所述内容类似的数据将出现在输出中。

要了解有关此构建项目的详细信息，请记下输出中的 id 值，然后查看查看构建详细信息（AWS CLI）。

运行批量构建 (AWS CLI)

1. 使用以下方法之一运行 start-build-batch 命令：

```
aws codebuild start-build-batch --project-name <project-name>
```

如果您要运行的构建项目使用的是最新版本的构建输入项目和构建项目现有设置，请使用此方法。

```
aws codebuild start-build-batch --generate-cli-skeleton > <json-file>
```

如果您要运行的构建具有早期版本的构建输入项目，或者如果您要覆盖构建输出项目、环境变量、构建规范或默认构建超时期限的设置，请使用此方法。

2. 如果您使用--project-name选项运行start-build-batch命令，请*<project-name>*替换为构建项目的名称，然后跳到此过程的步骤 6。要获取构建项目的列表，请参阅查看构建项目名称。
3. 如果您运行带 --idempotency-token 选项的 start-build-batch 命令，则 start-build-batch 请求将附带唯一的区分大小写的标识符或令牌。令牌在发出请求后的 5 分钟内有效。如果您使用相同的令牌重复start-build-batch请求，但更改了参数，则 CodeBuild 会返回参数不匹配错误。
4. 如果您使用--generate-cli-skeleton选项运行start-build-batch命令，则 JSON-formatted 数据将输出到*<json-file>*文件中。该文件与start-build命令生成的骨架类似，但添加了以下对象。有关常见对象的更多信息，请参阅运行构建 (AWS CLI)。

修改此文件以添加任何构建覆盖，并保存结果。

```
"buildBatchConfigOverride": {
  "combineArtifacts": combineArtifacts,
  "restrictions": {
    "computeTypesAllowed": [
      allowedComputeTypes
```

```
      ],
      "maximumBuildsAllowed": maximumBuildsAllowed
    },
    "serviceRole": "batchServiceRole",
    "timeoutInMins": batchTimeout
  }
```

该buildBatchConfigOverride对象是一个包含此生成的批量生成配置替代项的ProjectBuildBatchConfig结构。

combineArtifacts

指定批量构建的构建构件是否应合并到单个构件位置的布尔值。

allowedComputeTypes

一组字符串，用于指定批量构建允许的计算类型。请参阅构建环境计算类型以了解这些值。

maximumBuildsAllowed

指定允许的最大构建数。

batchServiceRole

为批量构建项目指定服务角色 ARN。

batchTimeout

指定必须完成批量构建的最长时间（以分钟为单位）。

5. 切换到包含您刚才保存的文件的目录，然后再次运行 start-build-batch 命令。

```
aws codebuild start-build-batch --cli-input-json file://start-build.json
```

6. 如果成功，则BuildBatch对象的 JSON 表示形式出现在控制台输出中。有关此数据的示例，请参见StartBuildBatch 响应语法。

开始自动运行构建（AWS CLI）

如果您的源代码存储在 GitHub 或 E GitHub nterprise Server 存储库中，则每当将代码更改推送到存储库时，都可以使用 GitHub webhook 来 AWS CodeBuild 重新生成源代码。

运行 create-webhook 命令，如下所示：

```
aws codebuild create-webhook --project-name <project-name>
```

*<project-name>*是包含要重建的源代码的生成项目的名称。

对于 GitHub，输出中会显示类似以下内容的信息：

```
{
  "webhook": {
    "url": "<url>"
  }
}
```

*<url>*是 GitHub webhook 的网址。

对于 GitHub 企业服务器，输出中会显示类似以下内容的信息：



1. 从输出中复制私有密钥和负载 URL。你需要它们在 GitHub 企业服务器中添加 webhook。
2. 在 GitHub 企业服务器中，选择存储 CodeBuild 项目的存储库。选择设置，选择挂钩和服务，然后选择添加 webhook。
3. 输入负载 URL 和私有密钥，接受其他字段的默认值，然后选择添加 webhook。

停止自动运行构建（AWS CLI）

如果您的源代码存储在 GitHub 或 E GitHub nterprise Server 存储库中，则可以将 GitHub webhook 设置为在将代码更改推送到存储库时 AWS CodeBuild 重新生成源代码。有关更多信息，请参阅 开始自动运行构建（AWS CLI）。

如果您已启用了此行为，则可以通过运行 `delete-webhook` 命令将其关闭，如下所示：

```
aws codebuild delete-webhook --project-name <project-name>
```

- 其中，*<project-name>*是包含要重建的源代码的构建项目的名称。

如果此命令成功，则输出中不会出现任何信息和错误。

> **Note**
>
> 这只会从您的 CodeBuild 项目中删除 webhook。您还应该从您的存储库 GitHub 或 GitHub 企业服务器存储库中删除 webhook。

运行构建 (AWS SDK)

CodePipeline 要使用运行构建 AWS CodeBuild，请跳过这些步骤，使用 AWS CodeBuild 用 AWS CodePipeline 测试代码并运行构建改为按照中的说明进行操作。

有关与 AWS 软件开发工具包 CodeBuild 一起使用的信息，请参阅。AWS SDK 和工具参考

在 build 上运行 AWS Lambda 计算

AWS Lambda compute 为您的构建提供优化的启动速度。 AWS Lambda 由于启动延迟较低，支持更快的构建。 AWS Lambda 还能自动扩展，因此编译版本无需排队等候运行。但是，有些用例 AWS Lambda 不支持，如果它们对您有影响，请使用 EC2 计算。有关更多信息，请参阅 的局限性 AWS Lambda 计算。

主题

- 精选的运行时环境 docker 镜像中将包含哪些工具和运行时 AWS Lambda?
- 如果精选映像未包括我需要的工具，该怎么办？
- 哪些地区支持 AWS Lambda 计算进去 CodeBuild？
- 的局限性 AWS Lambda 计算
- 使用部署 Lambda 函数 AWS SAM 使用 CodeBuild Lambda Java
- 使用 CodeBuild Lambda 创建单页的 React 应用程序 Node.js
- 使用 Lambda Python 更新 L CodeBuild ambda 函数配置

精选的运行时环境 docker 镜像中将包含哪些工具和运行时 AWS Lambda?

AWS Lambda 支持以下工具： AWS CLI v2、 AWS SAM CLI、git、go、Java、、Python Node.js、pip、Ruby 和 .NET。

如果精选映像未包括我需要的工具，该怎么办？

如果精选映像不包括您需要的工具，则可以提供包括所需工具的自定义环境 Docker 映像。

> **Note**
>
> Lambda 不支持使用多架构容器映像的函数。有关更多信息，请参阅《AWS Lambda 开发人员指南》中的使用容器映像创建 Lambda 函数。

请注意，您需要以下 Amazon ECR 权限才能使用 Lambda 计算的自定义映像：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "arn:aws:ecr:us-east-1:111122223333:repository/image-repo"
        }
    ]
}
```

另请注意，要使用自定义映像，必须安装 `curl` 或 `wget`。

哪些地区支持 AWS Lambda 计算进去 CodeBuild？

在中 CodeBuild，以下地区支持 AWS Lambda 计算 AWS 区域：美国东部（弗吉尼亚北部）、美国东部（俄亥俄）、美国西部（俄勒冈）、亚太地区（孟买）、亚太地区（新加坡）、亚太地区（悉尼）、亚太地区（东京）、欧洲（法兰克福）、欧洲（爱尔兰）和南美洲（圣保罗）。有关 AWS 区域 何处 CodeBuild 提供服务的更多信息，请参阅按地区划分的AWS 服务。

的局限性 AWS Lambda 计算

有些用例 AWS Lambda 不支持，如果它们对你有影响，可以使用 EC2 计算：

- AWS Lambda 不支持需要 root 权限的工具。对于 `yum` 或 `rpm` 之类的工具，请使用 EC2 计算类型或其他不需要 root 权限的工具。
- AWS Lambda 不支持 Docker 构建或运行。
- AWS Lambda 不支持写入外部文件`/tmp`。包含的包管理器被配置为默认使用 `/tmp` 目录来下载和引用包。
- AWS Lambda 不支持该环境类型`LINUX_GPU_CONTAINER`，Windows Server Core 2019 也不支持。
- AWS Lambda 不支持缓存、自定义编译超时、队列超时、构建徽章、特权模式、自定义运行时环境或超过 15 分钟的运行时间。
- AWS Lambda 不支持 VPC 连接、固定范围的 CodeBuild 源 IP 地址、EFS、安装证书或使用会话管理器进行 SSH 访问。

使用部署 Lambda 函数 AWS SAM 使用 CodeBuild Lambda Java

AWS Serverless Application Model (AWS SAM) 是一个用于构建无服务器应用程序的开源框架。有关更多信息，请参阅上的AWS Serverless Application Model 存储库 GitHub。以下 Java 示例使用 Gradle 来构建和测试 AWS Lambda 函数。之后，使用 AWS SAM CLI 来部署 CloudFormation 模板和部署包。通过使用 CodeBuild Lambda，构建、测试和部署步骤均可自动处理，从而无需在单个构建中进行手动干预即可快速更新基础架构。

设置你的 AWS SAM repository

使用 AWS SAM CLI 创建 AWS SAM `Hello World`项目。

创建你的 AWS SAM Project

1. 按照AWS Serverless Application Model 开发人员指南中的说明在本地计算机上安装 AWS SAM CLI。
2. 运行 `sam init` 并选择以下项目配置。

```
Which template source would you like to use?: 1 - AWS Quick Start Templates
Choose an AWS Quick Start application template: 1 - Hello World Example
Use the most popular runtime and package type? (Python and zip) [y/N]: N
Which runtime would you like to use?: 8 - java21
What package type would you like to use?: 1 - Zip
Which dependency manager would you like to use?: 1 - gradle
Would you like to enable X-Ray tracing on the function(s) in your application? [y/
N]: N
Would you like to enable monitoring using CloudWatch Application Insights? [y/N]: N
Would you like to set Structured Logging in JSON format on your Lambda functions?
 [y/N]:  N
Project name [sam-app]: <insert project name>
```

3. 将 AWS SAM 项目文件夹上传到支持的源存储库。有关支持的源类型列表，请参阅ProjectSource。

创建 CodeBuild Lambda Java 项目

创建 AWS CodeBuild Lambda Java 项目并设置构建所需的 IAM 权限。

创建您的 CodeBuild Lambda Java 项目

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 如果显示 CodeBuild 信息页面，请选择创建构建项目。否则，请在导航窗格中，展开构建，选择构建项目，然后选择创建构建项目。
3. 在项目名称中，输入此构建项目的名称。每个 AWS 账户的构建项目名称必须是唯一的。您还可以包含构建项目的可选描述，以帮助其他用户了解此项目的用途。
4. 在源代码中，选择 AWS SAM 项目所在的源存储库。
5. 在环境中：
   - 在计算中，选择 Lambda。
   - 在运行时中，选择 Java。
   - 对于图片，选择 aws/codebuild /amazon linux-x86_64-lambda-standard: corretto21。

- 在服务角色中，选中新服务角色。记下角色名称。在本示例的后面部分更新项目的 IAM 权限时，您需要此信息。

6. 选择 Create build project（创建构建项目）。
7. 使用 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
8. 在导航窗格中，选择角色，然后选择与项目关联的服务角色。 CodeBuild 通过选择构建项目，选择 “编辑”、“环境” 和 “服务角色”，可以在中找到您的项目角色。
9. 选择 Trust relationships（信任关系）选项卡，然后选择 Edit trust policy（编辑信任策略）。
10. 将以下内联策略附加到您的 IAM 角色。这将用于稍后部署您的 AWS SAM 基础架构。有关更多信息，请参阅《 IAM 用户指南》中的添加和移除 IAM 身份权限。

    JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Action": [
                "cloudformation:*",
                "lambda:*",
                "iam:*",
                "apigateway:*",
                "s3:*"
            ],
            "Resource": "arn:aws:iam::*:role/Service*"
        }
    ]
}
```

设置项目 buildspec

为了构建、测试和部署您的 Lambda 函数，请从构建规范中 CodeBuild 读取和执行构建命令。

设置项目 buildspec

1. 在 CodeBuild 控制台中，选择您的构建项目，然后选择编辑和编译规范。
2. 在 Buildspec 中，选择插入构建命令，然后选择切换到编辑器。

3. 删除预先填入的构建命令并粘贴以下 buildspec。

```
version: 0.2
env:
  variables:
    GRADLE_DIR: "HelloWorldFunction"
phases:
  build:
    commands:
      - echo "Running unit tests..."
      - cd $GRADLE_DIR; gradle test; cd ..
      - echo "Running build..."
      - sam build --template-file template.yaml
      - echo "Running deploy..."
      - sam package --output-template-file packaged.yaml --resolve-s3 --template-
file template.yaml
      - yes | sam deploy
```

4. 选择 Update buildspec (更新构建规范)。

部署你的 AWS SAM Lambda 基础设施

使用 CodeBuild Lambda 自动部署您的 Lambda 基础设施

部署 Lambda 基础设施

1. 选择启动构建。这将自动构建、测试您的 AWS SAM 应用程序并将其部署到 AWS Lambda 使用 CloudFormation。
2. 构建完成后，导航到 AWS Lambda 控制台并在 AWS SAM 项目名称下搜索您的新 Lambda 函数。
3. 在函数概览下面选择 API Gateway，然后单击 API 端点 URL，测试您的 Lambda 函数。您应该会看到一个页面打开，其中包含以下消息：`"message": "hello world"`。

清除基础设施

为避免在本教程中使用的资源产生更多费用，请删除 AWS SAM 模板创建的资源，然后 CodeBuild。

清除基础设施

1. 导航到 CloudFormation 控制台并选择`aws-sam-cli-managed-default`。

2. 在资源中，清空部署存储桶 `SamCliSourceBucket`。
3. 删除 `aws-sam-cli-managed-default` 堆栈。
4. 删除与您的 AWS SAM 项目关联的 CloudFormation 堆栈。此堆栈应与您的 AWS SAM 项目同名。
5. 导航到 CloudWatch 控制台并删除与您的 CodeBuild 项目关联的 CloudWatch 日志组。
6. 导航到 CodeBuild 控制台并选择 “删除构建 CodeBuild 项目” 来删除您的项目。

使用 CodeBuild Lambda 创建单页的 React 应用程序 Node.js

创建 React 应用程序是一种创建单页 React 应用程序的方法。以下 Node.js 示例用于 Node.js 从 Create React App 中构建源工件并返回构建工件。

设置源存储库和构件存储桶

使用 yarn 和“创建 React 应用程序”为项目创建源存储库。

设置源存储库和构件存储桶

1. 在本地计算机上运行 `yarn create react-app <app-name>` 来创建简单的 React 应用程序。
2. 将 React 应用程序项目文件夹上传到支持的源存储库。有关支持的源类型的列表，请参阅ProjectSource。

创建一个 CodeBuild Lambda 项目 Node.js

创建一个 AWS CodeBuild Lambda 项目 Node.js 。

创建您的 CodeBuild Lambda 项目 Node.js

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 如果显示 CodeBuild 信息页面，请选择 “创建构建项目”。否则，请在导航窗格中，展开构建，选择构建项目，然后选择创建构建项目。
3. 在项目名称中，输入此构建项目的名称。每个 AWS 账户中的构建项目名称必须是唯一的。您还可以包含构建项目的可选描述，以帮助其他用户了解此项目的用途。
4. 在源代码中，选择 AWS SAM 项目所在的源存储库。
5. 在环境中：

- 在计算中，选择 Lambda。
- 对于 “运行时间”，选择Node.js。
- 对于图片，选择 aws/codebuild/amazon linux-x86_64-lambda-standard: nodejs20。

6. 在构件中：
   - 在类型中，选择 Amazon S3。
   - 在存储桶名称中，选择您之前创建的项目构件存储桶。
   - 在构件打包中，选择 Zip。
7. 选择 Create build project（创建构建项目）。

设置项目 buildspec

为了构建 React 应用程序，需要从 buildspec 文件中 CodeBuild 读取和执行构建命令。

设置项目 buildspec

1. 在 CodeBuild 控制台中，选择您的构建项目，然后选择 “编辑” 和 “Buildspec”。
2. 在 Buildspec 中，选择插入构建命令，然后选择切换到编辑器。
3. 删除预先填入的构建命令并粘贴以下 buildspec。

```
version: 0.2
phases:
  build:
    commands:
      - yarn
      - yarn add --dev jest-junit @babel/plugin-proposal-private-property-in-object
      - yarn run build
      - yarn run test -- --coverage --watchAll=false --testResultsProcessor="jest-
junit" --detectOpenHandles
artifacts:
  name: "build-output"
  files:
    - "**/*"
reports:
  test-report:
    files:
      - 'junit.xml'
    file-format: 'JUNITXML'
```

```
  coverage-report:
    files:
      - 'coverage/clover.xml'
    file-format: 'CLOVERXML'
```

4. 选择 Update buildspec (更新构建规范)。

构建和运行 React 应用程序

在 CodeBuild Lambda 上构建 React 应用程序，下载构建工件，然后在本地运行 React 应用程序。

构建和运行 React 应用程序

1. 选择启动构建。
2. 构建完成后，导航到您的 Amazon S3 项目构件存储桶并下载 React 应用程序构件。
3. 解压缩 React 构建构件并在项目文件夹中执行 `run npm install -g serve && serve -s build`。
4. `serve` 命令将在本地端口为静态站点提供服务，并输出到您的终端。您可以访问终端输出中 `Local:` 下面的 localhost URL，查看您的 React 应用程序。

要详细了解如何处理基于 React 的服务器的部署，请参阅创建 React 应用程序部署。

清除基础设施

为避免对您在本教程中使用的资源收取更多费用，请删除为您的 CodeBuild 项目创建的资源。

清除基础设施

1. 删除项目构件 Amazon S3 存储桶
2. 导航到 CloudWatch 控制台并删除与您的 CodeBuild 项目关联的 CloudWatch 日志组。
3. 导航到 CodeBuild 控制台并通过选择删除构建 CodeBuild 项目来删除您的项目。

使用 Lambda Python 更新 L CodeBuild ambda 函数配置

以下 Python 示例使用 Boto3 和 CodeBuild Lambda Python 来更新 Lambda 函数的配置。此示例可以扩展为以编程方式管理其他 AWS 资源。有关更多信息，请参阅 Boto3 文档。

先决条件

在账户中创建或查找 Lambda 函数。

此示例假设您已经在账户中创建了一个 Lambda 函数，并将用于 CodeBuild 更新 Lambda 函数的环境变量。有关通过设置 Lambda 函数的更多信息 CodeBuild，请参阅使用部署 Lambda 函数 AWS SAM 使用 CodeBuild Lambda Java示例或访问AWS Lambda。

设置源存储库

创建源存储库来存储 Boto3 python 脚本。

设置源存储库。

1. 将以下 python 脚本复制到名为 `update_lambda_environment_variables.py` 的新文件中。

```
import boto3
from os import environ


def update_lambda_env_variable(lambda_client):
    lambda_function_name = environ['LAMBDA_FUNC_NAME']
    lambda_env_variable = environ['LAMBDA_ENV_VARIABLE']
    lambda_env_variable_value = environ['LAMBDA_ENV_VARIABLE_VALUE']
    print("Updating lambda function " + lambda_function_name + " environment
 variable "
          + lambda_env_variable + " to " + lambda_env_variable_value)
    lambda_client.update_function_configuration(
        FunctionName=lambda_function_name,
        Environment={
            'Variables': {
                lambda_env_variable: lambda_env_variable_value
            }
        },
    )


if __name__ == "__main__":
    region = environ['AWS_REGION']
    client = boto3.client('lambda', region)
    update_lambda_env_variable(client)
```

2. 将 python 文件上传到支持的源存储库。有关支持的源类型列表，请参阅ProjectSource。

创建 CodeBuild Lambda Python 项目

创建 CodeBuild Lambda Python 项目。

创建您的 CodeBuild Lambda Java 项目

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 如果显示 CodeBuild 信息页面，请选择创建构建项目。否则，请在导航窗格中，展开构建，选择构建项目，然后选择创建构建项目。
3. 在项目名称中，输入此构建项目的名称。每个 AWS 账户的构建项目名称必须是唯一的。您还可以包含构建项目的可选描述，以帮助其他用户了解此项目的用途。
4. 在源代码中，选择 AWS SAM 项目所在的源存储库。
5. 在环境中：
   - 在计算中，选择 Lambda。
   - 在运行时中，选择 Python。
   - 对于图片，选择 aws/codebuild /amazonlinux-x86_64-lambda-standard: python3.12 。
   - 在服务角色中，选中新服务角色。记下角色名称。在本示例的后面部分更新项目的 IAM 权限时，您需要此信息。
6. 选择 Create build project（创建构建项目）。
7. 使用 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
8. 在导航窗格中，选择角色，然后选择与项目关联的服务角色。 CodeBuild 通过选择构建项目，选择 “编辑”、“环境” 和 “服务角色”，可以在中找到您的项目角色。
9. 选择 Trust relationships（信任关系）选项卡，然后选择 Edit trust policy（编辑信任策略）。
10. 将以下内联策略附加到您的 IAM 角色。这将用于稍后部署您的 AWS SAM 基础架构。有关更多信息，请参阅《 IAM 用户指南》中的添加和移除 IAM 身份权限。

    JSON

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

```
                "lambda:UpdateFunctionConfiguration"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

设置项目 buildspec

要更新 Lambda 函数，该脚本从 buildspec 中读取环境变量，以查找 Lambda 函数的名称、环境变量名称和环境变量值。

设置项目 buildspec

1. 在 CodeBuild 控制台中，选择您的构建项目，然后选择编辑和编译规范。
2. 在 Buildspec 中，选择插入构建命令，然后选择切换到编辑器。
3. 删除预先填入的构建命令并粘贴以下 buildspec。

```
version: 0.2
env:
  variables:
    LAMBDA_FUNC_NAME: "<lambda-function-name>"
    LAMBDA_ENV_VARIABLE: "FEATURE_ENABLED"
    LAMBDA_ENV_VARIABLE_VALUE: "true"
phases:
  install:
    commands:
       - pip3 install boto3
  build:
    commands:
       - python3 update_lambda_environment_variables.py
```

4. 选择 Update buildspec (更新构建规范)。

更新 Lambda 配置

使用 CodeBuild Lambda Python 自动更新您的 Lambda 函数的配置。

更新您的 Lambda 函数的配置

1. 选择启动构建。
2. 构建完成后，导航到您的 Lambda 函数。
3. 选择配置，然后选择环境变量。您应该会看到一个具有键 FEATURE_ENABLED 和值 true 的新环境变量。

清除基础设施

为避免在本教程中使用的资源产生更多费用，请删除为您的 CodeBuild 项目创建的资源。

清除基础设施

1. 导航到 CloudWatch 控制台并删除与您的 CodeBuild 项目关联的 CloudWatch 日志组。
2. 导航到 CodeBuild 控制台并选择 “删除构建 CodeBuild 项目” 来删除您的项目。
3. 如果您为此示例创建了 Lambda 函数，请选择操作和删除函数来清理您的 Lambda 函数。

扩展程序

如果您想扩展此示例以使用 AWS CodeBuild Lambda Python 管理其他 AWS 资源：

- 使用 Boto3 更新 Python 脚本来修改新资源。
- 更新与您的 CodeBuild 项目关联的 IAM 角色以获得新资源的权限。
- 将与新资源关联的所有新环境变量添加到 buildspec 中。

在预留容量实例集上运行构建

CodeBuild 提供以下计算队列：

- On-demand 舰队
- 预留容量实例集

使用按需队列， CodeBuild 为您的构建提供计算。建造完成后，机器将被摧毁。 On-demand 车队是完全管理的，包括自动扩展功能以应对需求激增。

> **Note**
>
> On-demand 舰队不支持 macOS。

CodeBuild 还提供预留容量队列，其中包含由 Amazon EC2 提供支持、由维护的 CodeBuild实例。使用预留容量实例集，您可以为构建环境配置一组专用实例。这些计算机保持空闲状态，随时准备立即处理构建或测试，并缩短构建时间。使用预留容量队列，您的计算机将始终处于运行状态，并且只要配置完毕，就会继续产生成本。

> **Important**
>
> 无论您运行实例多长时间，预留容量实例集的每个实例都会产生初始费用，之后可能会有额外的相关费用。有关更多信息，请参阅 https://aws.amazon.com/codebuild/pricing/。

主题

- 创建预留容量实例集
- 最佳实践
- 我可以在多个 CodeBuild 项目之间共享预留容量队列吗？
- 基于属性的计算是如何工作的？
- 我能否为我的实例集手动指定 Amazon EC2 实例？
- 哪些区域支持预留容量实例集？
- 如何配置 macOS 预留容量实例集？
- 如何为预留容量实例集配置自定义亚马逊机器映像（AMI）？
- 预留容量实例集的局限性
- 预留容量实例集属性
- 预留容量样本包括 AWS CodeBuild

创建预留容量实例集

按照以下说明创建预留容量实例集。

创建预留容量实例集

1. 登录 AWS 管理控制台 并打开 AWS CodeBuild 控制台，地址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择计算实例集，然后选择创建实例集。
3. 在计算实例集名称文本字段中，输入实例集的名称。
4. 从操作系统下拉菜单中，选择操作系统。
5. 从架构下拉菜单中，选择架构。
6. （可选）选择使用实例运行模式 - 可选，以便直接在 Amazon EC2 实例而不是 Docker 容器上运行。然后选择主要版本和次要版本。
7. （可选）在其他配置中，执行以下操作：

   - 选择配置 VPC - 可选以将实例集连接到 VPC，以便在使用期间访问私有资源。
     - 从 VPC 下拉菜单中，选择您的 CodeBuild 队列将要访问的 VPC。
     - 从子网下拉菜单中，选择 CodeBuild 应用于设置 VPC 配置的子网。
     - 从安全组下拉菜单中，选择 CodeBuild 应用于处理您的 VPC 的安全组。
     - 在实例集服务角色字段中，选择已有服务角色。

     > **Note**
     >
     > 确保实例集角色具有必要的权限。有关更多信息，请参阅 允许用户为实例集服务角色添加权限策略。

     - 如果您选择了 Amazon Linux 操作系统，请选择定义代理配置 - 可选，以便对您的预留容量实例应用网络访问控制。
     - 对于默认行为，选择在默认情况下是允许还是拒绝发往所有目标的传出流量。
     - 对于代理规则，选择添加代理规则，以便指定允许或拒绝网络访问控制的目标域或 IP。
   - 选择配置自定义 AMI - 可选，以使用自定义亚马逊机器映像（AMI）。
     - 从 AMI 下拉菜单中，为您的队列选择亚马逊系统映像 (AMI)。
     - 在实例集服务角色字段中，选择已有服务角色。

> **Note**
>
> 确保实例集角色具有必要的权限。有关更多信息，请参阅 允许用户为实例集服务角色添加权限策略。

8. 在容量配置中，从计算选择模式中选择以下选项之一：

   - 如果您选择引导式选择，请执行以下操作：
     - 对于计算，请选择此实例集中包含的实例类型。
     - 在容量文本字段中，输入实例集中的最少实例数。
     - （可选）在其他配置中，执行以下操作：
       - 选择配置扩展 - 可选，以根据此配置自动扩展您的实例集。从扩展模式 - 可选下拉菜单中，选择在需求超过实例集容量时的行为。
   - 如果选择自定义实例，请执行以下操作：
     - 从计算实例类型下拉菜单中，选择此实例集中包含的实例类型。
     - 在其他 EBS 卷大小 - 可选文本字段中，输入提供的 64 GB 磁盘空间之外的额外容量。
     - 在容量文本字段中，输入实例集中的最少实例数。
     - （可选）在其他配置中，执行以下操作：
       - 选择配置扩展 - 可选，以根据此配置自动扩展您的实例集。从扩展模式 - 可选下拉菜单中，选择在需求超过实例集容量时的行为。
9. 选择创建计算实例集。
10. 创建计算队列后，创建新 CodeBuild 项目或编辑现有项目。从环境中，选择预置模型下的预留容量，然后在实例集名称下选择指定的实例集。

最佳实践

使用预留容量实例集时，我们建议您遵循以下这些最佳实践。

- 我们建议使用源代码缓存模式，通过缓存源代码来帮助提高构建性能。
- 我们建议使用 Docker 层缓存，通过缓存现有 Docker 层来帮助提高构建性能。

我可以在多个 CodeBuild 项目之间共享预留容量队列吗？

可以，您可以通过在多个项目中使用实例集的容量来最大限度地提高其利用率。

> **Important**
>
> 使用预留容量特征时，同一账户内的其他项目可以访问实例集实例中缓存的数据，包括源文件、Docker 层和 buildspec 中指定的缓存目录。这是设计使然，让同一账户内的项目可以共享实例集实例。

基于属性的计算是如何工作的？

如果您选择 `ATTRIBUTE_BASED_COMPUTE` 作为实例集的 `computeType`，则可以在名为 `computeConfiguration` 的新字段中指定属性。这些属性包括 vCPU、内存、磁盘空间和 `machineType`。此 `machineType` 为 `GENERAL` 或 `NVME`。指定一个或部分可用属性后， CodeBuild 将从可用的支持的实例类型中选择一种计算类型作为最终版本`computeConfiguration`。

> **Note**
>
> CodeBuild 将选择符合所有输入要求的最便宜的实例。所选实例的内存、vCPU 和磁盘空间都将大于或等于输入要求。您可以在已创建或更新的实例集中检查已解析的 `computeConfiguration`。

如果您在中输入`computeConfiguration`了无法满足的 a CodeBuild，则会收到验证异常。另请注意，如果 `computeConfiguration` 不适用于按需情况，则按需实例集溢出行为将被覆盖为队列行为。

我能否为我的实例集手动指定 Amazon EC2 实例？

可以，您可以通过选择自定义实例或配置 API 参数 `InstanceType`，直接在控制台中输入所需的 Amazon EC2 实例。此字段用于以下 API： CreateFleet、 UpdateFleet CreateProject、UpdateProject 和 StartBuild。有关更多信息，请参阅 Compute instance type。

哪些区域支持预留容量实例集？

以下区域支持 Amazon Linux 和 Windows 队列的预留容量 AWS 区域：美国东部（弗吉尼亚北部）、美国东部（俄亥俄）、美国西部（俄勒冈）、亚太地区（孟买）、亚太地区（新加坡）、亚太地区（悉尼）、亚太地区（东京）、欧洲（爱尔兰）和南美洲（圣保罗）。有关 AWS 区域 何处 CodeBuild 提供服务的更多信息，请参阅按地区划分的AWS 服务。

以下地区支持预留容量 macOS Medium 队列 AWS 区域：美国东部（弗吉尼亚北部）、美国东部（俄亥俄）、美国西部（俄勒冈）、亚太地区（悉尼）和欧洲（法兰克福）。以下地区支持预留容量 macOS 大型机队 AWS 区域：美国东部（弗吉尼亚北部）、美国东部（俄亥俄州）、美国西部（俄勒冈）和亚太地区（悉尼）。

如何配置 macOS 预留容量实例集？

配置 macOS 预留容量实例集

1. 登录 AWS 管理控制台 并打开 AWS CodeBuild 控制台，地址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择计算实例集，然后选择创建实例集。
3. 在计算实例集名称文本字段中，输入实例集的名称。
4. 在操作系统下拉菜单中，选择 macOS。
5. 在计算字段中，选择以下计算机类型之一：Apple M2、24 GB 内存、8 个 vCPU 或 Apple M2、32 GB 内存、12 个 vCPU。
6. 在容量文本字段中，输入实例集中的最少实例数。
7. （可选）要为实例集使用自定义映像，请参阅如何为预留容量实例集配置自定义亚马逊机器映像（AMI）？，以确保您的亚马逊机器映像（AMI）满足所需的先决条件。
8. （可选）要使用您的实例集配置 VPC，请在其他配置中执行以下操作：
   - 从 VPC-可选下拉菜单中，选择您的 CodeBuild 队列将要访问的 VPC。
   - 从子网下拉菜单中，选择 CodeBuild 应用于设置 VPC 配置的子网。
   - 从安全组下拉菜单中，选择 CodeBuild 应用于处理您的 VPC 的安全组。
   - 在实例集服务角色字段中，选择已有服务角色。

   > **Note**
   >
   > 确保实例集角色具有必要的权限。有关更多信息，请参阅 允许用户为实例集服务角色添加权限策略。
9. 选择创建计算实例集并等待实例集实例启动。一旦启动，容量将在哪里*n*/*n*，提供的容*n*量在哪里。
10. 计算队列启动后，创建新 CodeBuild 项目或编辑现有项目。从环境中，选择预置模型下的预留容量，然后在实例集名称下选择指定的实例集。

如何为预留容量实例集配置自定义亚马逊机器映像（AMI）？

为预留容量实例集配置自定义亚马逊机器映像（AMI）

1. 登录 AWS 管理控制台 并打开 AWS CodeBuild 控制台，地址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择计算实例集，然后选择创建实例集。
3. 在计算实例集名称文本字段中，输入实例集的名称。
4. 为实例集选择自定义映像，并确保您的亚马逊机器映像（AMI）满足以下先决条件：
   - 如果环境类型为 `MAC_ARM`，请确保 AMI 架构为 64 位 `Mac-Arm`。
   - 如果环境类型为 `LINUX_EC2`，请确保 AMI 架构为 64 位 `x86`。
   - 如果环境类型为 `ARM_EC2`，请确保 AMI 架构为 64 位 `Arm`。
   - 如果环境类型为 `WINDOWS_EC2`，请确保 AMI 架构为 64 位 `x86`。
   - AMI 允许 CodeBuild 服务组织 ARN。有关组织 ARN 的列表，请参阅Amazon Machine Images (AMI)。
   - 如果 AMI 使用 AWS KMS 密钥加密，则该 AWS KMS 密钥还必须允许 CodeBuild 服务组织 ID 。有关组织 ID 的列表，请参阅Amazon Machine Images (AMI)。有关 AWS KMS 密钥的更多信息，请参阅 Amazon EC2 用户指南中的允许组织和组织单位使用 KMS 密钥。要授予 CodeBuild 组织使用 KMS 密钥的权限，请在密钥策略中添加以下语句：

```
{
    "Sid": "Allow access for organization root",
    "Effect": "Allow",
    "Principal": "*",
    "Action": [
        "kms:Describe*",
        "kms:List*",
        "kms:Get*",
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:CreateGrant"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
```

```
            "aws:PrincipalOrgID": "o-123example"
        }
    }
}
```

- 在实例集服务角色字段中，授予以下 Amazon EC2 权限：

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
               "ec2:DescribeImages",
               "ec2:DescribeSnapshots"
            ],
            "Resource": "*"
        }
    ]
}
```

预留容量实例集的局限性

在预留容量实例集不支持的某些用例中，如果它们对您产生影响，请改用按需实例集：

- 预留容量实例集不支持构建利用率指标。
- macOS 预留容量实例集不支持调试会话。

有关限制和限额的更多信息，请参阅计算实例集。

预留容量实例集属性

预留容量实例集包含以下属性。有关预留容量实例集的更多信息，请参阅在预留容量实例集上运行构建。

操作系统

操作系统 以下操作系统可用：

- Amazon Linux

- macOS
- Windows Server 2019
- Windows Server 2022

架构

处理器架构。以下架构可用：

- x86_64
- Arm64

环境类型

选择 Amazon Linux 时可用的环境类型。以下环境类型可用：

- Linux EC2
- Linux GPU

计算实例类型

实例集实例的计算配置。

引导式选择

通过选择 vCPU、内存和磁盘空间设置来指定不同的计算类型。有关按区域划分的计算类型可用性的信息，请参阅关于预留容量环境类型。

自定义实例

手动指定所需的实例类型。

Capacity

分配给实例集的计算机的初始数量，它定义了可以并行运行的构建数量。

溢出行为

定义构建数量超过实例集容量时的行为。

On-demand

Overflow 版本按 CodeBuild 需运行。

> **Note**
>
> 如果您选择在创建 VPC-connected 队列时将溢出行为设置为按需运行，请确保为项目服务角色添加所需的 VPC 权限。有关更多信息，请参阅允许 CodeBuild 访问创建 VPC 网络接口所需的 AWS 服务的示例策略声明。

> **Important**
>
> 如果您选择将溢出行为设置为按需，请注意，溢出构建将单独计费，类似于按需型 Amazon EC2。有关更多信息，请参阅 https://aws.amazon.com/codebuild/pricing/。

队列

构建运行将放在队列中，直到有计算机可用。这限制了额外成本，因为没有分配额外的计算机。

亚马逊机器映像（AMI）

实例集的亚马逊机器映像（AMI）属性。支持以下属性 CodeBuild：

AWS 区域	组织 ARN	组织 ID
us-east-1	arn:aws:organizations::851725618577:organization/o-c6wcu152r1	o-c6wcu152r1
us-east-2	arn:aws:organizations::992382780434:organization/o-seufr2suvq	o-seufr2suvq
us-west-2	arn:aws:organizations::381491982620:organization/o-0412o99a4r	o-0412o99a4r

AWS 区域	组织 ARN	组织 ID
ap-northeast-1	arn:aws:organizations::891376993293:organization/o-b6k3sjqavm	o-b6k3sjqavm
ap-south-1	arn:aws:organizations::891376924779:organization/o-krtah1lkeg	o-krtah1lkeg
ap-southeast-1	arn:aws:organizations::654654522137:organization/o-mcn8uvc3tp	o-mcn8uvc3tp
ap-southeast-2	arn:aws:organizations::767398067170:organization/o-6crt0f6bu4	o-6crt0f6bu4
eu-central-1	arn:aws:organizations::590183817084:organization/o-lb2lne3te6	o-lb2lne3te6
eu-west-1	arn:aws:organizations::891376938588:organization/o-ullrrg5qf0	o-ullrrg5qf0
sa-east-1	arn:aws:organizations::533267309133:organization/o-db63c45ozw	o-db63c45ozw

其他配置

VPC - 可选

您的 CodeBuild 队列将访问的 VPC。有关更多信息，请参阅 AWS CodeBuild 与亚马逊 Virtual Private Cloud 配合使用。

> **Note**
>
> 如果在调用 StartBuild API 时指定了队列替代，则 CodeBuild 将忽略项目 VPC 配置。

子网

CodeBuild 用于设置 VPC 配置的 VPC 子网。请注意，预留容量队列在单个可用区中仅支持一个子网。此外，确保您的子网包括 NAT 网关。

安全组

与您的 VPC 一起 CodeBuild 使用的 VPC 安全组。确保您的安全组允许出站连接。

实例集服务角色

根据您账户中的现有服务角色为您的实例集定义服务角色。

定义代理配置 - 可选

对预留容量实例应用网络访问控制的代理配置。有关更多信息，请参阅 AWS CodeBuild 与托管代理服务器一起使用。

> **Note**
>
> 代理配置不支持 VPC、Windows 或 macOS。

默认行为

定义传出流量的行为。

允许

默认情况下，允许流向所有目标的传出流量。

拒绝

默认情况下，拒绝流向所有目标的传出流量。

代理规则

指定允许或拒绝网络访问控制的目标域或 IP。

预留容量样本包括 AWS CodeBuild

这些样本可用于对预留容量舰队进行试验。 CodeBuild

主题

- 使用预留容量进行缓存示例

使用预留容量进行缓存示例

缓存可以存储构建环境的可重用部分，并在多个构建中使用它们。此示例演示了如何使用预留容量在构建项目中启用缓存。有关更多信息，请参阅 缓存构建以提高性能。

您可以先在项目设置中指定一种或多种缓存模式：

```
Cache:
        Type: LOCAL
        Modes:
          - LOCAL_CUSTOM_CACHE
          - LOCAL_DOCKER_LAYER_CACHE
          - LOCAL_SOURCE_CACHE
```

> **Note**
>
> 要使用 Docker 层缓存，请务必启用特权模式。

您的项目构建规范设置应如下所示：

```
version: 0.2
      phases:
        build:
          commands:
```

```
            - echo testing local source cache
            - touch /codebuild/cache/workspace/foobar.txt
            - git checkout -b cached_branch
            - echo testing local docker layer cache
            - docker run alpine:3.14 2>&1 | grep 'Pulling from' || exit 1
            - echo testing local custom cache
            - touch foo
            - mkdir bar && ln -s foo bar/foo2
            - mkdir bar/bar && touch bar/bar/foo3 && touch bar/bar/foo4
            - "[ -f foo ] || exit 1"
            - "[ -L bar/foo2 ] || exit 1"
            - "[ -f bar/bar/foo3 ] || exit 1"
            - "[ -f bar/bar/foo4 ] || exit 1"
      cache:
        paths:
           - './foo'
           - './bar/**/*'
           - './bar/bar/foo3'
```

您可以先用新项目运行构建，为缓存做种子。完成后，您应该使用重写的构建规范开始另一个构建，如下所示：

```
version: 0.2
      phases:
        build:
          commands:
            - echo testing local source cache
            - git branch | if grep 'cached_branch'; then (exit 0); else (exit 1); fi
            - ls /codebuild/cache/workspace | if grep 'foobar.txt'; then (exit 0); else
 (exit 1); fi
            - echo testing local docker layer cache
            - docker run alpine:3.14 2>&1 | if grep 'Pulling from'; then (exit 1); else
 (exit 0); fi
            - echo testing local custom cache
            - "[ -f foo ] || exit 1"
            - "[ -L bar/foo2 ] || exit 1"
            - "[ -f bar/bar/foo3 ] || exit 1"
            - "[ -f bar/bar/foo4 ] || exit 1"
      cache:
        paths:
           - './foo'
           - './bar/**/*'
           - './bar/bar/foo3'
```

批量运行构建

您可以使用批量生成 AWS CodeBuild 来运行项目的并行和协调构建。

主题

- 安全角色
- 批量构建类型
- 批量报告模式
- 更多信息

安全角色

批量构建为批量配置引入了全新的安全角色。这个新角色是必需的，因为 CodeBuild 必须能够代表你调用`StartBuild`、`StopBuild`和`RetryBuild`操作才能作为批处理的一部分运行构建。客户应该使用新角色，而不是他们在构建中使用的角色，原因有两个：

- 向构建角色授予 `StartBuild`、`StopBuild` 和 `RetryBuild` 权限后，将允许单个构建通过 buildspec 启动多个构建。
- CodeBuild 批量编译提供的限制限制了批处理中可用于构建的编译数量和计算类型。如果构建角色拥有这些权限，则构建本身就有可能绕过这些限制。

批量构建类型

CodeBuild 支持以下批量生成类型：

批量构建类型

- 构建图
- 构建列表
- 构建矩阵
- 构建扇出

构建图

构建图定义了一组任务，这些任务依赖于批量处理中的其他任务。

以下示例定义了构建图，展示如何创建依赖项链。

```
batch:
  fast-fail: false
  build-graph:
    - identifier: build1
      env:
        variables:
          BUILD_ID: build1
      ignore-failure: false
    - identifier: build2
      buildspec: build2.yml
      env:
        variables:
          BUILD_ID: build2
      depend-on:
        - build1
    - identifier: build3
      env:
        variables:
          BUILD_ID: build3
      depend-on:
        - build2
    - identifier: build4
      env:
        compute-type: ARM_LAMBDA_1GB
    - identifier: build5
      env:
        fleet: fleet_name
```

在本示例中：

- 先运行 build1，因为它没有依赖项。
- 由于 build2 对 build1 存在依赖关系，因此 build2 会在完成 build1 后运行。
- 由于 build3 对 build2 存在依赖关系，因此 build3 会在完成 build2 后运行。

有关构建图 buildspec 语法的更多信息，请参阅batch/build-##。

构建列表

构建列表定义了多个并行运行的任务。

以下示例定义了构建列表。build1 和 build2 构建将并行运行。

```
batch:
  fast-fail: false
  build-list:
    - identifier: build1
      env:
        variables:
          BUILD_ID: build1
      ignore-failure: false
    - identifier: build2
      buildspec: build2.yml
      env:
        variables:
          BUILD_ID: build2
      ignore-failure: true
    - identifier: build3
      env:
        compute-type: ARM_LAMBDA_1GB
    - identifier: build4
      env:
        fleet: fleet_name
    - identifier: build5
      env:
        compute-type: GENERAL_LINUX_XLAGRE
```

有关构建列表 buildspec 语法的更多信息，请参阅batch/build-##。

构建矩阵

编译矩阵定义了并行运行的具有不同配置的任务。 CodeBuild 为每种可能的配置组合创建单独的构建。

以下示例显示了一个具有两个 buildspec 文件和三个环境变量值的构建矩阵。

```
batch:
  build-matrix:
    static:
      ignore-failure: false
    dynamic:
      buildspec:
        - matrix1.yml
        - matrix2.yml
      env:
```

```
        variables:
          MY_VAR:
            - VALUE1
            - VALUE2
            - VALUE3
```

在此示例中， CodeBuild 创建了六个构建：

- matrix1.yml 与 $MY_VAR=VALUE1
- matrix1.yml 与 $MY_VAR=VALUE2
- matrix1.yml 与 $MY_VAR=VALUE3
- matrix2.yml 与 $MY_VAR=VALUE1
- matrix2.yml 与 $MY_VAR=VALUE2
- matrix2.yml 与 $MY_VAR=VALUE3

每个构建都将具有以下设置：

- ignore-failure 设置为 false
- env/type 设置为 LINUX_CONTAINER
- env/image 设置为 aws/codebuild/amazonlinux-x86_64-standard:4.0
- env/privileged-mode 设置为 true

这些构建并行运行。

有关构建矩阵 buildspec 语法的更多信息，请参阅batch/build-##。

构建扇出

构建扇出定义的任务将在批量中拆分为多个构建。这可以用于并行运行测试。 CodeBuild 根据parallelism字段中设置的值为每个测试用例分片创建单独的构建。

以下示例定义了一个构建扇出，用于创建五个并行运行的构建。

```
version: 0.2

batch:
   fast-fail: false
   build-fanout:
```

```
     parallelism: 5
     ignore-failure: false

phases:
  install:
    commands:
      - npm install
   build:
    commands:
      - mkdir -p test-results
      - cd test-results
      - |
        codebuild-tests-run \
         --test-command 'npx jest --runInBand --coverage' \
         --files-search "codebuild-glob-search '**/test/**/*.test.js'" \
         --sharding-strategy 'equal-distribution'
```

在此示例中，假设有 100 个测试需要运行，则 CodeBuild 创建了五个构建，每个版本并行运行 20 个测试。

有关构建图 buildspec 语法的更多信息，请参阅batch/build-fanout。

批量报告模式

如果项目的源提供商是 Bitbucket 或 E GitHub nterprise，并且您的项目配置为向源提供商报告构建状态，则您可以选择如何将批量构建状态发送给源提供商。 GitHub您可以选择将状态作为批处理的单个汇总状态报告发送，也可以单独报告批处理中每个构建的状态。

有关更多信息，请参阅以下主题：

- 批量配置（创建）
- 批量配置（更新）

更多信息

有关更多信息，请参阅以下主题：

- 批量构建 buildspec 参考
- 批量配置
- 运行批量构建 (AWS CLI)

- 停止批量构建 AWS CodeBuild

在批量构建中执行并行测试

您可以使用 AWS CodeBuild 在批量构建中执行并行测试。并行测试执行是一种测试方法，其中多个测试用例在不同的环境、计算机或浏览器上同时运行，而不是按顺序执行。这种方法可以显著缩短总体测试执行时间并提高测试效率。在中 CodeBuild，您可以将测试拆分到多个环境中并行运行。

并行测试执行的主要优势包括：

1. 缩短执行时间：按顺序需要数小时的测试可以在几分钟内完成。
2. 提高资源利用率：高效地利用可用的计算资源。
3. 更早的反馈：更快地完成测试意味着可以更快地向开发人员提供反馈。
4. Cost-effective-从长远来看，可以节省时间和计算成本。

在实施并行测试执行时，通常会考虑两种主要方法：独立环境和多线程。虽然这两种方法都旨在实现并发测试执行，但它们在实施和有效性方面存在显著差异。独立环境会创建隔离的实例，其中每个测试套件独立运行，而多线程使用不同的线程在同一个进程空间内同时执行多个测试。

与多线程相比，独立环境的主要优势包括：

1. 隔离：每个测试都在完全隔离的环境中运行，以防止测试之间的干扰。
2. 资源冲突：不存在多线程中经常发生的共享资源竞争。
3. 稳定性：不太容易出现争用条件和同步问题。
4. 更易于调试：当测试失败时，由于每个环境都是独立的，因此更容易确定原因。
5. 状态管理：轻松管理困扰多线程测试的共享状态问题。
6. 更好的可扩展性：可以轻松地添加更多环境，而不会增加复杂性。

主题

- 中的支持 AWS CodeBuild
- 在批量构建中启用并行测试执行
- 使用 codebuild-tests- run CLI 命令
- 使用 codebuild-glob- search CLI 命令
- 关于测试拆分

- 自动合并各个构建报告
- 各种测试框架的并行测试执行示例

中的支持 AWS CodeBuild

AWS CodeBuild 通过其批量构建功能为并行测试执行提供强大支持，该功能专为利用单独的环境执行而设计。这种实施与隔离测试环境的优点完美吻合。

使用测试分配进行批量构建

CodeBuild的批量构建功能允许创建多个同时运行的构建环境。每个环境都作为一个完全独立的单元运行，有自己的计算资源、运行时环境和依赖项。通过批量构建配置，您可以指定它们需要多少个并行环境，以及如何在这些环境间分配测试。

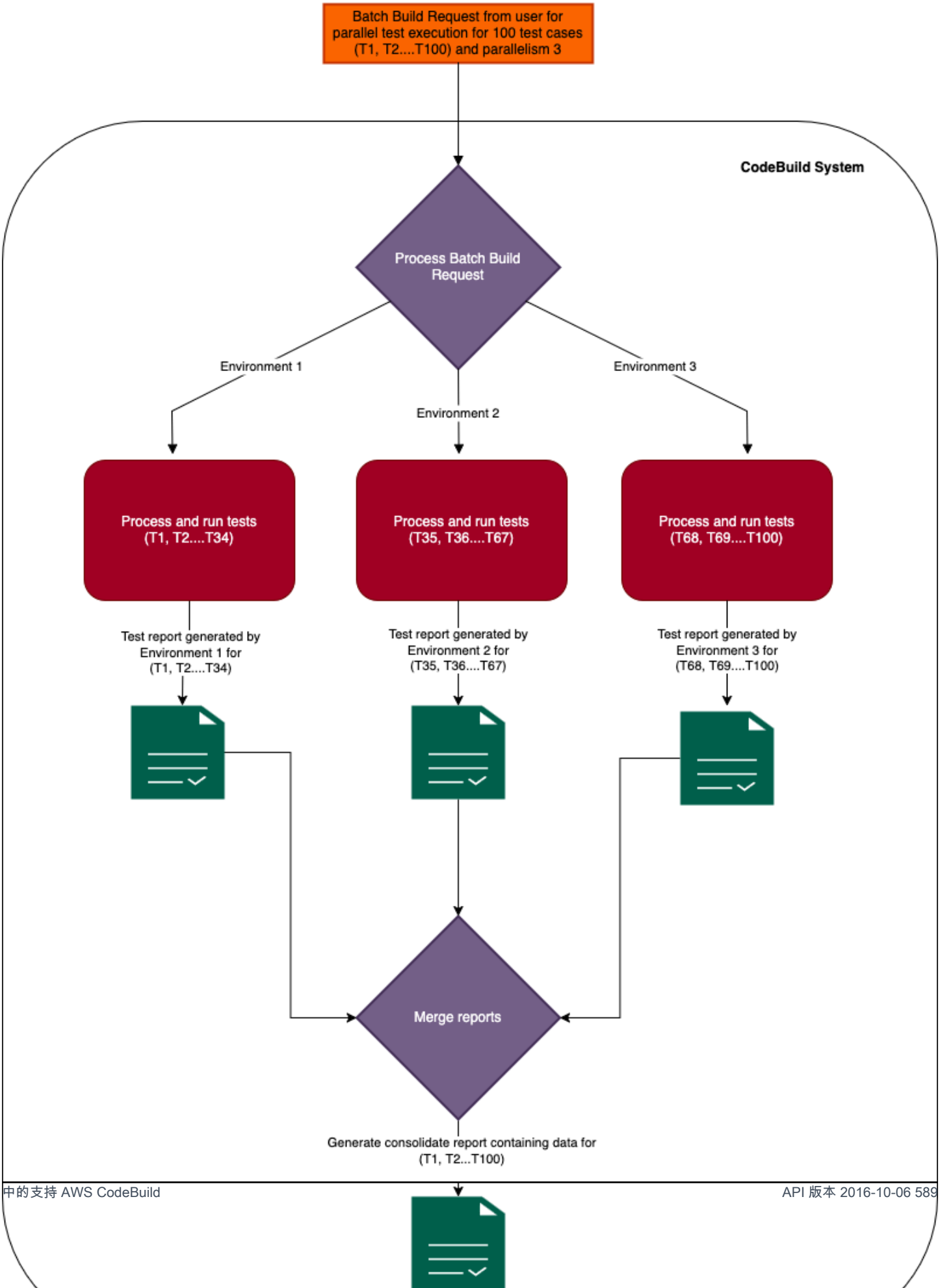
测试分片 CLI

CodeBuild 通过其 CLI 工具包括内置的测试分发机制`codebuild-tests-run`，该机制可自动将测试划分到不同的环境中。

报告聚合

的实现的关键优势之一 CodeBuild是它能够无缝处理测试结果汇总。当测试在不同的环境中执行时， CodeBuild 会自动收集来自每个环境的测试报告并将其合并为批量构建级别的统一测试报告。这种整合提供了测试结果的全面视图，同时保持了并行执行的效率优势。

下图解释了中执行并行测试的完整概念 AWS CodeBuild。


Batch Build Request from user for parallel test execution for 100 test cases (T1, T2....T100) and parallelism 3
CodeBuild System
Process Batch Build Request
Environment 1
Environment 2
Environment 3
Process and run tests (T1, T2....T34)
Process and run tests (T35, T36....T67)
Process and run tests (T68, T69....T100)
Test report generated by Environment 1 for (T1, T2....T34)
Test report generated by Environment 2 for (T35, T36....T67)
Test report generated by Environment 3 for (T68, T69....T100)
Merge reports
Generate consolidate report containing data for (T1, T2...T100)

在批量构建中启用并行测试执行

要并行运行测试，请更新批量构建 buildspec 文件，以包含构建扇出字段和要在 parallelism 字段中拆分测试套件的并行构建的数量，如下所示。该parallelism字段指定设置了多少独立执行者来执行测试套件。

要在多个并行执行环境中运行测试，请将 parallelism 字段设置为大于零的值。在下面的示例中，设置parallelism为五，这意味着 CodeBuild 启动五个相同的构建，并行执行测试套件的一部分。

可以使用 codebuild-tests-run CLI 命令来拆分和运行测试。测试文件将被拆分，并且将在每个构建中运行测试的一部分。这减少了运行完整测试套件所花的总时间。在以下示例中，测试将分成五个部分，并根据测试的名称计算拆分点。

```
version: 0.2

batch:
  fast-fail: false
  build-fanout:
    parallelism: 5
    ignore-failure: false

phases:
  install:
    commands:
      - npm install jest-junit --save-dev
  pre_build:
    commands:
      - echo 'prebuild'
  build:
    commands:
      - |
        codebuild-tests-run \
         --test-command 'npx jest --runInBand --coverage' \
         --files-search "codebuild-glob-search '**/_tests_/**/*.test.js'" \
         --sharding-strategy 'equal-distribution'

  post_build:
    commands:
      - codebuild-glob-search '**/*.xml'
      - echo "Running post-build steps..."
      - echo "Build completed on `date`"
```

```
reports:
  test-reports:
    files:
      - '**/junit.xml'
    base-directory: .
    discard-paths: yes
    file-format: JUNITXML
```

如果为 build-fanout 版本配置了报告，则会分别为每个版本生成测试报告，可以在控制台中相应版本的“报告” 选项卡下查看这些报告。 AWS CodeBuild

有关如何批量执行并行测试的更多信息，请参阅[各种测试框架的并行测试执行示例]。

使用 **codebuild-tests-** run CLI 命令

AWS CodeBuild 提供了一个 CLI，它将测试命令和测试文件位置作为输入。带有此输入的 CLI 将根据测试文件名将测试拆分为`parallelism`字段中指定的分片数。将测试文件分配给分片由分片策略决定。

```
codebuild-tests-run \
    --files-search "codebuild-glob-search '**/__tests__/*.js'" \
    --test-command 'npx jest --runInBand --coverage' \
    --sharding-strategy 'equal-distribution'
```

下表说明了 `codebuild-tests-run` CLI 命令的字段。

字段名称	Type	必需或可选	定义
`test-command`	字符串	必填	此命令用于运行测试。
`files-search`	字符串	必填	此命令提供了测试文件列表。您可以使用 AWS CodeBuild 提供的 codebuild-glob-search CLI 命令或您选择的任何其他文件搜索工具。

字段名称	Type	必需或可选	定义
			Note 确保 `files-search` 命令输出文件名，每个文件名用换行分隔。
`sharding-strategy`	枚举	可选	有效值：`equal-distribution`（默认值）、`stability` • `equal-distribution`：根据测试文件名均匀地对测试文件进行分片。 • `stability`：使用一致的文件名哈希对测试文件进行分片。 有关更多信息，请参阅 关于测试拆分。

codebuild-tests-run CLI 首先使用 files-search 参数中提供的命令识别测试文件列表。然后，它使用指定的分片策略确定为当前分片（环境）指定的测试文件子集。最后，测试文件的这一子集格式化为以空格分隔的列表，并在执行之前附加到在 test-command 参数中提供的命令的末尾。

对于不接受空格分隔列表的测试框架，codebuild-tests-run CLI 通过 CODEBUILD_CURRENT_SHARD_FILES 环境变量提供了一种灵活的替代方案。此变量包含为当前构建分片指定的测试文件路径的换行分隔列表。通过利用此环境变量，您可以轻松地适应各种测试框架要求，并适应那些期望输入格式不同于空格分隔列表的要求。此外，还可以根据测试框架的需要格式化测

试文件名。以下是在 Linux 上通过 Django 框架使用 CODEBUILD_CURRENT_SHARD_FILES 的示例。此处，CODEBUILD_CURRENT_SHARD_FILES 用于获取 Django 支持的点符号文件路径：

```
codebuild-tests-run \
    –files-search "codebuild-glob-search '/tests/test_.py'" \
    –test-command 'python3 manage.py test $(echo "$CODEBUILD_CURRENT_SHARD_FILES" | sed
 -E "s/\//__/g; s/\.py$//; s/__/./g")' \
    –sharding-strategy 'equal-distribution'
```

> **Note**
>
> 请注意，CODEBUILD_CURRENT_SHARD_FILES 环境变量只能在 codebuild-tests-run CLI 的范围内使用。
> 另外，如果您在测试命令内使用 CODEBUILD_CURRENT_SHARD_FILES，请将 CODEBUILD_CURRENT_SHARD_FILES 放在双引号内，如上面的示例所示。

使用 **codebuild-glob-** search CLI 命令

AWS CodeBuild 提供了一个名为的内置 CLI 工具codebuild-glob-search，允许您基于一个或多个 glob 模式在工作目录中搜索文件。当您要对项目中的特定文件或目录运行测试时，此工具可能特别有用。

用量

codebuild-glob-search CLI 的使用语法如下：

```
codebuild-glob-search <glob_pattern1> [<glob_pattern2> ...]
```

- *<glob_pattern1>*、*<glob_pattern2>* 等：一个或多个全局模式，用于与工作目录中的文件进行匹配。
- *：匹配任何字符序列（路径分隔符除外）。
- **：匹配任何字符序列（包括路径分隔符）。

> **Note**
>
> 确保全局字符串带有引号。要检查模式匹配的结果，请使用 echo 命令。

```
version: 0.2

phases:
  build:
    commands:
      - echo $(codebuild-glob-search '**/__tests__/*.js')
      - codebuild-glob-search '**/__tests__/*.js' | xargs -n 1 echo
```

Output

CLI 将输出与所提供的全局模式相匹配的文件路径的换行分隔列表。返回的文件路径将是工作目录的相对路径。

如果找不到与提供的模式相匹配的文件，CLI 将输出一条消息，指出未找到任何文件。

请注意，将从搜索结果中排除由于任何给定模式而找到的目录。

示例

如果只想搜索测试目录及其子目录中带有 `.js` 扩展名的文件，则可以在 `codebuild-glob-search` CLI 中使用以下命令：

```
codebuild-glob-search '**/__tests__/*.js'
```

该命令将在`__tests__`目录及其子目录中搜索所有`.js`扩展名为的文件，如模式所示。

关于测试拆分

AWS CodeBuild的测试拆分功能可让您在多个计算实例上并行执行测试套件，从而缩短总体测试运行时间。此功能是通过 CodeBuild 项目设置中的批处理配置和 buildspec 文件中的`codebuild-tests-run`实用程序启用的。

根据指定的分片策略对测试进行拆分。 CodeBuild 提供两种分片策略，如下所示：

Equal-distribution

`equal-distribution` 分片策略根据测试文件名的字母顺序将测试划分到并行构建中。这种方法首先对测试文件进行排序，然后使用基于分块的方法来分配它们，从而确保将相似的文件组合在一起进行测试。建议在处理相对较小的测试文件集时使用。虽然此方法旨在为每个分片分配大致相等

数量的文件，且最大差异为 1，但它并不能保证稳定性。在后续构件中添加或移除测试文件时，现有文件的分配可能会发生变化，这可能导致在分片之间重新分配。

稳定性

`stability` 分片策略采用一致的哈希算法在分片之间拆分测试，确保文件分配保持稳定。添加或移除新文件时，此方法可确保现有的文件到分片分配基本保持不变。对于大型测试套件，建议使用稳定性选项将测试均匀地分配到各个分片。该机制旨在提供近乎相等的分配，确保每个分片接收的文件数量相似，且差异最小。虽然稳定性策略不能保证理想的平等分配，但它提供了近乎相等的分配，即使在添加或移除文件时也能保持各构建之间文件分配的一致性。

要启用测试拆分，您需要在 CodeBuild 项目设置中配置批处理部分，指定所需`parallelism`级别和其他相关参数。此外，还需要在 buildspec 文件中包含 `codebuild-tests-run` 实用程序以及相应的测试命令和拆分方法。

自动合并各个构建报告

在 fanout 批量构建中， AWS CodeBuild 支持将单个构建报告自动合并为合并的批次级报告。此功能提供一个批量中所有构建的测试结果和代码覆盖的全面视图。

工作原理

执行`fanout`批量生成时，每个单独的生成都会生成测试报告。 CodeBuild 然后自动将来自不同版本的相同报告合并到一个统一的报告中，该报告附加到批量生成中。这些合并报告可通过 BatchGetBuildBatches API 的`reportArns`字段轻松访问，也可以在控制台的“报告”选项卡中查看。这种合并功能也扩展到自动发现的报告。

合并报告是在编译规范中指定或由自动发现的报告组下创建的。 CodeBuild您可以直接在这些报告组下分析合并报告的趋势，从而为跨同一批量构建项目的历史构建的整体构建性能和质量指标提供宝贵的见解。

对于批次中的每个单独构建， CodeBuild 会自动创建单独的报告组。这些报告组遵循特定的命名约定，将批量构建报告组名称与 `BuildFanoutShard<shard_number>` 后缀组合在一起，其中 `shard_number` 表示在其中创建报告组的分片编号。这种整理方式可让您跟踪和分析整合构建级别和各个构建级别的趋势，从而可以灵活地监控和评估其构建过程。

批量构建报告遵循与各个构建报告相同的结构。报告选项卡中的以下关键字段特定于批量构建报告：

批量构建报告状态

批量构建报告的状态遵循特定的规则，具体取决于报告类型：

- 测试报告：
  - 成功：当所有单独的构建报告都成功时，状态设置为成功。
  - 失败：如果任何单个构建报告失败，则状态将设置为失败。
  - 未完成：如果有任何单个构建报告缺失或状态为未完成，则状态将标记为未完成。
- 代码覆盖率报告：
  - 完成：当所有单独的构建报告都完成时，状态设置为完成。
  - 失败：如果任何单个构建报告失败，则状态将设置为失败。
  - 未完成：如果有任何单个构建报告缺失或状态为未完成，则状态将标记为未完成。

测试摘要

合并测试报告整合了所有单个构建报告中的以下字段：

- duration-in-nano-seconds：所有单个构建报告中的最大测试持续时间（以纳秒为单位）。
- total：所有测试用例的总数，即每个构建中的测试总数之和。
- status-counts：提供测试状态（例如“通过”、“失败”或“跳过”）的整合视图，这些状态是通过汇总所有单个构建中每种状态类型的计数计算得出的。

代码覆盖摘要

合并的代码覆盖报告使用以下计算方法合并了所有单个构建中的字段：

- branches-covered：各个报告中所有覆盖的分支的总和。
- branches-missed：各个报告中所有错过的分支的总和。
- branch-coverage-percentage：`(Total covered branches / Total branches) * 100`
- lines-covered：各个报告中所有覆盖的行的总和。
- lines-missed：各个报告中所有错过的行的总和。
- lines-coverage-percentage：`(Total covered lines / Total lines) * 100`

执行 ID

批量构建 ARN。

测试用例

合并后的报告包含来自各个版本的所有测试用例的综合列表，可通过 DescribeTestCases API 和控制台中的批量构建报告进行访问。

代码覆盖

合并后的代码覆盖率报告为所有单独编译的每个文件提供了合并的行和分支覆盖率信息，可通过 DescribeCodeCoverages API 和控制台中的批量生成报告进行访问。注意：对于分布在不同分片上的多个测试文件所涵盖的文件，合并报告使用以下选择标准：

1. 主要选择基于分片中最高的行覆盖率。
2. 如果多个分片上的行覆盖率相等，则会选择分支覆盖率最高的分片。

各种测试框架的并行测试执行示例

您可以使用 `codebuild-tests-run` CLI 命令在各并行执行环境之间拆分和运行测试。下一节提供了各种框架下的 `buildspec.yml` 示例，其中展示了 `codebuild-tests-run` 命令的用法。

- 下面每个示例包含的 `parallelism` 级别均为五，这意味着，系统将创建五个相同的执行环境来拆分测试。您可以通过修改 `build-fanout` 部分中的 `parallelism` 值来选择适合您项目的 `parallelism` 级别。
- 以下每个示例都显示了将测试配置为按测试文件名（默认值）拆分。这样可以将测试均衡地分布在各并行执行环境之间。

在开始之前，请参阅在批量构建中执行并行测试以了解更多信息。

有关使用 `codebuild-tests-run` CLI 命令时的完整选项列表，请参阅使用 `codebuild-tests-run` CLI 命令。

主题

- 使用 Django 配置并行测试
- 使用 Elixir 配置并行测试
- 使用 Go 配置并行测试
- 使用 Java 配置并行测试（Maven）
- 使用 Javascript 配置并行测试（Jest）
- 使用 Kotlin 配置并行测试
- 使用 PHPUnit 配置并行测试
- 使用 Pytest 配置并行测试
- 使用 Ruby 配置并行测试（Cucumber）
- 使用 Ruby 配置并行测试（RSpec）

使用 Django 配置并行测试

下面是一个 buildspec.yml 示例，显示了如何在 Ubuntu 平台上使用 Django 并行执行测试：

```
version: 0.2

batch:
  fast-fail: false
  build-fanout:
    parallelism: 5

phases:
  install:
    commands:
      - echo 'Installing Python dependencies'
      - sudo yum install -y python3 python3-pip
      - python3 -m ensurepip --upgrade
      - python3 -m pip install django
  pre_build:
    commands:
      - echo 'Prebuild'
  build:
    commands:
      - echo 'Running Django Tests'
      - |
        codebuild-tests-run \
         --test-command 'python3 manage.py test $(echo "$CODEBUILD_CURRENT_SHARD_FILES"
 | sed -E "s/\//__/g; s/\.py$//; s/__/./g")' \
         --files-search "codebuild-glob-search '**/tests/*test_*.py'" \
         --sharding-strategy 'equal-distribution'
  post_build:
    commands:
      - echo 'Test execution completed'
```

上面的示例显示了环境变量 CODEBUILD_CURRENT_SHARD_FILES 的用法。此处，CODEBUILD_CURRENT_SHARD_FILES 用于获取 Django 支持的点式标记法文件路径。如上所示，请将 CODEBUILD_CURRENT_SHARD_FILES 括在双引号中。

使用 Elixir 配置并行测试

以下 buildspec.yml 示例显示在 Ubuntu 平台上使用 Elixir 并行执行测试：

```
version: 0.2
```

```
batch:
  fast-fail: false
  build-fanout:
    parallelism: 5

phases:
  install:
    commands:
      - echo 'Installing Elixir dependencies'
      - sudo apt update
      - sudo DEBIAN_FRONTEND=noninteractive apt install -y elixir
      - elixir --version
      - mix --version
  pre_build:
    commands:
      - echo 'Prebuild'
  build:
    commands:
      - echo 'Running Elixir Tests'
      - |
        codebuild-tests-run \
         --test-command 'mix test' \
         --files-search "codebuild-glob-search '**/test/**/*_test.exs'" \
         --sharding-strategy 'equal-distribution'
  post_build:
    commands:
      - echo "Test execution completed"
```

使用 Go 配置并行测试

以下是显示在 Linux 平台上使用 Go 执行并行测试的示例：buildspec.yml

```
version: 0.2

batch:
  fast-fail: false
  build-fanout:
    parallelism: 5
    ignore-failure: false

phases:
  install:
```

```
    commands:
      - echo 'Fetching Go version'
      - go version
  pre_build:
    commands:
      - echo 'prebuild'
  build:
    commands:
      - echo 'Running go Tests'
      - go mod init calculator
      - cd calc
      - |
        codebuild-tests-run \
         --test-command "go test -v calculator.go" \
         --files-search "codebuild-glob-search '**/*test.go'"
  post_build:
    commands:
      - echo "Test execution completed"
```

在上面的示例中，calculator.go函数包含用于测试的简单数学函数，所有测试文件和calculator.go文件都在calc文件夹中。

使用 Java 配置并行测试（Maven）

以下是显示在 Linux 平台上使用 Java 执行并行测试的示例：buildspec.yml

```
version: 0.2

batch:
  fast-fail: false
  build-fanout:
    parallelism: 5
    ignore-failure: false

phases:
  pre_build:
    commands:
      - echo 'prebuild'
  build:
    commands:
      - echo "Running mvn test"
      - |
        codebuild-tests-run \
```

```
          --test-command 'mvn test -Dtest=$(echo "$CODEBUILD_CURRENT_SHARD_FILES" | sed
 "s|src/test/java/||g; s/\.java//g; s|/|.|g; s/ /,/g" | tr "\n" "," | sed "s/,$//")' \
          --files-search "codebuild-glob-search '**/test/**/*.java'"

  post_build:
    commands:
      - echo "Running post-build steps..."
      - echo "Test execution completed"
```

在给定的示例中，环境变量 CODEBUILD_CURRENT_SHARD_FILES 包含当前分片中的测试文件，用换行分隔。这些文件会按照 Maven 的 -Dtest 参数接受的格式转换为一个以逗号分隔的类名称列表。

使用 Javascript 配置并行测试（Jest）

以下 buildspec.yml 示例显示在 Ubuntu 平台上使用 Javascript 并行执行测试：

```
version: 0.2

batch:
  fast-fail: true
  build-fanout:
    parallelism: 5
    ignore-failure: false

phases:
  install:
    commands:
      - echo 'Installing Node.js dependencies'
      - apt-get update
      - apt-get install -y nodejs
      - npm install
      - npm install --save-dev jest-junit
  pre_build:
    commands:
      - echo 'prebuild'
  build:
    commands:
      - echo 'Running JavaScript Tests'
      - |
         codebuild-tests-run \
          --test-command "npx jest" \
          --files-search "codebuild-glob-search '**/test/**/*.test.js'" \
          --sharding-strategy 'stability'
```

```
    post_build:
      commands:
        - echo 'Test execution completed'
```

使用 Kotlin 配置并行测试

以下是显示在 Linux 平台上使用 Kotlin 执行并行测试的示例：`buildspec.yml`

```
version: 0.2

batch:
  fast-fail: false
  build-fanout:
    parallelism: 2
    ignore-failure: false

phases:
  install:
    runtime-versions:
      java: corretto11
    commands:
      - echo 'Installing dependencies'
      - KOTLIN_VERSION="1.8.20" # Replace with your desired version
      - curl -o kotlin-compiler.zip -L "https://github.com/JetBrains/kotlin/releases/
download/v${KOTLIN_VERSION}/kotlin-compiler-${KOTLIN_VERSION}.zip"
      - unzip kotlin-compiler.zip -d /usr/local
      - export PATH=$PATH:/usr/local/kotlinc/bin
      - kotlin -version
      - curl -O https://repo1.maven.org/maven2/org/junit/platform/junit-platform-
console-standalone/1.8.2/junit-platform-console-standalone-1.8.2.jar
  pre_build:
    commands:
      - echo 'prebuild'
  build:
    commands:
      - echo 'Running Kotlin Tests'
      - |
        codebuild-tests-run \
          --test-command 'kotlinc src/main/kotlin/*.kt $(echo
 "$CODEBUILD_CURRENT_SHARD_FILES" | tr "\n" " ") -d classes -cp junit-platform-console-
standalone-1.8.2.jar' \
          --files-search "codebuild-glob-search 'src/test/kotlin/*.kt'"
      - |
```

```
        codebuild-tests-run \
          --test-command '
            java -jar junit-platform-console-standalone-1.8.2.jar --class-path classes
 \
              $(for file in $CODEBUILD_CURRENT_SHARD_FILES; do
                 class_name=$(basename "$file" .kt)
                 echo "--select-class $class_name"
               done)
          ' \
          --files-search "codebuild-glob-search 'src/test/kotlin/*.kt'"
  post_build:
    commands:
      - echo "Test execution completed"
```

在上面的示例中，使用了 codebuild-tests-run CLI 两次。在第一次运行期间，kotlinc 编译文件。CODEBUILD_CURRENT_SHARD_FILES 变量会检索分配给当前分片的测试文件，然后将其转换为以空格分隔的列表。在第二次运行中，JUnit 执行测试。同样，CODEBUILD_CURRENT_SHARD_FILES 会获取分配给当前分片的测试文件，但这次它们会转换为类名称。

使用 PHPUnit 配置并行测试

以下是显示在 Linux 平台buildspec.yml上使用 PHPUnit 执行并行测试的示例：

```
version: 0.2

batch:
   fast-fail: false
   build-fanout:
     parallelism: 5
     ignore-failure: false

phases:
   install:
     commands:
       - echo 'Install dependencies'
       - composer require --dev phpunit/phpunit
   pre_build:
     commands:
       - echo 'prebuild'
   build:
     commands:
       - echo 'Running phpunit Tests'
       - composer dump-autoload
```

```
      - |
        codebuild-tests-run \
         --test-command "./vendor/bin/phpunit --debug" \
         --files-search "codebuild-glob-search '**/tests/*Test.php'"
  post_build:
      commands:
        - echo 'Test execution completed'
```

使用 Pytest 配置并行测试

以下 buildspec.yml 示例显示在 Ubuntu 平台上使用 Pytest 并行执行测试：

```
version: 0.2

batch:
  fast-fail: false
  build-fanout:
    parallelism: 5
    ignore-failure: false

phases:
  install:
    commands:
      - echo 'Installing Python dependencies'
      - apt-get update
      - apt-get install -y python3 python3-pip
      - pip3 install --upgrade pip
      - pip3 install pytest
  build:
    commands:
      - echo 'Running Python Tests'
      - |
         codebuild-tests-run \
          --test-command 'python -m pytest' \
          --files-search "codebuild-glob-search 'tests/test_*.py'" \
          --sharding-strategy 'equal-distribution'
  post_build:
    commands:
      - echo "Test execution completed"
```

以下 buildspec.yml 示例显示在 Windows 平台上使用 Pytest 并行执行测试：

```
version: 0.2
```

```
batch:
  fast-fail: false
  build-fanout:
    parallelism: 5
    ignore-failure: false

phases:
  install:
    commands:
      - echo 'Installing Python dependencies'
      - pip install pytest
  pre_build:
    commands:
      - echo 'prebuild'
  build:
    commands:
      - echo 'Running pytest'
      - |
        & codebuild-tests-run `
         --test-command 'pytest @("$env:CODEBUILD_CURRENT_SHARD_FILES" -split \"`r?`n
\")'  `
         --files-search "codebuild-glob-search '**/test_*.py' '**/*_test.py'" `
         --sharding-strategy 'equal-distribution'
  post_build:
    commands:
      - echo "Test execution completed"
```

在上面的示例中，CODEBUILD_CURRENT_SHARD_FILES环境变量用于获取分配给当前分片并作为数组传递给 pytest 命令的测试文件。

使用 Ruby 配置并行测试（Cucumber）

以下是显示在 Linux 平台上使用 Cucumber 执行并行测试的示例：buildspec.yml

```
version: 0.2

batch:
  fast-fail: false
  build-fanout:
    parallelism: 5
    ignore-failure: false
```

```
phases:
  install:
    commands:
      - echo 'Installing Ruby dependencies'
      - gem install bundler
      - bundle install
  pre_build:
    commands:
      - echo 'prebuild'
  build:
    commands:
      - echo 'Running Cucumber Tests'
      - cucumber --init
      - |
        codebuild-tests-run \
         --test-command "cucumber" \
         --files-search "codebuild-glob-search '**/*.feature'"
  post_build:
    commands:
      - echo "Test execution completed"
```

使用 Ruby 配置并行测试（RSpec）

以下 `buildspec.yml` 示例显示在 Ubuntu 平台上使用 RSpec 并行执行测试：

```
version: 0.2

batch:
  fast-fail: false
  build-fanout:
    parallelism: 5
    ignore-failure: false

phases:
  install:
    commands:
      - echo 'Installing Ruby dependencies'
      - apt-get update
      - apt-get install -y ruby ruby-dev build-essential
      - gem install bundler
      - bundle install
  build:
    commands:
```

```
      - echo 'Running Ruby Tests'
      - |
         codebuild-tests-run \
          --test-command 'bundle exec rspec' \
          --files-search "codebuild-glob-search 'spec/**/*_spec.rb'" \
          --sharding-strategy 'equal-distribution'
  post_build:
    commands:
      - echo "Test execution completed"
```

缓存构建以提高性能

构建项目时，可以使用缓存来节省时间。缓存可以存储构建环境的可重用部分，并在多个构建中使用它们。您的构建项目可以使用两种缓存类型中的一种：Amazon S3 或本地。如果使用本地缓存，则必须选择三种缓存模式中的一种或多种：源缓存、Docker 层缓存和自定义缓存。

> **Note**
>
> Docker 层缓存模式仅适用于 Linux 环境。如果选择此模式，则必须在特权模式下运行构建。CodeBuild 授予特权模式的项目授予其容器访问所有设备的权限。有关更多信息，请参阅 Docker 文档网站上的运行时权限和 Linux 功能。

主题

- Amazon S3 缓存
- 本地缓存
- 指定本地缓存

Amazon S3 缓存

Amazon S3 缓存将缓存存储在跨多个构建主机可用的 Amazon S3 存储桶中。对于构建成本高于下载成本的中小型构建构件，这是一个很好的选择。

要在构建中使用 Amazon S3，您可以为要缓存的文件指定路径`buildspec.yml`。 CodeBuild 将自动存储缓存并将其更新到项目上配置的 Amazon S3 位置。如果您不指定文件路径， CodeBuild 将尽最大努力缓存公共语言依赖关系，以帮助您加快构建速度。您可以在构建日志中查看缓存详细信息。

此外，如果您想拥有多个版本的缓存，可以在中定义缓存密钥buildspec.yml。 CodeBuild 将缓存存储在此缓存密钥的上下文下，并创建一个唯一的缓存副本，该副本一旦创建就不会更新。也可以跨项目共享缓存键。动态键、缓存版本控制和跨构建共享缓存等功能只有在指定键时才可用。

要了解有关 buildspec 文件中的缓存语法的更多信息，请参阅 buildspec 参考中的cache。

主题

- 生成动态键
- codebuild-hash-files
- 缓存版本
- 项目之间的缓存共享
- Buildspec 示例

生成动态键

缓存密钥可以包含 shell 命令和环境变量以使其独一无二，从而可以在密钥更改时自动更新缓存。例如，您可以使用 package-lock.json 文件的哈希值定义一个键。当该文件中的依赖项发生变化时，此哈希值（即，缓存键）也会发生变化，从而导致自动创建新缓存。

```
cache:
    key: npm-key-$(codebuild-hash-files package-lock.json)
```

CodeBuild 将评估表达式$(codebuild-hash-files package-lock.json)以获得最终密钥：

```
npm-key-abc123
```

您也可以使用环境变量定义缓存键，例如 CODEBUILD_RESOLVED_SOURCE_VERSION。这样可以确保每当源发生变化时，系统都会生成一个新的键，导致自动保存一个新缓存：

```
cache:
   key: npm-key-$CODEBUILD_RESOLVED_SOURCE_VERSION
```

CodeBuild 将评估表达式并获得最终密钥：

```
npm-key-046e8b67481d53bdc86c3f6affdd5d1afae6d369
```

codebuild-hash-files

`codebuild-hash-files`是一个 CLI 工具，它使用 glob 模式计算 CodeBuild 源目录中一组文件的 SHA-256 哈希值：

```
codebuild-hash-files <glob-pattern-1> <glob-pattern-2> ...
```

下面是一些使用 `codebuild-hash-files` 的示例：

```
codebuild-hash-files package-lock.json
codebuild-hash-files '**/*.md'
```

缓存版本

缓存版本是根据所缓存目录的路径生成的一个哈希值。如果两个缓存的版本不同，则在匹配过程中它们将被视为不同的缓存。例如，以下两个缓存被认为是不同的，因为它们引用不同的路径：

```
version: 0.2

phases:
  build:
    commands:
      - pip install pandas==2.2.3 --target pip-dependencies
cache:
  key: pip-dependencies
  paths:
    - "pip-dependencies/**/*"
```

```
version: 0.2

phases:
  build:
    commands:
      - pip install pandas==2.2.3 --target tmp/pip-dependencies
cache:
  key: pip-dependencies
  paths:
    - "tmp/pip-dependencies/**/*"
```

项目之间的缓存共享

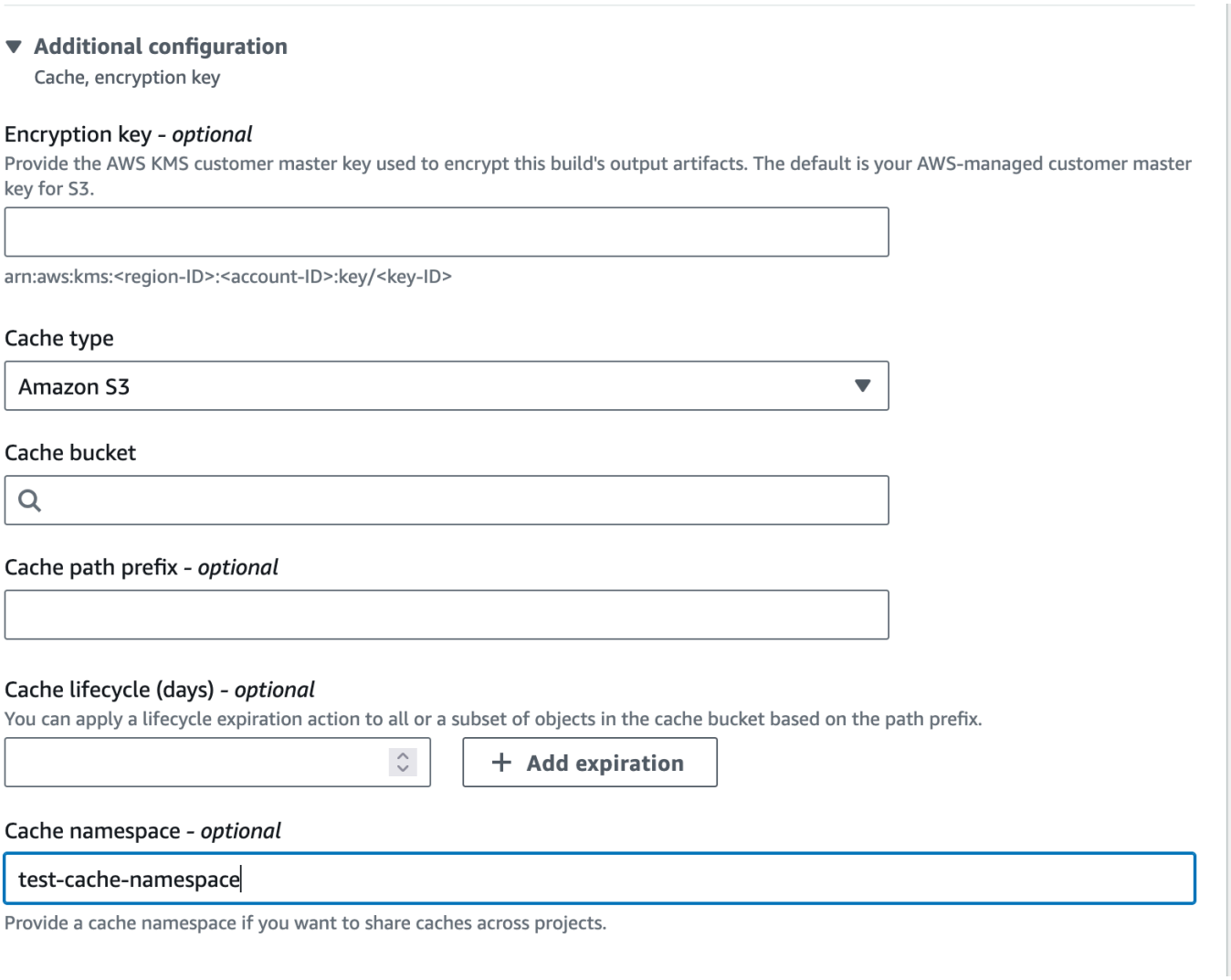
您可以使用 cache 部分下的 cacheNamespace API 字段在多个项目之间共享缓存。此字段用于定义缓存的范围。要共享缓存，必须执行以下操作：

- 使用相同的 cacheNamespace。
- 指定相同的缓存 key。
- 定义相同的缓存路径。
- 使用相同的 Amazon S3 存储桶和 pathPrefix（如果已设置）。

这样可以确保一致性并可在项目间共享缓存。

指定缓存命名空间（控制台）

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 选择创建项目。有关更多信息，请参阅 创建构建项目（控制台） 和 运行构建（控制台）。
3. 在构件中，选择其他配置。
4. 对于缓存类型，选择 Amazon S3。
5. 对于缓存命名空间 - 可选，输入命名空间值。



6. 继续使用默认值，然后选择创建构建项目。

指定缓存命名空间 (AWS CLI)

您可以使用中的`--cache`参数 AWS CLI 来指定缓存命名空间。

```
--cache '{"type": "S3", "location": "your-s3-bucket", "cacheNamespace": "test-cache-namespace"}'
```

Buildspec 示例

以下是常用语言的若干 buildspec 示例：

主题

- 缓存 Node.js 依赖关系

- 缓存 Python 依赖项
- 缓存 Ruby 依赖项
- 缓存 Go 依赖项

缓存 Node.js 依赖关系

如果您的项目包含package-lock.json文件并npm用于管理 Node.js 依赖关系，则以下示例显示如何设置缓存。默认情况下，npm 将依赖项安装到 node_modules 目录中。

```
version: 0.2

phases:
  build:
    commands:
      - npm install
cache:
  key: npm-$(codebuild-hash-files package-lock.json)
  paths:
    - "node_modules/**/*"
```

缓存 Python 依赖项

如果您的项目包含一个 requirements.txt 文件并使用 pip 来管理 Python 依赖项，则以下示例展示了如何配置缓存。默认情况下，pip 将软件包安装到系统的 site-packages 目录中。

```
version: 0.2

phases:
  build:
    commands:
      - pip install -r requirements.txt
cache:
  key: python-$(codebuild-hash-files requirements.txt)
  paths:
    - "/root/.pyenv/versions/${python_version}/lib/python${python_major_version}/site-
packages/**/*"
```

此外，您还可以将依赖项安装到一个特定目录中，并为此目录配置缓存。

```
version: 0.2
```

```
phases:
  build:
    commands:
      - pip install -r requirements.txt --target python-dependencies
cache:
  key: python-$(codebuild-hash-files requirements.txt)
  paths:
    - "python-dependencies/**/*"
```

缓存 Ruby 依赖项

如果您的项目包含 Gemfile.lock 文件并使用 Bundler 来管理 gem 依赖项，则以下示例演示了如何有效地配置缓存。

```
version: 0.2

phases:
  build:
    commands:
      - bundle install --path vendor/bundle
cache:
  key: ruby-$(codebuild-hash-files Gemfile.lock)
  paths:
    - "vendor/bundle/**/*"
```

缓存 Go 依赖项

如果您的项目包含 go.sum 文件并使用 Go 模块来管理依赖项，则以下示例演示如何配置缓存。默认情况下，将 Go 模块下载并存储在 ${GOPATH}/pkg/mod 目录中。

```
version: 0.2

phases:
  build:
    commands:
      - go mod download
cache:
  key: go-$(codebuild-hash-files go.sum)
  paths:
    - "/go/pkg/mod/**/*"
```

本地缓存

本地缓存将缓存本地存储在构建主机上，并且仅可用于该构建主机。对于大中型构建构件，这是一个很好的选择，因为构建主机上的缓存立即可用。如果您不经常构建，这不是最好的选择。这意味着构建性能不受网络传输时间的影响。

如果您选择本地缓存，则必须选择以下一个或多个缓存模式：

- 源缓存模式用于缓存主要和辅助源的 Git 元数据。创建缓存后，后续构建仅拉取两次提交之间发生的更改。对于具有干净工作目录和源为大型 Git 存储库的项目，此模式是一个不错的选择。如果您选择此选项且您的项目不使用 Git 存储库（AWS CodeCommit GitHub、 GitHub 企业服务器或 Bitbucket），则该选项将被忽略。
- Docker 层缓存模式缓存现有 Docker 层。对于构建或拉取大型 Docker 映像的项目，此模式是一个不错的选择。它可以防止因从网络中拉取大型 Docker 映像而导致的性能问题。

  > **Note**
  >
  > - 您只能在 Linux 环境中使用 Docker 层缓存。
  > - 必须设置 `privileged` 标志以使您的项目具有所需的 Docker 权限。
  >
  >   默认情况下，为非 VPC 构建启用 Docker 进程守护程序。如果您想使用 Docker 容器进行 VPC 构建，请参阅 Docker 文档网站上的运行时权限和 Linux 功能并启用特权模式。此外，Windows 不支持特权模式。
  > - 在使用 Docker 层缓存之前，您应考虑安全影响。

- 自定义缓存模式用于缓存您在 buildspec 文件中指定的目录。如果您的构建方案不适合另外两种本地缓存模式之一，则此模式是一个不错的选择。如果您使用自定义缓存：
  - 只能指定目录进行缓存。不能指定单独的文件。
  - 用于引用缓存目录的符号链接。
  - 缓存目录在下载项目源代码之前链接到您的构建。如果缓存的项目具有相同的名称，则会覆盖源项目。使用 buildspec 文件中的缓存路径指定目录。有关更多信息，请参阅 buildspec 语法。
  - 避免在源代码和缓存中使用相同的目录名。 Locally-cached 目录可能会覆盖或删除源存储库中同名目录的内容。

> **Note**
>
> LINUX_GPU_CONTAINER 环境类型和 BUILD_GENERAL1_2XLARGE 计算类型不支持本地缓存。有关更多信息，请参阅 构建环境计算模式和类型。

> **Note**
>
> 当您配置为使用 VPC 时 CodeBuild ，不支持本地缓存。有关将 VPC 与一起使用的更多信息 CodeBuild，请参阅AWS CodeBuild 与亚马逊 Virtual Private Cloud 配合使用。

指定本地缓存

您可以使用 AWS CLI、控制台、SDK 或 CloudFormation 指定本地缓存。有关本地缓存的更多信息，请参阅本地缓存。

主题

- 指定本地缓存 (CLI)
- 指定本地缓存（控制台）
- 指定本地缓存 (CloudFormation)

指定本地缓存 (CLI)

您可以使用中的--cache参数 AWS CLI 来指定三种本地缓存类型中的每一种类型。

- 指定源缓存：

```
--cache type=LOCAL,mode=[LOCAL_SOURCE_CACHE]
```

- 指定 Docker 层缓存：

```
--cache type=LOCAL,mode=[LOCAL_DOCKER_LAYER_CACHE]
```

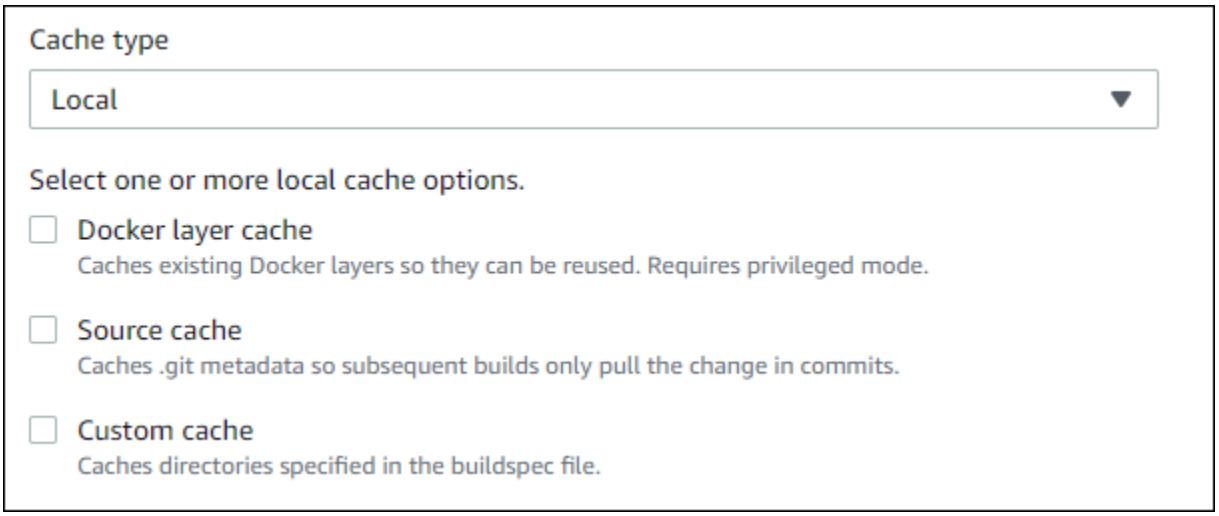
- 指定自定义缓存：

```
--cache type=LOCAL,mode=[LOCAL_CUSTOM_CACHE]
```

有关更多信息，请参阅 创建构建项目 (AWS CLI)。

指定本地缓存（控制台）

您可以使用控制台的构件部分指定缓存。对于缓存类型，选择 Amazon S3 或本地。如果您选择本地，请选择三个本地缓存选项中的一个或多个。



有关更多信息，请参阅 创建构建项目（控制台）。

指定本地缓存 (CloudFormation)

如果您使用 CloudFormation 来指定本地缓存，请在Cache属性上指定LOCAL。Type以下示例 YAML-formatted CloudFormation 代码指定了所有三种本地缓存类型。您可以指定这些类型的任意组合。如果您使用 Docker 层缓存，在 Environment 下，您必须将 PrivilegedMode 设置为 true，将 Type 设置为 LINUX_CONTAINER。

```
CodeBuildProject:
    Type: AWS::CodeBuild::Project
    Properties:
      Name: MyProject
      ServiceRole: <service-role>
      Artifacts:
        Type: S3
        Location: <bucket-name>
        Name: myArtifact
        EncryptionDisabled: true
        OverrideArtifactName: true
      Environment:
        Type: LINUX_CONTAINER
```

```
      ComputeType: BUILD_GENERAL1_SMALL
      Image: aws/codebuild/standard:5.0
      Certificate: <bucket/cert.zip>
      # PrivilegedMode must be true if you specify LOCAL_DOCKER_LAYER_CACHE
      PrivilegedMode: true
    Source:
      Type: GITHUB
      Location: <github-location>
      InsecureSsl: true
      GitCloneDepth: 1
      ReportBuildStatus: false
    TimeoutInMinutes: 10
    Cache:
      Type: LOCAL
      Modes: # You can specify one or more cache mode,
        - LOCAL_CUSTOM_CACHE
        - LOCAL_DOCKER_LAYER_CACHE
        - LOCAL_SOURCE_CACHE
```

> **Note**
>
> 默认情况下，为非 VPC 构建启用 Docker 进程守护程序。如果您想使用 Docker 容器进行 VPC 构建，请参阅 Docker 文档网站上的运行时权限和 Linux 功能并启用特权模式。此外，Windows 不支持特权模式。

有关更多信息，请参阅 创建构建项目 (CloudFormation)。

调试内置 AWS CodeBuild

AWS CodeBuild 提供了两种在开发和故障排除期间调试版本的方法。您可以使用 CodeBuild 沙盒环境实时调查问题并验证修复程序，也可以使用 S AWS ystems Manager 会话管理器连接到生成容器并查看容器状态。

使用 CodeBuild 沙盒调试构建

CodeBuild 沙盒环境在安全和隔离的环境中提供交互式调试会话。您可以通过 AWS 管理控制台 或直接与环境进行交互 AWS CLI，执行命令并逐步验证您的构建过程。它使用经济实惠的每秒计费模式，并支持与您的构建环境相同的源提供商和 AWS 服务的原生集成。您也可以使用 SSH 客户端或从集成式开发环境（IDE）连接到沙盒环境。

要了解有关 CodeBuild 沙盒定价的更多信息，请访问定CodeBuild 价文档。有关详细说明，请访问使用 CodeBuild 沙箱调试构建文档。

使用会话管理器调试构建

AWS Systems Manager 会话管理器允许在实际执行环境中直接访问正在运行的构建。这种方法可让您连接到活动的构建容器并实时检查构建过程。您可以检查文件系统，监控正在运行的进程，并在出现问题时对其进行故障排除。

有关详细说明，请访问使用会话管理器调试构建文档。

使用 CodeBuild 沙箱调试构建

在中 AWS CodeBuild，您可以使用 CodeBuild 沙箱来运行自定义命令来调试构建，并对构建进行故障排除。

主题

- 先决条件
- 使用 CodeBuild 沙箱（控制台）调试构建
- 使用 CodeBuild 沙箱调试构建 (AWS CLI)
- 教程：使用 SSH 连接到沙盒
- 问题排查 AWS CodeBuild 沙箱 SSH 连接问题

先决条件

在使用 CodeBuild 沙箱之前，请确保您的 CodeBuild 服务角色具有以下 SSM 策略：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssmmessages:CreateControlChannel",
                "ssmmessages:CreateDataChannel",
                "ssmmessages:OpenControlChannel",
                "ssmmessages:OpenDataChannel"
```

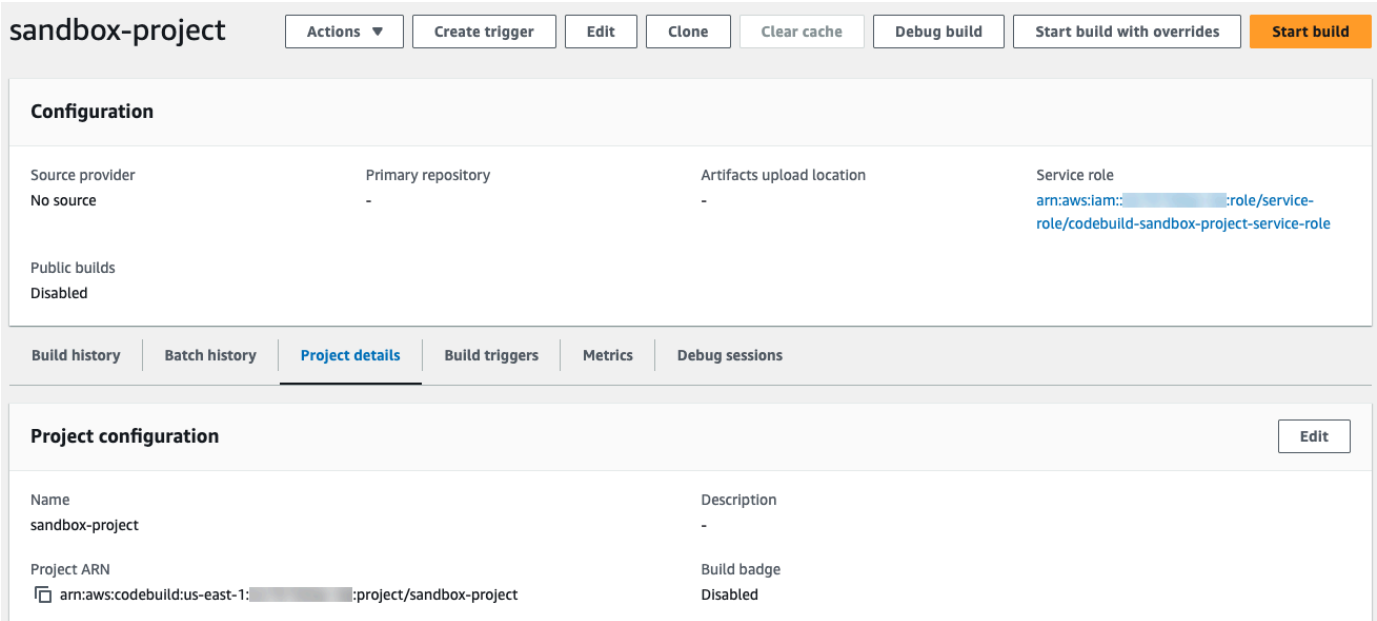
```
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ssm:StartSession"
            ],
            "Resource": [
                "arn:aws:codebuild:us-east-1:111122223333:build/*",
                "arn:aws:ssm:us-east-1::document/AWS-StartSSHSession"
            ]
        }
    ]
}
```

使用 CodeBuild 沙箱（控制台）调试构建

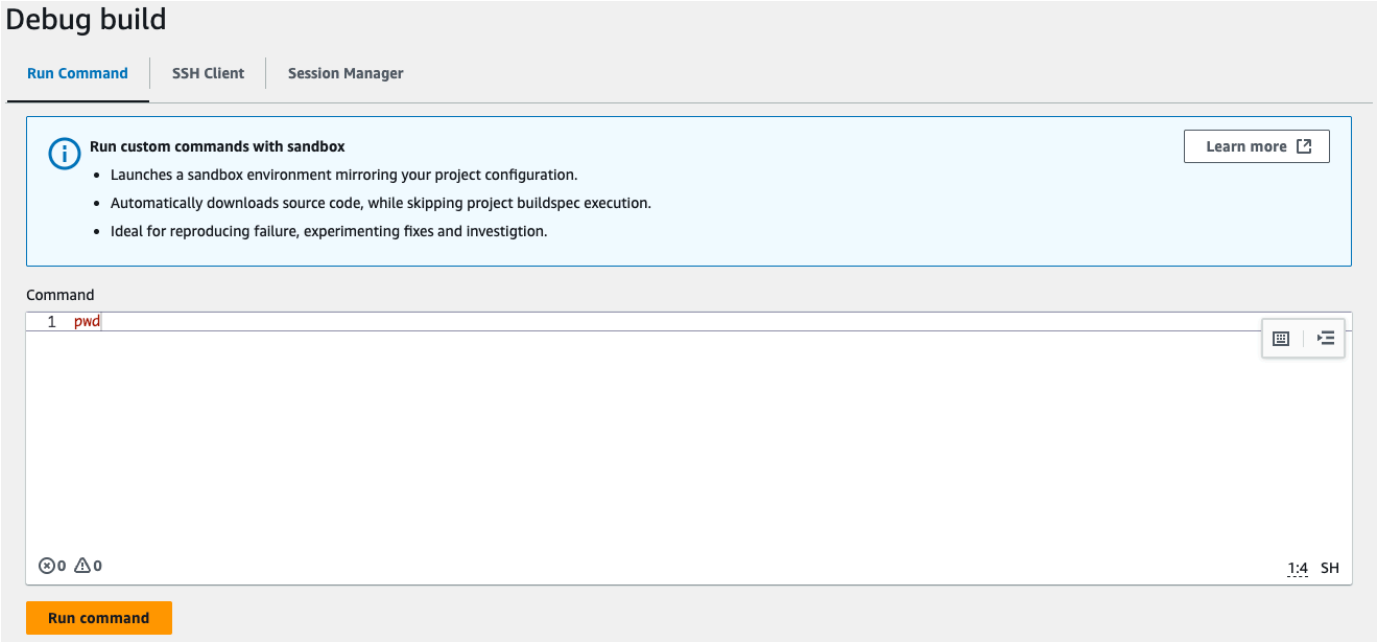
使用以下说明在控制台中运行命令并将 SSH 客户端与 CodeBuild 沙箱连接。

使用 CodeBuild 沙箱（控制台）运行命令

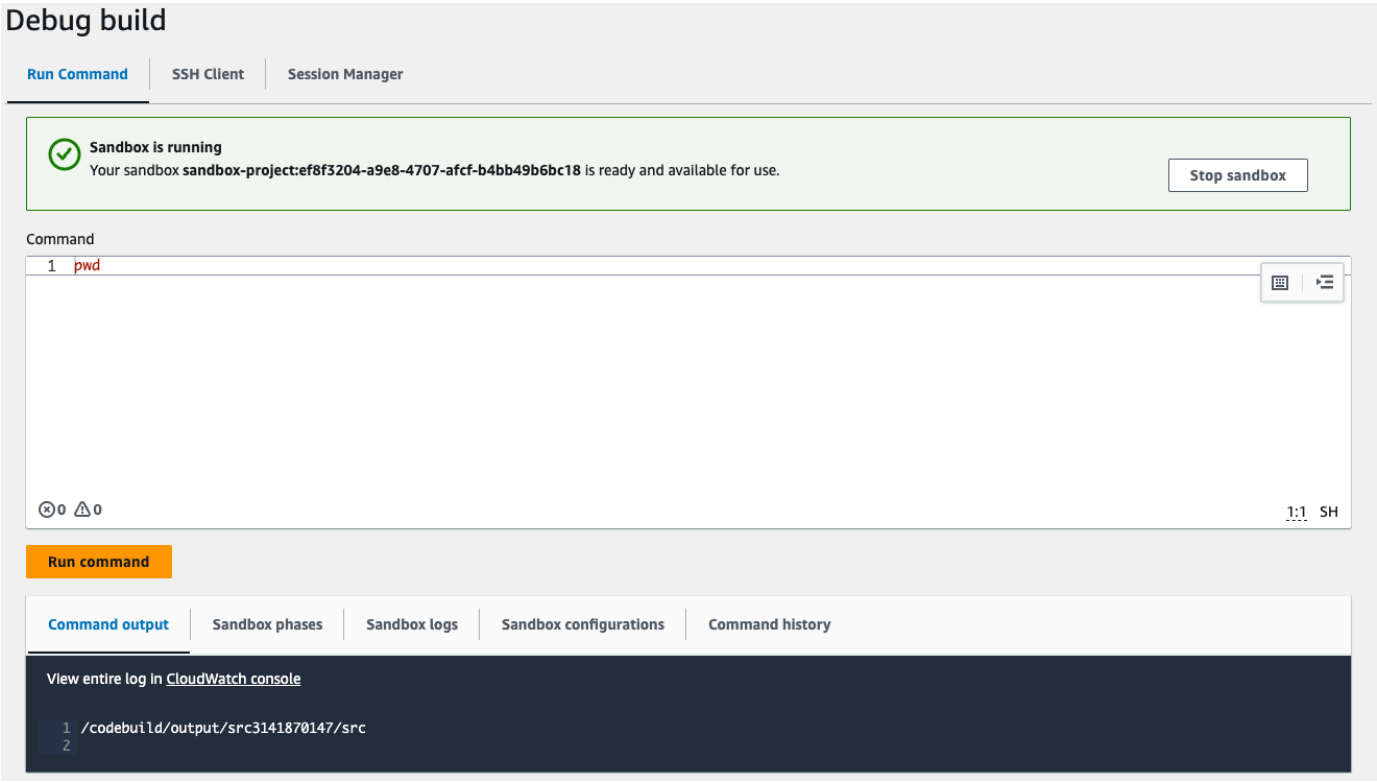
1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择构建项目。选择构建项目，然后选择调试构建。



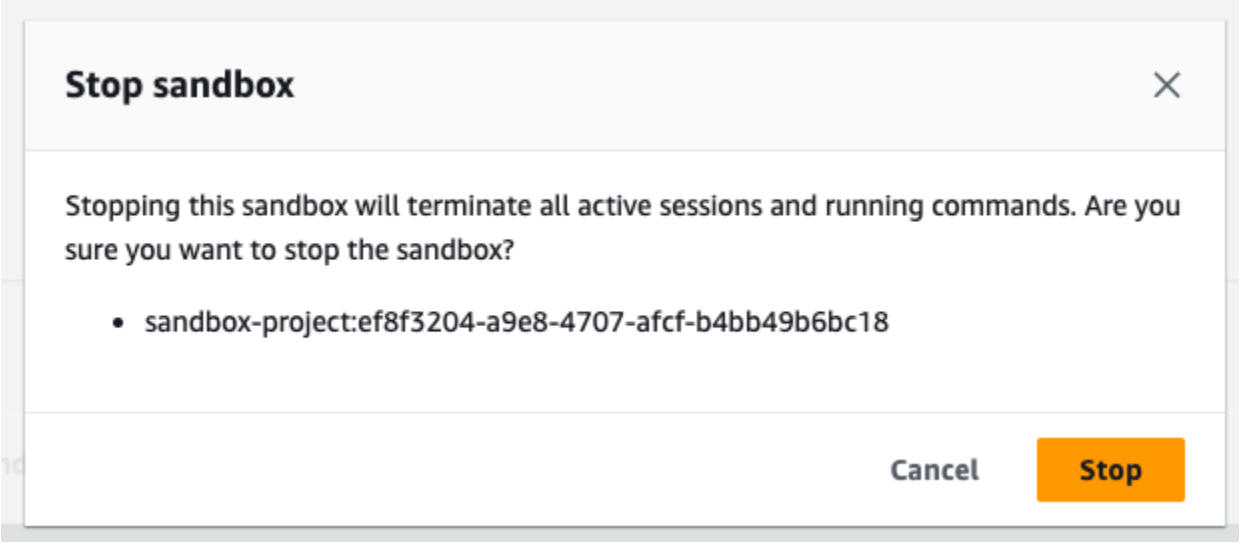
3. 在运行命令选项卡中，输入您的自定义命令，然后选择运行命令。

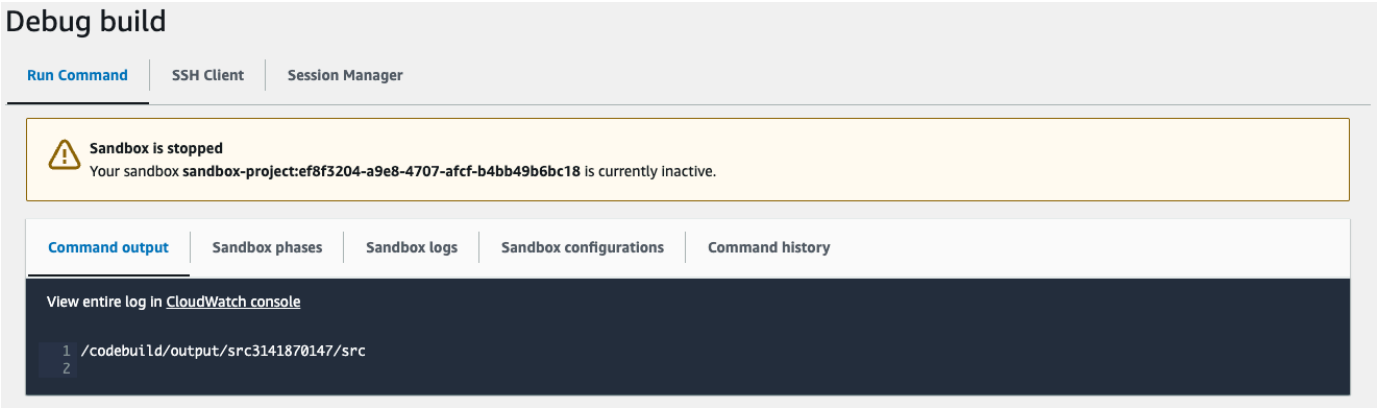


4. 然后，您的 CodeBuild 沙箱将被初始化并开始运行您的自定义命令。完成后，输出将显示在输出选项卡中。



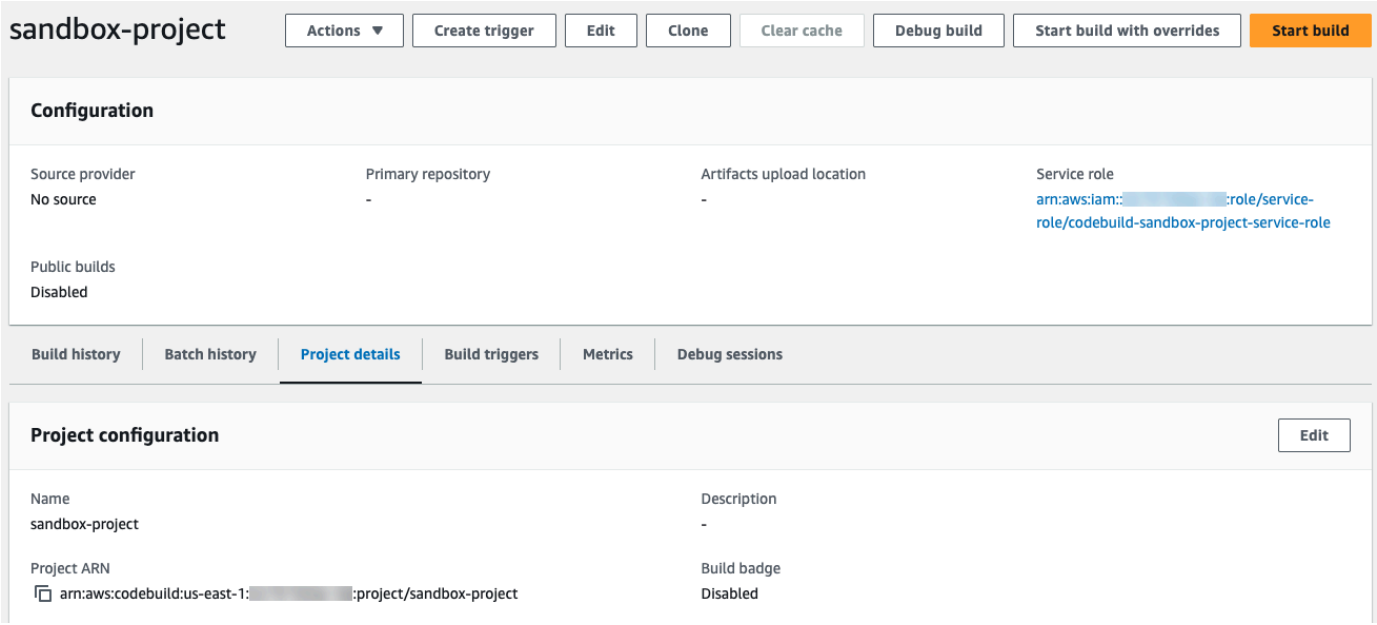
5. 完成故障排除后，可以通过选择停止沙盒来停止沙盒。然后选择停止以确认将停止沙盒。



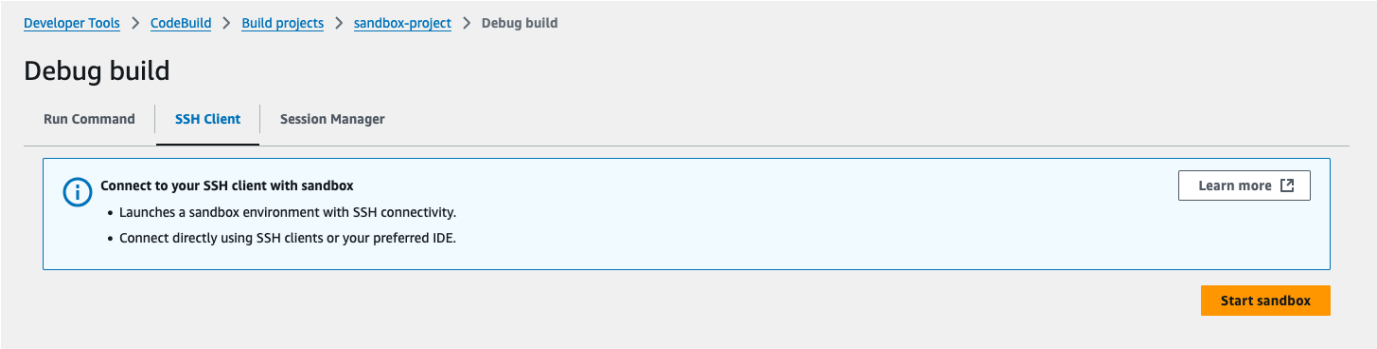


使用 CodeBuild 沙箱（控制台）连接您的 SSH 客户端

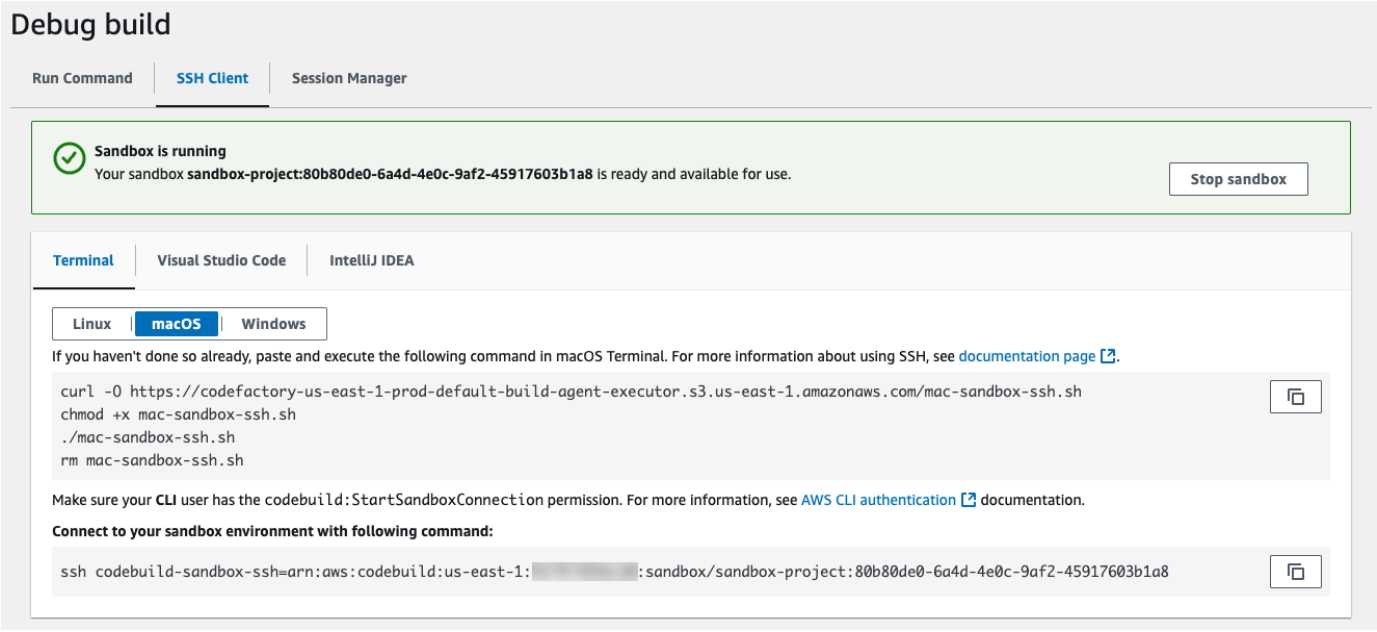
1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择构建项目。选择构建项目，然后选择调试构建。



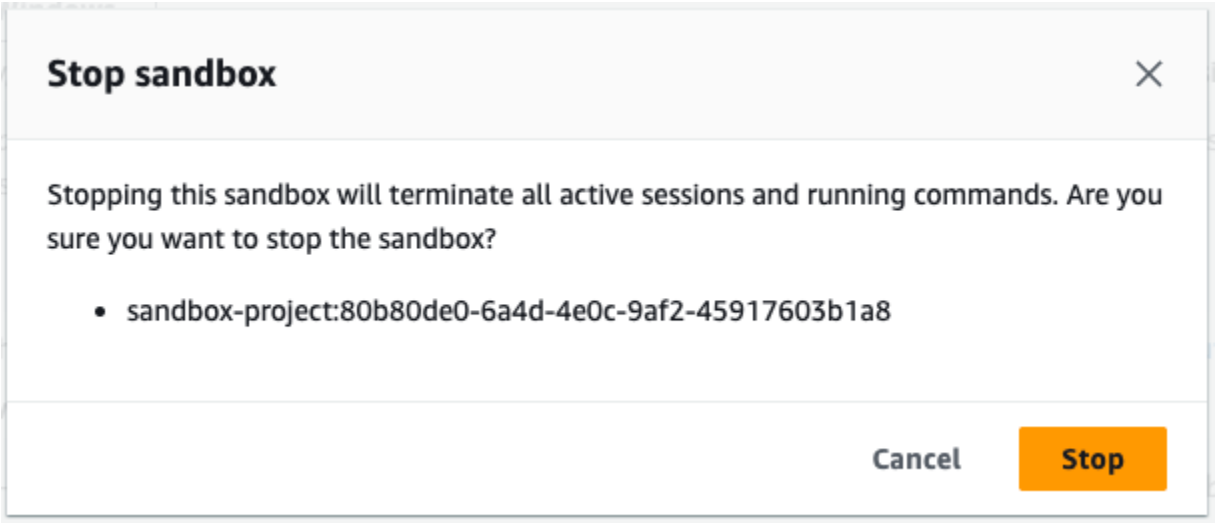
3. 在 SSH 客户端选项卡中，选择启动沙盒。

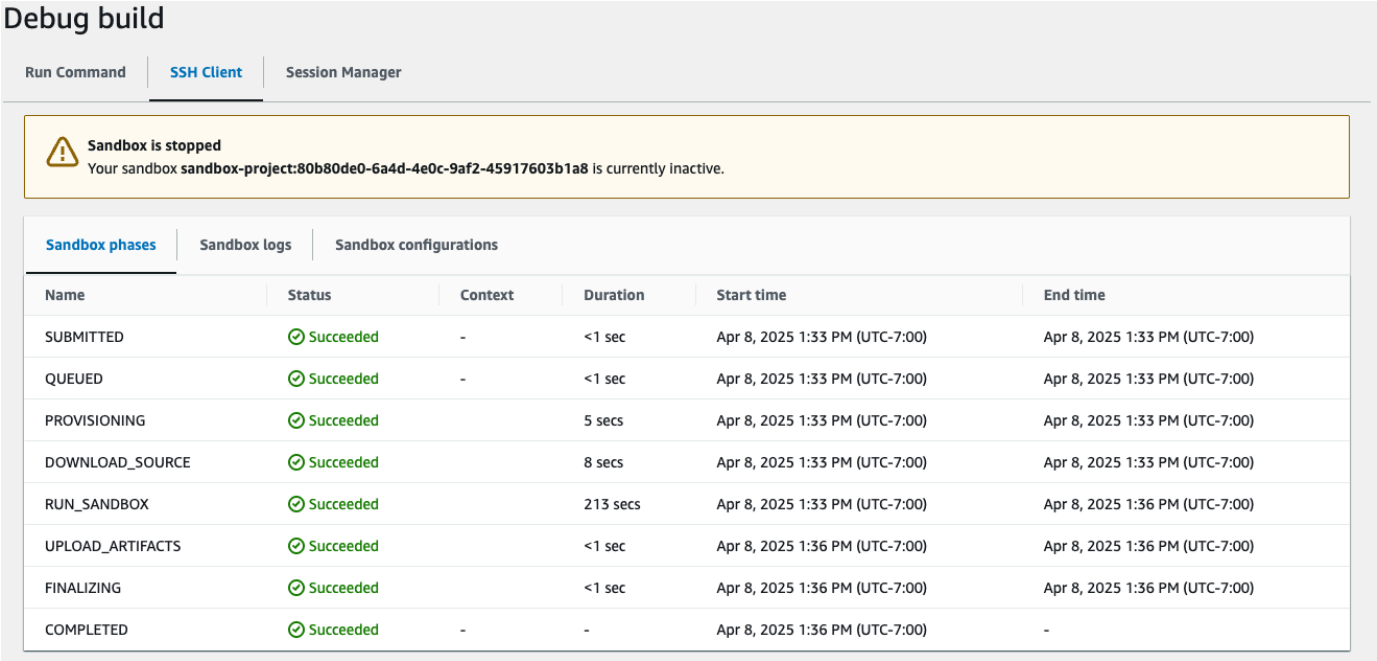


4. CodeBuild 沙箱开始运行后，按照控制台说明将 SSH 客户端与沙箱连接。



5. 完成故障排除后，可以通过选择停止沙盒来停止沙盒。然后选择停止以确认将停止沙盒。





使用 CodeBuild 沙箱调试构建 (AWS CLI)

使用以下说明运行命令并将 SSH 客户端与 CodeBuild 沙箱连接。

启动 CodeBuild 沙箱 (AWS CLI)

CLI command

```
aws codebuild start-sandbox --project-name $PROJECT_NAME
```

- `--project-name`: CodeBuild 项目名称

Sample request

```
aws codebuild start-sandbox --project-name "project-name"
```

Sample response

```
{
    "id": "project-name",
    "arn": "arn:aws:codebuild:us-west-2:962803963624:sandbox/project-name",
    "projectName": "project-name",
    "requestTime": "2025-02-06T11:24:15.560000-08:00",
```

```
    "status": "QUEUED",
    "source": {
        "type": "S3",
        "location": "arn:aws:s3:::cofa-e2e-test-1-us-west-2-beta-default-build-
sources/eb-sample-jetty-v4.zip",
        "insecureSsl": false
    },
    "environment": {
        "type": "LINUX_CONTAINER",
        "image": "aws/codebuild/standard:6.0",
        "computeType": "BUILD_GENERAL1_SMALL",
        "environmentVariables": [{
                "name": "foo",
                "value": "bar",
                "type": "PLAINTEXT"
            },
            {
                "name": "bar",
                "value": "baz",
                "type": "PLAINTEXT"
            }
        ],
        "privilegedMode": false,
        "imagePullCredentialsType": "CODEBUILD"
    },
    "timeoutInMinutes": 10,
    "queuedTimeoutInMinutes": 480,
    "logConfig": {
        "cloudWatchLogs": {
            "status": "ENABLED",
            "groupName": "group",
            "streamName": "stream"
        },
        "s3Logs": {
            "status": "ENABLED",
            "location": "codefactory-test-pool-1-us-west-2-beta-default-build-logs",
            "encryptionDisabled": false
        }
    },
    "encryptionKey": "arn:aws:kms:us-west-2:962803963624:alias/SampleEncryptionKey",
    "serviceRole": "arn:aws:iam::962803963624:role/BuildExecutionServiceRole",
    "currentSession": {
        "id": "0103e0e7-52aa-4a3d-81dd-bfc27226fa54",
        "currentPhase": "QUEUED",
```

```
        "status": "QUEUED",
        "startTime": "2025-02-06T11:24:15.626000-08:00",
        "logs": {
            "groupName": "group",
            "streamName": "stream/0103e0e7-52aa-4a3d-81dd-bfc27226fa54",
            "deepLink": "https://console.aws.amazon.com/cloudwatch/home?
region=us-west-2#logsV2:log-groups/log-group/group/log-events/stream
$252F0103e0e7-52aa-4a3d-81dd-bfc27226fa54",
            "s3DeepLink": "https://s3.console.aws.amazon.com/s3/object/
codefactory-test-pool-1-us-west-2-beta-default-build-logs/0103e0e7-52aa-4a3d-81dd-
bfc27226fa54.gz?region=us-west-2",
            "cloudWatchLogsArn": "arn:aws:logs:us-west-2:962803963624:log-
group:group:log-stream:stream/0103e0e7-52aa-4a3d-81dd-bfc27226fa54",
            "s3LogsArn": "arn:aws:s3:::codefactory-test-pool-1-us-west-2-beta-
default-build-logs/0103e0e7-52aa-4a3d-81dd-bfc27226fa54.gz",
            "cloudWatchLogs": {
                "status": "ENABLED",
                "groupName": "group",
                "streamName": "stream"
            },
            "s3Logs": {
                "status": "ENABLED",
                "location": "codefactory-test-pool-1-us-west-2-beta-default-build-
logs",
                "encryptionDisabled": false
            }
        }
    }
}
```

获取有关沙箱状态的信息 (AWS CLI)

CLI command

```
aws codebuild batch-get-sandboxes --ids $SANDBOX_IDs
```

Sample request

```
aws codebuild stop-sandbox --id "arn:aws:codebuild:us-west-2:962803963624:sandbox/
project-name"
```

- `--ids`：sandboxIds 或 sandboxArns 的逗号分隔列表。

您可以提供沙盒 ID 或沙盒 ARN：

- 沙盒 ID：*<codebuild-project-name>*:*<UUID>*

  例如 `project-name:d25be134-05cb-404a-85da-ac5f85d2d72c`。

- 沙盒 ARN: arn: aws: codebuild::: sandbox/: *<region>* *<account-id>* *<codebuild-project-name>* *<UUID>*

  例如 `arn:aws:codebuild:us-west-2:962803963624:sandbox/project-name:d25be134-05cb-404a-85da-ac5f85d2d72c`。

Sample response

```
{
    "sandboxes": [{
        "id": "project-name",
        "arn": "arn:aws:codebuild:us-west-2:962803963624:sandbox/project-name",
        "projectName": "project-name",
        "requestTime": "2025-02-06T11:24:15.560000-08:00",
        "endTime": "2025-02-06T11:39:21.587000-08:00",
        "status": "STOPPED",
        "source": {
            "type": "S3",
            "location": "arn:aws:s3:::cofa-e2e-test-1-us-west-2-beta-default-build-
sources/eb-sample-jetty-v4.zip",
            "insecureSsl": false
        },
        "environment": {
            "type": "LINUX_CONTAINER",
            "image": "aws/codebuild/standard:6.0",
            "computeType": "BUILD_GENERAL1_SMALL",
            "environmentVariables": [{
                    "name": "foo",
                    "value": "bar",
                    "type": "PLAINTEXT"
                },
                {
                    "name": "bar",
                    "value": "baz",
                    "type": "PLAINTEXT"
                }
            ],
```

```
            "privilegedMode": false,
            "imagePullCredentialsType": "CODEBUILD"
        },
        "timeoutInMinutes": 10,
        "queuedTimeoutInMinutes": 480,
        "logConfig": {
            "cloudWatchLogs": {
                "status": "ENABLED",
                "groupName": "group",
                "streamName": "stream"
            },
            "s3Logs": {
                "status": "ENABLED",
                "location": "codefactory-test-pool-1-us-west-2-beta-default-build-
logs",
                "encryptionDisabled": false
            }
        },
        "encryptionKey": "arn:aws:kms:us-west-2:962803963624:alias/
SampleEncryptionKey",
        "serviceRole": "arn:aws:iam::962803963624:role/BuildExecutionServiceRole",
        "currentSession": {
            "id": "0103e0e7-52aa-4a3d-81dd-bfc27226fa54",
            "currentPhase": "COMPLETED",
            "status": "STOPPED",
            "startTime": "2025-02-06T11:24:15.626000-08:00",
            "endTime": "2025-02-06T11:39:21.600000-08:00",
            "phases": [{
                    "phaseType": "SUBMITTED",
                    "phaseStatus": "SUCCEEDED",
                    "startTime": "2025-02-06T11:24:15.577000-08:00",
                    "endTime": "2025-02-06T11:24:15.606000-08:00",
                    "durationInSeconds": 0
                },
                {
                    "phaseType": "QUEUED",
                    "phaseStatus": "SUCCEEDED",
                    "startTime": "2025-02-06T11:24:15.606000-08:00",
                    "endTime": "2025-02-06T11:24:16.067000-08:00",
                    "durationInSeconds": 0
                },
                {
                    "phaseType": "PROVISIONING",
                    "phaseStatus": "SUCCEEDED",
```

```
                    "startTime": "2025-02-06T11:24:16.067000-08:00",
                    "endTime": "2025-02-06T11:24:20.519000-08:00",
                    "durationInSeconds": 4,
                    "contexts": [{
                        "statusCode": "",
                        "message": ""
                    }]
                },
                {
                    "phaseType": "DOWNLOAD_SOURCE",
                    "phaseStatus": "SUCCEEDED",
                    "startTime": "2025-02-06T11:24:20.519000-08:00",
                    "endTime": "2025-02-06T11:24:22.238000-08:00",
                    "durationInSeconds": 1,
                    "contexts": [{
                        "statusCode": "",
                        "message": ""
                    }]
                },
                {
                    "phaseType": "RUNNING_SANDBOX",
                    "phaseStatus": "TIMED_OUT",
                    "startTime": "2025-02-06T11:24:22.238000-08:00",
                    "endTime": "2025-02-06T11:39:21.560000-08:00",
                    "durationInSeconds": 899,
                    "contexts": [{
                        "statusCode": "BUILD_TIMED_OUT",
                        "message": "Build has timed out. "
                    }]
                },
                {
                    "phaseType": "COMPLETED",
                    "startTime": "2025-02-06T11:39:21.560000-08:00"
                }
            ],
            "logs": {
                "groupName": "group",
                "streamName": "stream/0103e0e7-52aa-4a3d-81dd-bfc27226fa54",
                "deepLink": "https://console.aws.amazon.com/cloudwatch/
home?region=us-west-2#logsV2:log-groups/log-group/group/log-events/stream
$252F0103e0e7-52aa-4a3d-81dd-bfc27226fa54",
                "s3DeepLink": "https://s3.console.aws.amazon.com/s3/object/
codefactory-test-pool-1-us-west-2-beta-default-build-logs/0103e0e7-52aa-4a3d-81dd-
bfc27226fa54.gz?region=us-west-2",
```

```
                "cloudWatchLogsArn": "arn:aws:logs:us-west-2:962803963624:log-
group:group:log-stream:stream/0103e0e7-52aa-4a3d-81dd-bfc27226fa54",
                "s3LogsArn": "arn:aws:s3:::codefactory-test-pool-1-us-west-2-beta-
default-build-logs/0103e0e7-52aa-4a3d-81dd-bfc27226fa54.gz",
                "cloudWatchLogs": {
                    "status": "ENABLED",
                    "groupName": "group",
                    "streamName": "stream"
                },
                "s3Logs": {
                    "status": "ENABLED",
                    "location": "codefactory-test-pool-1-us-west-2-beta-default-
build-logs",
                    "encryptionDisabled": false
                }
            }
        }
    }],
    "sandboxesNotFound": []
}
```

停止沙箱 (AWS CLI)

CLI command

```
aws codebuild stop-sandbox --id $SANDBOX-ID
```

- `--id`：sandboxId 或 sandboxArn。

Sample request

```
aws codebuild stop-sandbox --id "arn:aws:codebuild:us-west-2:962803963624:sandbox/
project-name"
```

Sample response

```
{
    "id": "project-name",
    "arn": "arn:aws:codebuild:us-west-2:962803963624:sandbox/project-name",
    "projectName": "project-name",
    "requestTime": "2025-02-06T11:24:15.560000-08:00",
```

```
    "status": "STOPPING",
    "source": {
        "type": "S3",
        "location": "arn:aws:s3:::cofa-e2e-test-1-us-west-2-beta-default-build-
sources/eb-sample-jetty-v4.zip",
        "insecureSsl": false
    },
    "environment": {
        "type": "LINUX_CONTAINER",
        "image": "aws/codebuild/standard:6.0",
        "computeType": "BUILD_GENERAL1_SMALL",
        "environmentVariables": [{
                "name": "foo",
                "value": "bar",
                "type": "PLAINTEXT"
            },
            {
                "name": "bar",
                "value": "baz",
                "type": "PLAINTEXT"
            }
        ],
        "privilegedMode": false,
        "imagePullCredentialsType": "CODEBUILD"
    },
    "timeoutInMinutes": 10,
    "queuedTimeoutInMinutes": 480,
    "logConfig": {
        "cloudWatchLogs": {
            "status": "ENABLED",
            "groupName": "group",
            "streamName": "stream"
        },
        "s3Logs": {
            "status": "ENABLED",
            "location": "codefactory-test-pool-1-us-west-2-beta-default-build-logs",
            "encryptionDisabled": false
        }
    },
    "encryptionKey": "arn:aws:kms:us-west-2:962803963624:alias/SampleEncryptionKey",
    "serviceRole": "arn:aws:iam::962803963624:role/BuildExecutionServiceRole",
    "currentSession": {
        "id": "0103e0e7-52aa-4a3d-81dd-bfc27226fa54",
        "currentPhase": "RUN_SANDBOX",
```

```
        "status": "STOPPING",
        "startTime": "2025-02-06T11:24:15.626000-08:00",
        "phases": [{
                "phaseType": "SUBMITTED",
                "phaseStatus": "SUCCEEDED",
                "startTime": "2025-02-08T14:33:26.144000-08:00",
                "endTime": "2025-02-08T14:33:26.173000-08:00",
                "durationInSeconds": 0
            },
            {
                "phaseType": "QUEUED",
                "phaseStatus": "SUCCEEDED",
                "startTime": "2025-02-08T14:33:26.173000-08:00",
                "endTime": "2025-02-08T14:33:26.702000-08:00",
                "durationInSeconds": 0
            },
            {
                "phaseType": "PROVISIONING",
                "phaseStatus": "SUCCEEDED",
                "startTime": "2025-02-08T14:33:26.702000-08:00",
                "endTime": "2025-02-08T14:33:30.530000-08:00",
                "durationInSeconds": 3,
                "contexts": [{
                    "statusCode": "",
                    "message": ""
                }]
            },
            {
                "phaseType": "DOWNLOAD_SOURCE",
                "phaseStatus": "SUCCEEDED",
                "startTime": "2025-02-08T14:33:30.530000-08:00",
                "endTime": "2025-02-08T14:33:33.478000-08:00",
                "durationInSeconds": 2,
                "contexts": [{
                    "statusCode": "",
                    "message": ""
                }]
            },
            {
                "phaseType": "RUN_SANDBOX",
                "startTime": "2025-02-08T14:33:33.478000-08:00"
            }
        ],
        "logs": {
```

```
            "groupName": "group",
            "streamName": "stream/0103e0e7-52aa-4a3d-81dd-bfc27226fa54",
            "deepLink": "https://console.aws.amazon.com/cloudwatch/home?
region=us-west-2#logsV2:log-groups/log-group/group/log-events/stream
$252F0103e0e7-52aa-4a3d-81dd-bfc27226fa54",
            "s3DeepLink": "https://s3.console.aws.amazon.com/s3/object/
codefactory-test-pool-1-us-west-2-beta-default-build-logs/0103e0e7-52aa-4a3d-81dd-
bfc27226fa54.gz?region=us-west-2",
            "cloudWatchLogsArn": "arn:aws:logs:us-west-2:962803963624:log-
group:group:log-stream:stream/0103e0e7-52aa-4a3d-81dd-bfc27226fa54",
            "s3LogsArn": "arn:aws:s3:::codefactory-test-pool-1-us-west-2-beta-
default-build-logs/0103e0e7-52aa-4a3d-81dd-bfc27226fa54.gz",
            "cloudWatchLogs": {
                "status": "ENABLED",
                "groupName": "group",
                "streamName": "stream"
            },
            "s3Logs": {
                "status": "ENABLED",
                "location": "codefactory-test-pool-1-us-west-2-beta-default-build-
logs",
                "encryptionDisabled": false
            }
        }
    }
}
```

开始执行命令 (AWS CLI)

CLI command

```
aws codebuild start-command-execution --command $COMMAND --type $TYPE --sandbox-id
 $SANDBOX-ID
```

- `--command`：需要执行的命令。
- `--sandbox-id`：sandboxId 或 sandboxArn。
- `--type`：命令类型，SHELL。

Sample request

```
aws codebuild start-command-execution --command "echo "Hello World"" --type SHELL --
sandbox-id "arn:aws:codebuild:us-west-2:962803963624:sandbox/project-name
```

Sample response

```
{
    "id": "e1c658c2-02bb-42a8-9abb-94835241fcd6",
    "sandboxId": "f7126a4a-b0d5-452f-814c-fea73718f805",
    "submitTime": "2025-02-06T20:12:02.683000-08:00",
    "status": "SUBMITTED",
    "command": "echo \"Hello World\"",
    "type": "SHELL",
    "logs": {
        "groupName": "group",
        "streamName": "stream",
        "deepLink": "https://console.aws.amazon.com/cloudwatch/home?region=us-
west-2#logsV2:log-groups/log-group/group/log-events/stream",
        "s3DeepLink": "https://s3.console.aws.amazon.com/s3/object/codefactory-test-
pool-1-us-west-2-beta-default-build-logs/f7126a4a-b0d5-452f-814c-fea73718f805.gz?
region=us-west-2",
        "cloudWatchLogsArn": "arn:aws:logs:us-west-2:962803963624:log-
group:group:log-stream:stream",
        "s3LogsArn": "arn:aws:s3:::codefactory-test-pool-1-us-west-2-beta-default-
build-logs/f7126a4a-b0d5-452f-814c-fea73718f805.gz",
        "cloudWatchLogs": {
            "status": "ENABLED",
            "groupName": "group",
            "streamName": "stream"
        },
        "s3Logs": {
            "status": "ENABLED",
            "location": "codefactory-test-pool-1-us-west-2-beta-default-build-logs",
            "encryptionDisabled": false
        }
    }
}
```

获取有关命令执行的信息 (AWS CLI)

CLI command

```
aws codebuild batch-get-command-executions --command-execution-ids $COMMAND-IDs --
sandbox-id $SANDBOX-IDs
```

- `--command-execution-ids`：commandExecutionIds 的逗号分隔列表。
- `--sandbox-id`：sandboxId 或 sandboxArn。

Sample request

```
aws codebuild batch-get-command-executions --command-execution-
ids"c3c085ed-5a8f-4531-8e95-87d547f27ffd" --sandbox-id "arn:aws:codebuild:us-
west-2:962803963624:sandbox/project-name"
```

Sample response

```
{
    "commandExecutions": [{
        "id": "c3c085ed-5a8f-4531-8e95-87d547f27ffd",
        "sandboxId": "cd71e456-2a4c-4db4-ada5-da892b0bba05",
        "submitTime": "2025-02-10T20:18:17.118000-08:00",
        "startTime": "2025-02-10T20:18:17.939000-08:00",
        "endTime": "2025-02-10T20:18:17.976000-08:00",
        "status": "SUCCEEDED",
        "command": "echo \"Hello World\"",
        "type": "SHELL",
        "exitCode": "0",
        "standardOutputContent": "Hello World\n",
        "logs": {
            "groupName": "group",
            "streamName": "stream",
            "deepLink": "https://console.aws.amazon.com/cloudwatch/home?region=us-
west-2#logsV2:log-groups/log-group/group/log-events/stream",
            "s3DeepLink": "https://s3.console.aws.amazon.com/s3/object/
codefactory-test-pool-1-us-west-2-beta-default-build-logs/cd71e456-2a4c-4db4-ada5-
da892b0bba05.gz?region=us-west-2",
            "cloudWatchLogsArn": "arn:aws:logs:us-west-2:962803963624:log-
group:group:log-stream:stream",
```

```
            "s3LogsArn": "arn:aws:s3:::codefactory-test-pool-1-us-west-2-beta-
default-build-logs/cd71e456-2a4c-4db4-ada5-da892b0bba05.gz",
            "cloudWatchLogs": {
                "status": "ENABLED",
                "groupName": "group",
                "streamName": "stream"
            },
            "s3Logs": {
                "status": "ENABLED",
                "location": "codefactory-test-pool-1-us-west-2-beta-default-build-
logs",
                "encryptionDisabled": false
            }
        }
    }],
    "commandExecutionsNotFound": []
}
```

列出沙箱的命令执行情况 (AWS CLI)

CLI command

```
aws codebuild list-command-executions-for-sandbox --sandbox-id $SANDBOX-ID --next-
token $NEXT_TOKEN --max-results $MAX_RESULTS --sort-order $SORT_ORDER
```

- `--next-token`：用于获取分页结果的下一个令牌（如果有）。您将从先前执行的列表沙盒中获得此值。
- `--max-results`：（可选）要检索的最大沙盒记录数。
- `--sort-order`：应检索沙盒记录的顺序。

Sample request

```
aws codebuild list-command-executions-for-sandbox --sandbox-id
 "arn:aws:codebuild:us-west-2:962803963624:sandbox/project-name"
```

Sample response

```
{
    "commandExecutions": [{
            "id": "aad6687e-07bc-45ab-a1fd-f5440229b528",
```

```
            "sandboxId": "cd71e456-2a4c-4db4-ada5-da892b0bba05",
            "submitTime": "2025-02-10T20:18:35.304000-08:00",
            "startTime": "2025-02-10T20:18:35.615000-08:00",
            "endTime": "2025-02-10T20:18:35.651000-08:00",
            "status": "FAILED",
            "command": "fail command",
            "type": "SHELL",
            "exitCode": "127",
            "standardErrContent": "/codebuild/output/tmp/script.sh: 4: fail: not
 found\n",
            "logs": {
                "groupName": "group",
                "streamName": "stream",
                "deepLink": "https://console.aws.amazon.com/cloudwatch/home?
region=us-west-2#logsV2:log-groups/log-group/group/log-events/stream",
                "s3DeepLink": "https://s3.console.aws.amazon.com/s3/object/
codefactory-test-pool-1-us-west-2-beta-default-build-logs/cd71e456-2a4c-4db4-ada5-
da892b0bba05.gz?region=us-west-2",
                "cloudWatchLogsArn": "arn:aws:logs:us-west-2:962803963624:log-
group:group:log-stream:stream",
                "s3LogsArn": "arn:aws:s3:::codefactory-test-pool-1-us-west-2-beta-
default-build-logs/cd71e456-2a4c-4db4-ada5-da892b0bba05.gz",
                "cloudWatchLogs": {
                    "status": "ENABLED",
                    "groupName": "group",
                    "streamName": "stream"
                },
                "s3Logs": {
                    "status": "ENABLED",
                    "location": "codefactory-test-pool-1-us-west-2-beta-default-
build-logs",
                    "encryptionDisabled": false
                }
            }
        },
        {
            "id": "c3c085ed-5a8f-4531-8e95-87d547f27ffd",
            "sandboxId": "cd71e456-2a4c-4db4-ada5-da892b0bba05",
            "submitTime": "2025-02-10T20:18:17.118000-08:00",
            "startTime": "2025-02-10T20:18:17.939000-08:00",
            "endTime": "2025-02-10T20:18:17.976000-08:00",
            "status": "SUCCEEDED",
            "command": "echo \"Hello World\"",
            "type": "SHELL",
```

```
            "exitCode": "0",
            "standardOutputContent": "Hello World\n",
            "logs": {
                "groupName": "group",
                "streamName": "stream",
                "deepLink": "https://console.aws.amazon.com/cloudwatch/home?
region=us-west-2#logsV2:log-groups/log-group/group/log-events/stream",
                "s3DeepLink": "https://s3.console.aws.amazon.com/s3/object/
codefactory-test-pool-1-us-west-2-beta-default-build-logs/cd71e456-2a4c-4db4-ada5-
da892b0bba05.gz?region=us-west-2",
                "cloudWatchLogsArn": "arn:aws:logs:us-west-2:962803963624:log-
group:group:log-stream:stream",
                "s3LogsArn": "arn:aws:s3:::codefactory-test-pool-1-us-west-2-beta-
default-build-logs/cd71e456-2a4c-4db4-ada5-da892b0bba05.gz",
                "cloudWatchLogs": {
                    "status": "ENABLED",
                    "groupName": "group",
                    "streamName": "stream"
                },
                "s3Logs": {
                    "status": "ENABLED",
                    "location": "codefactory-test-pool-1-us-west-2-beta-default-
build-logs",
                    "encryptionDisabled": false
                }
            }
        }
    ]
}
```

列出沙箱 (AWS CLI)

CLI command

```
aws codebuild list-sandboxes --next-token $NEXT_TOKEN --max-results $MAX_RESULTS --
sort-order $SORT_ORDER
```

Sample request

```
aws codebuild list-sandboxes
```

Sample response


```
{
    "ids": [
        "s3-log-project-integ-test-temp1739250628149 85d64e0f-7880-41df-9a3c-
fb6597a266d2:827a5243-0841-4b69-a720-4438796f6967",
        "s3-log-project-integ-test-temp1739249999716bbd438dd-8bb8-47bd-
ba6b-0133ac65b3d3:e2fa4eab-73af-42e3-8903-92fddaf9f378",
        "s3-log-project-integ-test-
temp17392474779450fbdacc2-2d6e-4190-9ad5-28f891bb7415:cd71e456-2a4c-4db4-ada5-
da892b0bba05",
        "s3-log-project-integ-test-temp1739224628416430 1421c-5030-4fa1-b4d3-
ca15e44771c5:9e26ab3f-65e4-4896-a19c-56b1a95e630a",
        "s3-log-project-integ-test-temp17392136731949705 6d8d-6d8e-4f5a-a37c-
a62f5686731f:22d91b06-df1e-4e9c-a664-c0abb8d5920b",
        "s3-log-project-integ-test-temp1739213439503f6283f19-390c-4dc8-95a9-
c8480113384a:82cc413e-fc46-47ab-898f-ae23c83a613f",
        "s3-log-project-integ-test-temp1739054385570b1f1ddc2-0a23-4062-
bd0c-24e9e4a99b99:c02562f3-2396-42ec-98da-38e3fe5da13a",
        "s3-log-project-integ-test-temp173905400540237dab1ac-1fde-4dfb-a8f5-
c0114333dc89:d2f30493-f65e-4fa0-a7b6-08a5e77497b9",
        "s3-log-project-integ-test-
temp17390534055719c534090-7bc4-48f1-92c5-34acaec5bf1e:df5f1c8a-f017-43b7-91ba-
ad2619e2c059",
        "s3-log-project-integ-test-temp1739052719086a61813cc-
ebb9-4db4-9391-7f43cc984ee4:d61917ec-8037-4647-8d52-060349272c4a",
        "s3-log-project-integ-test-temp173898670094078b67edb-
c42f-42ed-9db2-4b5c1a5fc66a:ce33dfbc-beeb-4466-8c99-a3734a0392c7",
        "s3-log-project-integ-test-
temp17389863425584d21b7cd-32e2-4f11-9175-72c89ecaffef:046dadf0-1f3a-4d51-a2c0-
e88361924acf",
        "s3-log-project-integ-test-
temp1738985884273977ccd23-394b-46cc-90d3-7ab94cf764dc:0370dc41-9339-4b0a-91ed-51929761b244",
        "s3-log-project-integ-test-temp1738985365972241b614f-8e41-4387-
bd25-2b8351fbc9e0:076c392a-9630-47d8-85a9-116aa34edfff",
        "s3-log-project-integ-test-
temp1738985043988a51a9e2b-09d6-4d24-9c3c-1e6e21ac9fa8:6ea3949c-435b-4177-
aa4d-614d5956244c",
        "s3-log-project-integ-test-temp1738984123354c68b31ad-49d1-4f4b-981d-
b66c00565ff6:6c3fff6c-815b-48b5-ada3-737400a6dee8",
        "s3-log-project-integ-test-
temp1738977263715d4d5bf6c-370a-48bf-8ea6-905358a6cf92:968a0f54-724a-42d1-9207-6ed854b2fae8",
```

```
        "s3-log-project-integ-test-
temp173897358796816ce8d7d-2a5e-41ef-855b-4a94a8d2795d:80f9a7ce-930a-402e-934e-
d8b511d68b04",
        "s3-log-project-integ-test-temp17389730633301af5e452-0966-467c-
b684-4e36d47f568c:cabbe989-2e8a-473c-af25-32edc8c28646",
        "s3-log-project-integ-test-temp1738901503813173fd468-
b723-4d7b-9f9f-82e88d17f264:f7126a4a-b0d5-452f-814c-fea73718f805",
        "s3-log-project-integ-test-temp1738890502472c13616fb-
bd0f-4253-86cc-28b74c97a0ba:c6f197e5-3a53-45b6-863e-0e6353375437",
        "s3-log-project-integ-test-
temp1738890304468361 0daf3-8da7-43c6-8580-9978432432ce:d20aa317-8838-4966-
bbfc-85b908213df1",
        "s3-log-project-integ-test-temp173888857196780b5ab8b-e54b-44fd-a222-
c5a374fffe96:ab4b9970-ffae-47a0-b3a8-7b6790008cad",
        "s3-log-project-integ-test-temp1738888336931c11d378d-e74d-49a4-
a723-3b92e6f7daac:4922f0e8-9b7d-4119-9c9f-115cd85e703e",
        "s3-log-project-integ-test-temp17388881717651612a397-c23f-4d88-
ba87-2773cd3fc0c9:be91c3fc-418e-4feb-8a3a-ba58ff8f4e8a",
        "s3-log-project-integ-test-
temp17388879727174c3c62ed-6195-4afb-8a03-59674d0e1187:a48826a8-3c0d-43c5-
a1b5-1c98a0f978e9",
        "s3-log-project-integ-test-temp1738885948597cef305e4-b8b4-46b0-a65b-
e2d0a7b83294:c050e77d-e3f8-4829-9a60-46149628fe96",
        "s3-log-project-integ-test-temp173888561463001a7d2a8-
e4e4-4434-94db-09d3da9a9e17:8c3ac3f5-7111-4297-aec9-2470d3ead873",
        "s3-log-project-integ-test-
temp1738869855076eb19cafd-04fe-41bd-8aa0-40826d0c0d27:d25be134-05cb-404a-85da-
ac5f85d2d72c",
        "s3-project-integ-test-temp1738868157467148eacfc-d39b-49fc-a137-
e55381cd2978:4909557b-c221-4814-b4b6-7d9e93d37c35",
        "s3-project-integ-test-temp1738820926895abec0af2-
e33d-473c-9cf4-2122dd9d6876:8f5cf218-71d6-40a4-a4be-6cacebd7765f",
        "s3-project-integ-test-temp173881998877574f969a6-1c2e-4441-b463-
ab175b45ce32:04396851-c901-4986-9117-585528e3877f",
        "s3-project-integ-test-temp17388189812309abd2604-29ba-4cf6-
b6bf-073207b7db9c:540075c7-f5ec-41e8-9341-2233c09247eb",
        "s3-project-integ-test-temp1738818843474d3ea9ac1-b609-461b-
bbdb-2da245c9bc96:865d4c3c-fbfe-4ece-9c92-d0c928341404",
        "s3-project-integ-test-temp1738818542236006e9169-e6d9-4344-9b59-
f557e7aec619:1f9ffa87-da15-4290-83e2-eebdd877497b",
        "s3-project-integ-test-
temp173881809557486ad11fd-7931-48d7-81d5-499cea52a6bc:c4c2efc4-685f-4e13-8b0f-1ef85ec300b1",
        "s3-project-integ-test-temp1738817941033229410 20-3f0b-49c3-b836-
fcd818ec9484:0344cfba-de48-456d-b2a8-6566bd4a5d6e",
```

```
        "s3-project-integ-test-temp1738817680747b93d0d0b-ea16-497f-9559-
af25ee6dcfdf:654a3a55-d92a-4dc6-8da8-56fd4d40d7e1",
        "s3-project-integ-test-temp17388174027191255c3da-086c-4270-b047-
acac0b7bee0d:b7e82740-2c69-42fc-ab5a-dbf15bc016a1",
        "s3-project-integ-test-temp1738817099799016e7fa3-b9b5-46a2-
bcd5-0888c646743f:8705a6a4-79ff-427a-a1c3-85c4e8fe462e",
        "s3-project-integ-test-temp1738816479281bb0c3606-5ebf-4623-
bed5-12b60e9d3512:f23fc74b-a981-4835-8e28-375fcd4c99e4",
        "s3-project-integ-test-
temp1738816263585c939a133-4d37-482c-9238-1dbff34b7674:ca28e234-0045-4ae6-8732-938b17597f50",
        "s3-project-integ-test-
temp173881580873072d18733-8fe4-43b1-83f7-95f25bb27ccf:c6f0f55b-5736-47c7-
a3aa-1b8461a6d5ed"
    ]
}
```

教程：使用 SSH 连接到沙盒

本教程向您展示如何使用 SSH 客户端连接到 CodeBuild 沙箱。

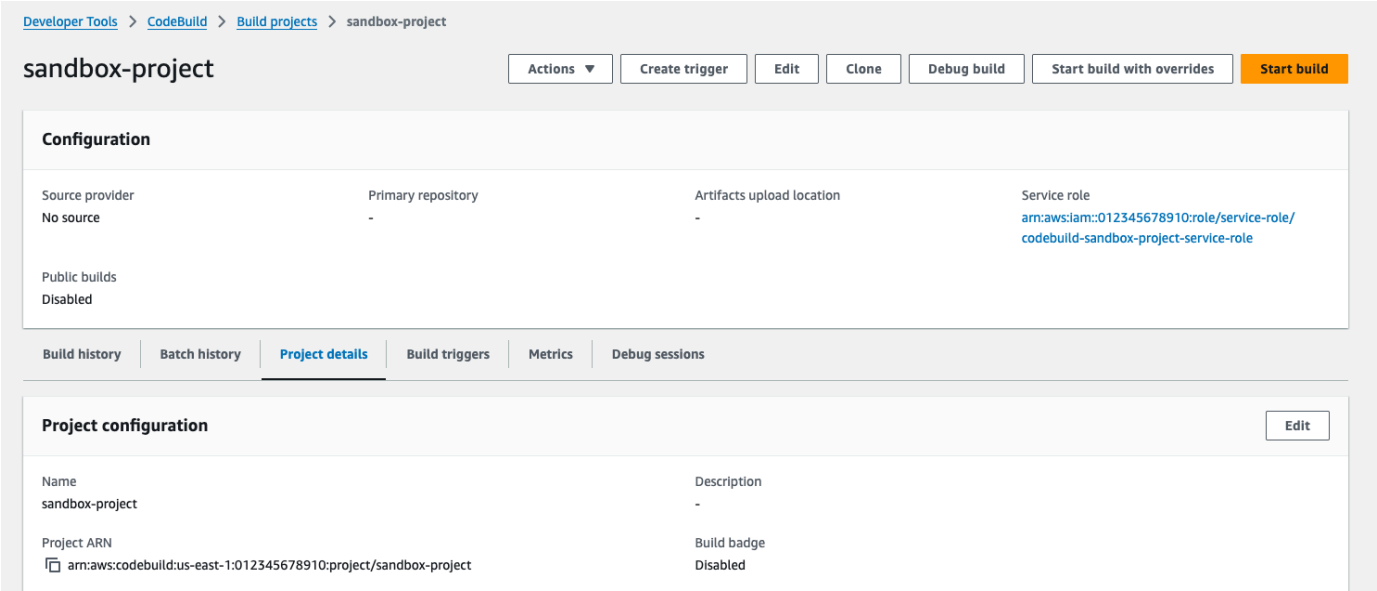
要完成本教程，您首先必须：

- 确保你有一个现有的 AWS CodeBuild 项目。
- 设置为您的 CodeBuild 项目角色配置的适当的 IAM 权限。
- 在本地计算机 AWS CLI 上安装和配置。

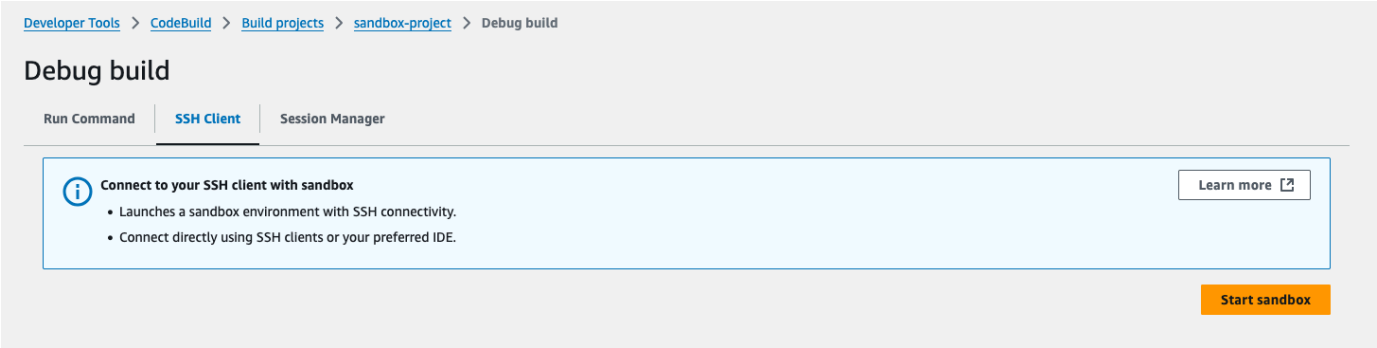
步骤 1：启动沙盒

在控制台中启动 CodeBuild 沙箱

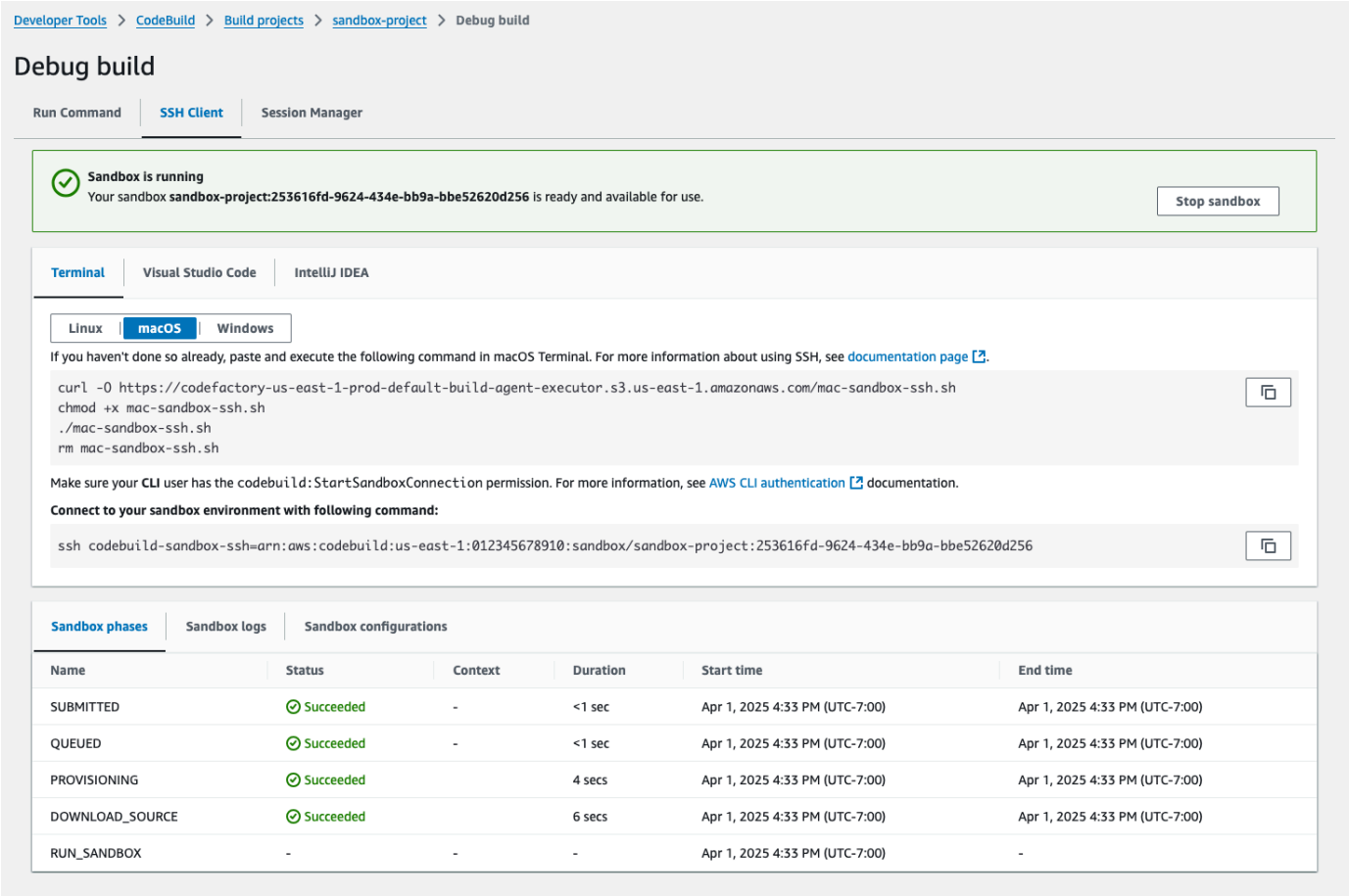
1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择构建项目。选择构建项目，然后选择调试构建。



3. 在 SSH 客户端选项卡中，选择启动沙盒。



4. 沙盒初始化过程可能需要一些时间。当沙盒的状态更改为 RUN_SANDDBOX 时，您可以连接到沙盒。

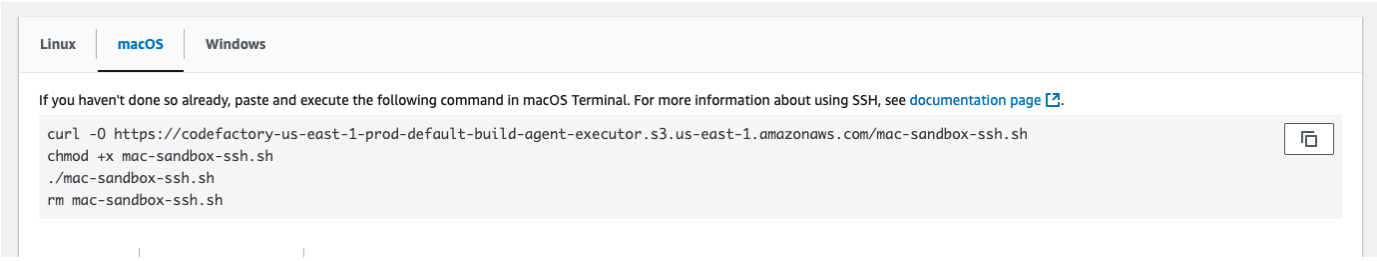


步骤 2：修改本地 SSH 配置

如果您是首次连接到沙盒，则需要使用以下步骤执行一次性设置过程：

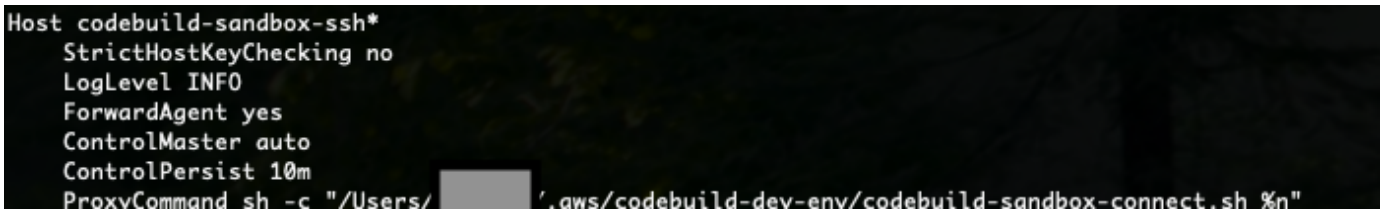
在控制台中修改本地 SSH 配置

1. 找到适用于您的操作系统的安装命令。
2. 打开本地终端，然后复制并执行提供的命令，以下载并运行脚本来设置本地 SSH 配置。例如，如果您的操作系统是 macOS，请使用以下命令：



3. 配置脚本将添加连接到沙盒所需的配置。系统将提示您接受这些更改。

4. 成功配置后，将为 CodeBuild 沙箱创建一个新的 SSH 配置条目。



步骤 3：连接到沙盒

在控制台中修改本地 SSH 配置

1. 配置 AWS CLI 身份验证并确保您的 AWS CLI 用户拥有`codebuild:StartSandboxConnection`权限。有关更多信息，请参阅《AWS 命令行界面版本 1 的用户指南》中的在 AWS CLI中使用 IAM 用户凭证进行身份验证。
2. 使用以下命令连接到您的沙盒：

```
ssh codebuild-sandbox-ssh=arn:aws:codebuild:us-east-1:<account-id>:sandbox/<sandbox-id>
```

> **Note**
>
> 要对连接失败进行故障排除，请使用 -v 标志启用详细输出。例如 `ssh -v codebuild-sandbox-ssh=arn:aws:codebuild:us-east-1:<account-id>:sandbox/<sandbox-id>`。
> 有关其它故障排除指南，请参阅问题排查 AWS CodeBuild 沙箱 SSH 连接问题。

第 4 步：查看结果

连接后，您可以调试构建失败、测试构建命令、试验配置更改以及使用沙盒验证环境变量和依赖项。

问题排查 AWS CodeBuild 沙箱 SSH 连接问题

使用本主题中的信息来帮助您识别、诊断和解决 CodeBuild 沙箱 SSH 连接问题。

主题

- StartSandboxConnectionInvalidInputExceptionSSH 进入 CodeBuild 沙盒环境时出错
- 错误：SSH 进入 CodeBuild 沙盒环境时出现 “无法找到凭据”

- StartSandboxConnectionAccessDeniedExceptionSSH 进入 CodeBuild 沙盒环境时出错
- SSH 进入 CodeBuild 沙盒环境时出现错误：“ssh：无法解析主机名”

StartSandboxConnectionInvalidInputExceptionSSH 进入 CodeBuild 沙盒环境时出错

问题：尝试使用命令连接到 CodeBuild 沙盒环境时`ssh codebuild-sandbox-ssh=<sandbox-arn>`，可能会遇到`InvalidInputException`错误，例如：

```
An error occurred (InvalidInputException) when calling the StartSandboxConnection
operation: Failed to start SSM session for {sandbox-arn}
User: arn:aws:sts::<account-ID>:assumed-role/<service-role-name>/AWSCodeBuild-<UUID>
is not authorized to perform: ssm:StartSession on resource.
```

```
An error occurred (InvalidInputException) when calling the StartSandboxConnection
operation: Failed to start SSM session for
sandbox <sandbox-arn>: codebuild:<UUID> is not connected.
```

可能的原因：

- 缺少 Amazon EC2 Systems Manager 代理：构建映像未正确地安装或配置 SSM 代理。
- 权限不足： CodeBuild 项目服务角色缺少所需的 SSM 权限。

建议的解决方案：如果您针对构建使用自定义映像，请执行以下操作。

1. 安装 SSM 代理 有关更多信息，请参阅中的在适用于 Linux 的 Amazon EC2 实例上手动安装和卸载 SSM 代理。SSM 代理必须是 `3.0.1295.0` 或更高版本。
2. 将文件复制https://github.com/aws/aws-codebuild-docker-images/blob/master/ubuntu/standard/7.0/amazon-ssm-agent.json到图像中的`/etc/amazon/ssm/`目录中。这将在 SSM 代理中启用容器模式。
3. 确保 CodeBuild 项目的服务角色具有以下权限，然后重新启动沙盒环境：

```
{
   "Effect": "Allow",
      "Action": [
         "ssmmessages:CreateControlChannel",
         "ssmmessages:CreateDataChannel",
         "ssmmessages:OpenControlChannel",
         "ssmmessages:OpenDataChannel"
```

```
      ],
      "Resource": "*"
},
{
   "Effect": "Allow",
   "Action": [
      "ssm:StartSession"
    ],
    "Resource": [
       "arn:aws:codebuild:region:account-id:build/*",
       "arn:aws:ssm:region::document/AWS-StartSSHSession"
    ]
}
```

错误：SSH 进入 CodeBuild 沙盒环境时出现 “无法找到凭据”

问题：尝试使用命令连接到 CodeBuild 沙盒环境时`ssh codebuild-sandbox-ssh=<sandbox-arn>`，可能会遇到以下凭据错误：

```
Unable to locate credentials. You can configure credentials by running
"aws configure".
```

可能的原因：未在本地环境中正确配置 AWS 证书。

推荐的解决方案：按照官方文档配置您的 AWS CLI 证书：配置版本 2 AWS 命令行界面用户指南 AWS CLI中的设置。

`StartSandboxConnectionAccessDeniedException`SSH 进入 CodeBuild 沙盒环境时出错

问题：尝试使用命令连接到 CodeBuild 沙盒环境时`ssh codebuild-sandbox-ssh=<sandbox-arn>`，可能会遇到以下权限错误：

```
An error occurred (AccessDeniedException) when calling the StartSandboxConnection
operation:
User: arn:aws:sts::account-id:assumed-role/role-name
is not authorized to perform: codebuild:StartSandboxConnection on resource:
sandbox-arn
because no identity-based policy allows the codebuild:StartSandboxConnection action
```

可能的原因：您的 AWS 凭证缺少执行此操作所需的 CodeBuild 权限。

推荐的解决方案：确保与您的 AWS CLI 证书关联的 IAM 用户或角色具有以下权限：

```
{
    "Effect": "Allow",
    "Action": [
       "codebuild:StartSandboxConnection"
     ],
     "Resource": [
        "arn:aws:codebuild:region:account-id:sandbox/*"
     ]
}
```

SSH 进入 CodeBuild 沙盒环境时出现错误：“ssh：无法解析主机名”

问题：尝试使用命令连接到 CodeBuild 沙盒环境时`ssh codebuild-sandbox-ssh=<sandbox-arn>`，会遇到以下主机名解析错误：

```
ssh: Could not resolve hostname
```

可能的原因：此错误通常发生在本地环境中未正确执行所需的 CodeBuild 沙盒连接脚本时。

建议的解决方案：

1. 下载 CodeBuild 沙箱连接脚本。
2. 在终端中执行脚本以建立必要的 SSH 配置。
3. 重试与沙盒环境的 SSH 连接。

使用会话管理器调试构建

在中 AWS CodeBuild，您可以暂停正在运行的构建，然后使用 AWS Systems Manager 会话管理器连接到生成容器并查看容器的状态。

> **Note**
>
> 此功能在 Windows 环境中不可用。

主题

- 先决条件

- 暂停构建
- 启动构建。
- 连接到构建容器
- 恢复构建

先决条件

要允许会话管理器与构建会话一起使用，必须为构建启用会话连接。有两个先决条件：

- CodeBuild Linux 标准精选映像已经安装了 SSM 代理并启用了 SSM 代理。 ContainerMode

  如果您在针对构建使用自定义映像，请执行以下操作：

  1. 安装 SSM 代理 有关更多信息，请参阅《 AWS Systems Manager 用户指南》中的在 Linux EC2 实例上手动安装 SSM 代理。SSM 代理必须是 3.0.1295.0 或更高版本。
  2. 将文件复制https://github.com/aws/aws-codebuild-docker-images/blob/master/ubuntu/standard/5.0/amazon-ssm-agent.json到图像中的`/etc/amazon/ssm/`目录中。这将在 SSM 代理中启用容器模式。

  > **Note**
  >
  > 自定义映像需要最新的 SSM 代理才能使此功能按预期运行。

- CodeBuild 服务角色必须具有以下 SSM 策略：

  JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
     {
       "Effect": "Allow",
       "Action": [
          "ssmmessages:CreateControlChannel",
          "ssmmessages:CreateDataChannel",
          "ssmmessages:OpenControlChannel",
          "ssmmessages:OpenDataChannel"
       ],
       "Resource": "*"
```

```
      }
    ]
}
```

在开始构建时，您可以让 CodeBuild 控制台自动将此策略附加到您的服务角色。您也可以将此策略手动附加到您的服务角色。

- 如果您在 Systems Manager 首选项中启用了“审核和记录会话活动”，则 CodeBuild 服务角色还必须具有其他权限。权限会有所不同，具体取决于日志的存储位置。

  CloudWatch 日志

  如果使用 CloudWatch 日志存储日志，请向 CodeBuild 服务角色添加以下权限：

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "logs:DescribeLogGroups",
            "Resource": "arn:aws:logs:us-east-1:111122223333:log-group:*:*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:us-east-1:111122223333:log-
group:MyLogGroup:*"
        }
    ]
}
```

  Amazon S3

  如果使用 Amazon S3 存储您的日志，请向 CodeBuild 服务角色添加以下权限：

  JSON

```
{
  "Version":"2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetEncryptionConfiguration",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::<bucket-name>",
        "arn:aws:s3:::<bucket-name>/*"
      ]
    }
  ]
}
```

有关更多信息，请参阅《AWS Systems Manager 用户指南》中的审核和记录会话活动。

暂停构建

要暂停构建，请在构建规范文件中的任意构建阶段插入 codebuild-breakpoint 命令。此时将暂停构建，这样您就可以连接到构建容器并查看当前状态下的容器。

例如，将以下内容添加到您的构建规范文件中的构建阶段。

```
phases:
  pre_build:
    commands:
      - echo Entered the pre_build phase...
      - echo "Hello World" > /tmp/hello-world
      - codebuild-breakpoint
```

此代码会创建 /tmp/hello-world 文件，然后在此时暂停构建。

启动构建。

要允许会话管理器与构建会话一起使用，必须为构建启用会话连接。为此，开始构建时，请按照以下步骤执行：

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择构建项目。选择构建项目，然后选择使用覆盖启动构建。

3. 选择高级构建覆盖。
4. 在环境部分中，选择启用会话连接选项。如果未选择此选项，则会忽略所有 codebuild-breakpoint 和 codebuild-resume 命令。
5. 执行任何其他所需更改，然后选择启动构建。
6. 在控制台中监控构建状态。当会话可用时，AWS 会话管理器链接将显示在构建状态部分中。

连接到构建容器

您可以通过以下两种方式之一连接到构建容器：

CodeBuild 控制台

在 Web 浏览器中，打开 AWS 会话管理器链接以连接到构建容器。将打开一个终端会话，允许您浏览和控制构建容器。

AWS CLI

> **Note**
>
> 您的本地计算机必须安装会话管理器插件才能执行此过程。有关更多信息，请参阅《AWS Systems Manager 用户指南》中的“安装适用于 AWS CLI 的会话管理器插件”。

1. 使用构建 ID 调用 batch-get-builds api 以获取有关构建的信息，包括会话目标标识符。会话目标标识符属性名称因 aws 命令的输出类型而异。这就是为什么要将 `--output json` 添加到命令中。

```
aws codebuild batch-get-builds --ids <buildID> --region <region> --output json
```

2. 复制 `sessionTarget` 属性值。`sessionTarget` 属性名称会因 aws 命令的输出类型而异。这就是在上一步骤中将 `--output json` 添加到命令中的原因。
3. 使用以下命令连接到构建容器。

```
aws ssm start-session --target <sessionTarget> --region <region>
```

在此示例中，请验证 `/tmp/hello-world` 文件是否存在且包含文本 `Hello World`。

恢复构建

完成对构建容器的检查后，从容器 shell 中发出 codebuild-resume 命令。

```
$ codebuild-resume
```

删除内部版本 AWS CodeBuild

您可以使用 AWS CLI 或 S AWS DK 删除中的内部版本。 AWS CodeBuild

主题

- 删除构建 (AWS CLI)
- 删除构建 (AWS SDK)

删除构建 (AWS CLI)

运行 batch-delete-builds 命令：

```
aws codebuild batch-delete-builds --ids ids
```

在上述命令中，替换以下占位符：

- *ids*: 必填字符串。要删除的构建的 ID。要指定多个构建，请用空格将每个构建 ID 分隔开。要获取构建 ID 的列表，请参阅以下主题：
  - 查看构建 ID 的列表（AWS CLI）
  - 查看构建项目的构建 ID 列表 (AWS CLI)

如果成功，buildsDeleted 数组将显示在输出中，其中包含已成功删除的每个构建的 Amazon 资源名称 (ARN)。有关未成功删除的构建的信息将显示在输出中的 buildsNotDeleted 数组中。

例如，如果您运行此命令：

```
aws codebuild batch-delete-builds --ids my-demo-build-project:f8b888d2-5e1e-4032-8645-
b115195648EX my-other-demo-build-project:a18bc6ee-e499-4887-b36a-8c90349c7eEX
```

与以下内容类似的信息将显示在输出中：

```
{
  "buildsNotDeleted": [
    {
      "id": "arn:aws:codebuild:us-west-2:123456789012:build/my-demo-build-
project:f8b888d2-5e1e-4032-8645-b115195648EX",
      "statusCode": "BUILD_IN_PROGRESS"
    }
  ],
  "buildsDeleted": [
    "arn:aws:codebuild:us-west-2:123456789012:build/my-other-demo-build-
project:a18bc6ee-e499-4887-b36a-8c90349c7eEX"
  ]
}
```

删除构建 (AWS SDK)

有关与 AWS 软件开发工具包 AWS CodeBuild 一起使用的信息，请参阅。AWS SDK 和工具参考

重试手动构建 AWS CodeBuild

您可以使用 AWS CodeBuild 控制台或 AWS SDK 手动重试单一构建或批量构建。 AWS CLI AWS CodeBuild

主题

- 手动重试构建（控制台）
- 手动重试构建 (AWS CLI)
- 手动重试构建 (AWS 软件开发工具包）

手动重试构建（控制台）

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 请执行以下操作之一：
   - 如果显示***build-project-name*:*build-ID***页面，请选择 “重试构建”。
   - 在导航窗格中，选择构建历史记录。在构建列表中，选中构建的框，然后选择重试构建。
   - 在导航窗格中，选择构建项目。在构建项目列表中的名称列，选择构建项目名称的链接。在构建列表中，选中构建的框，然后选择重试构建。

> **Note**
>
> 默认情况下，仅显示最新的 100 个构建或构建项目。要查看更多构建或构建项目，请选择齿轮图标，然后为每页构建数或每页项目数选择其他值，或者使用向后和向前箭头。

手动重试构建 (AWS CLI)

- 运行 retry-build 命令：

```
aws codebuild retry-build --id <build-id> --idempotency-token <idempotencyToken>
```

在上述命令中，替换以下占位符：

- *<build-id>*: 必填字符串。要重试的构建或批量构建的 ID。要获取构建 ID 的列表，请参阅以下主题：
  - 查看构建 ID 的列表（AWS CLI）
  - 查看批量构建 ID 的列表 (AWS CLI)
  - 查看构建项目的构建 ID 列表 (AWS CLI)
  - 查看构建项目的批量构建 ID 列表 (AWS CLI)
- `--idempotency-token`：可选。如果您运行带该选项的 retry-build 命令，则 `retry-build` 请求将附带唯一的区分大小写的标识符或令牌。令牌在发出请求后的 5 分钟内有效。如果您使用相同的令牌重复`retry-build`请求，但更改了参数，则 CodeBuild 会返回参数不匹配错误。

手动重试构建 (AWS 软件开发工具包)

有关与 AWS SDK AWS CodeBuild 一起使用的更多信息，请参阅。AWS SDK 和工具参考

在中自动重试构建 AWS CodeBuild

您可以使用 AWS CodeBuild 控制台或 AWS 软件开发工具包在中自动重试构建。 AWS CLI AWS CodeBuild启用自动重试后， CodeBuild 将在构建失败后自动`RetryBuild`使用项目的服务角色进行调用，直至达到指定限制。例如，如果将自动重试限制设置为两次，则 CodeBuild 将调用 `RetryBuild` API 自动重试您的构建，最多再重试两次。

> **Note**
>
> CodeBuild 不支持自动重试。 CodePipeline

主题

- 自动重试构建（控制台）
- 自动重试构建（AWS CLI）
- 自动重试构建 (AWS SDK)

自动重试构建（控制台）

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 选择创建项目。有关更多信息，请参阅 创建构建项目（控制台） 和 运行构建（控制台）。
   - 在环境中：
     - 对于Auto-retry 限制，输入生成失败后所需的最大自动重试次数。
3. 在环境中，选择其他配置。
4. 继续使用默认值，然后选择创建构建项目。

自动重试构建（AWS CLI）

- 运行 create-project 命令：

```
aws codebuild create-project \
    --name "<project-name>" \
    --auto-retry-limit <auto-retry-limit> \
    --source "<source>" \
    --artifacts {<artifacts>} \
    --environment "{\"type\": \"environment-type>\",\"image\": \"image-type>\",
\"computeType\": \"compute-type>\"}" \
    --service-role "service-role>"
```

替换上一命令中的以下占位符：

- *<auto-retry-limit>*：将自动重试限制设置为生成失败后所需的最大自动重试次数。
- *<project-name>*、*<source>*、*<artifacts>environment-type>*、*image-type>*、*compute-type>*、和*service-role>*：设置所需的项目配置设置。

自动重试构建 (AWS SDK)

有关与 AWS 软件开发工具包 AWS CodeBuild 一起使用的更多信息，请参阅。AWS SDK 和工具参考

停止构建 AWS CodeBuild

您可以使用 AWS CodeBuild 控制台或 AWS SDK 来停止构建。 AWS CLI AWS CodeBuild

主题

- 停止构建（控制台）
- 停止构建 (AWS CLI)
- 停止构建 (AWS 软件开发工具包）

停止构建（控制台）

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 请执行以下操作之一：
   - 如果显示***build-project-name*:*build-ID***页面，请选择停止构建。
   - 在导航窗格中，选择构建历史记录。在构建列表中，选中构建的框，然后选择停止构建。
   - 在导航窗格中，选择构建项目。在构建项目列表中的名称列，选择构建项目名称的链接。在构建列表中，选中构建的框，然后选择停止构建。

> **Note**
>
> 默认情况下，仅显示最新的 100 个构建或构建项目。要查看更多构建或构建项目，请选择齿轮图标，然后为每页构建数或每页项目数选择其他值，或者使用向后和向前箭头。
> 如果 AWS CodeBuild 无法成功停止构建（例如，如果生成过程已经完成），“停止” 按钮将被禁用或可能不会出现。

停止构建 (AWS CLI)

- 运行 stop-build 命令：

```
aws codebuild stop-build --id id
```

在上述命令中，替换以下占位符：

- *id*: 必填字符串。要停止的构建的 ID。要获取构建 ID 的列表，请参阅以下主题：
  - 查看构建 ID 的列表（AWS CLI）
  - 查看构建项目的构建 ID 列表 (AWS CLI)

如果 AWS CodeBuild 成功停止构建，则输出中build对象的buildStatus值为STOPPED。

如果 CodeBuild 无法成功停止构建（例如，如果生成已经完成），则输出中build对象的buildStatus值为最终生成状态（例如，SUCCEEDED）。

停止构建 (AWS 软件开发工具包)

有关与 AWS SDK AWS CodeBuild 一起使用的更多信息，请参阅。AWS SDK 和工具参考

停止批量构建 AWS CodeBuild

您可以使用 AWS CodeBuild 控制台或 AWS SDK 停止批量构建。 AWS CLI AWS CodeBuild

> **Note**
> 如果您在批量构建中使用 Lambda 计算，则无法停止正在进行的 Lambda 构建。

主题

- 停止批量构建（控制台）
- 停止批量构建 (AWS CLI)
- 停止批量构建 (AWS 软件开发工具包)

停止批量构建（控制台）

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 请执行以下操作之一：
   - 如果显示***build-project-name*:*build-ID***页面，请选择停止构建。
   - 在导航窗格中，选择构建历史记录。在构建列表中，选中构建的框，然后选择停止构建。
   - 在导航窗格中，选择构建项目。在构建项目列表中的名称列，选择构建项目名称的链接。在构建列表中，选中构建的框，然后选择停止构建。

> **Note**
>
> 默认情况下，仅显示最新的 100 个构建或构建项目。要查看更多构建或构建项目，请选择齿轮图标，然后为每页构建数或每页项目数选择其他值，或者使用向后和向前箭头。

停止批量构建 (AWS CLI)

- 运行 `stop-build-batch` 命令：

```
aws codebuild stop-build-batch --id <batch-build-id>
```

在上述命令中，替换以下占位符：

- *<batch-build-id>*: 必填字符串。要停止的批量构建的标识符。要获取批量构建标识符的列表，请参阅以下主题：
  - 查看批量构建 ID 的列表 (AWS CLI)
  - 查看构建项目的批量构建 ID 列表 (AWS CLI)

停止批量构建 (AWS 软件开发工具包)

有关与 AWS SDK AWS CodeBuild 一起使用的更多信息，请参阅。AWS SDK 和工具参考

触发器会自动 AWS CodeBuild 构建

您可以在项目上创建触发器以安排每小时、每天或每周进行一次构建。您也可以编辑触发器以使用带有 Amaz CloudWatch on cron 表达式的自定义规则。例如，通过使用 cron 表达式，您可以安排在每个工作日的特定时间进行构建。有关创建和编辑触发器的信息，请参阅创建 AWS CodeBuild 触发器和编辑 AWS CodeBuild 触发器。

主题

- 创建 AWS CodeBuild 触发器
- 编辑 AWS CodeBuild 触发器

创建 AWS CodeBuild 触发器

您可以在项目上创建触发器以安排每小时、每天或每周进行一次构建。您也可以使用带有 Amazon CloudWatch cron 表达式的自定义规则来创建触发器。例如，通过使用 cron 表达式，您可以安排在每个工作日的特定时间进行构建。

> **Note**
>
> 无法通过生成触发器、Amazon EventBridge 事件或 AWS Step Functions 任务启动批量构建。

主题

- 创建 AWS CodeBuild 触发器（控制台）
- 以编程方式创建 AWS CodeBuild 触发器

创建 AWS CodeBuild 触发器（控制台）

按照以下过程使用 AWS 管理控制台创建触发器。

创建触发器

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择构建项目。
3. 选择要将触发器添加到的构建项目的链接，然后选择构建触发器选项卡。

> **Note**
>
> 默认情况下，会显示 100 个最新的构建项目。要查看更多构建项目，请选择齿轮图标，然后为每页项目数选择不同值，或使用向后和向前箭头。

4. 选择创建触发器。
5. 在触发器名称中输入名称。
6. 从频率下拉列表中，选择触发器的频率。如果要使用 Cron 表达式创建频率，请选择自定义。
7. 为触发器的频率指定参数。您可以在文本框中输入您的选项的前几个字符以筛选下拉菜单项。

> **Note**
>
> 开始小时和分钟是从零开始的。开始分钟是一个介于 0 和 59 之间的数字。开始小时是一个介于 0 和 23 之间的数字。例如，每天 12:15 开始的每日触发器的起始时间为 12，开始分钟为 15 P.M。每天午夜开始的每日触发器的开始小时为 0，开始分钟为 0。每天 11:59 开始的每日触发器 P.M。起始时间为 23，开始分钟为 59。

频率	必需参数	Details
每小时	开始分钟	使用开始分钟下拉菜单。
每天	开始分钟	使用开始分钟下拉菜单。
	开始小时	使用开始小时下拉菜单。
Weekly	开始分钟	使用开始分钟下拉菜单。
	开始小时	使用开始小时下拉菜单。
	开始日	使用开始日下拉菜单。
自定义	Cron 表达式	在 Cron 表达式中输入 Cron 表达式。Cron 表达式有六个必填字段，各字段之间以空格分隔。这些字段分别指定分钟、小时、月中日、月、

频率	必需参数	Details
		周中日和年的开始值。您可以使用通配符指定范围、其他值等等。例如，cron 表达式在每个工作日 9:00 **`0 9 ? * MON-FRI *`** 安排一次构建。A.M有关更多信息，请参阅 Amazon Ev CloudWatch ents 用户指南中的 Cron 表达式。

8. 选择启用此触发器。
9. （可选）展开高级部分。在源版本中，键入源的版本。
   - 对于 Amazon S3，输入与您要构建的输入构件的版本相对应的版本 ID。如果源版本留空，则使用最新版本。
   - 对于 AWS CodeCommit，键入提交 ID。如果源版本留空，则使用默认分支的 HEAD 提交 ID。
   - 对于 GitHub 或 E GitHub nterprise，键入与要构建的源代码版本相对应的提交 ID、拉取请求 ID、分支名称或标签名称。如果您要指定拉取请求 ID，则必须使用格式 `pr/pull-request-ID`（例如，`pr/25`）。如果您要指定分支名称，则将使用分支的 HEAD 提交 ID。如果源版本留空，则将使用默认分支的 HEAD 提交 ID。
   - 对于 Bitbucket，键入提交 ID、分支名称或与您要构建的源代码版本对应的标签名称。如果您要指定分支名称，则将使用分支的 HEAD 提交 ID。如果源版本留空，则将使用默认分支的 HEAD 提交 ID。
10. （可选）指定介于 5 分钟和 2160 分钟（36 小时）之间的超时。此值指定 AWS CodeBuild 尝试生成多长时间后才会停止。如果小时和分钟保留为空，则使用项目中指定的默认超时值。
11. 选择创建触发器。

以编程方式创建 AWS CodeBuild 触发器

CodeBuild 将 Amazon EventBridge 规则用于生成触发器。您可以使用 EventBridge API 以编程方式为您的 CodeBuild 项目创建生成触发器。有关更多信息，请参阅 Amazon EventBridge API 参考。

编辑 AWS CodeBuild 触发器

您可以在项目上编辑触发器以安排每小时、每天或每周进行一次构建。您也可以编辑触发器以使用带有 Amaz CloudWatch on cron 表达式的自定义规则。例如，通过使用 cron 表达式，您可以安排在每个工作日的特定时间进行构建。有关创建触发器的信息，请参阅创建 AWS CodeBuild 触发器。

主题

- 编辑 AWS CodeBuild 触发器（控制台）
- 以编程方式编辑 AWS CodeBuild 触发器

编辑 AWS CodeBuild 触发器（控制台）

按照以下过程使用 AWS 管理控制台编辑触发器。

要编辑触发器，请执行以下操作：

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择构建项目。
3. 选择要更改的构建项目的链接，然后选择构建触发器选项卡。

> **Note**
>
> 默认情况下，会显示 100 个最新的构建项目。要查看更多构建项目，请选择齿轮图标，然后为每页项目数选择不同值，或使用向后和向前箭头。

4. 选择您要更改的触发器旁边的单选按钮，然后选择编辑。
5. 从频率下拉列表中，选择触发器的频率。如果要使用 Cron 表达式创建频率，请选择自定义。
6. 为触发器的频率指定参数。您可以在文本框中输入您的选项的前几个字符以筛选下拉菜单项。

> **Note**
>
> 开始小时和分钟是从零开始的。开始分钟是一个介于 0 和 59 之间的数字。开始小时是一个介于 0 和 23 之间的数字。例如，每天 12:15 开始的每日触发器的起始时间为 12，开始分钟为 15 P.M。每天午夜开始的每日触发器的开始小时为 0，开始分钟为 0。每天 11:59 开始的每日触发器 P.M。起始时间为 23，开始分钟为 59。

频率	必需参数	Details
每小时	开始分钟	使用开始分钟下拉菜单。
每天	开始分钟	使用开始分钟下拉菜单。
	开始小时	使用开始小时下拉菜单。
Weekly	开始分钟	使用开始分钟下拉菜单。
	开始小时	使用开始小时下拉菜单。
	开始日	使用开始日下拉菜单。
自定义	Cron 表达式	在 Cron 表达式中输入 Cron 表达式。Cron 表达式有六个必填字段，各字段之间以空格分隔。这些字段分别指定分钟、小时、月中日、月、周中日和年的开始值。您可以使用通配符指定范围、其他值等等。例如，cron 表达式在每个工作日 9:00 **`0 9 ? * MON-FRI *`** 安排一次构建。A.M有关更多信息，请参阅 Amazon Ev CloudWatch ents 用户指南中的 Cron 表达式。

7. 选择启用此触发器。

> **Note**
>
> 您可以使用 Amazon CloudWatch 控制台https://console.aws.amazon.com/cloudwatch/编辑源版本、超时以及中未提供的其他选项 AWS CodeBuild。

以编程方式编辑 AWS CodeBuild 触发器

CodeBuild 将 Amazon EventBridge 规则用于生成触发器。您可以使用 EventBridge API 以编程方式编辑 CodeBuild 项目的生成触发器。有关更多信息，请参阅 Amazon EventBridge API 参考。

在中查看版本详情 AWS CodeBuild

您可以使用 AWS CodeBuild 控制台或 AWS 软件开发工具包查看有关由 CodeBuild管理的版本的详细信息。 AWS CLI

主题

- 查看构建详细信息（控制台）
- 查看构建详细信息（AWS CLI）
- 查看版本详情 (AWS SDK)
- 构建阶段过渡

查看构建详细信息（控制台）

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 请执行以下操作之一：
   - 在导航窗格中，选择构建历史记录。在构建列表中的构建运行列，选择构建的链接。
   - 在导航窗格中，选择构建项目。在构建项目列表中的名称列，选择构建项目名称的链接。然后，在构建列表中的构建运行列，选择构建的链接。

> **Note**
>
> 默认情况下，仅显示最新的 10 个构建或构建项目。要查看更多构建或构建项目，请选择齿轮图标，然后为每页构建数或每页项目数选择其他值，或者使用向后和向前箭头。

查看构建详细信息（AWS CLI）

有关 AWS CLI 搭配使用的更多信息 AWS CodeBuild，请参阅命令行参考。

运行 batch-get-builds 命令：

```
aws codebuild batch-get-builds --ids ids
```

替换以下占位符：

- *ids*: 必填字符串。要查看其详细信息的一个或多个构建 ID。要指定多个构建 ID，请用空格分隔各个构建 ID。您最多可以指定 100 个构建 ID。要获取构建 ID 的列表，请参阅以下主题：
  - 查看构建 ID 的列表（AWS CLI）
  - 查看构建项目的构建 ID 列表 (AWS CLI)

例如，如果您运行此命令：

```
aws codebuild batch-get-builds --ids codebuild-demo-project:e9c4f4df-3f43-41d2-
ab3a-60fe2EXAMPLE codebuild-demo-project:815e755f-bade-4a7e-80f0-efe51EXAMPLE my-other-
project:813bb6c6-891b-426a-9dd7-6d8a3EXAMPLE
```

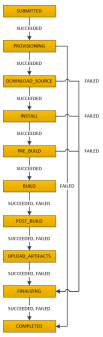
如果命令成功，与查看汇总的构建信息中所述内容类似的数据将出现在输出中。

查看版本详情 (AWS SDK)

有关与 AWS 软件开发工具包 AWS CodeBuild 一起使用的更多信息，请参阅。AWS SDK 和工具参考

构建阶段过渡

分阶段 AWS CodeBuild 进行构建：



> **Important**
>
> 系统始终尝试 UPLOAD_ARTIFACTS 阶段，即使 BUILD 阶段失败。

在中查看构建 ID 列表 AWS CodeBuild

您可以使用 AWS CodeBuild 控制台或 AWS 软件开发工具包查看由 CodeBuild管理的版本的构建 ID 列表。 AWS CLI

主题

- 查看构建 ID 的列表（控制台）
- 查看构建 ID 的列表（AWS CLI）
- 查看批量构建 ID 的列表 (AWS CLI)
- 查看构建 ID (AWS SDK) 列表

查看构建 ID 的列表（控制台）

1. 打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择构建历史记录。

> **Note**
>
> 默认情况下，仅显示 10 个最新构建。要查看更多构建，请选择齿轮图标，然后为每页构建数选择不同值，或使用向后和向前箭头。

查看构建 ID 的列表（AWS CLI）

有关 AWS CLI 搭配使用的更多信息 CodeBuild，请参阅命令行参考。

- 运行 list-builds 命令：

```
aws codebuild list-builds --sort-order sort-order --next-token next-token
```

替换上一命令中的以下占位符：

- *sort-order*：用于指示如何列出构建 ID 的可选字符串。有效值包括 ASCENDING 和 DESCENDING。

- *next-token*: 可选字符串。在上次运行时，如果列表中有 100 个以上的项目，则只能返回前 100 个项目，以及名为下一个令牌的唯一字符串。要获取列表中的下一批项目，请再次运行此命令，将下一个令牌添加到调用中。要获取列表中的所有项目，请利用每个后续的下一个令牌运行此命令，直到不再返回下一个令牌。

例如，如果您运行此命令：

```
aws codebuild list-builds --sort-order ASCENDING
```

与以下内容类似的结果可能会出现在输出中：

```
{
  "nextToken": "4AEA6u7J...The full token has been omitted for brevity...MzY2OA==",
  "ids": [
    "codebuild-demo-project:815e755f-bade-4a7e-80f0-efe51EXAMPLE"
    "codebuild-demo-project:84a7f3d1-d40e-4956-b4cf-7a9d4EXAMPLE"
    ... The full list of build IDs has been omitted for brevity ...
    "codebuild-demo-project:931d0b72-bf6f-4040-a472-5c707EXAMPLE"
  ]
}
```

如果您再次运行此命令：

```
aws codebuild list-builds --sort-order ASCENDING --next-token 4AEA6u7J...The full
 token has been omitted for brevity...MzY2OA==
```

与以下内容类似的结果可能会出现在输出中：

```
{
  "ids": [
    "codebuild-demo-project:49015049-21cf-4b50-9708-df115EXAMPLE",
    "codebuild-demo-project:543e7206-68a3-46d6-a4da-759abEXAMPLE",
    ... The full list of build IDs has been omitted for brevity ...
    "codebuild-demo-project:c282f198-4582-4b38-bdc0-26f96EXAMPLE"
  ]
}
```

查看批量构建 ID 的列表 (AWS CLI)

有关 AWS CLI 搭配使用的更多信息 CodeBuild，请参阅命令行参考。

- 运行 list-build-batches 命令：

```
aws codebuild list-build-batches --sort-order sort-order --next-token next-token
```

替换上一命令中的以下占位符：

- *sort-order*：用于指示如何列出批量构建 ID 的可选字符串。有效值包括 ASCENDING 和 DESCENDING。
- *next-token*: 可选字符串。在上次运行时，如果列表中有 100 个以上的项目，则只能返回前 100 个项目，以及名为下一个令牌的唯一字符串。要获取列表中的下一批项目，请再次运行此命令，将下一个令牌添加到调用中。要获取列表中的所有项目，请利用每个后续的下一个令牌运行此命令，直到不再返回下一个令牌。

例如，如果您运行此命令：

```
aws codebuild list-build-batches --sort-order ASCENDING
```

与以下内容类似的结果可能会出现在输出中：

```
{
  "nextToken": "4AEA6u7J...The full token has been omitted for brevity...MzY2OA==",
  "ids": [
    "codebuild-demo-project:815e755f-bade-4a7e-80f0-efe51EXAMPLE"
    "codebuild-demo-project:84a7f3d1-d40e-4956-b4cf-7a9d4EXAMPLE"
    ... The full list of build IDs has been omitted for brevity ...
    "codebuild-demo-project:931d0b72-bf6f-4040-a472-5c707EXAMPLE"
  ]
}
```

如果您再次运行此命令：

```
aws codebuild list-build-batches --sort-order ASCENDING --next-token 4AEA6u7J...The
 full token has been omitted for brevity...MzY2OA==
```

与以下内容类似的结果可能会出现在输出中：

```
{
  "ids": [
    "codebuild-demo-project:49015049-21cf-4b50-9708-df115EXAMPLE",
    "codebuild-demo-project:543e7206-68a3-46d6-a4da-759abEXAMPLE",
    ... The full list of build IDs has been omitted for brevity ...
    "codebuild-demo-project:c282f198-4582-4b38-bdc0-26f96EXAMPLE"
  ]
}
```

查看构建 ID (AWS SDK) 列表

有关与 AWS 软件开发工具包 CodeBuild 一起使用的更多信息，请参阅。AWS SDK 和工具参考

在中查看构建项目的构建 ID 列表 AWS CodeBuild

您可以使用 AWS CodeBuild 控制台或 AWS SDK 来查看中 CodeBuild构建项目的构建 ID 列表。 AWS CLI

主题

- 查看构建项目的构建 ID 列表（控制台）
- 查看构建项目的构建 ID 列表 (AWS CLI)
- 查看构建项目的批量构建 ID 列表 (AWS CLI)
- 查看构建项目的构建 ID 列表 (AWS 软件开发工具包）

查看构建项目的构建 ID 列表（控制台）

1. 在处打开 CodeBuild 控制台https://console.aws.amazon.com/codebuild/。
2. 在导航窗格中，选择构建项目。在构建项目列表中的名称列中，选择构建项目。

> **Note**
>
> 默认情况下，仅显示最新的 100 个构建或构建项目。要查看更多构建或构建项目，请选择齿轮图标，然后为每页构建数或每页项目数选择其他值，或者使用向后和向前箭头。

查看构建项目的构建 ID 列表 (AWS CLI)

有关 AWS CLI 与配合使用的更多信息 AWS CodeBuild，请参阅命令行参考。

运行 list-builds-for-project 命令，如下所示：

```
aws codebuild list-builds-for-project --project-name project-name --sort-order sort-order --next-token next-token
```

替换上一命令中的以下占位符：

- *project-name*：必填字符串，用于指明要列出构建 ID 的构建项目的名称。要获取构建项目的列表，请参阅查看构建项目名称的列表 (AWS CLI)。
- *sort-order*：用于指示如何列出构建 ID 的可选字符串。有效值包括 ASCENDING 和 DESCENDING。
- *next-token*：可选字符串。在上次运行时，如果列表中有 100 个以上的项目，则只能返回前 100 个项目，以及名为下一个令牌的唯一字符串。要获取列表中的下一批项目，请再次运行此命令，将下一个令牌添加到调用中。要获取列表中的所有项目，请借助随后返回的每个后续令牌不断运行此命令，直到不再返回后续令牌。

例如，如果你运行一个类似这样的命令：

```
aws codebuild list-builds-for-project --project-name codebuild-demo-project --sort-order ASCENDING
```

输出中可能会显示类似于以下内容的结果：

```
{
  "nextToken": "4AEA6u7J...The full token has been omitted for brevity...MzY2OA==",
  "ids": [
    "codebuild-demo-project:9b175d16-66fd-4e71-93a0-50a08EXAMPLE"
    "codebuild-demo-project:a9d1bd09-18a2-456b-8a36-7d65aEXAMPLE"
    ... The full list of build IDs has been omitted for brevity ...
    "codebuild-demo-project:fe70d102-c04f-421a-9cfa-2dc15EXAMPLE"
  ]
}
```

如果您再次运行此命令：

```
aws codebuild list-builds-for-project --project-name codebuild-demo-project --
sort-order ASCENDING --next-token 4AEA6u7J...The full token has been omitted for
 brevity...MzY2OA==
```

您可能会看到类似于以下输出的结果：

```
{
  "ids": [
    "codebuild-demo-project:98253670-7a8a-4546-b908-dc890EXAMPLE"
    "codebuild-demo-project:ad5405b2-1ab3-44df-ae2d-fba84EXAMPLE"
    ... The full list of build IDs has been omitted for brevity ...
    "codebuild-demo-project:f721a282-380f-4b08-850a-e0ac1EXAMPLE"
  ]
}
```

查看构建项目的批量构建 ID 列表 (AWS CLI)

有关 AWS CLI 与配合使用的更多信息 AWS CodeBuild，请参阅命令行参考。

运行 list-build-batches-for-project 命令，如下所示：

```
aws codebuild list-build-batches-for-project --project-name project-name --sort-
order sort-order --next-token next-token
```

替换上一命令中的以下占位符：

- *project-name*：必填字符串，用于指明要列出构建 ID 的构建项目的名称。要获取构建项目的列表，请参阅查看构建项目名称的列表 (AWS CLI)。
- *sort-order*：用于指示如何列出构建 ID 的可选字符串。有效值包括 ASCENDING 和 DESCENDING。
- *next-token*：可选字符串。在上次运行时，如果列表中有 100 个以上的项目，则只能返回前 100 个项目，以及名为下一个令牌的唯一字符串。要获取列表中的下一批项目，请再次运行此命令，将下一个令牌添加到调用中。要获取列表中的所有项目，请借助随后返回的每个后续令牌不断运行此命令，直到不再返回后续令牌。

例如，如果你运行一个类似这样的命令：

```
aws codebuild list-build-batches-for-project --project-name codebuild-demo-project --
sort-order ASCENDING
```

输出中可能会显示类似于以下内容的结果：

```
{
  "nextToken": "4AEA6u7J...The full token has been omitted for brevity...MzY2OA==",
  "ids": [
    "codebuild-demo-project:9b175d16-66fd-4e71-93a0-50a08EXAMPLE"
    "codebuild-demo-project:a9d1bd09-18a2-456b-8a36-7d65aEXAMPLE"
    ... The full list of build IDs has been omitted for brevity ...
    "codebuild-demo-project:fe70d102-c04f-421a-9cfa-2dc15EXAMPLE"
  ]
}
```

如果您再次运行此命令：

```
aws codebuild list-build-batches-for-project --project-name codebuild-demo-project
 --sort-order ASCENDING --next-token 4AEA6u7J...The full token has been omitted for
 brevity...MzY2OA==
```

您可能会看到类似于以下输出的结果：

```
{
  "ids": [
    "codebuild-demo-project:98253670-7a8a-4546-b908-dc890EXAMPLE"
    "codebuild-demo-project:ad5405b2-1ab3-44df-ae2d-fba84EXAMPLE"
    ... The full list of build IDs has been omitted for brevity ...
    "codebuild-demo-project:f721a282-380f-4b08-850a-e0ac1EXAMPLE"
  ]
}
```

查看构建项目的构建 ID 列表 (AWS 软件开发工具包)

有关与 AWS SDK AWS CodeBuild 一起使用的更多信息，请参阅。AWS SDK 和工具参考

中的测试报告 AWS CodeBuild

您可以在中创建 CodeBuild 包含有关在构建期间运行的测试的详细信息的报告。您可以创建诸如单元测试、配置测试和功能测试等测试。

支持以下测试报告文件格式：

- Cucumber JSON (.json)
- JUnit XML (.xml)
- NUnit XML (.xml)
- NUnit3 XML (.xml)
- TestNG XML (.xml)
- Visual Studio TRX (.trx)
- Visual Studio TRX XML (.xml)

> **Note**
>
> 支持的最新版本的 `cucumber-js` 是 7.3.2。

使用任何测试框架创建测试用例，这些测试框架可以采用任何一种格式创建报告文件（例如 Surefire JUnit 插件，TestNG 或 Cucumber）。

要创建测试报告，请将报告组名称添加到构建项目的 buildspec 文件中，该文件包含有关测试用例的信息。运行构建项目时，系统将运行测试用例并创建测试报告。每次测试用例运行时，都会在报告组中创建一个新的测试报告。您不需要在运行测试之前创建报告组。如果您指定报告组名称，则会在运行报告时为您 CodeBuild 创建一个报告组。如果要使用已存在的报告组，请在 buildspec 文件中指定其 ARN。

您可以使用测试报告帮助解决在构建运行期间发生的问题。如果您从构建项目的多个构建获得了许多测试报告，您可以使用测试报告查看趋势以及测试和失败率，以帮助您优化构建。

报告在创建后 30 天过期。您无法查看已过期的测试报告。如果您希望将测试报告保留 30 天以上，可以将测试结果的原始数据文件导出到 Amazon S3 存储桶。导出的测试文件不会过期。有关 S3 存储桶的信息在创建报告组时指定。

> **Note**
>
> 项目中指定的 CodeBuild 服务角色用于获得上传到 S3 存储桶的权限。

主题

- 创建测试报告
- 创建代码覆盖率报告
- Auto-discover 中的报告 CodeBuild
- 报告组
- 测试框架
- 查看测试报告
- 测试报告权限
- 测试报告状态

创建测试报告

要创建测试报告，您运行的构建项目应在 buildspec 文件中配置有一到五个报告组。测试报告在运行期间创建。它包含为报告组指定的测试用例的结果。对于使用相同构建规范文件的每个后续构建，系统将生成一个新的测试报告。

创建测试报告

1. 创建构建项目。有关信息，请参阅在中创建构建项目 AWS CodeBuild。
2. 使用测试报告信息配置项目的 buildspec 文件：

   a. 添加 `reports:` 部分并指定现有报告组的 ARN 或报告组的名称。

      如果您指定 ARN，则 CodeBuild 使用该报告组。

      如果您指定名称，则使用您的项目名称和您指定的名称（格式为 *<project-name>*-）为您 CodeBuild 创建一个报告组*<report-group-name>*。如果指定的报告组已经存在，则 CodeBuild 使用该报告组。

   b. 在报告组下，指定包含测试结果的文件的位置。如果您使用多个报告组，请为每个报告组指定测试结果文件位置。每次运行构建项目时都会创建一个新的测试报告。有关更多信息，请参阅指定测试文件。

c. 在 build 或 post_build 序列的 commands 部分中，指定将运行您为报告组指定的测试用例的命令。有关更多信息，请参阅 指定测试命令。

下面是一个 buildspec reports 部分示例：

```
reports:
  php-reports:
    files:
      - "reports/php/*.xml"
    file-format: "JUNITXML"
  nunit-reports:
    files:
      - "reports/nunit/*.xml"
    file-format: "NUNITXML"
```

3. 运行构建项目中的构建。有关更多信息，请参阅 手动运行 AWS CodeBuild 构建。
4. 构建完成后，从项目页面上的构建历史记录中选择新的构建运行。选择报告以查看测试报告。有关更多信息，请参阅 查看构建的测试报告。

创建代码覆盖率报告

CodeBuild 允许您为测试生成代码覆盖率报告。提供以下代码覆盖率报告：

行覆盖率

行覆盖率衡量您的测试涵盖了多少语句。语句是一条指令，不包括注释或条件。

```
line coverage = (total lines covered)/(total number of lines)
```

分支覆盖率

分支覆盖率衡量您的测试覆盖了控制结构（例如 if 或 case 语句）中所有可能的分支中的多少个分支。

```
branch coverage = (total branches covered)/(total number of branches)
```

支持以下代码覆盖率报告文件格式：

- JaCoCo XML
- SimpleCov JSON[1]

- Clover XML
- XML 覆盖范围
- LCOV INFO

[1] CodeBuild 接受 simplecov 生成的 JSON 代码覆盖率报告，而不是 s implecov-json。

创建代码覆盖率报告

要创建代码覆盖率报告，您需要运行一个在其 buildspec 文件中配置了至少一个代码覆盖率报告组的生成项目。 CodeBuild 将解释代码覆盖率结果并为运行提供代码覆盖率报告。对于使用相同构建规范文件的每个后续构建，系统将生成一个新的测试报告。

创建测试报告

1. 创建构建项目。有关信息，请参阅在中创建构建项目 AWS CodeBuild。
2. 使用测试报告信息配置项目的构建规范文件：

   a. 添加一个`reports:`分区并指定报告组的名称。 CodeBuild 使用您的项目名称和以`project-name-`格式指定的名称为您创建报告组`report-group-name-in-buildspec`。如果已存在要使用的报告组，请指定其 ARN。如果您使用名称而不是 ARN，则 CodeBuild 会创建一个新的报告组。有关更多信息，请参阅 Reports syntax in the buildspec file。

   b. 在报告组下，指定包含代码覆盖率结果的文件的位置。如果您使用多个报告组，请为每个报告组指定结果文件位置。每次运行构建项目时都会创建一个新的代码覆盖率报告。有关更多信息，请参阅 指定测试文件。

      这是一个为位于 test- JaCoCo 中的 XML 结果文件生成代码覆盖率报告的示例`results/jacoco-coverage-report.xml`。

      ```
      reports:
        jacoco-report:
          files:
            - 'test-results/jacoco-coverage-report.xml'
          file-format: 'JACOCOXML'
      ```

   c. 在 `build` 或 `post_build` 序列的 `commands` 部分中，指定用来运行代码覆盖率分析的命令。有关更多信息，请参阅 指定测试命令。

3. 运行构建项目中的构建。有关更多信息，请参阅 手动运行 AWS CodeBuild 构建。

4. 构建完成后，从项目页面上的构建历史记录中选择新的构建运行。选择报告来查看代码覆盖率报告。有关更多信息，请参阅 查看构建的测试报告。

Auto-discover 中的报告 CodeBuild

借助自动发现功能， CodeBuild 可以在构建阶段完成后搜索所有构建文件，搜索任何支持的报告文件类型，并自动创建新的测试和代码覆盖率报告组和报告。对于任何已发现的报告类型，使用以下模式 CodeBuild 创建新的报告组：

```
<project-name>-<report-file-format>-AutoDiscovered
```

> **Note**
>
> 如果发现的报告文件具有相同的格式类型，则它们将被置于同一个报告组或报告中。

通过项目环境变量配置报告自动发现：

`CODEBUILD_CONFIG_AUTO_DISCOVER`

此变量确定在构建期间是否禁用报告自动发现。默认情况下，所有构建均启用报告自动发现。要禁用此特征，请将 `CODEBUILD_CONFIG_AUTO_DISCOVER` 设置为 `false`。

`CODEBUILD_CONFIG_AUTO_DISCOVER_DIR`

（可选）此变量决定在何处 CodeBuild 搜索潜在报告文件。请注意，`**/*`默认情况下会进行 CodeBuild 搜索。

在构建阶段可以修改这些环境变量。例如，如果您只想为 `main` git 分支上的构建启用报告自动发现，则可以在构建过程中选中 git 分支，如果构建不在 `main` 分支上，则将 `CODEBUILD_CONFIG_AUTO_DISCOVER` 设置为 false。可以使用控制台或使用项目环境变量来禁用报告自动发现。

主题

- 使用控制台配置报告自动发现
- 使用项目环境变量配置报告自动发现

使用控制台配置报告自动发现

按照以下过程使用控制台来配置报告自动发现。

使用控制台配置报告自动发现

1. 创建构建项目或选择要编辑的构建项目。有关更多信息，请参阅 在中创建构建项目 AWS CodeBuild 或 在中更改构建项目设置 AWS CodeBuild。
2. 在环境中，选择其他配置。
3. 要禁用报告自动发现，在报告自动发现中，选择禁用报告自动发现。
4. （可选）在Auto-discover 目录-可选中，输入目录模式 CodeBuild 以搜索支持的报表格式文件。请注意，**/*默认情况下会进行 CodeBuild 搜索。

使用项目环境变量配置报告自动发现

按照以下过程使用项目环境变量来配置报告自动发现。

使用项目环境变量配置报告自动发现

1. 创建构建项目或选择要编辑的构建项目。有关更多信息，请参阅 在中创建构建项目 AWS CodeBuild 或 在中更改构建项目设置 AWS CodeBuild。
2. 在环境变量中，执行以下操作：
   a. 要禁用报告自动发现，在名称中输入 **CODEBUILD_CONFIG_AUTO_DISCOVER**，在值中输入 **false**。这将禁用报告自动发现。
   b. （可选）在 “名称” 中输入 **CODEBUILD_CONFIG_AUTO_DISCOVER_DIR**，对于 “值”，输入 CodeBuild 应在其中搜索支持的报表格式文件的目录。例如，`output/*xml` 在 `output` 目录中搜索 `.xml` 文件

报告组

报告组包含测试报告并指定共享设置。您可以使用 buildspec 文件指定要在构建时运行的测试用例以及运行它们的命令。对于在构建项目中配置的每个报告组，每次运行构建项目都会创建一个测试报告。多次运行配置有一个报告组的构建项目会在该报告组中创建多个测试报告，每个报告都包含为该报告组指定的相同测试用例的结果。

测试用例在构建项目的 buildspec 文件中针对报告组进行指定。您可以在一个构建项目中指定最多五个报告组。运行构建时，将运行所有测试用例。将创建一个新的测试报告，其中包含为报告组指定的每个测试用例的结果。每次运行新构建时，都会运行测试用例，并创建一个包含新测试结果的新测试报告。

报告组可用于多个构建项目。使用一个报告组创建的所有测试报告共享相同配置，如导出选项和权限，即使使用不同构建项目创建测试报告也是如此。在多个构建项目中使用一个报告组创建的测试报告可以包含运行不同测试用例集的结果（每个构建项目对应一个测试用例组）。这是因为您可以在每个项目的 buildspec 文件中为报告组指定不同的测试用例文件。您还可以通过编辑构建项目的 buildspec 文件来更改构建项目中的报告组的测试用例文件。后续运行构建会创建新的测试报告，其中包含更新的 buildspec 中的测试用例文件的结果。

主题

- 创建报告组
- 报告组命名
- 共享报告组
- 指定测试文件
- 指定测试命令
- 在中标记报告组 AWS CodeBuild
- 更新报告组

创建报告组

您可以使用 CodeBuild 控制台 AWS CLI、或 buildspec 文件来创建报告组。您的 IAM 角色必须具有创建报告组所需的权限。有关更多信息，请参阅 测试报告权限。

主题

- 创建报告组（buildspec）
- 创建报告组（控制台）
- 创建报告组（CLI）
- 创建报告组 (CloudFormation)

创建报告组（buildspec）

使用 buildspec 创建的报告组不会导出原始测试结果文件。您可以查看报告组并指定导出设置。有关更多信息，请参阅 更新报告组。

使用 buildspec 文件创建报告组

1. 选择与您 AWS 账户中的报告组无关的报告组名称。
2. 使用此名称配置 buildspec 文件的 `reports` 部分。在此示例中，报告组名称为 `new-report-group`，并使用 JUnit 框架创建使用测试用例：

```
reports:
 new-report-group: #surefire junit reports
   files:
     - '**/*'
   base-directory: 'surefire/target/surefire-reports'
```

也可以使用 buildspec 中的环境变量来指定报告组名称：

```
version: 0.2
env:
  variables:
    REPORT_GROUP_NAME: "new-report-group"
phases:
  build:
    commands:
      - ...
...
reports:
 $REPORT_GROUP_NAME:
   files:
     - '**/*'
   base-directory: 'surefire/target/surefire-reports'
```

有关更多信息，请参阅指定测试文件和Reports syntax in the buildspec file。

3. 在 `commands` 部分中，指定运行测试的命令。有关更多信息，请参阅 指定测试命令。
4. 运行构建。构建完成后，将使用 `project-name-report-group-name` 格式的名称创建一个新的报告组。有关更多信息，请参阅 报告组命名。

创建报告组（控制台）

按照以下过程使用 AWS 管理控制台创建报告组。

创建报告组

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择报告组。
3. 选择创建报告组。
4. 对于报告组名称，输入报告组的名称。
5. （可选）对于标签，输入您希望支持 AWS 服务使用的任何标签的名称和值。使用添加行添加标签。最多可以添加 50 个标签。
6. 如果您想将测试报告结果的原始数据上传到 Amazon S3 存储桶：

   a. 选择导出到 Amazon S3。

   b. 对于 S3 存储桶名称，请输入 S3 存储桶的名称。

   c. （可选）对于 S3 存储桶拥有者，请输入拥有 S3 存储桶的账户的 AWS 账户标识符。这允许将报告数据导出到 Amazon S3 桶，该存储桶由运行构建的账户以外的账户拥有。

   d. 对于路径前缀，请输入要上传测试结果的 S3 存储桶中的路径。

   e. 选择将测试结果数据压缩为 zip 文件以便压缩原始测试结果数据文件。

   f. 展开其他配置以显示加密选项。选择下列选项之一：

   - 用 AWS 托管式密钥 于 Amazon S3 的默认 AWS 托管密钥。有关更多信息，请参阅《AWS Key Management Service 用户指南》中的客户托管式 CMK。这是默认加密选项。
   - 选择自定义密钥将使用您创建和配置的客户托管密钥。对于AWS KMS 加密密钥，请输入加密密钥的 ARN。其格式为 `arn:aws:kms:<region-id>: <aws-account-id>:key/<key-id>` 。有关更多信息，请参阅《AWS Key Management Service 用户指南》中的创建 KMS 密钥。
   - 禁用构件加密将禁用加密。如果要共享测试结果或将其发布到静态网站，则可以选择此选项。（动态网站可以运行代码来解密测试结果。）

   有关静态数据加密的更多信息，请参阅数据加密。

   > **Note**
   >
   > 项目中指定的 CodeBuild 服务角色用于获得上传到 S3 存储桶的权限。

7. 选择创建报告组。

创建报告组（CLI）

按照以下过程使用 AWS CLI创建报告组。

创建报告组

1. 创建一个名为 `CreateReportGroup.json`的文件。
2. 根据您的要求，将以下 JSON 代码段之一复制到 `CreateReportGroup.json`：

   - 使用以下 JSON 指定测试报告组将原始测试结果文件导出到 Amazon S3 存储桶。

```
{
  "name": "<report-name>",
  "type": "TEST",
  "exportConfig": {
    "exportConfigType": "S3",
    "s3Destination": {
      "bucket": "<bucket-name>",
      "bucketOwner": "<bucket-owner>",
      "path": "<path>",
      "packaging": "NONE | ZIP",
      "encryptionDisabled": "false",
      "encryptionKey": "<your-key>"
    },
    "tags": [
      {
        "key": "tag-key",
        "value": "tag-value"
      }
    ]
  }
}
```

   - *<bucket-name>*替换为您的 Amazon S3 存储桶名称和*<path>*存储桶中要导出文件的路径。
   - 如果要压缩导出的文件，对于 `packaging`，请指定 `ZIP`。否则，请指定 `NONE`。
   - `bucketOwner` 是可选的，仅当 Amazon S3 存储桶由运行构建的账户以外的账户拥有时才是必需的。
   - 使用 `encryptionDisabled` 指定是否要加密导出的文件。如果要加密导出的文件，请输入客户托管密钥。有关更多信息，请参阅 更新报告组。

- 使用以下 JSON 指定测试报告不会导出原始测试文件：

```
{
  "name": "<report-name>",
  "type": "TEST",
  "exportConfig": {
    "exportConfigType": "NO_EXPORT"
  }
}
```

> **Note**
>
> 项目中指定的 CodeBuild 服务角色用于获得上传到 S3 存储桶的权限。

3. 运行以下命令：

```
aws codebuild create-report-group --cli-input-json file://
CreateReportGroupInput.json
```

创建报告组 (CloudFormation)

按照以下说明，使用 CloudFormation 模板创建报告组

使用 CloudFormation 模板创建报告组

您可以使用 CloudFormation 模板文件来创建和配置报告组。有关更多信息，请参阅 CloudFormation 用户指南。

以下 CloudFormation YAML 模板创建了一个不导出原始测试结果文件的报告组。

```
Resources:
  CodeBuildReportGroup:
    Type: AWS::CodeBuild::ReportGroup
    Properties:
      Name: my-report-group-name
      Type: TEST
      ExportConfig:
        ExportConfigType: NO_EXPORT
```

以下 CloudFormation YAML 模板创建了一个报告组，用于将原始测试结果文件导出到 Amazon S3 存储桶。

```
Resources:
  CodeBuildReportGroup:
    Type: AWS::CodeBuild::ReportGroup
    Properties:
      Name: my-report-group-name
      Type: TEST
      ExportConfig:
        ExportConfigType: S3
        S3Destination:
          Bucket: amzn-s3-demo-bucket
          Path: path-to-folder-for-exported-files
          Packaging: ZIP
          EncryptionKey: my-KMS-encryption-key
          EncryptionDisabled: false
```

> **Note**
>
> 项目中指定的 CodeBuild 服务角色用于获得上传到 S3 存储桶的权限。

报告组命名

当您使用 AWS CLI 或 AWS CodeBuild 控制台创建报告组时，需要为该报告组指定一个名称。如果您使用构建规范创建新的报告组，则使用 *project-name*-*report-group-name-specified-in-buildspec* 格式对其进行命名。通过运行该构建项目的构建所创建的所有报告都隶属于具有新名称的新报告组。

如果您不 CodeBuild 想创建新的报告组，请在构建项目的 buildspec 文件中指定该报告组的 ARN。您可以在多个构建项目中指定一个报告组的 ARN。运行每个构建项目后，报告组包含由每个构建项目创建的测试报告。

例如，如果您创建一个名为 `my-report-group` 的报告组，然后在两个名为 `my-project-1` 和 `my-project-2` 的不同构建项目中使用该报告组名称，并创建两个项目的构建，则会创建两个新的报告组。结果将生成三个具有以下名称的报告组：

- `my-report-group`：没有任何测试报告。

- my-project-1-my-report-group：包含由名为 my-project-1 的构建项目所运行的测试的结果报告。
- my-project-2-my-report-group：包含由名为 my-project-2 的构建项目所运行的测试的结果报告。

如果您在两个项目中使用名为 my-report-group 的报告组的 ARN，然后运行每个项目的构建，则只会生成一个报告组（my-report-group）。该报告组所含的测试报告包含由两个构建项目运行的测试的结果。

如果您选择的报告组名称不属于您的 AWS 账户中的报告组，然后在 buildspec 文件中将该名称用于报告组，并运行其构建项目的构建，则会创建一个新的报告组。新报告组的名称格式为 *project-name*-*new-group-name*。例如，如果您的 AWS 账户中没有名为的报表组new-report-group，并在名为的构建项目中指定了该报表组test-project，则生成运行会创建一个名为的新报告组test-project-new-report-group。

共享报告组

报告组共享允许多个 AWS 账户或用户查看报告组、其未过期的报告及其报告的测试结果。在此模型中，拥有报告组的账户（拥有者）将与其他账户（使用者）共享该报告组。使用者无法编辑报告组。报告在创建后 30 天过期。

主题

- 共享报告组
- 相关服务
- 访问与您共享的报告组
- 取消共享已共享的报告组
- 标识已共享的报告组
- 共享报告组权限

共享报告组

共享报告组时，将向使用者授予对报告组及其报告的只读访问权限。消费者可以使用 AWS CLI 来查看报告组、其报告以及每份报告的测试用例结果。使用者不能执行以下操作：

- 在 CodeBuild 控制台中查看共享报告组或其报告。
- 编辑共享报告组。

- 使用项目中的共享报告组的 ARN 运行报告。指定共享报告组的项目构建将失败。

您可以使用 CodeBuild 控制台将报告组添加到现有资源共享。如果要将报告组添加到新的资源共享，则必须首先在 AWS RAM 控制台中创建资源共享。

要与组织单位或整个组织共享报告组，您必须启用与 AWS Organizations的共享。有关更多信息，请参阅《AWS RAM 用户指南》中的允许与 AWS Organizations共享。

您可以使用 CodeBuild 控制台、控制 AWS RAM 台或 AWS CLI 共享您拥有的报表组。

先决条件

要共享举报组，必须由您的 AWS 账户拥有该群组。无法共享已与您共享的报告组。

共享您拥有的报表组（CodeBuild 控制台）

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择报告组。
3. 选择要共享的项目，然后选择共享。有关更多信息，请参阅《AWS RAM 用户指南》中的创建资源共享。

共享您拥有的报表群组 (AWS RAM 控制台）

请参阅《AWS RAM 用户指南》中的创建资源共享。

共享您拥有的报表群组 (AWS RAM 命令）

使用 create-resource-share 命令。

共享您拥有的报表组（CodeBuild 命令）

使用 put-resource-policy 命令：

1. 创建一个名为 `policy.json` 的文件，并将以下内容复制到该文件中。

   JSON

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

```
    "Principal":{
      "AWS":"111122223333"
    },
    "Action":[
      "codebuild:BatchGetReportGroups",
      "codebuild:BatchGetReports",
      "codebuild:ListReportsForReportGroup",
      "codebuild:DescribeTestCases"],
    "Resource":"arn:aws:iam::*:role/Service*"
  }]
}
```

2. 使用报告组 ARN 和标识符更新 `policy.json`，以便共享该报告组。以下示例向 Alice 和 123456789012 标识的 AWS 账户的根用户授予对具有 ARN `arn:aws:codebuild:us-west-2:123456789012:report-group/my-report-group` 的报告组的只读访问权限。

   JSON

```
{
  "Version":"2012-10-17",
  "Statement":[{
    "Effect":"Allow",
    "Principal":{
      "AWS": [
         "arn:aws:iam::123456789012:user/Alice",
         "123456789012"
       ]
    },
    "Action":[
      "codebuild:BatchGetReportGroups",
      "codebuild:BatchGetReports",
      "codebuild:ListReportsForReportGroup",
      "codebuild:DescribeTestCases"],
    "Resource":"arn:aws:codebuild:us-west-2:123456789012:report-group/my-report-group"
  }]
}
```

3. 运行如下命令。

```
aws codebuild put-resource-policy --resource-arn report-group-arn --policy file://policy.json
```

相关服务

报告群组共享与 AWS Resource Access Manager (AWS RAM) 集成，该服务使您可以与任何 AWS 账户或通过任何账户共享 AWS 资源 AWS Organizations。使用 AWS RAM，您可以通过创建资源共享来共享自己拥有的资源，该资源共享指定资源以及与之共享资源的使用者。消费者可以是个人 AWS 账户 AWS Organizations、中的组织单位或中的整个组织 AWS Organizations。

有关更多信息，请参阅 AWS RAM 《用户指南》。

访问与您共享的报告组

要访问共享报告组，使用者的 IAM 角色需要 `BatchGetReportGroups` 权限。您可以将以下策略附加到其 IAM 角色：

```
{
    "Effect": "Allow",
    "Resource": [
        "*"
    ],
    "Action": [
        "codebuild:BatchGetReportGroups"
    ]
}
```

有关更多信息，请参阅 将基于身份的策略用于 AWS CodeBuild。

取消共享已共享的报告组

取消共享的报告组（包括其报告及其测试用例结果）只能由其拥有者访问。如果您取消共享报告组，则之前与之共享的任何 AWS 账户或用户都无法访问该报告组、其报告或报告中测试用例的结果。

要取消共享您拥有的已共享报告组，必须从资源共享中将其删除。您可以使用 AWS RAM 控制台或执行 AWS CLI 此操作。

取消共享您拥有的共享报告群组（AWS RAM 控制台）

请参阅《AWS RAM 用户指南》中的更新资源共享。

取消共享您拥有的共享报告群组（AWS RAM 命令）

使用 disassociate-resource-share 命令。

取消共享您拥有的报表组（CodeBuild 命令）

运行 delete-resource-policy 命令，并指定要取消共享的报告组的 ARN：

```
aws codebuild delete-resource-policy --resource-arn report-group-arn
```

标识已共享的报告组

所有者和消费者可以使用 AWS CLI 来识别共享报告组。

要标识和获取有关共享报告组及其报告的信息，请使用以下命令：

- 要查看与您共享的报告组的 ARN，请运行 list-shared-report-groups：

  ```
  aws codebuild list-shared-report-groups
  ```

- 要查看报告组中的报告的 ARN，请使用报告组 ARN 运行 list-reports-for-report-group：

  ```
  aws codebuild list-reports-for-report-group --report-group-arn report-group-arn
  ```

- 要查看有关报告中的测试用例的信息，请使用报告 ARN 运行 describe-test-cases：

  ```
  aws codebuild describe-test-cases --report-arn report-arn
  ```

  输出内容如下所示：

  ```
  {
      "testCases": [
          {
              "status": "FAILED",
              "name": "Test case 1",
              "expired": 1575916770.0,
              "reportArn": "report-arn",
              "prefix": "Cucumber tests for agent",
              "message": "A test message",
              "durationInNanoSeconds": 1540540,
              "testRawDataPath": "path-to-output-report-files"
          },
          {
              "status": "SUCCEEDED",
              "name": "Test case 2",
  ```

```
            "expired": 1575916770.0,
            "reportArn": "report-arn",
            "prefix": "Cucumber tests for agent",
            "message": "A test message",
            "durationInNanoSeconds": 1540540,
            "testRawDataPath": "path-to-output-report-files"
        }
    ]
}
```

共享报告组权限

拥有者的权限

报告组拥有者可以编辑报告组，并在项目中指定该报告组以运行报告。

使用者的权限

报告组使用者可以查看报告组、报告以及报告的测试用例结果。使用者无法编辑报告组或其报告，也无法使用报告组创建报告。

指定测试文件

您可以在构建项目 buildspec 文件的 `reports` 部分中为每个报告组指定测试结果文件及其位置。有关更多信息，请参阅 Reports syntax in the buildspec file。

以下是 `reports` 部分的示例，该示例为构建项目指定了两个报告组。其中一个使用 ARN 指定，另一个使用名称指定。`files` 部分指定包含测试用例结果的文件。可选的 `base-directory` 部分指定测试用例文件所在的目录。可选的 `discard-paths` 部分指定是否丢弃将测试结果文件上传到 Amazon S3 存储桶的路径。

```
reports:
  arn:aws:codebuild:your-region:your-aws-account-id:report-group/report-group-name-1:
 #surefire junit reports
    files:
      - '**/*'
    base-directory: 'surefire/target/surefire-reports'
    discard-paths: false

  sampleReportGroup: #Cucumber reports from json plugin
    files:
```

```
      - 'cucumber-json/target/cucumber-json-report.json'
    file-format: CUCUMBERJSON #Type of the report, defaults to JUNITXML
```

指定测试命令

您可以在 buildspec 文件的 commands 部分中指定运行测试用例的命令。这些命令运行您在 buildspec 文件的 reports 部分中为报告组指定的测试用例。以下是 commands 部分的示例，其中包含用于运行测试文件中的测试的命令：

```
commands:
    - echo Running tests for surefire junit
    - mvn test -f surefire/pom.xml -fn
    - echo
    - echo Running tests for cucumber with json plugin
    - mvn test -Dcucumber.options="--plugin json:target/cucumber-json-report.json" -f
 cucumber-json/pom.xml -fn
```

有关更多信息，请参阅 buildspec 语法。

在中标记报告组 AWS CodeBuild

标签是您或 AWS 分配给 AWS 资源的自定义属性标签。每个 AWS 标签由两部分组成：

- 标签键（例如，CostCenter、Environment、Project 或 Secret）。标签密钥区分大小写。
- 一个称为标签值的可选字段（例如，111122223333、Production 或团队名称）。省略标签值与使用空字符串效果相同。与标签键一样，标签值区分大小写。

这些被统称为键-值对。有关报告组可拥有的标签数量限制以及标签键和值的限制，请参阅标签。

标签可帮助您识别和整理 AWS 资源。许多 AWS 服务都支持标记，因此您可以为来自不同服务的资源分配相同的标签，以表明这些资源是相关的。例如，您可以为 CodeBuild 报告组分配与分配给 Amazon S3 存储桶相同的标签。有关使用标签的更多信息，请参阅标记最佳实践白皮书。

在中 CodeBuild，主要资源是报告组和项目。您可以使用 CodeBuild 控制台、 AWS CLI、 CodeBuild API 或 AWS 软件开发工具包为报告组添加、管理和移除标签。除了使用标签标识、组织和跟踪报告组以外，您还可以在 IAM 策略中使用标签以帮助控制哪些用户可以查看并与报告组交互。有关基于标签的访问策略示例，请参阅使用标签控制对以下内容的访问 AWS CodeBuild 资源。

主题

- 为报告组添加标签
- 查看报告组的标签
- 编辑报告组的标签
- 从报告组中移除标签

为报告组添加标签

向报告组添加标签可以帮助您识别和整理 AWS 资源并管理对资源的访问权限。首先，为报告组添加一个或多个标签（键值对）。请记住，报告组可以拥有的标签数量有限制。键和值字段中可以使用的字符有限制。有关更多信息，请参阅 标签。有了标签后，您可以创建 IAM 策略以根据这些标签管理对报告组的访问。您可以使用 CodeBuild 控制台或 AWS CLI 向报告组添加标签。

> **Important**
>
> 为报告组添加标签会影响对该报告组的访问。为报告组添加标签之前，请务必查看是否存在任何 IAM 策略可能使用标签来控制对资源（如报告组）的访问。有关基于标签的访问策略示例，请参阅使用标签控制对以下内容的访问 AWS CodeBuild 资源。

有关在创建报告组时为其添加标签的更多信息，请参阅创建报告组（控制台）。

主题

- 为报告组添加标签（控制台）
- 为报告组添加标签（AWS CLI）

为报告组添加标签（控制台）

您可以使用 CodeBuild 控制台向 CodeBuild 报告组添加一个或多个标签。

1. 打开 CodeBuild 控制台，网址为https://console.aws.amazon.com/codebuild/。
2. 在报告组中，选择要在其中添加标签的报告组的名称。
3. 在导航窗格中，选择设置。
4. 如果此报告组中尚未添加标签，请选择添加标签。您也可以选择编辑，然后选择添加标签。
5. 在键中，输入标签的名称。您可以在值中添加可选的标签值。
6. （可选）要添加其他标签，请再次选择添加标签。

7. 添加完标签后，选择提交。

为报告组添加标签（AWS CLI）

要在创建报告组时为其添加标签，请参阅创建报告组（CLI）。在 `CreateReportGroup.json` 中，添加您的标签。

要向现有报告组添加标签，请参阅更新报告组（CLI）并在 `UpdateReportGroupInput.json` 中添加标签。

在这些步骤中，我们假设您已安装最新版本的 AWS CLI 或已更新到当前版本。有关更多信息，请参阅安装 AWS Command Line Interface。

查看报告组的标签

标签可以帮助您识别和整理 AWS 资源并管理对资源的访问权限。有关使用标签的更多信息，请参阅标记最佳实践白皮书。有关基于标签的访问策略示例，请参阅Deny or allow actions on report groups based on resource tags。

查看报告组的标签（控制台）

您可以使用 CodeBuild 控制台查看与 CodeBuild 报告组关联的标签。

1. 打开 CodeBuild 控制台，网址为https://console.aws.amazon.com/codebuild/。
2. 在报告组中，选择要在其中查看标签的报告组的名称。
3. 在导航窗格中，选择设置。

查看报告组的标签（AWS CLI）

按照以下步骤使用 AWS CLI 来查看报告组的 AWS 标签。如果尚未添加标签，则返回的标签列表为空。

1. 使用控制台或 AWS CLI 查找您的报告组的 ARN。记下它。

   AWS CLI

   运行如下命令。

   ```
   aws list-report-groups
   ```

此命令返回的 JSON-formatted 信息类似于以下内容：

```
{
    "reportGroups": [
        "arn:aws:codebuild:region:123456789012:report-group/report-group-1",
        "arn:aws:codebuild:region:123456789012:report-group/report-group-2",
        "arn:aws:codebuild:region:123456789012:report-group/report-group-3"
    ]
}
```

报告组 ARN 以其名称结尾，您可以使用该名称来识别您的报告组的 ARN。

Console

1. 打开 CodeBuild 控制台，网址为https://console.aws.amazon.com/codebuild/。
2. 在报告组中，选择带有您要查看的标签的报告组的名称。
3. 在配置中，找到您的报告组的 ARN。

2. 运行如下命令。为 `--report-group-arns` 参数使用您记下的 ARN。

```
aws codebuild batch-get-report-groups --report-group-arns
 arn:aws:codebuild:region:123456789012:report-group/report-group-name
```

如果成功，此命令将返回包含类似以下内容的`tags`部分 JSON-formatted 的信息：

```
{
    ...
    "tags": {
        "Status": "Secret",
        "Project": "TestBuild"
    }
    ...
}
```

编辑报告组的标签

您可以更改与报告组关联的标签值。您也可以更改键的名称，这相当于移除当前的标签并使用新名称和与另一个键相同的值添加一个不同的标签。请记住，键和值字段中可以使用的字符有限制。有关更多信息，请参阅 标签。

> **Important**
>
> 编辑报告组的标签会影响对该报告组的访问。编辑报告组的标签名称（键）或值之前，请务必查看是否存在任何 IAM 策略可能使用标签的键或值来控制对资源（如报告组）的访问。有关基于标签的访问策略示例，请参阅Deny or allow actions on report groups based on resource tags。

编辑报告组的标签（控制台）

您可以使用 CodeBuild 控制台编辑与 CodeBuild 报告组关联的标签。

1. 打开 CodeBuild 控制台，网址为https://console.aws.amazon.com/codebuild/。
2. 在报告组中，选择要在其中编辑标签的报告组的名称。
3. 在导航窗格中，选择设置。
4. 选择编辑。
5. 请执行以下操作之一：
   - 要更改标签，则在键中输入新名称。更改标签的名称相当于删除标签并使用新的键名添加新标签。
   - 要更改标签的值，则输入新值。如果您想将标签值清空，请删除当前的值并将字段保留为空白。
6. 编辑完标签后，选择提交。

编辑报告组的标签（AWS CLI）

要添加、更改或移除报告组中的标签，请参阅更新报告组（CLI）。更新 `UpdateReportGroupInput.json` 中的标签。

从报告组中移除标签

您可以移除与报告组关联的一个或多个标签。移除标签不会从与该标签关联的其他 AWS 资源中删除该标签。

> **Important**
>
> 删除报告组的标签会影响对该报告组的访问。从报告组中移除标签之前，请务必查看是否存在任何 IAM 策略可能使用标签的键或值来控制对资源（如报告组）的访问。有关基于标签的访问策略示例，请参阅使用标签控制对以下内容的访问 AWS CodeBuild 资源。

从报告组中删除标签（控制台）

您可以使用 CodeBuild 控制台删除标签和 CodeBuild报告组之间的关联。

1. 打开 CodeBuild 控制台，网址为https://console.aws.amazon.com/codebuild/。
2. 在报告组中，选择要从其中移除标签的报告组的名称。
3. 在导航窗格中，选择设置。
4. 选择编辑。
5. 找到要移除的标签，然后选择移除标签。
6. 移除标签之后，选择提交。

从报告组中移除标签（AWS CLI）

按照以下步骤使用 AWS CLI 从 CodeBuild 报告组中移除标签。移除标签并不会删除标签，而只是删除它和报告组之间的关联。

> **Note**
>
> 如果删除 CodeBuild 报告组，则会从已删除的报告组中移除所有标签关联。您无需在删除报告组之前移除标签。

要从报告组中删除一个或多个标签，请参阅编辑报告组的标签（AWS CLI）。使用不包含要删除的标签的更新列表来更新 JSON-formatted 数据中的`tags`部分。如果要删除所有标签，请将 `tags` 部分更新为：

```
"tags: []"
```

更新报告组

当您更新报告组时，您可以指定是否将原始测试结果数据导出到 Amazon S3 存储桶中的文件的相关信息。如果您选择导出到 S3 存储桶，可以为报告组指定以下内容：

- 是否将原始测试结果文件压缩为 ZIP 文件。
- 是否对原始测试结果文件进行加密。您可以使用以下选项之一指定加密：
  - AWS 托管式密钥 对于亚马逊 S3 来说。
  - 您创建和配置的客户托管密钥。

有关更多信息，请参阅 数据加密。

如果您使用更新报表组，则还可以更新或添加标签。 AWS CLI 有关更多信息，请参阅 在中标记报告组 AWS CodeBuild。

> **Note**
>
> 项目中指定的 CodeBuild 服务角色用于获得上传到 S3 存储桶的权限。

主题

- 更新报告组（控制台）
- 更新报告组（CLI）

更新报告组（控制台）

按照以下过程使用 AWS 管理控制台更新报告组。

更新报告组的步骤

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择报告组。
3. 选择要更新的报告组。
4. 选择编辑。
5. 选择或清除备份到 Amazon S3。如果选择此选项，请指定您的导出设置：
   a. 对于 S3 存储桶名称，请输入 S3 存储桶的名称。

b. 对于路径前缀，请输入要上传测试结果的 S3 存储桶中的路径。

c. 选择将测试结果数据压缩为 zip 文件以便压缩原始测试结果数据文件。

d. 展开其他配置以显示加密选项。选择下列选项之一：

- 用 AWS 托管式密钥 于 Amazon S3 的默认 AWS 托管密钥。有关更多信息，请参阅《AWS Key Management Service 用户指南》中的客户托管式 CMK。这是默认加密选项。
- 选择自定义密钥将使用您创建和配置的客户托管密钥。对于AWS KMS 加密密钥，请输入加密密钥的 ARN。其格式为 `arn:aws:kms:<region-id>: <aws-account-id>:key/<key-id>` 。有关更多信息，请参阅《AWS Key Management Service 用户指南》中的创建 KMS 密钥。
- 禁用构件加密将禁用加密。如果要共享测试结果或将其发布到静态网站，则可以选择此选项。（动态网站可以运行代码来解密测试结果。）

更新报告组（CLI）

按照以下过程使用 AWS CLI更新报告组。

更新报告组的步骤

1. 创建一个名为 `UpdateReportGroupInput.json`的文件。
2. 将以下内容复制到 `UpdateReportGroupInput.json`。

```
{
    "arn": "",
    "exportConfig": {
        "exportConfigType": "S3",
        "s3Destination": {
            "bucket": "bucket-name",
            "path": "path",
            "packaging": "NONE | ZIP",
            "encryptionDisabled": "false",
            "encryptionKey": "your-key"
         }
     },
     "tags": [
        {
            "key": "tag-key",
            "value": "tag-value"
        }
```

```
    ]
}
```

3. 在 arn 行中输入报告组的 ARN，例如 "arn":"arn:aws:codebuild:*region*:*123456789012*:report-group/*report-group-1*")。
4. 使用要应用到报告组的更新来更新 UpdateReportGroupInput.json。
   - 如果要更新报告组以将原始测试结果文件导出到 S3 存储桶，请更新 exportConfig 部分。将 bucket-name 替换为 S3 存储桶名称，并将 path 替换为 S3 存储桶中您要将文件导出到的路径。如果要压缩导出的文件，对于 packaging，请指定 ZIP。否则，请指定 NONE。使用 encryptionDisabled 指定是否要加密导出的文件。如果要加密导出的文件，请输入客户托管密钥。
   - 如果要更新报告组，以使其不会将原始测试结果文件导出到 S3 存储桶，请使用以下 JSON 更新 exportConfig 部分：

     ```
     {
       "exportConfig": {
           "exportConfigType": "NO_EXPORT"
       }
     }
     ```

   - 如果要更新报告组的标签，请更新 tags 部分。您可以更改、添加或删除标签。如果要删除所有标签，请使用以下 JSON 来更新：

     ```
     "tags": []
     ```

5. 运行以下命令：

```
aws codebuild update-report-group \
--cli-input-json file://UpdateReportGroupInput.json
```

测试框架

本节中的主题演示如何在中 AWS CodeBuild 为各种测试框架设置测试报告。

主题

- 使用 Jasmine 设置测试报告

- 使用 Jest 设置测试报告
- 使用 pytest 设置测试报告
- 使用 RSpec 设置测试报告

使用 Jasmine 设置测试报告

以下过程演示了如何 AWS CodeBuild 使用 JasmineBDD 测试框架设置测试报告。

该过程需要以下先决条件：

- 你有一个现有的 CodeBuild 项目。
- 你的 Node.js 项目是一个设置为使用 Jasmine 测试框架的项目。

将 `jasmine-reporters` 程序包添加到项目 `package.json` 文件的 `devDependencies` 部分。该软件包包含一系列可与 Jasmine 一起使用的 JavaScript 报告器类。

```
npm install --save-dev jasmine-reporters
```

如果它尚未存在，请将 `test` 脚本添加到项目的 `package.json` 文件中。`test` 脚本确保在运行 npm test 时调用 Jasmine。

```
{
  "scripts": {
    "test": "npx jasmine"
  }
}
```

CodeBuild 支持以下 Jasmine 测试报告器：

JUnitXmlReporter

用于以 `JunitXml` 格式生成报告。

NUnitXmlReporter

用于以 `NunitXml` 格式生成报告。

默认情况下，使用 Jasmine 的 Node.js 项目将有一个`spec`子目录，其中包含 Jasmine 配置和测试脚本。

要将 Jasmine 配置为以 JunitXML 格式生成报告，请通过将以下代码添加到测试中来实例化 JUnitXmlReporter 报告程序。

```
var reporters = require('jasmine-reporters');

var junitReporter = new reporters.JUnitXmlReporter({
  savePath: <test report directory>,
  filePrefix: <report filename>,
  consolidateAll: true
});

jasmine.getEnv().addReporter(junitReporter);
```

要将 Jasmine 配置为以 NunitXML 格式生成报告，请通过将以下代码添加到测试中来实例化 NUnitXmlReporter 报告程序。

```
var reporters = require('jasmine-reporters');

var nunitReporter = new reporters.NUnitXmlReporter({
  savePath: <test report directory>,
  filePrefix: <report filename>,
  consolidateAll: true
});

jasmine.getEnv().addReporter(nunitReporter)
```

测试报告将导出到由 *<test report directory>* /指定的文件中*<report filename>*。

在您的buildspec.yml文件中， add/update 有以下部分。

```
version: 0.2

phases:
  pre_build:
    commands:
      - npm install
  build:
    commands:
      - npm build
      - npm test

reports:
```

```
  jasmine_reports:
    files:
      - <report filename>
    file-format: JUNITXML
    base-directory: <test report directory>
```

如果您使用NunitXml报告格式，请将file-format值更改为以下值。

```
    file-format: NUNITXML
```

使用 Jest 设置测试报告

以下过程演示如何 AWS CodeBuild 使用 Jest 测试框架设置测试报告。

该过程需要以下先决条件：

- 你有一个现有 CodeBuild 项目。
- 您的项目是为使用 Jest 测试框架而设置的 Node.js 项目。

将jest-junit软件包添加到项目package.json文件的devDependencies部分。 CodeBuild 使用此包生成JunitXml格式的报告。

```
npm install --save-dev jest-junit
```

如果它尚未存在，请将 test 脚本添加到项目的 package.json 文件中。test 脚本确保在运行 npm test 时调用 Jest。

```
{
  "scripts": {
    "test": "jest"
  }
}
```

通过将以下内容添加到 Jest 配置文件中，将 Jest 配置为使用 JunitXml 报告程序。如果您的项目没有 Jest 配置文件，请在项目的根目录中创建一个名为 jest.config.js 的文件，然后添加以下内容。测试报告将导出到 *<test report directory>* /指定的文件中*<report filename>*。

```
module.exports = {
  reporters: [
    'default',
```

```
    [ 'jest-junit', {
      outputDirectory: <test report directory>,
      outputName: <report filename>,
    } ]
  ]
};
```

在您的buildspec.yml文件中， add/update 有以下几节。

```
version: 0.2

phases:
  pre_build:
    commands:
      - npm install
  build:
    commands:
      - npm build
      - npm test

reports:
  jest_reports:
    files:
      - <report filename>
    file-format: JUNITXML
    base-directory: <test report directory>
```

使用 pytest 设置测试报告

以下过程演示如何 AWS CodeBuild 使用 pytest 测试框架设置测试报告。

该过程需要以下先决条件：

- 你有一个现有 CodeBuild 项目。
- 您的项目是一个 Python 项目，此项目设置为使用 pytest 测试框架。

将以下条目添加到 buildspec.yml 文件的 build 或 post_build 阶段。此代码会自动发现当前目录中的测试，并将测试报告导出到*<test report directory>*/*<report filename>*指定的文件中。报告使用 JunitXml 格式。

```
      - python -m pytest --junitxml=<test report directory>/<report filename>
```

在您的buildspec.yml文件中， add/update 有以下几节。

```
version: 0.2

phases:
  install:
    runtime-versions:
      python: 3.7
    commands:
      - pip3 install pytest
  build:
    commands:
      - python -m pytest --junitxml=<test report directory>/<report filename>

reports:
  pytest_reports:
    files:
      - <report filename>
    base-directory: <test report directory>
    file-format: JUNITXML
```

使用 RSpec 设置测试报告

以下过程演示如何 AWS CodeBuild 使用 RSpec 测试框架设置测试报告。

该过程需要以下先决条件：

- 你有一个现有 CodeBuild 项目。
- 您的项目是一个 Ruby 项目，此项目设置为使用 RSpec 测试框架。

Add/update 您的buildspec.yml文件中有以下内容。此代码在*<test source directory>*目录中运行测试，并将测试报告导出到 *<test report directory>* /指定的文件中*<report filename>*。报告使用 JunitXml 格式。

```
version: 0.2

phases:
  install:
    runtime-versions:
      ruby: 2.6
  pre_build:
```

```
    commands:
      - gem install rspec
      - gem install rspec_junit_formatter
  build:
    commands:
      - rspec <test source directory>/* --format RspecJunitFormatter --out <test report
 directory>/<report filename>
reports:
    rspec_reports:
        files:
            - <report filename>
        base-directory: <test report directory>
        file-format: JUNITXML
```

查看测试报告

您可以查看有关测试报告的详细信息，例如，有关其测试用例、通过和失败次数及其运行时间的信息。您可以查看按编译运行、报告组或您的 AWS 帐户分组的测试报告。在控制台中选择测试报告以查看其详细信息和测试用例的结果。

您可以查看未过期的测试报告。测试报告在创建后 30 天过期。您无法在 CodeBuild 中查看过期报告。

主题

- 查看构建的测试报告
- 查看报告组的测试报告
- 查看您的测试报告 AWS 账户

查看构建的测试报告

查看构建的测试报告

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 找到要查看的构建。如果您知道运行构建（创建测试报告）的项目，请执行以下操作：

   1. 在导航窗格中，选择构建项目，然后选择项目，该项目具有运行要查看的测试报告的构建。
   2. 选择构建历史记录，然后选择构建，该构建运行创建的要查看的报告。

   您还可以在您的 AWS 账户的构建历史记录中查找构建：

1. 在导航窗格中，选择构建历史记录，然后选择构建，该构建已创建要查看的报告。

3. 在构建页面中，选择报告，然后选择测试报告以查看其详细信息。

查看报告组的测试报告

查看报告组中的测试报告

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择报告组。
3. 选择包含要查看的测试报告的报告组。
4. 选择测试报告以查看其详细信息。

查看您的测试报告 AWS 账户

要查看您的测试报告 AWS 账户

1. 在处打开 AWS CodeBuild 控制台https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择报告历史记录。
3. 选择测试报告以查看其详细信息。

测试报告权限

本主题介绍与测试报告相关权限有关的重要信息。

主题

- 测试报告的 IAM 角色
- 测试报告操作的权限
- 测试报告权限示例

测试报告的 IAM 角色

要运行测试报告，并更新项目以包含测试报告，您的 IAM 角色需要以下权限。这些权限包含在预定义的 AWS 托管策略中。如果要将测试报告添加到现有构建项目，则必须自行添加这些权限。

- CreateReportGroup
- CreateReport
- UpdateReport
- BatchPutTestCases

要运行代码覆盖率报告，您的 IAM 角色还必须包含 BatchPutCodeCoverages 权限。

> **Note**
>
> BatchPutTestCases、CreateReport、UpdateReport 和 BatchPutCodeCoverages 不是公共权限。您无法为这些权限调用相应的 AWS CLI 命令或 SDK 方法。

要确保您拥有这些权限，您可以将以下策略附加到您的 IAM 角色：

```
{
    "Effect": "Allow",
    "Resource": [
        "*"
    ],
    "Action": [
        "codebuild:CreateReportGroup",
        "codebuild:CreateReport",
        "codebuild:UpdateReport",
        "codebuild:BatchPutTestCases",
        "codebuild:BatchPutCodeCoverages"
    ]
}
```

我们建议您将此策略仅限定用于您必须使用的报告组。以下内容将权限仅限定用于策略中具有两个 ARN 的报告组：

```
{
    "Effect": "Allow",
    "Resource": [
        "arn:aws:codebuild:your-region:your-aws-account-id:report-group/report-group-name-1",
        "arn:aws:codebuild:your-region:your-aws-account-id:report-group/report-group-name-2"
```

```
    ],
    "Action": [
        "codebuild:CreateReportGroup",
        "codebuild:CreateReport",
        "codebuild:UpdateReport",
        "codebuild:BatchPutTestCases",
        "codebuild:BatchPutCodeCoverages"
    ]
}
```

以下内容将权限仅限定用于通过运行名为 `my-project` 的项目构建而创建的报告组：

```
{
    "Effect": "Allow",
    "Resource": [
        "arn:aws:codebuild:your-region:your-aws-account-id:report-group/my-project-*"
    ],
    "Action": [
        "codebuild:CreateReportGroup",
        "codebuild:CreateReport",
        "codebuild:UpdateReport",
        "codebuild:BatchPutTestCases",
        "codebuild:BatchPutCodeCoverages"
    ]
}
```

> **Note**
>
> 项目中指定的 CodeBuild 服务角色用于获得上传到 S3 存储桶的权限。

测试报告操作的权限

您可以为以下测试报告 CodeBuild API 操作指定权限：

- `BatchGetReportGroups`
- `BatchGetReports`
- `CreateReportGroup`
- `DeleteReportGroup`
- `DeleteReport`

- DescribeTestCases
- ListReportGroups
- ListReports
- ListReportsForReportGroup
- UpdateReportGroup

有关更多信息，请参阅 AWS CodeBuild 权限参考。

测试报告权限示例

有关测试报告相关示例策略的信息，请参阅以下内容：

- 允许用户更改报告组
- 允许用户创建报告组
- 允许用户删除报告
- 允许用户删除报告组
- 允许用户获取有关报告组的信息
- 允许用户获取有关报告的信息
- 允许用户获取报告组列表
- 允许用户获取报告列表
- 允许用户获取报告组的报告列表
- 允许用户获取报告的测试用例的列表

测试报告状态

测试报告可能处于以下状态之一：

- GENERATING：测试用例仍在运行中。
- DELETING：正在删除测试报告。删除测试报告时，还将删除其测试用例。不会删除导出到 S3 存储桶的原始测试结果数据文件。
- INCOMPLETE：测试报告未完成。由于以下原因之一，可能会返回此状态：
  - 指定此报告的测试用例的报告组配置有问题。例如，buildspec 文件中报告组下的测试用例路径可能不正确。

  - 运行构建的 IAM 用户没有运行测试的权限。有关更多信息，请参阅 测试报告权限。
  - 由于发生与测试无关的错误，构建未完成。
- SUCCEEDED：所有测试用例都成功。
- FAILED：部分测试用例未成功。

每个测试用例都会返回一个状态。测试用例可能处于以下状态之一：

- SUCCEEDED：测试用例通过。
- FAILED：测试用例失败。
- ERROR：测试用例导致意外错误。
- SKIPPED：测试用例未运行。
- UNKNOWN：测试用例返回 SUCCEEDED、FAILED、ERROR 或 SKIPPED 以外的状态。

测试报告最多可包含 500 个测试用例结果。如果运行的测试用例超过 500 个，则使用状态对测试进行 CodeBuild 优先级排序，FAILED并截断测试用例结果。

AWS CodeBuild 与亚马逊 Virtual Private Cloud 配合使用

通常， AWS CodeBuild 无法访问 VPC 中的资源。要启用访问权限，您必须在 CodeBuild 项目 VPC-specific 配置中提供其他配置信息。这包括 VPC ID、VPC 子网 ID 和 VPC 安全组 ID。 VPC-enabled 然后，构建可以访问您的 VPC 内的资源。有关在 Amazon VPC 中设置 VPC 的更多信息，请参阅《Amazon VPC 用户指南》。

主题

- 使用案例
- VPC 的最佳实操
- VPC 的限制
- 在您的 CodeBuild 项目中允许 Amazon VPC 访问权限
- 排查 VPC 设置的问题
- 使用 VPC 端点
- AWS CodeBuild 与托管代理服务器一起使用
- 使用 AWS CodeBuild 使用代理服务器
- CloudFormation VPC 模板

使用案例

通过 AWS CodeBuild 构建实现的 VPC 连接可以：

- 针对在私有子网上隔离的 Amazon RDS 数据库中的数据，从您的构建中运行集成测试。
- 直接从测试中查询 Amazon ElastiCache 集群中的数据。
- 与托管于 Amazon EC2、Amazon ECS 或使用内部 Elastic Load Balancing 的服务上的内部 Web 服务交互。
- 从自托管的内部工件存储库中检索依赖项，例如适用于 Python 的 PyPi、适用于 Java 的 Maven 和适用于 npm 的 npm。 Node.js
- 访问配置为仅允许通过 Amazon VPC 端点访问的 S3 存储桶中的对象。
- 利用与您的子网关联的 NAT 网关或 NAT 实例的弹性 IP 地址，来查询需要固定 IP 地址的外部 Web 服务。

您的构建可以访问您的 VPC 中托管的任何资源。

VPC 的最佳实操

设置要使用的 VPC 时，请使用此清单 CodeBuild。

- 设置具有公有和私有子网以及一个 NAT 网关的 VPC。NAT 网关必须位于公有子网中。有关更多信息，请参阅《Amazon VPC 用户指南》中的具有公有和私有子网 (NAT) 的 VPC。

  > **Important**
  >
  > 您需要一个 NAT 网关或 NAT 实例才能 CodeBuild 与您的 VPC 配合使用，这样它 CodeBuild 才能访问公有终端节点（例如，在运行构建时运行 CLI 命令）。您不能使用互联网网关代替 NAT 网关或 NAT 实例，因为 CodeBuild 不支持为其创建的网络接口分配弹性 IP 地址，而且 Amazon EC2 不支持为在 Amazon EC2 实例启动之外创建的任何网络接口自动分配公有 IP 地址。

- 将多个可用区包含在您的 VPC 中。
- 确保您的安全组不允许您的内部版本入站（入口）流量。 CodeBuild 对出站流量没有具体要求，但您必须允许访问构建所需的任何互联网资源，例如 GitHub 或 Amazon S3。

  有关更多信息，请参阅《Amazon VPC 用户指南》中的安全组规则。
- 为您的构建设置单独的子网。
- 当您将 CodeBuild 项目设置为访问您的 VPC 时，请仅选择私有子网。

有关在 Amazon VPC 中设置 VPC 的更多信息，请参阅《Amazon VPC 用户指南》。

有关使用配置 VPC CloudFormation 以使用 VP CodeBuild C 功能的更多信息，请参阅CloudFormation VPC 模板。

VPC 的限制

- 共享 VPC CodeBuild 不支持来自的 VPC 连接。

在您的 CodeBuild 项目中允许 Amazon VPC 访问权限

在您的 VPC 配置中包含以下设置：

- 对于 VPC ID，请选择 CodeBuild 使用的 VPC ID。

- 对于子网，请选择一个带有 NAT 转换的私有子网，该子网包含或具有指向所 CodeBuild用资源的路由。
- 对于 Security Groups (安全组)，选择 CodeBuild 用来支持对 VPC 中资源的访问的安全组。

要使用控制台创建构建项目，请参阅创建构建项目（控制台）。创建或更改 CodeBuild 项目时，在 VPC 中，选择您的 VPC ID、子网和安全组。

要使用创建生成项目，请参阅创建构建项目 (AWS CLI)。 AWS CLI 如果您使用的是 w AWS CLI it CodeBuild h，则用于代表 IAM 用户与服务交互的服务角色必须附加策略。 CodeBuild 有关信息，请参阅允许 CodeBuild 访问 AWS 创建 VPC 网络接口所需的服务。

该*vpcConfig*对象应包括您的*vpcIdsecurityGroupIds*、和*subnets*。

- *vpcId*：必需。 CodeBuild 使用的 VPC ID。运行此命令获取您的区域中的所有 Amazon VPC ID 的列表：

```
aws ec2 describe-vpcs
```

- *subnets*：必需。包含使用的资源的子网 ID CodeBuild。要获取这些 ID，请运行此命令：

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=<vpc-id>" --region us-east-1
```

> **Note**
>
> 将 us-east-1 替换为您的区域。

- *securityGroupIds*：必需。用于 CodeBuild 允许访问 VPC 中的资源的安全组 ID。要获取这些 ID，请运行此命令：

```
aws ec2 describe-security-groups --filters "Name=vpc-id,Values=<vpc-id>" --region us-east-1
```

> **Note**
>
> 将 us-east-1 替换为您的区域。

排查 VPC 设置的问题

使用错误消息中显示的信息可帮助您确定、诊断和解决问题。

下面是一些帮助您排查常见的 CodeBuild VPC 错误的指导信息：`Build does not have internet connectivity. Please check subnet network configuration`

1. 确保您的互联网网关已连接到 VPC。
2. 确保您的公有子网的路由表指向互联网网关。
3. 确保您的网络 ACL 允许流量流动。
4. 确保您的安全组允许流量流动。
5. 排查 NAT 网关的问题。
6. 确保私有子网的路由表指向 NAT 网关。
7. 确保用于代表 IAM 用户与服务交互的服务角色具有此策略中的权限。 CodeBuild 有关更多信息，请参阅 CodeBuild 允许与其他人互动 AWS 务。

   CodeBuild 如果缺少权限，您可能会收到一条错误消息，上面写着`Unexpected EC2 error: UnauthorizedOperation`。如果 CodeBuild 没有使用 VPC 所需的 Amazon EC2 权限，则可能会发生此错误。

使用 VPC 端点

您可以通过配置为使用接口 VPC 终端节点 AWS CodeBuild 来提高构建的安全性。接口终端节点由 PrivateLink一种可用于私有访问 Amazon EC2 的技术和 CodeBuild 私有 IP 地址提供支持。PrivateLink 将您的托管实例与 Amazon EC2 之间的所有网络流量限制到亚马逊网络。 CodeBuild（托管实例无法访问 Internet。）而且，您无需 Internet 网关、NAT 设备或虚拟专用网关。不要求您配置 PrivateLink，但推荐进行配置。有关 PrivateLink 和 VPC 终端节点的更多信息，请参阅什么是 AWS PrivateLink？ 。

在您创建 VPC 端点前

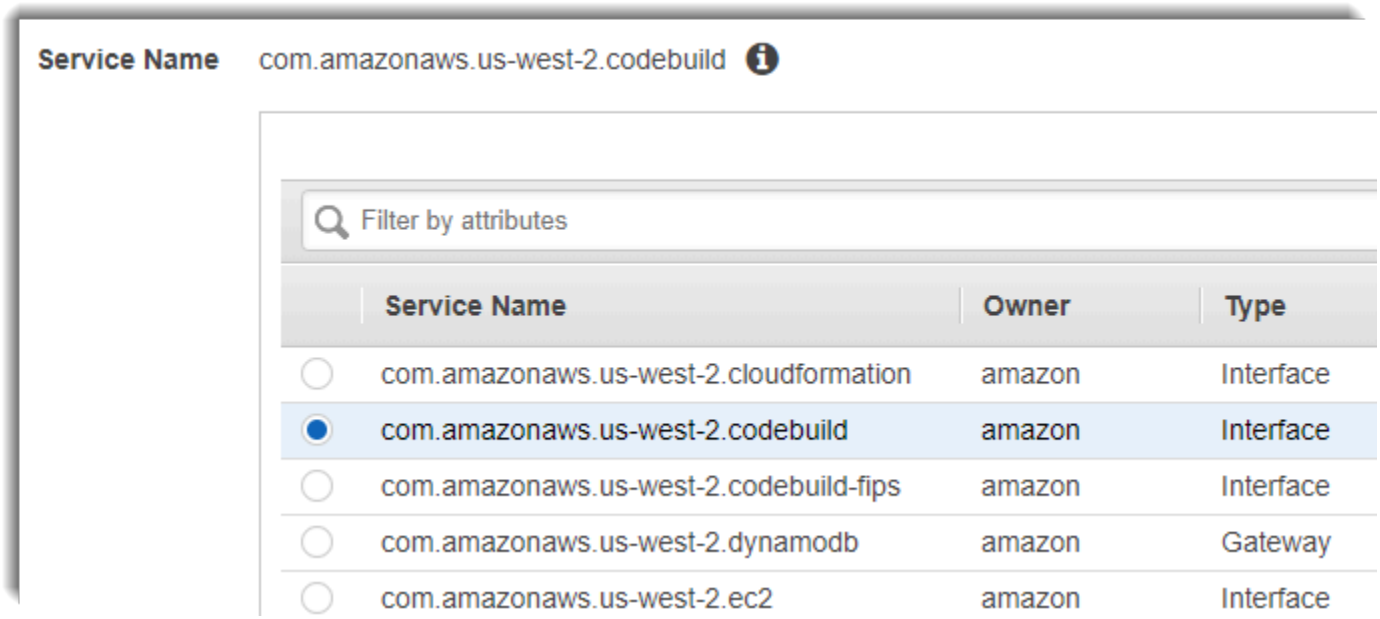
在为配置 VPC 终端节点之前 AWS CodeBuild，请注意以下限制和限制。

> **Note**
>
> 如果您想与不支持 Amazon VPC PrivateLink 连接 CodeBuild 的 AWS 服务一起使用，请使用 NAT 网关。

- VPC 终端节点仅支持通过 Amazon Route 53 的 Amazon-provided DNS。如果您希望使用自己的 DNS，可以使用条件 DNS 转发。有关更多信息，请参阅《Amazon VPC 用户指南》中的 DHCP 选项集。
- VPC 端点当前不支持跨区域请求。请确保在与存储构建输入和输出的任何 S3 存储桶相同的 AWS 区域中创建终端节点。您可以使用 Amazon S3 控制台或 get-bucket-location 命令来查找存储桶的位置。使用 A Region-specific mazon S3 终端节点访问您的存储桶（例如，`<bucket-name>.s3-us-west-2.amazonaws.com`）。有关 Amazon S3 Region-specific 终端节点的更多信息，请参阅中的亚马逊简单存储服务Amazon Web Services 一般参考。如果您使用向 Amazon S3 发出请求，请将默认区域设置为创建存储桶的相同区域，或者在请求中使用`--region`参数。 AWS CLI

为创建 VPC 终端节点 CodeBuild

按照创建接口端点中的说明操作，创建端点 `com.amazonaws.region.codebuild`。这是的 VPC 终端节点 AWS CodeBuild。



*region*表示所 CodeBuild支持的 AWS 区域（例如us-east-2美国东部（俄亥俄州）地区的区域标识符。有关支持的 AWS 区域列表，请参阅《 AWS 一般参考》CodeBuild中的。终端节点已预先填充您在登录时指定的区域。 AWS如果更改您的区域，VPC 端点会相应地更新。

为创建 VPC 终端节点策略 CodeBuild

您可以为 Amazon VPC 终端节点创建策略，您可以在其中指定： AWS CodeBuild

- 可执行操作的主体。
- 可执行的操作。
- 可用于执行操作的资源。

以下示例策略指定所有委托人只能启动和查看 project-name 项目的构建。

```
{
    "Statement": [
        {
            "Action": [
                "codebuild:ListBuildsForProject",
                "codebuild:StartBuild",
                "codebuild:BatchGetBuilds"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:codebuild:region-ID:account-ID:project/project-name",
            "Principal": "*"
        }
    ]
}
```

有关更多信息，请参阅《Amazon VPC 用户指南》中的使用 VPC 端点控制对服务的访问。

AWS CodeBuild 与托管代理服务器一起使用

要在托管代理服务器中运行 AWS CodeBuild 预留容量队列，必须使用代理规则将代理服务器配置为允许或拒绝进出外部站点的流量。请注意，VPC、Windows 或 macOS 不支持在托管代理服务器中运行预留容量实例集。

> **Important**
>
> 根据代理配置在实例集中的存在时间，会产生额外费用。有关更多信息，请参阅 https://aws.amazon.com/codebuild/pricing/。

主题

- 为预留容量实例集配置托管代理配置
- 运行 CodeBuild 预留容量队伍

为预留容量实例集配置托管代理配置

要为预留容量实例集配置托管代理服务器，必须在控制台中或使用 AWS CLI创建实例集时启用此特征。您需要定义几个属性：

定义代理配置 - 可选

对预留容量实例应用网络访问控制的代理配置。

默认行为

定义传出流量的行为。

允许

默认情况下，允许流向所有目标的传出流量。

拒绝

默认情况下，拒绝流向所有目标的传出流量。

代理规则

指定要为其限制网络访问控制的目标域。

要在控制台中定义代理配置，请参阅创建预留容量实例集以获取说明。要使用定义代理配置 AWS CLI，您可以通过修改以下 JSON 语法并保存结果来实现：

```
"proxyConfiguration": {
    "defaultBehavior": "ALLOW_ALL" | "DENY_ALL",
    "orderedProxyRules": [
```

```
        {
            "type": "DOMAIN" | "IP",
            "effect": "ALLOW" | "DENY",
            "entities": [
                "destination"
            ]
        }
    ]
}
```

JSON 文件可能看起来类似于以下内容：

```
"proxyConfiguration": {
    "defaultBehavior": "DENY_ALL",
    "orderedProxyRules": [
        {
            "type": "DOMAIN",
            "effect": "ALLOW",
            "entities": [
                "github.com"
            ]
        }
    ]
}
```

运行 CodeBuild 预留容量队伍

使用托管代理服务器运行 AWS CodeBuild 预留容量队列时， CodeBuild 将自动使用托管代理地址设置预留容量队列HTTP_PROXY和HTTPS_PROXY环境变量。如果您的相关软件有自己的配置且不遵循环境变量设置，则可以参考这些值，并在构建命令中更新软件配置，以便正确地引导构建流量通过托管代理服务器。有关更多信息，请参阅在中创建构建项目 AWS CodeBuild和在中更改构建项目设置 AWS CodeBuild。

使用 AWS CodeBuild 使用代理服务器

您可以与代理服务器 AWS CodeBuild 一起使用来管理进出互联网的 HTTP 和 HTTPS 流量。要 CodeBuild 使用代理服务器运行，您需要在公有子网和 CodeBuild VPC 的私有子网中安装代理服务器。

在代理服务器 CodeBuild 中运行有两种主要用例：

- 它不再需要您的 VPC 中的 NAT 网关或 NAT 实例。
- 它允许您指定代理服务器中的实例可以访问的 URL 以及代理服务器拒绝访问的 URL。

您可以 CodeBuild 与两种类型的代理服务器一起使用。对于两者，代理服务器在公有子网中 CodeBuild 运行并在私有子网中运行。

- 显式代理：如果使用显式代理服务器，则必须在 CodeBuild 项目级别配置NO_PROXYHTTP_PROXY、和HTTPS_PROXY环境变量。有关更多信息，请参阅在中更改构建项目设置 AWS CodeBuild和在中创建构建项目 AWS CodeBuild。
- 透明代理：如果使用透明代理服务器，则不需要特殊配置。

主题

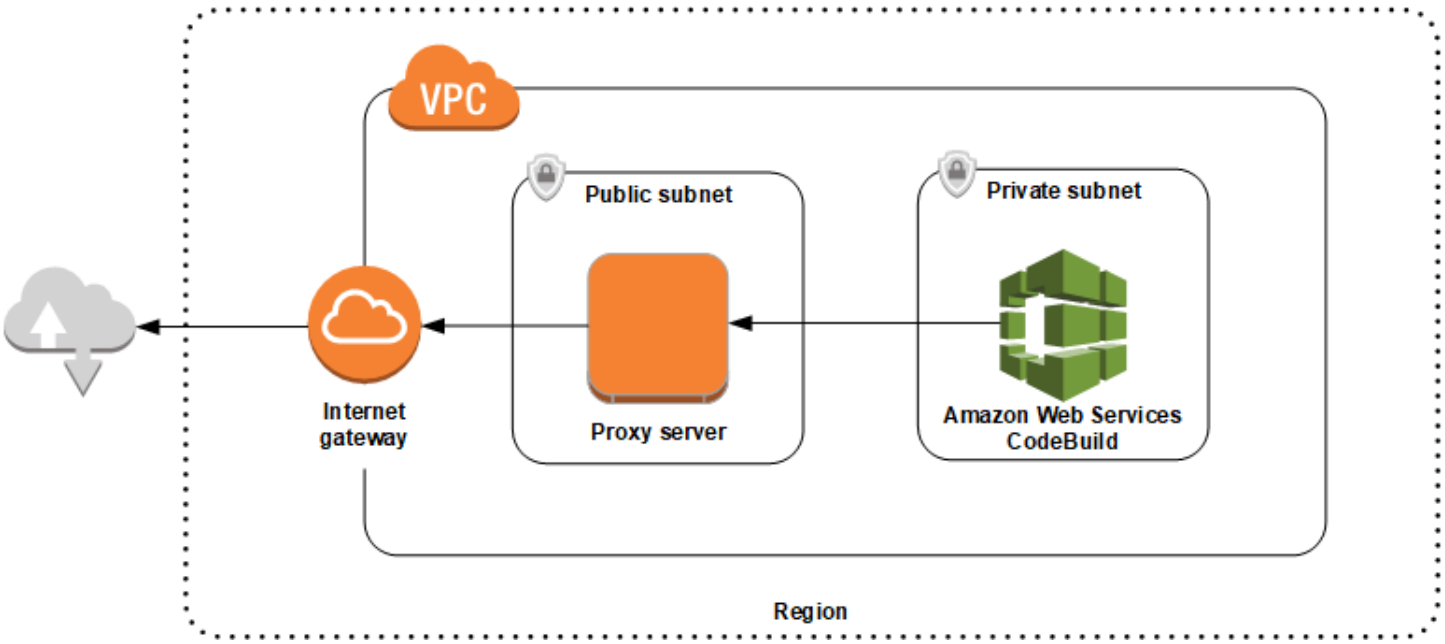
- 设置在代理服务器 CodeBuild 中运行所需的组件
- CodeBuild 在显式代理服务器中运行
- CodeBuild 在透明的代理服务器上运行
- 在代理服务器中运行程序包管理器和其他工具

设置在代理服务器 CodeBuild 中运行所需的组件

你需要这些组件才能 AWS CodeBuild 在透明或显式的代理服务器中运行：

- VPC。
- 代理服务器的 VPC 中的一个公有子网。
- CodeBuild 的 VPC 中的一个私有子网。
- 允许在 VPC 和互联网之间进行通信的互联网网关。

下图显示了组件的交互方式。



设置 VPC、子网和网络网关

要在透明或显式代理服务器 AWS CodeBuild 中运行，需要执行以下步骤。

1. 创建 VPC。有关信息，请参阅《Amazon VPC 用户指南》中的创建 VPC。
2. 在您的 VPC 中创建两个子网。一个是名为 `Public Subnet` 的公有子网，代理服务器将在其中运行。另一个是名为的私有子网，`Private Subnet`在其中 CodeBuild 运行。

   有关信息，请参阅在 VPC 中创建子网。
3. 创建 Internet 网关，并将其连接到您的 VPC。有关更多信息，请参阅创建并附加 Internet 网关。
4. 向默认路由表中添加一条规则，用于路由来自 VPC 的传出流量 (0.0.0.0/0) 到互联网网关。有关信息，请参阅在路由表中添加和删除路由。
5. 向您的 VPC 的默认安全组添加一条规则，允许来自您的 VPC (0.0.0) 的 SSH 流量 (TCP 22)。0/0 ）。
6. 请按照《Amazon EC2 用户指南》中的使用启动实例向导启动实例来启动 Amazon Linux 实例。当您运行该向导时，请选择以下选项：
   - 在选择实例类型中，选择一个 Amazon Linux 亚马逊机器映像（ AMI ）。
   - 在子网中，选择您在本主题的前面步骤中创建的公有子网。如果您使用了建议的名称，则该名称是公有子网。
   - 在Auto-assign 公有 IP 中，选择启用。
   - 在配置安全组页面上，对于分配安全组，选择选择现有安全组。接下来，选择默认安全组。

- 选择启动后，选择现有密钥对或创建密钥对。

选择所有其他选项的默认设置。

7. 在您的 EC2 实例运行后，禁用 source/destination 检查。有关信息，请参阅 Amazon VPC 用户指南中的禁用 Source/Destination 检查。
8. 在 VPC 中创建路由表。向路由表中添加一条规则，该规则将发往 Internet 的流量路由到您的代理服务器。将此路由表与私有子网关联。这是必需的，这样，您的私有子网中 CodeBuild 运行的实例的出站请求始终通过代理服务器路由。

安装和配置代理服务器

有许多可供选择的代理服务器。此处使用开源代理服务器 Squid 来演示如何在代理服务器中 AWS CodeBuild 运行。您可以将相同的概念应用于其他代理服务器。

要安装 Squid，请通过运行以下命令使用 yum 存储库：

```
sudo yum update -y
sudo yum install -y squid
```

安装 Squid 后，请按照本主题后面的说明操作来编辑其 `squid.conf` 文件。

为 HTTPS 流量配置 Squid

对于 HTTPS，HTTP 流量封装在一个传输层安全性协议（TLS）连接中。Squid 使用一项名SslPeekAndSplice为的功能从包含所请求的互联网主机的 TLS 启动中检索服务器名称指示 (SNI)。这是必需的，因此 Squid 不需要解密 HTTPS 流量。要启用 SslPeekAndSplice，Squid 需要证书。使用 OpenSSL 创建此证书：

```
sudo mkdir /etc/squid/ssl
cd /etc/squid/ssl
sudo openssl genrsa -out squid.key 2048
sudo openssl req -new -key squid.key -out squid.csr -subj "/C=XX/ST=XX/L=squid/O=squid/
CN=squid"
sudo openssl x509 -req -days 3650 -in squid.csr -signkey squid.key -out squid.crt
sudo cat squid.key squid.crt | sudo tee squid.pem
```

> **Note**
>
> 对于 HTTP，Squid 不需要配置。它可以从所有 HTTP/1.1 请求消息中检索主机标头字段，该字段指定了所请求的互联网主机。

CodeBuild 在显式代理服务器中运行

要 AWS CodeBuild 在显式代理服务器中运行，必须将代理服务器配置为允许或拒绝进出外部站点的流量，然后配置HTTP_PROXY和HTTPS_PROXY环境变量。

主题

- 将 Squid 配置为显式代理服务器
- 创建 CodeBuild 项目
- 显式代理服务器示例 s quid.conf 文件

将 Squid 配置为显式代理服务器

要将 Squid 代理服务器配置为显式，您必须对其 `/etc/squid/squid.conf` 文件进行以下修改：

- 删除以下默认访问控制列表（ACL）规则。

```
acl localnet src 10.0.0.0/8
acl localnet src 172.16.0.0/12
acl localnet src 192.168.0.0/16
acl localnet src fc00::/7
acl localnet src fe80::/10
```

在您删除的默认 ACL 规则的位置添加以下内容。第一行允许来自您的 VPC 的请求。接下来的两行授予您的代理服务器访问 AWS CodeBuild可能使用的目标 URL 的权限。编辑最后一行中的正则表达式以指定 S3 存储桶或 AWS 区域中的 CodeCommit 存储库。例如：

- 如果您的源是 Amazon S3，请使用命令 acl download_src dstdom_regex .*s3\.us-west-1\.amazonaws\.com 来授权访问 `us-west-1` 区域中的 S3 存储桶。
- 如果您的来源是 AWS CodeCommit，`git-codecommit.<your-region>.amazonaws.com`请使用将 AWS 区域添加到允许列表。

```
acl localnet src 10.1.0.0/16 #Only allow requests from within the VPC
```

```
acl allowed_sites dstdomain .github.com #Allows to download source from GitHub
acl allowed_sites dstdomain .bitbucket.com #Allows to download source from Bitbucket
acl download_src dstdom_regex .*\.amazonaws\.com #Allows to download source from
 Amazon S3 or CodeCommit
```

- 将 `http_access allow localnet` 替换为以下项：

```
http_access allow localnet allowed_sites
http_access allow localnet download_src
```

- 如果您希望构建上传日志和构件，请执行以下任一操作：

1. 在 `http_access deny all` 语句之前，插入以下语句。它们允许 CodeBuild 访问 CloudWatch 和亚马逊 S3。需要访问 CloudWatch 权限 CodeBuild 才能创建 CloudWatch 日志。上传构件和 Amazon S3 缓存需要访问 Amazon S3。

   - ```
     https_port 3130 cert=/etc/squid/ssl/squid.pem ssl-bump intercept
     acl SSL_port port 443
     http_access allow SSL_port
     acl allowed_https_sites ssl::server_name .amazonaws.com
     acl step1 at_step SslBump1
     acl step2 at_step SslBump2
     acl step3 at_step SslBump3
     ssl_bump peek step1 all
     ssl_bump peek step2 allowed_https_sites
     ssl_bump splice step3 allowed_https_sites
     ssl_bump terminate step2 all
     ```

     - 保存 `squid.conf` 后，运行以下命令：

       ```
       sudo iptables -t nat -A PREROUTING -p tcp --dport 443 -j REDIRECT --to-port 3130
       sudo service squid restart
       ```

2. 将 `proxy` 添加到您的 buildspec 文件。有关更多信息，请参阅 buildspec 语法。

   ```
   version: 0.2
   proxy:
     upload-artifacts: yes
     logs: yes
   phases:
     build:
       commands:
         - command
   ```
   ```

> **Note**
>
> 如果您收到 RequestError 超时错误，请参阅RequestError CodeBuild 在代理服务器上运行时出现超时错误。

有关更多信息，请参阅本主题后面的显式代理服务器示例 s quid.conf ##。

创建 CodeBuild 项目

要 AWS CodeBuild 使用显式代理服务器运行，请使用您为代理服务器创建的 EC2 实例的私有 IP 地址HTTP_PROXY和项目级别的端口 3128 来设置其和HTTPS_PROXY环境变量。私有 IP 地址看起来类似于 `http://your-ec2-private-ip-address:3128`。有关更多信息，请参阅在中创建构建项目 AWS CodeBuild和在中更改构建项目设置 AWS CodeBuild。

使用以下命令查看 Squid 代理服务器访问日志：

```
sudo tail -f /var/log/squid/access.log
```

显式代理服务器示例 s **quid.conf ##**

以下是为显式代理服务器配置的 squid.conf 文件的示例。

```
 acl localnet src 10.0.0.0/16 #Only allow requests from within the VPC
 # add all URLS to be whitelisted for download source and commands to be run in build
environment
 acl allowed_sites dstdomain .github.com    #Allows to download source from github
 acl allowed_sites dstdomain .bitbucket.com #Allows to download source from bitbucket
 acl allowed_sites dstdomain ppa.launchpad.net #Allows to run apt-get in build
environment
 acl download_src dstdom_regex .*\.amazonaws\.com #Allows to download source from S3
or CodeCommit
 acl SSL_ports port 443
 acl Safe_ports port 80  # http
 acl Safe_ports port 21  # ftp
 acl Safe_ports port 443  # https
 acl Safe_ports port 70  # gopher
 acl Safe_ports port 210  # wais
 acl Safe_ports port 1025-65535 # unregistered ports
 acl Safe_ports port 280  # http-mgmt
 acl Safe_ports port 488  # gss-http
 acl Safe_ports port 591  # filemaker
```

```
  acl Safe_ports port 777  # multiling http
  acl CONNECT method CONNECT
  #
  # Recommended minimum Access Permission configuration:
  #
  # Deny requests to certain unsafe ports
  http_access deny !Safe_ports
  # Deny CONNECT to other than secure SSL ports
  http_access deny CONNECT !SSL_ports
  # Only allow cachemgr access from localhost
  http_access allow localhost manager
  http_access deny manager
  # We strongly recommend the following be uncommented to protect innocent
  # web applications running on the proxy server who think the only
  # one who can access services on "localhost" is a local user
  #http_access deny to_localhost
  #
  # INSERT YOUR OWN RULE(S) HERE TO ALLOW ACCESS FROM YOUR CLIENTS
  #
  # Example rule allowing access from your local networks.
  # Adapt localnet in the ACL section to list your (internal) IP networks
  # from where browsing should be allowed
  http_access allow localnet allowed_sites
  http_access allow localnet download_src
  http_access allow localhost
  # Add this for CodeBuild to access CWL end point, caching and upload artifacts S3
 bucket end point
  https_port 3130 cert=/etc/squid/ssl/squid.pem ssl-bump intercept
  acl SSL_port port 443
  http_access allow SSL_port
  acl allowed_https_sites ssl::server_name .amazonaws.com
  acl step1 at_step SslBump1
  acl step2 at_step SslBump2
  acl step3 at_step SslBump3
  ssl_bump peek step1 all
  ssl_bump peek step2 allowed_https_sites
  ssl_bump splice step3 allowed_https_sites
  ssl_bump terminate step2 all
  # And finally deny all other access to this proxy
  http_access deny all
  # Squid normally listens to port 3128
  http_port 3128
  # Uncomment and adjust the following to add a disk cache directory.
  #cache_dir ufs /var/spool/squid 100 16 256
```

```
  # Leave coredumps in the first cache dir
  coredump_dir /var/spool/squid
  #
  # Add any of your own refresh_pattern entries above these.
  #
  refresh_pattern ^ftp:  1440 20% 10080
  refresh_pattern ^gopher: 1440 0% 1440
  refresh_pattern -i (/cgi-bin/|\?) 0 0% 0
  refresh_pattern .  0 20% 4320
```

CodeBuild 在透明的代理服务器上运行

要 AWS CodeBuild 在透明的代理服务器中运行，必须将代理服务器配置为可以访问与之交互的网站和域名。

主题

- 将 Squid 配置为透明代理服务器
- 创建 CodeBuild 项目

将 Squid 配置为透明代理服务器

要将代理服务器配置为透明，您必须授予其访问您希望其访问的域和网站的权限。要 AWS CodeBuild 使用透明代理服务器运行，必须向其授予访问权限amazonaws.com。您还必须授予访问其他网站的 CodeBuild 权限。这些会有所不同，具体取决于您创建 CodeBuild 项目的方式。示例网站是 Bitbucket GitHub、Yum 和 Maven 等存储库的网站。要授予 Squid 访问特定域和网站的权限，请使用类似于以下内容的命令来更新 squid.conf 文件。此示例命令授予对 amazonaws.com、github.com 和 bitbucket.com 的访问权限。您可以编辑此示例以授予对其他网站的访问权限。

```
cat | sudo tee /etc/squid/squid.conf #EOF
visible_hostname squid
#Handling HTTP requests
http_port 3129 intercept
acl allowed_http_sites dstdomain .amazonaws.com
#acl allowed_http_sites dstdomain domain_name [uncomment this line to add another
 domain]
http_access allow allowed_http_sites
#Handling HTTPS requests
https_port 3130 cert=/etc/squid/ssl/squid.pem ssl-bump intercept
acl SSL_port port 443
http_access allow SSL_port
```

```
acl allowed_https_sites ssl::server_name .amazonaws.com
acl allowed_https_sites ssl::server_name .github.com
acl allowed_https_sites ssl::server_name .bitbucket.com
#acl allowed_https_sites ssl::server_name [uncomment this line to add another website]
acl step1 at_step SslBump1
acl step2 at_step SslBump2
acl step3 at_step SslBump3
ssl_bump peek step1 all
ssl_bump peek step2 allowed_https_sites
ssl_bump splice step3 allowed_https_sites
ssl_bump terminate step2 all
http_access deny all
EOF
```

来自私有子网中的实例的传入请求必须重定向到 Squid 端口。Squid 在端口 3129 上侦听 HTTP 流量（而不是 80），并在端口 3130 上侦听 HTTPS 流量（而不是 443）。使用 iptables 命令可路由流量：

```
sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-port 3129
sudo iptables -t nat -A PREROUTING -p tcp --dport 443 -j REDIRECT --to-port 3130
sudo service iptables save
sudo service squid start
```

创建 CodeBuild 项目

配置代理服务器后，无需进行更多配置即可 AWS CodeBuild 在私有子网中使用它。每个 HTTP 和 HTTPS 请求都经过公共代理服务器。使用以下命令查看 Squid 代理服务器访问日志：

```
sudo tail -f /var/log/squid/access.log
```

在代理服务器中运行程序包管理器和其他工具

按照以下过程在代理服务器中运行软件包管理器和其他工具。

要在代理服务器中运行一个工具，如程序包管理器，请执行以下操作：

1. 通过将语句添加到您的 `squid.conf` 文件中，将该工具添加到代理服务器的允许列表中。
2. 在 buildspec 文件中添加指向代理服务器的私有终端节点的命令行。

以下示例演示了如何为 apt-get、curl 和 maven 执行此操作。如果您使用其他工具，则相同的原则将适用。将其添加到squid.conf文件中的允许列表中，并将命令添加到您的 buildspec 文件中，以 CodeBuild 了解代理服务器的终端节点。

在代理服务器### apt-get

1. 将以下语句添加到您的 squid.conf 文件中，以便将 apt-get 添加到代理服务器中的允许列表。前三行允许 apt-get 在构建环境中运行。

```
acl allowed_sites dstdomain ppa.launchpad.net # Required for apt-get to run in the
 build environment
acl apt_get dstdom_regex .*\.launchpad.net # Required for CodeBuild to run apt-get
 in the build environment
acl apt_get dstdom_regex .*\.ubuntu.com    # Required for CodeBuild to run apt-get
 in the build environment
http_access allow localnet allowed_sites
http_access allow localnet apt_get
```

2. 在构建规范文件中添加以下语句，以便 apt-get 命令在 /etc/apt/apt.conf.d/00proxy 中查找代理配置。

```
echo 'Acquire::http::Proxy "http://<private-ip-of-proxy-server>:3128";
 Acquire::https::Proxy "http://<private-ip-of-proxy-server>:3128";
 Acquire::ftp::Proxy "http://<private-ip-of-proxy-server>:3128";' > /etc/apt/
apt.conf.d/00proxy
```

在代理服务器### curl

1. 将以下内容添加到您的 squid.conf 文件中，以便将 curl 添加到构建环境中的允许列表。

```
acl allowed_sites dstdomain ppa.launchpad.net # Required to run apt-get in the
 build environment
acl allowed_sites dstdomain google.com # Required for access to a webiste. This
 example uses www.google.com.
http_access allow localnet allowed_sites
http_access allow localnet apt_get
```

2. 在 buildspec 文件中添加以下语句，以便 curl 使用专用代理服务器访问您添加到 squid.conf 的网站。在此示例中，网站为 google.com。

```
curl -x <private-ip-of-proxy-server>:3128 https://www.google.com
```

在代理服务器### maven

1. 将以下内容添加到您的 squid.conf 文件中，以便将 maven 添加到构建环境中的允许列表。

```
acl allowed_sites dstdomain ppa.launchpad.net # Required to run apt-get in the
 build environment
acl maven dstdom_regex .*\.maven.org # Allows access to the maven repository in the
 build environment
http_access allow localnet allowed_sites
http_access allow localnet maven
```

2. 在 buildspec 文件中添加以下语句。

```
maven clean install -DproxySet=true -DproxyHost=<private-ip-of-proxy-server> -
DproxyPort=3128
```

CloudFormation VPC 模板

CloudFormation 通过使用模板文件将资源集合作为一个单元（堆栈）一起创建和删除，使您能够以可预测的方式重复创建和配置 AWS 基础架构部署。有关更多信息，请参阅 CloudFormation 《用户指南》。

以下是用于配置要使用 AWS CodeBuild的 VPC 的 CloudFormation YAML 模板。该文件也可在 samples.zip 中找到。

```
Description:  This template deploys a VPC, with a pair of public and private subnets
 spread
  across two Availability Zones. It deploys an internet gateway, with a default
  route on the public subnets. It deploys a pair of NAT gateways (one in each AZ),
  and default routes for them in the private subnets.

Parameters:
  EnvironmentName:
    Description: An environment name that is prefixed to resource names
    Type: String

  VpcCIDR:
```

```
    Description: Please enter the IP range (CIDR notation) for this VPC
    Type: String
    Default: 10.192.0.0/16

  PublicSubnet1CIDR:
    Description: Please enter the IP range (CIDR notation) for the public subnet in the
 first Availability Zone
    Type: String
    Default: 10.192.10.0/24

  PublicSubnet2CIDR:
    Description: Please enter the IP range (CIDR notation) for the public subnet in the
 second Availability Zone
    Type: String
    Default: 10.192.11.0/24

  PrivateSubnet1CIDR:
    Description: Please enter the IP range (CIDR notation) for the private subnet in
 the first Availability Zone
    Type: String
    Default: 10.192.20.0/24

  PrivateSubnet2CIDR:
    Description: Please enter the IP range (CIDR notation) for the private subnet in
 the second Availability Zone
    Type: String
    Default: 10.192.21.0/24

Resources:
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: !Ref VpcCIDR
      EnableDnsSupport: true
      EnableDnsHostnames: true
      Tags:
        - Key: Name
          Value: !Ref EnvironmentName

  InternetGateway:
    Type: AWS::EC2::InternetGateway
    Properties:
      Tags:
        - Key: Name
```

```
        Value: !Ref EnvironmentName

  InternetGatewayAttachment:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      InternetGatewayId: !Ref InternetGateway
      VpcId: !Ref VPC

  PublicSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [ 0, !GetAZs '' ]
      CidrBlock: !Ref PublicSubnet1CIDR
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Public Subnet (AZ1)

  PublicSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [ 1, !GetAZs  '' ]
      CidrBlock: !Ref PublicSubnet2CIDR
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Public Subnet (AZ2)

  PrivateSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [ 0, !GetAZs  '' ]
      CidrBlock: !Ref PrivateSubnet1CIDR
      MapPublicIpOnLaunch: false
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Subnet (AZ1)

  PrivateSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
```

```
      VpcId: !Ref VPC
      AvailabilityZone: !Select [ 1, !GetAZs  '' ]
      CidrBlock: !Ref PrivateSubnet2CIDR
      MapPublicIpOnLaunch: false
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Subnet (AZ2)

  NatGateway1EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc

  NatGateway2EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc

  NatGateway1:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway1EIP.AllocationId
      SubnetId: !Ref PublicSubnet1

  NatGateway2:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway2EIP.AllocationId
      SubnetId: !Ref PublicSubnet2

  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Public Routes

  DefaultPublicRoute:
    Type: AWS::EC2::Route
    DependsOn: InternetGatewayAttachment
    Properties:
```

```
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref InternetGateway

  PublicSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet1

  PublicSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet2


  PrivateRouteTable1:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Routes (AZ1)

  DefaultPrivateRoute1:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway1

  PrivateSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      SubnetId: !Ref PrivateSubnet1

  PrivateRouteTable2:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
```

```
          Value: !Sub ${EnvironmentName} Private Routes (AZ2)

  DefaultPrivateRoute2:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway2

  PrivateSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      SubnetId: !Ref PrivateSubnet2

  NoIngressSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupName: "no-ingress-sg"
      GroupDescription: "Security group with no ingress rule"
      VpcId: !Ref VPC

Outputs:
  VPC:
    Description: A reference to the created VPC
    Value: !Ref VPC

  PublicSubnets:
    Description: A list of the public subnets
    Value: !Join [ ",", [ !Ref PublicSubnet1, !Ref PublicSubnet2 ]]

  PrivateSubnets:
    Description: A list of the private subnets
    Value: !Join [ ",", [ !Ref PrivateSubnet1, !Ref PrivateSubnet2 ]]

  PublicSubnet1:
    Description: A reference to the public subnet in the 1st Availability Zone
    Value: !Ref PublicSubnet1

  PublicSubnet2:
    Description: A reference to the public subnet in the 2nd Availability Zone
    Value: !Ref PublicSubnet2

  PrivateSubnet1:
```

```
    Description: A reference to the private subnet in the 1st Availability Zone
    Value: !Ref PrivateSubnet1

  PrivateSubnet2:
    Description: A reference to the private subnet in the 2nd Availability Zone
    Value: !Ref PrivateSubnet2

  NoIngressSecurityGroup:
    Description: Security group with no ingress rule
    Value: !Ref NoIngressSecurityGroup
```

登录和监控 AWS CodeBuild

日志记录和监控是维护 AWS 解决方案的可靠性、可用性和性能的重要组成部分。 AWS CodeBuild 您应该从 AWS 解决方案的所有部分收集监控数据，以便在出现多点故障时可以更轻松地进行调试。AWS 提供了以下工具，用于监控您的 CodeBuild 资源和版本以及响应潜在事件。

主题

- Log AWS CodeBuild 使用 API 调用 AWS CloudTrail
- 使用以下 CodeBuild 命令监控构建 CloudWatch

Log AWS CodeBuild 使用 API 调用 AWS CloudTrail

AWS CodeBuild 与 AWS CloudTrail一项服务集成，该服务提供用户、角色或 AWS 服务在中执行的操作的记录 CodeBuild。 CloudTrail 将所有 API 调用捕获 CodeBuild 为事件，包括来自 CodeBuild 控制台的调用和对 CodeBuild API 的代码调用。如果您创建跟踪，则可以启用向 S3 存储桶持续交付事件，包括的事件 CodeBuild。 CloudTrail 如果您未配置跟踪，仍可以在 CloudTrail 控制台的事件历史记录中查看最新事件。使用收集的信息 CloudTrail，您可以确定向谁发出了请求 CodeBuild、发出请求的 IP 地址、谁提出了请求、何时提出请求以及其他详细信息。

要了解更多信息 CloudTrail，请参阅AWS CloudTrail 用户指南。

主题

- 关于我们 AWS CodeBuild 中的信息 CloudTrail
- 关于我们 AWS CodeBuild 日志文件条目

关于我们 AWS CodeBuild 中的信息 CloudTrail

CloudTrail 在您创建 AWS 账户时已在您的账户上启用。当活动发生在中时 CodeBuild，该活动将与 CloudTrail 事件历史记录中的其他 AWS 服务事件一起记录在事件中。您可以在您的 AWS 账户中查看、搜索和下载最近的事件。有关更多信息，请参阅《AWS CloudTrail 用户指南》中的查看带有 CloudTrail 事件历史记录的事件。

要持续记录 AWS 账户中的事件（包括的事件） CodeBuild，请创建跟踪。跟踪可以 CloudTrail 将日志文件传送到 S3 存储桶。默认情况下，在控制台中创建跟踪记录时，此跟踪记录应用于所有区域。跟踪记录 AWS 分区中所有区域的事件，并将日志文件传送到您指定的 S3 存储桶。您可以配置其他 AWS 服务，以进一步分析和处理 CloudTrail 日志中收集的事件数据。有关更多信息，请参阅:

- 创建跟踪概览
- CloudTrail 支持的服务和集成
- 为配置亚马逊 SNS 通知 CloudTrail
- 接收来自多个区域的 CloudTrail 日志文件和接收来自多个账户的 CloudTrail 日志文件

所有 CodeBuild 操作均由 API 参考记录 CloudTrail 并记录在 CodeBuild API 参考中。例如，调用`CreateProject`（在 AWS CLI、`create-project`）、`StartBuild`（在 AWS CLI、`start-project`）和`UpdateProject`（在 AWS CLI，中`update-project`）操作会在 CloudTrail 日志文件中生成条目。

每个事件或日志条目都包含有关生成请求的人员信息。身份信息有助于您确定以下内容：

- 请求是使用根凭证还是用户凭证发出的。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

有关更多信息，请参阅《CloudTrail 用户指南》中的 userIdentity 元素。AWS CloudTrail

关于我们 AWS CodeBuild 日志文件条目

跟踪是一种配置，允许将事件作为日志文件传送到您指定的 S3 存储桶。 CloudTrail 日志文件包含一个或多个日志条目。事件代表来自任何来源的单个请求，包括有关请求的操作、操作的日期和时间、请求参数等的信息。 CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此它们不按任何特定顺序出现。

> **Note**
>
> 为了保护敏感信息，以下内容隐藏在 CodeBuild 日志中：
>
> - AWS 访问密钥 ID。有关更多信息，请参阅《AWS Identity and Access Management 用户指南》中的管理 IAM 用户的访问密钥。
> - 使用参数存储指定的字符串。有关更多信息，请参阅《Amazon EC2 Systems Manager 用户指南》中的 Systems Manager Parameter Store 和 Systems Manager Parameter Store 控制台演练。
> - 使用指定的字符串 AWS Secrets Manager。有关更多信息，请参阅 密钥管理。

以下示例显示了演示在中创建构建项目的 CloudTrail 日志条目 CodeBuild。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "FederatedUser",
    "principalId": "account-ID:user-name",
    "arn": "arn:aws:sts::account-ID:federated-user/user-name",
    "accountId": "account-ID",
    "accessKeyId": "access-key-ID",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2016-09-06T17:59:10Z"
      },
      "sessionIssuer": {
        "type": "IAMUser",
        "principalId": "access-key-ID",
        "arn": "arn:aws:iam::account-ID:user/user-name",
        "accountId": "account-ID",
        "userName": "user-name"
      }
    }
  },
  "eventTime": "2016-09-06T17:59:11Z",
  "eventSource": "codebuild.amazonaws.com",
  "eventName": "CreateProject",
  "awsRegion": "region-ID",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "user-agent",
  "requestParameters": {
    "awsActId": "account-ID"
  },
  "responseElements": {
    "project": {
      "environment": {
        "image": "image-ID",
        "computeType": "BUILD_GENERAL1_SMALL",
        "type": "LINUX_CONTAINER",
        "environmentVariables": []
      },
      "name": "codebuild-demo-project",
      "description": "This is my demo project",
```

```
      "arn": "arn:aws:codebuild:region-ID:account-ID:project/codebuild-demo-
project:project-ID",
      "encryptionKey": "arn:aws:kms:region-ID:key-ID",
      "timeoutInMinutes": 10,
      "artifacts": {
        "location": "arn:aws:s3:::codebuild-region-ID-account-ID-output-bucket",
        "type": "S3",
        "packaging": "ZIP",
        "outputName": "MyOutputArtifact.zip"
      },
      "serviceRole": "arn:aws:iam::account-ID:role/CodeBuildServiceRole",
      "lastModified": "Sep 6, 2016 10:59:11 AM",
      "source": {
        "type": "GITHUB",
        "location": "https://github.com/my-repo.git"
      },
      "created": "Sep 6, 2016 10:59:11 AM"
    }
  },
  "requestID": "9d32b228-745b-11e6-98bb-23b67EXAMPLE",
  "eventID": "581f7dd1-8d2e-40b0-aeee-0dbf7EXAMPLE",
  "eventType": "AwsApiCall",
  "recipientAccountId": "account-ID"
}
```

使用以下 CodeBuild 命令监控构建 CloudWatch

您可以使用亚马逊 CloudWatch 来查看您的构建，报告出现问题，并在适当时自动采取措施。可以监控两个级别的构建：

项目级别

这些指标适用于指定项目中的所有构建。要查看项目的指标，请为 CloudWatch 中的维度指定 `ProjectName`。

AWS 账户等级

这些指标适用于一个账户中的所有构建。要查看 AWS 账户级别的指标，请勿在 CloudWatch 中输入维度。 AWS 账号级别不提供版本资源利用率指标。

CloudWatch 指标显示您的构建在一段时间内的行为。例如，可以监控：

- 随着时间的推移，一个构建项目或一个 AWS 账户中尝试了多少次构建。
- 随着时间的推移，一个构建项目或一个 AWS 账户中有多少次成功构建。
- 随着时间的推移，构建项目或 AWS 账户中有多少构建失败。
- 随着时间的推移，在构建项目或 AWS 账户中运行构建所 CodeBuild 花费的时间。
- 构建或整个构建项目的构建资源利用率。构建资源利用率指标包括 CPU、内存和存储利用率等指标。

有关更多信息，请参阅 查看 CodeBuild 指标。

CodeBuild CloudWatch 指标

可以跟踪每个 AWS 账户或构建项目的以下指标。有关 CloudWatch与一起使用的更多信息 CodeBuild，请参阅使用以下 CodeBuild 命令监控构建 CloudWatch。

BuildDuration

测量构建的 BUILD 阶段的持续时间。

单位：秒

有效 CloudWatch 统计数据：平均值（推荐）、最大值、最小值

Builds

测量所触发构建的数量。

单位：计数

有效 CloudWatch 统计数据：总和

DownloadSourceDuration

测量构建的 DOWNLOAD_SOURCE 阶段的持续时间。

单位：秒

有效 CloudWatch 统计数据：平均值（推荐）、最大值、最小值

Duration

测量随着时间的推移所有构建的持续时间。

单位：秒

有效 CloudWatch 统计数据：平均值（推荐）、最大值、最小值

FailedBuilds

测量因为客户端错误或超时而失败的构建的数量。

单位：计数

有效 CloudWatch 统计数据：总和

FinalizingDuration

测量构建的 FINALIZING 阶段的持续时间。

单位：秒

有效 CloudWatch 统计数据：平均值（推荐）、最大值、最小值

InstallDuration

测量构建的 INSTALL 阶段的持续时间。

单位：秒

有效 CloudWatch 统计数据：平均值（推荐）、最大值、最小值

PostBuildDuration

测量构建的 POST_BUILD 阶段的持续时间

单位：秒

有效 CloudWatch 统计数据：平均值（推荐）、最大值、最小值

PreBuildDuration

测量构建的 PRE_BUILD 阶段的持续时间。

单位：秒

有效 CloudWatch 统计数据：平均值（推荐）、最大值、最小值

ProvisioningDuration

测量构建的 PROVISIONING 阶段的持续时间。

单位：秒

有效 CloudWatch 统计数据：平均值（推荐）、最大值、最小值

QueuedDuration

测量构建的 QUEUED 阶段的持续时间。

单位：秒

有效 CloudWatch 统计数据：平均值（推荐）、最大值、最小值

SubmittedDuration

测量构建的 SUBMITTED 阶段的持续时间。

单位：秒

有效 CloudWatch 统计数据：平均值（推荐）、最大值、最小值

SucceededBuilds

测量成功构建的数量。

单位：计数

有效 CloudWatch 统计数据：总和

UploadArtifactsDuration

测量构建的 UPLOAD_ARTIFACTS 阶段的持续时间。

单位：秒

有效 CloudWatch 统计数据：平均值（推荐）、最大值、最小值

CodeBuild CloudWatch 资源利用率指标

> **Note**
>
> CodeBuild 资源利用率指标仅在以下区域可用：
>
> - 亚太地区（东京）区域
> - 亚太地区（首尔）区域
> - 亚太地区（孟买） 区域
> - 亚太地区（新加坡）区域
> - Asia Pacific (Sydney) Region

- 加拿大（中部）区域
- 欧洲地区（法兰克福）区域
- 欧洲地区（爱尔兰）区域
- 欧洲地区（伦敦）区域
- 欧洲地区（巴黎）区域
- South America (São Paulo) Region
- US East (N. Virginia) Region
- 美国东部（俄亥俄州）区域
- 美国西部（北加利福尼亚）区域
- 美国西部（俄勒冈州）区域

可以跟踪以下资源利用率指标。有关 CloudWatch与一起使用的更多信息 CodeBuild，请参阅使用以下 CodeBuild 命令监控构建 CloudWatch。

CPUUtilized

构建容器使用的分配的 CPU 处理单元数量。

单位：CPU 单元数量

有效 CloudWatch 统计数据：平均值（推荐）、最大值、最小值

CPUUtilizedPercent

构建容器使用的分配的处理单元所占百分比。

单位：百分比

有效 CloudWatch 统计数据：平均值（推荐）、最大值、最小值

MemoryUtilized

构建容器使用的内存兆字节数。

单位：MB

有效 CloudWatch 统计数据：平均值（推荐）、最大值、最小值

MemoryUtilizedPercent

构建容器使用的分配的内存所占百分比。

单位：百分比

有效 CloudWatch 统计数据：平均值（推荐）、最大值、最小值

StorageReadBytes

构建容器使用的存储读取速度。

单位： Bytes/second

有效 CloudWatch 统计数据：平均值（推荐）、最大值、最小值

StorageWriteBytes

构建容器使用的存储写入速度。

单位： Bytes/second

有效 CloudWatch 统计数据：平均值（推荐）、最大值、最小值

CodeBuild CloudWatch 维度

CodeBuild 提供以下 CloudWatch 指标维度。如果未指定这些指标，则指标适用于当前 AWS 账户。

BuildId, BuildNumber, ProjectName

提供针对构建标识符、内部版本号和项目名称的指标。

ProjectName

提供针对项目名称的指标。

CodeBuild CloudWatch 警报

您可以使用 CloudWatch 控制台根据 CodeBuild 指标创建警报，以便在构建出现问题时做出反应。以下要点说明了对警报最有用的两个指标。有关 CloudWatch与一起使用的更多信息 CodeBuild，请参阅使用以下 CodeBuild 命令监控构建 CloudWatch。

- `FailedBuild`。可以创建在预先确定的秒数内检测到特定数量的失败构建时触发的警报。在中 CloudWatch，您可以指定秒数以及有多少失败的构建会触发警报。
- `Duration`。可以创建在构建所用时间长于预期时触发的警报。指定在启动构建之后、完成构建之前必须经历多少秒才会触发警报。

有关如何为 CodeBuild 指标创建警报的信息，请参阅使用 CloudWatch 警报监控 CodeBuild 构建。有关警报的更多信息，请参阅亚马逊 CloudWatch 用户指南中的创建亚马逊 CloudWatch 警报。

查看 CodeBuild 指标

AWS CodeBuild 代表您监控功能并通过 Amazon 报告指标 CloudWatch。上述指标包括构建总数量、失败构建数量、成功构建数量和构建的持续时间。

您可以使用 CodeBuild 控制台或控制 CloudWatch 台来监控的指标 CodeBuild。以下过程演示如何查看指标。

主题

- 查看编译指标（CodeBuild 控制台）
- 查看构建指标（Amazon CloudWatch 控制台）

查看编译指标（CodeBuild 控制台）

> **Note**
>
> 您无法自定义用于在 CodeBuild控制台中显示的指标或图表。如果您想自定义显示效果，请使用 Amazon CloudWatch 控制台查看您的构建指标。

Account-level 指标

查看 AWS 账户级别指标

1. 登录 AWS 管理控制台 并打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择账户指标。

Project-level 指标

查看项目级别指标

1. 登录 AWS 管理控制台 并打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择构建项目。

3. 在构建项目列表中的名称列中，选择要查看其指标的项目。
4. 请选择指标选项卡。

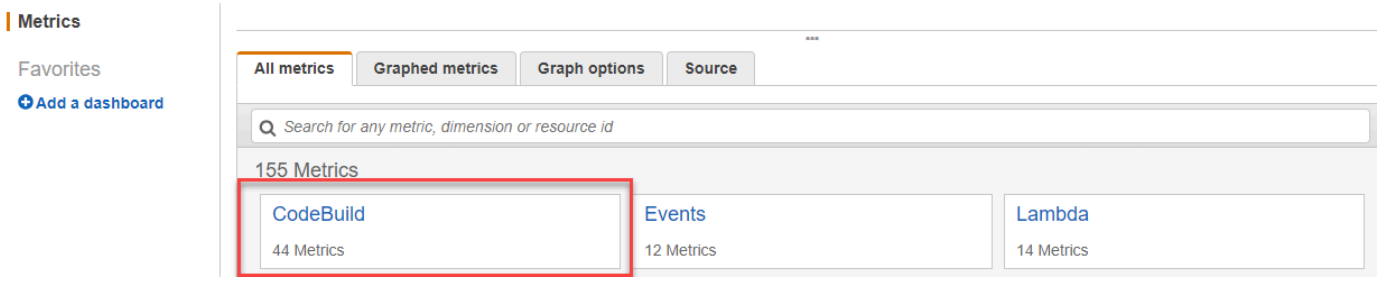
查看构建指标（Amazon CloudWatch 控制台）

您可以通过 CloudWatch控制台自定义指标和用于显示这些指标的图表。

Account-level 指标

查看账户级别指标

1. 登录 AWS 管理控制台 并打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 在导航窗格中，选择指标。
3. 在全部指标选项卡上，选择 CodeBuild。

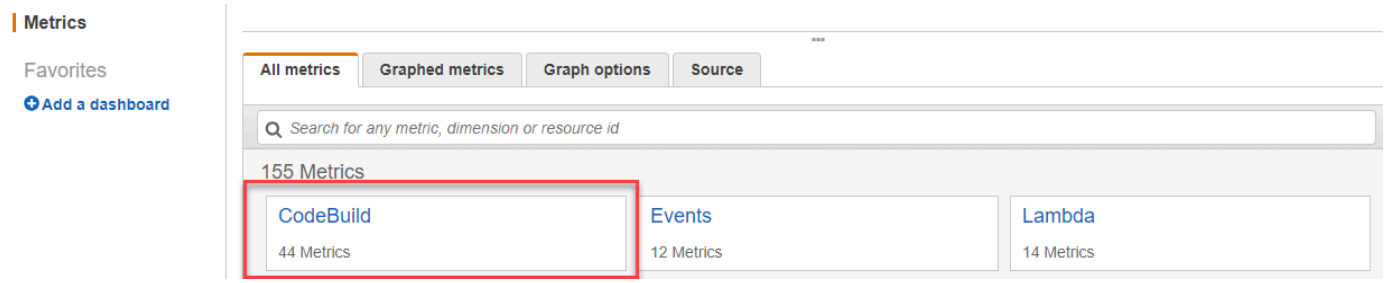


4. 选择账户指标。
5. 选择一个或多个项目和指标。对于每个项目，您可以选择SucceededBuilds、FailedBuilds、生成和持续时间指标。此页面上的图表中将显示所有选定项目和指标组合。

Project-level 指标

查看项目级别指标

1. 登录 AWS 管理控制台 并打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 在导航窗格中，选择指标。
3. 在全部指标选项卡上，选择 CodeBuild。



4. 选择按项目。
5. 选择一个或多个项目和指标组合。对于每个项目，您可以选择SucceededBuilds、FailedBuilds、生成和持续时间指标。此页面上的图表中将显示所有选定项目和指标组合。
6. （可选）可以自定义指标和图表。例如，从统计数据列的下拉列表中，可以选择要显示的不同统计数据。或从周期列的下拉菜单中，可以选择要用于监控指标的不同时间段。

   有关更多信息，请参阅 Amazon CloudWatch 用户指南中的图表指标和查看可用指标。

查看 CodeBuild 资源利用率指标

AWS CodeBuild 代表您监控构建资源利用率，并通过 Amazon 报告指标 CloudWatch。其中包括 CPU、内存和存储利用率等指标。

> **Note**
>
> CodeBuild 仅记录运行时间超过一分钟的版本的资源利用率指标。

您可以使用 CodeBuild 控制台或控制 CloudWatch 台来监控的资源利用率指标 CodeBuild。

> **Note**
>
> CodeBuild 资源利用率指标仅在以下区域可用：
>
> - 亚太地区（东京）区域
> - 亚太地区（首尔）区域
> - 亚太地区（孟买）区域
> - 亚太地区（新加坡）区域
> - 亚太地区（悉尼）区域

- 加拿大（中部）区域
- 欧洲地区（法兰克福）区域
- 欧洲地区（爱尔兰）区域
- 欧洲地区（伦敦）区域
- 欧洲地区（巴黎）区域
- 南美洲（圣保罗）区域
- US East (N. Virginia) Region
- 美国东部（俄亥俄州）区域
- 美国西部（北加利福尼亚）区域
- 美国西部（俄勒冈州）区域

以下过程演示如何访问您的资源利用率指标。

主题

- 访问资源利用率指标（CodeBuild 控制台）
- 访问资源利用率指标（Amazon CloudWatch 控制台）

访问资源利用率指标（CodeBuild 控制台）

> **Note**
>
> 您无法自定义用于在 CodeBuild控制台中显示的指标或图表。如果您想自定义显示效果，请使用 Amazon CloudWatch 控制台查看您的构建指标。

Project-level 资源利用率指标

要访问项目级别的资源利用率指标

1. 登录 AWS 管理控制台 并打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择构建项目。
3. 在构建项目列表中的名称列中，选择要查看其利用率指标的项目。
4. 请选择指标选项卡。资源利用率指标显示在资源利用率指标部分。

5. 要在 CloudWatch控制台中查看项目级别的资源利用率指标，请在 “资源利用率指标” 部分 CloudWatch中选择 “查看”。

Build-level 资源利用率指标

要访问构建级别资源利用率指标

1. 登录 AWS 管理控制台 并打开 AWS CodeBuild 控制台，网址为https://console.aws.amazon.com/codesuite/codebuild/home。
2. 在导航窗格中，选择构建历史记录。
3. 在构建列表中的构建运行列中，选择要查看其利用率指标的构建。
4. 选择资源利用率选项卡。
5. 要在 CloudWatch 控制台中查看构建级别的资源利用率指标，请在 “资源利用率指标” 部分 CloudWatch中选择 “查看”。

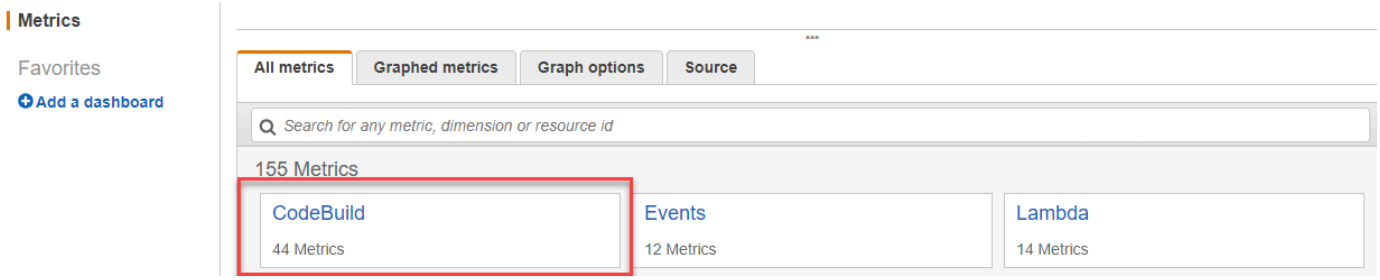
访问资源利用率指标（Amazon CloudWatch 控制台）

Amazon CloudWatch 控制台可用于访问 CodeBuild 资源利用率指标。

Project-level 资源利用率指标

要访问项目级别的资源利用率指标

1. 登录 AWS 管理控制台 并打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 在导航窗格中，选择指标。
3. 在全部指标选项卡上，选择 CodeBuild。



4. 选择按项目。
5. 选择一个或多个项目和指标组合，以添加到图表中。此页面上的图表中将显示所有选定项目和指标组合。

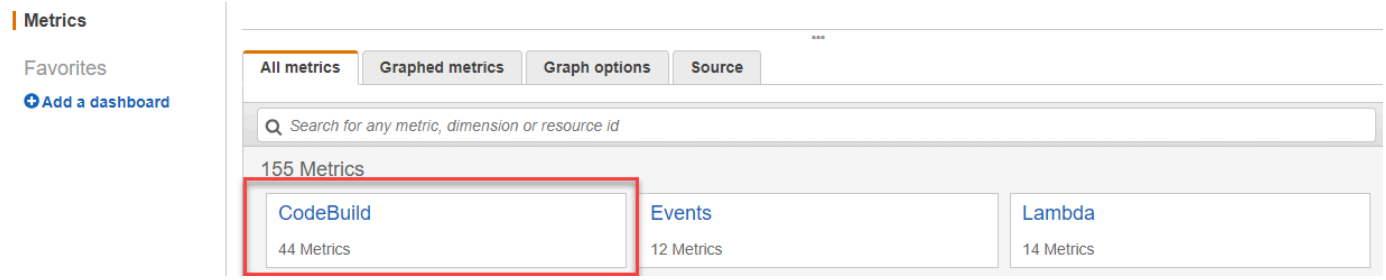
6. （可选）您可以在绘成图表的指标选项卡中自定义指标和图表。例如，从统计数据列的下拉列表中，可以选择要显示的不同统计数据。或从周期列的下拉菜单中，可以选择要用于监控指标的不同时间段。

   有关更多信息，请参阅 Amazon CloudWatch 用户指南中的绘制指标和查看可用指标。

Build-level 资源利用率指标

要访问构建级别资源利用率指标

1. 登录 AWS 管理控制台 并打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 在导航窗格中，选择指标。
3. 在全部指标选项卡上，选择 CodeBuild。



4. 选择BuildId、 BuildNumber、 ProjectName。
5. 选择一个或多个构建和指标组合，以添加到图表中。此页面上的图表中将显示所有选定构建和指标组合。
6. （可选）您可以在绘成图表的指标选项卡中自定义指标和图表。例如，从统计数据列的下拉列表中，可以选择要显示的不同统计数据。或从周期列的下拉菜单中，可以选择要用于监控指标的不同时间段。

   有关更多信息，请参阅 Amazon CloudWatch 用户指南中的绘制指标和查看可用指标。

使用 CloudWatch 警报监控 CodeBuild 构建

您可以为自己的版本创建 CloudWatch 警报。警报将监控某个指标在一个时间段（由您指定）的变化情况，并根据相对于指定阈值的指标值每隔若干个时间段执行一个或多个操作。使用原生 CloudWatch 警报功能，您可以指定超过阈值 CloudWatch 时支持的任何操作。例如，可以指定 15 分钟内如果账户中有三个以上的构建失败，则发送 Amazon SNS 通知。

为 CodeBuild 指标创建 CloudWatch 警报

1. 登录 AWS 管理控制台 并打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 在导航窗格中，选择告警。
3. 选择创建告警。
4. 在 “按类别划分的CloudWatch 指标” 下，选择CodeBuild指标。如果您知道您只需要项目级别指标，则选择按项目。如果您知道您只需要账户级别指标，则选择账户指标。
5. 在创建警报上，请选择选择指标（如果尚未选择）。
6. 选择要为之创建警报的指标。选项为按项目或账户指标。
7. 选择下一步或定义警报，然后创建警报。有关更多信息，请参阅亚马逊 CloudWatch 用户指南中的创建亚马逊 CloudWatch 警报。有关设置触发警报时的 Amazon SNS 通知的更多信息，请参阅《Amazon SNS 开发人员指南》中的设置 Amazon SNS 同志。
8. 选择创建警报。

中的安全性 AWS CodeBuild

云安全 AWS 是重中之重。作为 AWS 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是双方共同承担 AWS 的责任。责任共担模式将其描述为云的 安全性和云中 的安全性：

- 云安全 — AWS 负责保护在云中运行 AWS 服务的基础架构 AWS Cloud。 AWS 还为您提供可以安全使用的服务。 Third-party 作为AWS 合规计划的一部分，审计师定期测试和验证我们安全的有效性。要了解适用的合规计划 AWS CodeBuild，请参阅按合规计划划分的范围内的AWS 服务。
- 云端安全-您的责任由您使用的 AWS 服务决定。您还需要对其它因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

本文档可帮助您了解在使用时如何应用分担责任模型 CodeBuild。以下主题向您介绍如何进行配置 CodeBuild 以满足您的安全和合规性目标。您还将学习如何使用其他 AWS 服务来帮助您监控和保护您的 CodeBuild 资源。

主题

- 中的数据保护 AWS CodeBuild
- 中的身份和访问管理 AWS CodeBuild
- 的合规性验证 AWS CodeBuild
- 韧性在 AWS CodeBuild
- 中的基础设施安全 AWS CodeBuild
- 在以下位置访问您的源提供商 CodeBuild
- Cross-service 混乱的副手预防

中的数据保护 AWS CodeBuild

AWS 责任共担模式适用于保护 AWS CodeBuild中的数据。如本模型所述 AWS ，负责保护运行所有内容的全球基础架构 AWS Cloud。您负责维护对托管在此基础结构上的内容的控制。您还负责您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题解答 AWS条款。 有关欧洲数据保护的信息，请参阅通用数据保护条例（GDPR）中心。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得履行其工作职责所需的权限。还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证（MFA）。
- 用于 SSL/TLS 与 AWS 资源通信。我们要求使用 TLS 1.2，并建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 AWS CloudTrail。有关使用 CloudTrail 跟踪捕获 AWS 活动的信息，请参阅《AWS CloudTrail 用户指南》中的使用跟 CloudTrail 踪。
- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。
- 使用高级托管安全服务（如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在 AWS 通过命令行界面或 API 进行访问时需要经过 FIPS 140-3 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准（FIPS）第 140-3 版》https://aws.amazon.com/compliance/fips/。

强烈建议您切勿将机密信息或敏感信息（如客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括您使用控制台、API CodeBuild 或 SDK 或以其他 AWS 服务 方式使用控制台 AWS CLI、API 或 AWS SDK 的情况。在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供 URL，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

为了保护敏感信息， CodeBuild 日志中隐藏了以下内容：

- 在 CodeBuild 项目环境变量或 buildspec `env/parameter-store` 部分中使用参数存储指定的字符串。有关更多信息，请参阅《Amazon EC2 Systems Manager 用户指南》中的 Systems Manager Parameter Store 和 Systems Manager Parameter Store 控制台演练。
- AWS Secrets Manager 在 CodeBuild 项目环境变量或 buildspec `env/secrets-manager` 部分中使用指定的字符串。有关更多信息，请参阅 密钥管理。

> **Note**
>
> 掩码与 Amazon EC2 Systems Manager 参数存储中存储的确切值相匹配，或者 AWS Secrets Manager。如果构建命令在密钥出现在日志中之前对其进行转换，则结果将是一个未被屏蔽的不同字符串。例如，使用 Base64 对其进行编码、颠倒它、更改大小写或将其与其他文本连接在一起会创建一个不同的值，该值可能会出现在构建日志中。

要屏蔽某个值，请将其存储在编译版本使用的形式中。例如，如果您的版本需要 Base64-encoded 值，请将该 Base64-encoded 字符串直接存储在 AWS Secrets Manager 或 Amazon EC2 Systems Manager 参数存储中。这样，记录的值就会与存储的值相匹配。

有关数据保护的更多信息，请参阅 AWS 安全性博客上的 AWS 责任共担模式和 GDPR 博客文章。

主题

- 数据加密
- 密钥管理
- 流量隐私

数据加密

加密是 CodeBuild 安全的重要组成部分。某些加密（例如，针对传输中的数据的加密）是默认提供的，无需您执行任何操作。其他加密（例如，针对静态数据的加密）可在创建项目或构建时进行配置。

- 静态数据加密-默认使用 AWS 托管式密钥加密生成工件，例如缓存、日志、导出的原始测试报告数据文件和生成结果。如果您不想使用这些 KMS 密钥，则必须创建并配置一个客户托管密钥。有关更多信息，请参阅《AWS Key Management Service 用户指南》中的创建 KMS 密钥和 AWS Key Management Service 概念。
  - 您可以将 CodeBuild 用于加密生成输出项目的 AWS KMS 密钥的标识符存储在`CODEBUILD_KMS_KEY_ID`环境变量中。有关更多信息，请参阅 构建环境中的环境变量。
  - 可以在创建构建项目时指定客户托管密钥。有关更多信息，请参阅 Set the Encryption Key Using the Console 和使用 CLI 设置加密密钥。

  默认情况下，您的构建队列的 Amazon Elastic Block Store 卷是使用加密的 AWS 托管式密钥。
- 传输中的数据加密-使用签名版本 4 签名流程签名的 TLS 连接保护客户之间 CodeBuild CodeBuild 以及客户之间及其下游依赖关系之间的所有通信。所有 CodeBuild 端点都使用由管理的 SHA-256证书 AWS 私有证书颁发机构。有关更多信息，请参阅签名版本 4 签名流程和什么是 ACM PCA。
- 构建构件加密-与构建项目关联的 CodeBuild 服务角色需要访问 KMS 密钥才能加密其构建输出工件。默认情况下，在您的 AWS 账户中 CodeBuild 使用 AWS 托管式密钥 适用于 Amazon S3 的。如果您不想使用此 AWS 托管式密钥，则必须创建并配置一个客户托管密钥。有关更多信息，请参阅加密构建输出和《AWS KMS 开发人员指南》中的创建密钥。

密钥管理

可以通过加密保护您的内容免遭未经授权的使用。将您的加密密钥存储在中 AWS Secrets Manager，然后向与构建项目关联的 CodeBuild 服务角色授予从您的 Secrets Manager 账户获取加密密钥的权限。有关更多信息，请参阅使用客户托管密钥加密构建输出、在中创建构建项目 AWS CodeBuild、手动运行 AWS CodeBuild 构建和教程：存储和检索密钥。

在生成命令中使用CODEBUILD_KMS_KEY_ID环境变量来获取 AWS KMS 密钥标识符。有关更多信息，请参阅 构建环境中的环境变量。

可以使用 Secrets Manager 保护存储用于运行时环境的 Docker 映像的私有注册表的凭证。有关更多信息，请参阅 带有 AWS Secrets Manager 示例的私有注册表 CodeBuild。

流量隐私

您可以通过配置为使用接口 VPC 终端节点 CodeBuild 来提高构建的安全性。为此，您无需互联网网关、NAT 设备或虚拟私有网关。尽管建议这样做 PrivateLink，但也无需进行配置。有关更多信息，请参阅 使用 VPC 端点。有关 PrivateLink 和 VPC 终端节点的更多信息，请参阅AWS PrivateLink和通过访问 AWS 服务 PrivateLink。

中的身份和访问管理 AWS CodeBuild

访问 AWS CodeBuild 需要凭证。这些证书必须具有访问 AWS 资源的权限，例如在 S3 存储桶中存储和检索构建项目以及查看 Amazon CloudWatch Logs 以获取构建。以下各节介绍如何使用 AWS Identity and Access Management(IAM) 和 CodeBuild 帮助保护对资源的访问：

管理您的访问权限概述 AWS CodeBuild 资源

每个 AWS 资源都归一个 AWS 账户所有，创建或访问资源的权限受权限策略的约束。账户管理员可以向 IAM 身份（即：用户、组和角色）附加权限策略。

> **Note**
>
> 账户管理员（或管理员用户）是具有管理员权限的用户。有关更多信息，请参阅《IAM 用户指南》中的 IAM 最佳实操。

在您授予权限时，您要决定谁将获得权限、这些人可以访问的资源以及可以对这些资源执行的操作。

主题

- AWS CodeBuild 资源和运营
- 了解资源所有权
- 管理对 资源的访问
- 指定策略元素：操作、效果和主体

AWS CodeBuild 资源和运营

在中 AWS CodeBuild，主要资源是构建项目。在策略中，您可以使用 Amazon 资源名称 (ARN) 标识策略应用到的资源。构建项目也是资源，且具有相关联的 ARN。有关更多信息，请参阅中的 Amazon 资源名称 (ARN) 和 AWS 服务命名空间。Amazon Web Services 一般参考

资源类型	ARN 格式
构建项目	arn:aws:codebuild: *region-ID* :*account-ID* :project/ *project-name*
构建	arn:aws:codebuild: *region-ID* :*account-ID* :build/*build-ID*
报告组	arn:aws:codebuild: *region-ID* :*account-ID* :report-group/ *report-group-name*
报告	arn:aws:codebuild: *region-ID* :*account-ID* :report/*report-ID*
Fleet	arn:aws:codebuild: *region-ID* :*account-ID* :fleet/*fleet-ID*
所有 CodeBuild 资源	arn:aws:codebuild:*
指定 AWS 区域中指定账户拥有的所有 CodeBuild 资源	arn:aws:codebuild: *region-ID* :*account-ID* :*

> **Important**
>
> 使用预留容量特征时，同一账户内的其他项目可以访问实例集实例中缓存的数据，包括源文件、Docker 层和 buildspec 中指定的缓存目录。这是设计使然，让同一账户内的项目可以共享实例集实例。

> **Note**
>
> 在 ARN 中，大多数 AWS 服务都将冒号 (:) 或正斜杠 (/) 视为同一个字符。但是，在资源模式和规则中 CodeBuild 使用精确匹配。请务必在创建事件模式时使用正确的字符，以使其与资源中的 ARN 语法匹配。

例如，您可以在语句中使用其 ARN 指明特定的构建项目 (*myBuildProject*)，如下所示：

```
"Resource": "arn:aws:codebuild:us-east-2:123456789012:project/myBuildProject"
```

要指定所有资源，或者如果 API 操作不支持 ARN，请在 Resource 元素中使用通配符（*），如下所示：

```
"Resource": "*"
```

某些 CodeBuild API 操作接受多个资源（例如，BatchGetProjects）。要在单个语句中指定多种资源，请使用逗号将它们隔开，如下所示：

```
"Resource": [
  "arn:aws:codebuild:us-east-2:123456789012:project/myBuildProject",
  "arn:aws:codebuild:us-east-2:123456789012:project/myOtherBuildProject"
]
```

CodeBuild 提供了一组使用 CodeBuild 资源的操作。有关列表，请参阅AWS CodeBuild 权限参考。

了解资源所有权

该 AWS 账户拥有在账户中创建的资源，无论谁创建了这些资源。具体而言，资源所有者是对 AWS 资源创建请求进行身份验证的委托人实体（即根账户、用户或 IAM 角色）的账户。以下示例说明了它的工作原理：

- 如果您使用账户的根账户证书创建规则，则您的 AWS 账户就是该 CodeBuild 资源的所有者。 AWS
- 如果您在 AWS 账户中创建用户并向该用户授予创建 CodeBuild 资源的权限，则该用户可以创建 CodeBuild 资源。但是，该用户所属的您的 AWS 账户拥有这些 CodeBuild资源。
- 如果您在 AWS 账户中创建具有创建 CodeBuild 资源权限的 IAM 角色，则任何能够担任该角色的人都可以创建 CodeBuild资源。该角色所属的 AWS 账户拥有这些 CodeBuild资源。

管理对 资源的访问

权限策略规定谁可以访问哪些资源。

> **Note**
>
> 本节讨论如何在 AWS CodeBuild中使用 IAM。这里不提供有关 IAM 服务的详细信息。有关完整的 IAM 文档，请参阅《IAM 用户指南》中的什么是 IAM？。有关 IAM 策略语法和说明的信息，请参阅《IAM 用户指南》中的 AWS IAM 策略参考。

附加到 IAM 身份的策略称为基于身份的策略（IAM 策略）。附加到资源的策略称为基于资源的策略。CodeBuild 支持基于身份的策略，以及针对某些只读 API 的基于资源的策略，以实现跨账户资源共享。

对 S3 存储桶的安全访问

我们强烈建议您在您的 IAM 角色中包含以下权限，以验证与您的 CodeBuild 项目关联的 S3 存储桶是否归您或您信任的人所有。这些权限不包含在 AWS 托管策略和角色中。您必须自行添加。

- `s3:GetBucketAcl`
- `s3:GetBucketLocation`

如果您的项目使用的 S3 存储桶的拥有者发生更改，则必须验证您是否仍拥有存储桶，如果没有，请更新您的 IAM 角色的权限。有关更多信息，请参阅允许用户与之互动 CodeBuild和CodeBuild 允许与其他人互动 AWS 务。

指定策略元素：操作、效果和主体

对于每种 AWS CodeBuild 资源，该服务都定义了一组 API 操作。要授予这些 API 操作的权限，请 CodeBuild 定义一组可在策略中指定的操作。某些 API 操作可能需要多个操作的权限才能执行 API 操作。有关更多信息，请参阅AWS CodeBuild 资源和运营和AWS CodeBuild 权限参考。

以下是基本的策略元素：

- 资源 – 您使用 Amazon 资源名称 (ARN) 来标识策略应用到的资源。
- 操作 – 您可以使用操作关键字来标识要允许或拒绝的资源操作。例如，`codebuild:CreateProject` 权限授予用户执行 `CreateProject` 操作的权限。
- 效果 – 用于指定当用户请求操作时的效果（可以是允许或拒绝）。如果没有显式授予（允许）对资源的访问权限，则隐式拒绝访问。您也可显式拒绝对资源的访问。您可以执行此操作，以确保用户无法访问资源，即使有其他策略授予了访问权限也是如此。
- 主体 – 在基于身份的策略（IAM 策略）中，附加了策略的用户是隐式主体。对于基于资源的策略，您可以指定您希望将权限授予的用户、账户、服务或其他实体。

有关 IAM 策略语法和介绍的更多信息，请参阅《IAM 用户指南》中的 AWS IAM 策略参考。

有关显示所有 CodeBuild API 操作及其适用的资源的表格，请参阅AWS CodeBuild 权限参考。

将基于身份的策略用于 AWS CodeBuild

本主题提供了基于身份的策略的示例，这些示例展示了账户管理员如何将权限策略附加到 IAM 身份（即用户、组和角色），从而授予对 AWS CodeBuild 资源执行操作的权限。

> **Important**
>
> 我们建议您先阅读介绍性主题，这些主题解释了管理 CodeBuild 资源访问权限的基本概念和选项。有关更多信息，请参阅 管理您的访问权限概述 AWS CodeBuild 资源。

主题

- 使用所需的权限 AWS CodeBuild 控制台
- 所需的权限 AWS CodeBuild 连接到 Amazon 弹性容器注册表
- 所需的权限 AWS CodeBuild 用于连接到源提供商的控制台
- AWS 的托管（预定义）策略 AWS CodeBuild
- CodeBuild 托管策略和通知
- CodeBuild 更新到 AWS 托管策略
- Customer-managed 策略示例

以下是一个权限策略示例，仅允许用户在 123456789012 账户的 us-east-2 区域中获取任何以 my 名称开头的构建项目的相关信息：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:BatchGetProjects",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:project/my*"
    }
  ]
}
```

使用所需的权限 AWS CodeBuild 控制台

使用 AWS CodeBuild 控制台的用户必须拥有允许该 AWS 账户描述其他 AWS 资源的最低权限集。您必须拥有来自以下服务的权限：

- AWS CodeBuild
- Amazon CloudWatch
- CodeCommit（如果您要将源代码存储在 AWS CodeCommit 存储库中）
- Amazon Elastic Container Registry (Amazon ECR)（如果您使用的构建环境依赖于 Amazon ECR 存储库中的 Docker 映像）

> **Note**
>
> 截至 2022 年 7 月 26 日，默认 IAM 政策已更新。有关更多信息，请参阅 所需的权限 AWS CodeBuild 连接到 Amazon 弹性容器注册表。

- Amazon Elastic Container Service (Amazon ECS)（如果您使用的构建环境依赖于 Amazon ECR 存储库中的 Docker 映像）
- AWS Identity and Access Management (IAM)
- AWS Key Management Service (AWS KMS)
- Amazon Simple Storage Service（Amazon S3）

如果您创建比必需的最低权限更为严格的 IAM 策略，控制台将无法按预期正常运行。

所需的权限 AWS CodeBuild 连接到 Amazon 弹性容器注册表

自 2022 年 7 月 26 日起， AWS CodeBuild 已更新其亚马逊 ECR 权限的默认 IAM 政策。以下权限已从默认策略中删除：

```
"ecr:PutImage",
"ecr:InitiateLayerUpload",
"ecr:UploadLayerPart",
"ecr:CompleteLayerUpload"
```

对于在 2022 年 7 月 26 日之前创建的 CodeBuild 项目，我们建议您使用以下 Amazon ECR 政策更新您的政策：

```
"Action": [
    "ecr:BatchCheckLayerAvailability",
    "ecr:GetDownloadUrlForLayer",
    "ecr:BatchGetImage"
]
```

有关更新您的策略的更多信息，请参阅允许用户与之互动 CodeBuild。

所需的权限 AWS CodeBuild 用于连接到源提供商的控制台

AWS CodeBuild 控制台使用以下 API 操作连接到源提供商（例如 GitHub 存储库）。

- `codebuild:ListConnectedOAuthAccounts`
- `codebuild:ListRepositories`
- `codebuild:PersistOAuthToken`
- `codebuild:ImportSourceCredentials`

您可以使用 AWS CodeBuild 控制台将源提供程序（例如 GitHub 存储库）与您的构建项目相关联。为此，您必须先将前面的 API 操作添加到与您用于访问 AWS CodeBuild 控制台的用户关联的 IAM 访问策略中。

`ListConnectedOAuthAccounts`、`ListRepositories` 和 `PersistOAuthToken` API 操作不应由您的代码调用。因此，这些 API 操作不包含在 AWS CLI 和 AWS 软件开发工具包中。

AWS 的托管（预定义）策略 AWS CodeBuild

AWS 通过提供由创建和管理的独立 IAM 策略来解决许多常见用例 AWS。这些 AWS 托管策略为常见用例授予必要的权限，因此您可以不必调查需要哪些权限。的托管策略 CodeBuild 还提供在其他服务（例如 IAM、Amazon EC2、Amazon ECR AWS CodeCommit、Amazon ECR、Amazon SNS 和 A CloudWatch mazon Events）中执行操作的权限，以履行已获得相关政策的用户的责任。例如，该`AWSCodeBuildAdminAccess`策略是一个管理级别的用户策略，允许拥有此策略的用户创建和管理项目构建 CloudWatch的事件规则，为项目相关事件（名称前缀为的主题`arn:aws:codebuild:`）创建和管理通知的 Amazon SNS 主题，以及管理中的项目和报告组。 CodeBuild有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略。

以下 AWS 托管策略是特定的，您可以将其附加到账户中的用户 AWS CodeBuild。

AWSCodeBuildAdminAccess

提供完全访问权限， CodeBuild 包括管理 CodeBuild 生成项目的权限。

AWSCodeBuildDeveloperAccess

提供对生成项目的访问权限， CodeBuild 但不允许管理生成项目。

AWSCodeBuildReadOnlyAccess

提供对的只读访问权限 CodeBuild。

要访问 CodeBuild 创建的生成输出项目，您还必须附加名为的 AWS 托管策略`AmazonS3ReadOnlyAccess`。

要创建和管理 CodeBuild 服务角色，还必须附加名为的 AWS 托管策略`IAMFullAccess`。

此外，您还可以创建您自己的自定义 IAM policy，以授予 CodeBuild 操作和资源的相关权限。您可以将这些自定义策略附加到需要这些权限的 用户或组。

主题

- AWSCodeBuildAdminAccess
- AWSCodeBuildDeveloperAccess
- AWSCodeBuildReadOnlyAccess

AWSCodeBuildAdminAccess

该AWSCodeBuildAdminAccess策略提供对构建项目的完全访问权限 CodeBuild，包括管理 CodeBuild 生成项目的权限。此政策仅适用于管理员级别的用户，以授予他们对您 AWS 账户中的 CodeBuild 项目、报告组和相关资源的完全控制权，包括删除项目和报告组的权限。

有关完整的托管策略，请参阅 IAM 托管策略参考 AWSCodeBuildAdminAccess中的。

AWSCodeBuildDeveloperAccess

该AWSCodeBuildDeveloperAccess策略允许访问项目 CodeBuild 和报表组相关资源的所有功能。此政策不允许用户删除 CodeBuild 项目或报告组，或者其他 AWS 服务（例如 CloudWatch 活动）中的相关资源。建议对大多数用户应用此策略。

有关完整的托管策略，请参阅 IAM 托管策略参考 AWSCodeBuildDeveloperAccess中的。

AWSCodeBuildReadOnlyAccess

该AWSCodeBuildReadOnlyAccess政策授予对 CodeBuild 其他 AWS 服务中的相关资源的只读访问权限。将此策略应用于可以查看和运行构建、查看项目和查看报告组但无法对它们作出任何更改的用户。

有关完整的托管策略，请参阅 IAM 托管策略参考 AWSCodeBuildReadOnlyAccess中的。

CodeBuild 托管策略和通知

CodeBuild 支持通知，它可以通知用户生成项目的重要更改。的托管策略 CodeBuild 包括通知功能的策略声明。有关更多信息，请参阅什么是通知？。

只读托管策略中的通知的相关权限

AWSCodeBuildReadOnlyAccess 托管策略包含以下语句，以允许对通知进行只读访问。应用此托管策略的用户可以查看资源的通知，但无法创建、管理或订阅这些通知。

```
  {
       "Sid": "CodeStarNotificationsPowerUserAccess",
       "Effect": "Allow",
       "Action": [
           "codestar-notifications:DescribeNotificationRule"
       ],
       "Resource": "*",
       "Condition" : {
           "ArnLike" : {"codestar-notifications:NotificationsForResource" :
"arn:aws:codebuild:*:*:project/*"}
```

```
        }
    },
    {
        "Sid": "CodeStarNotificationsListAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:ListNotificationRules",
            "codestar-notifications:ListEventTypes",
            "codestar-notifications:ListTargets"
        ],
        "Resource": "*"
    }
```

其他托管策略中的通知的相关权限

AWSCodeBuildDeveloperAccess 托管策略包含以下语句，以允许用户创建、编辑和订阅通知。用户无法删除通知规则或管理资源的标签。

```
    {
        "Sid": "CodeStarNotificationsReadWriteAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:CreateNotificationRule",
            "codestar-notifications:DescribeNotificationRule",
            "codestar-notifications:UpdateNotificationRule",
            "codestar-notifications:Subscribe",
            "codestar-notifications:Unsubscribe"
        ],
        "Resource": "*",
        "Condition" : {
            "ArnLike" : {"codestar-notifications:NotificationsForResource" :
 "arn:aws:codebuild:*:*:project/*"}
        }
    },
    {
        "Sid": "CodeStarNotificationsListAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:ListNotificationRules",
            "codestar-notifications:ListTargets",
            "codestar-notifications:ListTagsforResource",
            "codestar-notifications:ListEventTypes"
        ],
```

```
        "Resource": "*"
    },
    {
        "Sid": "SNSTopicListAccess",
        "Effect": "Allow",
        "Action": [
            "sns:ListTopics"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CodeStarNotificationsChatbotAccess",
        "Effect": "Allow",
        "Action": [
            "chatbot:DescribeSlackChannelConfigurations",
            "chatbot:ListMicrosoftTeamsChannelConfigurations"
          ],
       "Resource": "*"
    }
```

有关 IAM 和通知的更多信息，请参阅用于 AWS CodeStar 通知的 Identity and Access Management。

CodeBuild 更新到 AWS 托管策略

查看 CodeBuild 自该服务开始跟踪这些更改以来 AWS 托管策略更新的详细信息。要获得有关此页面更改的自动提示，请订阅 AWS CodeBuild 《用户指南》文档历史记录 的 RSS 源。

更改	描述	日期
AWSCodeBuildAdminAccess 和 AWSCodeBuildDeveloperAccess – 现有策略的更新	CodeBuild 向这些策略添加了支持 Session Manager 交互式编译调试的ssmmessages:OpenDataChannel 权限。 AWSCodeBuildAdminAccess 和AWSCodeBuildDeveloperAccess 策略现在包括会话管理器会话资源 (arn:aws:ssm:*:*:se	2025 年 12 月 1 日

更改	描述	日期
	ssion/*) 的ssmmessages:OpenDataChannel 操作，以支持在此 WebSocket API 上强制执行 Sigv4。	
AWSCodeBuildAdminAccess 、AWSCodeBuildDeveloperAccess 和 AWSCodeBuildReadOnlyAccess – 现有策略更新	CodeBuild 已将资源更新为这些政策。 AWSCodeBuildAdminAccess 、AWSCodeBuildDeveloperAccess 和 AWSCodeBuildReadOnlyAccess 策略已更改为更新一个现有资源。原始资源 arn:aws:codebuild:* 已更新为 arn:aws:codebuild:*:*:project/* 。	2024 年 11 月 15 日
AWSCodeBuildAdminAccess 、AWSCodeBuildDeveloperAccess 和 AWSCodeBuildReadOnlyAccess – 现有策略更新	CodeBuild 在这些政策中添加了支持 AWS CodeConnections 品牌重塑的资源。 AWSCodeBuildAdminAccess 、AWSCodeBuildDeveloperAccess 和 AWSCodeBuildReadOnlyAccess 策略已更改为添加一项资源，arn:aws:codeconnections:*:*:* 。	2024 年 4 月 18 日

更改	描述	日期
AWSCodeBuildAdminAccess 和 AWSCodeBuildDeveloperAccess – 现有策略更新	CodeBuild 在这些政策中添加了支持在聊天应用程序中使用 Amazon Q Developer 的额外通知类型的权限。 AWSCodeBuildAdminAccess 和 AWSCodeBuildDeveloperAccess 策略已经过更改，来添加权限 chatbot:ListMicrosoftTeamsChannelConfigurations 。	2023 年 5 月 16 日
CodeBuild 已开始跟踪更改	CodeBuild 开始跟踪其 AWS 托管策略的更改。	2023 年 5 月 16 日

Customer-managed 策略示例

本节的用户策略示例介绍如何授予执行 AWS CodeBuild 操作的权限。当您使用 CodeBuild API、AWS 软件开发工具包或 AWS CLI时，这些策略会起作用。当您使用控制台时，您必须授予特定于控制台的其他权限。有关信息，请参阅使用所需的权限 AWS CodeBuild 控制台。

您可以使用以下示例 IAM 策略来限制您的用户和角色的 CodeBuild 访问权限。

主题

- 允许用户获取有关构建项目的信息
- 允许用户获取有关实例集的信息
- 允许用户获取有关报告组的信息
- 允许用户获取有关报告的信息
- 允许用户创建构建项目
- 允许用户创建实例集
- 允许用户创建报告组
- 允许用户删除实例集
- 允许用户删除报告组

- 允许用户删除报告
- 允许用户删除构建项目
- 允许用户获取构建项目名称的列表
- 允许用户更改有关构建项目的信息
- 允许用户更改实例集
- 允许用户更改报告组
- 允许用户获取有关构建的信息
- 允许用户获取构建项目的构建 ID 的列表
- 允许用户获取构建 ID 的列表
- 允许用户获取实例集列表
- 允许用户获取报告组列表
- 允许用户获取报告列表
- 允许用户获取报告组的报告列表
- 允许用户获取报告的测试用例的列表
- 允许用户开始运行构建
- 允许用户尝试停止构建
- 允许用户尝试删除构建
- 允许用户获取有关由管理的 Docker 镜像的信息 CodeBuild
- 允许用户为实例集服务角色添加权限策略
- 允许 CodeBuild 访问 AWS 创建 VPC 网络接口所需的服务
- 使用拒绝语句来防止 AWS CodeBuild 避免与源提供商断开连接

允许用户获取有关构建项目的信息

以下示例策略语句允许用户在 123456789012 账户的 us-east-2 区域中获取任何以名称 my 开头的构建项目的信息：

JSON

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

```
    {
      "Effect": "Allow",
      "Action": "codebuild:BatchGetProjects",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:project/my*"
    }
  ]
}
```

允许用户获取有关实例集的信息

以下示例策略语句允许用户在 123456789012 账户的 us-east-2 区域中获取有关实例集的信息：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:BatchGetFleets",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:fleet/*"
    }
  ]
}
```

允许用户获取有关报告组的信息

以下示例策略语句允许用户在 123456789012 账户的 us-east-2 区域中获取有关报告组的信息：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:BatchGetReportGroups",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:report-group/*"
    }
  ]
```

```
}
```

允许用户获取有关报告的信息

以下示例策略语句允许用户在 123456789012 账户的 us-east-2 区域中获取有关报告的信息：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:BatchGetReports",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:report-group/*"
    }
  ]
}
```

允许用户创建构建项目

以下示例策略声明允许用户使用任意名称创建构建项目，但只能在账户us-east-2所在区域中创建构建项目，123456789012并且只能使用指定的 CodeBuild 服务角色：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:CreateProject",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:project/*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::111122223333:role/CodeBuildServiceRole"
    }
```

```
  ]
}
```

以下示例策略声明允许用户使用任何名称创建构建项目，但只能在账户us-east-2所在区域中创建构建项目，123456789012并且只能使用指定的 CodeBuild 服务角色。它还强制用户只能将指定的服务角色与任何其他服务一起使用， AWS CodeBuild 而不能使用任何其他 AWS 服务。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:CreateProject",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:project/*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::111122223333:role/CodeBuildServiceRole",
      "Condition": {
          "StringEquals": {"iam:PassedToService": "codebuild.amazonaws.com"}
      }
    }
  ]
}
```

允许用户创建实例集

以下示例策略语句允许用户在 123456789012 账户的 us-east-2 区域中创建实例集：

JSON

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

```
      "Action": "codebuild:CreateFleet",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:fleet/*"
    }
  ]
}
```

允许用户创建报告组

以下示例策略语句允许用户在 123456789012 账户的 us-east-2 区域中创建报告组：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:CreateReportGroup",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:report-group/*"
    }
  ]
}
```

允许用户删除实例集

以下示例策略语句允许用户在 123456789012 账户的 us-east-2 区域中删除实例集：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:DeleteFleet",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:fleet/*"
    }
  ]
}
```

允许用户删除报告组

以下示例策略语句允许用户在 123456789012 账户的 us-east-2 区域中删除报告组：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:DeleteReportGroup",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:report-group/*"
    }
  ]
}
```

允许用户删除报告

以下示例策略语句允许用户在 123456789012 账户的 us-east-2 区域中删除报告：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:DeleteReport",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:report-group/*"
    }
  ]
}
```

允许用户删除构建项目

以下示例策略语句允许用户在 123456789012 账户的 us-east-2 区域中删除任何以名称 my 开头的构建项目：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:DeleteProject",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:project/my*"
    }
  ]
}
```

允许用户获取构建项目名称的列表

以下示例策略语句允许用户获取同一账户的构建项目名称的列表：

JSON

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

允许用户更改有关构建项目的信息

以下示例策略语句仅允许用户在 123456789012 账户的 us-east-2 区域中更改有关使用任何名称的构建项目的信息，并且只能使用指定的 AWS CodeBuild 服务角色：

JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:UpdateProject",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:project/*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::111122223333:role/CodeBuildServiceRole"
    }
  ]
}
```

允许用户更改实例集

以下示例策略语句允许用户在 123456789012 账户的 us-east-2 区域中更改实例集：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:UpdateFleet",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:fleet/*"
    }
  ]
}
```

允许用户更改报告组

以下示例策略语句允许用户在 123456789012 账户的 us-east-2 区域中更改报告组：

JSON

```
{
  "Version":"2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:UpdateReportGroup",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:report-group/*"
    }
  ]
}
```

允许用户获取有关构建的信息

以下示例策略语句允许用户在 123456789012 账户的 us-east-2 区域中获取名为 my-build-project 和 my-other-build-project 的构建项目的信息：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:BatchGetBuilds",
      "Resource": [
        "arn:aws:codebuild:us-east-2:111122223333:project/my-build-project",
        "arn:aws:codebuild:us-east-2:111122223333:project/my-other-build-project"
      ]
    }
  ]
}
```

允许用户获取构建项目的构建 ID 的列表

以下示例策略语句允许用户在 123456789012 账户的 us-east-2 区域中获取名为 my-build-project 和 my-other-build-project 的构建项目的构建 ID 列表：

JSON

```
{
  "Version":"2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:ListBuildsForProject",
      "Resource": [
        "arn:aws:codebuild:us-east-2:111122223333:project/my-build-project",
        "arn:aws:codebuild:us-east-2:111122223333:project/my-other-build-project"
      ]
    }
  ]
}
```

允许用户获取构建 ID 的列表

以下示例策略语句允许用户获取同一账户的所有构建 ID 的列表：

JSON

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

允许用户获取实例集列表

以下示例策略语句允许用户在 123456789012 账户的 us-east-2 区域中获取有关实例集的列表：

JSON

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

```
      "Action": "codebuild:ListFleets",
      "Resource": "*"
    }
  ]
}
```

允许用户获取报告组列表

以下示例策略语句允许用户在 123456789012 账户的 us-east-2 区域中获取有关报告组的列表：

JSON

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

允许用户获取报告列表

以下示例策略语句允许用户在 123456789012 账户的 us-east-2 区域中获取有关报告的列表：

JSON

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

允许用户获取报告组的报告列表

以下示例策略语句允许用户在 123456789012 账户的 us-east-2 区域中获取报告组的报告列表：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:ListReportsForReportGroup",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:report-group/*"
    }
  ]
}
```

允许用户获取报告的测试用例的列表

以下示例策略语句允许用户在 123456789012 账户的 us-east-2 区域中获取报告的测试用例列表：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:DescribeTestCases",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:report-group/*"
    }
  ]
}
```

允许用户开始运行构建

以下示例策略语句允许用户在 123456789012 账户的 us-east-2 区域中运行任何以名称 my 开头的构建项目：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:StartBuild",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:project/my*"
    }
  ]
}
```

允许用户尝试停止构建

以下示例策略语句仅允许用户在 123456789012 账户的 us-east-2 区域中尝试停止任何以名称 my 开头的运行中构建项目：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:StopBuild",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:project/my*"
    }
  ]
}
```

允许用户尝试删除构建

以下示例策略语句仅允许用户在 123456789012 账户的 us-east-2 区域中尝试为任何以名称 my 开头的构建项目删除构建：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:BatchDeleteBuilds",
      "Resource": "arn:aws:codebuild:us-east-2:111122223333:project/my*"
    }
  ]
}
```

允许用户获取有关由管理的 Docker 镜像的信息 CodeBuild

以下示例策略语句允许用户获取有关由 CodeBuild 管理的所有 Docker 映像的信息：

JSON

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

允许用户为实例集服务角色添加权限策略

以下示例资源策略语句允许用户为实例集服务角色添加 VPC 权限策略：

JSON

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

```
        {
            "Sid": "CodeBuildFleetVpcCreateNI",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface"
            ],
            "Resource": [
                "arn:aws:ec2:us-west-2:111122223333:subnet/subnet-id-1",
                "arn:aws:ec2:us-west-2:111122223333:security-group/security-
group-id-1",
                "arn:aws:ec2:us-west-2:111122223333:network-interface/*"
            ]
        },
        {
            "Sid": "CodeBuildFleetVpcPermission",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ec2:ModifyNetworkInterfaceAttribute",
                "ec2:DeleteNetworkInterface"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CodeBuildFleetVpcNIPermission",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterfacePermission"
            ],
            "Resource": "arn:aws:ec2:us-west-2:111122223333:network-interface/*",
            "Condition": {
                "StringEquals": {
                    "ec2:Subnet": [
                        "arn:aws:ec2:us-west-2:111122223333:subnet/subnet-id-1"
                    ]
                }
            }
        }
    ]
```

```
}
```

以下示例资源策略语句允许用户为实例集服务角色添加自定义 Amazon 托管的映像（AMI）权限策略：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ec2:DescribeImages",
            "Resource": "*"
        }
    ]
}
```

以下示例信任策略语句允许用户为实例集服务角色添加权限策略：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "CodeBuildFleetVPCTrustPolicy",
      "Effect": "Allow",
      "Principal": {
        "Service": "codebuild.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "111122223333"
        }
      }
    }
  ]
}
```

允许 CodeBuild 访问 AWS 创建 VPC 网络接口所需的服务

以下示例策略声明授予在具有两个子网的 VPC 中创建网络接口的 AWS CodeBuild 权限：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeVpcs"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterfacePermission"
            ],
            "Resource": "arn:aws:ec2:us-west-2:111122223333:network-interface/*",
            "Condition": {
                "StringEquals": {
                    "ec2:AuthorizedService": "codebuild.amazonaws.com"
                },
                "ArnEquals": {
                    "ec2:Subnet": [
                        "arn:aws:ec2:us-west-2:111122223333:subnet/subnet-id-1",
                        "arn:aws:ec2:us-west-2:111122223333:subnet/subnet-id-2"
                    ]
                }
            }
        }
    ]
}
```

使用拒绝语句来防止 AWS CodeBuild 避免与源提供商断开连接

以下示例策略语句使用 Deny 语句阻止 AWS CodeBuild 与源提供商断开连接。它使用 codebuild:DeleteOAuthToken（codebuild:PersistOAuthToken 和 codebuild:ImportSourceCredentials 的倒数）连接到源提供商。有关更多信息，请参阅 所需的权限 AWS CodeBuild 用于连接到源提供商的控制台。

JSON

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

AWS CodeBuild 权限参考

您可以在 AWS CodeBuild 策略中使用 AWS-wide 条件键来表达条件。有关列表，请参阅《IAM 用户指南》中的可用键。

请在策略的 Action 字段中指定这些操作。要指定操作，请在 API 操作名称之前使用 codebuild: 前缀（例如，codebuild:CreateProject 和 codebuild:StartBuild）。要在单个语句中指定多项操作，请使用逗号将它们隔开（例如，"Action": ["codebuild:CreateProject", "codebuild:StartBuild"]）。

使用通配符

您可以在策略的 Resource 字段中指定带或不带通配符（*）的 ARN 作为资源值。您可以使用通配符指定多个操作或资源。例如，codebuild:*指定所有 CodeBuild 动作并codebuild:Batch*指定以单词开头的所有 CodeBuild 动作Batch。以下示例授予对以 my 名称开头的所有构建项目的访问权限：

```
arn:aws:codebuild:us-east-2:123456789012:project/my*
```

CodeBuild API 操作和操作所需的权限

BatchDeleteBuilds

操作：codebuild:BatchDeleteBuilds

删除构建所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

BatchGetBuilds

操作：codebuild:BatchGetBuilds

获取有关构建项目的信息所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

BatchGetProjects

操作：codebuild:BatchGetProjects

获取有关构建项目的信息所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

BatchGetReportGroups

操作：codebuild:BatchGetReportGroups

获取有关报告组的信息所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:report-group/*report-group-name*

BatchGetReports

操作：codebuild:BatchGetReports

获取有关报告的信息所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:report-group/*report-group-name*

BatchPutTestCases[1]

操作：codebuild:BatchPutTestCases

创建或更新测试报告所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:report-group/*report-group-name*

CreateProject

操作：codebuild:CreateProject、iam:PassRole

创建构建项目所必需的。

资源：

- arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*
- arn:aws:iam::*account-ID*:role/*role-name*

CreateReport[1]

操作：codebuild:CreateReport

创建测试报告所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:report-group/*report-group-name*

CreateReportGroup

操作：codebuild:CreateReportGroup

创建报告组所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:report-group/*report-group-name*

CreateWebhook

操作：codebuild:CreateWebhook

创建 Webhook 所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

DeleteProject

操作：codebuild:DeleteProject

删除 CodeBuild 项目所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

DeleteReport

操作：codebuild:DeleteReport

删除报告所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:report-group/*report-group-name*

DeleteReportGroup

操作：codebuild:DeleteReportGroup

删除报告组所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:report-group/*report-group-name*

DeleteSourceCredentials

操作：codebuild:DeleteSourceCredentials

删除一组包含 GitHub 企业服务器或 Bitbucket 存储库凭据信息的SourceCredentialsInfo对象所必需的。 GitHub

资源：*

DeleteWebhook

操作：codebuild:DeleteWebhook

创建 Webhook 所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

DescribeTestCases

操作：codebuild:DescribeTestCases

返回测试用例的分页列表所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:report-group/*report-group-name*

ImportSourceCredentials

操作：codebuild:ImportSourceCredentials

导入一组包含有关 GitHub 企业服务器或 Bitbucke GitHub t 存储库凭证信息的SourceCredentialsInfo对象所必需的。

资源：*

InvalidateProjectCache

操作：codebuild:InvalidateProjectCache

重置项目缓存所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

ListBuildBatches

操作：codebuild:ListBuildBatches

获取构建批处理 ID 的列表所必需的。

资源：*

ListBuildBatchesForProject

操作：codebuild:ListBuildBatchesForProject

获取特定项目的构建批处理 ID 列表所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

ListBuilds

操作：codebuild:ListBuilds

获取构建 ID 的列表所必需的。

资源：*

ListBuildsForProject

操作：codebuild:ListBuildsForProject

获取构建项目的构建 ID 列表所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

ListCuratedEnvironmentImages

操作：codebuild:ListCuratedEnvironmentImages

获取由 AWS CodeBuild管理的所有 Docker 映像的相关信息所必需的。

资源：*（必需，但不引用可寻址的 AWS 资源）

ListProjects

操作：codebuild:ListProjects

获取构建项目名称的列表所必需的。

资源：*

ListReportGroups

操作：codebuild:ListReportGroups

获取报告组列表所必需的。

资源：*

ListReports

操作：codebuild:ListReports

获取报告列表所必需的。

资源：*

ListReportsForReportGroup

操作：codebuild:ListReportsForReportGroup

获取报告组的报告列表所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:report-group/*report-group-name*

RetryBuild

操作：codebuild:RetryBuild

重试构建所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

StartBuild

操作：codebuild:StartBuild

开始运行构建项目所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

StopBuild

操作：codebuild:StopBuild

尝试停止运行中构建项目所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

UpdateProject

操作：codebuild:UpdateProject、iam:PassRole

更改构建项目的相关信息所必需的。

资源：

- arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*
- arn:aws:iam::*account-ID*:role/*role-name*

UpdateProjectVisibility

操作：codebuild:UpdateProjectVisibility、iam:PassRole

更改项目构建的公共可见性所必需的。

资源：

- arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*
- arn:aws:iam::*account-ID*:role/*role-name*

UpdateReport[1]

操作：codebuild:UpdateReport

创建或更新测试报告所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:report-group/*report-group-name*

UpdateReportGroup

操作：codebuild:UpdateReportGroup

更新报告组所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:report-group/*report-group-name*

UpdateWebhook

操作：codebuild:UpdateWebhook

更新 Webhook 所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

[1] 仅用于权限。对于此操作没有 API。

使用标签控制对以下内容的访问 AWS CodeBuild 资源

IAM 策略声明中的条件是语法的一部分，您可以使用该语法为 CodeBuild 基于项目的操作指定权限。您可以创建一个策略（该策略基于与项目关联的标签来允许或拒绝对这些项目执行操作），然后将该策略应用于为管理用户而配置的 IAM 组。有关使用控制台或将标签应用于项目的信息 AWS CLI，请参阅在中创建构建项目 AWS CodeBuild。有关使用 CodeBuild SDK 应用标签的信息，请参阅 CodeBuildAPI 参考中的CreateProject 和标签。有关使用标签控制 AWS 资源访问的信息，请参阅 IAM 用户指南中的使用 AWS 资源标签控制对资源的访问权限。

> **Important**
>
> 使用预留容量特征时，同一账户内的其他项目可以访问实例集实例中缓存的数据，包括源文件、Docker 层和 buildspec 中指定的缓存目录。这是设计使然，让同一账户内的项目可以共享实例集实例。

Example示例 1：根据资源标签限制 CodeBuild 项目操作

以下示例拒绝对用键 Environment、键值 Production 标记的项目执行所有 BatchGetProjects 操作。除了托管用户策略，用户的管理员还必须将此 IAM 策略附加到未经授权的用户。aws:ResourceTag 条件键用于基于其标签控制对资源的访问。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codebuild:BatchGetProjects"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEquals": {
          "aws:ResourceTag/Environment": "Production"
        }
      }
    }
  ]
}
```

Example示例 2：根据请求标签限制 CodeBuild 项目操作

如果请求包含键为 Environment、键值为 Production 的标签，则以下策略拒绝用户对 CreateProject 操作的权限。此外，该策略将阻止这些未经授权的用户修改项目，方法是使用 aws:TagKeys 条件键在请求包含键为 Environment 的标签时不允许 UpdateProject。除了托管用户策略之外，管理员还必须将此 IAM 策略附加到无权执行这些操作的用户。aws:RequestTag 条件键用于控制可以在 IAM 请求中传递哪些标签

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codebuild:CreateProject"
      ],
      "Resource": "*",
      "Condition": {
```

```
        "ForAnyValue:StringEquals": {
          "aws:RequestTag/Environment": "Production"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
        "codebuild:UpdateProject"
      ],
      "Resource": "*",
      "Condition": {
        "ForAnyValue:StringEqualsIgnoreCase": {
          "aws:TagKeys": ["Environment"]
        }
      }
    }
  ]
}
```

Example示例 3：根据资源标签拒绝或允许对报告组执行操作

您可以根据与资源（项目和报告组）关联的 AWS 标签创建允许或拒绝对这些 CodeBuild 资源（项目和报告组）执行操作的策略，然后将这些策略应用于您为管理用户而配置的 IAM 群组。例如，您可以创建一个策略，拒绝对 AWS 标签键`Status`和密钥值为的任何报告组 CodeBuild 执行所有操作`Secret`，然后将该策略应用于您为普通开发者创建的 IAM 群组 (*`Developers`*)。然后，您需要确保在这些带标签的报告组上工作的开发人员不是该常规*`Developers`*组的成员，而是属于未应用限制性策略的其他 IAM 群组 (`SecretDevelopers`)。

以下示例拒绝对标有密钥`Status`和键值的报表组 CodeBuild 执行所有操作`Secret`：

JSON

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

```
        "codebuild:BatchGetReportGroups",
        "codebuild:CreateReportGroup",
        "codebuild:DeleteReportGroup",
        "codebuild:ListReportGroups",
        "codebuild:ListReportsForReportGroup",
        "codebuild:UpdateReportGroup"
      ],
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "aws:RequestedRegion": "us-east-1"
        }
      }
    }
  ]
}
```

Example示例 4：将 CodeBuild 操作限制为 AWSCodeBuildDeveloperAccess 基于资源标签

您可以创建允许对未使用特定标签标记的所有报告组和项目执行 CodeBuild 操作的策略。例如，以下策略为除了使用指定标签标记的报告组和项目以外的所有其他报告组和项目提供与 AWSCodeBuildDeveloperAccess 等效的权限：

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "codebuild:StartBuild",
            "codebuild:StopBuild",
            "codebuild:BatchGet*",
            "codebuild:GetResourcePolicy",
            "codebuild:DescribeTestCases",
            "codebuild:List*",
            "codecommit:GetBranch",
            "codecommit:GetCommit",
            "codecommit:GetRepository",
            "codecommit:ListBranches",
```

```
            "cloudwatch:GetMetricStatistics",
            "events:DescribeRule",
            "events:ListTargetsByRule",
            "events:ListRuleNamesByTarget",
            "logs:GetLogEvents",
            "s3:GetBucketLocation",
            "s3:ListAllMyBuckets"
         ],
         "Resource": "*",
         "Condition": {
            "StringNotEquals": {
               "aws:ResourceTag/Status": "Secret",
               "aws:ResourceTag/Team": "Saanvi"
            }
         }
      }
   ]
}
```

在控制台中查看资源

AWS CodeBuild 控制台需要`ListRepositories`权限才能在您登录的 AWS 地区显示您的 AWS 账户的存储库列表。该控制台还包括一个转到资源功能，可对资源快速执行不区分大小写的搜索。此搜索是在您 AWS 登录所在 AWS 地区的账户中执行的。将显示以下服务中的以下资源：

- AWS CodeBuild：构建项目
- AWS CodeCommit：存储库
- AWS CodeDeploy：应用程序
- AWS CodePipeline：管道

要在所有服务中跨资源执行此搜索，您必须具有如下权限：

- CodeBuild: `ListProjects`
- CodeCommit: `ListRepositories`
- CodeDeploy: `ListApplications`
- CodePipeline: `ListPipelines`

如果您没有针对某个服务的权限，搜索将不会针对该服务的资源返回结果。即使您有权查看资源，但如果特定资源明确 Deny 查看，也不会返回这些资源。

的合规性验证 AWS CodeBuild

我们全新的 AWS 注册体验不是为受监管的工作负载而设计的。如果您正在使用我们的全新 AWS 注册体验，但又想 AWS 用于受监管的工作负载，则可以注册 AWS （高级）或激活适合您 AWS 环境的高级功能。

要了解是否属于特定合规计划的范围，请参阅AWS 服务 “按合规计划划分的范围” ”，然后选择您感兴趣的合规计划。 AWS 服务 有关一般信息，请参阅AWS 合规计划AWS。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的 “下载报告” 中的 “ AWS Artifact。

您在使用 AWS 服务 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。有关您在使用时的合规责任的更多信息 AWS 服务，请参阅AWS 安全文档。

韧性在 AWS CodeBuild

AWS 全球基础设施是围绕 AWS 区域和可用区构建的。 AWS 区域提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络相连。利用可用区，您可以设计和操作在可用区之间无中断地自动实现失效转移的应用程序和数据库。与传统的单个或多个数据中心基础架构相比，可用区具有更高的可用性、容错性和可扩展性。

有关 AWS 区域和可用区的更多信息，请参阅AWS 全球基础设施。

中的基础设施安全 AWS CodeBuild

作为一项托管服务 AWS CodeBuild ，受 AWS 全球网络安全的保护。有关 AWS 安全服务以及如何 AWS 保护基础设施的信息，请参阅AWS 云安全。要使用基础设施安全的最佳实践来设计您的 AWS 环境，请参阅 S AWS ecurity Pillar Well-Architected Fram ework 中的基础设施保护。

您可以使用 AWS 已发布的 API 调用 CodeBuild 通过网络进行访问。客户端必须支持以下内容：

- 传输层安全性协议（TLS）。我们要求使用 TLS 1.2，并建议使用 TLS 1.3。
- 具有完全向前保密 (PFS) 的密码套件，例如 DHE（短暂的）或 ECDHE（椭圆曲线短暂的 Diffie-Hellman）。 Diffie-Hellman大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

在以下位置访问您的源提供商 CodeBuild

对于我们的 GitHub E GitHub nterprise Server，您可以使用个人访问令牌、Secrets Manager 密钥、连接或 OAuth 应用程序来访问源提供商。对于 Bitbucket，您可以使用访问令牌、应用程序密码、API 令牌、Secrets Manager 密钥、连接或 OAuth 应用程序来访问源提供商。

主题

- 在 Secrets Manager 密钥中创建和存储令牌
- GitHub 和 GitHub 企业服务器访问权限 CodeBuild
- 位桶访问权限在 CodeBuild
- GitLab 进入 CodeBuild

在 Secrets Manager 密钥中创建和存储令牌

如果您选择使用 Secrets Manager 存储访问令牌，则可以使用现有密钥连接或创建新密钥。要创建新密钥，请执行以下操作：

AWS 管理控制台

要在中创建 Secrets Manager 密钥 AWS 管理控制台

1. 对于源提供商，请选择 Bitbucket 或 E GitHub nter prise。GitHub
2. 对于凭证，执行以下操作之一：
   - 选择默认来源凭证，使用您账户的默认来源凭证应用于所有项目。
     a. 如果未连接到源提供商，请选择管理默认来源凭证。
     b. 对于凭证类型，请选择除之外的凭据类型。AWS CodeConnections
     c. 对于服务，选择 Secrets Manager，对于密钥，选择新密钥。
     d. 在密钥名称中，输入密钥的名称。
     e. （可选）在密钥描述 - 可选中，为密钥输入描述。
     f. 根据您选择的来源提供商，输入您的令牌或用户名和应用程序密码，然后选择保存。要获取 Bitbucket API 令牌，请改为输入您的 Atlassian 账户电子邮件地址和 API 令牌。
   - 选择自定义来源凭证，以便使用自定义来源凭证来覆盖您账户的默认设置。
     a. 对于凭证类型，请选择除之外的凭据类型。AWS CodeConnections

b. 在连接中，选择创建密钥。

c. 在密钥名称中，输入密钥的名称。

d. （可选）在密钥描述 - 可选中，为密钥输入描述。

e. 根据您选择的源提供商，输入您的令牌或用户名和应用程序密码，然后选择创建。要获取 Bitbucket API 令牌，请改为输入您的 Atlassian 账户电子邮件地址和 API 令牌。

AWS CLI

要在中创建 Secrets Manager 密钥 AWS CLI

- 打开终端（Linux、macOS 或 Unix）或命令提示符（Windows）。使用运行 S AWS CLI ecrets Manager create-secret 命令。

```
aws secretsmanager create-secret --region <aws-region> \
            --name '<secret-name>' \
            --description '<secret-description>' \
            --secret-string '{
                "ServerType":"<server-type>",
                "AuthType":"<auth-type>",
                "Token":"<token>"
                }' \
            --tags Key=codebuild:source,Value='' \
                Key=codebuild:source:type,Value=<type> \
                Key=codebuild:source:provider,Value=<provider>
```

CodeBuild 接受的 Secrets Manager 密钥必须与 CodeBuild 项目位于相同的账户和 AWS 区域中，并且必须采用以下 JSON 格式：

```
{
            "ServerType": ServerType,
            "AuthType: AuthType,
            "Token": string,
            "Username": string // Optional and is only used for Bitbucket app
 password or API token
       }
```

字段	有效值	说明
ServerType	GITHUB GITHUB_ENTERPRISE BITBUCKET	您的 Secrets Manager 密钥的第三方源提供商。
AuthType	PERSONAL_ACCESS_TOKEN BASIC_AUTH	凭证使用的访问令牌类型。对于 GitHub，只有 PERSONAL_ACCESS_TOKEN 有效。BASIC_AUTH 仅对 Bitbucket 应用程序密码或 API 令牌有效。
令牌	*string*	对于 GitHub 我们的 GitHub 企业，这是个人访问令牌。对于 Bitbucket，这是访问令牌、Bitbucket 应用程序密码或 Bitbucket API 令牌。
用户名	*string*	为 BASIC_AUTH 时的 Bitbuck AuthType et 用户名。如果您使用 Bitbucket API 令牌，则这是与您的 Atlassian 账户关联的电子邮件地址。对于其他类型的源提供商，此参数无效。

此外，在密钥上 CodeBuild 使用以下资源标签，以确保在创建或编辑项目时可以轻松选择密钥。

标签密钥	标签值	说明
codebuild:source:provider	GitHub github_enterprise bitbucket	告诉这个秘密是为 CodeBuild 哪个提供者准备的。
codebuild:source:type	personal_access_token basic_auth	告诉 CodeBuild 这个密钥中的访问令牌的类型。

GitHub 和 GitHub 企业服务器访问权限 CodeBuild

对于 GitHub，您可以使用个人访问令牌、OAuth 应用程序、Secrets Manager 密钥或 GitHub 应用程序连接来访问源提供商。对于 E GitHub nterprise Server，您可以使用个人访问令牌、Secrets Manager 密钥或 GitHub 应用程序连接来访问源提供商。

主题

- GitHub GitHub 和 GitHub 企业服务器的应用程序连接
- GitHub 和 GitHub 企业服务器访问令牌
- GitHub OAuth 应用程序

GitHub GitHub 和 GitHub 企业服务器的应用程序连接

您可以使用 GitHub App 进行连接 CodeBuild。 GitHub 通过支持应用程序连接AWS CodeConnections。

源提供程序访问权限使您可以通过订阅使用或教程：配置 CodeBuild-hosted GitHub 操作运行器在中GitHub webhook 事件 CodeBuild使用 CreateWebhook来触发构建。

> **Note**
>
> AWS CodeConnections 在少于. 的地区中可用 CodeBuild。您可以在中使用跨区域连接。CodeBuild在选择加入区域创建的连接不能用于其他区域。有关更多信息，请参阅 AWS CodeConnections 终端节点和限额。

主题

- 步骤 1：创建与 GitHub 应用程序（控制台）的连接
- 步骤 2：向 CodeBuild 项目 IAM 角色授予使用连接的访问权限
- 步骤 3：配置 CodeBuild 为使用新连接
- GitHub 应用程序问题疑难解答

步骤 1：创建与 GitHub 应用程序（控制台）的连接

使用以下步骤使用 CodeBuild 控制台为您的项目添加连接 GitHub。

要创建与的连接 GitHub

- 按照《开发者工具用户指南》中的说明进行操作，创建与的连接 GitHub。

> **Note**
>
> 您可以使用其他账户共享的连接，而不是创建或使用 AWS 账户中的现有连接。有关更多信息，请参阅与 AWS 账户共享连接。

步骤 2：向 CodeBuild 项目 IAM 角色授予使用连接的访问权限

您可以授予 CodeBuild 项目 IAM 角色访问权限，以使用您的连接出售的 GitHub 令牌。

授予 CodeBuild 项目 IAM 角色访问权限

1. 按照项目的说明为您的 CodeBuild 项目创建 IAM 角色。CodeBuild 允许与其他人互动 AWS 务 CodeBuild
2. 按照说明进行操作时，将以下 IAM 策略添加到您的 CodeBuild 项目角色以授予对连接的访问权限。

   JSON

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

```
            "Action": [
                "codeconnections:GetConnectionToken",
                "codeconnections:GetConnection"
            ],
            "Resource": [
                "arn:aws:iam::*:role/Service*"
            ]
        }
    ]
}
```

步骤 3：配置 CodeBuild 为使用新连接

您可以将连接配置为账户级别凭证，并在项目中使用。

AWS 管理控制台

要将连接配置为账户级别的凭证，请参阅 AWS 管理控制台

1. 对于源提供商，请选择GitHub。
2. 对于凭证，执行以下操作之一：
   - 选择默认来源凭证，使用您账户的默认来源凭证应用于所有项目。
     a. 如果您未连接到 GitHub，请选择 “管理默认来源凭据”。
     b. 对于 “凭据类型”，选择 “GitHub 应用程序”。
     c. 在连接中，选择使用现有连接或创建新连接。
   - 选择自定义来源凭证，以便使用自定义来源凭证来覆盖您账户的默认设置。
     a. 对于 “凭据类型”，选择 “GitHub 应用程序”。
     b. 在连接中，选择使用现有连接或创建新连接。

AWS CLI

要将连接配置为账户级别的凭证，请参阅 AWS CLI

- 打开终端（Linux、macOS 或 Unix）或命令提示符（Windows）。 AWS CLI 使用运行import-source-credentials命令，`--token`为您的连接指定`--auth-type`、`--server-type`和。

  使用以下命令：

```
aws codebuild import-source-credentials --auth-type CODECONNECTIONS --server-
type GITHUB --token <connection-arn>
```

您也可以为 CodeBuild 项目设置多个代币。有关更多信息，请参阅 将多个令牌配置为源级别凭证。

GitHub 应用程序问题疑难解答

以下信息可以帮助您解决 GitHub 应用程序的常见问题。

主题

- 安装 AWS 不想要的 GitHub 区域中的应用程序的连接器
- GitHub 应用程序连接无权访问存储库
- 这些区域有： AWS 服务的 IAM 角色缺少必要的 IAM 权限。

安装 AWS 不想要的 GitHub 区域中的应用程序的连接器

问题：您 GitHub 从 GitHub Marketplace 安装了连接 AWS 器，但连接是在不需要的区域创建的。如果您尝试在 GitHub 网站上重新配置应用程序，则该应用程序将无法运行，因为该应用程序已安装在您的 GitHub 帐户中。

可能的原因：该应用程序已安装在您的 GitHub 帐户中，因此您只能重新配置应用程序权限。

推荐的解决方案：您可以在期望区域使用安装 ID 创建新连接。

1. 打开 AWS CodeConnections 控制台，https://console.aws.amazon.com/codesuite/settings/connections然后使用控制 AWS 台导航栏中的区域选择器导航到所需区域。
2. 按照《开发者工具用户指南》中的说明进行操作，创建与的连接 GitHub。

> **Note**
>
> 由于您已经为 GitHub 应用程序安装了 C AWS onnector，因此您可以选择它来代替安装新应用程序。

GitHub 应用程序连接无权访问存储库

问题：使用连接的 AWS 服务（例如 CodeBuild 或 CodePipeline）报告它无权访问存储库或存储库不存在。一些可能的错误消息包括：

- `Authentication required for primary source.`
- `Unable to create webhook at this time. Please try again later.`
- `Failed to create webhook. GitHub API limit reached. Please try again later.`

可能的原因：您可能一直在使用该 GitHub 应用程序，但尚未授予 webhook 权限范围。

推荐的解决方案：要授予所需的权限范围，请按照导航到要查看或修改的 GitHub 应用程序中的说明配置已安装的应用程序。在权限部分下面，您会看到该应用程序没有 webhook 权限，并且您可以选择查看新请求的权限。查看并接受新权限。有关更多信息，请参阅批准 GitHub 应用程序的更新权限。

可能的原因：连接按预期运行，但突然无法访问存储库。

可能的解决方案：首先查看您的授权和安装情况，然后验证 GitHub 应用程序是否已授权并已安装。如果 GitHub 应用程序安装已暂停，则需要将其取消暂停。如果 GitHub 应用程序未获得 UAT（用户访问令牌）连接的授权，或者未为 IAT（安装访问令牌）连接安装该应用程序，则现有连接将无法再使用，您将需要创建一个新连接。请注意，重新安装该 GitHub 应用程序不会恢复与旧安装关联的先前连接。

可能的解决方案：如果连接是 UAT 连接，请确保该连接不是同时使用的，例如在多次 CodeBuild 并发运行的构建中使用。这是因为如果连接刷新了即将到期的令牌，则会 GitHub 立即使先前发行的 UAT 失效。如果您需要将 UAT 连接用于多个并发 CodeBuild 构建，则可以创建多个连接并独立使用每个连接。

可能的解决方案：如果在过去 6 个月内未使用过 UAT 连接，则该连接将失效。 GitHub要修复此问题，请创建新的连接。

可能的原因：您可能在未安装应用程序的情况下使用 UAT 连接。

推荐的解决方案：尽管创建 UAT 连接不需要将连接与 GitHub 应用程序安装相关联，但需要安装才能访问存储库。按照说明查看安装情况，确保 GitHub 应用程序已安装。如果未安装，请导航至GitHub 应用程序页面以安装该应用程序。有关 UAT 访问权限的更多信息，请参阅关于用户访问令牌。

这些区域有： AWS 服务的 IAM 角色缺少必要的 IAM 权限。

问题：您看到以下任何错误消息：

- `Access denied to connection <connection-arn>`

- `Failed to get access token from <connection-arn>`

推荐的解决方案：通常使用与 AWS 服务的连接，例如 CodePipeline 或 CodeBuild。当您为 AWS 服务提供 IAM 角色时，该 AWS 服务可以使用该角色的权限代表您执行操作。确保 IAM 角色具有必要的权限。有关必需的 IAM 权限的更多信息，请参阅向 CodeBuild 项目 IAM 角色授予使用连接的访问权限以及AWS CodeStar 通知的身份和访问管理，以及开发者工具控制台用户指南 AWS CodeConnections 中的开发者工具控制台用户指南。

GitHub 和 GitHub 企业服务器访问令牌

访问令牌先决条件

在开始之前，必须向 GitHub 访问令牌添加适当的权限范围。

对于 GitHub，您的个人访问令牌必须具有以下范围。

- repo：授予私有存储库的完全控制权。
- repo: status：授予对公共和私有仓库提交状态的 read/write 访问权限。
- admin:repo_hook：授予存储库挂钩的完全控制权。如果您的令牌具有 `repo` 范围，则不需要此权限范围。
- admin:org_hook：对组织挂钩授予完全控制权限。仅当您使用组织 webhook 特征时，才需要此范围。

有关更多信息，请参阅网站上的 “了解 OAuth 应用程序的 GitHub 范围”。

如果您使用的是细粒度个人访问令牌，则根据您的使用场景，您的个人访问令牌可能需要以下权限：

- 内容: Read-only: 授予对私有仓库的访问权限。如果使用私有存储库作为源，则需要此权限。
- 提交状态：读写：授予创建提交状态的权限。如果您的项目设置了 webhook，或者启用了报告构建状态特征，则需要此权限。
- Webhook：读写：授予管理 webhook 的权限。如果您的项目设置了 webhook，则需要此权限。
- 拉取请求: Read-only: 授予访问拉取请求的权限。如果您的 webhook 对拉取请求事件设置了 `FILE_PATH` 筛选条件，则需要此权限。
- 管理：读写：如果您使用自托管 GitHub 的 Actions 运行器功能，则需要此权限。 CodeBuild有关更多详细信息，请参阅为存储库创建注册令牌和教程：配置 CodeBuild-hosted GitHub 操作运行器。

> **Note**
>
> 如果要访问组织存储库，请务必将该组织指定为访问令牌的资源所有者。

有关更多信息，请参阅网站上的细粒度个人访问令牌所需的权限。 GitHub

GitHub 使用访问令牌 Connect（控制台）

要使用控制台 GitHub 使用访问令牌将您的项目与之连接，请在创建项目时执行以下操作。有关信息，请参阅创建构建项目（控制台）。

1. 对于源提供商，请选择GitHub。
2. 对于凭证，执行以下操作之一：
   - 选择使用账户凭证将您账户的默认源凭证应用于所有项目。
     a. 如果您未连接到 GitHub，请选择 “管理账户凭证”。
     b. 对于凭证类型，选择个人访问令牌。
   - 如果您选择为服务使用账户级别凭证，请选择要用于存储令牌的服务，然后执行以下操作：
     a. 如果您选择使用 Secrets Manager，则可以选择使用现有密钥连接或创建新密钥，然后选择保存。有关如何创建新密钥的更多信息，请参阅在 Secrets Manager 密钥中创建和存储令牌。
     b. 如果您选择使用 CodeBuild，请输入您的 GitHub 个人访问令牌，然后选择保存。
   - 选择仅为此项目使用覆盖凭证，以便使用自定义源凭证来覆盖账户的凭证设置。
     a. 从填充的凭证列表中，选择个人访问令牌下的选项之一。
     b. 您也可以通过在描述中选择 “创建新的个人访问令牌连接” 来创建新的个人访问令牌。

GitHub 使用访问令牌 (CLI) 连接

按照以下步骤使用访问令牌将您的项目连接到 GitHub 该项目。 AWS CLI 有关 AWS CLI 搭配使用的信息 AWS CodeBuild，请参阅命令行参考。

1. 运行 import-source-credentials 命令：

```
aws codebuild import-source-credentials --generate-cli-skeleton
```

JSON-formatted 数据出现在输出中。将数据复制到安装的本地计算机或实例上某个位置的文件（例如*import-source-credentials.json*）。 AWS CLI 按照下面所示修改复制的数据，并保存您的结果。

```
{
    "serverType": "server-type",
    "authType": "auth-type",
    "shouldOverwrite": "should-overwrite",
    "token": "token",
    "username": "username"
    }
```

替换以下内容：

- *server-type*：必填值。用于此凭证的源提供商。有效值为 GITHUB、BITBUCKET、GITHUB_ENTERPRISE、GITLAB 和 GITLAB_SELF_MANAGED。
- *auth-type*：必填值。用于连接到存储库的身份验证类型。有效值为 OAUTH、BASIC_AUTH、PERSONAL_ACCESS_TOKEN、CODECONNECTIONS 和 SECRETS_MANAGER。对于 GitHub，仅允许使用个人访问令牌。仅允许使用 Bitbucket 应用程序密码或 API 令牌进行 BASIC_AUTH。
- *should-overwrite*: 可选值。设置为 `false` 可防止覆盖存储库源凭证。设置为 `true` 可覆盖存储库源凭证。默认值为 `true`。
- *token*：必填值。对于 GitHub 我们的 GitHub 企业服务器，这是个人访问令牌。对于 Bitbucket，这是个人访问令牌、应用程序密码或 API 令牌。对于身份验证类型 CODECONNECTIONS，这是连接 ARN。对于身份验证类型 SECRETS_MANAGER，这是密钥 ARN。
- *username*: 可选值。对于 GitHub 和 GitHub 企业服务器源提供程序，将忽略此参数。对于 Bitbucket 应用程序的密码，这是你的 Bitbucket 用户名。对于 Bitbucket API 令牌，这是与您的 Atlassian 账户关联的电子邮件地址。

2. 要使用访问令牌连接您的账户，请切换到包含您在步骤 1 中保存的 `import-source-credentials.json` 文件的目录，然后重新运行 import-source-credentials 命令。

```
aws codebuild import-source-credentials --cli-input-json file://import-source-
credentials.json
```

JSON-formatted 数据以亚马逊资源名称 (ARN) 出现在输出中。

```
{
    "arn": "arn:aws:codebuild:region:account-id:token/server-type"
    }
```

> **Note**
>
> 如果您再次使用相同的服务器类型和身份验证类型运行 import-source-credentials 命令，则会更新存储的访问令牌。

在您的账户与访问令牌关联后，您可以使用`create-project`来创建您的 CodeBuild 项目。有关更多信息，请参阅 创建构建项目 (AWS CLI)。

3. 要查看连接的访问令牌，请运行 list-source-credentials 命令。

```
aws codebuild list-source-credentials
```

输出中会出现一个 JSON-formatted `sourceCredentialsInfos`对象：

```
{
        "sourceCredentialsInfos": [
            {
                "authType": "auth-type",
                "serverType": "server-type",
                "arn": "arn"
            }
        ]
    }
```

`sourceCredentialsObject` 包含连接的源凭证信息的列表：

- `authType` 是凭证使用的身份验证类型。这可以是 `OAUTH`、`BASIC_AUTH`、`PERSONAL_ACCESS_TOKEN`、`CODECONNECTIONS` 或 `SECRETS_MANAGER`。
- `serverType` 是源提供商类型。这可以是 `GITHUB`、`GITHUB_ENTERPRISE`、`BITBUCKET`、`GITLAB` 或 `GITLAB_SELF_MANAGED`。
- `arn` 是令牌的 ARN。

4. 要断开与源提供商的连接并删除其访问令牌，请使用其 ARN 运行 delete-source-credentials 命令。

```
aws codebuild delete-source-credentials --arn arn-of-your-credentials
```

JSON-formatted 数据将与已删除凭证的 ARN 一起返回。

```
{
    "arn": "arn:aws:codebuild:region:account-id:token/server-type"
    }
```

GitHub OAuth 应用程序

GitHub 使用 OAuth 连接（控制台）

要使用控制台将您的项目连接到 GitHub 使用 OAuth 应用程序，请在创建项目时执行以下操作。有关信息，请参阅创建构建项目（控制台）。

1. 对于源提供商，请选择GitHub。
2. 对于凭证，执行以下操作之一：
   - 选择使用账户凭证将您账户的默认源凭证应用于所有项目。
     a. 如果您未连接到 GitHub，请选择 “管理账户凭证”。
     b. 对于凭证类型，选择 OAuth 应用程序。
   - 如果您选择为服务使用账户级别凭证，请选择要用于存储令牌的服务，然后执行以下操作：
     a. 如果您选择使用 Secrets Manager，则可以选择使用现有密钥连接或创建新密钥，然后选择保存。有关如何创建新密钥的更多信息，请参阅在 Secrets Manager 密钥中创建和存储令牌。
     b. 如果您选择使用CodeBuild然后选择保存。
   - 选择仅为此项目使用覆盖凭证，以便使用自定义源凭证来覆盖账户的凭证设置。
     a. 从填充的凭证列表中，选择 OAuth 应用程序下的选项之一。
     b. 您还可以通过在描述中选择创建新的 Oauth 应用程序令牌连接来创建新的 OAuth 应用程序令牌。

要查看您的授权的 OAuth 应用程序，请导航到 “开启的应用程序” GitHub，并确认列出了名为 aws-codesu AWS CodeBuild (*region*) ite 拥有的应用程序。

位桶访问权限在 CodeBuild

对于 Bitbucket，您可以使用访问令牌、应用程序密码、API 令牌、OAuth 应用程序或 Bitbucket 连接来访问源提供商。

主题

- Bitbucket 应用程序连接
- Bitbucket 应用程序密码、API 令牌或访问令牌
- Bitbucket OAuth 应用程序

Bitbucket 应用程序连接

你可以使用 Bitbucket 进行连接。 CodeBuild通过 AWS CodeConnections 支持 Bitbucket 应用程序连接。

> **Note**
>
> AWS CodeConnections 在少于. 的地区中可用 CodeBuild。您可以在中使用跨区域连接。CodeBuild在选择加入区域创建的连接不能用于其他区域。有关更多信息，请参阅 AWS CodeConnections 终端节点和限额。

主题

- 步骤 1：创建到 Bitbucket 的连接（控制台）
- 步骤 2：向 CodeBuild 项目 IAM 角色授予使用连接的访问权限
- 步骤 3：配置 CodeBuild 为使用新连接
- Bitbucket OAuth 轮换刷新令

步骤 1：创建到 Bitbucket 的连接（控制台）

使用以下步骤使用 CodeBuild 控制台在 Bitbucket 中为您的项目添加连接。

要创建到 Bitbucket 的连接

- 按照《开发人员工具用户指南》中的说明来创建与 Bitbucket 的连接。

> **Note**
>
> 您可以使用其他账户共享的连接，而不是创建或使用 AWS 账户中的现有连接。有关更多信息，请参阅与 AWS 账户共享连接。

步骤 2：向 CodeBuild 项目 IAM 角色授予使用连接的访问权限

您可以向 CodeBuild 项目 IAM 角色授予使用您的连接出售的 Bitbucket 令牌的访问权限。

授予 CodeBuild 项目 IAM 角色访问权限

1. 按照项目的说明为您的 CodeBuild 项目创建 IAM 角色。CodeBuild 允许与其他人互动 AWS 务 CodeBuild
2. 按照说明进行操作时，将以下 IAM 策略添加到您的 CodeBuild 项目角色以授予对连接的访问权限。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codeconnections:GetConnectionToken",
                "codeconnections:GetConnection"
            ],
            "Resource": [
                "arn:aws:iam::*:role/Service*"
            ]
        }
    ]
}
```

步骤 3：配置 CodeBuild 为使用新连接

您可以将连接配置为账户级别凭证，并在项目中使用。

AWS 管理控制台

要将连接配置为账户级别的凭证，请参阅 AWS 管理控制台

1. 对于源提供商，选择 Bitbucket。
2. 对于凭证，执行以下操作之一：
   - 选择默认来源凭证，使用您账户的默认来源凭证应用于所有项目。
     a. 如果未连接到 Bitbucket，请选择管理默认来源凭证。
     b. 对于 “凭证类型”，选择AWS CodeConnections。
     c. 在连接中，选择使用现有连接或创建新连接。
   - 选择自定义来源凭证，以便使用自定义来源凭证来覆盖您账户的默认设置。
     a. 对于 “凭证类型”，选择AWS CodeConnections。
     b. 在连接中，选择使用现有连接或创建新连接。

AWS CLI

要将连接配置为账户级别的凭证，请参阅 AWS CLI

- 打开终端（Linux、macOS 或 Unix）或命令提示符（Windows）。 AWS CLI 使用运行import-source-credentials命令，`--token`为您的连接指定`--auth-type`、`--server-type`和。

  使用以下命令：

```
aws codebuild import-source-credentials --auth-type CODECONNECTIONS --server-type BITBUCKET --token <connection-arn>
```

有关在 CodeBuild 项目中设置多个令牌的更多信息，请参阅将多个令牌配置为源级别凭证。

Bitbucket OAuth 轮换刷新令

Atlassian 现在对 Bitbucket OAuth 强制使用一次性轮换刷新令牌。每次使用刷新令牌时，Bitbucket 都会使其失效并返回一个新令牌。有关更多信息，请参阅 Atlassian 开发者网站上的 Bitbucket OAuth 一次性刷新令牌 (CHANGE-3052)。

密钥 Manager-stored 凭证所需的操作

如果你在 Secrets Manager 中存储 Bitbucket OAuth 凭证，请向你的 CodeBuild 服务角色添加`secretsmanager:PutSecretValue`权限。有了此权限， CodeBuild 可以在每次刷新令牌后将更新的刷新令牌写回您的密钥。如果没有此权限，令牌刷新后的下一次构建将失败。

以下 IAM 策略声明示例，授予所需的权限：

```
{
    "Effect": "Allow",
    "Action": "secretsmanager:PutSecretValue",
    "Resource": "arn:aws:secretsmanager:*:*:secret:codebuild/*"
}
```

如果您使用该 CodeBuild-managed 选项，则无需进行任何更改。 CodeBuild 自动处理代币轮换。

迁移到 AWS CodeConnections （推荐）

我们建议迁移到以 AWS CodeConnections 进行 Bitbucket 集成。 AWS CodeConnections 使用单独的授权机制，轮换刷新令牌不受影响。您无需管理令牌或添加权限。

要迁移到 AWS CodeConnections，请完成以下步骤：

1. 打开 开发人员工具控制台。选择 “设置”、“连接”，然后选择 “创建连接”。选择 Bitbucket 作为提供商。
2. 在您的 Bitbucket 工作空间中授权 AWS CodeConnections 该应用程序。这需要在 Bitbucket 中拥有管理工作空间的权限。
3. 向您的 CodeBuild 服务角色添加以下权限：`codeconnections:GetConnectionToken`和`codeconnections:GetConnection`。
4. 将连接设置为默认凭据：

   ```
   aws codebuild import-source-credentials --auth-type CODECONNECTIONS --server-type
    BITBUCKET --token <connection-arn>
   ```

5. 所有项目都会自动使用默认凭据开关。单独更新使用每个项目的自定义凭据的项目。使用控制台或 `UpdateProject` API。

Bitbucket 应用程序密码、API 令牌或访问令牌

Bitbucket 应用程序密码已弃用

Atlassian 已弃用 Bitbucket 应用程序密码，转而使用 API 令牌。要继续使用基本身份验证 CodeBuild，请在您的 Atlassian 账户设置中创建 API 令牌。当您将其连接时 CodeBuild，请使用与您的 Atlassian 账户关联的电子邮件地址作为用户名。 CodeBuild 自动使用正确的凭据进行 Git 操作和 Bitbucket API 调用。
有关更多信息，请参阅 Atlassian 网站上的 “使用 API 令牌”。

比特桶 OAuth 代币轮换

如果您使用 Bitbucket OAuth 凭证，我们建议您迁移到。Bitbucket 应用程序连接有关继续将 OAuth 与 Secrets Manager 配合使用时所需更改的信息，请参阅。Bitbucket OAuth 应用程序

先决条件

在开始之前，您必须在 Bitbucket 应用程序密码、API 令牌或访问令牌中添加适当的权限范围。

对于 Bitbucket，您的应用程序密码、API 令牌或访问令牌必须具有以下范围。

- repository:read：授予对授权用户有权访问的所有存储库的读取访问权限。
- pullrequest:read：授予对拉取请求的读取访问权限。如果您的项目具有 Bitbucket webhook，则您的应用程序密码或访问令牌必须具有此权限范围。
- Webhook：授予对 Webhook 的访问权限。如果您的项目具有 webhook 操作，则您的应用程序密码或访问令牌必须具有此权限范围。
- 账户：授予对用户账户信息的读取权限。

有关更多信息，请参阅 Bitbucket 网站上的 Bitbucket 云 REST API 的权限范围和 Bitbucket 云上的 OAuth。

使用应用程序密码或 API 令牌连接 Bitbucket（控制台）

要使用控制台使用应用程序密码或 API 令牌将您的项目连接到 Bitbucket，请在创建项目时执行以下操作。有关信息，请参阅创建构建项目（控制台）。

> **Note**
>
> 如果您使用 API 令牌，请输入与您的 Atlassian 账户关联的电子邮件地址作为用户名。

1. 对于源提供商，选择 Bitbucket。
2. 对于凭证，执行以下操作之一：
   - 选择使用账户凭证将您账户的默认源凭证应用于所有项目。
     a. 如果未连接到 Bitbucket，请选择管理账户凭证。
     b. 对于凭据类型，选择 API 令牌（以前显示为应用程序密码）。
   - 如果您选择为服务使用账户级别凭证，请选择要用于存储令牌的服务，然后执行以下操作：
     a. 如果您选择使用 Secrets Manager，则可以选择使用现有密钥连接或创建新密钥，然后选择保存。有关如何创建新密钥的更多信息，请参阅在 Secrets Manager 密钥中创建和存储令牌。
     b. 如果您选择使用 CodeBuild，请输入您的 Bitbucket 用户名和应用程序密码，然后选择“保存”。要获取 Bitbucket API 令牌，请改为输入您的 Atlassian 账户电子邮件地址和 API 令牌。
   - 选择仅为此项目使用覆盖凭证，以便使用自定义源凭证来覆盖账户的凭证设置。
     a. 从填充的凭据列表中，选择应用程序密码或 API 令牌下的选项之一。
     b. 您还可以通过选择描述中的创建连接链接来创建新的应用程序密码或 API 令牌连接。

使用访问令牌连接 Bitbucket（控制台）

要在控制台中使用访问令牌将您的项目连接到 Bitbucket，请在创建项目时执行以下操作。有关信息，请参阅创建构建项目（控制台）。

1. 对于源提供商，选择 Bitbucket。
2. 对于凭证，执行以下操作之一：
   - 选择使用账户凭证将您账户的默认源凭证应用于所有项目。
     a. 如果未连接到 Bitbucket，请选择管理账户凭证。
     b. 对于凭证类型，选择个人访问令牌。
   - 如果您选择为服务使用账户级别凭证，请选择要用于存储令牌的服务，然后执行以下操作：

a. 如果您选择使用 Secrets Manager，则可以选择使用现有密钥连接或创建新密钥，然后选择保存。有关如何创建新密钥的更多信息，请参阅在 Secrets Manager 密钥中创建和存储令牌。

b. 如果您选择使用 CodeBuild，请输入您的 Bitbucket 个人访问令牌，然后选择 “保存”。

- 选择仅为此项目使用覆盖凭证，以便使用自定义源凭证来覆盖账户的凭证设置。

a. 从填充的凭证列表中，选择个人访问令牌下的选项之一。

b. 您也可以通过在描述中选择 “创建新的个人访问令牌连接” 来创建新的个人访问令牌。

使用应用程序密码或访问令牌 (CLI) 连接 Bitbucket

按照以下步骤使用应用程序密码或访问令牌将您的项目连接到 Bitbucket。 AWS CLI 有关 AWS CLI 搭配使用的信息 AWS CodeBuild，请参阅命令行参考。

1. 运行 import-source-credentials 命令：

```
aws codebuild import-source-credentials --generate-cli-skeleton
```

JSON-formatted 数据出现在输出中。将数据复制到安装的本地计算机或实例上某个位置的文件（例如*import-source-credentials.json*）。 AWS CLI 按照下面所示修改复制的数据，并保存您的结果。

```
{
    "serverType": "BITBUCKET",
    "authType": "auth-type",
    "shouldOverwrite": "should-overwrite",
    "token": "token",
    "username": "username"
    }
```

替换以下内容：

- *server-type*：必填值。用于此凭证的源提供商。有效值为 GITHUB、BITBUCKET、GITHUB_ENTERPRISE、GITLAB 和 GITLAB_SELF_MANAGED。
- *auth-type*：必填值。用于连接到存储库的身份验证类型。有效值为 OAUTH、BASIC_AUTH、PERSONAL_ACCESS_TOKEN、CODECONNECTIONS 和

SECRETS_MANAGER。对于 GitHub，仅允许使用个人访问令牌。仅允许使用 Bitbucket 应用程序密码或 API 令牌进行 BASIC_AUTH。

- *should-overwrite*: 可选值。设置为 `false` 可防止覆盖存储库源凭证。设置为 `true` 可覆盖存储库源凭证。默认值为 `true`。
- *token*：必填值。对于 GitHub 我们的 GitHub 企业服务器，这是个人访问令牌。对于 Bitbucket，这是个人访问令牌、应用程序密码或 API 令牌。对于身份验证类型 CODECONNECTIONS，这是连接 ARN。对于身份验证类型 SECRETS_MANAGER，这是密钥 ARN。
- *username*: 可选值。对于 GitHub 和 GitHub 企业服务器源提供程序，将忽略此参数。对于 Bitbucket 应用程序的密码，这是你的 Bitbucket 用户名。对于 Bitbucket API 令牌，这是与您的 Atlassian 账户关联的电子邮件地址。

2. 要将您的账户与应用程序密码、API 令牌或访问令牌关联起来，请切换到包含您在步骤 1 中保存的`import-source-credentials.json`文件的目录，然后再次运行该import-source-credentials命令。

```
aws codebuild import-source-credentials --cli-input-json file://import-source-credentials.json
```

JSON-formatted 数据以亚马逊资源名称 (ARN) 出现在输出中。

```
{
    "arn": "arn:aws:codebuild:region:account-id:token/server-type"
    }
```

> **Note**
>
> 如果您再次使用相同的服务器类型和身份验证类型运行 import-source-credentials 命令，则会更新存储的访问令牌。

在您的账户使用应用程序密码或 API 令牌关联后，您可以使用`create-project`来创建您的 CodeBuild 项目。有关更多信息，请参阅 创建构建项目 (AWS CLI)。

3. 要查看连接的应用程序密码或访问令牌，请运行 list-source-credentials 命令。

```
aws codebuild list-source-credentials
```

输出中会出现一个 JSON-formatted sourceCredentialsInfos对象：

```
{
        "sourceCredentialsInfos": [
            {
                "authType": "auth-type",
                "serverType": "BITBUCKET",
                "arn": "arn"
            }
        ]
    }
```

sourceCredentialsObject 包含连接的源凭证信息的列表：

- authType 是凭证使用的身份验证类型。这可以是 OAUTH、BASIC_AUTH、PERSONAL_ACCESS_TOKEN、CODECONNECTIONS 或 SECRETS_MANAGER。
- serverType 是源提供商类型。这可以是 GITHUB、GITHUB_ENTERPRISE、BITBUCKET、GITLAB 或 GITLAB_SELF_MANAGED。
- arn 是令牌的 ARN。

4. 要断开与源提供商的连接并删除其应用程序密码或访问令牌，请使用其 ARN 运行 delete-source-credentials 命令。

```
aws codebuild delete-source-credentials --arn arn-of-your-credentials
```

JSON-formatted 数据将与已删除凭证的 ARN 一起返回。

```
{
    "arn": "arn:aws:codebuild:region:account-id:token/server-type"
    }
```

Bitbucket OAuth 应用程序

> **Bitbucket OAuth 代币轮换所需的操作**
>
> Atlassian 现在对 Bitbucket OAuth 强制使用一次性轮换刷新令牌。我们建议迁移到Bitbucket 应用程序连接，这样可以完全省去令牌管理。有关迁移的说明，请参阅迁移到 AWS CodeConnections （推荐）。
> 如果你想继续使用 OAuth 和 Secrets Manager，请为你的 CodeBuild 服务角色添加`secretsmanager:PutSecretValue`权限。有关更多信息，请参阅 密钥 Manager-stored 凭证所需的操作。
> 如果您使用该 CodeBuild-managed 选项，则无需进行任何更改。 CodeBuild 自动处理代币轮换。

使用 OAuth 连接 Bitbucket（控制台）

要在控制台中使用 OAuth 应用程序将您的项目连接到 Bitbucket，请在创建项目时执行以下操作。有关信息，请参阅创建构建项目（控制台）。

1. 对于源提供商，选择 Bitbucket。
2. 对于凭证，执行以下操作之一：
   - 选择使用账户凭证将您账户的默认源凭证应用于所有项目。
     a. 如果未连接到 Bitbucket，请选择管理账户凭证。
     b. 对于凭证类型，选择 OAuth 应用程序。
   - 如果您选择为服务使用账户级别凭证，请选择要用于存储令牌的服务，然后执行以下操作：
     a. 如果您选择使用 Secrets Manager，则可以选择使用现有密钥连接或创建新密钥，然后选择保存。有关创建新密钥的更多信息，请参阅在 Secrets Manager 密钥中创建和存储令牌。
     b. 如果您选择使用CodeBuild然后选择保存。
   - 选择仅为此项目使用覆盖凭证，以便使用自定义源凭证来覆盖账户的凭证设置。
     a. 从填充的凭证列表中，选择 OAuth 应用程序下的选项之一。
     b. 您还可以通过在描述中选择创建新的 Oauth 应用程序令牌连接来创建新的 OAuth 应用程序令牌。

要查看您的授权 OAuth 应用程序，请导航到 Bitbucket 上的应用程序授权，并确认已列出名为 `AWS CodeBuild (region)` 的应用程序。

GitLab 进入 CodeBuild

对于 GitLab，您可以使用 GitLab 连接来访问源提供商。

主题

- Connect t CodeBuild o GitLab

Connect t CodeBuild o GitLab

Connections 允许您使用授权和建立将您的第三方提供商与您的 AWS 资源关联的配置 AWS CodeConnections。要将您的第三方存储库关联为构建项目的源，您应使用连接。

要在中添加 GitLab 或 GitLab 自行管理的源提供商 CodeBuild，您可以选择：

- 使用 CodeBuild 控制台的 “创建构建项目” 向导或 “编辑源” 页面选择GitLab或GitLab 自管理提供程序选项。要添加源提供商，请参阅创建与 GitLab （控制台）的连接。控制台可帮助您创建连接资源。
- 使用 CLI 来创建连接资源，请参阅 创建与 GitLab (CLI) 的连接，以便使用 CLI 创建连接资源。

> **Note**
>
> 您也可以使用开发人员工具控制台，在设置下创建连接。参阅创建连接。

> **Note**
>
> 在中授权安装此连接 GitLab，即表示您授予我们的服务权限，通过访问您的帐户来处理您的数据，并且您可以随时通过卸载应用程序来撤消这些权限。

创建连接到 GitLab

本节介绍如何连接 GitLab 到 CodeBuild。有关 GitLab 连接的更多信息，请参阅 Connect t CodeBuild o GitLab。

开始前的准备工作：

- 您必须已经使用创建了账户 GitLab。

> **Note**
>
> 连接只能访问用于创建并授权连接的账户所拥有的存储库。

> **Note**
>
> 您可以创建与拥有所有者角色的存储库的连接 GitLab，然后可以将该连接与包含诸如之类的资源的存储库一起使用 CodeBuild。对于群组中的仓库，您无需成为群组拥有者。

- 要为构建项目指定来源，您必须已经在上创建了存储库 GitLab。

主题

- 创建与 GitLab （控制台）的连接
- 创建与 GitLab (CLI) 的连接

创建与 GitLab （控制台）的连接

使用以下步骤使用 CodeBuild 控制台为中的项目（存储库）添加连接 GitLab。

> **Note**
>
> 您可以使用其他账户共享的连接，而不是创建或使用 AWS 账户中的现有连接。有关更多信息，请参阅与 AWS 账户共享连接。

创建或编辑您的构建项目

1. 登录 CodeBuild 控制台。
2. 选择下列选项之一。
   - 选择创建构建项目。按照中的创建构建项目（控制台）步骤完成第一个屏幕，然后在 “来源” 部分的 “源提供者” 下选择GitLab。
   - 选择编辑现有构建项目。选择编辑，然后选择源。在 “编辑源” 页面的 “源提供商” 下，选择GitLab。

3. 选择下列选项之一：

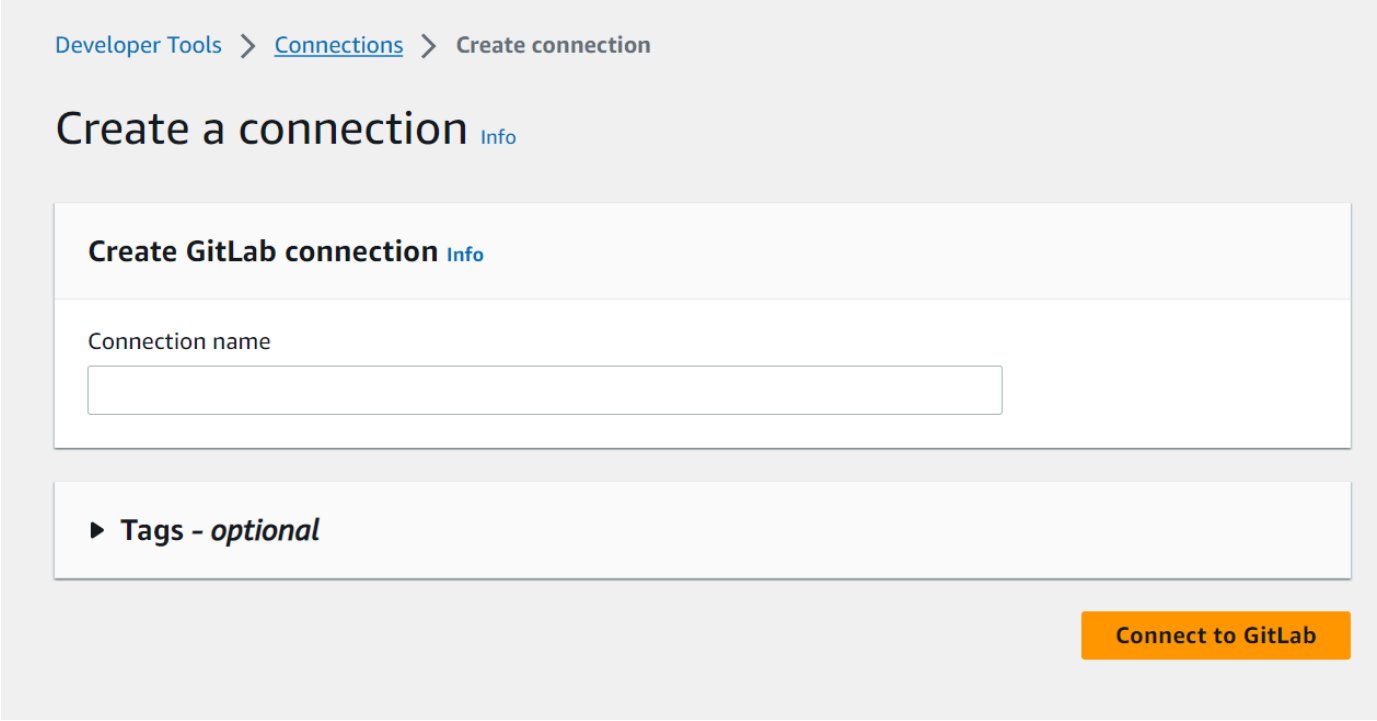
   - 在连接下，选择默认连接。默认连接将默认 GitLab 连接应用于所有项目。
   - 在连接下，选择自定义连接。自定义连接应用的自定义 GitLab连接会覆盖您账户的默认设置。

4. 请执行以下操作之一：

   - 在 “默认连接” 或 “自定义连接” 下，如果您尚未创建与提供商的连接，请选择 “创建新 GitLab连接”。继续执行步骤 5，以便创建连接。
   - 在连接下，如果您已创建到提供程序的连接，请选择该连接。继续执行步骤 10。

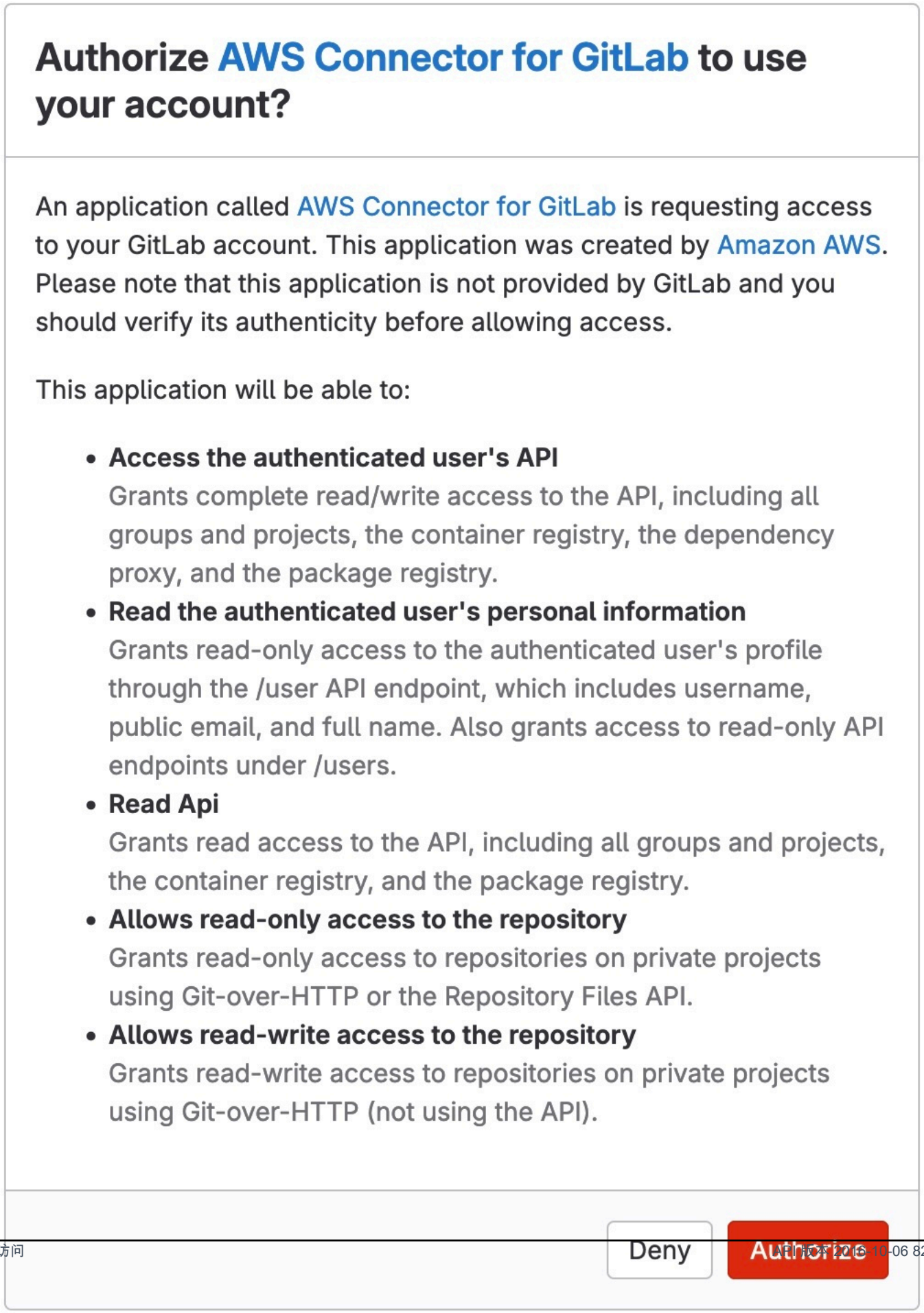
   > **Note**
   >
   > 如果在创建 GitLab 连接之前关闭了弹出窗口，则需要刷新页面。

5. 要创建与 GitLab 存储库的连接，请在 “选择提供者” 下选择GitLab。在连接名称中，输入要创建的连接的名称。选择 Connect to GitLab。



6. GitLab 显示的登录页面时，使用您的凭据登录，然后选择登录。
7. 如果这是您首次授权连接，则会显示一个授权页面，其中包含一条消息，请求授权该连接访问您的 GitLab 账户。

   选择授权。


Authorize AWS Connector for GitLab to use your account?
An application called AWS Connector for GitLab is requesting access to your GitLab account. This application was created by Amazon AWS. Please note that this application is not provided by GitLab and you should verify its authenticity before allowing access.
This application will be able to:
Access the authenticated user's API
Grants complete read/write access to the API, including all groups and projects, the container registry, the dependency proxy, and the package registry.
Read the authenticated user's personal information
Grants read-only access to the authenticated user's profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.
Read Api
Grants read access to the API, including all groups and projects, the container registry, and the package registry.
Allows read-only access to the repository
Grants read-only access to repositories on private projects using Git-over-HTTP or the Repository Files API.
Allows read-write access to the repository
Grants read-write access to repositories on private projects using Git-over-HTTP (not using the API).
Deny
Authorize

8. 浏览器返回到连接控制台页面。在 “GitLab 连接设置” 下，新连接显示在 “连接名称” 中。
9. 选择连接。

   成功创建 GitLab 连接后，将在顶部显示成功横幅。
10. 在创建构建项目页面的默认连接或自定义连接下拉列表中，确保列出了您的连接 ARN。如果未列出，请点击刷新按钮以使其显示。
11. 在 Reposit or GitLab y 中，通过使用命名空间指定项目路径来选择项目名称。例如，对于组级存储库，请按以下格式输入存储库名称：`group-name/repository-name`。有关路径和命名空间的更多信息，请参阅 https://docs.gitlab.com/ee/api/projects.html#get-single-p`path_with_namespace` roject 中的字段。有关中命名空间的更多信息 GitLab，请参阅https://docs.gitlab.com/ee/user/namespace/。

    > **Note**
    >
    > 对于中的群组 GitLab，必须使用命名空间手动指定项目路径。例如，对于组 `mygroup` 中名为 `myrepo` 的存储库，请输入以下内容：`mygroup/myrepo`。您可以在的 URL 中找到带有命名空间的项目路径 GitLab。

12. 在源版本 - 可选中，输入拉取请求 ID、分支、提交 ID、标签或引用以及提交 ID。有关更多信息，请参阅 源版本示例，包含 AWS CodeBuild。

    > **Note**
    >
    > 我们建议您选择看起来不像提交 ID 的 Git 分支名称，例如 `811dd1ba1aba14473856cee38308caed7190c0d` 或 `5392f7`。这可以帮助您避免 Git 签出与实际提交发生冲突。

13. 在 Git 克隆深度 - 可选中，可以创建一个浅克隆，其历史记录会截断至指定数量的提交。如果您需要完整克隆，请选择完整。
14. 在构建状态 - 可选中，如果您希望向源提供商报告构建的开始和完成状态，请选择在构建开始和完成时向源提供商报告构建状态。

    为了能够向源提供商报告构建状态，与源提供商关联的用户必须拥有对存储库的写入权限。如果用户没有写入权限，则无法更新构建状态。有关更多信息，请参阅 源提供商访问权限。

创建与 GitLab (CLI) 的连接

您可以使用 AWS Command Line Interface (AWS CLI) 来创建连接。

为此，请使用 create-connection 命令。

> **Important**
>
> 默认情况下，通过 AWS CLI 或创建的连接 AWS CloudFormation 处于PENDING状态。使用 CLI 或创建连接后 CloudFormation，使用控制台编辑连接以使其处于状态AVAILABLE。

创建连接

- 按照开发者工具控制台用户指南中的创建连接 GitLab (CLI) 中的说明进行操作。

Cross-service 混乱的副手预防

混淆代理问题是一个安全性问题，即不具有某操作执行权限的实体可能会迫使具有更高权限的实体执行该操作。在中 AWS，跨服务模仿可能会导致混乱的副手问题。 Cross-service 当一个服务（调用服务）调用另一个服务（被调用的服务）时，可能会发生模仿行为。可以操纵调用服务，使用其权限以在其他情况下该服务不应有访问权限的方式对另一个客户的资源进行操作。为防止这种情况， AWS 提供可帮助您保护所有服务的数据的工具，而这些服务中的服务主体有权限访问账户中的资源。

我们建议在资源策略中使用aws:SourceArn和aws:SourceAccount全局条件上下文密钥来限制为资源 AWS CodeBuild 提供其他服务的权限。如果您只希望将一个资源与跨服务访问相关联，请使用 aws:SourceArn。如果您想允许该账户中的任何资源与跨服务使用操作相关联，请使用 aws:SourceAccount。

防范混淆代理问题最有效的方法是，使用 aws:SourceArn 全局条件上下文键和资源的完整 ARN。如果不知道资源的完整 ARN，或者正在指定多个资源，请针对 ARN 未知部分使用带有通配符字符（*）的 aws:SourceArn 全局上下文条件键。例如 arn:aws:codebuild:*:*123456789012*:*。

如果 aws:SourceArn 值不包含账户 ID，例如 Amazon S3 存储桶 ARN，您必须使用两个全局条件上下文键来限制权限。

的值aws:SourceArn必须是 CodeBuild 项目 ARN。

以下示例显示了如何在中使用aws:SourceArn和aws:SourceAccount全局条件上下文键 CodeBuild 来防止出现混淆的副手问题。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "codebuild.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceArn": "arn:aws:codebuild:us-
east-1:111122223333:project/MyProject"
                }
            }
        }
    ]
}
```

高级主题

此部分包含几个高级主题，这些主题对于经验更丰富的 AWS CodeBuild 用户很有用。

主题

- 允许用户与之互动 CodeBuild
- CodeBuild 允许与其他人互动 AWS 务
- 使用客户托管密钥加密构建输出
- CodeBuild 使用进行交互 AWS CLI
- 的命令行参考 AWS CodeBuild
- AWS 的 SDK 和工具参考 AWS CodeBuild
- 将此服务与 AWS SDK
- 指定 AWS CodeBuild 端点
- 使用 AWS CodeBuild 用 AWS CodePipeline 测试代码并运行构建
- 使用 AWS CodeBuild 使用 Codecov
- 使用 AWS CodeBuild 和詹金斯在一起
- AWS CodeBuild 与无服务器应用程序一起使用
- 的第三方通知 AWS CodeBuild 适用于 Windows
- 使用 CodeBuild 条件键作为 IAM 服务角色变量来控制构建访问权限
- AWS CodeBuild 条件键

允许用户与之互动 CodeBuild

如果您按照中的通过控制台开始使用步骤 AWS CodeBuild 进行首次访问，则很可能不需要本主题中的信息。但是，当您继续使用时 CodeBuild，您可能需要做一些事情，例如让组织中的其他用户和群组能够与之交互 CodeBuild。

要允许 IAM 用户或群组与之交互 AWS CodeBuild，您必须向他们授予访问权限 CodeBuild。本节介绍了如何使用 IAM 控制台或 AWS CLI完成此操作。

如果您要 CodeBuild 使用 AWS 根账户（不推荐）或 AWS 账户中的管理员用户进行访问，则无需按照以下说明进行操作。

有关 AWS 根账户和管理员用户的信息，请参阅用户指南中的 AWS 账户 root 用户和创建您的第一个 AWS 账户 root 用户和群组。

向 IAM 群组或用户添加 CodeBuild 访问权限（控制台）

1. 使用 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。

   您应该已经使用以下任一 AWS 管理控制台 方式登录了：

   - 您的 AWS 根账号。我们不建议这么做。有关更多信息，请参阅《用户指南》中的 AWS 账户 根用户。
   - 您 AWS 账户中的管理员用户。有关更多信息，请参阅用户指南中的创建您的第一个 AWS 账户 root 用户和群组。
   - 您 AWS 账户中有权执行以下最低限度操作的用户：

   ```
   iam:AttachGroupPolicy
   iam:AttachUserPolicy
   iam:CreatePolicy
   iam:ListAttachedGroupPolicies
   iam:ListAttachedUserPolicies
   iam:ListGroups
   iam:ListPolicies
   iam:ListUsers
   ```

   有关更多信息，请参阅《用户指南》中的 IAM 策略概述。
2. 在导航窗格中，选择策略。
3. 要向 IAM 群组或 IAM 用户添加一组自定义 AWS CodeBuild 访问权限，请直接跳至本过程中的步骤 4。

   要向 IAM 群组或 IAM 用户添加一组默认 CodeBuild 访问权限，请选择策略类型、AWS 托管，然后执行以下操作：

   - 要向添加完全访问权限 CodeBuild，请选中名为的复选框 AWSCodeBuildAdminAccess，选择 “策略操作”，然后选择 “附加”。选中目标 IAM 组或用户旁的框，然后选择附加策略。对名为 AmazonS3ReadOnlyAccess 和 IAMFullAccess 的策略重复执行此操作。
   - 要 CodeBuild 为除构建项目管理之外的所有内容添加访问权限，请选中名为的复选框 AWSCodeBuildDeveloperAccess，选择 “策略操作”，然后选择 “附加”。选中目标 IAM 组或用户旁的框，然后选择附加策略。对名为 AmazonS3ReadOnlyAccess 的策略重复执行此操作。

- 要向添加只读访问权限 CodeBuild，请选中名为的复选框AWSCodeBuildReadOnlyAccess。选中目标 IAM 组或用户旁的框，然后选择附加策略。对名为 AmazonS3ReadOnlyAccess 的策略重复执行此操作。

现在，您已向 IAM 群组或用户添加了一组默认 CodeBuild 访问权限。跳过此过程中的其余步骤。

4. 请选择创建策略。
5. 在创建策略页面上的创建您自己的策略旁，选择选择。
6. 在审查策略页面上，为策略名称输入策略的名称（例如，**CodeBuildAccessPolicy**）。如果您使用其他名称，请确保在本过程中始终使用它。
7. 对于策略文档，输入以下内容，然后选择创建策略。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CodeBuildAccessPolicy",
            "Effect": "Allow",
            "Action": [
                "codebuild:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CodeBuildRolePolicy",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::111122223333:role/role-name"
        },
        {
            "Sid": "CloudWatchLogsAccessPolicy",
            "Effect": "Allow",
            "Action": [
                "logs:FilterLogEvents",
                "logs:GetLogEvents"
            ],
```

```
            "Resource": "*"
        },
        {
            "Sid": "S3AccessPolicy",
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:GetObject",
                "s3:List*",
                "s3:PutObject"
            ],
            "Resource": "*"
        },
        {
            "Sid": "S3BucketIdentity",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketAcl",
                "s3:GetBucketLocation"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> 此策略允许访问所有 CodeBuild 操作和可能的大量 AWS 资源。要限制特定 CodeBuild 操作的权限，请更改 CodeBuild 政策声明`codebuild:*`中的值。有关更多信息，请参阅 Identity and access management。要限制对特定 AWS 资源的访问权限，请更改`Resource`对象的值。有关更多信息，请参阅 Identity and access management。

8. 在导航窗格中，选择组或用户。
9. 在群组或用户列表中，选择要添加 CodeBuild 访问权限的 IAM 群组或 IAM 用户的名称。
10. 对于组，在组设置页面上的权限选项卡上，展开托管策略，然后选择附加策略。

    对于用户，在用户设置页面上的权限选项卡上，选择添加权限。
11. 对于群组，在 “连接策略” 页面上，选择 CodeBuildAccessPolicy，然后选择 “连接策略” 。

对于用户，在添加权限页面上，选择直接附加现有策略。选择 CodeBuildAccessPolicy，选择“下一步：审阅”，然后选择“添加权限”。

为 IAM 群组或用户添加 CodeBuild 访问权限 (AWS CLI)

1. 如前面的过程所述，确保您已 AWS 使用 AWS CLI 与其中一个 IAM 实体对应的访问 AWS 密钥和私有访问密钥进行配置。有关更多信息，请参阅《 AWS Command Line Interface用户指南》中的开始设置AWS Command Line Interface。
2. 要向 IAM 群组或 IAM 用户添加一组自定义 AWS CodeBuild 访问权限，请跳到本过程中的步骤 3。

   要向 IAM 群组或 IAM 用户添加一组默认 CodeBuild 访问权限，请执行以下操作：

   运行以下任一命令，具体取决于您是否要为 IAM 组或用户添加权限：

   ```
   aws iam attach-group-policy --group-name group-name --policy-arn policy-arn

   aws iam attach-user-policy --user-name user-name --policy-arn policy-arn
   ```

   您必须运行该命令三次，将*group-name*或*user-name*替换为 IAM 组名称或用户名，并为以下每项策略 Amazon 资源名称 (ARN) 替换*policy-arn*一次：

   - 要向添加完全访问权限 CodeBuild，请使用以下策略 ARN：
     - arn:aws:iam::aws:policy/AWSCodeBuildAdminAccess
     - arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
     - arn:aws:iam::aws:policy/IAMFullAccess
   - 要 CodeBuild 为除构建项目管理之外的所有内容添加访问权限，请使用以下策略 ARN：
     - arn:aws:iam::aws:policy/AWSCodeBuildDeveloperAccess
     - arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
   - 要向添加只读访问权限 CodeBuild，请使用以下策略 ARN：
     - arn:aws:iam::aws:policy/AWSCodeBuildReadOnlyAccess
     - arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess

   现在，您已向 IAM 群组或用户添加了一组默认 CodeBuild 访问权限。跳过此过程中的其余步骤。

3. 在安装的本地工作站或实例的空目录中 AWS CLI ，创建名为put-group-policy.json或的文件put-user-policy.json。如果您使用其他文件名，请确保在本过程中始终使用它。

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "CodeBuildAccessPolicy",
            "Effect": "Allow",
            "Action": [
                "codebuild:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "CodeBuildRolePolicy",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::111122223333:role/role-name"
        },
        {
            "Sid": "CloudWatchLogsAccessPolicy",
            "Effect": "Allow",
            "Action": [
                "logs:FilterLogEvents",
                "logs:GetLogEvents"
            ],
            "Resource": "*"
        },
        {
            "Sid": "S3AccessPolicy",
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:GetObject",
                "s3:List*",
                "s3:PutObject"
            ],
```

```
            "Resource": "*"
        },
        {
            "Sid": "S3BucketIdentity",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketAcl",
                "s3:GetBucketLocation"
            ],
            "Resource": "*"
        }
    ]
}
```

> **Note**
>
> 此策略允许访问所有 CodeBuild 操作和可能的大量 AWS 资源。要限制特定 CodeBuild 操作的权限，请更改 CodeBuild 政策声明`codebuild:*`中的值。有关更多信息，请参阅 Identity and access management。要限制对特定 AWS 资源的访问权限，请更改相关`Resource`对象的值。有关更多信息，请参阅Identity and access management或特定 AWS 服务的安全文档。

4. 切换到您保存该文件的目录，然后运行以下任一命令。您可以为 `CodeBuildGroupAccessPolicy` 和 `CodeBuildUserAccessPolicy` 使用不同的值。如果您使用其他值，请确保在此处使用它们。

   对于 IAM 组：

   ```
   aws iam put-group-policy --group-name group-name --policy-name
    CodeBuildGroupAccessPolicy --policy-document file://put-group-policy.json
   ```

   对于用户：

   ```
   aws iam put-user-policy --user-name user-name --policy-name
    CodeBuildUserAccessPolicy --policy-document file://put-user-policy.json
   ```

   在前面的命令中，将*group-name*或*user-name*替换为目标 IAM 组或用户的名称。

CodeBuild 允许与其他人互动 AWS 务

如果您按照中的通过控制台开始使用步骤 AWS CodeBuild 进行首次访问，则很可能不需要本主题中的信息。但是，当您继续使用时 CodeBuild，您可能需要做一些事情，例如 CodeBuild 允许与其他 AWS 服务进行交互。

CodeBuild 要允许代表你与依赖 AWS 服务进行交互，你需要一个 AWS CodeBuild 服务角色。您可以使用 CodeBuild 或 AWS CodePipeline 控制台创建 CodeBuild 服务角色。有关信息，请参阅：

- 创建构建项目（控制台）
- 创建使用 CodeBuild （CodePipeline 控制台）的管道
- 向管道添加 CodeBuild 生成操作（CodePipeline 控制台）
- 更改构建项目的设置（控制台）

如果您不打算使用这些控制台，则本节介绍如何 CodeBuild使用 IAM 控制台或 AWS CLI。

> **Important**
>
> CodeBuild 使用服务角色执行所有代表您执行的操作。如果该角色包含用户不应具有的权限，则您可能无意中提升了用户的权限。确保该角色授予最低权限。
> 此页上描述的服务角色包含一项策略，可授予使用 CodeBuild 时所需的最低权限。您可能需要根据使用案例添加额外的权限。

创建 CodeBuild 服务角色（控制台）

1. 使用 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。

   您应该已使用以下任一身份登录到控制台：

   - 您的 AWS 根账户。我们不建议这么做。有关更多信息，请参阅《用户指南》中的 AWS 账户 根用户。
   - 您 AWS 账户中的管理员用户。有关更多信息，请参阅用户指南中的创建您的第一个 AWS 账户 root 用户和群组。
   - 您 AWS 账户中有权执行以下最低限度操作的用户：

     ```
     iam:AddRoleToInstanceProfile
     iam:AttachRolePolicy
     ```

```
iam:CreateInstanceProfile
iam:CreatePolicy
iam:CreateRole
iam:GetRole
iam:ListAttachedRolePolicies
iam:ListPolicies
iam:ListRoles
iam:PassRole
iam:PutRolePolicy
iam:UpdateAssumeRolePolicy
```

有关更多信息，请参阅《用户指南》中的 IAM 策略概述。

2. 在导航窗格中，选择策略。
3. 选择创建策略。
4. 在创建策略页面上，选择 JSON。
5. 对于 JSON 策略，输入以下内容，然后选择查看策略：

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchLogsPolicy",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CodeCommitPolicy",
      "Effect": "Allow",
      "Action": [
        "codecommit:GitPull"
      ],
      "Resource": "*"
    },
    {
```

```
      "Sid": "S3GetObjectPolicy",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": "*"
    },
    {
      "Sid": "S3PutObjectPolicy",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": "*"
    },
    {
      "Sid": "ECRPullPolicy",
      "Effect": "Allow",
      "Action": [
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage"
      ],
      "Resource": "*"
    },
    {
      "Sid": "ECRAuthPolicy",
      "Effect": "Allow",
      "Action": [
        "ecr:GetAuthorizationToken"
      ],
      "Resource": "*"
    },
    {
      "Sid": "S3BucketIdentity",
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketAcl",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
```

```
}
```

> **Note**
>
> 此政策包含允许访问潜在大量 AWS 资源的声明。 AWS CodeBuild 要限制访问特定 AWS 资源，请更改`Resource`数组的值。有关更多信息，请参阅该 AWS 服务的安全文档。

6. 在查看策略页面上，对于策略名称，为策略输入一个名称（例如，**`CodeBuildServiceRolePolicy`**），然后选择创建策略。

> **Note**
>
> 如果您使用其他名称，请确保在本过程中始终使用它。

7. 在导航窗格中，选择角色。
8. 选择创建角色。
9. 在 “创建角色” 页面上，选择 “AWS 服务” CodeBuild，然后选择Next:Permissions。
10. 在 “附加权限策略” 页面上，选择 CodeBuildServiceRolePolicy，然后选择 “下一步：查看” 。
11. 在创建角色并审查页面上，对于角色名称，输入角色的名称（例如，**`CodeBuildServiceRole`**），然后选择创建角色。

创建 CodeBuild 服务角色 (AWS CLI)

1. 如前面的过程所述，确保您已 AWS 使用 AWS CLI 与其中一个 IAM 实体对应的访问 AWS 密钥和私有访问密钥进行配置。有关更多信息，请参阅《 AWS Command Line Interface用户指南》中的开始设置AWS Command Line Interface。
2. 在安装的本地工作站或实例的空目录中 AWS CLI ，创建两个名为`create-role.json`和的文件`put-role-policy.json`。如果您选择了其他文件名称，请确保在整个过程中使用它们。

   `create-role.json`:

   JSON

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

```
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "codebuild.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

> **Note**
>
> 建议您使用 aws:SourceAccount 和 aws:SourceArn 条件键来防止出现混淆代理人问题。例如，您可以使用以下条件块编辑上述信任策略：aws:SourceAccount是 CodeBuild 项目的所有者，aws:SourceArn也是 CodeBuild 项目 ARN。

如果您想将您的服务角色限制为一个 AWS 账户，create-role.json可能如下所示：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "codebuild.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": [
                        "111122223333"
                    ]
                }
            }
        }
    ]
}
```

如果您想将服务角色限制为特定 CodeBuild 项目，`create-role.json`可能如下所示：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "codebuild.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceArn": "arn:aws:codebuild:us-
east-1:111122223333:project/MyProject"
                }
            }
        }
    ]
}
```

> **Note**
>
> 如果您不知道或尚未决定 CodeBuild 项目的名称，并希望对特定的 ARN 模式进行信任政策限制，则可以将 ARN 的该部分替换为通配符 (*)。创建项目后，您可以更新信任策略。

`put-role-policy.json`:

JSON

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

```
      "Sid": "CloudWatchLogsPolicy",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CodeCommitPolicy",
      "Effect": "Allow",
      "Action": [
        "codecommit:GitPull"
      ],
      "Resource": "*"
    },
    {
      "Sid": "S3GetObjectPolicy",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": "*"
    },
    {
      "Sid": "S3PutObjectPolicy",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": "*"
    },
    {
      "Sid": "S3BucketIdentity",
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketAcl",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
```

```
}
```

> **Note**
>
> 此政策包含允许访问潜在大量 AWS 资源的声明。 AWS CodeBuild 要限制访问特定 AWS 资源，请更改`Resource`数组的值。有关更多信息，请参阅该 AWS 服务的安全文档。

3. 切换到您保存上述文件的目录，然后按照这个顺序运行以下两个命令，一次运行一个。您可以为 `CodeBuildServiceRole` 和 `CodeBuildServiceRolePolicy` 使用不同的值，但请务必在此处使用它们。

```
aws iam create-role --role-name CodeBuildServiceRole --assume-role-policy-document
 file://create-role.json
```

```
aws iam put-role-policy --role-name CodeBuildServiceRole --policy-name
 CodeBuildServiceRolePolicy --policy-document file://put-role-policy.json
```

使用客户托管密钥加密构建输出

如果您按照中的通过控制台开始使用步骤 AWS CodeBuild 进行首次访问，则很可能不需要本主题中的信息。但是，在继续使用时 CodeBuild，您可能需要执行诸如加密编译构件之类的操作。

AWS CodeBuild 要加密其构建输出项目，它需要访问 KMS 密钥。默认情况下，在您的 AWS 账户中 CodeBuild 使用 AWS 托管式密钥 适用于 Amazon S3 的。

如果您不想使用 AWS 托管式密钥，则必须自己创建和配置客户管理的密钥。本部分介绍了如何通过 IAM 控制台执行此操作。

有关客户自主管理型密钥的信息，请参阅《AWS KMS 开发人员指南》中的 AWS Key Management Service 概念和创建密钥。

要配置客户管理的密钥以供使用 CodeBuild，请按照AWS KMS 开发人员指南中修改密钥策略的 “如何修改密钥策略” 部分中的说明进行操作。然后将以下语句（介于*### BEGIN ADDING STATEMENTS HERE ###*和之间*### END ADDING STATEMENTS HERE ###*）添加到密钥策略中。为了简洁起见，也为了帮您查找添加语句的位置，此处使用了省略号 (`...`)。请勿删除任何语句，也不要将这些省略号键入密钥策略中。

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:ViaService": "s3.us-east-1.amazonaws.com",
          "kms:CallerAccount": "111122223333"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    }
  ]
}
```

- *region-ID*代表与之关联的 Amazon S3 存储桶 CodeBuild 所在 AWS 区域的 ID（例如，`us-east-1`）。
- *account-ID*代表拥有客户管理密钥的 AWS 账户的 ID。
- *CodeBuild-service-role*代表您在本主题前面创建或确定的 CodeBuild 服务角色的名称。

> **Note**
>
> 要通过 IAM 控制台创建或配置客户管理的密钥，您必须 AWS 管理控制台 首先使用以下任一方式登录：
>
> - 您的 AWS 根账户。我们不建议这么做。有关更多信息，请参阅《用户指南》中的账户根用户。
> - 您 AWS 账户中的管理员用户。有关更多信息，请参阅用户指南中的创建您的第一个 AWS 账户 root 用户和群组。
> - 您 AWS 账户中有权创建或修改客户管理密钥的用户。有关更多信息，请参阅《AWS KMS 开发者指南》中的使用 AWS KMS 控制台所需的权限。

CodeBuild 使用进行交互 AWS CLI

如果您按照中的通过控制台开始使用步骤 AWS CodeBuild 进行首次访问，则很可能不需要本主题中的信息。但是，当您继续使用时 CodeBuild，您可能需要做一些事情，例如允许用户使用来 CodeBuild 代替 CodeBuild 控制台、控制台或 AWS 软件开发工具包（或除此之外使用）进行交互。 AWS CLI CodePipeline

要安装和配置 AWS CLI，请参阅《AWS Command Line Interface 用户指南》 AWS Command Line Interface中的 “使用进行设置”。

安装后 AWS CLI，完成以下任务：

1. 运行以下命令以确认您安装的设备是否 AWS CLI 支持 CodeBuild：

   ```
   aws codebuild list-builds
   ```

   如果成功，将在输出中显示与以下内容类似的信息：

   ```
   {
     "ids": []
   }
   ```

   空方括号表示您尚未运行任何构建。

2. 如果输出一个错误，您必须卸载当前版本的 AWS CLI ，然后安装最新版本。有关更多信息，请参阅《 AWS CLI用户指南》中的卸载AWS Command Line Interface 和安装 AWS Command Line Interface。

的命令行参考 AWS CodeBuild

AWS CLI 提供了用于自动化的 AWS CodeBuild命令。使用本主题中的信息作为《AWS Command Line Interface 用户指南》和《适用于 AWS CodeBuild的AWS CLI 参考》的补充。

不是您要找的内容？ 如果要使用 AWS 软件开发工具包进行调用 CodeBuild，请参阅。AWS SDK 和工具参考

要使用本主题中的信息，您应该已经安装 AWS CLI 并对其进行了配置，以便与一起使用 CodeBuild，如中所述CodeBuild 使用进行交互 AWS CLI。

要使用 AWS CLI 来指定其终端节点 CodeBuild，请参阅指定 AWS CodeBuild 端点（ AWS CLI)。

运行此命令以获取 CodeBuild 命令列表。

```
aws codebuild help
```

运行此命令以获取有关 CodeBuild 命令的信息，其中*command-name*是命令的名称。

```
aws codebuild command-name help
```

CodeBuild 命令包括：

- `batch-delete-builds`：删除一个或多个内部版本 CodeBuild。有关更多信息，请参阅 删除构建 (AWS CLI)。
- `batch-get-builds`：获取有关 CodeBuild 中的多个构建的信息。有关更多信息，请参阅 查看构建详细信息（ AWS CLI ）。
- `batch-get-projects`：获取有关一个或多个指定构建项目的信息。有关更多信息，请参阅 查看构建项目的详细信息 (AWS CLI)。
- `create-project`：创建构建项目。有关更多信息，请参阅 创建构建项目 (AWS CLI)。
- `delete-project`：删除构建项目。有关更多信息，请参阅 删除构建项目 (AWS CLI)。
- `list-builds`：列出 CodeBuild 中的构建的 Amazon 资源名称 (ARN)。有关更多信息，请参阅 查看构建 ID 的列表（ AWS CLI ）。

- `list-builds-for-project`：获取与指定构建项目相关联的构建 ID 的列表。有关更多信息，请参阅 查看构建项目的构建 ID 列表 (AWS CLI)。
- `list-curated-environment-images`: 获取由管理的 Docker 映像列表 CodeBuild ，您可以将其用于构建。有关更多信息，请参阅 Docker 镜像由 CodeBuild。
- `list-projects`：获取构建项目名称的列表。有关更多信息，请参阅查看构建项目名称的列表 (AWS CLI)。
- `start-build`：开始运行构建。有关更多信息，请参阅 运行构建 (AWS CLI)。
- `stop-build`：尝试停止运行指定的构建。有关更多信息，请参阅 停止构建 (AWS CLI)。
- `update-project`：更改指定构建项目的信息。有关更多信息，请参阅 更改构建项目的设置 (AWS CLI)。

AWS 的 SDK 和工具参考 AWS CodeBuild

要使用其中一个 AWS SDK 或工具实现自动化 AWS CodeBuild，请参阅以下资源。

如果您想使用 AWS CLI 来运行 CodeBuild，请参阅命令行参考。

支持 AWS 软件开发工具包和工具 AWS CodeBuild

以下 AWS SDK 和工具支持 CodeBuild：

- 适用于 C++ 的AWS 开发工具包。有关更多信息，请参阅 C++ AWS 开发工具包 API 参考的 Aws:: CodeBuild 命名空间部分。
- 适用于 Go 的AWS 开发工具包。有关更多信息，请参阅《适用于 Go 的AWS SDK API 参考》的 codebuild 部分。
- AWS 适用于 Java 的 SDK。有关更多信息，请参阅适用于 Java 的AWS SDK API 参考的 `com.amazonaws.services.codebuild` 和 `com.amazonaws.services.codebuild.model` 部分。
- 浏览器 JavaScript 中的AWS 软件开发工具包和用于 JavaScript 中的AWS 软件开发工具包 Node.js。欲了解更多信息，请参阅 Class: AWS。CodeBuild适用于 JavaScript API 参考的 S AWS DK 部分。
- 适用于 .NET 的AWS 开发工具包。有关更多信息，请参阅.NET AWS 开发工具包 API 参考中的Amazon.CodeBuild和Amazon.CodeBuild.Model命名空间部分。
- 适用于 PHP 的AWS 开发工具包。有关更多信息，请参阅适用于 PHP 的 CodeBuild 软件开发工具包 API 参考的命名空间 Aws\AWS 部分。

- 适用于 Python 的AWS 开发工具包 (Boto3)。有关更多信息，请参阅 Boto 3 文档CodeBuild的 部分。
- 适用于 Ruby 的AWS 开发工具包。有关更多信息，请参阅适用于 Ruby 的 CodeBuild 软件开发工具包 API 参考的模块：Aws::AWS 部分。
- 用于 AWS ... 的 PowerShell工具 有关更多信息，请参阅《 PowerShell Cmdlet AWS 工具参考》AWS CodeBuild部分。

将此服务与 AWS SDK

AWS 软件开发套件 (SDK) 可用于许多流行的编程语言。每个软件开发工具包都提供 API、代码示例和文档，使开发人员能够更轻松地以其首选语言构建应用程序。

SDK 文档	代码示例
适用于 C++ 的 AWS SDK	适用于 C++ 的 AWS SDK 代码示例
AWS CLI	AWS CLI 代码示例
适用于 Go 的 AWS SDK	适用于 Go 的 AWS SDK 代码示例
适用于 Java 的 AWS SDK	适用于 Java 的 AWS SDK 代码示例
适用于 JavaScript 的 AWS SDK	适用于 JavaScript 的 AWS SDK 代码示例
适用于 Kotlin 的 AWS SDK	适用于 Kotlin 的 AWS SDK 代码示例
适用于 .NET 的 AWS SDK	适用于 .NET 的 AWS SDK 代码示例
适用于 PHP 的 AWS SDK	适用于 PHP 的 AWS SDK 代码示例
AWS Tools for PowerShell	AWS Tools for PowerShell 代码示例
AWS 适用于 Python 的 SDK 版本	AWS 适用于 Python 的 SDK v4 代码示例
适用于 Python (Boto3) 的 AWS SDK	适用于 Python (Boto3) 的 AWS SDK 代码示例
适用于 Ruby 的 AWS SDK	适用于 Ruby 的 AWS SDK 代码示例
适用于 Rust 的 AWS SDK	适用于 Rust 的 AWS SDK 代码示例

SDK 文档	代码示例
适用于 SAP ABAP 的 AWS SDK	适用于 SAP ABAP 的 AWS SDK 代码示例
适用于 Swift 的 AWS SDK	适用于 Swift 的 AWS SDK 代码示例

有关特定于此服务的示例，请参阅CodeBuild 使用 AWS SDK 的代码示例。

> **示例可用性**
>
> 找不到所需的内容？ 通过使用此页面底部的提供反馈链接请求代码示例。

指定 AWS CodeBuild 端点

您可以使用 AWS Command Line Interface (AWS CLI) 或其中一个 AWS 软件开发工具包来指定使用的 AWS CodeBuild终端节点。每个可用区域都有一个终端节点。 CodeBuild 除了一个区域端点之外，四个区域还有联邦信息处理标准 (FIPS) 端点。有关联邦信息处理标准端点的更多信息，请参阅 FIPS 140-2 概述。

可以选择指定端点。如果您没有明确告知要使用 CodeBuild 哪个终端节点，则该服务将使用与您的 AWS 账户使用的区域关联的终端节点。 CodeBuild永远不会默认为 FIPS 端点。如果您希望使用 FIPS 终端节点，则必须使用以下方法之一将 CodeBuild 与其关联。

> **Note**
>
> 您可以使用别名或区域名称来使用 AWS SDK 指定终端节点。如果您使用 AWS CLI，则必须使用完整的端点名称。

有关可与之一起使用的终端节点 CodeBuild，请参阅CodeBuild 区域和终端节点。

主题

- 指定 AWS CodeBuild 端点（AWS CLI）
- 指定 AWS CodeBuild 端点（AWS SDK）

指定 AWS CodeBuild 端点（AWS CLI)

您可以使用 AWS CLI 来指定端点，通过在任何 CodeBuild 命令中使用`--endpoint-url`参数来访问该端点。 AWS CodeBuild 例如，运行此命令以获取在美国东部（弗吉尼亚州北部）中使用联邦信息处理标准 (FIPS) 端点的项目构建名称列表：

```
aws codebuild list-projects --endpoint-url https://codebuild-fips.us-
east-1.amazonaws.com
```

在端`https://`点的开头加上。

该`--endpoint-url` AWS CLI 参数适用于所有 AWS 服务。有关此 AWS CLI 参数和其他参数的更多信息，请参阅AWS CLI 命令参考。

指定 AWS CodeBuild 端点（AWS SDK)

您可以使用 S AWS DK 来指定访问的终端节点。 AWS CodeBuild 尽管此示例使用AWS 适用于 Java 的软件开发工具包，但您可以使用其他 AWS SDK 指定终端节点。

在构造 AWSCodeBuild 客户端时使用该`withEndpointConfiguration`方法。以下是要使用的格式：

```
AWSCodeBuild awsCodeBuild = AWSCodeBuildClientBuilder.standard().
    withEndpointConfiguration(new AwsClientBuilder.EndpointConfiguration("endpoint",
 "region")).
    withCredentials(new AWSStaticCredentialsProvider(sessionCredentials)).
    build();
```

有关信息`AWSCodeBuildClientBuilder`，请参阅 Class AWSCodeBuildClientBuilder 。

在 `withCredentials` 中使用的凭证的类型必须为 `AWSCredentialsProvider`。有关更多信息，请参阅使用 AWS 证书。

请勿`https://`在端点的开头加入。

如果您希望指定非 FIPS 端点，则可以使用区域而非实际端点。例如，要在美国东部（弗吉尼亚州北部）区域中指定端点，您可以使用 `us-east-1` 而不是完整的端点名称 `codebuild.us-east-1.amazonaws.com`。

如果您要指定 FIPS 端点，可以使用别名来简化代码。只有 FIPS 端点有别名。其他端点必须使用其区域或完整名称指定。

下表列出了四个可用 FIPS 端点的各自的别名。

区域名称	区域	端点	Alias
美国东部（弗吉尼亚州北部）	us-east-1	codebuild-fips.us-east-1.amazonaws.com	us-east-1-fips
美国东部（俄亥俄州）	us-east-2	codebuild-fips.us-east-2.amazonaws.com	us-east-2-fips
美国西部（北加利福尼亚）	us-west-1	codebuild-fips.us-west-1.amazonaws.com	us-west-1-fips
美国西部（俄勒冈州）	us-west-2	codebuild-fips.us-west-2.amazonaws.com	us-west-2-fips

要指定使用美国西部（俄勒冈州）区域中的 FIPS 端点，请使用别名：

```
AWSCodeBuild awsCodeBuild = AWSCodeBuildClientBuilder.standard().
    withEndpointConfiguration(new AwsClientBuilder.EndpointConfiguration("us-west-2-
fips", "us-west-2")).
    withCredentials(new AWSStaticCredentialsProvider(sessionCredentials)).
    build();
```

指定使用美国东部（弗吉尼亚州北部）区域中的非 FIPS 端点：

```
AWSCodeBuild awsCodeBuild = AWSCodeBuildClientBuilder.standard().
    withEndpointConfiguration(new AwsClientBuilder.EndpointConfiguration("us-east-1",
 "us-east-1")).
```

```
    withCredentials(new AWSStaticCredentialsProvider(sessionCredentials)).
    build();
```

指定使用亚太地区（孟买）区域中的非 FIPS 端点：

```
AWSCodeBuild awsCodeBuild = AWSCodeBuildClientBuilder.standard().
    withEndpointConfiguration(new AwsClientBuilder.EndpointConfiguration("ap-south-1",
 "ap-south-1")).
    withCredentials(new AWSStaticCredentialsProvider(sessionCredentials)).
    build();
```

使用 AWS CodeBuild 用 AWS CodePipeline 测试代码并运行构建

您可以使用测试代码并使用运行构建 AWS CodePipeline ，从而实现发布过程自动化 AWS CodeBuild。

下表列出了可用于执行这些操作的任务和方法。本主题不介绍如何使用 AWS 开发工具包完成这些任务。

Task	可用方法	本主题中介绍的方法
使用创建持续交付 (CD) 管道 CodePipeline ，通过以下方式自动构建 CodeBuild	• CodePipeline 控制台 • AWS CLI • AWS 软件开发工具包	• 使用 CodePipeline 控制台 • 使用 AWS CLI • 您可以调整本主题中的信息以使用 AWS 开发工具包。有关更多信息，请参阅《适用于 Amazon Web Services 的工具》的 SDK 部分中您的编程语言对应的 `create-pipeline` 操作文档，或参阅《AWS CodePipeline API 参考》中的 `CreatePipeline` 。
使用 CodeBuild 中的现有管道添加测试和构建自动化 CodePipeline	• CodePipeline 控制台 • AWS CLI • AWS 软件开发工具包	• 使用 CodePipeline 控制台添加生成自动化 • 使用 CodePipeline 控制台添加测试自动化 • 为此 AWS CLI，您可以调整本主题中的信息，以创建包含 CodeBuild 生成操作或测试操作的管道。有关更多信息，请参阅《AWS CodePipeline 用户指南》中的编辑CodePipeline 管道 (AWS CLI) 和管道结构参考。

Task	可用方法	本主题中介绍的方法
		• 您可以调整本主题中的信息以使用 AWS 开发工具包。有关更多信息，请参阅《适用于 Amazon Web Services 的工具》的 SDK 部分中您的编程语言对应的 `update-pipeline` 操作文档，或参阅《AWS CodePipeline API 参考》中的 `UpdatePipeline` 。

主题

- 先决条件
- 创建使用 CodeBuild （CodePipeline 控制台）的管道
- 创建使用 CodeBuild (AWS CLI) 的管道
- 向管道添加 CodeBuild 生成操作（CodePipeline 控制台）
- 向管道添加 CodeBuild 测试操作（CodePipeline 控制台）

先决条件

1. 回答计划构建中的问题。
2. 如果您使用用户 CodePipeline 而不是 AWS 根账户或管理员用户进行访问，请将名为的托管策略附加`AWSCodePipelineFullAccess`到该用户（或该用户所属的 IAM 组）。不建议使用 AWS 根账户。此策略向用户授予在 CodePipeline 中创建管道的权限。有关更多信息，请参阅《用户指南》中的附加托管式策略。

   > **Note**
   >
   > 向该用户（或该用户所属的 IAM 组）附加策略的 IAM 实体在 IAM 中必须拥有附加策略的权限。有关更多信息，请参阅《用户指南》中的委派权限来管理 IAM 用户、组和凭证。

3. 如果您的 AWS 账户中还没有可用的 CodePipeline 服务角色，请创建服务角色。 CodePipeline 使用此服务角色与其他 AWS 服务进行交互 AWS CodeBuild，包括代表您进行交互。例如，要使用创建 CodePipeline 服务角色，请运行 IAM `create-role` 命令： AWS CLI

   对于 Linux、macOS 或 Unix：

   ```
   aws iam create-role --role-name AWS-CodePipeline-CodeBuild-
   Service-Role --assume-role-policy-document '{"Version":
   ```

```
 "2012-10-17",       "Statement":{"Effect":"Allow","Principal":
{"Service":"codepipeline.amazonaws.com"},"Action":"sts:AssumeRole"}}'
```

对于 Windows：

```
aws iam create-role --role-name AWS-CodePipeline-CodeBuild-Service-Role --assume-
role-policy-document "{\"Version\":\"2012-10-17\",       \"Statement\":{\"Effect
\":\"Allow\",\"Principal\":{\"Service\":\"codepipeline.amazonaws.com\"},\"Action\":
\"sts:AssumeRole\"}}"
```

> **Note**
>
> 创建此 CodePipeline 服务角色的 IAM 实体必须在 IAM 中拥有创建服务角色的权限。

4. 创建 CodePipeline 服务角色或确定现有服务角色后，如果默认 CodePipeline 服务角色策略尚未成为该角色策略的一部分，则必须按照查看AWS CodePipeline 用户指南中的默认 CodePipeline 服务角色策略中所述将默认服务角色策略添加到该服务角色。

> **Note**
>
> 添加此 CodePipeline 服务角色策略的 IAM 实体必须在 IAM 中拥有向服务角色添加服务角色策略的权限。

5. 创建源代码并将其上传到 CodeBuild 和支持的存储库类型 CodePipeline，例如 Amazon S3 CodeCommit、Bitbucket 或 GitHub。源代码应包含构建规范文件，不过您也可在本主题稍后部分定义构建项目时，声明一个构建规范文件。有关更多信息，请参阅Buildspec 参考。

> **Important**
>
> 如果您计划使用管道来部署已构建的源代码，则构建输出构件必须与您使用的部署系统兼容。
>
> - 有关信息 OpsWorks，请参阅《OpsWorks 用户指南》 OpsWorks中的 “应用程序源代码” 和 “ CodePipeline 搭配使用”。

创建使用 CodeBuild （ CodePipeline 控制台）的管道

使用以下过程创建用于构建和部署源代码的管道。 CodeBuild

要创建仅测试源代码的管道：

- 执行以下步骤来创建管道，然后从管道中删除构建和测试阶段。然后使用本主题中的 向管道添加 CodeBuild 测试操作（CodePipeline 控制台）步骤，将使用 CodeBuild 的测试操作添加到管道。
- 使用本主题中的其他步骤之一来创建管道，然后使用本主题中的 向管道添加 CodeBuild 测试操作（CodePipeline 控制台）步骤，将使用 CodeBuild 的测试操作添加到管道。

使用中的创建管道向导 CodePipeline 来创建使用以下内容的管道 CodeBuild

1. 使用以下 AWS 管理控制台 方式登录：

   - 您的 AWS 根账号。我们不建议这么做。有关更多信息，请参阅《用户指南》中的账户根用户。
   - 您 AWS 账户中的管理员用户。有关更多信息，请参阅用户指南中的创建您的第一个 AWS 账户 root 用户和群组。
   - 您 AWS 账户中有权使用以下最低限度操作的用户：

   ```
   codepipeline:*
   iam:ListRoles
   iam:PassRole
   s3:CreateBucket
   s3:GetBucketPolicy
   s3:GetObject
   s3:ListAllMyBuckets
   s3:ListBucket
   s3:PutBucketPolicy
   codecommit:ListBranches
   codecommit:ListRepositories
   codedeploy:GetApplication
   codedeploy:GetDeploymentGroup
   codedeploy:ListApplications
   codedeploy:ListDeploymentGroups
   elasticbeanstalk:DescribeApplications
   elasticbeanstalk:DescribeEnvironments
   lambda:GetFunctionConfiguration
   lambda:ListFunctions
   opsworks:DescribeStacks
   opsworks:DescribeApps
   opsworks:DescribeLayers
   ```

2. 在处打开 AWS CodePipeline 控制台https://console.aws.amazon.com/codesuite/codepipeline/home。

3. 在 AWS 区域选择器中，选择您的构建项目 AWS 资源所在的 AWS 区域。这必须 CodeBuild 是受支持的地 AWS 区。有关更多信息，请参阅Amazon Web Services 一般参考中的 AWS CodeBuild。
4. 创建管道。如果显示 CodePipeline 信息页面，请选择 “创建管道” 。如果显示管道页面，请选择创建管道。
5. 在步骤 1：选择管道设置页面上，对于管道名称，输入管道的名称（例如，**CodeBuildDemoPipeline**）。如果您选择其他名称，请确保在本过程中始终使用它。
6. 对于角色名称，执行以下操作之一：

   选择新服务角色，然后在角色名称中，输入新服务角色的名称。

   选择 Existing service role (现有服务角色)，然后选择已创建或标识为此主题的先决条件一部分的 CodePipeline 服务角色。
7. 对于构件存储，执行下列操作之一：

   - 选择默认位置，在为管道选择的 AWS 区域中为管道使用默认对象存储，例如指定为默认位置的 S3 对象存储桶。
   - 如果您已经在与管道相同的 AWS 区域中创建了现有对象存储，例如 S3 对象存储桶，请选择自定义位置。

   > **Note**
   >
   > 这不是管道的源代码的源存储桶。这是管道的项目存储。每个管道都需要单独的对象存储，例如 S3 存储桶，与管道位于同一 AWS 区域。
8. 选择下一步。
9. 在步骤 2：添加资源阶段页面上，对于源提供商，执行下列操作之一：

   - 如果您的源代码存储在 S3 存储桶中，请选择 Amazon S3。对于存储桶，选择包含源代码的 S3 存储桶。对于 S3 对象密钥，输入包含源代码的文件的名称（例如，*file-name*.zip）。选择下一步。
   - 如果您的源代码存储在存储 AWS CodeCommit 库中，请选择CodeCommit。对于存储库名称，请选择包含源代码的存储库的名称。对于分支名称，请选择包含要构建的源代码版本的分支名称。选择下一步。

- 如果您的源代码存储在存储 GitHub 库中，请选择GitHub。选择 “连接到” GitHub，然后按照说明进行身份验证 GitHub。对于存储库，请选择包含源代码的存储库的名称。对于分支，请选择包含要构建的源代码版本的分支名称。

选择下一步。

10. 在 Step 3: Add build stage (步骤 3: 添加构建阶段) 页面上，对于 Build provider (构建提供商)，选择 CodeBuild。
11. 如果您已有要使用的构建项目，则对于项目名称，选择构建项目的名称并跳到本过程的下一步。

    如果您需要创建新的 CodeBuild 构建项目，请按照中的说明进行操作创建构建项目（控制台）并返回此过程。

    如果您选择现有构建项目，则该项目必须已定义构建输出构件设置（即使 CodePipeline 会覆盖这些设置）。有关更多信息，请参阅 更改构建项目的设置（控制台）。

    > **Important**
    >
    > 如果您为 CodeBuild 项目启用 webhook，并将该项目用作构建步骤 CodePipeline，则每次提交都会创建两个相同的构建。一个生成通过 Webhook 触发，另一个生成通过 CodePipeline 触发。由于账单基于每个构建，因此您需要为这两个构建付费。因此，如果您正在使用 CodePipeline，我们建议您在中 CodeBuild禁用 webhook。在 AWS CodeBuild 控制台中，清除 Webhook 框。有关更多信息，请参阅 更改构建项目的设置（控制台）。

12. 在步骤 4: 添加部署阶段页面上，执行下列操作之一：

    - 如果您不想部署构建输出项目，请在系统提示时选择跳过并确认此选择。
    - 如果要部署构建输出项目，对于部署提供商，选择部署提供商，然后在系统提示时指定设置。

    选择下一步。

13. 在查看页面上，查看您的选择，然后选择创建管道。
14. 管道成功运行后，您可以获取构建输出构件。 CodePipeline 控制台中显示管道后，在 “构建” 操作中，选择工具提示。记下输出对象的值（例如，MyAppBuild）。

> **Note**
>
> 您还可以通过在 CodeBuild 控制台的编译详细信息页面上选择“生成构件”链接来获取编译输出构件。要前往此页面，请跳过此过程中的剩余步骤，并参阅查看构建详细信息（控制台）。

15. 打开 Amazon S3 控制台，网址为 https://console.aws.amazon.com/s3/。
16. 在存储桶列表中，请打开管道使用的存储桶。此存储桶的名称应遵循格式 `codepipeline-`*`region-ID`*`-`*`random-number`*。您可以使用运行 CodePipeline get-pipeline命令 AWS CLI 来获取存储桶的名称，其中*`my-pipeline-name`*是管道的显示名称：

    ```
    aws codepipeline get-pipeline --name my-pipeline-name
    ```

    在输出中，该 `pipeline` 对象包含一个 `artifactStore` 对象，其中包含带有存储桶名称的 `location` 值。
17. 打开与您的管道名称匹配的文件夹（根据管道名称的长度，文件夹名称可能被截断），然后打开与您之前记下的输出构件的值匹配的文件夹。
18. 提取文件内容。如果该文件夹中有多个文件，请提取具有最新上一次修改时间戳的文件的内容。（您可能需要为文件提供 `.zip` 扩展名，这样，您可以将其用于您系统内的 ZIP 实用工具。）构建输出构件将位于文件的提取内容中。
19. 如果您指示 CodePipeline 部署构建输出对象，请使用部署提供商的说明访问部署目标上的构建输出构件。

创建使用 CodeBuild (AWS CLI) 的管道

使用以下过程创建用于生成源代码 CodeBuild 的管道。

要使用创建用于部署您构建的源代码或仅测试源代码的管道，您可以调整编辑管道 (AWS CLI) 中的说明和AWS CodePipeline 用户指南中的CodePipeline管道结构参考。 AWS CLI

1. 在中创建或标识构建项目 CodeBuild。有关更多信息，请参阅 创建构建项目。

> **Important**
>
> 生成项目必须定义生成输出项目设置（即使 CodePipeline 覆盖它们）。有关更多信息，请参阅创建构建项目 (AWS CLI)中 `artifacts` 的描述。

2. 确保您已 AWS CLI 使用与本主题中 AWS 描述的 IAM 实体之一相对应的访问 AWS 密钥和私有访问密钥配置。有关更多信息，请参阅《AWS Command Line Interface 用户指南》中的开始设置 AWS Command Line Interface。
3. 创建代表管道结构的 JSON-formatted 文件。将文件命名为 `create-pipeline.json` 或类似名称。例如，此 JSON-formatted 结构创建了一个管道，其中包含引用 S3 输入存储桶的源操作和使用 CodeBuild以下内容的生成操作：

```
{
  "pipeline": {
    "roleArn": "arn:aws:iam::<account-id>:role/<AWS-CodePipeline-service-role-name>",
    "stages": [
      {
        "name": "Source",
        "actions": [
          {
            "inputArtifacts": [],
            "name": "Source",
            "actionTypeId": {
              "category": "Source",
              "owner": "AWS",
              "version": "1",
              "provider": "S3"
            },
            "outputArtifacts": [
              {
                "name": "MyApp"
              }
            ],
            "configuration": {
              "S3Bucket": "<bucket-name>",
              "S3ObjectKey": "<source-code-file-name.zip>"
            },
            "runOrder": 1
          }
        ]
```

```
      },
      {
        "name": "Build",
        "actions": [
          {
            "inputArtifacts": [
              {
                "name": "MyApp"
              }
            ],
            "name": "Build",
            "actionTypeId": {
              "category": "Build",
              "owner": "AWS",
              "version": "1",
              "provider": "CodeBuild"
            },
            "outputArtifacts": [
              {
                "name": "default"
              }
            ],
            "configuration": {
              "ProjectName": "<build-project-name>"
            },
            "runOrder": 1
          }
        ]
      }
    ],
    "artifactStore": {
      "type": "S3",
      "location": "<CodePipeline-internal-bucket-name>"
    },
    "name": "<my-pipeline-name>",
    "version": 1
  }
}
```

在此 JSON-formatted 数据中：

- 的值`roleArn`必须与您在先决条件中创建或标识的 CodePipeline 服务角色的 ARN 相匹配。

- `configuration` 中 `S3Bucket` 和 `S3ObjectKey` 的值假定源代码存储在 S3 存储桶中。有关其他源代码存储库类型的设置，请参阅《AWS CodePipeline 用户指南》中的CodePipeline 管道结构参考。
- 的值`ProjectName`是您在本过程前面创建的 CodeBuild 生成项目的名称。
- `location` 的值是此管道所用的 S3 存储桶的名称。有关更多信息，请参阅AWS CodePipeline 用户指南中的创建用作 S3 存储桶对象存储 CodePipeline的策略。
- `name` 的值是此管道的名称。所有管道名称对您的账户都必须是唯一的。

尽管这些数据仅描述了源操作和生成操作，但您可以为与测试、部署生成输出构件、调用 AWS Lambda 函数等相关的活动添加操作。有关更多信息，请参阅《AWS CodePipeline 用户指南》中的 AWS CodePipeline 管道结构参考。

4. 切换到包含 JSON 文件的文件夹，然后运行 CodePipelinecreate-pipeline命令，指定文件名：

```
aws codepipeline create-pipeline --cli-input-json file://create-pipeline.json
```

> **Note**
>
> 您必须在支持的 AWS 区域中创建管道。 CodeBuild 有关更多信息，请参阅Amazon Web Services 一般参考 中的 AWS CodeBuild。

JSON-formatted 数据显示在输出中，并 CodePipeline 创建管道。

5. 要获取有关管道状态的信息，请运行 CodePipeline get-pipeline-state命令，指定管道的名称：

```
aws codepipeline get-pipeline-state --name <my-pipeline-name>
```

在输出中，查找确认构建成功的信息。省略号 (`...`) 用于显示为简洁起见而省略的数据。

```
{
  ...
  "stageStates": [
    ...
    {
      "actionStates": [
        {
          "actionName": "CodeBuild",
```

```
          "latestExecution": {
            "status": "SUCCEEDED",
            ...
          },
          ...
        }
      ]
    }
  ]
}
```

如果您过早运行此命令，您可能不会看到有关构建操作的信息。您可能需要多次运行此命令，直到管道已完成构建操作的运行。

6. 成功构建后，请按照以下说明操作，获取构建输出项目。打开 Amazon S3 控制台，网址为 https://console.aws.amazon.com/s3/。

> **Note**
>
> 您还可以通过在 控制台的相关构建详细信息页面上选择 Build artifacts CodeBuild 链接来获取构建输出项目。要前往此页面，请跳过此过程中的剩余步骤，并参阅查看构建详细信息（控制台）。

7. 在存储桶列表中，请打开管道使用的存储桶。此存储桶的名称应遵循格式 `codepipeline-<region-ID>-<random-number>`。您可以从`create-pipeline.json`文件中获取存储桶名称，也可以运行 CodePipeline get-pipeline命令来获取存储桶的名称。

```
aws codepipeline get-pipeline --name <pipeline-name>
```

在输出中，该 `pipeline` 对象包含一个 `artifactStore` 对象，其中包含带有存储桶名称的 `location` 值。

8. 打开与您的管道名称相匹配的文件夹 (例如，*<pipeline-name>*)。
9. 在该文件夹中，打开名为 `default` 的文件夹。
10. 提取文件内容。如果该文件夹中有多个文件，请提取具有最新上一次修改时间戳的文件的内容。(您可能需要为文件提供 `.zip` 扩展名，这样，您可以将其用于您系统内的 ZIP 实用工具。) 构建输出构件将位于文件的提取内容中。

向管道添加 CodeBuild 生成操作（CodePipeline 控制台）

1. 使用以下 AWS 管理控制台 方式登录：

   - 您的 AWS 根账号。我们不建议这么做。有关更多信息，请参阅《用户指南》中的账户根用户。
   - 您 AWS 账户中的管理员用户。有关更多信息，请参阅用户指南中的创建您的第一个 AWS 账户 root 用户和群组。
   - 您 AWS 账户中有权执行以下最低限度操作的用户：

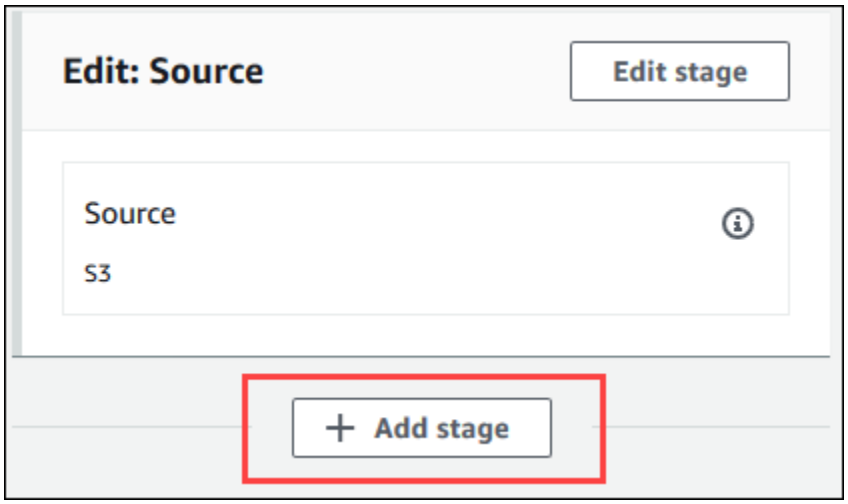
   ```
   codepipeline:*
   iam:ListRoles
   iam:PassRole
   s3:CreateBucket
   s3:GetBucketPolicy
   s3:GetObject
   s3:ListAllMyBuckets
   s3:ListBucket
   s3:PutBucketPolicy
   codecommit:ListBranches
   codecommit:ListRepositories
   codedeploy:GetApplication
   codedeploy:GetDeploymentGroup
   codedeploy:ListApplications
   codedeploy:ListDeploymentGroups
   elasticbeanstalk:DescribeApplications
   elasticbeanstalk:DescribeEnvironments
   lambda:GetFunctionConfiguration
   lambda:ListFunctions
   opsworks:DescribeStacks
   opsworks:DescribeApps
   opsworks:DescribeLayers
   ```

2. 在处打开 CodePipeline 控制台https://console.aws.amazon.com/codesuite/codepipeline/home。
3. 在 AWS 区域选择器中，选择您的管道所在的 AWS 区域。这必须 CodeBuild 是受支持的地区。有关更多信息，请参阅Amazon Web Services 一般参考中的 CodeBuild。
4. 在管道页面上，选择管道的名称。
5. 在管道详细信息页面的源操作中，选择工具提示。记下输出对象的值（例如，MyApp）。

> **Note**
>
> 此过程向您演示如何将构建操作添加到源和测试阶段之间的构建阶段内。如果您要在其他位置添加构建操作，在您要添加构建操作的位置之前的操作上选择工具提示，并记下输出构件的值。

6. 选择编辑。
7. 在源和测试阶段之间，选择添加阶段。

> **Note**
>
> 此过程向您演示如何在源和测试阶段之间添加构建阶段。要将构建操作添加到现有的阶段，请选择阶段中的编辑阶段，然后跳到此过程的步骤 8。要在其他位置添加构建阶段，请在所需位置选择添加阶段。



8. 对于阶段名称，输入构建阶段的名称（例如，**Build**）。如果您选择了其他名称，请在整个过程中使用该名称。
9. 在选定阶段内，选择添加操作。

> **Note**
>
> 此过程向您演示如何在构建阶段内添加构建操作。要在其他位置添加构建操作，请在所需位置选择添加操作。您可能需要先在您要添加构建操作的现有阶段内选择编辑阶段。

10. 在编辑操作中，对于操作名称，输入操作的名称（例如，**CodeBuild**）。如果您选择了其他名称，请在整个过程中使用该名称。
11. 对于 Action provider (操作提供商)，选择 CodeBuild。
12. 如果您已有要使用的构建项目，则对于项目名称，选择构建项目的名称并跳到本过程的下一步。

    如果您需要创建新的 CodeBuild 构建项目，请按照中的说明进行操作创建构建项目（控制台）并返回此过程。

    如果您选择现有构建项目，则该项目必须已定义构建输出构件设置（即使 CodePipeline 会覆盖这些设置）。有关更多信息，请参阅创建构建项目（控制台）或更改构建项目的设置（控制台）中构件的描述。

    > **Important**
    >
    > 如果您为 CodeBuild 项目启用 webhook，并将该项目用作构建步骤 CodePipeline，则每次提交都会创建两个相同的构建。一个生成通过 Webhook 触发，另一个生成通过 CodePipeline 触发。由于账单基于每个构建，因此您需要为这两个构建付费。因此，如果您正在使用 CodePipeline，我们建议您在中 CodeBuild禁用 webhook。在 CodeBuild 控制台中，清除 Webhook 复选框。有关更多信息，请参阅 更改构建项目的设置（控制台）。

13. 对于输入构件，选择您在此过程的前面记下的输出构件。
14. 对于输出构件，输入输出构件的名称（例如，**MyAppBuild**）。
15. 选择添加操作。
16. 选择保存，然后选择保存以保存对管道的更改。
17. 选择发布更改。
18. 管道成功运行后，您可以获取构建输出构件。 CodePipeline 控制台中显示管道后，在“构建”操作中，选择工具提示。记下输出对象的值（例如，MyAppBuild）。

    > **Note**
    >
    > 您还可以通过在 CodeBuild 控制台的编译详细信息页面上选择“生成构件”链接来获取编译输出构件。要访问此页面，请参阅查看构建详细信息（控制台），然后跳到此过程的步骤 31。

19. 打开 Amazon S3 控制台，网址为 https://console.aws.amazon.com/s3/。

20. 在存储桶列表中，请打开管道使用的存储桶。此存储桶的名称应遵循格式 `codepipeline-*region-ID*-*random-number*`。您可以使用运行 CodePipeline get-pipeline命令 AWS CLI 来获取存储桶的名称：

    ```
    aws codepipeline get-pipeline --name my-pipeline-name
    ```

    在输出中，该 `pipeline` 对象包含一个 `artifactStore` 对象，其中包含带有存储桶名称的 `location` 值。

21. 打开与您的管道名称匹配的文件夹（根据管道名称的长度，文件夹名称可能被截断），然后打开与您在此过程的前面记下的输出构件的值匹配的文件夹。

22. 提取文件内容。如果该文件夹中有多个文件，请提取具有最新上一次修改时间戳的文件的内容。（您可能需要为文件提供 `.zip` 扩展名，这样，您可以将其用于您系统内的 ZIP 实用工具。）构建输出构件将位于文件的提取内容中。

23. 如果您指示 CodePipeline 部署构建输出对象，请使用部署提供商的说明访问部署目标上的构建输出构件。

向管道添加 CodeBuild 测试操作（CodePipeline 控制台）

1. 使用以下 AWS 管理控制台 方式登录：

   - 您的 AWS 根账户。我们不建议这么做。有关更多信息，请参阅《用户指南》中的账户根用户。
   - 您 AWS 账户中的管理员用户。有关更多信息，请参阅用户指南中的创建您的第一个 AWS 账户 root 用户和群组。
   - 您 AWS 账户中有权执行以下最低限度操作的用户：

     ```
     codepipeline:*
     iam:ListRoles
     iam:PassRole
     s3:CreateBucket
     s3:GetBucketPolicy
     s3:GetObject
     s3:ListAllMyBuckets
     s3:ListBucket
     s3:PutBucketPolicy
     codecommit:ListBranches
     codecommit:ListRepositories
     codedeploy:GetApplication
     codedeploy:GetDeploymentGroup
     ```

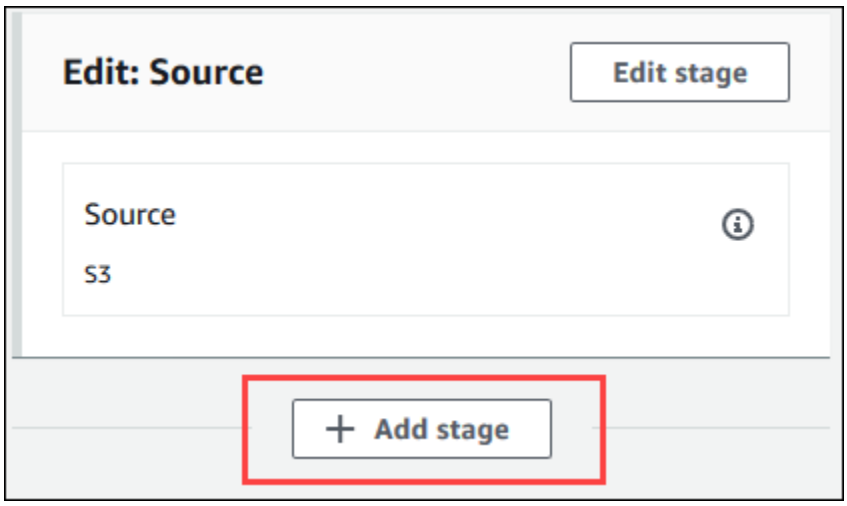
```
codedeploy:ListApplications
codedeploy:ListDeploymentGroups
elasticbeanstalk:DescribeApplications
elasticbeanstalk:DescribeEnvironments
lambda:GetFunctionConfiguration
lambda:ListFunctions
opsworks:DescribeStacks
opsworks:DescribeApps
opsworks:DescribeLayers
```

2. 在处打开 CodePipeline 控制台https://console.aws.amazon.com/codesuite/codepipeline/home。
3. 在 AWS 区域选择器中，选择您的管道所在的 AWS 区域。这必须 CodeBuild 是支持的地 AWS 区。有关更多信息，请参阅Amazon Web Services 一般参考中的 AWS CodeBuild。
4. 在管道页面上，选择管道的名称。
5. 在管道详细信息页面的源操作中，选择工具提示。记下输出对象的值（例如，MyApp）。

> **Note**
>
> 此过程向您演示如何将测试操作添加到源和测试阶段之间的测试阶段内。如果您要在其他位置添加测试操作，请将鼠标指针停留在之前的操作上，然后记下输出项目的值。

6. 选择编辑。
7. 紧接着源阶段，选择添加阶段。

> **Note**
>
> 此过程向您演示如何在管道中紧接着源阶段添加测试阶段。要将测试操作添加到现有的阶段，请选择阶段中的编辑阶段，然后跳到此过程的步骤 8。要在其他位置添加测试阶段，请在所需位置选择添加阶段。



8. 对于阶段名称，输入测试阶段的名称（例如，**Test**）。如果您选择了其他名称，请在整个过程中使用该名称。
9. 在选定阶段中，选择添加操作。

> **Note**
>
> 此过程向您演示如何在测试阶段内添加测试操作。要在其他位置添加测试操作，请在所需位置选择添加操作。您可能需要先在您要添加测试操作的现有阶段内选择编辑阶段。

10. 在编辑操作中，对于操作名称，输入操作的名称（例如，**Test**）。如果您选择了其他名称，请在整个过程中使用该名称。
11. 对于 Action provider (操作提供商)，选择 Test (测试) 下的 CodeBuild。
12. 如果您已有要使用的构建项目，则对于项目名称，选择构建项目的名称并跳到本过程的下一步。

    如果您需要创建新的 CodeBuild 构建项目，请按照中的说明进行操作[创建构建项目（控制台）](#)并返回此过程。

> **Important**
>
> 如果您为 CodeBuild 项目启用 webhook，并将该项目用作构建步骤 CodePipeline，则每次提交都会创建两个相同的构建。一个生成通过 Webhook 触发，另一个生成通过 CodePipeline 触发。由于账单基于每个构建，因此您需要为这两个构建付费。因此，如果您正在使用 CodePipeline，我们建议您在中 CodeBuild禁用 webhook。在 CodeBuild

控制台中，清除 Webhook 复选框。有关更多信息，请参阅 更改构建项目的设置（控制台）。

13. 对于输入构件，选择您在此过程的前面记下的输出构件的值。
14. （可选）如果您希望测试操作来生成输出构件，并且相应地设置构建规范，那么对于输出构件，请输入您要分配给输出构件的值。
15. 选择保存。
16. 选择发布更改。
17. 管道成功运行后，您可以获取测试结果。在管道的测试阶段，选择CodeBuild超链接以在 CodeBuild 控制台中打开相关的构建项目页面。
18. 在构建项目页面上的构建历史记录中，选择构建运行超链接。
19. 在构建运行页面的构建日志中，选择查看整个日志超链接，在亚马逊 CloudWatch 控制台中打开构建日志。
20. 滚动浏览构建日志，查看测试结果。

使用 AWS CodeBuild 使用 Codecov

Codecov 是一种用于测量代码的测试覆盖率的工具。Codecov 可标识您的代码中哪些方法和语句未经测试。可通过结果确定在何处编写测试以提高代码质量。Codecov 可用于支持的三个源存储库 CodeBuild： GitHub、 GitHub 企业服务器和 Bitbucket。如果您的构建项目使用 GitHub 企业服务器，则必须使用 Codecov 企业版。

当您运行与 Codecov 集成的 CodeBuild 项目的构建版本时，分析存储库中代码的 Codecov 报告会上传到 Codecov。构建日志包含指向报告的链接。此示例介绍如何将 Python 和 Java 构建项目与 Codecov 集成。有关 Codecov 支持语言的列表，请参阅 Codecov 网站上的 Codecov 支持语言。

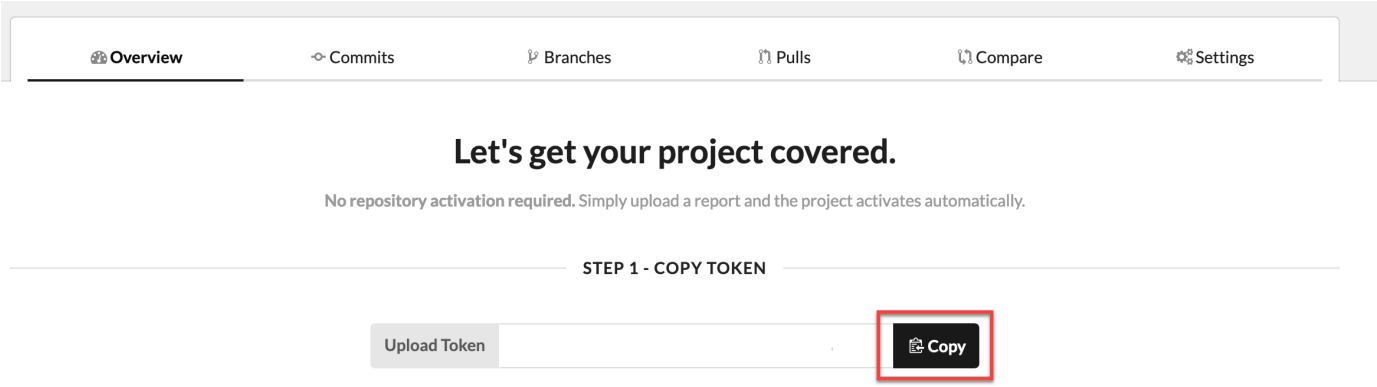
将 Codecov 集成到构建项目中

使用以下步骤将 Codecov 集成到构建项目中。

将 Codecov 与构建项目集成

1. 访问https://codecov.io/signup并注册一个 GitHub 或 Bitbucket 源代码存储库。如果您使用 GitHub Enterprise，请参阅 Codecov 网站上的 Codecov Enterprise。
2. 在 Codecov 中，添加要覆盖的存储库。

3. 在显示令牌信息时，选择复制。



4. 将复制的令牌作为名为 CODECOV_TOKEN 的环境变量添加到构建项目中。有关更多信息，请参阅[更改构建项目的设置（控制台）](#)。
5. 在存储库中创建一个名为 my_script.sh 的文本文件。在文件中输入以下内容：

```
#/bin/bash
bash <(curl -s https://codecov.io/bash) -t $CODECOV_TOKEN
```

6. 根据构建项目的使用情况，选择 Python 或 Java 选项卡，然后按照以下步骤操作。

Java

1. 将以下 JaCoCo 插件添加到您的存储库pom.xml中。

```
<build>
  <plugins>
    <plugin>
      <groupId>org.jacoco</groupId>
      <artifactId>jacoco-maven-plugin</artifactId>
      <version>0.8.2</version>
      <executions>
          <execution>
              <goals>
                  <goal>prepare-agent</goal>
              </goals>
          </execution>
          <execution>
              <id>report</id>
              <phase>test</phase>
              <goals>
                  <goal>report</goal>
              </goals>
```

```
          </execution>
        </executions>
      </plugin>
    </plugins>
</build>
```

2. 在构建规范文件中输入以下命令。有关更多信息，请参阅 buildspec 语法。

```
build:
  - mvn test -f pom.xml -fn
postbuild:
  - echo 'Connect to CodeCov'
  - bash my_script.sh
```

Python

在构建规范文件中输入以下命令。有关更多信息，请参阅 buildspec 语法。

```
build:
  - pip install coverage
  - coverage run -m unittest discover
postbuild:
  - echo 'Connect to CodeCov'
  - bash my_script.sh
```

7. 运行构建项目的构建。指向为项目生成的 Codecov 报告的链接将显示在构建日志中。使用链接查看 Codecov 报告。有关更多信息，请参阅手动运行 AWS CodeBuild 构建和Log AWS CodeBuild 使用 API 调用 AWS CloudTrail：构建日志中的 Codecov 信息与以下内容类似：

```
[Container] 2020/03/09 16:31:04 Running command bash my_script.sh

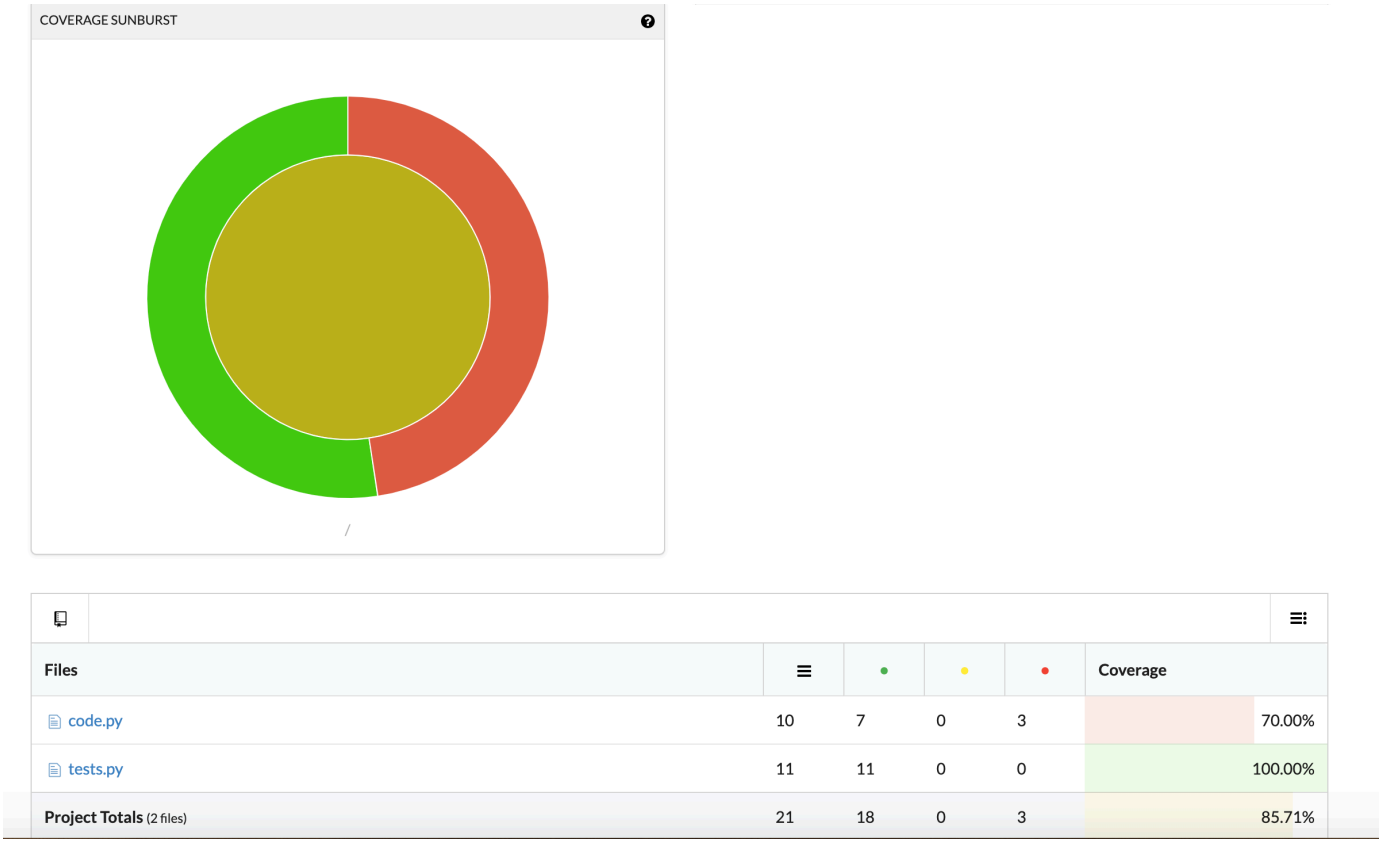

  _____          _
 / ____|        | |
| |     ___   __| | ___  ___ _____   __
| |    / _ \ / _` |/ _ \/ __/ _ \ \ / /
| |___| (_) | (_| |  __/ (_| (_) \ V /
 \_____\___/ \__,_|\___|\___\___/ \_/
                              Bash-20200303-bc4d7e6

·[0;90m==>·[0m AWS CodeBuild detected.
... The full list of Codecov log entries has been omitted for brevity ...
```

```
    .
    .[0;32m->.[0m View reports at .[0;36mhttps://codecov.io/github/user/test_py/commit/commit-id.[0m

[Container] 2020/03/09 16:31:07 Phase complete: POST_BUILD State: SUCCEEDED
```

报告与以下内容类似：



使用 AWS CodeBuild 和詹金斯在一起

你可以使用 Jenkins 插件 AWS CodeBuild 来 CodeBuild 与你的 Jenkins 构建任务集成。您可以使用插件将您的构建作业发送给 CodeBuild，而不是发送给 Jenkins 构建节点。这样便无需预置、配置和管理 Jenkins 构建节点。

主题

- 设置 Jenkins
- 安装插件
- 使用插件

设置 Jenkins

有关使用插件设置 Jenkins 以及下载 AWS CodeBuild 插件源代码的信息，请参阅https://github.com/awslabs/aws-codebuild-jenkins-plugin。

安装插件

如果您已设置 Jenkins 服务器并希望仅安装 AWS CodeBuild 插件，请在您的 Jenkins 实例上的插件管理器中搜索 **CodeBuild Plugin for Jenkins**。

使用插件

要将 AWS CodeBuild 使用来自 VPC 外部的来源

1. 在 CodeBuild 控制台中创建项目。有关更多信息，请参阅 创建构建项目（控制台）。
   - 选择要运行构建的 AWS 区域。
   - （可选）设置 Amazon VPC 配置，以允许 CodeBuild 构建容器访问您的 VPC 中的资源。
   - 记下您的项目的名称。您在步骤 3 中需要它。
   - （可选）如果您的源存储库本身不支持 CodeBuild，则可以将 Amazon S3 设置为项目的输入源类型。
2. 在 IAM 控制台中，创建一个供 Jenkins 插件使用的用户。
   - 当您为该用户创建凭证时，请选择编程访问。
   - 创建如下所示的策略，然后将该策略附加到您的用户。

     JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Resource": [
                "arn:aws:logs:us-east-1:111122223333:log-group:/aws/
codebuild/{{projectName}}:*"
            ],
            "Action": [
                "logs:GetLogEvents"
            ]
```

```
        },
        {
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::{{inputBucket}}"
            ],
            "Action": [
                "s3:GetBucketVersioning"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::{{inputBucket}}/{{inputObject}}"
            ],
            "Action": [
                "s3:PutObject"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::{{outputBucket}}/*"
            ],
            "Action": [
                "s3:GetObject"
            ]
        },
        {
            "Effect": "Allow",
            "Resource": [
                "arn:aws:codebuild:us-east-1:111122223333:project/
{{projectName}}"
            ],
            "Action": [
                "codebuild:StartBuild",
                "codebuild:BatchGetBuilds",
                "codebuild:BatchGetProjects"
            ]
        }
    ]
}
```

3. 在 Jenkins 中创建一个自由式项目。

- 在配置页面上，选择添加构建步骤，然后选择在 CodeBuild 上运行构建任务。
- 配置您的构建步骤。
  - 为区域、凭证和项目名称提供值。
  - 选择使用项目源。
  - 保存配置并从 Jenkins 运行构建任务。

4. 对于源代码管理，选择您希望如何检索您的源。你可能需要在 Jenkins 服务器上安装该 GitHub 插件（或源存储库提供商的 Jenkins 插件）。

- 在配置页面上，选择添加构建步骤，然后选择在 AWS CodeBuild上运行构建任务。
- 配置您的构建步骤。
  - 为区域、凭证和项目名称提供值。
  - 选择使用 Jenkins 源。
  - 保存配置并从 Jenkins 运行构建任务。

要再次使用 AWS CodeBuild 带有 Jenkins 管道插件的插件

- 在您的 Jenkins 流水线项目页面上，使用代码片段生成器生成流水线脚本，该脚本 CodeBuild 作为步骤添加到您的管道中。它应生成如下所示的脚本：

```
awsCodeBuild projectName: 'project', credentialsType: 'keys', region: 'us-west-2',
 sourceControlType: 'jenkins'
```

AWS CodeBuild 与无服务器应用程序一起使用

AWS Serverless Application Model (AWS SAM) 是一个用于构建无服务器应用程序的开源框架。有关更多信息，请参阅上的AWS 无服务器应用程序模型存储库。 GitHub

您可以使用 AWS CodeBuild 打包和部署 AWS SAM 符合标准的无服务器应用程序。对于部署步骤， CodeBuild 可以使用 AWS CloudFormation。要使用和自动构建和部署无服务器应用程序 CloudFormation， CodeBuild 可以使用 AWS CodePipeline。

有关更多信息，请参阅《AWS Serverless Application Model 开发人员指南》中的部署无服务器应用程序。

相关资源

- 有关入门的信息 AWS CodeBuild，请参阅开始使用 AWS CodeBuild 使用控制台。
- 有关中问题疑难解答的信息 CodeBuild，请参阅故障排除 AWS CodeBuild。
- 有关中配额的信息 CodeBuild，请参阅的配额 AWS CodeBuild。

的第三方通知 AWS CodeBuild 适用于 Windows

当你用 CodeBuild 于 Windows 版本时，你可以选择使用一些第三方包和模块来使你构建的应用程序能够在 Microsoft Windows 操作系统上运行并与某些第三方产品进行互操作。以下列表包含适用的第三方法律条款，可管理您对指定的第三方程序包和模块的使用。

主题

- 1) 基本 Docker 映像 — windowsservercore
- 2) 基于 Windows 的 Docker 映像 – choco
- 3) 基于 Windows 的 Docker 映像 – git -- 版本 2.16.2
- 4) 基于 Windows 的 Docker 映像 – microsoft-build-tools -- 版本 15.0.26320.2
- 5) 基于 Windows 的 Docker 映像 – nuget.commandline -- 版本 4.5.1
- 7) 基于 Windows 的 Docker 映像 – netfx-4.6.2-devpack
- 8) 基于 Windows 的 Docker 映像 – visualfsharptools，v 4.0
- 9) 基于 Windows 的 Docker 映像 – netfx-pcl-reference-assemblies-4.6
- 10) 基于 Windows 的 Docker 映像 – visualcppbuildtools v 14.0.25420.1
- 11) 基于 Windows 的 Docker 映像 – microsoft-windows-netfx3-ondemand-package.cab
- 12) 基于 Windows 的 Docker 映像 – dotnet-sdk

1) 基本 Docker 映像 — windowsservercore

（许可条款可在以下网址获得：https://hub.docker.com/_/microsoft-windows-servercore ）

许可：请求并使用此适用于 Windows 容器的容器操作系统映像即表明您承认、了解并同意以下补充许可条款：

MICROSOFT 软件补充许可条款

容器操作系统映像

Microsoft Corporation（或者您所在地的其附属公司）（简称为“我们”或“Microsoft”）将此容器操作系统映像补充（下称“补充”）的许可授予您。根据授予的许可，您可以将此补充与基本主机操作系统软件（下称“主机软件”）一起使用，其目的仅用于帮助您在主机软件中运行容器功能。主机软件许可条款适用于您对补充的使用。如果您未获得主机软件的许可，就不能使用它。您可以将此补充用于主机软件的每个授予有效许可的副本。

其他许可要求 AND/OR 使用权

您按照前述段落中的规定使用补充许可可能会导致创建或修改容器映像（下称“容器映像”），而其中包含特定补充组件。为明确起见，容器映像是独立的，不同于虚拟机映像或虚拟设备映像。根据这些许可条款，在遵循下列条件的情况下，我们授予您重新分发此类补充组件的有限许可：

(i) 您仅可以在自己的容器映像中，将补充组件作为映像的一部分使用，

(ii) 只要您的容器映像中的重要主功能与补充在实质上是分离且不同的，您就可以在容器映像中使用此类补充；以及

(iii) 您同意在您的容器映像中包括这些许可条款（或者我们或托管商要求的类似条款），用于对您的最终用户可能使用补充组件正确授予许可。

我们保留未在此处明确授予的所有其他权限。

使用本补充内容即表示您接受这些条款。如果您不接受这些条款，请勿使用本补充内容。

作为此适用于 Windows 容器的容器操作系统映像的补充许可条款的一部分，您还受基础的 Windows Server 主机软件许可条款的约束，这些条款位于：https://www.microsoft.com/en-us/useterms.

2) 基于 Windows 的 Docker 映像 – choco

（许可条款可在以下网址获得：https://github.com/chocolatey/choco/blob/master/LICENSE）

2011 年版权所有-现在的 RealDimensions 软件有限责任公司

根据 Apache 许可版本 2.0 授予许可（简称“许可”），如果不遵守许可，您不可使用这些文件。您可以在以下地址获取许可的副本

http://www.apache.org/licenses/LICENSE-2.0

除非适用的法律要求或以书面方式表示同意，否则，根据该许可分发的软件按“原样”分发，无任何明示或暗示的保证或条件。请参阅许可证以了解在许可证下特定语言的适用权限和限制。

3) 基于 Windows 的 Docker 映像 – git -- 版本 2.16.2

（许可条款可在以下网址获得：https://chocolatey.org/packages/git/2.16.2）

根据GNU通用公共许可证第2版获得许可，可在以下网址获得：https://www.gnu.org/licenses/old-licenses/gpl-2.0.html。

4) 基于 Windows 的 Docker 映像 – microsoft-build-tools -- 版本 15.0.26320.2

（许可条款可在以下网址获得：https://www.visualstudio.com/license-terms/mt171552/）

MICROSOFT VISUAL STUDIO 2015 扩展、VISUAL STUDIO SHELL 和 C++ 可重新分发

这些许可条款是 Microsoft Corporation（或您所在地的其附属公司）与您达成的协议。它们适用于以上所述软件。这些条款还适用于软件的任意 Microsoft 服务或更新，除非另有附加条款。

如果您遵循这些许可条款，您将拥有以下权利。

1. 安装和使用权利。您可以安装和使用该软件任意数量的副本。
2. 特定组件的条款。
   a. 实用程序。该软件可能包含实用程序列表中的某些项目，网址为https://docs.microsoft.com/en-us/visualstudio/productinfo/2015-redistribution-vs。如果软件中包含这些项目，您可以复制并安装这些项目到您或其他第三方的计算机上，用于调试和部署您使用软件开发的应用程序及数据库。请注意，实用程序设计用于临时使用，Microsoft 可能无法独立于软件的其余部分为实用程序打补丁或进行更新，并且一些实用程序在性质上会使得其他人可以访问安装该实用程序的计算机。因此，在您完成调试或部署应用程序及数据库之后，您应删除已安装的所有实用程序。对于任何第三方使用或访问您安装在任意计算机上的实用程序，Microsoft 不承担任何责任。
   b. Microsoft 平台。该软件可能包括来自微软 Windows、微软 Windows 服务器、微软 SQL 服务器、微软 Exchange、微软 Office 和微软的组件 SharePoint。这些组件受独立协议及自己的产品支持政策控制，如位于组件安装目录中或随软件提供的“Licenses”文件夹中的许可条款所述。
   c. 第三方组件。如软件附带 ThirdPartyNotices 文件中所述，该软件可能包括带有单独法律声明或受其他协议约束的第三方组件。即使此类组件受其他协议控制，以下免责声明以及对损害赔偿的限制或排除仍适用。软件还包含在开源许可下授予许可的组件，具有源代码可用性义务。这些许可证的副本（如果适用）包含在 ThirdPartyNotices 文件中。您可以根据相关开源许可的要求，

将 5.00 美元的汇票或支票发送到以下地址，从我们这里获取源代码：Source Code Compliance Team, Microsoft Corporation, 1 Microsoft Way, Redmond, WA 98052。在您付款的备注栏中，请注明以下列出的一个或多个组件的源代码：

- Remote Tools for Visual Studio 2015；
- Standalone Profiler for Visual Studio 2015；
- IntelliTraceCollector 适用于Visual Studio 2015；
- Microsoft VC++ Redistributable 2015；
- Multibyte MFC Library for Visual Studio 2015；
- Microsoft Build Tools 2015；
- Feedback Client；
- Visual Studio 2015 Integrated Shell；或
- Visual Studio 2015 Isolated Shell。

我们也可以在以下网址提供源代码的副本http://thirdpartysource.microsoft.com。

3. DATA。软件可能会收集有关您以及您使用软件的信息，并将信息发送给 Microsoft。Microsoft 可能会使用此信息来提供服务和改进我们的产品及服务。您可以选择退出其中的多种情况，但并非全部，如产品文档中所述。此外，软件中还有一些功能，可能使您能够从应用程序的用户收集数据。如果您使用这些功能在应用程序中启用数据收集，则必须遵循适用的法律，包括向应用程序的用户提供相应的说明。您可以在帮助文档和隐私声明中了解有关数据收集和使用的更多信息，网址为https://privacy.microsoft.com/en-us/privacystatement。您使用软件操作即表明您同意这些做法。
4. 许可范围。该软件授予许可，而非销售。本协议仅向您提供使用软件的一些权利。Microsoft 保留所有其他权利。除非适用法律在此限制之外赋予您更多的权利，否则您只能在本协议中明确允许的情况下使用该软件。在使用时，您必须遵守软件中仅允许您以特定方式使用它的任意技术限制。您不能
   - 规避软件中的任何技术限制；
   - 逆向工程、反编译或反汇编软件，或者尝试这样做（除非且仅限于在第三方许可条款要求的范围内，该条款规定软件中可能包含的特定开源组件的使用）
   - 删除、最小化、阻止或修改软件中 Microsoft 或其供应商的通知；
   - 以违反法律的方式使用软件；或者
   - 共享、发布、租借或租用软件，或者将软件独立托管为解决方案供其他人使用。
5. 出口限制。您必须遵守所有适用于该软件的国内和国际出口法律及法规，这包括对目的地、最终用户和最终用途的限制。有关出口限制的更多信息，请访问（又名ms/exporting）。
6. 支持服务。由于此软件“按原样”提供，我们可能不会为它提供支持服务。

7. 完整协议。本协议以及您使用的补充、更新、 Internet-based 服务和支持服务的条款是软件和支持服务的完整协议。
8. 适用的法律。如果您在美国购买本软件，华盛顿州法律管辖对本协议的解释以及违反协议的索赔，您居住州的法律适用于所有其他索赔。如果您在任何其他国家/地区购买本软件，则适用该国家/地区的法律。
9. 消费者权利；区域差异。本协议描述了特定法律权利。根据所在的州或国家/地区，您可能拥有其他权利，包括消费者权利。除了与 Microsoft 的关系之外，您还可能对与您购买该软件的一方的拥有相应权利。如果您所在的州或国家/地区的法律不允许，本协议不会更改这些其他权利。例如，如果您在以下区域之一购买了本软件，或者有强制性国家/地区法律适用，则以下条款适用于您：
   a. 澳大利亚。您在澳大利亚消费者法下获得了法定担保，本协议中的任何内容都无意影响这些权利。
   b. 加拿大。如果您在加拿大购买此软件，您可通过关闭自动更新功能、断开设备与 Internet 的连接（但是，在您重新连接到 Internet 时，软件将恢复检查和安全更新）或者卸载软件来停止接收更新。产品文档（如果有）可能会说明如何关闭特定设备或软件的更新。
   c. 德国和奥地利。
      i. 担保。正确授予许可的软件，将基本按照随该软件所提供的任意 Microsoft 材料中所述执行。但是，Microsoft 不提供任何与所许可软件相关的合同担保。
      ii. 责任限制。在出现故意行为、重大过失、基于产品责任法的索赔时，以及出现死亡、人身伤害或物理伤害时，Microsoft 根据成文法律承担责任。根据前述条款 (ii)，Microsoft 仅在违背了实质性合同义务时，在承担责任有助于正当履行此协议时，违反许可会危害到此协议的目的时，以及符合当事人值得长期信任的情况下（称为“基本义务”），Microsoft 才会承担轻微过失责任。在其他轻微过失的情况下，Microsoft 不承担轻微过失的责任。
10. 免责声明。该软件已获得许可” AS-IS。您承担使用它的风险。MICROSOFT 不提供任何明示的担保、保证或条件。在当地法律允许的范围内，MICROSOFT 排除对适销性、特定用途的适用性等方面的默示担保。 NON-INFRINGEMENT
11. 对损害赔偿的限制或排除。您只能向微软及其供应商追回的直接损害赔偿金不超过以下金额 U.S。5.00 美元。您无法获得任何其他损害赔偿，包括后果性损害、利润损失、间接损害或事故损害。此限制适用于 (a) 第三方 Internet 站点或第三方应用程序上与软件、服务、内容（包括代码）相关的任意内容；(b) 违反合同；违反担保、保证或条件；严格责任；疏忽；或法律允许情况下其他侵权行为的索赔。

    即使 Microsoft 已知或者应该知道造成损害的可能性，此条款仍适用。由于您所在的国家/地区可能不允许排除或限制事故损害、后果性损害或其他损害，上述限制或排除可能不适用于您。

最终用户许可协议 ID：vs ShellsRedist 2015_update3_ _ <ENU>

5) 基于 Windows 的 Docker 映像 – nuget.commandline -- 版本 4.5.1

（许可条款可在以下网址获得：https://github.com/NuGet/Home/blob/dev/LICENSE.txt）

版权所有 (c) .NET Foundation。保留所有权利。

根据 Apache 许可版本 2.0 授予许可（简称“许可”），如果不遵守许可，您不可使用这些文件。您可以在以下地址获取许可的副本

http://www.apache.org/licenses/LICENSE-2.0

除非适用的法律要求或以书面方式表示同意，否则，根据该许可分发的软件按“原样”分发，无任何明示或暗示的保证或条件。请参阅许可证以了解在许可证下特定语言的适用权限和限制。

7) 基于 Windows 的 Docker 映像 – netfx-4.6.2-devpack

MICROSOFT 软件补充许可条款

用于 MICROSOFT WINDOWS 操作系统的 .NET FRAMEWORK 和相关语言包

Microsoft Corporation（或您所在地的其附属公司）向您授予此补充的许可。如果您获得使用 Microsoft Windows 操作系统软件（下称“软件”）的许可，则可以使用此补充。如果您未获得软件的许可，就不能使用它。您可以将此补充用于软件的每个授予有效许可的副本。

以下许可条款描述了此补充的额外使用条款。软件的这些条款和许可条款适用于您对补充的使用。如果存在冲突，这些补充许可条款适用。

使用本补充内容即表示您接受这些条款。如果您不接受这些条款，请勿使用本补充内容。

如果您遵循这些许可条款，您将拥有以下权利。

1. 可分发代码。此补充由可分发代码组成。“可分发代码”是在您遵守以下条款的情况下，允许您在自己开发的程序中分发的代码。
   a. 使用和分发权利。
      - 您可以复制和分发本补充的对象代码形式。
      - 第三方分发。您可以允许程序分发者复制和分发可分发代码作为这些程序的一部分。

b. 分发要求。对于您分发的任何可分发代码，您必须

- 在程序中向其添加重要主功能；
- 对于文件扩展名为 .lib 的任意可分发代码，仅分发通过您程序的链接器运行此类可分发代码的结果；
- 仅在可分发代码未经修改的情况下，将其作为安装程序的一部分分发；
- 要求分发者和外部最终用户同意不低于本协议要求的保护条款；
- 在您的程序上显示有效的版权声明；以及
- 对于与分发或使用您的程序相关的索赔，为 Microsoft 辩护、补偿和使其免受损害，包括律师费。

c. 分发限制。您不能

- 更改可分发代码中的任何著作权、商标或专利通知；
- 在您的程序名称中使用 Microsoft 商标，或者以暗示程序来自 Microsoft 或者得到 Microsoft 认可的方式使用 Microsoft 商标；
- 将可分发代码分发在 Windows 平台之外的平台上运行；
- 在可分发代码中包括恶意、欺骗性或者非法程序；或者
- 修改或分发任意可分发代码的源代码，以使其任何部分都受限于排除许可。排除许可是针对使用、修改或分发的条件，要求
  - 代码以源代码的形式公布或分发；或
  - 其他人有权修改它。

2. 补充的支持服务。如 www.support.microsoft所述，微软为该软件提供支持服务。com/common/international.aspx 。

8) 基于 Windows 的 Docker 映像 – visualfsharptools，v 4.0

（许可条款可在以下网址获得：https://github.com/dotnet/fsharp/blob/main/License.txt ）

版权所有 (c) Microsoft Corporation。保留所有权利。

根据 Apache 许可版本 2.0 授予许可（简称“许可”），如果不遵守许可，您不可使用这些文件。您可以在以下地址获取许可的副本

http://www.apache.org/licenses/LICENSE-2.0

除非适用的法律要求或以书面方式表示同意，否则，根据该许可分发的软件按“原样”分发，无任何明示或暗示的保证或条件。请参阅许可证以了解在许可证下特定语言的适用权限和限制。

9) 基于 Windows 的 Docker 映像 – netfx-pcl-reference-assemblies-4.6

Microsoft 软件许可条款

MICROSOFT .NET 可移植类库引用程序集 – 4.6

这些许可条款是 Microsoft Corporation（或您所在地的其附属公司）与您达成的协议。请仔细阅读。它们适用于以上所述软件。该条款也适用于此软件的任意 Microsoft

- 更新、
- 补充、
- Internet-based 服务，以及
- 支持服务，

除非随这些项目另有其他条款。如果是这样，则那些条款也适用。

使用本软件即表示您接受这些条款。如果您不接受这些条款，请勿使用本软件。

如果您遵循这些许可条款，您将拥有以下永久权利。

1. 安装和使用权利。您可以安装和使用任何数量的软件副本来设计、开发和测试您的程序。
2. 其他许可要求 AND/OR 使用权。
   a. 可分发代码。您可以在您开发的开发人员工具程序中分发本软件，以便您程序的客户可以开发可用于任何设备或操作系统的可移植库，前提是您遵守以下条款。
      i. 使用和分发权利。本软件是“可分发代码”。
         - 可分发代码。您可以复制和分发本软件的对象代码形式。
         - 第三方分发。您可以允许程序分发者复制和分发可分发代码作为这些程序的一部分。
      ii. 分发要求。对于您分发的任何可分发代码，您必须
         - 在程序中向其添加重要主功能；
         - 要求分发者和您的客户同意不低于本协议要求的保护条款；
         - 在您的程序上显示有效的版权声明；以及
         - 对于与分发或使用您的程序相关的索赔，为 Microsoft 辩护、补偿和使其免受损害，包括律师费。

iii. 分发限制。您不能

- 更改可分发代码中的任何著作权、商标或专利通知；
- 在您的程序名称中使用 Microsoft 商标，或者以暗示程序来自 Microsoft 或者得到 Microsoft 认可的方式使用 Microsoft 商标；
- 在可分发代码中包括恶意、欺骗性或者非法程序；或者
- 修改或分发可分发代码，以使其任何部分都受限于排除许可。排除许可是针对使用、修改或分发的条件，要求
  - 代码以源代码的形式公布或分发；或
  - 其他人有权修改它。

3. 许可范围。该软件授予许可，而非销售。本协议仅向您提供使用软件的一些权利。Microsoft 保留所有其他权利。除非适用法律在此限制之外赋予您更多的权利，否则您只能在本协议中明确允许的情况下使用该软件。在使用时，您必须遵守软件中仅允许您以特定方式使用它的任意技术限制。您不能
   - 规避软件中的任何技术限制；
   - 逆向工程、反编译或反汇编本软件，尽管有此限制，但在适用法律明确允许的范围内可以执行上述操作；
   - 发布本软件供其他人复制；或者
   - 出租、租赁或出借本软件。
4. 反馈。您可以提供关于本软件的反馈。如果您向 Microsoft 提供关于本软件的反馈，则表示您向 Microsoft 免费提供以任何方式和任何用途使用、共享和商业化您的反馈的权利。您还可以免费向第三方提供其产品、技术和服务所需的任何专利权，以便使用包含反馈的 Microsoft 软件或服务的任何特定部分或者与之进行交互。如果反馈受下面这样的许可证的约束，则您不能提供反馈：该许可证要求 Microsoft 向第三方提供其软件或文档的许可，因为我们在相应软件或文档中包含了您的反馈。这些权利不受本协议的约束。
5. 转让给第三方。本软件的第一个用户可以将本软件和本协议直接转让给第三方。在转让之前，该第三方必须同意本协议适用于本软件的转让和使用。第一个用户在将本软件独立于设备进行转让之前，必须先卸载本软件。第一个用户不能保留任何副本。
6. 出口限制。本软件受美国出口法律和法规的约束。您必须遵守适用于本软件的所有国内和国际出口法律和法规。这些法律包括对目的地、最终用户和最终用途的限制。有关更多信息，请参阅 www.microsoft。com/exporting。
7. 支持服务。由于此软件“按原样”提供，我们可能不会为它提供支持服务。
8. 完整协议。本协议以及您使用的补充、更新、 Internet-based 服务和支持服务的条款是软件和我们提供的任何支持服务的完整协议。

9. 适用的法律。

   a. 美国。如果您在美国购买本软件，华盛顿州法律管辖本协议的解释，并适用于违反本协议的索赔，无论法律原则是否冲突都是如此。您居住的州的法律管辖所有其他索赔，包括根据国家消费者保护法、不正当竞争法和侵权行为提起的索赔。

   b. 在美国以外的国家或地区。如果您在任何其他国家/地区购买本软件，则适用该国家/地区的法律。

10 法律效力。本协议描述了特定法律权利。根据贵国法律，您可能拥有其他权利。您还可能拥有从其获得本软件的一方的相应权利。如果您所在的国家/地区的法律不允许，根据这些法律，本协议不会更改您的权利。

11 免责声明。该软件已获得许可” AS-IS。您承担使用它的风险。MICROSOFT 不提供任何明示的担保、保证或条件。根据所在地区的法律，您可能拥有其他本许可证无法更改的消费者权利或法定担保。在当地法律允许的范围内，MICROSOFT 排除对适销性、特定用途的适用性等方面的默示担保。 NON-INFRINGEMENT

   对于澳大利亚 – 您在澳大利亚消费者法下获得了法定担保，本协议中的任何条款都无意影响这些权利。

12 对补救措施和损害赔偿的限制和排除。您只能向微软及其供应商追回的直接损害赔偿金不超过以下金额 U.S。5.00 美元。您无法获得任何其他损害赔偿，包括后果性损害、利润损失、间接损害或事故损害。

   此限制适用于

   - 与第三方 Internet 站点或第三方程序中的软件、服务、内容（包括代码）相关的任何内容；以及
   - 违反合同；违反担保、保证或条件；严格责任；疏忽；或适用法律允许情况下其他侵权行为的索赔。

   即使 Microsoft 已知或者应该知道造成损害的可能性，此条款仍适用。由于您所在的国家/地区可能不允许排除或限制事故损害、后果性损害或其他损害，上述限制或排除可能不适用于您。

10) 基于 Windows 的 Docker 映像 – visualcppbuildtools v 14.0.25420.1

（许可条款可在以下网址获得：https://www.visualstudio.com/license-terms/mt644918/ ）

Microsoft Visual C++ Build Tools

Microsoft 软件许可条款

Microsoft Visual C++ Build Tools

这些许可条款是 Microsoft Corporation（或您所在地的其附属公司）与您达成的协议。它们适用于以上所述软件。这些条款还适用于软件的任意 Microsoft 服务或更新，除非另有其他条款。

如果您遵循这些许可条款，您将拥有以下权利。

1. 安装和使用权利。
   a. 可以有一位用户使用本软件的副本来开发和测试他们的应用程序。
2. DATA。软件可能会收集有关您以及您使用软件的信息，并将信息发送给 Microsoft。Microsoft 可能会使用此信息来提供服务和改进我们的产品及服务。您可以选择退出其中的多种情况，但并非全部，如产品文档中所述。此外，软件中还有一些功能，可能使您能够从应用程序的用户收集数据。如果您使用这些功能在应用程序中启用数据收集，则必须遵循适用的法律，包括向应用程序的用户提供相应的说明。您可以在帮助文档和隐私声明中了解有关数据收集和使用的更多信息，网址为 http://go.microsoft.com/fwlink/？linkid=528096 。您使用软件操作即表明您同意这些做法。
3. 特定组件的条款。
   a. 构建服务器。该软件可能包含 BuildServer.TXT文件中列出的某些 Build Server 组件，BuildeServer 列表中列出的 and/or 任何文件均遵循本 Microsoft 软件许可条款。如果这些项目包含在本软件中，您可以复制并安装到您的构建计算机。您和您组织中的其他人可以在您的构建计算机上使用这些项目，仅用于编译、构建、验证和归档应用程序，或者作为构建过程的一部分运行质量或性能测试。
   b. Microsoft 平台。该软件可能包括来自微软 Windows、微软 Windows 服务器、微软 SQL 服务器、微软 Exchange、微软 Office 和微软的组件 SharePoint。这些组件受独立协议及自己的产品支持政策控制，如位于组件安装目录中或随软件提供的“Licenses”文件夹中的许可条款所述。
   c. 第三方组件。如软件附带 ThirdPartyNotices 文件中所述，该软件可能包括带有单独法律声明或受其他协议约束的第三方组件。即使此类组件受其他协议控制，以下免责声明以及对损害赔偿的限制或排除仍适用。
   d. 程序包管理器。本软件软件可能包含软件包管理器（如 Nuget），它允许您下载其他 Microsoft 和第三方软件包以供您的应用程序使用。这些软件包遵循它们自己的许可证，而不是本协议。Microsoft 不会分发、许可任何第三方软件包或者为其提供任何担保。
4. 许可范围。该软件授予许可，而非销售。本协议仅向您提供使用软件的一些权利。Microsoft 保留所有其他权利。除非适用法律在此限制之外赋予您更多的权利，否则您只能在本协议中明确允许的情况下使用该软件。在使用时，您必须遵守软件中仅允许您以特定方式使用它的任意技术限制。有关更多信息，请参阅 https://docs.microsoft.com/en-us/legal/information-protection/software-license-terms #1-安装和使用权限。您不能
   - 规避软件中的任何技术限制；

- 逆向工程、反编译或反汇编软件，或者尝试这样做，除非且仅限于在第三方许可条款要求的范围内，该条款规定本软件中可能包含的特定开源组件的使用；
- 删除、最小化、阻止或修改 Microsoft 或其供应商的任何通知；
- 以违反法律的方式使用软件；或者
- 共享、发布、租借或租用软件，或者将软件独立托管为解决方案供其他人使用。

5. 出口限制。您必须遵守所有适用于该软件的国内和国际出口法律及法规，这包括对目的地、最终用户和最终用途的限制。有关出口限制的更多信息，请访问（又名ms/exporting）。
6. 支持服务。由于此软件“按原样”提供，我们可能不会为它提供支持服务。
7. 完整协议。本协议以及您使用的补充、更新、 Internet-based 服务和支持服务的条款是软件和支持服务的完整协议。
8. 适用的法律。如果您在美国购买本软件，华盛顿州法律管辖对本协议的解释以及违反协议的索赔，您居住州的法律适用于所有其他索赔。如果您在任何其他国家/地区购买本软件，则适用该国家/地区的法律。
9. 消费者权利；区域差异。本协议描述了特定法律权利。根据所在的州或国家/地区，您可能拥有其他权利，包括消费者权利。除了与 Microsoft 的关系之外，您还可能对与您购买该软件的一方的拥有相应权利。如果您所在的州或国家/地区的法律不允许，本协议不会更改这些其他权利。例如，如果您在以下区域之一购买了本软件，或者有强制性国家/地区法律适用，则以下条款适用于您：
   - 澳大利亚。您在澳大利亚消费者法下获得了法定担保，本协议中的任何内容都无意影响这些权利。
   - 加拿大。如果您在加拿大购买此软件，您可通过关闭自动更新功能、断开设备与 Internet 的连接（但是，在您重新连接到 Internet 时，软件将恢复检查和安全更新）或者卸载软件来停止接收更新。产品文档（如果有）可能会说明如何关闭特定设备或软件的更新。
   - 德国和奥地利。
     - 担保。正确授予许可的软件，将基本按照随该软件所提供的任意 Microsoft 材料中所述执行。但是，Microsoft 不提供任何与所许可软件相关的合同担保。
     - 责任限制。如果发生故意行为、重大过失、基于“产品责任法案”的索赔，以及在发生死亡或人身或身体伤害的情况下，Microsoft 将依照法定法律承担法律责任。

       在遵守上述第 (ii) 条的前提下，如果 Microsoft 违反此类重大合同义务，Microsoft 将仅对轻微过失承担责任，履行这一条有助于本协议的正常履行，违反这一条会危及本协议的用途以及一方可以不断信任的合规性（所谓的“基本义务”）。在其他轻微过失的情况下，Microsoft 不承担轻微过失的责任。
10. 法律效力。本协议描述了特定法律权利。根据您所在州或国家/地区的法律，您可能拥有其他权利。如果您所在州或国家/地区的法律不允许，根据这些法律，本协议不会更改您的权利。在不限制前述

条款的情况下，对于澳大利亚，您在澳大利亚消费者法下获得了法定担保，本协议中的任何条款都无意影响这些权利

11免责声明。该软件已获得许可” AS-IS。您承担使用它的风险。MICROSOFT 不提供任何明示的担保、保证或条件。在当地法律允许的范围内，MICROSOFT 排除对适销性、特定用途的适用性等方面的默示担保。 NON-INFRINGEMENT

12对损害赔偿的限制或排除。您只能向微软及其供应商追回的直接损害赔偿金不超过以下金额 U.S。5.00 美元。您无法获得任何其他损害赔偿，包括后果性损害、利润损失、间接损害或事故损害。

此限制适用于 (a) 第三方 Internet 站点或第三方应用程序上与软件、服务、内容（包括代码）相关的任意内容；(b) 违反合同；违反担保、保证或条件；严格责任；疏忽；或法律允许情况下其他侵权行为的索赔。

即使 Microsoft 已知或者应该知道造成损害的可能性，此条款仍适用。由于您所在的国家/地区可能不允许排除或限制事故损害、后果性损害或其他损害，上述限制或排除可能不适用于您。

11) 基于 Windows 的 Docker 映像 – microsoft-windows-netfx3-ondemand-package.cab

MICROSOFT 软件补充许可条款

Microsoft .NET Framework 3.5 SP1，适用于 Microsoft Windows 操作系统

Microsoft Corporation（或您所在地的其附属公司）向您授予此补充的许可。如果您获得使用 Microsoft Windows 操作系统软件（本补充内容适用）（下称“软件”）的许可，则可以使用本补充内容。如果您未获得软件的许可，就不能使用它。您可以将本补充内容的副本用于软件的每个授予有效许可的副本。

以下许可条款描述了此补充的额外使用条款。软件的这些条款和许可条款适用于您对补充的使用。如果存在冲突，这些补充许可条款适用。

使用本补充内容即表示您接受这些条款。如果您不接受这些条款，请勿使用本补充内容。

如果您遵循这些许可条款，您将拥有以下权利。

1. 补充的支持服务。如 www.support.microsoft所述，微软为该软件提供支持服务。com/common/international.aspx 。

2. Microsoft .NET 基准测试。本软件包括 Windows 操作系统（.NET 组件）的 .NET Framework、Windows Communication Foundation、Windows Presentation Foundation 和 Windows Workflow Foundation 组件。您可以对 .NET 组件执行内部基准测试。只要您符合 http://go.microsoft.com/fwlink/？中规定的条件，您就可以披露.NET 组件的任何基准测试结果 linkid=66406 。

   尽管你可能与微软签订了任何其他协议，但如果你披露此类基准测试结果，微软有权披露其对你的产品进行的与适用.NET 组件竞争的基准测试结果，前提是它符合上规定的相同条件 http://go.microsoft.com/fwlink/？linkid=66406 。

12) 基于 Windows 的 Docker 映像 – dotnet-sdk

（可在以下网址获得 https://github.com/dotnet/core/blob/main/LICENSE.TXT ）

MIT 许可证（MIT）

版权所有 (c) Microsoft Corporation

特此免费允许任何获得本软件和相关文档文件（以下简称“软件”）副本的人不受限制地使用本软件，包括但不限于使用、复制、修改、合并、发布、分发、再许可、 and/or 出售本软件副本的权利，以及允许本软件提供给的人这样做的权利，但须符合以下条件：

上述版权声明和本许可声明应包含在本软件的所有副本或主要部分中。

此软件按“原样”提供，无任何明示或暗示的保证，包括但不限于有关适销性、针对特定目的的适用性和不侵权的保证。任何情况下，作者或版权所有者都不应承担任何索赔、损害赔偿或其他责任，无论是因软件或使用或其他软件处理引起的或与其相关的合同行为、侵权行为或其他行为。

使用 CodeBuild 条件键作为 IAM 服务角色变量来控制构建访问权限

使用 CodeBuild 构建 ARN，您可以使用上下文密钥来限制构建资源的访问权限，从而缩小 CodeBuild 服务角色中的资源访问范围。对于 CodeBuild，可用于控制构建访问行为的密钥是`codebuild:buildArn`和`codebuild:projectArn`。使用构建项目 ARN，您可以验证对资源的调用是否来自特定的构建项目。要验证这一点，请在 IAM 基于身份的策略中使用 `codebuild:buildArn` 或 `codebuild:projectArn` 条件键。

要在策略中使用 `codebuild:buildArn` 或 `codebuild:projectArn` 条件键，请将其作为条件与任何 ARN 条件运算符结合使用。键的值必须是解析为有效 ARN 的 IAM 变量。在下面的示例策略中，支持的唯一访问权限将是使用 `${codebuild:projectArn}` IAM 变量的项目 ARN 访问构建项目。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::bucket-name/${codebuild:projectArn}/*"
        }
    ]
}
```

AWS CodeBuild 条件键

AWS CodeBuild 提供了一组条件密钥，您可以在 IAM 策略中使用这些密钥来对项目和队列等 CodeBuild 资源执行组织策略。条件密钥涵盖了大多数 CodeBuild API 请求上下文，包括网络设置、凭据配置和计算限制。

主题

- 对您的项目和实例集强制执行 VPC 连接设置
- 防止对项目 buildspec 进行未经授权的修改
- 限制构建的计算类型
- 控制环境变量设置
- 在条件键名称中使用变量
- 检查 API 请求中是否存在属性

对您的项目和实例集强制执行 VPC 连接设置

此策略允许调用者在创建 CodeBuild 项目和队列时使用选定的 VPC、子网和安全组。有关多值上下文键的更多信息，请参见Single-valued 与多值上下文键对比。

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "codebuild:CreateProject",
            "codebuild:CreateFleet"
        ],
        "Resource": "*",
        "Condition": {
            "ForAnyValue:StringEquals": {
                "codebuild:vpcConfig.vpcId": [
                    "vpc-01234567890abcdef",
                    "vpc-abcdef01234567890"
                ],
                "codebuild:vpcConfig.subnets": [
                    "subnet-1234abcd",
                    "subnet-5678abcd"
                ],
                "codebuild:vpcConfig.securityGroupIds": [
                    "sg-12345678abcdefghij",
                    "sg-01234567abcdefghij"
                ]
            }
        }
    }]
}
```

防止对项目 buildspec 进行未经授权的修改

此策略不支持调用方覆盖 `buildspecOverride` 字段中的 buildspec。

> **Note**
>
> `codebuild:source.buildspec` 条件键仅支持 Null 运算符来检查 API 字段是否存在。它不评估构建规范的内容。

JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": "codebuild:StartBuild",
        "Resource": "*"
    }, {
        "Effect": "Deny",
        "Action": "codebuild:StartBuild",
        "Resource": "*",
        "Condition": {
            "Null": {
                "codebuild:source.buildspec": "false"
            }
        }
    }]
}
```

限制构建的计算类型

此策略支持创建只能使用 `c5.large` 或 `m5.large` 计算实例类型构建的实例集。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": "codebuild:CreateFleet",
        "Resource": "*",
        "Condition": {
            "ForAnyValue:StringEquals": {
                "codebuild:computeConfiguration.instanceType": ["c5.large",
 "m5.large"]
            }
        }
    }]
}
```

控制环境变量设置

此策略支持调用方将 STAGE 环境变量覆盖为 BETA 或 GAMMA。它还显式拒绝将 STAGE 覆盖为 PRODUCTION，并拒绝覆盖 MY_APP_VERSION 环境变量。有关多值上下文键的信息，请参见Single-valued 与多值上下文键对比。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:StartBuild"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "codebuild:environment.environmentVariables/STAGE.value": [
                        "BETA",
                        "GAMMA"
                    ]
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": [
                "codebuild:StartBuild"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "codebuild:environment.environmentVariables/STAGE.value":
 "PRODUCTION"
                },
                "ForAnyValue:StringEquals": {
                    "codebuild:environment.environmentVariables.name": [
                        "MY_APP_VERSION"
                    ]
                }
```

```
            }
        }
    ]
}
```

在条件键名称中使用变量

您可以在条件键名称中使用变量，例如 `secondarySources/${sourceIdentifier}.location` 和 `secondaryArtifacts/${artifactIdentifier}.location`，其中，您可以在 IAM 策略中指定辅助源或辅助构件标识符。下面的策略支持调用方为辅助源 `mySecondSource` 创建具有特定源位置的项目。

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "codebuild:CreateProject",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "codebuild:secondarySources/mySecondSource.location": "my-
source-location"
                }
            }
        }
    ]
}
```

检查 API 请求中是否存在属性

CodeBuild 支持条件密钥来检查 API 请求中是否存在某些字段。该策略在创建或更新项目时强制执行 VPC 要求。

```
{
    "Version": "2012-10-17"             ,
```

```
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "codebuild:CreateProject",
            "codebuild:UpdateProject"
        ],
        "Resource": "*",
        "Condition": {
            "Null": {
                "codebuild:vpcConfig.vpcId": "false"
            }
        }
    }]
}
```

CodeBuild 使用 AWS SDK 的代码示例

以下代码示例说明如何 CodeBuild 使用 AWS 软件开发套件 (SDK)。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景的上下文查看操作。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

代码示例

- CodeBuild 使用 AWS SDK 的基本示例
  - CodeBuild 使用 AWS SDK 的操作
    - CreateProject与 AWS SDK 或 CLI 配合使用
    - ListBuilds与 AWS SDK 或 CLI 配合使用
    - ListProjects与 AWS SDK 或 CLI 配合使用
    - StartBuild与 AWS SDK 或 CLI 配合使用

CodeBuild 使用 AWS SDK 的基本示例

以下代码示例展示了如何在 AWS SDK 中使用基础知识。 AWS CodeBuild

示例

- CodeBuild 使用 AWS SDK 的操作
  - CreateProject与 AWS SDK 或 CLI 配合使用
  - ListBuilds与 AWS SDK 或 CLI 配合使用
  - ListProjects与 AWS SDK 或 CLI 配合使用
  - StartBuild与 AWS SDK 或 CLI 配合使用

CodeBuild 使用 AWS SDK 的操作

以下代码示例演示了如何使用 AWS 软件开发工具包执行单个 CodeBuild操作。每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关设置和运行代码的说明。

以下示例仅包括最常用的操作。有关完整列表，请参阅 AWS CodeBuild API 参考。

示例

- CreateProject与 AWS SDK 或 CLI 配合使用
- ListBuilds与 AWS SDK 或 CLI 配合使用
- ListProjects与 AWS SDK 或 CLI 配合使用
- StartBuild与 AWS SDK 或 CLI 配合使用

`CreateProject`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `CreateProject`。

CLI

AWS CLI

示例 1：创建 AWS CodeBuild 构建项目

以下`create-project`示例使用 S3 存储桶中的源文件创建 CodeBuild 构建项目

```
aws codebuild create-project \
    --name "my-demo-project" \
    --source "{\"type\": \"S3\",\"location\": \"codebuild-us-west-2-123456789012-input-bucket/my-source.zip\"}" \
    --artifacts {"\"type\": \"S3\",\"location\": \"codebuild-us-west-2-123456789012-output-bucket\""} \
    --environment "{\"type\": \"LINUX_CONTAINER\",\"image\": \"aws/codebuild/standard:1.0\",\"computeType\": \"BUILD_GENERAL1_SMALL\"}" \
    --service-role "arn:aws:iam::123456789012:role/service-role/my-codebuild-service-role"
```

输出：

```
{
    "project": {
        "arn": "arn:aws:codebuild:us-west-2:123456789012:project/my-demo-project",
        "name": "my-cli-demo-project",
        "encryptionKey": "arn:aws:kms:us-west-2:123456789012:alias/aws/s3",
        "serviceRole": "arn:aws:iam::123456789012:role/service-role/my-codebuild-service-role",
        "lastModified": 1556839783.274,
        "badge": {
```

```
            "badgeEnabled": false
        },
        "queuedTimeoutInMinutes": 480,
        "environment": {
            "image": "aws/codebuild/standard:1.0",
            "computeType": "BUILD_GENERAL1_SMALL",
            "type": "LINUX_CONTAINER",
            "imagePullCredentialsType": "CODEBUILD",
            "privilegedMode": false,
            "environmentVariables": []
        },
        "artifacts": {
            "location": "codebuild-us-west-2-123456789012-output-bucket",
            "name": "my-cli-demo-project",
            "namespaceType": "NONE",
            "type": "S3",
            "packaging": "NONE",
            "encryptionDisabled": false
        },
        "source": {
            "type": "S3",
            "location": "codebuild-us-west-2-123456789012-input-bucket/my-
source.zip",
            "insecureSsl": false
        },
        "timeoutInMinutes": 60,
        "cache": {
            "type": "NO_CACHE"
        },
        "created": 1556839783.274
    }
}
```

示例 2：使用 JSON 输入文件作为参数创建 AWS CodeBuild 构建项目

以下`create-project`示例通过在 JSON 输入文件中传递所有必需的参数来创建 CodeBuild 构建项目。通过仅使用 `--generate-cli-skeleton parameter` 运行命令来创建输入文件模板。

```
aws codebuild create-project --cli-input-json file://create-project.json
```

输入 JSON 文件 `create-project.json` 包含以下内容：

```
{
    "name": "codebuild-demo-project",
    "source": {
        "type": "S3",
        "location": "codebuild-region-ID-account-ID-input-bucket/MessageUtil.zip"
    },
    "artifacts": {
        "type": "S3",
        "location": "codebuild-region-ID-account-ID-output-bucket"
    },
    "environment": {
        "type": "LINUX_CONTAINER",
        "image": "aws/codebuild/standard:1.0",
        "computeType": "BUILD_GENERAL1_SMALL"
    },
    "serviceRole": "serviceIAMRole"
}
```

输出：

```
{
    "project": {
        "name": "codebuild-demo-project",
        "serviceRole": "serviceIAMRole",
        "tags": [],
        "artifacts": {
            "packaging": "NONE",
            "type": "S3",
            "location": "codebuild-region-ID-account-ID-output-bucket",
            "name": "message-util.zip"
        },
        "lastModified": 1472661575.244,
        "timeoutInMinutes": 60,
        "created": 1472661575.244,
        "environment": {
            "computeType": "BUILD_GENERAL1_SMALL",
            "image": "aws/codebuild/standard:1.0",
            "type": "LINUX_CONTAINER",
            "environmentVariables": []
        },
        "source": {
            "type": "S3",
```

```
            "location": "codebuild-region-ID-account-ID-input-bucket/
MessageUtil.zip"
        },
        "encryptionKey": "arn:aws:kms:region-ID:account-ID:alias/aws/s3",
        "arn": "arn:aws:codebuild:region-ID:account-ID:project/codebuild-demo-
project"
    }
}
```

有关更多信息，请参阅《AWS CodeBuild 用户指南》中的创建构建项目 (AWS CLI)。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考CreateProject中的。

JavaScript

适用于 JavaScript (v3) 的软件开发工具包

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创建项目。

```
import {
  ArtifactsType,
  CodeBuildClient,
  ComputeType,
  CreateProjectCommand,
  EnvironmentType,
  SourceType,
} from "@aws-sdk/client-codebuild";

// Create the AWS CodeBuild project.
export const createProject = async (
  projectName = "MyCodeBuilder",
  roleArn = "arn:aws:iam::xxxxxxxxxxxx:role/CodeBuildAdmin",
  buildOutputBucket = "xxxx",
  githubUrl = "https://...",
) => {
  const codeBuildClient = new CodeBuildClient({});
```

```
  const response = await codeBuildClient.send(
    new CreateProjectCommand({
      artifacts: {
        // The destination of the build artifacts.
        type: ArtifactsType.S3,
        location: buildOutputBucket,
      },
      // Information about the build environment. The combination of
 "computeType" and "type" determines the
      // requirements for the environment such as CPU, memory, and disk space.
      environment: {
        // Build environment compute types.
        // https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-ref-
compute-types.html
        computeType: ComputeType.BUILD_GENERAL1_SMALL,
        // Docker image identifier.
        // See https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-
ref-available.html
        image: "aws/codebuild/standard:7.0",
        // Build environment type.
        type: EnvironmentType.LINUX_CONTAINER,
      },
      name: projectName,
      // A role ARN with permission to create a CodeBuild project, write to the
 artifact location, and write CloudWatch logs.
      serviceRole: roleArn,
      source: {
        // The type of repository that contains the source code to be built.
        type: SourceType.GITHUB,
        // The location of the repository that contains the source code to be
 built.
        location: githubUrl,
      },
    }),
  );
  console.log(response);
  //   {
  //     '$metadata': {
  //       httpStatusCode: 200,
  //       requestId: 'b428b244-777b-49a6-a48d-5dffedced8e7',
  //       extendedRequestId: undefined,
  //       cfId: undefined,
  //       attempts: 1,
```

```
  //       totalRetryDelay: 0
  //     },
  //     project: {
  //       arn: 'arn:aws:codebuild:us-east-1:xxxxxxxxxxxx:project/MyCodeBuilder',
  //       artifacts: {
  //         encryptionDisabled: false,
  //         location: 'xxxxxx-xxxxxxx-xxxxxx',
  //         name: 'MyCodeBuilder',
  //         namespaceType: 'NONE',
  //         packaging: 'NONE',
  //         type: 'S3'
  //       },
  //       badge: { badgeEnabled: false },
  //       cache: { type: 'NO_CACHE' },
  //       created: 2023-08-18T14:46:48.979Z,
  //       encryptionKey: 'arn:aws:kms:us-east-1:xxxxxxxxxxxx:alias/aws/s3',
  //       environment: {
  //         computeType: 'BUILD_GENERAL1_SMALL',
  //         environmentVariables: [],
  //         image: 'aws/codebuild/standard:7.0',
  //         imagePullCredentialsType: 'CODEBUILD',
  //         privilegedMode: false,
  //         type: 'LINUX_CONTAINER'
  //       },
  //       lastModified: 2023-08-18T14:46:48.979Z,
  //       name: 'MyCodeBuilder',
  //       projectVisibility: 'PRIVATE',
  //       queuedTimeoutInMinutes: 480,
  //       serviceRole: 'arn:aws:iam::xxxxxxxxxxxx:role/CodeBuildAdmin',
  //       source: {
  //         insecureSsl: false,
  //         location: 'https://...',
  //         reportBuildStatus: false,
  //         type: 'GITHUB'
  //       },
  //       timeoutInMinutes: 60
  //     }
  //   }
  return response;
};
```

- 有关更多信息，请参阅《适用于 JavaScript 的 AWS SDK 开发人员指南》https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/client/codebuild/。
- 有关 API 的详细信息，请参阅 适用于 JavaScript 的 AWS SDK API 参考CreateProject中的。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`ListBuilds`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `ListBuilds`。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
//! List the CodeBuild builds.
/*!
  \param sortType: 'SortOrderType' type.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::CodeBuild::listBuilds(Aws::CodeBuild::Model::SortOrderType sortType,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::CodeBuild::CodeBuildClient codeBuildClient(clientConfiguration);

    Aws::CodeBuild::Model::ListBuildsRequest listBuildsRequest;
    listBuildsRequest.SetSortOrder(sortType);

    Aws::String nextToken; // Used for pagination.

    do {
        if (!nextToken.empty()) {
            listBuildsRequest.SetNextToken(nextToken);
        }
```

```
       Aws::CodeBuild::Model::ListBuildsOutcome listBuildsOutcome =
codeBuildClient.ListBuilds(
               listBuildsRequest);

       if (listBuildsOutcome.IsSuccess()) {
           const Aws::Vector<Aws::String> &ids =
listBuildsOutcome.GetResult().GetIds();
           if (!ids.empty()) {

               std::cout << "Information about each build:" << std::endl;
               Aws::CodeBuild::Model::BatchGetBuildsRequest getBuildsRequest;
               getBuildsRequest.SetIds(listBuildsOutcome.GetResult().GetIds());
               Aws::CodeBuild::Model::BatchGetBuildsOutcome getBuildsOutcome =
codeBuildClient.BatchGetBuilds(
                       getBuildsRequest);

               if (getBuildsOutcome.IsSuccess()) {
                   const Aws::Vector<Aws::CodeBuild::Model::Build> &builds =
getBuildsOutcome.GetResult().GetBuilds();
                   std::cout << builds.size() << " build(s) found." <<
std::endl;
                   for (auto val: builds) {
                       std::cout << val.GetId() << std::endl;
                   }
               } else {
                   std::cerr << "Error getting builds"
                             << getBuildsOutcome.GetError().GetMessage() <<
std::endl;
                   return false;
               }
           } else {
               std::cout << "No builds found." << std::endl;
           }

           // Get the next token for pagination.

           nextToken = listBuildsOutcome.GetResult().GetNextToken();
       } else {
           std::cerr << "Error listing builds"
                     << listBuildsOutcome.GetError().GetMessage()
                     << std::endl;
           return false;
       }
```

```
    } while (!nextToken.

            empty()

            );

    return true;
}
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK API 参考ListBuilds中的。

CLI

AWS CLI

获取构 AWS CodeBuild 建 ID 列表。

以下list-builds示例获取按升序排序的 CodeBuild ID 列表。

```
aws codebuild list-builds --sort-order ASCENDING
```

输出包括一个 nextToken 值，该值表示还有更多可用的输出。

```
{
    "nextToken": "4AEA6u7J...The full token has been omitted for
 brevity...MzY2OA==",
    "ids": [
        "codebuild-demo-project:815e755f-bade-4a7e-80f0-efe51EXAMPLE"
        "codebuild-demo-project:84a7f3d1-d40e-4956-b4cf-7a9d4EXAMPLE"
            ... The full list of build IDs has been omitted for brevity ...
        "codebuild-demo-project:931d0b72-bf6f-4040-a472-5c707EXAMPLE"
    ]
}
```

再次运行此命令并提供上一个响应中的 nextToken 值作为参数，从而获取输出的下一部分。重复此操作，直到在响应中不再收到 nextToken 值。

```
aws codebuild list-builds --sort-order ASCENDING --next-
token 4AEA6u7J...The full token has been omitted for brevity...MzY2OA==
```

输出的下一部分：

```
{
    "ids": [
        "codebuild-demo-project:49015049-21cf-4b50-9708-df115EXAMPLE",
        "codebuild-demo-project:543e7206-68a3-46d6-a4da-759abEXAMPLE",
            ... The full list of build IDs has been omitted for brevity ...
        "codebuild-demo-project:c282f198-4582-4b38-bdc0-26f96EXAMPLE"
    ]
}
```

有关更多信息，请参阅《AWS CodeBuild 用户指南》中的查看构建 ID 列表 (AWS CLI)

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ListBuilds中的。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`ListProjects`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `ListProjects`。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
//! List the CodeBuild projects.
/*!
  \param sortType: 'SortOrderType' type.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::CodeBuild::listProjects(Aws::CodeBuild::Model::SortOrderType
 sortType,
                                     const Aws::Client::ClientConfiguration
 &clientConfiguration) {
```

```
    Aws::CodeBuild::CodeBuildClient codeBuildClient(clientConfiguration);

    Aws::CodeBuild::Model::ListProjectsRequest listProjectsRequest;
    listProjectsRequest.SetSortOrder(sortType);

    Aws::String nextToken; // Next token for pagination.
    Aws::Vector<Aws::String> allProjects;

    do {
        if (!nextToken.empty()) {
            listProjectsRequest.SetNextToken(nextToken);
        }

        Aws::CodeBuild::Model::ListProjectsOutcome outcome =
 codeBuildClient.ListProjects(
                listProjectsRequest);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::String> &projects =
 outcome.GetResult().GetProjects();
            allProjects.insert(allProjects.end(), projects.begin(),
 projects.end());
            nextToken = outcome.GetResult().GetNextToken();
        }

        else {
            std::cerr << "Error listing projects" <<
 outcome.GetError().GetMessage()
                      << std::endl;
        }

    } while (!nextToken.empty());

    std::cout << allProjects.size() << " project(s) found." << std::endl;
    for (auto project: allProjects) {
        std::cout << project << std::endl;
    }

    return true;
}
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK API 参考[ListProjects](#)中的。

CLI

AWS CLI

获取 AWS CodeBuild 构建项目名称列表。

以下`list-projects`示例获取按名称升序排序的 CodeBuild 构建项目列表。

```
aws codebuild list-projects --sort-by NAME --sort-order ASCENDING
```

输出包括一个 `nextToken` 值，该值表示还有更多可用的输出。

```
{
    "nextToken": "Ci33ACF6...The full token has been omitted for brevity...U
+AkMx8=",
    "projects": [
        "codebuild-demo-project",
        "codebuild-demo-project2",
            ... The full list of build project names has been omitted for
 brevity ...
        "codebuild-demo-project99"
    ]
}
```

再次运行此命令并提供来自上一个响应的 `nextToken` 值作为参数，从而获取输出的下一部分。重复此操作，直到在响应中不再收到 `nextToken` 值。

```
aws codebuild list-projects  --sort-by NAME --sort-order ASCENDING --next-
token Ci33ACF6...The full token has been omitted for brevity...U+AkMx8=

{
    "projects": [
        "codebuild-demo-project100",
        "codebuild-demo-project101",

  ... The full list of build project names has been omitted for brevity ...
        "codebuild-demo-project122"

  ]
}
```

有关更多信息，请参阅《AWS CodeBuild 用户指南》中的 “查看构建项目名称列表 (AWS CLI)”。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ListProjects中的。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`StartBuild`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `StartBuild`。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
//! Start an AWS CodeBuild project build.
/*!
  \param projectName: A CodeBuild project name.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::CodeBuild::startBuild(const Aws::String &projectName,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::CodeBuild::CodeBuildClient codeBuildClient(clientConfiguration);

    Aws::CodeBuild::Model::StartBuildRequest startBuildRequest;
    startBuildRequest.SetProjectName(projectName);

    Aws::CodeBuild::Model::StartBuildOutcome outcome =
 codeBuildClient.StartBuild(
            startBuildRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully started build" << std::endl;
```

```
        std::cout << "Build ID: " << outcome.GetResult().GetBuild().GetId()
                  << std::endl;
    }

    else {
        std::cerr << "Error starting build" << outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 适用于 C++ 的 AWS SDK API 参考StartBuild中的。

CLI

AWS CLI

开始运行 AWS CodeBuild 构建项目的构建。

以下start-build示例为指定 CodeBuild 项目启动构建。构建会覆盖有关在超时前可在队列中等待的分钟数的项目设置，还会覆盖项目的构件设置。

```
aws codebuild start-build \
    --project-name "my-demo-project" \
    --queued-timeout-in-minutes-override 5 \
    --artifacts-override {"\"type\": \"S3\",\"location\":
 \"arn:aws:s3:::artifacts-override\",\"overrideArtifactName\":true"}
```

输出：

```
{
    "build": {
        "serviceRole": "arn:aws:iam::123456789012:role/service-role/my-codebuild-
service-role",
        "buildStatus": "IN_PROGRESS",
        "buildComplete": false,
        "projectName": "my-demo-project",
        "timeoutInMinutes": 60,
        "source": {
            "insecureSsl": false,
```

```
            "type": "S3",
            "location": "codebuild-us-west-2-123456789012-input-bucket/my-
source.zip"
        },
        "queuedTimeoutInMinutes": 5,
        "encryptionKey": "arn:aws:kms:us-west-2:123456789012:alias/aws/s3",
        "currentPhase": "QUEUED",
        "startTime": 1556905683.568,
        "environment": {
            "computeType": "BUILD_GENERAL1_MEDIUM",
            "environmentVariables": [],
            "type": "LINUX_CONTAINER",
            "privilegedMode": false,
            "image": "aws/codebuild/standard:1.0",
            "imagePullCredentialsType": "CODEBUILD"
        },
        "phases": [
            {
                "phaseStatus": "SUCCEEDED",
                "startTime": 1556905683.568,
                "phaseType": "SUBMITTED",
                "durationInSeconds": 0,
                "endTime": 1556905684.524
            },
            {
                "startTime": 1556905684.524,
                "phaseType": "QUEUED"
            }
        ],
        "logs": {
            "deepLink": "https://console.aws.amazon.com/cloudwatch/home?
region=us-west-2#logEvent:group=null;stream=null"
        },
        "artifacts": {
            "encryptionDisabled": false,
            "location": "arn:aws:s3:::artifacts-override/my-demo-project",
            "overrideArtifactName": true
        },
        "cache": {
            "type": "NO_CACHE"
        },
        "id": "my-demo-project::12345678-a1b2-c3d4-e5f6-11111EXAMPLE",
        "initiator": "my-aws-account-name",
```

```
        "arn": "arn:aws:codebuild:us-west-2:123456789012:build/my-demo-
project::12345678-a1b2-c3d4-e5f6-11111EXAMPLE"
    }
}
```

有关更多信息，请参阅《AWS CodeBuild 用户指南》中的运行构建 (AWS CLI)。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考StartBuild中的。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅将此服务与 AWS SDK。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

公共构建 API

AWS CodeBuild 控制台和公共生成结果网站使用公共构建 API 来获取有关公共构建项目的信息。有关更多信息，请参阅 the section called “获取公共构建项目 URL”。

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含 CodeBuild 公共构建 API。

主题

- 公共构建操作
- 公共版本数据类型

公共构建操作

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含 CodeBuild 公共构建 API。

CodeBuild 公共构建 API 支持以下操作：

操作

- DescribeBuildBatchesForPublicProject
- DescribeBuildsForPublicProject
- GetCloudWatchLogsForPublicBuild
- GetPresignedUrlsForPublicBuild
- GetPublicBuild
- GetPublicBuildBatch
- GetPublicProject

DescribeBuildBatchesForPublicProject

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含此 API 元素。

请求语法

```
{
   "filter": {
      "status": "string"
   },
   "maxResults": number,
   "nextToken": "string",
   "publicProjectAlias": "string",
   "sortOrder": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅常用参数。

请求接受采用 JSON 格式的以下数据。

> **Note**
>
> 下表中首先描述了必需参数。

公众 ProjectAlias

类型：字符串

长度限制：最小长度为 1。长度上限为 1024。

模式：^[0-9a-zA-Z%+=]+$

是否必需：是

过滤

指定检索批处理版本时的筛选条件。

类型：BuildBatchFilter 对象

必需：否

最大结果

类型：整数

有效范围：最小值为 1。最大值为 100。

必需：否

nextToken

类型：字符串

必需：否

排序顺序

类型：字符串

有效值: ASCENDING | DESCENDING

必需：否

响应语法

```
{
   "nextToken": "string",
   "publicBuildBatches": [
      {
         "buildBatchNumber": number,
         "buildBatchStatus": "string",
         "endTime": number,
         "publicBuildBatchAlias": "string",
         "sourceVersion": "string",
         "startTime": number
      }
   ]
```

```
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

nextToken

类型：字符串

公众 BuildBatches

类型：BuildBatchForDescribeBuildBatchesPublic 对象数组

错误

有关所有操作中常见的错误的信息，请参阅常见错误。

InvalidInputException

HTTP 状态代码：400

ResourceNotFoundException

HTTP 状态代码：400

DescribeBuildsForPublicProject

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含此 API 元素。

请求语法

```
{
   "maxResults": number,
   "nextToken": "string",
   "publicProjectAlias": "string",
   "sortOrder": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅常用参数。

请求接受采用 JSON 格式的以下数据。

> **Note**
>
> 下表中首先描述了必需参数。

公众 ProjectAlias

类型：字符串

长度限制：最小长度为 1。长度上限为 1024。

模式：^[0-9a-zA-Z%+=]+$

是否必需：是

最大结果

类型：整数

有效范围：最小值为 1。最大值为 100。

必需：否

nextToken

类型：字符串

必需：否

排序顺序

类型：字符串

有效值: ASCENDING | DESCENDING

必需：否

响应语法

```
{
   "builds": [
      {
         "artifacts": {
            "artifactIdentifier": "string",
            "packaging": "string",
            "type": "string"
         },
         "buildComplete": boolean,
         "buildNumber": number,
         "buildStatus": "string",
         "endTime": number,
         "environment": {
            "computeType": "string",
            "environmentVariables": [
               {
                  "name": "string",
                  "type": "string",
                  "value": "string"
               }
            ],
            "image": "string",
            "type": "string"
         },
         "id": "string",
         "initiator": "string",
```

```
        "logsStatus": {
           "cloudWatchLogsStatus": "string",
           "s3LogsStatus": "string"
        },
        "phases": [
           {
              "contexts": [
                 {
                    "message": "string",
                    "statusCode": "string"
                 }
              ],
              "durationInSeconds": number,
              "endTime": number,
              "phaseStatus": "string",
              "phaseType": "string",
              "startTime": number
           }
        ],
        "projectName": "string",
        "queuedTimeoutInMinutes": number,
        "resolvedSourceVersion": "string",
        "secondaryArtifacts": [
           {
              "artifactIdentifier": "string",
              "packaging": "string",
              "type": "string"
           }
        ],
        "secondarySources": [
           {
              "buildspec": "string",
              "gitCloneDepth": number,
              "gitSubmodulesConfig": {
                 "fetchSubmodules": boolean
              },
              "location": "string",
              "sourceIdentifier": "string",
              "type": "string"
           }
        ],
        "secondarySourceVersions": [
           {
              "sourceIdentifier": "string",
```

```
                "sourceVersion": "string"
            }
         ],
         "source": {
            "buildspec": "string",
            "gitCloneDepth": number,
            "gitSubmodulesConfig": {
               "fetchSubmodules": boolean
            },
            "location": "string",
            "sourceIdentifier": "string",
            "type": "string"
         },
         "sourceVersion": "string",
         "startTime": number,
         "timeoutInMinutes": number
      }
   ],
   "nextToken": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

构建

类型：PublicBuild 对象数组

nextToken

类型：字符串

错误

有关所有操作常见错误的信息，请参阅常见错误。

InvalidInputException

HTTP 状态代码：400

ResourceNotFoundException

HTTP 状态代码：400

GetCloudWatchLogsForPublicBuild

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含此 API 元素。

请求语法

```
{
   "maxResult": number,
   "nextToken": "string",
   "publicBuildAlias": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅常用参数。

请求接受采用 JSON 格式的以下数据。

> **Note**
>
> 下表中首先描述了必需参数。

公众 BuildAlias

类型：字符串

长度限制：最小长度为 1。长度上限为 1024。

模式：`^[0-9a-zA-Z%+=]+:\p{XDigit}{8}(-\p{XDigit}{4}){3}-\p{XDigit}{12}$`

是否必需：是

最大结果

类型：整数

有效范围：最小值为 1。最大值为 10000。

必需：否

nextToken

类型：字符串

必需：否

响应语法

```
{
   "logs": [ "string" ],
   "nextToken": "string"
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

日志

类型：字符串数组

nextToken

类型：字符串

长度限制：最小长度为 1。

错误

有关所有操作中常见的错误的信息，请参阅常见错误。

InvalidInputException

HTTP 状态代码：400

ResourceNotFoundException

HTTP 状态代码：400

GetPresignedUrlsForPublicBuild

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含此 API 元素。

请求语法

```
{
   "publicBuildAlias": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅常用参数。

请求接受采用 JSON 格式的以下数据。

> **Note**
>
> 下表中首先描述了必需参数。

公众 BuildAlias

类型：字符串

长度限制：最小长度为 1。长度上限为 1024。

模式：`^[0-9a-zA-Z%+=]+:\p{XDigit}{8}(-\p{XDigit}{4}){3}-\p{XDigit}{12}$`

必需：是

响应语法

```
{
   "artifacts": {
```

```
      "expiredAt": number,
      "identifier": "string",
      "md5Checksum": "string",
      "presignedUrl": "string",
      "s3Arn": "string",
      "sha256Checksum": "string"
   },
   "log": {
      "expiredAt": number,
      "identifier": "string",
      "md5Checksum": "string",
      "presignedUrl": "string",
      "s3Arn": "string",
      "sha256Checksum": "string"
   },
   "secondaryArtifacts": [
      {
         "expiredAt": number,
         "identifier": "string",
         "md5Checksum": "string",
         "presignedUrl": "string",
         "s3Arn": "string",
         "sha256Checksum": "string"
      }
   ]
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

文物

类型：S3可下载 对象

日志

类型：S3可下载 对象

次要文物

类型：S3可下载 对象数组

错误

有关所有操作常见错误的信息，请参阅常见错误。

InvalidInputException

HTTP 状态代码：400

ResourceNotFoundException

HTTP 状态代码：400

GetPublicBuild

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含此 API 元素。

请求语法

```
{
   "publicBuildAlias": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅常用参数。

请求接受采用 JSON 格式的以下数据。

> **Note**
>
> 下表中首先描述了必需参数。

公众 BuildAlias

类型：字符串

长度限制：最小长度为 1。长度上限为 1024。

模式：`^[0-9a-zA-Z%+=]+:\p{XDigit}{8}(-\p{XDigit}{4}){3}-\p{XDigit}{12}$`

必需：是

响应语法

```
{
   "build": {
      "artifacts": {
         "artifactIdentifier": "string",
         "packaging": "string",
```

```
      "type": "string"
   },
   "buildComplete": boolean,
   "buildNumber": number,
   "buildStatus": "string",
   "endTime": number,
   "environment": {
      "computeType": "string",
      "environmentVariables": [
         {
            "name": "string",
            "type": "string",
            "value": "string"
         }
      ],
      "image": "string",
      "type": "string"
   },
   "id": "string",
   "initiator": "string",
   "logsStatus": {
      "cloudWatchLogsStatus": "string",
      "s3LogsStatus": "string"
   },
   "phases": [
      {
         "contexts": [
            {
               "message": "string",
               "statusCode": "string"
            }
         ],
         "durationInSeconds": number,
         "endTime": number,
         "phaseStatus": "string",
         "phaseType": "string",
         "startTime": number
      }
   ],
   "projectName": "string",
   "queuedTimeoutInMinutes": number,
   "resolvedSourceVersion": "string",
   "secondaryArtifacts": [
      {
```

```
            "artifactIdentifier": "string",
            "packaging": "string",
            "type": "string"
         }
      ],
      "secondarySources": [
         {
            "buildspec": "string",
            "gitCloneDepth": number,
            "gitSubmodulesConfig": {
               "fetchSubmodules": boolean
            },
            "location": "string",
            "sourceIdentifier": "string",
            "type": "string"
         }
      ],
      "secondarySourceVersions": [
         {
            "sourceIdentifier": "string",
            "sourceVersion": "string"
         }
      ],
      "source": {
         "buildspec": "string",
         "gitCloneDepth": number,
         "gitSubmodulesConfig": {
            "fetchSubmodules": boolean
         },
         "location": "string",
         "sourceIdentifier": "string",
         "type": "string"
      },
      "sourceVersion": "string",
      "startTime": number,
      "timeoutInMinutes": number
   }
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

建造

类型：PublicBuild 对象

错误

有关所有操作中常见的错误的信息，请参阅常见错误。

InvalidInputException

HTTP 状态代码：400

ResourceNotFoundException

HTTP 状态代码：400

GetPublicBuildBatch

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含此 API 元素。

请求语法

```
{
   "publicBuildBatchAlias": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅常用参数。

请求接受采用 JSON 格式的以下数据。

> **Note**
>
> 下表中首先描述了必需参数。

公众 BuildBatchAlias

类型：字符串

长度限制：最小长度为 1。长度上限为 1024。

模式：`^[0-9a-zA-Z%+=]+:[a-z0-9]{8}(-[a-z0-9]{4}){3}-[a-z0-9]{12}$`

必需：是

响应语法

```
{
   "publicBuildBatch": {
      "artifacts": {
         "artifactIdentifier": "string",
         "packaging": "string",
```

```
            "type": "string"
         },
         "buildBatchConfig": {
            "combineArtifacts": boolean,
            "restrictions": {
               "computeTypesAllowed": [ "string" ],
               "maximumBuildsAllowed": number
            },
            "timeoutInMins": number
         },
         "buildBatchNumber": number,
         "buildBatchStatus": "string",
         "buildGroups": [
            {
               "currentBuildSummary": {
                  "buildStatus": "string",
                  "primaryArtifact": {
                     "identifier": "string",
                     "location": "string",
                     "type": "string"
                  },
                  "publicBuildAlias": "string",
                  "requestedOn": number,
                  "secondaryArtifacts": [
                     {
                        "identifier": "string",
                        "location": "string",
                        "type": "string"
                     }
                  ]
               },
               "dependsOn": [ "string" ],
               "identifier": "string",
               "ignoreFailure": boolean,
               "priorBuildSummaryList": [
                  {
                     "buildStatus": "string",
                     "primaryArtifact": {
                        "identifier": "string",
                        "location": "string",
                        "type": "string"
                     },
                     "publicBuildAlias": "string",
                     "requestedOn": number,
```

```
                "secondaryArtifacts": [
                   {
                      "identifier": "string",
                      "location": "string",
                      "type": "string"
                   }
                ]
             }
          ]
       }
    ],
    "buildTimeoutInMinutes": number,
    "complete": boolean,
    "currentPhase": "string",
    "endTime": number,
    "environment": {
       "computeType": "string",
       "environmentVariables": [
          {
             "name": "string",
             "type": "string",
             "value": "string"
          }
       ],
       "image": "string",
       "type": "string"
    },
    "id": "string",
    "initiator": "string",
    "logsStatus": {
       "cloudWatchLogsStatus": "string",
       "s3LogsStatus": "string"
    },
    "phases": [
       {
          "contexts": [
             {
                "message": "string",
                "statusCode": "string"
             }
          ],
          "durationInSeconds": number,
          "endTime": number,
          "phaseStatus": "string",
```

```
            "phaseType": "string",
            "startTime": number
         }
      ],
      "projectName": "string",
      "publicBuildBatchAlias": "string",
      "queuedTimeoutInMinutes": number,
      "resolvedSourceVersion": "string",
      "secondaryArtifacts": [
         {
            "artifactIdentifier": "string",
            "packaging": "string",
            "type": "string"
         }
      ],
      "secondarySources": [
         {
            "buildspec": "string",
            "gitCloneDepth": number,
            "gitSubmodulesConfig": {
               "fetchSubmodules": boolean
            },
            "location": "string",
            "sourceIdentifier": "string",
            "type": "string"
         }
      ],
      "secondarySourceVersions": [
         {
            "sourceIdentifier": "string",
            "sourceVersion": "string"
         }
      ],
      "source": {
         "buildspec": "string",
         "gitCloneDepth": number,
         "gitSubmodulesConfig": {
            "fetchSubmodules": boolean
         },
         "location": "string",
         "sourceIdentifier": "string",
         "type": "string"
      },
      "sourceVersion": "string",
```

```
        "startTime": number
    }
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

公众 BuildBatch

类型：PublicBuildBatch 对象

错误

有关所有操作中常见的错误的信息，请参阅常见错误。

InvalidInputException

HTTP 状态代码：400

ResourceNotFoundException

HTTP 状态代码：400

GetPublicProject

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含此 API 元素。

请求语法

```
{
   "publicProjectAlias": "string"
}
```

请求参数

有关所有操作的通用参数的信息，请参阅常用参数。

请求接受采用 JSON 格式的以下数据。

> **Note**
>
> 下表中首先描述了必需参数。

公众 ProjectAlias

类型：字符串

长度限制：最小长度为 1。长度上限为 1024。

模式：^[0-9a-zA-Z%+=]+$

必需：是

响应语法

```
{
   "project": {
      "artifacts": {
         "artifactIdentifier": "string",
         "location": "string",
```

```
      "type": "string"
   },
   "buildBatchConfig": {
      "combineArtifacts": boolean,
      "restrictions": {
         "computeTypesAllowed": [ "string" ],
         "maximumBuildsAllowed": number
      },
      "timeoutInMins": number
   },
   "concurrentBuildLimit": number,
   "description": "string",
   "environment": {
      "computeType": "string",
      "environmentVariables": [
         {
            "name": "string",
            "type": "string",
            "value": "string"
         }
      ],
      "image": "string",
      "type": "string"
   },
   "name": "string",
   "queuedTimeoutInMinutes": number,
   "secondaryArtifacts": [
      {
         "artifactIdentifier": "string",
         "location": "string",
         "type": "string"
      }
   ],
   "secondarySources": [
      {
         "buildspec": "string",
         "gitCloneDepth": number,
         "gitSubmodulesConfig": {
            "fetchSubmodules": boolean
         },
         "location": "string",
         "sourceIdentifier": "string",
         "type": "string"
      }
```

```
      ],
      "secondarySourceVersions": [
         {
            "sourceIdentifier": "string",
            "sourceVersion": "string"
         }
      ],
      "source": {
         "buildspec": "string",
         "gitCloneDepth": number,
         "gitSubmodulesConfig": {
            "fetchSubmodules": boolean
         },
         "location": "string",
         "sourceIdentifier": "string",
         "type": "string"
      },
      "sourceVersion": "string",
      "timeoutInMinutes": number,
      "webhook": {
         "branchFilter": "string",
         "buildType": "string",
         "filterGroups": [
            [
               {
                  "excludeMatchedPattern": boolean,
                  "pattern": "string",
                  "type": "string"
               }
            ]
         ],
         "payloadUrl": "string",
         "url": "string"
      }
   }
}
```

响应元素

如果此操作成功，则该服务将会发送回 HTTP 200 响应。

服务以 JSON 格式返回以下数据。

项目

类型：PublicProject 对象

错误

有关所有操作中常见的错误的信息，请参阅常见错误。

InvalidInputException

HTTP 状态代码：400

ResourceNotFoundException

HTTP 状态代码：400

公共版本数据类型

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含 CodeBuild 公共构建 API。

CodeBuild 公共构建 API 支持以下数据类型：

数据类型

- BuildBatchForDescribeBuildBatchesPublic
- PublicBuild
- PublicBuildArtifacts
- PublicBuildBatch
- PublicBuildGroup
- PublicBuildSummary
- PublicLogsStatus
- PublicProject
- PublicProjectArtifacts
- PublicProjectBuildBatchConfig

- PublicProjectEnvironment
- PublicProjectSource
- PublicWebhook
- S3可下载

BuildBatchForDescribeBuildBatchesPublic

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含此 API 元素。

内容

> **Note**
>
> 下表中首先描述了必需参数。

buildBatchNumber

类型：长整型

必需：否

buildBatchStatus

类型：字符串

有效值: `PENDING | SUCCEEDED | FAILED | FAULT | TIMED_OUT | IN_PROGRESS | STOPPED`

必需：否

endTime

类型：时间戳

必需：否

公众 BuildBatchAlias

类型：字符串

长度限制：最小长度为 1。长度上限为 1024。

模式：`^[0-9a-zA-Z%+=]+:[a-z0-9]{8}(-[a-z0-9]{4}){3}-[a-z0-9]{12}$`

必需：否

sourceVersion

类型：字符串

长度限制：最小长度为 1。

必需：否

startTime

类型：时间戳

必需：否

PublicBuild

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含此 API 元素。

内容

> **Note**
>
> 下表中首先描述了必需参数。

构件

类型：PublicBuildArtifacts 对象

必需：否

构建完成

类型：布尔值

必需：否

内部版本号

类型：长整型

必需：否

构建状态

类型：字符串

有效值：`PENDING | SUCCEEDED | FAILED | FAULT | TIMED_OUT | IN_PROGRESS | STOPPED`

必需：否

endTime

类型：时间戳

必需：否

环境

类型：PublicProjectEnvironment 对象

必需：否

id

类型：字符串

长度限制：最小长度为 1。

必需：否

发起者

类型：字符串

必需：否

日志状态

类型：PublicLogsStatus 对象

必需：否

阶段

类型：BuildPhase 对象数组

必需：否

项目名称

类型：字符串

长度限制：最小长度为 1。

必需：否

排队 TimeoutInMinutes

类型：整数

必需：否

已解决 SourceVersion

类型：字符串

长度限制：最小长度为 1。

必需：否

次要文物

类型：PublicBuildArtifacts 对象数组

必需：否

secondarySources

类型：PublicProjectSource 对象数组

数组成员：最少 0 个物品。最多包含 12 个项目。

必需：否

次要的 SourceVersions

类型：ProjectSourceVersion 对象数组

数组成员：最少 0 个物品。最多包含 12 个项目。

必需：否

source

类型：PublicProjectSource 对象

必需：否

sourceVersion

类型：字符串

长度限制：最小长度为 1。

必需：否

startTime

类型：时间戳

必需：否

timeoutInMinutes

类型：整数

必需：否

PublicBuildArtifacts

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含此 API 元素。

内容

> **Note**
>
> 下表中首先描述了必需参数。

神器标识符

类型：字符串

必需：否

包装

类型：字符串

必需：否

type

类型：字符串

有效值: `CODEPIPELINE | S3 | NO_ARTIFACTS | DOCKER_IMAGE`

必需：否

PublicBuildBatch

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含此 API 元素。

内容

> **Note**
>
> 下表中首先描述了必需参数。

构件

类型：PublicBuildArtifacts 对象

必需：否

buildBatchConfig

类型：PublicProjectBuildBatchConfig 对象

必需：否

buildBatchNumber

类型：长整型

必需：否

buildBatchStatus

类型：字符串

有效值：`PENDING | SUCCEEDED | FAILED | FAULT | TIMED_OUT | IN_PROGRESS | STOPPED`

必需：否

建立群组

类型：PublicBuildGroup 对象数组

必需：否

buildTimeoutInMinutes

类型：整数

必需：否

完成

类型：布尔值

必需：否

当前阶段

类型：字符串

必需：否

endTime

类型：时间戳

必需：否

环境

类型：PublicProjectEnvironment 对象

必需：否

id

类型：字符串

长度限制：最小长度为 1。

必需：否

发起者

类型：字符串

必需：否

日志状态

类型：PublicLogsStatus 对象

必需：否

阶段

类型：BuildBatchPhase 对象数组

必需：否

项目名称

类型：字符串

长度限制：最小长度为 1。

必需：否

公众 BuildBatchAlias

类型：字符串

长度限制：最小长度为 1。长度上限为 1024。

模式：`^[0-9a-zA-Z%+=]+:[a-z0-9]{8}(-[a-z0-9]{4}){3}-[a-z0-9]{12}$`

必需：否

排队 TimeoutInMinutes

类型：整数

必需：否

已解决 SourceVersion

类型：字符串

长度限制：最小长度为 1。

必需：否

次要文物

类型：PublicBuildArtifacts 对象数组

必需：否

secondarySources

类型：PublicProjectSource 对象数组

数组成员：最少 0 个物品。最多包含 12 个项目。

必需：否

次要的 SourceVersions

类型：ProjectSourceVersion 对象数组

数组成员：最少 0 个物品。最多包含 12 个项目。

必需：否

source

类型：PublicProjectSource 对象

必需：否

sourceVersion

类型：字符串

长度限制：最小长度为 1。

必需：否

startTime

类型：时间戳

必需：否

PublicBuildGroup

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含此 API 元素。

内容

> **Note**
>
> 下表中首先描述了必需参数。

currentBuildSummary

类型：PublicBuildSummary 对象

必需：否

视情况而定

类型：字符串数组

长度约束：最小长度为 1。

必需：否

identifier

类型：字符串

必需：否

忽略失败

类型：布尔值

必需：否

先前的 BuildSummaryList

类型：PublicBuildSummary 对象数组

必需：否

PublicBuildSummary

> **Note**
>
> 此 API 元素不包含在 AWS CLI 或 AWS 软件开发工具包中。

内容

> **Note**
>
> 下表中首先描述了必需参数。

生成状态

类型：字符串

必需：否

主要神器

代表已解析的构建工件。已解析的工件是指构建并部署到目标的工件，例如 Amazon S3。

类型：ResolvedArtifact 对象

必需：否

公众 BuildAlias

类型：字符串

长度限制：最小长度为 1。长度上限为 1024。

模式：`^[0-9a-zA-Z%+=]+:\p{XDigit}{8}(-\p{XDigit}{4}){3}-\p{XDigit}{12}$`

必需：否

RequestedOn

类型：时间戳

必需：否

次要神器

类型：ResolvedArtifact 对象数组

必需：否

PublicLogsStatus

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含此 API 元素。

内容

> **Note**
>
> 下表中首先描述了必需参数。

云 WatchLogsStatus

类型：字符串

有效值：`ENABLED | DISABLED`

必需：否

S3LogsStatus

类型：字符串

有效值：`ENABLED | DISABLED`

必需：否

PublicProject

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含此 API 元素。

内容

> **Note**
>
> 下表中首先描述了必需参数。

构件

类型：PublicProjectArtifacts 对象

必需：否

buildBatchConfig

类型：PublicProjectBuildBatchConfig 对象

必需：否

并发 BuildLimit

类型：整数

必需：否

描述

类型：字符串

长度限制：最小长度为 0。最大长度为 255。

必需：否

环境

类型：PublicProjectEnvironment 对象

必需：否

name

类型：字符串

长度限制：最小长度为 2。最大长度为 150。

模式：`[A-Za-z0-9][A-Za-z0-9\-_]{1,254}`

必需：否

排队 TimeoutInMinutes

类型：整数

有效范围：最小值为 5。最大值为 480。

必需：否

次要文物

类型：PublicProjectArtifacts 对象数组

数组成员：最少 0 个物品。最多包含 12 个项目。

必需：否

secondarySources

类型：PublicProjectSource 对象数组

数组成员：最少 0 个物品。最多包含 12 个项目。

必需：否

次要的 SourceVersions

类型：ProjectSourceVersion 对象数组

数组成员：最少 0 个物品。最多包含 12 个项目。

必需：否

source

类型：PublicProjectSource 对象

必需：否

sourceVersion

类型：字符串

长度限制：最小长度为 1。

必需：否

timeoutInMinutes

类型：整数

有效范围：最小值为 5。最大值为 480。

必需：否

webhook

类型：PublicWebhook 对象

必需：否

PublicProjectArtifacts

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含此 API 元素。

内容

> **Note**
>
> 下表中首先描述了必需参数。

type

类型：字符串

有效值：`CODEPIPELINE | S3 | NO_ARTIFACTS | DOCKER_IMAGE`

是否必需：是

神器标识符

类型：字符串

必需：否

位置

类型：字符串

必需：否

PublicProjectBuildBatchConfig

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含此 API 元素。

内容

> **Note**
>
> 下表中首先描述了必需参数。

combineArtifacts

类型：布尔值

必需：否

限制

指定批量构建的限制。

类型：BatchRestrictions 对象

必需：否

timeoutInMins

类型：整数

必需：否

PublicProjectEnvironment

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含此 API 元素。

内容

> **Note**
>
> 下表中首先描述了必需参数。

计算类型

类型：字符串

有效值：`BUILD_GENERAL1_SMALL | BUILD_GENERAL1_MEDIUM | BUILD_GENERAL1_LARGE | BUILD_GENERAL1_2XLARGE`

必需：否

环境变量

类型：EnvironmentVariable 对象数组

必需：否

image

类型：字符串

长度限制：最小长度为 1。

必需：否

type

类型：字符串

有效值：`WINDOWS_CONTAINER | WINDOWS_SERVER_2019_CONTAINER | LINUX_CONTAINER | LINUX_GPU_CONTAINER | ARM_CONTAINER | MAC`

必需：否

PublicProjectSource

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含此 API 元素。

内容

> **Note**
>
> 下表中首先描述了必需参数。

type

类型：字符串

有效值: `NO_SOURCE | CODECOMMIT | CODEPIPELINE | GITHUB | S3 | BITBUCKET | GITHUB_ENTERPRISE | GITLAB | GITLAB_SELF_MANAGED`

是否必需：是

buildspec

类型：字符串

必需：否

git CloneDepth

类型：整数

有效范围：最小值为 0。

必需：否

git SubmodulesConfig

有关 AWS CodeBuild 构建项目的 Git 子模块配置的信息。

类型：GitSubmodulesConfig 对象

必需：否

位置

类型：字符串

必需：否

源标识符

类型：字符串

必需：否

PublicWebhook

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含此 API 元素。

内容

> **Note**
>
> 下表中首先描述了必需参数。

分支过滤器

类型：字符串

必需：否

构建类型

类型：字符串

必需：否

过滤器组

类型：WebhookFilter对象数组数组

必需：否

payloadURL

类型：字符串

长度限制：最小长度为 1。

必需：否

url

类型：字符串

长度限制：最小长度为 1。

必需：否

S3可下载

> **Note**
>
> AWS CLI 或 AWS 软件开发工具包中不包含此 API 元素。

内容

> **Note**
>
> 下表中首先描述了必需参数。

已过期

类型：时间戳

必需：否

identifier

类型：字符串

必需：否

md5checksum

类型：字符串

必需：否

presigneDurl

类型：字符串

必需：否

s3arn

类型：字符串

必需：否

sha256CheckSum

类型：字符串

必需：否

故障排除 AWS CodeBuild

使用本主题中的信息来帮助您识别、诊断和解决问题。要了解如何记录和监控 CodeBuild 版本以解决问题，请参阅日志记录和监控。

主题

- 来自错误存储库的 Apache Maven 构建参考构件
- 默认情况下，以根用户身份运行构建命令
- 如果文件名不是-，则生成可能会失败U.S。构建可能会失败
- 当从 Amazon EC2 Parameter Store 获取参数时，构建可能失败
- 无法在 CodeBuild 控制台中访问分支筛选器
- 无法查看构建是成功还是失败
- 未向源提供商报告构建状态
- 无法找到并选择 Windows Server Core 2019 平台的基本映像
- 构建规范文件中的前期命令无法被后续命令识别
- 尝试下载缓存时出现错误：“访问被拒绝”
- 使用自定义构建映像时出现错误“BUILD_CONTAINER_UNABLE_TO_PULL_IMAGE”
- 错误：“构建容器在完成构建之前发现已失效。构建容器因内存不足而死亡，或者不支持 Docker 镜像。 ErrorCode: 500 英寸
- 错误：运行构建时出现“无法连接到 Docker 进程守护程序”
- 创建或更新构建项目时出现错误：AssumeRole“无权执行：st CodeBuild s:”
- 错误：“调用时出错 GetBucketAcl：要么存储桶所有者已更改，要么服务角色不再有权调用 s3:GetBucketAcl”
- 运行构建时收到错误：“无法上传构件：arn 无效”
- 错误：“Git 克隆失败：无法访问 '您的存储库网址': SSL 证书问题：自签名证书”
- 运行构建时收到错误：“必须使用指定的终端节点来寻址当前尝试访问的存储桶”
- 错误：“此构建映像需要至少选择一个运行时版本。”
- 构建队列中的构建失败时出现错误“QUEUED: INSUFFICIENT_SUBNET”
- 错误：“无法下载缓存： RequestError：发送请求失败原因是：x509：无法加载系统根目录且未提供根目录”

- 错误：“无法从 S3 下载证书。 AccessDenied”
- 错误：“找不到凭证”
- RequestError CodeBuild 在代理服务器上运行时出现超时错误
- bourne shell（sh）必须存在于构建映像中
- 警告：“跳过运行时安装。此构建映像不支持运行时版本选择”（在运行构建时出现）
- 错误：打开 CodeBuild 控制台时出现 “无法验证 JobWorker 身份”
- 构建启动失败
- 访问本地缓存版本中的 GitHub 元数据
- AccessDenied：报告组的存储桶所有者与 S3 存储桶的所有者不匹配...
- 使用以下命令创建 CodeBuild 项目时出现错误：“您的凭证缺少一个或多个必需的权限范围” AWS CodeConnections
- 错误：使用 Ubuntu 安装命令构建时出现“Sorry, no terminal at all requested - can't get input”
- 使用 Secrets Manager OAuth 后，Bitbucket 构建因身份验证错误而失败

来自错误存储库的 Apache Maven 构建参考构件

问题：当你将 Maven 与 AWS CodeBuild提供的 Java 编译环境一起使用时，Maven 会从安全的中央 Maven 存储库中提取编译和插件依赖关系，网址为。https://repo1.maven.org/maven2即使您构建项目的 `pom.xml` 文件明确声明会改用其他位置，也会发生这种情况。

可能的原因： CodeBuild-provided Java 编译环境中包含一个名为的文件`settings.xml`，该文件已预先安装在构建环境的`/root/.m2`目录中。该`settings.xml`文件包含以下声明，这些声明指示 Maven 始终从安全的中央 Maven 存储库中提取构建和插件依赖项，网址为。https://repo1.maven.org/maven2

```
<settings>
  <activeProfiles>
    <activeProfile>securecentral</activeProfile>
  </activeProfiles>
  <profiles>
    <profile>
      <id>securecentral</id>
      <repositories>
        <repository>
          <id>central</id>
          <url>https://repo1.maven.org/maven2</url>
          <releases>
```

```
            <enabled>true</enabled>
          </releases>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>central</id>
          <url>https://repo1.maven.org/maven2</url>
          <releases>
            <enabled>true</enabled>
          </releases>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>
</settings>
```

建议的解决方案：执行以下操作：

1. 向源代码中添加 `settings.xml` 文件。
2. 在此 `settings.xml` 文件中，使用上述 `settings.xml` 格式作为指导，声明您希望 Maven 从哪些存储库中提取构建和插件依赖项。
3. 在构建项目`install`阶段，请指示将`settings.xml`文件复制 CodeBuild 到构建环境的`/root/.m2`目录中。例如，考虑说明此行为的 `buildspec.yml` 文件中的以下代码段。

```
version 0.2

phases:
  install:
    commands:
      - cp ./settings.xml /root/.m2/settings.xml
```

默认情况下，以根用户身份运行构建命令

问题：以 root 用户身份 AWS CodeBuild 运行您的构建命令。即使您的相关构建映像的 Dockerfile 将 USER 指令设置为另一位用户，也会发生这种情况。

原因：默认情况下，以 root 用户身份 CodeBuild 运行所有构建命令。

建议的解决方案：无。

如果文件名不是-，则生成可能会失败U.S。构建可能会失败

问题：当您运行使用文件名不包含的文件的版本时U.S。字符（例如中文字符）的文件的构建时，构建将失败。

可能的原因：提供的构建环境 AWS CodeBuild 的默认区域设置为POSIX。 POSIX本地化设置与 CodeBuild 不兼容，文件名不兼容U.S。字符和可能导致相关构建失败的文件名。

建议的解决方案：将以下命令添加到构建规范文件的 pre_build 部分。这些命令可供构建环境使用 U.S。英语 UTF-8 表示其本地化设置，它更 CodeBuild 兼容，文件名包含非-U.S。的文件名。

对于基于 Ubuntu 的构建环境：

```
pre_build:
  commands:
    - export LC_ALL="en_US.UTF-8"
    - locale-gen en_US en_US.UTF-8
    - dpkg-reconfigure -f noninteractive locales
```

对于基于 Amazon Linux 的构建环境：

```
pre_build:
  commands:
    - export LC_ALL="en_US.utf8"
```

当从 Amazon EC2 Parameter Store 获取参数时，构建可能失败

问题：当构建尝试获取存储在 Amazon EC2 Parameter Store 中的一个或多个参数的值时，处于 DOWNLOAD_SOURCE 阶段的构建将失败并返回错误 Parameter does not exist。

可能的原因：构建项目所依赖的服务角色无权调用ssm:GetParameters操作，或者构建项目使用由生成 AWS CodeBuild 并允许调用ssm:GetParameters操作的服务角色，但参数的名称不是以开头的/CodeBuild/。

建议的解决方案：

- 如果服务角色不是由生成的 CodeBuild，请更新其定义 CodeBuild 以允许调用ssm:GetParameters操作。例如，以下策略语句允许调用 ssm:GetParameters 操作以获取名称以 /CodeBuild/ 开头的参数：

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": "ssm:GetParameters",
            "Effect": "Allow",
            "Resource": "arn:aws:ssm:us-east-1:111122223333:parameter/
CodeBuild/*"
        }
    ]
}
```

- 如果服务角色是由生成的 CodeBuild，请更新其定义以允许 CodeBuild 使用名称以外的名称访问 Amazon EC2 Parameter Store 中的参数/CodeBuild/。例如，以下策略语句允许调用 ssm:GetParameters 操作以获取具有指定名称的参数：

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": "ssm:GetParameters",
            "Effect": "Allow",
            "Resource": "arn:aws:ssm:us-
east-1:111122223333:parameter/PARAMETER_NAME"
        }
    ]
}
```

无法在 CodeBuild 控制台中访问分支筛选器

问题：创建或更新 AWS CodeBuild 项目时，控制台中无法使用分支筛选器选项。

可能的原因：分支筛选选项已被弃用。它已被 Webhook 筛选条件组取代，后者可以更好地控制触发新 CodeBuild 中的构建的 Webhook 事件。

建议的解决方案：要迁移在引入 Webhook 筛选条件之前创建的分支筛选条件，请使用正则表达式 ^refs/heads/*branchName*$ 创建带 HEAD_REF 筛选条件的 Webhook 筛选条件组。例如，如果

您的分支筛选条件正则表达式是 ^branchName$，那么您放入 HEAD_REF 筛选条件的经过更新的正则表达式是 ^refs/heads/branchName$。有关更多信息，请参阅Bitbucket Webhook 事件和过滤 GitHub webhook 事件（控制台）。

无法查看构建是成功还是失败

问题：无法查看重试构建是成功还是失败。

可能的原因：未启用报告构建状态的选项。

推荐的解决方案：在创建或更新 CodeBuild 项目时启用 “报告构建状态”。此选项告知 CodeBuild 在触发构建时报告状态。有关更多信息，请参阅 AWS CodeBuild API 参考BuildStatus中的报告。

未向源提供商报告构建状态

问题：允许向源提供商（例如 GitHub 或 Bitbucket）报告构建状态后，构建状态未更新。

可能的原因：与源提供商关联的用户不具备访问存储库的权限。

建议的解决方案：为了能够向源提供商报告构建状态，与源提供商关联的用户必须拥有对存储库的写入权限。如果用户没有写入权限，则无法更新构建状态。有关更多信息，请参阅 源提供商访问权限。

无法找到并选择 Windows Server Core 2019 平台的基本映像

问题：无法找到或选择 Windows Server Core 2019 平台的基本映像。

可能的原因：您使用的 AWS 区域不支持此图片。

建议的解决方案：使用以下支持 Windows Server Core 2019 平台基本映像的 AWS 区域之一：

- 美国东部（弗吉尼亚州北部）
- 美国东部（俄亥俄州）
- 美国西部（俄勒冈州）
- 欧洲地区（爱尔兰）

构建规范文件中的前期命令无法被后续命令识别

问题：buildspec 文件中的一个或多个命令的结果无法被同一 buildspec 文件中的后续命令识别。例如，某个命令可能会设置本地环境变量，但稍后运行的命令可能无法获取该本地环境变量的值。

可能的原因：在 buildspec 文件版本 0.1 中， AWS CodeBuild 将在构建环境内的默认 Shell 的单独实例中运行每个命令。这表示各个命令独立于其他所有命令而运行。默认情况下，您无法运行依赖于任何先前命令的状态的单个命令。

推荐的解决方案：我们建议您使用 buildspec 版本 0.2，它可以解决此问题。如果您必须使用构建规范版本 0.1，建议您使用 Shell 命令链接运算符（例如，Linux 中的 &&）将多个命令合并为一个命令。或者，您也可以在源代码中包括一个带有多个命令的 Shell 脚本，然后从 buildspec 文件中的单个命令调用该 Shell 脚本。有关更多信息，请参阅构建环境中的 Shell 和命令和构建环境中的环境变量。

尝试下载缓存时出现错误：“访问被拒绝”

问题：当尝试下载已启用缓存的构建项目上的缓存时，您收到 `Access denied` 错误。

可能的原因：

- 您刚刚已将缓存配置为您的构建项目的一部分。
- 最近已通过 `InvalidateProjectCache` API 使缓存失效。
- 使用的服务角色对`s3:GetObject`存放缓存的 S3 存储桶 CodeBuild 没有`s3:PutObject`权限。

建议的解决方案：在首次使用时，在更新缓存配置后立即看到此错误是正常的。如果此错误持续存在，则您应该检查您的服务角色对包含缓存的 S3 存储桶是否具有 `s3:GetObject` 和 `s3:PutObject` 权限。有关更多信息，请参阅《Amazon S3 开发人员指南》中的指定 S3 权限。

使用自定义构建映像时出现错误“BUILD_CONTAINER_UNABLE_TO_PULL_IMAGE”

问题：当您尝试运行使用自定义构建映像的构建时，构建将失败并返回错误 `BUILD_CONTAINER_UNABLE_TO_PULL_IMAGE`。

可能的原因：构建映像的整体未压缩大小大于构建环境计算类型的可用磁盘空间。要检查构建映像的大小，请使用 Docker 运行 *`docker images REPOSITORY:TAG`* 命令。有关按计算类型分类的可用磁盘空间的列表，请参阅构建环境计算模式和类型。

建议的解决方案：对较大的计算类型使用更多的可用磁盘空间，或者减小自定义构建映像的大小。

可能的原因： AWS CodeBuild 无权从您的亚马逊弹性容器注册表 (Amazon ECR) Container Registry 中提取构建映像。

推荐的解决方案：在 Amazon ECR 中更新存储库中的权限，以便 CodeBuild 可以将您的自定义构建映像拉入构建环境。有关更多信息，请参阅Amazon ECR 示例。

可能的原因：您请求的 Amazon ECR 图片在您的 AWS 账户使用的 AWS 地区不可用。

推荐的解决方案：使用与您的 AWS 账户 AWS 所在区域相同的 Amazon ECR 映像。

可能的原因：您在无法访问公共互联网的 VPC 中使用私有注册表。 CodeBuild 无法从 VPC 中的私有 IP 地址提取镜像。有关更多信息，请参阅 带有 AWS Secrets Manager 示例的私有注册表 CodeBuild。

建议的解决方案：如果您在 VPC 中使用私有注册表，请确保 VPC 具有公共互联网访问权限。

可能的原因：如果错误消息包含“toomanyrequests”，并且映像是从 Docker Hub 获取的，则此错误表示已达到 Docker Hub 的拉取限制。

建议的解决方案：使用 Docker Hub 私有注册表，或者从 Amazon ECR 获取您的映像。有关使用私有注册表的更多信息，请参阅带有 AWS Secrets Manager 示例的私有注册表 CodeBuild。有关使用 Amazon ECR 的更多信息，请参阅的亚马逊 ECR 示例 CodeBuild。

错误：“构建容器在完成构建之前发现已失效。构建容器因内存不足而死亡，或者不支持 Docker 镜像。 ErrorCode: 500 英寸

问题：当你尝试在中使用微软 Windows 或 Linux 容器时 AWS CodeBuild，此错误发生在配置阶段。

可能的原因：

- 不支持容器操作系统版本 CodeBuild。
- 在容器中指定了 HTTP_PROXY 和/或 HTTPS_PROXY。

建议的解决方案：

- 对于微软 Windows，请使用带有容器操作系统版本为:10.0.x 的 Windows 容器 microsoft/windowsservercore（例如，: 10.0.14393.2125）。 microsoft/windowsservercore
- 对于 Linux，请在 Docker 映像中清除 HTTP_PROXY 和 HTTPS_PROXY 设置，或在构建项目中指定 VPC 配置。

错误：运行构建时出现“无法连接到 Docker 进程守护程序”

问题：您的构建失败，并在构建日志中收到了类似于 `Cannot connect to the Docker daemon at unix:/var/run/docker.sock. Is the docker daemon running?` 的错误。

可能的原因：您未在特权模式下运行构建。

推荐的解决方案：要修复此错误，必须启用特权模式并按照以下说明更新 buildspec。

要在特权模式下运行构建，请按照以下步骤操作：

1. 打开 CodeBuild 控制台，网址为https://console.aws.amazon.com/codebuild/。
2. 在导航窗格中选择构建项目，然后选择您的构建项目。
3. 从编辑中，选择环境。
4. 选择其他配置。
5. 在特权中，选择如果要构建 Docker 映像或希望您的构建获得提升的特权，请启用此标志。
6. 选择更新环境。
7. 选择启动构建来重试您的构建。

您还需要在容器内启动 Docker 进程守护程序。buildspec 的 `install` 阶段可能看上去与以下示例类似：

```
phases:
  install:
    commands:
      - nohup /usr/local/bin/dockerd --host=unix:///var/run/docker.sock --
host=tcp://127.0.0.1:2375 --storage-driver=overlay2 &
      - timeout 15 sh -c "until docker info; do echo .; sleep 1; done"
```

有关 buildspec 文件中引用的 OverlayFS 存储驱动程序的更多信息，请参阅 Docker 网站上的使用 OverlayFS 存储驱动程序。

> **Note**
>
> 如果基本操作系统是 Alpine Linux，请在 `buildspec.yml` 中向 `timeout` 添加 `-t` 参数：
>
> ```
> - timeout -t 15 sh -c "until docker info; do echo .; sleep 1; done"
> ```

要详细了解如何使用构建和运行 Docker 镜像 AWS CodeBuild，请参阅自定义镜像示例中的 Docker CodeBuild。

创建或更新构建项目时出现错误：AssumeRole“无权执行：st CodeBuild s:”

问题：当您尝试创建或更新构建项目时，您会收到错误 `Code:InvalidInputException, Message:CodeBuild is not authorized to perform: sts:AssumeRole on arn:aws:iam::account-ID:role/service-role-name`。

可能的原因：

- 对于您尝试创建或更新构建项目的 AWS 区域， AWS Security Token Service (AWS STS) 已被停用。
- 与构建项目关联的 AWS CodeBuild 服务角色不存在或没有足够的信任权限 CodeBuild。
- 与构建项目关联的 AWS CodeBuild 服务角色大小写与实际的 IAM 角色不匹配。

建议的解决方案：

- 确保 AWS STS 在您尝试创建或更新构建项目的 AWS 区域处于激活状态。有关更多信息，请参阅 IAM 用户指南 AWS STS 中的在 AWS 区域中激活和停用。
- 确保您的 AWS 账户中存在目标 CodeBuild 服务角色。如果您没有使用控制台，请确保在创建或更新构建项目时没有拼错服务角色的 Amazon 资源名称（ARN）。请注意，IAM 角色区分大小写，因此请检查 IAM 角色的大小写是否正确。
- 确保目标 CodeBuild 服务角色具有足够的信任权限 CodeBuild。有关更多信息，请参阅 CodeBuild 允许与其他人互动 AWS 务 中的信任关系策略声明。

错误：“调用时出错 GetBucketAcl：要么存储桶所有者已更改，要么服务角色不再有权调用 s3:GetBucketAcl”

问题：运行构建时，您收到一个有关 S3 存储桶所有权更改和 `GetBucketAcl` 权限更改的错误。

可能的原因：您将 `s3:GetBucketAcl` 和 `s3:GetBucketLocation` 权限添加到了 IAM 角色。这些权限可保护您项目的 S3 存储桶，并确保只有您可以访问它。添加完这些权限后，S3 存储桶的拥有者会发生更改。

建议的解决方案：确认您是 S3 存储桶的拥有者，然后重新将权限添加到您的 IAM 角色。有关更多信息，请参阅 对 S3 存储桶的安全访问。

运行构建时收到错误：“无法上传构件：arn 无效”

问题：在运行构建时，UPLOAD_ARTIFACTS 构建阶段失败并出现错误 `Failed to upload artifacts: Invalid arn`。

可能的原因：您的 S3 输出存储桶（ AWS CodeBuild 存储其生成输出的存储桶）位于与 CodeBuild 构建项目不同的 AWS 区域。

推荐的解决方案：更新构建项目的设置，使其指向与构建项目位于同一 AWS 区域的输出存储桶。

错误：“Git 克隆失败：无法访问 '#######': SSL 证书问题：自签名证书”

问题： 当您尝试运行构建项目时，构建失败并出现此错误。

可能的原因：您的源存储库具有一个自签名证书，但您在构建项目的过程中未选择从您的 S3 存储桶安装此证书。

建议的解决方案：

- 编辑您的项目。对于证书，选择从 S3 安装证书。对于证书存储桶，选择存储您的 SSL 证书的 S3 存储桶。对于证书的对象键，键入您的 S3 对象键的名称。
- 编辑您的项目。选择 “不安全 SSL” 可在连接到 GitHub 企业服务器项目存储库时忽略 SSL 警告。

> **Note**
>
> 建议您仅将不安全的 SSL 用于测试。它不应在生产环境中使用。

运行构建时收到错误：“必须使用指定的终端节点来寻址当前尝试访问的存储桶”

问题：在运行构建时，DOWNLOAD_SOURCE 构建阶段失败并出现错误 `The bucket you are attempting to access must be addressed using the specified endpoint. Please send all future requests to this endpoint`。

可能的原因：您预先构建的源代码存储在 S3 存储桶中，而该存储桶位于与 AWS CodeBuild 构建项目不同的 AWS 区域。

建议的解决方案：更新构建项目的设置，以指向包含预构建的源代码的存储桶。确保该存储桶与构建项目位于同一 AWS 区域。

错误：“此构建映像需要至少选择一个运行时版本。”

问题：在运行构建时，DOWNLOAD_SOURCE 构建阶段失败并出现错误 YAML_FILE_ERROR: This build image requires selecting at least one runtime version。

可能的原因：您的构建使用 1.0 版或更高版本的 Amazon Linux 2（AL2）标准映像或者 2.0 版或更高版本的 Ubuntu 标准映像，并且未在构建规范文件中指定运行时。

推荐的解决方案：如果您使用aws/codebuild/standard:2.0 CodeBuild 托管映像，则必须在 buildspec 文件的runtime-versions部分中指定运行时版本。例如，您可以对使用 PHP 的项目使用以下 buildspec 文件：

```
version: 0.2

phases:
  install:
    runtime-versions:
        php: 7.3
  build:
    commands:
      - php --version
artifacts:
  files:
    -  README.md
```

> **Note**
>
> 如果您指定 runtime-versions 部分且使用的映像不是 Ubuntu 标准映像 2.0 或更高版本或 Amazon Linux 2 (AL2) 标准映像 1.0 或更高版本，则构建会发出警告“Skipping install of runtimes. Runtime version selection is not supported by this build image”。

有关更多信息，请参阅 Specify runtime versions in the buildspec file。

构建队列中的构建失败时出现错误“QUEUED: INSUFFICIENT_SUBNET”

问题：构建队列中的构建失败，出现类似于 QUEUED: INSUFFICIENT_SUBNET 的错误。

可能的原因：为 VPC 指定的 IPv4 CIDR 块使用了预留 IP 地址。每个子网 CIDR 块中的前四个 IP 地址和最后一个 IP 地址无法供您使用，而且无法分配到一个实例。例如，在具有 CIDR 块 10.0.0.0/24 的子网中，以下五个 IP 地址是保留的：

- 10.0.0.0:：网络地址。
- 10.0.0.1：由 AWS VPC 路由器保留。
- 10.0.0.2: 由... 保留 AWS。DNS 服务器的 IP 地址始终为 VPC 网络范围的基址 + 2；但是，我们也保留了每个子网范围基址 + 2 的 IP 地址。对于包含多个 CIDR 块的 VPC，DNS 服务器的 IP 地址位于主要 CIDR 中。有关更多信息，请参阅《Amazon VPC 用户指南》中的 Amazon DNS 服务器。
- 10.0.0.3: 由保留 AWS 以备将来使用。
- 10.0.0.255：网络广播地址。我们不支持 VPC 中的广播。该地址是预留的。

建议的解决方案：检查您的 VPC 是否使用了预留 IP 地址。将任何预留的 IP 地址替换为未预留的 IP 地址。有关更多信息，请参阅《Amazon VPC 用户指南》中的 VPC 和子网大小调整。

错误：“无法下载缓存： RequestError：发送请求失败原因是：x509：无法加载系统根目录且未提供根目录”

问题： 当您尝试运行构建项目时，构建失败并出现此错误。

可能的原因：您将缓存配置为您的构建项目的一部分并使用包含过期根证书的较旧 Docker 映像。

推荐的解决方案：更新 AWS CodeBuild 项目中正在使用的 Docker 镜像。有关更多信息，请参阅 Docker 镜像由 CodeBuild。

错误：“无法从 S3 下载证书。 AccessDenied”

问题： 当您尝试运行构建项目时，构建失败并出现此错误。

可能的原因：

- 您选择了错误的证书 S3 存储桶。
- 您输入了错误的证书对象键。

建议的解决方案：

- 编辑您的项目。对于证书存储桶，选择存储您的 SSL 证书的 S3 存储桶。
- 编辑您的项目。对于证书的对象键，键入您的 S3 对象键的名称。

错误：“找不到凭证”

问题：当你尝试运行 AWS CLI、使用 AWS SDK 或调用其他类似组件作为构建的一部分时，你会遇到与 AWS CLI、 AWS SDK 或组件直接相关的构建错误。例如，您可能会收到构建错误，如 `Unable to locate credentials`。

可能的原因：

- 编译环境中 AWS CLI、 AWS SDK 或组件的版本与不兼容 AWS CodeBuild。
- 您在使用 Docker 的构建环境中运行 Docker 容器，默认情况下，该容器无权访问 AWS 凭证。

建议的解决方案：

- 确保您的构建环境具有以下版本或更高版本的 AWS CLI、 AWS SDK 或组件。
  - AWS CLI：1.10.47
  - AWS 适用于 C++ 的 SDK：0.2.19
  - AWS 适用于 Go 的 SDK：1.2.5
  - AWS 适用于 Java 的 SDK：1.11.16
  - AWS 适用于 JavaScript：2.4.7 的 SDK
  - AWS 适用于 PHP 的 SDK：3.18.28
  - AWS 适用于 Python 的 SDK (Boto3)：1.4.0
  - AWS 适用于 Ruby 的 SDK：2.3.22
  - Botocore：1.4.37
  - CoreCLR：3.2.6-beta
  - Node.js: 2.4.7

- 如果您需要在构建环境中运行 Docker 容器并且该容器需要 AWS 证书，则必须将证书从构建环境传递到容器。在您的 buildspec 文件中，包含与以下内容类似的 Docker `run` 命令。此示例使用 `aws s3 ls` 命令列出您的可用 S3 存储桶。该`-e`选项会传递容器访问 AWS 证书所需的环境变量。

```
docker run -e AWS_DEFAULT_REGION -e AWS_CONTAINER_CREDENTIALS_RELATIVE_URI your-image-tag aws s3 ls
```

- 如果您正在构建 Docker 映像，并且构建需要 AWS 证书（例如，从 Amazon S3 下载文件），则必须按如下方式将编译环境中的证书传递到 Docker 构建过程。

  1. 在您的源代码的用于 Docker 映像的 Dockerfile 中，指定以下 `ARG` 指令。

```
ARG AWS_DEFAULT_REGION
ARG AWS_CONTAINER_CREDENTIALS_RELATIVE_URI
```

  2. 在您的 buildspec 文件中，包含与以下内容类似的 Docker `build` 命令。这些`--build-arg`选项设置了 Docker 构建过程访问 AWS 凭证所需的环境变量。

```
docker build --build-arg AWS_DEFAULT_REGION=$AWS_DEFAULT_REGION --build-arg
 AWS_CONTAINER_CREDENTIALS_RELATIVE_URI=$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI -
t your-image-tag .
```

RequestError CodeBuild 在代理服务器上运行时出现超时错误

问题：您收到类似于以下内容的 `RequestError` 错误：

- `RequestError: send request failed caused by: Post https://logs.<your-region>.amazonaws.com/: dial tcp 52.46.158.105:443: i/o timeout`来自 CloudWatch 日志。
- Amazon S3 中的 `Error uploading artifacts: RequestError: send request failed caused by: Put https://your-bucket.s3.your-aws-region.amazonaws.com/*: dial tcp 52.219.96.208:443: connect: connection refused`。

可能的原因：

- `ssl-bump` 未正确配置。
- 贵组织的安全策略不允许您使用 `ssl_bump`。
- 您的 buildspec 文件没有使用 `proxy` 元素指定的代理设置。

建议的解决方案：

- 确保 `ssl-bump` 已正确配置。如果您对代理服务器使用 Squid，请参阅将 Squid 配置为显式代理服务器。
- 按照以下步骤为 Amazon S3 和 CloudWatch 日志使用私有终端节点：

  1. 在您的私有子网路由表中，删除您添加的、将发往互联网的流量路由到您的代理服务器的规则。有关信息，请参阅《Amazon VPC 用户指南》中的在 VPC 中创建子网。
  2. 创建私有 Amazon S3 终端节点和 CloudWatch 日志终端节点，并将它们与您的 Amazon VPC 的私有子网关联。有关信息，请参阅《Amazon VPC 用户指南》中的 VPC 端点服务。
  3. 确认已选中 Amazon VPC 中的启用私有 DNS 名称。有关更多信息，请参阅《Amazon VPC 用户指南》中的创建接口端点
- 如果您不将 `ssl-bump` 用于显式代理服务器，请使用 `proxy` 元素将代理配置添加到您的 buildspec 文件。有关更多信息，请参阅CodeBuild 在显式代理服务器中运行和buildspec 语法。

```
version: 0.2
proxy:
  upload-artifacts: yes
  logs: yes
phases:
  build:
    commands:
```

bourne shell（sh）必须存在于构建映像中

问题：您使用的构建映像不是由提供的 AWS CodeBuild，并且您的构建失败并显示消息`Build container found dead before completing the build`。

可能的原因：Bourne 外壳 (sh) 未包含在您的构建镜像中。 CodeBuild `sh`需要运行构建命令和脚本。

推荐的解决方案：`sh`如果您的构建映像中不存在，请务必在开始使用您的映像的更多构建之前将其包含在内。（CodeBuild 已包含`sh`在其构建镜像中。）

警告：“跳过运行时安装。此构建映像不支持运行时版本选择”（在运行构建时出现）

问题：在运行构建时，构建日志包含此警告。

可能的原因：您的构建未使用 1.0 版或更高版本的 Amazon Linux 2（AL2）标准映像或者 2.0 版或更高版本的 Ubuntu 标准映像，并且在 buildspec 文件的 `runtime-versions` 部分中指定了运行时。

建议的解决方案：确保 buildspec 文件不包含 `runtime-versions` 部分。仅当使用 Amazon Linux 2（AL2）标准映像或更高版本或者 Ubuntu 标准映像版本 2.0 或更高版本时，才需要 `runtime-versions` 部分。

错误：打开 CodeBuild 控制台时出现 “无法验证 JobWorker 身份”

问题：打开 CodeBuild 控制台时，会显示 “无法验证 JobWorker 身份” 错误消息。

可能的原因：用于控制台访问的 IAM 角色的标签以 `jobId` 作为键。此标签密钥是为之保留的 CodeBuild ，如果存在则会导致此错误。

建议的解决方案：将任何具有 `jobId` 键的自定义 IAM 角色标签更改为具有其他键，例如 `jobIdentifier`。

构建启动失败

问题：启动构建时，您会收到构建启动失败错误消息。

可能的原因：已达到并发构建的数量。

推荐的解决方案：等到其他构建完成，或者增加项目的并发生成限制，然后重新开始构建。有关更多信息，请参阅 项目配置。

访问本地缓存版本中的 GitHub 元数据

问题：在某些情况下，缓存构建中的 .git 目录是文本文件，而不是目录。

可能的原因：为版本启用本地源代码缓存后， CodeBuild 会为该`.git`目录创建一个 gitlink。这意味着该 `.git` 目录实际上是一个包含目录路径的文本文件。

建议的解决方案：在所有情况下，都使用以下命令获取 Git 元数据目录。无论 `.git` 采用何种格式，此命令都将起作用：

```
git rev-parse --git-dir
```

AccessDenied：报告组的存储桶所有者与 S3 存储桶的所有者不匹配...

问题：将测试数据上传到 Amazon S3 存储桶 CodeBuild 时，无法将测试数据写入存储桶。

可能的原因：

- 为报告组存储桶所有者指定的账户与 Amazon S3 存储桶的所有者不匹配。
- 服务角色不具备写入存储桶的权限。

建议的解决方案：

- 更改报告组存储桶所有者，使其与 Amazon S3 存储桶所有者匹配。
- 修改服务角色来提供写入 Amazon S3 存储桶的权限。

使用以下命令创建 CodeBuild 项目时出现错误：“您的凭证缺少一个或多个必需的权限范围” AWS CodeConnections

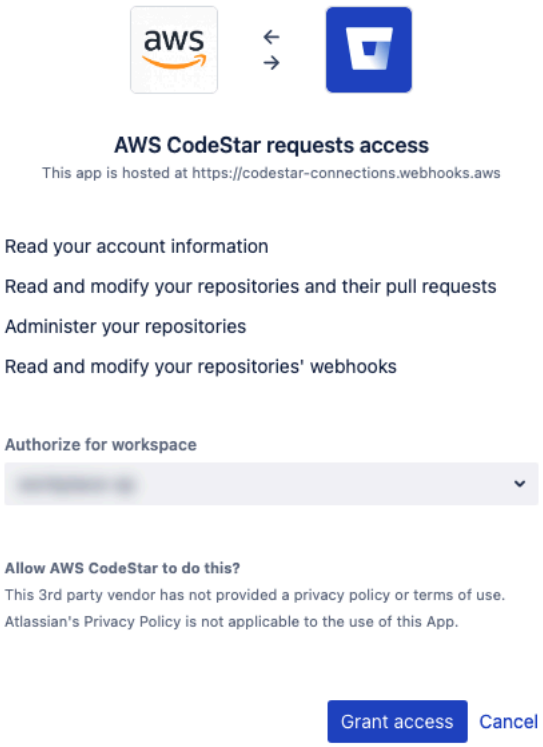
问题：使用创建 CodeBuild 项目时 AWS CodeConnections，您无权安装 Bitbucket webhook。

可能的原因：

- 您的 Bitbucket 账户可能尚未接受新的权限范围。

建议的解决方案：

- 要接受新的权限，您应该已经收到一封标题为 Bitbucket 发送的标题为 “需要操作——范围 AWS CodeStar 已更改” 的电子邮件。`notifications-noreply@bitbucket.org`该电子邮件包含一个链接，用于向您现有 B AWS CodeConnections itbucket 应用程序安装授予 webhook 权限。
- 如果您找不到该电子邮件，则可以通过导航到 `https://bitbucket.org/site/addons/reauthorize?account=<workspace-name>&addon_key=aws-codestar` 或 `https://bitbucket.org/site/addons/reauthorize?addon_key=aws-codestar`，并选择要向 webhook 授予相关权限的工作区来授予权限。



错误：使用 Ubuntu 安装命令构建时出现“Sorry, no terminal at all requested - can't get input”

问题：如果您正在运行 GPU 容器有特权的构建，则可能要按照以下过程安装 NVIDIA 容器工具包。在最新版本的 CodeBuild 镜像中，使用最新的`amazonlinux`精选镜像 CodeBuild 预安装和配置 d`nvidia-container-toolkit` ocker。`ubuntu`按照此过程操作将导致使用 Ubuntu 安装命令进行构建时失败，并出现以下错误：

```
Running command curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | gpg --
dearmor --no-tty -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg
gpg: Sorry, no terminal at all requested - can't get input
curl: (23) Failed writing body
```

可能的原因：gpg 键已存在于同一位置。

建议的解决方案：映像中已安装了 `nvidia-container-toolkit`。如果您看到这个错误，则可以跳过安装并在 buildspec 中重启 docker 进程。

使用 Secrets Manager OAuth 后，Bitbucket 构建因身份验证错误而失败

问题：使用 Bitbucket 作为源代码提供商构建失败，并出现源证书错误。Bitbucket 连接是使用 OAuth AWS Secrets Manager 作为存储服务配置的。

可能的原因：Atlassian 对 Bitbucket OAuth 强制使用一次性轮换刷新令牌。每次使用刷新令牌时，Bitbucket 都会使其失效并返回一个新令牌。有关更多信息，请参阅 Atlassian 开发者网站上的 Bitbucket OAuth 一次性刷新令牌 (CHANGE-3052)。

推荐的解决方案：为您的 CodeBuild 服务角色添加`secretsmanager:PutSecretValue`权限。有关所需的 IAM 权限的更多信息，请参阅密钥 Manager-stored 凭证所需的操作。

的配额 AWS CodeBuild

下表列出了中的当前配额 AWS CodeBuild。除非另有说明，否则这些配额适用于每个 AWS 账户的每个受支持 AWS 区域。

服务配额

以下是该 AWS CodeBuild 服务的默认配额。

Name	默认值	可调整	说明
每个项目的关联标签数	每个受支持的区域：50 个	否	可以与构建项目相关联的最大标签数
构建项目	每个受支持的区域：5000 个	是	最大构建项目数
构建超时（分钟）	每个受支持的区域：2160 个	否	最大构建超时（以分钟为单位）
针对构建信息的并发请求	每个受支持的区域：100 个	否	您可以随时使用 AWS CLI 或 AWS SDK 请求相关信息的最大版本数。
针对构建项目信息的并发请求	每个受支持的区域：100 个	否	您可以随时使用 AWS CLI 或 AWS SDK 请求相关信息的最大构建项目数量。
同时运行适用于 ARM Lambda/10GB 环境的构建	每个受支持的区域：1 个	是	ARM Lambda/10GB 环境中同时运行的版本的最大数量
同时运行适用于 ARM Lambda/1GB 环境的构建	每个受支持的区域：1 个	是	ARM Lambda/1GB 环境中同时运行的版本的最大数量

Name	默认值	可调整	说明
同时运行适用于 ARM Lambda/2GB 环境的构建	每个受支持的区域：1 个	是	ARM Lambda/2GB 环境中同时运行的版本的最大数量
同时运行适用于 ARM Lambda/4GB 环境的构建	每个受支持的区域：1 个	是	ARM Lambda/4GB 环境中同时运行的版本的最大数量
同时运行适用于 ARM Lambda/8GB 环境的构建	每个受支持的区域：1 个	是	ARM Lambda/8GB 环境中同时运行的版本的最大数量
同时运行针对环境的 ARM/2XLarge 构建	每个受支持的区域：1 个	是	同时运行的环境版本的最 ARM/2XLarge 大数量
同时运行针对环境的 ARM/Large 构建	每个受支持的区域：1 个	是	同时运行的环境版本的最 ARM/Large 大数量
同时运行针对环境的 ARM/Medium 构建	每个受支持的区域：1 个	是	同时运行的环境版本的最 ARM/Medium 大数量
同时运行针对环境的 ARM/Small 构建	每个受支持的区域：1 个	是	同时运行的环境版本的最 ARM/Small 大数量
同时运行针对环境的 ARM/XLarge 构建	每个受支持的区域：1 个	是	同时运行的环境版本的最 ARM/XLarge 大数量
适用于 Linux GPU Large 环境的并发运行构建	每个受支持的区域：0 个	是	Linux GPU/Large 环境中同时运行的版本的最大数量
适用于 Linux GPU Small 环境的并发运行构建	每个受支持的区域：0 个	是	Linux GPU/Small 环境中同时运行的版本的最大数量

Name	默认值	可调整	说明
同时运行适用于 Linux Lambda/10GB 环境的版本	每个受支持的区域：1 个	是	Linux Lambda/10GB 环境中同时运行的版本的最大数量
同时运行适用于 Linux Lambda/1GB 环境的版本	每个受支持的区域：1 个	是	Linux Lambda/1GB 环境中同时运行的版本的最大数量
同时运行适用于 Linux Lambda/2GB 环境的版本	每个受支持的区域：1 个	是	Linux Lambda/2GB 环境中同时运行的版本的最大数量
同时运行适用于 Linux Lambda/4GB 环境的版本	每个受支持的区域：1 个	是	Linux Lambda/4GB 环境中同时运行的版本的最大数量
同时运行适用于 Linux Lambda/8GB 环境的版本	每个受支持的区域：1 个	是	Linux Lambda/8GB 环境中同时运行的版本的最大数量
同时运行针对环境的 Linux/2XLarge 构建	每个受支持的区域：0 个	是	同时运行的环境版本的最 Linux/2XLarge 大数量
同时运行针对环境的 Linux/Large 构建	每个受支持的区域：1 个	是	同时运行的环境版本的最 Linux/Large 大数量
同时运行针对环境的 Linux/Medium 构建	每个受支持的区域：1 个	是	同时运行的环境版本的最 Linux/Medium 大数量
同时运行针对环境的 Linux/Small 构建	每个受支持的区域：1 个	是	同时运行的环境版本的最 Linux/Small 大数量
同时运行针对环境的 Linux/XLarge 构建	每个受支持的区域：1 个	是	同时运行的环境版本的最 Linux/XLarge 大数量

Name	默认值	可调整	说明
同时运行适用于 Windows 服务器 2019/Large 环境的版本	每个受支持的区域：1 个	是	Windows 服务器 2019/Large 环境中同时运行的版本的最大数量
同时运行适用于 Windows 服务器 2019/Medium 环境的版本	每个受支持的区域：1 个	是	Windows 服务器 2019/Medium 环境中同时运行的版本的最大数量
同时运行适用于 Windows 服务器 2022/2XLarge 环境的版本	每个受支持的区域：1 个	是	Windows 服务器 2022/2XLarge 环境中同时运行的版本的最大数量
同时运行适用于 Windows 服务器 2022/Large 环境的版本	每个受支持的区域：1 个	是	Windows 服务器 2022/Large 环境中同时运行的版本的最大数量
同时运行适用于 Windows 服务器 2022/Medium 环境的版本	每个受支持的区域：1 个	是	Windows 服务器 2022/Medium 环境中同时运行的版本的最大数量
同时运行适用于 Windows 服务器 2022/XLarge 环境的版本	每个受支持的区域：1 个	是	Windows 服务器 2022/XLarge 环境中同时运行的版本的最大数量
同时运行针对环境的 Windows/Large 构建	每个受支持的区域：1 个	是	同时运行的环境版本的最 Windows/Large 大数量
同时运行针对环境的 Windows/Medium 构建	每个受支持的区域：1 个	是	同时运行的环境版本的最 Windows/Medium 大数量
构建超时的最短时间（分钟）	每个受支持的区域：5 个	否	最小构建超时（以分钟为单位）
VPC 配置下的安全组	每个受支持的区域：5 个	否	可用于 VPC 配置的安全组

Name	默认值	可调整	说明
VPC 配置下的子网	每个受支持的区域：16 个	否	可用于 VPC 配置的子网

> **Note**
>
> 内部指标将决定并发运行构建的默认限额。

最大并发运行构建数的限额因计算类型的不同而有所不同。对于某些平台和计算类型，默认值为 20。要请求更高的并发构建限额，或者如果您收到“账户不能有多于 X 个处于活动状态的构建”错误，请使用上面的链接提出请求。有关定价的更多信息，请参阅 AWS CodeBuild 定价。

其他限制

构建项目

资源	默认
构建项目描述中允许使用的字符	Any
构建项目名称中允许使用的字符	字母 A-Z 和 a-z、数字 0-9，以及特殊字符 - 和 _
构建项目名称的长度	2 到 150 个字符
构建项目描述的最大长度	255 个字符
您可以添加到项目的最大报告数	5
可以在构建项目中为所有相关构建的构建超时指定的分钟数	5 到 2160（36 个小时）

Builds

资源	默认
构建历史记录的最长保留时间	1 年
可以为单个构建的构建超时指定的分钟数	5 到 2160（36 个小时）

计算实例集

资源	默认
计算实例集的并发数量	10
同时运行 ARM/Small 环境队列的实例	1
同时运行 ARM/Large 环境队列的实例	1
同时运行 Linux/Small 环境队列的实例	1
同时运行 Linux/Medium 环境队列的实例	1
同时运行 Linux/Large 环境队列的实例	1
同时运行 Linux/XLarge 环境队列的实例	1
同时运行 Linux/2XLarge 环境队列的实例	0
同时运行 Linux GPU/Small 环境队列的实例	0
同时运行 Linux GPU/Large 环境队列的实例	0
同时运行 Windows 服务器 2019/Medium 环境队列的实例	1
同时运行 Windows 服务器 2019/Large 环境队列的实例	1

资源	默认
同时运行 Windows 服务器 2022/Medium 环境队列的实例	1
同时运行 Windows 服务器 2022/Large 环境队列的实例	1
同时运行 Mac ARM/Medium 环境队列的实例	1
同时运行 Mac ARM/Large 环境队列的实例	1

Reports

资源	默认
测试报告创建后可用的最长持续时间	30 天
测试用例消息的最大长度	5000 个字符
测试用例名称的最大长度	1000 个字符
每个 AWS 账户的最大报告组数	5000
每份报告的最大测试用例数	500

标签

标签限制适用于 CodeBuild 生成项目和 CodeBuild 报表组资源上的标签。

资源	默认
资源标签键名称	Unicode 字母、数字、空格和允许字符的任意组合，长度 UTF-8 介于 1 到 127 个字符之间。允许使用的字符为 `+ - = . _ : / @` 标签键名称必须是唯一的，而且每个键只能有一个值。标签键名称不能：

资源	默认
	• 以 aws: 开头 • 只包含空格 • 以空格结尾 • 包含表情符号或以下任意字符：? ^ * [\ ~ ! # $ % & * ( ) > < | " ' ` [ ] { } ;
资源标签值	Unicode 字母、数字、空格和允许字符的任意组合，长度 UTF-8 介于 0 到 255 个字符之间。允许使用的字符为 + - = . _ : / @ 一个键只能有一个值，但许多键可以具有相同的值。标签键值不能包含表情符号或以下任意字符： ? ^ * [\ ~ ! # $ % & * ( ) > < | " ' ` [ ] { } ;

AWS CodeBuild 《用户指南》文档历史记录

下表描述了自上次发布以来对文档所做的重要更改 AWS CodeBuild。要获得本文档的更新通知，您可以订阅 RSS 源。

- 最新 API 版本：2016-10-06

变更	说明	日期
更新内容： CodeBuild 标准映像的可用运行时	为 x86_64 标准镜像添加了 Node.js 26 运行时和 Rust 1.97。	2026年9月22日
更新内容： CodeBuild 标准映像的可用运行时	在支持的运行时中添加了亚马逊 Linux 2023 AArch64 standard: 4.0 镜像。	2026年9月22日
更新内容：webhook 过滤器正则表达式语法	澄清了 webhook 过滤器模式使用 RE2 正则表达式语法以及不支持哪些结构。	2026 年 8 月 3 日
更新的内容：Lambda 计算映像	为.NET 10、、Go 1.25和Python 3.14添加了 Lambda 计算映像。Corretto 25 Node.js 24	2026 年 7 月 29 日
更新的内容：Bitbucket 应用程序密码、API 令牌或访问令牌	添加对 Bitbucket API 令牌的支持，该令牌取代了过时的应用程序密码。	2026年7月19日
自定义 GitHub 企业角色映射和维护角色名称修复	添加了有关如何根据权限层次结构将 GitHub 企业自定义存储库角色映射到标准角色的文档。将维护级别的角色名称更正为`GITHUB_MAINTAIN` 默认值和示例。	2026 年 4 月 13 日

更新内容： CodeBuild 标准映像的可用运行时	更新了可用的运行时版本并添加了新图像。	2026 年 4 月 7 日
更新的内容：EC2 计算映像	添加了亚马逊 Linux 2023 x86_64 standard: 6.0 和 Ubuntu 24.04 standard: 8.0 计算映像。	2026 年 4 月 7 日
更新内容：运行时版本选择行为	添加了指南，说明当先前的默认版本到期 (EOL) 时，默认运行时版本可能会发生变化。	2026年3月25日
更新内容： CodeBuild 标准映像的可用运行时	更新了可用的运行时版本并添加了新图像。	2026年3月25日
更新的内容：的 AWS 托管（预定义）策略 AWS CodeBuild	AWSCodeBuildAdminAccess 和 AWSCodeBuildDeveloperAccess 策略已更新，包括会话管理器交互式编译调试所需的`ssmmessages:OpenDataChannel` 权限。	2025 年 12 月 1 日
拉取请求评论批准	CodeBuild 支持拉取请求构建策略，为拉取请求触发的构建提供额外控制。	2025 年 8 月 7 日
更新的内容：的 AWS 托管（预定义）策略 AWS CodeBuild	AWSCodeBuildAdminAccess AWSCodeBuildDeveloperAccess、和 AWSCodeBuildReadOnlyAccess 政策已更新。原始资源 `arn:aws:codeconnections:*:*:connection/*` 已更新为 `arn:aws:codeconnections:*:*:*` 。	2025 年 6 月 10 日

CodeBuild 条件键的新参考	添加了一个新的参考页面，其中包含使用 CodeBuild 条件键的示例。请参阅 AWS CodeBuild 条件键。	2025 年 5 月 15 日
新增内容：的 Docker 镜像构建服务器示例 CodeBuild	CodeBuild 现在支持将您的 Docker 编译版本卸载到托管镜像构建服务器。	2025 年 5 月 15 日
新的计算类型：CUSTOM_INSTANCE_TYPE	CodeBuild 现在允许您使用创建具有特定实例类型的预留容量队列CUSTOM_INSTANCE_TYPE 。	2025 年 4 月 23 日
对 CodeBuild 沙盒的新支持	添加了有关使用新 CodeBuild 沙箱的信息。请参阅使用 CodeBuild 沙箱https://docs.aws.amazon.com/codebuild/latest/userguide/sandbox.html调试构建。	2025 年 4 月 7 日
新的 Windows 环境类型	CodeBuild 现在支持 Windows XL 和 2XL 环境类型。有关更多信息，请参阅构建环境计算类型。	2025 年 3 月 31 日
更新了 Amazon S3 缓存	CodeBuild 现在支持 Amazon S3 缓存的新缓存行为。	2025 年 3 月 28 日
新内容： GitHub 操作运行器配置选项	CodeBuild 现在CODEBUILD_CONFIG_GITHUB_ACTIONS_ENTERPRISE_REGISTRATION_NAME 支持企业级注册。	2025 年 3 月 11 日
新增内容：添加新的 webhook 筛选条件类型	添加对新 webhook 筛选条件类型（ ORGANIZATION_NAME ）的支持。	2025 年 3 月 11 日

新增内容：使用 Fastlane 以及 S3 证书存储进行 Apple 代码签名的教程	在 “ CodeBuild 使用 S3 进行证书存储” 中添加使用 Fastlane 进行 Apple 代码签名的新教程	2025 年 2 月 5 日
新内容：使用带 GitHub 证书存储的 Fastlane 进行 Apple 代码签名教程	在 “使用证书存储” 中添加 CodeBuild 使用 GitHub Fastlane 进行 Apple 代码签名的新教程	2025 年 2 月 5 日
新增内容：Buildkite 运行程序	为 Buildkite 运行程序添加新内容	2025 年 1 月 31 日
新增内容：Buildkite 手动 webhook	添加对 Buildkite 手动 webhook 的支持。	2025 年 1 月 31 日
新增内容：批量构建 buildspec 参考	添加对预留容量实例集和 Lambda 环境中的批量构建的支持。	2025 年 1 月 8 日
新增内容：在批量构建中执行并行测试	添加在批量构建中进行并行测试的新内容。	2025 年 1 月 2 日
新增内容：自动重试构建	CodeBuild 现在支持自动重试 webhook 构建。	2024 年 12 月 18 日
新增内容：为自托管运行程序配置私有注册表凭证	在使用来自非私有注册表的自定义映像时，添加对设置注册表凭证的支持。	2024 年 12 月 13 日
新内容： GitHub 操作运行器配置选项	CodeBuild GitHub Actions 自托管运行器现在允许您在组织级别注册跑步者并配置特定的跑步者群组 ID。	2024 年 12 月 12 日
新增内容：添加失败时属性 RETRY	CodeBuild 现在允许您在构建规范RETRY中配置 on-failure 属性为。	2024 年 12 月 12 日

新内容： GitLab 手动 webhook	添加对 GitLab 手动 webhook 的支持。	2024 年 12 月 11 日
更新了内容：更新了别名	更新 Linux-based 标准运行时图像的别名。	2024 年 11 月 22 日
更新内容：运行器支持标签覆盖 CodeBuild-hosted GitLab	添加对 GitLab 运行器自定义图像标签覆盖的支持。	2024 年 11 月 22 日
更新内容： CodeBuild-hosted GitHub 操作运行器支持标签覆盖	为 Act GitHub ions 运行器添加对自定义图像标签覆盖的支持。	2024 年 11 月 22 日
更新的内容：的 AWS 托管（预定义）策略 AWS CodeBuild	AWSCodeBuildAdminAccess AWSCodeBuildDeveloperAccess、和 AWSCodeBuildReadOnlyAccess 政策已更新。原始资源 `arn:aws:codebuild:*` 已更新为 `arn:aws:codebuild:*:*:project/*` 。	2024 年 11 月 15 日
更新内容：预留容量	预留容量实例集现在支持非容器构建：ARM EC2、Linux EC2 和 Windows EC2。	2024 年 11 月 12 日
更新内容：预留容量	预留容量实例集现在支持基于属性的计算。	2024 年 11 月 6 日
新增内容：自动重试构建	CodeBuild 现在允许您为构建启用自动重试。	2024 年 10 月 25 日
新内容：在托管代理服务器 CodeBuild 中为预留容量队列运行	为预留容量实例集添加代理配置支持。	2024 年 10 月 15 日
新内容： Self-managed GitLab 跑步者	为自我管理 GitLab 的跑步者添加新内容	2024 年 9 月 17 日

新内容： GitLab 群组网络挂钩	添加对 GitLab 群组 webhook 的支持。	2024 年 9 月 17 日
新内容：在 INSTALL、PRE_BUILD 和 POST_BUILD 阶段运行 buildspec 命令	增加了对 `-with-buildspec` 的支持。	2024 年 8 月 20 日
更新内容：预留容量	预留容量实例集现在支持 macOS。	2024 年 8 月 19 日
新内容： GitHub 应用程序连接	添加对 GitHub 应用程序连接的支持。	2024 年 8 月 14 日
新增内容：Bitbucket 应用程序连接	添加对 Bitbucket 应用程序连接的支持。	2024 年 8 月 14 日
新增内容：里面有多个访问令牌 CodeBuild	添加对从 AWS Secrets Manager 或通过 AWS CodeConnections 连接中的机密信息向第三方提供商采购访问令牌的支持。	2024 年 8 月 14 日
更新内容：预留容量	预留容量实例集现在支持 ARM Medium、ARM XLarge 和 ARM 2XLarge 计算类型。	2024 年 8 月 5 日
更新内容：预留容量	CodeBuild 现在支持 Windows 上预留容量队列的 VPC 连接。	2024 年 8 月 1 日
新 ARM 计算类型	CodeBuild 现在支持 ARM Medium、ARM XLarge 和 ARM 2xLarge 计算类型。有关更多信息，请参阅构建环境计算类型。	2024 年 7 月 10 日
更新内容：SHA 签名	更新 x86_64 和 ARM 的安全哈希算法（SHA）签名。	2024 年 6 月 19 日

新内容： GitHub 全球和组织网络挂钩	添加对 GitHub 全球和组织网络挂钩的支持。	2024 年 6 月 17 日
新增内容：添加新的 webhook 筛选条件类型	添加对新 webhook 筛选条件类型（ `REPOSITORY_NAME` ）的支持。	2024 年 6 月 17 日
更新的磁盘空间	`ARM Small` 和 `ARM Large` 计算类型现在增加了磁盘空间。	2024 年 6 月 4 日
新内容： GitHub 手动 webhook	添加对 GitHub 手动 webhook 的支持。	2024 年 5 月 23 日
更新内容：预留容量	CodeBuild 现在支持亚马逊 Linux 上的预留容量队列的 VPC 连接。	2024 年 5 月 15 日
更新的内容：Lambda 计算映像	为 .NET 8（ `al-lambda/aarch64/dotnet8` 和 `al-lambda/x86_64/dotnet8` ）添加了 Lambda 支持	2024 年 5 月 8 日
更新配额：构建超时	将最大构建超时配额更新为 2160 分钟（ 36 小时）。	2024 年 5 月 1 日
更新的内容：的 AWS 托管（预定义）策略 AWS CodeBuild	AWSCodeBuildAdminAccess AWSCodeBuildDeveloperAccess、和 AWSCodeBuildReadOnlyAccess 政策已更新，以反映 AWS CodeConnections 品牌重塑。	2024 年 4 月 30 日
新增内容：Bitbucket 应用程序密码、API 令牌或访问令牌	增加了对 Bitbucket 访问令牌的支持。	2024 年 4 月 11 日
新增内容：Auto-discover 中的报告 CodeBuild	CodeBuild 现在支持报告自动发现。	2024 年 4 月 4 日

新内容： Self-hosted GitHub 动作运行器	为自托管的 Act GitHub ions 运行器添加新内容	2024 年 4 月 2 日
新内容： GitLab 连接	添加对连接 GitLab 和 GitHub 自我管理连接的支持。	2024 年 3 月 25 日
新增内容：添加新的 webhook 事件和筛选条件类型	添加对新 webhook 事件（ RELEASED 和 PRERELEASED ）和筛选条件类型（ TAG_NAME 和 RELEASE_NAME ）的支持。	2024 年 3 月 15 日
新增内容：添加新的 webhook 事件：PULL_REQUEST_CLOSED	添加对以下新 webhook 事件的支持：PULL_REQUEST_CLOSED 。	2024 年 2 月 20 日
更新的内容：Docker 镜像由 CodeBuild	添加了对 Windows Server 2019（ windows-base:2019-3.0 ）的支持	2024 年 2 月 7 日
更新的内容：Docker 镜像由 CodeBuild	添加对适用于 Amazon Linux 2023（ al2/aarch64/standard/3.0 ）的新运行时系统的支持	2024 年 1 月 29 日
新内容：预留容量	CodeBuild 现在支持预留容量队列。 CodeBuild	2024 年 1 月 18 日
新的计算类型	CodeBuild 现在支持 Linux XLarge 计算类型。有关更多信息，请参阅构建环境计算类型。	2024 年 1 月 8 日
更新的内容：Docker 镜像由 CodeBuild	添加对适用于 Amazon Linux 2（ al2/standard/5.0 ）和 Ubuntu（ ubuntu/standard/7.0 ）的新运行时系统的支持	2023 年 12 月 14 日

更新的内容：Docker 镜像由 CodeBuild	添加对新 Lambda 计算映像的支持	2023 年 12 月 8 日
新内容： AWS Lambda 计算	为 AWS Lambda 计算添加新内容	2023 年 11 月 6 日
更新的内容：Docker 镜像由 CodeBuild	增加了对 Amazon Linux 2 (`al2/standard/5.0`) 的支持	2023 年 5 月 17 日
对以下各项的托管策略的更改 CodeBuild	现已提供有关 AWS 托管策略更新的详细信息。 CodeBuild 有关更多信息，请参阅 CodeBuild 对 AWS 托管策略的更新。	2023 年 5 月 16 日
更新的内容：Docker 镜像由 CodeBuild	删除了对 Amazon Linux 2 (`al2/standard/3.0`) 的支持，并增加了对 Amazon Linux 2 (`al2/standard/corretto8`) 和 Amazon Linux 2 (`al2/standard/corretto11`) 的支持	2023 年 5 月 9 日
更新的内容：Docker 镜像由 CodeBuild	增加了对 Ubuntu 22.04 (`ubuntu/standard/7.0`) 的支持	2023 年 4 月 13 日
更新的内容：Docker 镜像由 CodeBuild	删除了对 Ubuntu 18.04 (`ubuntu/standard/4.0`) 和 Amazon Linux 2 (`al2/aarch64/standard/1.0`) 的支持	2023 年 3 月 31 日

更新内容：移除 VPC 限制	移除以下限制：如果您配置 CodeBuild 为使用 VPC，则不支持本地缓存。从 02/28 /22 开始，您的 VPC 构建将花费更长的时间，因为每次构建都将使用新的 Amazon EC2 实例。	2023 年 3 月 1 日
更新的内容：Docker 镜像由 CodeBuild	删除了对 Ubuntu 18.04 (`ubuntu/standard/3.0`) 和 Amazon Linux 2 (`al2/standard/2.0`) 的支持	2022 年 6 月 30 日
Amazon ECR 示例：限制映像访问	使用 CodeBuild 凭证提取 Amazon ECR 图像时，您可以限制对特定 CodeBuild 项目的图像访问权限。有关更多信息，请参阅 Amazon ECR 示例。	2022 年 3 月 10 日
增加了区域支持	现在，以下其他区域支持该 `ARM_CONTAINER` 计算类型：亚太地区（首尔）、加拿大（中部）、欧洲地区（伦敦）和欧洲地区（巴黎）。有关更多信息，请参阅构建环境计算类型。	2022 年 3 月 10 日
新的 VPC 限制	如果您配置 CodeBuild 为使用 VPC，则不支持本地缓存。从 02/28 /22 开始，您的 VPC 构建将花费更长的时间，因为每次构建都将使用新的 Amazon EC2 实例。	2022 年 2 月 25 日

批量报告模式	CodeBuild 现在允许您选择如何将批量生成状态发送给项目的源提供商。有关更多信息，请参阅批量报告模式。	2021 年 10 月 4 日
新的计算类型	CodeBuild 现在支持小型 ARM 计算类型。有关更多信息，请参阅构建环境计算类型。	2021 年 9 月 13 日
公共构建项目	CodeBuild 现在允许您无需访问 AWS 帐户即可向公众提供构建项目的构建结果。有关更多信息，请参阅公共构建项目。	2021 年 8 月 11 日
针对批量构建的会话调试	CodeBuild 现在支持批量构建的会话调试。有关更多信息，请参阅构建图和构建列表。	2021 年 3 月 3 日
项目级别的并发构建限制	CodeBuild 现在允许您限制构建项目的并发构建次数。有关更多信息，请参阅项目配置和并发BuildLimit。	2021 年 2 月 16 日
新的构建规范属性：s3-prefix	CodeBuild 现在为工件提供了 `s3-prefix` buildspec 属性，允许您为上传到 Amazon S3 的项目指定路径前缀。有关更多信息，请参阅 s3-prefix。	2021 年 2 月 9 日
新的构建规范属性：on-failure	CodeBuild 现在为构建阶段提供了 `on-failure` buildspec 属性，允许您确定构建阶段失败时会发生什么。有关更多信息，请参阅 on-failure。	2021 年 2 月 9 日

新的构建规范属性：exclude-paths	CodeBuild 现在为构件提供了 `exclude-paths` buildspec 属性，允许您从构建工件中排除路径。有关更多信息，请参阅 exclude-paths。	2021 年 2 月 9 日
新的构建规范属性：enable-symlinks	CodeBuild 现在为构件提供了 `enable-symlinks` buildspec 属性，允许您在 ZIP 工件中保留符号链接。有关更多信息，请参阅 enable-symlinks。	2021 年 2 月 9 日
构建规范构件名称增强功能	CodeBuild 现在允许该`artifacts/name` 属性包含路径信息。有关更多信息，请参阅名称。	2021 年 2 月 9 日
代码覆盖率报告	CodeBuild 现在提供代码覆盖率报告。有关更多信息，请参阅代码覆盖率报告。	2020 年 7 月 30 日
批量构建	CodeBuild 现在支持运行项目的并行和协调构建。有关更多信息，请参阅批量构建 CodeBuild。	2020 年 7 月 30 日
Windows Server 2019 映像	CodeBuild 现在提供了 Windows Server Core 2019 版本映像。有关更多信息，请参阅提供的 Docker 镜像 CodeBuild。	2020 年 7 月 20 日

会话管理器	CodeBuild 现在允许您暂停正在运行的构建，然后使用 AWS Systems Manager 会话管理器连接到构建容器并查看容器的状态。有关更多信息，请参阅会话管理器。	2020 年 7 月 20 日
更新的主题	CodeBuild 现在支持在 buildspec 文件中指定要在其构建环境中使用的外壳。有关更多信息，请参阅构建规范参考。	2020 年 6 月 25 日
使用测试框架测试报告	添加了几个主题，描述如何使用多个 CodeBuild 测试框架生成测试报告。有关更多信息，请参阅使用测试框架测试报告。	2020 年 5 月 29 日
更新的主题	CodeBuild 现在支持向报告组添加标签。有关更多信息，请参阅 ReportGroup。	2020 年 5 月 21 日
支持测试报告	CodeBuild 对测试报告的支持现已正式推出。	2020 年 5 月 21 日
更新的主题	CodeBuild 现在支持为 Github 和 Bitbucket 创建创建 webhook 过滤器，仅当头部提交消息与指定表达式匹配时才会触发构建。有关更多信息，请参阅GitHub 拉取请求和 webhook 过滤器示例和 Bitbucket 拉取请求和 webhook 过滤器示例。	2020 年 5 月 6 日

新主题	CodeBuild 现在支持共享生成项目和报表组资源。有关更多信息，请参阅使用共享项目和使用共享报告组。	2019 年 12 月 13 日
新增和更新的主题	CodeBuild 现在支持构建项目运行期间的测试报告。有关更多信息，请参阅使用测试报告、创建测试报告和使用 AWS CLI 示例创建测试报告。	2019 年 11 月 25 日
更新的主题	CodeBuild 现在支持 Linux GPU 和 Arm 环境类型以及2xlarge计算类型。有关更多信息，请参阅构建环境计算类型。	2019 年 11 月 19 日
更新的主题	CodeBuild 现在支持所有版本的内部版本号、导出环境变量和 AWS Secrets Manager 集成。有关更多信息，请参阅构建规范语法中的导出变量和 Secrets Manager。	2019 年 11 月 6 日
新主题	CodeBuild 现在支持通知规则。您可以使用通知规则向用户通知构建项目中的重要更改。有关更多信息，请参阅创建通知规则。	2019 年 11 月 5 日
更新的主题	CodeBuild 现在支持安卓版本 29 和 Go 版本 1.13 运行时。有关更多信息，请参阅 CodeBuild 提供的 Docker 映像和构建规范语法。	2019 年 9 月 10 日

更新的主题	在创建项目时，您现在可以选择 Amazon Linux 2 (AL2) 托管映像。有关更多信息，请参阅 CodeBuild 提供的 Docker 映像和 CodeBuild 的构建规范文件示例中的运行时版本。	2019 年 8 月 16 日
更新的主题	创建项目时，您现在可以选择禁用 S3 日志的加密，如果您使用 Git-based 源存储库，则可以选择 Git 子模块。有关更多信息，请参阅中的创建构建项目 CodeBuild。	2019 年 3 月 8 日
新主题	CodeBuild 现在支持本地缓存。创建构建时，可以在四种模式中的一种或多种模式中指定本地缓存。有关更多信息，请参阅在中构建缓存 CodeBuild。	2019 年 2 月 21 日
新主题	CodeBuild 现在支持 webhook 过滤器组来指定触发生成的事件。有关更多信息，请参阅筛选 GitHub webhook 事件和过滤 Bitbucket webhook 事件。	2019 年 2 月 8 日
新主题	《CodeBuild 用户指南》现在显示了如何 CodeBuild 与代理服务器一起使用。有关更多信息，请参见 CodeBuild 与代理服务器一起使用。	2019 年 2 月 4 日

更新的主题	CodeBuild 现在支持使用另一个 AWS 账户中的亚马逊 ECR 图片。已更新多个主题以反映这一变化，包括 Amazon ECR 示例 CodeBuild、创建构建项目和创建 CodeBuild 服务角色。	2019 年 1 月 24 日
支持专用 Docker 注册表	CodeBuild 现在支持使用存储在私有注册表中的 Docker 镜像作为运行时环境。有关更多信息，请参阅私有注册表与 AWS Secrets Manager 示例。	2019 年 1 月 24 日
更新的主题	CodeBuild 现在支持使用访问令牌连接 GitHub（使用个人访问令牌）和 Bitbucket（使用应用程序密码）存储库。有关更多信息，请参阅创建构建项目（控制台）和将访问令牌与源提供商结合使用。	2018 年 12 月 6 日
更新的主题	CodeBuild 现在支持新的构建指标，用于衡量构建中每个阶段的持续时间。有关更多信息，请参阅 CodeBuild CloudWatch 指标。	2018 年 11 月 15 日
VPC 端点策略主题	亚马逊 VPC 终端节点 CodeBuild 目前支持政策。有关更多信息，请参阅为 CodeBuild 创建 VPC 终端节点策略。	2018 年 11 月 9 日
更新了内容	更新了主题来反映新控制台体验。	2018 年 10 月 30 日

Amazon EFS 示例	CodeBuild 可以在构建期间使用项目构建规范文件中的命令挂载 Amazon EFS 文件系统。有关更多信息，请参阅的 Amazon EFS 示例 CodeBuild 。	2018 年 10 月 26 日
Bitbucket Webhook	CodeBuild 当您使用 Bitbucket 作为存储库时，现在支持网络挂钩。有关更多信息，请参阅 CodeBuild 的 Bitbucket 拉取请求示例。	2018 年 10 月 2 日
S3 日志	CodeBuild 现在支持在 S3 存储桶中生成日志。以前，您只能使用日志生成 CloudWatch 日志。有关更多信息，请参阅创建项目。	2018 年 9 月 17 日
多个输入源和多个输出工件	CodeBuild 现在支持使用多个输入源和发布多组工件的项目。有关更多信息，请参阅多个输入源和输入构件示例以及与多个输入源 CodeBuild 和输出构件的CodePipeline 集成示例。	2018 年 8 月 30 日
语义版本控制示例	《 CodeBuild 用户指南》现在有一个基于用例的示例，该示例演示了如何使用语义版本控制在构建时创建构件名称。有关更多信息，请参阅使用语义版本控制指定构建构件示例。	2018 年 8 月 14 日

新的静态网站示例	《CodeBuild 用户指南》现在有一个基于用例的示例，演示了如何在 S3 存储桶中托管构建输出。此示例利用了对未加密构建构件的最新支持。有关更多信息，请参阅创建将构建输出托管在 S3 存储桶中的静态网站。	2018 年 8 月 14 日
支持使用语义版本控制覆盖构件名称	现在，您可以使用语义版本控制来指定用于命名构建构 CodeBuild 件的格式。这很有用，因为具有硬编码名称的构建构件将覆盖之前使用同一硬编码名称的构建构件。例如，如果每天触发一个构建多次，则现在可以为此构建的构件名称添加时间戳。每个构建构件名称是唯一的，不会覆盖之前构建的构件。	2018 年 8 月 7 日
支持未加密的构建构件	CodeBuild 现在支持使用未加密构建构件的构建。有关更多信息，请参阅创建构建项目（控制台）。	2018 年 7 月 26 日
支持亚马逊 CloudWatch 指标和警报	CodeBuild 现在提供与 CloudWatch 指标和警报的集成。您可以使用 CodeBuild 或 CloudWatch 控制台监控项目和账户级别的构建。有关更多信息，请参阅监控构建。	2018 年 7 月 19 日
支持报告构建的状态	CodeBuild 现在可以向源提供商报告构建的开始和完成状态。有关更多信息，请参见在中创建构建项目 CodeBuild。	2018 年 7 月 10 日

环境变量已添加到 CodeBuild 文档中	构建环境中的环境变量页面使用 CODEBUILD_BUILD_ID、CODEBUILD_LOG_PATH 和 CODEBUILD_START_TIME 环境变量进行了更新。	2018 年 7 月 9 日
支持构建规范文件中的 finally 语句块	CodeBuild 文档已更新，其中包含有关 buildspec 文件中可选finally块的详细信息。finally 语句块命令总是在其相应的命令语句块命令之后运行。有关更多信息，请参阅构建规范语法。	2018 年 6 月 20 日
CodeBuild 代理更新通知	CodeBuild 文档已更新，详细介绍了如何使用 Amazon SNS，以便在新版本的 CodeBuild 代理发布时收到通知。有关更多信息，请参阅接收新 AWS CodeBuild 代理版本的通知。	2018 年 6 月 15 日

早期更新

下表描述了 2018 年 6 月之前每次发布 AWS CodeBuild 用户指南 时进行的重要更改。

更改	描述	日期
支持 Windows 构建	CodeBuild 现在支持微软 Windows 服务器平台的构建，包括在 Windows 上为.NET Core 2.0 预先打包的编译环境。有关更多信息，请参阅运行微软 Windows 示例 CodeBuild。	2018 年 5 月 25 日

更改	描述	日期
支持构建幂等性	当您使用 AWS Command Line Interface (AWS CLI) 运行`start-build` 命令时，可以指定构建是指数性的。有关更多信息，请参阅 运行构建 (AWS CLI)。	2018 年 5 月 15 日
支持覆盖多个构建项目设置	现在，在创建构建时，可以覆盖多个构建项目设置。覆盖仅针对该构建。有关更多信息，请参阅 手动运行 AWS CodeBuild 构建。	2018 年 5 月 15 日
VPC 端点支持	现在，您可以使用 VPC 端点提高构建的安全性。有关更多信息，请参阅 使用 VPC 端点。	2018 年 3 月 18 日
支持触发器	现在，您可以创建触发器，按固定频率安排构建。有关更多信息，请参阅 创建 AWS CodeBuild 触发器。	2018 年 3 月 28 日
FIPS 端点文档	现在，您可以了解如何使用 AWS Command Line Interface (AWS CLI) 或 AWS SDK CodeBuild 来指示使用四个联邦信息处理标准 (FIPS) 端点之一。有关更多信息，请参阅 指定 AWS CodeBuild 端点。	2018 年 3 月 28 日
AWS CodeBuild 可在亚太地区（孟买）、欧洲（巴黎）和南美洲（圣保罗）使用	AWS CodeBuild 现已在亚太地区（孟买）、欧洲（巴黎）和南美洲（圣保罗）地区上市。有关更多信息，请参阅Amazon Web Services 一般参考中的 AWS CodeBuild。	2018 年 3 月 28 日

更改	描述	日期
GitHub 企业服务器支持	CodeBuild 现在可以根据存储在 GitHub 企业服务器存储库中的源代码进行构建。有关更多信息，请参阅 运行 GitHub 企业服务器示例。	2018 年 1 月 25 日
Git 克隆深度支持	CodeBuild 现在支持创建浅层克隆，将历史记录截断为指定的提交次数。有关更多信息，请参阅 创建构建项目。	2018 年 1 月 25 日
VPC 支持	VPC-enabled 构建版本现在可以访问您的 VPC 内的资源。有关更多信息，请参阅 VPC 支持。	2017 年 11 月 27 日
依赖项缓存支持	CodeBuild 现在支持依赖缓存。这 CodeBuild 允许将构建环境的某些可重复使用的部分保存在缓存中，并在构建中使用。	2017 年 11 月 27 日
构建徽章支持	CodeBuild 现在支持使用编译徽章，它提供可嵌入的、动态生成的图像（徽章），用于显示项目最新版本的状态。有关更多信息，请参阅 构建徽章示例。	2017 年 11 月 27 日
AWS Config 整合	AWS Config 现在支持 CodeBuild 作为 AWS 资源，这意味着该服务可以跟踪您的 CodeBuild 项目。有关的更多信息 AWS Config，请参阅AWS Config 样本。	2017 年 10 月 20 日

更改	描述	日期
在 GitHub 存储库中自动重建更新的源代码	如果您的源代码存储在存储 GitHub 库中，则无论何时将代码更改推送 AWS CodeBuild 到存储库，您都可以启用重建源代码的功能。有关更多信息，请参阅 运行 GitHub 拉取请求和 webhook 过滤器示例。	2017 年 9 月 21 日
在 Amazon EC2 Systems Manager Parameter Store 中存储和检索敏感或大型环境变量的新方法	现在，您可以使用 AWS CodeBuild 控制台或 AWS CLI 来检索存储在 Amazon EC2 系统管理器参数存储中的敏感或大型环境变量。现在也可以使用 AWS CodeBuild 控制台将这些类型的环境变量存储在 Amazon EC2 Systems Manager Parameter Store 中。以前，您只能通过将这些类型的环境变量包含在构建规范中或运行构建命令以自动化 AWS CLI来检索这些变量。您只能通过使用 Amazon EC2 Systems Manager Parameter Store 控制台来存储这些类型的环境变量。有关更多信息，请参阅创建构建项目、更改构建项目设置和手动运行构建。	2017 年 9 月 14 日
构建删除支持	您现在可以在 AWS CodeBuild 中删除构建。有关更多信息，请参阅 删除构建。	2017 年 8 月 31 日

更改	描述	日期
使用生成规范检索存储在 Amazon EC2 Systems Manager Parameter Store 中的敏感或大型环境变量的更新方法	AWS CodeBuild 现在可以更轻松地使用构建规范来检索存储在 Amazon EC2 系统管理器参数存储中的敏感或大型环境变量。以前，您只能通过运行构建命令自动化 AWS CLI来检索这些类型的环境变量。有关更多信息，请参阅buildspec 语法中的 `parameter-store` 映射。	2017 年 8 月 10 日
AWS CodeBuild 支持 Bitbucket	CodeBuild 现在可以使用 Bitbucket 存储库中存储的源代码进行构建。有关更多信息，请参阅创建构建项目和手动运行构建：	2017 年 8 月 10 日
AWS CodeBuild 在美国西部（加利福尼亚北部）、欧洲（伦敦）和加拿大（中部）提供	AWS CodeBuild 现已在美国西部（加利福尼亚北部）、欧洲（伦敦）和加拿大（中部）地区推出。有关更多信息，请参阅Amazon Web Services 一般参考 中的 AWS CodeBuild。	2017 年 6 月 29 日
已支持其他构建规范文件名称和位置	现在，您可以不为构建项目指定源代码根目录处的名为 `buildspec.yml` 的默认构建规范文件，而是指定使用文件名称或位置与之不同的构建规范文件。有关更多信息，请参阅 buildspec 文件名称和存储位置。	2017 年 6 月 27 日

更改	描述	日期
更新了构建通知示例	CodeBuild 现在通过亚马逊 CloudWatch 活动和亚马逊简单通知服务 (亚马逊 SNS) 为生成通知提供内置支持。先前的构建通知示例已更新为演示此新行为。	2017 年 6 月 22 日
添加了自定义映像中的 Docker 示例	添加了演示如何使用 CodeBuild 自定义 Docker 构建镜像来构建和运行 Docker 镜像的示例。有关更多信息，请参阅自定义映像示例中的 Docker。	2017 年 6 月 7 日
获取 GitHub 拉取请求的源代码	当您使用依赖存储 GitHub库中存储 CodeBuild 的源代码运行构建时，您现在可以指定要构建的 GitHub 拉取请求 ID。您还可以改为指定提交 ID、分支名称或标签名称。有关更多信息，请参阅运行构建（控制台）中的源版本值或运行构建 (AWS CLI) 中的 `sourceVersion` 值。	2017 年 6 月 6 日

更改	描述	日期
更新了构建规范版本	发布了新版本的构建规范格式。版本 0.2 解决了在默认 shell 的单独实例中 CodeBuild 运行每个编译命令的问题。此外，在版本 0.2 中，`environment_variables` 已重命名为 `env`，而且 `plaintext` 已重命名为 `variables`。有关更多信息，请参阅 的构建规范参考 CodeBuild。	2017 年 5 月 9 日
用于构建镜像的 Dockerfile 可在中找到 GitHub	提供的许多编译镜像的定义 AWS CodeBuild 都以 Dockerfile 的形式提供。 GitHub有关更多信息，请参阅 Docker 镜像由 CodeBuild中表的“定义”列。	2017 年 5 月 2 日
AWS CodeBuild 可在欧洲（法兰克福）、亚太地区（新加坡）、亚太地区（悉尼）和亚太地区（东京）使用	AWS CodeBuild 现已在欧洲（法兰克福）、亚太地区（新加坡）、亚太地区（悉尼）和亚太地区（东京）地区推出。有关更多信息，请参阅Amazon Web Services 一般参考 中的 AWS CodeBuild。	2017 年 3 月 21 日
CodePipeline 的测试操作支持 CodeBuild	现在，您可以在使用的测试操作 CodePipeline 中添加到管道中 CodeBuild。有关更多信息，请参阅 向管道添加 CodeBuild 测试操作（CodePipeline 控制台）。	2017 年 3 月 8 日

更改	描述	日期
构建规范文件支持从选定的顶级目录中获取构建输出	Buildspec 文件现在允许您指定各个顶级目录，您可以指示 CodeBuild 将这些目录的内容包含在构建输出构件中。为此，您可以使用 `base-directory` 映射。有关更多信息，请参阅 buildspec 语法。	2017 年 2 月 8 日
Built-in 环境变量	AWS CodeBuild 提供额外的内置环境变量供您的构建使用。这些包括描述启动了构建项目的实体的环境变量、指向源代码存储库的 URL 以及源代码的版本 ID 等。有关更多信息，请参阅 构建环境中的环境变量。	2017 年 1 月 30 日
AWS CodeBuild 在美国东部（俄亥俄州）可用	AWS CodeBuild 现已在美国东部（俄亥俄州）地区推出。有关更多信息，请参阅Amazon Web Services 一般参考 中的 AWS CodeBuild。	2017 年 1 月 19 日
Shell 和命令行为信息	CodeBuild 在构建环境的默认 shell 的单独实例中运行你指定的每条命令。这种默认行为可对您的命令产生一些意想不到的副作用。我们推荐了一些方法，用于在需要时处理这种默认行为。有关更多信息，请参阅 构建环境中的 Shell 和命令。	2016 年 12 月 9 日

更改	描述	日期
环境变量信息	CodeBuild 提供了多个环境变量，您可以在编译命令中使用这些变量。您也可以定义自己的环境变量。有关更多信息，请参阅 构建环境中的环境变量。	2016 年 12 月 7 日
故障排除主题	现已提供故障排除信息。有关更多信息，请参阅 故障排除 AWS CodeBuild。	2016 年 12 月 5 日
Jenkins 插件初始版本	这是 CodeBuild Jenkins 插件的初始版本。有关更多信息，请参阅 使用 AWS CodeBuild 和詹金斯在一起。	2016 年 12 月 5 日
用户指南初始版本	这是 CodeBuild 用户指南 的初始版本。	2016 年 12 月 1 日

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。