用户指南

AWS CodePipeline

API 版本 2015-07-09

AWS CodePipeline: 用户指南

Table of Contents

什么是 AWS CodePipeline？

AWS CodePipeline 是一项持续交付服务，可用于对发布软件所需的步骤进行建模、可视化和自动化。您可以快速建模和配置软件发布过程的不同阶段。 CodePipeline 自动执行持续发布软件更改所需的步骤。有关定价的信息 CodePipeline，请参阅定价。

主题

持续交付和持续集成

CodePipeline 是一项持续交付服务，可自动构建、测试软件并将其部署到生产环境中。

持续交付是实现发布流程自动化的软件开发方法。每个软件更改都将自动生成、测试并部署到生产环境中。在最终推送到生产环境之前，可由人员、自动化测试或业务规则决定最后的推送何时发生。虽然每次成功的软件更改都可以通过持续交付立即发布到生产环境中，但并非所有更改都需要立即发布。

持续集成是一种软件开发实践，其中团队成员使用版本控制系统，并将其工作频繁地集成到同一位置，如主分支。每项更改都经过生成和验证，以尽可能快地检测到集成错误。持续交付会自动执行整个软件发布过程，一直到最后的生产部署，而持续集成重点关注自动生成和测试代码。

有关更多信息，请参阅《实践持续集成和持续交付 AWS：使用加速软件交付》 DevOps。

您可以使用 CodePipeline 控制台、 AWS Command Line Interface (AWS CLI)、 AWS 软件开发工具包或它们的任意组合来创建和管理您的管道。

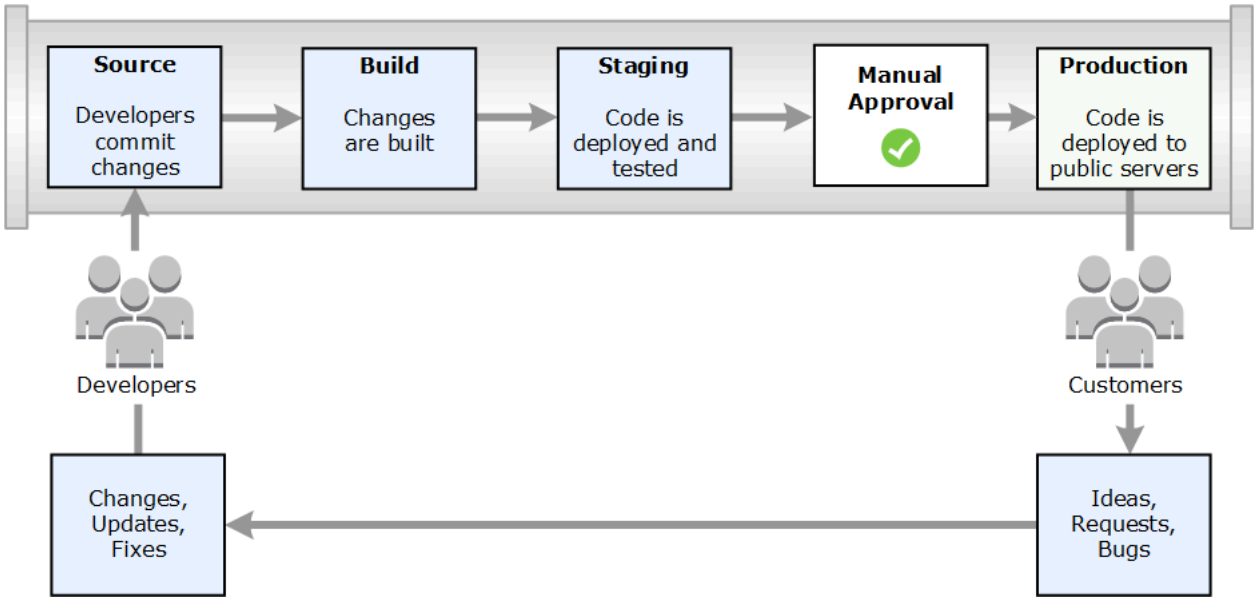
我能用什么做 CodePipeline呢？

您可以使用 CodePipeline 来帮助您在云中自动构建、测试和部署应用程序。具体来说，您可以：

- 自动化发布流程：从源存储库开始，一直到构建、测试和部署，实现发布流程的端到端 CodePipeline 完全自动化。您可以通过在 Source 阶段之外的任何阶段中包括手动审批操作，来防止更改在管道中继续处理。您可以在需要的时间、按需要的方式在所选系统上跨一个实例或多个实例发布。
- 建立一致的发布流程：为每次代码更改定义一组一致的步骤。 CodePipeline 根据您的标准运行发布的每个阶段。
- 加快交付速度，同时提高质量：您可以自动执行发布过程，以允许开发人员逐步测试和发布代码，并加快向客户发布新功能的速度。
- 使用您常用的工具：您可以将现有源代码、生成和部署工具纳入管道中。有关目前支持的第三方工具 AWS 服务 的完整列表 CodePipeline，请参阅产品和服务与 CodePipeline。
- 一目了然地查看进度：您可以查看管道的实时状态，检查任何警报的详细信息，重试失败的阶段或操作，查看每个阶段的最新管道执行中使用的源修订的详细信息，以及手动重新运行任意管道。
- 查看管道详细历史信息：您可以查看有关管道执行的详细信息，包括开始时间和结束时间、运行持续时间和执行 ID。

快速浏览一下 CodePipeline

下图显示了使用发布过程的示例 CodePipeline。



在此示例中，当开发人员向源存储库提交更改时， CodePipeline会自动检测更改。系统将生成这些更改，如果配置了任何测试，则会运行这些测试。测试完成后，将生成的代码部署到暂存服务器进行测试。在登台服务器上， CodePipeline 运行更多测试，例如集成测试或负载测试。成功完成这些测试后，在已添加到管道中的手动批准操作获得批准后，会将经过测试和批准的代码 CodePipeline 部署到生产实例。

CodePipeline 可以使用 CodeDeploy、 AWS Elastic Beanstalk或将应用程序部署到 EC2 实例 AWS OpsWorks Stacks。 CodePipeline 还可以使用 Amazon ECS 将基于容器的应用程序部署到服务中。开发人员还可以使用随附的集成点 CodePipeline 来插入其他工具或服务，包括构建服务、测试提供者或其他部署目标或系统。

管道可以很简单，也可以很复杂，具体由您的发布过程决定。

我该如何开始 CodePipeline？

首先，请执行 CodePipeline以下操作：

1. 阅读CodePipeline 概念 本节，了解 CodePipeline 工作原理。
2. CodePipeline 按照中的步骤@@ 准备使用入门 CodePipeline。
3. CodePipeline 按照CodePipeline 教程教程中的步骤进行@@ 实验。
4. 按照中的步骤操作， CodePipeline 用于您的新项目或现有项目在中创建管道 CodePipeline。

CodePipeline 概念

如果您了解中使用的概念和术语，则可以更轻松地对自动发布流程进行建模和配置 AWS CodePipeline。以下是您在使用时需要了解的一些概念 CodePipeline。

有关 DevOps 管道的示例，请参见DevOps 管道示例。

以下术语用于 CodePipeline：

主题

- 管线
- 管道执行
- 舞台操作
- 操作执行

- 执行类型
- 操作类型
- 构件
- 源修订
- 触发
- Variables

管线

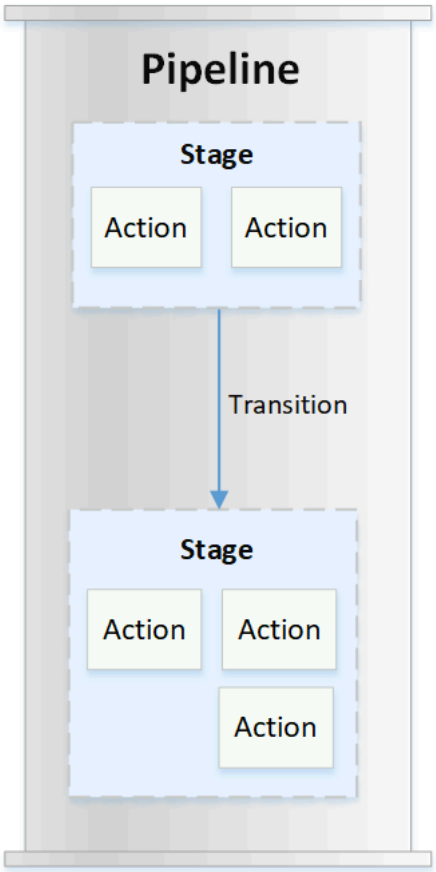
管道是一种工作流程结构，描述软件更改如何经过发布过程。每个管道由一系列阶段 组成。

阶段

阶段用于隔离环境并限制该环境中并发更改数的逻辑单元。每个阶段都包含在应用程序构件上执行的操作。您的源代码是一个构件示例。阶段可能是构建阶段，在该阶段中构建源代码并运行测试。它也可以是部署阶段，在该阶段中将代码部署到运行时环境。每个阶段都由一系列串行或并行操作 组成。

转换

转换 是指管道执行移动到管道中下一阶段的点。您可以禁用某个阶段的入站转换，以防止执行进入该阶段，然后您可以启用转换以允许执行继续。当多个执行达到某个已禁用转换时，在启用转换时，只有最新的执行继续到下一个阶段。这意味着在禁用了转换时，较新的执行将继续取代等待中的执行，然后在启用转换后，继续执行的执行是进行取代的执行。



操作

操作 是对应用程序代码执行的一组操作，并配置为使操作在管道中的指定点运行。这可能包括来自代码更改的源操作、将应用程序部署到实例的操作等。例如，部署阶段可能包含将代码部署到计算服务（例如 Amazon EC2 或 AWS Lambda）的部署操作。

有效的 CodePipeline 操作类型为`sourcebuild`、`test`、`deploy`、`approval`、和`invoke`。有关操作提供程序的列表，请参阅 中的有效操作类型和提供者 CodePipeline 。

操作能够以串行或并行方式运行。有关阶段中串行和并行操作的信息，请参阅操作结构要求中的`runOrder` 信息。

管道执行

执行 是管道发布的一组更改。每个管道执行均唯一并有自己的 ID。一个执行对应于一组更改，例如合并提交或手动发布最新提交。两个执行可以在不同的时间发布相同的一组更改。

虽然管道可以同时处理多个执行，但一个管道阶段一次只处理一个执行。为此，阶段在执行的处理期间将锁定。两个管道执行不能同时占据同一个阶段。等待进入占用阶段的执行称为入站执行。入站执行仍可能失败、被取代或手动停止。有关入站执行如何工作的更多信息，请参阅入站执行的工作原理。

管道执行按顺序遍历管道阶段。管道的有效状态为 InProgress、Stopping、Stopped、Succeeded、Superseded 和 Failed。

有关更多信息，请参阅PipelineExecution。

停止的执行

可以手动停止管道执行，以便正在进行的管道执行不会继续通过管道。如果手动停止，管道执行会显示 Stopping 状态，直到它完全停止。然后它显示 Stopped 状态。可以重试 Stopped 管道执行。

可以使用两种方法停止管道执行：

- 停止并等待
- 停止并放弃

有关停止执行的使用案例信息和这些选项的序列详细信息，请参阅管道执行的停止方式。

失败的执行

如果执行失败，则会停止执行并且不会完全遍历管道。其状态为 FAILED 状态，并且阶段取消锁定。更近的执行可以跟上并进入取消锁定的阶段，然后将其锁定。您可以重试失败的执行，除非失败的执行已被取代或不可重试。您可以将失败的阶段回滚到之前成功执行的阶段。

执行模式

要通过管道传递最新的一组更改，将会传递较新的执行，并替换已在管道中运行的较早的执行。发生这种情况时，较新的执行将取代较早的执行。一个执行在特定点（即阶段之间的点）被更新的执行取代。SUPERSEDED 是默认的执行模式。

在 SUPERSEDED 模式下，如果执行正在等待进入锁定阶段，则最近的执行可能会赶上并取代它。较新的执行现在等待阶段取消锁定，而被取代的执行停止，且状态为 SUPERSEDED。当某个管道执行被取代时，该执行将停止，并且不会完全遍历管道。当某个执行在此阶段中被取代之后，您无法再重试该执行。其他可用的执行模式包括并行或队列模式。

有关执行模式和锁定阶段的更多信息，请参阅如何在 SUPERSEDED 模式下处理执行。

舞台操作

当管道执行通过一个阶段时，该阶段正在完成其中的所有操作。有关阶段操作的工作原理的信息以及有关锁定阶段的信息，请参见如何在 SUPERSEDED 模式下处理执行。

有效的阶段状态为 InProgress、Stopping、Stopped、Succeeded 和 Failed。除非失败的阶段不可重试，否则您都可以重试这些阶段。有关更多信息，请参阅StageExecution。您可以将阶段回滚到之前成功执行的指定阶段。可以将舞台配置为在出现故障时自动回滚，如中所述配置阶段回滚。有关更多信息，请参阅RollbackStage。

操作执行

操作执行 是完成在指定构件上执行操作的已配置操作的过程。这可以是输入构件和/或输出构件。例如，构建操作可能会在输入构件上运行构建命令，如编译应用程序源代码。操作执行详细信息包括操作执行 ID、相关的管道执行源触发器以及操作的输入和输出构件。

有效的操作状态为 InProgress、Abandoned、Succeeded 和 Failed。有关更多信息，请参阅ActionExecution。

执行类型

管道或阶段执行可以是标准执行，也可以是回滚执行。

对于标准类型，执行具有唯一的 ID，并且是完整的管道运行。管道回滚有一个要回滚的阶段，而该阶段的成功执行是要回滚到的目标执行。目标管道执行用于检索要重新运行的阶段的源版本和变量。

操作类型

操作类型是预先配置的动作，可在中进行选择。 CodePipeline操作类型按照它的所有者、提供方、版本和类别来定义。操作类型可提供自定义参数，用于完成管道中的操作任务。

有关您可以根据操作类型集成到管道中的第三方产品和服务的信息，请参阅与动 CodePipeline 作类型的集成。 AWS 服务

有关中操作类型支持的集成模型的信息 CodePipeline，请参阅集成模型参考。

有关第三方提供商如何在中设置和管理操作类型的信息 CodePipeline，请参阅使用操作类型。

构件

构件 是指通过管道操作处理的数据集合，例如应用程序源代码、构建应用程序、依赖关系、定义文件、模板等。构件由一些操作生成，由另外一些操作使用。在管道中，构件可以是操作处理的一组文件（输入构件），或者是已完成操作的更新输出（输出构件）。

操作将输出传递给另一个操作，以便使用管道工件存储桶进行进一步处理。 CodePipeline 将工件复制到工件存储区，操作将在那里捡起它们。有关构件的更多信息，请参阅 输入和输出构件。

源修订

当您进行源代码更改时，将创建新版本。源修订 是触发管道执行的源更改版本。执行会处理源代码修订。对于 GitHub 和 CodeCommit 存储库，这是提交。对于 S3 存储桶或操作，则为对象版本。

可以使用您指定的源修订（例如提交）启动管道执行。执行将处理指定的修订，并覆盖本应用于执行的修订。有关更多信息，请参阅 使用源修订覆盖启动管道。

触发

触发器 是会启动管道的事件。一些触发器（如手动启动管道）可供管道中的所有源操作提供程序使用。某些触发器依赖于管道的源提供程序。例如，必须使用来自 Amazon CloudWatch 的事件资源配置事件 CloudWatch ，这些资源将管道 ARN 添加为事件规则中的目标。建议使用 Amazon Ev CloudWatch ents 触发器，用于自动检测具有 CodeCommit 或 S3 源操作的管道的更改。Webhook 是一种为第三方存储库事件配置的触发器。例如，WebHookv2 是一种触发器类型，它允许使用 Git 标签启动与第三方源提供商（例如 GitHub .com、En GitHub terprise Server、.com、 GitLab 自我管理或 Bitbuc GitLab ket Cloud）的管道。在工作流配置中，您可以为触发器（例如推送或拉取请求）指定过滤器。您可以在 Git 标签、分支或文件路径上筛选代码推送事件。您可以根据事件（已打开、更新、已关闭）、分支或文件路径筛选拉取请求事件。

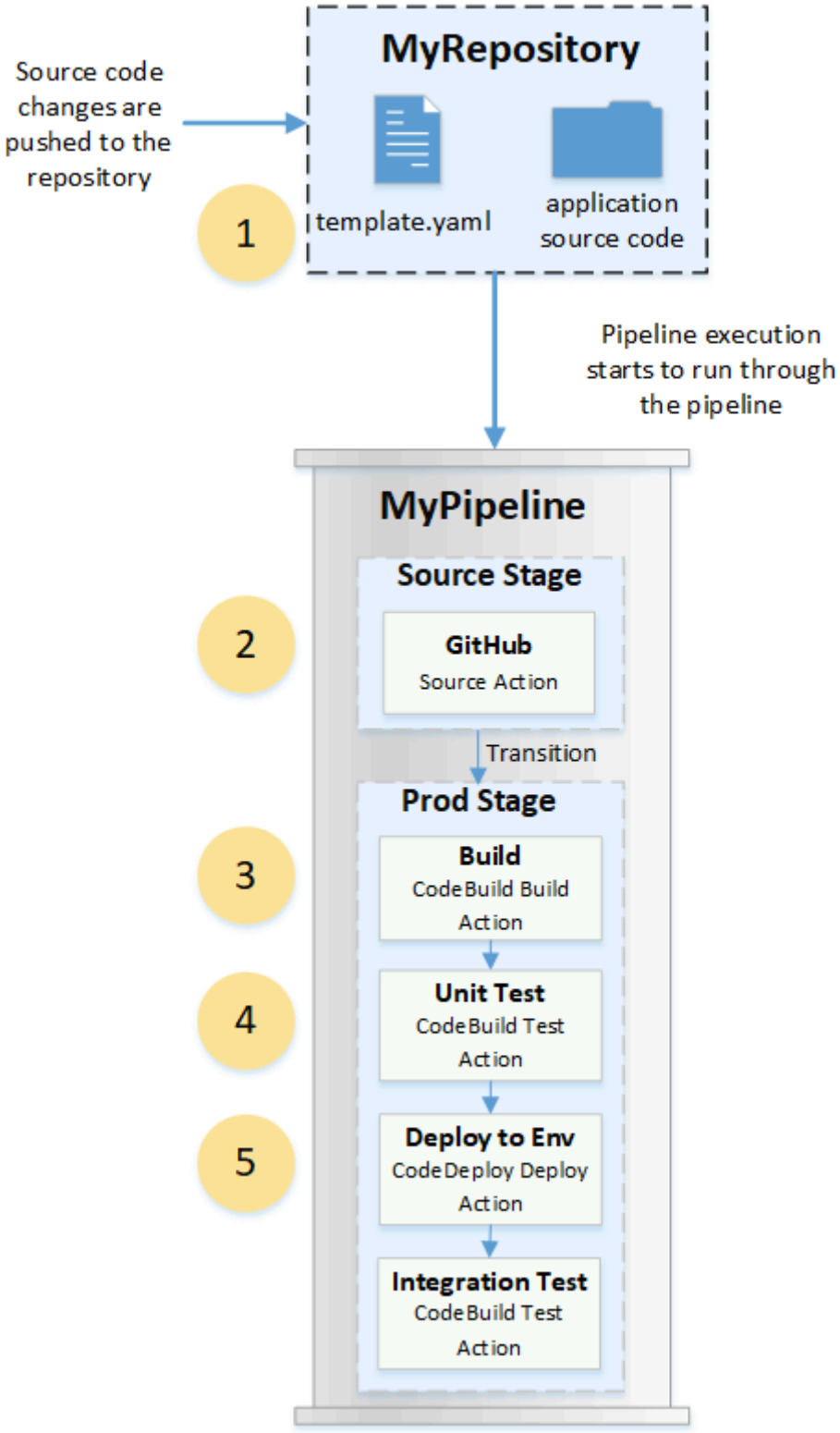
有关触发器的更多信息，请参阅在中启动管道 CodePipeline。有关演示使用 Git 标签作为管道触发器的教程，请参阅教程：使用 Git 标签启动管道。

Variables

变量 是一个值，可用于动态配置管道中的操作。变量可以在管道级别声明，也可以由管道中的操作发出。变量值在管道执行时进行解析，可以在执行历史记录中查看。对于在管道级别声明的变量，可以在管道配置中定义默认值，也可以在给定的执行中覆盖默认值。对于由操作发出的变量，值在操作成功完成后才可用。有关更多信息，请参阅 Variables。

DevOps 管道示例

以管道为例，两阶段 DevOps 管道可能有一个名为 Source 的源阶段和一个名为 Pro d 的第二阶段。在此示例中，管道使用最新更改来更新应用程序，并持续部署最新结果。在部署最新的应用程序之前，管道会构建并测试 Web 应用程序。在此示例中，一组开发人员在名为的 GitHub 存储库中设置了基础架构模板和一个 Web 应用程序的源代码 MyRepository。



例如，开发人员将修复推送到 Web 应用程序的索引页面，并将发生以下情况：

1. 应用程序源代码保存在配置为管道中 GitHub 源操作的存储库中。当开发者将提交推送到存储库时，CodePipeline 会检测到推送的更改，然后从源阶段开始执行管道。
2. GitHub 源操作成功完成（也就是说，最新更改已下载并存储到该执行所特有的构件存储桶中）。然后， GitHub 源操作生成的输出构件（即存储库中的应用程序文件）将用作输入工件，供下一阶段的操作处理。
3. 管道执行从源阶段转换到生产阶段。Prod Stag e 中的第一个操作运行在管道中创建 CodeBuild 并配置为生成操作的生成项目。构建任务提取构建环境映像，然后在虚拟容器中构建 Web 应用程序。
4. Prod Stag e的下一个操作是在管道中创建 CodeBuild 并配置为测试操作的单元测试项目。
5. 单元测试的代码接下来由生产阶段中的部署操作处理，该操作将应用程序部署到生产环境。部署操作成功完成后，该阶段的最后一个操作是在管道中创建 CodeBuild 并配置为测试操作的集成测试项目。测试操作调用在 Web 应用程序上安装和运行测试工具（如链接检查器）的 shell 脚本。成功完成后，输出是一个构建 Web 应用程序和一组测试结果。

开发人员可以向管道添加操作，以便在构建并针对每个更改测试应用程序后，对其进行部署或进一步测试。

有关更多信息，请参阅 管道执行的工作原理。

管道执行的工作原理

本节概述了 CodePipeline 处理一组变更的方式。 CodePipeline跟踪手动启动管道或更改源代码时开始的每次管道执行。 CodePipeline 使用以下执行模式来处理每次执行在管道中的进展方式。

- 被取代模式：较新的执行可能会超过较旧的执行。这是默认模式。
- 队列模式：执行按排队的顺序逐一处理。这需要管道类型 V2。
- 并行模式：在并行模式下，执行是同时独立运行的。执行不会等到其他运行完成后才开始或完成。这需要管道类型 V2。

管道执行的启动方式

您可以在更改源代码或手动启动管道时开始执行。您也可以通过您安排的 Amazon Ev CloudWatch ents 规则触发执行。例如，将源代码更改推送到配置为管道源操作的存储库时，管道检测到更改并开始执行。

> **Note**
>
> 如果管道包含多个源操作，将再次运行所有这些操作，即使仅在一个源操作中检测到更改也是如此。

在管道执行中如何处理源代码修订

对于以源代码更改（源代码修订版）开始的每个管道执行，源版本的确定方式如下。

- 对于有 CodeCommit 源的管道，HEAD CodePipeline 将在推送提交的那一刻被克隆。例如，推送提交，这会启动执行 1 的管道。在推送第二个提交的那一刻，这将启动执行管道 2。

  > **Note**
  >
  > 对于处于并行模式且带有 CodeCommit 源的管道，无论触发管道执行的提交如何，源操作都将在启动时克隆 HEAD。有关更多信息，请参阅 CodeCommit 或者并行模式下的 S3 源版本可能与 EventBridge 事件不匹配 。

- 对于具有 S3 源的管道，将使用 S3 存储桶更新 EventBridge 事件。例如，该事件是在源存储桶中更新文件时生成的，这会启动执行管道 1。在第二次存储桶更新事件发生的那一刻，这将启动执行 2 的管道。

  > **Note**
  >
  > 对于处于 PARALLEL 模式且具有 S3 源的管道，无论触发执行的是哪个图像标签，源操作都将始终以最新的图像标签开头。有关更多信息，请参阅 CodeCommit 或者并行模式下的 S3 源版本可能与 EventBridge 事件不匹配 。

- 对于具有连接源的管道，例如连接到 Bitbucket 的管道，HEAD 将在推送提交的那一 CodePipeline 刻被克隆。例如，对于处于 PARALLEL 模式的管道，将推送提交，这将启动管道以执行 1，而第二次管道执行使用第二次提交。

管道执行的停止方式

要使用控制台停止管道执行，您可以在管道可视化页面、执行历史记录页面或详细历史记录页面上选择停止执行。要使用 CLI 停止管道执行，请使用 `stop-pipeline-execution` 命令。有关更多信息，请参阅 在中停止管道执行 CodePipeline。

可以使用两种方法停止管道执行：

- 停止并等待：允许完成所有进行中的操作执行，并且不会启动后续操作。管道执行不会继续到后续阶段。您不能对已处于 Stopping 状态的执行使用此选项。
- 停止并放弃：放弃所有进行中的操作执行并且不完成，不会启动后续操作。管道执行不会继续到后续阶段。您可以对已处于 Stopping 状态的执行使用此选项。

> **Note**
>
> 此选项会导致任务失败或任务次序混乱。

每个选项都会导致管道和操作执行阶段的顺序不同，如下所示。

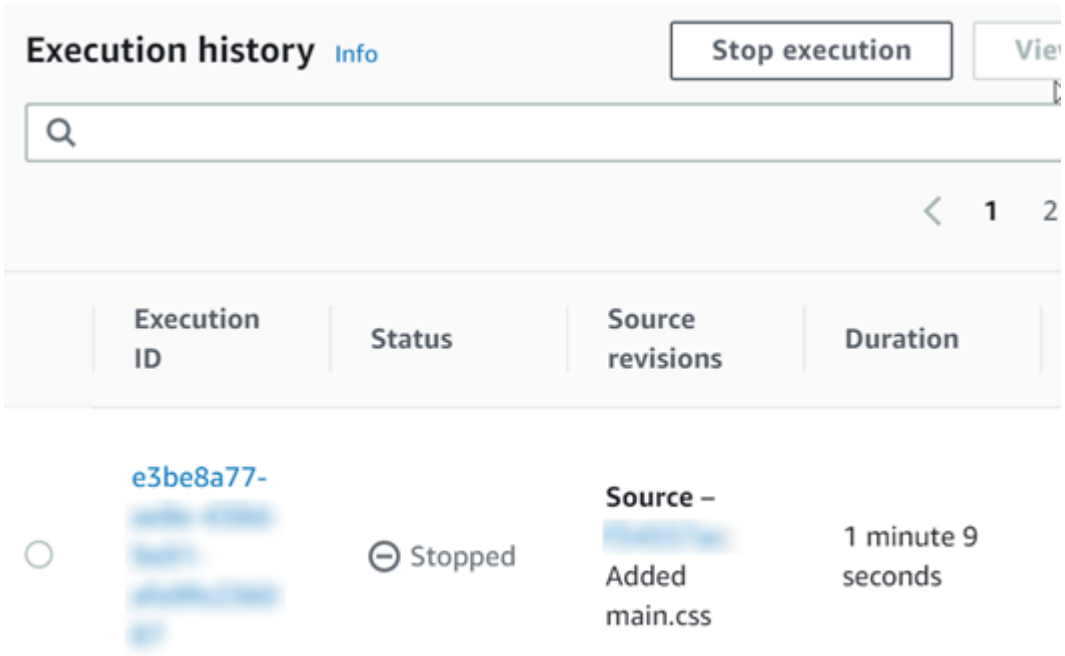
选项 1：停止并等待

当您选择停止并等待时，选定的执行将继续，直到进行中的操作完成。例如，在生成操作正在进行时，以下管道执行会停止。

1. 在管道视图中显示成功消息横幅，并且生成操作会继续，直到完成。管道执行状态为正在停止。

   在历史记录视图中，进行中的操作（例如生成操作）的状态为进行中，直到生成操作完成。操作正在进行时，管道执行状态为正在停止。
2. 停止过程完成后，执行停止。如果已成功完成生成操作，则其状态为已成功，并且管道执行显示已停止状态。后续操作不会启动。重试按钮已启用。

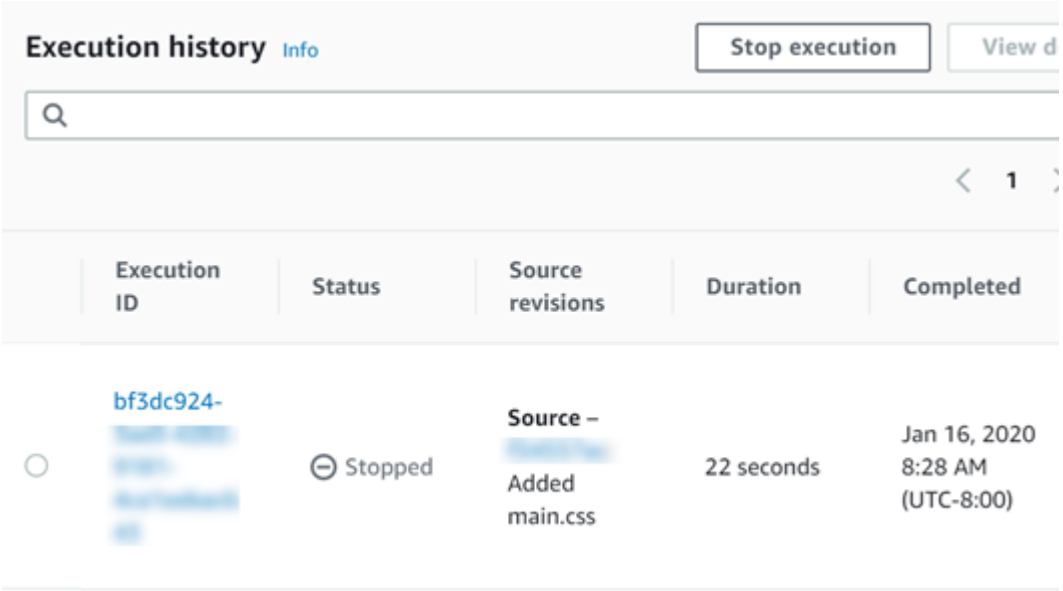
   在历史记录视图中，完成进行中的操作之后，执行状态为已停止。



选项 2：停止并放弃

当您选择停止并放弃时，选定的执行不会等待进行中的操作完成。放弃这些操作。例如，在生成操作进行期间，停止并放弃以下管道执行。

1. 在管道视图中显示成功消息横幅，生成操作显示进行中状态，管道执行显示正在停止状态。
2. 管道执行停止后，生成操作显示已放弃状态，管道执行显示已停止状态。后续操作不会启动。重试按钮已启用。
3. 在历史记录视图中，执行状态为已停止。



停止管道执行的使用案例

我们建议您使用停止并等待选项来停止管道执行。此选项更安全，因为它可以避免管道中可能出现的失败或 out-of-sequence 任务。在中放弃操作后 CodePipeline，操作提供者会继续执行与该操作相关的所有任务。对于 AWS CloudFormation 操作，管道中的部署操作将被放弃，但堆栈更新可能会继续进行并导致更新失败。

举个可能导致 out-of-sequence 任务的放弃操作的示例，如果您通过 S3 部署操作部署一个大文件 (1GB)，并且在部署已经进行时选择停止并放弃该操作，则该操作将在中放弃 CodePipeline，但会在 Amazon S3 中继续执行。Amazon S3 没有收到任何取消上传的指令。接下来，如果您使用非常小的文件启动新的管道执行，则现在有两个部署正在进行。由于新执行的文件很小，因此新部署会在旧部署仍在上传的情况下完成。当旧部署完成后，旧文件会覆盖新文件。

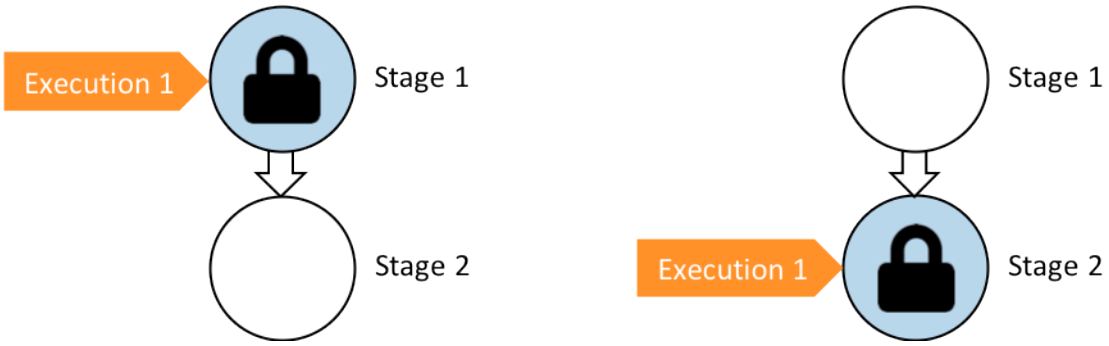
如果有自定义操作，不妨使用停止并放弃选项。例如，如果您有一个自定义操作，其中包含的工作不需要在开始新执行以修复错误之前完成，则可以放弃该操作。

如何在 SUPERSEDED 模式下处理执行

处理执行的默认模式是取代模式。执行包含一组由执行选取并处理的更改。管道可以同时处理多个执行。每个执行都单独在管道中运行。管道按顺序处理每个执行，并且可能使用较后的执行取代较早的执行。以下规则用于在 SUPERSEDED 模式下处理管道中的执行。

规则 1：处理执行时锁定阶段

由于每个阶段一次只能处理一个执行，因此阶段在处理过程中锁定。当执行完成某个阶段时，它会过渡到管道中的下一个阶段。

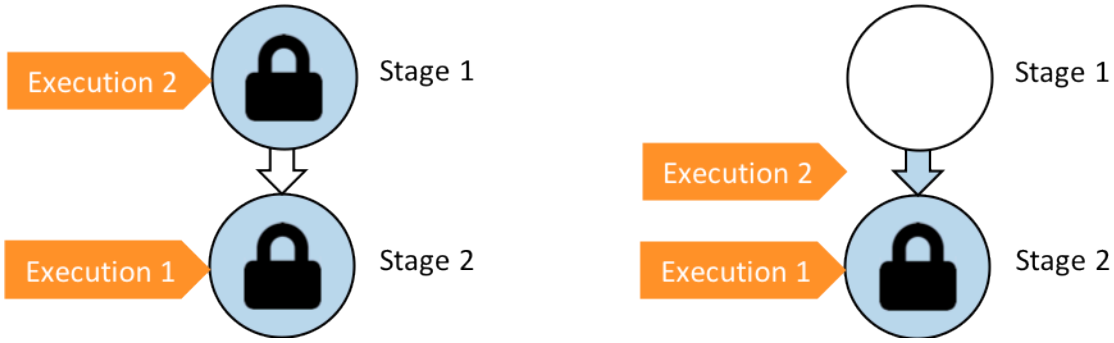


之前：Stage 1 is locked as Execution 1 enters. 之后： Stage 2 is locked as Execution 1 enters.

规则 2：后续执行等待阶段取消锁定

当某个阶段处于锁定状态时，等待中的执行将停留在锁定阶段之前。在阶段被视为完成之前，为该阶段配置的所有操作必须都已成功完成。执行失败将释放阶段锁定。当执行停止时，执行不会在阶段中继续，并且该阶段会解锁。

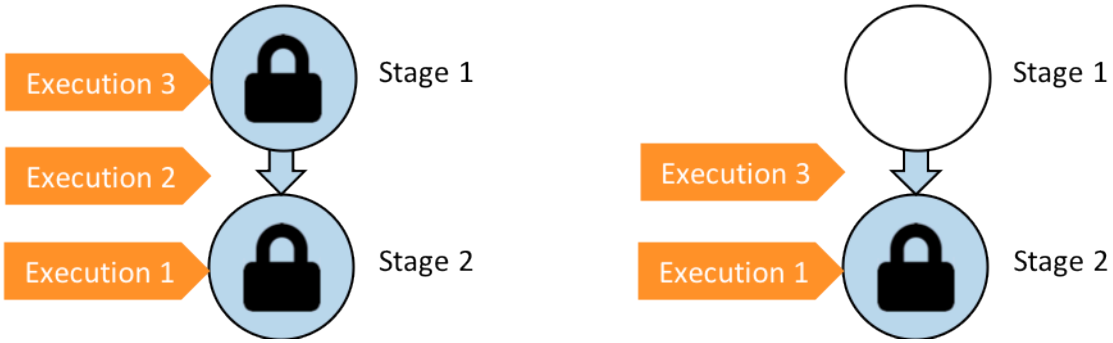
> **Note**
>
> 在停止执行之前，我们建议您禁用阶段前的转换。这样，当阶段由于停止执行而解锁时，阶段不接受后续的管道执行。



之前： Stage 2 is locked as Execution 1 enters. 之后： Execution 2 exits Stage 1 and waits between stages.

规则 3：更新的执行将取代等待中的执行

只会取代处于阶段之间的执行。一个锁定的阶段在该阶段之前挂起一个执行，等待该阶段完成。更新的执行会取代等待中的执行，并在阶段解锁后立即继续进入下一阶段。被取代的执行不会继续。在此示例中，执行 2 在等待锁定阶段时被执行 3 取代。执行 3 下一步进入阶段。



之前：在执行 3 进入阶段 1 时，执行 2 在阶段之间等待。之后：执行 3 退出阶段 1。执行 2 被执行 3 取代。

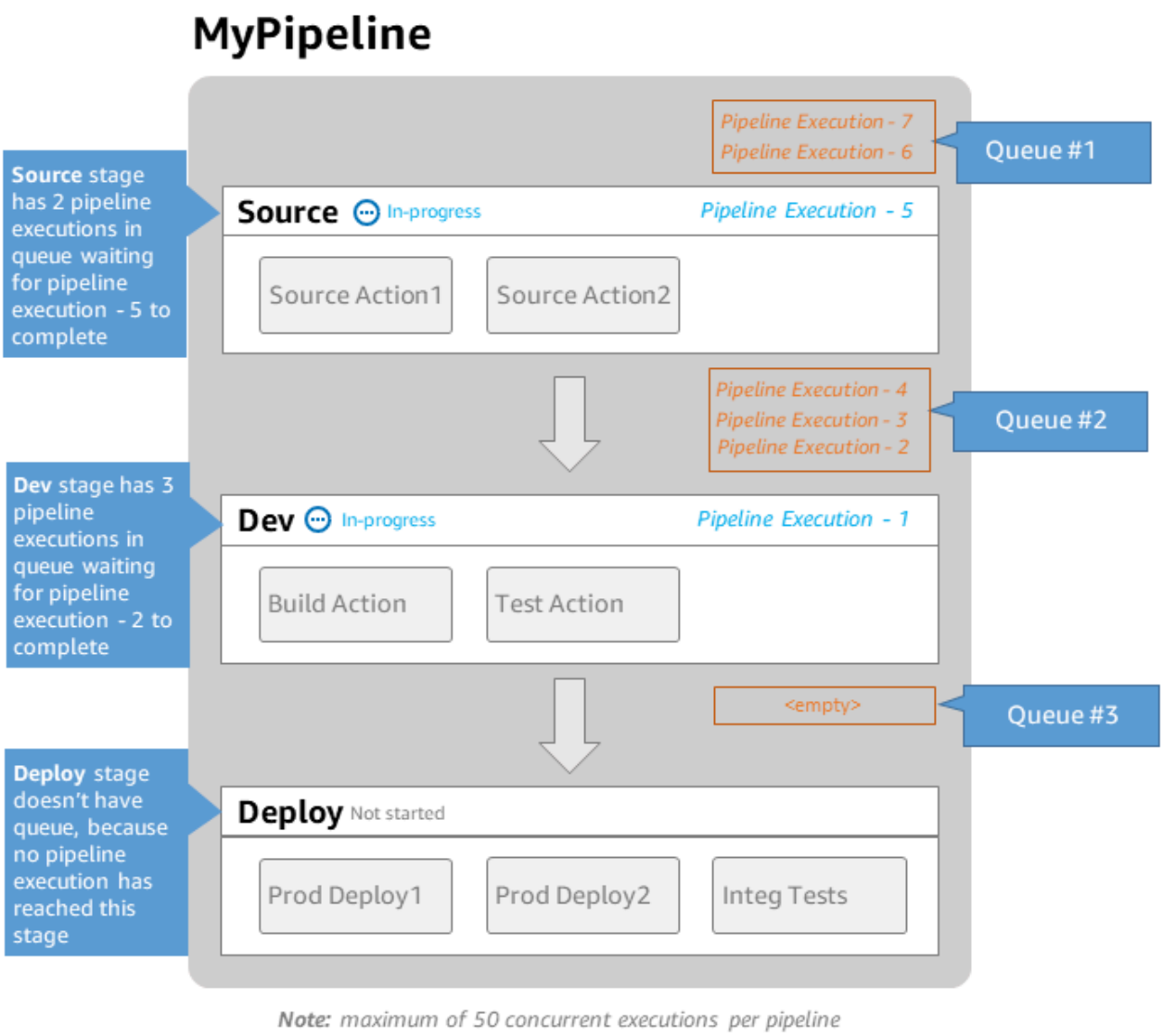
有关查看和切换执行模式的注意事项的更多信息，请参阅设置或更改管道执行模式。有关使用执行模式的配额的更多信息，请参阅中的配额 AWS CodePipeline。

如何在排队模式下处理执行

对于处于 QUEUED 模式的管道，处理执行时阶段会被锁定；但是，等待的执行不会超过已经开始的执行。

等待的执行按到达阶段的顺序聚集在锁定阶段的入口点，形成等待的执行队列。在 QUEUED 模式下，您可以在同一个管道中拥有多个队列。当排队的执行进入某个阶段时，该阶段将被锁定，其他执行都无法进入。此行为与 SUPERSEDED 模式相同。执行完成该阶段后，该阶段将处于解锁状态，为下一次执行做好了准备。

下图显示了排队模式管道中的各个阶段是如何处理执行的。例如，当源阶段处理执行 5 时，6 和 7 的执行形成队列 #1 并在阶段入口点等待。队列中的下一次执行将在阶段解锁后处理。



有关查看和切换执行模式的注意事项的更多信息，请参阅设置或更改管道执行模式。有关使用执行模式的配额的更多信息，请参阅中的配额 AWS CodePipeline。

如何在并行模式下处理执行

对于处于 PARALLEL 模式的管道，执行相互独立，无需等待其他执行完成后再开始。没有队列。要查看管道中的并行执行，请使用执行历史视图。

在开发环境中使用并行模式，在这种环境中，每个功能都有自己的功能分支，并部署到其他用户未共享的目标。

有关查看和切换执行模式的注意事项的更多信息，请参阅设置或更改管道执行模式。有关使用执行模式的配额的更多信息，请参阅中的配额 AWS CodePipeline。

管理管道流

管道执行流程可以由以下环节控制：

- 一个过渡，用于控制执行进入阶段的流程。可以启用或禁用过渡。禁用过渡后，管道执行将无法进入阶段。等待进入禁用了过渡的阶段的管道执行称为入站执行。启用过渡之后，入站执行将进入阶段并将其锁定。

  与等待锁定阶段的执行类似，在禁用过渡时，等待进入阶段的执行仍可被新执行取代。重新启用已禁用过渡时，最新的执行以及在禁用过渡时已被取代的较早执行将进入阶段。
- 审批操作 可防止在授予权限之前管道过渡到下一个操作（例如，授权身份的手动批准）。例如，如果您希望控制管道过渡到最终生产阶段的时间，则可以使用审批操作。

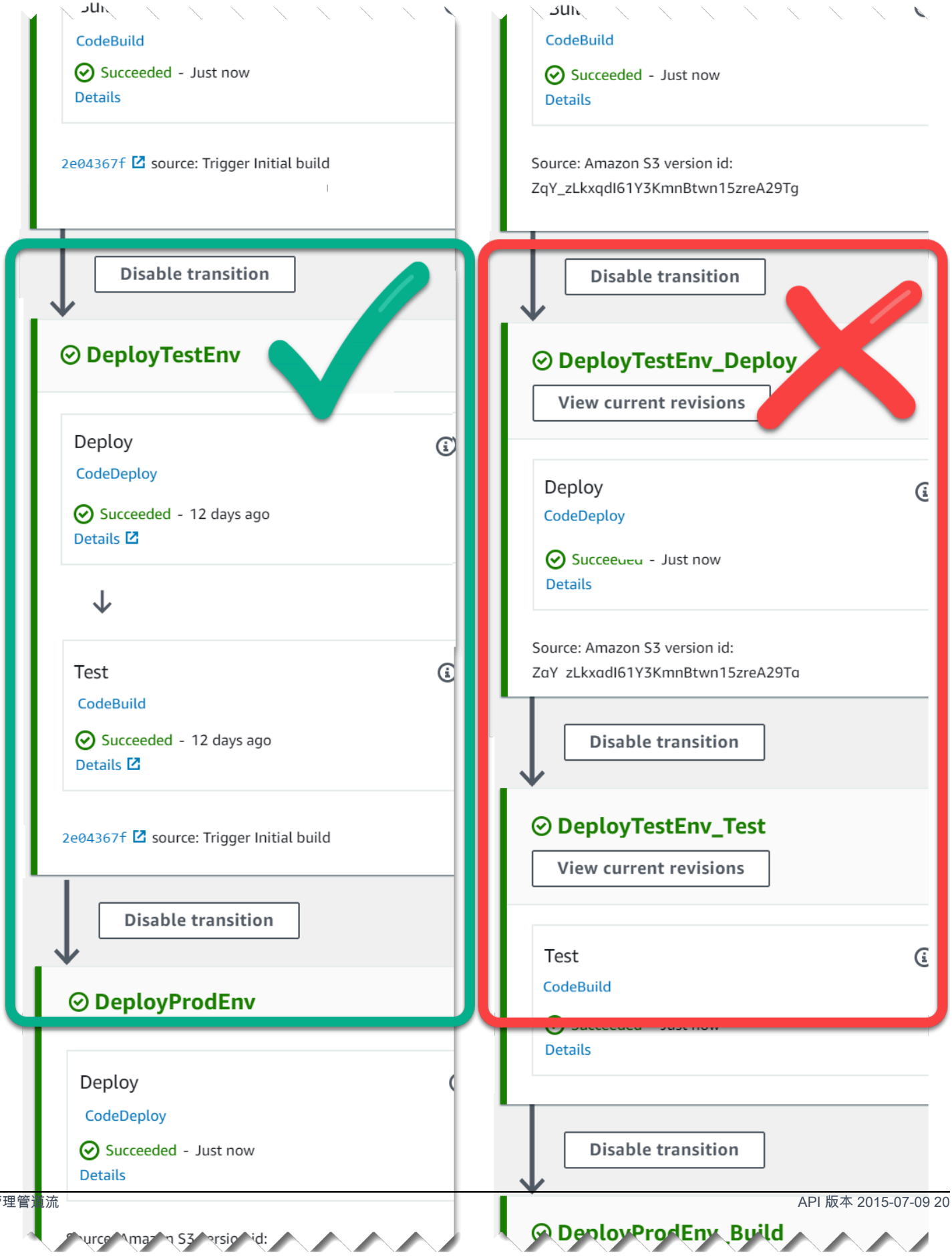
  > **Note**
  >
  > 具有审批操作的阶段被锁定，直到审批操作获得批准、被拒绝或者超时。超时审批操作的处理方式与失败操作相同。

- 当阶段中的操作未成功完成时即失败。修订不会过渡到该阶段的下一个操作或管道中的下一阶段。可能会发生以下情况：
  - 您手动重试包含失败操作的阶段。这将恢复执行（它重试失败的操作，如果操作成功，将在阶段/管道中继续）。
  - 另一个执行进入失败阶段并取代失败的执行。此时，无法重试失败的执行。

推荐的管道结构

在决定代码更改应如何通过您的管道流程时，最好对一个阶段中的相关操作分组，以便在阶段锁定时，操作均会处理相同的执行。您可以为每个应用程序环境或可用区等创建一个阶段。 AWS 区域具有太多阶段（也就是说太细致）的管道可能会允许过多的并发更改，而在一个较大阶段中具有许多操作（太粗糙）的管道可能需要太长时间才能发布更改。

例如，在同一阶段执行部署操作后的测试操作可以确保测试已部署的相同更改。在此示例中，更改部署到测试环境并进行测试，然后将来自测试环境的最新更改部署到生产环境。在推荐的示例中，测试环境和生产环境是独立的阶段。


CodeBuild
Succeeded - Just now
Details
2e04367f source: Trigger Initial build
Disable transition
DeployTestEnv
Deploy
CodeDeploy
Succeeded - 12 days ago
Details
Test
CodeBuild
Succeeded - 12 days ago
Details
2e04367f source: Trigger Initial build
Disable transition
DeployProdEnv
Deploy
CodeDeploy
Succeeded - Just now
Details
CodeBuild
Succeeded - Just now
Details
Source: Amazon S3 version id:
ZqY_zLkxqdI61Y3KmnBtwn15zreA29Tg
Disable transition
DeployTestEnv_Deploy
View current revisions
Deploy
CodeDeploy
Succeeded - Just now
Details
Source: Amazon S3 version id:
Disable transition
DeployTestEnv_Test
View current revisions
Test
CodeBuild
Details
Disable transition
DeployProdEnv_Build

左：相关测试、部署和批准操作分组在一起（推荐）。右：相关操作位于独立的阶段中（不推荐）。

入站执行的工作原理

入站执行是一种在向前移动之前等待不可用的阶段、过渡或操作变为可用状态的执行。下一阶段、过渡或操作可能不可用，原因是：

- 另一执行已进入下一阶段并将其锁定。
- 进入下一阶段的过渡已禁用。

如果您想控制当前执行是否有时间在后续阶段完成，或者您想在某个时间点停止所有操作，则可以禁用过渡以保持入站执行。要确定是否有入站执行，可以在控制台中查看管道或查看 get-pipeline-state 命令的输出。

入站执行需要考虑以下注意事项：

- 一旦操作、过渡或锁定阶段可用，正在进行的入站执行就会进入该阶段并继续通过管道。
- 在入站执行等待时，可以手动将其停止。入站执行可以具有 `InProgress`、`Stopped` 或 `Failed` 状态。
- 当入站执行已停止或失败时，无法重试，因为没有失败的操作可供重试。当入站执行已停止并启用过渡时，停止的入站执行不会继续到该阶段。

您可以查看或停止入站执行。

输入和输出构件

CodePipeline 与开发工具集成以检查代码更改，然后在持续交付过程的所有阶段进行构建和部署。构件是指通过管道中的操作处理的文件，例如包含应用程序代码的文件或文件夹、索引页文件、脚本等。例如，Amazon S3 源操作项目是一个文件名（或文件路径），其中为管道源操作提供了应用程序源代码文件，这些文件通常以 ZIP 文件形式提供，例如以下示例项目名称： SampleApp_Windows.zip。源操作的输出构件，即应用程序源代码文件，是源操作的输出构件，也是下一操作（例如构建操作）的输入构件。又例如，构建操作可能会运行构建命令以编译输入构件（源操作中的应用程序源代码文件）的应用程序源代码。有关构件参数（例如操作）的详细信息，请参阅特定操作AWS CodeBuild的 CodeBuild 操作配置参考页面。

操作使用存储在您在创建管道时选择的 Amazon S3 项目存储桶中的输入和输出项目。 CodePipeline 根据舞台中的操作类型，压缩和传输输入或输出工件的文件。

> **Note**
>
> 构件桶与用作管道源文件位置的桶不同，其中所选源操作为 S3。

例如：

1. CodePipeline 在提交源存储库时触发管道运行，提供源阶段的输出工件（任何要构建的文件）。
2. 上一步骤中的输出构件（要构建的任意文件）将作为构建阶段的输入构件。构建阶段的输出构件（构建应用程序）可能是为容器构建的更新的应用程序或更新的 Docker 镜像。
3. 上一步骤中的输出构件（构建应用程序）将作为部署阶段的输入构件，例如， AWS Cloud中的暂存或生产环境。您可以将应用程序部署到部署队列中，也可以将基于容器的应用程序部署到在 ECS 集群中运行的任务。

创建或编辑操作时，您可以为操作指定输入和输出构件。例如，对于具有源和部署阶段的两阶段管道，在编辑操作中，您可以为部署操作的输入构件选择源操作的构件名称。

- 当您使用控制台创建第一个管道时， CodePipeline 会在同一个管道中创建一个 Amazon S3 存储桶 AWS 区域 ， AWS 账户 并存储所有管道的项目。每次使用控制台在该区域创建另一个管道时，都会在存储桶中为该管道 CodePipeline 创建一个文件夹。在自动化发布过程运行时，它将使用该文件夹来存储管道的项目。*###### codepipeline-region-12345 EXAMPLE### AWS ########## ##12345 EXAMPLE ### 12 ##################*

  > **Note**
  >
  > 如果您在创建管道的区域中已经有一个以 codepipeline-region 开头的存储桶，CodePipeline 请将其用作默认存储桶。它还遵循字母表顺序；例如 codepipeline-region-abcexample 先于 codepipeline-region-defexample 选择。

  CodePipeline 截断对象名称，这可能会导致某些存储桶名称看起来相似。尽管工件名称似乎已被截断，但 CodePipeline 映射到工件存储桶的方式不会受到名称被截断的工件的影响。管道可以正常工作。这不是文件夹或构件的问题。管道名称不能超过 100 个字符。尽管构件文件夹名称似乎已缩短，但它对您的管道来说仍然是唯一的。

创建或编辑管道时，管道中必须有一个工件存储桶 AWS 区域， AWS 账户 并且您计划在其中执行操作的每个区域都必须有一个构件存储桶。如果您使用控制台创建管道或跨区域操作，则默认对象存储桶由 CodePipeline 在您有操作的区域中进行配置。

如果您使用创建管道，则可以 AWS CLI 将该管道的项目存储在任何 Amazon S3 存储桶中，前提是该存储桶与管道位于相同 AWS 账户 且 AWS 区域 与管道相同。如果您担心超出您的账户允许的 Amazon S3 桶限制，则可以这么做。如果您使用创建或编辑管道，并且添加了跨区域操作（ AWS 提供商位于与您的管道不同的区域的操作），则必须为计划执行操作的每个其他区域提供一个构件存储桶。 AWS CLI

- 每个操作都有一个类型。根据类型，操作可能具有以下一项或两项：
  - 输入项目，它是操作在运行过程中使用或处理的项目。
  - 输出构件，它是操作的输出。

  管道中的每个输出构件必须具有唯一的名称。操作的每个输入项目必须与管道中的以前操作的输出构件匹配，无论该操作直接位于某个阶段中的操作前面，还是相差几个阶段。

  项目可以由多个操作处理。

管道类型

CodePipeline 提供了以下管道类型，这些类型在特性和价格上有所不同，因此您可以根据应用程序的需求定制管道功能和成本。

- V1 类型的管道具有 JSON 结构，其中包含标准的管道、阶段和操作级参数。
- V2 类型管道与 V1 类型管道结构相同，还有用于发布安全性和触发器配置的其他参数。

有关定价的信息 CodePipeline，请参阅定价。

有关每种管道类型中参数的详细信息，请参阅CodePipeline 管道结构参考页面。有关选择哪种管道类型的信息，请参阅哪种类型的管道适合我？。

哪种类型的管道适合我？

管道类型由每个管道版本支持的一组特点与特征决定。

下面是每种管道类型的使用案例和特征摘要。

	V1 类型	V2 类型
特性		
使用案例	• 标准部署	• 通过在运行时传递管道级变量进行配置的部署 • 将管道配置为在 Git 标签上启动的部署
操作级变量	支持	支持
并行执行模式	不支持	支持
管道级变量	不支持	支持
排队执行模式	不支持	支持
流水线阶段的回滚	不支持	支持
源修订覆盖	不支持	支持
触发和筛选 Git 标签、拉取请求、分支或文件路径	不支持	支持

有关定价的信息 CodePipeline，请参阅定价。

在 V1 类型管道上使用以下脚本来分析将管道移至 V2 类型管道的成本。

运行管道类型的成本分析（脚本）

1. 打开终端窗口。运行以下命令创建一个名为 PipelineCostAnalyzer.py 的新 python 脚本。

```
vi PipelineCostAnalyzer.py
```

2. 将以下代码复制并粘贴到 PipelineCostAnalyzer.py 脚本中。

```
import boto3
import sys
import math
from datetime import datetime, timedelta, timezone
```

```
if len(sys.argv) < 3:
    raise Exception("Please provide region name and pipeline name as arguments.
 Example usage: python PipelineCostAnalyzer.py us-east-1 MyPipeline")
session = boto3.Session(profile_name='default', region_name=sys.argv[1])
pipeline = sys.argv[2]
codepipeline = session.client('codepipeline')

def analyze_cost_in_v2(pipeline_name):
    if codepipeline.get_pipeline(name=pipeline)['pipeline']['pipelineType'] ==
 'V2':
        raise Exception("Provided pipeline is already of type V2.")
    total_action_executions = 0
    total_blling_action_executions = 0
    total_action_execution_minutes = 0
    cost = 0.0
    hasNextToken = True
    nextToken = ""

    while hasNextToken:
        if nextToken=="":
            response =
 codepipeline.list_action_executions(pipelineName=pipeline_name)
        else:
            response =
 codepipeline.list_action_executions(pipelineName=pipeline_name,
 nextToken=nextToken)
        if 'nextToken' in response:
            nextToken = response['nextToken']
        else:
            hasNextToken= False
        for action_execution in response['actionExecutionDetails']:
            start_time = action_execution['startTime']
            end_time = action_execution['lastUpdateTime']
            if (start_time < (datetime.now(timezone.utc) - timedelta(days=30))):
                hasNextToken= False
                continue
            total_action_executions += 1
            if (action_execution['status'] in ['Succeeded', 'Failed', 'Stopped']):
                action_owner = action_execution['input']['actionTypeId']['owner']
                action_category = action_execution['input']['actionTypeId']
['category']
                if (action_owner == 'Custom' or (action_owner == 'AWS' and
 action_category == 'Approval')):
```

```
                    continue

                total_blling_action_executions += 1
                action_execution_minutes = (end_time -
 start_time).total_seconds()/60
                action_execution_cost = math.ceil(action_execution_minutes) * 0.02
                total_action_execution_minutes += action_execution_minutes
                cost = round(cost + action_execution_cost, 2)

    print ("{:<40}".format('Activity in last 30 days:'))
    print ("| {:<40} | {:<10}".format('_____________________________________',
 '____________________'))
    print ("| {:<40} | {:<10}".format('Total action executions:',
 total_action_executions))
    print ("| {:<40} | {:<10}".format('Total billing action executions:',
 total_blling_action_executions))
    print ("| {:<40} | {:<10}".format('Total billing action execution minutes:',
 round(total_action_execution_minutes, 2)))
    print ("| {:<40} | {:<10}".format('Cost of moving to V2 in $:', cost - 1))

analyze_cost_in_v2(pipeline)
```

3. 根据给定 AWS 区域的 V1 管道运行脚本。

   运行以下命令运行名为 PipelineCostAnalyzer.py 的 python 脚本。在此示例中，区域为 us-west-2。

```
python3 PipelineCostAnalyzer.py us-west-2
```

4. 在脚本的以下示例输出中，我们可以看到操作执行列表、符合计费条件的操作执行列表、这些操作执行的总运行时间，最后是将管道更新为 V2 类型的费用。

```
Activity in last 30 days:
 | _____________________________________      | ____________________
 | Total action executions:                   | 9
 | Total billing action executions:           | 9
 | Total billing action execution minutes:    | 5.59
 | Cost of moving to V2 in $:                 | -0.76
```

> **Note**
>
> 在此示例中，最后一行的负值表示移至 V2 类型的管道所要节省的金额。

入门 CodePipeline

如果您不熟悉 CodePipeline，则可以在按照本章中的步骤进行设置后，按照本指南中的教程进行设置。

CodePipeline 控制台在可折叠面板中包含有用的信息，您可以通过页面上的信息图标或任何 “信息” 链接打开该面板。

()。

您可以随时关闭此面板。

CodePipeline 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。

在 AWS CodePipeline 首次使用之前，必须先创建 AWS 账户 并创建您的第一个管理用户。

主题

步骤 1：创建 AWS 账户 管理员用户

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

报名参加 AWS 账户

1. 打开 https://portal.aws.amazon.com/billing/signup。
2. 按照屏幕上的说明进行操作。

   在注册时，将接到一通电话，要求使用电话键盘输入一个验证码。

当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为安全最佳实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后会向您发送一封确认电子邮件。在任何时候，您都可以通过转至 https://aws.amazon.com/ 并选择我的账户来查看当前的账户活动并管理您的账户。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。AWS Management Console在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的以根用户身份登录。
2. 为您的根用户启用多重身份验证 (MFA)。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的启用 AWS IAM Identity Center。
2. 在 IAM Identity Center 中，为用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的创建权限集。
2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的添加组。

步骤 2：应用管理访问的托管策略 CodePipeline

您必须授予与之交互的权限 CodePipeline。为此，最快的方法是对管理用户应用 `AWSCodePipeline_FullAccess` 托管式策略。

> **Note**
>
> 该`AWSCodePipeline_FullAccess`策略包括允许控制台用户将 IAM 角色传递给 CodePipeline 或其他角色的权限 AWS 服务。这样可允许服务代入此角色并代表您执行操作。将该策略附加到用户、角色或组时，将应用 `iam:PassRole` 权限。确保该策略仅应用于受信任的用户。当具有这些权限的用户使用控制台创建或编辑管道时，以下选项可供使用：
>
> - 创建 CodePipeline 服务角色或选择现有角色并将该角色传递给 CodePipeline
> - 可以选择创建用于变更检测 CloudWatch 的事件规则并将 CloudWatch 事件服务角色传递给 Events CloudWatch
>
> 有关更多信息，请参阅向用户授予将角色传递给的权限 AWS 服务。

> **Note**
>
> 该`AWSCodePipeline_FullAccess`策略允许访问 IAM 用户有权访问的所有 CodePipeline 操作和资源，以及在管道中创建阶段时可能执行的所有操作，例如创建包括 CodeDeploy Elastic Beanstalk 或 Amazon S3 的阶段。作为最佳实践，您仅应向个人授予他们履行职责所需的权

限。有关如何限制 IAM 用户只能使用一组有限的 CodePipeline 操作和资源的更多信息，请参阅从 CodePipeline 服务角色删除权限。

要提供访问权限，请为您的用户、组或角色添加权限：

- 中的用户和群组 AWS IAM Identity Center：

  创建权限集合。按照《AWS IAM Identity Center 用户指南》中创建权限集的说明进行操作。

- 通过身份提供商在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中为第三方身份提供商创建角色（联合身份验证）的说明进行操作。

- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

第 3 步：安装 AWS CLI

要从本地开发计算机上的 AWS CLI 调用 CodePipeline 命令，必须安装 AWS CLI。如果您打算仅使用本指南中的 CodePipeline 控制台步骤开始使用此步骤，则此步骤是可选的。

要安装和配置 AWS CLI

1. 在您的本地计算机上，下载并安装 AWS CLI。这将使您能够 CodePipeline 从命令行进行交互。有关更多信息，请参阅使用 AWS 命令行界面进行设置。

   > **Note**
   >
   > CodePipeline 仅适用于 1.7.38 及更高 AWS CLI 版本。要确定您可能已安装 AWS CLI 的版本，请运行该命令aws --version。要将旧版本的升级 AWS CLI 到最新版本，请按照卸载中的说明进行操作 AWS CLI，然后按照安装中的 AWS Command Line Interface说明进行操作。

2. AWS CLI 使用configure命令进行配置，如下所示：

```
aws configure
```

出现提示时，请指定您将与之配合使用的 IAM 用户的访问 AWS 密钥和私有访问密钥 CodePipeline。 AWS 当系统提示您提供默认区域名称时，请指定您将要创建管道的区域，比如 us-east-2。系统提示指定默认输出格式时，指定 json。例如：

```
AWS Access Key ID [None]: Type your target AWS access key ID here, and then press
 Enter
AWS Secret Access Key [None]: Type your target AWS secret access key here, and then
 press Enter
Default region name [None]: Type us-east-2 here, and then press Enter
Default output format [None]: Type json here, and then press Enter
```

> **Note**
>
> 有关 IAM、访问密钥和秘密密钥的更多信息，请参阅管理 IAM 用户的访问密钥和如何获取凭证？。
> 有关可用区域和终端节点的更多信息 CodePipeline，请参阅终AWS CodePipeline 端节点和配额。

第 4 步：打开控制台 CodePipeline

- 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

后续步骤

您已满足先决条件。你可以开始使用 CodePipeline了。要开始使用 CodePipeline，请参阅CodePipeline 教程。

产品和服务与 CodePipeline

默认情况下， AWS CodePipeline 已与许多合作伙伴产品 AWS 服务 和服务集成。使用以下各节中的信息来帮助您进行配置 CodePipeline ，使其与所使用的产品和服务集成。

下列相关资源在您使用此服务的过程中会有所帮助。

主题

与动 CodePipeline 作类型的集成

本主题中的集成信息按 CodePipeline 操作类型组织。

主题

源操作集成

以下信息按 CodePipeline 操作类型组织，可以帮助您进行配置 CodePipeline 以与以下源操作提供程序集成。

主题

- CodeCommit 源操作
- GitHub （版本 1）源操作

Amazon ECR 源操作

Amazon ECR 是一项 AWS Docker 镜像存储库服务。使用 Docker 推送和拉取命令将 Docker 映像上传到您的存储库。Amazon ECR 存储库 URI 和映像在 Amazon ECS 任务定义中用于引用源映像信息。

了解更多：

- 要查看配置参数和 JSON/YAML 示例代码片段，请参阅 Amazon ECR
- 在中创建管道 CodePipeline
- 教程：使用 Amazon ECR 源和 ECS 目标部署创建管道 CodeDeploy

Amazon S3 源操作

Amazon S3 是一种面向互联网的存储服务。您可以通过 Amazon S3 随时在 Web 上的任何位置存储和检索的任意大小的数据。您可以配置 CodePipeline 为使用受版本控制的 Amazon S3 存储桶作为代码的源操作。

> **Note**
>
> Amazon S3 也可以作为部署操作包含在管道中。

了解更多：

- 要查看配置参数和 JSON/YAML 示例代码片段，请参阅 Amazon S3 源操作
- 步骤 1：为您的应用程序创建一个 S3 存储桶
- 创建管道（CLI）
- CodePipeline 使用亚马逊 EventBridge （前身为 Amazon Ev CloudWatch ents）来检测您的 Amazon S3 源存储桶中的更改。请参阅 与以下内容的常规集成 CodePipeline。

与 Bitbucket Cloud GitHub（版本 2）、 GitHub 企业服务器、 GitLab .com 和 GitLab 自我管理的连接

连接（CodeStarSourceConnection操作）用于访问您的第三方 Bitbucket Cloud、 GitHub E GitHub nterprise Server、 GitLab .com 或 GitLab自我管理的存储库。

> **Note**
>
> 此功能不适用于亚太地区（香港）、亚太地区（海得拉巴）、亚太地区（雅加达）、亚太地区（墨尔本）、亚太地区（墨尔本）、亚太地区（大阪）、非洲（开普敦）、中东（巴林）、中东（阿联酋）、欧洲（西班牙）、欧洲（苏黎世）、以色列（特拉维夫）或 AWS GovCloud（美国西部）地区。要参考其他可用操作，请参阅 产品和服务与 CodePipeline。有关在欧洲地区（米兰）区域使用此操作的注意事项，请参阅CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、 GitLab .com 和 GitLab 自我管理操作中的说明。

Bitbucket Cloud	您可以配置 CodePipeline 为使用 Bitbucket Cloud 存储库作为代码的来源。您必须事先创建一个 Bitbucket 账户和至少一个 Bitbucket Cloud 存储库。您可以通过创建管道或编辑现有管道，为 Bitbucket Cloud 存储库添加一个源操作。 **Note** 您可以创建到 Bitbucket Cloud 存储库的连接。不支持已安装的 Bitbucket 提供程序类型（如 Bitbucket 服务器）。 您可以设置称为连接 的资源，以允许您的管道访问第三方代码存储库。创建连接时，您需要使用第三方代码存储库安装该 AWS CodeStar 应用程序，然后将其与您的连接关联。 对于 Bitbucket Cloud，请使用控制台中的 Bitbucket 选项或 CLI 中的 CodestarSourceConnection 操作。请参阅 Bitbucket Cloud 连接。 您可以对此操作使用完整克隆选项以引用存储库 Git 元数据，以便下游操作可以直接执行 Git 命令。此选项只能由 CodeBuild 下游操作使用。 了解更多：

	• 要查看配置参数和 JSON/YAML 示例代码片段，请参阅 CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、GitLab .com 和 GitLab 自我管理操作。 • 要查看使用 Bitbucket Cloud 源创建管道的“开始使用”教程，请参阅开始使用连接。
GitHub 或 GitHub 企业云	您可以配置 CodePipeline 为使用 GitHub 存储库作为代码的源。您之前必须创建过一个 GitHub 账户和至少一个 GitHub存储库。您可以通过创建管道或编辑现有管道来为 GitHub 仓库添加源操作。 您可以设置称为连接 的资源，以允许您的管道访问第三方代码存储库。创建连接时，您需要使用第三方代码存储库安装该 AWS CodeStar 应用程序，然后将其与您的连接关联。 使用控制台中的 GitHub （版本 2）提供程序选项或 CLI 中的`CodestarSourceConnection` 操作。请参阅 GitHub 连接。 您可以对此操作使用完整克隆选项以引用存储库 Git 元数据，以便下游操作可以直接执行 Git 命令。此选项只能由 CodeBuild 下游操作使用。 了解更多： • 要查看配置参数和 JSON/YAML 示例代码片段，请参阅 CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、GitLab .com 和 GitLab 自我管理操作 • 有关向您展示如何连接到 GitHub 存储库和使用 “完全克隆” 选项的教程，请参阅教程：使用带有 GitHub 管道源的完整克隆。 • 当前 GitHub 操作是的版本 2 源操作 GitHub。版本 1 GitHub 操作通过 OAuth 令牌身份验证进行管理。虽然我们不建议使用 GitHub 版本 1 操作，但具有 GitHub 版本 1 操作的现有管道将继续运行而不会产生任何影响。现在，您可以在管道中使用CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、 GitLab .com 和 GitLab 自我管理操作源操作来管理 GitHub 应用程序的 GitHub 源操作。如果您的管道使用版本 1 GitHub 操作，请参阅中更新该管道以使用版本 2 GitHub 操作的步骤将 GitHub 版本 1 的源操作更新为 GitHub 版本 2 的源操作。

GitHub 企业服务器	您可以配置 CodePipeline 为使用 GitHub 企业服务器存储库作为代码的源。您之前必须创建过一个 GitHub 账户和至少一个 GitHub 存储库。您可以通过创建管道或编辑现有管道来为 E GitHub nterprise Server 存储库添加源操作。 您可以设置称为连接 的资源，以允许您的管道访问第三方代码存储库。创建连接时，您需要使用第三方代码存储库安装该 AWS CodeStar 应用程序，然后将其与您的连接关联。 使用控制台中的GitHub 企业服务器提供者选项或 CLI 中的`CodestarSourceConnection` 操作。请参阅 GitHub 企业服务器连接。 **Important** AWS CodeStar 由于版本中存在已知问题，Connections 不支持 GitHub 企业服务器版本 2.22.0。要执行连接，请升级到版本 2.22.1 或最新的可用版本。 您可以对此操作使用完整克隆选项以引用存储库 Git 元数据，以便下游操作可以直接执行 Git 命令。此选项只能由 CodeBuild 下游操作使用。 了解更多： • 要查看配置参数和 JSON/YAML 示例代码片段，请参阅 CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、GitLab .com 和 GitLab 自我管理操作 • 有关向您展示如何连接到 GitHub 存储库和使用 “完全克隆” 选项的教程，请参阅教程：使用带有 GitHub 管道源的完整克隆。
GitLab.com	您可以配置 CodePipeline 为使用 GitLab .com 存储库作为代码的源。您之前必须创建过 GitLab .com 账户和至少一个 GitLab .com 存储库。您可以通过创建管道或编辑现有管道来为 GitLab .com 存储库添加源操作。 使用控制台中的GitLab提供者选项或在 CLI 中使用`GitLab`提供程序的`CodestarSourceConnection` 操作。请参阅 GitLab.com 连接。 了解更多：

	• 要查看配置参数和 JSON/YAML 示例代码片段，请参阅 CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、GitLab .com 和 GitLab 自我管理操作
GitLab 自我管理	您可以配置 CodePipeline 为使用 GitLab 自行管理的安装作为代码的来源。您之前必须已创建 GitLab 帐户并订阅了自我管理 GitLab （企业版或社区版）。您可以通过创建管道或编辑现有 GitLab 管道来为自行管理的存储库添加源操作。 您可以设置称为连接 的资源，以允许您的管道访问第三方代码存储库。创建连接时，您需要使用第三方代码存储库安装该 AWS CodeStar 应用程序，然后将其与您的连接关联。 使用控制台中的GitLab 自我管理提供商选项或 CLI 中的`CodestarSourceConnection` 操作。请参阅 用于 GitLab 自我管理的连接。 您可以对此操作使用完整克隆选项以引用存储库 Git 元数据，以便下游操作可以直接执行 Git 命令。此选项只能由 CodeBuild 下游操作使用。 了解更多： • 要查看配置参数和 JSON/YAML 示例代码片段，请参阅 CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、GitLab .com 和 GitLab 自我管理操作 • 有关创建与此提供程序类型连接的步骤，请参阅用于 GitLab 自我管理的连接。

CodeCommit 源操作

CodeCommit是一项版本控制服务，可用于在云端私下存储和管理资产（例如文档、源代码和二进制文件）。您可以配置 CodePipeline 为使用 CodeCommit 存储库中的分支作为代码的源。创建存储库并将其与本地计算机上的工作目录关联。然后，您可以在某个阶段的源操作中创建使用分支的管道。您可以通过创建管道或编辑现有管道来连接到 CodeCommit 存储库。

您可以对此操作使用完整克隆选项以引用存储库 Git 元数据，以便下游操作可以直接执行 Git 命令。此选项只能由 CodeBuild 下游操作使用。

了解更多：

- 要查看配置参数和 JSON/YAML 示例代码片段，请参阅 CodeCommit。
- 教程：创建简单的管道（CodeCommit存储库）
- CodePipeline 使用 Amazon Ev CloudWatch ents 来检测用作管道来源的 CodeCommit 存储库中的更改。每个源操作具有相应的事件规则。此事件规则在存储库中发生更改时启动管道。请参阅 与以下内容的常规集成 CodePipeline。

GitHub （版本 1）源操作

GitHub 版本 1 操作由 OAuth 应用程序管理。在可用区域中，您还可以在管道中使用CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、 GitLab .com 和 GitLab 自我管理操作源操作来管理您的 GitHub Apps GitHub 源操作。如果您的管道使用 GitHub 版本 1 操作，请参阅中更新该管道以使用 GitHub 版本 2 操作的步骤将 GitHub 版本 1 的源操作更新为 GitHub 版本 2 的源操作。

> **Note**
>
> 虽然我们不建议使用 GitHub 版本 1 操作，但具有 GitHub 版本 1 操作的现有管道将继续运行而不会产生任何影响。

了解更多：

- 有关基于 OAuth 的 GitHub 访问与基于应用程序 GitHub 的访问对比的更多信息，请参阅。https://docs.github.com/en/developers/apps/differences-between-github-apps-and-oauth-apps
- 要查看包含版本 1 GitHub 操作详细信息的附录，请参阅附录 A： GitHub 版本 1 源代码操作。

生成操作集成

以下信息按 CodePipeline 操作类型组织，可以帮助您进行配置 CodePipeline 以与以下构建操作提供程序集成。

主题

- CodeBuild 生成操作
- CloudBees 生成操作

- Jenkins 构建操作
- TeamCity 生成操作

CodeBuild 生成操作

CodeBuild是一项完全托管的生成服务，它可以编译源代码、运行单元测试并生成可随时部署的工件。

您可以将生成操作 CodeBuild 作为生成操作添加到管道的生成阶段。有关更多信息，请参阅《 CodePipeline 操作配置参考》AWS CodeBuild。

> **Note**
>
> CodeBuild 也可以作为测试操作包含在管道中，带或不带生成输出。

了解更多：

- 要查看配置参数和 JSON/YAML 示例代码片段，请参阅 AWS CodeBuild。
- 什么是 CodeBuild？
- CodeBuild— 完全托管的构建服务

CloudBees 生成操作

您可以配置 CodePipeline CloudBees为使用在管道中的一个或多个操作中生成或测试代码。

了解更多：

- re: Invent 2017：云优先 AWS

Jenkins 构建操作

您可以配置 CodePipeline 为使用 Jenkins CI 在管道中的一个或多个操作中生成或测试代码。您之前必须已创建一个 Jenkins 项目，并为该项目安装和配置了 Jenkins CodePipeline 插件。您可以通过创建新管道或编辑现有管道来连接到 Jenkins 项目。

对 Jenkins 的访问权限应按项目来配置。你必须在你想要使用的每个 Jenkins 实例上安装适用于 Jenkins 的 CodePipeline 插件。 CodePipeline您还必须配置 CodePipeline 对 Jenkins 项目的访问权

限。通过将 Jenkins 项目配置为仅接受 HTTPS/SSL 连接来保护您的 Jenkins 项目。如果您的 Jenkins 项目安装在 Amazon EC2 实例上，请考虑通过 AWS CLI 在每个实例上安装来提供您的 AWS 证书。然后，使用您要用于连接的凭据在这些实例上 AWS 配置配置文件。这是通过 Jenkins Web 界面添加和存储它们的替代方法。

了解更多：

- 访问 Jenkins
- 教程：创建一个四阶段管道

TeamCity 生成操作

您可以配置 CodePipeline TeamCity为使用在管道中的一个或多个操作中生成和测试代码。

了解更多：

- TeamCity 插件用于 CodePipeline

测试操作集成

以下信息按 CodePipeline 操作类型组织，可以帮助您进行配置 CodePipeline 以与以下测试操作提供程序集成。

主题

- CodeBuild 测试动作
- AWS Device Farm 测试动作
- Ghost Inspector 测试操作
- OpenText LoadRunner 云端测试操作

CodeBuild 测试动作

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。

您可以作为测试操作 CodeBuild 添加到管道中。有关更多信息，请参阅《 CodePipeline操作配置参考》AWS CodeBuild。

> **Note**
>
> CodeBuild 也可以作为生成操作包含在管道中，并带有必需的生成输出工件。

了解更多：

- 要查看配置参数和 JSON/YAML 示例代码片段，请参阅 AWS CodeBuild。
- 什么是 CodeBuild？

AWS Device Farm 测试动作

AWS Device Farm 是一项应用程序测试服务，您可以用它在实际物理手机和平板电脑上测试您的 Android、iOS 和 Web 应用程序并与其交互。您可以配置 CodePipeline AWS Device Farm 为用于在管道中的一个或多个操作中测试您的代码。 AWS Device Farm 允许您上传自己的测试或使用内置的、无脚本的兼容性测试。由于测试是并行执行的，因此多个设备上的测试将在几分钟内开始。包含高级结果、低级日志、 pixel-to-pixel 屏幕截图和性能数据的测试报告将在测试完成后更新。 AWS Device Farm 支持测试原生和混合安卓、iOS 和 Fire OS 应用程序，包括使用 Titanium PhoneGap、Xamarin、Unity 和其他框架创建的应用程序。它支持 Android 应用程序的远程访问，使您可以直接与测试设备交互。

了解更多：

- 要查看配置参数和 JSON/YAML 示例代码片段，请参阅 AWS Device Farm。
- 什么是 AWS Device Farm？
- AWS Device Farm 在 CodePipeline 测试阶段使用

Ghost Inspector 测试操作

您可以配置 CodePipeline 为使用 Ghost Inspec tor 在管道中的一个或多个操作中测试您的代码。

了解更多：

- 用于与 Ghost Inspector 集成的服务文档 CodePipeline

OpenText LoadRunner 云端测试操作

您可以配置 CodePipeline 为在管道中的一个或多个操作中使用 OpenText LoadRunner Cloud。

了解更多：

- LoadRunner 用于与集成的云文档 CodePipeline

部署操作集成

以下信息按 CodePipeline 操作类型组织，可以帮助您进行配置 CodePipeline 以与以下部署操作提供程序集成。

主题

- Amazon S3 部署操作
- AWS AppConfig 部署动作
- AWS CloudFormation 部署动作
- AWS CloudFormation StackSets 部署动作
- Amazon ECS 部署操作
- Elastic Beanstalk 部署操作
- AWS OpsWorks 部署动作
- Service Catalog 部署操作
- Amazon Alexa 部署操作
- CodeDeploy 部署动作
- XebiaLabs 部署动作

Amazon S3 部署操作

Amazon S3 是一种面向互联网的存储服务。您可以通过 Amazon S3 随时在 Web 上的任何位置存储和检索的任意大小的数据。您可以向使用 Amazon S3 作为部署提供程序的管道添加操作。

> **Note**
>
> Amazon S3 也可以作为源操作包含在管道中。

了解更多：

- 在中创建管道 CodePipeline

- 教程：创建以 Amazon S3 作为部署提供程序的管道

AWS AppConfig 部署动作

AWS AppConfig 是一种创建、管理和快速部署应用程序配置的功能。 AWS Systems Manager 您可以 AppConfig 与 EC2 实例上托管的应用程序、容器 AWS Lambda、移动应用程序或 IoT 设备一起使用。

了解更多：

- CodePipeline 的操作配置参考 AWS AppConfig
- 教程：创建 AWS AppConfig 用作部署提供者的管道

AWS CloudFormation 部署动作

AWS CloudFormation为开发人员和系统管理员提供了一种创建和管理相关 AWS 资源集合的简便方法，使用模板来配置和更新这些资源。您可以使用服务的示例模板，也可创建自己的模板。模板描述了运行应用程序所需的 AWS 资源以及任何依赖关系或运行时参数。

AWS 无服务器应用程序模型 (AWS SAM) 扩展 AWS CloudFormation 为定义和部署无服务器应用程序提供了一种简化的方法。 AWS SAM 支持 Amazon API Gateway API、 AWS Lambda 函数和亚马逊 DynamoDB 表。您可以使用 CodePipeline AWS CloudFormation 和 AWS SAM 来持续交付您的无服务器应用程序。

您可以向 AWS CloudFormation 用作部署提供者的管道添加操作。当你 AWS CloudFormation 用作部署提供者时，可以在管道执行过程中对 AWS CloudFormation 堆栈和变更集执行操作。 AWS CloudFormation 可以在管道运行时创建、更新、替换和删除堆栈和更改集。因此，可以根据您在 AWS CloudFormation 模板 AWS 和参数定义中提供的规范，在管道执行期间创建、配置、更新或终止自定义资源。

了解更多：

- CodePipeline 的操作配置参考 AWS CloudFormation
- 使用@@ 持续交付 CodePipeline — 了解如何使用 CodePipeline 为其构建持续交付工作流程 AWS CloudFormation。
- 自动部署基于 Lambda 的应用程序 — 了解如何使用 AWS 无服务器应用程序模型以及 AWS CloudFormation 如何为基于 Lambda 的应用程序构建持续交付工作流程。

AWS CloudFormation StackSets 部署动作

AWS CloudFormation为您提供了一种跨多个账户和 AWS 地区部署资源的方式。

您可以 AWS CloudFormation 将 CodePipeline 与配合使用来更新您的堆栈集定义并将更新部署到您的实例。

您可以向管道中添加以下操作以 AWS CloudFormation StackSets 用作部署提供者。

- CloudFormationStackSet
- CloudFormationStackInstances

了解更多：

- CodePipeline 的操作配置参考 AWS CloudFormation StackSets
- 教程：使用 AWS CloudFormation StackSets 部署操作创建管道

Amazon ECS 部署操作

Amazon ECS 是一项可高度扩展的高性能容器管理服务，使您可以在 AWS Cloud中运行基于容器的应用程序。在创建管道时，您可以选择 Amazon ECS 作为部署提供方。源控制存储库中的代码发生变更后，将会触发管道构建新的 Docker 映像，将其推送到您的容器注册表，然后将更新后的映像部署到 Amazon ECS 中。您还可以使用中的 ECS（蓝/绿）提供商操作将流量路由和部署 CodePipeline 到 Amazon ECS。 CodeDeploy

了解更多：

- 什么是 Amazon ECS？
- 教程：使用持续部署 CodePipeline
- 在中创建管道 CodePipeline
- 教程：使用 Amazon ECR 源和 ECS 目标部署创建管道 CodeDeploy

Elastic Beanstalk 部署操作

Elastic Beanstalk 是一项服务，用于在熟悉的服务器（例如 Apache、Nginx、Passenger 和 IIS）上部署和扩展使用 Java、.NET、PHP、Node.js、Python、Ruby、Go 和 Docker 开发的 Web 应用程

序和服务。您可以配置 CodePipeline 为使用 Elastic Beanstalk 来部署您的代码。您可以创建 Elastic Beanstalk 应用程序和环境，以便在某个阶段的部署操作中使用，这可以发生在您创建管道之前，或者在您使用创建管道向导时。

> **Note**
>
> 此功能在亚太地区（海得拉巴）、亚太地区（墨尔本）、中东（阿联酋）、欧洲（西班牙）或欧洲（苏黎世）区域不可用。要参考其他可用操作，请参阅 产品和服务与 CodePipeline。

了解更多：

- 开始使用 Elastic Beanstalk
- 在中创建管道 CodePipeline

AWS OpsWorks 部署动作

AWS OpsWorks 是一项配置管理服务，可帮助您使用 Chef 配置和操作各种形状和大小的应用程序。使用 AWS OpsWorks Stacks，您可以定义应用程序的架构和每个组件的规格，包括软件包安装、软件配置和存储等资源。您可以配置 CodePipeline 为用于 AWS OpsWorks Stacks 将您的代码与自定义 Chef 食谱和应用程序一起部署。 AWS OpsWorks

- 自定义 Chef Cookbook AWS OpsWorks s — 使用 Chef Cookbooks 来处理诸如安装和配置软件包以及部署应用程序之类的任务。
- 应用程序- AWS OpsWorks 应用程序由要在应用程序服务器上运行的代码组成。应用程序代码存储在存储库中，如 Amazon S3 桶。

在创建管道之前，您需要创建 AWS OpsWorks 堆栈和层。您可以在创建管道之前或使用 “创建管道” 向导时创建要在部署操作中使用的 AWS OpsWorks 应用程序。

CodePipeline 目前 AWS OpsWorks 仅在美国东部（弗吉尼亚北部）区域 (us-east-1) 提供支持。

了解更多：

- CodePipeline 与一起使用 AWS OpsWorks Stacks
- 说明书和诀窍
- AWS OpsWorks 应用程序

Service Catalog 部署操作

S@@ ervice Catalog 使组织能够创建和管理获准在上 AWS使用的产品目录。

> **Note**
>
> 此功能在亚太地区（海得拉巴）、亚太地区（雅加达）、亚太地区（墨尔本）、亚太地区（大阪）、中东（阿联酋）、欧洲（西班牙）、欧洲（苏黎世）或以色列（特拉维夫）区域不可用。要参考其他可用操作，请参阅 产品和服务与 CodePipeline。

您可以配置为 CodePipeline 将产品模板的更新和版本部署到 Service Catalog。您可以创建要在部署操作中使用的 Service Catalog 产品，然后使用创建管道向导创建管道。

了解更多：

- 教程：创建部署到 Service Catalog 的管道
- 在中创建管道 CodePipeline

Amazon Alexa 部署操作

Amazon Alexa Skills Kit 可让您构建基于云的技能并将其分发给启用 Alexa 的设备的用户。

> **Note**
>
> 此功能在亚太地区（香港）或欧洲地区（米兰）区域不可用。要使用该区域可用的其他部署操作，请参阅部署操作集成。

您可以向管道中添加使用 Alexa Skills Kit 作为部署提供方的操作。您的管道检测到源更改，然后您的管道在 Alexa 服务中部署 Alexa 技能的更新。

了解更多：

- 教程：创建部署 Amazon Alexa 技能的管道

CodeDeploy 部署动作

CodeDeploy协调对 Amazon EC2 实例、本地实例或两者的应用程序部署。您可以配置 CodePipeline 为使用 CodeDeploy 来部署您的代码。您可以在创建管道之前或使用 “创建管道” 向导时创建要用于部署操作的 CodeDeploy 应用程序、部署和部署组。

了解更多：

- 步骤 3：在中创建应用程序 CodeDeploy
- 教程：创建简单的管道（CodeCommit存储库）

XebiaLabs 部署动作

您可以配置 CodePipeline XebiaLabs为使用在管道中的一个或多个操作中部署代码。

了解更多：

- 将 XL 部署与配合使用 CodePipeline

审批操作与 Amazon Simple Notification Service 集成

Amazon SNS 是一项快速、灵活、完全托管的推送通知服务，可让您将单个消息或多个消息发送给大量收件人。Amazon SNS 提供了一种简单且经济高效的方式，以便向移动设备用户、电子邮件收件人发送推送通知，甚至向其他分布式服务发送消息。

当您在中创建手动批准请求时 CodePipeline，您可以选择在 Amazon SNS 中向某个主题发布消息，这样订阅该请求的所有 IAM 用户都会收到通知，该批准操作已准备就绪，可供审核。

了解更多：

- 什么是 Amazon SNS？
- 向服务角色授予 Amazon SNS 权限 CodePipeline

调用操作集成

以下信息按 CodePipeline 操作类型组织，可以帮助您配置 CodePipeline 为与以下调用操作提供程序集成。

主题

- Lambda 调用操作
- Snyk 调用操作
- Step Functions 调用操作

Lambda 调用操作

利用 Lambda，您可以在不预调配或管理服务器的情况下运行代码。您可以配置 CodePipeline 为使用 Lambda 函数来增加管道的灵活性和功能。您可以创建 Lambda 函数，以便在某个阶段的部署操作中使用，这可以发生在您创建管道之前，或者在您使用创建管道向导时。

了解更多：

- CodePipeline 的操作配置参考 AWS Lambda
- 在中调用管道中的 AWS Lambda 函数 CodePipeline

Snyk 调用操作

您可以配置 CodePipeline 为使用 Snyk，通过检测和修复安全漏洞以及更新应用程序代码和容器镜像中的依赖关系来保护开源环境的安全。您还可以使用中的 Snyk 操作 CodePipeline 来自动执行管道中的安全测试控制。

了解更多：

- CodePipeline 的操作配置参考 Snyk 操作结构参考
- 使用 Snyk 自动 AWS CodePipeline 进行漏洞扫描

Step Functions 调用操作

Step Functions 使您可以创建和配置状态机。您可以配置 CodePipeline 为使用 Step Functions 调用操作来触发状态机执行。

了解更多：

- CodePipeline 的操作配置参考 AWS Step Functions
- 教程：在管道中使用 AWS Step Functions 调用操作

与以下内容的常规集成 CodePipeline

以下 AWS 服务 集成不是基于 CodePipeline 操作类型的。

Amazon CloudWatch	亚马逊 CloudWatch监控您的 AWS 资源。 了解更多： • 什么是亚马逊 CloudWatch？
Amazon EventBridge	Amazon EventBridge 是一项 Web 服务，它 AWS 服务 根据您定义的规则检测您的更改，并在发生更改 AWS 服务 时调用一个或多个指定的操作。 • 发生变化时自动开始管道执行 — 您可以在 Amazon 中设置的规则中配置 CodePipeline 为目标 EventBridge。这会将管道设置为在其他服务发生更改时自动启动。 了解更多： • 什么是亚马逊 EventBridge？ • 在中启动管道 CodePipeline. • CodeCommit 源操作和 EventBridge • 当管道状态发生变化时接收通知-您可以设置 EventBridge 规则来检测管道、阶段或操作的执行状态变化并做出反应。 了解更多： • 监视 CodePipeline 事件 • 教程：设置 CloudWatch 事件规则以接收有关管道状态更改的电子邮件通知
AWS Cloud9	AWS Cloud9 是一个在线 IDE，您可以通过 Web 浏览器访问它。该 IDE 提供丰富的代码编辑体验，对多种编程语言和运行时调试程序的支持以及内置终端。在后台，Amazon EC2 实例托管 AWS Cloud9 开发环境。有关更多信息，请参阅《AWS Cloud9 用户指南》。 了解更多： • 设置 AWS Cloud9

AWS CloudTrail	CloudTrail捕获账户或代表 AWS 账户进行的 AWS API 调用和相关事件，并将日志文件传输到您指定的 Amazon S3 存储桶。您可以配置 CloudTrail 为捕获来自 CodePipeline控制台的 API 调用 AWS CLI、来自和 CodePipeline API 的 CodePipeline 命令。 了解更多： • 使用记录 CodePipeline API 调用 AWS CloudTrail
AWS CodeStar 通知	您可以设置通知 以使用户了解重要更改，例如管道开始执行的时间。有关更多信息，请参阅 创建通知规则。

AWS Key Management Service	AWS KMS 是一项托管服务，可让您轻松创建和控制加密您的数据所用的加密密钥。默认情况下， CodePipeline 用于 AWS KMS 加密存储在 Amazon S3 存储桶中的管道的项目。 了解更多： • 要创建使用来自一个账户的源存储桶、项目存储桶和服务角色以及来自不同 AWS AWS 账户的 CodeDeploy 资源的管道，您必须创建客户管理的 KMS 密钥，将密钥添加到管道中，并设置账户策略和角色以启用跨账户访问。有关更多信息，请参阅 在中 CodePipeline 创建使用其他 AWS 账户资源的管道。 • 要从一个 AWS 账户创建将 AWS CloudFormation 堆栈部署到另一个 AWS 账户的管道，您必须创建客户管理的 KMS 密钥，将密钥添加到管道中，并设置账户策略和角色以将堆栈部署到另一个 AWS 账户。有关更多信息，请参阅 CodePipeline 如何使用在其他账户中部署 AWS CloudFormation 堆栈？ • 要为管道的 S3 工件存储桶配置服务器端加密，您可以使用默认的 AWS 托管 KMS 密钥或创建客户管理的 KMS 密钥并将存储桶策略设置为使用加密密钥。有关更多信息，请参阅 为存储在 Amazon S3 中的项目配置服务器端加密 CodePipeline。 对于 AWS KMS key，您可以使用密钥 ID、密钥 ARN 或别名 ARN。 **Note** 别名只能在创建 KMS 密钥的账户中识别。对于跨账户操作，您只能使用密钥 ID 或密钥 ARN 来标识密钥。跨账户操作涉及使用其他账户（AccountB）的角色，因此指定密钥 ID 将使用其他账户（AccountB）的密钥。

来自社区的示例

以下各部分提供的链接指向博客文章、文章和社区提供的示例。

> **Note**
>
> 这些链接仅供参考，不应被视为全面的清单或对示例内容的认可。 AWS 对外部内容的内容或准确性概不负责。

主题

集成示例：博客文章

- 从第三方 Git 仓库跟踪 AWS CodePipeline 构建状态

  了解如何设置将在第三方存储库中显示您的管道和构建操作状态的资源，使开发人员无需切换上下文即可轻松跟踪状态。

  发布时间：2021 年 3 月

- 使用、、和 AWS CodeCommit， AWS CodeBuild完成 CI/CD AWS CodeDeployAWS CodePipeline

  了解如何设置管道，使用 CodeCommit CodePipeline CodeBuild、和 CodeDeploy 服务在一组 Amazon EC2 Linux 实例上编译、构建和安装受版本控制的 Java 应用程序。

  发布时间：2020 年 9 月

- 如何使用从部署 GitHub 到 Amazon EC2 CodePipeline

  了解如何 CodePipeline 从头开始设置以将开发、测试和生产分支部署到不同的部署组。了解如何使用和配置 IAM 角色、 CodeDeploy 代理和 CodeDeploy以及 CodePipeline。

  发布时间：2020 年 4 月

- 为 AWS Step Functions 测试和创建 CI/CD 管道

  了解如何设置将会协调 Step Functions 状态机和管道的资源。

  发布时间：2020 年 3 月

- 实现 DevSecOps 使用 CodePipeline

了解如何使用 CI/CD 管道实现预防和侦查安全控制的自动化。 CodePipeline 这篇文章介绍了如何使用管道创建简单的安全组，并在源代码、测试和生产阶段进行安全检查，以改善 AWS 账户的安全状况。

发布时间：2017 年 3 月

- 使用 CodePipeline、、 CodeBuild Amazon ECR 和 Amazon ECR 持续部署到 Amazon ECS AWS CloudFormation

  了解如何创建到 Amazon Elastic Container Service (Amazon ECS) 的持续部署管道。应用程序使用 CodePipeline、、Amazon ECR 和 CodeBuild，作为 Docker 容器交付。 AWS CloudFormation

  - 从 ECS 参考架构：持续部署存储库中下载示例 AWS CloudFormation 模板和使用该模板创建自己的持续部署管道的说明。 GitHub

发布时间：2017 年 1 月

- 无服务器应用程序的持续部署

  了解如何使用集合 AWS 服务 为您的无服务器应用程序创建持续部署管道。您将使用无服务器应用程序模型 (SAM) 来定义应用程序及其资源， CodePipeline 并协调应用程序部署。

  - 查看示例应用程序，该应用程序使用 Go with the Gin 框架和 API Gateway 代理填充码编写。

发布时间：2016 年 12 月

- 使用 CodePipeline 和 Dynatrace 扩展 DevOps 部署

  了解如何使用 Dynatrace 监控解决方案扩展管道 CodePipeline、在提交代码之前自动分析测试执行以及保持最佳交付时间。

发布时间：2016 年 11 月

- 在 Using AWS Elastic Beanstalk 中 CodePipeline 创建流水线 AWS CloudFormation 和 CodeCommit

  了解如何在 CodePipeline 管道中为应用程序实现持续交付 AWS Elastic Beanstalk。所有 AWS 资源都是通过使用 AWS CloudFormation 模板自动配置的。本演练还包含 CodeCommit 和 AWS Identity and Access Management (IAM)。

发布时间：2016 年 5 月

- 自动化 CodeCommit 和 CodePipeline 进入 AWS CloudFormation

用于 AWS CloudFormation 为使用 CodeCommit、 CodePipeline、和的持续交付管道自动配置 AWS 资源 AWS Identity and Access Management。 CodeDeploy

发布时间：2016 年 4 月

- 在中创建跨账户管道 AWS CodePipeline

  了解如何通过使用 AWS Identity and Access Management自动预配置对 AWS CodePipeline 中的管道的跨账户访问。在 AWS CloudFormation 模板中包含示例。

  发布时间：2016 年 3 月

- 探究 ASP.NET 内核第 2 部分：持续交付

  了解如何使用和为 ASP.NET Core 应用程序创建完整的持续交付系统 CodeDeploy 。 AWS CodePipeline

  发布时间：2016 年 3 月

- 使用 AWS CodePipeline 控制台创建管道

  在基于的演练中学习如何使用 AWS CodePipeline 控制台创建两阶段管道。 AWS CodePipeline 教程：创建一个四阶段管道

  发布时间：2016 年 3 月

- 用它来模拟 AWS CodePipeline 管道 AWS Lambda

  学习如何调用 Lambda 函数，该函数允许您在设计软件交付流程时，在管道开始运行之前可视化 CodePipeline 软件交付过程中的操作和阶段。在设计管道结构时，您可以使用 Lambda 函数来测试管道是否将成功完成。

  发布时间：2016 年 2 月

- 在 usin AWS Lambda g 中 CodePipeline 运行函数 AWS CloudFormation

  了解如何创建一个 AWS CloudFormation 堆栈，以配置用户指南任务中使用的所有 AWS 资源在中调用管道中的 AWS Lambda 函数 CodePipeline。

  发布时间：2016 年 2 月

- 在中配置自定义 CodePipeline 操作 AWS CloudFormation

  在中了解 AWS CloudFormation 如何使用配置自定义操作 CodePipeline。

发布时间：2016 年 1 月

- CodePipeline 使用置备 AWS CloudFormation

  了解如何在 CodePipeline 使用中配置基本的持续交付管道 AWS CloudFormation。

  发布时间：2015 年 12 月

- CodePipeline AWS OpsWorks 使用自定义操作从部署到 AWS Lambda

  了解如何配置您的管道以及要部署到 AWS OpsWorks 的 AWS Lambda 函数 CodePipeline。

  发布时间：2015 年 7 月

CodePipeline 教程

完成中的步骤后入门 CodePipeline，您可以尝试本用户指南中的其中一个 AWS CodePipeline 教程：

我想使用向导创建一个管道，用于将示例应用程序从 Amazon S3 存储桶部署 CodeDeploy 到运行 Amazon Linux 的 Amazon EC2 实例。使用向导创建两阶段管道后，我想再添加一个阶段。	请参阅 教程：创建一个简单的管道（S3 存储桶）。
我想创建一个两阶段管道，用于将示例应用程序从 CodeCommit 存储库部署 CodeDeploy 到运行 Amazon Linux 的 Amazon EC2 实例。	请参阅 教程：创建简单的管道（CodeCommit存储库）。
我想向在第一个教程中创建的三阶段管道添加构建阶段。新的阶段使用 Jenkins 来构建应用程序。	请参阅 教程：创建一个四阶段管道。
我想设置一个 CloudWatch 事件规则，当我的管道、阶段或操作的执行状态发生变化时，该规则就会发送通知。	请参阅 教程：设置 CloudWatch 事件规则以接收有关管道状态更改的电子邮件通知。
我想创建一个管道，其 GitHub 源代码可以用和构建和测试 Android 应用程序 AWS Device Farm。 CodeBuild	请参阅 教程：创建用于构建和测试您的 Android 应用程序的管道 AWS Device Farm。
我想创建一个包含 Amazon S3 源的管道，该管道使用 AWS Device Farm测试 iOS 应用程序。	请参阅 教程：创建用于测试 iOS 应用的管道 AWS Device Farm。
我想创建一个管道，用于将我的产品模板部署到 Service Catalog。	请参阅 教程：创建部署到 Service Catalog 的管道。
我想使用示例模板通过 AWS CloudFormation 控制台创建一个简单的管道（使用 Amazon S3 CodeCommit、或 GitHub 源代码）。	请参阅 教程：使用创建管道 AWS CloudFormation。
我想创建一个两阶段管道，该管道使用 CodeDeploy 和 Amazon ECS，用于将映像从	请参阅 教程：使用 Amazon ECR 源和 ECS 目标部署创建管道 CodeDeploy 。

Amazon ECR 存储库部署到 Amazon ECS 集群和服务的蓝/绿。	
我想创建一个将我的无服务器应用程序持续发布到 AWS Serverless Application Repository的管道。	请参阅 教程：创建将您的无服务器应用程序发布到的管道 AWS Serverless Application Repository。

其他用户指南中的以下教程为将其他内容集成 AWS 服务 到您的管道中提供了指导：

- 创建在《AWS CodeBuild 用户指南》 CodeBuild中使用的管道
- 在《AWS OpsWorks 用户指南》 AWS OpsWorks Stacks中@@ CodePipeline 与一起使用
- 在《AWS CloudFormation 用户指南》 CodePipeline中@@ 进行持续交付
- AWS Elastic Beanstalk 开发者指南 中的开始使用 Elastic Beanstalk
- 使用设置持续部署管道 CodePipeline

教程：使用 Git 标签启动管道

在本教程中，您将创建一个连接到 GitHub 存储库的管道，其中为 Git 标签触发器类型配置了源操作。在提交时创建 Git 标签后，您的管道就会启动。此示例说明如何创建允许根据标签名称语法筛选标签的管道。有关使用 glob 模式筛选的更多信息，请参阅使用语法中的 glob 模式。

本教程 GitHub 通过`CodeStarSourceConnection`操作类型连接到。

> **Note**
>
> 此功能在亚太地区（香港）、非洲（开普敦）、中东（巴林）或欧洲（苏黎世）区域不可用。要参考其他可用操作，请参阅 产品和服务与 CodePipeline。有关在欧洲地区（米兰）区域使用此操作的注意事项，请参阅CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、GitHub 企业服务器、 GitLab .com 和 GitLab 自我管理操作中的说明。

主题

- 先决条件
- 第 1 步：打开 CloudShell 并克隆您的存储库
- 步骤 2：创建在 Git 标签上触发的管道
- 第 3 步：标记要发布的提交

- 步骤 4：发布更改并查看日志

先决条件

在开始之前，您必须执行以下操作：

- 使用您的 GitHub 账户创建 GitHub 存储库。
- 准备好您的 GitHub 凭证。当你使用 AWS Management Console 来建立连接时，系统会要求你使用自己的 GitHub 凭据登录。

第 1 步：打开 CloudShell 并克隆您的存储库

您可以使用命令行界面来克隆存储库、执行提交和添加标签。本教程启动命令行界面的 CloudShell 实例。

1. 登录到 AWS Management Console。
2. 在顶部导航栏中，选择图 AWS 标。此时显示 AWS Management Console 主页。
3. 在顶部导航栏中，选择图 AWS CloudShell 标。 CloudShell 打开。等待 CloudShell 环境创建完成。

   > **Note**
   >
   > 如果您没有看到该 CloudShell 图标，请确保您所在的区域由支持 CloudShell。本教程假设您位于美国西部（俄勒冈州）区域。

4. 在中 GitHub，导航到您的存储库。选择代码，然后选择 HTTPS。复制路径。用于克隆 Git 存储库的地址复制到您的剪贴板。
5. 运行以下命令以克隆存储库。

   ```
   git clone https://github.com/<account>/MyGitHubRepo.git
   ```

6. 在出现提示`Password`时输入您的 GitHub 帐户`Username`。对于 `Password` 条目，您必须使用用户创建的令牌，而不是您的账户密码。

步骤 2：创建在 Git 标签上触发的管道

在此部分中，您将使用以下操作创建管道：

- 一个与您的 GitHub 仓库连接和操作的源阶段。
- 带有生成操作的 AWS CodeBuild 生成阶段。

使用向导创建管道

1. 登录 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。
2. 在欢迎页面、入门页面或管道页面上，选择创建管道。
3. 在步骤 1：选择管道设置的管道名称中，输入 **MyGitHubTagsPipeline**。
4. 在管道类型中，保留默认选择 V2。管道类型有不同的特点和价格。有关更多信息，请参阅 管道类型。
5. 在服务角色中，选择新建服务角色。

> **Note**
>
> 如果您选择使用现有的 CodePipeline 服务角色，请确保已`codestar-connections:UseConnection`将 IAM 权限添加到您的服务角色策略中。有关 CodePipeline 服务角色的说明，请参阅为 CodePipeline 服务角色添加权限。

6. 在高级设置下，保留原定设置值。在构件存储中，选择默认位置以将默认构件存储（如指定为默认值的 Amazon S3 项目存储桶）用于为管道选择的区域中的管道。

> **Note**
>
> 这不是源代码的源存储桶。这是管道的项目存储。每个管道都需要一个单独的构件存储，例如 S3 存储桶。

选择下一步。

7. 在步骤 2：添加源阶段页面上，添加源阶段：
   a. 在源提供程序中，选择 GitHub（版本 2）。
   b. 在连接下，选择一个现有连接或创建一个新连接。要创建或管理 GitHub 源操作的连接，请参阅GitHub 连接。
   c. 在存储库名称中，选择存储 GitHub库的名称。
   d. 在管道触发器下，选择 Git 标签。

在包含字段中输入 `release*`。

在默认分支中，选择在管道手动启动或使用非 Git 标签的源事件时要指定的分支。如果更改的来源不是触发器，或者管道执行是手动启动的，则使用的更改将是来自默认分支的 HEAD 提交。

> **Important**
>
> 使用 Git 标签触发器类型启动的管道配置为使用 WebHookv2 事件，而不会使用 Webhook 事件（对所有推送事件进行更改检测）来启动管道。

选择下一步。

8. 在添加构建阶段，添加一个构建阶段：

   a. 在构建提供程序中，选择 AWS CodeBuild。允许区域默认为管道区域。

   b. 选择创建项目。

   c. 在项目名称中，输入此构建项目的名称。

   d. 在环境映像中，选择托管映像。对于操作系统，选择 Ubuntu。

   e. 对于运行时，选择标准。对于映像，选择 aws/codebuild/standard:5.0。

   f. 对于服务角色，选择新建服务角色。

   > **Note**
   >
   > 记下您的 CodeBuild 服务角色的名称。在本教程的最后一步，您会用到此角色名称。

   g. 在构建规范下，为构建规范选择插入构建命令。选择切换到编辑器，然后将以下内容粘贴到构建命令。

```
version: 0.2
#env:
  #variables:
     # key: "value"
     # key: "value"
  #parameter-store:
     # key: "value"
     # key: "value"
  #git-credential-helper: yes
```

```
phases:
  install:
    #If you use the Ubuntu standard image 2.0 or later, you must specify
 runtime-versions.
    #If you specify runtime-versions and use an image other than Ubuntu
 standard image 2.0, the build fails.
    runtime-versions:
      nodejs: 12
    #commands:
      # - command
      # - command
  #pre_build:
    #commands:
      # - command
      # - command
  build:
    commands:
      -
  #post_build:
    #commands:
      # - command
      # - command
artifacts:
  files:
     - '*'
    # - location
  name: $(date +%Y-%m-%d)
  #discard-paths: yes
  #base-directory: location
#cache:
  #paths:
    # - paths
```

h. 选择 “继续” CodePipeline。这将返回到 CodePipeline控制台并创建一个使用您的构建命令进行配置的 CodeBuild 项目。构建项目使用服务角色来管理 AWS 服务 权限。此步骤可能需要几分钟时间。

i. 选择下一步。

9. 在步骤 4：添加部署阶段页面上，选择跳过部署阶段，并通过再次选择跳过接受警告消息。选择下一步。

10. 在步骤 5：审核中，选择创建管道。

第 3 步：标记要发布的提交

创建管道并指定 Git 标签后，可以在 GitHub仓库中标记提交。在这些步骤中，您将使用 `release-1` 标签标记提交。Git 存储库中的每个提交都必须有唯一的 Git 标签。当您选择提交并对其进行标记时，可将来自不同分支的更改合并到您的管道部署中。请注意，标签名称版本不适用于中版本的概念 GitHub。

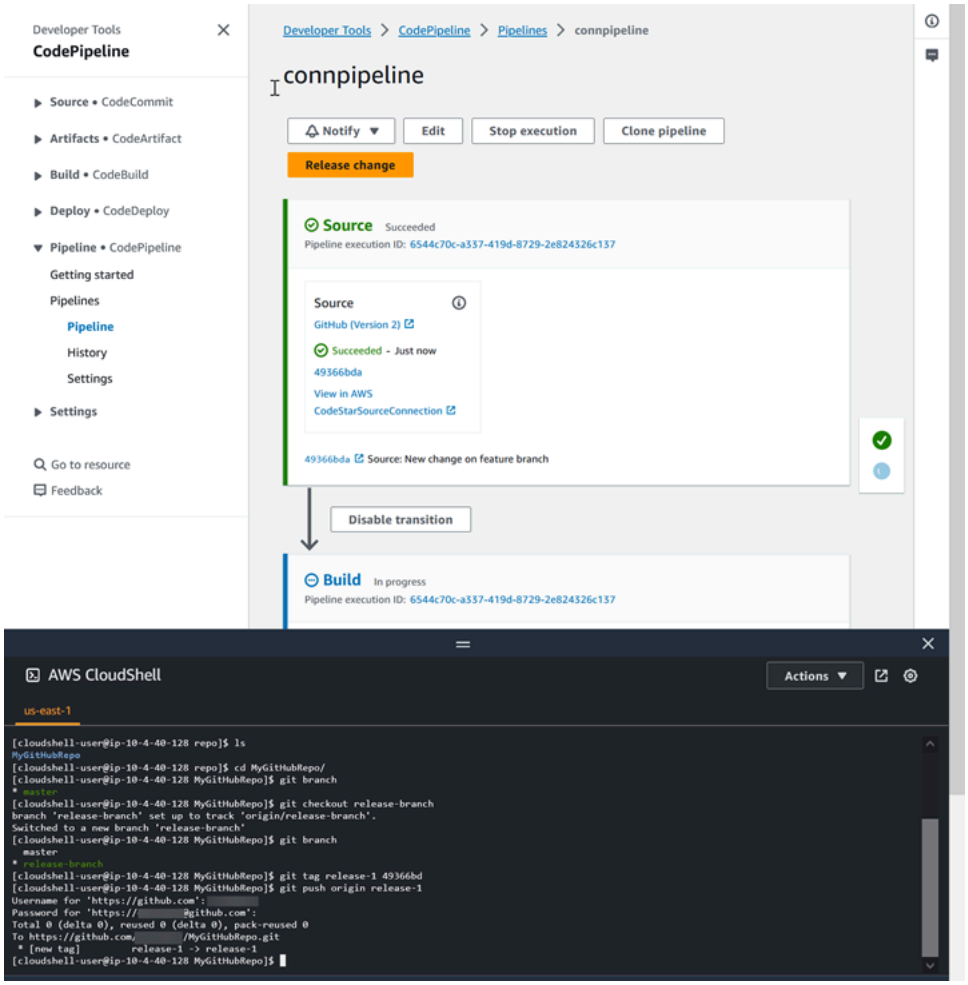
1. 参考您想要标记的复制的提交 ID。要查看每个分支中的提交，请在 CloudShell 终端中输入以下命令以捕获要标记的提交 ID：

   ```
   git log
   ```

2. 在 CloudShell 终端中，输入命令来标记你的提交并将其推送到 Origin。标记提交后，您可以使用 git push 命令将标签推送到源。在以下示例中，输入以下命令将 `release-1` 标签用于 ID 为 49366bd 的第二次提交。此标签将按管道 `release*` 标签筛选条件筛选并启动管道。

   ```
   git tag release-1 49366bd
   ```

   ```
   git push origin release-1
   ```



步骤 4：发布更改并查看日志

1. 管道成功运行后，在成功构建阶段，选择查看日志。

   在“日志”下，查看 CodeBuild 生成输出。这些命令将输出所输入变量的值。

2. 在历史记录页面中，查看触发器列。查看触发器类型 GitTag ：release-1。

教程：筛选拉取请求的分支名称以启动管道

在本教程中，您将创建一个连接到 GitHub .com存储库的管道，在该存储库中，源操作配置为使用筛选拉取请求的触发器配置启动管道。当指定分支发生指定的拉取请求事件时，您的管道就会启动。此示例说明如何创建允许筛选分支名称的管道。有关使用触发器的更多信息，请参阅在管道中触发筛选 JSON (CLI)。有关使用 glob 格式的正则表达式模式进行筛选的更多信息，请参阅。使用语法中的 glob 模式

本教程通过CodeStarSourceConnection操作类型连接到 GitHub .com。

主题

先决条件

在开始之前，您必须执行以下操作：

- 使用您的 GitHub .com 账户创建 GitHub .com 存储库。
- 准备好您的 GitHub 凭证。当你使用 AWS Management Console 来建立连接时，系统会要求你使用自己的 GitHub 凭据登录。

步骤 1：创建管道以启动指定分支的拉取请求

在此部分中，您将使用以下操作创建管道：

- 与您的 GitHub .com存储库的连接和操作的源阶段。
- 带有生成操作的 AWS CodeBuild 生成阶段。

使用向导创建管道

1. 登录 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。
2. 在欢迎页面、入门页面或管道页面上，选择创建管道。
3. 在步骤 1：选择管道设置的管道名称中，输入 **MyFilterBranchesPipeline**。
4. 在管道类型中，保留默认选择 V2。管道类型有不同的特点和价格。有关更多信息，请参阅 管道类型。
5. 在服务角色中，选择新建服务角色。

> **Note**
>
> 如果您选择使用现有的 CodePipeline 服务角色，请确保已`codeconnections:UseConnection`将 IAM 权限添加到您的服务角色策略中。有关 CodePipeline 服务角色的说明，请参阅为 CodePipeline 服务角色添加权限。

6. 在高级设置下，保留原定设置值。在构件存储中，选择默认位置以将默认构件存储（如指定为默认值的 Amazon S3 项目存储桶）用于为管道选择的区域中的管道。

> **Note**
>
> 这不是源代码的源存储桶。这是管道的项目存储。每个管道都需要一个单独的构件存储，例如 S3 存储桶。

选择下一步。

7. 在步骤 2：添加源阶段页面上，添加源阶段：

   a. 在源提供程序中，选择 GitHub （版本 2）。

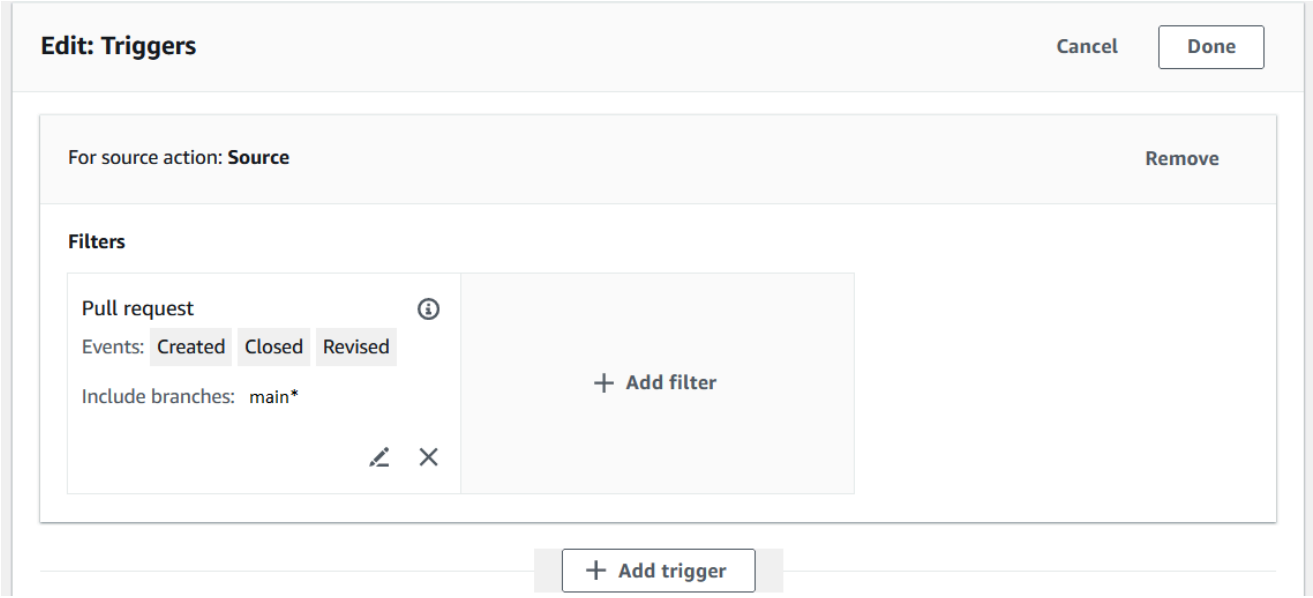
   b. 在连接下，选择一个现有连接或创建一个新连接。要创建或管理 GitHub 源操作的连接，请参阅GitHub 连接。

   c. 在存储库名称中，选择您的 GitHub .com存储库的名称。

   d. 在 “触发器类型” 下，选择 “指定过滤器”。

      在事件类型下，选择拉取请求。选择拉取请求下的所有事件，以便针对已创建、更新或已关闭的拉取请求发生事件。

      在 “分支” 下的 “包含” 字段中输入`main*`。



> **Important**
>
> 将为 WebHookv2 事件配置以此触发器类型开头的管道，并且不会使用 Webhook 事件（对所有推送事件进行更改检测）来启动管道。

选择下一步。

8. 在 “添加构建” 阶段的 “生成提供者” 中，选择AWS CodeBuild。允许区域默认为管道区域。按照中的说明选择或创建构建项目教程：使用 Git 标签启动管道。此操作仅在本教程中用作创建管道所需的第二阶段。
9. 在步骤 4：添加部署阶段页面上，选择跳过部署阶段，并通过再次选择跳过接受警告消息。选择下一步。
10. 在步骤 5：审核中，选择创建管道。

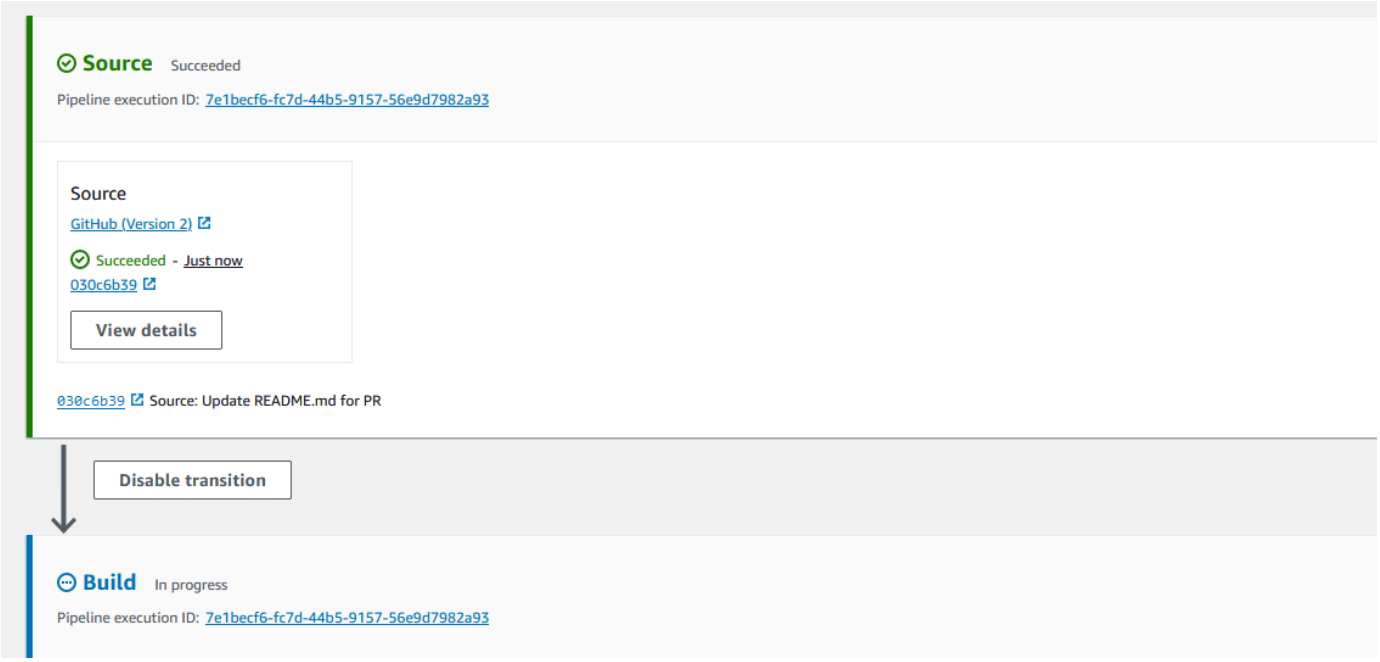
第 2 步：在 GitHub .com 中创建并合并拉取请求以开始执行管道

在本节中，您将创建和合并拉取请求。这将启动您的管道，打开的拉取请求执行一次，已关闭的拉取请求执行一次。

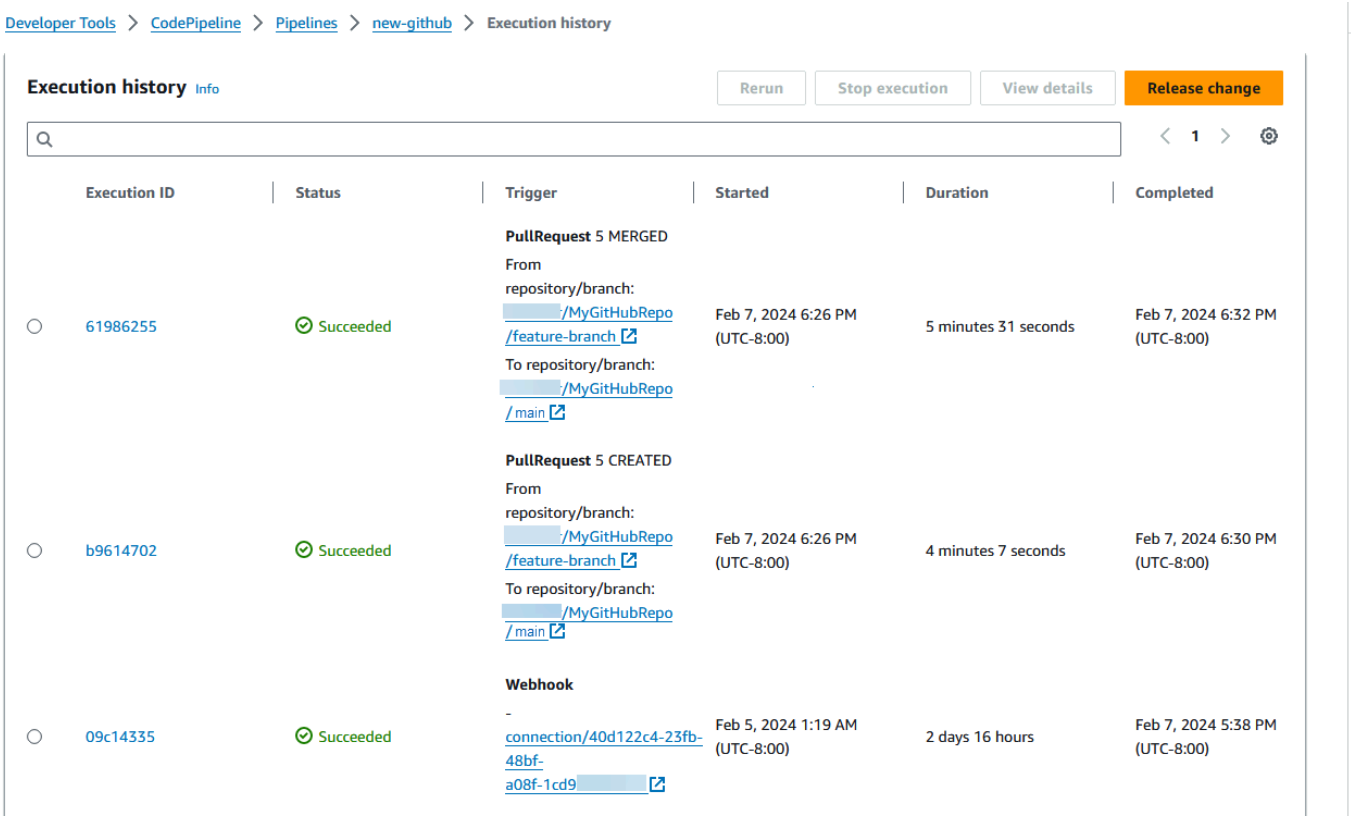
创建拉取请求并启动管道

1. 在 GitHub .com 中，通过更改功能分支上的 README.md 并向该分支提出拉取请求来创建拉取请求。`main`使用类似的消息提交更改`Update README.md for PR`。

2. 管道从源版本开始，将拉取请求的源消息显示为 Update README.m d for PR。



3. 选择 History (历史记录)。在管道执行历史记录中，查看启动管道执行的 CREATED 和 MERGED 拉取请求状态事件。



教程：使用管道级变量

在本教程中，您将创建一个管道，在管道级别添加变量，然后运行输出变量值的 CodeBuild 生成操作。

主题

先决条件

在开始之前，您必须执行以下操作：

- 创建 CodeCommit 存储库。
- 向存储库添加 .txt 文件。

步骤 1：创建管道并构建项目

在此部分中，您将使用以下操作创建管道：

- 与您的 CodeCommit 存储库建立连接的源阶段。
- 带有生成操作的 AWS CodeBuild 生成阶段。

使用向导创建管道

1. 通过 https://console.aws.amazon.com/codepipeline/ 登录主 CodePipeline 机。
2. 在欢迎页面、入门页面或管道页面上，选择创建管道。
3. 在步骤 1：选择管道设置的管道名称中，输入 **MyVariablesPipeline**。
4. 在管道类型中，保留默认选择 V2。管道类型有不同的特点和价格。有关更多信息，请参阅 管道类型。
5. 在服务角色中，选择新建服务角色。

> **Note**
>
> 如果您选择使用现有的 CodePipeline 服务角色，请确保已`codeconnections:UseConnection`将 IAM 权限添加到您的服务角色策略中。有关 CodePipeline 服务角色的说明，请参阅为 CodePipeline 服务角色添加权限。

6. 在变量下，选择添加变量。在名称中，输入 `timeout`。在默认值中，输入 1000。在“描述”中，输入以下描述：**`Timeout`**。

   这将创建一个变量，当管道执行开始时，您可以在其中声明变量值。变量名必须匹配 `[A-Za-z0-9@\-_]+`，可以是除空字符串之外的任何名称。

7. 在高级设置下，保留原定设置值。在构件存储中，选择默认位置以将默认构件存储（如指定为默认值的 Amazon S3 项目存储桶）用于为管道选择的区域中的管道。

   > **Note**
   >
   > 这不是源代码的源存储桶。这是管道的项目存储。每个管道都需要一个单独的构件存储，例如 S3 存储桶。

   选择下一步。

8. 在步骤 2：添加源阶段页面上，添加源阶段：

   a. 在源提供程序中，选择 AWS CodeCommit。

   b. 在存储库名称和分支名称中，选择您的存储库和分支。

   选择下一步。

9. 在添加构建阶段，添加一个构建阶段：

   a. 在构建提供程序中，选择 AWS CodeBuild。允许区域默认为管道区域。

   b. 选择创建项目。

   c. 在项目名称中，输入此构建项目的名称。

   d. 在环境映像中，选择托管映像。对于操作系统，选择 Ubuntu。

   e. 对于运行时，选择标准。对于映像，选择 aws/codebuild/standard:5.0。

   f. 对于服务角色，选择新建服务角色。

> **Note**
>
> 记下您的 CodeBuild 服务角色的名称。在本教程的最后一步，您会用到此角色名称。

g. 在构建规范下，为构建规范选择插入构建命令。选择切换到编辑器，然后将以下内容粘贴到构建命令。在 buildspec 中，客户变量 $CUSTOM_VAR1 将用于在构建日志中输出管道变量。您将在下一步骤中创建 $CUSTOM_VAR1 输出变量作为环境变量。

```
version: 0.2
#env:
  #variables:
     # key: "value"
     # key: "value"
  #parameter-store:
     # key: "value"
     # key: "value"
  #git-credential-helper: yes
phases:
  install:
    #If you use the Ubuntu standard image 2.0 or later, you must specify
 runtime-versions.
    #If you specify runtime-versions and use an image other than Ubuntu
 standard image 2.0, the build fails.
    runtime-versions:
      nodejs: 12
    #commands:
      # - command
      # - command
  #pre_build:
    #commands:
      # - command
      # - command
  build:
    commands:
      - echo $CUSTOM_VAR1
  #post_build:
    #commands:
      # - command
      # - command
artifacts:
  files:
     - '*'
```

```
    # - location
  name: $(date +%Y-%m-%d)
  #discard-paths: yes
  #base-directory: location
#cache:
  #paths:
    # - paths
```

h. 选择 “继续” CodePipeline。这将返回到 CodePipeline控制台并创建一个使用您的构建命令进行配置的 CodeBuild 项目。构建项目使用服务角色来管理 AWS 服务 权限。此步骤可能需要几分钟时间。

i. 在环境变量 - 可选下，要创建环境变量作为将由管道级变量解析的构件操作的输入变量，请选择添加环境变量。这将创建在 buildspec 中指定为 $CUSTOM_VAR1 的变量。在名称中，输入 CUSTOM_VAR1。在值中，输入 #{variables.timeout}。在类型中，选择 Plaintext。

环境变量的 #{variables.timeout} 值基于管道级变量命名空间 variables 和步骤 5 中为管道创建的管道级变量 timeout。

j. 选择下一步。

10. 在步骤 4：添加部署阶段页面上，选择跳过部署阶段，并通过再次选择跳过接受警告消息。选择下一步。
11. 在步骤 5：审核中，选择创建管道。

步骤 2：发布更改并查看日志

1. 管道成功运行后，在成功的构建阶段上，选择查看详细信息。

   在详细信息页面上，选择日志选项卡。查看编 CodeBuild 译输出。这些命令将输出所输入变量的值。

2. 在左侧导航窗格中，选择历史记录。

   选择最近执行，然后选择变量选项卡。查看管道变量的解析值。

教程：创建一个简单的管道（S3 存储桶）

创建管道的最简单方法是在 AWS CodePipeline 控制台中使用创建管道向导。

在本教程中，您将创建一个使用版本控制的 S3 存储桶的两阶段管道 CodeDeploy并发布示例应用程序。

> **Note**
>
> 当 Amazon S3 是您的管道的源提供程序时，您可以将一个或多个源文件压缩到单个 .zip 文件中，然后将 .zip 文件上传到源桶。您也可以上传单个解压缩的文件；但是，需要 .zip 文件的下游操作将失败。

创建此简单管道后，您将另外添加一个阶段，禁用然后再启用阶段之间的过渡。

> **Important**
>
> 在此过程中，您在管道中添加的许多操作都涉及在创建管道之前需要创建的 AWS 资源。 AWS 源操作的资源必须始终在您创建管道的同一 AWS 区域创建。例如，如果您在美国东部（俄亥俄州）地区创建管道，则您的 CodeCommit 存储库必须位于美国东部（俄亥俄州）区域。
> 您可以在创建管道时添加跨区域操作。 AWS 跨区域操作的资源必须位于您计划执行操作的同一 AWS 区域。有关更多信息，请参阅 在中添加跨区域操作 CodePipeline。

在开始之前，您应完成入门 CodePipeline中的先决条件。

主题

步骤 1：为您的应用程序创建一个 S3 存储桶

您可以将源文件或应用程序存储在任何受版本控制的位置。在本教程中，您将为示例应用程序文件创建一个 S3 桶，并对该桶启用版本控制。启用版本控制后，您可将示例应用程序复制到该存储桶。

创建 S3 存储桶

1. 登录控制台，网址为 AWS Management Console。打开 S3 控制台。

2. 请选择创建存储桶。
3. 对于存储桶名称，输入您的存储桶的名称（例如 **awscodepipeline-demobucket-example-date**）。

> **Note**
>
> 由于 Amazon S3 中的所有桶名称必须是唯一的，因此请使用您自己的名称，而不是示例中显示的名称。您可以通过添加日期来更改示例名称。请记下该名称，因为您需要在本教程的其余部分中使用该名称。

在区域中，选择您要在其中创建管道的区域（例如美国西部(俄勒冈州)），然后选择创建桶。

4. 创建存储桶后，系统会显示成功横幅。选择转到存储桶详细信息。
5. 在属性选项卡上，选择版本控制。选择启用版本控制，然后选择保存。

启用版本控制后，Amazon S3 会在桶中存储每个对象的每个版本。

6. 在权限选项卡上，保留默认设置。有关 S3 存储桶和对象权限的更多信息，请参阅在策略中指定权限。
7. 接下来，下载一个示例，并将该示例保存到本地计算机上的文件夹或目录中。

   a. 选择以下选项之一。如果要对 Windows Server 实例执行本教程中的步骤，请选择 `SampleApp_Windows.zip`。

   - 如果您想使用部署到亚马逊 Linux 实例 CodeDeploy，请在此处下载示例应用程序：SampleApp_Linux.zip。
   - 如果你想使用部署到 Windows 服务器实例 CodeDeploy，请在此处下载示例应用程序：SampleApp_Windows.zip。

   示例应用程序包含以下用于部署的文件 CodeDeploy：

   - `appspec.yml`— 应用程序规范文件（AppSpec文件）是一个 YAML 格式的文件，CodeDeploy 用于管理部署。有关该 AppSpec 文件的更多信息，请参阅《AWS CodeDeploy 用户指南》中的CodeDeploy AppSpec 文件参考。
   - `index.html` - 索引文件包含已部署的示例应用程序的主页。
   - `LICENSE.txt` - 许可证文件包含示例应用程序的许可证信息。

- 脚本文件 - 示例应用程序使用脚本将文本文件写入实例上的某个位置。为多个 CodeDeploy 部署生命周期事件分别写入一个文件，如下所示：
  - （仅限 Linux 示例）`scripts` 文件夹 - 该文件夹包含以下 shell 脚本，用于安装依赖项以及启动和停止自动部署的示例应用程序：`install_dependencies`、`start_server` 和 `stop_server`。
  - （仅限 Windows 示例）`before-install.bat` - 此批处理脚本用于 `BeforeInstall` 部署生命周期事件，它将运行以删除此示例在先前部署中写入的旧文件，并在您的实例上创建一个位置用于写入新文件。

b. 下载压缩（zipped）文件。不要解压缩该文件。

8. 在 Amazon S3 控制台中，为您的桶上传文件：

a. 选择上传。

b. 拖放文件或选择添加文件并通过浏览找到该文件。

c. 选择上传。

第 2 步：创建 Amazon EC2 Windows 实例并安装 CodeDeploy 代理

> **Note**
>
> 本教程提供了创建 Amazon EC2 Windows 实例的示例步骤。有关创建 Amazon EC2 Linux 实例的示例步骤，请参阅步骤 3：创建 Amazon EC2 Linux 实例并安装 CodeDeploy 代理。当系统提示您输入要创建的实例数时，请指定 2 个实例。

在此步骤中，您将创建 Windows Server Amazon EC2 实例 以向其部署示例应用程序。作为此过程的一部分，您将创建一个实例角色，其策略允许在实例上安装和管理 CodeDeploy 代理。 CodeDeploy 代理是一个软件包，允许在 CodeDeploy 部署中使用实例。您还可以附加策略，允许实例获取 CodeDeploy 代理用于部署您的应用程序的文件，并允许该实例由 SSM 管理。

创建实例角色

1. 通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在控制台控制面板中，选择角色。
3. 选择创建角色。

4. 在选择受信任实体的类型下，选择 AWS 服务。在选择使用案例下，选择 EC2，然后选择下一步: 权限。
5. 搜索名为 **AmazonEC2RoleforAWSCodeDeploy** 的策略并将其选中。
6. 搜索名为 **AmazonSSMManagedInstanceCore** 的策略并将其选中。选择下一步：标签。
7. 选择下一步：审核。输入角色的名称（例如 **EC2InstanceRole**）。

> **Note**
>
> 记下您的角色名称以便在下一步中使用。在创建实例时选择此角色。

选择 创建角色。

启动实例

1. 通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。
2. 在侧面导航栏中，选择实例，然后从页面顶部选择启动实例。
3. 在名称和标签下，对于名称，输入 **MyCodePipelineDemo**。这会为实例分配标签键 **Name** 和标签值 **MyCodePipelineDemo**。稍后，您将创建一个将示例 CodeDeploy 应用程序部署到实例的应用程序。 CodeDeploy根据标签选择要部署的实例。
4. 在应用程序和操作系统映像(Amazon 机器映像) 下，选择 Windows 选项。（此 AMI 称为 Microsoft Windows Server 2019 Base，标记为“符合免费套餐条件”，可以在快速入门下找到。）
5. 在实例类型下，选择符合免费套餐条件的 `t2.micro` 类型作为实例的硬件配置。
6. 在密钥对(登录) 下，选择或创建密钥对。

   也可以选择继续操作但不提供密钥对。

> **Note**
>
> 在本教程中，您可以继续操作而不提供密钥对。要使用 SSH 连接到实例，请创建或使用密钥对。

7. 在网络设置下，执行以下操作：

   在自动分配公有 IP 中，确保状态为启用。

   - 在分配安全组旁边，选择创建新安全组。

- 在 SSH 对应的行中，在源下选择我的 IP。
- 选择添加安全组，选择 HTTP，然后在源类型下选择我的 IP。

8. 展开高级详细信息。在 IAM 实例配置文件中，选择您在上一过程中创建的 IAM 角色（例如 **EC2InstanceRole**）。
9. 在摘要下的实例数下，输入 2。
10. 选择启动实例。
11. 选择查看所有实例以关闭确认页面并返回控制台。
12. 您可以在实例页面上查看启动的状态。启动实例时，其初始状态为 pending。实例启动后，其状态变为 running，并且会收到一个公有 DNS 名称。（如果公有 DNS 列不显示，请选择显示/隐藏图标，然后选择公有 DNS。）
13. 可能需要花几分钟时间，实例才能准备好让您连接到它。检查您的实例是否通过了状态检查。您可以在状态检查列中查看此信息。

步骤 3：在中创建应用程序 CodeDeploy

在中 CodeDeploy，应用程序是您要部署的代码的标识符，以名称的形式出现。 CodeDeploy 使用此名称来确保在部署期间引用修订版、部署配置和部署组的正确组合。在本教程稍后创建管道时，您可以选择在此步骤中创建的 CodeDeploy 应用程序的名称。

您首先要创建一个 CodeDeploy 要使用的服务角色。如果已创建服务角色，则无需创建其他服务角色。

创建 CodeDeploy 服务角色

1. 通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在控制台控制面板中，选择角色。
3. 选择创建角色。
4. 在选择受信任实体下，选择 AWS 服务。在 Use case（使用案例）下，选择 CodeDeploy。CodeDeploy从列出的选项中进行选择。选择下一步。AWSCodeDeployRole 托管策略已附加到角色。
5. 选择下一步。
6. 输入角色的名称（例如 **CodeDeployRole**），然后选择创建角色。

要在中创建应用程序 CodeDeploy

1. 打开 CodeDeploy 控制台，网址为 https://console.aws.amazon.com/codedeploy。
2. 如果未出现 “应用程序” 页面，请在 AWS CodeDeploy 菜单上选择 “应用程序”。
3. 选择创建应用程序。
4. 在应用程序名称中，输入 `MyDemoApplication`。
5. 在 Compute Platform（计算平台）中，选择 EC2/On-premises（EC2/本地）。
6. 选择创建应用程序。

要在中创建部署组 CodeDeploy

1. 在显示应用程序的页面上，选择创建部署组。
2. 在 Deployment group name（部署组名称）中，输入 **`MyDemoDeploymentGroup`**。
3. 在服务角色中，选择您之前创建的服务角色。您使用的服务角色必须至少 AWS CodeDeploy 具有创建服务角色中所述的信任和权限 CodeDeploy。要获得服务角色 ARN，请参阅获得服务角色 ARN（控制台）。
4. 在部署类型下，选择就地。
5. 在环境配置下，选择 Amazon EC2 实例。在键字段中选择名称，然后在值字段中输入 **`MyCodePipelineDemo`**。

   > **Important**
   >
   > 在这里，您必须为名称键选择您在创建 EC2 实例时为该实例分配的值。如果您使用 **`MyCodePipelineDemo`** 以外的内容标记实例，则在此处，请务必使用该内容。

6. 在 “使用 S AWS ystems Manager 配置代理” 下，选择 “立即并安排更新”。这将在实例上安装代理。Windows 实例已经配置了 SSM 代理，现在将使用该 CodeDeploy 代理进行更新。
7. 在部署设置下，选择 `CodeDeployDefault.OneAtaTime`。
8. 在负载均衡器下，确保未选中启用负载均衡框。您无需为此示例设置负载均衡器或选择目标组。取消选中此复选框后，不会显示负载均衡器选项。
9. 在高级部分中，保留默认值。
10. 选择 Create deployment group（创建部署组）。

第 4 步：在中创建您的第一个管道 CodePipeline

在这部分的教程中，您将创建管道。示例将自动通过管道运行。

创建 CodePipeline 自动发布流程

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在欢迎页面、入门页面或管道页面上，选择创建管道。
3. 在步骤 1：选择管道设置的管道名称中，输入 **MyFirstPipeline**。

   > **Note**
   >
   > 如果您为管道选择另一个名称，请确保在本教程的剩下部分使用该名称替代 **MyFirstPipeline**。创建管道后，便无法再更改其名称。管道名称受一些限制的约束。有关更多信息，请参阅 中的配额 AWS CodePipeline。

4. 在本教程中，为管道类型选择 V1。也可以选择 V2；但请注意，不同管道类型具有不同的特性和价格。有关更多信息，请参阅 管道类型。
5. 在服务角色中，执行下列操作之一：

   - 选择 “新建服务角色” CodePipeline 以允许在 IAM 中创建新的服务角色。
   - 选择现有服务角色以使用已在 IAM 中创建的服务角色。在角色名称中，从列表中选择您的服务角色。

6. 将高级设置中的各项设置保留为默认值，然后选择下一步。
7. 在步骤 2: 添加源阶段的源提供程序中，选择 Amazon S3。在存储桶中，输入您在步骤 1：为您的应用程序创建一个 S3 存储桶中创建的 S3 存储桶的名称。在S3 对象键中，输入带或不带文件路径的对象键。请记住加上文件的扩展名。例如，对于 `SampleApp_Windows.zip`，输入示例文件名，如以下示例所示：

   ```
   SampleApp_Windows.zip
   ```

   选择下一步。

   在更改检测选项下面，保留默认值。这 CodePipeline 允许使用 Amazon E CloudWatch vents 来检测源存储桶中的更改。

   选择下一步。

8. 在步骤 3：添加构建阶段中，选择跳过构建阶段，并通过再次选择跳过接受警告消息。选择下一步。
9. 在步骤 4：添加部署阶段中，在部署提供程序中，选择CodeDeploy 。“区域” 字段默认为与您的管道 AWS 区域 相同。在应用程序名称中，输入 `MyDemoApplication`，或选择刷新按钮，然后从列表中选择应用程序名称。在部署组中，输入 **`MyDemoDeploymentGroup`** 或者从列表中选择它，然后选择下一步。

   > **Note**
   >
   > 名称“Deploy”是在步骤 4：添加部署阶段步骤中创建的阶段的默认名称，就像“Source”是管道第一阶段的默认名称一样。

10. 在步骤 5：审核中，查看信息，然后选择创建管道。
11. 管道开始运行。当 CodePipeline 示例向部署中的每个 Amazon EC2 实例部署网页时，您可以查看进度以及成功和失败消息。 CodeDeploy

恭喜您！您刚刚在中创建了一个简单的管道 CodePipeline。管道具有两个阶段：

- 一个名为 Source 的源阶段，此阶段会检测存储在 S3 存储桶中的受版本控制的示例应用程序中的更改，并将这些更改提取到管道中。
- 部署阶段，用于将这些更改部署到 EC2 实例。 CodeDeploy

现在，验证结果。

验证您的管道是否已成功运行

1. 查看管道的初始进度。每个阶段的状态将从还没有任何执行变为正在进行，然后变为 成功或失败。管道将在几分钟内完成首次运行。
2. 操作状态显示成功之后，在部署阶段的状态区域中，选择详细信息。这将打开 CodeDeploy 控制台。
3. 在部署组选项卡的部署生命周期事件下，选择一个实例 ID。此操作将打开 EC2 控制台。
4. 在描述选项卡上的公有 DNS中，复制地址，然后将其粘贴到 Web 浏览器的地址栏中。查看上传到 S3 存储桶的示例应用程序的索引页面。

   此网页为您上传到 S3 桶的示例应用程序显示。

有关阶段和操作以及管道如何工作的更多信息，请参阅CodePipeline 概念 。

（可选）步骤 5：在管道中添加另一个阶段

现在，在管道中添加另一个阶段，使用 CodeDeploy从暂存服务器部署到生产服务器。首先，在中创建另一个部署组 CodeDeploy。 CodePipelineDemoApplication 然后，您添加一个包含使用此部署组的操作的阶段。要添加另一个阶段，您可以使用 CodePipeline 控制台或在 AWS CLI JSON 文件中检索和手动编辑管道的结构，然后运行update-pipeline命令使用您的更改更新管道。

主题

- 在中创建第二个部署组 CodeDeploy
- 将部署组作为另一个阶段添加到管道中

在中创建第二个部署组 CodeDeploy

> **Note**
>
> 在这部分教程中，您将创建第二个部署组，但要将其部署到与之前相同的 Amazon EC2 实例。这仅用于演示目的。它被故意设计为无法向您显示错误是如何显示的。 CodePipeline

要在中创建第二个部署组 CodeDeploy

1. 打开 CodeDeploy 控制台，网址为 https://console.aws.amazon.com/codedeploy。
2. 选择应用程序，然后在应用程序列表中选择 `MyDemoApplication`。
3. 选择部署组选项卡，然后选择创建部署组。
4. 在创建部署组页面的部署组名称中，输入第二个部署组的名称（例如 **`CodePipelineProductionFleet`**）。
5. 在服务角色中，选择与初始部署相同的 CodeDeploy 服务角色（不是 CodePipeline 服务角色）。
6. 在部署类型下，选择就地。
7. 在环境配置下，选择 Amazon EC2 实例。在键框中选择名称，然后在值框中从列表中选择 `MyCodePipelineDemo`。保留部署设置的默认配置。
8. 在部署配置中，选择 `CodeDeployDefault.OneAtaTime`。
9. 在负载均衡器中，取消选择启用负载均衡。
10. 选择 Create deployment group（创建部署组）。

将部署组作为另一个阶段添加到管道中

有了另一个部署组后，就可以添加一个使用此部署组的阶段来部署到之前所使用的相同 EC2 实例。您可以使用 CodePipeline控制台或 AWS CLI 添加此阶段。

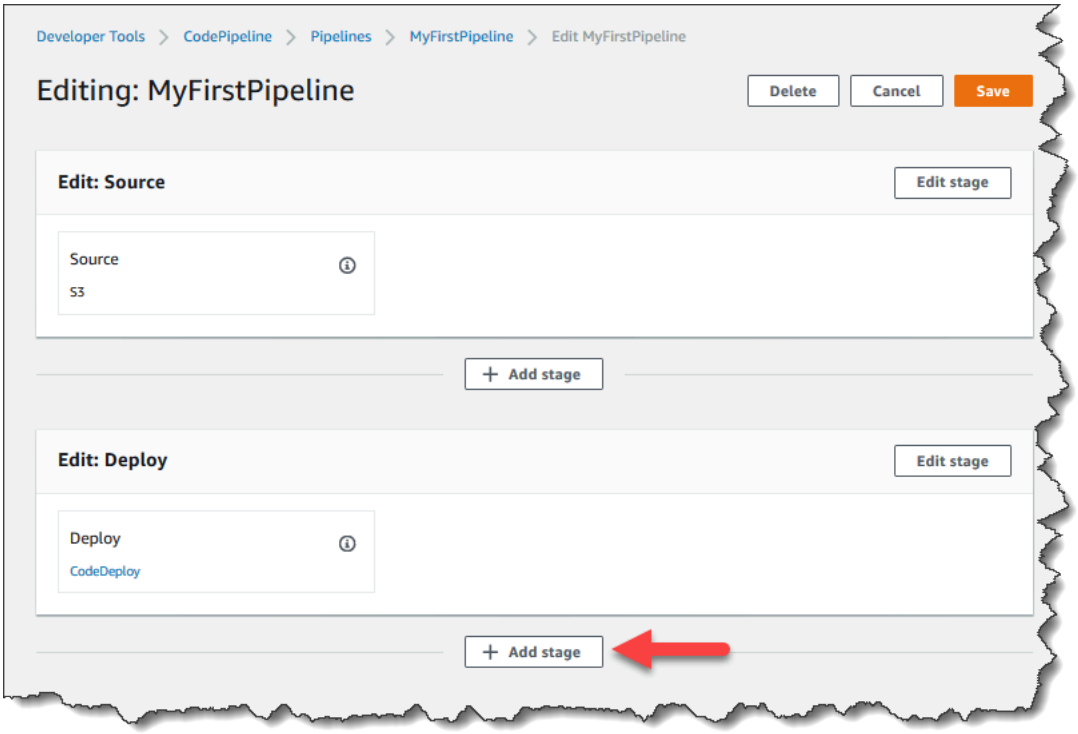
主题

- 创建第三个阶段（控制台）
- 创建第三个阶段（CLI）

创建第三个阶段（控制台）

您可以使用 CodePipeline 控制台添加使用新部署组的新阶段。由于此部署组将部署到您已经使用的 EC2 实例，因此该阶段的部署操作会失败。

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在名称中，选择您创建的管道的名称 MyFirstPipeline。
3. 在管道详细信息页中，选择编辑。
4. 在编辑页面上，选择 + 添加阶段以紧随 Deploy 阶段之后添加一个阶段。



5. 在添加阶段的阶段名称中，输入 **`Production`**。选择添加阶段。

6. 在新阶段中，选择 + 添加操作组。
7. 在编辑操作的操作名称中，输入 **Deploy-Second-Deployment**。在 “操作提供者” 中的 “部署” 下，选择CodeDeploy。
8. 在该 CodeDeploy 部分的应用程序名称中，MyDemoApplication从下拉列表中进行选择，就像创建管道时一样。在部署组中，选择您刚刚创建的部署组 **CodePipelineProductionFleet**。在输入构件中，从源操作中选择输入构件。选择保存。
9. 在编辑页面上，选择保存。在保存管道更改中，选择保存。
10. 虽然新阶段已添加到您的管道中，但该阶段显示为 No executions yet 状态，因为没有发生触发管道再次运行的更改。您必须手动重新运行最新修订，以了解编辑后的管道的运行情况。在管道详细信息页面中，选择发布更改，然后在系统提示时选择发布。这会通过管道运行在源操作中指定的每个源位置中提供的最新修订。

    或者，要使用重新运行管道，请从本地 Linux、macOS 或 Unix 计算机上的终端或本地 Windows 计算机上的命令提示符处运行start-pipeline-execution该命令，指定管道的名称。 AWS CLI 这将第二次通过管道运行您的源存储桶中的应用程序。

    ```
    aws codepipeline start-pipeline-execution --name MyFirstPipeline
    ```

    该命令返回 pipelineExecutionId 对象。
11. 返回 CodePipeline 控制台，在管道列表中 MyFirstPipeline，选择打开视图页面。

    管道显示三个阶段以及项目经历这三个阶段时的状态。管道经历所有阶段可能需要 5 分钟的时间。您将看到前两个阶段的部署取得成功，正如之前一样，但 Production 阶段显示 Deploy-Second-Deployment 操作失败。
12. 在 Deploy-Second-Deployment 操作中，选择 Details。您将被重定向到 CodeDeploy部署页面。在这种情况下，操作失败是因为第一个实例组部署到所有 EC2 实例，导致第二个部署组没有实例可用。

    > **Note**
    >
    > 此故障是特意设计的，是为了演示某个管道阶段出现故障会怎么样。

创建第三个阶段（CLI）

尽管使用 AWS CLI 向管道添加阶段比使用控制台更为复杂，但它可以更清楚地了解管道的结构。

为管道创建第三个阶段

1. 在本地 Linux、macOS 或 Unix 计算机上打开终端会话，或者在本地 Windows 计算机上打开命令提示符，运行 get-pipeline 命令，以显示您刚创建的管道的结构。对于 **MyFirstPipeline**，您可以键入以下命令：

```
aws codepipeline get-pipeline --name "MyFirstPipeline"
```

此命令返回的结构 MyFirstPipeline。输出的第一部分应类似于以下内容：

```
{
    "pipeline": {
        "roleArn": "arn:aws:iam::80398EXAMPLE:role/AWS-CodePipeline-Service",
        "stages": [
    ...
```

输出的最后部分包括管道元数据，应类似于以下内容：

```
    ...
        ],
        "artifactStore": {
            "type": "S3"
            "location": "codepipeline-us-east-2-250656481468",
        },
        "name": "MyFirstPipeline",
        "version": 4
    },
    "metadata": {
        "pipelineArn": "arn:aws:codepipeline:us-
east-2:80398EXAMPLE:MyFirstPipeline",
        "updated": 1501626591.112,
        "created": 1501626591.112
    }
}
```

2. 将此结构复制并粘贴到纯文本编辑器中，并将文件保存为 **pipeline.json**。为了方便起见，请将此文件保存在运行 aws codepipeline 命令的相同目录中。

> **Note**
>
> 您可以使用 get-pipeline 命令将 JSON 直接添加到文件中，如下所示：

```
aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
```

3. 将部署阶段部分复制并粘贴在前两个阶段之后。由于它是一个部署阶段，就像部署阶段一样，您可以将其用作第三个阶段的模板。

4. 更改阶段的名称和部署组详细信息。

   以下示例显示将在部署阶段结束后添加到 pipeline.json 文件的 JSON。使用新值编辑突出显示的元素。请记住，使用逗号分隔部署和生产阶段定义。

```
,
{
    "name": "Production",
     "actions": [
        {
         "inputArtifacts": [
             {
              "name": "MyApp"
             }
          ],
          "name": "Deploy-Second-Deployment",
          "actionTypeId": {
              "category": "Deploy",
              "owner": "AWS",
              "version": "1",
              "provider": "CodeDeploy"
              },
          "outputArtifacts": [],
          "configuration": {
              "ApplicationName": "CodePipelineDemoApplication",
              "DeploymentGroupName": "CodePipelineProductionFleet"
               },
          "runOrder": 1
        }
    ]
}
```

5. 如果您要使用通过 get-pipeline 命令检索到的管道结构，则必须删除 JSON 文件中的 `metadata` 行。否则，update-pipeline 命令无法使用它。删除 `"metadata": { }` 行以及 `"created"`、`"pipelineARN"` 和 `"updated"` 字段。

例如，从结构中删除以下各行：

```
"metadata": {
  "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
  "created": "date",
  "updated": "date"
  }
```

保存该文件。

6. 运行 update-pipeline 命令并指定管道 JSON 文件，类似于以下内容：

```
aws codepipeline update-pipeline --cli-input-json file://pipeline.json
```

该命令会返回更新后管道的整个结构。

> **Important**
>
> 务必在文件名前包含 `file://`。此命令中需要该项。

7. 运行 start-pipeline-execution 命令，并且指定管道的名称。这将第二次通过管道运行您的源存储桶中的应用程序。

```
aws codepipeline start-pipeline-execution --name MyFirstPipeline
```

该命令返回 `pipelineExecutionId` 对象。

8. 打开 CodePipeline 控制台并MyFirstPipeline从管道列表中进行选择。

管道显示三个阶段以及项目经历这三个阶段时的状态。管道经历所有阶段可能需要 5 分钟的时间。虽然前两个阶段的部署取得成功，正如之前一样，但 Production 阶段显示 Deploy-Second-Deployment 操作失败。

9. 在 Deploy-Second-Deployment 操作中，选择 Details 以查看故障详细信息。您将被重定向到 CodeDeploy 部署的详细信息页面。在这种情况下，操作失败是因为第一个实例组部署到所有 EC2 实例，导致第二个部署组没有实例可用。

> **Note**
>
> 此故障是特意设计的，是为了演示某个管道阶段出现故障会怎么样。

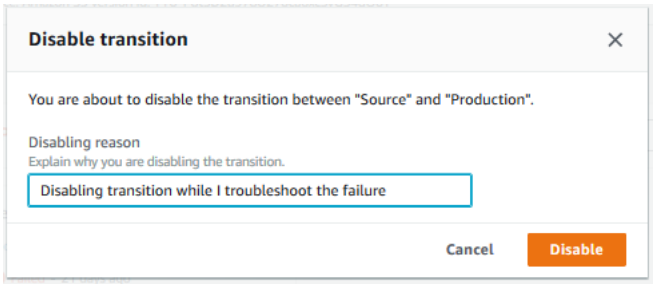
（可选）步骤 6：禁用和启用阶段之间的过渡 CodePipeline

您可以启用或禁用管道中阶段之间的过渡。禁用阶段之间的过渡允许您手动控制一个阶段和另一个阶段之间的过渡。例如，您可能希望运行管道的前两个阶段，但是禁用向第三阶段的过渡，直到您准备好部署到生产环境，或者您要排查该阶段的问题或故障。

禁用和启用 CodePipeline 管道中各个阶段之间的过渡

1. 打开 CodePipeline 控制台并MyFirstPipeline从管道列表中进行选择。
2. 在管道详细信息页面中，选择第二个阶段（部署）与您在上一部分中添加的第三个阶段（生产）之间的禁用过渡按钮。
3. 在禁用过渡中，键入禁用阶段过渡的原因，然后选择禁用。

   各个阶段之间的箭头显示一个图标和颜色变化，以及启用过渡按钮。



4. 再次将示例应用程序上传到 S3 存储桶。由于存储桶启用了版本控制，因此这一更改将启动管道。
5. 返回管道详细信息页面，查看各个阶段的状态。管道视图会随之发生变化，以显示前两个阶段的进度和成功消息，但第三个阶段不会发生任何变化。此过程可能需要几分钟时间。
6. 通过选择两个阶段之间的启用过渡按钮，即可启用过渡。在 Enable transition 对话框中，选择 Enable。这一阶段将在几分钟内开始运行，并尝试处理已经历管道的前两个阶段的项目。

> **Note**
>
> 如果您希望第三阶段成功，请在启用过渡之前编辑 CodePipelineProductionFleet 部署组，并指定部署应用程序的另一组 EC2 实例。有关如何执行此操作的更多信息，请参阅更改部署组设置。如果您创建更多的 EC2 实例，可能会产生额外成本。

步骤 7：清理资源

您可以将在本教程中创建的一些资源用于教程：创建一个四阶段管道。例如，您可以重复使用 CodeDeploy 应用程序和部署。您可以使用提供者（例如）配置生成操作 CodeBuild，这是云中完全托管的生成服务。您还可以配置一个将提供方与生成服务器或系统结合使用的生成操作，例如 Jenkins。

但是，在完成本教程和任何其他教程之后，您应该删除管道及其使用的资源，以避免为继续使用这些资源付费。首先，删除管道，然后删除 CodeDeploy 应用程序及其关联的 Amazon EC2 实例，最后删除 S3 存储桶。

清理本教程中使用的资源

1. 要清理您的 CodePipeline 资源，请按照中删除管道中的说明进行操作 AWS CodePipeline。
2. 要清理 CodeDeploy 资源，请按照清理资源（控制台）中的说明进行操作。
3. 要删除 S3 存储桶，请按照删除或清空存储桶中的说明操作。如果您不打算创建更多管道，请删除为存储管道构件而创建的 S3 存储桶。有关此存储桶的更多信息，请参阅CodePipeline 概念 。

教程：创建简单的管道（CodeCommit存储库）

在本教程中，您将使用 CodePipeline 将 CodeCommit 存储库中维护的代码部署到单个 Amazon EC2 实例。当您将更改推送到 CodeCommit 存储库时，您的管道就会被触发。该管道使用 CodeDeploy 部署服务将您的更改部署到 Amazon EC2 实例。

管道具有两个阶段：

- 源操作的源阶段（ CodeCommit 来源）。
- 您的部署操作的部署阶段（ CodeDeploy 部署）。

最简单的入门方法 AWS CodePipeline 是在 CodePipeline 控制台中使用创建管道向导。

> **Note**
>
> 在开始之前，请确保已设置好要使用的 Git 客户端 CodeCommit。有关说明，请参阅设置 CodeCommit。

步骤 1：创建 CodeCommit 存储库

首先，在中创建存储库 CodeCommit。您的管道在运行时将从该存储库获取源代码。您还可以创建一个本地存储库，在将代码推送到存储 CodeCommit 库之前，在其中维护和更新代码。

创建存储 CodeCommit 库

1. 打开 CodeCommit 控制台，网址为 https://console.aws.amazon.com/codecommit/。
2. 在区域选择器中，选择要创建存储库和管道 AWS 区域 的位置。有关更多信息，请参阅AWS 区域和端点。
3. 在存储库页面上，选择创建存储库。
4. 在创建存储库页面上的存储库名称中，输入存储库的名称（例如，**MyDemoRepo**）。
5. 选择创建。

> **Note**
>
> 本教程的其余步骤将使用**MyDemoRepo**存储 CodeCommit 库的名称。如果您选择其他名称，请确保在本教程中通篇使用它。

设置本地存储库

在此步骤中，您将设置本地存储库以连接到您的远程 CodeCommit存储库。

> **Note**
>
> 您无需设置本地存储库。还可以使用此控制台上传文件（如第 2 步：向 CodeCommit 存储库添加示例代码所述）。

1. 在控制台中打开您的新存储库，选择页面右上角的克隆 URL，然后选择克隆 SSH。用于克隆 Git 存储库的地址复制到您的剪贴板。
2. 在您的终端窗口或命令行中，导航到您要用来存储本地存储库的本地目录。在本教程中，我们使用 `/tmp`。
3. 运行以下命令以克隆存储库，使用您在上一步中复制的地址替换 SSH 地址。此命令创建一个名为 `MyDemoRepo` 的目录。您将示例应用程序复制到此目录。

```
git clone ssh://git-codecommit.us-west-2.amazonaws.com/v1/repos/MyDemoRepo
```

第 2 步：向 CodeCommit 存储库添加示例代码

在此步骤中，您将下载为示例演练创建的 CodeDeploy示例应用程序的代码，并将其添加到 CodeCommit 存储库中。

1. 接下来，下载一个示例，并将该示例保存到本地计算机上的文件夹或目录中。

   a. 选择以下选项之一。如果要对 Linux 实例执行本教程中的步骤，请选择 `SampleApp_Linux.zip`。

   - 如果您想使用部署到亚马逊 Linux 实例 CodeDeploy，请在此处下载示例应用程序：SampleApp_Linux.zip。
   - 如果你想使用部署到 Windows 服务器实例 CodeDeploy，请在此处下载示例应用程序：SampleApp_Windows.zip。

   示例应用程序包含以下用于部署的文件 CodeDeploy：

   - `appspec.yml`— 应用程序规范文件（AppSpec文件）是一个 YAML 格式的文件，CodeDeploy 用于管理部署。有关该 AppSpec 文件的更多信息，请参阅AWS CodeDeploy 用户指南中的CodeDeploy AppSpec 文件参考。
   - `index.html` - 索引文件包含已部署的示例应用程序的主页。
   - `LICENSE.txt` - 许可证文件包含示例应用程序的许可证信息。
   - 脚本文件 - 示例应用程序使用脚本将文本文件写入实例上的某个位置。为多个 CodeDeploy 部署生命周期事件分别写入一个文件，如下所示：

- （仅限 Linux 示例）`scripts` 文件夹 - 该文件夹包含以下 shell 脚本，用于安装依赖项以及启动和停止自动部署的示例应用程序：`install_dependencies`、`start_server` 和 `stop_server`。
- （仅限 Windows 示例）`before-install.bat` - 此批处理脚本用于 `BeforeInstall` 部署生命周期事件，它将运行以删除此示例在先前部署中写入的旧文件，并在您的实例上创建一个位置用于写入新文件。

b. 下载压缩（zipped）文件。

2. 将文件从 SampleApp_Linux.zip 解压缩到之前创建的本地目录（例如，`/tmp/MyDemoRepo`或`c:\temp\MyDemoRepo`）。

   务必将文件直接放到本地存储库中。不要包括 `SampleApp_Linux` 文件夹。例如，在您的本地 Linux、macOS 或 Unix 计算机上，您的目录和文件层次结构应如下所示：

```
/tmp
   #-- MyDemoRepo
       #-- appspec.yml
       #-- index.html
       #-- LICENSE.txt
       #-- scripts
           #-- install_dependencies
           #-- start_server
           #-- stop_server
```

3. 要将文件上传到您的存储库，请使用以下方法之一。

   a. 要使用 CodeCommit 控制台上传文件，请执行以下操作：

      i. 打开 CodeCommit 控制台，然后从 “存储库” 列表中选择您的存储库。

      ii. 选择添加文件，然后选择上传文件。

      iii. 选择选择文件，然后浏览找到您的文件。要在文件夹下添加文件，请选择创建文件，然后输入带有文件名的文件夹名称，例如 `scripts/install_dependencies`。将文件内容粘贴到新文件中。

      通过输入您的用户名和电子邮件地址来提交更改。

      选择提交更改。

      iv. 对每个文件重复此步骤。

      您的存储库内容应如下所示：

```
      #-- appspec.yml
      #-- index.html
      #-- LICENSE.txt
      #-- scripts
          #-- install_dependencies
          #-- start_server
          #-- stop_server
```

b. 使用 git 命令上传文件：

i. 将目录更改为本地存储库：

```
(For Linux, macOS, or Unix) cd /tmp/MyDemoRepo
(For Windows) cd c:\temp\MyDemoRepo
```

ii. 运行以下命令以立即暂存您的所有文件：

```
git add -A
```

iii. 运行以下命令以提交带有提交消息的文件：

```
git commit -m "Add sample application files"
```

iv. 运行以下命令将本地存储库中的文件推送到存储 CodeCommit库：

```
git push
```

4. 您下载并添加到本地存储库中的文件现已添加到 CodeCommit `MyDemoRepo`存储库的`main`分支中，可以包含在管道中。

步骤 3：创建 Amazon EC2 Linux 实例并安装 CodeDeploy 代理

在此步骤中，您将创建要在其中部署示例应用程序的 Amazon EC2 实例。作为此过程的一部分，创建一个允许在实例上安装和管理 CodeDeploy 代理的实例角色。 CodeDeploy 代理是一个软件包，允许在 CodeDeploy 部署中使用实例。您还可以附加策略，允许实例获取 CodeDeploy 代理用于部署您的应用程序的文件，并允许该实例由 SSM 管理。

创建实例角色

1. 通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在控制台控制面板中，选择角色。
3. 选择创建角色。
4. 在选择受信任实体的类型下，选择 AWS 服务。在选择使用案例下，选择 EC2。在选择您的使用案例下，选择 EC2。选择下一步：权限。
5. 搜索名为 **AmazonEC2RoleforAWSCodeDeploy** 的策略并将其选中。
6. 搜索名为 **AmazonSSMManagedInstanceCore** 的策略并将其选中。选择下一步：标签。
7. 选择下一步：审核。输入角色的名称（例如 **EC2InstanceRole**）。

> **Note**
>
> 记下您的角色名称以便在下一步中使用。在创建实例时选择此角色。

选择 创建角色。

启动实例

1. 通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。
2. 在侧面导航栏中，选择实例，然后从页面顶部选择启动实例。
3. 在名称中，输入 **MyCodePipelineDemo**。这会为实例分配标签键 **Name** 和标签值 **MyCodePipelineDemo**。稍后，您将创建一个将示例 CodeDeploy 应用程序部署到此实例的应用程序。 CodeDeploy根据标签选择要部署的实例。
4. 在“应用程序和操作系统映像（亚马逊系统映像）”下，找到带有 AWS 徽标的 Amazon Linux AMI 选项，并确保将其选中。（此 AMI 称为 Amazon Linux 2 AMI (HVM)，并标记为“符合免费套餐条件”。）
5. 在实例类型下，选择符合免费套餐条件的 `t2.micro` 类型作为实例的硬件配置。
6. 在密钥对(登录) 下，选择或创建密钥对。

   也可以选择继续操作但不提供密钥对。

> **Note**
>
> 在本教程中，您可以继续操作而不提供密钥对。要使用 SSH 连接到实例，请创建或使用密钥对。

7. 在网络设置下，执行以下操作：

   在自动分配公有 IP 中，确保状态为启用。

   - 在分配安全组旁边，选择创建新安全组。
   - 在 SSH 对应的行中，在源下选择我的 IP。
   - 选择添加安全组，选择 HTTP，然后在源类型下选择我的 IP。
8. 展开高级详细信息。在 IAM 实例配置文件中，选择您在上一过程中创建的 IAM 角色（例如 **`EC2InstanceRole`**）。
9. 在摘要下的实例数下，输入 1。
10. 选择启动实例。
11. 您可以在实例页面上查看启动的状态。启动实例时，其初始状态为 `pending`。实例启动后，其状态变为 `running`，并且会收到一个公有 DNS 名称。（如果公有 DNS 列不显示，请选择显示/隐藏图标，然后选择公有 DNS。）

步骤 4：在中创建应用程序 CodeDeploy

在中 CodeDeploy，应用程序是一种包含要部署的软件应用程序的资源。稍后，您将此应用程序与配合使用 CodePipeline ，自动将示例应用程序部署到您的 Amazon EC2 实例。

首先，创建一个允许执行部署 CodeDeploy 的角色。然后，您将创建一个 CodeDeploy 应用程序。

创建 CodeDeploy 服务角色

1. 通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。
2. 在控制台控制面板中，选择角色。
3. 选择创建角色。
4. 在选择受信任实体下，选择 AWS 服务。在 Use case（使用案例）下，选择 CodeDeploy。CodeDeploy从列出的选项中进行选择。选择下一步。`AWSCodeDeployRole` 托管策略已附加到角色。

5. 选择下一步。
6. 输入角色的名称（例如 **CodeDeployRole**），然后选择创建角色。

要在中创建应用程序 CodeDeploy

1. 打开 CodeDeploy 控制台，网址为 https://console.aws.amazon.com/codedeploy。
2. 如果未显示应用程序页面，请在菜单上选择应用程序。
3. 选择创建应用程序。
4. 在应用程序名称中，输入 **MyDemoApplication**。
5. 在 Compute Platform（计算平台）中，选择 EC2/On-premises（EC2/本地）。
6. 选择创建应用程序。

在中创建部署组 CodeDeploy

部署组是定义了部署相关设置（例如，要部署到哪些实例以及部署它们的速度有多快）的资源。

1. 在显示应用程序的页面上，选择创建部署组。
2. 在 Deployment group name（部署组名称）中，输入 **MyDemoDeploymentGroup**。
3. 在服务角色中，选择您之前创建的服务角色的 ARN（例如，**arn:aws:iam::*account_ID*:role/CodeDeployRole**）。
4. 在部署类型下，选择就地。
5. 在环境配置下，选择 Amazon EC2 实例。在键字段中，输入 **Name**。在值字段中，输入您用于标记实例的名称（例如，**MyCodePipelineDemo**）。
6. 在 “使用 S AWS ystems Manager 配置代理” 下，选择 “立即并安排更新”。这将在实例上安装代理。Linux 实例已经配置了 SSM 代理，现在将使用 CodeDeploy 代理进行更新。
7. 在部署配置中，选择 CodeDeployDefault.OneAtaTime。
8. 在负载均衡器下，确保未选中启用负载均衡。您无需为此示例设置负载均衡器或选择目标组。
9. 选择 Create deployment group（创建部署组）。

第 5 步：在中创建您的第一个管道 CodePipeline

现在您已经可以创建并运行您的第一个管道。在此步骤中，您将创建一个在代码推送到 CodeCommit 存储库时自动运行的管道。

创建 CodePipeline 管道

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

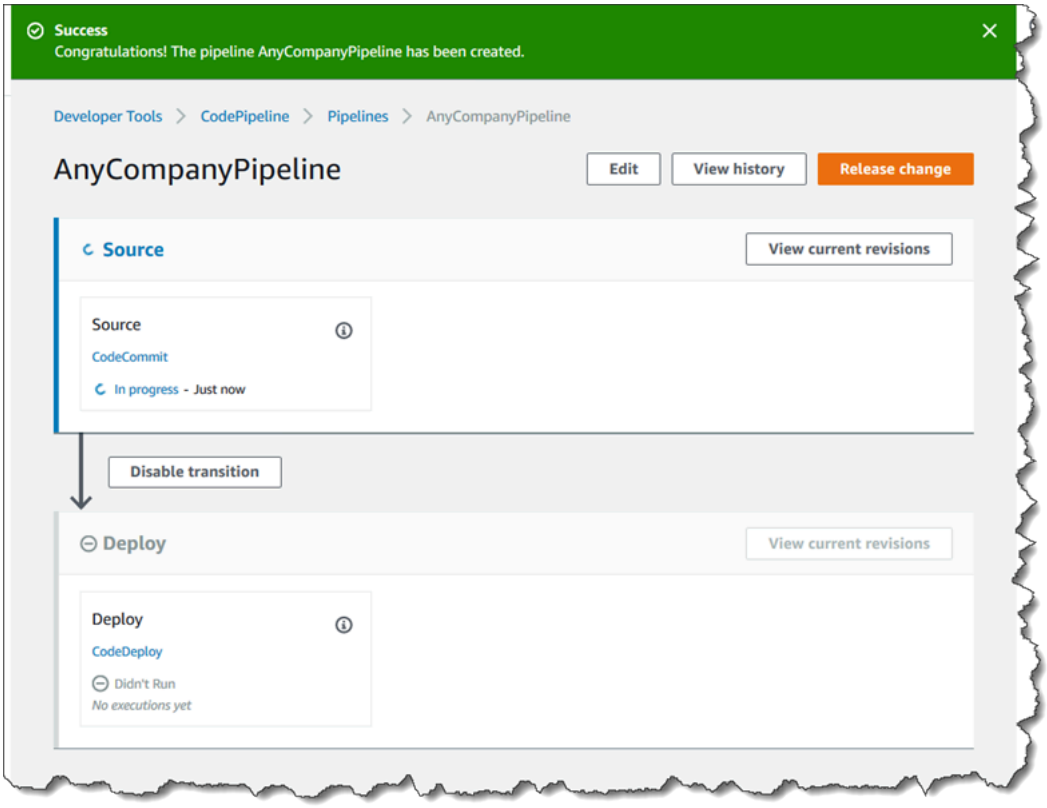
   打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。
2. 选择创建管道。
3. 在步骤 1：选择管道设置的管道名称中，输入 **MyFirstPipeline**。
4. 在本教程中，为管道类型选择 V1。也可以选择 V2；但请注意，不同管道类型具有不同的特性和价格。有关更多信息，请参阅 管道类型。
5. 在服务角色中，选择新建服务角色 CodePipeline 以允许在 IAM 中创建服务角色。
6. 将高级设置中的各项设置保留为默认值，然后选择下一步。
7. 在步骤 2：添加源舞台中，在源提供程序中，选择CodeCommit。在存储库名称中，选择您在中创建的 CodeCommit 存储库的名称步骤 1：创建 CodeCommit 存储库。在分支名称中，选择 `main`，然后选择下一步。

   选择存储库名称和分支后，将显示一条消息，显示要为此管道创建的 Amazon Ev CloudWatch ents 规则。

   在更改检测选项下面，保留默认值。这 CodePipeline 允许使用 Amazon E CloudWatch vents 来检测源存储库中的更改。

   选择下一步。
8. 在步骤 3：添加构建阶段中，选择跳过构建阶段，并通过再次选择跳过接受警告消息。选择下一步。

   > **Note**
   >
   > 在本教程中，您将部署不需要生成服务的代码，因此您可以跳过此步骤。但是，如果您的源代码需要在部署到实例之前进行构建，则可以在此步骤CodeBuild中进行配置。
9. 在步骤 4：添加部署阶段中，在部署提供程序中，选择CodeDeploy。在应用程序名称中，选择 **MyDemoApplication**。在部署组中，选择 **MyDemoDeploymentGroup**，然后选择下一步。
10. 在步骤 5：审核中，查看信息，然后选择创建管道。
11. 创建管道后它会开始运行。它会从您的 CodeCommit存储库下载代码并创建到您的 EC2 实例的 CodeDeploy 部署。当 CodePipeline 示例将网页部署到部署中的 Amazon EC2 实例时，您可以查看进度以及成功和失败消息。 CodeDeploy



恭喜您！您刚刚在中创建了一个简单的管道 CodePipeline。

接下来，您将验证结果。

验证您的管道是否成功运行

1. 查看管道的初始进度。每个阶段的状态将从还没有任何执行变为正在进行，然后变为 成功或失败。管道将在几分钟内完成首次运行。
2. 在管道状态显示成功后，在 “部署” 阶段的状态区域中选择CodeDeploy。这将打开 CodeDeploy 控制台。如果未显示成功，请参阅 故障排除 CodePipeline。
3. 在部署选项卡上，选择部署 ID。在部署页面上的部署生命周期事件下，选择实例 ID。此操作将打开 EC2 控制台。
4. 在描述选项卡上的公有 DNS 中，复制地址（例如，`ec2-192-0-2-1.us-west-2.compute.amazonaws.com`），然后将其粘贴到 Web 浏览器的地址栏中。

   将显示您下载并推送到 CodeCommit 存储库的示例应用程序的网页。

有关阶段和操作以及管道如何工作的更多信息，请参阅CodePipeline 概念 。

第 6 步：修改 CodeCommit 仓库中的代码

您的管道配置为在 CodeCommit仓库发生代码更改时运行。在此步骤中，您将对作为 CodeCommit 存储库中示例 CodeDeploy应用程序一部分的 HTML 文件进行更改。当您推送这些更改时，您的管道会再次运行，并且您作出的更改将在您之前访问的网址中显示。

1. 将目录更改为本地存储库：

```
(For Linux, macOS, or Unix) cd /tmp/MyDemoRepo
(For Windows) cd c:\temp\MyDemoRepo
```

2. 使用文本编辑器修改 index.html 文件：

```
(For Linux or Unix)gedit index.html
(For OS X)open -e index.html
(For Windows)notepad index.html
```

3. 修订 index.html 文件的内容，以更改网页的背景颜色和一些文本，然后保存该文件。

```
<!DOCTYPE html>
<html>
<head>
  <title>Updated Sample Deployment</title>
  <style>
    body {
      color: #000000;
      background-color: #CCFFCC;
      font-family: Arial, sans-serif;
      font-size:14px;
    }

    h1 {
      font-size: 250%;
      font-weight: normal;
      margin-bottom: 0;
    }

    h2 {
      font-size: 175%;
      font-weight: normal;
      margin-bottom: 0;
    }
```

```
  </style>
</head>
<body>
  <div align="center"><h1>Updated Sample Deployment</h1></div>
  <div align="center"><h2>This application was updated using CodePipeline,
 CodeCommit, and CodeDeploy.</h2></div>
  <div align="center">
    <p>Learn more:</p>
    <p><a href="https://docs.aws.amazon.com/codepipeline/latest/
userguide/">CodePipeline User Guide</a></p>
    <p><a href="https://docs.aws.amazon.com/codecommit/latest/
userguide/">CodeCommit User Guide</a></p>
    <p><a href="https://docs.aws.amazon.com/codedeploy/latest/
userguide/">CodeDeploy User Guide</a></p>
  </div>
</body>
</html>
```

4. 通过运行以下命令逐一提交更改并将其推送到 CodeCommit 存储库：

```
git commit -am "Updated sample application files"
```

```
git push
```

验证您的管道是否已成功运行

1. 查看管道的初始进度。每个阶段的状态将从还没有任何执行变为正在进行，然后变为 成功或失败。管道应在几分钟内完成运行。
2. 操作状态显示成功之后，请刷新您之前在浏览器中访问的演示页面。

   此时将显示更新后的网页。

步骤 7：清理资源

您可以将在本教程中创建的一些资源用于本指南中的其他教程。例如，您可以重复使用 CodeDeploy 应用程序和部署。但是，在完成本教程和任何其他教程之后，您应该删除管道及其使用的资源，以避免为继续使用这些资源付费。首先，删除管道，然后删除 CodeDeploy 应用程序及其关联的 Amazon EC2 实例，最后删除 CodeCommit存储库。

清理本教程中使用的资源

1. 要清理您的 CodePipeline 资源，请按照中删除管道中的说明进行操作 AWS CodePipeline。
2. 要清理 CodeDeploy 资源，请按照清理部署演练资源中的说明进行操作。
3. 要删除 CodeCommit 存储库，请按照删除存储 CodeCommit库中的说明进行操作。

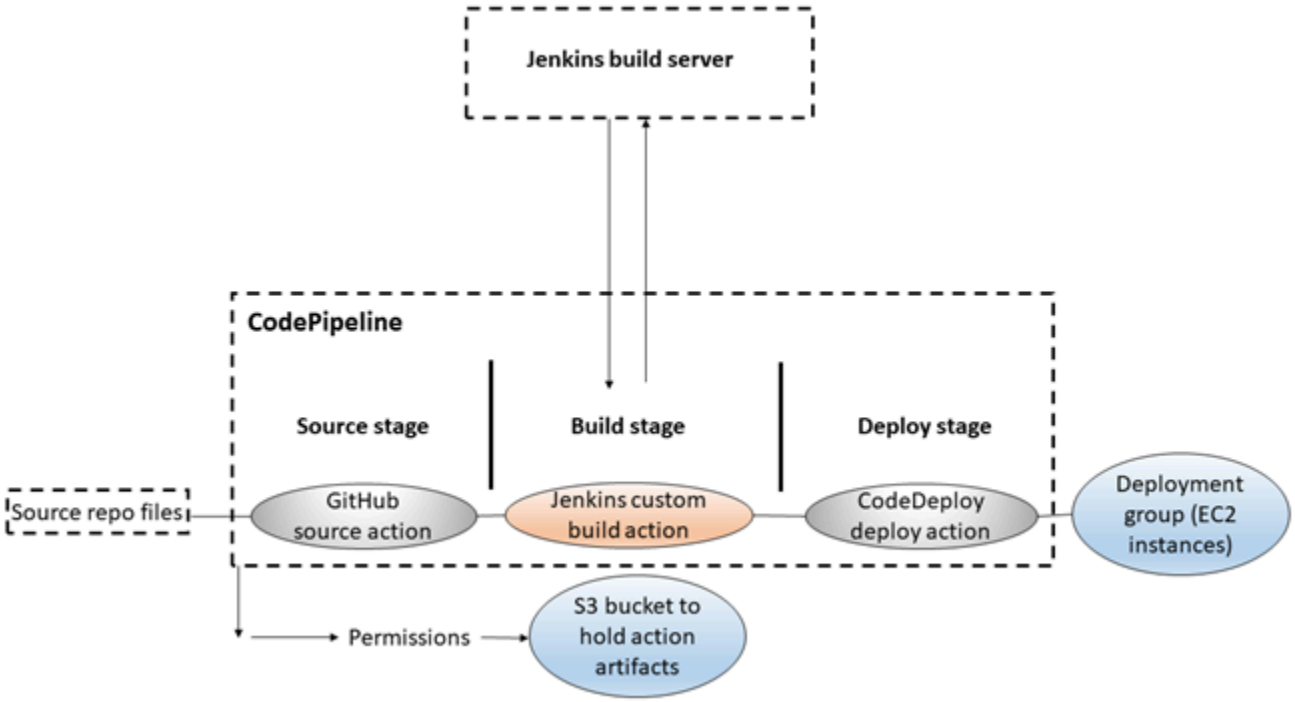
步骤 8：延伸阅读

详细了解 CodePipeline 工作原理：

- 有关阶段和操作以及管道如何工作的更多信息，请参阅CodePipeline 概念 。
- 有关您可以使用执行的操作的信息 CodePipeline，请参阅与动 CodePipeline 作类型的集成。
- 试用这个更高级的教程：教程：创建一个四阶段管道。它创建一个多阶段管道，包括一个在部署之前构建代码的步骤。

教程：创建一个四阶段管道

在教程：创建一个简单的管道（S3 存储桶）或教程：创建简单的管道（CodeCommit存储库）中创建了第一个管道后，接下来就可以开始创建更复杂的管道。本教程将引导你创建四阶段管道，该管道使用GitHub存储库作为源代码，使用 Jenkins 生成服务器来生成项目，使用 CodeDeploy 应用程序将生成的代码部署到暂存服务器。下图显示最初的三阶段管道。



创建管道后，您将编辑它以添加包含测试操作的阶段来测试代码，同样使用 Jenkins。

创建此管道之前，必须先配置所需的资源。例如，如果要为源代码使用 GitHub 存储库，则必须先创建存储库，然后才能将其添加到管道中。作为设置的一部分，本教程将指导您在 EC2 实例上设置 Jenkins 以进行演示。

> **Important**
>
> 在此过程中，您在管道中添加的许多操作都涉及在创建管道之前需要创建的 AWS 资源。 AWS 源操作的资源必须始终在您创建管道的同一 AWS 区域创建。例如，如果您在美国东部（俄亥俄州）地区创建管道，则您的 CodeCommit 存储库必须位于美国东部（俄亥俄州）区域。
> 您可以在创建管道时添加跨区域操作。 AWS 跨区域操作的资源必须位于您计划执行操作的同一 AWS 区域。有关更多信息，请参阅 在中添加跨区域操作 CodePipeline。

在开始学习本教程之前，您应该已经完成入门 CodePipeline中的一般先决条件。

主题

- 步骤 1：满足先决条件
- 步骤 2：在中创建管道 CodePipeline

步骤 1：满足先决条件

要与 Jenkins 集成， AWS CodePipeline 需要您在要使用的任何 Jenkins 实例上安装适用于 Jenkins 的 CodePipeline 插件。 CodePipeline您还应该配置一个专用 IAM 用户或角色，用于在 Jenkins 项目和 CodePipeline之间获得权限。集成 Jenkins 的最简单方法 CodePipeline 是在使用你为 Jenkins 集成创建的 IAM 实例角色的 EC2 实例上安装 Jenkins。为了使管道中 Jenkins 操作的链接能够成功连接，您必须在服务器或 EC2 实例上配置代理和防火墙设置，以允许与 Jenkins 项目使用的端口进行入站连接。在允许连接这些端口（例如，如果您已保护 Jenkins 以仅使用 HTTPS 连接，端口为 443 和 8443；如果您允许 HTTP 连接，端口为 80 和 8080）之前，请确保已将 Jenkins 配置为对用户进行身份验证并强制执行访问控制。有关更多信息，请参阅保护 Jenkins。

> **Note**
>
> 本教程使用一个代码示例并配置将该示例从 Haml 转换为 HTML 的生成步骤。您可以按照中的步骤从 GitHub 存储库下载开源示例代码将示例复制或克隆到 GitHub 存储库中。您将需要 GitHub 存储库中的整个示例，而不仅仅是.zip 文件。
>
> 本指南还假定：
>
> - 您熟悉安装和管理 Jenkins 以及创建 Jenkins 项目的过程。
> - 您已将适用于 Ruby 的 Rake 和 Haml Gem 安装在托管 Jenkins 项目的同一计算机或实例上。
> - 您已设置所需的系统环境变量，以便可以从终端或命令行运行 Rake 命令（例如，在 Windows 系统上，修改 PATH 变量以包括安装 Rake 的目录）。

主题

将示例复制或克隆到 GitHub 存储库中

克隆示例并推送到 GitHub 存储库

1. 从 GitHub 存储库下载示例代码，或将存储库克隆到本地计算机。有两个示例包：
   - 如果您要将示例部署到亚马逊 Linux、RHEL 或 Ubuntu 服务器实例，请选择 _linux.zip。codepipeline-jenkins-aws-codedeploy
   - 如果您要将示例部署到 Windows 服务器实例，请选择 CodePipeline-Jenkins-.zip。AWSCodeDeploy_Windows
2. 在存储库中，选择 Fork，将示例存储库克隆到您的 Github 账户的存储库中。有关更多信息，请参阅GitHub文档。

创建 IAM 角色以用于 Jenkins 集成

作为最佳实践，可以考虑启动 EC2 实例来托管您的 Jenkins 服务器，并使用 IAM 角色向该实例授予与 CodePipeline之交互所需的权限。

1. 登录 AWS Management Console 并打开 IAM 控制台，网址为 https://console.aws.amazon.com/iam/。
2. 在 IAM 控制台的导航窗格中，选择角色，然后选择创建角色。
3. 在选择受信任实体的类型下，选择 AWS 服务。在选择将使用此角色的服务下，选择 EC2。在选择您的使用案例下，选择 EC2。
4. 选择下一步：权限。在附加权限策略页面上，选择 `AWSCodePipelineCustomActionAccess` 托管策略，然后选择下一步: 标签。选择 下一步: 审核。
5. 在 “审阅” 页面的 “角色名称” 中，输入要专门为 Jenkins 集成创建的角色名称（例如 *`JenkinsAccess`*），然后选择创建角色。

创建要在其中安装 Jenkins 的 EC2 实例时，请务必在步骤 3：配置实例详细信息中选择实例角色（例如 *`JenkinsAccess`*）。

有关实例角色和 Amazon EC2 的更多信息，请参阅适用于 Amazon EC2 的 IAM 角色、使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授权和创建向 AWS 服务委托权限的角色。

安装和配置 Jenkins 和适用于 Jenkins 的 CodePipeline 插件

安装 Jenkins 和适用于 Jenkins 的 CodePipeline 插件

1. 创建要在其中安装 Jenkins 的 EC2 实例，在步骤 3：配置实例详细信息中，确保选择您创建的实例角色（例如 *JenkinsAccess*）。有关创建 EC2 实例的更多信息，请参阅 Amazon EC2 用户指南 中的启动 Amazon EC2 实例。

   > **Note**
   >
   > 如果您已经拥有您想要使用的 Jenkins 资源，您可以执行此操作，但是您必须创建一个专用的 IAM 用户，为该用户应用 AWSCodePipelineCustomActionAccess 托管策略，然后在您的 Jenkins 资源上配置和使用该用户的访问凭证。如果您想使用 Jenkins UI 来提供凭证，请将 Jenkins 配置为仅允许 HTTPS。有关更多信息，请参阅 故障排除 CodePipeline。

2. 在 EC2 实例上安装 Jenkins。有关更多信息，请参阅针对安装 Jenkins 和启动并访问 Jenkins 的 Jenkins 文档以及 产品和服务与 CodePipeline中的details of integration with Jenkins。
3. 启动 Jenkins，然后在主页上，选择管理 Jenkins。
4. 在管理 Jenkins 页中，选择管理插件。
5. 选择可用选项卡，然后在筛选条件搜索框中，输入 **AWS CodePipeline**。从列表中选择 Jenkins 的CodePipeline 插件，然后选择立即下载并在重启后安装。
6. 在正在安装插件/更新页中，选择当安装结束且无作业正在运行时重启 Jenkins。
7. 选择返回控制面板。
8. 在主页中，选择新项目。
9. 在项目名称中，输入 Jenkins 项目的名称（例如，*MyDemoProject*）。选择自由式项目，然后选择确定。

   > **Note**
   >
   > 请确保您的项目名称符合的要求 CodePipeline。有关更多信息，请参阅 中的配额 AWS CodePipeline。

10. 在项目的配置页中，选择如有必要，执行并发构建复选框。在源代码管理中，选择 AWS CodePipeline。如果您已在 EC2 实例上安装了 Jenkins，并使用您为 AWS CLI 与 Jenkins 之间的 CodePipeline 集成而创建的 IAM 用户的配置文件配置了，请将所有其他字段留空。

11. 选择“高级”，然后在“提供者”中输入操作提供者的名称 CodePipeline （例如，***MyJenkinsProviderName***）。确保此名称是唯一的且容易记住。在此教程的后面部分，当您向管道添加生成操作时，会用到这个名称，并且在添加测试操作时，会再次用到它。

> **Note**
>
> 此操作名称必须符合中操作的命名要求 CodePipeline。有关更多信息，请参阅 中的配额 AWS CodePipeline。

12. 在构建触发器中，清除所有复选框，然后选择轮询 SCM。在计划中，输入五个星号并用空格隔开，如下所示：

```
* * * * *
```

这 CodePipeline 每分钟进行一次民意调查。

13. 在构建中，选择添加构建步骤。选择执行 Shell（Amazon Linux、RHEL 或 Ubuntu Server）执行批处理命令 (Windows Server)，然后输入以下内容：

```
rake
```

> **Note**
>
> 确保您的环境配置有运行 rake 所需的变量和设置，否则，生成会失败。

14. 选择“添加生成后期操作”，然后选择“AWS CodePipeline 发布者”。选择添加，然后在构建输出位置中，将位置留空。此配置是默认的，会在生成过程结束时创建一个压缩文件。
15. 选择保存以保存您的 Jenkins 项目。

步骤 2：在中创建管道 CodePipeline

在这部分教程中，您将使用创建管道向导来创建管道。

创建 CodePipeline 自动发布流程

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

2. 如有必要，请使用区域选择器将区域更改为您的管道资源所在的区域。例如，如果您在 us-east-2 中为上一教程创建了资源，请确保将区域选择器设置为“美国东部（俄亥俄州）”。

   有关可用区域和终端节点的更多信息 CodePipeline，请参阅终AWS CodePipeline 端节点和配额。
3. 在欢迎页面、入门页面或管道页面上，选择创建管道。
4. 在步骤 1：选择管道设置页面上，在管道名称中，输入管道的名称。
5. 在本教程中，为管道类型选择 V1。也可以选择 V2；但请注意，不同管道类型具有不同的特性和价格。有关更多信息，请参阅 管道类型。
6. 在服务角色中，选择新建服务角色 CodePipeline 以允许在 IAM 中创建服务角色。
7. 将高级设置中的各项设置保留为默认值，然后选择下一步。
8. 在 “步骤 2：添加源舞台” 页面上，在源提供程序中，选择GitHub。
9. 在连接下，选择一个现有连接或创建一个新连接。要创建或管理 GitHub 源操作的连接，请参阅GitHub 连接。
10. 在步骤 3: 添加构建阶段中，选择添加 Jenkins。在提供者名称中，输入您在 Jenkins CodePipeline 插件中提供的操作的名称（例如 *MyJenkinsProviderName*）。此名称必须与 Jenkins CodePipeline 插件中的名称完全匹配。在服务器 URL 中，输入在其中安装 Jenkins 的 EC2 实例的 URL。在项目名称中，输入您在 Jenkins 中创建的项目的名称，例如 *MyDemoProject*，然后选择下一步。
11. 在步骤 4：添加部署阶段中，重复使用您在中创建的 CodeDeploy 应用程序和部署组教程：创建一个简单的管道（S3 存储桶）。在部署提供商中，选择CodeDeploy。在应用程序名称中，输入 **CodePipelineDemoApplication**，或选择刷新按钮，然后从列表中选择应用程序名称。在部署组中，输入 **CodePipelineDemoFleet** 或者从列表中选择它，然后选择下一步。

    > **Note**
    >
    > 您可以使用自己的 CodeDeploy 资源或创建新的资源，但可能会产生额外费用。

12. 在步骤 5：审核中，查看信息，然后选择创建管道。
13. 管道会自动启动示例并通过管道运行示例。当管道将 Haml 示例构建为 HTML 并将其部署到部署中的每个 Amazon EC2 实例时，您可以查看进度以及成功和失败消息。 CodeDeploy

步骤 3：向管道中添加另一个阶段

现在，您将添加一个测试阶段，然后向该阶段添加测试操作，该操作使用示例中包含的 Jenkins 测试来确定网页是否具有任何内容。该测试仅用于演示目的。

> **Note**
>
> 如果您不想向管道中添加另一个阶段，则可以向管道的 Staging 阶段添加一个测试操作，该操作可以在部署操作之前或之后。

向管道中添加测试阶段

主题

- 查找实例的 IP 地址
- 创建一个 Jenkins 项目用于测试部署
- 创建第四个阶段

查找实例的 IP 地址

验证您将代码部署到的实例的 IP 地址

1. 当管道状态显示为成功后，在 Staging 阶段的状态区域中，选择详细信息。
2. 在部署详细信息部分，在实例 ID 中，选择成功部署的一个实例的实例 ID。
3. 复制实例的 IP 地址（例如，*192.168.0.4*）。您将在 Jenkins 测试中用到此 IP 地址。

创建一个 Jenkins 项目用于测试部署

创建 Jenkins 项目

1. 在您将 Jenkins 安装到的实例中，打开 Jenkins，然后从主页中选择新项目。
2. 在项目名称中，输入 Jenkins 项目的名称（例如，*MyTestProject*）。选择自由式项目，然后选择确定。

   > **Note**
   >
   > 确保您的项目名称符合 CodePipeline 要求。有关更多信息，请参阅 中的配额 AWS CodePipeline。

3. 在项目的配置页中，选择如有必要，执行并发构建复选框。在源代码管理中，选择 AWS CodePipeline。如果您已在 EC2 实例上安装了 Jenkins，并使用您为 AWS CLI 与 Jenkins 之间的 CodePipeline 集成而创建的 IAM 用户的配置文件配置了，请将所有其他字段留空。

> **Important**
>
> 如果您正在配置 Jenkins 项目，但该项目未安装在 Amazon EC2 实例上，或者该项目安装在运行 Windows 操作系统的 EC2 实例上，请根据您的代理主机和端口设置填写所需的字段，并提供您为在 Jenkins 和之间集成而配置的 IAM 用户或角色的证书。 CodePipeline

4. 选择高级，然后在目录中，选择测试。
5. 在 Prov id er 中，输入与生成项目相同的名称（例如，*MyJenkinsProviderName*）。在本教程的稍后部分，您在向您的管道添加测试操作时将用到此名称。

> **Note**
>
> 此名称必须符合操作的 CodePipeline 命名要求。有关更多信息，请参阅 中的配额 AWS CodePipeline。

6. 在构建触发器中，清除所有复选框，然后选择轮询 SCM。在计划中，输入五个星号并用空格隔开，如下所示：

```
* * * * *
```

这 CodePipeline 每分钟进行一次民意调查。

7. 在构建中，选择添加构建步骤。如果要部署到 Amazon Linux、RHEL 或 Ubuntu Server 实例，请选择执行 Shell。然后输入以下内容，其中 IP 地址是您之前复制的 EC2 实例的地址：

```
TEST_IP_ADDRESS=192.168.0.4 rake test
```

如果您要部署到 Windows Server 实例，请选择执行批处理命令，然后输入以下内容，其中 IP 地址是指您前面复制的 EC2 实例的地址：

```
set TEST_IP_ADDRESS=192.168.0.4 rake test
```

> **Note**
>
> 测试假定为默认端口 80。如果您要指定一个不同的端口，请添加一个测试端口语句，如下所示：

```
TEST_IP_ADDRESS=192.168.0.4 TEST_PORT=8000 rake test
```

8. 选择 “添加生成后期操作”，然后选择 “AWS CodePipeline 发布者”。不要选择添加。
9. 选择保存以保存您的 Jenkins 项目。

创建第四个阶段

向您的管道添加一个包括 Jenkins 测试操作的阶段

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在名称中，选择您创建的管道的名称 MySecondPipeline。
3. 在管道详细信息页中，选择编辑。
4. 在编辑页面上，选择 + 阶段，紧随构建阶段之后添加一个阶段。
5. 在新阶段的名称字段中，输入名称（例如 **Testing**），然后选择 + 添加操作组。
6. 在 “操作名称” 中，输入 *MyJenkinsTest-Action*。在测试提供者中，选择您在 Jenkins 中指定的提供者名称（例如，*MyJenkinsProviderName*）。在项目名称中，输入您在 Jenkins 中创建的项目的名称（例如，*MyTestProject*）。在 “输入工件” 中，从 Jenkins 版本中选择默认名称为的工件 *BuildArtifact*，然后选择 “完成”。

   > **Note**
   >
   > 由于 Jenkins 测试操作对 Jenkins 构建步骤中构建的应用程序起作用，请为输入构件使用构建构件来执行测试操作。

   有关输入和输出构件以及管道结构的更多信息，请参阅 CodePipeline 管道结构参考。
7. 在编辑页中，选择保存管道更改。在保存管道更改对话框中，选择保存并继续。
8. 虽然新阶段已添加到您的管道中，但该阶段会显示为 No executions yet 状态，因为更改并不会触发管道的另一次运行。要通过修订后的管道运行示例，在管道详细信息页中，选择发布更改。

   管道视图会显示管道中的阶段和操作以及经历这四个阶段的修订的状态。管道完成所有阶段需要的时间将取决于项目的大小、生成和测试操作的复杂程度以及其他因素。

步骤 4：清理资源

完成本教程之后，您应该删除管道及其使用的资源，以避免为继续使用这些资源付费。如果您不打算继续使用 CodePipeline，请删除管道，然后删除 CodeDeploy 应用程序及其关联的 Amazon EC2 实例，最后删除用于存储项目的 Amazon S3 存储桶。如果您不打算继续使用其他资源，例如 GitHub 存储库，则还应考虑是否删除它们。

清理本教程中使用的资源

1. 在您本地的 Linux、macOS 或 Unix 计算机上打开终端会话或在您本地的 Windows 计算机上打开命令提示符，并运行 delete-pipeline 命令，删除您创建的管道。对于 **MySecondPipeline**，您可以输入以下命令：

   ```
   aws codepipeline delete-pipeline --name "MySecondPipeline"
   ```

   该命令不返回任何内容。
2. 要清理 CodeDeploy 资源，请按照清理中的说明进行操作。
3. 要清理实例资源，请删除您在其中安装 Jenkins 的 EC2 实例。有关更多信息，请参阅清除您的实例。
4. 如果您不打算创建更多管道或 CodePipeline 再次使用，请删除用于存储管道项目的 Amazon S3 存储桶。要删除存储桶，请按照删除存储桶中的说明进行操作。
5. 如果您不打算将其他资源再次用于该管道，请考虑根据该特定资源的指导删除这些资源。例如，如果要删除 GitHub 存储库，请按照在 GitHub 网站上删除存储库中的说明进行操作。

教程：设置 CloudWatch 事件规则以接收有关管道状态更改的电子邮件通知

在中设置管道后 AWS CodePipeline，您可以设置 CloudWatch 事件规则，以便在管道的执行状态或管道中的阶段或操作发生变化时发送通知。有关使用 CloudWatch 事件设置管道状态更改通知的更多信息，请参阅监视 CodePipeline 事件。

在本教程中，您配置一个通知以在管道状态变为 FAILED 时发送电子邮件。本教程在创建 CloudWatch 事件规则时使用输入转换器方法。它转换消息架构详细信息以在用户可读的文本中传送消息。

> **Note**
>
> 在为本教程创建资源（例如 Amazon SNS 通知和 CloudWatch 事件规则）时，请确保在与您的管道相同的 AWS 区域创建这些资源。

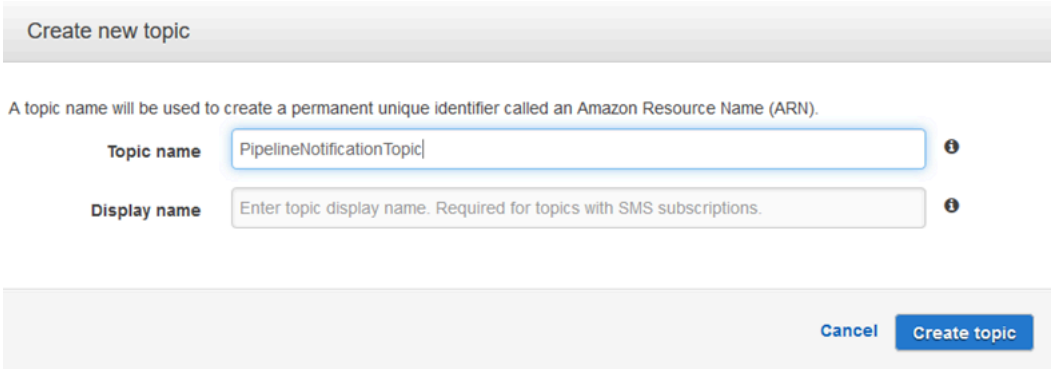
主题

步骤 1：使用 Amazon SNS 设置电子邮件通知

Amazon SNS 协调使用的主题以将消息传送到订阅端点或客户端。可以使用 Amazon SNS 创建一个通知主题，然后使用您的电子邮件地址订阅该主题。Amazon SNS 主题将作为目标添加到您的 CloudWatch 事件规则中。有关更多信息，请参阅《Amazon Simple Notification Service 开发人员指南》https://docs.aws.amazon.com/sns/latest/dg/。

在 Amazon SNS 中创建或标识主题。 CodePipeline 将使用 CloudWatch 活动通过 Amazon SNS 向该主题发送通知。要创建主题，请执行以下操作:

1. 打开亚马逊 SNS 控制台，网址为 https://console.aws.amazon.com/sns。
2. 选择创建主题。
3. 在 Create new topic (创建新主题) 对话框中，为 Topic name (主题名称) 键入主题名（例如 **`PipelineNotificationTopic`**）。



4. 选择创建主题。

   有关更多信息，请参阅 Amazon SNS 开发者指南 中的创建主题。

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. 在 Amazon SNS 控制台中，从主题列表中，选中新主题旁边的复选框。选择操作，然后选择“订阅主题”。
2. 在创建订阅对话框中，确认在主题 ARN 中显示一个 ARN。
3. 对于协议，选择电子邮件。
4. 对于终端节点，请键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。要接收电子邮件通知，收件人必须在该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 开发者指南 中的订阅主题。

步骤 2：创建规则并将 SNS 主题添加为目标

创建以 CodePipeline 作为 CloudWatch 事件源的事件通知规则。

1. 打开 CloudWatch 控制台，网址为 https://console.aws.amazon.com/cloudwatch/。
2. 在导航窗格中，选择 Events（事件）。
3. 选择创建规则。在事件源下面，选择 AWS CodePipeline。对于事件类型，请选择管道执行状态更改。
4. 选择特定状态，然后选择 **FAILED**。
5. 对于事件模式预览窗格，请选择编辑以打开 JSON 编辑器。添加 **pipeline** 参数以及管道名称，如以下示例中名为“myPipeline”的管道所示。

   您可以复制此处的事件模式并将其粘贴到控制台中：

```
{
  "source": [
    "aws.codepipeline"
  ],
  "detail-type": [
    "CodePipeline Pipeline Execution State Change"
  ],
  "detail": {
    "state": [
```

```
      "FAILED"
    ],
    "pipeline": [
      "myPipeline"
    ]
  }
}
```

6. 对于目标，选择添加目标。
7. 在目标列表中，选择 SNS 主题。对于主题，请输入刚创建的主题。
8. 展开配置输入，然后选择输入转换器。
9. 在输入路径框中，键入以下键值对。

   ```
   { "pipeline" : "$.detail.pipeline" }
   ```

   在输入模板框中，键入以下内容：

   ```
   "The Pipeline <pipeline> has failed."
   ```

10. 选择 Configure details（配置详细信息）。
11. 在配置规则详细信息页中，键入一个名称和可选的描述。对于状态，请将已启用框保持选中状态。
12. 选择创建规则。
13. 确认现在 CodePipeline 正在发送构建通知。例如，检查您的收件箱中现在是否有构建通知电子邮件。
14. 要更改规则的行为，请在 CloudWatch 控制台中选择规则，然后选择操作、编辑。编辑该规则，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 控制台中选择规则，然后选择操作，禁用。

    要删除规则，请在 CloudWatch 控制台中选择规则，然后选择操作，删除。

步骤 3：清理资源

完成本教程之后，您应该删除管道及其使用的资源，以避免为继续使用这些资源付费。

有关如何清理 SNS 通知和删除亚马逊 CloudWatch 事件规则的信息，请参阅清理（取消订阅亚马逊 SNS 主题）和《亚马逊活动 API 参考》DeleteRule中的参考资料。 CloudWatch

教程：创建用于构建和测试您的 Android 应用程序的管道 AWS Device Farm

您可以使用 AWS CodePipeline 来配置持续集成流程，每次推送提交时都会在该流程中构建和测试您的应用程序。本教程介绍如何创建和配置管道，以便使用 GitHub 存储库中的源代码构建和测试您的 Android 应用程序。管道会检测到新 GitHub 提交的CodeBuild到来，然后使用构建应用程序，并使用 Device Far m对其进行测试。

> **Important**
>
> 在此过程中，您在管道中添加的许多操作都涉及在创建管道之前需要创建的 AWS 资源。 AWS 源操作的资源必须始终在您创建管道的同一 AWS 区域创建。例如，如果您在美国东部（俄亥俄州）地区创建管道，则您的 CodeCommit 存储库必须位于美国东部（俄亥俄州）区域。
> 您可以在创建管道时添加跨区域操作。 AWS 跨区域操作的资源必须位于您计划执行操作的同一 AWS 区域。有关更多信息，请参阅 在中添加跨区域操作 CodePipeline。

您可以使用现有的 Android 应用程序和测试定义或使用 Device Farm 提供的示例应用程序和测试定义进行此尝试。

> **Note**

开始之前

1. 登录 AWS Device Farm 控制台并选择 “创建新项目”。
2. 选择您的项目。在浏览器中，复制新项目的 URL。URL 包含项目 ID。
3. 复制并保留此项目 ID。您可以在中创建管道时使用它 CodePipeline。

   以下是项目的示例 URL。要提取项目 ID，请复制 `projects/` 之后的值。在此示例中，项目 ID 为 `eec4905f-98f8-40aa-9afc-4c1cfexample`。

   ```
   https://<region-URL>/devicefarm/home?region=us-west-2#/projects/
   eec4905f-98f8-40aa-9afc-4c1cfexample/runs
   ```

配置 CodePipeline 为使用您的 Device Farm 测试

1. 在应用程序代码的根目录buildspec.yml中添加并提交一个名为的文件，然后将其推送到存储库。 CodeBuild 使用此文件执行构建应用程序所需的命令和访问构件。

```
version: 0.2

phases:
  build:
    commands:
      - chmod +x ./gradlew
      - ./gradlew assembleDebug
artifacts:
  files:
     - './android/app/build/outputs/**/*.apk'
  discard-paths: yes
```

2. （可选）如果使用 Calabash 或 Appium 测试应用程序，请将测试定义文件添加到存储库。在后续步骤中，可将 Device Farm 配置为使用定义来执行测试套件。

   如果使用 Device Farm 内置测试，可跳过此步骤。

3. 要创建管道并添加一个源阶段，请执行以下操作：

   a. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。

   b. 选择创建管道。在步骤 1：选择管道设置页面上，在管道名称中，输入管道的名称。

   c. 在本教程中，为管道类型选择 V1。也可以选择 V2；但请注意，不同管道类型具有不同的特性和价格。有关更多信息，请参阅 管道类型。

   d. 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。您还可以选择使用现有服务角色（如果已有）。

> **Note**
>
> 如果您使用在 2018 年 7 月之前创建的 CodePipeline 服务角色，则需要为 Device Farm 添加权限。要执行此操作，请打开 IAM 控制台，找到该角色，然后向该角色的策略添加以下权限。有关更多信息，请参阅 向 CodePipeline 服务角色添加权限。

```
{
     "Effect": "Allow",
```

```
      "Action": [
         "devicefarm:ListProjects",
         "devicefarm:ListDevicePools",
         "devicefarm:GetRun",
         "devicefarm:GetUpload",
         "devicefarm:CreateUpload",
         "devicefarm:ScheduleRun"
      ],
      "Resource": "*"
}
```

e. 将高级设置中的各项设置保留为默认值，然后选择下一步。

f. 在 “步骤 2：添加源舞台” 页面上，在源提供程序中，选择GitHub。

g. 在连接下，选择一个现有连接或创建一个新连接。要创建或管理 GitHub 源操作的连接，请参阅GitHub 连接。

h. 在存储库中选择源存储库。

i. 在分支中选择要使用的分支。

j. 保留源操作的其余默认值。选择下一步。

4. 在添加构建阶段，添加一个构建阶段：

a. 在构建提供程序中，选择 AWS CodeBuild。允许区域默认为管道区域。

b. 选择创建项目。

c. 在项目名称中，输入此构建项目的名称。

d. 在环境映像中，选择托管映像。对于操作系统，选择 Ubuntu。

e. 对于运行时，选择标准。对于映像，选择 aws/codebuild/standard:5.0。

CodeBuild 使用此安装了 Android Studio 的操作系统映像来构建您的应用程序。

f. 对于服务角色，请选择您现有的 CodeBuild 服务角色或创建一个新的服务角色。

g. 对于构建规范，选择使用 buildspec 文件。

h. 选择 “继续” CodePipeline。这将返回到 CodePipeline控制台并创建一个使用存储库`buildspec.yml`中的进行配置的 CodeBuild 项目。该构建项目使用服务角色来管理 AWS 服务 权限。此步骤可能需要几分钟时间。

i. 选择下一步。

5. 在步骤 4：添加部署阶段页面上，选择跳过部署阶段，并通过再次选择跳过接受警告消息。选择下一步。

6. 在步骤 5：审核中，选择创建管道。您应看到一个显示源和构建阶段的示意图。
7. 将 Device Farm 测试操作添加到管道：

   a. 在右上角，选择编辑。
   b. 在示意图底部，选择 + 添加阶段。在阶段名称中，输入名称，例如 **Test**。
   c. 选择 + 添加操作组。
   d. 在操作名称中输入名称。
   e. 在操作提供程序中，选择 AWS Device Farm。允许区域默认为管道区域。
   f. 在输入构件中，选择与测试阶段之前的那个阶段的输出构件相匹配的输入构件，例如 `BuildArtifact`。

      在 AWS CodePipeline 控制台中，将鼠标悬停在管道图中的信息图标上，可以找到每个阶段的输出工件的名称。如果您的管道直接从 S ourc e 阶段测试您的应用程序，请选择SourceArtifact。如果管道包含 “构建” 阶段，请选择BuildArtifact。
   g. 在中 ProjectId，输入您的 Device Farm 项目 ID。使用本教程开头的步骤来检索您的项目 ID。
   h. 在中 DevicePoolArn，输入设备池的 ARN。要获取项目可用的设备池 ARN，包括热门设备的 ARN，请 AWS 使用 CLI 输入以下命令：

      ```
      aws devicefarm list-device-pools --arn arn:aws:devicefarm:us-
      west-2:account_ID:project:project_ID
      ```
   i. 在中 AppType，输入安卓。

      以下是的有效值列表 AppType：

      - iOS
      - Android
      - Web
   j. 在 App 中，输入已编译的应用程序包的路径。该路径相对于测试阶段的输入构件的根。通常，此路径类似于 `app-release.apk`。
   k. 在中 TestType，输入您的测试类型，然后在测试中输入测试定义文件的路径。路径相对于测试的输入项目的根。

      以下是的有效值列表 TestType：

      - APPIUM_JAVA_JUNIT
      - APPIUM_JAVA_TESTNG

- APPIUM_NODE
- APPIUM_RUBY
- APPIUM_PYTHON
- APPIUM_WEB_JAVA_JUNIT
- APPIUM_WEB_JAVA_TESTNG
- APPIUM_WEB_NODE
- APPIUM_WEB_RUBY
- APPIUM_WEB_PYTHON
- BUILTIN_FUZZ
- INSTRUMENTATION
- XCTEST
- XCTEST_UI

> **Note**
>
> 不支持自定义环境节点。

l. 在其余字段中，提供适合测试和应用程序类型的配置。

m. （可选）在高级中，为测试运行提供配置信息。

n. 选择保存。

o. 在所编辑的阶段上，选择完成。在 AWS CodePipeline 窗格中，选择保存，然后选择警告消息上的保存。

p. 要提交所做的更改并开始管道构建，请选择发布更改，然后选择发布。

教程：创建用于测试 iOS 应用的管道 AWS Device Farm

您可以使用 AWS CodePipeline 来轻松配置持续集成流程，每次源存储桶更改时，都会在该流程中测试您的应用程序。本教程介绍如何创建和配置管道以测试从 S3 存储桶中构建的 iOS 应用程序。该管道通过 Amazon CloudWatch Events 检测已保存更改的到来，然后使用 De vice Farm 来测试构建的应用程序。

> **Important**
>
> 在此过程中，您在管道中添加的许多操作都涉及在创建管道之前需要创建的 AWS 资源。 AWS 源操作的资源必须始终在您创建管道的同一 AWS 区域创建。例如，如果您在美国东部（俄亥俄州）地区创建管道，则您的 CodeCommit 存储库必须位于美国东部（俄亥俄州）区域。
> 您可以在创建管道时添加跨区域操作。 AWS 跨区域操作的资源必须位于您计划执行操作的同一 AWS 区域。有关更多信息，请参阅 在中添加跨区域操作 CodePipeline。

您可以使用现有的 iOS 应用程序或使用示例 iOS 应用程序进行此测试。

> **Note**

开始之前

1. 登录 AWS Device Farm 控制台并选择 “创建新项目”。
2. 选择您的项目。在浏览器中，复制新项目的 URL。URL 包含项目 ID。
3. 复制并保留此项目 ID。您可以在中创建管道时使用它 CodePipeline。

   以下是项目的示例 URL。要提取项目 ID，请复制 `projects/` 之后的值。在此示例中，项目 ID 为 `eec4905f-98f8-40aa-9afc-4c1cfexample`。

   ```
   https://<region-URL>/devicefarm/home?region=us-west-2#/projects/
   eec4905f-98f8-40aa-9afc-4c1cfexample/runs
   ```

配置 CodePipeline 为使用您的 Device Farm 测试（Amazon S3 示例）

1. 创建或使用已启用版本控制的 S3 存储桶。按照步骤 1：为您的应用程序创建一个 S3 存储桶中的说明创建 S3 存储桶。
2. 在桶的 Amazon S3 控制台中，选择上传，然后按照说明上传 .zip 文件。

   您的示例应用程序必须打包在 .zip 文件中。
3. 要创建管道并添加一个源阶段，请执行以下操作：

a. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。

b. 选择创建管道。在步骤 1：选择管道设置页面上，在管道名称中，输入管道的名称。

c. 在本教程中，为管道类型选择 V1。也可以选择 V2；但请注意，不同管道类型具有不同的特性和价格。有关更多信息，请参阅 管道类型。

d. 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。您还可以选择使用现有服务角色（如果已有）。

> **Note**
>
> 如果您使用在 2018 年 7 月之前创建的 CodePipeline 服务角色，则必须为 Device Farm 添加权限。要执行此操作，请打开 IAM 控制台，找到该角色，然后向该角色的策略添加以下权限。有关更多信息，请参阅 向 CodePipeline 服务角色添加权限。
>
> ```
> {
>      "Effect": "Allow",
>      "Action": [
>         "devicefarm:ListProjects",
>         "devicefarm:ListDevicePools",
>         "devicefarm:GetRun",
>         "devicefarm:GetUpload",
>         "devicefarm:CreateUpload",
>         "devicefarm:ScheduleRun"
>      ],
>      "Resource": "*"
> }
> ```

e. 将高级设置中的各项设置保留为默认值，然后选择下一步。

f. 在步骤 2：添加源阶段页面上，在源提供程序 中，选择 Amazon S3。

g. 在 Amazon S3 位置中，输入 .zip 文件的桶（例如 `my-storage-bucket`）和对象键（例如 `s3-ios-test-1.zip`）。

h. 选择下一步。

4. 在构建中，为管道创建一个占位符构建阶段。这样，可以在向导中创建管道。在使用向导创建两阶段管道后，将不再需要此占位符生成阶段。完成该管道后，将在步骤 5 中删除第二个阶段并添加新的测试阶段。

a. 在构建提供程序中，选择添加 Jenkins。此生成选择是一个占位符。不会使用。

b. 在提供商名称中输入名称。该名称是一个占位符。不会使用。

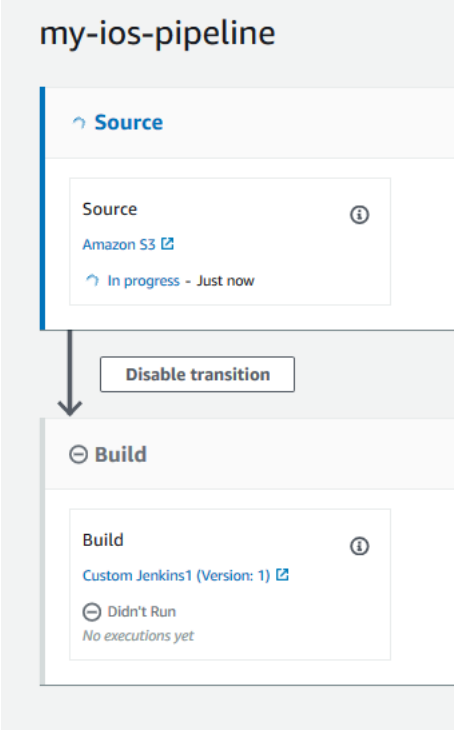
c. 在服务器 URL 中输入文本。该文本是一个占位符。不会使用。

d. 在项目名称中输入名称。该名称是一个占位符。不会使用。

e. 选择下一步。

f. 在步骤 4：添加部署阶段页面上，选择跳过部署阶段，并通过再次选择跳过接受警告消息。

g. 在步骤 5：审核中，选择创建管道。您应看到一个显示源和构建阶段的示意图。



5. 将 Device Farm 测试操作添加到管道，如下所述：

a. 在右上角，选择编辑。

b. 选择编辑阶段。选择删除。此操作将删除不再需要用于管道创建的占位符阶段。

c. 在示意图底部，选择 + 添加阶段。

d. 在“阶段名称”中，输入阶段名称，例如“测试”，然后选择添加阶段。

e. 选择 + 添加操作组。

f. 在操作名称中，输入一个名称，例如 DeviceFarmTest。

g. 在操作提供程序中，选择 AWS Device Farm。允许区域默认为管道区域。

h. 在输入构件中，选择与测试阶段之前的那个阶段的输出构件相匹配的输入构件，例如 `SourceArtifact`。

在 AWS CodePipeline 控制台中，将鼠标悬停在管道图中的信息图标上，可以找到每个阶段的输出工件的名称。如果您的管道直接从 S ourc e 阶段测试您的应用程序，请选择SourceArtifact。如果管道包含 “构建” 阶段，请选择BuildArtifact。

i. 在中 ProjectId，选择您的 Device Farm 项目 ID。使用本教程开头的步骤来检索您的项目 ID。

j. 在中 DevicePoolArn，输入设备池的 ARN。要获取项目可用的设备池 ARN，包括热门设备的 ARN，请 AWS 使用 CLI 输入以下命令：

```
aws devicefarm list-device-pools --arn arn:aws:devicefarm:us-
west-2:account_ID:project:project_ID
```

k. 在中 AppType，输入 iOS。

以下是的有效值列表 AppType：

- iOS
- Android
- Web

l. 在 App 中，输入已编译的应用程序包的路径。该路径相对于测试阶段的输入构件的根。通常，此路径类似于 `ios-test.ipa`。

m. 在中 TestType，输入您的测试类型，然后在测试中输入测试定义文件的路径。路径相对于测试的输入项目的根。

如果您正在使用其中一个 Device Farm 内置测试，请输入 Device Farm 项目中配置的测试类型，例如 BUILTIN_FUZZ。在中 FuzzEventCount，输入以毫秒为单位的时间，例如 6000。在中 FuzzEventThrottle，输入以毫秒为单位的时间，例如 50。

如果没有使用其中一个 Device Farm 内置测试，请选择测试类型，然后在测试中输入测试定义文件的路径。路径相对于测试的输入项目的根。

以下是的有效值列表 TestType：

- APPIUM_JAVA_JUNIT
- APPIUM_JAVA_TESTNG
- APPIUM_NODE
- APPIUM_RUBY
- APPIUM_PYTHON

- APPIUM_WEB_JAVA_JUNIT
- APPIUM_WEB_JAVA_TESTNG
- APPIUM_WEB_NODE
- APPIUM_WEB_RUBY
- APPIUM_WEB_PYTHON
- BUILTIN_FUZZ
- INSTRUMENTATION
- XCTEST
- XCTEST_UI

> **Note**
>
> 不支持自定义环境节点。

n. 在其余字段中，提供适合测试和应用程序类型的配置。

o. （可选）在高级中，为测试运行提供配置信息。

p. 选择保存。

q. 在所编辑的阶段上，选择完成。在 AWS CodePipeline 窗格中，选择 “保存”，然后在警告消息上选择 “保存”。

r. 要提交所做的更改并开始管道执行，请选择发布更改，然后选择发布。

教程：创建部署到 Service Catalog 的管道

Service Catalog 使您能够基于 AWS CloudFormation 模板创建和配置产品。本教程向您展示如何创建和配置管道，以便将您的产品模板部署到 Service Catalog，并交付您在源存储库（已在 GitHub CodeCommit、或 Amazon S3 中创建）中所做的更改。

> **Note**
>
> 当 Amazon S3 是管道的源提供程序时，您必须将所有源文件打包为单个 .zip 文件上传到桶。否则，源操作将失败。

首先，您在 Service Catalog 中创建一个产品，然后在 AWS CodePipeline中创建管道。本教程提供两个设置部署配置的选项：

- 在 Service Catalog 中创建产品并将模板文件上传到您的源存储库。在 CodePipeline 控制台中提供产品版本和部署配置（无需单独的配置文件）。请参阅 选项 1：在不使用配置文件的情况下部署到 Service Catalog。

  > **Note**
  >
  > 可以按 YAML 或 JSON 格式创建模板文件。

- 在 Service Catalog 中创建产品并将模板文件上传到您的源存储库。在单独的配置文件中提供产品版本和部署配置。请参阅 选项 2：使用配置文件部署到 Service Catalog。

选项 1：在不使用配置文件的情况下部署到 Service Catalog

在此示例中，您上传了 S3 存储桶的示例 AWS CloudFormation 模板文件，然后在 Service Catalog 中创建您的产品。接下来，创建管道并在 CodePipeline 控制台中指定部署配置。

步骤 1：将示例模板文件上传到源存储库

1. 打开文本编辑器。通过将以下内容粘贴到文件中创建一个示例模板。将该文件保存为 `S3_template.json`。

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "CloudFormation Sample Template S3_Bucket: Sample template showing
 how to create a privately accessible S3 bucket. **WARNING** This template creates
 an S3 bucket. You will be billed for the resources used if you create a stack from
 this template.",
  "Resources": {
    "S3Bucket": {
      "Type": "AWS::S3::Bucket",
      "Properties": {}
    }
  },
  "Outputs": {
    "BucketName": {
      "Value": {
        "Ref": "S3Bucket"
      },
```

```
      "Description": "Name of Amazon S3 bucket to hold website content"
    }
  }
}
```

此模板 AWS CloudFormation 允许创建可由 Service Catalog 使用的 S3 存储桶。

2. 将 S3_template.json文件上传到 AWS CodeCommit 存储库中。

步骤 2：在 Service Catalog 中创建产品

1. 以 IT 管理员身份登录到 Service Catalog 控制台，转到产品页面，然后选择上传新产品。
2. 在上传新产品页面上，完成以下操作：
   a. 在产品名称中，输入您要用于新产品的名称。
   b. 在描述中，输入产品目录描述。此说明显示在产品列表中，可帮助用户选择正确的产品。
   c. 在提供者中，输入 IT 部门或管理员的名称。
   d. 选择下一步。
3. （可选）在输入支持详细信息中，输入产品支持联系人信息，然后选择下一步。
4. 在版本详细信息中，完成以下内容：
   a. 选择上传模板文件。浏览您的 S3_template.json 文件并将其上传。
   b. 在版本标题中，输入产品版本名称（例如 **devops S3 v2**）。
   c. 在描述中，输入区分此版本与其他版本的详细信息。
   d. 选择下一步。
5. 在审核页面上，确认信息正确，然后选择创建。
6. 在产品页面上，在浏览器中复制新产品的 URL。这包含产品 ID。复制并保留此产品 ID。您可以在中创建管道时使用它 CodePipeline。

   以下是名为 my-product 的产品的 URL。要提取产品 ID，请复制等号（=）和符号（&）之间的值。在此示例中，产品 ID 为 prod-example123456。

```
https://<region-URL>/servicecatalog/home?region=<region>#/admin-products?
productCreated=prod-example123456&createdProductTitle=my-product
```

> **Note**
>
> 复制您的产品的 URL，然后再离开该页面。离开此页面后，必须使用 CLI 获取产品 ID。

几秒钟后，您的产品会显示在产品页面上。您可能需要刷新浏览器才能在列表中看到此产品。

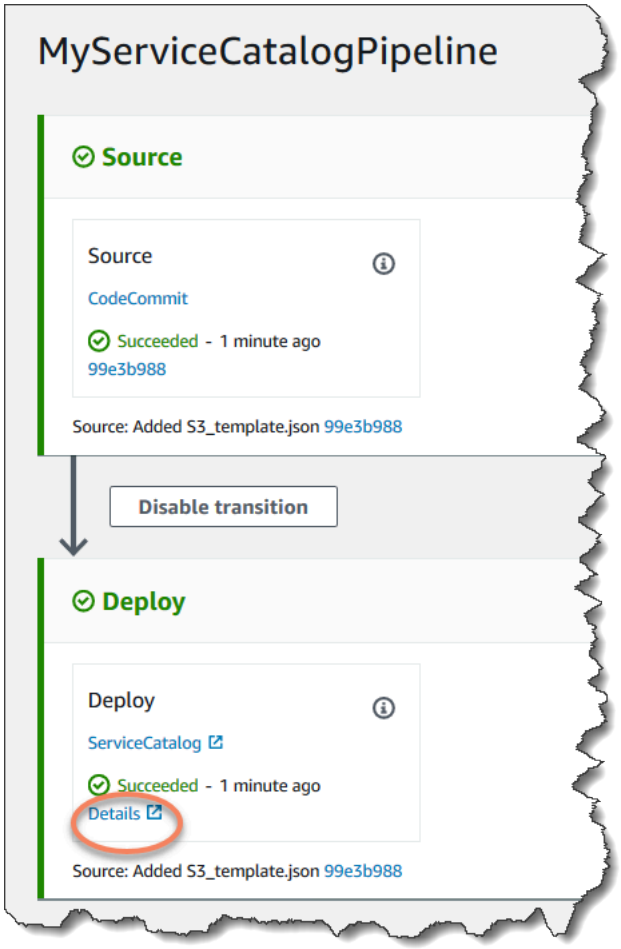
步骤 3：创建管道

1. 要命名您的管道并选择用于管道的参数，请执行以下操作：
   a. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。
   b. 选择开始使用。选择创建管道，然后输入管道名称。
   c. 在本教程中，为管道类型选择 V1。也可以选择 V2；但请注意，不同管道类型具有不同的特性和价格。有关更多信息，请参阅 管道类型。
   d. 在服务角色中，选择新建服务角色 CodePipeline 以允许在 IAM 中创建服务角色。
   e. 将高级设置中的各项设置保留为默认值，然后选择下一步。
2. 要添加源阶段，请执行以下操作：
   a. 在源提供程序中，选择 AWS CodeCommit。
   b. 在存储库名称和分支名称中，输入您要用于自己的源操作的存储库和分支。
   c. 选择下一步。
3. 在添加构建阶段中，选择跳过构建阶段，然后再次选择跳过以接受警告消息。
4. 在添加部署阶段中，完成以下操作：
   a. 在部署提供商中，选择AWS Service Catalog。
   b. 对于部署配置，请选择输入部署配置。
   c. 在产品 ID 中，粘贴您从 Service Catalog 控制台复制的产品 ID。
   d. 在模板文件路径中，输入存储模板文件的位置的相对路径。
   e. 在产品类型中，选择 AWS CloudFormation 模板。
   f. 在产品版本名称中，输入您在 Service Catalog 中指定的产品版本的名称。如果您想将模板更改部署到新的产品版本，请输入尚未用于同一产品中任何先前产品版本的产品版本名称。
   g. 对于输入项目，选择源输入项目。

h. 选择下一步。

5. 在审核中，审核您的管道设置，然后选择创建。

6. 管道成功运行后，在部署阶段上选择详细信息。这会在 Service Catalog 中打开您的产品。



7. 在您的产品信息中，选择您的版本名称以打开产品模板。查看模板部署。

步骤 4：在 Service Catalog 中推送更改并验证产品

1. 在 CodePipeline 控制台中查看您的管道，然后在源代码舞台上选择 Deta ils。您的源 AWS CodeCommit 存储库将在控制台中打开。选择编辑，然后在文件中进行更改（例如，更改描述）。

```
"Description": "Name of Amazon S3 bucket to hold and version website content"
```

2. 提交并推送您的更改。推送更改后，您的管道启动。当管道运行完成时，在部署阶段上选择详细信息以在 Service Catalog 中打开您的产品。

3. 在您的产品信息中，选择新的版本名称以打开产品模板。查看部署的模板更改。

选项 2：使用配置文件部署到 Service Catalog

在此示例中，您上传了 S3 存储桶的示例 AWS CloudFormation 模板文件，然后在 Service Catalog 中创建您的产品。您还可以上传指定您的部署配置的单独配置文件。接下来，您可以创建您的管道，并指定部署文件的位置。

步骤 1：将示例模板文件上传到源存储库

1. 打开文本编辑器。通过将以下内容粘贴到文件中创建一个示例模板。将该文件保存为 `S3_template.json`。

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "CloudFormation Sample Template S3_Bucket: Sample template showing
 how to create a privately accessible S3 bucket. **WARNING** This template creates
 an S3 bucket. You will be billed for the resources used if you create a stack from
 this template.",
  "Resources": {
    "S3Bucket": {
      "Type": "AWS::S3::Bucket",
      "Properties": {}
    }
  },
  "Outputs": {
    "BucketName": {
      "Value": {
        "Ref": "S3Bucket"
      },
      "Description": "Name of Amazon S3 bucket to hold website content"
    }
  }
}
```

   此模板 AWS CloudFormation 允许创建可由 Service Catalog 使用的 S3 存储桶。

2. 将 `S3_template.json`文件上传到 AWS CodeCommit 存储库中。

步骤 2：创建产品部署配置文件

1. 打开文本编辑器。为您的产品创建配置文件。配置文件用于定义 Service Catalog 部署参数/首选项。创建管道时要使用此文件。

   此示例提供了“devops S3 v2”的 `ProductVersionName` 和 `MyProductVersionDescription` 的 `ProductVersionDescription`。如果您想将模板更改部署到新的产品版本，只需输入尚未用于同一产品中任何先前产品版本的产品版本名称。

   将该文件保存为 `sample_config.json`。

   ```
   {
       "SchemaVersion": "1.0",
       "ProductVersionName": "devops S3 v2",
       "ProductVersionDescription": "MyProductVersionDescription",
       "ProductType": "CLOUD_FORMATION_TEMPLATE",
       "Properties": {
           "TemplateFilePath": "/S3_template.json"
       }
   }
   ```

   此文件将在您的管道每次运行时为您创建产品版本信息。

2. 将 `sample_config.json`文件上传到 AWS CodeCommit 存储库中。确保您将此文件上载到您的源存储库。

步骤 3：在 Service Catalog 中创建产品

1. 以 IT 管理员身份登录到 Service Catalog 控制台，转到产品页面，然后选择上传新产品。
2. 在上传新产品页面上，完成以下操作：

   a. 在产品名称中，输入您要用于新产品的名称。

   b. 在描述中，输入产品目录描述。此说明出现在产品列表中，可帮助用户选择正确的产品。

   c. 在提供者中，输入 IT 部门或管理员的名称。

   d. 选择下一步。

3. （可选）在输入支持详细信息中，输入产品支持联系人信息，然后选择下一步。
4. 在版本详细信息中，完成以下内容：

   a. 选择上传模板文件。浏览您的 `S3_template.json` 文件并将其上传。

b. 在版本标题中，输入产品版本的名称（例如，“devops S3 v2”）。

c. 在描述中，输入区分此版本与其他版本的详细信息。

d. 选择下一步。

5. 在审核页面上，确认信息正确，然后选择确认并上传。

6. 在产品页面上，在浏览器中复制新产品的 URL。这包含产品 ID。复制并保留此产品 ID。您在中创建管道时使用 CodePipeline。

   以下是名为 `my-product` 的产品的 URL。要提取产品 ID，请复制等号（=）和符号（&）之间的值。在此示例中，产品 ID 为 `prod-example123456`。

   ```
   https://<region-URL>/servicecatalog/home?region=<region>#/admin-products?
   productCreated=prod-example123456&createdProductTitle=my-product
   ```

   > **Note**
   >
   > 复制您的产品的 URL，然后再离开该页面。离开此页面后，必须使用 CLI 获取产品 ID。

   几秒钟后，您的产品会显示在产品页面上。您可能需要刷新浏览器才能在列表中看到此产品。

步骤 4：创建管道

1. 要命名您的管道并选择用于管道的参数，请执行以下操作：

   a. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。

   b. 选择开始使用。选择创建管道，然后输入管道名称。

   c. 在服务角色中，选择新建服务角色 CodePipeline 以允许在 IAM 中创建服务角色。

   d. 将高级设置中的各项设置保留为默认值，然后选择下一步。

2. 要添加源阶段，请执行以下操作：

   a. 在源提供程序中，选择 AWS CodeCommit。

   b. 在存储库名称和分支名称中，输入您要用于自己的源操作的存储库和分支。

   c. 选择下一步。

3. 在添加构建阶段中，选择跳过构建阶段，然后再次选择跳过以接受警告消息。

4. 在添加部署阶段中，完成以下操作：

   a. 在部署提供商中，选择AWS Service Catalog。

   b. 选择使用配置文件。

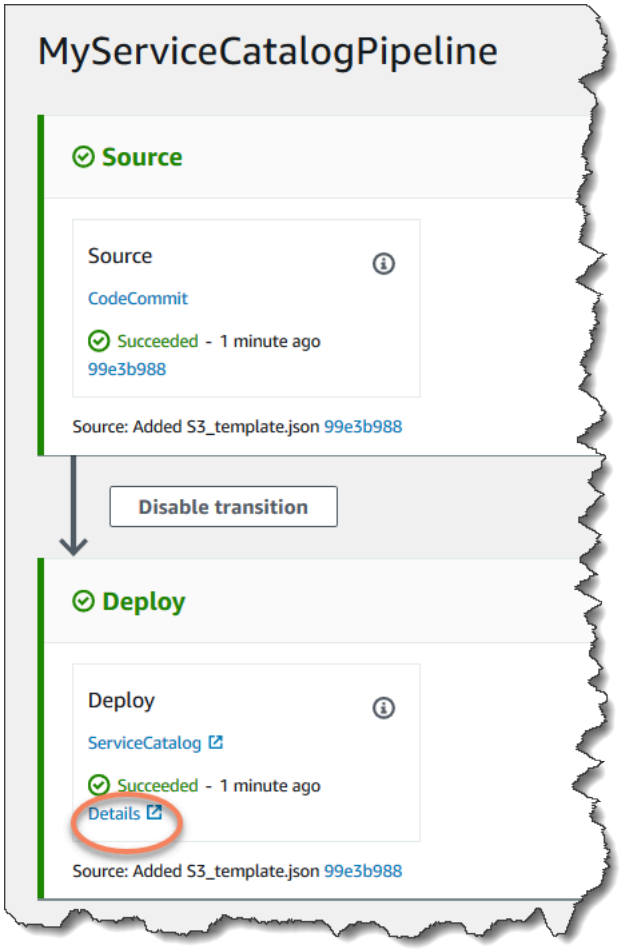
   c. 在产品 ID 中，粘贴您从 Service Catalog 控制台复制的产品 ID。

   d. 在配置文件路径中，输入存储库中配置文件的文件路径。

   e. 选择下一步。

5. 在审核中，审核您的管道设置，然后选择创建。

6. 管道成功运行后，在部署阶段上选择详细信息，以便在 Service Catalog 中打开您的产品。



7. 在您的产品信息中，选择您的版本名称以打开产品模板。查看模板部署。

步骤 5：在Service Catalog 中推送更改并验证产品

1. 在 CodePipeline 控制台中查看您的管道，然后在源代码舞台上选择 Deta ils。您的源 AWS CodeCommit 存储库将在控制台中打开。选择编辑，然后在文件中进行更改（例如，更改描述）。

   ```
   "Description": "Name of Amazon S3 bucket to hold and version website content"
   ```

2. 提交并推送您的更改。推送更改后，您的管道启动。当管道运行完成时，在部署阶段上选择详细信息以在 Service Catalog 中打开您的产品。
3. 在您的产品信息中，选择新的版本名称以打开产品模板。查看部署的模板更改。

教程：使用创建管道 AWS CloudFormation

这些示例提供了示例模板，允许您使用这些模板 AWS CloudFormation 来创建管道，该管道在每次源代码更改时都会将您的应用程序部署到您的实例。示例模板创建了一个可以在 AWS CodePipeline中查看的管道。该渠道通过 Amazon CloudWatch Events 检测已保存更改的到来。

主题

- 示例 1：使用创建 AWS CodeCommit 管道 AWS CloudFormation
- 示例 2：使用 AWS CloudFormation创建 Amazon S3 管道

示例 1：使用创建 AWS CodeCommit 管道 AWS CloudFormation

本演练向您展示如何使用 AWS CloudFormation 控制台创建基础架构，其中包括连接到 CodeCommit 源存储库的管道。在本教程中，您将使用提供的示例模板文件创建资源堆栈，其中包括您的项目存储、管道和更改检测资源，例如您的 Amazon Ev CloudWatch ents 规则。在中创建资源堆栈后 AWS CloudFormation，您可以在 AWS CodePipeline 控制台中查看您的管道。该管道是一个分为两个阶段的管道，包括 CodeCommit 源阶段和 CodeDeploy 部署阶段。

先决条件：

您必须已创建以下资源才能与 AWS CloudFormation 示例模板一起使用：

- 您必须已创建一个源存储库。您可以使用您在中创建的 AWS CodeCommit 存储库教程：创建简单的管道（CodeCommit存储库）。

- 您必须已创建 CodeDeploy 应用程序和部署组。您可以使用您在中创建的 CodeDeploy资源教程：创建简单的管道（CodeCommit存储库）。
- 选择以下链接之一下载用于创建管道的示例 AWS CloudFormation 模板文件：YAML | JSON

  解压缩该文件并将其放在您的本地计算机上。
- 下载 SampleApp_Linux.zip 示例应用程序文件。

在中创建您的管道 AWS CloudFormation

1. 从 SampleApp_Linux.zip 解压文件，然后将文件上传到您的 AWS CodeCommit 存储库。您必须将解压缩的文件上传到存储库的根目录。您可以按照第 2 步：向 CodeCommit 存储库添加示例代码中的说明将文件推送到您的存储库。
2. 打开 AWS CloudFormation 控制台并选择 “创建堆栈”。选择使用新资源（标准）。
3. 在指定模板下，选择上传模板。选中选择文件，然后从您的本地计算机选择模板文件。选择下一步。
4. 在堆栈名称中，输入管道的名称。将显示由示例模板指定的参数。输入以下参数：

   a. 在中 ApplicationName，输入您的 CodeDeploy应用程序的名称。

   b. 在中 BetaFleet，输入您的 CodeDeploy 部署组的名称。

   c. 在中 BranchName，输入要使用的存储库分支。

   d. 在中 RepositoryName，输入您的 CodeCommit 源存储库的名称。
5. 选择下一步。在接下来的页面上接受默认值，然后选择下一步。
6. 在 “能力” 中，选择 “我确认 AWS CloudFormation 可能会创建 IAM 资源”，然后选择 “创建堆栈”。
7. 在堆栈创建完成后，查看事件列表以检查是否存在任何错误。

   故障排除

   在中创建管道的 IAM 用户 AWS CloudFormation 可能需要额外的权限才能为管道创建资源。策略中需要以下权限才能 AWS CloudFormation 为 CodeCommit管道创建所需的 Amazon E CloudWatch vents 资源：

```
{
     "Effect": "Allow",
     "Action": [
        "events:PutRule",
        "events:PutEvents",
```

```
            "events:PutTargets",
            "events:DeleteRule",
            "events:RemoveTargets",
            "events:DescribeRule"
        ],
        "Resource": "resource_ARN"
}
```

8. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。

   在管道下，选择您的管道并选择查看。该图显示了您的管道源和部署阶段。

   > **Note**
   >
   > 要查看已创建的管道，请在 AWS CloudFormation中找到堆栈的资源选项卡下的逻辑 ID 列。记下管道的物理 ID 列中的名称。在中 CodePipeline，您可以在创建堆栈的区域中查看具有相同物理 ID（管道名称）的管道。

9. 在您的源存储库中，提交并推送更改。您的更改检测资源会拾取更改，并且您的管道启动。

示例 2：使用 AWS CloudFormation创建 Amazon S3 管道

本演练向您展示如何使用 AWS CloudFormation 控制台创建基础设施，其中包括连接到 Amazon S3 源存储桶的管道。在本教程中，您将使用提供的示例模板文件创建资源堆栈，其中包括您的源存储桶、项目存储、管道和更改检测资源，例如您的 Amazon Ev CloudWatch ents 规则和 CloudTrail跟踪。在中创建资源堆栈后 AWS CloudFormation，您可以在 AWS CodePipeline 控制台中查看您的管道。该管道是一个分为两个阶段的管道，包括一个 Amazon S3 源阶段和一个 CodeDeploy部署阶段。

先决条件：

您必须拥有以下资源才能与 AWS CloudFormation 示例模板一起使用：

- 您必须已创建 Amazon EC2 实例，并在实例上安装了 CodeDeploy 代理。您必须已创建 CodeDeploy 应用程序和部署组。使用您在中创建的 Amazon EC2 和 CodeDeploy 资源教程：创建简单的管道（CodeCommit存储库）。
- 选择以下链接下载用于使用 Amazon S3 源创建管道的示例 AWS CloudFormation 模板文件：
  - 为您的管道下载示例模板：YAML | JSON
  - 下载 CloudTrail 存储桶和跟踪的示例模板：YAML | JSON

  - 解压缩文件并将其放在您的本地计算机上。
- 从 SampleApp_Linux.zip 下载示例应用程序。

  将 .zip 文件保存在本地计算机上。创建堆栈后，您将上传 .zip 文件。

在中创建您的管道 AWS CloudFormation

1. 打开 AWS CloudFormation 控制台，然后选择 “创建堆栈”。选择使用新资源（标准）。
2. 在选择一个模板中，选择上传模板。选中选择文件，然后从您的本地计算机选择模板文件。选择下一步。
3. 在堆栈名称中，输入管道的名称。将显示由示例模板指定的参数。输入以下参数：

   a. 在中 ApplicationName，输入您的 CodeDeploy 应用程序的名称。您可以替换 `DemoApplication` 默认名称。

   b. 在中 BetaFleet，输入您的 CodeDeploy 部署组的名称。您可以替换 `DemoFleet` 默认名称。

   c. 在 SourceObjectKey 中输入 `SampleApp_Linux.zip`。在模板创建存储桶和管道后，将此文件上传到存储桶。
4. 选择下一步。在接下来的页面上接受默认值，然后选择下一步。
5. 在 “能力” 中，选择 “我确认 AWS CloudFormation 可能会创建 IAM 资源”，然后选择 “创建堆栈”。
6. 在堆栈创建完成后，查看事件列表以检查是否存在任何错误。

   故障排除

   在中创建管道的 IAM 用户 AWS CloudFormation 可能需要额外的权限才能为管道创建资源。策略中需要以下权限 AWS CloudFormation 才能允许为 Amazon S3 管道创建所需的 Amazon Ev CloudWatch ents 资源：

```
{
     "Effect": "Allow",
     "Action": [
        "events:PutRule",
        "events:PutEvents",
        "events:PutTargets",
        "events:DeleteRule",
        "events:RemoveTargets",
        "events:DescribeRule"
     ],
     "Resource": "resource_ARN"
```

```
}
```

7. 在 AWS CloudFormation堆栈的 “资源” 选项卡中，查看为您的堆栈创建的资源。

> **Note**
>
> 要查看已创建的管道，请在 AWS CloudFormation中找到堆栈的资源选项卡下的逻辑 ID 列。记下管道的物理 ID 列中的名称。在中 CodePipeline，您可以在创建堆栈的区域中查看具有相同物理 ID（管道名称）的管道。

选择名称中包含 `sourcebucket` 标签的 S3 存储桶，例如 `s3-cfn-codepipeline-sourcebucket-y04EXAMPLE.`。不要选择管道构件存储桶。

源存储桶为空，因为资源是由 AWS CloudFormation新创建的。打开 Amazon S3 控制台，并找到您的 `sourcebucket` 桶。选择 Upload (上传)，然后按照说明上传 `SampleApp_Linux.zip` .zip 文件。

> **Note**
>
> 当 Amazon S3 是管道的源提供程序时，您必须将所有源文件打包为单个 .zip 文件上传到桶。否则，源操作将失败。

8. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。

   在管道下，选择您的管道，然后选择查看。该图显示了您的管道源和部署阶段。

9. 完成以下过程中的步骤来创建您的 AWS CloudTrail 资源。

在中创建您的 AWS CloudTrail 资源 AWS CloudFormation

1. 打开 AWS CloudFormation 控制台，然后选择 “创建堆栈”。
2. 在选择一个模板中，选择将模板上传到 Amazon S3。选择 “浏览”，然后从本地计算机上为 AWS CloudTrail 资源选择模板文件。选择下一步。
3. 在堆栈名称中，输入资源堆栈的名称。将显示由示例模板指定的参数。输入以下参数：
   - 在中 SourceObjectKey，接受示例应用程序的 zip 文件的默认值。
4. 选择下一步。在接下来的页面上接受默认值，然后选择下一步。

5. 在“能力”中，选择“我确认 AWS CloudFormation 可能会创建 IAM 资源”，然后选择“创建”。
6. 在堆栈创建完成后，查看事件列表以检查是否存在任何错误。

   策略中需要以下权限 AWS CloudFormation 才能创建 Amazon S3 管道所需的 CloudTrail 资源：

```
{
     "Effect": "Allow",
     "Action": [
        "cloudtrail:CreateTrail",
        "cloudtrail:DeleteTrail",
        "cloudtrail:StartLogging",
        "cloudtrail:StopLogging",
        "cloudtrail:PutEventSelectors"
     ],
     "Resource": "resource_ARN"
}
```

7. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。

   在管道下，选择您的管道，然后选择查看。该图显示了您的管道源和部署阶段。
8. 在您的源存储桶中，提交并推送更改。您的更改检测资源会拾取更改，并且您的管道启动。

教程：创建使用 AWS CloudFormation 部署操作中的变量的管道

在本教程中，您将使用 AWS CodePipeline 控制台创建带有部署操作的管道。管道运行时，模板会创建一个堆栈，另外还创建一个 `outputs` 文件。堆栈模板生成的输出是中 AWS CloudFormation 操作生成的变量 CodePipeline。

在从模板创建堆栈的操作中，您可以指定一个变量命名空间。然后，后续操作可以使用由 `outputs` 文件生成的变量。在此示例中，您将根据 AWS CloudFormation 操作生成的`StackName`变量创建更改集。手动批准后，执行更改集，然后创建删除堆栈操作，此操作根据 `StackName` 变量删除堆栈。

主题

- 先决条件：创建 AWS CloudFormation 服务角色和 CodeCommit 存储库
- 步骤 1：下载、编辑和上传示例 AWS CloudFormation 模板
- 步骤 2：创建管道
- 步骤 3：添加 AWS CloudFormation 部署操作以创建更改集

- 步骤 4：添加手动审批操作
- 步骤 5：添加 CloudFormation部署操作以执行更改集
- 步骤 6：添加 CloudFormation部署操作以删除堆栈

先决条件：创建 AWS CloudFormation 服务角色和 CodeCommit 存储库

您必须已经具备以下各项：

- 存储 CodeCommit 库。您可以使用您在中创建的 AWS CodeCommit 存储库教程：创建简单的管道（CodeCommit存储库）。
- 此示例从模板创建 Amazon DocumentDB 堆栈。您必须使用 AWS Identity and Access Management (IAM) 为 Amazon DocumentDB 创建具有以下权限的 AWS CloudFormation 服务角色。

```
"rds:DescribeDBClusters",
"rds:CreateDBCluster",
"rds:DeleteDBCluster",
"rds:CreateDBInstance"
```

步骤 1：下载、编辑和上传示例 AWS CloudFormation 模板

下载示例 AWS CloudFormation 模板文件并将其上传到您的 CodeCommit 存储库。

1. 导航到您所在区域的示例模板页面。例如，对应于 us-west-2 的页面位于 https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/sample-templates-services-us-west-2.html。在 Amazon DocumentDB 下，下载 Amazon DocumentDB 集群的模板。文件名为 `documentdb_full_stack.yaml`。
2. 解压缩 `documentdb_full_stack.yaml` 文件，然后在文本编辑器中打开它。进行以下更改。

   a. 对于此示例，将以下 `Purpose:` 参数添加到模板中的 `Parameters` 部分。

```
Purpose:
  Type: String
  Default: testing
  AllowedValues:
    - testing
    - production
```

```
    Description: The purpose of this instance.
```

b. 对于此示例，将以下 StackName 输出添加到模板中的 Outputs: 部分。

```
StackName:
  Value: !Ref AWS::StackName
```

3. 将模板文件上传到您的 AWS CodeCommit 存储库。您必须将解压缩和编辑的模板文件上传到存储库的根目录。

   要使用 CodeCommit 控制台上传文件，请执行以下操作：

   a. 打开 CodeCommit 控制台，然后从 “存储库” 列表中选择您的存储库。

   b. 选择添加文件，然后选择上传文件。

   c. 选择选择文件，然后浏览找到您的文件。通过输入您的用户名和电子邮件地址来提交更改。选择提交更改。

   文件在存储库的根级别看起来应该像这样：

```
documentdb_full_stack.yaml
```

步骤 2：创建管道

在此部分中，您将使用以下操作创建管道：

- 一个源阶段，其 CodeCommit 操作中的源构件就是您的模板文件。
- 包含部署操作的 AWS CloudFormation 部署阶段。

分别向由向导创建的源和部署阶段中的每个操作分配一个变量命名空间 SourceVariables 和 DeployVariables。由于已向操作分配命名空间，因此在此示例中配置的变量可用于下游操作。有关更多信息，请参阅 Variables。

使用向导创建管道

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在欢迎页面、入门页面或管道页面上，选择创建管道。

3. 在步骤 1：选择管道设置的管道名称中，输入 **MyCFNDeployPipeline**。
4. 在本教程中，为管道类型选择 V1。也可以选择 V2；但请注意，不同管道类型具有不同的特性和价格。有关更多信息，请参阅 管道类型。
5. 在服务角色中，执行下列操作之一：
   - 选择 “新建服务角色” CodePipeline 以允许在 IAM 中创建服务角色。
   - 选择现有服务角色。在角色名称中，从列表中选择您的服务角色。
6. 在构件存储中：
   a. 选择默认位置以将默认构件存储（如指定为默认值的 Amazon S3 构件桶）用于为管道选择的区域中的管道。
   b. 如果您在管道所在的区域中已有构件存储（例如，Amazon S3 构件桶），请选择自定义位置。

   > **Note**
   >
   > 这不是源代码的源存储桶。这是管道的项目存储。每个管道都需要一个单独的构件存储，例如 S3 存储桶。创建或编辑管道时，管道区域中必须有一个工件存储桶，并且每个运行操作的 AWS 区域都必须有一个工件存储桶。
   > 有关更多信息，请参阅 输入和输出构件 和 CodePipeline 管道结构参考。

   选择下一步。
7. 在步骤 2：添加源阶段中：
   a. 在源提供程序中，选择 AWS CodeCommit。
   b. 在存储库名称中，选择您在中创建的 CodeCommit 存储库的名称步骤 1：创建 CodeCommit 存储库。
   c. 在分支名称中，选择包含最新的代码更新的分支的名称。

   选择存储库名称和分支后，将显示要为此管道创建的 Amazon CloudWatch Events 规则。

   选择下一步。
8. 在步骤 3：添加构建阶段中，选择跳过构建阶段，并通过再次选择跳过接受警告消息。

   选择下一步。

9. 在步骤 4：添加部署阶段中：

   a. 在操作名称中，选择部署。在部署提供商中，选择CloudFormation。

   b. 在操作模式中，选择创建或更新堆栈。

   c. 在堆栈名称中，为堆栈输入名称。这是模板将创建的堆栈的名称。

   d. 在输出文件名中，输入输出文件的名称，如 **outputs**。这是创建堆栈后由操作创建的文件的名称。

   e. 展开高级。在参数覆盖下，作为键值对输入模板覆盖。例如，此模板需要以下覆盖。

   ```
   {
   "DBClusterName": "MyDBCluster",
   "DBInstanceName": "MyDBInstance",
   "MasterUser": "UserName",
   "MasterPassword": "Password",
   "DBInstanceClass": "db.r4.large",
   "Purpose": "testing"}
   ```

   如果不输入覆盖，则模板会创建一个具有默认值的堆栈。

   f. 选择下一步。

   g. 选择创建管道。允许您的管道运行。您的两阶段管道已完成并准备好添加其他阶段。

步骤 3：添加 AWS CloudFormation 部署操作以创建更改集

在管道中创建下一个操作， AWS CloudFormation 允许在手动批准操作之前创建变更集。

1. 打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。

   在管道下，选择您的管道并选择查看。该图显示了您的管道源和部署阶段。

2. 选择以编辑管道，或继续在编辑模式下显示管道。
3. 选择以编辑部署阶段。
4. 添加一个部署操作，该操作将为在上一个操作中创建的堆栈创建更改集。您可以在阶段中的现有操作之后添加此操作。

   a. 在操作名称中，输入 Change_Set。对于操作提供程序，选择 AWS CloudFormation 。

   b. 在输入构件中，选择SourceArtifact。

   c. 在操作模式中，选择创建或替换更改集。

d. 在堆栈名称中，如下所示输入变量语法。这是为其创建更改集的堆栈的名称，其中，向操作分配默认命名空间 DeployVariables。

```
#{DeployVariables.StackName}
```

e. 在更改集名称中，输入更改集的名称。

```
my-changeset
```

f. 在参数覆盖中，将 Purpose 参数从 testing 更改为 production。

```
{
"DBClusterName": "MyDBCluster",
"DBInstanceName": "MyDBInstance",
"MasterUser": "UserName",
"MasterPassword": "Password",
"DBInstanceClass": "db.r4.large",
"Purpose": "production"}
```

g. 选择完成以保存操作。

步骤 4：添加手动审批操作

在管道中创建手动审批操作。

1. 选择以编辑管道，或继续在编辑模式下显示管道。
2. 选择以编辑部署阶段。
3. 在创建更改集的部署操作之后添加手动审批操作。此操作允许您在管道执行更改集 AWS CloudFormation 之前验证中已创建的资源变更集。

步骤 5：添加 CloudFormation部署操作以执行更改集

在管道中创建下一个操作， AWS CloudFormation 允许在手动批准操作之后执行变更集。

1. 打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。

   在管道下，选择您的管道并选择查看。该图显示了您的管道源和部署阶段。
2. 选择以编辑管道，或继续在编辑模式下显示管道。

3. 选择以编辑部署阶段。
4. 添加将执行在上一个手动操作中审批的更改集的部署操作：
   a. 在操作名称中，输入 Execute_Change_Set。对于操作提供程序，选择 AWS CloudFormation。
   b. 在输入构件中，选择SourceArtifact。
   c. 在操作模式中，选择执行更改集。
   d. 在堆栈名称中，如下所示输入变量语法。这是为其创建更改集的堆栈的名称。

      ```
      #{DeployVariables.StackName}
      ```

   e. 在更改集名称中，输入您在上一个操作中创建的更改集的名称。

      ```
      my-changeset
      ```

   f. 选择完成以保存操作。
   g. 继续管道运行。

步骤 6：添加 CloudFormation部署操作以删除堆栈

在管道中创建最后一个操作， AWS CloudFormation 允许从输出文件中的变量中获取堆栈名称并删除堆栈。

1. 打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。

   在管道下，选择您的管道并选择查看。该图显示了您的管道源和部署阶段。
2. 选择以编辑管道。
3. 选择以编辑部署阶段。
4. 添加将删除堆栈的部署操作：
   a. 在操作名称中，选择DeleteStack。在部署提供商中，选择CloudFormation。
   b. 在操作模式下，选 删除堆栈。
   c. 在堆栈名称中，如下所示输入变量语法。这是操作将删除的堆栈的名称。
   d. 选择完成以保存操作。
   e. 选择保存以保存管道。

管道在保存时运行。

教程：使用 Amazon ECS 标准部署 CodePipeline

本教程可帮助您使用 Amazon ECS 创建完整的 end-to-end 持续部署 (CD) 管道 CodePipeline。

> **Note**
>
> 本教程适用于 Amazon ECS 的标准部署操作 CodePipeline。有关中使用 Amazon ECS 进行 CodeDeploy 蓝/绿部署操作的教程 CodePipeline，请参阅。教程：使用 Amazon ECR 源和 ECS 目标部署创建管道 CodeDeploy

先决条件

您必须先部署一些资源，然后才能使用本教程创建您的 CD 管道。以下是您在开始操作之前需要的资源：

> **Note**
>
> 所有这些资源都应在同一个 AWS 区域内创建。

- 包含您的 Dockerfile 和应用程序源的源代码控制存储库（本教程使用 CodeCommit）。有关更多信息，请参阅《AWS CodeCommit 用户指南》中的 “创建 CodeCommit存储库”。
- 一个 Docker 映像存储库（本教程使用 Amazon ECR），其中包含您从 Dockerfile 和应用程序源构建的映像。有关更多信息，请参阅 Amazon Elastic Container Registry 用户指南 中的创建存储库和推送映像。
- 一个 Amazon ECS 任务定义，该定义会引用在您的映像存储库中托管的 Docker 映像。有关更多信息，请参阅 Amazon Elastic Container Service 开发者指南 中的创建任务定义。

> **Important**
>
> Amazon ECS 的标准部署操作会根据 Amazon ECS 服务使用的修订版 CodePipeline 创建自己的任务定义修订版。如果您在不更新 Amazon ECS 服务的情况下为任务定义创建新的修订，则部署操作将忽略这些修订。

以下是本教程中使用的一个任务定义示例。您对 `name` 和 `family` 使用的值将在下一步中用于构建规范文件。

```
{
  "ipcMode": null,
  "executionRoleArn": "role_ARN",
  "containerDefinitions": [
    {
      "dnsSearchDomains": null,
      "environmentFiles": null,
      "logConfiguration": {
        "logDriver": "awslogs",
        "secretOptions": null,
        "options": {
          "awslogs-group": "/ecs/hello-world",
          "awslogs-region": "us-west-2",
          "awslogs-stream-prefix": "ecs"
        }
      },
      "entryPoint": null,
      "portMappings": [
        {
          "hostPort": 80,
          "protocol": "tcp",
          "containerPort": 80
        }
      ],
      "command": null,
      "linuxParameters": null,
      "cpu": 0,
      "environment": [],
      "resourceRequirements": null,
      "ulimits": null,
      "dnsServers": null,
```

```
      "mountPoints": [],
      "workingDirectory": null,
      "secrets": null,
      "dockerSecurityOptions": null,
      "memory": null,
      "memoryReservation": 128,
      "volumesFrom": [],
      "stopTimeout": null,
      "image": "image_name",
      "startTimeout": null,
      "firelensConfiguration": null,
      "dependsOn": null,
      "disableNetworking": null,
      "interactive": null,
      "healthCheck": null,
      "essential": true,
      "links": null,
      "hostname": null,
      "extraHosts": null,
      "pseudoTerminal": null,
      "user": null,
      "readonlyRootFilesystem": null,
      "dockerLabels": null,
      "systemControls": null,
      "privileged": null,
      "name": "hello-world"
    }
  ],
  "placementConstraints": [],
  "memory": "2048",
  "taskRoleArn": null,
  "compatibilities": [
    "EC2",
    "FARGATE"
  ],
  "taskDefinitionArn": "ARN",
  "family": "hello-world",
  "requiresAttributes": [],
  "pidMode": null,
  "requiresCompatibilities": [
    "FARGATE"
  ],
  "networkMode": "awsvpc",
  "cpu": "1024",
```

```
  "revision": 1,
  "status": "ACTIVE",
  "inferenceAccelerators": null,
  "proxyConfiguration": null,
  "volumes": []
}
```

- 一个 Amazon ECS 集群，该集群运行着一项使用前述任务定义的服务。有关更多信息，请参阅 Amazon Elastic Container Service 开发者指南 中的创建集群和创建服务。

在满足这些先决条件后，您可以继续完成本教程并创建您的 CD 管道。

步骤 1：将构建规范文件添加至您的源存储库

本教程用于 CodeBuild 构建 Docker 镜像并将镜像推送到 Amazon ECR。向您的源代码存储库中添加一个`buildspec.yml`文件以说明 CodeBuild如何执行此操作。下面的示例构建规范将执行以下操作：

- 构建前阶段：
  - 登录 Amazon ECR。
  - 将存储库 URI 设置为您的 ECR 映像并添加包含源的 Git 提交 ID 的前七个字符的映像标签。
- 构建阶段：
  - 构建 Docker 映像并使用 Git 提交 ID 将该映像标记为 `latest`。
- 构建后阶段：
  - 使用两个标签将该映像推送至 ECR 存储库。
  - 在根目录中编写一个称为 `imagedefinitions.json` 文件，其中包含您的 Amazon ECS 服务的容器名称以及映像和标签。您的 CD 管道的部署阶段将使用此信息来创建您的服务的任务定义的新修订，然后它会将该服务更新为使用此新任务定义。ECS 作业辅助角色需要 `imagedefinitions.json` 文件。

粘贴此示例文本以创建您的 `buildspec.yml` 文件，然后替换映像和任务定义的值。此文本使用示例账户 ID 111122223333。

```
version: 0.2

phases:
  pre_build:
    commands:
```

```
      - echo Logging in to Amazon ECR...
      - aws --version
      - aws ecr get-login-password --region $AWS_DEFAULT_REGION | docker login --
username AWS --password-stdin 111122223333.dkr.ecr.us-west-2.amazonaws.com
      - REPOSITORY_URI=012345678910.dkr.ecr.us-west-2.amazonaws.com/hello-world
      - COMMIT_HASH=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-7)
      - IMAGE_TAG=${COMMIT_HASH:=latest}
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $REPOSITORY_URI:latest .
      - docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker images...
      - docker push $REPOSITORY_URI:latest
      - docker push $REPOSITORY_URI:$IMAGE_TAG
      - echo Writing image definitions file...
      - printf '[{"name":"hello-world","imageUri":"%s"}]' $REPOSITORY_URI:$IMAGE_TAG >
 imagedefinitions.json
artifacts:
    files: imagedefinitions.json
```

构建规范为先决条件中提供的示例任务定义而编写，本教程中的 Amazon ECS 服务使用了该定义。REPOSITORY_URI 值对应于 image 存储库 (没有任何映像标签)，此文件末尾附近的 *hello-world* 值对应于该服务的任务定义中的容器名称。

将 **buildspec.yml** 文件添加至您的源存储库

1. 打开文本编辑器，然后将上述构建规范复制并粘贴到新文件中。
2. 将 REPOSITORY_URI 值 (*012345678910.dkr.ecr.us-west-2.amazonaws.com/hello-world*) 替换为您的 Docker 映像的 Amazon ECR 存储库 URI（无任何映像标签）。将 *hello-world* 替换为引用您的 Docker 映像的服务的任务定义中的容器名称。
3. 提交您的 buildspec.yml 文件并将其推送至源存储库。

   a. 添加文件。

   ```
   git add .
   ```

   b. 提交更改。

```
git commit -m "Adding build specification."
```

c. 推送提交。

```
git push
```

步骤 2：创建持续部署管道

使用向 CodePipeline 导创建管道阶段并将源存储库连接到 ECS 服务。

创建管道

1. 打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。
2. 在 Welcome 页面上，选择 Create pipeline。

   如果这是您第一次使用 CodePipeline，则会显示介绍性页面，而不是 “欢迎”。选择 Get Started Now (立即开始)。
3. 在 “步骤 1：名称” 页面上，在 “管道名称” 中，键入管道的名称。在本教程中，管道名称为 hello-world。
4. 在本教程中，为管道类型选择 V1。也可以选择 V2；但请注意，不同管道类型具有不同的特性和价格。有关更多信息，请参阅 管道类型。选择 下一步。
5. 在步骤 2：添加源阶段页面上，对于源提供程序，请选择 AWS CodeCommit。

   a. 在存储库名称中，选择要用作管道源位置的存储 CodeCommit库名称。

   b. 对于 Branch name (分支名称)，选择要使用的分支，然后选择 Next (下一步)。
6. 在步骤 3：添加构建阶段页面上，对于构建提供程序，请选择 AWS CodeBuild，然后选择创建项目。

   a. 对于 Project name，为您的构建项目选择唯一名称。在本教程中，项目名称为 hello-world。

   b. 对于环境映像，选择托管映像。

   c. 对于操作系统，选择 Amazon Linux 2。

   d. 对于运行时，选择标准。

   e. 对于映像，请选择 **aws/codebuild/amazonlinux2-x86_64-standard:3.0**。

   f. 对于 Image version (映像版本) 和 Environment type (环境类型)，请使用默认值。

g. 选择 Enable this flag if you want to build Docker images or want your builds to get elevated privileges (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)。

h. 取消选择CloudWatch 日志。您可能需要展开高级。

i. 选择 “继续” CodePipeline。

j. 选择下一步。

> **Note**
>
> 该向导会为您的生成项目创建一个名为 codebuild-service ***build-project-name***-role 的 CodeBuild 服务角色。记下此角色名称，因为稍后您要向该角色添加 Amazon ECR 权限。

7. 在步骤 4：添加部署阶段页面上，对于部署提供程序，请选择 Amazon ECS。

   a. 对于集群名称，请选择在其中运行您的服务的 Amazon ECS 集群。在本教程中，该集群为 default。

   b. 对于 Service name (服务名称)，选择要更新的服务，然后选择 Next (下一步)。在本教程中，服务名称为 hello-world。

8. 在 Step 5: Review (步骤 5: 审核) 页面上，审查您的管道配置，然后选择 Create pipeline (创建管道) 以创建管道。

> **Note**
>
> 既然管道已创建好，它将尝试经历不同的管道阶段。但是，向导创建的默认 CodeBuild 角色没有权限执行`buildspec.yml`文件中包含的所有命令，因此生成阶段将失败。下一部分将为构建阶段添加权限。

步骤 3：为角色添加 Amazon ECR 权限 CodeBuild

该 CodePipeline 向导为 CodeBuild 构建项目创建了一个名为 codebuild-service ***build-project-name***- role 的 IAM 角色。在本教程中，名称为 codebuild-hello-world-service-role。由于 `buildspec.yml` 文件会调用 Amazon ECR API 操作，因此该角色必须具有授权进行这些 Amazon ECR 调用的策略。以下过程帮助您将适当权限附加到该角色。

向角色添加 Amazon ECR 权限 CodeBuild

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在左侧导航窗格中，选择 Roles（角色）。
3. 在搜索框中，键入 codebuild-，然后选择 CodePipeline 向导创建的角色。在本教程中，角色名称为 codebuild-hello-world-service-role。
4. 在 Summary (摘要) 页面上，选择 Attach policies (附加策略)。
5. 选中 AmazonEC2 ContainerRegistryPowerUser 政策左侧的复选框，然后选择附加政策。

步骤 4：测试您的管道

您的管道应该具备运行 end-to-end 原生 AWS 持续部署所需的一切。现在，通过将代码更改推送至您的源存储库来测试管道的功能。

测试您的管道

1. 对您的已配置源存储库进行代码更改，然后提交并推送更改。
2. 打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。
3. 从列表中选择您的管道。
4. 监视管道经历不同阶段的进度。您的管道应完成且您的 Amazon ECS 服务运行通过您的代码更改创建的 Docker 映像。

教程：使用 Amazon ECR 源和 ECS 目标部署创建管道 CodeDeploy

在本教程中，您将配置一个管道， AWS CodePipeline 该管道使用支持 Docker 映像的蓝/绿部署部署来部署容器应用程序。在蓝/绿部署中，您可以在运行旧版本应用程序的同时启动新版本应用程序，并在重新路由流量之前测试新版本。您还可以监控部署流程并且在存在问题的情况下迅速回滚。

> **Note**
>
> 本教程适用于 Amazon ECS 的 CodeDeploy 蓝/绿部署操作。 CodePipeline有关使用中的 Amazon ECS 标准部署操作的教程 CodePipeline，请参阅教程：使用 Amazon ECS 标准部署 CodePipeline。

已完成的管道会检测您的映像的更改，该图像存储在映像存储库（例如 Amazon ECR）中，并用于 CodeDeploy 将流量路由和部署到 Amazon ECS 集群和负载均衡器。 CodeDeploy 使用侦听器将流量重新路由到 AppSpec 文件中指定的已更新容器的端口。如需了解如何在蓝绿部署中使用负载均衡器、生产侦听器、目标组和 Amazon ECS 应用程序，请参阅教程：部署 Amazon ECS 服务。

管道还配置为使用存储您的 Amazon ECS 任务定义的源位置，例如。 CodeCommit在本教程中，您将配置这些 AWS 资源中的每一个，然后创建包含每种资源操作的阶段的管道。

在源代码发生更改时或者新基本映像上传到 Amazon ECR 时，您的持续交付管道将自动构建和部署容器映像。

此流程使用以下项目：

- 一个 Docker 映像文件，该文件指定容器名称和 Amazon ECR 映像存储库的存储库 URI。
- 一个 Amazon ECS 任务定义，该定义列出您的 Docker 映像名称、容器名称、Amazon ECS 服务名称和负载均衡器配置。
- 一个 CodeDeploy AppSpec 文件，它指定 Amazon ECS 任务定义文件的名称、更新的应用程序容器的名称以及 CodeDeploy重新路由生产流量的容器端口。它还可以指定可选的网络配置以及您在部署生命周期事件挂钩期间可运行的 Lambda 函数。

> **Note**
>
> 当您把更改提交到 Amazon ECR 映像存储库时，管道源操作会为该提交创建 `imageDetail.json` 文件。有关 `imageDetail.json` 文件的信息，请参阅适用于 Amazon ECS 蓝绿部署的 imageDetail.json 文件。

在创建或编辑管道和更新或指定部署阶段更新或指定源项目时，请确保指向要使用的带最新名称和版本的源项目。在设置管道后，在对映像或任务定义进行更改时，您可能需要更新存储库中的源项目文件，然后在管道中编辑部署阶段。

主题

- 步骤 4：创建您的 Amazon ECS 集群和服务
- 步骤 5：创建您的 CodeDeploy应用程序和部署组（ECS 计算平台）
- 步骤 6：创建管道
- 步骤 7：对您的管道进行更改并验证部署

先决条件

您必须已创建以下资源：

- 存储 CodeCommit 库。您可以使用您在中创建的 AWS CodeCommit 存储库教程：创建简单的管道（CodeCommit存储库）。
- 启动 Amazon EC2 Linux 实例并安装 Docker 以创建映像，如本教程中所示。如果您已有要使用的映像，则可以跳过此先决条件。

步骤 1：创建映像并推送至 Amazon ECR 存储库

在本节中，您将使用 Docker 创建映像，然后使用创建 Amazon ECR 存储库并将该映像推送到存储库。 AWS CLI

> **Note**
>
> 如果您已有要使用的映像，则可以跳过此步骤。

创建映像

1. 登录到已安装 Docker 的 Linux 实例。

   下拉 `nginx` 的映像。此命令提供 `nginx:latest` 映像：

   ```
   docker pull nginx
   ```

2. 运行 docker images。您将在列表中看到该映像。

   ```
   docker images
   ```

创建 Amazon ECR 存储库并推送映像

1. 创建用于存储您的 映像的 Amazon ECR 存储库。记下输出中的 `repositoryUri`。

```
aws ecr create-repository --repository-name nginx
```

输出：

```
{
    "repository": {
        "registryId": "aws_account_id",
        "repositoryName": "nginx",
        "repositoryArn": "arn:aws:ecr:us-east-1:aws_account_id:repository/nginx",
        "createdAt": 1505337806.0,
        "repositoryUri": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/nginx"
    }
}
```

2. 使用上一步中的 `repositoryUri` 值标记映像。

```
docker tag nginx:latest aws_account_id.dkr.ecr.us-east-1.amazonaws.com/nginx:latest
```

3. 运行 aws ecr get-login-password 命令，如 `us-west-2` 区域和 111122223333 账户 ID 的此示例中所示。

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-west-2.amazonaws.com/nginx
```

4. 使用上一步中的 `repositoryUri` 将映像推送至 Amazon ECR。

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/nginx:latest
```

第 2 步：创建任务定义和 AppSpec 源文件并推送到 CodeCommit 存储库

在此部分中，您将创建一个任务定义 JSON 文件并将该文件注册到 Amazon ECS。然后，您可以为创建 AppSpec 文件 CodeDeploy 并使用 Git 客户端将这些文件推送到您的 CodeCommit 存储库。

为您的映像创建任务定义

1. 使用以下内容创建名为 taskdef.json 的文件。对于 image，输入您的映像名称，如 nginx。此值会在管道运行时进行更新。

> **Note**
>
> 请确保在任务定义中指定的执行角色包含 AmazonECSTaskExecutionRolePolicy。有关更多信息，请参阅 Amazon ECS 开发者指南 中的 Amazon ECS 任务执行 IAM 角色。

```
{
    "executionRoleArn": "arn:aws:iam::account_ID:role/ecsTaskExecutionRole",
    "containerDefinitions": [
        {
            "name": "sample-website",
            "image": "nginx",
            "essential": true,
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ]
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "networkMode": "awsvpc",
    "cpu": "256",
    "memory": "512",
    "family": "ecs-demo"
}
```

2. 使用 taskdef.json 文件注册任务定义。

```
aws ecs register-task-definition --cli-input-json file://taskdef.json
```

3. 注册任务定义后，编辑您的文件以删除映像名称并将 <IMAGE1_NAME> 占位符文本包含在映像字段中。

```
{
    "executionRoleArn": "arn:aws:iam::account_ID:role/ecsTaskExecutionRole",
    "containerDefinitions": [
        {
            "name": "sample-website",
            "image": "<IMAGE1_NAME>",
            "essential": true,
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ]
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "networkMode": "awsvpc",
    "cpu": "256",
    "memory": "512",
    "family": "ecs-demo"
}
```

创建 AppSpec 文件

- 该 AppSpec 文件用于 CodeDeploy 部署。该文件（包含可选字段）使用以下格式：

```
version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: "task-definition-ARN"
        LoadBalancerInfo:
          ContainerName: "container-name"
          ContainerPort: container-port-number
# Optional properties
```

```
        PlatformVersion: "LATEST"
        NetworkConfiguration:
            AwsvpcConfiguration:
              Subnets: ["subnet-name-1", "subnet-name-2"]
              SecurityGroups: ["security-group"]
              AssignPublicIp: "ENABLED"
Hooks:
- BeforeInstall: "BeforeInstallHookFunctionName"
- AfterInstall: "AfterInstallHookFunctionName"
- AfterAllowTestTraffic: "AfterAllowTestTrafficHookFunctionName"
- BeforeAllowTraffic: "BeforeAllowTrafficHookFunctionName"
- AfterAllowTraffic: "AfterAllowTrafficHookFunctionName"
```

有关该 AppSpec 文件的更多信息（包括示例），请参阅CodeDeploy AppSpec 文件引用。

使用以下内容创建名为 appspec.yaml 的文件。对于 TaskDefinition，请勿更改 <TASK_DEFINITION> 占位符文本。此值会在管道运行时进行更新。

```
version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: <TASK_DEFINITION>
        LoadBalancerInfo:
          ContainerName: "sample-website"
          ContainerPort: 80
```

将文件推送到您的 CodeCommit 存储库

1. 将文件推送或上传到您的 CodeCommit 存储库。这些文件是由 “创建管道” 向导为中部署操作创建的源项目 CodePipeline。您的文件在本地目录中应如下所示：

```
/tmp
  |my-demo-repo
    |-- appspec.yaml
    |-- taskdef.json
```

2. 选择要用于上传文件的方法：

   a. 在本地计算机上从克隆的存储库使用 Git 命令行：

i. 将目录更改为本地存储库：

```
(For Linux, macOS, or Unix) cd /tmp/my-demo-repo
(For Windows) cd c:\temp\my-demo-repo
```

ii. 运行以下命令以立即暂存您的所有文件：

```
git add -A
```

iii. 运行以下命令以提交带有提交消息的文件：

```
git commit -m "Added task definition files"
```

iv. 运行以下命令将本地存储库中的文件推送到存储 CodeCommit 库：

```
git push
```

b. 要使用 CodeCommit 控制台上传文件，请执行以下操作：

i. 打开 CodeCommit 控制台，然后从 “存储库” 列表中选择您的存储库。

ii. 选择添加文件，然后选择上传文件。

iii. 选择选择文件，然后浏览以找到您的文件。通过输入您的用户名和电子邮件地址来提交更改。选择提交更改。

iv. 对要上传的每个文件重复此步骤。

步骤 3：创建您的应用程序负载均衡器和目标组

在此部分中，您将创建一个 Amazon EC2 应用程序负载均衡器。在创建 Amazon ECS 服务时，您可以使用稍后将通过负载均衡器创建的子网名称和目标组值。您可以创建应用程序负载均衡器或网络负载均衡器。负载均衡器必须使用在不同的可用区中具有两个公有子网的 VPC。在这些步骤中，您将确认您的默认 VPC，创建负载均衡器，然后为负载均衡器创建两个目标组。有关更多信息，请参阅[您的网络负载均衡器的目标组](#)。

验证您的默认 VPC 和公有子网

1. 登录 AWS Management Console 并打开亚马逊 VPC 控制台，[网址为 https://console.aws.amazon.com/vpc/](#)。

2. 验证要使用的默认 VPC。在导航窗格中，选择您的 VPC。请注意哪个 VPC 在默认 VPC 列中显示为是。这是默认 VPC。它包含可供您选择的默认子网。
3. 选择子网。选择两个在默认子网列中显示是的子网。

> **Note**
>
> 记下子网 ID。本教程后面将会用到这些 ID。

4. 选择子网，然后选择描述选项卡。验证要使用的子网是否位于不同的可用区中。
5. 选择子网，然后选择路由表选项卡。为了验证要使用的每个子网是否为公有子网，请确认网关行包含在路由表中。

创建 Amazon EC2 应用程序负载均衡器

1. 登录 AWS Management Console 并打开亚马逊 EC2 控制台，网址为 https://console.aws.amazon.com/ec2/。
2. 在导航窗格中，选择负载均衡器。
3. 选择 Create Load Balancer（创建负载均衡器）。
4. 选择 Application Load Balancer（应用程序负载均衡器），然后选择 Create（创建）。
5. 在 Name（名称）中，输入负载均衡器的名称。
6. 在 Scheme（模式）中，选择 internet-facing（面向 internet）。
7. 在 IP 地址类型中，选择 ipv4。
8. 为您的负载均衡器配置两个侦听器端口：
   a. 在负载均衡器协议下，选择 HTTP。在负载均衡器端口下，输入 **80**。
   b. 选择添加侦听器。
   c. 在第二个侦听器的负载均衡器协议下，选择 HTTP。在 Load Balancer Port（负载均衡器端口）下，输入 **8080**。
9. 在可用区下，在 VPC 中，选择默认 VPC。接下来，选择要使用的两个默认子网。
10. 选择下一步：配置安全设置。
11. 选择下一步：配置安全组。
12. 选择选择现有安全组，然后记下安全组 ID。
13. 选择下一步：配置路由。

14. 在目标组中，选择新建目标组并配置您的第一个目标组：

    a. 在名称中，输入目标组名称（例如，**target-group-1**）。
    b. 在 Target type（目标类型）中，选择 IP。
    c. 在 Protocol（协议）中，选择 HTTP。在 Port（端口）中，输入 **80**。
    d. 选择下一步：注册目标。
15. 选择 Next: Review（下一步：审核），然后选择 Create（创建）。

为您的负载均衡器创建第二个目标组

1. 在预配置您的负载均衡器后，打开 Amazon EC2 控制台。在导航窗格中，选择目标组。
2. 选择创建目标组。
3. 在 Name（名称）中，输入目标组名称（例如，**target-group-2**）。
4. 在 Target type（目标类型）中，选择 IP。
5. 在 Protocol（协议）中，选择 HTTP。在 Port（端口）中，输入 **8080**。
6. 在 VPC 中，选择默认 VPC。
7. 选择创建。

> **Note**
>
> 要使部署运行，您必须为您的负载均衡器创建了两个目标组。您只需记下第一个目标组的 ARN。在下一步中，此 ARN 将用于 `create-service` JSON 文件。

更新您的负载均衡器以包含第二个目标组

1. 打开 Amazon EC2 控制台。在导航窗格中，选择负载均衡器。
2. 选择您的负载均衡器，然后选择侦听器选项卡。选择具有端口 8080 的侦听器，然后选择编辑。
3. 选择转发至旁的铅笔图标。选择您的第二个目标组，然后选择复选标记。选择更新以保存更新。

步骤 4：创建您的 Amazon ECS 集群和服务

在本节中，您将创建一个 Amazon ECS 集群和服务，用于在部署期间 CodeDeploy 路由流量（到 Amazon ECS 集群而不是 EC2 实例）。要创建 Amazon ECS 服务，您必须使用通过负载均衡器创建的子网名称、安全组和目标组值来创建服务。

> **Note**
>
> 使用这些步骤创建 Amazon ECS 集群时，您将使用仅限联网的集群模板，该模板预配置 AWS Fargate 容器。 AWS Fargate 是一项为您管理容器实例基础设施的技术。您不需要为 Amazon ECS 集群选择或手动创建 Amazon EC2 实例。

要创建 Amazon ECS 集群

1. 打开 Amazon ECS 经典控制台： https://console.aws.amazon.com/ecs/。
2. 在导航窗格中，选择集群。
3. 选择创建集群。
4. 选择使用 AWS Fargate 的仅限联网集群模板，然后选择下一步。
5. 在配置集群页面上，输入集群名称。您可以为资源添加可选标签。选择创建。

创建 Amazon ECS 服务

使用在 AWS CLI Amazon ECS 中创建您的服务。

1. 创建一个 JSON 文件并将其命名为 `create-service.json`。将以下内容粘贴到 JSON 文件中。

   对于 `taskDefinition` 字段，当您在 Amazon ECS 中注册任务定义时，请提供一个系列。这类似于使用修订号指定的任务定义的多个版本的名称。在此示例中，对文件中的系列和修订号使用“`ecs-demo:1`”。使用您在步骤 3：创建您的应用程序负载均衡器和目标组 中随负载均衡器创建的子网名称、安全组和目标组值。

   > **Note**
   >
   > 您需要在此文件中包含您的目标组 ARN。打开 Amazon EC2 控制台，然后在导航窗格中的负载均衡下，选择目标组。选择您的第一个目标组。从描述选项卡复制您的 ARN。

```
{
    "taskDefinition": "family:revision-number",
    "cluster": "my-cluster",
    "loadBalancers": [
```

```
        {
            "targetGroupArn": "target-group-arn",
            "containerName": "sample-website",
            "containerPort": 80
        }
    ],
    "desiredCount": 1,
    "launchType": "FARGATE",
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "CODE_DEPLOY"
    },
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "subnets": [
                "subnet-1",
                "subnet-2"
            ],
            "securityGroups": [
                "security-group"
            ],
            "assignPublicIp": "ENABLED"
        }
    }
}
```

2. 运行 create-service 命令，并指定 JSON 文件：

> **Important**
>
> 务必在文件名前包含 `file://`。此命令中需要该项。

此示例创建一个名为 `my-service` 的服务。

> **Note**
>
> 此示例命令创建一个名为 my-service 服务。如果您已有一个具有该名称的服务，则命令会返回错误。

```
aws ecs create-service --service-name my-service --cli-input-json file://create-
service.json
```

输出将返回您的服务的描述字段。

3. 运行 describe-services 命令以验证是否已创建您的服务。

```
aws ecs describe-services --cluster cluster-name --services service-name
```

步骤 5：创建您的 CodeDeploy应用程序和部署组（ECS 计算平台）

当您为 Amazon ECS 计算平台创建 CodeDeploy 应用程序和部署组时，将在部署期间使用该应用程序来引用正确的部署组、目标组、侦听器和流量重新路由行为。

创建 CodeDeploy 应用程序

1. 打开 CodeDeploy 控制台并选择创建应用程序。
2. 在应用程序名称中，输入要使用的名称。
3. 在计算平台中，选择 Amazon ECS。
4. 选择创建应用程序。

创建 CodeDeploy 部署组

1. 在应用程序页面的部署组选项卡上，选择创建部署组。
2. 在部署组名称中，输入一个描述部署组的名称。
3. 在服务角色中，选择一个授予 CodeDeploy对 Amazon ECS 访问权限的服务角色。若要创建新的服务角色，请按照以下步骤操作：

   a. 通过以下网址打开 IAM 控制台：https://console.aws.amazon.com/iam/。

   b. 在控制台控制面板中，选择角色。

   c. 选择创建角色。

   d. 在选择受信任实体的类型下，选择 AWS 服务。在“选择用例”下，选择CodeDeploy。在“选择您的用例”下，选择 CodeDeploy -ECS。选择下一步：权限。AWSCodeDeployRoleForECS 托管策略已附加到角色。

e. 选择下一步：标签，然后选择下一步：审核。

f. 输入角色的名称（例如 **CodeDeployECSRole**），然后选择创建角色。

4. 在环境配置中，选择 Amazon ECS 集群名称和服务名称。
5. 从负载均衡器中，选择将流量提供给 Amazon ECS 服务的负载均衡器的名称。
6. 在生产侦听器端口中，选择将生产流量路由至您的 Amazon ECS 服务的侦听器的端口和协议。在测试侦听器端口中，选择测试侦听器的端口和协议。
7. 从目标组 1 名称和目标组 2 名称中，选择用于在部署期间路由流量的目标组。请确保它们是您为负载均衡器创建的目标组。
8. 选择立即重新路由流量以确定成功部署后将流量重新路由到更新后的 Amazon ECS 任务的时长。
9. 选择 Create deployment group（创建部署组）。

步骤 6：创建管道

在此部分中，您将使用以下操作创建管道：

- 一种 CodeCommit 操作，其中源对象是任务定义和 AppSpec 文件。
- 一个带 Amazon ECR 源操作的源阶段，其中源构件是映像文件。
- 具有 Amazon ECS 部署操作的部署阶段，其中部署与 CodeDeploy 应用程序和部署组一起运行。

使用向导创建两阶段管道

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在欢迎页面、入门页面或管道页面上，选择创建管道。
3. 在步骤 1：选择管道设置的管道名称中，输入 **MyImagePipeline**。
4. 在本教程中，为管道类型选择 V1。也可以选择 V2；但请注意，不同管道类型具有不同的特性和价格。有关更多信息，请参阅 管道类型。
5. 在服务角色中，选择新建服务角色 CodePipeline以允许在 IAM 中创建服务角色。
6. 将高级设置中的各项设置保留为默认值，然后选择下一步。
7. 在步骤 2：添加源阶段的源提供程序中，选择 AWS CodeCommit。在存储库名称中，选择您在中创建的 CodeCommit存储库的名称步骤 1：创建 CodeCommit 存储库。在分支名称中，选择包含最新的代码更新的分支的名称。

   选择下一步。

8. 在步骤 3：添加构建阶段中，选择跳过构建阶段，并通过再次选择跳过接受警告消息。选择下一步。
9. 在步骤 4：添加部署阶段中：
   a. 在部署提供商中，选择 Amazon ECS（蓝/绿）。在应用程序名称中，输入应用程序名称或从列表中选择应用程序名称，例如 `codedeployapp`。在部署组中，输入部署组名称或从列表中选择部署组名称，例如 `codedeploydeplgroup`。

      > **Note**
      > 名称“Deploy”是在 步骤 4：部署步骤中创建的阶段的默认名称，正如“Source”是管道第一阶段的名称一样。

   b. 在 Amazon ECS 任务定义下，选择SourceArtifact。在字段中，输入 **`taskdef.json`**。
   c. 在 “AWS CodeDeploy AppSpec 文件” 下，选择SourceArtifact。在字段中，输入 **`appspec.yaml`**。

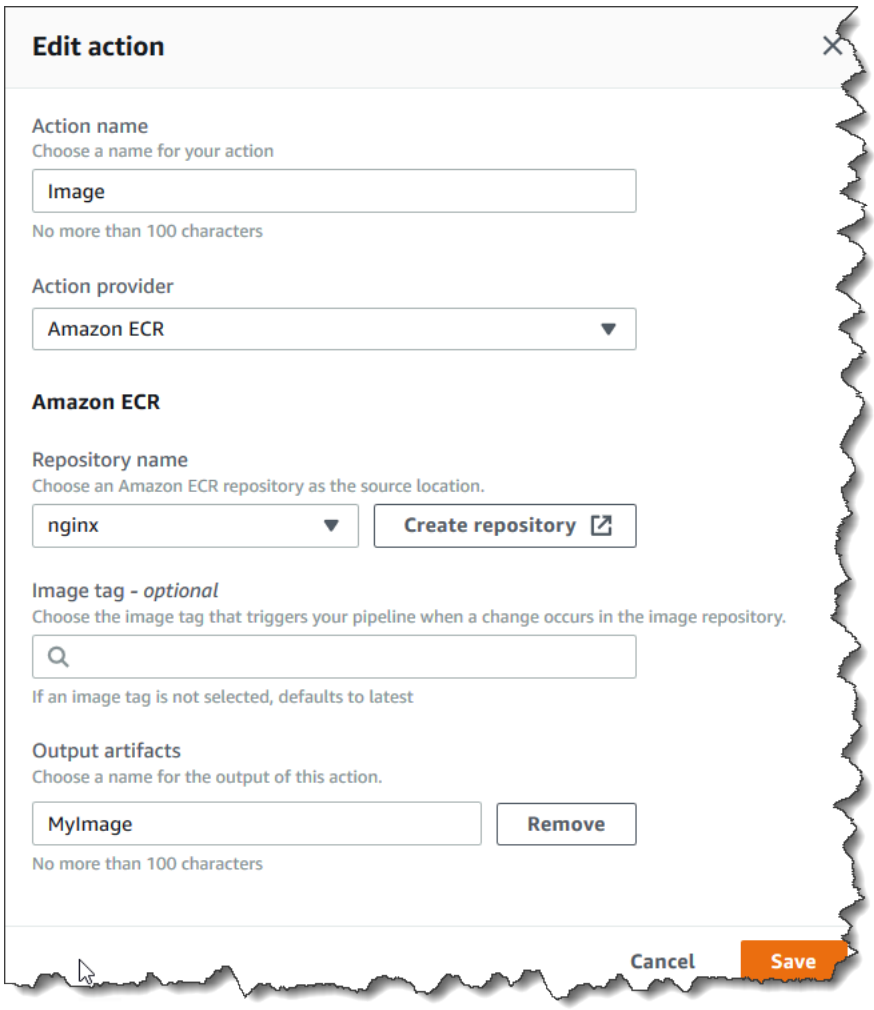
      > **Note**
      > 此时，请不要在动态更新任务定义映像下填写任何信息。

   d. 选择下一步。
10. 在步骤 5：审核中，查看信息，然后选择创建管道。

向管道添加 Amazon ECR 源操作

查看您的管道并向其添加 Amazon ECR 源操作。

1. 选择您的管道。在左上方，选择编辑。
2. 在源阶段中，选择编辑阶段。
3. 选择 CodeCommit 源操作旁边的 + 添加动作，即可添加并行动作。
4. 在操作名称中，输入名称（例如 **`Image`**）。
5. 在操作提供程序中，选择 Amazon ECR。



6. 在存储库名称中，选择 Amazon ECR 存储库的名称。
7. 在映像标签中，指定映像名称和版本（如果与最新版本不同）。
8. 在输出构件中，选择默认输出构件（例如 `MyImage`），其中包含您希望下一阶段使用的映像名称和存储库 URI 信息。
9. 在操作屏幕上选择保存。在阶段屏幕上选择完成。在管道上选择保存。一条消息显示了要为 Amazon CloudWatch ECR 源操作创建的 Amazon Events 规则。

将您的源构件连接到部署操作

1. 在部署阶段上选择编辑，然后选择图标以编辑 Amazon ECS（蓝色/绿色）操作。
2. 滚动到窗格底部。在输入项目中，选择添加。从您的新 Amazon ECR 存储库（例如，`MyImage`）添加源构件。
3. 在 “任务定义” 中 SourceArtifact，选择，然后输入验证**`taskdef.json`**。

4. 在AWS CodeDeploy AppSpec 文件中 SourceArtifact，选择，然后输入验证**appspec.yaml**信息。
5. 在动态更新任务定义图像中，在带图像 URI 的 Input Ar tifact 中 MyImage，选择，然后输入`taskdef.json`文件中使用的占位符文本： **IMAGE1_NAME**。选择保存。
6. 在 AWS CodePipeline 窗格中，选择 “保存管道更改”，然后选择 “保存更改”。查看已更新的管道。

   创建此示例管道后，控制台条目的操作配置按如下所示在管道结构中显示：

```
"configuration": {
  "AppSpecTemplateArtifact": "SourceArtifact",
  "AppSpecTemplatePath": "appspec.yaml",
  "TaskDefinitionTemplateArtifact": "SourceArtifact",
  "TaskDefinitionTemplatePath": "taskdef.json",
  "ApplicationName": "codedeployapp",
  "DeploymentGroupName": "codedeploydeplgroup",
  "Image1ArtifactName": "MyImage",
  "Image1ContainerName": "IMAGE1_NAME"
},
```

7. 要提交所做的更改并开始管道构建，请选择发布更改，然后选择发布。
8. 选择要查看的部署操作 CodeDeploy 并查看流量转移的进度。

> **Note**
>
> 您可能会看到一个部署步骤，显示可选的等待时间。默认情况下，在成功部署后 CodeDeploy 等待一小时才终止原始任务集。您可以利用此时间回滚或终止任务，但在任务集终止时，您的部署也会完成。

步骤 7：对您的管道进行更改并验证部署

对您的映像进行更改，然后将更改推送至您的 Amazon ECR 存储库。这将触发您的管道运行。验证是否已部署您的映像源更改。

教程：创建部署 Amazon Alexa 技能的管道

在本教程中，您将使用 Alexa Skills Kit 作为部署阶段中的部署提供程序来配置一个连续提供 Alexa 技能的管道。当您对源存储库中的源文件进行更改时，已完成的管道会检测对您的技能所做的更改。然后，管道使用 Alexa Skills Kit 部署到 Alexa 技能开发阶段。

> **Note**
>
> 此功能在亚太地区（香港）或欧洲地区（米兰）区域不可用。要使用该区域可用的其他部署操作，请参阅部署操作集成。

要将您的自定义技能创建为 Lambda 函数，请参阅将自定义技能作为 Lambda AWS 函数托管。您还可以创建一个使用 Lambda 源文件的管道和一个用于为您的技能部署对 Lambda 的更改的 CodeBuild 项目。例如，要创建新的技能和相关的 Lambda 函数，您可以创建一个 AWS CodeStar 项目。请参阅在 AWS CodeStar 中创建 Alexa 技能项目。对于该选项，管道包括第三个构建阶段，其中包含一个 CodeBuild操作和一个位于部署阶段的操作 AWS CloudFormation。

先决条件

您必须已经具备以下各项：

- 存储 CodeCommit 库。您可以使用您在中创建的 AWS CodeCommit 存储库教程：创建简单的管道（CodeCommit存储库）。
- 一个 Amazon 开发人员账户。此账户拥有您的 Alexa 技能。您可以在 Alexa Skills Kit 免费创建一个账户。
- 一个 Alexa 技能。您可以使用获取自定义技能示例代码教程创建示例技能。
- 安装 ASK CLI，并将 `ask init` 与您的 AWS 凭证结合使用来配置它。请参阅安装和初始化 ASK CLI。

步骤 1：创建 Alexa 开发人员服务 LWA 安全配置文件

在本节中，您将创建一个要与 Login with Amazon（LWA）一起使用的安全配置文件。如果您已有配置文件，可跳过本步骤。

- 使用中的步骤generate-lwa-tokens创建安全配置文件。
- 在您创建配置文件后，请记下客户端 ID 和客户端密钥。
- 确保您按照说明中提供的内容输入允许的返回 URL。该 URL 允许 ASK CLI 命令重定向刷新令牌请求。

第 2 步：创建 Alexa 技能源文件并推送到你的 CodeCommit 存储库

在本节中，您将创建 Alexa 技能源文件并将其推送到管道用于源阶段的存储库。对于您在 Amazon 开发人员控制台中创建的技能，您可以生成并推送以下内容：

- 一个 `skill.json` 文件。
- 一个 `interactionModel/custom` 文件夹。

> **Note**
>
> 此目录结构符合 Alexa Skills Kit 技能包格式要求，如技能包格式中所述。如果您的目录结构未使用正确的技能包格式，则更改不会成功部署到 Alexa Skills Kit 控制台。

为您的技能创建源文件

1. 从 Alexa Skills Kit 开发人员控制台中检索您的技能 ID。使用此命令：

   ```
   ask api list-skills
   ```

   按名称查找您的技能，然后在 `skillId` 字段中复制关联的 ID。

2. 生成一个 `skill.json` 文件，其中包含您的技能详细信息。使用此命令：

   ```
   ask api get-skill -s skill-ID > skill.json
   ```

3. （可选）创建一个 `interactionModel/custom` 文件夹。

   使用此命令可在文件夹中生成交互模型文件。对于区域设置，本教程使用 en-US 作为文件名中的区域设置。

   ```
   ask api get-model --skill-id skill-ID --locale locale >
       ./interactionModel/custom/locale.json
   ```

将文件推送到您的 CodeCommit 存储库

1. 将文件推送或上传到您的 CodeCommit 存储库。这些文件是由创建管道向导针对您在 AWS CodePipeline中的部署操作创建的源项目。您的文件在本地目录中应如下所示：

```
skill.json
/interactionModel
  /custom
    |en-US.json
```

2. 选择要用于上传文件的方法：

   a. 在本地计算机上从克隆的存储库使用 Git 命令行：

      i. 运行以下命令以立即暂存您的所有文件：

      ```
      git add -A
      ```

      ii. 运行以下命令以提交带有提交消息的文件：

      ```
      git commit -m "Added Alexa skill files"
      ```

      iii. 运行以下命令将本地存储库中的文件推送到存储 CodeCommit 库：

      ```
      git push
      ```

   b. 要使用 CodeCommit 控制台上传您的文件，请执行以下操作：

      i. 打开 CodeCommit 控制台，然后从 “存储库” 列表中选择您的存储库。

      ii. 选择添加文件，然后选择上传文件。

      iii. 选择选择文件，然后浏览以找到您的文件。通过输入您的用户名和电子邮件地址来提交更改。选择提交更改。

      iv. 对要上传的每个文件重复此步骤。

步骤 3：使用 ASK CLI 命令来创建刷新令牌

CodePipeline 根据您的亚马逊开发者账户中的客户端 ID 和密钥使用刷新令牌来授权其代表您执行的操作。在本节中，您将使用 ASK CLI 来创建令牌。您将在使用创建管道向导时使用这些凭证。

使用您的 Amazon 开发人员账户凭据创建刷新令牌

1. 使用以下命令：

```
ask util generate-lwa-tokens
```

2. 在系统提示时，输入您的客户端 ID 和密钥，如下例所示：

```
? Please type in the client ID:
amzn1.application-client.example112233445566
? Please type in the client secret:
example112233445566
```

3. 将显示登录浏览器页面。使用 Amazon 开发人员账户凭证进行登录。
4. 返回到命令行屏幕。会在输出中生成访问令牌和刷新令牌。复制在输出中返回的刷新令牌。

步骤 4：创建管道

在此部分中，您将使用以下操作创建管道：

- 一个源舞台，其 CodeCommit 动作源工件是支持你技能的 Alexa 技能文件。
- 具有 Alexa Skills Kit 部署操作的部署阶段。

使用向导创建管道

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 选择要在其中创建项目及其资源的 AWS 区域。Alexa 技能运行时仅在以下区域可用：
   - 亚太地区（东京）
   - 欧洲地区（爱尔兰）
   - 美国东部（弗吉尼亚州北部）
   - 美国西部（俄勒冈州）
3. 在欢迎页面、入门页面或管道页面上，选择创建管道。
4. 在步骤 1：选择管道设置的管道名称中，输入 **MyAlexaPipeline**。
5. 在本教程中，为管道类型选择 V1。也可以选择 V2；但请注意，不同管道类型具有不同的特性和价格。有关更多信息，请参阅 管道类型。
6. 在服务角色中，选择新建服务角色 CodePipeline以允许在 IAM 中创建服务角色。
7. 将高级设置中的各项设置保留为默认值，然后选择下一步。
8. 在步骤 2：添加源阶段的源提供程序中，选择 AWS CodeCommit。在存储库名称中，选择您在中创建的 CodeCommit 存储库的名称步骤 1：创建 CodeCommit 存储库。在分支名称中，选择包含最新的代码更新的分支的名称。

选择存储库名称和分支后，将显示一条消息，显示要为此管道创建的 Amazon Ev CloudWatch ents 规则。

选择下一步。

9. 在步骤 3：添加构建阶段中，选择跳过构建阶段，并通过再次选择跳过接受警告消息。

选择下一步。

10. 在步骤 4：添加部署阶段中：

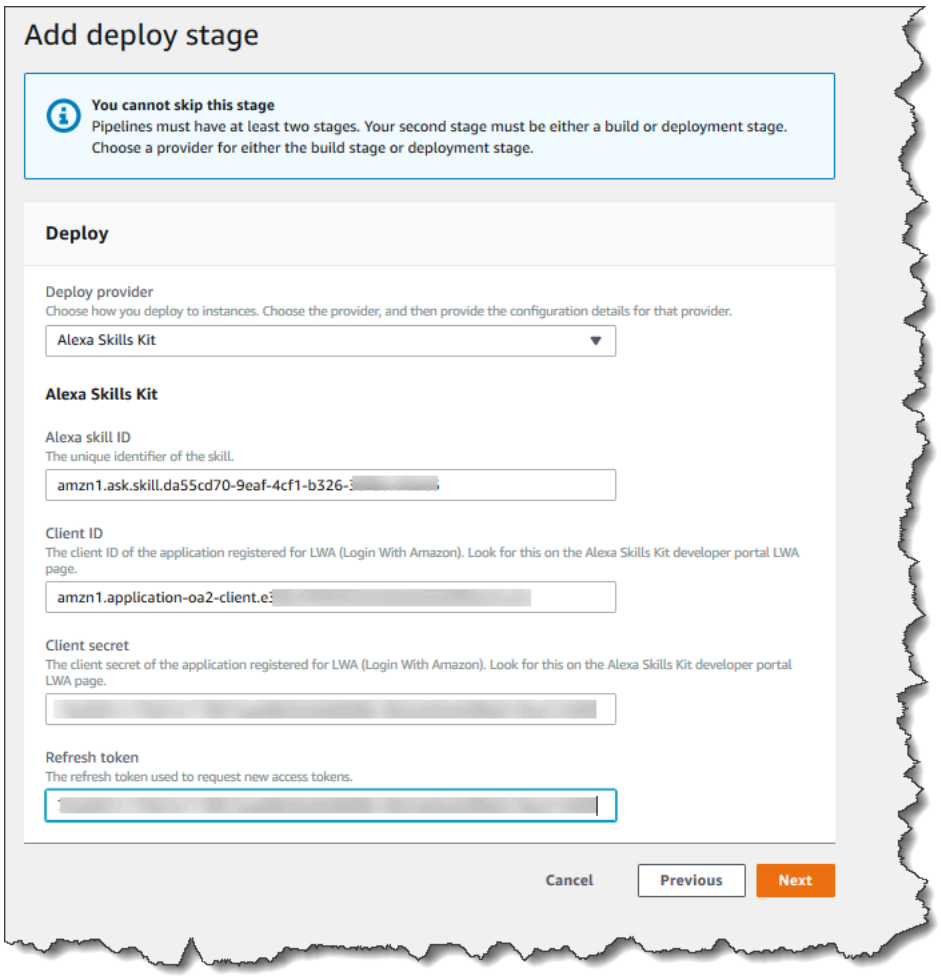
a. 在部署提供程序中，选择 Alexa Skills Kit。

b. 在 Alexa skill ID 中，输入在 Alexa Skills Kit 开发人员控制台中分配给您的技能的技能 ID。

c. 在客户端 ID 中，输入您注册的应用程序的 ID。

d. 在客户端密钥中，输入您在注册时选择的密钥。

e. 在刷新令牌中，输入您在步骤 3 中生成的令牌。



f. 选择下一步。

11. 在步骤 5：审核中，查看信息，然后选择创建管道。

步骤 5：对任何源文件进行更改并验证部署

对您的技能进行更改，然后将更改推送至您的存储库。这将触发您的管道运行。验证您的技能已在 Alexa Skills Kit 开发人员控制台中更新。

教程：创建以 Amazon S3 作为部署提供程序的管道

在本教程中，您将使用 Amazon S3 作为部署阶段中的部署操作提供程序来配置连续交付文件的管道。完成的管道会在您对源存储库中的源文件进行更改时检测更改。然后，管道使用 Amazon S3 将文件部署到您的桶。每次在源位置修改或添加网站文件时，部署都会使用最新文件创建网站。

> **Note**
>
> 即使您从源存储库中删除文件，S3 部署操作也不会删除与已删除文件对应的 S3 对象。

本教程提供了两个选项：

- 创建一个管道，用于将静态网站部署到您的 S3 公有存储桶。此示例创建了一个包含 AWS CodeCommit 源操作和 Amazon S3 部署操作的管道。请参阅 选项 1：将静态网站文件部署到 Amazon S3。
- 创建一个管道，该管道将示例 TypeScript 代码编译到您的 S3 存储桶中， JavaScript 并将 CodeBuild 输出项目部署到您的 S3 存储桶中进行存档。此示例创建了一个包含 Amazon S3 源操作、 CodeBuild 构建操作和 Amazon S3 部署操作的管道。请参阅 选项 2：将构建的存档文件从 S3 源桶部署到 Amazon S3。 。

> **Important**
>
> 在此过程中，您在管道中添加的许多操作都涉及在创建管道之前需要创建的 AWS 资源。 AWS 源操作的资源必须始终在您创建管道的同一 AWS 区域创建。例如，如果您在美国东部（俄亥俄州）地区创建管道，则您的 CodeCommit 存储库必须位于美国东部（俄亥俄州）区域。
> 您可以在创建管道时添加跨区域操作。 AWS 跨区域操作的资源必须位于您计划执行操作的同一 AWS 区域。有关更多信息，请参阅 在中添加跨区域操作 CodePipeline。

选项 1：将静态网站文件部署到 Amazon S3

在此示例中，您将下载示例静态网站模板文件，将文件上传到 AWS CodeCommit 存储库，创建存储桶，然后将其配置为托管。接下来，您可以使用 AWS CodePipeline 控制台创建管道并指定 Amazon S3 部署配置。

先决条件

您必须已经具备以下各项：

- 存储 CodeCommit 库。您可以使用您在中创建的 AWS CodeCommit 存储库教程：创建简单的管道（CodeCommit存储库）。
- 您的静态网站的源文件。使用此链接下载示例静态网站。该 sample-website.zip 下载生成以下文件：
  - 一个 `index.html` 文件
  - 一个 `main.css` 文件
  - 一个 `graphic.jpg` 文件
- 一个为网站托管配置的 S3 存储桶。请参阅在 Amazon S3 上托管静态网站。确保您在与管道相同的区域中创建存储桶。

> **Note**
>
> 若要托管网站，您的桶必须具有公共读取访问权限，以为每个人提供读取访问权限。您应该保留阻止对 S3 存储桶的公共访问的默认访问设置，但网站托管除外。

第 1 步：将源文件推送到您的 CodeCommit 存储库

在本节中，将源文件推送到管道用于源阶段的存储库。

将文件推送到您的 CodeCommit 存储库

1. 提取下载的示例文件。不要将 ZIP 文件上传到您的存储库。
2. 将文件推送或上传到您的 CodeCommit 存储库。这些文件是由 “创建管道” 向导创建的源项目，用于在中的部署操作 CodePipeline。您的文件在本地目录中应如下所示：

```
index.html
```

```
main.css
graphic.jpg
```

3. 你可以使用 Git 或 CodeCommit 控制台上传文件：

   a. 在本地计算机上从克隆的存储库使用 Git 命令行：

      i. 运行以下命令以立即暂存您的所有文件：

      ```
      git add -A
      ```

      ii. 运行以下命令以提交带有提交消息的文件：

      ```
      git commit -m "Added static website files"
      ```

      iii. 运行以下命令将本地存储库中的文件推送到存储 CodeCommit 库：

      ```
      git push
      ```

   b. 要使用 CodeCommit 控制台上传您的文件，请执行以下操作：

      i. 打开 CodeCommit 控制台，然后从 “存储库” 列表中选择您的存储库。

      ii. 选择添加文件，然后选择上传文件。

      iii. 选择选择文件，然后浏览找到您的文件。通过输入您的用户名和电子邮件地址来提交更改。选择提交更改。

      iv. 对要上传的每个文件重复此步骤。

步骤 2：创建管道

在此部分中，您将使用以下操作创建管道：

- 一个源代码阶段，其 CodeCommit 操作中的源构件是您网站的文件。
- 一个具有 Amazon S3 部署操作的部署阶段。

使用向导创建管道

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在欢迎页面、入门页面或管道页面上，选择创建管道。

3. 在步骤 1：选择管道设置的管道名称中，输入 **MyS3DeployPipeline**。
4. 在本教程中，为管道类型选择 V1。也可以选择 V2；但请注意，不同管道类型具有不同的特性和价格。有关更多信息，请参阅 管道类型。
5. 在服务角色中，选择新建服务角色 CodePipeline 以允许在 IAM 中创建服务角色。
6. 将高级设置中的各项设置保留为默认值，然后选择下一步。
7. 在步骤 2：添加源阶段的源提供程序中，选择 AWS CodeCommit。在存储库名称中，选择您在中创建的 CodeCommit存储库的名称步骤 1：创建 CodeCommit 存储库。在分支名称中，选择包含最新的代码更新的分支的名称。除非您自行创建了其他分支，否则仅 `main` 可用。

   选择存储库名称和分支后，将显示要为此管道创建的 Amazon CloudWatch Events 规则。

   选择下一步。
8. 在步骤 3：添加构建阶段中，选择跳过构建阶段，并通过再次选择跳过接受警告消息。

   选择下一步。
9. 在步骤 4：添加部署阶段中：

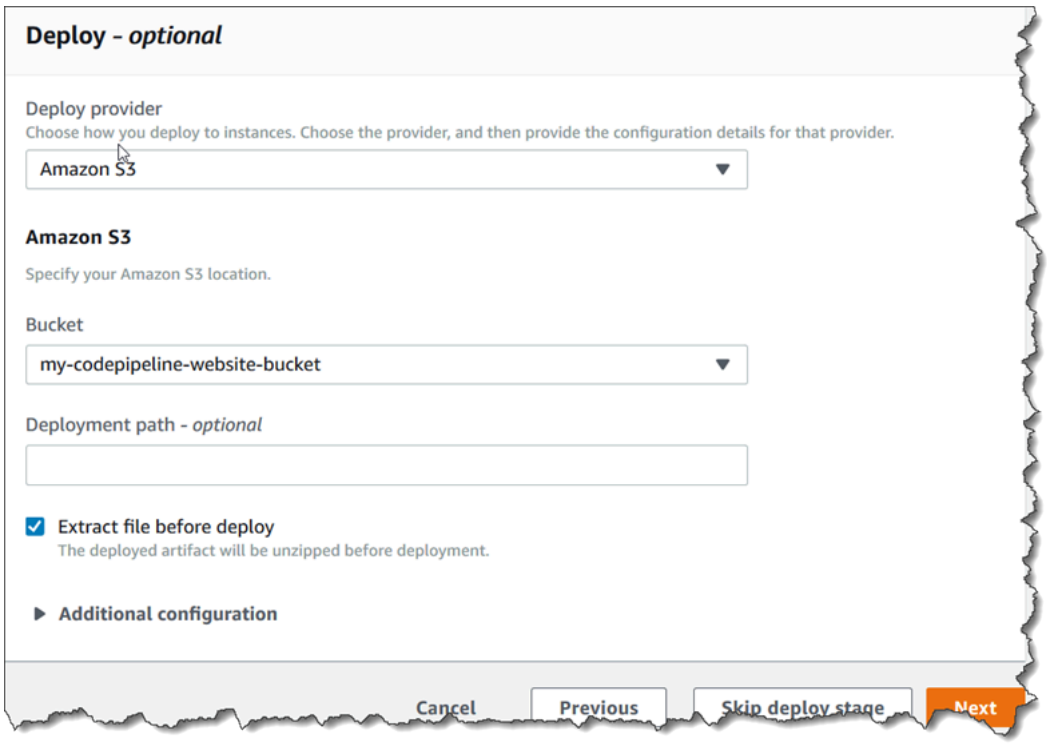
   a. 在部署提供程序中，选择 Amazon S3。

   b. 在存储桶中，输入您的公有存储桶的名称。

   c. 选择在部署前提取文件。

   > **Note**
   >
   > 如果不选择部署前提取文件，部署将会失败。这是因为管道中的 AWS CodeCommit 操作会压缩源工件，而您的文件是一个 ZIP 文件。

   当在部署前提取文件已选择时，将会显示部署路径。输入您要使用的路径的名称。这将在 Amazon S3 中创建一个文件夹结构，文件将提取到该文件夹结构中。对于此教程，将此字段保留为空。



d. （可选）在标准 ACL 中，您可以将一组预定义的授权（称作标准 ACL）应用于上传的构件。

e. （可选）在缓存控制中，输入缓存参数。您可以设置它以控制请求/响应的缓存行为。有关有效值，请参阅 HTTP 操作的 `Cache-Control` 标头字段 。

f. 选择下一步。

10. 在步骤 5：审核中，查看信息，然后选择创建管道。

11. 在管道成功运行后，打开 Amazon S3 控制台并验证您的文件是否显示在公有桶中，如下所示：

```
index.html
main.css
graphic.jpg
```

12. 访问您的终端节点以测试网站。您的终端节点遵循以下格式： `http://bucket-name.s3-website-region.amazonaws.com/`。

    示例终端节点：`http://my-bucket.s3-website-us-west-2.amazonaws.com/`。

    此时会显示示例网页。

步骤 3：对任何源文件进行更改并验证部署

对您的源文件进行更改，然后将更改推送到存储库。这将触发您的管道运行。验证您的网站是否已更新。

选项 2：将构建的存档文件从 S3 源桶部署到 Amazon S3。

在此选项中，构建阶段的构建命令将 TypeScript 代码编译成 JavaScript 代码，并将输出部署到带有时间戳的单独文件夹下的 S3 目标存储桶。首先，创建 TypeScript 代码和一个 buildspec.yml 文件。将源文件合并到 ZIP 文件中后，将源 ZIP 文件上传到 S3 源存储桶，然后使用 CodeBuild 阶段将构建的应用程序 ZIP 文件部署到您的 S3 目标存储桶。编译后的代码将作为存档保留在目标存储桶中。

先决条件

您必须已经具备以下各项：

- S3 源存储桶。您可以使用在教程：创建一个简单的管道（S3 存储桶）中创建的桶。
- S3 目标存储桶。请参阅在 Amazon S3 上托管静态网站。请务必使用与要创建的管道 AWS 区域 相同的方式创建存储桶。

> **Note**
>
> 此示例演示如何将文件部署到私有存储桶。请勿为网站托管启用目标桶，也不要附加任何将桶设为公共桶的策略。

步骤 1：创建源文件并将其上传到您的 S3 源存储桶

在本部分中，您将创建源文件，并将其上传到管道用于源阶段的桶。本节提供有关创建以下源文件的说明：

- 一个`buildspec.yml`文件，用于 CodeBuild 生成项目。
- 一个 `index.ts` 文件。

创建 buildspec.yml 文件

- 使用以下内容创建名为 `buildspec.yml` 的文件。这些编译命令安装 TypeScript 并使用 TypeScript编译器将代码重写`index.ts`为 JavaScript代码。

```
version: 0.2

phases:
  install:
    commands:
      - npm install -g typescript
  build:
    commands:
      - tsc index.ts
artifacts:
  files:
    - index.js
```

创建 index.ts 文件

- 使用以下内容创建名为 index.ts 的文件。

```
interface Greeting {
    message: string;
}

class HelloGreeting implements Greeting {
    message = "Hello!";
}

function greet(greeting: Greeting) {
    console.log(greeting.message);
}

let greeting = new HelloGreeting();

greet(greeting);
```

将文件上传到 S3 源存储桶

1. 您的文件在本地目录中应如下所示：

```
buildspec.yml
index.ts
```

压缩这些文件，并将文件命名为 `source.zip`。

2. 在 Amazon S3 控制台中，对于您的源桶，选择上传。选择添加文件，然后浏览找到您之前创建的 ZIP 文件。
3. 选择上传。这些文件是由 “创建管道” 向导创建的源项目，用于在中的部署操作 CodePipeline。您的文件应该在您的文件夹中如下所示：

```
source.zip
```

步骤 2：创建管道

在此部分中，您将使用以下操作创建管道：

- 一个具有 Amazon S3 操作的源阶段，其中源构件是用于可下载应用程序的文件。
- 一个具有 Amazon S3 部署操作的部署阶段。

使用向导创建管道

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在欢迎页面、入门页面或管道页面上，选择创建管道。
3. 在步骤 1：选择管道设置的管道名称中，输入 **`MyS3DeployPipeline`**。
4. 在服务角色中，选择新建服务角色 CodePipeline 以允许在 IAM 中创建服务角色。
5. 将高级设置中的各项设置保留为默认值，然后选择下一步。
6. 在步骤 2: 添加源阶段的源提供程序中，选择 Amazon S3。在存储桶中，选择您的源存储桶的名称。在 S3 对象键中，输入您的源 ZIP 文件。请确保包含 .zip 文件扩展名。

   选择下一步。
7. 在步骤 3：添加构建阶段中：

   a. 在构建提供程序中，选择 CodeBuild。
   b. 选择创建构建项目。在创建项目页面上：
   c. 在项目名称中，输入此构建项目的名称。
   d. 在环境中，选择托管映像。对于操作系统，选择 Ubuntu。
   e. 对于运行时，选择标准。对于运行时版本，选择 aws/codebuild/standard:1.0。

f. 在映像版本中，选择始终对此运行时版本使用最新映像。

g. 对于服务角色，请选择您的 CodeBuild 服务角色或创建一个服务角色。

h. 对于构建规范，选择使用 buildspec 文件。

i. 选择 “继续” CodePipeline。如果项目已成功创建，则会显示一条消息。

j. 选择下一步。

8. 在步骤 4：添加部署阶段中：

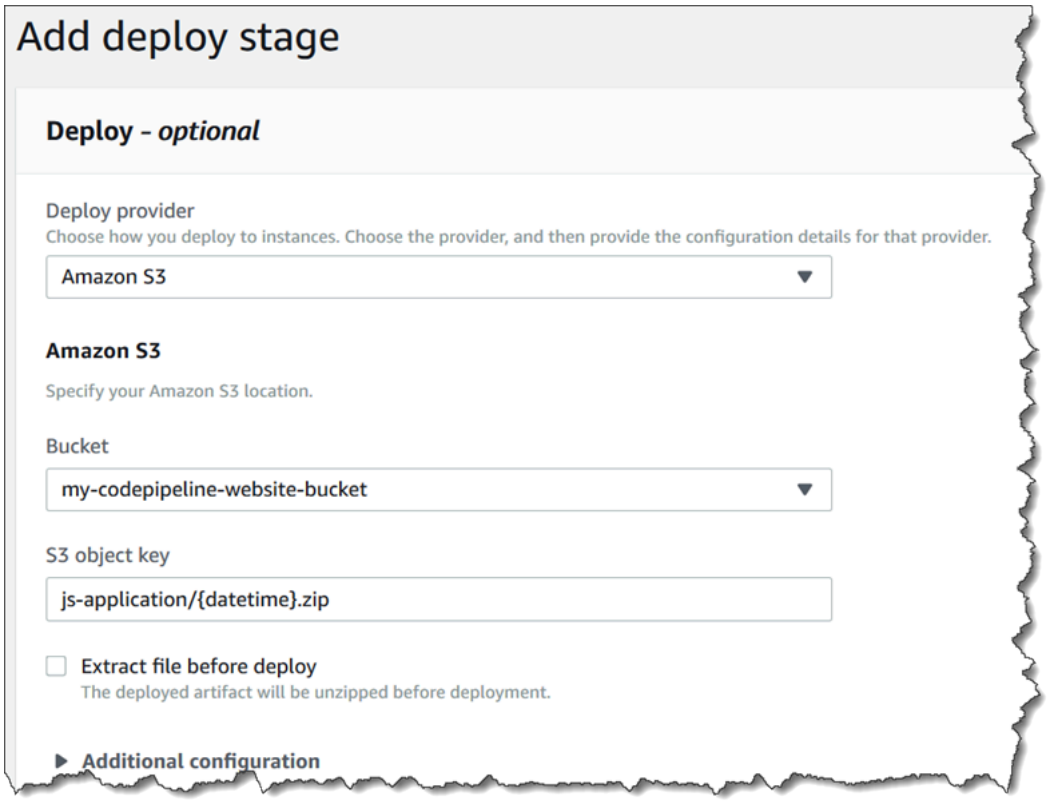
a. 在部署提供程序中，选择 Amazon S3。

b. 在存储桶中，输入您的 S3 目标存储桶的名称。

c. 确保已清除部署前提取文件。

当在部署前提取文件已清除时，将会显示 S3 对象键。输入您要使用的路径的名称：`js-application/{datetime}.zip`。

这将在 Amazon S3 中创建一个 `js-application` 文件夹，文件将提取到该文件夹中。在此文件夹中，`{datetime}` 变量在您的管道运行时会在每个输出文件上创建一个时间戳。



d. （可选）在标准 ACL 中，您可以将一组预定义的授权（称作标准 ACL）应用于上传的构件。

e. （可选）在缓存控制中，输入缓存参数。您可以设置它以控制请求/响应的缓存行为。有关有效值，请参阅 HTTP 操作的 Cache-Control 标头字段 。

f. 选择下一步。

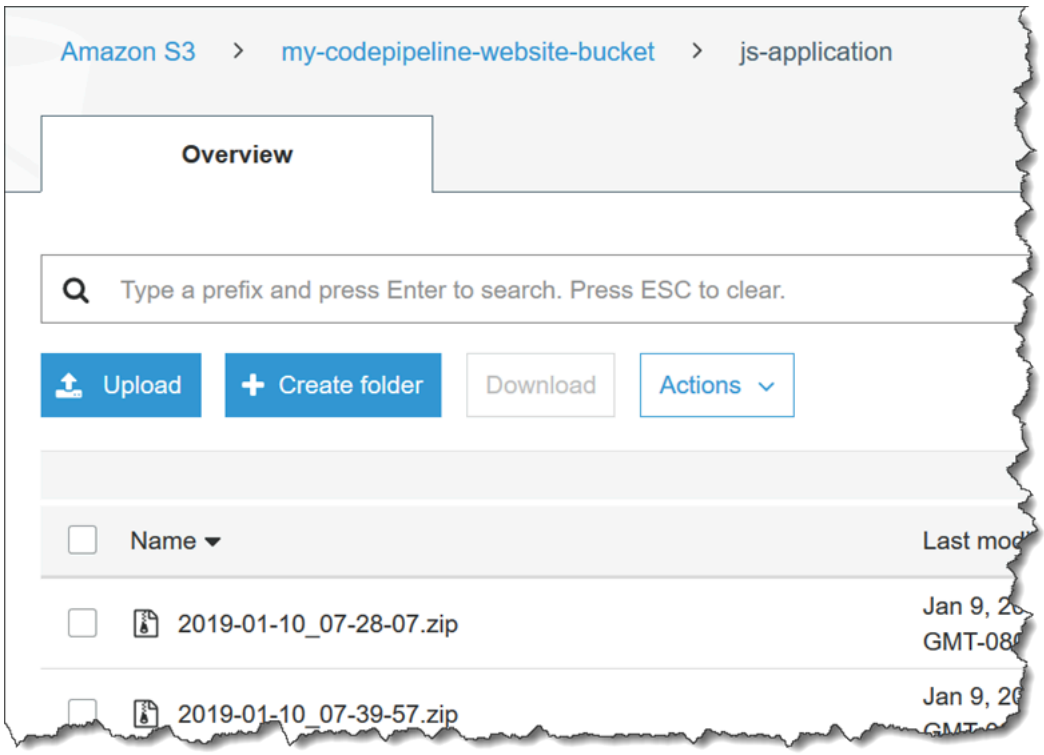
9. 在步骤 5：审核中，查看信息，然后选择创建管道。

10. 管道成功运行后，在 Amazon S3 控制台中查看您的桶。验证已部署的 ZIP 文件是否显示在 `js-application` 文件夹下的目标存储桶中。ZIP JavaScript 文件中包含的文件应为`index.js`。`index.js` 文件包含以下输出：

```
var HelloGreeting = /** @class */ (function () {
    function HelloGreeting() {
        this.message = "Hello!";
    }
    return HelloGreeting;
}());
function greet(greeting) {
    console.log(greeting.message);
}
var greeting = new HelloGreeting();
greet(greeting);
```

步骤 3：对任何源文件进行更改并验证部署

对源文件进行更改，然后将其上传到源存储桶。这将触发您的管道运行。查看目标存储桶并验证部署的输出文件是否在 `js-application` 文件夹中可用，如下所示：



教程：创建将您的无服务器应用程序发布到的管道 AWS Serverless Application Repository

您可以使用 AWS CodePipeline 将您的 AWS SAM 无服务器应用程序持续交付给。 AWS Serverless Application Repository

本教程介绍如何创建和配置管道来构建托管的无服务器应用程序并将其 AWS Serverless Application Repository 自动发布到。 GitHub 管道 GitHub 用作源提供者和 CodeBuild 生成提供者。要将您的无服务器应用程序发布到 AWS Serverless Application Repository，您需要部署一个应用程序（来自 AWS Serverless Application Repository），并将该应用程序创建的 Lambda 函数关联为管道中的 Invoke 操作提供者。然后，您无需编写任何代码即可持续向 AWS Serverless Application Repository提供应用程序更新。

> **Important**
>
> 在此过程中，您在管道中添加的许多操作都涉及在创建管道之前需要创建的 AWS 资源。 AWS 源操作的资源必须始终在您创建管道的同一 AWS 区域创建。例如，如果您在美国东部（俄亥俄州）地区创建管道，则您的 CodeCommit 存储库必须位于美国东部（俄亥俄州）区域。

您可以在创建管道时添加跨区域操作。 AWS 跨区域操作的资源必须位于您计划执行操作的同一 AWS 区域。有关更多信息，请参阅 在中添加跨区域操作 CodePipeline。

开始前的准备工作

在本教程中，我们假设您满足以下条件：

- 您熟悉AWS Serverless Application Model (AWS SAM) 和 AWS Serverless Application Repository。
- 您在中托管了一个无服务器应用程序 GitHub ，并已 AWS Serverless Application Repository 使用 AWS SAM CLI 将其发布到。要向发布示例应用程序 AWS Serverless Application Repository，请参阅《AWS Serverless Application Repository 开发人员指南》中的 “快速入门：发布应用程序”。要将自己的应用程序发布到 AWS Serverless Application Repository，请参阅《AWS Serverless Application Model 开发者指南》中的 “使用 AWS SAM CLI 发布应用程序”。

步骤 1：创建 buildspec.yml 文件

创建包含以下内容的buildspec.yml文件，并将其添加到无服务器应用程序的 GitHub 存储库中。将 t *emplate.yml* 替换为应用程序的 AWS SAM 模板，将存储###替换为存储已打包应用程序的 S3 存储桶。

```
version: 0.2
phases:
  install:
    runtime-versions:
        python: 3.8
  build:
    commands:
      - sam package --template-file template.yml --s3-bucket bucketname --output-
template-file packaged-template.yml
artifacts:
  files:
    - packaged-template.yml
```

步骤 2：创建并配置您的管道

按照以下步骤在要发布无服务器应用程序的 AWS 区域 位置创建管道。

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。
2. 如有必要，请切换到要发布无服务器应用程序 AWS 区域 的位置。
3. 选择创建管道。在 Choose pipeline settings (选择管道设置) 页面上，在 Pipeline name (管道名称) 中，输入管道的名称。
4. 在本教程中，为管道类型选择 V1。也可以选择 V2；但请注意，不同管道类型具有不同的特性和价格。有关更多信息，请参阅 管道类型。
5. 在服务角色中，选择新建服务角色 CodePipeline 以允许在 IAM 中创建服务角色。
6. 将高级设置中的各项设置保留为默认值，然后选择下一步。
7. 在添加源舞台页面的源提供者中，选择GitHub。
8. 在连接下，选择一个现有连接或创建一个新连接。要创建或管理 GitHub 源操作的连接，请参阅GitHub 连接。
9. 在存储库中，选择您的 GitHub 源存储库。
10. 在 Branch 中，选择您的 GitHub 分支。
11. 保留源操作的其余默认值。选择下一步。
12. 在 Add build stage (添加构建阶段) 页面上，添加一个构建阶段：
    a. 在构建提供程序中，选择 AWS CodeBuild。对于 Region (区域)，请使用管道区域。
    b. 选择创建项目。
    c. 在项目名称中，输入此构建项目的名称。
    d. 在环境映像中，选择托管映像。对于操作系统，请选择 Ubuntu。
    e. 对于 Runtime (运行时) 和 Runtime version (运行时版本)，选择无服务器应用程序所需的运行时和版本。
    f. 对于服务角色，选择新建服务角色。
    g. 对于构建规范，选择使用 buildspec 文件。
    h. 选择 “继续” CodePipeline。这将打开 CodePipeline 控制台并创建一个使用存储库`buildspec.yml`中的进行配置的 CodeBuild 项目。构建项目使用服务角色来管理 AWS 服务 权限。此步骤可能需要几分钟时间。
    i. 选择下一步。
13. 在 Add deploy stage (添加部署阶段) 页面上，选择 Skip deploy stage (跳过部署阶段)，并通过再次选择 Skip (跳过) 接受警告消息。选择下一步。
14. 选择创建管道。您应看到一个显示源和构建阶段的示意图。

15. 向 CodeBuild 服务角色授予访问存储打包应用程序的 S3 存储桶的权限。

    a. 在新管道的 “构建” 阶段，选择CodeBuild。

    b. 选择 Build details (构建详细信息) 选项卡。

    c. 在环境中，选择 CodeBuild 服务角色以打开 IAM 控制台。

    d. 展开 `CodeBuildBasePolicy` 选项，然后选择 Edit policy (编辑策略)。

    e. 选择 JSON。

    f. 添加包含以下内容的新策略声明。该语句 CodeBuild 允许将对象放入存储已打包应用程序的 S3 存储桶中。将 *`bucketname`* 替换为您的 S3 存储桶的名称。

```
{
    "Effect": "Allow",
    "Resource": [
        "arn:aws:s3:::bucketname/*"
    ],
    "Action": [
        "s3:PutObject"
    ]
}
```

    g. 选择查看策略。

    h. 选择保存更改。

步骤 3：部署发布应用程序

按照以下步骤部署包含 Lambda 函数的应用程序，该函数执行到 AWS Serverless Application Repository的发布操作。此应用程序是 aws-serverless-codepipeline-serverlessrepo-pub lish。

> **Note**
>
> 您必须将应用程序部署到与您的管道 AWS 区域 相同的位置。

1. 转到应用程序页面，然后选择 Deploy (部署)。
2. 选择 I acknowledge that this app creates custom IAM roles (我确认此应用程序创建自定义 IAM 角色)。
3. 选择部署。

4. 选择 “查看 AWS CloudFormation 堆栈” 以打开 AWS CloudFormation 控制台。
5. 展开资源部分。你看 ServerlessRepoPublish，属于这种类型AWS::Lambda::Function。记下此资源的物理 ID 以供下一步使用。在中创建新的发布操作时，您将使用此物理 ID CodePipeline。

步骤 4：创建发布操作

请按照以下步骤在管道中创建发布操作。

1. 打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。
2. 在左侧导航部分中，选择要编辑的管道。
3. 选择编辑。
4. 在当前管道的最后一个阶段之后，选择 + Add stage (+ 添加阶段)。在 Stage name (阶段名称) 中，输入名称，例如 **Publish**，然后选择 Add stage (添加阶段)。
5. 在新阶段中，选择 + 添加操作组。
6. 输入操作名称。从 Action provider (操作提供程序) 的 Invoke (调用) 中，选择 AWS Lambda。
7. 从输入构件中选择BuildArtifact。
8. 从函数名称中，选择您在上一步中记下的 Lambda 函数的物理 ID。
9. 对操作选择 Save (保存)。
10. 对阶段选择 Done (完成)。
11. 在右上角，选择 Save (保存)。
12. 要验证您的管道，请在中对您的应用程序进行更改 GitHub。例如，更改 AWS SAM 模板文件`Metadata`部分中应用程序的描述。提交更改并将其推送到您的 GitHub 分支。这将触发您的管道运行。管道完成后，检查您的应用程序是否已根据 AWS Serverless Application Repository 中的更改进行了更新。

教程：将变量与 Lambda 调用操作一起使用

Lambda 调用操作可以使用来自另一个操作的变量作为其输入的一部分，并返回新变量及其输出。有关中操作变量的信息 CodePipeline，请参见Variables。

在本教程结束时，您将具备：

- Lambda 调用操作：

  - 消耗 CodeCommit 源`CommitId`操作中的变量
  - 输出三个新变量：`dateTime`、`testRunId` 和 `region`
- 一个手动审批操作，它使用 Lambda 调用操作中的新变量以提供测试 URL 和测试运行 ID
- 使用新操作更新的管道

主题

先决条件

在开始之前，您必须具有以下内容：

- 您可以创建或使用带有 CodeCommit 源代码的管道教程：创建简单的管道（CodeCommit存储库）。
- 编辑现有管道，使 CodeCommit 源操作具有命名空间。将命名空间 `SourceVariables` 分配给操作。

第 1 步：创建 Lambda 函数

使用以下步骤可创建 Lambda 函数和 Lambda 执行角色。创建 Lambda 函数后，可以将 Lambda 操作添加到您的管道中。

创建 Lambda 函数和执行角色

1. 登录 AWS Management Console 并打开 AWS Lambda 控制台，网址为 https://console.aws.amazon.com/lambda/。
2. 选择创建函数。将 Author from scratch (从头开始创作) 保持选中状态：
3. 在 Function name (函数名称) 中，输入您的函数的名称，例如 **`myInvokeFunction`**。在 Runtime (运行时) 中，保持选中默认选项。
4. 展开 Choose or create an execution role (选择或创建执行角色)。选择 Create a new role with basic Lambda permissions (创建具有基本 Lambda 权限的新角色)。
5. 选择创建函数。

6. 要使用来自另一个操作的变量，必须将它传递给 Lambda 调用操作配置中的 UserParameters。您将在本教程的后面部分的管道中配置操作，但假设将传递变量，您将添加代码。

```
const commitId =
event["CodePipeline.job"].data.actionConfiguration.configuration.UserParameters;
```

要生成新变量，请将输入上名为 outputVariables 的属性设置为 putJobSuccessResult。请注意，您不能将变量作为 putJobFailureResult 的一部分生成。

```
const successInput = {
  jobId: jobId,
  outputVariables: {
    testRunId: Math.floor(Math.random() * 1000).toString(),
    dateTime: Date(Date.now()).toString(),
    region: lambdaRegion
  }
};
```

在您的新函数中，将 Edit code inline (编辑内联代码) 保持选中状态，然后将以下示例代码粘贴到 index.js 的下方。

```
var AWS = require('aws-sdk');

exports.handler = function(event, context) {
    var codepipeline = new AWS.CodePipeline();

    // Retrieve the Job ID from the Lambda action
    var jobId = event["CodePipeline.job"].id;

    // Retrieve the value of UserParameters from the Lambda action configuration in
 CodePipeline,
    // in this case it is the Commit ID of the latest change of the pipeline.
    var params =
 event["CodePipeline.job"].data.actionConfiguration.configuration.UserParameters;

    // The region from where the lambda function is being executed.
    var lambdaRegion = process.env.AWS_REGION;

    // Notify CodePipeline of a successful job
    var putJobSuccess = function(message) {
        var params = {
```

```
            jobId: jobId,
            outputVariables: {
                testRunId: Math.floor(Math.random() * 1000).toString(),
                dateTime: Date(Date.now()).toString(),
                region: lambdaRegion
            }
        };
        codepipeline.putJobSuccessResult(params, function(err, data) {
            if(err) {
                context.fail(err);
            } else {
                context.succeed(message);
            }
        });
    };

    // Notify CodePipeline of a failed job
    var putJobFailure = function(message) {
        var params = {
            jobId: jobId,
            failureDetails: {
                message: JSON.stringify(message),
                type: 'JobFailed',
                externalExecutionId: context.invokeid
            }
        };
        codepipeline.putJobFailureResult(params, function(err, data) {
            context.fail(message);
        });
    };

    var sendResult = function() {
        try {
            console.log("Testing commit - " + params);

            // Your tests here

            // Succeed the job
            putJobSuccess("Tests passed.");
        } catch (ex) {
            // If any of the assertions failed then fail the job
            putJobFailure(ex);
        }
    };
```

```
    sendResult();
};
```

7. 选择保存。
8. 复制屏幕顶部的 Amazon 资源名称 (ARN)。
9. 最后一步，打开 AWS Identity and Access Management (IAM) 控制台，网址为 https://console.aws.amazon.com/iam/。修改 Lambda 执行角色以添加以下策略：。AWSCodePipelineCustomActionAccess有关创建 Lambda 执行角色或修改角色策略的步骤，请参阅第 2 步：创建 Lambda 函数 。

步骤 2：向您的管道添加 Lambda 调用操作和手动审批操作

在此步骤中，将 Lambda 调用操作添加到您的管道中。可以在名为 Test 的阶段中添加操作。操作类型是一个调用动作。然后，您可以在调用操作后面添加手动审批操作。

向管道添加 Lambda 操作和手动审批操作

1. 打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。

   将显示与您的 AWS 账户关联的所有管道的名称。选择要添加操作的管道。
2. 将 Lambda 测试操作添加到您的管道中。

   a. 要编辑您的管道，请选择编辑。在现有管道中的源操作后面添加一个阶段。输入阶段的名称，例如 **Test**。

   b. 在新阶段中，选择要添加操作的图标。在 Action name (操作名称) 中，输入调用操作的名称，例如 **Test_Commit**。

   c. 对于操作提供程序，选择 AWS Lambda。

   d. 在 Input artifacts (输入构件) 中，选择源操作输出构件的名称，例如 `SourceArtifact`。

   e. 在函数名称中，选择您创建的 Lambda 函数的名称。

   f. 在用户参数中，输入提 CodeCommit 交 ID 的变量语法。此时将创建输出变量，该变量解析为每次管道运行时要复核和审批的提交。

   ```
   #{SourceVariables.CommitId}
   ```

   g. 在 Variable namespace (变量命名空间) 中，添加命名空间名称，例如 **TestVariables**。

   h. 选择完成。

3. 将手动审批操作添加到您的管道中。

   a. 如果您的管道仍处于编辑模式，则在调用操作后面添加一个阶段。输入阶段的名称，例如 **Approval**。

   b. 在新阶段中，选择要添加操作的图标。在 Action name (操作名称) 中，输入审批操作的名称，例如 **Change_Approval**。

   c. 在 Action provider (操作提供程序) 中，选择 Manual approval (手动审批)。

   d. 在 URL for review (用于审查的 URL) 中，通过为 `CommitId` 变量和 `region` 变量添加变量语法，来构建 URL。确保您使用分配给提供输出变量的操作的命名空间。

      在此示例中，带有 CodeCommit 动作变量语法的 URL 具有默认命名空间`SourceVariables`。Lambda 区域输出变量具有 `TestVariables` 命名空间。此 URL 如下所示：

      ```
      https://#{TestVariables.region}.console.aws.amazon.com/codesuite/codecommit/
      repositories/MyDemoRepo/commit/#{SourceVariables.CommitId}
      ```

      在 Comments (注释) 中，通过添加 `testRunId` 变量的变量语法，构建审批消息文本。对于此示例，使用 Lambda `testRunId` 输出变量的变量语法的 URL 具有 `TestVariables` 命名空间。输入以下消息。

      ```
      Make sure to review the code before approving this action. Test Run ID:
       #{TestVariables.testRunId}
      ```

4. 选择 Done (完成) 以关闭操作的编辑屏幕，然后选择 Done (完成) 以关闭阶段的编辑屏幕。要保存管道，请选择 Done (完成)。已完成的管道现在包含一个结构，该结构带有源、测试、审批和部署阶段。

   选择 Release change (发布更改) 以通过管道结构运行最新更改。

5. 当管道到达手动审批阶段时，选择 Review (审核)。解析的变量显示为提交 ID 的 URL。您的审批人可以选择该 URL 来查看提交。

6. 当成功运行管道后，您还可以在操作执行历史记录页面上查看变量值。

教程：在管道中使用 AWS Step Functions 调用操作

您可以使用 AWS Step Functions 来创建和配置状态机。本教程介绍如何将调用操作添加到管道以便从管道激活状态机执行。

在本教程中，您将执行以下任务：

- 在中创建标准状态机 AWS Step Functions。
- 直接输入状态机输入 JSON。还可以将状态机输入文件上传到 Amazon Simple Storage Service (Amazon S3) 桶。
- 通过添加状态机操作来更新管道。

主题

先决条件：创建或选择简单管道

在本教程中，您将向现有管道添加调用操作。您可以使用在教程：创建一个简单的管道（S3 存储桶）或教程：创建简单的管道（CodeCommit存储库）中创建的管道。

您可以将现有管道与一个源操作和至少一个两阶段结构结合使用，但在此示例中不使用源构件。

> **Note**
>
> 您可能需要通过添加运行此操作所需的其他权限来更新管道所使用的服务角色。为此，请打开 AWS Identity and Access Management (IAM) 控制台，找到该角色，然后将权限添加到该角色的策略中。有关更多信息，请参阅 向 CodePipeline 服务角色添加权限。

步骤 1：创建示例状态机

在 Step Functions 控制台中，使用 `HelloWorld` 示例模板创建状态机。有关说明，请参阅 AWS Step Functions 开发者指南 中的创建状态机。

步骤 2：将 Step Functions 调用操作添加到管道

将 Step Functions 调用操作添加到管道，如下所示：

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

将显示与您的 AWS 账户关联的所有管道的名称。

2. 在 Name 中，选择您要编辑的管道的名称。这将打开管道的详细视图，包括管道每个阶段中每个操作的状态。
3. 在管道详细信息页中，选择编辑。
4. 在简单管道的第二个阶段，选择 Edit stage (编辑阶段)。选择删除。这将立即删除您不再需要的第二个阶段。
5. 在示意图底部，选择 + 添加阶段。
6. 在 Stage name (阶段名称) 中，输入阶段的名称，例如 **`Invoke`**，然后选择 Add stage(添加阶段)。
7. 选择 + 添加操作组。
8. 在 Action name (操作名称) 中，输入名称，例如 **`Invoke`**。
9. 在操作提供程序中，选择 AWS Step Functions。允许区域默认为管道区域。
10. 在 Input artifacts (输入构件) 中，选择 `SourceArtifact`。
11. 在 State machine ARN (状态机 ARN) 中，选择之前创建的状态机的 Amazon 资源名称 (ARN)。
12. （可选）在 Execution name prefix (执行名称前缀) 中，输入要添加到状态机执行 ID 的前缀。
13. 在 Input type (输入类型) 中，选择 Literal (文本)。
14. 在 Input (输入) 中，输入 `HelloWorld` 示例状态机所需的输入 JSON。

> **Note**
>
> 状态机执行的输入与中 CodePipeline 用来描述操作输入工件的术语不同。

在此示例中，输入以下 JSON：

```
{"IsHelloWorldExample": true}
```

15. 选择完成。
16. 在所编辑的阶段上，选择 Done (完成)。在 AWS CodePipeline 窗格中，选择保存，然后选择警告消息上的保存。
17. 要提交所做的更改并开始管道执行，请选择发布更改，然后选择发布。

18. 在已完成的管道上，选择调用操作中的 AWS Step Functions。在 AWS Step Functions 控制台中，查看您的状态机执行 ID。ID 显示您的状态机名称 `HelloWorld` 和带有前缀 `my-prefix` 的状态机执行 ID。

```
arn:aws:states:us-west-2:account-ID:execution:HelloWorld:my-
prefix-0d9a0900-3609-4ebc-925e-83d9618fcca1
```

教程：创建 AWS AppConfig 用作部署提供者的管道

在本教程中，您将配置一个管道，该管道在部署阶段 AWS AppConfig 用作部署操作提供者，持续提供配置文件。

主题

先决条件

在开始本教程之前，您必须完成以下步骤：

- 本例将 S3 源用于您的管道。创建或使用已启用版本控制的 Amazon S3 桶。按照步骤 1：为您的应用程序创建一个 S3 存储桶中的说明创建 S3 存储桶。

步骤 1：创建您的 AWS AppConfig资源

在本节中，您将创建以下资源：

- 中的应用程序 AWS AppConfig 是一个逻辑代码单元，可为您的客户提供功能。
- 中的环境 AWS AppConfig 是由 AppConfig 目标组成的逻辑部署组，例如测试版或生产环境中的应用程序。
- 配置文件是一组影响应用程序行为的设置。配置文件使您 AWS AppConfig 能够在其存储位置访问您的配置。

- （可选）中的部署策略 AWS AppConfig 定义了配置部署的行为，例如在部署期间的任何给定时间应收到新部署配置的客户端的百分比。

创建应用程序、环境、配置文件和部署策略

1. 登录到 AWS Management Console。
2. 使用以下主题中的步骤在中创建您的资源 AWS AppConfig。
   - 创建应用程序。
   - 创建一个环境。
   - 创建 AWS CodePipeline 配置文件。
   - （可选）选择预定义的部署策略或创建自己的部署策略。

步骤 2：将文件上传到 S3 源桶

在本节中，创建一个或多个配置文件。然后将源文件压缩并推送到管道用于源阶段的桶。

创建配置文件

1. 在每个区域中为每个配置创建一个 `configuration.json` 文件。包括以下内容：

```
Hello World!
```

2. 使用以下步骤压缩和上传配置文件。

压缩和上传源文件

1. 创建包含文件的 .zip 文件并将文件命名为 .zip 文件 `configuration-files.zip`。例如，您的 .zip 文件可以使用以下结构：

```
.
### appconfig-configurations
    ### MyConfigurations
        ### us-east-1
        #   ### configuration.json
        ### us-west-2
            ### configuration.json
```

2. 在桶的 Amazon S3 控制台中，选择上传，然后按照说明上传 .zip 文件。

步骤 3：创建管道

在此部分中，您将使用以下操作创建管道：

- 一个具有 Amazon S3 操作的源阶段，其中源构件是用于配置的文件。
- 包含部署操作的 AppConfig 部署阶段。

使用向导创建管道

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在欢迎页面、入门页面或管道页面上，选择创建管道。
3. 在步骤 1：选择管道设置的管道名称中，输入 **MyAppConfigPipeline**。
4. 在本教程中，为管道类型选择 V1。也可以选择 V2；但请注意，不同管道类型具有不同的特性和价格。有关更多信息，请参阅 管道类型。
5. 在服务角色中，选择新建服务角色 CodePipeline 以允许在 IAM 中创建服务角色。
6. 将高级设置中的各项设置保留为默认值，然后选择下一步。
7. 在步骤 2: 添加源阶段的源提供程序中，选择 Amazon S3。在桶中，选择 S3 源桶的名称。

   在 S3 对象键中，输入您的 .zip 文件：`configuration-files.zip`。

   选择下一步。
8. 在步骤 3：添加构建阶段中，选择跳过构建阶段，并通过再次选择跳过接受警告消息。

   选择下一步。
9. 在步骤 4：添加部署阶段中：

   a. 在部署提供商中，选择AWS AppConfig。

   b. 在应用程序中，选择您在中创建的应用程序的名称 AWS AppConfig。该字段显示应用程序的 ID。

   c. 在环境中，选择您在中创建的环境的名称 AWS AppConfig。该字段显示您的环境的 ID。

   d. 在配置配置文件中，选择您在中创建的配置文件的名称 AWS AppConfig。该字段显示您的配置文件的 ID。

e. 在部署策略中，选择部署策略的名称。这可以是您在中创建的部署策略， AppConfig 也可以是您从中预定义的部署策略中选择的策略 AppConfig。该字段显示您的部署策略的 ID。

f. 在输入构件配置路径中，输入文件路径。确保您的输入构件配置路径与 S3 桶 .zip 文件中的目录结构相匹配。在此示例中，输入以下文件路径：`appconfig-configurations/MyConfigurations/us-west-2/configuration.json`。

g. 选择下一步。

10. 在步骤 5：审核中，查看信息，然后选择创建管道。

步骤 4：对任何源文件进行更改并验证部署

对源文件进行更改，然后将更改上传到桶。这将触发您的管道运行。通过查看版本来验证您的配置是否可用。

教程：使用带有 GitHub 管道源的完整克隆

您可以在中为 GitHub 源操作选择完全克隆选项 CodePipeline。使用此选项可在管道构建操作中运行 Git 元数据的 CodeBuild 命令。

在本教程中，您将创建一个连接到 GitHub 仓库的管道，对源数据使用完整克隆选项，并运行一个用于克隆仓库并对仓库执行 Git 命令的 CodeBuild 构建。

> **Note**
>
> 此功能不适用于亚太地区（香港）、非洲（开普敦）、中东（巴林）、欧洲（苏黎世）或 AWS GovCloud （美国西部）区域。要参考其他可用操作，请参阅 产品和服务与 CodePipeline。有关在欧洲地区（米兰）区域使用此操作的注意事项，请参阅CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、GitLab .com 和 GitLab 自我管理操作中的说明。

主题

先决条件

在开始之前，您必须执行以下操作：

- 使用您的 GitHub 账户创建 GitHub 存储库。
- 准备好您的 GitHub 凭证。当你使用 AWS Management Console 来建立连接时，系统会要求你使用自己的 GitHub 凭据登录。

步骤 1：创建自述文件

创建存储 GitHub 库后，使用以下步骤添加自述文件。

1. 登录您的 GitHub 存储库并选择您的存储库。
2. 要创建新文件，请选择添加文件 > 创建新文件。将文件命名为 `README.md` 文件并添加以下文本。

   ```
   This is a GitHub repository!
   ```

3. 选择提交更改。

   确保 `README.md` 文件位于存储库的根级别。

步骤 2：创建管道并构建项目

在此部分中，您将使用以下操作创建管道：

- 与您的 GitHub 仓库连接和操作的源阶段。
- 带有生成操作的 AWS CodeBuild 生成阶段。

使用向导创建管道

1. 通过 https://console.aws.amazon.com/codepipeline/ 登录主 CodePipeline 机。
2. 在欢迎页面、入门页面或管道页面上，选择创建管道。
3. 在步骤 1：选择管道设置的管道名称中，输入 **`MyGitHubPipeline`**。
4. 在本教程中，为管道类型选择 V1。也可以选择 V2；但请注意，不同管道类型具有不同的特性和价格。有关更多信息，请参阅 管道类型。
5. 在服务角色中，选择新建服务角色。

> **Note**
>
> 如果您选择使用现有的 CodePipeline 服务角色，请确保已`codeconnections:UseConnection`将 IAM 权限添加到您的服务角色策略中。有关 CodePipeline 服务角色的说明，请参阅为 CodePipeline 服务角色添加权限。

6. 在高级设置下，保留原定设置值。在构件存储中，选择默认位置以将默认构件存储（如指定为默认值的 Amazon S3 项目存储桶）用于为管道选择的区域中的管道。

> **Note**
>
> 这不是源代码的源存储桶。这是管道的项目存储。每个管道都需要一个单独的构件存储，例如 S3 存储桶。

选择下一步。

7. 在步骤 2：添加源阶段页面上，添加源阶段：

   a. 在源提供程序中，选择 GitHub （版本 2）。

   b. 在连接下，选择一个现有连接或创建一个新连接。要创建或管理 GitHub 源操作的连接，请参阅GitHub 连接。

   c. 在存储库名称中，选择存储 GitHub库的名称。

   d. 在分支名称中，键入要使用的存储库分支。

   e. 确保选择了在源代码更改时启动管道选项。

   f. 在输出构件格式下，选择完整克隆，对源存储库启用 Git 克隆选项。只有提供的操作 CodeBuild 才能使用 Git 克隆选项。在本教程步骤 3：更新 CodeBuild 服务角色策略以使用连接中，您将使用更新 CodeBuild 项目服务角色使用此选项的权限。

选择下一步。

8. 在添加构建阶段，添加一个构建阶段：

   a. 在构建提供程序中，选择 AWS CodeBuild。允许区域默认为管道区域。

   b. 选择创建项目。

   c. 在项目名称中，输入此构建项目的名称。

   d. 在环境映像中，选择托管映像。对于操作系统，选择 Ubuntu。

e. 对于运行时，选择标准。对于映像，选择 aws/codebuild/standard:5.0。

f. 对于服务角色，选择新建服务角色。

> **Note**
>
> 记下您的 CodeBuild 服务角色的名称。在本教程的最后一步，您会用到此角色名称。

g. 在构建规范下，为构建规范选择插入构建命令。选择切换到编辑器，然后将以下内容粘贴到构建命令。

> **Note**
>
> 在构建规范的 env 部分中，确保启用了 Git 命令的凭证助手，如本示例所示。

```
version: 0.2

env:
  git-credential-helper: yes
phases:
  install:
    #If you use the Ubuntu standard image 2.0 or later, you must specify
 runtime-versions.
    #If you specify runtime-versions and use an image other than Ubuntu
 standard image 2.0, the build fails.
    runtime-versions:
      nodejs: 12
      # name: version
    #commands:
      # - command
      # - command
  pre_build:
    commands:
      - ls -lt
      - cat README.md
  build:
    commands:
      - git log | head -100
      - git status
      - ls
      - git archive --format=zip HEAD > application.zip
```

```
  #post_build:
    #commands:
      # - command
      # - command
artifacts:
  files:
    - application.zip
    # - location
  #name: $(date +%Y-%m-%d)
  #discard-paths: yes
  #base-directory: location
#cache:
  #paths:
    # - paths
```

h. 选择 “继续” CodePipeline。这将返回到 CodePipeline控制台并创建一个使用您的构建命令进行配置的 CodeBuild 项目。构建项目使用服务角色来管理 AWS 服务 权限。此步骤可能需要几分钟时间。

i. 选择下一步。

9. 在步骤 4：添加部署阶段页面上，选择跳过部署阶段，并通过再次选择跳过接受警告消息。选择下一步。

10. 在步骤 5：审核中，选择创建管道。

步骤 3：更新 CodeBuild 服务角色策略以使用连接

初始管道运行将失败，因为必须更新 CodeBuild 服务角色才能使用连接。在服务角色策略中添加 `codeconnections:UseConnection` IAM 权限。有关在 IAM 控制台中更新策略的说明，请参阅添加连接 Bitbucket、 GitHub、En GitHub terprise Server 或 GitLab .com 的 CodeBuild GitClone 权限。

步骤 4：在构建输出中查看存储库命令

1. 成功更新服务角色后，在失败 CodeBuild 阶段选择 “重试”。

2. 管道成功运行后，在成功的构建阶段上，选择查看详细信息。

   在详细信息页面上，选择日志选项卡。查看编 CodeBuild 译输出。这些命令将输出所输入变量的值。

   这些命令输出 `README.md` 文件内容，列出目录中的文件，克隆存储库，查看日志，然后将存储库存档为 ZIP 文件。

教程：使用带有 CodeCommit 管道源的完整克隆

您可以在中为 CodeCommit 源操作选择完全克隆选项 CodePipeline。使用此选项 CodeBuild 允许在管道构建操作中访问 Git 元数据。

在本教程中，您将创建一个访问 CodeCommit 仓库的管道，对源数据使用完整克隆选项，并运行一个用于克隆仓库并对仓库执行 Git 命令的 CodeBuild 构建。

> **Note**
>
> CodeBuild 操作是唯一支持使用 Git 克隆选项提供的 Git 元数据的下游操作。此外，虽然您的管道可以包含跨账户操作，但操作和 CodeBuild 操作必须位于同一个账户中，完整克隆选项才能成功。 CodeCommit

主题

先决条件

在开始之前，您必须在与您的管道相同的 AWS 账户和区域中创建 CodeCommit 存储库。

步骤 1：创建自述文件

使用以下步骤将自述文件添加到您的源存储库。README 文件提供了供 CodeBuild 下游操作读取的示例源文件。

添加自述文件

1. 登录您的存储库并选择您的存储库。
2. 要创建新文件，请选择添加文件 > 创建文件。将文件命名为 `README.md` 文件并添加以下文本。

```
This is a CodeCommit repository!
```

3. 选择提交更改。

   确保 `README.md` 文件位于存储库的根级别。

步骤 2：创建管道并构建项目

在此部分中，您将使用以下操作创建管道：

- 带有源操作的 CodeCommit 源阶段。
- 带有生成操作的 AWS CodeBuild 生成阶段。

使用向导创建管道

1. 通过 https://console.aws.amazon.com/codepipeline/ 登录主 CodePipeline 机。
2. 在欢迎页面、入门页面或管道页面上，选择创建管道。
3. 在步骤 1：选择管道设置的管道名称中，输入 **`MyCodeCommitPipeline`**。
4. 在本教程中，为管道类型选择 V1。也可以选择 V2；但请注意，不同管道类型具有不同的特性和价格。有关更多信息，请参阅 管道类型。
5. 在服务角色中，执行下列操作之一：
   - 选择现有服务角色。
   - 选择您现有的 CodePipeline 服务角色。在服务角色策略中，该角色必须有 `codecommit:GetRepository` IAM 权限。请参阅为 CodePipeline 服务角色添加权限。
6. 在高级设置下，保留原定设置值。选择下一步。
7. 在步骤 2: 添加源阶段页面上，执行下列操作之一：

   a. 在源提供程序中，选择 CodeCommit。

   b. 在存储库名称中，选择存储库的名称。

   c. 在分支名称中，选择您的分支名称。

   d. 确保选择了在源代码更改时启动管道选项。

   e. 在输出构件格式下，选择完整克隆，对源存储库启用 Git 克隆选项。只有提供的操作 CodeBuild 才能使用 Git 克隆选项。

   选择下一步。
8. 在添加构建阶段中，执行以下操作：

a. 在构建提供程序中，选择 AWS CodeBuild。允许区域默认为管道区域。

b. 选择创建项目。

c. 在项目名称中，输入此构建项目的名称。

d. 在环境映像中，选择托管映像。对于操作系统，选择 Ubuntu。

e. 对于运行时，选择标准。对于映像，选择 aws/codebuild/standard:5.0。

f. 对于服务角色，选择新建服务角色。

> **Note**
>
> 记下您的 CodeBuild 服务角色的名称。在本教程的最后一步，您会用到此角色名称。

g. 在构建规范下，为构建规范选择插入构建命令。选择切换到编辑器，然后在构建命令下粘贴以下代码。

```
version: 0.2

env:
  git-credential-helper: yes
phases:
  install:
    #If you use the Ubuntu standard image 2.0 or later, you must specify
 runtime-versions.
    #If you specify runtime-versions and use an image other than Ubuntu
 standard image 2.0, the build fails.
    runtime-versions:
      nodejs: 12
      # name: version
    #commands:
      # - command
      # - command
  pre_build:
    commands:
      - ls -lt
      - cat README.md
  build:
    commands:
      - git log | head -100
      - git status
      - ls
      - git describe --all
```

```
  #post_build:
    #commands:
      # - command
      # - command
#artifacts:
  #files:
    # - location
  #name: $(date +%Y-%m-%d)
  #discard-paths: yes
  #base-directory: location
#cache:
  #paths:
    # - paths
```

h. 选择 “继续” CodePipeline。这将使您返回 CodePipeline 控制台并创建一个使用您的构建命令进行配置的 CodeBuild 项目。该构建项目使用服务角色来管理 AWS 服务 权限。此步骤可能需要几分钟时间。

i. 选择下一步。

9. 在步骤 4：添加部署阶段页面上，选择跳过部署阶段，并通过再次选择跳过接受警告消息。选择下一步。

10. 在步骤 5：审核中，选择创建管道。

步骤 3：更新 CodeBuild 服务角色策略以克隆存储库

初始管道运行将失败，因为您需要更新 CodeBuild 服务角色，使其具有从存储库中提取的权限。

在服务角色策略中添加 `codecommit:GitPull` IAM 权限。有关在 IAM 控制台中更新策略的说明，请参阅为 CodeCommit源操作添加 CodeBuild GitClone 权限。

步骤 4：在构建输出中查看存储库命令

查看构建输出

1. 成功更新服务角色后，在失败 CodeBuild 阶段选择 “重试”。

2. 管道成功运行后，在成功的构建阶段上，选择查看详细信息。

   在详细信息页面上，选择日志选项卡。查看编 CodeBuild 译输出。这些命令将输出所输入变量的值。

这些命令输出 README.md 文件内容，列出目录中的文件，克隆存储库，查看日志，然后运行 `git describe --all`。

教程：使用 AWS CloudFormation StackSets 部署操作创建管道

在本教程中，您将使用 AWS CodePipeline 控制台创建带有部署操作的管道，用于创建堆栈集和创建堆栈实例。管道运行时，模板会创建堆栈集，还会创建和更新部署了堆栈集的实例。

有两种方法可以管理堆栈集的权限：自我管理的 IAM 角色和托 AWS管的 IAM 角色。本教程提供了自托管权限的示例。

要在中最有效地使用 Stackset CodePipeline，您应该清楚地了解其背后的概念 AWS CloudFormation StackSets 及其工作原理。参见《AWS CloudFormation 用户指南》中的StackSets 概念。

主题

先决条件

对于堆栈集操作，您可以使用两个不同的账户：管理账户和目标账户。在管理员账户中创建堆栈集。在目标账户中创建属于堆栈集的单个堆栈。

使用管理员账户创建管理员角色

- 按照为堆栈集操作设置基本权限中的说明进行操作。您的角色必须命名为 **`AWSCloudFormationStackSetAdministrationRole`**。

在目标账户中创建服务角色

- 在信任管理员账户的目标账户中创建服务角色。按照为堆栈集操作设置基本权限中的说明进行操作。您的角色必须命名为 **AWSCloudFormationStackSetExecutionRole**。

步骤 1：上传示例 AWS CloudFormation 模板和参数文件

为您的堆栈集模板和参数文件创建源桶。下载示例 AWS CloudFormation 模板文件，设置参数文件，然后压缩文件，然后再上传到 S3 源存储桶。

> **Note**
>
> 即使唯一的源文件是模板，也要确保先压缩源文件，再上传到 S3 源桶。

创建 S3 源桶

1. 登录 AWS Management Console 并打开 Amazon S3 控制台，网址为 https://console.aws.amazon.com/s3/。
2. 选择创建存储桶。
3. 在桶名称中，输入桶的名称。

   在区域中，选择要在其中创建管道的区域。请选择创建存储桶。
4. 创建存储桶后，系统会显示成功横幅。选择转到存储桶详细信息。
5. 在属性选项卡上，选择版本控制。选择启用版本控制，然后选择保存。

创建 AWS CloudFormation 模板文件

1. 下载以下示例模板文件，用于生成堆栈集的 CloudTrail 配置:https://s3.amazonaws.com/cloudformation-stackset-sample-templates-us-east-1/EnableAWSCloudtrail.yml.
2. 将该文件保存为 `template.yml`。

创建 parameters.txt 文件

1. 创建包含部署参数的文件。参数是您要在运行时在堆栈中更新的值。以下示例文件将更新堆栈集的模板参数，以启用日志验证和全局事件。

```
[
  {
    "ParameterKey": "EnableLogFileValidation",
    "ParameterValue": "true"
  },
  {
    "ParameterKey": "IncludeGlobalEvents",
    "ParameterValue": "true"
  }
]
```

2. 将该文件保存为 parameters.txt。

创建 accounts.txt 文件

1. 创建包含账户（您要从中创建实例）的文件，如以下示例文件所示。

```
[
    "111111222222","333333444444"
]
```

2. 将该文件保存为 accounts.txt。

创建和上传源文件

1. 将文件合并为单个 ZIP 文件。您的文件应该在您的 ZIP 文件中如下所示。

```
template.yml
parameters.txt
accounts.txt
```

2. 将 ZIP 文件上传到 S3 桶。此文件是由 “创建管道” 向导为中的部署操作创建的源项目 CodePipeline。

步骤 2：创建管道

在此部分中，您将使用以下操作创建管道：

- 具有 S3 源操作的源阶段，其中源构件是您的模板文件和任何支持源文件。

- 一个部署阶段，其中包含用于创建 AWS CloudFormation 堆栈集的堆栈集部署操作。
- 带有 AWS CloudFormation 堆栈实例部署操作的部署阶段，用于在目标账户中创建堆栈和实例。

使用 CloudFormationStackSet 操作创建管道

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在欢迎页面、入门页面或管道页面上，选择创建管道。
3. 在步骤 1：选择管道设置的管道名称中，输入 **MyStackSetsPipeline**。
4. 在本教程中，为管道类型选择 V1。也可以选择 V2；但请注意，不同管道类型具有不同的特性和价格。有关更多信息，请参阅 管道类型。
5. 在服务角色中，选择新建服务角色 CodePipeline 以允许在 IAM 中创建服务角色。
6. 在构件存储中，保留默认值。

> **Note**
>
> 这不是源代码的源存储桶。这是管道的项目存储。每个管道都需要一个单独的构件存储，例如 S3 存储桶。创建或编辑管道时，管道区域中必须有一个工件存储桶，并且每个运行操作的 AWS 区域都必须有一个工件存储桶。
> 有关更多信息，请参阅 输入和输出构件 和 CodePipeline 管道结构参考。

选择下一步。

7. 在步骤 2：添加源阶段页面上，在源提供程序 中，选择 Amazon S3。
8. 在桶中，输入您在本教程中创建的 S3 源桶，例如 `BucketName`。在 S3 对象键中，输入 ZIP 文件的文件路径和文件名，例如 `MyFiles.zip`。
9. 选择下一步。
10. 在步骤 3：添加构建阶段中，选择跳过构建阶段，并通过再次选择跳过接受警告消息。

    选择下一步。
11. 在步骤 4：添加部署阶段中：

    a. 在部署提供程序中，选择 AWS CloudFormation 堆栈集。

    b. 在堆栈集名称中，为堆栈集输入名称。这是模板将创建的堆栈集的名称。

> **Note**
>
> 记下您的堆栈集名称。当您将第二个 StackSets 部署操作添加到管道中时，您将使用它。

c. 在模板路径中，输入构件名称和上传模板文件的文件路径。例如，使用默认源构件名称 SourceArtifact 输入以下内容。

```
SourceArtifact::template.yml
```

d. 在部署目标中，输入构件名称和上传账户文件的文件路径。例如，使用默认源构件名称 SourceArtifact 输入以下内容。

```
SourceArtifact::accounts.txt
```

e. 在部署目标中 AWS 区域，输入一个用于部署初始堆栈实例的区域，例如us-east-1。

f. 展开部署选项。在参数中，输入构件名称和上传参数文件的文件路径。例如，使用默认源构件名称 SourceArtifact 输入以下内容。

```
SourceArtifact::parameters.txt
```

要将参数作为文字输入而不是文件路径，请输入以下内容：

```
ParameterKey=EnableLogFileValidation,ParameterValue=true
ParameterKey=IncludeGlobalEvents,ParameterValue=true
```

g. 在功能中，选择 CAPABILITY_IAM 和 CAPABILITY_NAMED_IAM。

h. 在权限模式中，选择 SELF_MANAGED。

i. 在失效容限百分比中，输入 20。

j. 在最大并发百分比中，输入 25。

k. 选择下一步。

l. 选择创建管道。将显示您的管道。

m. 允许您的管道运行。

步骤 3：查看初始部署

查看初始部署的资源和状态。验证部署已成功创建堆栈集后，您可以将另一个操作添加到部署阶段。

查看资源

1. 打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。
2. 在管道下，选择您的管道并选择查看。该图显示了您的管道源和部署阶段。
3. 选择对 AWS CloudFormation 管道中CloudFormationStackSet操作的操作。您的堆栈集的模板、资源和事件显示在 AWS CloudFormation 控制台中。
4. 在左侧导航面板中，选择StackSets。在列表中，选择新的堆栈集。
5. 选择堆栈实例选项卡。验证已在 us-east-1 区域中为您提供的每个账户创建了一个堆栈实例。验证每个堆栈实例的状态为 CURRENT。

步骤 4：添加动 CloudFormationStackInstances作

在管道中创建下一个操作 AWS CloudFormation StackSets 以允许创建剩余堆栈实例。

在管道中创建下一操作

1. 打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。

   在管道下，选择您的管道并选择查看。该图显示了您的管道源和部署阶段。
2. 选择以编辑管道。管道将在编辑模式下显示。
3. 在部署阶段上，选择编辑。
4. 在 AWS CloudFormation 堆栈集部署操作下，选择添加操作组。
5. 在编辑操作页面上，添加操作详细信息：

   a. 在操作名称中，为操作输入名称。
   b. 在操作提供程序中，选择 AWS CloudFormation 堆栈实例。
   c. 在输入构件中，选择 SourceArtifact。
   d. 在堆栈集名称中，为堆栈集输入名称。这是您在第一个操作中提供的堆栈集的名称。
   e. 在部署目标中，输入构件名称和上传账户文件的文件路径。例如，使用默认源构件名称 SourceArtifact 输入以下内容。

```
SourceArtifact::accounts.txt
```

f. 在部署目标中 AWS 区域，输入用于部署剩余堆栈实例的区域，`eu-central-1`例如`us-east-2`和如下：

```
us-east2, eu-central-1
```

g. 在失效容限百分比中，输入 20。

h. 在最大并发百分比中，输入 25。

i. 选择保存。

j. 手动发布更改。更新的管道将在“部署”阶段显示两个操作。

步骤 5：查看部署的堆栈集资源

查看堆栈集部署的资源和状态。

查看资源

1. 打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。
2. 在管道下，选择您的管道，然后选择查看。该图显示了您的管道源和部署阶段。
3. 选择对 AWS CloudFormation 管道中**`AWS CloudFormation Stack Instances`**操作的操作。您的堆栈集的模板、资源和事件显示在 AWS CloudFormation 控制台中。
4. 在左侧导航面板中，选择StackSets。在列表中，选择您的堆栈集。
5. 选择堆栈实例选项卡。验证已在预期区域中创建或更新您提供的每个账户的所有剩余堆栈实例。验证每个堆栈实例的状态为 CURRENT。

第 6 步：更新您的堆栈集

更新您的堆栈集并将更新部署到实例。在此示例中，您还将更改您指定进行更新的部署目标。不属于更新的实例将变为已过期状态。

1. 打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。
2. 在管道下，选择您的管道，然后选择编辑。在部署阶段上，选择编辑。
3. 选择编辑管道中的 AWS CloudFormation 堆栈集操作。在描述中，使用堆栈集的新描述覆盖现有描述。
4. 选择编辑管道中的 AWS CloudFormation 堆栈实例操作。在部署目标中 AWS 区域，删除创建操作时输入的`us-east-2`值。

5. 保存更改。选择发布更改以运行管道。
6. 在 AWS CloudFormation中打开您的操作。选择 “StackSet 信息” 选项卡。在StackSet 描述中，验证是否显示了新的描述。
7. 选择堆栈实例选项卡。在状态下，验证 us-east-2 中堆栈实例的状态为 OUTDATED。

CodePipeline 最佳实践和用例

以下各节描述了的最佳实践 CodePipeline。

主题

的用例 CodePipeline

您可以创建与其他管道集成的管道 AWS 服务。这些产品可以是 AWS 服务 Amazon S3，也可以是第三方产品，例如 GitHub。本节提供了使用 CodePipeline 不同的产品集成自动发布代码的示例。有关按操作类型 CodePipeline 组织的集成的完整列表，请参阅CodePipeline 管道结构参考。

主题

CodePipeline 与 Amazon S3 一起使用 AWS CodeCommit，以及 AWS CodeDeploy

创建管道时，与 AWS 产品和服务 CodePipeline 集成，这些产品和服务在管道的每个阶段都充当操作提供者。当您在向导中选择阶段时，必须选择源阶段和至少一个生成或部署阶段。该向导将为您创建具有无法更改的默认名称的阶段。这些是在向导中设置完整的三阶段管道时创建的阶段名称：

- 默认名称为“Source”的源操作阶段。
- 默认名称为“Build”的生成操作阶段。
- 默认名称为“Staging”的部署操作阶段。

您可以使用本指南中的教程创建管道并指定阶段：

- 教程：创建一个简单的管道（S3 存储桶）中的步骤可帮助您使用向导创建具有两个默认阶段的管道：“Source”和“Staging”，其中您的 Amazon S3 存储库是源提供程序。本教程创建了一个管道，用于 AWS CodeDeploy 将示例应用程序从 Amazon S3 存储桶部署到运行 Amazon Linux 的 Amazon EC2 实例。
- 中的步骤教程：创建简单的管道（CodeCommit存储库）可帮助您使用向导创建带有 “源” 阶段的管道，该阶段使用您的 AWS CodeCommit 存储库作为源提供程序。本教程创建了一个管道，用于 AWS CodeDeploy 将示例应用程序从 AWS CodeCommit 存储库部署到运行 Amazon Linux 的 Amazon EC2 实例。

CodePipeline 与第三方操作提供者（GitHub和 Jenkins）一起使用

您可以创建与第三方产品（例如 GitHub 和 Jenkins）集成的管道。教程：创建一个四阶段管道 中的步骤将向您演示如何创建一个管道，以便：

- 从 GitHub 存储库中获取源代码，
- 使用 Jenkins 生成和测试源代码，
- 用于 AWS CodeDeploy 将构建和测试的源代码部署到运行亚马逊 Linux 或微软 Windows Server 的亚马逊 EC2 实例。

CodePipeline 与一起 AWS CodeStar 使用在代码项目中生成管道

AWS CodeStar 是一项基于云的服务，它为管理软件开发项目提供了统一的用户界面 AWS。 AWS CodeStar 与一起将 AWS 资源整合 CodePipeline 到项目开发工具链中。您可以使用 AWS CodeStar 控制面板自动创建完整代码项目所需的管道、存储库、源代码、构建规范文件、部署方法以及托管实例或无服务器实例。

要创建 AWS CodeStar 项目，您需要选择编码语言和要部署的应用程序类型。您可以创建以下项目类型：Web 应用程序、Web 服务或 Alexa 技能。

您可以随时将首选 IDE 集成到 AWS CodeStar 仪表板中。您还可以添加和删除团队成员并管理项目中团队成员的权限。有关向您展示 AWS CodeStar 如何使用为无服务器应用程序创建示例管道的教程，请参阅中的教程：创建和管理无服务器项目。 AWS CodeStar

CodePipeline 用于编译、生成和测试代码 CodeBuild

CodeBuild 是云端的托管生成服务，允许您在没有服务器或系统的情况下生成和测试代码。CodePipeline CodeBuild 与配合使用可自动通过管道运行修订，以便在源代码发生变化时持续交付软件版本。 有关更多信息，请参阅 CodePipeline 与一起使用 CodeBuild 来测试代码和运行构建。

CodePipeline 与 Amazon ECS 配合使用，将基于容器的应用程序持续交付到云端

Amazon ECS 是一种容器管理服务，可让您将基于容器的应用程序部署到云中的 Amazon ECS 实例。CodePipeline 与 Amazon ECS 配合使用，通过管道自动运行修订，以便在源映像存储库发生变化时持续部署基于容器的应用程序。有关更多信息，请参阅教程：使用进行持续部署 CodePipeline。

CodePipeline 与 Elastic Beanstalk 配合使用，可将 Web 应用程序持续交付到云端

Elastic Beanstalk 是一项计算服务，可让您将 Web 应用程序和服务部署到 Web 服务器。CodePipeline 与 Elastic Beanstalk 配合使用，可将 Web 应用程序持续部署到您的应用程序环境中。您还可以使用 AWS CodeStar 创建带有 Elastic Beanstalk 部署操作的管道。

CodePipeline 与一起使用可 AWS Lambda 持续交付基于 Lambda 和无服务器的应用程序

您可以使用 AWS Lambda 和来调 CodePipeline 用 AWS Lambda 函数，如部署无服务器应用程序中所述。您还可以使用 AWS Lambda 和 AWS CodeStar 来创建用于部署无服务器应用程序的管道。

CodePipeline 与 AWS CloudFormation 模板一起使用，持续交付到云端

您可以 AWS CloudFormation 与一起使用 CodePipeline 以实现持续交付和自动化。有关更多信息，请参阅使用持续交付 CodePipeline。 AWS CloudFormation 还用于为中创建的管道创建模板 AWS CodeStar。

标记资源

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）。标签键区分大小写。
- 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。

标签可帮助您识别和整理 AWS 资源。许多都 AWS 服务 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，对于分配给 Amazon S3 源桶的管道，您可以分配相同的标签。

有关使用标签的提示，请参阅AWS 回答博客上的AWS 标记策略文章。

您可以在中标记以下资源类型 CodePipeline：

- 将管道标记为 CodePipeline
- 在中标记自定义操作 CodePipeline

您可以使用 AWS CLI、 CodePipeline API 或 AWS 软件开发工具包来：

- 在您创建管道、自定义操作或 Webhook 时将标签添加到其上。
- 添加、管理和删除管道、自定义操作或 Webhook 的标签。

您还可以使用控制台添加、管理和删除管道的标签。

除了通过标签标识、组织和跟踪资源之外，您可以在 IAM 策略中使用标签，帮助控制哪些人可以查看您的资源并与之交互。有关基于标签的访问策略示例，请参阅使用标签控制对 CodePipeline资源的访问权限。

CodePipeline 与亚马逊 Virtual Private Cloud 配合使用

AWS CodePipeline 现在支持由提供支持的亚马逊虚拟私有云（亚马逊 VPC）终端节点AWS PrivateLink。这意味着您可以 CodePipeline 通过 VPC 中的私有终端节点直接连接，从而将所有流量保留在您的 VPC 和 AWS 网络内。

Amazon VPC 可用于在您定义的虚拟网络中启动 AWS 资源。 AWS 服务 利用 VPC，您可以控制网络设置，例如：

- IP 地址范围
- 子网
- 路由表
- 网络网关

接口 VPC 终端节点由 AWS PrivateLink一种 AWS 技术提供支持，该技术可促进 AWS 服务 使用弹性网络接口与私有 IP 地址之间的私有通信。要将您的 VPC 连接到 CodePipeline，您需要为定义接口 VPC 终端节点 CodePipeline。这类端点使您能够将自己的 VPC 连接到 AWS 服务。该端点 CodePipeline 无需互联网网关、网络地址转换 (NAT) 实例或 VPN 连接即可提供可靠、可扩展的连接。有关设置 VPC 的信息，请参阅 VPC 用户指南。

可用性

CodePipeline 目前支持以下 VPC 终端节点 AWS 区域：

- 美国东部（俄亥俄）
- 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）
- 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）
- 欧洲地区（米兰）*
- 欧洲地区（巴黎）

- 欧洲地区（斯德哥尔摩）
- 亚太地区（香港）*
- 亚太地区（孟买）
- 亚太地区（东京)
- 亚太地区（首尔）
- 亚太地区（新加坡）
- 亚太地区（悉尼）
- 南美洲（圣保罗）
- AWS GovCloud （美国西部）

* 您必须启用此区域，然后才能使用它。

为 CodePipeline 创建 VPC 端点

您可以使用 Amazon VPC 控制台创建 com.amazonaws.*region*.codepipeline VPC 端点。在控制台中，*##*是 AWS 区域 支持的区域标识符 CodePipeline，例如`us-east-2`美国东部（俄亥俄州）区域。有关更多信息，请参阅 Amazon VPC 用户指南中的创建接口终端节点。

使用您在登录到 AWS时指定的区域来预填充终端节点。如果您登录到其他区域，VPC 终端节点将使用新区域进行更新。

> **Note**
>
> 其他 AWS 服务 提供 VPC 支持并与之集成的产品 CodePipeline（例如 CodeCommit）可能不支持使用 Amazon VPC 终端节点进行集成。例如， CodePipeline 和之间的流量 CodeCommit 不能限制在 VPC 子网范围内。

排查 VPC 设置的问题

当您排查 VPC 问题时，请使用 Internet 连接错误消息中显示的信息帮助您发现、诊断和解决问题。

1. 确保您的互联网网关已连接到您的 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. 确保私有子网的路由表指向虚拟私有网关。
6. 确保使用的服务角色 CodePipeline 具有相应的权限。例如，如果 CodePipeline 没有使用亚马逊 VPC 所需的 Amazon EC2 权限，则可能会收到一条错误消息，上面写着 “意外 EC2 错误：” UnauthorizedOperation。

使用中的管道 CodePipeline

要在中定义自动发布流程 AWS CodePipeline，您需要创建一个管道，这是一个描述软件更改如何通过发布过程的工作流程结构。管道由您配置的阶段和操作组成。

> **Note**
>
> 在添加 “构建”、“部署”、“测试” 或 “调用” 阶段时 CodePipeline，除了随附的默认选项外，您还可以选择已创建的用于管道的自定义操作。自定义操作可用于运行内部开发的生成过程或测试套件等任务。将包含版本标识符，以帮助您区分供应商列表中的自定义操作的不同版本。有关更多信息，请参阅 在中创建和添加自定义操作 CodePipeline。

在创建管道之前，您必须先完成入门 CodePipeline中的步骤。

有关管道的更多信息，请参阅CodePipeline 概念 CodePipeline 教程、和，如果您想使用创建管道，请参阅、和CodePipeline 管道结构参考。 AWS CLI 要查看管道列表，请参阅在中查看管道和详细信息 CodePipeline。

主题

- 在中启动管道 CodePipeline
- 在中停止管道执行 CodePipeline
- 在中创建管道 CodePipeline
- 在中编辑管道 CodePipeline
- 在中查看管道和详细信息 CodePipeline
- 删除中的管道 CodePipeline
- 在中 CodePipeline 创建使用其他 AWS 账户资源的管道
- 迁移轮询管道以使用基于事件的更改检测
- 创建 CodePipeline 服务角色
- 将管道标记为 CodePipeline
- 创建通知规则

在中启动管道 CodePipeline

每个管道执行都可以基于不同的触发器启动。每个管道执行可以有不同类型的触发器，具体取决于管道的启动方式。每个执行的触发器类型都显示在管道的执行历史记录中。触发器类型可能取决于源操作提供方，如下所示：

> **Note**
>
> 不能为每个源操作指定多个触发器。

- 管道创建：创建管道后，管道执行将自动启动。这是执行历史记录中的 `CreatePipeline` 触发器类型。
- 修订对象的更改：此类别代表执行历史记录中的 `PutActionRevision` 触发器类型。
- 代码推送分支和提交的更改检测：此类别代表执行历史记录中的 `CloudWatchEvent` 触发器类型。当检测到源存储库中的源提交和分支发生更改时，您的管道会启动。此触发器类型使用自动化更改检测。使用此触发器类型的源操作提供者是 S3 和 CodeCommit。此类型也用于启动管道的计划。请参阅 按计划启动管道。
- 轮询源更改：此类别代表执行历史记录中的 `PollForSourceChanges` 触发器类型。当通过轮询检测到源存储库中的源提交和分支发生更改时，您的管道会启动。不建议使用此触发器类型，应改为使用自动化更改检测。使用此触发器类型的源操作提供者是 S3 和 CodeCommit。
- 第三方源的 Webhook 事件：此类别代表执行历史记录中的 `Webhook` 触发器类型。当 Webhook 事件检测到更改时，您的管道会启动。此触发器类型使用自动化更改检测。使用此触发器类型的源操作提供程序是为代码推送配置的连接（Bitbucket Cloud GitHub、E GitHub nterprise Server、GitLab .com 和 GitLab自我管理）。
- 第三方源的 WebhookV2 事件：此类别代表执行历史记录中的 `WebhookV2` 触发器类型。此类型适用于根据管道定义中定义的触发器触发的执行。当检测到含指定 Git 标签的版本时，您的管道会启动。您可以使用 Git 标签通过名称或其他标识符来标记提交，以帮助其他存储库用户了解其重要性。您还可以使用 Git 标签来标识存储库历史记录中的特定提交。此触发器类型禁用自动化更改检测。使用此触发类型的源操作提供程序是为 Git 标签（Bitbucket Cloud GitHub、En GitHub terprise Server 和 GitLab .com）配置的连接。
- 手动启动管道：此类别代表执行历史记录中的 `StartPipelineExecution` 触发器类型。您可以使用控制台或手动启动管道。 AWS CLI 有关信息，请参阅 手动启动管道。
- RollbackStage：此类别代表执行历史记录中的`RollbackStage`触发器类型。您可以使用控制台或手动或自动 AWS CLI 回滚舞台。有关信息，请参阅 配置阶段回滚。

当您在管道中添加使用自动化更改检测触发器类型的源操作时，这些操作将使用其他资源。由于更改检测需要这些额外的资源，所以将在单独的小节中详细介绍如何创建每个源操作。有关自动化更改检测所要求的每个源提供方和更改检测方法的详细信息，请参阅 源操作和更改检测方法。

主题

源操作和更改检测方法

向管道中添加源操作时，这些操作将使用表中描述的其他资源。

> **Note**
>
> CodeCommit 和 S3 源操作需要配置的更改检测资源（ EventBridge 规则），或者使用选项轮询存储库以获取源更改。对于具有 Bitbucket GitHub、或 E GitHub nterprise Server 源操作的管道，您无需设置 webhook 或默认进行轮询。连接操作会为您管理更改检测。

来源	使用其他资源？	步骤
Amazon S3	此源操作使用其他资源。使用 CLI 或创建 CloudFormation 此操作时，您还可以创建和管理这些资源。	请参阅在中创建管道 CodePipeline和亚马逊 S3 源代码操作 EventBridge 以及 AWS CloudTrail。
Bitbucket Cloud	此源操作使用连接资源。	请参阅 Bitbucket Cloud 连接。
AWS CodeCommit	亚马逊 EventBridge （推荐）。这是在控制台中创建或编辑 CodeCommit 源代码的管道的默认设置。	请参阅在中创建管道 CodePipeline和 CodeCommit 源操作和 EventBridge。
Amazon ECR	亚马逊 EventBridge。这是由向导为管道创建的，具有在控制台中创建或编辑的 Amazon ECR 源。	请参阅在中创建管道 CodePipeline和 Amazon ECR 源操作和 EventBridge 资源。

来源	使用其他资源？	步骤
GitHub 或 GitHub 企业云	此源操作使用连接资源。	请参阅 GitHub 连接。
GitHub 企业服务器	此源操作使用连接资源和主机资源。	请参阅 GitHub 企业服务器连接。
GitLab.com	此源操作使用连接资源。	请参阅 GitLab.com 连接。
GitLab 自我管理	此源操作使用连接资源和主机资源。	请参阅 用于 GitLab 自我管理的连接。

如果您的管道使用轮询功能，则可以更新该管道以使用推荐的检测方法。有关更多信息，请参阅 将轮询管道更新为采用建议的更改检测方法。

如果要为使用连接的源操作关闭更改检测，请参阅CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、 GitLab .com 和 GitLab 自我管理操作。

手动启动管道

默认情况下，在创建管道以及每次在源存储库中进行更改时，将会自动启动管道。但是，您可能希望再次通过管道重新运行最新的修订。您可以使用 CodePipeline 控制台或 start-pipeline-execution and 命令通过管道手动重新运行最新的修订版。 AWS CLI

主题

- 手动启动管道（控制台）
- 手动启动管道（CLI）

手动启动管道（控制台）

手动启动管道并通过管道运行最新的修订

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在名称中，选择要启动的管道的名称。

3. 在管道详细信息页中，选择发布更改。如果将管道配置为传递参数（管道变量），则选择发布更改会打开发布更改窗口。在管道变量中管道级变量的一个或多个字段中，输入要为此管道执行传递的一个或多个值。有关更多信息，请参阅 Variables。

   这会通过管道启动在源操作中指定的每个源位置中提供的最新修订。

手动启动管道（CLI）

手动启动管道并通过管道运行最新版本的项目

1. 打开终端（Linux、macOS 或 Unix）或命令提示符 (Windows)，并使用 AWS CLI 运行 start-pipeline-execution 命令，指定您要启动的管道的名称。例如，要通过名为以下的管道开始运行最后一次更改 *MyFirstPipeline*：

   ```
   aws codepipeline start-pipeline-execution --name MyFirstPipeline
   ```

   要启动在管道级配置变量的管道，请使用带有可选 --variables 参数的 start-pipeline-execution 命令启动管道并添加执行中将使用的变量。例如，要添加值为 1 的 `var1` 变量，请使用以下命令：

   ```
   aws codepipeline start-pipeline-execution --name MyFirstPipeline --variables
    name=var1,value=1
   ```

2. 要验证是否成功，请查看返回的对象。该命令将返回执行 ID，如下所示：

   ```
   {
       "pipelineExecutionId": "c53dbd42-This-Is-An-Example"
   }
   ```

   > **Note**
   >
   > 启动管道后，可以在 CodePipeline 控制台中或通过运行get-pipeline-state命令来监控其进度。有关更多信息，请参阅 查看管道（控制台） 和 查看管道详细信息和历史记录（CLI）。

按计划启动管道

您可以在中设置规则 EventBridge 以按计划启动管道。

创建安排管道启动的 EventBridge 规则（控制台）

创建以时间表作为事件源的 EventBridge 规则

1. 打开亚马逊 EventBridge 控制台，网址为 https://console.aws.amazon.com/events/。
2. 在导航窗格中，选择规则。
3. 选择创建规则，然后在规则详细信息下面选择计划。
4. 使用固定速率或表达式设置计划。有关更多信息，请参阅规则的计划表达式。
5. 在目标中，选择CodePipeline。
6. 为该计划输入管道执行的管道 ARN。

> **Note**
>
> 您可以在控制台的设置下找到管道 ARN。请参阅 查看管道 ARN 和服务角色 ARN（控制台）。

7. 选择以下选项之一来创建或指定一个 IAM 服务角色，该角色 EventBridge 授予调用与您的 EventBridge 规则关联的目标的权限（在本例中，目标是 CodePipeline）。
   - 选择 “为此特定资源创建新角色”，创建一个服务角色来授予启动管道执行的 EventBridge权限。
   - 选择使用现有角色输入服务角色，该角色授予启动管道执行的 EventBridge 权限。
8. 选择 Configure details（配置详细信息）。
9. 在配置规则详细信息页上，输入规则的名称和描述，然后选择状态以启用该规则。
10. 如果您对规则满意，请选择 Create rule。

创建安排管道启动的 EventBridge 规则 (CLI)

要使用创建规则，请调用put-rule命令，指定： AWS CLI

- 唯一地标识创建的规则的名称。在您创建的与 AWS 账户 CodePipeline 关联的所有管道中，此名称必须是唯一的。
- 规则的计划表达式。

创建以时间表作为事件源的 EventBridge 规则

1. 调用 put-rule 命令并包含 `--name` 和 `--schedule-expression` 参数。

示例：

以下示例命令用于--schedule-expression创建名为按计划MyRule2 EventBridge 进行筛选的规则。

```
aws events put-rule --schedule-expression 'cron(15 10 ? * 6L 2002-2005)' --name
 MyRule2
```

2. 授 EventBridge 予使用调 CodePipeline 用规则的权限。有关更多信息，请参阅使用适用于 Amazon EventBridge 的基于资源的政策。

   a. 使用以下示例创建允许 EventBridge担任服务角色的信任策略。将它命名为 `trustpolicyforEB.json`。

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Principal": {
                   "Service": "events.amazonaws.com"
               },
               "Action": "sts:AssumeRole"
           }
       ]
   }
   ```

   b. 使用以下命令创建 `Role-for-MyRule` 角色并附加信任策略。

   ```
   aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
    file://trustpolicyforEB.json
   ```

   c. 为名为 `MyFirstPipeline` 的管道创建权限策略 JSON，如此示例中所示。将权限策略命名为 `permissionspolicyforEB.json`。

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

```
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-west-2:80398EXAMPLE:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 使用以下命令将新的 CodePipeline-Permissions-Policy-for-EB 权限策略附加到您创建的 Role-for-MyRule 角色。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforCWE.json
```

使用源修订覆盖启动管道

您可以使用覆盖来启动一个管道，该管道具有您为管道执行提供的特定源修订 ID。例如，如果您想启动一个管道来处理来自您的 CodeCommit 来源的特定提交 ID，则可以在启动管道时将提交 ID 添加为替代。

源版本有四种类型revisionType：

- COMMIT_ID
- IMAGE_DIGEST
- S3_OBJECT_VERSION_ID
- S3_OBJECT_OBJECT_KEY

> **Note**
>
> 对于COMMIT_ID和IMAGE_DIGEST类型的源修订版，源修订版 ID 适用于存储库中所有分支的所有内容。

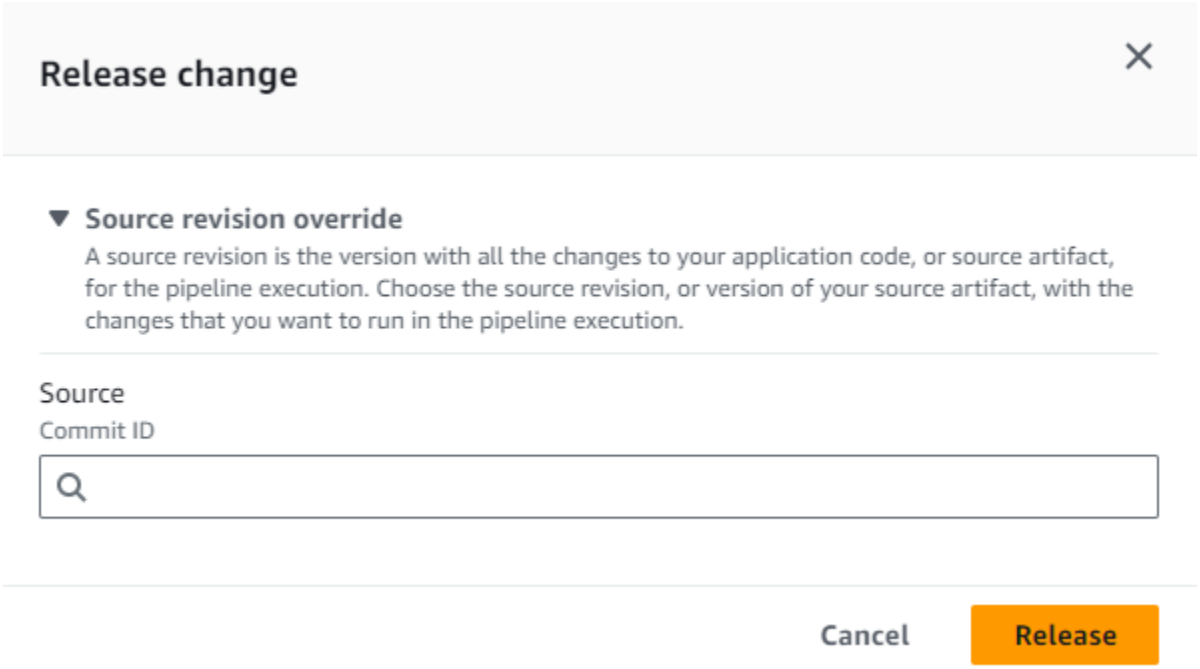
> **Note**
>
> 对于S3_OBJECT_VERSION_ID和S3_OBJECT_KEY类型的源修订版本，两种类型都可以单独使用，也可以将它们一起使用以使用特定的 ObjectKey 和版本标识来覆盖源代码。

主题

使用源修订覆盖启动管道（控制台）

手动启动管道并通过管道运行最新的修订

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在名称中，选择要启动的管道的名称。
3. 在管道详细信息页中，选择发布更改。选择发布更改将打开发布更改窗口。对于源修订覆盖，请选择箭头以展开字段。在源中，输入源修订 ID。例如，如果您的管道有 CodeCommit 源，请从要使用的字段中选择提交 ID。



使用源修订覆盖启动管道（CLI）

手动启动管道并通过管道运行构件的指定源修订 ID

1. 打开终端（Linux、macOS 或 Unix）或命令提示符 (Windows)，并使用 AWS CLI 运行 start-pipeline-execution 命令，指定您要启动的管道的名称。还可以使用 --source-revisions 参数来提供源修订 ID。源修订由 actionName、revisionType 和 revisionValue 组成。有效的 revisionType 值为 `COMMIT_ID | IMAGE_DIGEST | S3_OBJECT_VERSION_ID | S3_OBJECT_KEY`。

   在以下示例中，我们要通过一个名为 codecommit-pipeline 的管道，开始运行指定的更改。以下命令将指定一个名为 Source 的源操作、一个 `COMMIT_ID` 修订类型和一个 `78a25c18755ccac3f2a9eec099dEXAMPLE` 提交 ID。

   ```
   aws codepipeline start-pipeline-execution --name codecommit-pipeline --source-revisions
    actionName=Source,revisionType=COMMIT_ID,revisionValue=78a25c18755ccac3f2a9eec099dEXAMPLE
    --region us-west-1
   ```

2. 要验证是否成功，请查看返回的对象。该命令将返回执行 ID，如下所示：

   ```
   {
       "pipelineExecutionId": "c53dbd42-This-Is-An-Example"
   }
   ```

   > **Note**
   >
   > 启动管道后，可以在 CodePipeline 控制台中或通过运行get-pipeline-state命令来监控其进度。有关更多信息，请参阅 查看管道（控制台） 和 查看管道详细信息和历史记录（CLI）。

在中停止管道执行 CodePipeline

当管道执行开始通过管道运行时，它一次进入一个阶段，并在该阶段中的所有操作执行正在运行时锁定该阶段。必须以某种方式处理这些正在进行的操作，以便在停止管道执行时，允许完成或放弃操作。

可以使用两种方法停止管道执行：

- 停止并等待： AWS CodePipeline 等待停止执行，直到所有正在进行的操作都完成（也就是说，操作的Failed状态为Succeeded或）。此选项保留进行中的操作。执行处于 Stopping 状态，直到进行中的操作完成。然后执行处于 Stopped 状态。操作完成后阶段解锁。

  如果您选择停止并等待，并且在执行仍处于 Stopping 状态时改变主意，那么您可以选择放弃。

- 停止并放弃： AWS CodePipeline 停止执行，无需等待正在进行的操作完成。放弃进行中的操作时，执行在很短的时间内处于 Stopping 状态。执行停止后，操作执行处于 Abandoned 状态，而管道执行处于 Stopped 状态。阶段解锁。

  对于处于 Stopped 状态的管道执行，可以重试已停止执行的阶段中的操作。

> **Warning**
>
> 此选项会导致任务失败或任务次序混乱。

主题

停止管道执行（控制台）

您可以使用控制台停止管道执行。选择执行，然后选择停止管道执行的方法。

> **Note**
>
> 您也可以停止管道执行中的入站执行。要了解有关停止入站执行的更多信息，请参阅停止入站执行（控制台）。

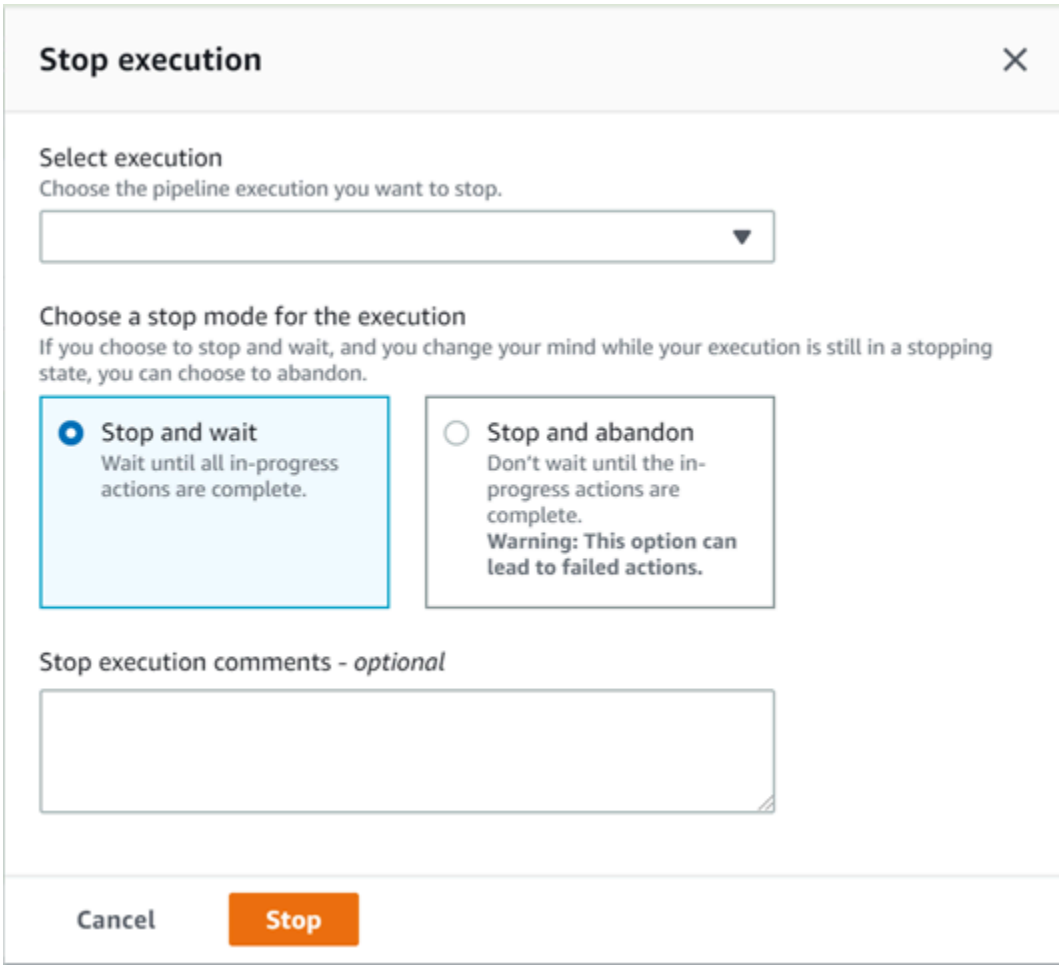
1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 请执行以下操作之一：

> **Note**
>
> 在停止执行之前，我们建议您禁用阶段前的转换。这样，当阶段由于停止执行而解锁时，该阶段不会接受后续管道执行。

- 在 Name (名称) 中，选择要停止执行的管道的名称。在管道详细信息页面上，选择 Stop execution (停止执行)。
- 选择 View history (查看历史记录)。在历史记录页面上，选择 Stop execution (停止执行)。

3. 在 Stop execution (停止执行) 页面上的 Select execution (选择执行) 下面，选择要停止的执行。

> **Note**
>
> 仅当执行仍在进行中时才会显示。不会显示已完成的执行。



4. 在 Select an action to apply to execution (选择要应用于执行的操作) 下，选择以下选项之一：

   - 要确保所有进行中的操作在完成之前不会停止执行，请选择 Stop and wait (停止并等待)。

     > **Note**
     >
     > 如果执行已处于 Stopping (正在停止) 状态，则无法选择“Stop and wait (停止并等待)”，但可以选择“Stop and abandon (停止并放弃)”。

   - 要停止而不等待进行中的操作完成，请选择 Stop and abandon (停止并放弃)。

     > **Warning**
     >
     > 此选项会导致任务失败或任务次序混乱。

5. （可选）输入注释。这些注释以及执行状态显示在执行的历史记录页面上。

6. 选择停止。

> **Important**
>
> 并且无法撤消。

7. 在管道可视化中查看执行状态，如下所示：

   - 如果您选择停止并等待，选定的执行将继续，直到完成进行中的操作。
     - 成功横幅消息显示在控制台顶部。
     - 在当前阶段，进行中的操作继续处于 InProgress 状态。操作正在进行时，管道执行处于 Stopping 状态。

       操作完成（即操作失败或成功）后，管道执行更改为 Stopped 状态，操作更改为 Failed 或 Succeeded 状态。您还可以在执行详细信息页面上查看操作状态。您可以在执行历史记录页面或执行详细信息页面上查看执行状态。
     - 管道执行短暂更改为 Stopping 状态，然后更改为 Stopped 状态。您可以在执行历史记录页面或执行详细信息页面上查看执行状态。
   - 如果您选择停止并放弃，则执行不会等待进行中的操作完成。
     - 成功横幅消息显示在控制台顶部。
     - 在当前阶段，进行中的操作将更改为 Abandoned 状态。您还可以在执行详细信息页面上查看操作状态。
     - 管道执行短暂更改为 Stopping 状态，然后更改为 Stopped 状态。您可以在执行历史记录页面或执行详细信息页面上查看执行状态。

   您可以在执行历史记录视图和详细历史记录视图中查看管道执行状态。

停止入站执行（控制台）

您可以使用控制台停止入站执行。入站执行是一种管道执行，它等待进入一个已禁用过渡的阶段。过渡启用后，InProgress 状态的入站执行将继续进入该阶段。Stopped 状态的入站执行不会进入该阶段。

> **Note**
>
> 入站执行停止后，无法重试。

如果未看到入站执行，说明禁用的阶段转换中没有待处理的执行。

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   将显示与您的 AWS 账户关联的所有管道的名称。
2. 选择要停止入站执行的管道的名称，执行以下任一操作：
   - 在管道视图中，选择入站执行 ID，然后选择停止执行。
   - 选择管道并选择查看历史记录。在执行历史记录中，选择入站执行 ID，然后选择停止执行。
3. 在停止执行模式中，按照上面部分中的步骤选择执行 ID 并指定停止方法。

   使用 get-pipeline-state 命令查看入站执行的状态。

停止管道执行 (CLI)

要使用手动停止管道，请使用带有以下参数的stop-pipeline-execution命令： AWS CLI

- 执行 ID（必需）
- 注释（可选）
- 管道名称（必需）
- 放弃标志（可选，默认为 false）

命令格式：

```
aws codepipeline stop-pipeline-execution --pipeline-name Pipeline_Name --pipeline-
execution-id Execution_ID [--abandon | --no-abandon] [--reason STOP_EXECUTION_REASON]
```

1. 打开终端（Linux、macOS 或 Unix）或命令提示符（Windows）。
2. 要停止管道执行，请选择以下选项之一：

- 要确保在所有进行中的操作完成之前不会停止执行，请选择停止并等待。您可以通过包括 no-abandon 参数来实现。如果未指定参数，则该命令默认为停止并等待。 AWS CLI 使用运行stop-pipeline-execution命令，指定管道的名称和执行 ID。例如，要停止名为、指定了 stop and *MyFirstPipeline*wait 选项的管道，请执行以下操作：

```
aws codepipeline stop-pipeline-execution --pipeline-name MyFirstPipeline --
pipeline-execution-id d-EXAMPLE --no-abandon
```

例如，要停止名为*MyFirstPipeline*、默认为 stop and wait 选项并选择包含注释的管道，请执行以下操作：

```
aws codepipeline stop-pipeline-execution --pipeline-name MyFirstPipeline --
pipeline-execution-id d-EXAMPLE --reason "Stopping execution after the build
 action is done"
```

> **Note**
>
> 如果执行已处于 Stopping (正在停止) 状态，则无法选择停止并等待。您可以选择停止并放弃已处于 Stopping (正在停止) 状态的执行。

- 要停止而不等待进行中的操作完成，请选择停止并放弃。包含 abandon 参数。 AWS CLI 使用运行stop-pipeline-execution命令，指定管道的名称和执行 ID。

例如，要停止名为*MyFirstPipeline*、指定放弃选项并选择包含注释的管道，请执行以下操作：

```
aws codepipeline stop-pipeline-execution --pipeline-name MyFirstPipeline --
pipeline-execution-id d-EXAMPLE --abandon --reason "Stopping execution for a bug
 fix"
```

停止入站执行 (CLI)

您可以使用 CLI 停止入站执行。入站执行是一种管道执行，它等待进入一个已禁用过渡的阶段。过渡启用后，InProgress 状态的入站执行将继续进入该阶段。Stopped 状态的入站执行不会进入该阶段。

> **Note**
>
> 入站执行停止后，无法重试。

如果未看到入站执行，说明禁用的阶段转换中没有待处理的执行。

要使用手动停止入站执行，请使用带有以下参数的stop-pipeline-execution命令： AWS CLI

- 入站执行 ID（必需）
- 注释（可选）
- 管道名称（必需）
- 放弃标志（可选，默认为 false）

命令格式：

```
aws codepipeline stop-pipeline-execution --pipeline-name Pipeline_Name --
pipeline-execution-id Inbound_Execution_ID [--abandon | --no-abandon] [--
reason STOP_EXECUTION_REASON]
```

按照上述过程中的步骤输入命令并指定停止方法。

使用 get-pipeline-state 命令查看入站执行的状态。

在中创建管道 CodePipeline

您可以使用 AWS CodePipeline 控制台或 AWS CLI 创建管道。管道必须包含至少两个阶段。管道的第一个阶段必须是源阶段。管道必须至少有一个其他阶段，可以是构建阶段或部署阶段。

您可以向管道中添加与您的管道不同的 AWS 区域中的操作。跨区域操作是指操作的提供者，而操作类型或提供者类型位于与您的管道不同的 AWS 区域的操作。 AWS 服务 有关更多信息，请参阅 在中添加跨区域操作 CodePipeline。

您还可以创建管道以生成基于容器的应用程序，并将 Amazon ECS 作为部署提供方以部署应用程序。在创建管道以使用 Amazon ECS 部署基于容器的应用程序之前，您必须创建一个映像定义文件，如映像定义文件参考中所述。

CodePipeline 当源代码变更被推送时，使用变更检测方法启动您的管道。这些检测方法基于源类型：

- CodePipeline 使用 Amazon Ev CloudWatch ents 来检测您的 CodeCommit 源存储库和分支或 S3 源存储桶中的更改。

> **Note**
>
> 在使用控制台创建或编辑管道时，将为您创建更改检测资源。如果您使用 AWS CLI 创建管道，则必须自己创建其他资源。有关更多信息，请参阅 CodeCommit 源操作和 EventBridge。

主题

创建管道（控制台）

要在控制台中创建管道，您必须提供源文件位置和有关您将用于操作的提供商的信息。

当您使用控制台创建管道时，必须包括一个源阶段和以下一个或两个阶段：

- 生成阶段。
- 部署阶段。

使用管道向导时， CodePipeline 会创建阶段（源代码、构建、暂存）的名称。这些名称不能更改。您可以对稍后添加的阶段使用更具体的名称（例如 BuildToGamma 或 DeployToProd）。

步骤 1：创建管道并为其命名

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 Welcome 页面上，选择 Create pipeline。

   如果这是您第一次使用 CodePipeline，请选择 “入门”。
3. 在步骤 1：选择管道设置页面上，在管道名称中，输入管道的名称。

   在单个 AWS 账户中，您在一个 AWS 区域中创建的每个管道都必须具有唯一的名称。名称可重用于不同区域的管道。

   > **Note**
   >
   > 创建管道后，便无法再更改其名称。有关其他限制的信息，请参阅中的配额 AWS CodePipeline。
4. 对于管道类型，选择以下选项之一：管道类型有不同的特点和价格。有关更多信息，请参阅 管道类型。

   - V1 类型的管道具有 JSON 结构，其中包含标准的管道、阶段和操作级参数。
   - V2 类型的管道与 V1 类型结构相同，并支持其他参数，例如 Git 标签上的触发器和管道级变量。
5. 在服务角色中，执行下列操作之一：

   - 选择 “新建服务角色” CodePipeline 以允许在 IAM 中创建新的服务角色。
   - 选择现有服务角色以使用已在 IAM 中创建的服务角色。在角色 ARN 中，从列表中选择您的服务角色 ARN。

   > **Note**
   >
   > 根据您的服务角色的创建时间，您可能需要更新其权限以支持其他权限 AWS 服务。有关信息，请参阅 向 CodePipeline 服务角色添加权限。

   有关服务角色及其策略语句的更多信息，请参阅管理 CodePipeline 服务角色。
6. （可选）在变量下，选择添加变量，以添加管道级变量。

有关管道级变量的更多信息，请参阅Variables。有关在管道执行时传递的管道级变量的教程，请参阅教程：使用管道级变量。

> **Note**
>
> 虽然添加管道级变量是可选的，但如果使用管道级变量指定了管道而没有提供变量值，管道执行将失败。

7. （可选）展开高级设置。
8. 在构件存储中，执行下列操作之一：
   a. 选择默认位置，在为管道选择的管道中使用默认项目存储，例如指定为默认的 S3 工件存储桶。 AWS 区域
   b. 如果您在管道所在的区域中已有构件存储（例如，S3 构件存储桶），请选择自定义位置。在存储桶中，选择存储桶名称。

> **Note**
>
> 这不是源代码的源存储桶。这是管道的项目存储。每个管道都需要一个单独的构件存储，例如 S3 存储桶。创建或编辑管道时，管道区域中必须有一个工件存储桶，并且每个运行操作的 AWS 区域都必须有一个工件存储桶。
> 有关更多信息，请参阅 输入和输出构件 和 CodePipeline 管道结构参考。

9. 在加密密钥中，执行下列操作之一：
   a. 要使用 CodePipeline 默认值 AWS KMS key 加密管道项目存储（S3 存储桶）中的数据，请选择默认 AWS 托管密钥。
   b. 要使用客户托管密钥对管道构件存储（S3 桶）中的数据进行加密，请选择客户托管密钥。选择密钥 ID、密钥 ARN 或别名 ARN。
10. 选择下一步。

步骤 2：创建源阶段

- 在步骤 2：添加源阶段页面上的源提供程序中，选择您的源代码存储到的存储库的类型，并指定其必需选项，然后选择下一步。

- 对于 Bitbucket Cloud GitHub（版本 2）、 GitHub 企业服务器、 GitLab .com 或 GitLab自行管理：

  1. 在连接下，选择一个现有连接或创建一个新连接。要创建或管理 GitHub 源操作的连接，请参阅GitHub 连接。
  2. 选择要用作管道的源位置的存储库。

     选择添加触发器或筛选触发器类型以启动管道。有关使用触发器的更多信息，请参阅筛选代码推送或拉取请求的触发器。有关使用 glob 模式筛选的更多信息，请参阅使用语法中的 glob 模式。
  3. 在输出构件格式中，为构件选择格式。

     - 要使用默认方法存储 GitHub 操作的输出对象，请选择CodePipeline默认。该操作访问存储库中的文件，并将工件 GitHub 存储在管道工件存储区的 ZIP 文件中。
     - 要存储包含存储库的 URL 引用的 JSON 文件，以便下游操作可以直接执行 Git 命令，请选择完全克隆。此选项只能由 CodeBuild 下游操作使用。

       如果选择此选项，则需要更新 CodeBuild 项目服务角色的权限，如所示故障排除 CodePipeline。如需查看教程以了解如何使用完整克隆选项，请参阅教程：使用带有 GitHub 管道源的完整克隆。

- 对于 Amazon S3：

  1. 在Amazon S3 位置中，提供 S3 存储桶名称和已启用版本控制的存储桶中对象的路径。存储桶名称和路径的格式与以下内容类似：

     ```
     s3://bucketName/folderName/objectName
     ```

     > **Note**
     >
     > 当 Amazon S3 是您的管道的源提供程序时，您可以将一个或多个源文件压缩到单个 .zip 文件中，然后将 .zip 文件上传到源桶。您也可以上传单个解压缩的文件；但是，需要 .zip 文件的下游操作将失败。
  2. 选择 S3 源存储桶后， CodePipeline 创建 Amazon E CloudWatch vents 规则和要为此管道创建的 AWS CloudTrail 跟踪。接受更改检测选项下面的默认值。这 CodePipeline 允许使用 Amazon Ev CloudWatch en AWS CloudTrail ts 并检测新管道的更改。选择下一步。

- 对于 AWS CodeCommit：

- 在存储库名称中，选择要用作管道源位置的 CodeCommit 存储库的名称。在分支名称中，从下拉列表中选择要使用的分支。
- 在输出构件格式中，为构件选择格式。
  - 要使用默认方法存储 CodeCommit 操作的输出对象，请选择CodePipeline默认。该操作访问存储库中的文件，并将工件 CodeCommit 存储在管道工件存储区的 ZIP 文件中。
  - 要存储包含存储库的 URL 引用的 JSON 文件，以便下游操作可以直接执行 Git 命令，请选择完全克隆。此选项只能由 CodeBuild 下游操作使用。

    如果选择此选项，则需要向您的 CodeBuild 服务角色添加`codecommit:GitPull`权限，如所示为 CodeCommit源操作添加 CodeBuild GitClone 权限。您还需要为 CodePipeline 服务角色添加`codecommit:GetRepository`权限，如所示向 CodePipeline 服务角色添加权限。如需查看教程以了解如何使用完整克隆选项，请参阅教程：使用带有 GitHub 管道源的完整克隆。
- 选择 CodeCommit 存储库名称和分支后，更改检测选项中会显示一条消息，显示要为此管道创建的 Amazon Ev CloudWatch ents 规则。接受更改检测选项下面的默认值。这 CodePipeline 允许使用 Amazon E CloudWatch vents 来检测您的新管道的更改。

- 对于 Amazon ECR：
  - 在存储库名称中，选择 Amazon ECR 存储库的名称。
  - 在映像标签中，指定映像名称和版本（如果与最新版本不同）。
  - 在输出构件中，选择默认输出对象，例如 MyApp，它包含您希望下一阶段使用的图像名称和存储库 URI 信息。

    有关使用 CodeDeploy蓝绿色部署（包括 Amazon ECR 源阶段）为 Amazon ECS 创建管道的教程，请参阅。教程：使用 Amazon ECR 源和 ECS 目标部署创建管道 CodeDeploy

当您在管道中包含一个 Amazon ECR 源阶段时，在您提交更改时，源操作生成 `imageDetail.json` 文件作为输出构件。有关 `imageDetail.json` 文件的信息，请参阅适用于 Amazon ECS 蓝绿部署的 imageDetail.json 文件。

> **Note**
>
> 对象和文件类型必须与您计划使用的部署系统兼容（例如，Elastic Beanstalk 或）。CodeDeploy受支持文件类型可能包括 .zip、.tar 和 .tgz 文件。有关 Elastic Beanstalk 的受支持容器类型的更多信息，请参阅自定义和配置 Elastic Beanstalk 环境和支持的平台。有

关使用部署修订版的更多信息 CodeDeploy，请参阅上传您的应用程序修订版和准备修订版。

步骤 3：创建生成阶段

如果您计划在创建部署阶段，则此步骤是可选的。

- 在步骤 3：添加生成阶段页面上，执行下列操作之一，然后选择下一步：
  - 如果您计划创建部署阶段，请选择跳过构建阶段。
  - 从构建提供程序中，选择构建服务的自定义操作提供程序，并提供该提供程序的配置详细信息。有关如何添加 Jenkins 作为构建提供程序的示例，请参阅教程：创建一个四阶段管道。
  - 从构建提供程序中，选择 AWS CodeBuild。

    在区域中，选择资源所在的 AWS 区域。Re g ion 字段指定为此操作类型和提供者类型创建 AWS 资源的位置。仅为操作提供方是 AWS 服务的操作显示此字段。“区域” 字段默认为与您的管道相同的 AWS 区域。

    在项目名称中选择您的构建项目。如果您已经在中创建了构建项目 CodeBuild，请选择它。或者，您可以在中创建构建项目， CodeBuild 然后返回此任务。按照《CodeBuild用户指南》中的“创建要使用的管道” CodeBuild 中的说明进行操作。

    在环境变量中，要将 CodeBuild环境变量添加到生成操作中，请选择添加环境变量。每个变量由三个条目组成：
    - 在名称中，输入环境变量的名称或键。
    - 在值中，输入环境变量的值。如果您为变量类型选择参数，请确保此值是您已经存储在 Sy AWS stems Manager 参数存储中的参数的名称。

      > **Note**
      >
      > 我们强烈不鼓励使用环境变量来存储敏感值，尤其是 AWS 证书。使用 CodeBuild 控制台或 AWS CLI 时，环境变量将以纯文本形式显示。对于敏感值，我们建议您改用参数 类型。

    - （可选）在类型中，输入环境变量的类型。有效值为明文或参数。默认值为明文。

    （可选）在构建类型中，选择下列选项之一：

- 要在单个构建操作执行中运行每次构建，请选择单次构建。
- 要在同一个构建操作执行中运行多次构建，请选择批处理构建。

（可选）如果您选择运行批处理构建，则可以选择将批处理中的所有构件合并到一个位置，将所有构建构件放入单个输出构件中。

步骤 4：创建部署阶段

如果已创建了构建阶段，则此步骤为可选步骤。

- 在步骤 4：添加部署阶段页面上，执行下列操作之一，然后选择下一步：
  - 如果您在上一步中创建了构建阶段，则选择跳过部署阶段。

    > **Note**
    >
    > 如果您已跳过构建阶段，则不会显示此选项。

  - 在部署提供程序中，选择您为部署提供程序创建的自定义操作。

    在区域中，仅对于跨区域操作，请选择创建资源的 AWS 区域。区域字段指定为此操作类型和提供方类型创建 AWS 资源的位置。此字段仅在操作提供方是 AWS 服务的情况下对操作显示。“区域” 字段默认为与您的管道相同的 AWS 区域。
  - 在部署提供程序中，适用于默认提供商的字段如下所示：
    - CodeDeploy

      在应用程序名称中，输入或选择现有 CodeDeploy 应用程序的名称。在部署组中，输入应用程序的部署组的名称。选择下一步。您也可以在 CodeDeploy控制台中创建应用程序、部署组或两者兼而有之。
    - AWS Elastic Beanstalk

      在应用程序名称中，输入或选择现有 Elastic Beanstalk 应用程序的名称。在环境名称中，输入应用程序的环境。选择下一步。您还可以在 Elastic Beanstalk 控制台中创建应用程序和/或环境。
    - AWS OpsWorks Stacks

在堆栈中，输入或选择要使用的堆栈的名称。在层中，选择目标实例所属的层。在 App 中，选择您要更新和部署的应用程序。如果您需要创建一个应用程序，请选择在 AWS OpsWorks 中创建一个新的。

有关将应用程序添加到堆栈和层中的信息 AWS OpsWorks，请参阅《AWS OpsWorks 用户指南》中的 “添加应用程序”。

有关如何使用中的简单管道 CodePipeline 作为在 AWS OpsWorks 层上运行的代码的源代码的 end-to-end 示例，请参阅 CodePipeline 与一起使用 AWS OpsWorks Stacks。

- AWS CloudFormation

  请执行以下操作之一：

  - 在操作模式下，选择创建或更新堆栈，输入堆栈名称和模板文件名，然后选择 AWS CloudFormation 要担任的角色的名称。（可选）输入配置文件的名称，然后选择 IAM 功能选项。
  - 在操作模式下，选择 “创建或替换更改集”，输入堆栈名称和更改集名称，然后选择 AWS CloudFormation 要担任的角色的名称。（可选）输入配置文件的名称，然后选择 IAM 功能选项。

  有关将 AWS CloudFormation 功能集成到管道中的信息 CodePipeline，请参阅《AWS CloudFormation 用户指南》 CodePipeline中的使用持续交付。

- Amazon ECS

  在集群名称中，输入或选择现有 Amazon ECS 集群的名称。在服务名称中，输入或选择在集群上运行的服务的名称。您还可以创建集群和服务。在映像文件名中，键入描述服务的容器和映像的映像定义文件的名称。

  > **Note**
  >
  > Amazon ECS 部署操作需要一个 `imagedefinitions.json` 文件作为部署操作的输入。该文件的默认文件名为 imagedefinitions.json。如果您选择使用不同的文件名，则必须在创建管道部署阶段时提供。有关更多信息，请参阅 适用于 Amazon ECS 标准部署操作的 imagedefinitions.json 文件。

选择 下一步。

> **Note**
>
> 确保为 Amazon ECS 集群配置了两个或更多实例。Amazon ECS 集群必须包含至少两个实例，以便一个作为主实例进行维护，另一个用于容纳新部署。

有关使用管道部署基于容器的应用程序的教程，请参阅教程：使用进行持续部署。CodePipeline

- Amazon ECS（蓝/绿）

  输入 CodeDeploy 应用程序和部署组、Amazon ECS 任务定义和 AppSpec 文件信息，然后选择 “下一步”。

  > **Note**
  >
  > Amazon ECS（蓝/绿）操作需要 imageDetail.json 文件作为部署操作的输入构件。由于 Amazon ECR 源操作会创建此文件，所以包括 Amazon ECR 源操作的管道无需提供 `imageDetail.json` 文件。有关更多信息，请参阅 适用于 Amazon ECS 蓝绿部署的 imageDetail.json 文件。

  有关使用创建蓝绿色部署到 Amazon ECS 集群的管道的教程 CodeDeploy，请参阅教程：使用 Amazon ECR 源和 ECS 目标部署创建管道 CodeDeploy 。

- AWS Service Catalog

  如果您要使用控制台中的字段指定配置，请选择输入部署配置，如果您有单独的配置文件，请选择配置文件。输入产品和配置信息，然后选择下一步。

  有关使用管道将产品更改部署到 Service Catalog 的教程，请参阅教程：创建部署到 Service Catalog 的管道。

- Alexa Skills Kit

  在 Alexa Skill ID 中，为您的 Alexa 技能输入技能 ID。在客户端 ID 和客户端密钥中，输入使用 Login with Amazon（LWA）安全配置文件生成的凭证。在刷新令牌中，输入使用 ASK CLI 命令生成的刷新令牌，以检索刷新令牌。选择下一步。

有关使用管道部署 Alexa 技能并生成 LWA 凭据的教程，请参阅教程：创建部署 Amazon Alexa 技能的管道。

- Amazon S3

在存储桶中，您要使用的 S3 存储桶的名称。如果部署阶段的输入项目是 ZIP 文件，请选择在部署前提取文件。如果选择了在部署前提取文件，则您可以（可选）为要将 ZIP 文件解压缩到的部署路径输入值。如果未选择，则您需要在 S3 对象键中输入一个值。

> **Note**
>
> 大多数源和构建阶段输出构件都会被压缩。除了 Amazon S3 之外的所有管道源提供方都会先压缩源文件，然后再将它们作为输入构件提供给下一个操作。

（可选）在标准 ACL 中，输入要应用于部署到 Amazon S3 的对象的标准 ACL。

> **Note**
>
> 应用标准 ACL 将覆盖已应用于对象的任何现有 ACL。

（可选）在缓存控制中，为从存储桶下载对象的请求指定缓存控制参数。有关有效值的列表，请参阅 HTTP 操作的 `Cache-Control` 标头字段。要在缓存控件中输入多个值，请在每个值之间使用逗号。您可以在每个逗号后添加一个空格（可选），如本示例所示。

前述示例条目显示在 CLI 中，如下所示：

```
"CacheControl": "public, max-age=0, no-transform"
```

选择下一步。

有关使用 Amazon S3 部署操作提供方创建管道的教程，请参阅教程：创建以 Amazon S3 作为部署提供程序的管道。

步骤 5：审核管道

- 在步骤 5：审核页面上，查看您的管道配置，然后选择创建管道以创建管道，或者选择上一步返回并编辑您的选择。要退出向导而不创建管道，请选择 Cancel。

创建了管道后，就可以在控制台中进行查看。管道将在创建之后开始运行。有关更多信息，请参阅在中查看管道和详细信息 CodePipeline。有关对管道进行更改的更多信息，请参阅在中编辑管道 CodePipeline。

创建管道（CLI）

要使用创建管道，您需要创建一个 JSON 文件来定义管道结构，然后使用`--cli-input-json`参数运行create-pipeline命令。 AWS CLI

> **Important**
>
> 您不能 AWS CLI 使用创建包含合作伙伴操作的管道。您必须改用 CodePipeline 控制台。

有关管道结构的更多信息，请参阅《 CodePipeline API 参考》中的 an CodePipeline 管道结构参考 d create-pipelin e。

要创建 JSON 文件，请使用示例管道 JSON 文件，编辑该文件，然后在运行 create-pipeline 命令时调用该文件。

先决条件：

您需要在中为其创建的服务角色的 ARN。 CodePipeline 入门 CodePipeline运行命令时，您可以在管道 JSON 文件中使用 CodePipeline 服务角色 ARN。create-pipeline有关创建服务角色的更多信息，请参阅创建 CodePipeline 服务角色。与控制台不同，在中运行create-pipeline命令 AWS CLI 不能选择为你创建 CodePipeline 服务角色。服务角色必须已存在。

您需要用于存储管道构件的 S3 存储桶的名称。该存储桶必须与管道位于同一区域。运行 create-pipeline 命令时，可以在管道 JSON 文件中使用存储桶名称。与控制台不同，在中运行create-pipeline命令 AWS CLI 不会创建用于存储项目的 S3 存储桶。该存储桶必须已经存在。

> **Note**
>
> 您也可以使用 get-pipeline 命令获取该管道的 JSON 结构的副本，然后在纯文本编辑器中修改该结构。

创建 JSON 文件

1. 在终端（Linux、macOS 或 Unix）或命令提示符 (Windows) 处，在本地目录中创建新的文本文件。
2. （可选）您可以添加一个或多个管道级变量。您可以在 CodePipeline 操作配置中引用此值。您可以在创建管道时添加变量名称和值，也可以在控制台中启动管道时选择赋值。

   > **Note**
   >
   > 虽然添加管道级变量是可选的，但如果使用管道级变量指定了管道而没有提供变量值，管道执行将失败。

   管道级变量是在管道运行时解析的。所有变量都是不可变的，这意味着它们在赋值后无法更新。已解析值的管道级变量将显示在每个执行的历史记录中。

   您可以使用管道结构中的变量属性提供管道级变量。在以下示例中，变量 `Variable1` 的值为 `Value1`。

   ```
   "variables": [
        {
            "name": "Timeout",
            "defaultValue": "1000",
            "description": "description"
        }
    ]
   ```

   将此结构添加到您的管道 JSON 中，或者添加到以下步骤中的示例 JSON 中。有关变量的更多信息，包括命名空间信息，请参阅Variables。
3. 在纯文本编辑器中打开该文件并编辑值，以反映您要创建的结构。您必须至少更改管道的名称。您还应考虑是否要更改：

   - 用于存储此管道的构件的 S3 存储桶。

- 代码的源位置。
- 部署提供程序。
- 您希望如何部署代码。
- 管道的标签。

下面的两阶段示例管道结构突出显示了您应考虑为您的管道更改的值。您的管道可能包含两个以上的阶段：

```
{
    "pipeline": {
        "roleArn": "arn:aws:iam::80398EXAMPLE::role/AWS-CodePipeline-Service",
        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "inputArtifacts": [],
                        "name": "Source",
                        "actionTypeId": {
                            "category": "Source",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "S3"
                        },
                        "outputArtifacts": [
                            {
                                "name": "MyApp"
                            }
                        ],
                        "configuration": {
                            "S3Bucket": "awscodepipeline-demobucket-example-date",
                            "S3ObjectKey": "ExampleCodePipelineSampleBundle.zip",
                            "PollForSourceChanges": "false"
                        },
                        "runOrder": 1
                    }
                ]
            },
            {
                "name": "Staging",
                "actions": [
```

```
                {
                    "inputArtifacts": [
                        {
                            "name": "MyApp"
                        }
                    ],
                    "name": "Deploy-CodeDeploy-Application",
                    "actionTypeId": {
                        "category": "Deploy",
                        "owner": "AWS",
                        "version": "1",
                        "provider": "CodeDeploy"
                    },
                    "outputArtifacts": [],
                    "configuration": {
                        "ApplicationName": "CodePipelineDemoApplication",
                        "DeploymentGroupName": "CodePipelineDemoFleet"
                    },
                    "runOrder": 1
                }
            ]
        }
    ],
    "artifactStore": {
        "type": "S3",
        "location": "codepipeline-us-east-2-250656481468"
    },
    "name": "MyFirstPipeline",
    "version": 1,
    "variables": [
        {
            "name": "Timeout",
            "defaultValue": "1000",
            "description": "description"
        }
      ]
    },
    "triggers": [
        {
            "providerType": "CodeStarSourceConnection",
            "gitConfiguration": {
                "sourceActionName": "Source",
                "push": [
                    {
```

```
                            "tags": {
                                "includes": [
                                    "v1"
                                ],
                                "excludes": [
                                    "v2"
                                ]
                            }
                        }
                    ]
                }
            }
        ]
    "metadata": {
        "pipelineArn": "arn:aws:codepipeline:us-
east-2:80398EXAMPLE:MyFirstPipeline",
        "updated": 1501626591.112,
        "created": 1501626591.112
    },
    "tags": [{
      "key": "Project",
      "value": "ProjectA"
    }]
}
```

此示例通过在管道上包含 `Project` 标签键和 `ProjectA` 值来为管道添加标记。有关在中为资源添加标签的更多信息 CodePipeline，请参阅标记资源。

确保按如下所示设置 JSON 文件中的 `PollForSourceChanges` 参数：

```
                "PollForSourceChanges": "false",
```

CodePipeline 使用 Amazon Ev CloudWatch ents 来检测您的 CodeCommit 源存储库和分支或 S3 源存储桶中的更改。下一步包括手动为您的管道创建这些资源的说明。将此标记设置为 `false` 将禁用定期检查，当您使用建议的更改检测方法时不需要定期检查。

4. 要在您的管道所在区域之外的区域创建构建、测试或部署操作，您必须向管道结构添加以下内容。有关说明，请参阅在中添加跨区域操作 CodePipeline。

   - 将 `Region` 参数添加到您的操作的管道结构。
   - 使用`artifactStores`参数为您要执行操作的每个 AWS 区域指定一个对象存储桶。

5. 如果您对其结构感到满意，请使用类似 **pipeline.json** 的名称保存您的文件。

要创建管道

1. 运行 create-pipeline 命令，并使用 `--cli-input-json` 参数指定您之前创建的 JSON 文件。

   要创建*MySecondPipeline*使用名为 pipeline.json 的 JSON 文件命名的管道，该文件在 JSON name 中包含名称 *MySecondPipeline*“” 作为值，您的命令将如下所示：

   ```
   aws codepipeline create-pipeline --cli-input-json file://pipeline.json
   ```

   > **Important**
   >
   > 务必在文件名前包含 `file://`。此命令中需要该项。

   该命令会返回您创建的整个管道的结构。

2. 要查看管道，可以打开 CodePipeline 控制台并从管道列表中进行选择，也可以使用get-pipeline-state命令。有关更多信息，请参阅 在中查看管道和详细信息 CodePipeline。
3. 如果您使用 CLI 创建管道，您必须手动为您的管道创建建议的更改检测资源：
   - 对于包含 CodeCommit 存储库的管道，您必须手动创建 CloudWatch 事件规则，如中所述为 CodeCommit 源创建 EventBridge 规则 (CLI)。
   - 对于包含 Amazon S3 源的管道，您必须手动创建 CloudWatch 事件规则和 AWS CloudTrail 跟踪，如中所述亚马逊 S3 源代码操作 EventBridge 以及 AWS CloudTrail。

Amazon ECR 源操作和 EventBridge 资源

要在中添加 Amazon ECR 源操作 CodePipeline，您可以选择：

- 使用 CodePipeline 控制台创建管道向导 (创建管道（控制台）) 或编辑操作页面选择 Amazon ECR 提供商选项。控制台会创建一条 EventBridge 规则，当源发生变化时，该规则会启动您的管道。
- 使用 CLI 为 ECR 操作添加操作配置并创建其他资源，如下所示：
  - 使用 Amazon ECR 中的 ECR 示例操作配置来创建操作，如创建管道（CLI） 中所示。
  - 更改检测方法默认为通过轮询源来启动管道。应禁用定期检查并手动创建更改检测规则。使用以下方法之一：为 Amazon ECR 来源创建 EventBridge 规则（控制台）、为 Amazon ECR 来源 (CLI)

创建 EventBridge 规则或为 Amazon ECR 来源创建 EventBridge 规则（AWS CloudFormation 模板）。

主题

- 为 Amazon ECR 来源创建 EventBridge 规则（控制台）
- 为 Amazon ECR 来源 (CLI) 创建 EventBridge 规则
- 为 Amazon ECR 来源创建 EventBridge 规则（AWS CloudFormation 模板）

为 Amazon ECR 来源创建 EventBridge 规则（控制台）

创建用于 CodePipeline 操作的 EventBridge 规则（Amazon ECR 来源）

1. 打开亚马逊 EventBridge 控制台，网址为 https://console.aws.amazon.com/events/。
2. 在导航窗格中，选择事件。
3. 选择创建规则，然后在事件源下的服务名称中，选择 Elastic Container Registry (ECR)。
4. 在事件源中，选择事件模式。

   选择编辑，然后将以下示例事件模式粘贴到事件源窗口中，以获取映像标签为 `cli-testing` 的 `eb-test` 存储库：

```
{
    "detail-type": [
        "ECR Image Action"
    ],
    "source": [
        "aws.ecr"
    ],
    "detail": {
        "action-type": [
            "PUSH"
        ],
        "image-tag": [
            "latest"
        ],
        "repository-name": [
            "eb-test"
        ],
        "result": [
            "SUCCESS"
```

```
        ]
    }
}
```

> **Note**
>
> 要查看 Amazon ECR 事件支持的完整事件模式，请参阅 Amazon ECR 事件和/或 EventBridge亚马逊弹性容器注册表事件。

5. 选择保存。

   在事件模式预览窗格中，查看该规则。

6. 在目标中，选择CodePipeline。
7. 输入该规则将启动的管道的管道 ARN。

> **Note**
>
> 在运行 get-pipeline 命令后，您可以在元数据输出中找到管道 ARN。管道 ARN 是使用以下格式构造的：
> arn:aws:codepipeline:*region*:*account*:*pipeline-name*
> 示例管道 ARN：
> `arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline`

8. 创建或指定一个 IAM 服务角色，该角色授予调用与您的 EventBridge 规则关联的目标的 EventBridge 权限（在本例中，目标是 CodePipeline）。
   - 选择 “为此特定资源创建新角色” 以创建服务角色，该角色 EventBridge授予您启动管道执行的权限。
   - 选择 “使用现有角色” 输入一个服务角色，该角色 EventBridge 授予您启动管道执行的权限。
9. 检查规则设置以确保它符合您的要求。
10. 选择 Configure details（配置详细信息）。
11. 在配置规则详细信息页上，输入规则的名称和描述，然后选择状态以启用该规则。
12. 如果您对规则满意，请选择 Create rule。

为 Amazon ECR 来源 (CLI) 创建 EventBridge 规则

调用 put-rule 命令，在命令中指定：

- 唯一地标识创建的规则的名称。在您创建的与 AWS 账户 CodePipeline 关联的所有管道中，此名称必须是唯一的。
- 规则使用的源事件模式和详细信息字段。有关更多信息，请参阅 Amazon EventBridge 和事件模式。

创建以 Amazon ECR 作为事件源和目标 CodePipeline 的 EventBridge 规则

1. 添加用于调 EventBridge CodePipeline 用规则的权限。有关更多信息，请参阅使用适用于 Amazon EventBridge 的基于资源的政策。

   a. 使用以下示例创建允许 EventBridge 担任服务角色的信任策略。将信任策略命名为 `trustpolicyforEB.json`。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "events.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

   b. 使用以下命令创建 `Role-for-MyRule` 角色并附加信任策略。

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

   c. 为名为 `MyFirstPipeline` 的管道创建权限策略 JSON，如此示例中所示。将权限策略命名为 `permissionspolicyforEB.json`。

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

```
            "Resource": [
                "arn:aws:codepipeline:us-west-2:80398EXAMPLE:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 使用以下命令将 CodePipeline-Permissions-Policy-for-EB 权限策略附加到 Role-for-MyRule 角色。

我为何做出此更改？ 将此策略添加到角色会为创建权限 EventBridge。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. 调用 put-rule 命令并包含 --name、--event-pattern 和 --role-arn 参数。

我为何做出此更改？ 您必须创建具有相应规则（指定必须如何进行映像推送）和目标（指定由事件启动的管道）的事件。

以下示例命令创建一个名为 MyECRRepoRule 的规则。

```
aws events put-rule --name "MyECRRepoRule" --event-pattern "{\"detail-type\":[\"ECR Image Action\"],\"source\":[\"aws.ecr\"],\"detail\":{\"action-type\":[\"PUSH\"],\"image-tag\":[\"latest\"],\"repository-name\":[\"eb-test\"],\"result\":[\"SUCCESS\"]}}" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-MyRule"
```

> **Note**
>
> 要查看 Amazon ECR 事件支持的完整事件模式，请参阅 Amazon ECR 事件和/或 EventBridge亚马逊弹性容器注册表事件。

3. 要添加 CodePipeline 为目标，请调用put-targets命令并添加以下参数：

- --rule 参数与您使用 put-rule 创建的 rule_name 结合使用。
- --targets 参数与目标列表中该目标的列表 Id 以及目标管道的 ARN 结合使用。

以下示例命令为名为 MyECRRepoRule 的规则指定此内容，目标 Id 由数字 1 组成，这指示此内容位于规则的目标列表中，而这是目标 1。示例命令还指定管道示例 Arn 和规则示例 RoleArn。管道在存储库中发生更改时启动。

```
aws events put-targets --rule MyECRRepoRule --targets
 Id=1,Arn=arn:aws:codepipeline:us-
west-2:80398EXAMPLE:TestPipeline,RoleArn=arn:aws:iam::80398EXAMPLE:role/Role-for-
MyRule
```

为 Amazon ECR 来源创建 EventBridge 规则（AWS CloudFormation 模板）

AWS CloudFormation 要使用创建规则，请使用此处所示的模板片段。

更新您的管道 AWS CloudFormation 模板并创建 EventBridge 规则

1. 在模板下的模板中`Resources`，使用`AWS::IAM::Role` AWS CloudFormation 资源配置允许您的事件启动管道的 IAM 角色。此条目将创建一个使用两个策略的角色：

   - 第一个策略允许代入角色。
   - 第二个策略提供启动管道所需的权限。

   我为何做出此更改？ 您必须创建一个可以代入的 EventBridge 角色才能在我们的管道中开始执行。

   YAML

   ```
   EventRole:
     Type: AWS::IAM::Role
     Properties:
       AssumeRolePolicyDocument:
         Version: 2012-10-17
         Statement:
           -
             Effect: Allow
             Principal:
               Service:
                 - events.amazonaws.com
             Action: sts:AssumeRole
       Path: /
       Policies:
         -
           PolicyName: eb-pipeline-execution
           PolicyDocument:
   ```

```
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Sub arn:aws:codepipeline:${AWS::Region}:
${AWS::AccountId}:${AppPipeline}
```

JSON

```
{
    "EventRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": [
                                "events.amazonaws.com"
                            ]
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "Path": "/",
            "Policies": [
                {
                    "PolicyName": "eb-pipeline-execution",
                    "PolicyDocument": {
                        "Version": "2012-10-17",
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Action": "codepipeline:StartPipelineExecution",
                                "Resource": {
                                    "Fn::Sub": "arn:aws:codepipeline:
${AWS::Region}:${AWS::AccountId}:${AppPipeline}"
                                }
                            }
```

```
                        ]
                    }
                }
            ]
        }
    }
}
...
```

2. 在模板下Resources，使用AWS::Events::Rule AWS CloudFormation 资源为 Amazon ECR 来源添加 EventBridge 规则。此事件模式会创建一个事件，以监控向存储库提交的操作。当 EventBridge 检测到存储库状态更改时，将在目标管道StartPipelineExecution上调用该规则。

   我为何做出此更改？ 您必须创建具有相应规则（指定必须如何进行映像推送）和目标（指定由事件启动的管道）的事件。

   此代码段使用标签为 latest、名为 eb-test 的映像。

   YAML

```
EventRule:
  Type: 'AWS::Events::Rule'
  Properties:
    EventPattern:
      detail:
        action-type: [PUSH]
        image-tag: [latest]
        repository-name: [eb-test]
        result: [SUCCESS]
      detail-type: [ECR Image Action]
      source: [aws.ecr]
    Targets:
      - Arn: !Sub arn:aws:codepipeline:${AWS::Region}:${AWS::AccountId}:
${AppPipeline}
        RoleArn: !GetAtt
          - EventRole
          - Arn
        Id: codepipeline-AppPipeline
```

JSON

```
{
    "EventRule": {
        "Type": "AWS::Events::Rule",
        "Properties": {
            "EventPattern": {
                "detail": {
                    "action-type": [
                        "PUSH"
                    ],
                    "image-tag": [
                        "latest"
                    ],
                    "repository-name": [
                        "eb-test"
                    ],
                    "result": [
                        "SUCCESS"
                    ]
                },
                "detail-type": [
                    "ECR Image Action"
                ],
                "source": [
                    "aws.ecr"
                ]
            },
            "Targets": [
                {
                    "Arn": {
                        "Fn::Sub": "arn:aws:codepipeline:${AWS::Region}:
${AWS::AccountId}:${AppPipeline}"
                    },
                    "RoleArn": {
                        "Fn::GetAtt": [
                            "EventRole",
                            "Arn"
                        ]
                    },
                    "Id": "codepipeline-AppPipeline"
                }
            ]
```

```
        }
    }
},
```

> **Note**
>
> 要查看 Amazon ECR 事件支持的完整事件模式，请参阅 Amazon ECR 事件和/或 EventBridge亚马逊弹性容器注册表事件。

3. 将更新后的模板保存到本地计算机，然后打开 AWS CloudFormation 控制台。
4. 选择堆栈，然后选择为当前堆栈创建更改集。
5. 上传模板，然后查看 AWS CloudFormation中列出的更改。这些是要对堆栈进行的更改。您应在列表中看到新资源。
6. 选择执行。

亚马逊 S3 源代码操作 EventBridge 以及 AWS CloudTrail

要在中添加 Amazon S3 源操作 CodePipeline，您可以选择以下任一选项：

- 使用 CodePipeline 控制台的 “创建管道” 向导 (创建管道（控制台）) 或 “编辑” 操作页面选择 S3 提供程序选项。控制台会创建一条 EventBridge 规则和一条 CloudTrail 跟踪，当源发生变化时，它会启动您的管道。
- 使用 AWS CLI 为操作添加操作配置并创建其他资源，如下所示：S3
  - 使用 Amazon S3 源操作 中的 S3 示例操作配置来创建操作，如创建管道（CLI） 中所示。
  - 更改检测方法默认为通过轮询源来启动管道。应禁用定期检查并手动创建更改检测规则和跟踪。使用以下方法之一：为 Amazon S3 来源创建 EventBridge 规则（控制台）、为 Amazon S3 来源 (CLI) 创建 EventBridge 规则或为 Amazon S3 来源创建 EventBridge 规则（AWS CloudFormation 模板）。

AWS CloudTrail 是一项用于记录和筛选您的 Amazon S3 源存储桶上的事件的服务。跟踪会将筛选后的源更改发送到 EventBridge 规则。该 EventBridge 规则会检测源代码更改，然后启动您的管道。

要求：

- 如果您没有创建跟踪，请使用现有 AWS CloudTrail 跟踪记录您的 Amazon S3 源存储桶中的事件，并将筛选的事件发送到 EventBridge规则。

- 创建或使用 AWS CloudTrail 可存储其日志文件的现有 S3 存储桶。 AWS CloudTrail 必须具有将日志文件传输到 Amazon S3 存储桶所需的权限。此存储桶无法配置为申请方付款存储桶。当您在控制台中创建或更新跟踪时创建 Amazon S3 存储桶时，会将所需的权限 AWS CloudTrail 附加到存储桶。有关更多信息，请参阅 Amazon S3 存储桶政策 CloudTrail。

为 Amazon S3 来源创建 EventBridge 规则（控制台）

在中设置规则之前 EventBridge，必须创建 AWS CloudTrail 跟踪。有关更多信息，请参阅在控制台中创建跟踪。

> **Important**
>
> 如果您使用控制台创建或编辑管道，则会为您创建 EventBridge 规则和 AWS CloudTrail 跟踪。

创建跟踪

1. 打开控制 AWS CloudTrail 台。
2. 在导航窗格中，选择 Trails（跟踪记录）。
3. 选择创建跟踪。对于跟踪名称，输入跟踪的名称。
4. 在存储位置下，创建或指定要用于存储日志文件的存储桶。默认情况下，Amazon S3 存储桶和对象都是私有的。只有资源所有者（创建存储桶的 AWS 账户）才能访问存储桶及其对象。存储桶必须具有允许访问存储桶中对象的 AWS CloudTrail 权限的资源策略。
5. 在跟踪日志桶和文件夹下，指定 Amazon S3 桶和对象前缀（文件夹名称）以记录文件夹中所有对象的数据事件。对于每个跟踪，您可以添加最多 250 个 Amazon S3 对象。填写所需的加密密钥信息，然后选择下一步。
6. 对于事件类型，选择管理事件。
7. 对于管理事件，选择编辑。跟踪将记录指定桶和前缀上的 Amazon S3 对象级 API 活动（例如，`GetObject` 和 `PutObject`）。
8. 选择写入。
9. 如果您对跟踪满意，请选择创建跟踪。

使用 Amazon S3 源创建针对您的管道的 EventBridge 规则

1. 打开亚马逊 EventBridge 控制台，网址为 https://console.aws.amazon.com/events/。

2. 在导航窗格中，选择规则。保留选中的默认总线或选择一个事件总线。选择创建规则。
3. 在名称中，输入规则的名称。
4. 对于规则类型，选择具有事件模式的规则。选择下一步。
5. 在 “事件来源” 下，选择AWS 事件或 EventBridge 合作伙伴事件。
6. 在示例事件类型下，选择 AWS 事件。
7. 在示例事件中，键入 S3 作为要筛选的关键字。选择 AWS API 调用方式 CloudTrail。
8. 在创建方法下，选择自定义模式（JSON 编辑器）。

   粘贴下面提供的事件模式。确保添加桶名称和 S3 对象键（或键名称），它们作为 `requestParameters` 唯一标识桶中的对象。在本例中，针对名为 `my-bucket` 的桶和对象键 `my-files.zip` 创建了一条规则。在使用编辑窗口指定资源时，您的规则将更新为使用自定义事件模式。

   下面是一个可供复制并粘贴的示例事件模式：

```
{
    "source": [
        "aws.s3"
    ],
    "detail-type": [
        "AWS API Call via CloudTrail"
    ],
    "detail": {
        "eventSource": [
            "s3.amazonaws.com"
        ],
        "eventName": [
            "CopyObject",
            "CompleteMultipartUpload",
            "PutObject"
        ],
        "requestParameters": {
            "bucketName": [
                "my-bucket"
            ],
            "key": [
                "my-files.zip"
            ]
        }
    }
```

```
}
```

9. 选择下一步。
10. 在目标类型中，选择 AWS 服务。
11. 在选择目标中，选择CodePipeline。在管道 ARN 中，输入该规则启动的管道的管道 ARN。

> **Note**
>
> 要获取管道 ARN，请运行 get-pipeline 命令。管道 ARN 将显示在输出中。它是使用以下格式构造的：
> arn:aws:codepipeline:*region*:*account*:*pipeline-name*
> 示例管道 ARN：
> arn: aws: codepipeline: us-east-2:80398 示例：MyFirstPipeline

12. 要创建或指定一个 IAM 服务角色来授予调用与您的 EventBridge 规则关联的目标的 EventBridge 权限（在本例中，目标是 CodePipeline），请执行以下操作：
    - 选择 “为此特定资源创建新角色” 以创建服务角色，该角色 EventBridge授予您启动管道执行的权限。
    - 选择 “使用现有角色” 输入一个服务角色，该角色 EventBridge 授予您启动管道执行的权限。
13. 选择下一步。
14. 在标签页面上，选择下一步。
15. 在查看和创建页面上，检查规则配置。如果您对规则满意，请选择 Create rule。

为 Amazon S3 来源 (CLI) 创建 EventBridge 规则

创建 AWS CloudTrail 跟踪并启用日志记录

要使用创建跟踪，请调用create-trail命令，指定： AWS CLI

- 跟踪名称。
- 已将 AWS CloudTrail的存储桶策略应用于的存储桶。

有关更多信息，请参阅使用AWS 命令行界面创建跟踪。

1. 调用 create-trail 命令并包含 `--name` 和 `--s3-bucket-name` 参数。

我为何做出此更改？ 这将为您的 S3 源存储桶创建所需的 CloudTrail跟踪。

以下示例命令使用 `--name` 和 `--s3-bucket-name` 创建一个名为 `my-trail` 的跟踪和一个名为 `myBucket` 的存储桶。

```
aws cloudtrail create-trail --name my-trail --s3-bucket-name myBucket
```

2. 调用 start-logging 命令并包含 `--name` 参数。

   我为何做出此更改？ 此命令启动您的源存储桶的 CloudTrail 日志记录并将事件发送到 EventBridge。

   例如：

   以下命令使用 `--name` 启动针对名为 `my-trail` 的跟踪的日志记录。

   ```
   aws cloudtrail start-logging --name my-trail
   ```

3. 调用 put-event-selectors 命令并包含 `--trail-name` 和 `--event-selectors` 参数。使用事件选择器指定您希望您的跟踪记录源存储桶的数据事件并将事件发送到 EventBridge 规则。

   我为何做出此更改？ 此命令筛选事件。

   例如：

   以下示例命令使用 `--trail-name` 和 `--event-selectors` 指定源存储桶的数据事件以及名为 `myBucket/myFolder` 的前缀。

   ```
   aws cloudtrail put-event-selectors --trail-name my-trail --event-selectors
    '[{ "ReadWriteType": "WriteOnly", "IncludeManagementEvents":false,
    "DataResources": [{ "Type": "AWS::S3::Object", "Values": ["arn:aws:s3:::myBucket/
   myFolder/file.zip"] }] }]'
   ```

创建以 Amazon S3 作为事件源和 CodePipeline 目标的 EventBridge 规则并应用权限策略

1. 授 EventBridge 予使用调 CodePipeline 用规则的权限。有关更多信息，请参阅[使用适用于 Amazon EventBridge 的基于资源的政策。]

   a. 使用以下示例创建允许 EventBridge 担任服务角色的信任策略。将它命名为 `trustpolicyforEB.json`。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "events.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

b. 使用以下命令创建 Role-for-MyRule 角色并附加信任策略。

我为何做出此更改？ 将此信任策略添加到角色会为创建权限 EventBridge。

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. 为名为 MyFirstPipeline 的管道创建权限策略 JSON，如此处所示。将权限策略命名为 permissionspolicyforEB.json。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-west-2:80398EXAMPLE:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 使用以下命令将新的 CodePipeline-Permissions-Policy-for-EB 权限策略附加到您创建的 Role-for-MyRule 角色。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. 调用 put-rule 命令并包含 `--name`、`--event-pattern` 和 `--role-arn` 参数。

   以下示例命令创建名为 `MyS3SourceRule` 的规则。

```
aws events put-rule --name "MyS3SourceRule" --event-pattern "{\"source\":
[\"aws.s3\"],\"detail-type\":[\"AWS API Call via CloudTrail\"],\"detail\":
{\"eventSource\":[\"s3.amazonaws.com\"],\"eventName\":[\"CopyObject\",\"PutObject
\",\"CompleteMultipartUpload\"],\"requestParameters\":{\"bucketName\":[\"my-bucket
\"],\"key\":[\"my-key\"]}}}
 --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-MyRule"
```

3. 要添加 CodePipeline 为目标，请调用put-targets命令并添加`--rule`和`--targets`参数。

   以下示例命令为名为 `MyS3SourceRule` 的规则指定此内容，目标 `Id` 由数字 1 组成，这指示此内容位于规则的目标列表中，而这是目标 1。此命令还为管道指定一个示例 ARN。管道在存储库中发生更改时启动。

```
aws events put-targets --rule MyS3SourceRule --targets
 Id=1,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
```

编辑管道的 PollForSourceChanges参数

> **Important**
>
> 使用此方法创建管道时，如果 `PollForSourceChanges` 参数未明确设置为 false，则默认为 true。添加基于事件的更改检测时，必须将参数添加到输出并将其设置为 false 以禁用轮询。否则，您的管道将针对单个源更改启动两次。有关更多信息，请参阅 PollForSourceChanges 参数的默认设置。

1. 运行 get-pipeline 命令以将管道结构复制到 JSON 文件中。例如，对于名为 `MyFirstPipeline` 的管道，运行以下命令：

```
aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
```

该命令不会返回任何结果，但您创建的文件将出现在您运行命令所在的目录中。

2. 在任何纯文本编辑器中打开 JSON 文件并通过将名为 `storage-bucket` 的存储桶的 `PollForSourceChanges` 参数更改为 `false` 来编辑源阶段，如此示例中所示。

   我为何做出此更改？ 将此参数设置为 `false` 将关闭定期检查，因此您只能使用基于事件的更改检测。

   ```
   "configuration": {
       "S3Bucket": "storage-bucket",
       "PollForSourceChanges": "false",
       "S3ObjectKey": "index.zip"
   },
   ```

3. 如果您要使用通过 get-pipeline 命令检索到的管道结构，则必须删除 JSON 文件中的 `metadata` 行。否则，update-pipeline 命令无法使用它。删除 `"metadata": { }` 行以及 `"created"`、`"pipelineARN"` 和 `"updated"` 字段。

   例如，从结构中删除以下各行：

   ```
   "metadata": {
       "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
       "created": "date",
       "updated": "date"
   },
   ```

   保存该文件。

4. 要应用更改，请运行 update-pipeline 命令，指定管道 JSON 文件：

   > **Important**
   >
   > 务必在文件名前包含 `file://`。此命令中需要该项。

   ```
   aws codepipeline update-pipeline --cli-input-json file://pipeline.json
   ```

   该命令会返回编辑后的管道的整个结构。

> **Note**
>
> update-pipeline 命令会停止管道。如果在运行 update-pipeline 命令时正在通过管道运行修订，则该运行会被停止。您必须手动启动管道，通过升级后的管道运行此修订。使用 start-pipeline-execution 命令手动启动您的管道。

为 Amazon S3 来源创建 EventBridge 规则（AWS CloudFormation 模板）

AWS CloudFormation 要使用创建规则，请更新您的模板，如下所示。

创建以 Amazon S3 作为事件源和 CodePipeline 目标的 EventBridge 规则并应用权限策略

1. 在模板下的模板中`Resources`，使用`AWS::IAM::Role` AWS CloudFormation 资源配置允许您的事件启动管道的 IAM 角色。此条目将创建一个使用两个策略的角色：

   - 第一个策略允许代入角色。
   - 第二个策略提供启动管道所需的权限。

   我为何做出此更改？ 添加`AWS::IAM::Role`资源可以 AWS CloudFormation 为创建权限 EventBridge。此资源已添加到您的 AWS CloudFormation 堆栈中。

   YAML

```
EventRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: 2012-10-17
      Statement:
        -
          Effect: Allow
          Principal:
            Service:
              - events.amazonaws.com
          Action: sts:AssumeRole
    Path: /
    Policies:
      -
        PolicyName: eb-pipeline-execution
```

```
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]


...
```

JSON

```
 "EventRole": {
   "Type": "AWS::IAM::Role",
   "Properties": {
     "AssumeRolePolicyDocument": {
       "Version": "2012-10-17",
       "Statement": [
         {
           "Effect": "Allow",
           "Principal": {
             "Service": [
               "events.amazonaws.com"
             ]
           },
           "Action": "sts:AssumeRole"
         }
       ]
     },
     "Path": "/",
     "Policies": [
       {
         "PolicyName": "eb-pipeline-execution",
         "PolicyDocument": {
           "Version": "2012-10-17",
           "Statement": [
             {
               "Effect": "Allow",
               "Action": "codepipeline:StartPipelineExecution",
               "Resource": {
                 "Fn::Join": [
```

```
                    "",
                    [
                      "arn:aws:codepipeline:",
                      {
                        "Ref": "AWS::Region"
                      },
                      ":",
                      {
                        "Ref": "AWS::AccountId"
                      },
                      ":",
                      {
                        "Ref": "AppPipeline"
                      }
                    ]
                  ]

...
```

2. 使用AWS::Events::Rule AWS CloudFormation 资源添加 EventBridge 规则。此事件模式会创建一个事件，以监控 Amazon S3 源桶上的 CopyObject、PutObject 和 CompleteMultipartUpload。此外，还包括您管道的目标。当发生 CopyObject、PutObject 或 CompleteMultipartUpload 时，此规则将对目标管道调用 StartPipelineExecution。

   我为何做出此更改？ 添加AWS::Events::Rule资源 AWS CloudFormation 即可创建事件。此资源已添加到您的 AWS CloudFormation 堆栈中。

   YAML

```
EventRule:
  Type: AWS::Events::Rule
  Properties:
    EventPattern:
      source:
        - aws.s3
      detail-type:
        - 'AWS API Call via CloudTrail'
      detail:
        eventSource:
          - s3.amazonaws.com
        eventName:
          - CopyObject
```

```
            - PutObject
            - CompleteMultipartUpload
          requestParameters:
            bucketName:
              - !Ref SourceBucket
            key:
              - !Ref SourceObjectKey
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline


...
```

JSON

```
 "EventRule": {
   "Type": "AWS::Events::Rule",
   "Properties": {
     "EventPattern": {
       "source": [
         "aws.s3"
       ],
       "detail-type": [
         "AWS API Call via CloudTrail"
       ],
       "detail": {
         "eventSource": [
           "s3.amazonaws.com"
         ],
         "eventName": [
           "CopyObject",
           "PutObject",
           "CompleteMultipartUpload"
         ],
         "requestParameters": {
           "bucketName": [
             {
```

```
                "Ref": "SourceBucket"
              }
            ],
            "key": [
              {
                "Ref": "SourceObjectKey"
              }
            ]
          }
        }
      },
      "Targets": [
        {
          "Arn": {
            "Fn::Join": [
              "",
              [
                "arn:aws:codepipeline:",
                {
                  "Ref": "AWS::Region"
                },
                ":",
                {
                  "Ref": "AWS::AccountId"
                },
                ":",
                {
                  "Ref": "AppPipeline"
                }
              ]
            ]
          },
          "RoleArn": {
            "Fn::GetAtt": [
              "EventRole",
              "Arn"
            ]
          },
          "Id": "codepipeline-AppPipeline"
        }
      ]
    }
  }
},
```

```
...
```

3. 将此代码段添加到第一个模板，以允许跨堆栈功能：

   YAML

   ```
   Outputs:
     SourceBucketARN:
       Description: "S3 bucket ARN that Cloudtrail will use"
       Value: !GetAtt SourceBucket.Arn
       Export:
         Name: SourceBucketARN
   ```

   JSON

   ```
     "Outputs" : {
       "SourceBucketARN" : {
         "Description" : "S3 bucket ARN that Cloudtrail will use",
         "Value" : { "Fn::GetAtt": ["SourceBucket", "Arn"] },
         "Export" : {
           "Name" : "SourceBucketARN"
         }
       }

   ...
   ```

4. 将更新后的模板保存到本地计算机上，然后打开 AWS CloudFormation 控制台。
5. 选择堆栈，然后选择为当前堆栈创建更改集。
6. 上传更新的模板，然后查看 AWS CloudFormation中所列的更改。以下是将对堆栈进行的更改。您应在列表中看到新资源。
7. 选择执行。

编辑管道的 PollForSourceChanges参数

> **Important**
>
> 使用此方法创建管道时，如果 `PollForSourceChanges` 参数未明确设置为 false，则默认为 true。添加基于事件的更改检测时，必须将参数添加到输出并将其设置为 false 以禁用轮询。否

则，您的管道将针对单个源更改启动两次。有关更多信息，请参阅 PollForSourceChanges 参数的默认设置。

- 在模板中，将 PollForSourceChanges 更改为 false。如果您未在管道定义中包含 PollForSourceChanges，请添加它并将它设置为 false。

  我为何做出此更改？ 将 PollForSourceChanges 更改为 false 将关闭定期检查，因此您只能使用基于事件的更改检测。

  YAML

```
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
              - Name: SourceOutput
            Configuration:
              S3Bucket: !Ref SourceBucket
              S3ObjectKey: !Ref SourceObjectKey
              PollForSourceChanges: false
            RunOrder: 1
```

  JSON

```
{
   "Name": "SourceAction",
   "ActionTypeId": {
     "Category": "Source",
     "Owner": "AWS",
     "Version": 1,
     "Provider": "S3"
   },
   "OutputArtifacts": [
     {
       "Name": "SourceOutput"
```

```
      }
    ],
    "Configuration": {
      "S3Bucket": {
        "Ref": "SourceBucket"
      },
      "S3ObjectKey": {
        "Ref": "SourceObjectKey"
      },
      "PollForSourceChanges": false
    },
    "RunOrder": 1
  }
```

为您的 Amazon S3 管道的 CloudTrail 资源创建第二个模板

- 在单独的模板中Resources，使用AWS::S3::BucketAWS::S3::BucketPolicy、和AWS::CloudTrail::Trail AWS CloudFormation 资源为其提供简单的存储桶定义和跟踪 CloudTrail。

  我为何做出此更改？ 鉴于当前每个账户只能有五条 CloudTrail 跟踪，因此必须单独创建和管理跟踪。（参见中的限制 AWS CloudTrail。）但是，您可以在单个跟踪上包括许多 Amazon S3 桶，因此您可以创建跟踪一次，然后根据需要为其他管道添加 Amazon S3 桶。将以下内容粘贴到第二个示例模板文件中。

  YAML

```
###############################################################################
# Prerequisites:
#   - S3 SourceBucket and SourceObjectKey must exist
###############################################################################

Parameters:
  SourceObjectKey:
    Description: 'S3 source artifact'
    Type: String
    Default: SampleApp_Linux.zip

Resources:
  AWSCloudTrailBucketPolicy:
```

```
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref AWSCloudTrailBucket
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Sid: AWSCloudTrailAclCheck
            Effect: Allow
            Principal:
              Service:
                - cloudtrail.amazonaws.com
            Action: s3:GetBucketAcl
            Resource: !GetAtt AWSCloudTrailBucket.Arn
          -
            Sid: AWSCloudTrailWrite
            Effect: Allow
            Principal:
              Service:
                - cloudtrail.amazonaws.com
            Action: s3:PutObject
            Resource: !Join [ '', [ !GetAtt AWSCloudTrailBucket.Arn, '/
AWSLogs/', !Ref 'AWS::AccountId', '/*' ] ]
            Condition:
              StringEquals:
                s3:x-amz-acl: bucket-owner-full-control
  AWSCloudTrailBucket:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
  AwsCloudTrail:
    DependsOn:
      - AWSCloudTrailBucketPolicy
    Type: AWS::CloudTrail::Trail
    Properties:
      S3BucketName: !Ref AWSCloudTrailBucket
      EventSelectors:
        -
          DataResources:
            -
              Type: AWS::S3::Object
              Values:
                - !Join [ '', [ !ImportValue SourceBucketARN, '/', !Ref
 SourceObjectKey ] ]
          ReadWriteType: WriteOnly
```

```
          IncludeManagementEvents: false
      IncludeGlobalServiceEvents: true
      IsLogging: true
      IsMultiRegionTrail: true


...
```

JSON

```
{
  "Parameters": {
    "SourceObjectKey": {
      "Description": "S3 source artifact",
      "Type": "String",
      "Default": "SampleApp_Linux.zip"
    }
  },
  "Resources": {
    "AWSCloudTrailBucket": {
      "Type": "AWS::S3::Bucket",
        "DeletionPolicy": "Retain"
    },
    "AWSCloudTrailBucketPolicy": {
      "Type": "AWS::S3::BucketPolicy",
      "Properties": {
        "Bucket": {
          "Ref": "AWSCloudTrailBucket"
        },
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "AWSCloudTrailAclCheck",
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  "cloudtrail.amazonaws.com"
                ]
              },
              "Action": "s3:GetBucketAcl",
              "Resource": {
                "Fn::GetAtt": [
```

```
                "AWSCloudTrailBucket",
                "Arn"
              ]
            }
          },
          {
            "Sid": "AWSCloudTrailWrite",
            "Effect": "Allow",
            "Principal": {
              "Service": [
                "cloudtrail.amazonaws.com"
              ]
            },
            "Action": "s3:PutObject",
            "Resource": {
              "Fn::Join": [
                "",
                [
                  {
                    "Fn::GetAtt": [
                      "AWSCloudTrailBucket",
                      "Arn"
                    ]
                  },
                  "/AWSLogs/",
                  {
                    "Ref": "AWS::AccountId"
                  },
                  "/*"
                ]
              ]
            },
            "Condition": {
              "StringEquals": {
                "s3:x-amz-acl": "bucket-owner-full-control"
              }
            }
          }
        ]
      }
    }
  },
  "AwsCloudTrail": {
    "DependsOn": [
```

```
        "AWSCloudTrailBucketPolicy"
      ],
      "Type": "AWS::CloudTrail::Trail",
      "Properties": {
        "S3BucketName": {
          "Ref": "AWSCloudTrailBucket"
        },
        "EventSelectors": [
          {
            "DataResources": [
              {
                "Type": "AWS::S3::Object",
                "Values": [
                  {
                    "Fn::Join": [
                      "",
                      [
                        {
                          "Fn::ImportValue": "SourceBucketARN"
                        },
                        "/",
                        {
                          "Ref": "SourceObjectKey"
                        }
                      ]
                    ]
                  }
                ]
              }
            ],
            "ReadWriteType": "WriteOnly",
            "IncludeManagementEvents": false
          }
        ],
        "IncludeGlobalServiceEvents": true,
        "IsLogging": true,
        "IsMultiRegionTrail": true
      }
    }
  }
}

...
```

Bitbucket Cloud 连接

Connections 允许您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。要将您的第三方存储库关联为管道的源，您应使用连接。

> **Note**
>
> 此功能不适用于亚太地区（香港）、亚太地区（海得拉巴）、亚太地区（雅加达）、亚太地区（墨尔本）、亚太地区（墨尔本）、亚太地区（大阪）、非洲（开普敦）、中东（巴林）、中东（阿联酋）、欧洲（西班牙）、欧洲（苏黎世）、以色列（特拉维夫）或 AWS GovCloud（美国西部）地区。要参考其他可用操作，请参阅 产品和服务与 CodePipeline。有关在欧洲地区（米兰）区域使用此操作的注意事项，请参阅CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、 GitLab .com 和 GitLab 自我管理操作中的说明。

要在中添加 Bitbucket Cloud 源操作 CodePipeline，您可以选择以下任一选项：

- 使用 CodePipeline 控制台的 “创建管道” 向导或 “编辑” 操作页面选择 Bitbucket 提供程序选项。参阅创建到 Bitbucket Cloud 的连接（控制台）以添加操作。控制台可帮助您创建连接资源。

  > **Note**
  >
  > 您可以创建到 Bitbucket Cloud 存储库的连接。不支持已安装的 Bitbucket 提供程序类型（如 Bitbucket 服务器）。

- 使用 CLI，添加提供方为 `Bitbucket` 的 `CreateSourceConnection` 操作的操作配置，如下所示：
  - 要创建连接资源，请参阅 创建到 Bitbucket Cloud 的连接 (CLI)，以便使用 CLI 创建连接资源。
  - 使用CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、 GitLab .com 和 GitLab 自我管理操作中的 `CreateSourceConnection` 示例操作配置来添加操作，如创建管道（CLI）中所示。

> **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。

开始前的准备工作：

- 您必须已创建第三方存储库提供方（如 Bitbucket Cloud）的账户。
- 您必须事先创建第三方代码存储库，如 Bitbucket Cloud 存储库。

> **Note**
>
> Bitbucket Cloud 连接只能访问用于创建连接的 Bitbucket Cloud 账户所拥有的存储库。如果要将应用程序安装在 Bitbucket Cloud 工作区中，则需要管理工作区权限。否则，安装应用程序的选项将不会显示。

主题

创建到 Bitbucket Cloud 的连接（控制台）

使用这些步骤使用 CodePipeline 控制台为您的 Bitbucket 存储库添加连接操作。

> **Note**
>
> 您可以创建到 Bitbucket Cloud 存储库的连接。不支持已安装的 Bitbucket 提供程序类型（如 Bitbucket 服务器）。

步骤 1：创建或编辑您的管道

创建或编辑您的管道

1. 登录 CodePipeline 控制台。
2. 选择以下选项之一。
   - 选择创建管道。按照创建管道 中的步骤操作，完成第一个屏幕，然后选择下一步。在源页面的源提供程序下，选择 Bitbucket。
   - 选择编辑现有管道。选择编辑，然后选择编辑阶段。选择添加或编辑您的源操作。在编辑操作页面的操作名称下，输入您的操作的名称。在操作提供程序中，选择 Bitbucket。

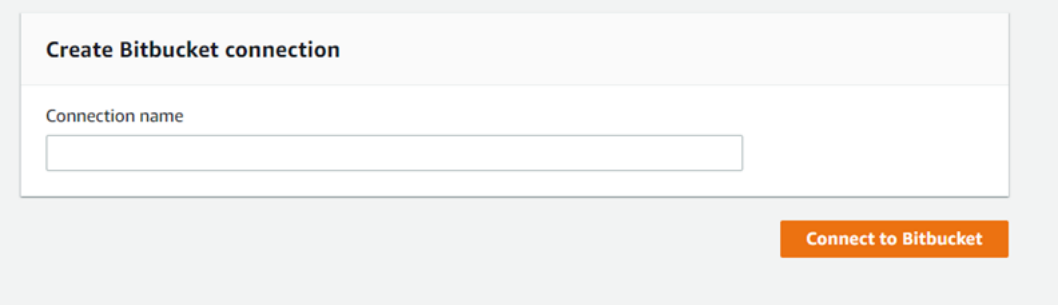
3. 请执行以下操作之一：

   - 在连接下，如果您尚未创建到提供程序的连接，请选择连接到 Bitbucket。继续执行步骤 2：创建到 Bitbucket 的连接。
   - 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 3：保存连接的源操作。

步骤 2：创建到 Bitbucket Cloud 的连接

创建到 Bitbucket Cloud 的连接

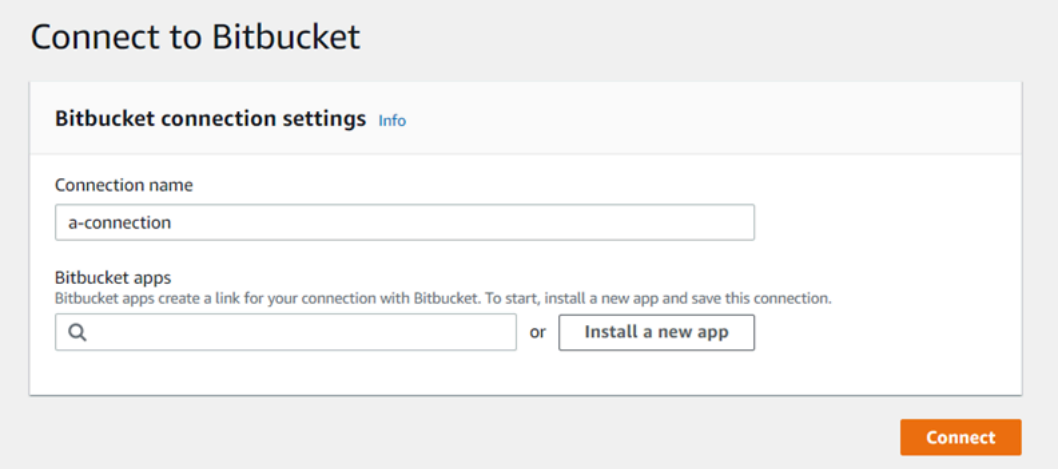
1. 在连接到 Bitbucket 设置页面上，输入您的连接名称，然后选择连接到 Bitbucket。



   此时将出现 Bitbucket 应用程序字段。

2. 在 Bitbucket 应用程序下，选择一个应用程序安装，或者选择安装新应用程序来创建一个应用程序安装。

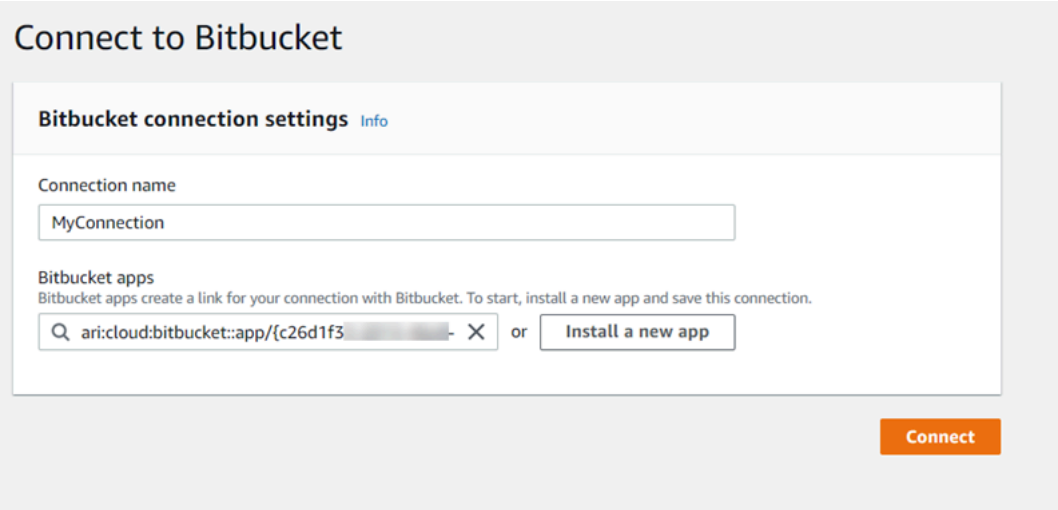
> **Note**
>
> 您只需为每个 Bitbucket 工作区或账户安装一次该应用。如果已经安装 Bitbucket 应用，请选择它，然后移至步骤 4。



3. 如果显示 Bitbucket Cloud 的登录页面，请使用您的凭证登录，然后选择继续。
4. 在应用程序安装页面上，一条消息显示该 AWS CodeStar 应用程序正在尝试连接到您的 Bitbucket 帐户。

   如果您使用的是 Bitbucket 工作区，请更改工作区的 Authorize for（授权）选项。只有您拥有管理员访问权限的工作区才会显示。

   选择授予访问权限。
5. 在 Bitbucket 应用程序中，将显示新安装的连接 ID。选择连接。创建的连接将显示在连接列表中。



步骤 3：保存您的 Bitbucket Cloud 源操作

使用向导或编辑操作页面上的这些步骤，将源操作与连接信息一起保存。

完成并保存您的源操作和连接

1. 在存储库名称中，选择第三方存储库的名称。
2. 如果您的操作是操作，则可以在 Pip elin e 触发器下添加触发器。 CodeConnections 要配置管道触发器配置并选择使用触发器进行筛选，请在中查看更多详细信息筛选代码推送或拉取请求的触发器。
3. 在输出构件格式中，您必须为构件选择格式。

   - 要使用默认方法存储 Bitbucket Cloud 操作的输出项目，请选择CodePipeline 默认。操作会访问 Bitbucket Cloud 存储库中的文件，并将构件以 ZIP 文件格式存储在管道构件存储中。
   - 要存储包含存储库的 URL 引用的 JSON 文件，以便下游操作可以直接执行 Git 命令，请选择完全克隆。此选项只能由 CodeBuild 下游操作使用。

     如果选择此选项，则需要更新 CodeBuild 项目服务角色的权限，如所示添加连接 Bitbucket、GitHub、En GitHub terprise Server 或 GitLab .com 的 CodeBuild GitClone 权限。
4. 在向导上选择下一步，或者在编辑操作页面上选择保存。

创建到 Bitbucket Cloud 的连接 (CLI)

您可以使用 AWS Command Line Interface (AWS CLI) 来创建连接。

> **Note**
>
> 您可以创建到 Bitbucket Cloud 存储库的连接。不支持已安装的 Bitbucket 提供程序类型（如 Bitbucket 服务器）。

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 AWS CloudFormation，使用控制台编辑连接以使其处于状态`AVAILABLE`。

创建连接

1. 打开终端（Linux、macOS 或 Unix）或命令提示符（Windows）。 AWS CLI 使用运行create-connection命令，`--connection-name`为您的连接指定`--provider-type`和。在此示例中，第三方提供方名称为 `Bitbucket`，指定的连接名称为 `MyConnection`。

   ```
   aws codestar-connections create-connection --provider-type Bitbucket --connection-name MyConnection
   ```

   如果成功，该命令将返回类似以下内容的连接 ARN 信息。

   ```
   {
       "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/aEXAMPLE-8aad-4d5d-8878-dfcab0bc441f"
   }
   ```

2. 使用控制台完成连接。有关更多信息，请参阅更新挂起的连接。

3. 管道默认会在向连接源存储库推送代码时检测更改。要配置手动发布或 Git 标签管道触发器配置，请执行以下操作之一：

   - 要将管道触发器配置为仅通过手动发布启动，请在配置中添加以下行：

     ```
     "DetectChanges": "false",
     ```

   - 要将管道触发器配置配置配置为使用触发器进行筛选，请在中查看更多详细信息筛选代码推送或拉取请求的触发器。例如，以下内容将 Git 标签添加到管道 JSON 定义的管道级别。在此示例中，`release-v0` 和 `release-v1` 是要包含的 Git 标签，`release-v2` 是要排除的 Git 标签。

     ```
     "triggers": [
                 {
                     "providerType": "CodeStarSourceConnection",
                     "gitConfiguration": {
                         "sourceActionName": "Source",
                         "push": [
                             {
                                 "tags": {
                                     "includes": [
                                         "release-v0", "release-v1"
                                     ],
                                     "excludes": [
                                         "release-v2"
     ```

```
                            ]
                        }
                    }
                ]
            }
        }
    ]
```

CodeCommit 源操作和 EventBridge

要在中添加 CodeCommit 源操作 CodePipeline，可以选择：

- 使用 CodePipeline 控制台的 “创建管道” 向导 (创建管道（控制台）) 或 “编辑” 操作页面选择CodeCommit提供者选项。控制台会创建一条 EventBridge规则，当源发生变化时，该规则会启动您的管道。
- 使用 AWS CLI 为操作添加操作配置并创建其他资源，如下所示：`CodeCommit`
  - 使用 CodeCommit 中的 `CodeCommit` 示例操作配置来创建操作，如创建管道（CLI） 中所示。
  - 更改检测方法默认为通过轮询源来启动管道。应禁用定期检查并手动创建更改检测规则。使用以下方法之一：为 CodeCommit 来源创建 EventBridge 规则（控制台）、为 CodeCommit 源创建 EventBridge 规则 (CLI)或为 CodeCommit 来源（AWS CloudFormation 模板）创建 EventBridge 规则 。

主题

- 为 CodeCommit 来源创建 EventBridge 规则（控制台）
- 为 CodeCommit 源创建 EventBridge 规则 (CLI)
- 为 CodeCommit 来源（AWS CloudFormation 模板）创建 EventBridge 规则

为 CodeCommit 来源创建 EventBridge 规则（控制台）

> **Important**
>
> 如果您使用控制台创建或编辑管道，则会为您创建 EventBridge 规则。

创建用于 CodePipeline 操作的 EventBridge 规则

1. 打开亚马逊 EventBridge 控制台，网址为 https://console.aws.amazon.com/events/。
2. 在导航窗格中，选择规则。保留选中的默认总线或选择一个事件总线。选择创建规则。
3. 在名称中，输入规则的名称。
4. 对于规则类型，选择具有事件模式的规则。选择下一步。
5. 在 “事件来源” 下，选择AWS 事件或 EventBridge 合作伙伴事件。
6. 在示例事件类型下，选择 AWS 事件。
7. 在示例事件中，键入 CodeCommit 作为要筛选的关键字。选择 “CodeCommit 存储库状态更改”。
8. 在创建方法下，选择自定义模式（JSON 编辑器）。

   粘贴下面提供的事件模式。以下是带有名为的分支的`MyTestRepo`存储库的 “事件” 窗口中的示例CodeCommit事件模式`main`：

```
{
  "source": [
    "aws.codecommit"
  ],
  "detail-type": [
    "CodeCommit Repository State Change"
  ],
  "resources": [
    "arn:aws:codecommit:us-west-2:80398EXAMPLE:MyTestRepo"
  ],
  "detail": {
    "referenceType": [
      "branch"
    ],
    "referenceName": [
      "main"
    ]
  }
}
```

9. 在目标中，选择CodePipeline。
10. 输入该规则将启动的管道的管道 ARN。

> **Note**
>
> 在运行 get-pipeline 命令后，您可以在元数据输出中找到管道 ARN。管道 ARN 是使用以下格式构造的：
> arn:aws:codepipeline:*region*:*account*:*pipeline-name*
> 示例管道 ARN：
> `arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline`

11. 要创建或指定一个 IAM 服务角色来授予调用与您的 EventBridge 规则关联的目标的 EventBridge 权限（在本例中，目标是 CodePipeline），请执行以下操作：
    - 选择 “为此特定资源创建新角色” 以创建服务角色，该角色 EventBridge授予您启动管道执行的权限。
    - 选择 “使用现有角色” 输入一个服务角色，该角色 EventBridge 授予您启动管道执行的权限。
12. 选择下一步。
13. 在标签页面上，选择下一步。
14. 在查看和创建页面上，检查规则配置。如果您对规则满意，请选择 Create rule。

为 CodeCommit 源创建 EventBridge 规则 (CLI)

调用 put-rule 命令，在命令中指定：

- 唯一地标识创建的规则的名称。在您创建的与 AWS 账户 CodePipeline 关联的所有管道中，此名称必须是唯一的。
- 规则使用的源事件模式和详细信息字段。有关更多信息，请参阅 Amazon EventBridge 和事件模式。

创建以事件源 CodeCommit 为目标的 EventBridge 规则 CodePipeline

1. 添加用于调 EventBridge CodePipeline 用规则的权限。有关更多信息，请参阅使用适用于 Amazon EventBridge 的基于资源的政策。
    a. 使用以下示例创建允许 EventBridge 担任服务角色的信任策略。将信任策略命名为 `trustpolicyforEB.json`。

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

```
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "events.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

b. 使用以下命令创建 Role-for-MyRule 角色并附加信任策略。

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. 为名为 MyFirstPipeline 的管道创建权限策略 JSON，如此示例中所示。将权限策略命名为 permissionspolicyforEB.json。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-west-2:80398EXAMPLE:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 使用以下命令将 CodePipeline-Permissions-Policy-for-EB 权限策略附加到 Role-for-MyRule 角色。

我为何做出此更改？ 将此策略添加到角色会为创建权限 EventBridge。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. 调用 put-rule 命令并包含 --name、--event-pattern 和 --role-arn 参数。

我为何做出此更改？ 此命令将允许 AWS CloudFormation 创建事件。

以下示例命令创建一个名为 MyCodeCommitRepoRule 的规则。

```
aws events put-rule --name "MyCodeCommitRepoRule" --event-pattern "{\"source\":
[\"aws.codecommit\"],\"detail-type\":[\"CodeCommit Repository State Change\"],
\"resources\":[\"repository-ARN\"],\"detail\":{\"referenceType\":[\"branch\"],
\"referenceName\":[\"main\"]}}" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-
MyRule"
```

3. 要添加 CodePipeline 为目标，请调用put-targets命令并添加以下参数：

   - --rule 参数与您使用 put-rule 创建的 rule_name 结合使用。
   - --targets 参数与目标列表中该目标的列表 Id 以及目标管道的 ARN 结合使用。

   以下示例命令为名为 MyCodeCommitRepoRule 的规则指定此内容，目标 Id 由数字 1 组成，这指示此内容位于规则的目标列表中，而这是目标 1。示例命令还为管道指定一个示例 ARN。管道在存储库中发生更改时启动。

```
aws events put-targets --rule MyCodeCommitRepoRule --targets
 Id=1,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
```

编辑管道的 PollForSourceChanges参数

> **Important**
>
> 使用此方法创建管道时，如果 PollForSourceChanges 参数未明确设置为 false，则默认为 true。添加基于事件的更改检测时，必须将参数添加到输出并将其设置为 false 以禁用轮询。否则，您的管道将针对单个源更改启动两次。有关更多信息，请参阅 PollForSourceChanges 参数的默认设置。

1. 运行 get-pipeline 命令以将管道结构复制到 JSON 文件中。例如，对于名为 MyFirstPipeline 的管道，运行以下命令：

```
aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
```

该命令不会返回任何结果，但您创建的文件将出现在您运行命令所在的目录中。

2. 在任何纯文本编辑器中打开 JSON 文件并通过将 `PollForSourceChanges` 参数更改为 `false` 来编辑源阶段，如此示例中所示。

   我为何做出此更改？ 将此参数更改为 `false` 将关闭定期检查，因此您只能使用基于事件的更改检测。

   ```
   "configuration": {
       "PollForSourceChanges": "false",
       "BranchName": "main",
       "RepositoryName": "MyTestRepo"
   },
   ```

3. 如果您要使用通过 get-pipeline 命令检索到的管道结构，请删除 JSON 文件中的 `metadata` 行。否则，update-pipeline 命令无法使用它。删除 `"metadata": { }` 行以及 `"created"`、`"pipelineARN"` 和 `"updated"` 字段。

   例如，从结构中删除以下各行：

   ```
   "metadata": {
       "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
       "created": "date",
       "updated": "date"
   },
   ```

   保存该文件。

4. 要应用更改，请运行 update-pipeline 命令，指定管道 JSON 文件：

   > **Important**
   >
   > 务必在文件名前包含 `file://`。此命令中需要该项。

   ```
   aws codepipeline update-pipeline --cli-input-json file://pipeline.json
   ```

   该命令会返回编辑后的管道的整个结构。

> **Note**
>
> update-pipeline 命令会停止管道。如果在运行 update-pipeline 命令时正在通过管道运行修订，则该运行会被停止。您必须手动启动管道，通过升级后的管道运行此修订。使用 **start-pipeline-execution** 命令手动启动您的管道。

为 CodeCommit 来源（AWS CloudFormation 模板）创建 EventBridge 规则

AWS CloudFormation 要使用创建规则，请更新您的模板，如下所示。

更新您的管道 AWS CloudFormation 模板并创建 EventBridge 规则

1. 在模板下的模板中`Resources`，使用`AWS::IAM::Role` AWS CloudFormation 资源配置允许您的事件启动管道的 IAM 角色。此条目将创建一个使用两个策略的角色：

   - 第一个策略允许代入角色。
   - 第二个策略提供启动管道所需的权限。

   我为何做出此更改？ 添加`AWS::IAM::Role`资源可以 AWS CloudFormation 为创建权限 EventBridge。此资源已添加到您的 AWS CloudFormation 堆栈中。

   YAML

```
EventRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: 2012-10-17
      Statement:
        -
          Effect: Allow
          Principal:
            Service:
              - events.amazonaws.com
          Action: sts:AssumeRole
    Path: /
    Policies:
      -
```

```
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
```

JSON

```
"EventRole": {
  "Type": "AWS::IAM::Role",
  "Properties": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": [
              "events.amazonaws.com"
            ]
          },
          "Action": "sts:AssumeRole"
        }
      ]
    },
    "Path": "/",
    "Policies": [
      {
        "PolicyName": "eb-pipeline-execution",
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Action": "codepipeline:StartPipelineExecution",
              "Resource": {
                "Fn::Join": [
                  "",
                  [
```

```
                    "arn:aws:codepipeline:",
                    {
                      "Ref": "AWS::Region"
                    },
                    ":",
                    {
                      "Ref": "AWS::AccountId"
                    },
                    ":",
                    {
                      "Ref": "AppPipeline"
                    }
                  ]

...
```

2. 在模板的下方`Resources`，使用`AWS::Events::Rule` AWS CloudFormation 资源添加 EventBridge 规则。此事件模式会创建一个事件，以监控向存储库推送更改的操作。当 EventBridge 检测到存储库状态更改时，将在目标管道`StartPipelineExecution`上调用该规则。

   我为何做出此更改？ 添加`AWS::Events::Rule`资源 AWS CloudFormation 即可创建事件。此资源已添加到您的 AWS CloudFormation 堆栈中。

   YAML

```
 EventRule:
   Type: AWS::Events::Rule
   Properties:
     EventPattern:
       source:
         - aws.codecommit
       detail-type:
         - 'CodeCommit Repository State Change'
       resources:
         - !Join [ '', [ 'arn:aws:codecommit:', !Ref 'AWS::Region', ':', !Ref
'AWS::AccountId', ':', !Ref RepositoryName ] ]
       detail:
         event:
           - referenceCreated
           - referenceUpdated
         referenceType:
           - branch
```

```
          referenceName:
            - main
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline
```

JSON

```
"EventRule": {
  "Type": "AWS::Events::Rule",
  "Properties": {
    "EventPattern": {
      "source": [
        "aws.codecommit"
      ],
      "detail-type": [
        "CodeCommit Repository State Change"
      ],
      "resources": [
        {
          "Fn::Join": [
            "",
            [
              "arn:aws:codecommit:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
                "Ref": "RepositoryName"
              }
            ]
          ]
        }
```

```
      ],
      "detail": {
        "event": [
          "referenceCreated",
          "referenceUpdated"
        ],
        "referenceType": [
          "branch"
        ],
        "referenceName": [
          "main"
        ]
      }
    },
    "Targets": [
      {
        "Arn": {
          "Fn::Join": [
            "",
            [
              "arn:aws:codepipeline:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
                "Ref": "AppPipeline"
              }
            ]
          ]
        },
        "RoleArn": {
          "Fn::GetAtt": [
            "EventRole",
            "Arn"
          ]
        },
        "Id": "codepipeline-AppPipeline"
      }
    ]
```

```
  }
},
```

3. 将更新后的模板保存到本地计算机，然后打开 AWS CloudFormation 控制台。
4. 选择堆栈，然后选择为当前堆栈创建更改集。
5. 上传模板，然后查看 AWS CloudFormation中列出的更改。这些是要对堆栈进行的更改。您应在列表中看到新资源。
6. 选择执行。

编辑管道的 PollForSourceChanges参数

> **Important**
>
> 许多情况下，当您创建管道时，PollForSourceChanges 参数默认为 true。添加基于事件的更改检测时，必须将参数添加到输出并将其设置为 false 以禁用轮询。否则，您的管道将针对单个源更改启动两次。有关更多信息，请参阅 PollForSourceChanges 参数的默认设置。

- 在模板中，将 PollForSourceChanges 更改为 false。如果您未在管道定义中包含 PollForSourceChanges，请添加它并将它设置为 false。

  我为何做出此更改？ 将此参数更改为 false 将关闭定期检查，因此您只能使用基于事件的更改检测。

  YAML

```
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: CodeCommit
            OutputArtifacts:
              - Name: SourceOutput
            Configuration:
              BranchName: !Ref BranchName
              RepositoryName: !Ref RepositoryName
```

```
                PollForSourceChanges: false
              RunOrder: 1
```

JSON

```
{
  "Name": "Source",
  "Actions": [
    {
      "Name": "SourceAction",
      "ActionTypeId": {
        "Category": "Source",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "CodeCommit"
      },
      "OutputArtifacts": [
        {
          "Name": "SourceOutput"
        }
      ],
      "Configuration": {
        "BranchName": {
          "Ref": "BranchName"
        },
        "RepositoryName": {
          "Ref": "RepositoryName"
        },
        "PollForSourceChanges": false
      },
      "RunOrder": 1
    }
  ]
},
```

GitHub 连接

您可以使用连接来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。

> **Note**
>
> 此功能不适用于亚太地区（香港）、亚太地区（海得拉巴）、亚太地区（雅加达）、亚太地区（墨尔本）、亚太地区（墨尔本）、亚太地区（大阪）、非洲（开普敦）、中东（巴林）、中东（阿联酋）、欧洲（西班牙）、欧洲（苏黎世）、以色列（特拉维夫）或 AWS GovCloud（美国西部）地区。要参考其他可用操作，请参阅 产品和服务与 CodePipeline。有关在欧洲地区（米兰）区域使用此操作的注意事项，请参阅CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、 GitLab .com 和 GitLab 自我管理操作中的说明。

要在中为您的企业云存储库 GitHub 或 GitHub 企业云存储库添加源操作 CodePipeline，您可以选择：

- 使用 CodePipeline 控制台的 “创建管道” 向导或 “编辑” 操作页面选择 GitHub （版本 2）提供者选项。参阅创建与 GitHub 企业服务器（控制台）的连接以添加操作。控制台可帮助您创建连接资源。

  > **Note**
  >
  > 有关指导您如何添加 GitHub 连接和在管道中使用 “完全克隆” 选项的教程，请参阅教程：使用带有 GitHub 管道源的完整克隆。

- 使用 CLI，通过创建管道（CLI）中所示的步骤，添加提供方为 `GitHub` 的 `CodeStarSourceConnection` 操作的操作配置。

> **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。

开始前的准备工作：

- 您必须已使用创建账户 GitHub。
- 您必须已经创建了 GitHub 代码存储库。
- 如果您的 CodePipeline 服务角色是在 2019 年 12 月 18 日之前创建的，则可能需要更新其权限才能`codestar-connections:UseConnection`用于 AWS CodeStar 连接。有关说明，请参阅向 CodePipeline 服务角色添加权限。

> **Note**
>
> 要创建连接，您必须是 GitHub 组织所有者。对于不属于组织的存储库，您必须是存储库拥有者。

主题

创建与 GitHub（控制台）的连接

使用这些步骤使用 CodePipeline 控制台为您的 GitHub或 GitHub 企业云存储库添加连接操作。

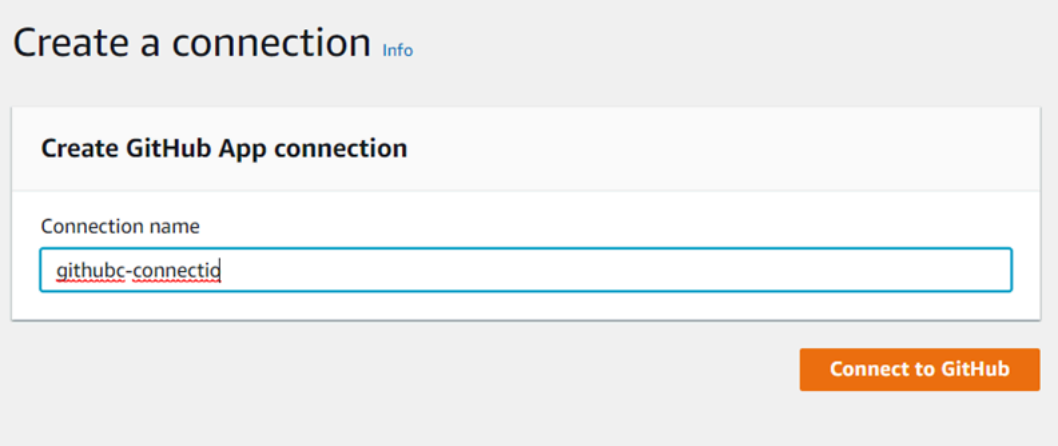
> **Note**
>
> 在这些步骤中，您可以在存储库访问下选择特定的存储库。任何未被选中的存储库都将无法访问或看见 CodePipeline。

步骤 1：创建或编辑您的管道

1. 登录 CodePipeline 控制台。
2. 选择以下选项之一。
   - 选择创建管道。按照创建管道 中的步骤操作，完成第一个屏幕，然后选择下一步。在 “来源” 页面的 “源提供程序” 下，选择 GitHub （版本 2）。
   - 选择编辑现有管道。选择编辑，然后选择编辑阶段。选择添加或编辑您的源操作。在编辑操作页面的操作名称下，输入您的操作的名称。在操作提供者中，选择 GitHub （版本 2）。
3. 请执行以下操作之一：
   - 在 “连接” 下，如果您尚未创建与提供商的连接，请选择 “连接到” GitHub。继续执行步骤 2：创建与的连接 GitHub。
   - 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 3：保存连接的源操作。

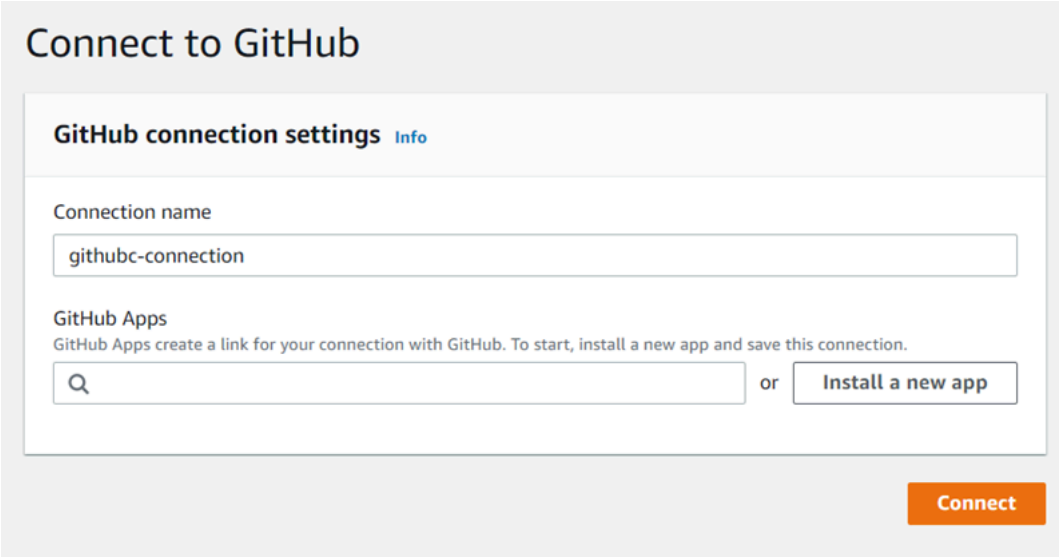
步骤 2：创建与的连接 GitHub

选择创建连接后，将出现 “Connect t GitHub o” 页面。



要创建与的连接 GitHub

1. 在 “GitHub 连接设置” 下，您的连接名称显示在 “连接名称” 中。选择 “连接到” GitHub。此时将显示访问请求页面。
2. 选择 “为 AWS 连接器授权” GitHub。连接页面将显示并显示 “GitHub 应用程序” 字段。



3. 在 “GitHub 应用程序” 下，选择应用程序安装或选择 “安装新应用程序” 来创建一个。

> **Note**
>
> 您可以为与特定提供程序的所有连接安装一个应用程序。如果您已经安装了 GitHub应用程序 AWS 连接器，请选择它并跳过此步骤。

4. 在 “安装 AWS 连接器 GitHub” 页面上，选择要安装应用程序的帐户。

> **Note**
>
> 您只需为每个 GitHub 帐户安装一次应用程序。如果您之前已安装了应用程序，则可以选择配置，继续进入应用程序安装的修改页面，也可以使用后退按钮返回到控制台。

5. 在 “安装 AWS 连接器 GitHub” 页面上，保留默认值，然后选择 “安装”。
6. 在 “Connect t o GitHub” 页面上，新安装的连接 ID 显示在 “GitHub 应用程序” 中。选择连接。

第 3 步：保存 GitHub 源操作

使用编辑操作页面上的这些步骤，将源操作与连接信息一起保存。

保存 GitHub 源操作

1. 在存储库名称中，选择第三方存储库的名称。
2. 如果您的操作是操作，则可以在 Pip elin e 触发器下添加触发器。 CodeConnections 要配置管道触发器配置并选择使用触发器进行筛选，请在中查看更多详细信息筛选代码推送或拉取请求的触发器。
3. 在输出构件格式中，您必须为构件选择格式。
   - 要使用默认方法存储 GitHub 操作的输出对象，请选择CodePipeline 默认。该操作访问存储库中的文件，并将工件 GitHub 存储在管道工件存储区的 ZIP 文件中。
   - 要存储包含存储库的 URL 引用的 JSON 文件，以便下游操作可以直接执行 Git 命令，请选择完全克隆。此选项只能由 CodeBuild 下游操作使用。

     如果选择此选项，则需要更新 CodeBuild 项目服务角色的权限，如所示添加连接 Bitbucket、GitHub、En GitHub terprise Server 或 GitLab .com 的 CodeBuild GitClone 权限。如需查看教程以了解如何使用完整克隆选项，请参阅教程：使用带有 GitHub 管道源的完整克隆。
4. 在向导上选择下一步，或者在编辑操作页面上选择保存。

创建与 GitHub (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 AWS CloudFormation，使用控制台编辑连接以使其处于状态AVAILABLE。

创建连接

1. 打开终端（Linux、macOS 或 Unix）或命令提示符（Windows）。 AWS CLI 使用运行create-connection命令，--connection-name为您的连接指定--provider-type和。在此示例中，第三方提供方名称为 GitHub，指定的连接名称为 MyConnection。

   ```
   aws codestar-connections create-connection --provider-type GitHub --connection-name
    MyConnection
   ```

   如果成功，该命令将返回类似以下内容的连接 ARN 信息。

   ```
   {
       "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/
   aEXAMPLE-8aad-4d5d-8878-dfcab0bc441f"
   }
   ```

2. 使用控制台完成连接。有关更多信息，请参阅更新挂起的连接。
3. 管道默认会在向连接源存储库推送代码时检测更改。要配置手动发布或 Git 标签管道触发器配置，请执行以下操作之一：

   - 要将管道触发器配置为仅通过手动发布启动，请在配置中添加以下行：

     ```
     "DetectChanges": "false",
     ```

   - 要将管道触发器配置配置配置为使用触发器进行筛选，请在中查看更多详细信息筛选代码推送或拉取请求的触发器。例如，以下内容添加到管道 JSON 定义的管道级别。在此示例中，release-v0 和 release-v1 是要包含的 Git 标签，release-v2 是要排除的 Git 标签。

     ```
     "triggers": [
                 {
                     "providerType": "CodeStarSourceConnection",
                     "gitConfiguration": {
                         "sourceActionName": "Source",
                         "push": [
     ```

```
                    {
                        "tags": {
                            "includes": [
                                "release-v0", "release-v1"
                            ],
                            "excludes": [
                                "release-v2"
                            ]
                        }
                    }
                ]
            }
        }
    ]
```

GitHub 企业服务器连接

Connections 允许您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。要将您的第三方存储库关联为管道的源，您应使用连接。

> **Note**
>
> 此功能不适用于亚太地区（香港）、亚太地区（海得拉巴）、亚太地区（雅加达）、亚太地区（墨尔本）、亚太地区（墨尔本）、亚太地区（大阪）、非洲（开普敦）、中东（巴林）、中东（阿联酋）、欧洲（西班牙）、欧洲（苏黎世）、以色列（特拉维夫）或 AWS GovCloud（美国西部）地区。要参考其他可用操作，请参阅 产品和服务与 CodePipeline。有关在欧洲地区（米兰）区域使用此操作的注意事项，请参阅CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、 GitLab .com 和 GitLab 自我管理操作中的说明。

要在中添加 GitHub 企业服务器源操作 CodePipeline，可以选择以下任一选项：

- 使用 CodePipeline 控制台的 “创建管道” 向导或 “编辑” 操作页面选择GitHub 企业服务器提供者选项。参阅创建与 GitHub 企业服务器（控制台）的连接以添加操作。控制台可帮助您创建主机资源和连接资源。
- 使用 CLI，添加提供方为 `GitHubEnterpriseServer` 的 `CreateSourceConnection` 操作的操作配置，并创建您的资源：
  - 要创建连接资源，请参阅创建主机并连接到 GitHub 企业服务器 (CLI)，以便使用 CLI 创建主机资源和连接资源。

- 使用CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、GitLab .com 和 GitLab 自我管理操作中的 `CreateSourceConnection` 示例操作配置来添加操作，如创建管道（CLI）中所示。

> **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。

开始前的准备工作：

- 您必须已在 GitHub 企业服务器上创建帐户并在基础架构上安装了 GitHub 企业服务器实例。

  > **Note**
  >
  > 每个 VPC 一次只能与一台主机（GitHub 企业服务器实例）关联。

- 您必须已经使用 GitHub 企业服务器创建了代码存储库。

主题

- 创建与 GitHub 企业服务器（控制台）的连接
- 创建主机并连接到 GitHub 企业服务器 (CLI)

创建与 GitHub 企业服务器（控制台）的连接

使用这些步骤使用 CodePipeline 控制台为您的 GitHub企业服务器存储库添加连接操作。

> **Note**
>
> GitHub 企业服务器连接仅提供对用于创建连接的 GitHub 企业服务器帐户所拥有的存储库的访问权限。

开始前的准备工作：

要将主机连接到 E GitHub nterprise Server，必须完成为连接创建主机资源的步骤。请参阅管理连接主机。

步骤 1：创建或编辑您的管道

创建或编辑您的管道

1. 登录 CodePipeline 控制台。
2. 选择以下选项之一。
   - 选择创建管道。按照创建管道 中的步骤操作，完成第一个屏幕，然后选择下一步。在 “源” 页上的 “源提供商” 下，选择 “GitHub 企业服务器”。
   - 选择编辑现有管道。选择编辑，然后选择编辑阶段。选择添加或编辑您的源操作。在编辑操作页面的操作名称下，输入您的操作的名称。在操作提供者中，选择GitHub 企业服务器。
3. 请执行以下操作之一：
   - 在 “连接” 下，如果您尚未创建与提供商的连接，请选择 Connection to En GitHub terprise Server。继续执行步骤 2：创建与 GitHub 企业服务器的连接。
   - 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 3：保存连接的源操作。

创建与 GitHub企业服务器的连接

选择创建连接后，将显示 “Connect to En GitHub terprise Server” 页面。

> **Important**
>
> AWS CodeConnections 由于版本中存在已知问题，不支持 GitHub 企业服务器版本 2.22.0。要执行连接，请升级到版本 2.22.1 或最新的可用版本。

连接到 GitHub 企业服务器

1. 在连接名称，输入您的连接的名称。
2. 在 URL 中，输入服务器的终端节点。

> **Note**
>
> 如果所提供的 URL 已用于为连接设置 GitHub企业服务器，则系统将提示您选择先前为该终端节点创建的主机资源 ARN。

3. 如果您已将服务器启动到 Amazon VPC 中，并且想要连接 VPC，请选择 Use a VPC (使用 VPC) 并完成以下操作。

   a. 在 VPC ID 中，选择您的 VPC ID。请务必为安装 GitHub 企业服务器实例的基础设施选择 VPC，或者选择可通过 VPN 或 Direct Connect 访问 GitHub企业服务器实例的 VPC。

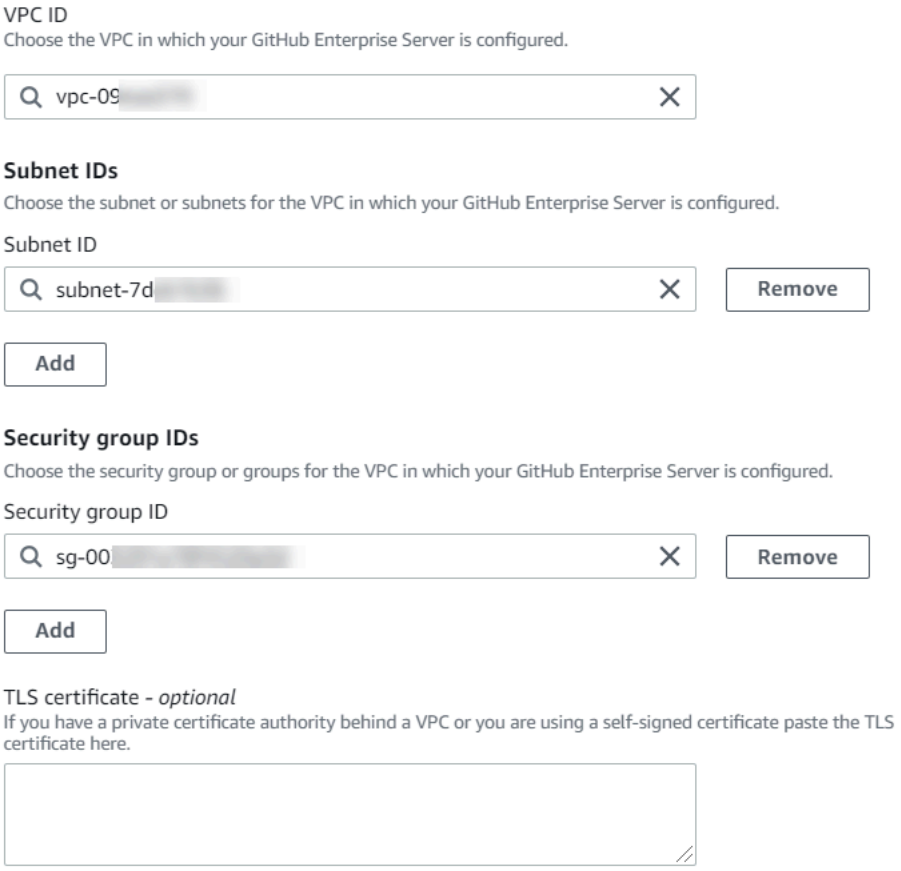
   b. 在 Subnet ID (子网 ID) 下，选择 Add (添加)。在字段中，选择要用于主机的子网 ID。您可以选择最多 10 个子网。

      请务必为安装 GitHub 企业服务器实例的基础架构选择子网，或者选择可以通过 VPN 或 Direct Connect 访问已安装的 GitHub 企业服务器实例的子网。

   c. 在 Security group IDs (安全组 ID) 下，选择 Add (添加)。在字段中，选择要用于主机的安全组。您最多可以选择 10 个安全组。

      请务必为安装 GitHub 企业服务器实例的基础架构选择安全组，或者选择可以通过 VPN 或 Direct Connect 访问已安装的 GitHub 企业服务器实例的安全组。

   d. 如果您配置了私有 VPC，并且已将 GitHub 企业服务器实例配置为使用非公共证书颁发机构执行 TLS 验证，请在 TLS 证书中输入您的证书 ID。TLS 证书值应该是证书的公有密钥。



4. 选择 “连接到 GitHub 企业服务器”。创建的连接显示为待处理状态。将使用您提供的服务器信息为连接创建一个主机资源。对于主机名，将使用 URL。
5. 选择更新待处理的连接。
6. 如果出现提示，请在 GitHub 企业登录页面上使用您的 GitHub企业凭据登录。
7. 在创建 GitHub 应用程序页面上，为您的应用程序选择一个名称。
8. 在 GitHub 授权页面上，选择授权<app-name>。
9. 在应用程序安装页面上，一条消息显示连接器应用程序已准备就绪，可以安装了。如果您有多个组织，系统可能会提示您选择要安装该应用程序的组织。

   选择要安装应用程序的存储库设置。选择安装。
10. 连接页面显示已创建的连接处于可用状态。

步骤 3：保存您的 GitHub 企业服务器源操作

使用向导或编辑操作页面上的这些步骤，将源操作与连接信息一起保存。

完成并保存您的源操作和连接

1. 在存储库名称中，选择第三方存储库的名称。
2. 如果您的操作是操作，则可以在 Pip elin e 触发器下添加触发器。 CodeConnections 要配置管道触发器配置并选择使用触发器进行筛选，请在中查看更多详细信息筛选代码推送或拉取请求的触发器。
3. 在输出构件格式中，您必须为构件选择格式。
   - 要使用默认方法存储 GitHub 企业服务器操作的输出对象，请选择CodePipeline默认。该操作访问 GitHub 企业服务器存储库中的文件，并将工件存储在管道对象存储区的 ZIP 文件中。
   - 要存储包含存储库的 URL 引用的 JSON 文件，以便下游操作可以直接执行 Git 命令，请选择完全克隆。此选项只能由 CodeBuild 下游操作使用。
4. 在向导上选择下一步，或者在编辑操作页面上选择保存。

创建主机并连接到 GitHub 企业服务器 (CLI)

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 AWS CloudFormation，使用控制台编辑连接以使其处于状态AVAILABLE。

您可以使用 AWS Command Line Interface (AWS CLI) 为已安装的连接创建主机。

> **Note**
>
> 每个 GitHub 企业服务器帐户只能创建一次主机。您与特定 GitHub 企业服务器帐户的所有连接都将使用同一台主机。

您可以使用主机来表示安装第三方提供程序的基础设施的终端节点。使用 CLI 完成主机创建后，主机将处于待处理状态。然后，您应设置或注册主机，使之转为可用状态。主机可用后，完成创建连接的步骤。

为此，请使用 create-host 命令。

> **Important**
>
> 默认情况下，通过创建的主机 AWS CLI 处于Pending状态。使用 CLI 创建主机后，可使用控制台或 CLI 设置主机以使其状态为 Available。

要创建主机

1. 打开终端（Linux、macOS 或 Unix）或命令提示符（Windows）。 AWS CLI 使用运行create-host命令，--provider-endpoint为您的连接指定--name--provider-type、和。在此示例中，第三方提供程序名称为 GitHubEnterpriseServer，终端节点为 my-instance.dev。

   ```
   aws codestar-connections create-host --name MyHost --provider-type
    GitHubEnterpriseServer --provider-endpoint "https://my-instance.dev"
   ```

   如果成功，该命令将返回类似以下内容的主机 Amazon 资源名称（ARN）信息。

   ```
   {
   ```

```
    "HostArn": "arn:aws:codestar-connections:us-west-2:account_id:host/My-
Host-28aef605"
}
```

完成此步骤后，主机处于 PENDING 状态。

2. 使用控制台完成主机设置并将主机变为 Available 状态。

创建与 GitHub 企业服务器的连接

1. 打开终端（Linux、macOS 或 Unix）或命令提示符（Windows）。 AWS CLI 使用运行create-connection命令，--connection-name为您的连接指定--host-arn和。

```
aws codestar-connections create-connection --host-arn arn:aws:codestar-
connections:us-west-2:account_id:host/MyHost-234EXAMPLE --connection-name
 MyConnection
```

如果成功，该命令将返回类似以下内容的连接 ARN 信息。

```
{
    "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/
aEXAMPLE-8aad"
}
```

2. 使用控制台设置待处理的连接。
3. 管道默认会在向连接源存储库推送代码时检测更改。要配置手动发布或 Git 标签管道触发器配置，请执行以下操作之一：

   - 要将管道触发器配置为仅通过手动发布启动，请在配置中添加以下行：

     ```
     "DetectChanges": "false",
     ```

   - 要将管道触发器配置配置配置为使用触发器进行筛选，请在中查看更多详细信息筛选代码推送或拉取请求的触发器。例如，以下内容添加到管道 JSON 定义的管道级别。在此示例中，release-v0 和 release-v1 是要包含的 Git 标签，release-v2 是要排除的 Git 标签。

     ```
     "triggers": [
                 {
                     "providerType": "CodeStarSourceConnection",
                     "gitConfiguration": {
     ```

```
                "sourceActionName": "Source",
                "push": [
                    {
                        "tags": {
                            "includes": [
                                "release-v0", "release-v1"
                            ],
                            "excludes": [
                                "release-v2"
                            ]
                        }
                    }
                ]
            }
        }
    ]
```

GitLab.com 连接

Connections 允许您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。要将您的第三方存储库关联为管道的源，您应使用连接。

> **Note**
>
> 此功能不适用于亚太地区（香港）、亚太地区（海得拉巴）、亚太地区（雅加达）、亚太地区（墨尔本）、亚太地区（墨尔本）、亚太地区（大阪）、非洲（开普敦）、中东（巴林）、中东（阿联酋）、欧洲（西班牙）、欧洲（苏黎世）、以色列（特拉维夫）或 AWS GovCloud（美国西部）地区。要参考其他可用操作，请参阅 产品和服务与 CodePipeline。有关在欧洲地区（米兰）区域使用此操作的注意事项，请参阅CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、 GitLab .com 和 GitLab 自我管理操作中的说明。

要在中添加 GitLab .com 源操作 CodePipeline，您可以选择以下任一选项：

- 使用 CodePipeline 控制台的 “创建管道” 向导或 “编辑” 操作页面选择GitLab提供者选项。参阅创建到 GitLab .com（控制台）的连接以添加操作。控制台可帮助您创建连接资源。
- 使用 CLI，添加提供方为 `GitLab` 的 `CreateSourceConnection` 操作的操作配置，如下所示：
  - 要创建连接资源，请参阅 创建到 GitLab .com 的连接 (CLI)，以便使用 CLI 创建连接资源。

- 使用CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、GitLab .com 和 GitLab 自我管理操作中的 `CreateSourceConnection` 示例操作配置来添加操作，如创建管道（CLI）中所示。

> **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 GitLab .com中授权安装此连接，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。

开始前的准备工作：

- 您必须已经在 GitLab .com上创建了账户。

> **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 CodePipeline。对于群组中的仓库，您无需成为群组拥有者。

- 要为您的管道指定一个源，必须事先在 gitlab.com 上创建存储库。

主题

- 创建到 GitLab .com（控制台）的连接
- 创建到 GitLab .com 的连接 (CLI)

创建到 GitLab .com（控制台）的连接

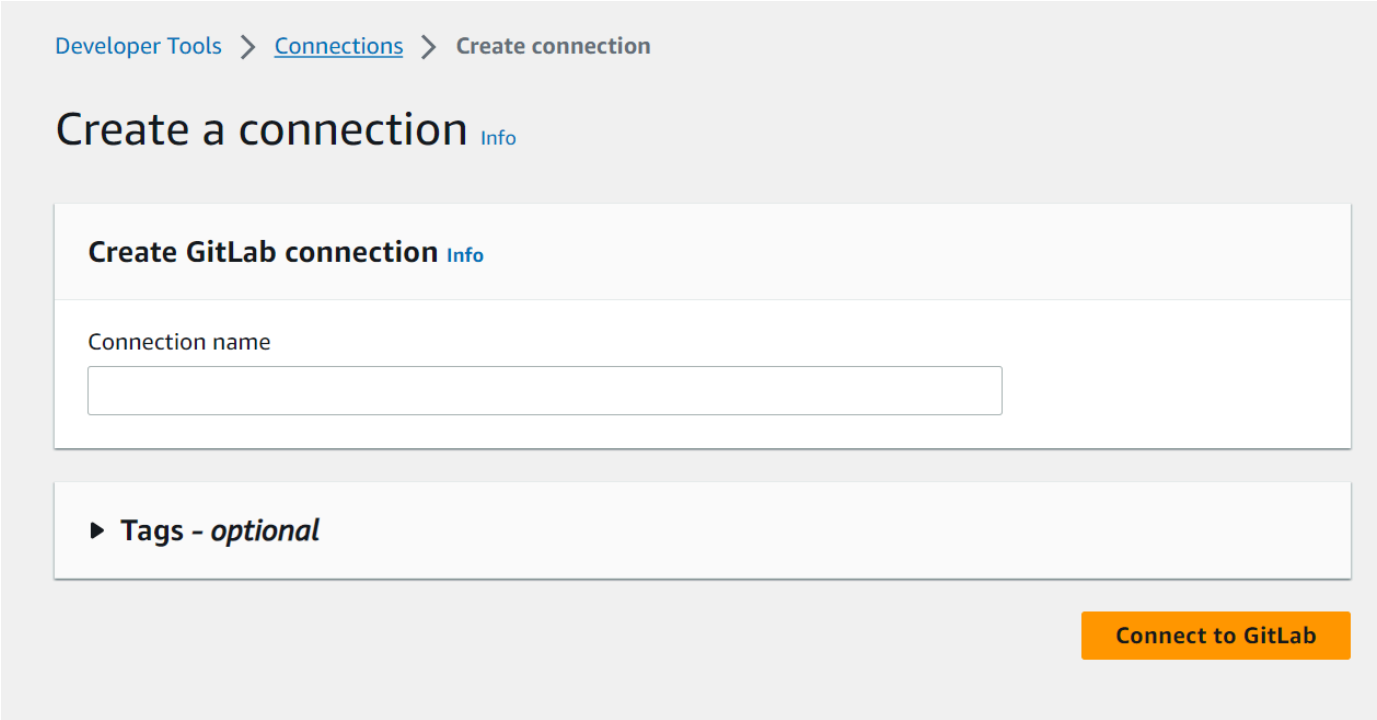
使用以下步骤使用 CodePipeline 控制台为中的项目（存储库）添加连接操作 GitLab。

创建或编辑您的管道

1. 登录 CodePipeline 控制台。
2. 选择以下选项之一。
   - 选择创建管道。按照创建管道 中的步骤操作，完成第一个屏幕，然后选择下一步。在 “来源” 页面上的 “源提供者” 下，选择GitLab。
   - 选择编辑现有管道。选择编辑，然后选择编辑阶段。选择添加或编辑您的源操作。在编辑操作页面的操作名称下，输入您的操作的名称。对于操作提供程序，选择 GitLab。
3. 请执行以下操作之一：
   - 在 “连接” 下，如果您尚未创建与提供商的连接，请选择 Connection to GitLab。继续执行步骤 4，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 9。

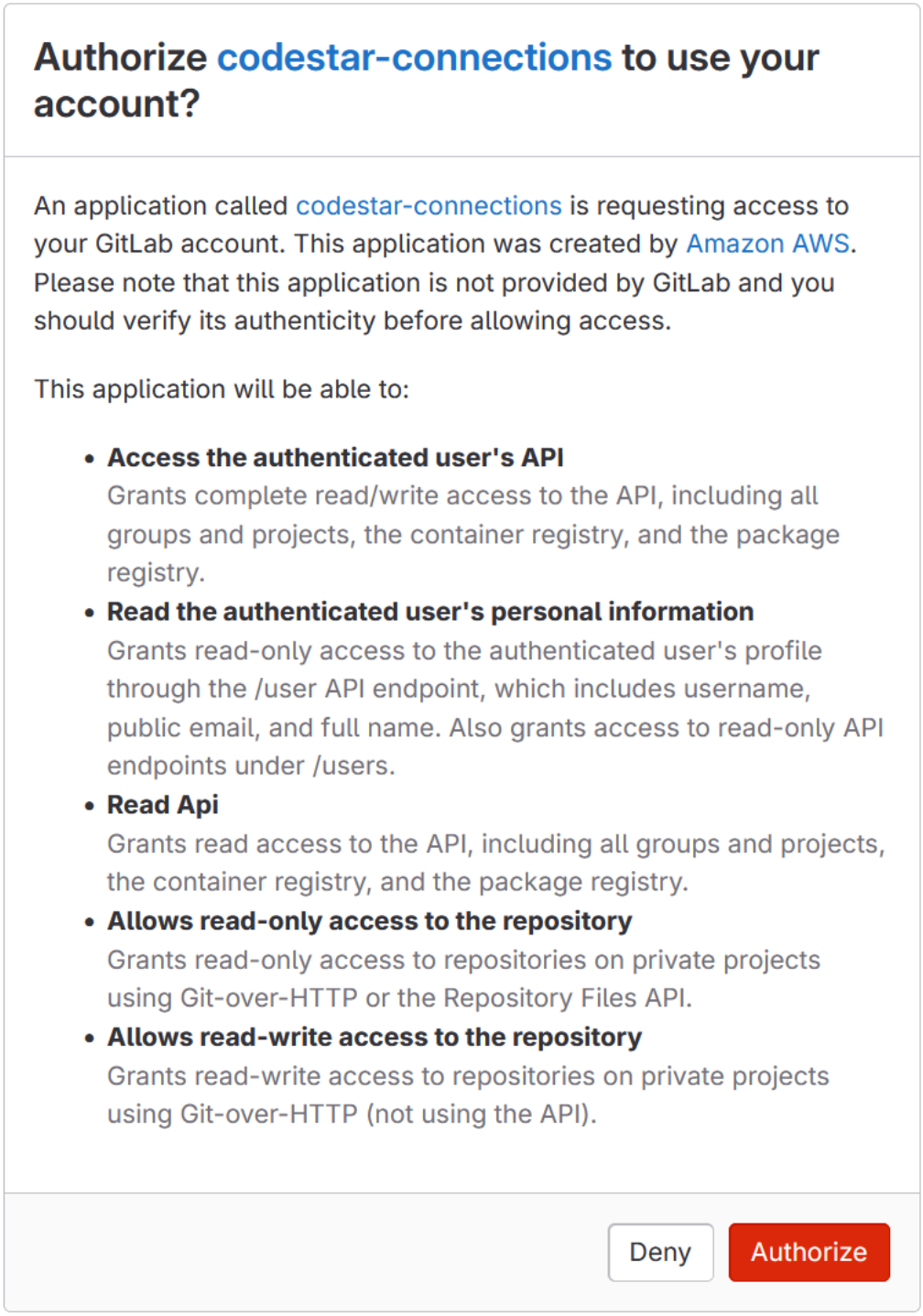
   > **Note**
   >
   > 如果您在创建 GitLab .com 连接之前关闭了弹出窗口，则需要刷新页面。

4. 要创建到 GitLab .com 存储库的连接，请在 “选择提供商” 下选择GitLab。在连接名称中，输入要创建的连接的名称。选择 Connect t GitLab o。



5. 当显示 GitLab .com的登录页面时，使用您的凭据登录，然后选择登录。
6. 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 .com账户。

   选择授权。



7. 浏览器返回到连接控制台页面。在 “创建 GitLab 连接” 下，新连接显示在 “连接名称” 中。
8. 选择 Connect t GitLab o。

   您将返回到 CodePipeline 控制台。

> **Note**
>
> 成功创建 GitLab .com 连接后，将在主窗口中显示成功横幅。
> 如果您之前未在当前计算机 GitLab 上登录过，则需要手动关闭弹出窗口。

9. 在存储库名称中， GitLab 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 # 中的`path_with_namespace`字段get-single-project。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，您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。

10. 如果您的操作是操作，则可以在 Pip elin e 触发器下添加触发器。 CodeConnections 要配置管道触发器配置并选择使用触发器进行筛选，请在中查看更多详细信息筛选代码推送或拉取请求的触发器。
11. 在分支名称中，选择您希望管道在其中检测源更改的分支。

> **Note**
>
> 如果未自动填充分支名称，说明您没有存储库的所有者访问权限。原因要么是项目名称无效，要么是所使用的连接无权访问项目/存储库。

12. 在输出构件格式中，您必须为构件选择格式。

    - 要使用默认方法存储 GitLab .com 操作的输出对象，请选择CodePipeline 默认。该操作访问 GitLab .com存储库中的文件，并将工件存储在管道工件存储区的 ZIP 文件中。
    - 要存储包含存储库的 URL 引用的 JSON 文件，以便下游操作可以直接执行 Git 命令，请选择完全克隆。此选项只能由 CodeBuild 下游操作使用。

      如果选择此选项，则需要更新 CodeBuild 项目服务角色的权限，如所示添加连接 Bitbucket、GitHub、En GitHub terprise Server 或 GitLab .com 的 CodeBuild GitClone 权限。如需查看教程以了解如何使用完整克隆选项，请参阅教程：使用带有 GitHub 管道源的完整克隆。

13. 选择保存源操作并继续。

创建到 GitLab .com 的连接 (CLI)

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 AWS CloudFormation，使用控制台编辑连接以使其处于状态AVAILABLE。

创建连接

1. 打开终端（Linux、macOS 或 Unix）或命令提示符（Windows）。 AWS CLI 使用运行create-connection命令，--connection-name为您的连接指定--provider-type和。在此示例中，第三方提供方名称为 GitLab，指定的连接名称为 MyConnection。

   ```
   aws codestar-connections create-connection --provider-type GitLab --connection-name
    MyConnection
   ```

   如果成功，该命令将返回类似以下内容的连接 ARN 信息。

   ```
   {
       "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/
   aEXAMPLE-8aad-4d5d-8878-dfcab0bc441f"
   }
   ```

2. 使用控制台完成连接。有关更多信息，请参阅更新挂起的连接。
3. 管道默认会在向连接源存储库推送代码时检测更改。要配置手动发布或 Git 标签管道触发器配置，请执行以下操作之一：

   - 要将管道触发器配置为仅通过手动发布启动，请在配置中添加以下行：

     ```
     "DetectChanges": "false",
     ```

- 要将管道触发器配置配置配置为使用触发器进行筛选，请在中查看更多详细信息筛选代码推送或拉取请求的触发器。例如，以下内容添加到管道 JSON 定义的管道级别。在此示例中，`release-v0` 和 `release-v1` 是要包含的 Git 标签，`release-v2` 是要排除的 Git 标签。

```
"triggers": [
            {
                "providerType": "CodeStarSourceConnection",
                "gitConfiguration": {
                    "sourceActionName": "Source",
                    "push": [
                        {
                            "tags": {
                                "includes": [
                                    "release-v0", "release-v1"
                                ],
                                "excludes": [
                                    "release-v2"
                                ]
                            }
                        }
                    ]
                }
            }
        ]
```

用于 GitLab 自我管理的连接

Connections 允许您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。要将您的第三方存储库关联为管道的源，您应使用连接。

> **Note**
>
> 此功能不适用于亚太地区（香港）、亚太地区（海得拉巴）、亚太地区（雅加达）、亚太地区（墨尔本）、亚太地区（墨尔本）、亚太地区（大阪）、非洲（开普敦）、中东（巴林）、中东（阿联酋）、欧洲（西班牙）、欧洲（苏黎世）、以色列（特拉维夫）或 AWS GovCloud（美国西部）地区。要参考其他可用操作，请参阅 产品和服务与 CodePipeline。有关在欧洲地区（米兰）区域使用此操作的注意事项，请参阅CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、 GitLab .com 和 GitLab 自我管理操作中的说明。

要在中添加 GitLab 自我管理的源操作 CodePipeline，您可以选择：

- 使用 CodePipeline 控制台的 “创建管道” 向导或 “编辑” 操作页面选择GitLab自我管理的提供者选项。参阅创建与 GitLab自我管理（控制台）的连接以添加操作。控制台可帮助您创建主机资源和连接资源。
- 使用 CLI，添加提供方为 `GitLabSelfManaged` 的 `CreateSourceConnection` 操作的操作配置，并创建您的资源：
  - 要创建连接资源，请参阅创建主机并连接到 GitLab自我管理 (CLI)，以便使用 CLI 创建主机资源和连接资源。
  - 使用CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、GitLab .com 和 GitLab 自我管理操作中的 `CreateSourceConnection` 示例操作配置来添加操作，如创建管道（CLI） 中所示。

> **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。

开始前的准备工作：

- 您必须已经使用自己管理安装的 GitLab 企业版或 GitLab 社区版创建了帐户， GitLab 并且安装了企业版或社区版。有关更多信息，请参阅 https://docs.gitlab.com/ee/subscriptions/self_managed/。

> **Note**
>
> 连接只为用于创建并授权连接的账户提供访问权限。

> **Note**
>
> 您可以创建与拥有所有者角色的存储库的连接 GitLab，然后该连接可以与资源一起使用，例如 CodePipeline。对于群组中的仓库，您无需成为群组拥有者。

- 您必须已经创建了仅具有以下范围缩小权限的 GitLab 个人访问令牌 (PAT)：api。有关更多信息，请参阅 https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html。您必须是管理员才能创建和使用 PAT。

> **Note**
>
> 您的 PAT 用于对主机进行授权，不会以其它方式存储或由连接使用。要设置主机，您可以创建一个临时 PAT，然后在设置主机之后，可以删除 PAT。

- 您可以选择提前设置主机。无论是否使用 VPC，您都可以设置主机。有关 VPC 配置的详细信息以及有关创建主机的其他信息，请参阅创建主机。

主题

创建与 GitLab自我管理（控制台）的连接

使用这些步骤使用 CodePipeline 控制台为您的 GitLab自管理存储库添加连接操作。

> **Note**
>
> GitLab 自管理连接仅提供对用于创建连接的 GitLab 自我管理账户所拥有的存储库的访问权限。

开始前的准备工作：

要实现 GitLab 自我管理的主机连接，您必须完成为连接创建主机资源的步骤。请参阅管理连接主机。

步骤 1：创建或编辑您的管道

创建或编辑您的管道

1. 登录 CodePipeline 控制台。
2. 选择以下选项之一。
   - 选择创建管道。按照创建管道 中的步骤操作，完成第一个屏幕，然后选择下一步。在来源页面的来源提供商下，选择GitLab 自我管理。
   - 选择编辑现有管道。选择编辑，然后选择编辑阶段。选择添加或编辑您的源操作。在编辑操作页面的操作名称下，输入您的操作的名称。在操作提供者中，选择GitLab 自我管理。

3. 请执行以下操作之一：

   - 在 “连接” 下，如果您尚未创建与提供商的连接，请选择 “连接到 GitLab 自我管理”。继续执行步骤 2：创建 GitLab 自我管理连接。
   - 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 3：保存连接的源操作。

创建与 GitLab 自我管理的连接

选择创建连接后，将显示 “Connect t GitLab o 自我管理” 页面。

连接到 GitLab 自我管理

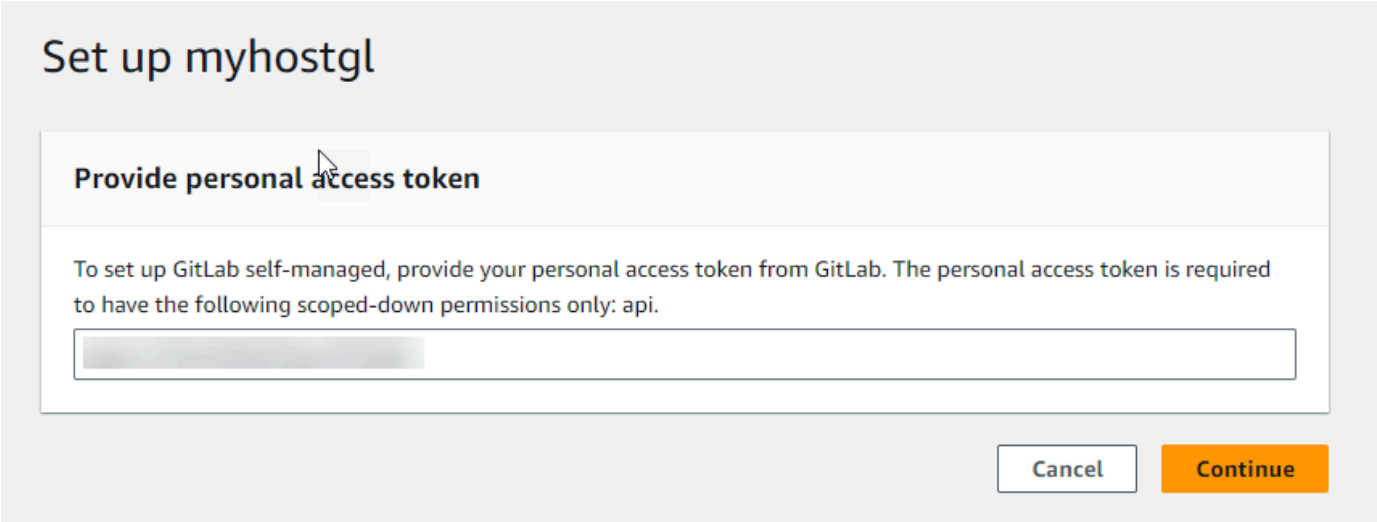
1. 在连接名称，输入您的连接的名称。
2. 在 URL 中，输入服务器的终端节点。

   > **Note**
   >
   > 如果提供的 URL 已用于为连接设置主机，系统将提示您选择之前为该端点创建的主机资源 ARN。

3. 如果您已将服务器启动到 Amazon VPC 中，并且想要连接 VPC，请选择使用 VPC 并填写 VPC 的信息。
4. 选择 Connect 以进行 GitLab 自我管理。创建的连接显示为待处理状态。将使用您提供的服务器信息为连接创建一个主机资源。对于主机名，将使用 URL。
5. 选择更新待处理的连接。
6. 如果页面打开时显示一条重定向消息，确认您是否要继续访问该提供程序，请选择继续。输入该提供程序的授权。
7. 将显示设置 ***host_name*** 页面。在提供个人访问令牌中，仅向你的 GitLab PAT 提供以下限定范围的权限：。`api`

   > **Note**
   >
   > 只有管理员才能创建和使用 PAT。

   选择继续。



8. 连接页面显示已创建的连接处于可用状态。

步骤 3：保存您 GitLab自行管理的源代码操作

使用向导或编辑操作页面上的这些步骤，将源操作与连接信息一起保存。

完成并保存您的源操作和连接

1. 在存储库名称中，选择第三方存储库的名称。
2. 如果您的操作是操作，则可以在 Pip elin e 触发器下添加触发器。 CodeConnections 要配置管道触发器配置并选择使用触发器进行筛选，请在中查看更多详细信息筛选代码推送或拉取请求的触发器。
3. 在输出构件格式中，您必须为构件选择格式。
   - 要使用默认方法存储 GitLab 自管理操作的输出对象，请选择CodePipeline默认。操作会访问存储库中的文件，并将构件以 ZIP 文件格式存储在管道构件存储中。
   - 要存储包含存储库的 URL 引用的 JSON 文件，以便下游操作可以直接执行 Git 命令，请选择完全克隆。此选项只能由 CodeBuild 下游操作使用。
4. 在向导上选择下一步，或者在编辑操作页面上选择保存。

创建主机并连接到 GitLab自我管理 (CLI)

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 AWS CloudFormation，使用控制台编辑连接以使其处于状态AVAILABLE。

您可以使用 AWS Command Line Interface (AWS CLI) 为已安装的连接创建主机。

您可以使用主机来表示安装第三方提供程序的基础设施的终端节点。使用 CLI 完成主机创建后，主机将处于待处理状态。然后，您应设置或注册主机，使之转为可用状态。主机可用后，完成创建连接的步骤。

为此，请使用 create-host 命令。

> **Important**
>
> 默认情况下，通过创建的主机 AWS CLI 处于Pending状态。使用 CLI 创建主机后，可使用控制台或 CLI 设置主机以使其状态为 Available。

要创建主机

1. 打开终端（Linux、macOS 或 Unix）或命令提示符（Windows）。 AWS CLI 使用运行create-host命令，--provider-endpoint为您的连接指定--name--provider-type、和。在此示例中，第三方提供程序名称为 GitLabSelfManaged，终端节点为 my-instance.dev。

   ```
   aws codestar-connections create-host --name MyHost --provider-type
    GitLabSelfManaged --provider-endpoint "https://my-instance.dev"
   ```

   如果成功，该命令将返回类似以下内容的主机 Amazon 资源名称（ARN）信息。

   ```
   {
       "HostArn": "arn:aws:codestar-connections:us-west-2:account_id:host/My-
   Host-28aef605"
   }
   ```

   完成此步骤后，主机处于 PENDING 状态。
2. 使用控制台完成主机设置并将主机变为 Available 状态。

创建与 GitLab 自我管理的连接

1. 打开终端（Linux、macOS 或 Unix）或命令提示符（Windows）。 AWS CLI 使用运行create-connection命令，`--connection-name`为您的连接指定`--host-arn`和。

```
aws codestar-connections create-connection --host-arn arn:aws:codestar-
connections:us-west-2:account_id:host/MyHost-234EXAMPLE --connection-name
 MyConnection
```

如果成功，该命令将返回类似以下内容的连接 ARN 信息。

```
{
    "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/
aEXAMPLE-8aad"
}
```

2. 使用控制台设置待处理的连接。
3. 管道默认会在向连接源存储库推送代码时检测更改。要配置手动发布或 Git 标签管道触发器配置，请执行以下操作之一：

   - 要将管道触发器配置为仅通过手动发布启动，请在配置中添加以下行：

```
"DetectChanges": "false",
```

   - 要将管道触发器配置配置配置为使用触发器进行筛选，请在中查看更多详细信息筛选代码推送或拉取请求的触发器。例如，以下内容添加到管道 JSON 定义的管道级别。在此示例中，`release-v0` 和 `release-v1` 是要包含的 Git 标签，`release-v2` 是要排除的 Git 标签。

```
"triggers": [
            {
                "providerType": "CodeStarSourceConnection",
                "gitConfiguration": {
                    "sourceActionName": "Source",
                    "push": [
                        {
                            "tags": {
                                "includes": [
                                    "release-v0", "release-v1"
                                ],
                                "excludes": [
                                    "release-v2"
```

```
                            ]
                        }
                    }
                ]
            }
        }
    ]
```

在中编辑管道 CodePipeline

管道描述了您 AWS CodePipeline 要遵循的发布流程，包括必须完成的阶段和操作。您可以编辑管道以添加或删除这些元素。不过，在编辑管道时，无法更改某些值，例如管道名称或管道元数据。

您可以使用管道编辑页面编辑管道类型、变量和触发器。还可以在管道中添加或更改阶段和操作。

与创建管道不同，编辑管道不会通过管道重新运行最新的修订。如果要通过刚刚编辑的管道运行最新修订，您必须手动重新运行它。否则，编辑的管道将在您下次对在源阶段中配置的源位置进行更改时运行。有关信息，请参阅 手动启动管道。

您可以向管道中添加与您的管道不同的 AWS 区域中的操作。如果 AWS 服务 是操作的提供者，并且此操作类型/提供者类型与您的管道位于不同的 AWS 区域，则这是跨区域操作。有关跨区域操作的更多信息，请参阅在中添加跨区域操作 CodePipeline。

CodePipeline 当源代码变更被推送时，使用变更检测方法启动您的管道。这些检测方法基于源类型：

- CodePipeline 使用 Amazon Ev CloudWatch ents 来检测您的 CodeCommit 源存储库或 Amazon S3 源存储桶中的更改。

> **Note**
>
> 您使用控制台时，将自动创建更改检测资源。在使用控制台创建或编辑管道时，将为您创建其他资源。如果您使用创建管道，则必须自己创建其他资源。 AWS CLI 有关创建或更新 CodeCommit 管道的更多信息，请参阅为 CodeCommit 源创建 EventBridge 规则 (CLI)。有关使用 CLI 创建或更新 Amazon S3 管道的更多信息，请参阅为 Amazon S3 来源 (CLI) 创建 EventBridge 规则。

主题

- 编辑管道（控制台）

- 编辑管道（AWS CLI）

编辑管道（控制台）

您可以使用 CodePipeline 控制台在管道中添加、编辑或移除阶段，也可以在阶段中添加、编辑或删除操作。

更新管道时， CodePipeline优雅地完成所有正在运行的操作，然后在已完成运行操作的阶段和管道执行中失败。更新管道后，您需要重新运行管道。有关运行管道的更多信息，请参阅手动启动管道。

编辑管道

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   将显示与您的 AWS 账户关联的所有管道的名称。
2. 在 Name 中，选择您要编辑的管道的名称。这将打开管道的详细视图，包括管道每个阶段中每个操作的状态。
3. 在管道详细信息页中，选择编辑。
4. 要编辑管道类型，请在编辑：管道属性卡上选择编辑。选择以下选项之一，然后选择完成。

   - V1 类型的管道具有 JSON 结构，其中包含标准的管道、阶段和操作级参数。
   - V2 类型的管道与 V1 类型结构相同，并支持其他参数，例如触发器和管道级变量。

   管道类型有不同的特点和价格。有关更多信息，请参阅 管道类型。
5. 要编辑管道变量，请在编辑：变量卡上选择编辑变量。添加或更改管道级别的变量，然后选择完成。

   有关管道级变量的更多信息，请参阅Variables。有关在管道执行时传递的管道级变量的教程，请参阅教程：使用管道级变量。

   > **Note**
   >
   > 虽然添加管道级变量是可选的，但如果使用管道级变量指定了管道而没有提供变量值，管道执行将失败。
6. 要编辑管道触发器，请在编辑：触发器卡上选择编辑触发器。添加或更改触发器，然后选择完成。

有关添加触发器的更多信息，请参阅创建与 Bitbucket Cloud GitHub （版本 2）、 GitHub 企业服务器、 GitLab .com 或 GitLab自我管理的连接的步骤，例如。GitHub 连接

7. 要在编辑页面上编辑阶段和操作，请执行以下操作之一：

   - 要编辑阶段，请选择编辑阶段。您可以添加与现有操作串行或并行运行的操作 ：

     您还可以为这些操作选择编辑图标，在此视图中编辑操作。要删除某个操作，请选择该操作上的删除图标。
   - 要编辑某个操作，请选择该操作的编辑图标，然后在编辑操作上更改这些值。标记为星号（*）的项目都是必填的。
     - 对于 CodeCommit 存储库名称和分支，将显示一条消息，显示要为此管道创建的 Amazon Ev CloudWatch ents 规则。如果您移除 CodeCommit 来源，则会显示一条消息，显示要删除的 Amazon Ev CloudWatch ents 规则。
     - 对于 Amazon S3 源存储桶，将显示一条消息，显示要为此管道创建的 Amazon Ev CloudWatch ents 规则和 AWS CloudTrail 跟踪。如果您移除 Amazon S3 来源，则会显示一条消息，显示要删除的 Amazon Ev CloudWatch ents 规则和 AWS CloudTrail 跟踪。如果其他管道正在使用该 AWS CloudTrail 跟踪，则不会移除该跟踪，并且会删除数据事件。
   - 要添加阶段，请在管道中您要添加阶段的时间点选择 + 添加阶段。为阶段提供一个名称，然后向该阶段至少添加一个操作。标记为星号（*）的项目都是必填的。
   - 要删除阶段，请选择该阶段上的删除图标。阶段及其所有操作都将被删除。
   - 要将阶段配置为在失败时自动回滚，请选择 “编辑阶段”，然后选中 “配置舞台失败时自动回滚” 复选框。

   例如，如果您想要向管道中的阶段添加一系列操作：

   1. 在您要添加操作的阶段中，选择编辑阶段，然后选择 + 添加操作组。
   2. 在编辑操作的操作名称中，键入操作的名称。操作提供程序按类别显示提供程序选项。查找类别（例如部署）。在类别下，选择提供方（例如 AWS CodeDeploy）。在区域中，选择您已在其中创建或计划在其中创建资源的 AWS 区域。Re g ion 字段指定为此操作类型和提供者类型创建 AWS 资源的位置。此字段仅在操作提供方是 AWS 服务的情况下对操作显示。“区域” 字段默认为与您的管道相同的 AWS 区域。

      有关中操作要求的更多信息 CodePipeline，包括输入和输出工件的名称及其使用方式，请参阅中的操作结构要求 CodePipeline。有关添加操作提供程序和使用每个提供程序的默认字段的示例，请参阅创建管道（控制台）。

要 CodeBuild 作为生成操作或测试操作添加到阶段，请参阅《CodeBuild 用户指南》中的 CodePipeline 与一起使用 CodeBuild 来测试代码和运行构建。

> **Note**
>
> 某些操作提供者（例如 GitHub）要求您先连接到提供者的网站，然后才能完成操作的配置。当您连接到提供商的网站时，请确保您使用该网站的凭证。请勿使用您的 AWS 凭证。

3. 配置完操作后，请选择保存。

> **Note**
>
> 您无法在控制台视图中重命名阶段。您可以添加具有您要更改的名称的阶段，然后删除旧阶段。确保您已添加在该阶段需要的所有操作，然后再删除旧操作。

8. 编辑完您的管道后，请选择保存以返回到摘要页面。

> **Important**
>
> 当您保存自己的更改后，便无法撤消这些更改。您必须重新编辑管道。如果当您保存更改时，修订正在通过您的管道运行，则运行将无法完成。如果您希望特定提交或更改通过编辑后的管道运行，则必须手动通过管道运行它。否则，下一个提交或更改将通过管道自动运行。

9. 要测试您的操作，请选择发布更改来处理通过管道的提交并将更改提交至管道的源阶段中指定的源。或者按照中的手动启动管道步骤 AWS CLI 使用手动发布更改。

编辑管道（AWS CLI）

您可以使用 update-pipeline 命令编辑管道。

更新管道时， CodePipeline 优雅地完成所有正在运行的操作，然后在已完成运行操作的阶段和管道执行中失败。更新管道后，您需要重新运行管道。有关运行管道的更多信息，请参阅手动启动管道。

> **Important**
>
> 尽管您可以使用 AWS CLI 来编辑包含合作伙伴操作的管道，但您不得手动编辑合作伙伴操作的 JSON。如果这样做，在更新管道后，合作伙伴操作将失败。

编辑管道

1. 打开终端会话（Linux、macOS 或 Unix）或命令提示符 (Windows)，然后运行 get-pipeline 命令以将管道结构复制到 JSON 文件中。例如，对于名为 **MyFirstPipeline** 的管道，输入以下命令：

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   该命令不会返回任何结果，但您创建的文件将出现在您运行命令所在的目录中。

2. 在任何纯文本编辑器中打开 JSON 文件，修改文件结构以反映您要对管道所进行的更改。例如，您可以添加或删除阶段，或者在现有阶段中添加另一个操作。

   以下示例介绍如何在 pipeline.json 文件中添加另一个部署阶段。此阶段会在名为 *Staging* 的第一个部署阶段之后运行。

   > **Note**
   >
   > 这只是文件的一部分，而不是整个结构。有关更多信息，请参阅 CodePipeline 管道结构参考。

   ```
   ,
           {
               "name": "Staging",
               "actions": [
                       {
                           "inputArtifacts": [
                               {
                                   "name": "MyApp"
                               }
                           ],
                           "name": "Deploy-CodeDeploy-Application",
                           "actionTypeId": {
                               "category": "Deploy",
   ```

```
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeDeploy"
                        },
                        "outputArtifacts": [],
                        "configuration": {
                            "ApplicationName": "CodePipelineDemoApplication",
                            "DeploymentGroupName": "CodePipelineDemoFleet"
                        },
                        "runOrder": 1
                    }
                ]
            },
        {
            "name": "Production",
            "actions":  [
                    {
                        "inputArtifacts": [
                            {
                                "name": "MyApp"
                            }
                        ],
                        "name": "Deploy-Second-Deployment",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeDeploy"
                        },
                        "outputArtifacts": [],
                        "configuration": {
                        "ApplicationName": "CodePipelineDemoApplication",
                        "DeploymentGroupName": "CodePipelineProductionFleet"
                        },
                        "runOrder": 1
                    }
                ]
        }
    ]
}
```

有关使用 CLI 在管道中添加审批操作的信息，请参阅[向中的管道添加手动批准操作 CodePipeline](#)。

确保按如下所示设置 JSON 文件中的 PollForSourceChanges 参数：

```
                "PollForSourceChanges": "false",
```

CodePipeline 使用 Amazon Ev CloudWatch ents 来检测您的 CodeCommit 源存储库和分支或 Amazon S3 源存储桶中的更改。下一步包括手动创建这些资源的说明。将此标记设置为 false 将禁用定期检查，当使用建议的更改检测方法时不需要定期检查。

3. 要在您的管道所在区域之外的区域添加构建、测试或部署操作，您必须向管道结构添加以下内容。有关详细说明，请参阅 在中添加跨区域操作 CodePipeline。

   - 将 Region 参数添加到您的操作的管道结构。
   - 使用 artifactStores 参数为包含操作的每个区域指定一个构件桶。

4. 如果您正在使用通过 get-pipeline 命令检索到的管道结构，则必须在 JSON 文件中修改结构。您必须从文件中删除 metadata 行以便 update-pipeline 命令可以使用它。从 JSON 文件中的管道结构中删除该部分（"metadata": { } 行以及 "created"、"pipelineARN" 和 "updated" 字段）。

   例如，从结构中删除以下各行：

   ```
   "metadata": {
     "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
     "created": "date",
     "updated": "date"
     }
   ```

   保存该文件。

5. 如果您使用 CLI 编辑管道，您必须手动为您的管道管理建议的更改检测资源：

   - 对于 CodeCommit 存储库，您必须创建 CloudWatch 事件规则，如中所述为 CodeCommit 源创建 EventBridge 规则 (CLI)。
   - 对于 Amazon S3 来源，您必须创建 CloudWatch 事件规则和 AWS CloudTrail 跟踪，如中所述亚马逊 S3 源代码操作 EventBridge 以及 AWS CloudTrail。

6. 要应用更改，请运行 update-pipeline 命令，指定管道 JSON 文件：

> **Important**
>
> 务必在文件名前包含 `file://`。此命令中需要该项。

```
aws codepipeline update-pipeline --cli-input-json file://pipeline.json
```

该命令会返回编辑后的管道的整个结构。

> **Note**
>
> update-pipeline 命令会停止管道。如果在运行 update-pipeline 命令时正在通过管道运行修订，则该运行会被停止。您必须手动启动管道，通过更新后的管道运行此修订。

7. 打开 CodePipeline 控制台并选择您刚才编辑的管道。

   管道将显示您的更改。当您下一次更改源位置时，管道会通过管道修订后的结构运行该修订。

8. 要通过管道修订后的结构手动运行最新修订，请运行 start-pipeline-execution 命令。有关更多信息，请参阅 手动启动管道。

有关管道结构和预期值的更多信息，请参阅CodePipeline 管道结构参考和 AWS CodePipeline API 参考。

在中查看管道和详细信息 CodePipeline

您可以使用 AWS CodePipeline 控制台或查看 AWS CLI 与您的 AWS 账户关联的管道的详细信息。

主题

- 查看管道（控制台）
- 在管道中查看操作详细信息（控制台）
- 查看管道 ARN 和服务角色 ARN（控制台）
- 查看管道详细信息和历史记录（CLI）

查看管道（控制台）

您可以查看管道的状态、过渡和构件更新。

> **Note**
>
> 一小时后，管道的详细视图将在浏览器中自动停止刷新。要查看当前信息，请刷新页面。

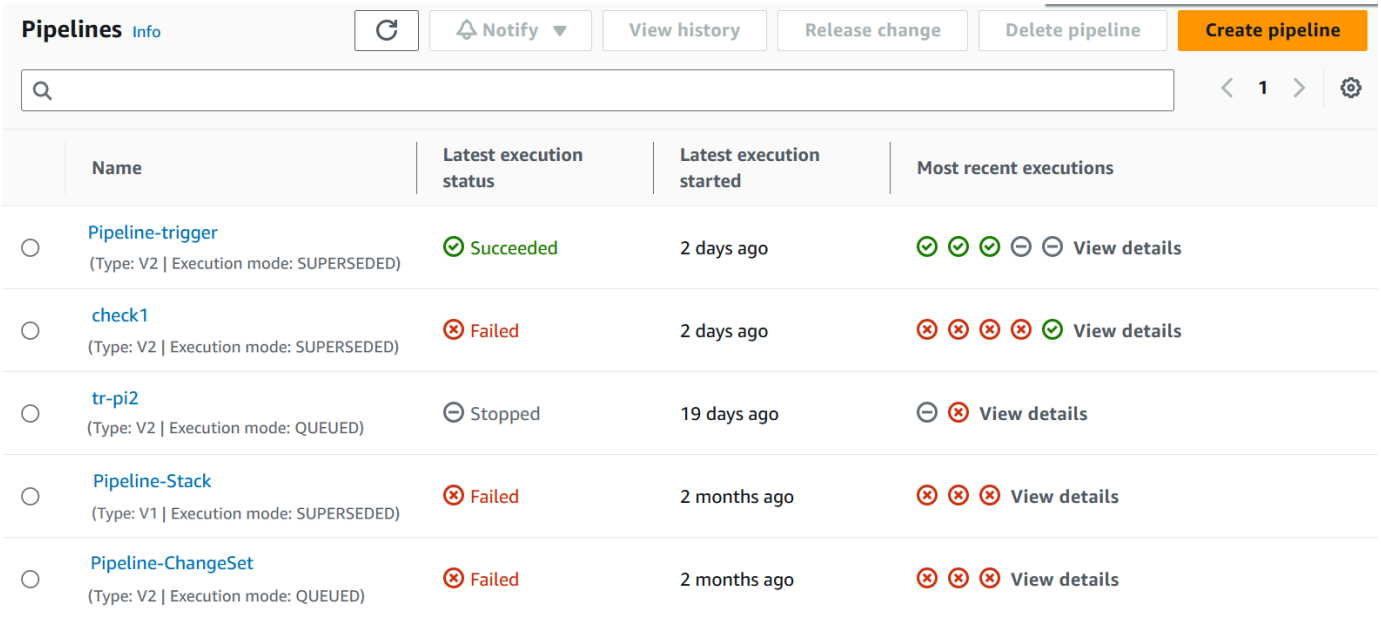
查看管道

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

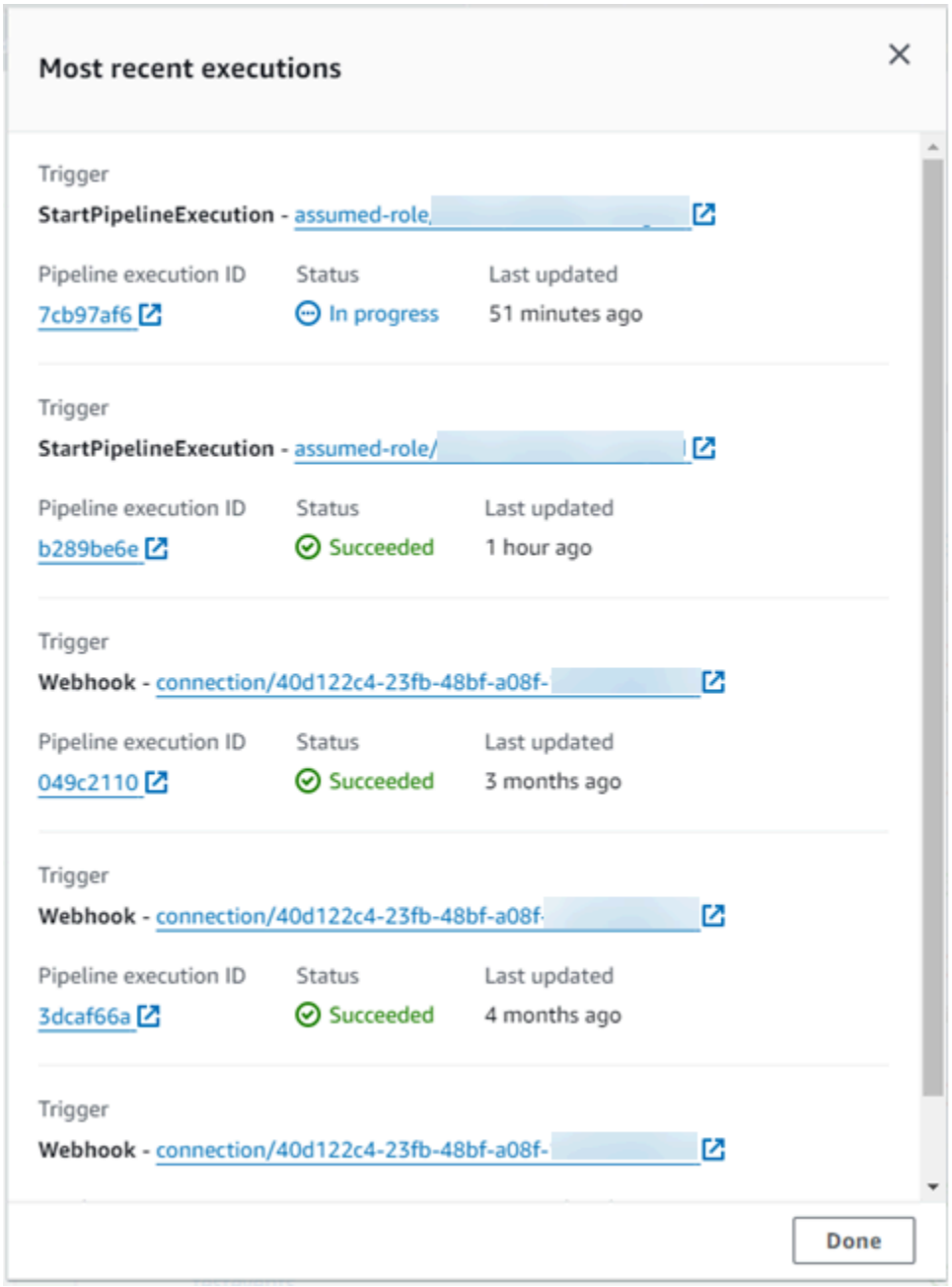
   将显示管道页面。其中显示了您在该区域内所有管道的列表。

   将显示与您的 AWS 账户关联的所有管道的名称、类型、状态、版本、创建日期和上次修改日期，以及最近开始执行的时间。

2. 还显示了最近五次执行的状态。



选择特定行旁边的查看详细信息可打开一个详细信息对话框，其中列出了最近的执行。

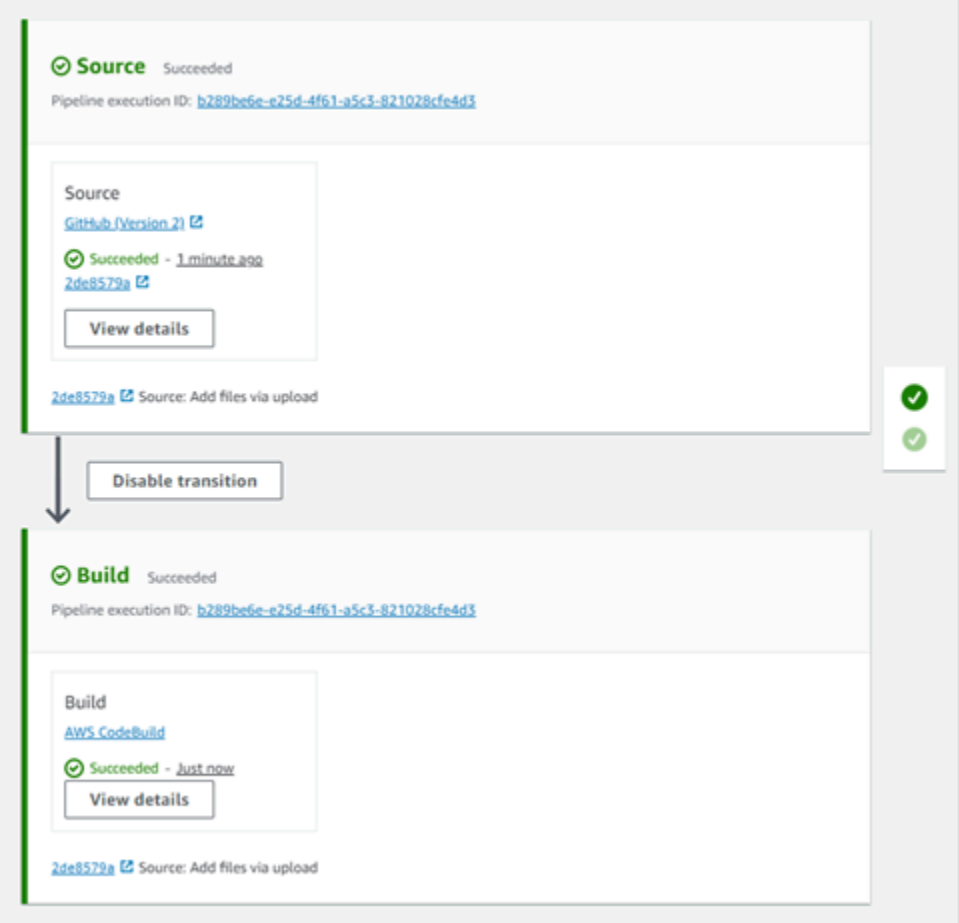


要查看有关管道的最新执行详细信息，请选择 View history (查看历史记录)。对于过去的执行，您可以查看与源项目相关联的修订详细信息，如执行 ID、状态、开始时间和结束时间、持续时间以及提交 ID 和消息。

> **Note**
>
> 对于处于并行执行模式的管道，主管道视图不显示管道结构或正在进行的执行。对于处于并行执行模式的管道，您可以通过从执行历史记录页面选择要查看的执行的 ID 来访问管

道结构。在左侧导航栏中选择 “历史记录”，选择并行执行的执行 ID，然后在 “可视化” 选项卡上查看管道。

3. 要查看单个管道的详细信息，在（名称）中，选择该管道。此时将显示管道的详细视图，包括每个阶段中每个操作的状态和过渡的状态。



图形视图会显示每个阶段的以下信息：

- 阶段名称。
- 为阶段配置的各个操作。
- 各阶段之间的过渡状态（已启用或已禁用），由各阶段之间的箭头的状态表示。已启用的过渡由箭头及其旁边的禁用过渡按钮指示。已禁用的过渡由其下带删除线的箭头及其旁边的启用过渡按钮指示。
- 表示阶段状态的色条：
  - 灰色：还没有任何执行
  - 蓝色：正在进行

- 绿色：已成功
- 红色：已失败

图形视图还会显示每个阶段中各个操作的以下信息：

- 操作的名称。
- 操作的提供者，例如 CodeDeploy。
- 上一次运行操作的时间。
- 操作是成功了还是失败了。
- 指向关于操作上一次运行的其他详细信息的链接（如果有）。
- 有关通过该阶段最新管道执行运行的源版本的详细信息，或者对于 CodeDeploy 部署，则是部署到目标实例的最新源版本的详细信息。
- 查看详细信息按钮，用于打开一个对话框，显示有关操作执行、日志和操作配置的详细信息。

> **Note**
>
> “日志” 选项卡适用于 CodeBuild 已在管道账户中运行的 AWS CloudFormation 操作。

4. 要查看操作提供程序的详细信息，请选择提供程序。例如，在前面的示例管道 CodeDeploy 中，如果您选择暂存阶段或生产阶段，则会显示为该阶段配置的部署组的 CodeDeploy 控制台页面。
5. 要查看某个操作的进度，可选择正在进行的操作旁边显示的选项（由正在进行消息指示）。如果操作正在进行，您将看到递增进度以及正在执行的步骤或操作。
6. 要批准或拒绝已被配置为手动审批的操作，请选择 Review。
7. 要重试阶段中未成功完成的操作，请选择 Retry。
8. 此时将显示操作最后一次运行的状态，包括操作的结果：成功或失败。

在管道中查看操作详细信息（控制台）

您可以查看管道的详细信息，包括每个阶段操作的详细信息。

> **Note**
>
> 一小时后，管道的详细视图将在浏览器中自动停止刷新。要查看当前信息，请刷新页面。

在管道中查看操作详细信息

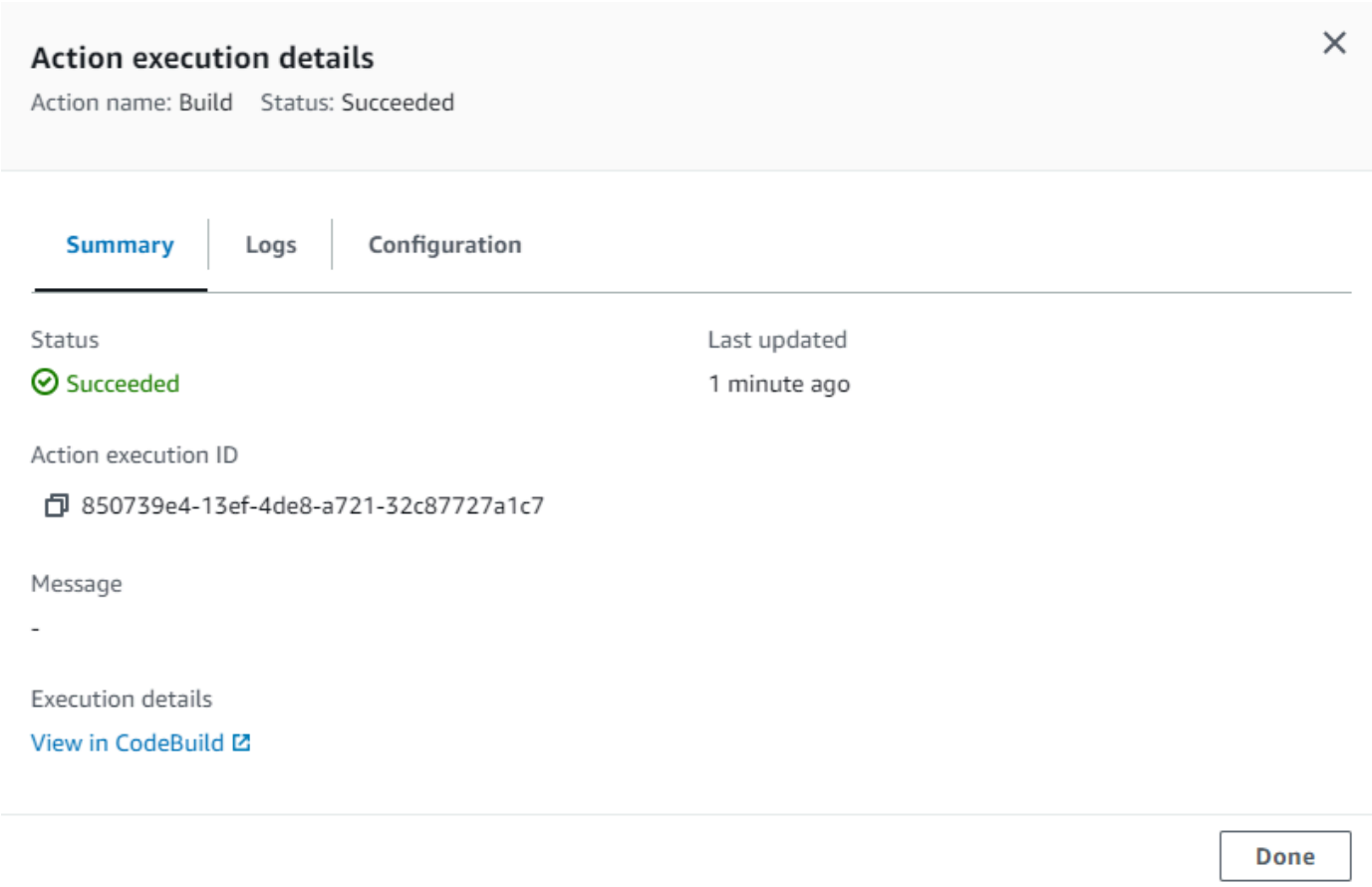
1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   将显示管道页面。

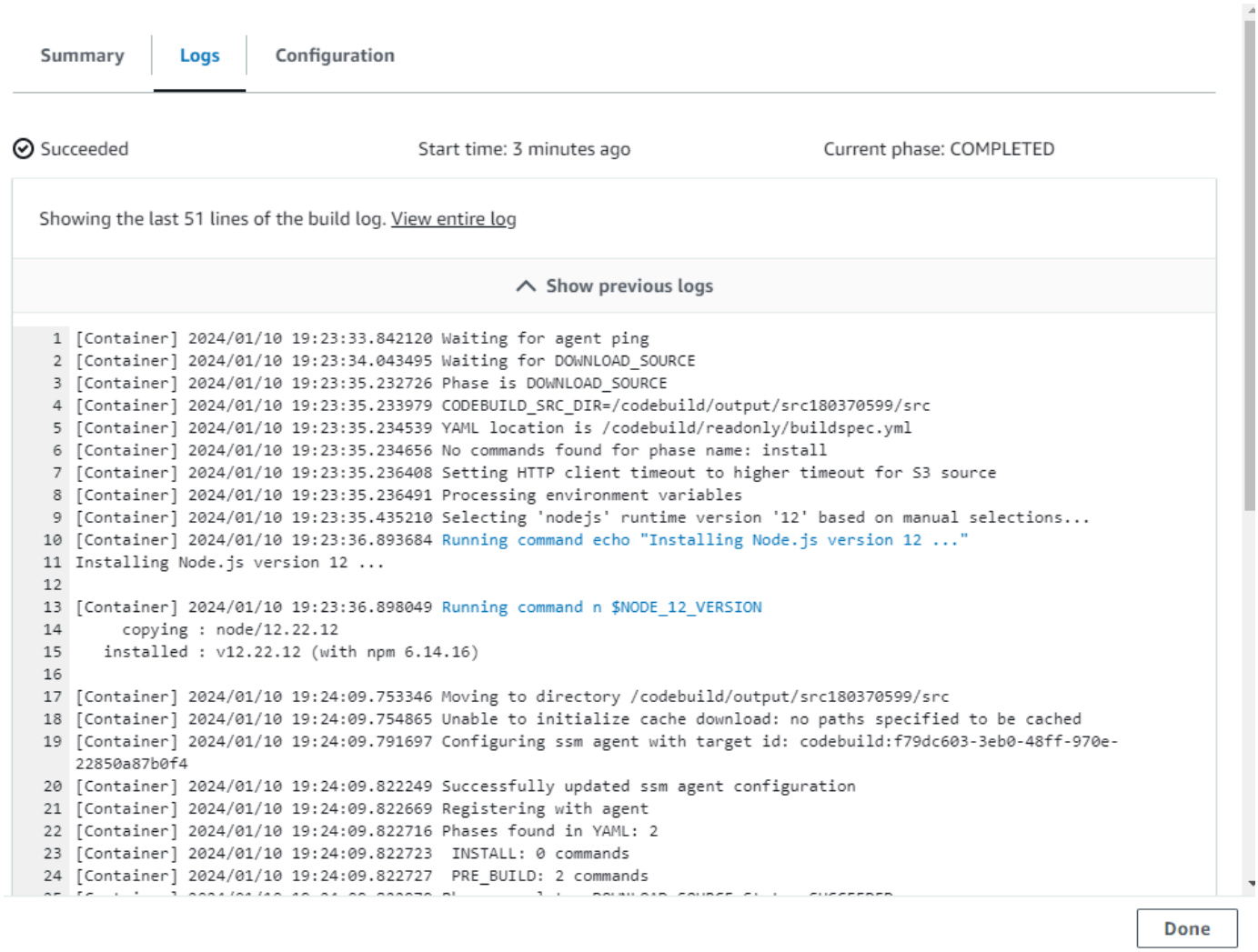
2. 在任何操作上，选择查看详细信息以打开一个对话框，其中包含有关操作执行、日志和操作配置的详细信息。

   > **Note**
   >
   > “日志” 选项卡可用于 CodeBuild 和 AWS CloudFormation 操作。

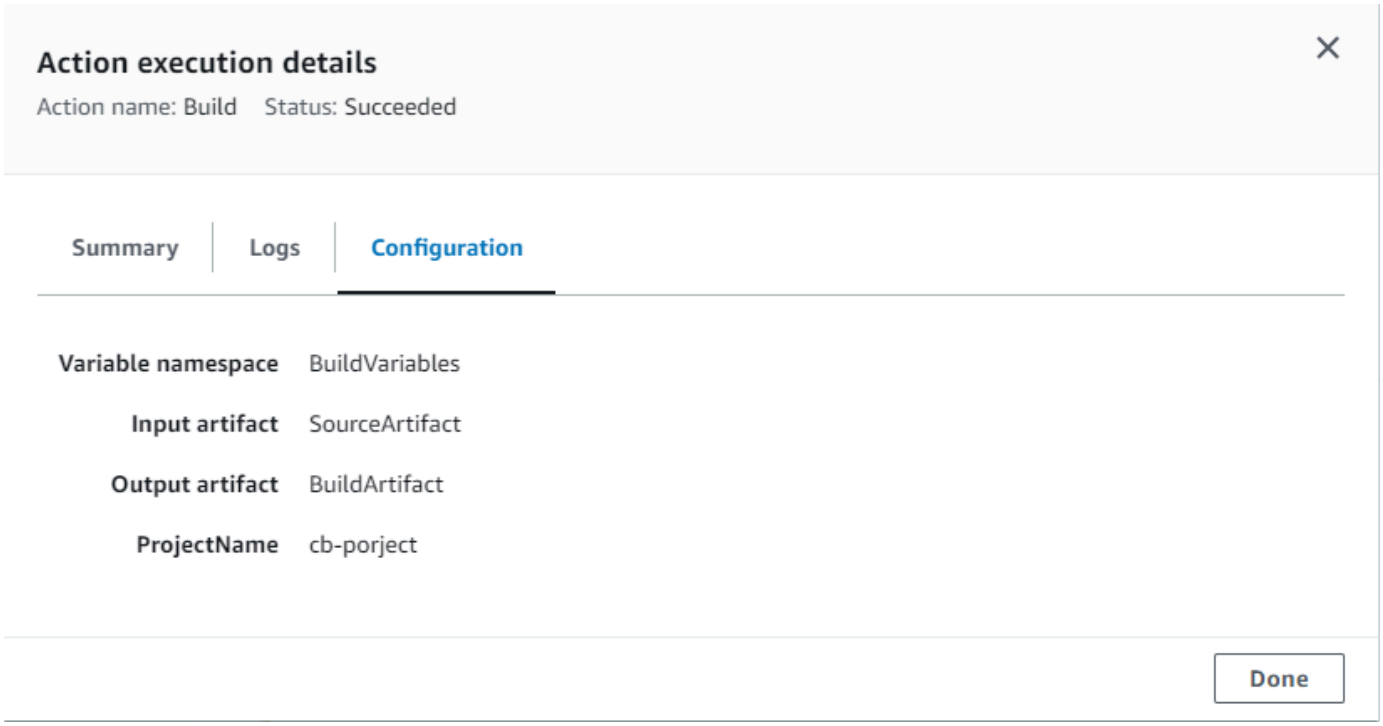
3. 要查看管道某阶段内某项操作的操作摘要，请选择该操作的查看详细信息，然后选择摘要选项卡。



4. 要查看带有日志的操作的操作日志，请在该操作上选择查看详细信息，然后选择日志选项卡。



5. 要查看某项操作的配置详细信息，请选择配置选项卡。



查看管道 ARN 和服务角色 ARN（控制台）

您可以使用控制台查看管道设置，例如管道 ARN、服务角色 ARN 和管道构件存储。

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   将显示与您的 AWS 账户关联的所有管道的名称。
2. 选择您的管道名称，然后在左侧导航窗格中选择设置。页面中显示以下内容：

   - 管道名称
   - 管道的 Amazon 资源名称 (ARN)。

     管道 ARN 是使用以下格式构造的：

     arn:aws:codepipeline:*region*:*account*:*pipeline-name*

     示例管道 ARN：

     ```
     arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline
     ```
   - 您的管道的 CodePipeline 服务角色 ARN

- 管道版本
- 管道构件存储的名称和位置

查看管道详细信息和历史记录（CLI）

您可以运行以下命令，以查看您的管道和管道执行的详细信息：

- list-pipelines命令查看与您的 AWS 账户关联的所有管道的摘要。
- get-pipeline 命令，用于查看单个管道的详细信息。
- list-pipeline-executions，用于查看管道的最近执行的摘要。
- get-pipeline-execution 用于查看有关管道执行的信息，包括有关构件的详细信息、管道执行 ID 以及管道的名称、版本和状态。
- get-pipeline-state 命令用于查看管道、阶段和操作状态。
- list-action-executions 用于查看管道的操作执行详细信息。

1. 打开终端（Linux、macOS 或 Unix）或命令提示符 (Windows)， AWS CLI 然后使用运行命令：list-pipelines

   ```
   aws codepipeline list-pipelines
   ```

   该命令会返回一个与您的 AWS 账户相关联的所有管道的列表。
2. 要查看管道的详细信息，请运行 get-pipeline 命令，指定管道的唯一名称。例如，要查看名为的管道的详细信息 *MyFirstPipeline*，请输入以下内容：

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline
   ```

   该命令会返回管道结构。

删除中的管道 CodePipeline

您可以随时编辑管道以更改其功能，但您可能决定将其删除。您可以使用 AWS CodePipeline 控制台或中的delete-pipeline命令 AWS CLI 来删除管道。

主题

删除管道（控制台）

删除管道

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   将显示与您的 AWS 账户关联的所有管道的名称和状态。
2. 在 Name 中，选择您要删除的管道的名称。
3. 在管道详细信息页中，选择编辑。
4. 在 Edit 页面上，选择 Delete。
5. 在字段中键入 **delete** 以确认，然后选择 Delete (删除)。

> **Important**
>
> 并且无法撤消。

删除管道 (CLI)

要使用手动删除管道 AWS CLI ，请使用 delet e-pipelin e 命令。

> **Important**
>
> 删除管道操作是不可逆的。没有确认对话框。命令运行后，管道将被删除，但管道中使用的任何资源都不会被删除。这样就可以更轻松地创建使用这些资源来自动发布软件的新管道。

删除管道

1. 打开终端（Linux、macOS 或 Unix）或命令提示符 (Windows)，然后使用运行delete-pipeline命令，指定要删除的管道的名称。 AWS CLI 例如，要删除名为以下的管道 *MyFirstPipeline*：

```
aws codepipeline delete-pipeline --name MyFirstPipeline
```

该命令不返回任何内容。

2. 删除任何不再需要的资源。

> **Note**
>
> 删除管道不会删除管道中使用的资源，例如您用于部署代码的 CodeDeploy 或 Elastic Beanstalk 应用程序，或者，如果您是通过控制台创建管道的，则会 CodePipeline 删除为存储管道项目而创建的 A CodePipeline mazon S3 存储桶。确保删除您不再需要的资源，这样以后您就不用再为它们付费。例如，当您首次使用控制台创建管道时， CodePipeline 会创建一个 Amazon S3 存储桶来存储所有管道的所有项目。如果您已经删除了您的所有管道，请按照删除存储桶中的步骤操作。

在中 CodePipeline 创建使用其他 AWS 账户资源的管道

您可能希望创建一个使用由另一个 AWS 账户创建或管理的资源的管道。例如，您可能想将一个账户用于管道，将另一个账户用于您的 CodeDeploy 资源。

> **Note**
>
> 当您创建一个包含多个账户的操作的管道时，必须配置您的操作，以使它们仍可以在跨账户管道的限制范围内访问构件。跨账户操作适用以下限制：
>
> - 通常，只有在以下情况下，操作才能使用构件：
> - 操作所属账户与管道账户相同，或者
> - 构件是在管道账户中为另一个账户中的操作创建的，或者
> - 构件是由操作所属账户中的先前操作生成的
>
> 换句话说，如果两个账户都不是管道账户，则无法将构件从一个账户传递到另一个账户。
> - 以下操作类型不支持跨账户操作：
> - Jenkins 构建操作

在此示例中，您必须创建要使用的 AWS Key Management Service (AWS KMS) 密钥，将密钥添加到管道中，并设置账户策略和角色以启用跨账户访问。对于 AWS KMS 密钥，您可以使用密钥 ID、密钥 ARN 或别名 ARN。

> **Note**
>
> 别名只能在创建 KMS 密钥的账户中识别。对于跨账户操作，您只能使用密钥 ID 或密钥 ARN 来标识密钥。跨账户操作涉及使用其他账户（AccountB）的角色，因此指定密钥 ID 将使用其他账户（AccountB）的密钥。

在本演练及其示例中，*AccountA* 是最初用于创建管道的账户。它可以访问用于存储管道项目的 Amazon S3 存储桶和使用的服务角色 AWS CodePipeline。*AccountB* 是最初用于创建所使用的 CodeDeploy 应用程序、部署组和服务角色的账户。 CodeDeploy

Ac countA ######## Account B ### CodeDeploy #####AccountA

- 请求 *AccountB* 的 ARN 或账户 ID（在本演练中，*AccountB* ID 为 *012ID_ACCOUNT_B*）。
- *############ AWS KMS ############ (CodePipeline_Service_R ole) # AccountB ###########*
- 创建亚马逊 S3 存储桶策略，授予 Ac *countB* 访问亚马逊 S3 存储桶的权限（例如，*codepipeline-us-east-2-* 1234567890）。
- *###### A #### B ##################### (_Service_Role)#CodePipeline*
- 编辑管道以使用客户托管 AWS KMS 密钥而不是默认密钥。

要使 *AccountB* 允许在 *AccountA* 中创建的管道访问其资源，*AccountB* 必须：

- 请求 *AccountA* 的 ARN 或账户 ID（在本演练中，*AccountA* ID 为 *012ID_ACCOUNT_A*）。
- 创建应用于为其配置的 Amazon EC2 实例角色的策略 CodeDeploy ，该策略允许访问 Amazon S3 存储桶 (*codepipeline-us-east-2-123* 4567890)。
- 创建应用于为其配置的 Amazon EC2 实例角色的策略 CodeDeploy ，该策略允许访问用于在 Ac *coun* tA 中加密管道项目的 AWS KMS 客户托管密钥。
- 配置并附加一个 IAM 角色 (*CrossAccount_Rol* e)，其信任关系策略允许 Accoun *tA* 中的 CodePipeline 服务角色代入该角色。
- 创建允许访问管道所需的部署资源的策略并将其附加到 *CrossAccount_Role#*
- *###### Amazon S3 ### (codepipeline-us-east-2-1234567890) ######### _Role#CrossAccount*

主题

先决条件：创建 AWS KMS 加密密钥

客户管理的密钥和所有 AWS KMS 密钥都特定于一个区域。您必须在创建管道的同一区域（例如us-east-2）创建客户托管 AWS KMS 密钥。

要在中创建客户管理的密钥 AWS KMS

1. 使用 Ac *coun AWS Management Console tA* 登录并打开控制台。 AWS KMS
2. 在左侧，选择 客户管理的密钥。
3. 选择 Create key。在配置密钥中，保留默认选中的对称，然后选择下一步。
4. 在别名中，输入用于此密钥的别名（例如，*PipelineName-Key*）。（可选）提供有关该密钥的描述和标签，然后选择下一步。
5. 在定义密钥管理权限中，选择您希望作为该密钥管理员的一个或多个角色，然后选择下一步。
6. 在 “定义密钥使用权限” 中的 “此帐户” 下，选择管道的服务角色名称（例如 CodePipeline _Service_Role）。在 “其他 AWS 账户” 下，选择 “添加其他 AWS 账户”。输入 *AccountB* 的账户 ID 以完成 ARN，然后选择下一步。
7. 在审核和编辑密钥策略中审核策略，然后选择完成。
8. 从密钥列表中选择密钥的别名并复制其 ARN（例如 ***arn:aws:kms:us-east-2:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE***）。在您编辑您的管道和配置策略时将会需要此密钥。

步骤 1：设置账户策略和角色

创建 AWS KMS 密钥后，您必须创建并附加将启用跨账户访问权限的策略。这需要来自 *AccountA* 和 *AccountB* 的操作。

主题

在将要创建管道的账户中配置策略和角色 (*AccountA*)

要创建使用与其他 AWS 账户关联的 CodeDeploy 资源的管道，Ac *countA* 必须为用于存储项目的 Amazon S3 存储桶和其服务角色配置策略。 CodePipeline

创建授予 AccountB 访问权限的 Amazon S3 桶策略（控制台）

1. 使用 Ac *coun AWS Management Console tA* 登录并打开亚马逊 S3 控制台，网址为 https://console.aws.amazon.com/s3/。
2. 在 Amazon S3 桶列表中，选择用于存储您的管道构件的 Amazon S3 桶。*##### #codepipeline-region-1234567EXAMPLE########### AWS ###1234567EXAMPLE ########################-2-1234567890##codepipeline-us-east*
3. 在 Amazon S3 桶的详细信息页面上，选择属性。
4. 在属性窗格中，展开 Permissions，然后选择 Add bucket policy。

   > **Note**
   >
   > 如果一个策略已附加到您的 Amazon S3 桶，请选择编辑桶策略。然后，您可以将以下示例中的语句添加到现有策略中。要添加新策略，请选择链接，然后按照 AWS 策略生成器中的说明进行操作。有关更多信息，请参阅 IAM 策略概述。

5. 在桶策略编辑器窗口中，键入以下策略。这将允许 *AccountB* 访问管道项目，并且如果某个操作 (比如自定义源或生成操作) 创建了这些管道项目，则将为 *AccountB* 提供添加输出项目的功能。

   在该示例中，*AccountB* 的 ARN 为 *012ID_ACCOUNT_B*。*### S3 #### ARN # -2-1234567890 codepipeline-us-east#*将这些 ARN 分别替换为要允许访问的账户的 ARN 和 Amazon S3 桶的 ARN：

```
{
 "Version": "2012-10-17",
 "Id": "SSEAndSSLPolicy",
 "Statement": [
{
  "Sid": "DenyUnEncryptedObjectUploads",
  "Effect": "Deny",
  "Principal": "*",
  "Action": "s3:PutObject",
  "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
  "Condition": {
 "StringNotEquals": {
```

```
  "s3:x-amz-server-side-encryption": "aws:kms"
  }
    }
  },
 {
   "Sid": "DenyInsecureConnections",
   "Effect": "Deny",
   "Principal": "*",
   "Action": "s3:*",
   "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
   "Condition": {
  "Bool": {
     "aws:SecureTransport": false
   }
  }
   },
 {
   "Sid": "",
   "Effect": "Allow",
   "Principal": {
  "AWS": "arn:aws:iam::012ID_ACCOUNT_B:root"
  },
   "Action": [
            "s3:Get*",
            "s3:Put*"
        ],
   "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*"
 },
 {
   "Sid": "",
   "Effect": "Allow",
   "Principal": {
       "AWS": "arn:aws:iam::012ID_ACCOUNT_B:root"
   },
   "Action": "s3:ListBucket",
   "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890"
 }
   ]
}
```

6. 选择 Save，然后关闭策略编辑器。
7. 选择保存以保存 Amazon S3 桶的权限。

为的服务角色创建策略 CodePipeline （控制台）

1. 使用 Ac *coun AWS Management Console tA* 登录并打开 IAM 控制台，网址为 https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择角色。
3. 在角色列表中的角色名称下，选择其服务角色的名称 CodePipeline。
4. 在权限选项卡上，选择添加内联策略。
5. 选择 JSON 选项卡，然后输入以下策略以允许 *AccountB* 代入该角色。在以下示例中，*012ID_ACCOUNT_B* 是 *AccountB* 的 ARN：

```
{
   "Version": "2012-10-17",
   "Statement": {
       "Effect": "Allow",
       "Action": "sts:AssumeRole",
       "Resource": [
           "arn:aws:iam::012ID_ACCOUNT_B:role/*"
       ]
   }
 }
```

6. 选择查看策略。
7. 在 Name (名称) 中，输入此策略的名称。选择 创建策略。

在拥有 AWS 资源的账户 (Ac *co* untB) 中配置策略和角色

当您在中创建应用程序、部署和部署组时 CodeDeploy，还会创建 Amazon EC2 实例角色。（如果您使用“运行部署演练”向导，系统将为您创建该角色，但您也可以手动创建该角色。）要使在 Ac *countA* 中创建的管道使用在 AccountB 中创建的 CodeDeploy 资源，您#须：

- 为实例角色配置策略，允许它访问存储管道构件的 Amazon S3 桶。
- 在配置为跨账户存取的 *AccountB* 中创建第二个角色。

  第二个角色不仅必须有权访问 Ac *countA* 中的 Amazon S3 存储桶，还必须包含允许访问 CodeDeploy 资源的策略和允许## *A* 中的 CodePipeline 服务角色代入该角色的信任关系策略。

> **Note**
>
> 这些策略专门用于设置要在使用其他 AWS 账户创建的管道中使用的 CodeDeploy 资源。其他 AWS 资源将需要针对其资源需求的具体政策。

为为 CodeDeploy （控制台）配置的 Amazon EC2 实例角色创建策略

1. 使用 A *ccoun AWS Management Console tB* 登录并打开 IAM 控制台，网址为 https://console.aws.amazon.com/iam/。
2. 在导航窗格中，选择角色。
3. 在角色列表中的角色名称下，选择用作 CodeDeploy 应用程序的 Amazon EC2 实例角色的服务角色的名称。该角色名称可能不同，而且多个实例角色可以由一个部署组使用。有关更多信息，请参阅为 Amazon EC2 实例创建 IAM 实例配置文件。
4. 在权限选项卡上，选择添加内联策略。
5. 选择 JSON 选项卡，然后输入以下策略以授予对 Ac *countA* 用来存储管道工件的 Amazon S3 存储桶的访问权限（在本示例中为 *codepipeline-us-east-2-* 1234567890）：

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "s3:Get*"
        ],
        "Resource": [
          "arn:aws:s3:::codepipeline-us-east-2-1234567890/*"
        ]
      },
      {
        "Effect": "Allow",
        "Action": [
          "s3:ListBucket"
        ],
        "Resource": [
          "arn:aws:s3:::codepipeline-us-east-2-1234567890"
        ]
      }
```

```
    ]
 }
```

6. 选择查看策略。
7. 在 Name (名称) 中，输入此策略的名称。选择 创建策略。
8. *########### Accoun* ***arn:aws:kms:us-east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE*** *tA ###### ### AccountB ########## ARN # AWS KMS ###*

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
           "kms:DescribeKey",
           "kms:GenerateDataKey*",
           "kms:Encrypt",
           "kms:ReEncrypt*",
           "kms:Decrypt"
         ],
        "Resource": [
           "arn:aws:kms:us-
east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE"
          ]
      }
   ]
}
```

> **Important**
>
> 您必须使用本政策中账户 *A* 的账户 ID 作为 AWS KMS 密钥资源 ARN 的一部分，如图所示，否则该策略将失效。

9. 选择查看策略。
10. 在 Name (名称) 中，输入此策略的名称。选择 创建策略。

现在，创建用于跨账户访问的 IAM 角色，并对其进行配置，以便 Accoun *tA* 中的 CodePipeline 服务角色可以代入该角色。此角色必须包含允许访问 CodeDeploy 资源和用于在 Ac *coun* tA 中存储项目的 Amazon S3 存储桶的策略。

在 IAM 中配置跨账户角色

1. 使用 A *ccoun AWS Management Console tB* 登录并打开 IAM 控制台，网址为 https://console.aws.amazon.com/iam。
2. 在导航窗格中，选择角色。选择 创建角色。
3. 在选择受信任实体的类型下，选择其他 AWS 账户。在 “指定可以使用此角色的账户 ID” 下，输入要在中创建管道的账户的账户 ID CodePipeline（ac *countA* ），然后选择下一步：权限。 AWS

   > **Important**
   >
   > 此步骤将在 *AccountB* 与 *AccountA* 之间创建信任关系策略。但是，这会授予对账户的根级访问权限，并 CodePipeline 建议将其范围缩小到Accoun *t* A中的 CodePipeline 服务角色。按照步骤 16 限制权限。

4. 在 “附加权限策略” 下，选择 AmazonS3 ReadOnlyAccess，然后选择 “下一步：标签”。

   > **Note**
   >
   > 这并不是您要使用的策略。您必须选择一个策略来完成向导。

5. 选择 下一步: 审核。在角色名称中键入此角色的名称（例如，*CrossAccount_R* ole）。您可以任意命名该角色，只要其遵循 IAM 中的命名约定即可。考虑为该角色使用一个明确指明其用途的名称。请选择 创建角色。
6. 从角色列表中选择您刚刚创建的角色（例如 *CrossAccount_Role########*的 “摘要” 页面。
7. 在权限选项卡上，选择添加内联策略。
8. 选择 JSON 选项卡，然后输入以下策略以允许访问 CodeDeploy 资源：

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

```
        "codedeploy:GetDeployment",
        "codedeploy:GetDeploymentConfig",
        "codedeploy:GetApplicationRevision",
        "codedeploy:RegisterApplicationRevision"
      ],
      "Resource": "*"
    }
  ]
}
```

9. 选择查看策略。
10. 在 Name (名称) 中，输入此策略的名称。选择 创建策略。
11. 在权限选项卡上，选择添加内联策略。
12. 在策略文档中，键入以下策略以允许该角色从 *AccountA* 的 Amazon S3 桶中检索输入构件，或者将输出构件放入该桶中：

```
{
   "Version": "2012-10-17",
   "Statement": [
     {
       "Effect": "Allow",
       "Action": [
         "s3:GetObject*",
         "s3:PutObject",
         "s3:PutObjectAcl"
       ],
       "Resource": [
         "arn:aws:s3:::codepipeline-us-east-2-1234567890/*"
       ]
     }
   ]
}
```

13. 选择查看策略。
14. 在 Name (名称) 中，输入此策略的名称。选择 创建策略。
15. 在 “权限” 选项卡上，在 “策略名称” 下的策略列表ReadOnlyAccess中找到 AmazonS3，然后选择策略旁边的删除图标 (X)。系统提示时，选择 Detach。
16. 选择 “信任关系” 选项卡，然后选择 “编辑信任策略”。选择左栏中的添加委托人选项。*########## IAM ###### AccountA #### CodePipeline ##### ARN#*arn:aws:iam::Account_A:root从 “AWS 委托人” 列表中删除，然后选择 “更新策略”。

步骤 2：编辑管道

您不能使用 CodePipeline 控制台创建或编辑使用与其他 AWS 账户关联的资源的管道。但是，您可以使用控制台创建管道的总体结构，然后使用编辑管道并添加这些资源。 AWS CLI 或者，您可以使用现有管道的结构并手动向其添加资源。

添加与其他 AWS 账户关联的资源 (AWS CLI)

1. 在终端（Linux、macOS 或 Unix）或命令提示符 (Windows) 中，对您要添加资源的管道运行 get-pipeline 命令。将命令输出复制到 JSON 文件。例如，对于名为的管道 MyFirstPipeline，您可以键入类似于以下内容的内容：

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   输出将发送到 *pipeline.json* 文件。

2. 在任何纯文本编辑器中打开 JSON 文件。进入项目存储后"type": "S3"，添加 KMS 加密密钥、ID 和类型信息，*codepipeline-us-east##* -2-1234567890 是用于存储管道项目的 Amazon S3 存储桶的名称，也是您刚刚创建的客户托***arn:aws:kms:us-east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE***管密钥的 ARN：

   ```
   {
     "artifactStore”: {
       "location": "codepipeline-us-east-2-1234567890",
       "type": "S3",
       "encryptionKey": {
         "id": "arn:aws:kms:us-
   east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE",
         "type": "KMS"
       }
     },
   ```

3. *############ A ccountB ### CodeDeploy ############ (CrossAccount_Role) #roleArn##*

   以下示例显示了添加名为的部署操作的 JSON *ExternalDeploy*。*###### “##” # #### AccountB #### CodeDeploy ###*在以下示例中，*AccountB* 的 ARN 是 *012ID_ACCOUNT_B*：

```
,
            {
                "name": "Staging",
                "actions": [
                    {
                        "inputArtifacts": [
                            {
                                "name": "MyAppBuild"
                            }
                        ],
                        "name": "ExternalDeploy",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeDeploy"
                        },
                        "outputArtifacts": [],
                        "configuration": {
                            "ApplicationName": "AccountBApplicationName",
                            "DeploymentGroupName": "AccountBApplicationGroupName"
                        },
                        "runOrder": 1,
                        "roleArn":
 "arn:aws:iam::012ID_ACCOUNT_B:role/CrossAccount_Role"
                    }
                ]
            }
```

> **Note**
>
> 这不是整个管道的 JSON，而只是一个阶段中操作的结构。

4. 您必须从文件中删除 metadata 行以便 update-pipeline 命令可以使用它。从 JSON 文件中的管道结构中删除该部分（"metadata": { } 行以及 "created"、"pipelineARN" 和 "updated" 字段）。

   例如，从结构中删除以下各行：

```
"metadata": {
  "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
```

```
  "created": "date",
  "updated": "date"
  }
```

保存该文件。

5. 要应用更改，请运行 update-pipeline 命令并指定一个管道 JSON 文件，类似于以下内容：

> **Important**
>
> 务必在文件名前包含 `file://`。此命令中需要该项。

```
aws codepipeline update-pipeline --cli-input-json file://pipeline.json
```

该命令会返回编辑后的管道的整个结构。

测试使用与其他 AWS 账户关联的资源的管道

1. 在终端（Linux、macOS 或 Unix）或命令提示符 (Windows) 中，运行 start-pipeline-execution 命令，指定管道的名称，类似下面这样：

```
aws codepipeline start-pipeline-execution --name MyFirstPipeline
```

有关更多信息，请参阅 手动启动管道。

2. 使用 Ac *coun AWS Management Console tA* 登录并打开 CodePipeline控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

将显示与您的 AWS 账户关联的所有管道的名称。

3. 在名称中，选择您刚编辑的管道的名称。这将打开管道的详细视图，包括管道每个阶段中每个操作的状态。

4. 观看管道中的进度。等待有关使用与其他 AWS 账户关联的资源的操作的成功消息。

> **Note**
>
> 使用 *AccountA* 登录时，如果您尝试查看操作的详细信息，您将收到一条错误消息。注销，然后使用 A *ccountB* 登录以在中查看部署详细信息。 CodeDeploy

迁移轮询管道以使用基于事件的更改检测

AWS CodePipeline 支持完整、 end-to-end 持续的交付，包括在发生代码更改时启动您的管道。在发生代码更改后，有两种支持的方法来启动管道：基于事件的更改检测和轮询。建议对管道使用基于事件的更改检测。

使用此处包含的过程，迁移（更新）轮询管道以对管道使用基于事件的更改检测方法。

推荐的管道基于事件的变更检测方法由管道来源决定，例如 CodeCommit。例如，在这种情况下，轮询管道需要使用 EventBridge迁移到基于事件的变更检测。

如何迁移轮询管道

要迁移轮询管道，请确定您的轮询管道，然后确定建议的基于事件的更改检测方法：

- 使用查看账户中的轮询管道中的步骤来确定轮询管道。
- 在此表中，找到您的管道源类型，然后选择具有要用于迁移轮询管道的实施的过程。每个部分都包含多种迁移方法，例如使用 CLI 或 AWS CloudFormation。

如何将管道迁移至建议的更改检测方法		
管道源	建议的基于事件的检测方法	迁移过程
AWS CodeCommit	EventBridge （推荐）。	请参阅 使用 CodeCommit 来源迁移轮询管道。
Amazon S3	EventBridge 并启用存储桶以接收事件通知（推荐）。	请参阅 迁移为事件启用了 S3 源的轮询管道。
Amazon S3	EventBridge 还有一条 AWS CloudTrail 小径。	请参阅 使用 S3 源和 CloudTrail 跟踪迁移轮询管道。
GitHub 第 1 版	连接（建议）	请参阅 将 GitHub 版本 1 源操作的轮询管道迁移到连接。
GitHub 第 1 版	Webhook	请参阅 将 GitHub 版本 1 源操作的轮询管道迁移到 webhook。

> **Important**
>
> 对于适用的管道操作配置更新（例如具有 GitHub版本 1 操作的管道），您必须在源操作的配置中将该PollForSourceChanges参数显式设置为 false，以停止管道轮询。因此，可能会错误地将管道配置为基于事件的变更检测和轮询，例如配置 EventBridge 规则并省略参数。PollForSourceChanges这将导致重复的管道执行，并且会将管道计入轮询管道总数限制，默认情况下，该值远低于基于事件的管道。有关更多信息，请参阅 中的配额 AWS CodePipeline。

查看账户中的轮询管道

首先，使用以下脚本之一来确定账户中的哪些管道配置为使用轮询。这些是迁移到基于事件的更改检测的管道。

查看账户中的轮询管道（脚本）

按照以下步骤使用脚本来确定账户中使用轮询的管道。

1. 打开终端窗口，然后执行以下操作之一：

   - 运行以下命令创建名为 PollingPipelinesExtractor.sh 的新脚本。

     ```
     vi PollingPipelinesExtractor.sh
     ```

   - 要使用 python 脚本，请运行以下命令创建一个名为 PollingPipelinesExtractor.py 的新 python 脚本。

     ```
     vi PollingPipelinesExtractor.py
     ```

2. 将以下代码复制并粘贴到PollingPipelinesExtractor脚本中。请执行以下操作之一：

   - 将以下代码复制并粘贴到 PollingPipelinesExtractor.sh 脚本中。

     ```
     #!/bin/bash

     set +x

     POLLING_PIPELINES=()
     LAST_EXECUTED_DATES=()
     ```

```
NEXT_TOKEN=null
HAS_NEXT_TOKEN=true
if [[ $# -eq 0 ]] ; then
    echo 'Please provide region name'
    exit 0
fi
REGION=$1


while [ "$HAS_NEXT_TOKEN" != "false" ]; do
    if [ "$NEXT_TOKEN" != "null" ];
        then
            LIST_PIPELINES_RESPONSE=$(aws codepipeline list-pipelines --region
 $REGION --next-token $NEXT_TOKEN)
        else
            LIST_PIPELINES_RESPONSE=$(aws codepipeline list-pipelines --region
 $REGION)
    fi
    LIST_PIPELINES=$(jq -r '.pipelines[].name' <<< "$LIST_PIPELINES_RESPONSE")
    NEXT_TOKEN=$(jq -r '.nextToken' <<< "$LIST_PIPELINES_RESPONSE")
    if [ "$NEXT_TOKEN" == "null" ];
        then
            HAS_NEXT_TOKEN=false
    fi

    for pipline_name in $LIST_PIPELINES
    do
        PIPELINE=$(aws codepipeline get-pipeline --name $pipline_name --region
 $REGION)
        HAS_POLLABLE_ACTIONS=$(jq '.pipeline.stages[].actions[] |
 select(.actionTypeId.category == "Source") | select(.actionTypeId.owner
 == ("ThirdParty","AWS")) | select(.actionTypeId.provider ==
 ("GitHub","S3","CodeCommit")) | select(.configuration.PollForSourceChanges ==
 ("true",null))' <<< "$PIPELINE")
        if [ ! -z "$HAS_POLLABLE_ACTIONS" ];
        then
            POLLING_PIPELINES+=("$pipline_name")
            PIPELINE_EXECUTIONS=$(aws codepipeline list-pipeline-executions --
pipeline-name $pipline_name --region $REGION)
            LAST_EXECUTION=$(jq -r '.pipelineExecutionSummaries[0]' <<<
 "$PIPELINE_EXECUTIONS")
            if [ "$LAST_EXECUTION" != "null" ];
                then
```

```
                    LAST_EXECUTED_TIMESTAMP=$(jq -r '.startTime' <<<
 "$LAST_EXECUTION")
                    LAST_EXECUTED_DATE="$(date -r ${LAST_EXECUTED_TIMESTAMP%.*})"
                else
                    LAST_EXECUTED_DATE="Not executed in last year"
            fi
            LAST_EXECUTED_DATES+=("$LAST_EXECUTED_DATE")
        fi
    done

done

fileName=$REGION-$(date +%s)
printf "| %-30s | %-30s |\n" "Polling Pipeline Name" "Last Executed Time"
printf "| %-30s | %-30s |\n" "_____________________" "___________________"
for i in "${!POLLING_PIPELINES[@]}"; do
  printf "| %-30s | %-30s |\n" "${POLLING_PIPELINES[i]}"
 "${LAST_EXECUTED_DATES[i]}"
  printf "${POLLING_PIPELINES[i]}," >> $fileName.csv
done

printf "\nSaving Polling Pipeline Names to file $fileName.csv."
```

- 将以下代码复制并粘贴到 PollingPipelinesExtractor.py 脚本中。

```
import boto3
import sys
import time
import math

hasNextToken = True
nextToken = ""
pollablePipelines = []
lastExecutedTimes = []
if len(sys.argv) == 1:
    raise Exception("Please provide region name.")
session = boto3.Session(profile_name='default', region_name=sys.argv[1])
codepipeline = session.client('codepipeline')

def is_pollable_action(action):
    actionTypeId = action['actionTypeId']
    configuration = action['configuration']
    return actionTypeId['owner'] in {"AWS", "ThirdParty"}
 and actionTypeId['provider'] in {"GitHub", "CodeCommit",
```

```
 "S3"} and ('PollForSourceChanges' not in configuration or
 configuration['PollForSourceChanges'] == 'true')

def has_pollable_actions(pipeline):
    hasPollableAction = False
    pipelineDefinition = codepipeline.get_pipeline(name=pipeline['name'])
['pipeline']
    for action in pipelineDefinition['stages'][0]['actions']:
        hasPollableAction = is_pollable_action(action)
        if hasPollableAction:
            break
    return hasPollableAction

def get_last_executed_time(pipelineName):

 pipelineExecutions=codepipeline.list_pipeline_executions(pipelineName=pipelineName)
['pipelineExecutionSummaries']
    if pipelineExecutions:
        return pipelineExecutions[0]['startTime'].strftime("%A %m/%d/%Y, %H:%M:
%S")
    else:
        return "Not executed in last year"

while hasNextToken:
    if nextToken=="":
        list_pipelines_response = codepipeline.list_pipelines()
    else:
        list_pipelines_response =
 codepipeline.list_pipelines(nextToken=nextToken)
    if 'nextToken' in list_pipelines_response:
        nextToken = list_pipelines_response['nextToken']
    else:
        hasNextToken= False
    for pipeline in list_pipelines_response['pipelines']:
        if has_pollable_actions(pipeline):
            pollablePipelines.append(pipeline['name'])
            lastExecutedTimes.append(get_last_executed_time(pipeline['name']))

fileName="{region}-
{timeNow}.csv".format(region=sys.argv[1],timeNow=math.trunc(time.time()))
file = open(fileName, 'w')

print ("{:<30} {:<30} {:<30}".format('Polling Pipeline Name', '|','Last Executed
 Time'))
```

```
print ("{:<30} {:<30} {:<30}".format('_____________________',
 '|','___________________'))
for i in range(len(pollablePipelines)):
    print("{:<30} {:<30} {:<30}".format(pollablePipelines[i], '|',
 lastExecutedTimes[i]))
    file.write("{pipeline},".format(pipeline=pollablePipelines[i]))
file.close()
print("\nSaving Polling Pipeline Names to file
 {fileName}".format(fileName=fileName))
```

3. 对于每个具有管道的区域，必须为此区域运行脚本。要运行脚本，请执行以下操作之一：

   - 运行以下命令运行名为 PollingPipelinesExtractor.sh 的脚本。在此示例中，区域为 us-west-2。

     ```
     ./PollingPipelinesExtractor.sh us-west-2
     ```

   - 对于 python 脚本，运行以下命令以运行名为 PollingPipelinesExtractor.py 的 python 脚本。在此示例中，区域为 us-west-2。

     ```
     python3 PollingPipelinesExtractor.py us-west-2
     ```

   在脚本的以下示例输出中，区域 us-west-2 返回了轮询管道列表，并显示了每个管道的上次执行时间。

   ```
    % ./pollingPipelineExtractor.sh us-west-2

   | Polling Pipeline Name | Last Executed Time           |
   | _____________________ | ___________________          |
   | myCodeBuildPipeline   | Wed Mar 8 09:35:49 PST 2023  |
   | myCodeCommitPipeline  | Mon Apr 24 22:32:32 PDT 2023 |
   | TestPipeline          | Not executed in last year    |

   Saving list of polling pipeline names to us-west-2-1682496174.csv...%
   ```

   分析脚本输出，并针对列表中的每个管道，更新轮询源以使用建议的基于事件的更改检测方法。

> **Note**
>
> 轮询管道由管道操作配置中的 `PollForSourceChanges` 参数决定。如果管道源配置省略了`PollForSourceChanges`参数，则 CodePipeline默认会轮询存储库以查看源代码更改。如果包括 `PollForSourceChanges` 并将其设置为 true，则此行为相同。有关更多信息，请参阅管道源操作的配置参数，例如Amazon S3 源操作中的 Amazon S3 源操作配置参数。

请注意，此脚本还会生成一个包含账户中轮询管道列表的 .csv 文件，并将此 .csv 文件保存到当前工作文件夹。

使用 CodeCommit 来源迁移轮询管道

您可以迁移轮询管道 EventBridge 以用于检测 CodeCommit源存储库或 Amazon S3 源存储桶中的更改。

CodeCommit--对于带有 CodeCommit 源的管道，请修改管道，以便自动进行更改检测 EventBridge。从以下方法中进行选择以实施迁移：

- 控制台：迁移轮询管道（CodeCommit 或 Amazon S3 源代码）（控制台）
- CLI：迁移轮询管道（CodeCommit 来源）(CLI)
- AWS CloudFormation: 迁移轮询管道（CodeCommit 来源）（AWS CloudFormation 模板）

迁移轮询管道（CodeCommit 或 Amazon S3 源代码）（控制台）

您可以使用 CodePipeline 控制台更新管道， EventBridge 以用于检测 CodeCommit 源存储库或 Amazon S3 源存储桶中的更改。

> **Note**
>
> 当您使用控制台编辑包含 CodeCommit 源存储库或 Amazon S3 源存储桶的管道时，将为您创建规则和 IAM 角色。如果您使用编辑管道，则必须自己创建 EventBridge 规则和 IAM 角色。AWS CLI 有关更多信息，请参阅 CodeCommit 源操作和 EventBridge。

使用以下步骤可编辑正在使用定期检查的管道。如果要创建管道，请参阅在中创建管道 CodePipeline。

编辑管道源阶段

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   将显示与您的 AWS 账户关联的所有管道的名称。
2. 在 Name 中，选择您要编辑的管道的名称。这将打开管道的详细视图，包括管道每个阶段中每个操作的状态。
3. 在管道详细信息页中，选择编辑。
4. 在 Edit stage (编辑阶段) 中，选择源操作上的编辑图标。
5. 展开 “更改检测选项”，然后选择 “使用 CloudWatch 事件”，以便在发生更改时自动启动我的管道（推荐）。

   将出现一条消息，显示要为此管道创建的 EventBridge 规则。选择更新。

   如果您更新的是具有 Amazon S3 源的管道，将显示以下消息。选择更新。
6. 编辑完您的管道后，请选择 Save pipeline changes 以返回到摘要页面。

   一条消息显示要为您的管道创建的 EventBridge 规则的名称。选择 保存并继续。
7. 要测试您的操作，请使用发布更改，将更改提交 AWS CLI 到管道源阶段中指定的源。

迁移轮询管道（CodeCommit 来源）(CLI)

按照以下步骤编辑正在使用轮询（定期检查）的管道，以使用 EventBridge 规则启动管道。如果要创建管道，请参阅在中创建管道 CodePipeline。

要使用构建事件驱动的管道 CodeCommit，请编辑管道的`PollForSourceChanges`参数，然后创建以下资源：

- EventBridge 事件
- IAM 角色，以允许该事件启动您的管道

编辑管道的 PollForSourceChanges参数

> **Important**
>
> 使用此方法创建管道时，如果 PollForSourceChanges 参数未明确设置为 false，则默认为 true。添加基于事件的更改检测时，必须将参数添加到输出并将其设置为 false 以禁用轮询。否则，您的管道将针对单个源更改启动两次。有关更多信息，请参阅 PollForSourceChanges 参数的默认设置。

1. 运行 get-pipeline 命令以将管道结构复制到 JSON 文件中。例如，对于名为 MyFirstPipeline 的管道，运行以下命令：

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   该命令不会返回任何结果，但您创建的文件将出现在您运行命令所在的目录中。

2. 在任何纯文本编辑器中打开 JSON 文件并通过将 PollForSourceChanges 参数更改为 false 来编辑源阶段，如此示例中所示。

   我为何做出此更改？ 将此参数更改为 false 将关闭定期检查，因此您只能使用基于事件的更改检测。

   ```
   "configuration": {
       "PollForSourceChanges": "false",
       "BranchName": "main",
       "RepositoryName": "MyTestRepo"
   },
   ```

3. 如果您要使用通过 get-pipeline 命令检索到的管道结构，请删除 JSON 文件中的 metadata 行。否则，update-pipeline 命令无法使用它。删除 "metadata": { } 行以及 "created"、"pipelineARN" 和 "updated" 字段。

   例如，从结构中删除以下各行：

   ```
   "metadata": {
       "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
       "created": "date",
       "updated": "date"
   },
   ```

保存该文件。

4. 要应用更改，请运行 update-pipeline 命令，指定管道 JSON 文件：

> **Important**
>
> 务必在文件名前包含 `file://`。此命令中需要该项。

```
aws codepipeline update-pipeline --cli-input-json file://pipeline.json
```

该命令会返回编辑后的管道的整个结构。

> **Note**
>
> update-pipeline 命令会停止管道。如果在运行 update-pipeline 命令时正在通过管道运行修订，则该运行会被停止。您必须手动启动管道，通过升级后的管道运行此修订。使用 **start-pipeline-execution** 命令手动启动您的管道。

创建以事件源 CodeCommit 为目标的 EventBridge 规则 CodePipeline

1. 添加用于调 EventBridge CodePipeline 用规则的权限。有关更多信息，请参阅使用适用于 Amazon EventBridge 的基于资源的政策。

   a. 使用以下示例创建允许 EventBridge 担任服务角色的信任策略。将信任策略命名为 `trustpolicyforEB.json`。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "events.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
```

```
}
```

b. 使用以下命令创建 Role-for-MyRule 角色并附加信任策略。

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. 为名为 MyFirstPipeline 的管道创建权限策略 JSON，如此示例中所示。将权限策略命名为 permissionspolicyforEB.json。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-west-2:80398EXAMPLE:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 使用以下命令将 CodePipeline-Permissions-Policy-for-EB 权限策略附加到 Role-for-MyRule 角色。

我为何做出此更改？ 将此策略添加到角色会为创建权限 EventBridge。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. 调用 put-rule 命令并包含 --name、--event-pattern 和 --role-arn 参数。

我为何做出此更改？ 此命令将允许 AWS CloudFormation 创建事件。

以下示例命令创建一个名为 MyCodeCommitRepoRule 的规则。

```
aws events put-rule --name "MyCodeCommitRepoRule" --event-pattern "{\"source\":
[\"aws.codecommit\"],\"detail-type\":[\"CodeCommit Repository State Change\"],
\"resources\":[\"repository-ARN\"],\"detail\":{\"referenceType\":[\"branch\"],
```

```
\"referenceName\":[\"main\"]}}" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-
MyRule"
```

3. 要添加 CodePipeline 为目标，请调用put-targets命令并添加以下参数：

   - `--rule` 参数与您使用 put-rule 创建的 `rule_name` 结合使用。
   - `--targets` 参数与目标列表中该目标的列表 `Id` 以及目标管道的 ARN 结合使用。

   以下示例命令为名为 `MyCodeCommitRepoRule` 的规则指定此内容，目标 `Id` 由数字 1 组成，这指示此内容位于规则的目标列表中，而这是目标 1。示例命令还为管道指定一个示例 ARN。管道在存储库中发生更改时启动。

```
aws events put-targets --rule MyCodeCommitRepoRule --targets
 Id=1,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
```

迁移轮询管道（CodeCommit 来源）（AWS CloudFormation 模板）

要使用构建事件驱动的管道 AWS CodeCommit，请编辑管道的`PollForSourceChanges`参数，然后将以下资源添加到模板中：

- 一条 EventBridge 规则
- 您的 EventBridge 规则的 IAM 角色

如果您 AWS CloudFormation 使用创建和管理管道，则您的模板将包含以下内容。

> **Note**
>
> 源阶段中的 `Configuration` 属性称为 `PollForSourceChanges`。如果该属性未包含在您的模板中，`PollForSourceChanges` 在默认情况下将设置为 `true`。

YAML

```
Resources:
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: codecommit-polling-pipeline
```

```
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: CodeCommit
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                BranchName: !Ref BranchName
                RepositoryName: !Ref RepositoryName
                PollForSourceChanges: true
              RunOrder: 1
```

JSON

```
"Stages": [
    {
        "Name": "Source",
  "Actions": [{
      "Name": "SourceAction",
      "ActionTypeId": {
    "Category": "Source",
    "Owner": "AWS",
    "Version": 1,
    "Provider": "CodeCommit"
      },
      "OutputArtifacts": [{
          "Name": "SourceOutput"
      }],
      "Configuration": {
          "BranchName": {
        "Ref": "BranchName"
    },
    "RepositoryName": {
        "Ref": "RepositoryName"
```

```
    },
    "PollForSourceChanges": true
            },
            "RunOrder": 1
        }]
    },
```

更新您的管道 AWS CloudFormation 模板并创建 EventBridge 规则

1. 在模板下的模板中Resources，使用AWS::IAM::Role AWS CloudFormation 资源配置允许您的事件启动管道的 IAM 角色。此条目将创建一个使用两个策略的角色：

   - 第一个策略允许代入角色。
   - 第二个策略提供启动管道所需的权限。

   我为何做出此更改？ 添加AWS::IAM::Role资源可以 AWS CloudFormation 为创建权限 EventBridge。此资源已添加到您的 AWS CloudFormation 堆栈中。

   YAML

```
EventRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: 2012-10-17
      Statement:
        -
          Effect: Allow
          Principal:
            Service:
              - events.amazonaws.com
          Action: sts:AssumeRole
    Path: /
    Policies:
      -
        PolicyName: eb-pipeline-execution
        PolicyDocument:
          Version: 2012-10-17
          Statement:
            -
              Effect: Allow
```

```
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
```

JSON

```
"EventRole": {
  "Type": "AWS::IAM::Role",
  "Properties": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": [
              "events.amazonaws.com"
            ]
          },
          "Action": "sts:AssumeRole"
        }
      ]
    },
    "Path": "/",
    "Policies": [
      {
        "PolicyName": "eb-pipeline-execution",
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Action": "codepipeline:StartPipelineExecution",
              "Resource": {
                "Fn::Join": [
                  "",
                  [
                    "arn:aws:codepipeline:",
                    {
                      "Ref": "AWS::Region"
                    },
                    ":",
                    {
```

```
                      "Ref": "AWS::AccountId"
                    },
                    ":",
                    {
                      "Ref": "AppPipeline"
                    }
                  ]

...
```

2. 在模板的下方Resources，使用AWS::Events::Rule AWS CloudFormation 资源添加 EventBridge 规则。此事件模式会创建一个事件，以监控向存储库推送更改的操作。当 EventBridge 检测到存储库状态更改时，将在目标管道StartPipelineExecution上调用该规则。

   我为何做出此更改？ 添加AWS::Events::Rule资源 AWS CloudFormation 即可创建事件。此资源已添加到您的 AWS CloudFormation 堆栈中。

   YAML

```
 EventRule:
   Type: AWS::Events::Rule
   Properties:
     EventPattern:
       source:
         - aws.codecommit
       detail-type:
         - 'CodeCommit Repository State Change'
       resources:
         - !Join [ '', [ 'arn:aws:codecommit:', !Ref 'AWS::Region', ':', !Ref
'AWS::AccountId', ':', !Ref RepositoryName ] ]
       detail:
         event:
           - referenceCreated
           - referenceUpdated
         referenceType:
           - branch
         referenceName:
           - main
     Targets:
       -
         Arn:
```

```
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
         RoleArn: !GetAtt EventRole.Arn
         Id: codepipeline-AppPipeline
```

JSON

```
"EventRule": {
  "Type": "AWS::Events::Rule",
  "Properties": {
    "EventPattern": {
      "source": [
        "aws.codecommit"
      ],
      "detail-type": [
        "CodeCommit Repository State Change"
      ],
      "resources": [
        {
          "Fn::Join": [
            "",
            [
              "arn:aws:codecommit:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
                "Ref": "RepositoryName"
              }
            ]
          ]
        }
      ],
      "detail": {
        "event": [
          "referenceCreated",
          "referenceUpdated"
```

```
        ],
        "referenceType": [
          "branch"
        ],
        "referenceName": [
          "main"
        ]
      }
    },
    "Targets": [
      {
        "Arn": {
          "Fn::Join": [
            "",
            [
              "arn:aws:codepipeline:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
                "Ref": "AppPipeline"
              }
            ]
          ]
        },
        "RoleArn": {
          "Fn::GetAtt": [
            "EventRole",
            "Arn"
          ]
        },
        "Id": "codepipeline-AppPipeline"
      }
    ]
  }
},
```

3. 将更新后的模板保存到本地计算机，然后打开 AWS CloudFormation 控制台。

4. 选择堆栈，然后选择为当前堆栈创建更改集。
5. 上传模板，然后查看 AWS CloudFormation中列出的更改。这些是要对堆栈进行的更改。您应在列表中看到新资源。
6. 选择执行。

编辑管道的 PollForSourceChanges参数

> **Important**
>
> 许多情况下，当您创建管道时，PollForSourceChanges 参数默认为 true。添加基于事件的更改检测时，必须将参数添加到输出并将其设置为 false 以禁用轮询。否则，您的管道将针对单个源更改启动两次。有关更多信息，请参阅 PollForSourceChanges 参数的默认设置。

- 在模板中，将 PollForSourceChanges 更改为 false。如果您未在管道定义中包含 PollForSourceChanges，请添加它并将它设置为 false。

  我为何做出此更改？ 将此参数更改为 false 将关闭定期检查，因此您只能使用基于事件的更改检测。

  YAML

```
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: CodeCommit
            OutputArtifacts:
              - Name: SourceOutput
            Configuration:
              BranchName: !Ref BranchName
              RepositoryName: !Ref RepositoryName
              PollForSourceChanges: false
            RunOrder: 1
```

JSON

```
{
  "Name": "Source",
  "Actions": [
    {
      "Name": "SourceAction",
      "ActionTypeId": {
        "Category": "Source",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "CodeCommit"
      },
      "OutputArtifacts": [
        {
          "Name": "SourceOutput"
        }
      ],
      "Configuration": {
        "BranchName": {
          "Ref": "BranchName"
        },
        "RepositoryName": {
          "Ref": "RepositoryName"
        },
        "PollForSourceChanges": false
      },
      "RunOrder": 1
    }
  ]
},
```

Example

当您使用创建这些资源时 AWS CloudFormation，系统会在创建或更新存储库中的文件时触发您的管道。下面是最终模板代码段：

YAML

```
Resources:
```

```
 EventRole:
   Type: AWS::IAM::Role
   Properties:
     AssumeRolePolicyDocument:
       Version: 2012-10-17
       Statement:
         -
           Effect: Allow
           Principal:
             Service:
               - events.amazonaws.com
           Action: sts:AssumeRole
     Path: /
     Policies:
       -
         PolicyName: eb-pipeline-execution
         PolicyDocument:
           Version: 2012-10-17
           Statement:
             -
               Effect: Allow
               Action: codepipeline:StartPipelineExecution
               Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region',
':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
 EventRule:
   Type: AWS::Events::Rule
   Properties:
     EventPattern:
       source:
         - aws.codecommit
       detail-type:
         - 'CodeCommit Repository State Change'
       resources:
         - !Join [ '', [ 'arn:aws:codecommit:', !Ref 'AWS::Region', ':', !Ref
'AWS::AccountId', ':', !Ref RepositoryName ] ]
       detail:
         event:
           - referenceCreated
           - referenceUpdated
         referenceType:
           - branch
         referenceName:
           - main
     Targets:
```

```
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: codecommit-events-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: CodeCommit
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                BranchName: !Ref BranchName
                RepositoryName: !Ref RepositoryName
                PollForSourceChanges: false
              RunOrder: 1


...
```

JSON

```
    "Resources": {

...

        "EventRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
```

```
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": [
                                    "events.amazonaws.com"
                                ]
                            },
                            "Action": "sts:AssumeRole"
                        }
                    ]
                },
                "Path": "/",
                "Policies": [
                    {
                        "PolicyName": "eb-pipeline-execution",
                        "PolicyDocument": {
                            "Version": "2012-10-17",
                            "Statement": [
                                {
                                    "Effect": "Allow",
                                    "Action": "codepipeline:StartPipelineExecution",
                                    "Resource": {
                                        "Fn::Join": [
                                            "",
                                            [
                                                "arn:aws:codepipeline:",
                                                {
                                                    "Ref": "AWS::Region"
                                                },
                                                ":",
                                                {
                                                    "Ref": "AWS::AccountId"
                                                },
                                                ":",
                                                {
                                                    "Ref": "AppPipeline"
                                                }
                                            ]
                                        ]
                                    }
                                }
```

```
                        ]
                    }
                }
            ]
        }
    },
    "EventRule": {
        "Type": "AWS::Events::Rule",
        "Properties": {
            "EventPattern": {
                "source": [
                    "aws.codecommit"
                ],
                "detail-type": [
                    "CodeCommit Repository State Change"
                ],
                "resources": [
                    {
                        "Fn::Join": [
                            "",
                            [
                                "arn:aws:codecommit:",
                                {
                                    "Ref": "AWS::Region"
                                },
                                ":",
                                {
                                    "Ref": "AWS::AccountId"
                                },
                                ":",
                                {
                                    "Ref": "RepositoryName"
                                }
                            ]
                        ]
                    }
                ],
                "detail": {
                    "event": [
                        "referenceCreated",
                        "referenceUpdated"
                    ],
                    "referenceType": [
                        "branch"
```

```
                    ],
                    "referenceName": [
                        "main"
                    ]
                }
            },
            "Targets": [
                {
                    "Arn": {
                        "Fn::Join": [
                            "",
                            [
                                "arn:aws:codepipeline:",
                                {
                                    "Ref": "AWS::Region"
                                },
                                ":",
                                {
                                    "Ref": "AWS::AccountId"
                                },
                                ":",
                                {
                                    "Ref": "AppPipeline"
                                }
                            ]
                        ]
                    },
                    "RoleArn": {
                        "Fn::GetAtt": [
                            "EventRole",
                            "Arn"
                        ]
                    },
                    "Id": "codepipeline-AppPipeline"
                }
            ]
        }
    },
    "AppPipeline": {
        "Type": "AWS::CodePipeline::Pipeline",
        "Properties": {
            "Name": "codecommit-events-pipeline",
            "RoleArn": {
                "Fn::GetAtt": [
```

```
                        "CodePipelineServiceRole",
                        "Arn"
                    ]
                },
                "Stages": [
                    {
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "AWS",
                                    "Version": 1,
                                    "Provider": "CodeCommit"
                                },
                                "OutputArtifacts": [
                                    {
                                        "Name": "SourceOutput"
                                    }
                                ],
                                "Configuration": {
                                    "BranchName": {
                                        "Ref": "BranchName"
                                    },
                                    "RepositoryName": {
                                        "Ref": "RepositoryName"
                                    },
                                    "PollForSourceChanges": false
                                },
                                "RunOrder": 1
                            }
                        ]
                    },

...
```

迁移为事件启用了 S3 源的轮询管道

对于具有 Amazon S3 源的管道，请修改管道，以便通过 EventBridge 启用事件通知的源存储桶自动进行更改检测。如果您正在使用 CLI 或 AWS CloudFormation 迁移管道，则推荐使用此方法。

> **Note**
>
> 这包括使用启用了事件通知功能的存储桶，在这种存储桶中，您无需创建单独的 CloudTrail 跟踪。如果您使用的是控制台，则会为您设置事件规则和 CloudTrail 跟踪。有关这些步骤，请参阅使用 S3 源和 CloudTrail 跟踪迁移轮询管道。

- CLI：使用 S3 源和跟 CloudTrail 踪 (CLI) 迁移轮询管道
- AWS CloudFormation: 使用 S3 源和 CloudTrail 跟踪（AWS CloudFormation 模板）迁移轮询管道

迁移为事件启用了 S3 源的轮询管道 (CLI)

按照以下步骤编辑正在使用轮询（定期检查）的管道， EventBridge 改为使用事件。如果要创建管道，请参阅在中创建管道 CodePipeline。

要构建具有 Amazon S3 源的事件驱动管道，您应编辑管道的 `PollForSourceChanges` 参数，然后创建以下资源：

- EventBridge 事件规则
- 允许 EventBridge 事件启动管道的 IAM 角色

创建以 Amazon S3 作为事件源和 CodePipeline 目标的 EventBridge 规则并应用权限策略

1. 授 EventBridge 予使用调 CodePipeline 用规则的权限。有关更多信息，请参阅使用适用于 Amazon EventBridge 的基于资源的政策。

   a. 使用以下示例创建允许 EventBridge担任服务角色的信任策略。将它命名为 `trustpolicyforEB.json`。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "events.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
```

```
        }
    ]
}
```

b. 使用以下命令创建 Role-for-MyRule 角色并附加信任策略。

我为何做出此更改？ 将此信任策略添加到角色会为创建权限 EventBridge。

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. 为名为 MyFirstPipeline 的管道创建权限策略 JSON，如此处所示。将权限策略命名为 permissionspolicyforEB.json。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-west-2:80398EXAMPLE:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 使用以下命令将新的 CodePipeline-Permissions-Policy-for-EB 权限策略附加到您创建的 Role-for-MyRule 角色。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. 调用 put-rule 命令并包含 --name、--event-pattern 和 --role-arn 参数。

以下示例命令创建名为 EnabledS3SourceRule 的规则。

```
aws events put-rule --name "EnabledS3SourceRule" --event-pattern "{\"source\":
[\"aws.s3\"],\"detail-type\":[\"Object Created\"],\"detail\":{\"bucket\":{\"name\":
[\"my-bucket\"]}}}" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-MyRule"
```

3. 要添加 CodePipeline 为目标，请调用put-targets命令并添加--rule和--targets参数。

   以下示例命令为名为 EnabledS3SourceRule 的规则指定此内容，目标 Id 由数字 1 组成，这指示此内容位于规则的目标列表中，而这是目标 1。此命令还为管道指定一个示例 ARN。管道在存储库中发生更改时启动。

   ```
   aws events put-targets --rule EnabledS3SourceRule --targets Id=codepipeline-
   AppPipeline,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
   ```

编辑管道的 PollForSourceChanges参数

> **Important**
>
> 使用此方法创建管道时，如果 PollForSourceChanges 参数未明确设置为 false，则默认为 true。添加基于事件的更改检测时，必须将参数添加到输出并将其设置为 false 以禁用轮询。否则，您的管道将针对单个源更改启动两次。有关更多信息，请参阅 PollForSourceChanges 参数的默认设置。

1. 运行 get-pipeline 命令以将管道结构复制到 JSON 文件中。例如，对于名为 MyFirstPipeline 的管道，运行以下命令：

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   该命令不会返回任何结果，但您创建的文件将出现在您运行命令所在的目录中。

2. 在任何纯文本编辑器中打开 JSON 文件并通过将名为 storage-bucket 的存储桶的 PollForSourceChanges 参数更改为 false 来编辑源阶段，如此示例中所示。

   我为何做出此更改？ 将此参数设置为 false 将关闭定期检查，因此您只能使用基于事件的更改检测。

   ```
   "configuration": {
       "S3Bucket": "storage-bucket",
       "PollForSourceChanges": "false",
       "S3ObjectKey": "index.zip"
   },
   ```

3. 如果您要使用通过 get-pipeline 命令检索到的管道结构，则必须删除 JSON 文件中的 `metadata` 行。否则，update-pipeline 命令无法使用它。删除 `"metadata": { }` 行以及 `"created"`、`"pipelineARN"` 和 `"updated"` 字段。

   例如，从结构中删除以下各行：

   ```
   "metadata": {
       "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
       "created": "date",
       "updated": "date"
   },
   ```

   保存该文件。

4. 要应用更改，请运行 update-pipeline 命令，指定管道 JSON 文件：

   > **Important**
   >
   > 务必在文件名前包含 `file://`。此命令中需要该项。

   ```
   aws codepipeline update-pipeline --cli-input-json file://pipeline.json
   ```

   该命令会返回编辑后的管道的整个结构。

   > **Note**
   >
   > update-pipeline 命令会停止管道。如果在运行 update-pipeline 命令时正在通过管道运行修订，则该运行会被停止。您必须手动启动管道，通过升级后的管道运行此修订。使用 start-pipeline-execution 命令手动启动您的管道。

在为事件启用 S3 源的情况下迁移轮询管道（AWS CloudFormation 模板）

此过程适用于源桶启用了事件的管道。

使用以下步骤，将具有 Amazon S3 源的管道从轮询编辑为基于事件的更改检测。

要构建具有 Amazon S3 的事件驱动管道，您应编辑管道的 `PollForSourceChanges` 参数，然后将以下资源添加到您的模板：

- EventBridge 规则和 IAM 角色以允许此事件启动您的管道。

如果您 AWS CloudFormation 使用创建和管理管道，则您的模板将包含以下内容。

> **Note**
>
> 源阶段中的 `Configuration` 属性称为 `PollForSourceChanges`。如果您的模板未包含该属性，`PollForSourceChanges` 在默认情况下将设置为 `true`。

YAML

```
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      RoleArn: !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: S3
              OutputArtifacts:
                -
                  Name: SourceOutput
              Configuration:
                S3Bucket: !Ref SourceBucket
                S3ObjectKey: !Ref S3SourceObjectKey
                PollForSourceChanges: true
              RunOrder: 1


...
```

JSON

```
        "AppPipeline": {
```

```
            "Type": "AWS::CodePipeline::Pipeline",
            "Properties": {
                "RoleArn": {
                    "Fn::GetAtt": ["CodePipelineServiceRole", "Arn"]
                },
                "Stages": [
                    {
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "AWS",
                                    "Version": 1,
                                    "Provider": "S3"
                                },
                                "OutputArtifacts": [
                                    {
                                        "Name": "SourceOutput"
                                    }
                                ],
                                "Configuration": {
                                    "S3Bucket": {
                                        "Ref": "SourceBucket"
                                    },
                                    "S3ObjectKey": {
                                        "Ref": "SourceObjectKey"
                                    },
                                    "PollForSourceChanges": true
                                },
                                "RunOrder": 1
                            }
                        ]
                    },


...
```

创建以 Amazon S3 作为事件源和 CodePipeline 目标的 EventBridge 规则并应用权限策略

1. 在模板下的模板中Resources，使用AWS::IAM::Role AWS CloudFormation 资源配置允许您的事件启动管道的 IAM 角色。此条目将创建一个使用两个策略的角色：

   - 第一个策略允许代入角色。
   - 第二个策略提供启动管道所需的权限。

   我为何做出此更改？ 添加AWS::IAM::Role资源可以 AWS CloudFormation 为创建权限 EventBridge。此资源已添加到您的 AWS CloudFormation 堆栈中。

   YAML

```
  EventRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]


...
```

JSON

```
"EventRole": {
  "Type": "AWS::IAM::Role",
  "Properties": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": [
              "events.amazonaws.com"
            ]
          },
          "Action": "sts:AssumeRole"
        }
      ]
    },
    "Path": "/",
    "Policies": [
      {
        "PolicyName": "eb-pipeline-execution",
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Action": "codepipeline:StartPipelineExecution",
              "Resource": {
                "Fn::Join": [
                  "",
                  [
                    "arn:aws:codepipeline:",
                    {
                      "Ref": "AWS::Region"
                    },
                    ":",
                    {
                      "Ref": "AWS::AccountId"
                    },
                    ":",
                    {
```

```
                        "Ref": "AppPipeline"
                      }
                    ]
                  ]

...
```

2. 使用AWS::Events::Rule AWS CloudFormation 资源添加 EventBridge 规则。此事件模式会创建一个事件，以监控 Amazon S3 源桶中对象的创建或删除。此外，还包括您管道的目标。创建对象后，此规则将在您的目标管道上调用 StartPipelineExecution。

   我为何做出此更改？ 添加AWS::Events::Rule资源 AWS CloudFormation 即可创建事件。此资源已添加到您的 AWS CloudFormation 堆栈中。

   YAML

```
  EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventBusName: default
      EventPattern:
        source:
          - aws.s3
        detail-type:
          - Object Created
        detail:
          bucket:
            name:
              - !Ref SourceBucket
      Name: EnabledS3SourceRule
      State: ENABLED
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline


...
```

JSON

```
"EventRule": {
  "Type": "AWS::Events::Rule",
  "Properties": {
"EventBusName": "default",
"EventPattern": {
    "source": [
 "aws.s3"
    ],
    "detail-type": [
  "Object Created"
    ],
    "detail": {
  "bucket": {
      "name": [
    "s3-pipeline-source-fra-bucket"
      ]
      }
          }
},
"Name": "EnabledS3SourceRule",
      "State": "ENABLED",
      "Targets": [
      {
        "Arn": {
          "Fn::Join": [
            "",
            [
              "arn:aws:codepipeline:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
                "Ref": "AppPipeline"
              }
            ]
```

```
            ]
          },
          "RoleArn": {
            "Fn::GetAtt": [
              "EventRole",
              "Arn"
            ]
          },
          "Id": "codepipeline-AppPipeline"
        }
      ]
    }
  }
},

...
```

3. 将更新的模板保存到本地计算机，并打开 AWS CloudFormation 控制台。
4. 选择堆栈，然后选择为当前堆栈创建更改集。
5. 上传更新的模板，然后查看 AWS CloudFormation中所列的更改。以下是将对堆栈进行的更改。您应在列表中看到新资源。
6. 选择执行。

编辑管道的 PollForSourceChanges参数

> **Important**
>
> 使用此方法创建管道时，如果 PollForSourceChanges 参数未明确设置为 false，则默认为 true。添加基于事件的更改检测时，必须将参数添加到输出并将其设置为 false 以禁用轮询。否则，您的管道将针对单个源更改启动两次。有关更多信息，请参阅 PollForSourceChanges 参数的默认设置。

- 在模板中，将 PollForSourceChanges 更改为 false。如果您未在管道定义中包含 PollForSourceChanges，请添加它并将它设置为 false。

  我为何做出此更改？ 将 PollForSourceChanges 更改为 false 将关闭定期检查，因此您只能使用基于事件的更改检测。

YAML

```
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
              - Name: SourceOutput
            Configuration:
              S3Bucket: !Ref SourceBucket
              S3ObjectKey: !Ref SourceObjectKey
              PollForSourceChanges: false
            RunOrder: 1
```

JSON

```
{
   "Name": "SourceAction",
   "ActionTypeId": {
     "Category": "Source",
     "Owner": "AWS",
     "Version": 1,
     "Provider": "S3"
   },
   "OutputArtifacts": [
     {
       "Name": "SourceOutput"
     }
   ],
   "Configuration": {
     "S3Bucket": {
       "Ref": "SourceBucket"
     },
     "S3ObjectKey": {
       "Ref": "SourceObjectKey"
     },
     "PollForSourceChanges": false
```

```
      },
      "RunOrder": 1
    }
```

Example

当您使用 AWS CloudFormation 创建这些资源时，系统会在创建或更新存储库中的文件时触发您的管道。

> **Note**
>
> 请勿在此处停止。尽管您的管道已创建，但您必须为 Amazon S3 管道创建第二个 AWS CloudFormation 模板。如果您不创建第二个模板，您的管道不会有任何更改检测功能。

YAML

```
Parameters:
  SourceObjectKey:
    Description: 'S3 source artifact'
    Type: String
    Default: SampleApp_Linux.zip
  ApplicationName:
    Description: 'CodeDeploy application name'
    Type: String
    Default: DemoApplication
  BetaFleet:
    Description: 'Fleet configured in CodeDeploy'
    Type: String
    Default: DemoFleet

Resources:
  SourceBucket:
    Type: AWS::S3::Bucket
    Properties:
      NotificationConfiguration:
        EventBridgeConfiguration:
          EventBridgeEnabled: true
      VersioningConfiguration:
        Status: Enabled
  CodePipelineArtifactStoreBucket:
```

```
    Type: AWS::S3::Bucket
  CodePipelineArtifactStoreBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref CodePipelineArtifactStoreBucket
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Sid: DenyUnEncryptedObjectUploads
            Effect: Deny
            Principal: '*'
            Action: s3:PutObject
            Resource: !Join [ '', [ !GetAtt CodePipelineArtifactStoreBucket.Arn, '/
*' ] ]
            Condition:
              StringNotEquals:
                s3:x-amz-server-side-encryption: aws:kms
          -
            Sid: DenyInsecureConnections
            Effect: Deny
            Principal: '*'
            Action: s3:*
            Resource: !Sub ${CodePipelineArtifactStoreBucket.Arn}/*
            Condition:
              Bool:
                aws:SecureTransport: false
  CodePipelineServiceRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - codepipeline.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: AWS-CodePipeline-Service-3
          PolicyDocument:
```

```
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action:
                  - codecommit:CancelUploadArchive
                  - codecommit:GetBranch
                  - codecommit:GetCommit
                  - codecommit:GetUploadArchiveStatus
                  - codecommit:UploadArchive
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
                  - codedeploy:CreateDeployment
                  - codedeploy:GetApplicationRevision
                  - codedeploy:GetDeployment
                  - codedeploy:GetDeploymentConfig
                  - codedeploy:RegisterApplicationRevision
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
                  - codebuild:BatchGetBuilds
                  - codebuild:StartBuild
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
                  - devicefarm:ListProjects
                  - devicefarm:ListDevicePools
                  - devicefarm:GetRun
                  - devicefarm:GetUpload
                  - devicefarm:CreateUpload
                  - devicefarm:ScheduleRun
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
                  - lambda:InvokeFunction
                  - lambda:ListFunctions
                Resource: 'resource_ARN'
              -
                Effect: Allow
```

```
              Action:
                - iam:PassRole
              Resource: 'resource_ARN'
            -
              Effect: Allow
              Action:
                - elasticbeanstalk:*
                - ec2:*
                - elasticloadbalancing:*
                - autoscaling:*
                - cloudwatch:*
                - s3:*
                - sns:*
                - cloudformation:*
                - rds:*
                - sqs:*
                - ecs:*
              Resource: 'resource_ARN'
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: s3-events-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: S3
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                S3Bucket: !Ref SourceBucket
                S3ObjectKey: !Ref SourceObjectKey
                PollForSourceChanges: false
              RunOrder: 1
        -
          Name: Beta
```

```
          Actions:
            -
              Name: BetaAction
              InputArtifacts:
                - Name: SourceOutput
              ActionTypeId:
                Category: Deploy
                Owner: AWS
                Version: 1
                Provider: CodeDeploy
              Configuration:
                ApplicationName: !Ref ApplicationName
                DeploymentGroupName: !Ref BetaFleet
              RunOrder: 1
      ArtifactStore:
        Type: S3
        Location: !Ref CodePipelineArtifactStoreBucket
  EventRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region',
':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
  EventRule:
    Type: AWS::Events::Rule
    Properties:
```

```
      EventBusName: default
      EventPattern:
        source:
          - aws.s3
        detail-type:
          - Object Created
        detail:
          bucket:
            name:
              - !Ref SourceBucket
      Name: EnabledS3SourceRule
      State: ENABLED
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline
```

JSON

```
{
    "Parameters": {
        "SourceObjectKey": {
            "Description": "S3 source artifact",
            "Type": "String",
            "Default": "SampleApp_Linux.zip"
        },
        "ApplicationName": {
            "Description": "CodeDeploy application name",
            "Type": "String",
            "Default": "DemoApplication"
        },
        "BetaFleet": {
            "Description": "Fleet configured in CodeDeploy",
            "Type": "String",
            "Default": "DemoFleet"
        }
    },
    "Resources": {
        "SourceBucket": {
```

```
            "Type": "AWS::S3::Bucket",
              "Properties": {
                "NotificationConfiguration": {
                    "EventBridgeConfiguration": {
                        "EventBridgeEnabled": true
                    }
                },
                "VersioningConfiguration": {
                    "Status": "Enabled"
                }
            }
        },
        "CodePipelineArtifactStoreBucket": {
            "Type": "AWS::S3::Bucket"
        },
        "CodePipelineArtifactStoreBucketPolicy": {
            "Type": "AWS::S3::BucketPolicy",
            "Properties": {
                "Bucket": {
                    "Ref": "CodePipelineArtifactStoreBucket"
                },
                "PolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Sid": "DenyUnEncryptedObjectUploads",
                            "Effect": "Deny",
                            "Principal": "*",
                            "Action": "s3:PutObject",
                            "Resource": {
                                "Fn::Join": [
                                    "",
                                    [
                                        {
                                            "Fn::GetAtt": [
                                                "CodePipelineArtifactStoreBucket",
                                                "Arn"
                                            ]
                                        },
                                        "/*"
                                    ]
                                ]
                            },
                            "Condition": {
```

```
                            "StringNotEquals": {
                                "s3:x-amz-server-side-encryption": "aws:kms"
                            }
                        }
                    },
                    {
                        "Sid": "DenyInsecureConnections",
                        "Effect": "Deny",
                        "Principal": "*",
                        "Action": "s3:*",
                        "Resource": {
                            "Fn::Join": [
                                "",
                                [
                                    {
                                        "Fn::GetAtt": [
                                            "CodePipelineArtifactStoreBucket",
                                            "Arn"
                                        ]
                                    },
                                    "/*"
                                ]
                            ]
                        },
                        "Condition": {
                            "Bool": {
                                "aws:SecureTransport": false
                            }
                        }
                    }
                ]
            }
        }
    },
    "CodePipelineServiceRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": [
```

```
                            "codepipeline.amazonaws.com"
                        ]
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        },
        "Path": "/",
        "Policies": [
            {
                "PolicyName": "AWS-CodePipeline-Service-3",
                "PolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Action": [
                                "codecommit:CancelUploadArchive",
                                "codecommit:GetBranch",
                                "codecommit:GetCommit",
                                "codecommit:GetUploadArchiveStatus",
                                "codecommit:UploadArchive"
                            ],
                            "Resource": "resource_ARN"
                        },
                        {
                            "Effect": "Allow",
                            "Action": [
                                "codedeploy:CreateDeployment",
                                "codedeploy:GetApplicationRevision",
                                "codedeploy:GetDeployment",
                                "codedeploy:GetDeploymentConfig",
                                "codedeploy:RegisterApplicationRevision"
                            ],
                            "Resource": "resource_ARN"
                        },
                        {
                            "Effect": "Allow",
                            "Action": [
                                "codebuild:BatchGetBuilds",
                                "codebuild:StartBuild"
                            ],
                            "Resource": "resource_ARN"
                        },
```

```
                {
                    "Effect": "Allow",
                    "Action": [
                        "devicefarm:ListProjects",
                        "devicefarm:ListDevicePools",
                        "devicefarm:GetRun",
                        "devicefarm:GetUpload",
                        "devicefarm:CreateUpload",
                        "devicefarm:ScheduleRun"
                    ],
                    "Resource": "resource_ARN"
                },
                {
                    "Effect": "Allow",
                    "Action": [
                        "lambda:InvokeFunction",
                        "lambda:ListFunctions"
                    ],
                    "Resource": "resource_ARN"
                },
                {
                    "Effect": "Allow",
                    "Action": [
                        "iam:PassRole"
                    ],
                    "Resource": "resource_ARN"
                },
                {
                    "Effect": "Allow",
                    "Action": [
                        "elasticbeanstalk:*",
                        "ec2:*",
                        "elasticloadbalancing:*",
                        "autoscaling:*",
                        "cloudwatch:*",
                        "s3:*",
                        "sns:*",
                        "cloudformation:*",
                        "rds:*",
                        "sqs:*",
                        "ecs:*"
                    ],
                    "Resource": "resource_ARN"
                }
```

```
                        ]
                    }
                }
            ]
        }
    },
    "AppPipeline": {
        "Type": "AWS::CodePipeline::Pipeline",
        "Properties": {
            "Name": "s3-events-pipeline",
            "RoleArn": {
                "Fn::GetAtt": [
                    "CodePipelineServiceRole",
                    "Arn"
                ]
            },
            "Stages": [
                {
                    "Name": "Source",
                    "Actions": [
                        {
                            "Name": "SourceAction",
                            "ActionTypeId": {
                                "Category": "Source",
                                "Owner": "AWS",
                                "Version": 1,
                                "Provider": "S3"
                            },
                            "OutputArtifacts": [
                                {
                                    "Name": "SourceOutput"
                                }
                            ],
                            "Configuration": {
                                "S3Bucket": {
                                    "Ref": "SourceBucket"
                                },
                                "S3ObjectKey": {
                                    "Ref": "SourceObjectKey"
                                },
                                "PollForSourceChanges": false
                            },
                            "RunOrder": 1
                        }
```

```
                    ]
                },
                {
                    "Name": "Beta",
                    "Actions": [
                        {
                            "Name": "BetaAction",
                            "InputArtifacts": [
                                {
                                    "Name": "SourceOutput"
                                }
                            ],
                            "ActionTypeId": {
                                "Category": "Deploy",
                                "Owner": "AWS",
                                "Version": 1,
                                "Provider": "CodeDeploy"
                            },
                            "Configuration": {
                                "ApplicationName": {
                                    "Ref": "ApplicationName"
                                },
                                "DeploymentGroupName": {
                                    "Ref": "BetaFleet"
                                }
                            },
                            "RunOrder": 1
                        }
                    ]
                }
            ],
            "ArtifactStore": {
                "Type": "S3",
                "Location": {
                    "Ref": "CodePipelineArtifactStoreBucket"
                }
            }
        }
    },
    "EventRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
```

```
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": [
                                    "events.amazonaws.com"
                                ]
                            },
                            "Action": "sts:AssumeRole"
                        }
                    ]
                },
                "Path": "/",
                "Policies": [
                    {
                        "PolicyName": "eb-pipeline-execution",
                        "PolicyDocument": {
                            "Version": "2012-10-17",
                            "Statement": [
                                {
                                    "Effect": "Allow",
                                    "Action": "codepipeline:StartPipelineExecution",
                                    "Resource": {
                                        "Fn::Join": [
                                            "",
                                            [
                                                "arn:aws:codepipeline:",
                                                {
                                                    "Ref": "AWS::Region"
                                                },
                                                ":",
                                                {
                                                    "Ref": "AWS::AccountId"
                                                },
                                                ":",
                                                {
                                                    "Ref": "AppPipeline"
                                                }
                                            ]
                                        ]
                                    }
                                }
                            ]
                        }
```

```
                    }
                ]
            }
        },
        "EventRule": {
            "Type": "AWS::Events::Rule",

            "Properties": {
                "EventBusName": "default",
          "EventPattern": {
              "source": [
            "aws.s3"
          ],
                "detail-type": [
        "Object Created"
          ],
          "detail": {
        "bucket": {
                        "name": [
                            {
                            "Ref": "SourceBucket"
                            }
                        ]
                    }
                }
    },
    "Name": "EnabledS3SourceRule",
          "State": "ENABLED",
                "Targets": [
                    {
                        "Arn": {
                            "Fn::Join": [
                                "",
                                [
                                    "arn:aws:codepipeline:",
                                    {
                                        "Ref": "AWS::Region"
                                    },
                                    ":",
                                    {
                                        "Ref": "AWS::AccountId"
                                    },
                                    ":",
                                    {
```

```
                                    "Ref": "AppPipeline"
                                }
                            ]
                        ]
                    },
                    "RoleArn": {
                        "Fn::GetAtt": [
                            "EventRole",
                            "Arn"
                        ]
                    },
                    "Id": "codepipeline-AppPipeline"
                }
            ]
        }
    }
    }
}
```

使用 S3 源和 CloudTrail 跟踪迁移轮询管道

对于具有 Amazon S3 源的管道，请修改管道，以便通过自动进行更改检测 EventBridge。从以下方法中进行选择以实施迁移：

- 控制台：迁移轮询管道（CodeCommit 或 Amazon S3 源代码）（控制台）
- CLI：使用 S3 源和跟 CloudTrail 踪 (CLI) 迁移轮询管道
- AWS CloudFormation: 使用 S3 源和 CloudTrail 跟踪（AWS CloudFormation 模板）迁移轮询管道

使用 S3 源和跟 CloudTrail 踪 (CLI) 迁移轮询管道

按照以下步骤编辑正在使用轮询（定期检查）的管道， EventBridge 改为使用事件。如果要创建管道，请参阅在中创建管道 CodePipeline。

要构建具有 Amazon S3 源的事件驱动管道，您应编辑管道的 `PollForSourceChanges` 参数，然后创建以下资源：

- AWS CloudTrail Amazon S3 可用于记录事件的跟踪、存储桶和存储桶策略。
- EventBridge 事件

- 允许 EventBridge 事件启动管道的 IAM 角色

创建 AWS CloudTrail 跟踪并启用日志记录

要使用创建跟踪，请调用create-trail命令，指定： AWS CLI

- 跟踪名称。
- 已将 AWS CloudTrail的存储桶策略应用于的存储桶。

有关更多信息，请参阅使用AWS 命令行界面创建跟踪。

1. 调用 create-trail 命令并包含 `--name` 和 `--s3-bucket-name` 参数。

   我为何做出此更改？ 这将为您的 S3 源存储桶创建所需的 CloudTrail跟踪。

   以下示例命令使用 `--name` 和 `--s3-bucket-name` 创建一个名为 `my-trail` 的跟踪和一个名为 `myBucket` 的存储桶。

   ```
   aws cloudtrail create-trail --name my-trail --s3-bucket-name myBucket
   ```

2. 调用 start-logging 命令并包含 `--name` 参数。

   我为何做出此更改？ 此命令启动您的源存储桶的 CloudTrail 日志记录并将事件发送到 EventBridge。

   例如：

   以下命令使用 `--name` 启动针对名为 `my-trail` 的跟踪的日志记录。

   ```
   aws cloudtrail start-logging --name my-trail
   ```

3. 调用 put-event-selectors 命令并包含 `--trail-name` 和 `--event-selectors` 参数。使用事件选择器指定您希望您的跟踪记录源存储桶的数据事件并将事件发送到 EventBridge 规则。

   我为何做出此更改？ 此命令筛选事件。

   例如：

   以下示例命令使用 `--trail-name` 和 `--event-selectors` 指定源存储桶的数据事件以及名为 `myBucket/myFolder` 的前缀。

```
aws cloudtrail put-event-selectors --trail-name my-trail --event-selectors
 '[{ "ReadWriteType": "WriteOnly", "IncludeManagementEvents":false,
 "DataResources": [{ "Type": "AWS::S3::Object", "Values": ["arn:aws:s3:::myBucket/
myFolder/file.zip"] }] }]'
```

创建以 Amazon S3 作为事件源和 CodePipeline 目标的 EventBridge 规则并应用权限策略

1. 授 EventBridge 予使用调 CodePipeline 用规则的权限。有关更多信息，请参阅使用适用于 Amazon EventBridge 的基于资源的政策。

   a. 使用以下示例创建允许 EventBridge 担任服务角色的信任策略。将它命名为 `trustpolicyforEB.json`。

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Principal": {
                   "Service": "events.amazonaws.com"
               },
               "Action": "sts:AssumeRole"
           }
       ]
   }
   ```

   b. 使用以下命令创建 `Role-for-MyRule` 角色并附加信任策略。

   我为何做出此更改？ 将此信任策略添加到角色会为创建权限 EventBridge。

   ```
   aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
    file://trustpolicyforEB.json
   ```

   c. 为名为 `MyFirstPipeline` 的管道创建权限策略 JSON，如此处所示。将权限策略命名为 `permissionspolicyforEB.json`。

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

```
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws:codepipeline:us-west-2:80398EXAMPLE:MyFirstPipeline"
            ]
        }
    ]
}
```

d. 使用以下命令将新的 CodePipeline-Permissions-Policy-for-EB 权限策略附加到您创建的 Role-for-MyRule 角色。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. 调用 put-rule 命令并包含 --name、--event-pattern 和 --role-arn 参数。

以下示例命令创建名为 MyS3SourceRule 的规则。

```
aws events put-rule --name "MyS3SourceRule" --event-pattern "{\"source\":
[\"aws.s3\"],\"detail-type\":[\"AWS API Call via CloudTrail\"],\"detail\":
{\"eventSource\":[\"s3.amazonaws.com\"],\"eventName\":[\"CopyObject\",\"PutObject
\",\"CompleteMultipartUpload\"],\"requestParameters\":{\"bucketName\":[\"my-bucket
\"],\"key\":[\"my-key\"]}}}
 --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-MyRule"
```

3. 要添加 CodePipeline 为目标，请调用put-targets命令并添加--rule和--targets参数。

以下示例命令为名为 MyS3SourceRule 的规则指定此内容，目标 Id 由数字 1 组成，这指示此内容位于规则的目标列表中，而这是目标 1。此命令还为管道指定一个示例 ARN。管道在存储库中发生更改时启动。

```
aws events put-targets --rule MyS3SourceRule --targets
 Id=1,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
```

编辑管道的 PollForSourceChanges参数

> **Important**
>
> 使用此方法创建管道时，如果 PollForSourceChanges 参数未明确设置为 false，则默认为 true。添加基于事件的更改检测时，必须将参数添加到输出并将其设置为 false 以禁用轮询。否则，您的管道将针对单个源更改启动两次。有关更多信息，请参阅 PollForSourceChanges 参数的默认设置。

1. 运行 get-pipeline 命令以将管道结构复制到 JSON 文件中。例如，对于名为 MyFirstPipeline 的管道，运行以下命令：

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   该命令不会返回任何结果，但您创建的文件将出现在您运行命令所在的目录中。

2. 在任何纯文本编辑器中打开 JSON 文件并通过将名为 storage-bucket 的存储桶的 PollForSourceChanges 参数更改为 false 来编辑源阶段，如此示例中所示。

   我为何做出此更改？ 将此参数设置为 false 将关闭定期检查，因此您只能使用基于事件的更改检测。

   ```
   "configuration": {
       "S3Bucket": "storage-bucket",
       "PollForSourceChanges": "false",
       "S3ObjectKey": "index.zip"
   },
   ```

3. 如果您要使用通过 get-pipeline 命令检索到的管道结构，则必须删除 JSON 文件中的 metadata 行。否则，update-pipeline 命令无法使用它。删除 "metadata": { } 行以及 "created"、"pipelineARN" 和 "updated" 字段。

   例如，从结构中删除以下各行：

   ```
   "metadata": {
       "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
       "created": "date",
       "updated": "date"
   },
   ```

保存该文件。

4. 要应用更改，请运行 update-pipeline 命令，指定管道 JSON 文件：

> **Important**
>
> 务必在文件名前包含 `file://`。此命令中需要该项。

```
aws codepipeline update-pipeline --cli-input-json file://pipeline.json
```

该命令会返回编辑后的管道的整个结构。

> **Note**
>
> update-pipeline 命令会停止管道。如果在运行 update-pipeline 命令时正在通过管道运行修订，则该运行会被停止。您必须手动启动管道，通过升级后的管道运行此修订。使用 start-pipeline-execution 命令手动启动您的管道。

使用 S3 源和 CloudTrail 跟踪（AWS CloudFormation 模板）迁移轮询管道

使用以下步骤，将具有 Amazon S3 源的管道从轮询编辑为基于事件的更改检测。

要构建具有 Amazon S3 的事件驱动管道，您应编辑管道的 `PollForSourceChanges` 参数，然后将以下资源添加到您的模板：

- EventBridge 要求必须记录所有 Amazon S3 事件。必须创建 AWS CloudTrail 跟踪、桶和桶策略，以便 Amazon S3 用于记录发生的事件。有关更多信息，请参阅记录数据事件以用于跟踪和记录管理事件以用于跟踪。
- EventBridge 规则和 IAM 角色以允许此事件启动我们的管道。

如果您 AWS CloudFormation 使用创建和管理管道，则您的模板将包含以下内容。

> **Note**
>
> 源阶段中的 Configuration 属性称为 PollForSourceChanges。如果您的模板未包含该属性，PollForSourceChanges 在默认情况下将设置为 true。

YAML

```
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      RoleArn: !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: S3
              OutputArtifacts:
                -
                  Name: SourceOutput
              Configuration:
                S3Bucket: !Ref SourceBucket
                S3ObjectKey: !Ref S3SourceObjectKey
                PollForSourceChanges: true
              RunOrder: 1


...
```

JSON

```
       "AppPipeline": {
           "Type": "AWS::CodePipeline::Pipeline",
           "Properties": {
               "RoleArn": {
                   "Fn::GetAtt": ["CodePipelineServiceRole", "Arn"]
```

```
                },
                "Stages": [
                    {
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "AWS",
                                    "Version": 1,
                                    "Provider": "S3"
                                },
                                "OutputArtifacts": [
                                    {
                                        "Name": "SourceOutput"
                                    }
                                ],
                                "Configuration": {
                                    "S3Bucket": {
                                        "Ref": "SourceBucket"
                                    },
                                    "S3ObjectKey": {
                                        "Ref": "SourceObjectKey"
                                    },
                                    "PollForSourceChanges": true
                                },
                                "RunOrder": 1
                            }
                        ]
                    },


...
```

创建以 Amazon S3 作为事件源和 CodePipeline 目标的 EventBridge 规则并应用权限策略

1. 在模板下的模板中`Resources`，使用`AWS::IAM::Role` AWS CloudFormation 资源配置允许您的事件启动管道的 IAM 角色。此条目将创建一个使用两个策略的角色：

   - 第一个策略允许代入角色。
   - 第二个策略提供启动管道所需的权限。

我为何做出此更改？ 添加AWS::IAM::Role资源可以 AWS CloudFormation 为创建权限 EventBridge。此资源已添加到您的 AWS CloudFormation 堆栈中。

YAML

```
  EventRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]


...
```

JSON

```
  "EventRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
```

```
          {
            "Effect": "Allow",
            "Principal": {
              "Service": [
                "events.amazonaws.com"
              ]
            },
            "Action": "sts:AssumeRole"
          }
        ]
      },
      "Path": "/",
      "Policies": [
        {
          "PolicyName": "eb-pipeline-execution",
          "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
              {
                "Effect": "Allow",
                "Action": "codepipeline:StartPipelineExecution",
                "Resource": {
                  "Fn::Join": [
                    "",
                    [
                      "arn:aws:codepipeline:",
                      {
                        "Ref": "AWS::Region"
                      },
                      ":",
                      {
                        "Ref": "AWS::AccountId"
                      },
                      ":",
                      {
                        "Ref": "AppPipeline"
                      }
                    ]
                  ]

...
```

2. 使用AWS::Events::Rule AWS CloudFormation 资源添加 EventBridge 规则。此事件模式会创建一个事件，以监控 Amazon S3 源桶上的 CopyObject、PutObject

和 CompleteMultipartUpload。此外，还包括您管道的目标。当发生 CopyObject、PutObject 或 CompleteMultipartUpload 时，此规则将对目标管道调用 StartPipelineExecution。

我为何做出此更改？ 添加AWS::Events::Rule资源 AWS CloudFormation 即可创建事件。此资源已添加到您的 AWS CloudFormation 堆栈中。

YAML

```
  EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventPattern:
        source:
          - aws.s3
        detail-type:
          - 'AWS API Call via CloudTrail'
        detail:
          eventSource:
            - s3.amazonaws.com
          eventName:
            - CopyObject
            - PutObject
            - CompleteMultipartUpload
          requestParameters:
            bucketName:
              - !Ref SourceBucket
            key:
              - !Ref SourceObjectKey
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline


...
```

JSON

```
"EventRule": {
  "Type": "AWS::Events::Rule",
  "Properties": {
    "EventPattern": {
      "source": [
        "aws.s3"
      ],
      "detail-type": [
        "AWS API Call via CloudTrail"
      ],
      "detail": {
        "eventSource": [
          "s3.amazonaws.com"
        ],
        "eventName": [
          "CopyObject",
          "PutObject",
          "CompleteMultipartUpload"
        ],
        "requestParameters": {
          "bucketName": [
            {
              "Ref": "SourceBucket"
            }
          ],
          "key": [
            {
              "Ref": "SourceObjectKey"
            }
          ]
        }
      }
    },
    "Targets": [
      {
        "Arn": {
          "Fn::Join": [
            "",
            [
              "arn:aws:codepipeline:",
```

```
                {
                  "Ref": "AWS::Region"
                },
                ":",
                {
                  "Ref": "AWS::AccountId"
                },
                ":",
                {
                  "Ref": "AppPipeline"
                }
              ]
            ]
          },
          "RoleArn": {
            "Fn::GetAtt": [
              "EventRole",
              "Arn"
            ]
          },
          "Id": "codepipeline-AppPipeline"
        }
      ]
    }
  }
},

...
```

3. 将此代码段添加到第一个模板，以允许跨堆栈功能：

YAML

```
Outputs:
  SourceBucketARN:
    Description: "S3 bucket ARN that Cloudtrail will use"
    Value: !GetAtt SourceBucket.Arn
    Export:
      Name: SourceBucketARN
```

JSON

```
  "Outputs" : {
```

```
    "SourceBucketARN" : {
      "Description" : "S3 bucket ARN that Cloudtrail will use",
      "Value" : { "Fn::GetAtt": ["SourceBucket", "Arn"] },
      "Export" : {
        "Name" : "SourceBucketARN"
      }
    }

...
```

4. 将更新后的模板保存到本地计算机上，然后打开 AWS CloudFormation 控制台。
5. 选择堆栈，然后选择为当前堆栈创建更改集。
6. 上传更新的模板，然后查看 AWS CloudFormation中所列的更改。以下是将对堆栈进行的更改。您应在列表中看到新资源。
7. 选择执行。

编辑管道的 PollForSourceChanges参数

> **Important**
>
> 使用此方法创建管道时，如果 PollForSourceChanges 参数未明确设置为 false，则默认为 true。添加基于事件的更改检测时，必须将参数添加到输出并将其设置为 false 以禁用轮询。否则，您的管道将针对单个源更改启动两次。有关更多信息，请参阅 PollForSourceChanges 参数的默认设置。

- 在模板中，将 PollForSourceChanges 更改为 false。如果您未在管道定义中包含 PollForSourceChanges，请添加它并将它设置为 false。

  我为何做出此更改？ 将 PollForSourceChanges 更改为 false 将关闭定期检查，因此您只能使用基于事件的更改检测。

  YAML

```
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
```

```
            Owner: AWS
            Version: 1
            Provider: S3
          OutputArtifacts:
            - Name: SourceOutput
          Configuration:
            S3Bucket: !Ref SourceBucket
            S3ObjectKey: !Ref SourceObjectKey
            PollForSourceChanges: false
          RunOrder: 1
```

JSON

```
{
   "Name": "SourceAction",
   "ActionTypeId": {
     "Category": "Source",
     "Owner": "AWS",
     "Version": 1,
     "Provider": "S3"
   },
   "OutputArtifacts": [
     {
       "Name": "SourceOutput"
     }
   ],
   "Configuration": {
     "S3Bucket": {
       "Ref": "SourceBucket"
     },
     "S3ObjectKey": {
       "Ref": "SourceObjectKey"
     },
     "PollForSourceChanges": false
   },
   "RunOrder": 1
 }
```

为您的 Amazon S3 管道 CloudTrail 资源创建第二个模板

- 在单独的模板中Resources，使用AWS::S3::BucketAWS::S3::BucketPolicy、和AWS::CloudTrail::Trail AWS CloudFormation 资源为其提供简单的存储桶定义和跟踪 CloudTrail。

  我为何做出此更改？ 鉴于当前每个账户只能有五条 CloudTrail 跟踪，因此必须单独创建和管理跟踪。（参见中的限制 AWS CloudTrail。）但是，您可以在单个跟踪上包括许多 Amazon S3 桶，因此您可以创建跟踪一次，然后根据需要为其他管道添加 Amazon S3 桶。将以下内容粘贴到第二个示例模板文件中。

  YAML

```
######################################################################################
# Prerequisites:
#   - S3 SourceBucket and SourceObjectKey must exist
######################################################################################

Parameters:
  SourceObjectKey:
    Description: 'S3 source artifact'
    Type: String
    Default: SampleApp_Linux.zip

Resources:
  AWSCloudTrailBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref AWSCloudTrailBucket
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Sid: AWSCloudTrailAclCheck
            Effect: Allow
            Principal:
              Service:
                - cloudtrail.amazonaws.com
            Action: s3:GetBucketAcl
            Resource: !GetAtt AWSCloudTrailBucket.Arn
          -
            Sid: AWSCloudTrailWrite
```

```
            Effect: Allow
            Principal:
              Service:
                - cloudtrail.amazonaws.com
            Action: s3:PutObject
            Resource: !Join [ '', [ !GetAtt AWSCloudTrailBucket.Arn, '/
AWSLogs/', !Ref 'AWS::AccountId', '/*' ] ]
            Condition:
              StringEquals:
                s3:x-amz-acl: bucket-owner-full-control
  AWSCloudTrailBucket:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
  AwsCloudTrail:
    DependsOn:
      - AWSCloudTrailBucketPolicy
    Type: AWS::CloudTrail::Trail
    Properties:
      S3BucketName: !Ref AWSCloudTrailBucket
      EventSelectors:
        -
          DataResources:
            -
              Type: AWS::S3::Object
              Values:
                - !Join [ '', [ !ImportValue SourceBucketARN, '/', !Ref
 SourceObjectKey ] ]
          ReadWriteType: WriteOnly
          IncludeManagementEvents: false
      IncludeGlobalServiceEvents: true
      IsLogging: true
      IsMultiRegionTrail: true


...
```

JSON

```
{
  "Parameters": {
    "SourceObjectKey": {
      "Description": "S3 source artifact",
      "Type": "String",
```

```
      "Default": "SampleApp_Linux.zip"
    }
  },
  "Resources": {
    "AWSCloudTrailBucket": {
      "Type": "AWS::S3::Bucket",
        "DeletionPolicy": "Retain"
    },
    "AWSCloudTrailBucketPolicy": {
      "Type": "AWS::S3::BucketPolicy",
      "Properties": {
        "Bucket": {
          "Ref": "AWSCloudTrailBucket"
        },
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "AWSCloudTrailAclCheck",
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  "cloudtrail.amazonaws.com"
                ]
              },
              "Action": "s3:GetBucketAcl",
              "Resource": {
                "Fn::GetAtt": [
                  "AWSCloudTrailBucket",
                  "Arn"
                ]
              }
            },
            {
              "Sid": "AWSCloudTrailWrite",
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  "cloudtrail.amazonaws.com"
                ]
              },
              "Action": "s3:PutObject",
              "Resource": {
                "Fn::Join": [
```

```
                "",
                [
                  {
                    "Fn::GetAtt": [
                      "AWSCloudTrailBucket",
                      "Arn"
                    ]
                  },
                  "/AWSLogs/",
                  {
                    "Ref": "AWS::AccountId"
                  },
                  "/*"
                ]
              ]
            },
            "Condition": {
              "StringEquals": {
                "s3:x-amz-acl": "bucket-owner-full-control"
              }
            }
          }
        ]
      }
    }
  },
  "AwsCloudTrail": {
    "DependsOn": [
      "AWSCloudTrailBucketPolicy"
    ],
    "Type": "AWS::CloudTrail::Trail",
    "Properties": {
      "S3BucketName": {
        "Ref": "AWSCloudTrailBucket"
      },
      "EventSelectors": [
        {
          "DataResources": [
            {
              "Type": "AWS::S3::Object",
              "Values": [
                {
                  "Fn::Join": [
                    "",
```

```
                    [
                      {
                        "Fn::ImportValue": "SourceBucketARN"
                      },
                      "/",
                      {
                        "Ref": "SourceObjectKey"
                      }
                    ]
                  ]
                }
              ]
            }
          ],
          "ReadWriteType": "WriteOnly",
          "IncludeManagementEvents": false
        }
      ],
      "IncludeGlobalServiceEvents": true,
      "IsLogging": true,
      "IsMultiRegionTrail": true
    }
  }
}

...
```

Example

当您使用 AWS CloudFormation 创建这些资源时，系统会在创建或更新存储库中的文件时触发您的管道。

> **Note**
>
> 请勿在此处停止。尽管您的管道已创建，但您必须为 Amazon S3 管道创建第二个 AWS CloudFormation 模板。如果您不创建第二个模板，您的管道不会有任何更改检测功能。

YAML

```
Resources:
```

```
  SourceBucket:
    Type: AWS::S3::Bucket
    Properties:
      VersioningConfiguration:
        Status: Enabled
  CodePipelineArtifactStoreBucket:
    Type: AWS::S3::Bucket
  CodePipelineArtifactStoreBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref CodePipelineArtifactStoreBucket
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Sid: DenyUnEncryptedObjectUploads
            Effect: Deny
            Principal: '*'
            Action: s3:PutObject
            Resource: !Join [ '', [ !GetAtt CodePipelineArtifactStoreBucket.Arn, '/
*' ] ]
            Condition:
              StringNotEquals:
                s3:x-amz-server-side-encryption: aws:kms
          -
            Sid: DenyInsecureConnections
            Effect: Deny
            Principal: '*'
            Action: s3:*
            Resource: !Join [ '', [ !GetAtt CodePipelineArtifactStoreBucket.Arn, '/
*' ] ]
            Condition:
              Bool:
                aws:SecureTransport: false
  CodePipelineServiceRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
```

```
            - codepipeline.amazonaws.com
        Action: sts:AssumeRole
    Path: /
    Policies:
      -
        PolicyName: AWS-CodePipeline-Service-3
        PolicyDocument:
          Version: 2012-10-17
          Statement:
            -
              Effect: Allow
              Action:
                - codecommit:CancelUploadArchive
                - codecommit:GetBranch
                - codecommit:GetCommit
                - codecommit:GetUploadArchiveStatus
                - codecommit:UploadArchive
              Resource: 'resource_ARN'
            -
              Effect: Allow
              Action:
                - codedeploy:CreateDeployment
                - codedeploy:GetApplicationRevision
                - codedeploy:GetDeployment
                - codedeploy:GetDeploymentConfig
                - codedeploy:RegisterApplicationRevision
              Resource: 'resource_ARN'
            -
              Effect: Allow
              Action:
                - codebuild:BatchGetBuilds
                - codebuild:StartBuild
              Resource: 'resource_ARN'
            -
              Effect: Allow
              Action:
                - devicefarm:ListProjects
                - devicefarm:ListDevicePools
                - devicefarm:GetRun
                - devicefarm:GetUpload
                - devicefarm:CreateUpload
                - devicefarm:ScheduleRun
              Resource: 'resource_ARN'
            -
```

```
                Effect: Allow
                Action:
                  - lambda:InvokeFunction
                  - lambda:ListFunctions
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
                  - iam:PassRole
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
                  - elasticbeanstalk:*
                  - ec2:*
                  - elasticloadbalancing:*
                  - autoscaling:*
                  - cloudwatch:*
                  - s3:*
                  - sns:*
                  - cloudformation:*
                  - rds:*
                  - sqs:*
                  - ecs:*
                Resource: 'resource_ARN'
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: s3-events-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: S3
              OutputArtifacts:
                - Name: SourceOutput
```

```
            Configuration:
              S3Bucket: !Ref SourceBucket
              S3ObjectKey: !Ref SourceObjectKey
              PollForSourceChanges: false
            RunOrder: 1
      -
        Name: Beta
        Actions:
          -
            Name: BetaAction
            InputArtifacts:
              - Name: SourceOutput
            ActionTypeId:
              Category: Deploy
              Owner: AWS
              Version: 1
              Provider: CodeDeploy
            Configuration:
              ApplicationName: !Ref ApplicationName
              DeploymentGroupName: !Ref BetaFleet
            RunOrder: 1
    ArtifactStore:
      Type: S3
      Location: !Ref CodePipelineArtifactStoreBucket
EventRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: 2012-10-17
      Statement:
        -
          Effect: Allow
          Principal:
            Service:
              - events.amazonaws.com
          Action: sts:AssumeRole
    Path: /
    Policies:
      -
        PolicyName: eb-pipeline-execution
        PolicyDocument:
          Version: 2012-10-17
          Statement:
            -
```

```
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region',
 ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
  EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventPattern:
        source:
          - aws.s3
        detail-type:
          - 'AWS API Call via CloudTrail'
        detail:
          eventSource:
            - s3.amazonaws.com
          eventName:
            - PutObject
            - CompleteMultipartUpload
          resources:
            ARN:
              - !Join [ '', [ !GetAtt SourceBucket.Arn, '/', !Ref
 SourceObjectKey ] ]
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline

Outputs:
  SourceBucketARN:
    Description: "S3 bucket ARN that Cloudtrail will use"
    Value: !GetAtt SourceBucket.Arn
    Export:
      Name: SourceBucketARN
```

JSON

```
    "Resources": {
        "SourceBucket": {
            "Type": "AWS::S3::Bucket",
            "Properties": {
```

```
                "VersioningConfiguration": {
                    "Status": "Enabled"
                }
            }
        },
        "CodePipelineArtifactStoreBucket": {
            "Type": "AWS::S3::Bucket"
        },
        "CodePipelineArtifactStoreBucketPolicy": {
            "Type": "AWS::S3::BucketPolicy",
            "Properties": {
                "Bucket": {
                    "Ref": "CodePipelineArtifactStoreBucket"
                },
                "PolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Sid": "DenyUnEncryptedObjectUploads",
                            "Effect": "Deny",
                            "Principal": "*",
                            "Action": "s3:PutObject",
                            "Resource": {
                                "Fn::Join": [
                                    "",
                                    [
                                        {
                                            "Fn::GetAtt": [
                                                "CodePipelineArtifactStoreBucket",
                                                "Arn"
                                            ]
                                        },
                                        "/*"
                                    ]
                                ]
                            },
                            "Condition": {
                                "StringNotEquals": {
                                    "s3:x-amz-server-side-encryption": "aws:kms"
                                }
                            }
                        },
                        {
                            "Sid": "DenyInsecureConnections",
```

```
                        "Effect": "Deny",
                        "Principal": "*",
                        "Action": "s3:*",
                        "Resource": {
                            "Fn::Join": [
                                "",
                                [
                                    {
                                        "Fn::GetAtt": [
                                            "CodePipelineArtifactStoreBucket",
                                            "Arn"
                                        ]
                                    },
                                    "/*"
                                ]
                            ]
                        },
                        "Condition": {
                            "Bool": {
                                "aws:SecureTransport": false
                            }
                        }
                    }
                ]
            }
        }
    },
    "CodePipelineServiceRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": [
                                "codepipeline.amazonaws.com"
                            ]
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
```

```
                "Path": "/",
                "Policies": [
                    {
                        "PolicyName": "AWS-CodePipeline-Service-3",
                        "PolicyDocument": {
                            "Version": "2012-10-17",
                            "Statement": [
                                {
                                    "Effect": "Allow",
                                    "Action": [
                                        "codecommit:CancelUploadArchive",
                                        "codecommit:GetBranch",
                                        "codecommit:GetCommit",
                                        "codecommit:GetUploadArchiveStatus",
                                        "codecommit:UploadArchive"
                                    ],
                                    "Resource": "resource_ARN"
                                },
                                {
                                    "Effect": "Allow",
                                    "Action": [
                                        "codedeploy:CreateDeployment",
                                        "codedeploy:GetApplicationRevision",
                                        "codedeploy:GetDeployment",
                                        "codedeploy:GetDeploymentConfig",
                                        "codedeploy:RegisterApplicationRevision"
                                    ],
                                    "Resource": "resource_ARN"
                                },
                                {
                                    "Effect": "Allow",
                                    "Action": [
                                        "codebuild:BatchGetBuilds",
                                        "codebuild:StartBuild"
                                    ],
                                    "Resource": "resource_ARN"
                                },
                                {
                                    "Effect": "Allow",
                                    "Action": [
                                        "devicefarm:ListProjects",
                                        "devicefarm:ListDevicePools",
                                        "devicefarm:GetRun",
                                        "devicefarm:GetUpload",
```

```
                                "devicefarm:CreateUpload",
                                "devicefarm:ScheduleRun"
                            ],
                            "Resource": "resource_ARN"
                        },
                        {
                            "Effect": "Allow",
                            "Action": [
                                "lambda:InvokeFunction",
                                "lambda:ListFunctions"
                            ],
                            "Resource": "resource_ARN"
                        },
                        {
                            "Effect": "Allow",
                            "Action": [
                                "iam:PassRole"
                            ],
                            "Resource": "resource_ARN"
                        },
                        {
                            "Effect": "Allow",
                            "Action": [
                                "elasticbeanstalk:*",
                                "ec2:*",
                                "elasticloadbalancing:*",
                                "autoscaling:*",
                                "cloudwatch:*",
                                "s3:*",
                                "sns:*",
                                "cloudformation:*",
                                "rds:*",
                                "sqs:*",
                                "ecs:*"
                            ],
                            "Resource": "resource_ARN"
                        }
                    ]
                }
            }
        ]
    }
},
"AppPipeline": {
```

```
            "Type": "AWS::CodePipeline::Pipeline",
            "Properties": {
                "Name": "s3-events-pipeline",
                "RoleArn": {
                    "Fn::GetAtt": [
                        "CodePipelineServiceRole",
                        "Arn"
                    ]
                },
                "Stages": [
                    {
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "AWS",
                                    "Version": 1,
                                    "Provider": "S3"
                                },
                                "OutputArtifacts": [
                                    {
                                        "Name": "SourceOutput"
                                    }
                                ],
                                "Configuration": {
                                    "S3Bucket": {
                                        "Ref": "SourceBucket"
                                    },
                                    "S3ObjectKey": {
                                        "Ref": "SourceObjectKey"
                                    },
                                    "PollForSourceChanges": false
                                },
                                "RunOrder": 1
                            }
                        ]
                    },
                    {
                        "Name": "Beta",
                        "Actions": [
                            {
                                "Name": "BetaAction",
```

```
                            "InputArtifacts": [
                                {
                                    "Name": "SourceOutput"
                                }
                            ],
                            "ActionTypeId": {
                                "Category": "Deploy",
                                "Owner": "AWS",
                                "Version": 1,
                                "Provider": "CodeDeploy"
                            },
                            "Configuration": {
                                "ApplicationName": {
                                    "Ref": "ApplicationName"
                                },
                                "DeploymentGroupName": {
                                    "Ref": "BetaFleet"
                                }
                            },
                            "RunOrder": 1
                        }
                    ]
                }
            ],
            "ArtifactStore": {
                "Type": "S3",
                "Location": {
                    "Ref": "CodePipelineArtifactStoreBucket"
                }
            }
        }
    },
    "EventRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": [
                                "events.amazonaws.com"
                            ]
```

```
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "Path": "/",
            "Policies": [
                {
                    "PolicyName": "eb-pipeline-execution",
                    "PolicyDocument": {
                        "Version": "2012-10-17",
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Action": "codepipeline:StartPipelineExecution",
                                "Resource": {
                                    "Fn::Join": [
                                        "",
                                        [
                                            "arn:aws:codepipeline:",
                                            {
                                                "Ref": "AWS::Region"
                                            },
                                            ":",
                                            {
                                                "Ref": "AWS::AccountId"
                                            },
                                            ":",
                                            {
                                                "Ref": "AppPipeline"
                                            }
                                        ]
                                    ]
                                }
                            }
                        ]
                    }
                }
            ]
        }
    },
    "EventRule": {
        "Type": "AWS::Events::Rule",
        "Properties": {
```

```
                "EventPattern": {
                    "source": [
                        "aws.s3"
                    ],
                    "detail-type": [
                        "AWS API Call via CloudTrail"
                    ],
                    "detail": {
                        "eventSource": [
                            "s3.amazonaws.com"
                        ],
                        "eventName": [
                            "PutObject",
                            "CompleteMultipartUpload"
                        ],
                        "resources": {
                            "ARN": [
                                {
                                    "Fn::Join": [
                                        "",
                                        [
                                            {
                                                "Fn::GetAtt": [
                                                    "SourceBucket",
                                                    "Arn"
                                                ]
                                            },
                                            "/",
                                            {
                                                "Ref": "SourceObjectKey"
                                            }
                                        ]
                                    ]
                                }
                            ]
                        }
                    }
                },
                "Targets": [
                    {
                        "Arn": {
                            "Fn::Join": [
                                "",
                                [
```

```
                                    "arn:aws:codepipeline:",
                                    {
                                        "Ref": "AWS::Region"
                                    },
                                    ":",
                                    {
                                        "Ref": "AWS::AccountId"
                                    },
                                    ":",
                                    {
                                        "Ref": "AppPipeline"
                                    }
                                ]
                            ]
                        },
                        "RoleArn": {
                            "Fn::GetAtt": [
                                "EventRole",
                                "Arn"
                            ]
                        },
                        "Id": "codepipeline-AppPipeline"
                    }
                ]
            }
        }
    },
    "Outputs" : {
        "SourceBucketARN" : {
            "Description" : "S3 bucket ARN that Cloudtrail will use",
            "Value" : { "Fn::GetAtt": ["SourceBucket", "Arn"] },
            "Export" : {
                "Name" : "SourceBucketARN"
            }
        }
    }
}


...
```

将 GitHub 版本 1 源操作的轮询管道迁移到连接

您可以迁移 GitHub 版本 1 的源操作以使用外部存储库的连接。对于使用 GitHub版本 1 源操作的管道，这是推荐的变更检测方法。

对于具有 GitHub 版本 1 源操作的管道，我们建议将管道修改为使用 GitHub 版本 2 操作，以便自动进行更改检测 AWS CodeConnections。有关使用连接的更多信息，请参阅GitHub 连接。

创建与 GitHub （控制台）的连接

您可以使用控制台创建与的连接 GitHub。

步骤 1：替换版本 1 的 GitHub 操作

使用管道编辑页面将您的版本 1 GitHub 操作替换为版本 2 GitHub 操作。

替换您的版本 1 GitHub 操作

1. 登录 CodePipeline 控制台。
2. 选择管道，然后选择编辑。在源阶段中，选择编辑阶段。将显示一条消息，建议您更新操作。
3. 在操作提供者中，选择 GitHub （版本 2）。
4. 请执行以下操作之一：

   - 在 “连接” 下，如果您尚未创建与提供商的连接，请选择 “连接到” GitHub。继续步骤 2：创建与的连接 GitHub。
   - 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 3：保存连接的源操作。

步骤 2：创建与的连接 GitHub

选择创建连接后，将显示 “Connect t GitHub o” 页面。

要创建与的连接 GitHub

1. 在 “GitHub 连接设置” 下，您的连接名称显示在 “连接名称” 中。

   在 “GitHub 应用程序” 下，选择应用程序安装或选择 “安装新应用程序” 来创建一个。

> **Note**
>
> 您可以为与特定提供程序的所有连接安装一个应用程序。如果您已经安装了该 GitHub 应用程序，请选择它并跳过此步骤。

2. 如果 GitHub 显示的授权页面，请使用您的凭据登录，然后选择继续。
3. 在应用程序安装页面上，一条消息显示该 AWS CodeStar 应用程序正在尝试连接到您的 GitHub 帐户。

> **Note**
>
> 您只需为每个 GitHub 账户安装一次该应用程序。如果您之前已安装了应用程序，则可以选择配置，继续进入应用程序安装的修改页面，也可以使用后退按钮返回到控制台。

4. 在安装 AWS CodeStar 页面上，选择安装。
5. 在 “Connect t o GitHub” 页面上，将显示新安装的连接 ID。选择连接。

第 3 步：保存您的 GitHub 源代码操作

在编辑操作页面上完成更新以保存新的源操作。

保存您的 GitHub 源代码操作

1. 在存储库中，输入第三方存储库的名称。在分支中，输入您希望管道在其中检测源更改的分支。

> **Note**
>
> 在存储库中键入 `owner-name/repository-name`，如以下示例所示：
>
> ```
> my-account/my-repository
> ```

2. 在输出构件格式中，为构件选择格式。

   - 要使用默认方法存储 GitHub 操作的输出对象，请选择CodePipeline默认。该操作访问存储库中的文件，并将工件 GitHub 存储在管道工件存储区的 ZIP 文件中。
   - 要存储包含存储库的 URL 引用的 JSON 文件，以便下游操作可以直接执行 Git 命令，请选择完全克隆。此选项只能由 CodeBuild 下游操作使用。

如果选择此选项，则需要更新 CodeBuild 项目服务角色的权限，如所示添加连接 Bitbucket、GitHub、En GitHub terprise Server 或 GitLab .com 的 CodeBuild GitClone 权限。如需查看教程以了解如何使用完整克隆选项，请参阅教程：使用带有 GitHub 管道源的完整克隆。

3. 在输出构件中，可以保留此操作的输出对象的名称，例如 SourceArtifact。选择完成以关闭编辑操作页面。
4. 选择完成以关闭阶段编辑页面。选择保存以关闭管道编辑页面。

创建与 GitHub (CLI) 的连接

您可以使用 AWS Command Line Interface (AWS CLI) 创建与的连接 GitHub。

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 AWS CloudFormation，使用控制台编辑连接以使其处于状态AVAILABLE。

要创建与的连接 GitHub

1. 打开终端（Linux、macOS 或 Unix）或命令提示符（Windows）。 AWS CLI 使用运行create-connection命令，--connection-name为您的连接指定--provider-type和。在此示例中，第三方提供方名称为 GitHub，指定的连接名称为 MyConnection。

```
aws codeconnections create-connection --provider-type GitHub --connection-name
 MyConnection
```

如果成功，该命令将返回类似以下内容的连接 ARN 信息。

```
{
    "ConnectionArn": "arn:aws:codeconnections:us-west-2:account_id:connection/
aEXAMPLE-8aad-4d5d-8878-dfcab0bc441f"
}
```

2. 使用控制台完成连接。

将 GitHub 版本 1 源操作的轮询管道迁移到 webhook

您可以将管道迁移到使用 webhook 来检测 GitHub 源存储库中的更改。这种向 webhook 的迁移仅适用于 GitHub 版本 1 的操作。

- 控制台：将轮询管道迁移到 webhook（GitHub 版本 1 源操作）（控制台）
- CLI：将轮询管道迁移到 Webhook（GitHub 版本 1 源操作）(CLI)
- AWS CloudFormation: 更新推送事件的管道（GitHub 版本 1 源操作）（AWS CloudFormation 模板）

将轮询管道迁移到 webhook（GitHub 版本 1 源操作）（控制台）

您可以使用 CodePipeline 控制台更新管道，以便使用 webhook 来检测 CodeCommit 源存储库中的更改。

按照以下步骤编辑 EventBridge 改用轮询（定期检查）的管道。如果要创建管道，请参阅在中创建管道 CodePipeline。

当您使用控制台时，您的管道的 `PollForSourceChanges` 参数将为您更改。 GitHub Webhook 已为您创建并注册。

编辑管道源阶段

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   将显示与您的 AWS 账户关联的所有管道的名称。
2. 在 Name 中，选择您要编辑的管道的名称。这将打开管道的详细视图，包括管道每个阶段中每个操作的状态。
3. 在管道详细信息页中，选择编辑。
4. 在 Edit stage (编辑阶段) 中，选择源操作上的编辑图标。
5. 展开 “更改检测选项”，然后选择 “使用 Amazon CloudWatch Events”，以便在发生更改时自动启动我的管道（推荐）。

   屏幕上会显示一条消息，提示您 CodePipeline 创建一个 webhook GitHub 以检测源代码更改： AWS CodePipeline 将为您创建 webhook。可以在以下选项中选择退出。选择更新。除了 webhook 之外，还 CodePipeline 会创建以下内容：

- 随机生成并用于授权连接的密钥 GitHub。
- Webhook URL，使用该区域的公有终端节点生成。

CodePipeline 向注册 webhook。 GitHub这将订阅 URL 以接收存储库事件。

6. 编辑完您的管道后，请选择 Save pipeline changes 以返回到摘要页面。

将出现一条消息，显示要为您的管道创建的 Webhook 的名称。选择 保存并继续。

7. 要测试您的操作，请使用发布更改，将更改提交 AWS CLI 到管道源阶段中指定的源。

将轮询管道迁移到 Webhook（GitHub 版本 1 源操作）(CLI)

执行以下步骤来编辑正在使用定期检查的管道，以改用 Webhook。如果要创建管道，请参阅在中创建管道 CodePipeline。

要构建一个事件驱动的管道，您应编辑管道的 `PollForSourceChanges` 参数，然后手动创建以下资源：

- GitHub webhook 和授权参数

创建和注册您的 Webhook

> **Note**
>
> 使用 CLI 或 AWS CloudFormation 创建管道并添加 webhook 时，必须禁用定期检查。要禁用定期检查，您必须明确添加该 `PollForSourceChanges` 参数并将其设置为 false，有关详细信息，请参阅下面的最终过程。否则，CLI 或 AWS CloudFormation 管道的默认`PollForSourceChanges`值为默认为 true，并且不会显示在管道结构输出中。有关 PollForSourceChanges 默认值的更多信息，请参阅 PollForSourceChanges 参数的默认设置。

1. 在文本编辑器中，创建并保存您要创建的 Webhook 的 JSON 文件。为名为 `my-webhook` 的 Webhook 使用此示例文件：

```
{
    "webhook": {
        "name": "my-webhook",
  "targetPipeline": "pipeline_name",
```

```
  "targetAction": "source_action_name",
  "filters": [{
      "jsonPath": "$.ref",
      "matchEquals": "refs/heads/{Branch}"
  }],
  "authentication": "GITHUB_HMAC",
  "authenticationConfiguration": {
      "SecretToken": "secret"
  }
    }
}
```

2. 调用 put-webhook 命令并包含 `--cli-input` 和 `--region` 参数。

   以下示例命令使用 webhook_json JSON 文件创建一个 Webhook。

```
aws codepipeline put-webhook --cli-input-json file://webhook_json.json --region
 "eu-central-1"
```

3. 在此示例中显示的输出中，返回了名为 my-webhook 的 Webhook 的 URL 和 ARN。

```
{
    "webhook": {
        "url": "https://webhooks.domain.com/
trigger111111111EXAMPLE11111111111111111",
        "definition": {
            "authenticationConfiguration": {
                "SecretToken": "secret"
            },
            "name": "my-webhook",
            "authentication": "GITHUB_HMAC",
            "targetPipeline": "pipeline_name",
            "targetAction": "Source",
            "filters": [
                {
                    "jsonPath": "$.ref",
                    "matchEquals": "refs/heads/{Branch}"
                }
            ]
        },
        "arn": "arn:aws:codepipeline:eu-central-1:ACCOUNT_ID:webhook:my-webhook"
    },
    "tags": [{
```

```
      "key": "Project",
      "value": "ProjectA"
    }]
}
```

此示例通过为 Webhook 包含 `Project` 标签键和 `ProjectA` 值来为 Webhook 添加标记。有关在中为资源添加标签的更多信息 CodePipeline，请参阅标记资源。

4. 调用 register-webhook-with-third-party 命令并包含 `--webhook-name` 参数。

   以下示例命令注册名为 `my-webhook` 的 Webhook。

```
aws codepipeline register-webhook-with-third-party --webhook-name my-webhook
```

编辑管道的 PollForSourceChanges参数

> **Important**
>
> 使用此方法创建管道时，如果 `PollForSourceChanges` 参数未明确设置为 false，则默认为 true。添加基于事件的更改检测时，必须将参数添加到输出并将其设置为 false 以禁用轮询。否则，您的管道将针对单个源更改启动两次。有关更多信息，请参阅 PollForSourceChanges 参数的默认设置。

1. 运行 get-pipeline 命令以将管道结构复制到 JSON 文件中。例如，对于名为 `MyFirstPipeline` 的管道，可以键入以下命令：

```
aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
```

   该命令不会返回任何结果，但您创建的文件将出现在您运行命令所在的目录中。

2. 在任何纯文本编辑器中打开 JSON 文件，并通过更改或添加 `PollForSourceChanges` 参数来编辑源阶段。在此示例中，对于名为 `UserGitHubRepo` 的存储库，该参数设置为 `false`。

   我为何做出此更改？ 更改此参数将关闭定期检查，因此您只能使用基于事件的更改检测。

```
"configuration": {
    "Owner": "name",
    "Repo": "UserGitHubRepo",
    "PollForSourceChanges": "false",
```

```
    "Branch": "main",
    "OAuthToken": "****"
},
```

3. 如果您使用的是通过 get-pipeline 命令检索到的管道结构，则必须通过从文件中删除 metadata 行来编辑 JSON 文件中的结构。否则，update-pipeline 命令无法使用它。从 JSON 文件中的管道结构中删除 "metadata" 部分，包括：{ } 以及 "created"、"pipelineARN" 和 "updated" 字段。

   例如，从结构中删除以下各行：

```
"metadata": {
    "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
    "created": "date",
    "updated": "date"
},
```

   保存该文件。

4. 要应用更改，请运行 update-pipeline 命令并指定一个管道 JSON 文件，类似于以下内容：

> **Important**
>
> 务必在文件名前包含 file://。此命令中需要该项。

```
aws codepipeline update-pipeline --cli-input-json file://pipeline.json
```

该命令会返回编辑后的管道的整个结构。

> **Note**
>
> update-pipeline 命令会停止管道。如果在运行 update-pipeline 命令时正在通过管道运行修订，则该运行会被停止。您必须手动启动管道，通过升级后的管道运行此修订。使用 start-pipeline-execution 命令手动启动您的管道。

更新推送事件的管道（GitHub 版本 1 源操作）（AWS CloudFormation 模板）

按照以下步骤将您的管道（带有 GitHub 来源）从定期检查（轮询）更新为使用 webhook 的基于事件的变更检测。

要使用构建事件驱动的管道 AWS CodeCommit，请编辑管道的`PollForSourceChanges`参数，然后向模板中添加 GitHub webhook 资源。

如果您 AWS CloudFormation 使用创建和管理管道，则模板的内容如下所示。

> **Note**
>
> 请注意源阶段中的 `PollForSourceChanges` 配置属性。如果您的模板未包含该属性，`PollForSourceChanges` 在默认情况下将设置为 `true`。

YAML

```
Resources:
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: github-polling-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: ThirdParty
                Version: 1
                Provider: GitHub
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                Owner: !Ref GitHubOwner
                Repo: !Ref RepositoryName
                Branch: !Ref BranchName
```

```
                OAuthToken:
 {{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
 PollForSourceChanges: true
             RunOrder: 1


...
```

JSON

```
        "AppPipeline": {
            "Type": "AWS::CodePipeline::Pipeline",
            "Properties": {
                "Name": "github-polling-pipeline",
                "RoleArn": {
                    "Fn::GetAtt": [
                        "CodePipelineServiceRole",
                        "Arn"
                    ]
                },
                "Stages": [
                    {
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "ThirdParty",
                                    "Version": 1,
                                    "Provider": "GitHub"
                                },
                                "OutputArtifacts": [
                                    {
                                        "Name": "SourceOutput"
                                    }
                                ],
                                "Configuration": {
                                    "Owner": {
                                        "Ref": "GitHubOwner"
                                    },
                                    "Repo": {
                                        "Ref": "RepositoryName"
```

```
                                    },
                                    "Branch": {
                                        "Ref": "BranchName"
                                    },
                                    "OAuthToken":
"{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}",
                                    "PollForSourceChanges": true
                                },
                                "RunOrder": 1
                            }
                        ]
                    },


...
```

在模板中添加参数并创建 Webhook

我们强烈建议您使用 AWS Secrets Manager 来存储您的证书。如果使用 Secrets Manager，则您必须已在 Secrets Manager 中配置并存储密钥参数。此示例使用对 Secrets Manager 的动态引用作为您的 webhook 的 GitHub凭证。有关更多信息，请参阅使用动态引用以指定模板值。

Important

传递密钥参数时，请勿直接将值输入到模板中。该值以明文形式提供，因此是可读的。出于安全考虑，请勿在 AWS CloudFormation 模板中使用纯文本来存储您的证书。

使用 CLI 或 AWS CloudFormation 创建管道并添加 webhook 时，必须禁用定期检查。

Note

要禁用定期检查，您必须明确添加该 `PollForSourceChanges` 参数并将其设置为 false，有关详细信息，请参阅下面的最终过程。否则，CLI 或 AWS CloudFormation 管道的默认`PollForSourceChanges`值为默认为 true，并且不会显示在管道结构输出中。有关 PollForSourceChanges 默认值的更多信息，请参阅 PollForSourceChanges 参数的默认设置。

1. 在模板中的 `Resources` 下，添加您的参数：

YAML

```
Parameters:
        GitHubOwner:
          Type: String

...
```

JSON

```
{
 "Parameters": {
  "BranchName": {
   "Description": "GitHub branch name",
   "Type": "String",
   "Default": "main"
  },
  "GitHubOwner": {
   "Type": "String"
  },

...
```

2. 使用该AWS::CodePipeline::Webhook AWS CloudFormation 资源添加 webhook。

> **Note**
>
> 您指定的 TargetAction 必须与管道中定义的源操作的 Name 属性匹配。

如果设置RegisterWithThirdParty为true，请确保与关联的用户OAuthToken可以在中设置所需的范围 GitHub。令牌和 webhook 需要以下 GitHub 范围：

- repo - 用于完全控制从公有和私有存储库读取项目并将项目提取到管道中的过程。
- admin:repo_hook - 用于完全控制存储库挂钩。

否则， GitHub 返回 404。有关返回的 404 的更多信息，请参阅https://help.github.com/articles/about-webhooks。

YAML

```
 AppPipelineWebhook:
   Type: AWS::CodePipeline::Webhook
   Properties:
     Authentication: GITHUB_HMAC
     AuthenticationConfiguration:
       SecretToken:
{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
     Filters:
       -
         JsonPath: "$.ref"
         MatchEquals: refs/heads/{Branch}
     TargetPipeline: !Ref AppPipeline
     TargetAction: SourceAction
     Name: AppPipelineWebhook
     TargetPipelineVersion: !GetAtt AppPipeline.Version
     RegisterWithThirdParty: true


...
```

JSON

```
"AppPipelineWebhook": {
    "Type": "AWS::CodePipeline::Webhook",
    "Properties": {
        "Authentication": "GITHUB_HMAC",
        "AuthenticationConfiguration": {
            "SecretToken":
 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}"
        },
        "Filters": [{
            "JsonPath": "$.ref",
            "MatchEquals": "refs/heads/{Branch}"
        }],
        "TargetPipeline": {
            "Ref": "AppPipeline"
        },
        "TargetAction": "SourceAction",
        "Name": "AppPipelineWebhook",
        "TargetPipelineVersion": {
```

```
            "Fn::GetAtt": [
              "AppPipeline",
              "Version"
            ]
        },
        "RegisterWithThirdParty": true
    }
},

...
```

3. 将更新的模板保存到本地计算机，然后打开 AWS CloudFormation 控制台。
4. 选择堆栈，然后选择为当前堆栈创建更改集。
5. 上传模板，然后查看 AWS CloudFormation中列出的更改。这些是要对堆栈进行的更改。您应在列表中看到新资源。
6. 选择执行。

编辑管道的 PollForSourceChanges参数

> **Important**
>
> 使用此方法创建管道时，如果 `PollForSourceChanges` 参数未明确设置为 false，则默认为 true。添加基于事件的更改检测时，必须将参数添加到输出并将其设置为 false 以禁用轮询。否则，您的管道将针对单个源更改启动两次。有关更多信息，请参阅 PollForSourceChanges 参数的默认设置。

- 在模板中，将 `PollForSourceChanges` 更改为 `false`。如果您未在管道定义中包含 `PollForSourceChanges`，请添加它并将它设置为 false。

  我为何做出此更改？ 将此参数更改为 `false` 将关闭定期检查，因此您只能使用基于事件的更改检测。

  YAML

```
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
```

```
                Category: Source
                Owner: ThirdParty
                Version: 1
                Provider: GitHub
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                Owner: !Ref GitHubOwner
                Repo: !Ref RepositoryName
                Branch: !Ref BranchName
                OAuthToken:
{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
                PollForSourceChanges: false
              RunOrder: 1
```

JSON

```
{
   "Name": "Source",
   "Actions": [{
 "Name": "SourceAction",
 "ActionTypeId": {
     "Category": "Source",
     "Owner": "ThirdParty",
     "Version": 1,
     "Provider": "GitHub"
},
"OutputArtifacts": [{
    "Name": "SourceOutput"
}],
"Configuration": {
    "Owner": {
  "Ref": "GitHubOwner"
    },
    "Repo": {
  "Ref": "RepositoryName"
    },
    "Branch": {
        "Ref": "BranchName"
    },
    "OAuthToken":
"{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}",
    PollForSourceChanges: false
```

```
 },
 "RunOrder": 1
    }]
```

Example

使用创建这些资源时 AWS CloudFormation，定义的 webhook 将在指定的 GitHub 存储库中创建。您的管道将在提交时触发。

YAML

```
Parameters:
  GitHubOwner:
    Type: String

Resources:
  AppPipelineWebhook:
    Type: AWS::CodePipeline::Webhook
    Properties:
      Authentication: GITHUB_HMAC
      AuthenticationConfiguration:
        SecretToken: {{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
      Filters:
        -
          JsonPath: "$.ref"
          MatchEquals: refs/heads/{Branch}
      TargetPipeline: !Ref AppPipeline
      TargetAction: SourceAction
      Name: AppPipelineWebhook
      TargetPipelineVersion: !GetAtt AppPipeline.Version
      RegisterWithThirdParty: true
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: github-events-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
```

```
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: ThirdParty
                Version: 1
                Provider: GitHub
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                Owner: !Ref GitHubOwner
                Repo: !Ref RepositoryName
                Branch: !Ref BranchName
                OAuthToken:
 {{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
                PollForSourceChanges: false
              RunOrder: 1

...
```

JSON

```
{
    "Parameters": {
        "BranchName": {
            "Description": "GitHub branch name",
            "Type": "String",
            "Default": "main"
        },
        "RepositoryName": {
            "Description": "GitHub repository name",
            "Type": "String",
            "Default": "test"
        },
        "GitHubOwner": {
            "Type": "String"
        },
        "ApplicationName": {
            "Description": "CodeDeploy application name",
            "Type": "String",
            "Default": "DemoApplication"
        },
        "BetaFleet": {
            "Description": "Fleet configured in CodeDeploy",
```

```
            "Type": "String",
            "Default": "DemoFleet"
        }
    },
    "Resources": {

...

        },
        "AppPipelineWebhook": {
            "Type": "AWS::CodePipeline::Webhook",
            "Properties": {
                "Authentication": "GITHUB_HMAC",
                "AuthenticationConfiguration": {
                    "SecretToken": {

 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}"
                    }
                },
                "Filters": [
                    {
                        "JsonPath": "$.ref",
                        "MatchEquals": "refs/heads/{Branch}"
                    }
                ],
                "TargetPipeline": {
                    "Ref": "AppPipeline"
                },
                "TargetAction": "SourceAction",
                "Name": "AppPipelineWebhook",
                "TargetPipelineVersion": {
                    "Fn::GetAtt": [
                        "AppPipeline",
                        "Version"
                    ]
                },
                "RegisterWithThirdParty": true
            }
        },
        "AppPipeline": {
            "Type": "AWS::CodePipeline::Pipeline",
            "Properties": {
                "Name": "github-events-pipeline",
                "RoleArn": {
```

```
                    "Fn::GetAtt": [
                        "CodePipelineServiceRole",
                        "Arn"
                    ]
                },
                "Stages": [
                    {
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "ThirdParty",
                                    "Version": 1,
                                    "Provider": "GitHub"
                                },
                                "OutputArtifacts": [
                                    {
                                        "Name": "SourceOutput"
                                    }
                                ],
                                "Configuration": {
                                    "Owner": {
                                        "Ref": "GitHubOwner"
                                    },
                                    "Repo": {
                                        "Ref": "RepositoryName"
                                    },
                                    "Branch": {
                                        "Ref": "BranchName"
                                    },
                                    "OAuthToken":
 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}",
                                    "PollForSourceChanges": false
                                },
                                "RunOrder": 1

...
```

创建 CodePipeline 服务角色

创建管道时，您可以创建服务角色或使用现有服务角色。

您可以使用 CodePipeline 控制台或 AWS CLI 创建 CodePipeline 服务角色。创建管道需要服务角色，管道始终与该服务角色相关联。

在 AWS 使用 CLI 创建管道之前，必须为管道创建 CodePipeline 服务角色。有关指定了服务角色和策略的示例 AWS CloudFormation 模板，请参阅中的教程教程：创建使用 AWS CloudFormation 部署操作中的变量的管道。

服务角色不是 AWS 托管角色，而是最初为创建管道而创建的，然后在服务角色策略中添加新权限时，您可能需要更新管道的服务角色。使用服务角色创建管道后，您无法将不同的服务角色应用于该管道。将建议的策略附加到服务角色。

有关服务角色的更多信息，请参阅管理 CodePipeline 服务角色。

创建 CodePipeline 服务角色（控制台）

使用控制台创建管道时，您可以使用管道创建向导创建 CodePipeline 服务角色。

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   选择创建管道并完成管道创建向导中的步骤 1: 选择管道设置页面。

   > **Note**
   >
   > 创建管道后，便无法再更改其名称。有关其他限制的信息，请参阅中的配额 AWS CodePipeline。

2. 在服务角色中，选择新服务角色 CodePipeline 以允许在 IAM 中创建新的服务角色。
3. 完成管道创建。您的管道服务角色可在 IAM 角色列表中查看，您可以通过使用 AWS CLI 运行 `get-pipeline` 命令来查看与管道关联的服务角色 ARN。

创建 CodePipeline 服务角色 (CLI)

在 AWS 使用 CLI 或创建管道之前 AWS CloudFormation，必须为管道创建 CodePipeline服务角色并附加服务角色策略和信任策略。要使用 CLI 创建您的服务角色，请按照以下步骤首先在运行 CLI 命令的目录中创建信任策略 JSON 和角色策略 JSON 作为单独的文件。

> **Note**
>
> 我们建议您只允许管理用户创建任何服务角色。拥有创建角色和附加任何策略权限的人员可以升级自己的权限。相反，应创建一个策略，允许他们仅创建他们所需的角色，或者让管理员代表他们创建服务角色。

1. 在终端窗口中，输入以下命令创建一个名为 `TrustPolicy.json` 的文件，在其中粘贴角色策略 JSON。此示例使用 VIM。

```
vim TrustPolicy.json
```

2. 将以下 JSON 粘贴到文件中。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "codepipeline.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

要保存并退出文件，请输入以下 VIM 命令：

```
:wq
```

3. 在终端窗口中，输入以下命令创建一个名为 `RolePolicy.json` 的文件，在其中粘贴角色策略 JSON。此示例使用 VIM。

```
vim RolePolicy.json
```

4. 将以下 JSON 粘贴到文件中。在政策声明 `Resource` 字段中为您的管道添加 Amazon 资源名称 (ARN)，确保尽可能缩小权限范围。

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecommit:CancelUploadArchive",
        "codecommit:GetBranch",
        "codecommit:GetCommit",
        "codecommit:GetUploadArchiveStatus",
        "codecommit:UploadArchive"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "codedeploy:CreateDeployment",
        "codedeploy:GetApplicationRevision",
        "codedeploy:GetDeployment",
        "codedeploy:GetDeploymentConfig",
        "codedeploy:RegisterApplicationRevision"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:BatchGetBuilds",
        "codebuild:StartBuild"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "devicefarm:ListProjects",
        "devicefarm:ListDevicePools",
        "devicefarm:GetRun",
        "devicefarm:GetUpload",
        "devicefarm:CreateUpload",
        "devicefarm:ScheduleRun"
      ],
      "Resource": "*"
    },
```

```
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction",
        "lambda:ListFunctions"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "elasticbeanstalk:*",
        "ec2:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "cloudformation:*",
        "rds:*",
        "sqs:*",
        "ecs:*"
      ],
      "Resource": "resource_ARN"
    }
  ]
}
```

要保存并退出文件，请输入以下 VIM 命令：

```
:wq
```

5. 输入以下命令创建角色并附加信任角色策略。该策略名称格式通常与角色名称格式相同。此示例使用角色名称 MyRole 和作为单独文件创建的策略 TrustPolicy。

```
aws iam create-role --role-name MyRole --assume-role-policy-document file://
TrustPolicy.json
```

6. 输入以下命令创建角色策略并将其附加到角色。该策略名称格式通常与角色名称格式相同。此示例使用角色名称 MyRole 和作为单独文件创建的策略 MyRole。

```
aws iam put-role-policy --role-name MyRole --policy-name RolePolicy --policy-
document file://RolePolicy.json
```

7. 要查看创建的角色名称和信任策略，请为名为 MyRole 的角色输入以下命令：

```
aws iam get-role --role-name MyRole
```

8. 使用 CL AWS I 或创建管道时，请使用服务角色 ARN。 AWS CloudFormation

将管道标记为 CodePipeline

标签是与资源关联的键值对。 AWS 您可以在中将标签应用于您的管道 CodePipeline。有关 CodePipeline 资源标记、用例、标签键和值限制以及支持的资源类型的信息，请参阅标记资源。

您可以使用 CLI 在创建管道时指定标签。您可以使用控制台或 CLI 来添加或删除标签，以及更新管道中标签的值。您最多可以为每个管道添加 50 个标签。

主题

- 为管道添加标签（控制台）
- 为管道添加标签 (CLI)

为管道添加标签（控制台）

您可以使用控制台或 CLI 来标记资源。管道是唯一可以通过控制台或 CLI 管理的 CodePipeline资源。

主题

- 为管道添加标签（控制台）
- 查看管道的标签（控制台）
- 编辑管道的标签（控制台）
- 删除管道的标签（控制台）

为管道添加标签（控制台）

您可以使用控制台向现有管道添加标签。

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 Pipelines (管道) 页面上，选择您要向其添加标签的管道。
3. 从导航窗格中，选择设置。
4. 在 Pipeline tags (管道标签) 中，选择 Edit (编辑)。
5. 在键和值字段中，输入您要添加的每组标签的键/值对。（值字段为可选项。）例如，在键中，输入 **Project**。在值中，输入 **ProjectA**。
6. （可选）选择添加标签以添加多行并输入多个标签。
7. 选择 Submit (提交)。标签在管道设置下列出。

查看管道的标签（控制台）

您可以使用控制台列出现有管道的标签。

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 Pipelines (管道) 页面上，选择您要查看其标签的管道。
3. 从导航窗格中，选择设置。
4. 在 Pipeline tags (管道标签) 下，在 Key (键) 和 Value (值) 列下查看管道的标签。

编辑管道的标签（控制台）

您可以使用控制台来编辑已添加到管道的标签。

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 Pipelines (管道) 页面上，选择您要更新其标签的管道。
3. 从导航窗格中，选择设置。
4. 在 Pipeline tags (管道标签) 中，选择 Edit (编辑)。
5. 在键和值字段中，根据需要更新每个字段的值。例如，对于 **Project** 键，在值中，将 **ProjectA** 更改为 **ProjectB**。

6. 选择 Submit (提交)。

删除管道的标签（控制台）

您可以使用控制台从管道删除标签。当您移除关联资源的标签时，对应标签会被删除。

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 Pipelines (管道) 页面上，选择您要删除其标签的管道。
3. 从导航窗格中，选择设置。
4. 在 Pipeline tags (管道标签) 中，选择 Edit (编辑)。
5. 接下来，对于您要删除的每个标签的键和值，选择删除标签。
6. 选择提交。

为管道添加标签 (CLI)

您可以使用 CLI 来标记资源。您必须使用控制台管理管道中的标签。

主题

- 为管道添加标签 (CLI)
- 查看管道的标签 (CLI)
- 编辑管道的标签 (CLI)
- 删除管道的标签 (CLI)

为管道添加标签 (CLI)

您可以使用控制台或 AWS CLI 来标记管道。

要在创建管道时向其添加标签，请参阅在中创建管道 CodePipeline。

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。

在终端或命令行运行 tag-resource 命令，指定要为其添加标签的管道的 Amazon 资源名称 (ARN)，以及要添加的标签的键/值。您可以为管道添加多个标签。例如，要*MyPipeline*使用两个标签标记名为

管道，一个标签键名为 Test，标签键名为 *DeploymentEnvironment*T *est*，标签键名为 true，标签键名为 *IscontainerBasedtrue*：

```
aws codepipeline tag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:MyPipeline --tags key=Project,value=ProjectA key=IscontainerBased,value=true
```

如果成功，该命令不返回任何内容。

查看管道的标签 (CLI)

按照以下步骤使用 AWS CLI 查看管道的 AWS 标签。如果尚未添加标签，则返回的列表为空。

在终端或命令行中，运行 list-tags-for-resource 命令。例如，要查看*MyPipeline*以 arn:aws:codepipeline:*us-west-2*:*account-id*:*MyPipeline* ARN 值命名的管道的标签键和标签值列表，请执行以下操作：

```
aws codepipeline list-tags-for-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:MyPipeline
```

如果成功，该命令返回类似以下内容的信息：

```
{
    "tags": {
        "Project": "ProjectA",
        "IscontainerBased": "true"
    }
}
```

编辑管道的标签 (CLI)

按照以下步骤 AWS CLI 使用编辑管道的标签。您可以更改现有键的值或添加另一个键。您还可以删除管道的标签，如下一节所示。

在终端或命令行运行 tag-resource 命令，指定要为其更新标签的管道的 ARN 并指定标签键和标签值：

```
aws codepipeline tag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:MyPipeline --tags key=Project,value=ProjectA
```

如果成功，该命令不返回任何内容。

删除管道的标签 (CLI)

按照以下步骤使用从 AWS CLI 管道中移除标签。当您移除关联资源的标签时，对应标签会被删除。

> **Note**
>
> 如果您删除某个管道，则会从已删除管道中移除关联的所有标签。您无需在删除该管道之前移除标签。

在终端或命令行运行 untag-resource 命令，指定要从中删除标签的管道的 ARN 以及要删除的标签的标签键。例如，要在名为 Project 的管道上删除多个标签 *MyPipeline*，标签键为 P *rojec* t 和 *IscontainerBased*：

```
aws codepipeline untag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:MyPipeline --tag-keys Project IscontainerBased
```

如果成功，该命令不返回任何内容。要验证与管道关联的标签，请运行 list-tags-for-resource 命令。

创建通知规则

您可以使用通知规则来向用户通知重要更改，例如在管道开始执行时。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 CodePipeline。

创建通知规则（控制台）

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。
2. 选择 Pipelines (管道)，然后选择要在其中添加通知的管道。
3. 在管道页面上，选择 Notify (通知)，然后选择 Create notification rule (创建通知规则)。您也可以转到管道的 Settings (设置) 页面，然后选择 Create notification rule (创建通知规则)。
4. 在 Notification name（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 CodePipeline 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 Events that trigger notifications（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，请参阅管道的通知规则的事件。
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 AWS Chatbot (Slack) 或 SNS 主题。在选择目标中，选择客户端名称（对于中配置的 Slack 客户端 AWS Chatbot）或亚马逊 SNS 主题的亚马逊资源名称 (ARN)（对于已经配置了通知所需策略的 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创建 AWS Chatbot 客户端。如果您选择 AWS Chatbot (Slack)，则会看到一个按钮，指示您在中 AWS Chatbot配置客户端。选择该选项将打开 AWS Chatbot 控制台。有关更多信息，请参阅配置通知和 AWS Chatbot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。您还可以设置通知之间的集成，并将通知发送 AWS Chatbot 到 Amazon Chime 聊天室或 Slack 频道。有关更多信息，请参阅配置通知和之间的集成 AWS Chatbot。

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 codestar-notifications 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。以下示例显示了名为 1234567890 *12* 的 AWS 账户*MyDemoPipeline*中名**MyNotificationRule**为管道的通知规则。管道执行开始时，通知将以完整详细信息类型发送到名为 *codestar-* notifications 的 Amazon SNS 主题：MyNotificationTopic

```
{
    "Name": "MyNotificationRule",
    "EventTypeIds": [
        "codepipeline-pipeline-pipeline-execution-started"
    ],
    "Resource": "arn:aws:codebuild:us-east-2:123456789012:MyDemoPipeline",
    "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 codestar-notifications 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"
}
```

在中使用触发器 CodePipeline

触发器允许您将管道配置为在特定事件类型或筛选的事件类型上启动，例如检测到特定分支或拉取请求的更改时。触发器可配置为使用中操作的连接的CodeStarSourceConnection源操作 CodePipeline GitHub，例如 Bitbucket 和 GitLab。

源操作（例如 CodeCommit 和 S3）使用变更检测，如本节中有关启动管道的详细介绍。

您可以向管道添加触发器，并将触发器配置为筛选特定事件

您可以使用控制台或 CLI 来指定触发器。

筛选代码推送或拉取请求的触发器

您可以为管道触发器配置过滤器，以便针对不同的 Git 事件（例如标签或分支推送、特定文件路径的更改、在特定分支中打开的拉取请求等）启动管道执行。您可以使用 AWS CodePipeline 控制台或中的create-pipeline和update-pipeline命令 AWS CLI 来配置触发器的过滤器。

您可以为以下触发器类型指定过滤器：

- Push

  当更改推送到源存储库时，推送触发器会启动管道。执行将使用来自你要推送到的分支（即目标分支）的提交。您可以根据分支、文件路径或 Git 标签筛选推送触发器。
- 拉取请求

  在源存储库中打开、更新或关闭拉取请求时，拉取请求触发器会启动管道。执行将使用你从源分支（即源分支）中提取的提交。您可以根据分支和文件路径筛选拉取请求触发器。

> **Note**
>
> 拉取请求支持的事件类型包括打开、更新或关闭（合并）。所有其他拉取请求事件都将被忽略。

管道定义允许您在相同的推送触发器配置中组合不同的过滤器。有关管道定义的详细信息，请参见在管道中触发筛选 JSON (CLI)。有效的组合是：

- 标签

- 分支
- 分支 + 文件路径

您可以按如下方式指定过滤器类型：

- 没有过滤器

  此触发器配置会在任何推送到作为操作配置一部分指定的默认分支时启动您的管道。

- 指定过滤器

  您可以添加一个过滤器，该过滤器在特定的过滤器（例如代码推送的分支名称）上启动管道，并获取确切的提交。这也将管道配置为不在发生任何更改时自动启动。

- 不要检测到更改

  这不会添加触发器，也不会在发生任何更改时自动启动管道。

下表为每种事件类型提供了有效的筛选选项。该表还显示了在操作配置中自动检测更改时，哪些触发器配置默认为 true 或 false。

触发器配置	事件类型	筛选条件选项	检测变化
添加触发器 — 无过滤器	none	none	true
添加触发器 — 在推送代码时进行筛选	推送事件	Git 标签、分支、文件路径	false
添加触发器 — 筛选拉取请求	拉取请求	分支、文件路径	false
没有触发器 — 不检测	none	none	false

> **Note**
>
> 此触发器类型使用自动更改检测（作为Webhook触发器类型）。使用此触发器类型的源操作提供程序是为代码推送配置的连接（Bitbucket Cloud GitHub、E GitHub nterprise Server、GitLab .com 和 GitLab自我管理）。

为了进行筛选，支持 glob 格式的正则表达式模式，详情请参见。使用语法中的 glob 模式

> **Note**
>
> 在某些情况下，对于具有按文件路径筛选的触发器的管道，当首次创建带有文件路径筛选器的分支时，管道可能无法启动。有关更多信息，请参阅 使用按文件路径进行触发器筛选的连接的管道可能不会在创建分支时启动。

主题

触发器过滤器的注意事项

使用触发器时，需要考虑以下注意事项。

- 对于带有分支和文件路径过滤器的触发器，首次推送分支时，管道将无法运行，因为无法访问为新创建的分支更改的文件列表。
- 如果推送（分支筛选器）和拉取请求（分支过滤器）触发配置相交，合并拉取请求可能会触发两个管道执行。

触发器过滤器示例

对于带有推送和拉取请求事件类型过滤器的 Git 配置，指定的过滤器可能会相互冲突。以下是推送和拉取请求事件的有效过滤器组合示例。

当客户在单个配置对象中组合筛选器时，这些筛选器将使用 AND 操作，这意味着只有完全匹配才会启动管道。以下示例显示了 Git 配置：

```
{
  "filePaths": {
    "includes": ["common/**/*.js"]
  },
  "branches": {
    "includes": ["feature/**"]
  }
}
```

使用上面的 Git 配置，此示例显示了一个事件，该事件将由于 AND 操作成功而启动管道执行。

```
{
  "ref": "refs/heads/feature/triggers",
  ...
  "commits": [
    {
      ...
      "modified": [
        "common/app.js"
      ]
      ...
    }
  ]
}
```

此示例显示了一个事件，该事件无法启动管道执行，因为分支可以筛选，但文件路径不能。

```
{
   "ref": "refs/heads/feature/triggers",
  ...
  "commits": [
    {
      ...
```

```
      "modified": [
        "src/Main.java"
      ]
      ...
    }
  ]
}
```

同时，推送数组中的触发器配置对象使用 OR 操作。这允许您配置多个触发器来启动同一管道的执行。有关 JSON 结构中字段定义的列表，请参阅在管道中触发筛选 JSON (CLI)。

筛选推送事件（控制台）

您可以使用控制台为推送事件添加过滤器，也可以包含或排除分支或文件路径。

筛选推送事件（控制台）

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   将显示与您的 AWS 账户关联的所有管道的名称和状态。
2. 在 Name 中，选择您要编辑的管道的名称。否则，请在管道创建向导中使用以下步骤。
3. 在管道详细信息页中，选择编辑。
4. 在 “编辑” 页面上，选择要编辑的源操作。选择 “编辑触发器”。选择 “指定过滤器”。
5. 在事件类型中，从以下选项中选择推送。

   - 当更改推送到源存储库时，选择 Push 即可启动管道。选择此选项后，这些字段就可以为分支和文件路径或 Git 标签指定筛选器。
   - 选择 P ull req uest 可在源存储库中打开、更新或关闭拉取请求时启动管道。选择此选项后，字段就可以为目标分支和文件路径指定过滤器。
6. 在筛选器类型中，选择以下选项之一。

   - 选择 Branch 以指定触发器监控的源存储库中的分支，以便知道何时开始工作流程运行。在 Include 中，以 glob 格式输入要为触发器配置指定的分支名称模式，以便在指定分支发生更改时启动管道。在 Exclude 中，以 glob 格式输入分支名称的正则表达式模式，以便触发器配置忽略这些模式，以免在指定分支发生更改时启动管道。请参阅使用语法中的 glob 模式了解更多信息。

> **Note**
>
> 如果 include 和 exclude 具有相同的模式，则默认设置为排除该模式。

你可以使用 glob 格式的正则表达式模式来定义你的分支名称。例如，用于匹配所有`main.*`以开头的分支`main.*`。请参阅使用语法中的 glob 模式了解更多信息。

对于推送触发器，请指定要推送到的分支，即目标分支。对于拉取请求触发器，请指定要向其打开拉取请求的目标分支。

- （可选）在 “文件路径” 下，为触发器指定文件路径。根据需要在 “包括” 和 “排除” 中输入名称。

  你可以使用 glob 格式的正则表达式模式来定义文件路径名。例如，使用匹配`prod.*`以开头的所有文件路径`prod.*`。请参阅使用语法中的 glob 模式了解更多信息。

- 选择标签将管道触发器配置配置配置配置为从 Git 标签开始。在 In clude 中，输入要为触发器配置指定的 glob 格式的标签名称模式，以便在指定标签发布时启动管道。在 Exclude 中，输入要为触发器配置指定的 glob 格式的标签名称的正则表达式模式，这样触发器配置就不会在释放一个或多个指定标签时启动管道。如果包含和排除二者具有相同的标签模式，则默认为排除该标签模式。

筛选拉取请求（控制台）

您可以使用控制台为具有指定事件的拉取请求添加过滤器，并包括或排除分支或文件路径。

筛选拉取请求（控制台）

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   将显示与您的 AWS 账户关联的所有管道的名称和状态。
2. 在 Name 中，选择您要编辑的管道的名称。否则，请在管道创建向导中使用以下步骤。
3. 在管道详细信息页中，选择编辑。
4. 在 “编辑” 页面上，选择要编辑的源操作。选择 “编辑触发器”。选择 “指定过滤器”。
5. 在事件类型中，从以下选项中选择拉取请求。

- 当更改推送到源存储库时，选择 Push 即可启动管道。选择此选项后，这些字段就可以为分支和文件路径或 Git 标签指定筛选器。
- 选择 P ull req uest 可在拉取请求被打开、更新或关闭到指定目标分支时启动管道。选择此选项后，字段就可以为分支和文件路径指定过滤器。

  您可以选择指定以下拉取请求事件进行筛选：

  - 拉取请求已创建
  - 对拉取请求进行了新的修订
  - 拉取请求已关闭

6. 在筛选器类型中，选择以下选项之一。

- 选择 Branch 以指定触发器监控的源存储库中的分支，以便知道何时开始工作流程运行。在 Inc lude 中，以 glob 格式输入要为触发器配置指定的分支名称模式，以便在指定分支发生更改时启动管道。在 Exclude 中，以 glob 格式输入分支名称的正则表达式模式，以便触发器配置忽略这些模式，以免在指定分支发生更改时启动管道。请参阅使用语法中的 glob 模式了解更多信息。

  > **Note**
  >
  > 如果 include 和 exclude 具有相同的模式，则默认设置为排除该模式。

  你可以使用 glob 格式的正则表达式模式来定义你的分支名称。例如，用于匹配所有`main.*`以开头的分支`main.*`。请参阅使用语法中的 glob 模式了解更多信息。

  对于推送触发器，请指定要推送到的分支，即目标分支。对于拉取请求触发器，请指定要向其打开拉取请求的目标分支。

- （可选）在“文件路径”下，为触发器指定文件路径名。根据需要在“包括”和“排除”中输入名称。

  你可以使用 glob 格式的正则表达式模式来定义文件路径名。例如，使用匹配`prod.*`以开头的所有文件路径`prod.*`。请参阅使用语法中的 glob 模式了解更多信息。

在管道中触发筛选 JSON (CLI)

您可以更新管道 JSON，为触发器添加过滤器。

要使用创建或更新您的管道，请使用create-pipeline或update-pipeline命令。 AWS CLI

以下 JSON 结构示例，为下的字段定义提供了参考create-pipeline。

```
{
    "pipeline": {
        "name": "MyServicePipeline",
        "triggers": [
            {
                "provider": "Connection",
                "gitConfiguration": {
                    "sourceActionName": "ApplicationSource",
                    "push": [
                        {
                            "filePaths": {
                                "includes": [
                                    "projectA/**",
                                    "common/**/*.js"
                                ],
                                "excludes": [
                                    "**/README.md",
                                    "**/LICENSE",
                                    "**/CONTRIBUTING.md"
                                ]
                            },
                            "branches": {
                                "includes": [
                                    "feature/**",
                                    "release/**"
                                ],
                                "excludes": [
                                    "mainline"
                                ]
                            },
                            "tags": {
                                "includes": [
                                    "release-v0", "release-v1"
                                ],
                                "excludes": [
                                    "release-v2"
                                ]
                            }
                        }
                    ],
```

```
                    "pullRequest": [
                        {
                            "events": [
                                "CLOSED"
                            ],
                            "branches": {
                                "includes": [
                                    "feature/**",
                                    "release/**"
                                ],
                                "excludes": [
                                    "mainline"
                                ]
                            },
                            "filePaths": {
                                "includes": [
                                    "projectA/**",
                                    "common/**/*.js"
                                ],
                                "excludes": [
                                    "**/README.md",
                                    "**/LICENSE",
                                    "**/CONTRIBUTING.md"
                                ]
                            }
                        }
                    ]
                }
            }
        ],
        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "name": "ApplicationSource",
                        "configuration": {
                            "BranchName": "mainline",
                            "ConnectionArn": "arn:aws:codestar-connections:eu-
central-1:111122223333:connection/fe9ff2e8-ee25-40c9-829e-65f8EXAMPLE",
                            "FullRepositoryId": "monorepo-example",
                            "OutputArtifactFormat": "CODE_ZIP"
                        }
                    }
```

```
                ]
            }
        ]
    }
}
```

JSON 结构中的字段定义如下：

- sourceActionName：具有 Git 配置的管道源操作的名称。
- push：使用过滤功能推送事件。这些事件在不同的推送过滤器之间使用 OR 运算，在过滤器内使用 AND 操作。
  - branches: 要筛选的分支。分支在包含和排除之间使用 AND 运算。
    - includes: 用于筛选将包含的分支的模式。包括使用 OR 操作。
    - excludes: 筛选将被排除的分支的模式。不包括使用 OR 操作。
  - filePaths：要筛选的文件路径名。
    - includes：用于筛选将包含的文件路径的模式。包括使用 OR 操作。
    - excludes: 筛选将被排除的文件路径的模式。不包括使用 OR 操作。
  - tags：要筛选的标签名称。
    - includes：用于筛选将包含的标签的模式。包括使用 OR 操作。
    - excludes：用于筛选要排除的标签的模式。不包括使用 OR 操作。
- pullRequest: 拉取请求事件，可筛选拉取请求事件和拉取请求过滤器。
  - events：按指定筛选已打开、更新或已关闭的拉取请求事件。
  - branches: 要筛选的分支。分支在包含和排除之间使用 AND 运算。
    - includes: 用于筛选将包含的分支的模式。包括使用 OR 操作。
    - excludes: 筛选将被排除的分支的模式。不包括使用 OR 操作。
  - filePaths：要筛选的文件路径名。
    - includes：用于筛选将包含的文件路径的模式。包括使用 OR 操作。
    - excludes: 筛选将被排除的文件路径的模式。不包括使用 OR 操作。

在 AWS CloudFormation 模板中触发筛选

您可以在中更新管道资源 AWS CloudFormation 以添加触发器筛选。

以下示例模板片段为触发器字段定义提供了 YAML 参考。有关字段定义的列表，请参见在管道中触发筛选 JSON (CLI)。

```
pipeline:
  name: MyServicePipeline
  executionMode: PARALLEL
  triggers:
    - provider: CodeConnection
      gitConfiguration:
        sourceActionName: ApplicationSource
        push:
          - filePaths:
              includes:
                - projectA/**
                - common/**/*.js
              excludes:
                - '**/README.md'
                - '**/LICENSE'
                - '**/CONTRIBUTING.md'
            branches:
              includes:
                - feature/**
                - release/**
              excludes:
                - mainline
          - tags:
              includes:
                - release-v0
                - release-v1
              excludes:
                - release-v2
        pullRequest:
          - events:
              - CLOSED
            branches:
              includes:
                - feature/**
                - release/**
              excludes:
                - mainline
            filePaths:
              includes:
                - projectA/**
```

```
                - common/**/*.js
              excludes:
                - '**/README.md'
                - '**/LICENSE'
                - '**/CONTRIBUTING.md'
  stages:
    - name: Source
      actions:
        - name: ApplicationSource
          configuration:
            BranchName: mainline
            ConnectionArn: arn:aws:codestar-connections:eu-
central-1:111122223333:connection/fe9ff2e8-ee25-40c9-829e-65f85EXAMPLE
            FullRepositoryId: monorepo-example
            OutputArtifactFormat: CODE_ZIP
```

在中管理执行 CodePipeline

要分析管道进度，您可以查看错误日志、查看管道和操作执行历史记录，以及重试失败的阶段或操作。

主题

在中查看执行情况 CodePipeline

您可以使用 AWS CodePipeline 控制台或查看执行状态、查看执行历史记录以及重试失败的阶段或操作。 AWS CLI

主题

查看管道执行历史记录（控制台）

您可以使用 CodePipeline 控制台查看您账户中所有管道的列表。还可以查看每个管道的详细信息，包括管道中最后一次运行操作的时间，阶段之间的过渡是启用还是禁用，是否有任何操作失败以及其他信息。您还可以查看历史记录页面，其中显示已记录历史记录的所有管道执行的详细信息。

> **Note**
>
> 在特定的执行模式之间切换时，管道视图和历史记录可能会发生变化。有关更多信息，请参阅设置或更改管道执行模式。

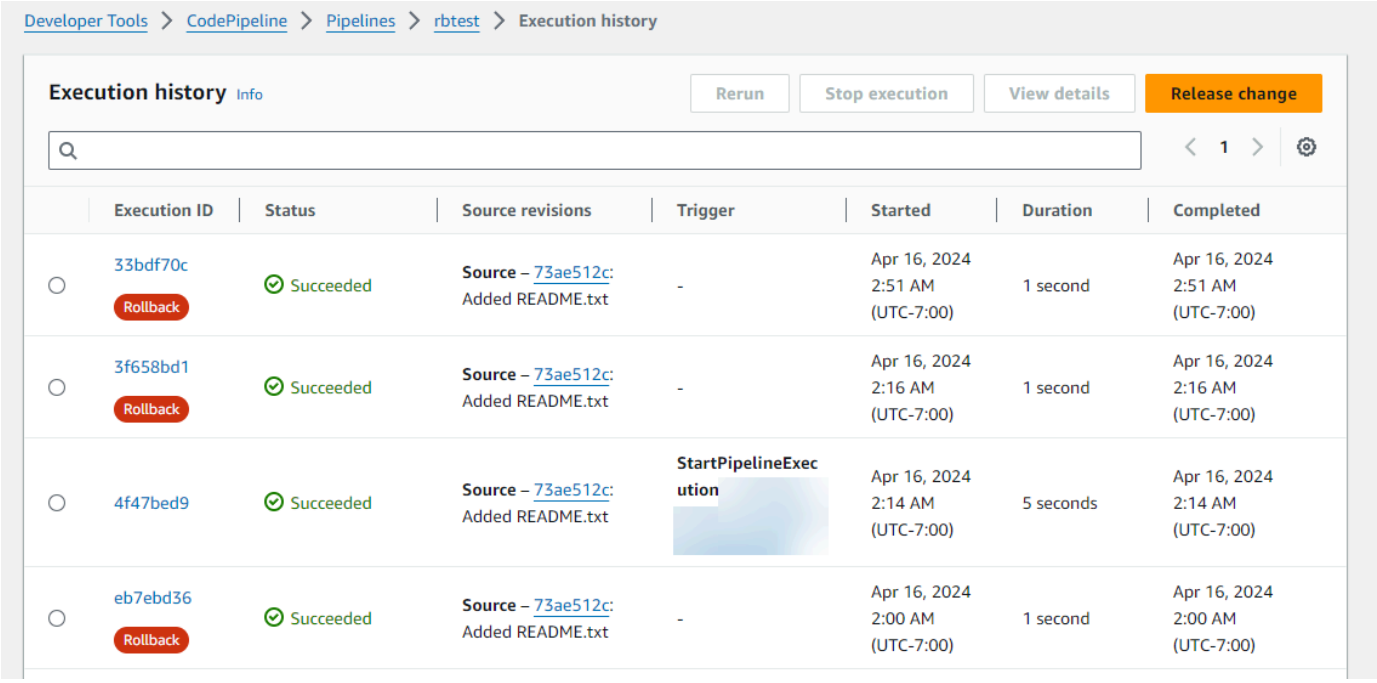
执行历史记录保留长达 12 个月。

您可以使用控制台查看管道中的执行历史记录，包括每次执行的状态、源修订和时间详细信息。

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   将显示与您的 AWS 账户关联的所有管道的名称及其状态。
2. 在名称中，选择管道的名称。
3. 选择 View history (查看历史记录)。

> **Note**
>
> 对于处于并行执行模式的管道，主管道视图不显示管道结构或正在进行的执行。对于处于并行执行模式的管道，您可以通过从执行历史记录页面选择要查看的执行的 ID 来访问管道结构。在左侧导航栏中选择 “历史记录”，选择并行执行的执行 ID，然后在 “可视化” 选项卡上查看管道。



4. 查看与管道的每次执行相关的状态、源修订、更改详细信息和触发器。已回滚的管道执行将在控制台的详细信息屏幕上显示执行类型 Rollbac k。对于触发自动回滚的失败执行，将显示失败的执行 ID。

5. 选择一个执行。详细信息视图显示执行详细信息、时间表选项卡、可视化选项卡和变量选项卡。管道级别变量的值在管道执行时进行解析，可以在每个执行的执行历史记录中查看。

> **Note**
>
> 对于管道操作的输出变量，可以在每个操作执行的历史记录下方的输出变量选项卡上查看。

查看执行状态（控制台）

您可以在执行历史记录页面上的 Status (状态) 中查看管道状态。选择一个执行 ID 链接，然后查看操作状态。

以下是管道、阶段和操作的有效状态：

> **Note**
>
> 以下管道状态也适用于属于入站执行的管道执行。要查看入站执行及其状态，请参阅查看入站执行（控制台）。

管道级别状态

管道状态	描述
InProgress	当前正在运行管道执行。
Stopping	由于发出了停止并等待或停止并放弃管道执行的请求，管道执行正在停止。
Stopped（已停止）	停止过程已完成，管道执行已停止。
成功	已成功完成管道执行。
已取代	在该管道执行等待完成下一阶段时，启动了较新的管道执行并通过管道。
失败	未成功完成管道执行。

阶段级别状态

阶段状态	描述
InProgress	当前正在运行阶段。
Stopping	由于发出了停止并等待或停止并放弃管道执行的请求，阶段执行正在停止。
Stopped（已停止）	停止过程已完成，阶段执行已停止。
成功	已成功完成阶段。
失败	阶段未成功完成。

操作级别权限

操作状态	描述
InProgress	当前正在运行操作。
已放弃	由于发出了停止并放弃管道执行的请求，已放弃操作。
成功	已成功完成操作。
失败	对于审批操作，FAILED (已失败) 状态表示审查者已拒绝操作，或者由于操作配置不正确而失败。

查看入站执行（控制台）

您可以使用控制台查看入站执行的状态和详细信息。在启用转换或阶段变为可用后，`InProgress` 状态的入站执行将会继续并进入该阶段。状态为 `Stopped` 的入站执行不会进入该阶段。如果对管道进行编辑，则入站执行状态将更改为 `Failed`。编辑管道时，所有进行中的执行都不会继续进行，执行状态将更改为 `Failed`。

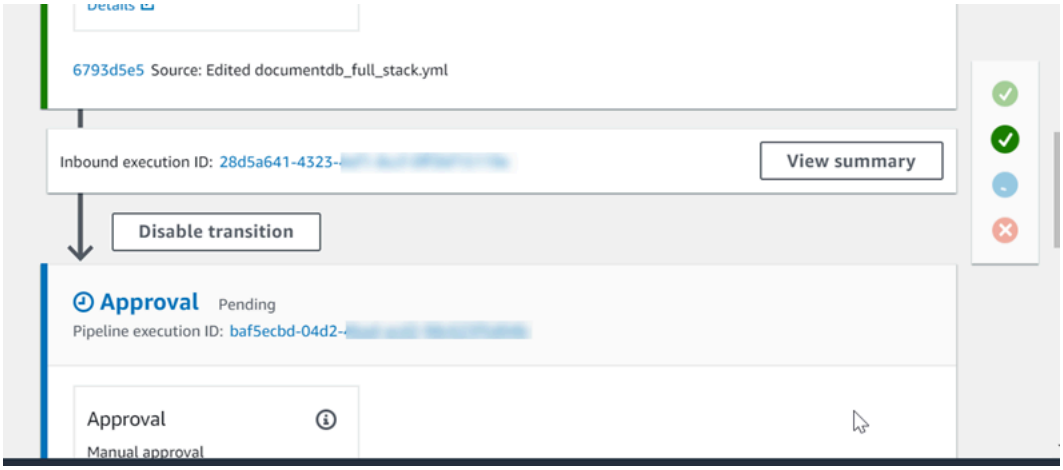
如果未看到入站执行，说明禁用的阶段转换中没有待处理的执行。

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

将显示与您的 AWS 账户关联的所有管道的名称。

2. 选择要查看入站执行的管道的名称，执行以下任一操作：

   - 选择 查看。在管道图中，在禁用转换前的入站执行 ID 字段中，您可以查看入站执行 ID。



   选择查看摘要以查看执行详细信息，如执行 ID、源触发器和下一阶段的名称。

   - 选择管道并选择查看历史记录。

查看管道执行源修订（控制台）

您可以查看在管道执行过程中使用的源构件（在管道的第一阶段产生的输出构件）的详细信息。详细信息包括标识符，例如提交 ID 和签入注释，如果使用 CLI，还包括管道构建操作的版本号。对于某些修订类型，您可以查看并打开提交的 URL。源修订由以下内容组成：

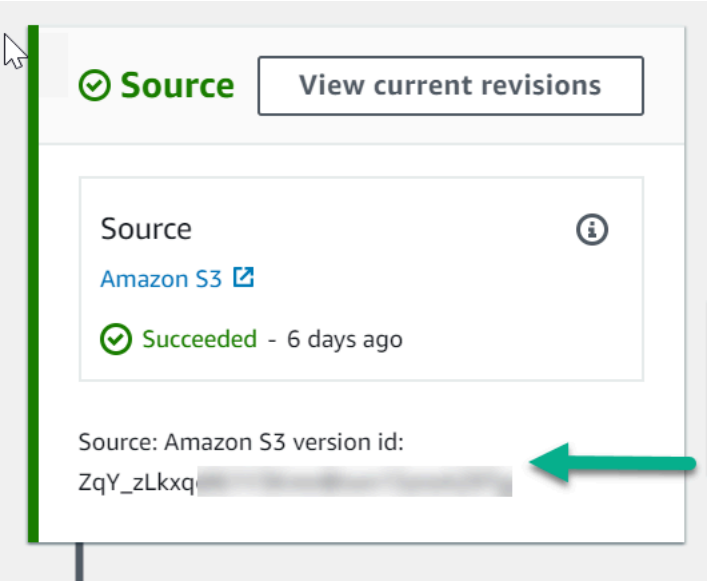
- 摘要：有关构件最新修订的摘要信息。对于 GitHub 和 CodeCommit 存储库，则为提交消息。对于 Amazon S3 存储桶或操作，在对象元数据中指定的用户提供的 codepipeline-artifact-revision-summary 密钥内容。
- revisionUrl：构件修订的修订 URL（例如，外部存储库 URL）。
- revisionId：构件修订的修订 ID。例如，对于 CodeCommit 或 GitHub 存储库中的源代码更改，这是提交 ID。对于存储在 GitHub 或存储 CodeCommit 库中的项目，提交 ID 链接到提交详细信息页面。

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

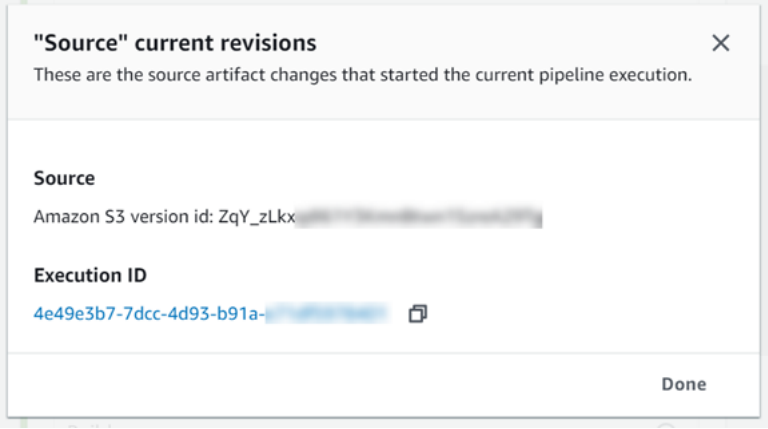
   此时将显示所有与您的 AWS 账户关联的管道的名称。

2. 选择您要查看源修订详细信息的管道的名称。请执行以下操作之一：

   - 选择 View history (查看历史记录)。在 Source revisions (源修订) 中，将列出每次执行的源更改。
   - 查找您要查看源修订详细信息的操作，然后在其阶段的底部找到修订信息：



   选择 View current revisions (查看当前修订) 以查看源信息。除存储在 Amazon S3 桶中的构件之外，此详细信息视图中的标识符（如提交 ID）会与构件的源信息页面关联。



查看操作执行（控制台）

您可以查看管道的操作详细信息，例如操作执行 ID、输入构件、输出构件和状态。您可以通过在控制台中选择管道，然后选择执行 ID 来查看操作详细信息。

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   将显示与您的 AWS 账户关联的所有管道的名称。

2. 选择要查看其操作详细信息的管道的名称，然后选择 View history (查看历史记录)。
3. 在 Execution ID (执行 ID) 中，选择要查看其操作执行详细信息的执行 ID。
4. 您可以在 Timeline (时间线) 选项卡上查看以下信息：

   a. 在 Action name (操作名称) 中，选择链接以打开操作的详细信息页面，您可以在其中查看状态、阶段名称、操作名称、配置数据和构件信息。

   b. 在 Provider (提供程序) 中，选择链接以查看操作提供程序详细信息。例如，在前面的示例管道 CodeDeploy 中，如果您选择暂存阶段或生产阶段，则会显示为该阶段配置的 CodeDeploy 应用程序的 CodeDeploy控制台页面。

查看操作构件和构件存储信息（控制台）

您可以查看操作的输入和输出构件详细信息。您还可以选择一个链接，将您带到该操作的构件信息。由于构件存储使用版本控制，因此每个操作执行都具有唯一的输入和输出构件位置。

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   将显示与您的 AWS 账户关联的所有管道的名称。

2. 选择要查看其操作详细信息的管道的名称，然后选择 View history (查看历史记录)。
3. 在 Execution ID (执行 ID) 中，选择要查看其操作详细信息的执行 ID。
4. 在 Timeline (时间线) 选项卡上的 Action name (操作名称) 中，选择用于打开操作详细信息页面的链接。
5. 在详细信息页面上的执行中，查看执行操作的状态和时间。
6. 在配置选项卡上，查看操作的资源配置（例如， CodeBuild 生成项目名称）。
7. 在构件中，查看构件类型和构件提供程序中的构件详细信息。选择 Artifact name (构件名称) 下的链接，以查看构件存储中的构件。
8. 在输出变量选项卡上，查看从操作执行管道中的操作解析的变量。

查看管道详细信息和历史记录（CLI）

您可以运行以下命令，以查看您的管道和管道执行的详细信息：

- list-pipelines命令查看与您的关联的所有管道的摘要 AWS 账户。
- get-pipeline 命令，用于查看单个管道的详细信息。
- list-pipeline-executions，用于查看管道的最近执行的摘要。
- get-pipeline-execution 用于查看有关管道执行的信息，包括有关构件的详细信息、管道执行 ID 以及管道的名称、版本和状态。
- get-pipeline-state 命令用于查看管道、阶段和操作状态。
- list-action-executions 用于查看管道的操作执行详细信息。

主题

使用 **list-pipeline-executions** (CLI) 查看执行历史记录

您可以查看管道执行历史记录。

- 要查看管道的过去执行的详细信息，请运行 list-pipeline-executions 命令，指定管道的唯一名称。例如，要查看名为的管道当前状态的详细信息 *MyFirstPipeline*，请输入以下内容：

  ```
  aws codepipeline list-pipeline-executions --pipeline-name MyFirstPipeline
  ```

  此命令返回有关已记录其历史记录的所有管道执行的摘要信息。摘要包括开始和结束时间、持续时间和状态。

  已回滚的管道执行将显示执行类型Rollback。对于触发自动回滚的失败执行，将显示失败的执行ID。

  以下示例显示了名为的管道的返回数据 *MyFirstPipeline*，该管道已执行了三次：

```
{
    "pipelineExecutionSummaries": [
        {
            "pipelineExecutionId": "eb7ebd36-353a-4551-90dc-18ca5EXAMPLE",
            "status": "Succeeded",
            "startTime": "2024-04-16T09:00:28.185000+00:00",
            "lastUpdateTime": "2024-04-16T09:00:29.665000+00:00",
            "sourceRevisions": [
                {
                    "actionName": "Source",
                    "revisionId": "revision_ID",
                    "revisionSummary": "Added README.txt",
                    "revisionUrl": "console-URL"
                }
            ],
            "trigger": {
                "triggerType": "StartPipelineExecution",
                "triggerDetail": "trigger_ARN"
            },
            "executionMode": "SUPERSEDED"
        },
        {
            "pipelineExecutionId": "fcd61d8b-4532-4384-9da1-2aca1EXAMPLE",
            "status": "Succeeded",
            "startTime": "2024-04-16T08:58:56.601000+00:00",
            "lastUpdateTime": "2024-04-16T08:59:04.274000+00:00",
            "sourceRevisions": [
                {
                    "actionName": "Source",
                    "revisionId": "revision_ID",
                    "revisionSummary": "Added README.txt",
                    "revisionUrl": "console_URL"
                }
            ],
            "trigger": {
                "triggerType": "StartPipelineExecution",
                "triggerDetail": "trigger_ARN"
            },
            "executionMode": "SUPERSEDED"
        }
```

要查看管道执行的更多详细信息，请运行 get-pipeline-execution，指定管道执行的唯一 ID。例如，要查看上一示例中第一次执行的更多详细信息，请输入以下内容：

```
aws codepipeline get-pipeline-execution --pipeline-name MyFirstPipeline --pipeline-execution-id 7cf7f7cb-3137-539g-j458-d7eu3EXAMPLE
```

该命令会返回有关管道执行的摘要信息，包括项目的详细信息、管道执行 ID 以及管道的名称、版本和状态。

以下示例显示了名为的管道的返回数据 *MyFirstPipeline*：

```
{
    "pipelineExecution": {
        "pipelineExecutionId": "3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE",
        "pipelineVersion": 2,
        "pipelineName": "MyFirstPipeline",
        "status": "Succeeded",
        "artifactRevisions": [
            {
                "created": 1496380678.648,
                "revisionChangeIdentifier": "1496380258.243",
                "revisionId": "7636d59f3c461cEXAMPLE8417dbc6371",
                "name": "MyApp",
                "revisionSummary": "Updating the application for feature 12-4820"
            }
        ]
    }
}
```

使用 **get-pipeline-state** (CLI) 查看管道状态

您可以使用 CLI 查看管道、阶段和操作状态。

- 要查看管道的当前状态的详细信息，请运行 get-pipeline-state 命令，指定管道的唯一名称。例如，要查看名为的管道当前状态的详细信息 *MyFirstPipeline*，请输入以下内容：

```
aws codepipeline get-pipeline-state --name MyFirstPipeline
```

该命令会返回管道所有阶段的当前状态以及这些阶段中操作的状态。

以下示例显示了名为的三阶段管道的返回数据 *MyFirstPipeline*，其中前两个阶段和操作显示成功，第三个阶段和操作显示失败，第二阶段和第三阶段之间的过渡被禁用：

```
{
    "updated": 1427245911.525,
    "created": 1427245911.525,
    "pipelineVersion": 1,
    "pipelineName": "MyFirstPipeline",
    "stageStates": [
        {
            "actionStates": [
                {
                    "actionName": "Source",
                    "entityUrl": "https://console.aws.amazon.com/s3/home?#",
                    "latestExecution": {
                        "status": "Succeeded",
                        "lastStatusChange": 1427298837.768
                    }
                }
            ],
            "stageName": "Source"
        },
        {
            "actionStates": [
                {
                    "actionName": "Deploy-CodeDeploy-Application",
                    "entityUrl": "https://console.aws.amazon.com/codedeploy/home?
#",
                    "latestExecution": {
                        "status": "Succeeded",
                        "lastStatusChange": 1427298939.456,
                        "externalExecutionUrl": "https://console.aws.amazon.com/?
#",
                        "externalExecutionId": ""c53dbd42-This-Is-An-Example"",
                        "summary": "Deployment Succeeded"
                    }
                }
            ],
            "inboundTransitionState": {
                "enabled": true
            },
            "stageName": "Staging"
```

```
        },
        {
            "actionStates": [
                {
                    "actionName": "Deploy-Second-Deployment",
                    "entityUrl": "https://console.aws.amazon.com/codedeploy/home?
#",
                    "latestExecution": {
                        "status": "Failed",
                        "errorDetails": {
                            "message": "Deployment Group is already deploying
 deployment ...",
                            "code": "JobFailed"
                        },
                        "lastStatusChange": 1427246155.648
                    }
                }
            ],
            "inboundTransitionState": {
                "disabledReason": "Disabled while I investigate the failure",
                "enabled": false,
                "lastChangedAt": 1427246517.847,
                "lastChangedBy": "arn:aws:iam::80398EXAMPLE:user/CodePipelineUser"
            },
            "stageName": "Production"
        }
    ]
}
```

使用 **get-pipeline-state** (CLI) 查看入站执行状态

您可以使用 CLI 查看入站执行状态。在启用转换或阶段变为可用后，InProgress 状态的入站执行将会继续并进入该阶段。状态为 Stopped 的入站执行不会进入该阶段。如果对管道进行编辑，则入站执行状态将更改为 Failed。编辑管道时，所有进行中的执行都不会继续进行，执行状态将更改为 Failed。

- 要查看管道的当前状态的详细信息，请运行 get-pipeline-state 命令，指定管道的唯一名称。例如，要查看名为的管道当前状态的详细信息 *MyFirstPipeline*，请输入以下内容：

```
aws codepipeline get-pipeline-state --name MyFirstPipeline
```

该命令会返回管道所有阶段的当前状态以及这些阶段中操作的状态。输出中还会显示每个阶段中的管道执行 ID，以及禁用转换的阶段是否有入站执行 ID。

以下示例显示名为的两阶段管道的返回数据 *MyFirstPipeline*，其中第一阶段显示已启用的过渡和成功的管道执行Beta，名为的第二阶段显示已禁用的过渡和入站执行 ID。入站执行状态可以是 InProgress、Stopped 或 FAILED。

```
{
    "pipelineName": "MyFirstPipeline",
    "pipelineVersion": 2,
    "stageStates": [
        {
            "stageName": "Source",
            "inboundTransitionState": {
                "enabled": true
            },
            "actionStates": [
                {
                    "actionName": "SourceAction",
                    "currentRevision": {
                        "revisionId": "PARcnxX_u0SMRBnKh83pHL09.zPRLLMu"
                    },
                    "latestExecution": {
                        "actionExecutionId": "14c8b311-0e34-4bda-EXAMPLE",
                        "status": "Succeeded",
                        "summary": "Amazon S3 version id: PARcnxX_u0EXAMPLE",
                        "lastStatusChange": 1586273484.137,
                        "externalExecutionId": "PARcnxX_u0EXAMPLE"
                    },
                    "entityUrl": "https://console.aws.amazon.com/s3/home?#"
                }
            ],
            "latestExecution": {
                "pipelineExecutionId": "27a47e06-6644-42aa-EXAMPLE",
                "status": "Succeeded"
            }
        },
        {
            "stageName": "Beta",
            "inboundExecution": {
                "pipelineExecutionId": "27a47e06-6644-42aa-958a-EXAMPLE",
                "status": "InProgress"
```

```
            },
            "inboundTransitionState": {
                "enabled": false,
                "lastChangedBy": "USER_ARN",
                "lastChangedAt": 1586273583.949,
                "disabledReason": "disabled"
            },
                    "currentRevision": {
            "actionStates": [
                {
                    "actionName": "BetaAction",
                    "latestExecution": {
                        "actionExecutionId": "a748f4bf-0b52-4024-98cf-EXAMPLE",
                        "status": "Succeeded",
                        "summary": "Deployment Succeeded",
                        "lastStatusChange": 1586272707.343,
                        "externalExecutionId": "d-KFGF3EXAMPLE",
                        "externalExecutionUrl": "https://us-
west-2.console.aws.amazon.com/codedeploy/home?#/deployments/d-KFGF3WTS2"
                    },
                    "entityUrl": "https://us-west-2.console.aws.amazon.com/
codedeploy/home?#/applications/my-application"
                }
            ],
            "latestExecution": {
                "pipelineExecutionId": "f6bf1671-d706-4b28-EXAMPLE",
                "status": "Succeeded"
            }
        }
    ],
    "created": 1585622700.512,
    "updated": 1586273472.662
}
```

使用 **get-pipeline-execution** (CLI) 查看状态和源版本号

您可以查看在管道执行过程中使用的源构件（在管道的第一阶段产生的输出构件）的详细信息。详细信息包括标识符（例如提交 ID）、签入注释、自创建或更新项目以来的时间，如果您使用 CLI，还包括生成操作的版本号。对于某些修订类型，您可以查看并打开构件版本对应的提交内容的 URL。源修订由以下内容组成：

- 摘要：有关构件最新修订的摘要信息。对于 GitHub 和 AWS CodeCommit 存储库，则为提交消息。对于 Amazon S3 存储桶或操作，在对象元数据中指定的用户提供的 codepipeline-artifact-revision-summary 密钥内容。
- revisionUrl：构件修订的提交 ID。对于存储在 GitHub 或存储 AWS CodeCommit 库中的项目，提交 ID 链接到提交详细信息页面。

您可以运行 get-pipeline-execution 命令来查看有关管道执行中包含的最新源修订的信息。首次运行 get-pipeline-state 命令以获取管道中所有阶段的详细信息后，可以确定适用于您需要其源修订详细信息的阶段的执行 ID。然后，您将在 get-pipeline-execution 命令中使用该执行 ID。（因为管道中的阶段可能在不同的管道运行期间最后一个成功完成，所以它们可以具有不同的执行 ID。）

换句话说，如果要查看当前处于 Staging 阶段的构件的详细信息，请运行 get-pipeline-state 命令，确定 Staging 阶段的当前执行 ID，然后使用该执行 ID 运行 get-pipeline-execution 命令。

查看管道中的状态和源修订版本

1. 打开终端（Linux、macOS 或 Unix）或命令提示符 (Windows)，并使用 AWS CLI 运行 get-pipeline-state 命令。对于名为的管道 *MyFirstPipeline*，您可以输入：

   ```
   aws codepipeline get-pipeline-state --name MyFirstPipeline
   ```

   该命令将返回管道的最新状态，包括每个阶段的最新管道执行 ID。

2. 要查看有关管道执行的详细信息，请运行 get-pipeline-execution 命令，指定管道的唯一名称以及您要查看构件详细信息的执行的管道执行 ID。例如，要查看名为、执行编号为 *MyFirstPipeline*3137f7cb-7cf7-039j-s83l-d7eu3Example 的管道的执行详细信息，您需要输入以下内容：

   ```
   aws codepipeline get-pipeline-execution --pipeline-name MyFirstPipeline --pipeline-execution-id 3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE
   ```

   该命令将返回有关作为管道执行的一部分且可识别有关管道的信息的每个源修订的信息。这里只包含有关该执行中包含的管道阶段的信息。管道中可能存在不属于该管道执行的其他阶段。

   以下示例显示了名为 “” 的管道部分的返回数据 *MyFirstPipeline*，其中名为 “MyApp” 的构件存储在存储 GitHub 库中：

3. ```
   {
       "pipelineExecution": {
   ```
   ```

```
        "artifactRevisions": [
            {
                "created": 1427298837.7689769,
                "name": "MyApp",
                "revisionChangeIdentifier": "1427298921.3976923",
                "revisionId": "7636d59f3c461cEXAMPLE8417dbc6371",
                "revisionSummary": "Updating the application for feature 12-4820",
                "revisionUrl": "https://api.github.com/repos/anycompany/MyApp/git/
commits/7636d59f3c461cEXAMPLE8417dbc6371"
            }
        ],
        "pipelineExecutionId": "3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE",
        "pipelineName": "MyFirstPipeline",
        "pipelineVersion": 2,
        "status": "Succeeded",
        "executionMode": "SUPERSEDED",
        "executionType": "ROLLBACK",
        "rollbackMetadata": {
            "rollbackTargetPipelineExecutionId": "4f47bed9-6998-476c-a49d-
e60beEXAMPLE"
         }
    }
}
```

使用 **list-action-executions** (CLI) 查看操作执行情况

您可以查看管道的操作执行详细信息，例如操作执行 ID、输入构件、输出构件、执行结果和状态。您提供执行 ID 筛选条件以返回管道执行中的操作列表：

> **Note**
>
> 在 2019 年 2 月 21 日或之后运行的执行有详细的执行历史记录。

- 要查看管道的操作执行，请执行下列操作之一：
  - 要查看管道中所有操作执行的详细信息，请运行 list-action-executions 命令，并在命令中指定管道的唯一名称。例如，要查看名为的管道中的操作执行情况 *MyFirstPipeline*，请输入以下内容：

```
aws codepipeline list-action-executions --pipeline-name MyFirstPipeline
```

下面显示了此命令的示例输出的一部分：

```
{
    "actionExecutionDetails": [
        {
            "actionExecutionId": "ID",
            "lastUpdateTime": 1552958312.034,
            "startTime": 1552958246.542,
            "pipelineExecutionId": "Execution_ID",
            "actionName": "Build",
            "status": "Failed",
            "output": {
                "executionResult": {
                    "externalExecutionUrl": "Project_ID",
                    "externalExecutionSummary": "Build terminated with state:
 FAILED",
                    "externalExecutionId": "ID"
                },
                "outputArtifacts": []
            },
            "stageName": "Beta",
            "pipelineVersion": 8,
            "input": {
                "configuration": {
                    "ProjectName": "java-project"
                },
                "region": "us-east-1",
                "inputArtifacts": [
                    {
                        "s3location": {
                            "bucket": "codepipeline-us-east-1-ID",
                            "key": "MyFirstPipeline/MyApp/Object.zip"
                        },
                        "name": "MyApp"
                    }
                ],
                "actionTypeId": {
                    "version": "1",
                    "category": "Build",
                    "owner": "AWS",
```

```
                    "provider": "CodeBuild"
                }
            }
        },

. . .
```

- 要查看管道中的所有操作执行，请运行 list-action-executions 命令，并在命令中指定管道的唯一名称和执行 ID。例如，要查看 *Execution_ID* 的操作执行，请输入以下内容：

```
aws codepipeline list-action-executions --pipeline-name MyFirstPipeline --filter
 pipelineExecutionId=Execution_ID
```

- 下面显示了此命令的示例输出的一部分：

```
{
    "actionExecutionDetails": [
        {
            "stageName": "Beta",
            "pipelineVersion": 8,
            "actionName": "Build",
            "status": "Failed",
            "lastUpdateTime": 1552958312.034,
            "input": {
                "configuration": {
                    "ProjectName": "java-project"
                },
                "region": "us-east-1",
                "actionTypeId": {
                    "owner": "AWS",
                    "category": "Build",
                    "provider": "CodeBuild",
                    "version": "1"
                },
                "inputArtifacts": [
                    {
                        "s3location": {
                            "bucket": "codepipeline-us-east-1-ID",
                            "key": "MyFirstPipeline/MyApp/Object.zip"
                        },
                        "name": "MyApp"
                    }
                ]
```

```
            },


. . .
```

设置或更改管道执行模式

您可以为管道设置执行模式，以指定如何处理多个执行。

有关管道执行模式的更多信息，请参见管道执行的工作原理。

> **Important**
>
> 对于处于并行模式的管道，当将管道执行模式编辑为 QUEUED 或 SUPERSEDED 时，管道状态不会将更新的状态显示为 PARALLEL。有关更多信息，请参阅 从 PARALLEL 模式更改的管道将显示之前的执行模式。

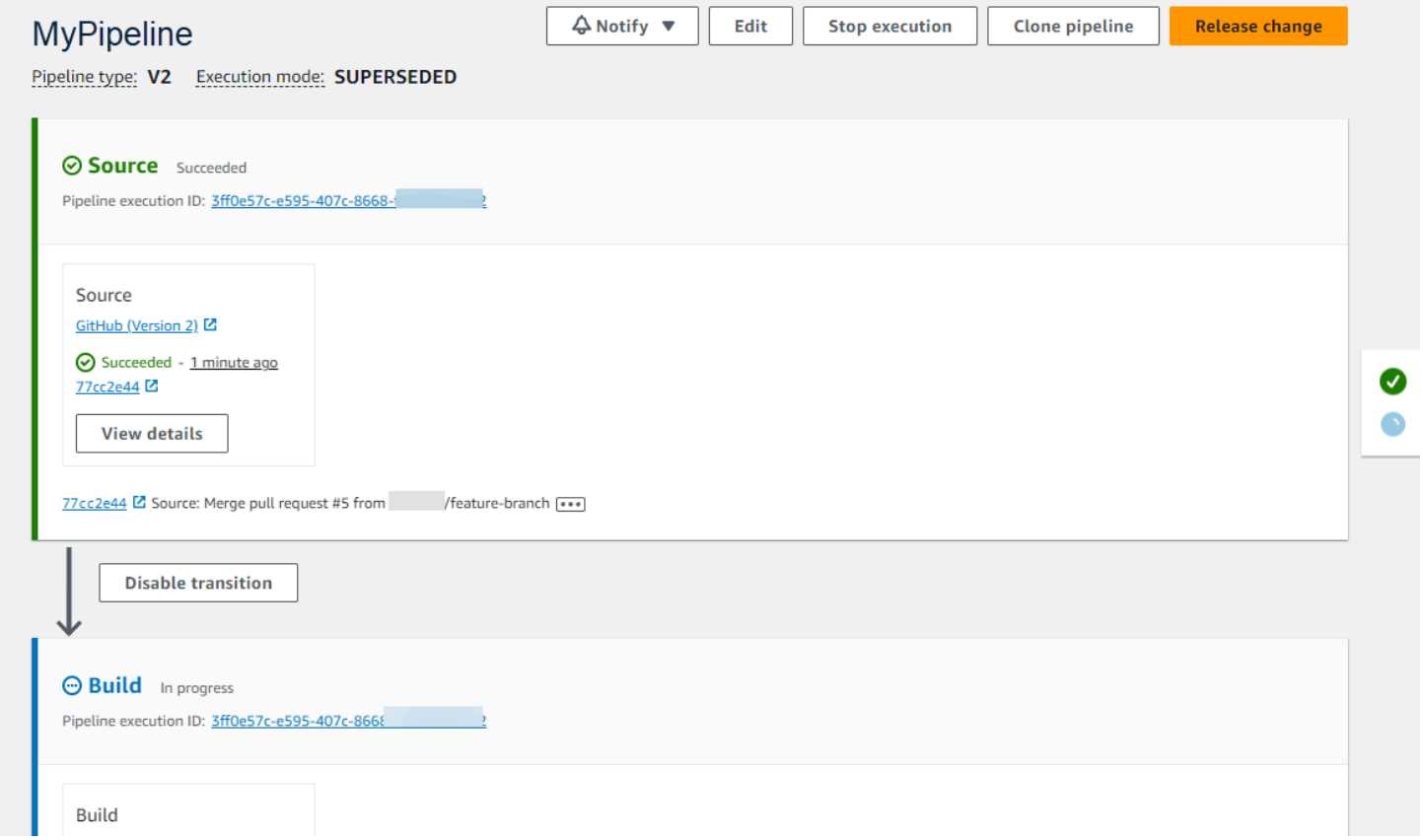
> **Important**
>
> 对于处于并行模式的管道，当将管道执行模式编辑为 QUEUED 或 SUPERSEDED 时，不会更新每种模式下管道的管道定义。有关更多信息，请参阅 如果在更改为 QUEUED 或 SUPERSEDED 模式时进行编辑，则处于并行模式的管道定义会过时。

查看执行模式的注意事项

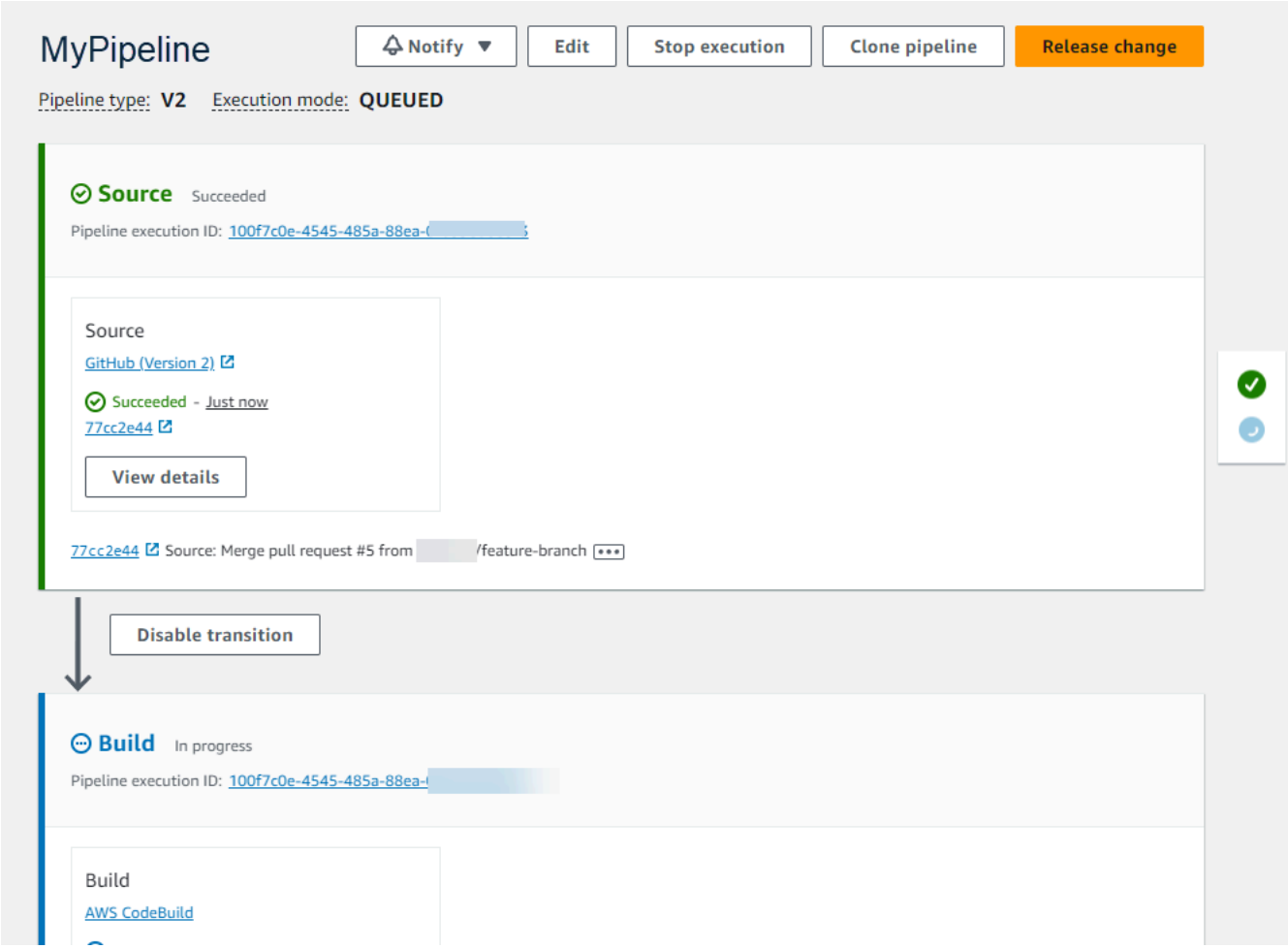
在特定执行模式下查看管道需要注意一些注意事项。

对于 SUPERSEDED 和 QUEUED 模式，使用管道视图查看正在进行的执行，然后单击执行 ID 查看详细信息和历史记录。对于并行模式，单击执行 ID 可在可视化选项卡上查看正在进行的执行。

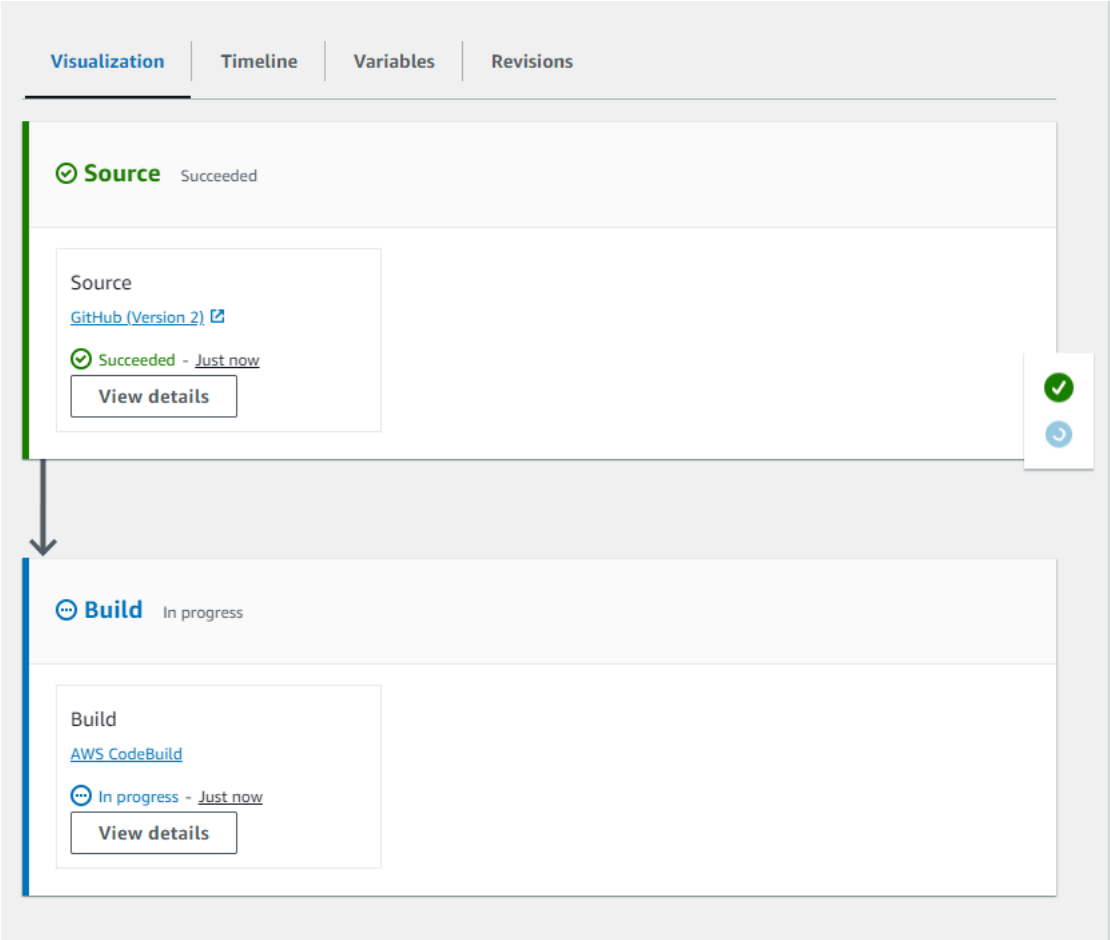
下图显示了 SUPERSEDED 模式在中的视图。 CodePipeline



下图显示了中排队模式的 CodePipeline视图。



下图显示了中并行模式的视图 CodePipeline。



在执行模式之间切换的注意事项

以下是更改管道模式时管道的注意事项。在编辑模式下从一种执行模式切换到另一种执行模式并保存更改时，某些视图或状态可能会进行调整。

例如，从并行模式切换到队列或取代模式时，在并行模式下启动的执行将继续运行。这些可以在执行历史页面上查看。管道视图将显示之前在 QUEUED 或 SUPERSEDED 模式下运行的执行，否则显示为空状态。

再举一个例子，当从 QUEUED 或 SUPERSEDED 模式切换到并行模式时，您将不再看到管道视图/状态页面。要以并行模式查看执行情况，请使用执行详细信息页面上的可视化选项卡。在 “取代” 或 “队列” 模式下开始的执行将被取消。

下表提供了更多详细信息。

模式更改	待处理和正在执行的执行详情	管道状态详情
已取代为 “已取代”/“取代” 为 “已排队”	• 正在进行的操作完成后，主动执行将被取消。 • 待处理的执行被取消。	管道状态（例如已取消）在第一模式和第二模式的版本之间保留。
已排队等候/已排队等候取代	• 正在进行的操作完成后，主动执行将被取消。 • 待处理的执行被取消。	管道状态（例如已取消）在第一模式和第二模式的版本之间保留。
平行到平行	允许所有执行独立于管道定义更新运行。	空。并行模式没有管道状态。
已取代为并行/已排队为并行	• 正在进行的操作完成后，主动执行将被取消。 • 待处理的执行被取消。	空。并行模式没有管道状态。

设置或更改管道执行模式（控制台）

您可以使用控制台设置管道执行模式。

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   将显示与您的 AWS 账户关联的所有管道的名称和状态。
2. 在 Name 中，选择您要编辑的管道的名称。
3. 在管道详细信息页中，选择编辑。
4. 在编辑页面上，选择编辑：管道属性。
5. 为您的管道选择模式。

   - 已取代
   - 已排队（需要管道类型 V2）
   - 并行（需要管道类型 V2）
6. 在 “编辑” 页面上，选择 “完成”。

设置管道执行模式 (CLI)

要使用 AWS CLI 来设置管道执行模式，请使用`create-pipeline`或`update-pipeline`命令。

1. 打开终端会话（Linux、macOS 或 Unix）或命令提示符 (Windows)，然后运行 get-pipeline 命令以将管道结构复制到 JSON 文件中。例如，对于名为 **MyFirstPipeline** 的管道，输入以下命令：

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   该命令不会返回任何结果，但您创建的文件将出现在您运行命令所在的目录中。

2. 在任何纯文本编辑器中打开 JSON 文件，然后修改文件结构以反映要设置的管道执行模式，例如 QUEUED。

   ```
   "executionMode": "QUEUED"
   ```

   以下示例说明如何在包含两个阶段的示例管道中将执行模式设置为 QUEUED。

   ```
   {
       "pipeline": {
           "name": "MyPipeline",
           "roleArn": "arn:aws:iam::111122223333:role/service-role/
   AWSCodePipelineServiceRole-us-east-1-dkpippe",
           "artifactStore": {
               "type": "S3",
               "location": "bucket"
           },
           "stages": [
               {
                   "name": "Source",
                   "actions": [
                       {
                           "name": "Source",
                           "actionTypeId": {
                               "category": "Source",
                               "owner": "AWS",
                               "provider": "CodeCommit",
                               "version": "1"
                           },
                           "runOrder": 1,
                           "configuration": {
                               "BranchName": "main",
   ```

```
                        "OutputArtifactFormat": "CODE_ZIP",
                        "PollForSourceChanges": "true",
                        "RepositoryName": "MyDemoRepo"
                    },
                    "outputArtifacts": [
                        {
                            "name": "SourceArtifact"
                        }
                    ],
                    "inputArtifacts": [],
                    "region": "us-east-1",
                    "namespace": "SourceVariables"
                }
            ]
        },
        {
            "name": "Build",
            "actions": [
                {
                    "name": "Build",
                    "actionTypeId": {
                        "category": "Build",
                        "owner": "AWS",
                        "provider": "CodeBuild",
                        "version": "1"
                    },
                    "runOrder": 1,
                    "configuration": {
                        "ProjectName": "MyBuildProject"
                    },
                    "outputArtifacts": [
                        {
                            "name": "BuildArtifact"
                        }
                    ],
                    "inputArtifacts": [
                        {
                            "name": "SourceArtifact"
                        }
                    ],
                    "region": "us-east-1",
                    "namespace": "BuildVariables"
                }
            ]
```

```
            }
        ],
        "version": 1,
        "executionMode": "QUEUED"
    }
}
```

3. 如果您正在使用通过 get-pipeline 命令检索到的管道结构，则必须在 JSON 文件中修改结构。您必须从文件中删除 `metadata` 行以便 update-pipeline 命令可以使用它。从 JSON 文件中的管道结构中删除该部分（`"metadata": { }` 行以及 `"created"`、`"pipelineARN"` 和 `"updated"` 字段）。

   例如，从结构中删除以下各行：

```
"metadata": {
  "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
  "created": "date",
  "updated": "date"
  }
```

   保存该文件。

4. 要应用更改，请运行 update-pipeline 命令，指定管道 JSON 文件：

> **Important**
>
> 务必在文件名前包含 `file://`。此命令中需要该项。

```
aws codepipeline update-pipeline --cli-input-json file://pipeline.json
```

该命令会返回编辑后的管道的整个结构。

> **Note**
>
> update-pipeline 命令会停止管道。如果在运行 update-pipeline 命令时正在通过管道运行修订，则该运行会被停止。您必须手动启动管道，通过更新后的管道运行此修订。

重试失败的阶段或阶段中失败的操作

您可以重试失败的阶段，而不必从头开始再次运行管道。为此，您可以重试阶段中失败的操作，也可以从阶段中的第一个操作开始重试所有操作。当重试阶段中失败的操作时，所有仍在进行的操作都会继续运行，而失败的操作将会重新触发。当从阶段中的第一个操作开始重试失败的阶段时，该阶段中不会有任何正在进行的操作。在重试某个阶段之前，它其中的操作必须全部失败，或者有些操作失败，有些操作成功。

> **Important**
>
> 重试失败的阶段时，会从其中第一个操作开始重试所有操作，而重试失败的操作则会重试阶段中所有失败的操作。这会覆盖同一执行中之前成功的操作的输出构件。
> 尽管构件可能会被覆盖，但之前成功的操作的执行历史记录仍会保留。

如果使用控制台查看管道，则可重试的阶段上将会出现重试阶段按钮或重试失败的操作按钮。

如果您使用的是 AWS CLI，则可以使用get-pipeline-state命令来确定是否有任何操作失败。

> **Note**
>
> 在以下情况下，您可能无法重试阶段：
>
> - 阶段中的所有操作都已成功，因此该阶段未处于失败状态。
> - 阶段失败后总体管道结构发生变化。
> - 阶段中的另一个重试尝试已在进行中。

主题

重试失败的阶段（控制台）

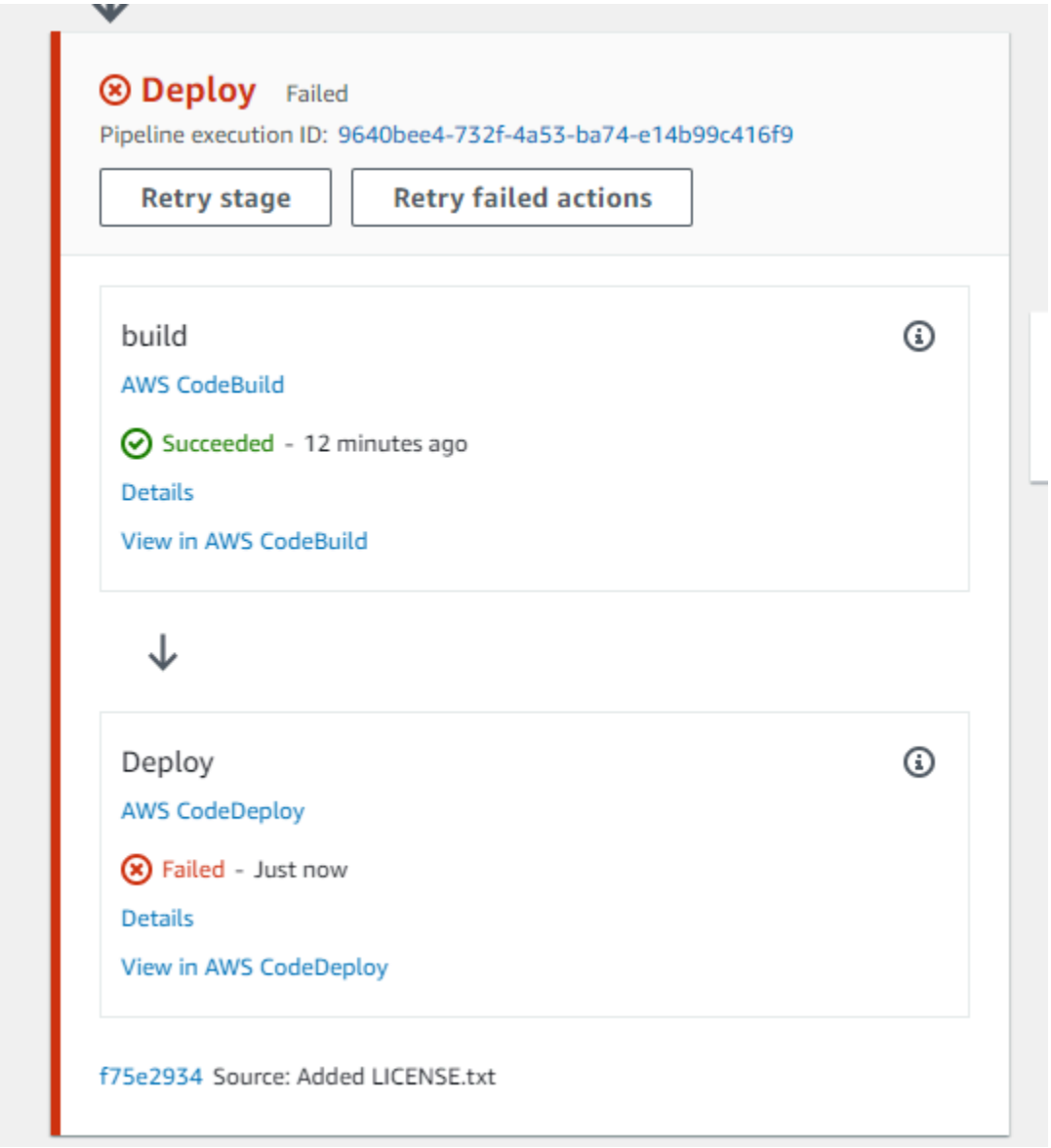
重试失败的阶段或阶段中失败的操作（控制台）

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

将显示与您的 AWS 账户关联的所有管道的名称。

2. 在名称中，选择管道的名称。
3. 找到包含失败操作的阶段，然后选择以下项之一：

   - 要重试阶段中的所有操作，请选择重试阶段。
   - 要仅重试阶段中失败的操作，请选择重试失败的操作。



如果该阶段中所有重试的操作都已成功完成，管道将继续运行。

重试失败的阶段 (CLI)

重试失败的阶段或阶段中失败的操作 (CLI)

要使用重试所有操作或所有失败的操作，请使用以下参数运行retry-stage-execution命令： AWS CLI

```
--pipeline-name <value>
--stage-name <value>
--pipeline-execution-id <value>
--retry-mode ALL_ACTIONS/FAILED_ACTIONS
```

> **Note**
>
> 您可以对 `retry-mode` 使用的值是 `FAILED_ACTIONS` 和 `ALL_ACTIONS`。

1. 在终端（Linux、macOS 或 Unix）或命令提示符 (Windows) 下，运行 retry-stage-execution 命令，如以下名为 `MyPipeline` 的管道示例中所示。

   ```
   aws codepipeline retry-stage-execution --pipeline-name MyPipeline --stage-name
    Deploy --pipeline-execution-id b59babff-5f34-EXAMPLE --retry-mode FAILED_ACTIONS
   ```

   输出会返回执行 ID：

   ```
   {
       "pipelineExecutionId": "b59babff-5f34-EXAMPLE"
   }
   ```

2. 您还可以使用 JSON 输入文件运行该命令。您首先要创建一个 JSON 文件来标识管道、包含失败操作的阶段以及该阶段中的最新管道执行。然后，运行带 `--cli-input-json` 参数的 retry-stage-execution 命令。要检索您需要用于 JSON 文件的详细信息，最简单的方法是使用 get-pipeline-state 命令。

   a. 在终端（Linux、macOS 或 Unix）或命令提示符 (Windows) 处，对管道运行 get-pipeline-state 命令。例如，对于名为的管道 MyFirstPipeline，您可以键入类似于以下内容的内容：

      ```
      aws codepipeline get-pipeline-state --name MyFirstPipeline
      ```

该命令的响应包括每个阶段的管道状态信息。在以下示例中，响应表示 Staging 阶段有一个或多个操作失败：

```
{
    "updated": 1427245911.525,
    "created": 1427245911.525,
    "pipelineVersion": 1,
    "pipelineName": "MyFirstPipeline",
    "stageStates": [
        {
            "actionStates": [...],
            "stageName": "Source",
            "latestExecution": {
                "pipelineExecutionId": "9811f7cb-7cf7-SUCCESS",
                "status": "Succeeded"
            }
        },
        {
            "actionStates": [...],
            "stageName": "Staging",
            "latestExecution": {
                "pipelineExecutionId": "3137f7cb-7cf7-EXAMPLE",
                "status": "Failed"
            }
        }
    ]
}
```

b. 在纯文本编辑器中，创建一个 JSON 格式文件，您需要记录以下内容：

- 包含失败操作的管道的名称
- 包含失败操作的阶段的名称
- 该阶段中最新管道执行的 ID
- 重试模式。

对于前面的 MyFirstPipeline 示例，您的文件将如下所示：

```
{
    "pipelineName": "MyFirstPipeline",
    "stageName": "Staging",
```

```
    "pipelineExecutionId": "3137f7cb-7cf7-EXAMPLE",
    "retryMode": "FAILED_ACTIONS"
}
```

c. 使用类似于 **retry-failed-actions.json** 的名称保存文件。

d. 调用您运行 retry-stage-execution 命令时创建的文件。例如：

> **Important**
>
> 务必在文件名前包含 `file://`。此命令中需要该项。

```
aws codepipeline retry-stage-execution --cli-input-json file://retry-failed-actions.json
```

e. 要查看重试的结果，请打开 CodePipeline 控制台并选择包含失败操作的管道，或者再次使用该get-pipeline-state命令。有关更多信息，请参阅 在中查看管道和详细信息 CodePipeline。

配置阶段回滚

你可以将一个阶段回滚到在该阶段成功的执行中。您可以预先配置阶段以便在出现故障时进行回滚，也可以手动回滚阶段。回滚操作将导致新的执行。为回滚选择的目标管道执行用于检索源版本和变量。

执行类型（标准执行或回滚）显示在管道历史记录、管道状态和管道执行详细信息中。

主题

- 回滚注意事项
- 手动回滚舞台
- 为自动回滚配置阶段
- 在执行列表中查看回滚状态
- 查看回滚状态详细信息

回滚注意事项

阶段回滚的注意事项如下：

- 您无法回滚源阶段。

- 只有在当前管道结构版本中启动之前的执行时，管道才能回滚到之前的执行。
- 您无法回滚到回滚执行类型的目标执行 ID。

手动回滚舞台

您可以使用控制台或 CLI 手动回滚阶段。只有在当前管道结构版本中启动之前的执行时，管道才能回滚到之前的执行。

您也可以将舞台配置为在失败时自动回滚，详情请参见为自动回滚配置阶段。

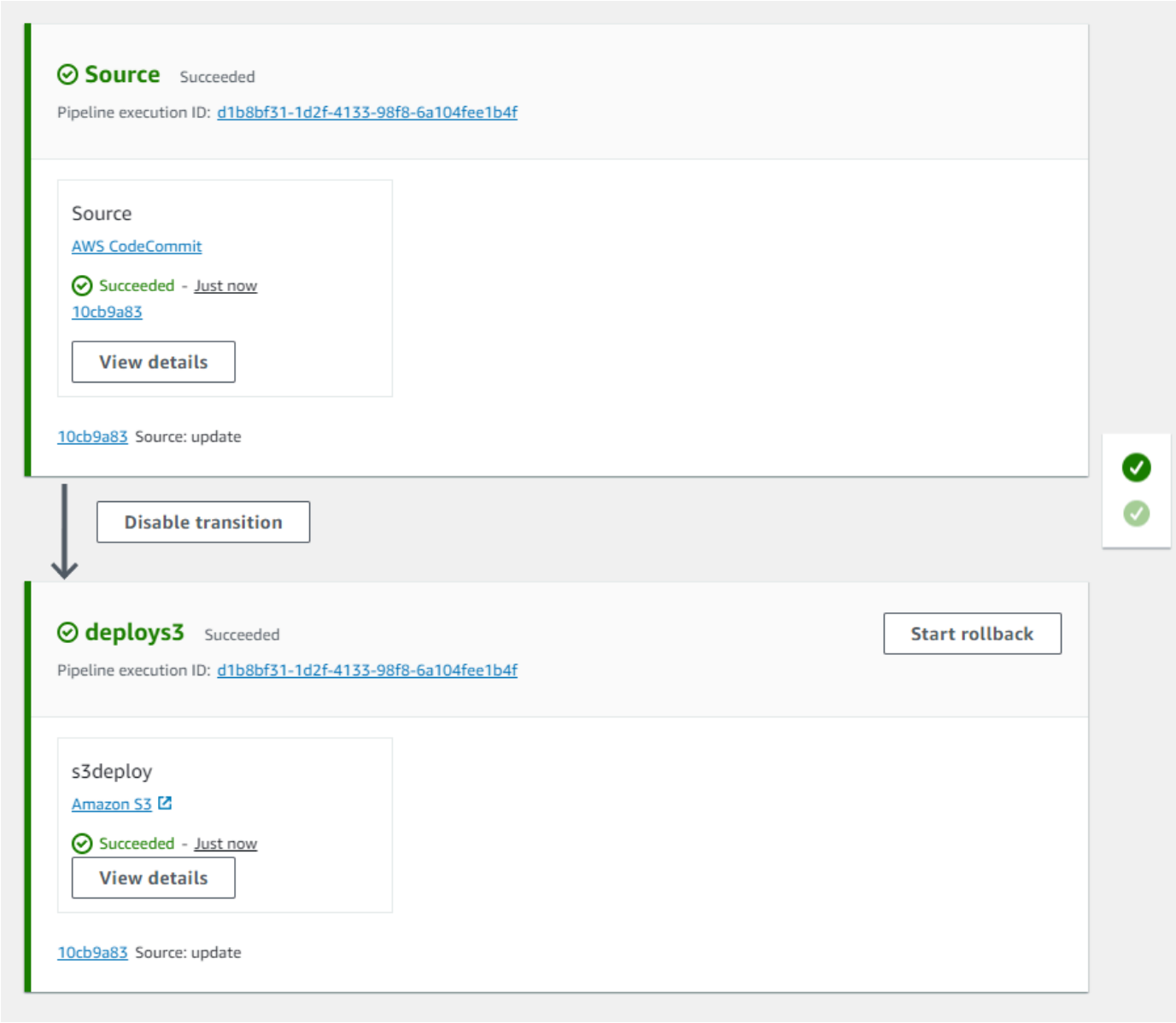
手动回滚舞台（控制台）

您可以使用控制台手动将阶段回滚到目标管道执行。回滚阶段后，控制台中的管道可视化上会显示一个 Rollback 标签。

手动回滚舞台（控制台）

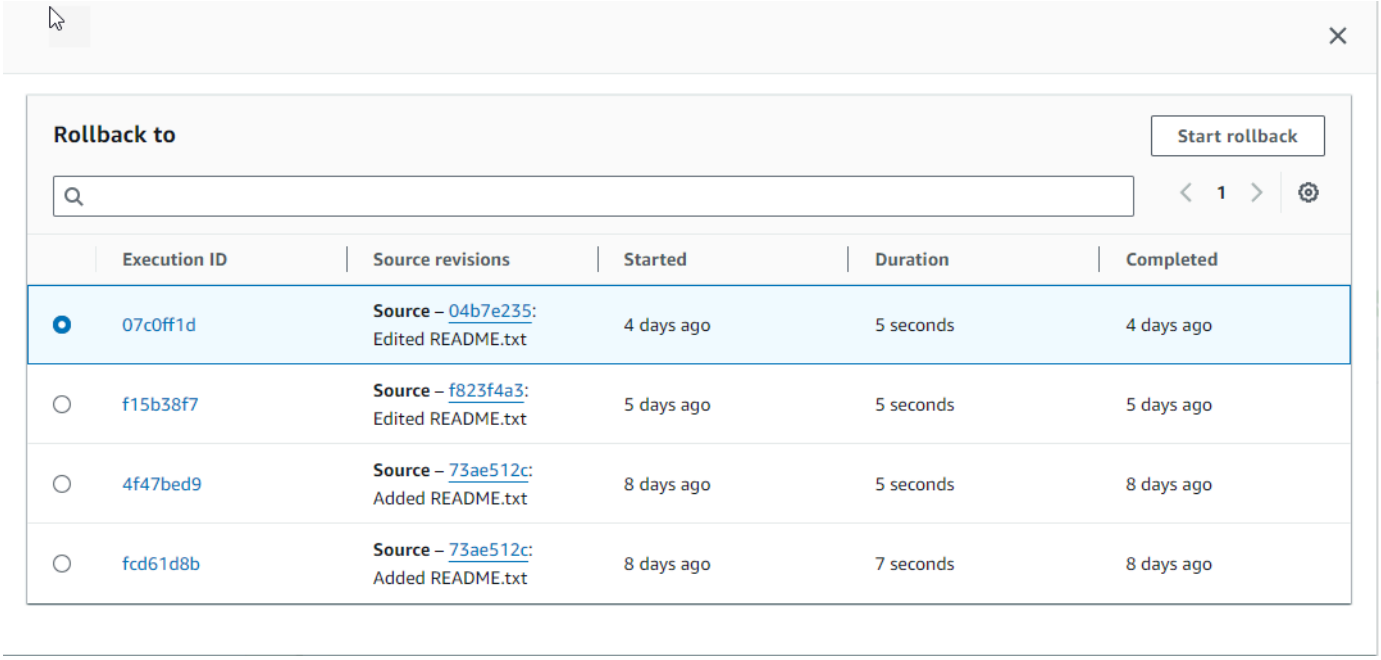
1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   将显示与您的 AWS 账户关联的所有管道的名称和状态。
2. 在名称中，选择包含要回滚的舞台的管道名称。

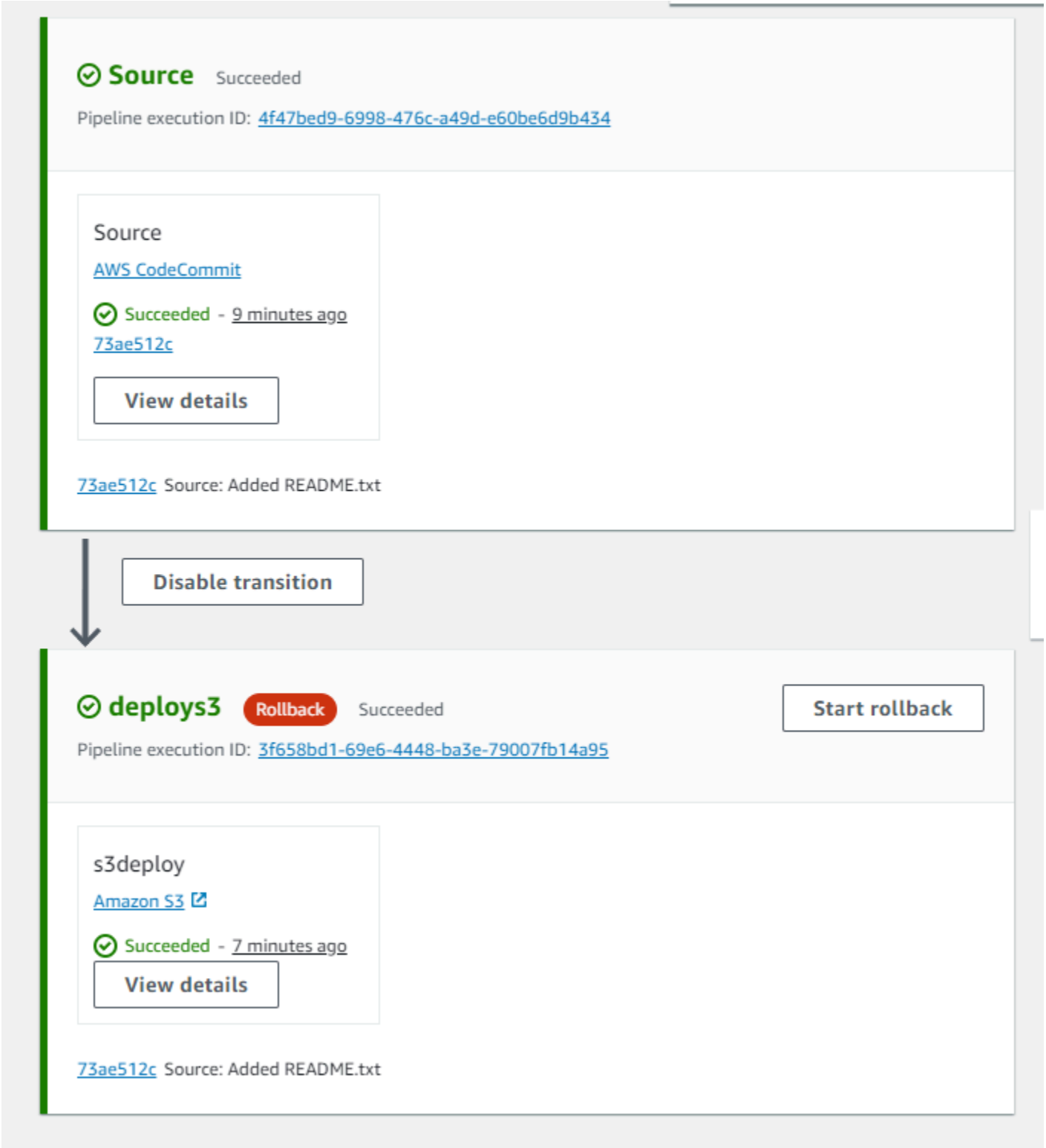


3. 在舞台上，选择 “开始回滚”。将显示 “回滚到页面”。
4. 选择要将舞台回滚到的目标执行。

> **Note**
>
> 可用的目标管道执行列表将是当前管道版本中从 2024 年 2 月 1 日开始的所有执行。



下图显示了使用新执行 ID 的回滚阶段的示例。



手动回滚阶段 (CLI)

要使用手动回滚阶段，请使用`rollback-stage`命令。 AWS CLI

您也可以手动回滚舞台，详情请参见手动回滚舞台。

> **Note**
>
> 可用的目标管道执行列表将是当前管道版本中从 2024 年 2 月 1 日开始的所有执行。

手动回滚阶段 (CLI)

1. 用于手动回滚的 CLI 命令需要先前在该阶段成功执行的管道的执行 ID。要获取您将指定的目标管道执行 ID，请使用带有筛选器的 list-pipeline-executions 命令，该筛选器将返回该阶段的成功执行。打开终端（Linux、macOS 或 Unix）或命令提示符 (Windows)，然后使用运行`list-pipeline-executions`命令，指定管道名称和舞台中成功执行的过滤器。 AWS CLI 在此示例中，输出将列出名为的管道 MyFirstPipeline 和名为的阶段中成功执行的管道执行情况`deploys3`。

   ```
   aws codepipeline list-pipeline-executions --pipeline-name MyFirstPipeline --filter
    succeededInStage={stageName=deploys3}
   ```

   在输出中，复制要为回滚指定的先前成功执行的执行 ID。在下一步中，您将使用它作为目标执行 ID。

2. 打开终端（Linux、macOS 或 Unix）或`rollback-stage`命令提示符 (Windows)，然后使用运行命令，指定管道名称、阶段名称和要回滚到的目标执行。 AWS CLI 例如，要为名为 Deploy 的管道回滚名为 Deploy 的阶段 *`MyFirstPipeline`*：

   ```
   aws codepipeline rollback-stage --pipeline-name MyFirstPipeline --stage-name Deploy
    --target-pipeline-execution-id bc022580-4193-491b-8923-9728dEXAMPLE
   ```

   输出返回新的回滚执行的执行 ID。这是一个单独的 ID，它使用指定目标执行的源版本和参数。

为自动回滚配置阶段

您可以将管道中的阶段配置为在失败时自动回滚。当该阶段失败时，该阶段将回滚到最近一次成功执行的状态。只有在当前管道结构版本中启动之前的执行时，管道才能回滚到之前的执行。由于自动回滚配置是管道定义的一部分，因此只有在管道阶段成功执行管道之后，您的管道阶段才会自动回滚。

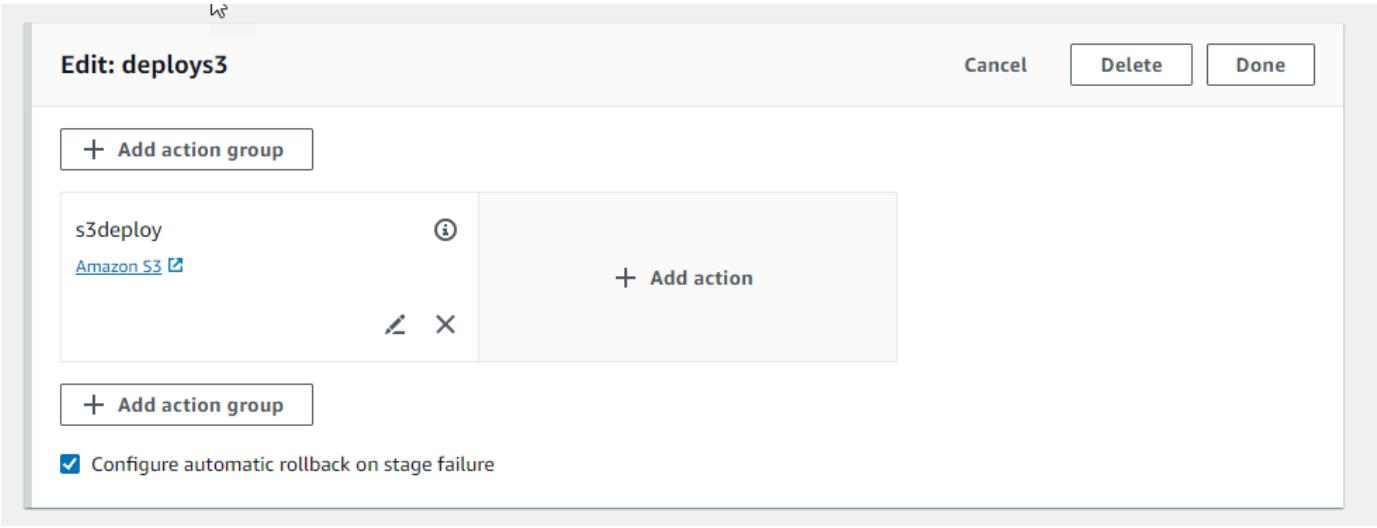
为自动回滚配置阶段（控制台）

您可以将阶段回滚到之前成功执行的指定阶段。有关更多信息，请参阅 CodePipeline API 指南RollbackStage中的。

为自动回滚配置阶段（控制台）

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   将显示与您的 AWS 账户关联的所有管道的名称和状态。
2. 在 Name 中，选择您要编辑的管道的名称。
3. 在管道详细信息页中，选择编辑。
4. 在“编辑”页面上，对于要编辑的操作，选择“编辑阶段”。
5. 选择“配置舞台失败时自动回滚”。将更改保存到您的管道中。



为自动回滚配置阶段 (CLI)

要使用 AWS CLI 将失败阶段配置为自动回滚到最近一次成功执行的阶段，请使用命令创建或更新管道，详见在中创建管道 CodePipeline和中在中编辑管道 CodePipeline。

- 打开终端（Linux、macOS 或 Unix）或命令提示符 (Windows)，然后使用运行`update-pipeline`命令，在管道结构中指定失败条件。 AWS CLI 以下示例为名为的分阶段配置自动回滚：`S3Deploy`

```
{
```

```
            "name": "S3Deploy",
            "actions": [
                {
                    "name": "s3deployaction",
                    "actionTypeId": {
                        "category": "Deploy",
                        "owner": "AWS",
                        "provider": "S3",
                        "version": "1"
                    },
                    "runOrder": 1,
                    "configuration": {
                        "BucketName": "static-website-bucket",
                        "Extract": "false",
                        "ObjectKey": "SampleApp.zip"
                    },
                    "outputArtifacts": [],
                    "inputArtifacts": [
                        {
                            "name": "SourceArtifact"
                        }
                    ],
                    "region": "us-east-1"
                }
            ],
            "onFailure": {
                "result": "ROLLBACK"
            }
        }
```

有关为阶段回滚配置故障条件的更多信息，请参阅 CodePipeline API 参考FailureConditions中的。

为自动回滚配置阶段 ()AWS CloudFormation

AWS CloudFormation 要使用将舞台配置为在失败时自动回滚，请使用`OnFailure`参数。失败时，该阶段将自动回滚到最近的成功执行。

```
OnFailure:
     Result: ROLLBACK
```

- 更新模板，如以下代码段所示。以下示例为名为的分阶段配置自动回滚：Release

```
AppPipeline:
  Type: AWS::CodePipeline::Pipeline
  Properties:
    RoleArn:
      Ref: CodePipelineServiceRole
    Stages:
      -
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
              -
                Name: SourceOutput
            Configuration:
              S3Bucket:
                Ref: SourceS3Bucket
              S3ObjectKey:
                Ref: SourceS3ObjectKey
            RunOrder: 1
      -
        Name: Release
        Actions:
          -
            Name: ReleaseAction
            InputArtifacts:
              -
                Name: SourceOutput
            ActionTypeId:
              Category: Deploy
              Owner: AWS
              Version: 1
              Provider: CodeDeploy
            Configuration:
              ApplicationName:
                Ref: ApplicationName
```

```
            DeploymentGroupName:
              Ref: DeploymentGroupName
          RunOrder: 1
     OnFailure:
          Result: ROLLBACK
  ArtifactStore:
    Type: S3
    Location:
      Ref: ArtifactStoreS3Location
    EncryptionKey:
      Id: arn:aws:kms:useast-1:ACCOUNT-ID:key/KEY-ID
      Type: KMS
  DisableInboundStageTransitions:
    -
      StageName: Release
      Reason: "Disabling the transition until integration tests are completed"
  Tags:
    - Key: Project
      Value: ProjectA
    - Key: IsContainerBased
      Value: 'true'
```

有关为阶段回滚配置故障条件的更多信息，请参阅OnFailure《AWS CloudFormation 用户指南》StageDeclaration中的。

在执行列表中查看回滚状态

您可以查看回滚执行的状态和目标执行 ID。

在执行列表中查看回滚状态（控制台）

您可以使用控制台在执行列表中查看回滚执行的状态和目标执行 ID。

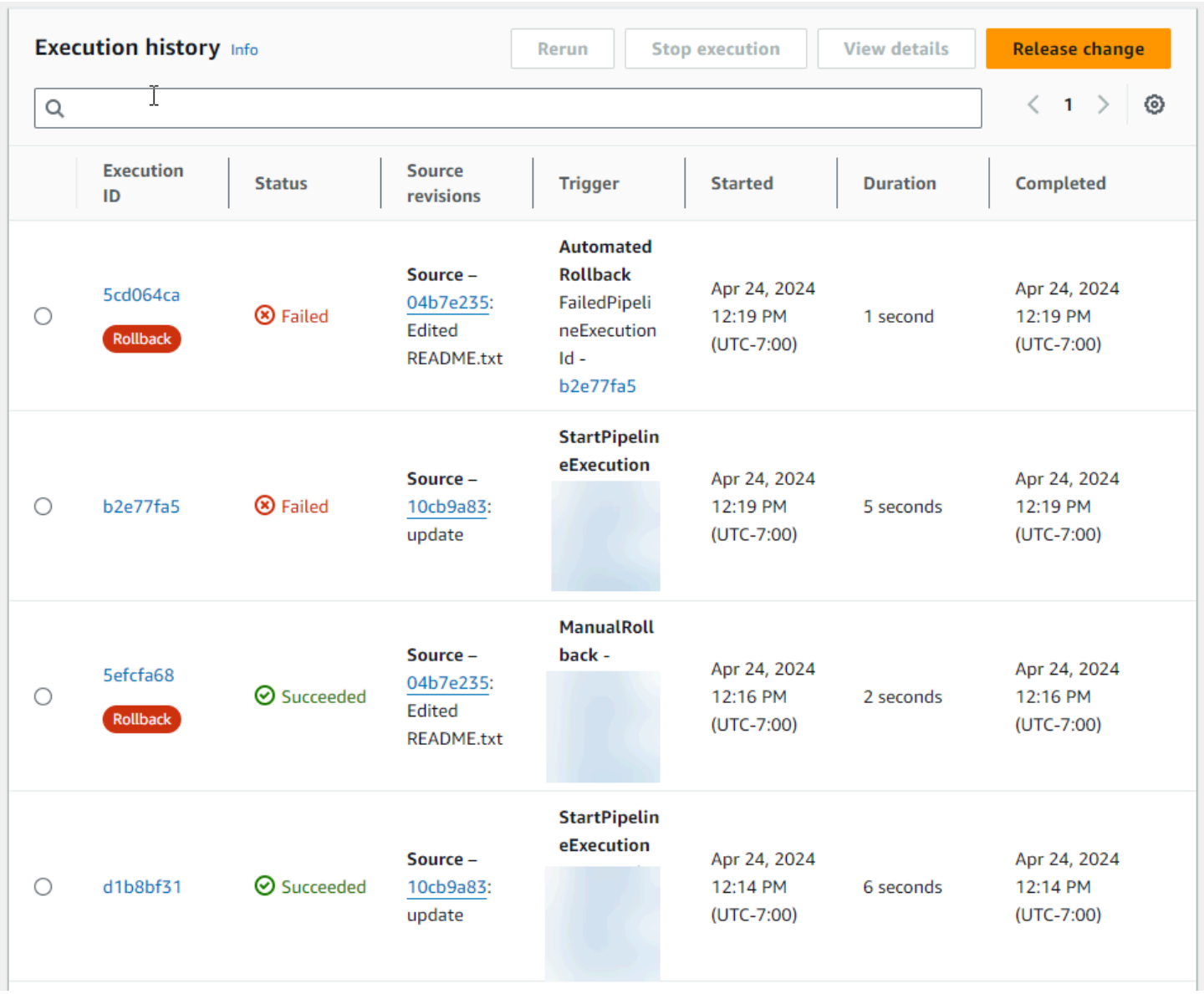
在执行列表中查看回滚执行状态（控制台）

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   将显示与您 AWS 账户 关联的所有管道的名称和状态。

2. 在名称中，选择要查看的管道的名称。

3. 选择 History (历史记录)。执行列表显示了 “Rollback” 标签。



选择要查看其详细信息的执行 ID。

使用 `list-pipeline-executions` (CLI) 查看回滚状态

您可以使用 CLI 查看回滚执行的状态和目标执行 ID。

- 打开终端（Linux、macOS 或 Unix）或命令提示符 (Windows)，并使用 AWS CLI 运行 list-pipeline-executions 命令：

```
aws codepipeline list-pipeline-executions --pipeline-name MyFirstPipeline
```

此命令返回与管道关联的所有已完成执行的列表。

以下示例显示了名为回滚执行启动管道*MyFirstPipeline*的管道的返回数据。

```
{
    "pipelineExecutionSummaries": [
        {
            "pipelineExecutionId": "eb7ebd36-353a-4551-90dc-18ca5EXAMPLE",
            "status": "Succeeded",
            "startTime": "2024-04-16T09:00:28.185000+00:00",
            "lastUpdateTime": "2024-04-16T09:00:29.665000+00:00",
            "sourceRevisions": [
                {
                    "actionName": "Source",
                    "revisionId": "revision_ID",
                    "revisionSummary": "Added README.txt",
                    "revisionUrl": "console-URL"
                }
            ],
            "trigger": {
                "triggerType": "ManualRollback",
                "triggerDetail": "{arn:aws:sts::<account_ID>:assumed-role/<role>"}"
            },
            "executionMode": "SUPERSEDED",
            "executionType": "ROLLBACK",
            "rollbackMetadata": {
                "rollbackTargetPipelineExecutionId":
 "f15b38f7-20bf-4c9e-94ed-2535eEXAMPLE"
            }
        },
        {
            "pipelineExecutionId": "fcd61d8b-4532-4384-9da1-2aca1EXAMPLE",
            "status": "Succeeded",
            "startTime": "2024-04-16T08:58:56.601000+00:00",
            "lastUpdateTime": "2024-04-16T08:59:04.274000+00:00",
            "sourceRevisions": [
                {
                    "actionName": "Source",
                    "revisionId": "revision_ID",
                    "revisionSummary": "Added README.txt",
```

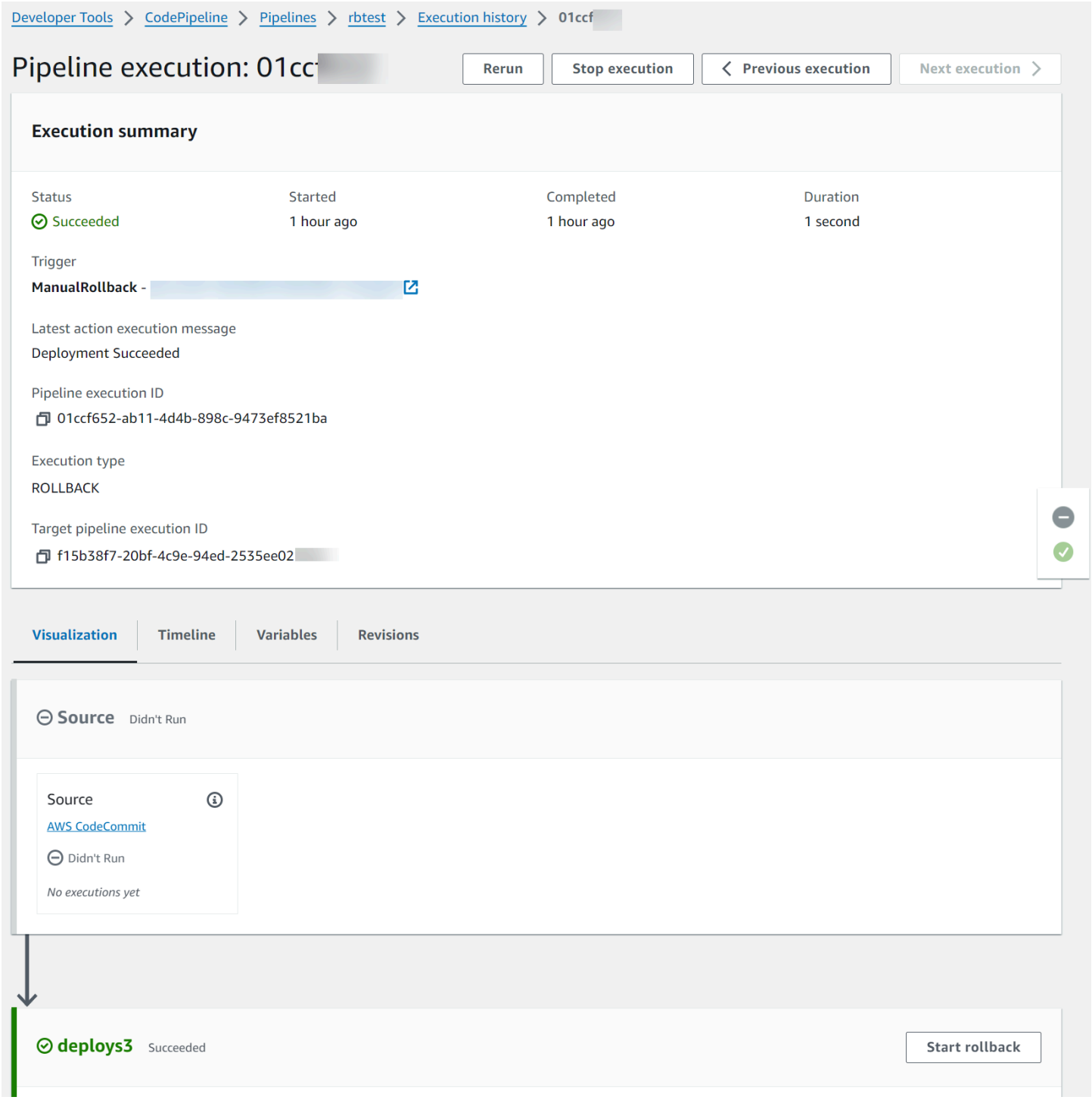
```
                    "revisionUrl": "console_URL"
                }
            ],
            "trigger": {
                "triggerType": "StartPipelineExecution",
                "triggerDetail": "arn:aws:sts::<account_ID>:assumed-role/<role>"
            },
            "executionMode": "SUPERSEDED"
        },
        {
            "pipelineExecutionId": "5cd064ca-bff7-425f-8653-f41d9EXAMPLE",
            "status": "Failed",
            "startTime": "2024-04-24T19:19:50.781000+00:00",
            "lastUpdateTime": "2024-04-24T19:19:52.119000+00:00",
            "sourceRevisions": [
                {
                    "actionName": "Source",
                    "revisionId": "<revision_ID>",
                    "revisionSummary": "Edited README.txt",
                    "revisionUrl": "<revision_URL>"
                }
            ],
            "trigger": {
                "triggerType": "AutomatedRollback",
                "triggerDetail": "{\"FailedPipelineExecutionId\":
\"b2e77fa5-9285-4dea-ae66-4389EXAMPLE\"}"
            },
            "executionMode": "SUPERSEDED",
            "executionType": "ROLLBACK",
            "rollbackMetadata": {
                "rollbackTargetPipelineExecutionId": "5efcfa68-d838-4ca7-
a63b-4a743EXAMPLE"
            }
         },
```

查看回滚状态详细信息

您可以查看回滚执行的状态和目标执行 ID。

在详情页面（控制台）上查看回滚状态

您可以使用控制台查看回滚执行的状态和目标管道执行 ID。



使用 **`get-pipeline-execution`** (CLI) 查看回滚详情

已回滚的管道执行将显示在输出中，以获取管道执行。

- 要查看管道的详细信息，请运行 get-pipeline-execution 命令，指定管道的唯一名称。例如，要查看名为的管道的详细信息 *MyFirstPipeline*，请输入以下内容：

```
aws codepipeline get-pipeline-execution --pipeline-name MyFirstPipeline --pipeline-execution-id 3f658bd1-69e6-4448-ba3e-79007EXAMPLE
```

该命令会返回管道结构。

以下示例显示了管道中名为的部分返回的数据 *MyFirstPipeline*，其中显示了回滚执行 ID 和元数据。

```
{
    "pipelineExecution": {
        "pipelineName": "MyFirstPipeline",
        "pipelineVersion": 6,
        "pipelineExecutionId": "2004a94e-8b46-4c34-a695-c8d20EXAMPLE",
        "status": "Succeeded",
        "artifactRevisions": [
            {
                "name": "SourceArtifact",
                "revisionId": "<ID>",
                "revisionSummary": "Added README.txt",
                "revisionUrl": "<console_URL>"
            }
        ],
        "trigger": {
            "triggerType": "ManualRollback",
            "triggerDetail": "arn:aws:sts::<account_ID>:assumed-role/<role>"
        },
        "executionMode": "SUPERSEDED",
        "executionType": "ROLLBACK",
        "rollbackMetadata": {
            "rollbackTargetPipelineExecutionId": "4f47bed9-6998-476c-a49d-e60beEXAMPLE"
        }
    }
}
```

使用 **get-pipeline-state** (CLI) 查看回滚状态

已回滚的管道执行将显示在输出中，用于获取管道状态。

- 要查看管道的详细信息，请运行 get-pipeline-state 命令，指定管道的唯一名称。例如，要查看名为的管道的状态详细信息 *MyFirstPipeline*，请输入以下内容：

```
aws codepipeline get-pipeline-state --name MyFirstPipeline
```

以下示例显示了使用回滚执行类型的返回数据。

```
{
    "pipelineName": "MyFirstPipeline",
    "pipelineVersion": 7,
    "stageStates": [
        {
            "stageName": "Source",
            "inboundExecutions": [],
            "inboundTransitionState": {
                "enabled": true
            },
            "actionStates": [
                {
                    "actionName": "Source",
                    "currentRevision": {
                        "revisionId": "<Revision_ID>"
                    },
                    "latestExecution": {
                        "actionExecutionId": "13bbd05d-
b439-4e35-9c7e-887cb789b126",
                        "status": "Succeeded",
                        "summary": "update",
                        "lastStatusChange": "2024-04-24T20:13:45.799000+00:00",
                        "externalExecutionId": "10cbEXAMPLEID"
                    },
                    "entityUrl": "console-url",
                    "revisionUrl": "console-url"
                }
            ],
            "latestExecution": {
                "pipelineExecutionId": "cf95a8ca-0819-4279-ae31-03978EXAMPLE",
                "status": "Succeeded"
            }
        },
        {
            "stageName": "deploys3",
```

```
            "inboundExecutions": [],
            "inboundTransitionState": {
                "enabled": true
            },
            "actionStates": [
                {
                    "actionName": "s3deploy",
                    "latestExecution": {
                        "actionExecutionId":
 "3bc4e3eb-75eb-45b9-8574-8599aEXAMPLE",
                        "status": "Succeeded",
                        "summary": "Deployment Succeeded",
                        "lastStatusChange": "2024-04-24T20:14:07.577000+00:00",
                        "externalExecutionId": "mybucket/SampleApp.zip"
                    },
                    "entityUrl": "console-URL"
                }
            ],
            "latestExecution": {
                "pipelineExecutionId": "fdf6b2ae-1472-4b00-9a83-1624eEXAMPLE",
                "status": "Succeeded",
                "type": "ROLLBACK"
            }
        }
    ],
    "created": "2024-04-15T21:29:01.635000+00:00",
    "updated": "2024-04-24T20:12:24.480000+00:00"
}
```

在中处理动作 CodePipeline

在中 AWS CodePipeline，动作是管道中某个阶段序列的一部分。它是在该阶段对项目执行的任务。管道操作以特定顺序、依次或并行发生，具体方式在该阶段的配置中确定。

CodePipeline 为六种类型的操作提供支持：

- 来源
- 构建
- 测试
- 部署
- 审批
- 调用

有关您可以根据操作类型集成到管道中的 AWS 服务 以及合作伙伴产品和服务的信息，请参阅与动 CodePipeline 作类型的集成。

主题

使用操作类型

操作类型是您作为提供方，使用 AWS CodePipeline中支持的集成模型之一为客户创建的预配置操作。

您可以请求、查看和更新操作类型。如果操作类型是为作为所有者的账户创建的，则可以使用 AWS CLI 来查看或更新您的操作类型属性和结构。如果您是操作类型的提供者或所有者，则您的客户可以选择该操作，并在该操作可用后将其添加到他们的管道中 CodePipeline。

> **Note**
>
> 您应创建 owner 字段为 custom 的作业，以便与作业辅助角色一起运行。您不能使用集成模型来创建它们。有关自定义操作的信息，请参阅在中创建和添加自定义操作 CodePipeline。

操作类型组件

以下组件构成了操作类型。

- 操作类型 ID：ID 由类别、所有者、提供方和版本组成。以下示例显示了一个操作类型 ID，其所有者为 ThirdParty，类别为 Test，提供方名为 TestProvider，版本为 1。

```
{
    "Category": "Test",
    "Owner": "ThirdParty",
    "Provider": "TestProvider",
    "Version": "1"
},
```

- 执行程序配置：创建操作时指定的集成模型或操作引擎。为操作类型指定执行程序时，可以选择以下两种类型之一：
  - Lambda：操作类型所有者将集成写为 Lambda 函数， CodePipeline 只要有任务可用于操作，就会调用该函数。
  - JobWorker：操作类型所有者以在客户管道上轮询可用作业的作业工作人员的身份编写集成。然后，作业工作人员运行作业，并使用 CodePipeline API 将作业结果提交回给 CodePipeline 。

  > **Note**
  >
  > 作业辅助角色集成模型不是首选的集成模型。

- 输入和输出构件：操作类型所有者为操作客户指定的构件的限制。
- 权限：一种权限策略，用于指定可以访问第三方操作类型的客户。可用的权限策略取决于为操作类型选择的集成模型。
- URL：客户可以与之交互的资源的深层链接，如操作类型所有者的配置页面。

主题

- 请求操作类型
- 向管道添加可用的操作类型（控制台）
- 查看操作类型
- 更新操作类型

请求操作类型

当第三方提供者请求新的 CodePipeline 操作类型时，将在中为操作类型所有者创建操作类型 CodePipeline，所有者可以管理和查看该操作类型。

操作类型可以是私有操作，也可以是公有操作。创建操作类型后，操作为私有操作。要请求将操作类型更改为公共操作，请联系 CodePipeline 服务团队。

在为 CodePipeline 团队创建操作定义文件、执行者资源和操作类型请求之前，必须选择集成模型。

步骤 1：选择您的集成模型

选择您的集成模型，然后创建该模型的配置。选择集成模型后，您必须配置自己的集成资源。

- 对于 Lambda 集成模型，您应创建 Lambda 函数并添加权限。向您的集成商 Lambda 函数添加权限，以向服务提供使用服务 CodePipeline 主体调用 CodePipeline 该函数的权限:. `codepipeline.amazonaws.com` 可以使用 AWS CloudFormation 或命令行添加权限。
  - 使用 AWS CloudFormation添加权限的示例：

```
  CodePipelineLambdaBasedActionPermission:
    Type: 'AWS::Lambda::Permission'
    Properties:
      Action: 'lambda:invokeFunction'
      FunctionName: {"Fn::Sub": "arn:${AWS::Partition}:lambda:${AWS::Region}:
${AWS::AccountId}:function:function-name"}
      Principal: codepipeline.amazonaws.com
```

  - 命令行文档
- 对于求职者集成模型，您可以使用允许的账户列表创建集成，在这些账户中，作业工作人员使用 CodePipeline API 轮询职位。

步骤 2：创建操作类型定义文件

您应使用 JSON 格式，在操作类型定义文件中定义一种操作类型。在该文件中，您可以包括操作类别、用于管理操作类型的集成模型，以及配置属性。

> **Note**
>
> 在创建公有操作后，您无法将 `properties` 下的操作类型属性从 `optional` 更改为 `required`。您也不能更改 `owner`。

有关操作类型定义文件参数的更多信息，请参阅 CodePipeline API 参考UpdateActionType中的ActionTypeDeclaration和。

操作类型定义文件中有八个部分：

- `description`：要更新的操作类型的描述。
- `executor`：与使用支持的集成模型（`Lambda` 或 `job worker`）创建的操作类型的执行程序相关的信息。根据您的执行程序类型，您只能提供 `jobWorkerExecutorConfiguration` 或 `lambdaExecutorConfiguration`。
  - `configuration`：基于所选择的集成模型，用于操作类型配置的资源。对于 Lambda 集成模型，请使用 Lambda 函数 ARN。对于作业辅助角色集成模型，请使用作业辅助角色从中运行的账户或账户列表。
  - `jobTimeout`：作业的超时(以秒为单位)。一个操作执行可以包含多个作业。这是单个作业的超时，而不是整个操作执行的超时。

    > **Note**
    >
    > 对于 Lambda 集成模型，最大超时为 15 分钟。

  - `policyStatementsTemplate`：政策声明，它指定了 CodePipeline 客户账户中成功运行操作执行所需的权限。
  - `type`：用于创建和更新操作类型的集成模型，可以是 `Lambda` 或 `JobWorker`。
- `id`：操作类型的类别、所有者、提供方和版本 ID：
  - `category`：可在阶段中执行的操作类型：源、构建、部署、测试、调用或批准。
  - `provider`：被调用的操作类型的提供方，如提供方公司或产品名称。提供方名称应在创建操作类型时提供。

  - owner：被调用的操作类型的创建者：AWS 或 ThirdParty。
  - version：用于对操作类型进行版本控制的字符串。对于第一个版本，应将版本号设置为 1。
- inputArtifactDetails：预计来自管道上一阶段的构件数量。
- outputArtifactDetails：预计操作类型阶段的结果所产生的构件数量。
- permissions：用于识别有权使用操作类型的账户的详细信息。
- properties：完成项目任务所需的参数。
  - description：显示给用户的属性的说明。
  - optional：配置属性是否为可选。
  - noEcho：日志中是否省略由客户输入的字段值。如果true，则在与 GetPipeline API 请求一起返回时，该值将被删除。
  - key：配置属性是否为一个键。
  - queryable：属性是否与轮询一起使用。一个操作类型最多可以有一个可查询属性。如果它有一个可查询属性，该属性必须是必需的且不是私有的。
  - name：向用户显示的属性名称。
- urls：向您的用户 CodePipeline 显示网址列表。
  - entityUrlTemplate：操作类型的外部资源（如配置页面）的 URL。
  - executionUrlTemplate：最新操作执行的详细信息的 URL。
  - revisionUrlTemplate： CodePipeline控制台中显示的指向客户可以更新或更改外部操作配置的页面的 URL。
  - thirdPartyConfigurationUrl：页面的 URL，用户可在该页面中注册外部服务，并对该服务提供的操作执行初始配置。

以下代码展示了一个操作类型定义文件示例。

```
{
   "actionType": {
      "description": "string",
      "executor": {
         "configuration": {
            "jobWorkerExecutorConfiguration": {
               "pollingAccounts": [ "string" ],
               "pollingServicePrincipals": [ "string" ]
            },
            "lambdaExecutorConfiguration": {
```

```
                "lambdaFunctionArn": "string"
             }
          },
          "jobTimeout": number,
          "policyStatementsTemplate": "string",
          "type": "string"
       },
       "id": {
          "category": "string",
          "owner": "string",
          "provider": "string",
          "version": "string"
       },
       "inputArtifactDetails": {
          "maximumCount": number,
          "minimumCount": number
       },
       "outputArtifactDetails": {
          "maximumCount": number,
          "minimumCount": number
       },
       "permissions": {
          "allowedAccounts": [ "string" ]
       },
       "properties": [
          {
             "description": "string",
             "key": boolean,
             "name": "string",
             "noEcho": boolean,
             "optional": boolean,
             "queryable": boolean
          }
       ],
       "urls": {
          "configurationUrl": "string",
          "entityUrlTemplate": "string",
          "executionUrlTemplate": "string",
          "revisionUrlTemplate": "string"
       }
    }
}
```

第 3 步：注册您的集成 CodePipeline

要向注册您的操作类型 CodePipeline，请联系 CodePipeline 服务团队并提出您的请求。

CodePipeline 服务团队通过更改服务代码库来注册新的操作类型集成。 CodePipeline 登记了两项新诉讼：一项公共诉讼和一项私人诉讼。您应使用私有操作进行测试，然后在准备就绪后，激活公有操作来为客户流量提供服务。

注册 Lambda 集成请求

- 使用以下表格向 CodePipeline 服务团队发送请求。

```
This issue will track the onboarding of [Name] in CodePipeline.


[Contact engineer] will be the primary point of contact for this integration.

Name of the action type as you want it to appear to customers: Example.com Testing

Initial onboard checklist:

1. Attach an action type definition file in JSON format. This includes the schema
 for the action type

2. A list of test accounts for the allowlist which can access the new action type
 [{account, account_name}]

3. The Lambda function ARN

4. List of AWS ## where your action will be available

5. Will this be available as a public action?
```

注册作业辅助角色集成请求

- 使用以下表格向 CodePipeline 服务团队发送请求。

```
This issue will track the onboarding of [Name] in CodePipeline.

[Contact engineer] will be the primary point of contact for this integration.
```

```
Name of the action type as you want it to appear to customers: Example.com Testing

Initial onboard checklist:

1. Attach an action type definition file in JSON format. This includes the schema
 for the action type.

2. A list of test accounts for the allowlist which can access the new action type
 [{account, account_name}]

3. URL information:
Website URL: https://www.example.com/%TestThirdPartyName%/%TestVersionNumber%

Example URL pattern where customers will be able to review their configuration
 information for the action: https://www.example.com/%TestThirdPartyName%/
%customer-ID%/%CustomerActionConfiguration%

Example runtime URL pattern: https://www.example.com/%TestThirdPartyName%/
%customer-ID%/%TestRunId%

4. List of AWS ## where your action will be available

5. Will this be available as a public action?
```

步骤 4：激活您的新集成

当您准备好公开使用新的集成时，请联系 CodePipeline 服务团队。

向管道添加可用的操作类型（控制台）

您可以将自己的操作类型添加到管道中，以便对其进行测试。为此，您可以创建新管道或编辑现有管道。

> **Note**
>
> 如果您的操作类型是源、构件或部署类操作，则可以通过创建管道来添加该操作。如果您的操作类型属于测试类别，则您必须通过编辑现有管道来添加。

从 CodePipeline 控制台向现有管道添加操作类型

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在管道列表中，选择要在其中添加操作类型的管道。
3. 在管道的摘要视图页面中，选择编辑。
4. 选择编辑阶段。在要向其添加操作类型的阶段中，选择添加操作组。此时将显示编辑操作页面。
5. 在编辑操作页面的操作名称下，输入操作的名称。这是为管道中的阶段显示的名称。
6. 在操作提供程序中，从列表中选择您的操作类型。

   请注意，列表中的值基于操作类型定义文件中指定的 `provider`。
7. 在输入构件中，按照以下格式输入构件名称：

   `Artifactname``::`*`FileName`*

   请注意，所允许的最小和最大数量基于操作类型定义文件中指定的 `inputArtifactDetails`。
8. 选择连接到 <Action_Name>。

   此时将打开一个浏览器窗口，并连接到您为操作类型创建的网站。
9. 以客户身份登录您的网站，完成客户为使用您的操作类型而应采取的步骤。您的步骤将会根据操作类别、网站和配置而有所不同，但通常会包含一个完成操作，使客户返回编辑操作页面。
10. 在 CodePipeline “编辑” 操作页面中，将显示该操作的其他配置字段。所显示的字段是您在操作定义文件中指定的配置属性。在为您的操作类型自定义的字段中输入信息。

    例如，如果操作定义文件指定了一个名为 `Host` 的属性，则操作的编辑操作页面上会显示一个带有主机标签的字段。
11. 在输出构件中，按照以下格式输入构件名称：

    `Artifactname``::`*`FileName`*

    请注意，所允许的最小和最大数量基于操作类型定义文件中指定的 `outputArtifactDetails`。
12. 选择完成以返回管道详细信息页面。

> **Note**
>
> 您的客户可以选择使用 CLI 将操作类型添加到他们的管道中。

13. 要测试您的操作，请向管道源阶段中指定的源提交更改，或者按照手动启动管道中的步骤进行操作。

要创建包含您的操作类型的管道，请按照在中创建管道 CodePipeline中的步骤操作，并从您将测试的众多阶段中选择自己的操作类型。

查看操作类型

您可以使用 CLI 查看操作类型。使用 get-action-type 命令可以查看使用集成模型创建的操作类型。

查看操作类型

1. 创建一个输入 JSON 文件并将该文件命名为 `file.json`。以 JSON 格式添加您的操作类型 ID，如以下示例所示。

```
{
    "category": "Test",
    "owner": "ThirdParty",
    "provider": "TestProvider",
    "version": "1"
}
```

2. 在终端窗口或命令行中，运行 get-action-type 命令。

```
aws codepipeline get-action-type --cli-input-json file://file.json
```

此命令会返回操作类型的操作定义输出。此示例显示了一个使用 Lambda 集成模型创建的操作类型。

```
{
    "actionType": {
        "executor": {
            "configuration": {
                "lambdaExecutorConfiguration": {
                    "lambdaFunctionArn": "arn:aws:lambda:us-west-2:<account-
id>:function:my-function"
                }
            },
            "type": "Lambda"
        },
        "id": {
```

```
            "category": "Test",
            "owner": "ThirdParty",
            "provider": "TestProvider",
            "version": "1"
        },
        "inputArtifactDetails": {
            "minimumCount": 0,
            "maximumCount": 1
        },
        "outputArtifactDetails": {
            "minimumCount": 0,
            "maximumCount": 1
        },
        "permissions": {
            "allowedAccounts": [
                "<account-id>"
            ]
        },
        "properties": []
    }
}
```

更新操作类型

您可以使用 CLI 来编辑使用集成模型创建的操作类型。

对于公有操作类型，您无法更新所有者，也无法将可选属性更改为必需属性，并且只能添加新的可选属性。

1. 使用 `get-action-type` 命令获取您的操作类型的结构。复制该结构。
2. 创建一个输入 JSON 文件并将其命名为 `action.json`。将您在上一步中复制的操作类型结构粘贴到其中。更改您要更新的任何参数。您也可以添加可选的参数。

   有关输入文件参数的更多信息，请参阅步骤 2：创建操作类型定义文件中的操作定义文件。

   下面的示例展示了如何更新使用 Lambda 集成模型创建的操作类型。此示例中将做出以下更改：

   - 将 `provider` 名称更改为 `TestProvider1`。
   - 添加 900 秒的作业超时限制。
   - 添加名为 `Host` 的操作配置属性，该属性将向使用该操作客户显示。

```
{
    "actionType": {
        "executor": {
            "configuration": {
                "lambdaExecutorConfiguration": {
                    "lambdaFunctionArn": "arn:aws:lambda:us-west-2:<account-
id>:function:my-function"
                }
            },
            "type": "Lambda",
            "jobTimeout": 900
        },
        "id": {
            "category": "Test",
            "owner": "ThirdParty",
            "provider": "TestProvider1",
            "version": "1"
        },
        "inputArtifactDetails": {
            "minimumCount": 0,
            "maximumCount": 1
        },
        "outputArtifactDetails": {
            "minimumCount": 0,
            "maximumCount": 1
        },
        "permissions": {
            "allowedAccounts": [
                "account-id"
            ]
        },
        "properties": {
         "description": "Owned build action parameter description",
         "optional": true,
         "noEcho": false,
         "key": true,
         "queryable": false,
         "name": "Host"
         }
    }
}
```

3. 在终端或命令行中，运行 update-action-type 命令。

```
aws codepipeline update-action-type --cli-input-json file://action.json
```

此命令会返回与更新后的参数相匹配的操作类型输出。

在中创建和添加自定义操作 CodePipeline

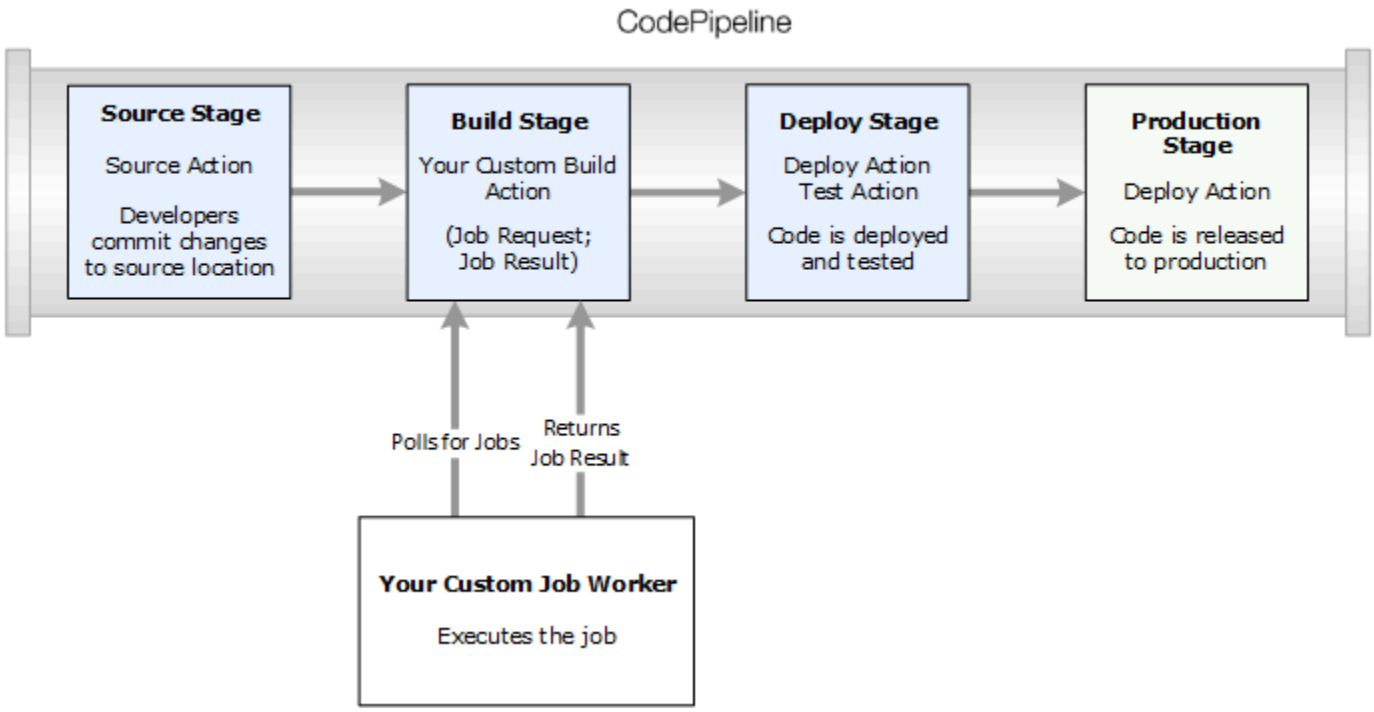
AWS CodePipeline 包括许多操作，可帮助您为自动发布流程配置构建、测试和部署资源。如果您的发布过程包括默认操作中不包含的活动，例如内部开发的构建过程或测试套件，则可以为此目的创建自定义操作，并将其包含在管道中。您可以使用在 AWS CLI 与您的 AWS 账户关联的管道中创建自定义操作。

您可以为以下操作类别创建自定义 AWS CodePipeline 操作：

- 用于构建或转换项目的自定义构建操作
- 用于将项目部署到一个或多个服务器、网站或存储库的自定义部署操作
- 用于配置和运行自动化测试的自定义测试操作
- 用于运行函数的自定义调用操作

在创建自定义操作时，您还必须创建一个作业辅助角色，该工作人员将轮询 CodePipeline 此自定义操作的任务请求，执行该作业，并将状态结果返回到 CodePipeline。该作业工作人员可以位于任何计算机或资源上，前提是它可以访问的公共终端节点 CodePipeline。要轻松管理访问权限和安全性，请考虑在 Amazon EC2 实例上托管您的作业辅助角色。

下图展示了包含自定义构建操作的管道的概要视图：



当管道包含作为阶段一部分的自定义操作时，该管道将创建一个作业请求。自定义作业辅助角色会检测到该请求并执行该作业（在本示例中，是使用第三方构建软件的自定义流程）。操作完成后，作业辅助角色会返回成功结果或失败结果。如果收到成功结果，管道将向下一个操作提供修订及其构件。如果返回失败结果，管道不会向管道中的下一个操作提供修订。

> **Note**
>
> 这些说明假设您已经完成了入门 CodePipeline中的步骤。

主题

- 创建自定义操作
- 为自定义操作创建作业辅助角色
- 向管道添加自定义操作

创建自定义操作

要使用创建自定义操作 AWS CLI

1. 打开文本编辑器，为您的自定义操作创建一个 JSON 文件，其中包括操作类别、操作提供程序以及您的自定义操作所需的任何设置。例如，要创建一个只需一个属性的自定义生成操作，您的 JSON 文件可能类似下面这样：

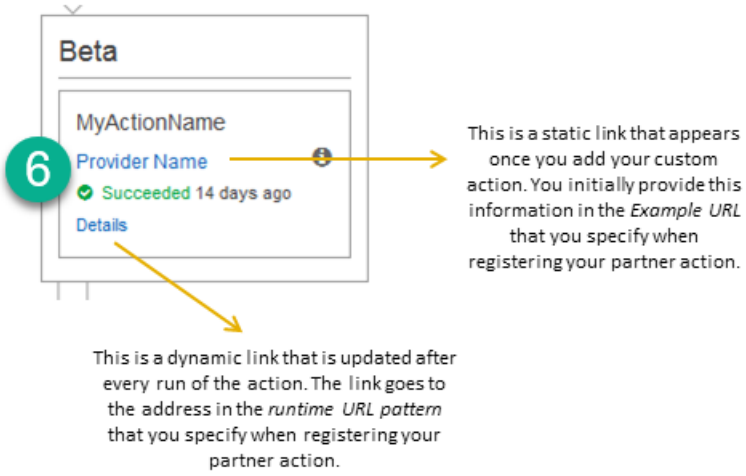
```
{
    "category": "Build",
    "provider": "My-Build-Provider-Name",
    "version": "1",
    "settings": {
        "entityUrlTemplate": "https://my-build-instance/job/{Config:ProjectName}/",
        "executionUrlTemplate": "https://my-build-instance/job/
{Config:ProjectName}/lastSuccessfulBuild/{ExternalExecutionId}/"
    },
    "configurationProperties": [{
        "name": "ProjectName",
        "required": true,
        "key": true,
        "secret": false,
        "queryable": false,
        "description": "The name of the build project must be provided when this
 action is added to the pipeline.",
        "type": "String"
    }],
    "inputArtifactDetails": {
        "maximumCount": integer,
        "minimumCount": integer
    },
    "outputArtifactDetails": {
        "maximumCount": integer,
        "minimumCount": integer
    },
    "tags": [{
      "key": "Project",
      "value": "ProjectA"
    }]
}
```

此示例通过在自定义操作上包含 Project 标签键和 ProjectA 值来为自定义操作添加标记。有关在中为资源添加标签的更多信息 CodePipeline，请参阅标记资源。

JSON 文件中有两个属性：entityUrlTemplate 和 executionUrlTemplate。只要配置属性是必需的且不为私有，您就可以遵照以下格式在 URL 模板内引用自定义操作的配置属性中的名称：{Config:*name*}。例如，在上面的示例中，该entityUrlTemplate值指的是配置属性*ProjectName*。

- entityUrlTemplate：用于提供操作服务提供方相关信息的静态链接。在示例中，构建系统包含一个指向每个构建项目的静态链接。链接格式可能不同，具体取决于您的构建提供程序（或者如果您创建的是其他操作类型，比如测试，则取决于其他服务提供方）。您必须提供此链接格式，以便在添加自定义操作时，用户可以选择此链接来打开浏览器，从而打开您的网站上提供生成项目 (或测试环境) 具体细节的页面。
- executionUrlTemplate：将会使用有关当前或最新运行的操作进行更新的动态链接。当您的自定义作业辅助角色更新作业的状态（例如成功、失败或正在进行）时，它还将提供一个 externalExecutionId 用于完成此链接。此链接可用于提供有关操作运行的详细信息。

例如，当您查看管道中的操作时，您将看到以下两个链接：



1

此静态链接在您添加自定义操作时显示，指向 entityUrlTemplate 中您在创建自定义操作时指定的地址。

2

此动态链接在每次操作运行后更新，指向 `executionUrlTemplate` 中您在创建自定义操作时指定的地址。

有关这些链接类型以及`RevisionURLTemplate`和的更多信息`ThirdPartyURL`，请参阅 CodePipeline API 参考CreateCustomActionType中的ActionTypeSettings和。有关操作结构要求以及如何创建操作的更多信息，请参阅 CodePipeline 管道结构参考。

2. 保存 JSON 文件并为其指定一个易于记住的名称（例如，*MyCustomAction*.json）。
3. 在您安装了 AWS CLI的计算机上打开终端会话 (Linux、OS X、Unix) 或命令提示符 (Windows)。
4. 使用运行aws codepipeline create-custom-action-type命令，指定您刚刚创建的 JSON 文件的名称。 AWS CLI

   例如，要创建一个构建自定义操作：

   > **Important**
   >
   > 务必在文件名前包含 `file://`。此命令中需要该项。

   ```
   aws codepipeline create-custom-action-type --cli-input-json
    file://MyCustomAction.json
   ```

5. 该命令将返回您创建的自定义操作的整个结构以及为您添加的 `JobList` 操作配置属性。在管道中添加自定义操作时，您可以使用 `JobList` 来指定您可以在提供程序的哪些项目中轮询作业。如果您不配置此操作，则您的自定义作业辅助角色轮询作业时将会返回所有可用作业。

   例如，上述命令可能会返回类似以下的结构：

   ```
   {
       "actionType": {
           "inputArtifactDetails": {
               "maximumCount": 1,
               "minimumCount": 1
          },
          "actionConfigurationProperties": [
               {
                   "secret": false,
   ```

```
                "required": true,
                "name": "ProjectName",
                "key": true,
                "description": "The name of the build project must be provided when
 this action is added to the pipeline."
            }
        ],
        "outputArtifactDetails": {
            "maximumCount": 0,
            "minimumCount": 0
        },
        "id": {
            "category": "Build",
            "owner": "Custom",
            "version": "1",
            "provider": "My-Build-Provider-Name"
        },
        "settings": {
            "entityUrlTemplate": "https://my-build-instance/job/
{Config:ProjectName}/",
            "executionUrlTemplate": "https://my-build-instance/job/mybuildjob/
lastSuccessfulBuild/{ExternalExecutionId}/"
        }
    }
}
```

> **Note**
>
> 作为create-custom-action-type命令输出的一部分，该id部分包括"owner": "Custom"。 CodePipeline 自动分配Custom为自定义操作类型的所有者。使用 create-custom-action-type 命令或 update-pipeline 命令时，无法分配或更改此值。

为自定义操作创建作业辅助角色

自定义操作需要作业工作人员轮 CodePipeline 询自定义操作的作业请求，执行任务并将状态结果返回到 CodePipeline。只要可以访问的公共终端节点，作业工作人员就可以位于任何计算机或资源上 CodePipeline。

有许多方法可以设计您的作业辅助角色。以下各节为开发定制作业工作人员提供了一些实用指导 CodePipeline。

主题

为作业辅助角色选择和配置权限管理策略

要在中为你的自定义操作开发自定义工作人员 CodePipeline，你需要一个集成用户和权限管理的策略。

最简单的策略是通过创建具有 IAM 实例角色的 Amazon EC2 实例，为您的自定义作业辅助角色添加所需的基础设施，从而允许您轻松纵向扩展集成所需的资源。您可以使用内置的集成 AWS 来简化自定义作业工作人员和之间的交互 CodePipeline。

设置 Amazon EC2 实例

1. 了解更多有关 Amazon EC2 的信息，并确定其是否为适合您的集成的正确选择。有关信息，请参阅 Amazon EC2 – 虚拟服务器托管。
2. 开始创建您的 Amazon EC2 实例。有关信息，请参阅开始使用 Amazon EC2 Linux 实例。

另一个需要考虑的策略是使用 IAM 的身份联合验证来集成您的现有身份提供程序系统和资源。如果您已经拥有企业身份提供程序，或已经配置为支持使用网络身份提供程序的用户，则此策略特别有用。联合身份允许您授予对 AWS 资源（包括）的安全访问权限 CodePipeline，而无需创建或管理 IAM 用户。您可以利用针对密码安全性要求和凭证轮换的功能和策略。您可以使用示例应用程序作为自己设计的模板。

若要设置身份联合验证

1. 了解有关 IAM 身份联合验证的更多信息。有关信息，请参阅管理联合身份验证。
2. 查看授予临时访问权限的情形中的示例，以确定最适合您的自定义操作需求的临时访问情形。
3. 查看与您的基础设施相关的联合身份验证代码示例，例如：
   - Active Directory 的联合身份验证示例应用程序使用案例
4. 开始配置联合身份验证。有关信息，请参阅 IAM 用户指南 中的身份提供程序和联合身份验证。

创建以下内容之一，以便在运行自定义操作和作业辅助工具时在您的 AWS 账户 下使用。

如果用户想在 AWS 外部进行交互，则需要编程访问权限 AWS Management Console。授予编程访问权限的方式取决于正在访问的用户类型 AWS。

要向用户授予编程式访问权限，请选择以下选项之一。

哪个用户需要编程式访问权限？	目的	方式
人力身份 （在 IAM Identity Center 中管理的用户）	使用临时证书签署向 AWS CLI、 AWS 软件开发工具包或 AWS API 发出的编程请求。	按照您希望使用的界面的说明进行操作。 • 有关的 AWS CLI，请参阅《AWS Command Line Interface 用户指南》 AWS IAM Identity Center中的 “配置 AWS CLI 要使用”。 • 有关 AWS 软件开发工具包、工具和 AWS API，请参阅《软件开发工具包和AWS 工具参考指南》中的 IAM 身份中心身份验证。
IAM	使用临时证书签署向 AWS CLI、 AWS 软件开发工具包或 AWS API 发出的编程请求。	按照 IAM 用户指南中的将临时证书与 AWS 资源配合使用中的说明进行操作。
IAM	（不推荐使用） 使用长期凭证签署向 AWS CLI、 AWS 软件开发工具包或 AWS API 发出的编程请求。	按照您希望使用的界面的说明进行操作。 • 有关信息 AWS CLI，请参阅用户指南中的使用 IAM 用户证书进行身份验证。AWS Command Line Interface • 有关 AWS SDK 和工具，请参阅 S AWS DK 和工具参考指南中的使用长期凭证进行身份验证。

哪个用户需要编程式访问权限？	目的	方式
		• 有关 AWS API，请参阅 IAM 用户指南中的管理 IAM 用户的访问密钥。

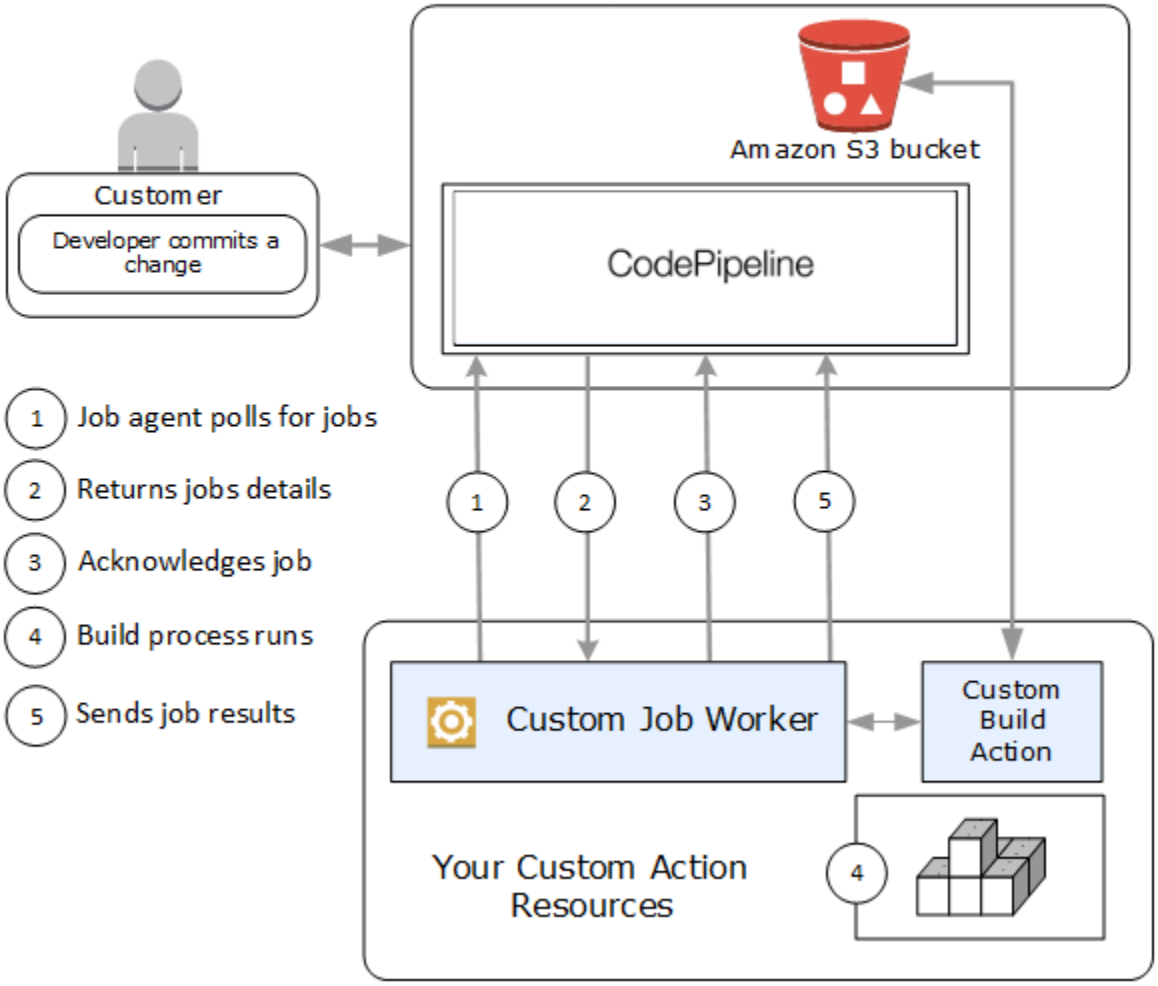
以下是一个您为了与自定义作业辅助角色一起使用而可能会创建的示例策略。此策略仅作为示例，按原样提供。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codepipeline:PollForJobs",
        "codepipeline:AcknowledgeJob",
        "codepipeline:GetJobDetails",
        "codepipeline:PutJobSuccessResult",
        "codepipeline:PutJobFailureResult"
      ],
      "Resource": [
        "arn:aws:codepipeline:us-east-2::actionType:custom/Build/MyBuildProject/1/"
      ]
    }
  ]
}
```

> **Note**
>
> 考虑使用 AWSCodePipelineCustomActionAccess 托管式策略。

为自定义操作开发作业辅助角色

在你选择了权限管理策略后，你应该考虑你的求职者将如何与之互动 CodePipeline。以下概要图表显示了构建过程的自定义操作和作业辅助角色的工作流。



1. 您的求职者使用以下方式 CodePipeline 对工作进行民意调查`PollForJobs`。
2. 当源阶段中的变更（例如，开发者提交更改）触发管道时，自动发布过程就会开始。该过程将持续执行，一直到配置自定义操作的阶段为止。当它到达你在此阶段的操作时，会对任务进行 CodePipeline 排队。如果您的作业辅助角色再次调用 `PollForJobs` 以获取状态，则会显示此作业。从 `PollForJobs` 中获取作业详细信息并将其传回给您的作业辅助角色。
3. 求职者打电话`AcknowledgeJob`发送 CodePipeline 工作确认。 CodePipeline 返回一条确认信息，表示作业工作人员应继续工作 (`InProgress`)，或者，如果您有多个求职者在轮询作业，而另一名求职者已经申请了该作业，则将返回`InvalidNonceException`错误响应。`InProgress`确认后，CodePipeline 等待结果返回。

4. 作业辅助角色对修订启动您的自定义操作，然后您的操作将开始执行。与任何其他操作一样，您的自定义操作也会将结果返回给作业辅助角色。在构建自定义操作的示例中，操作从 Amazon S3 桶中提取项目，对其进行构建，然后将成功构建的构件推送回 Amazon S3 桶。
5. 当操作正在运行时，作业辅助角色可以使用延续令牌（由作业辅助角色生成的作业状态的序列化，例如 JSON 格式的构建标识符或一个 Amazon S3 对象键），以及将用于填充 `executionUrlTemplate` 中的链接的 `ExternalExecutionId` 信息来调用 `PutJobSuccessResult`。这将使用一个有效链接来更新管道的控制台视图，该链接提供正在进行中的操作的具体细节。虽然不是必需的，但它是最佳实践，因为它使用户能够在自定义操作运行时查看其状态。

   调用 `PutJobSuccessResult` 后，作业即视为完成。在中创建了一个 CodePipeline 包含延续令牌的新作业。如果您的作业辅助角色再次调用 `PollForJobs`，则会显示此作业。此新任务可用于检查操作的状态，然后返回延续令牌，或者在操作完成后不返回延续令牌。

   > **Note**
   >
   > 如果您的作业辅助角色执行自定义操作的所有工作，则应考虑将作业辅助角色处理过程分为至少两个步骤。第一步是为您的操作建立详细信息页面。创建详细信息页面后，您可以将作业辅助角色的状态序列化，并将其作为延续令牌返回，同时遵守大小限制（请参阅 中的配额 AWS CodePipeline）。例如，您可以将操作的状态写入用作延续令牌的字符串中。作业辅助角色处理过程的第二步（以及后续步骤）将执行操作的实际工作。最后一步将成功或失败返回到 CodePipeline，最后一步没有延续标记。

   有关使用延续令牌的更多信息，请参阅 CodePipeline API 参考`PutJobSuccessResult`中的规范。
6. 自定义操作完成后，作业工作人员 CodePipeline 通过调用两个 API 中的一个，将自定义操作的结果返回到：
   - `PutJobSuccessResult`，不带延续令牌，表示自定义操作已成功运行
   - `PutJobFailureResult`，表示自定义操作未成功运行

   根据结果，管道将继续执行下一个操作（成功），或者停止（失败）。

自定义作业辅助角色架构和示例

在制定概要工作流后，您可以创建作业辅助角色。尽管自定义操作的细节最终将决定作业辅助角色需要什么，但自定义操作的大多数作业辅助角色都包括以下功能：

- CodePipeline 正在使用轮询作业`PollForJobs`。
- 确认任务并将结果返回到 CodePipeline 使用`AcknowledgeJob`、`PutJobSuccessResult`和`PutJobFailureResult`。
- 从管道的 Amazon S3 桶中检索构件和/或将构件放入桶。要从 Amazon S3 桶下载构件，您必须创建一个使用签名版本 4 (Sig V4) 签名的 Amazon S3 客户端。需要 Sig V4 才能使用。 AWS KMS

  要将构件上传到 Amazon S3 桶，您还必须另外配置 Amazon S3 `PutObject` 请求以使用加密。目前仅支持 AWS 密钥管理服务 (AWS KMS) 进行加密。 AWS KMS 使用 AWS KMS keys。为了知道是使用客户管理的 AWS 托管式密钥 密钥还是客户管理的密钥来上传工件，您的自定义作业工作人员必须查看作业数据并检查加密密钥属性。如果设置了该属性，则在配置时应使用该客户托管密钥 ID AWS KMS。如果键属性为空，则使用 AWS 托管式密钥。 CodePipeline AWS 托管式密钥 除非另行配置，否则使用。

  有关演示如何使用 Java 或.NET 创建 AWS KMS 参数的示例，请参阅使用 AWS 软件开发工具包 AWS Key Management Service 在 Amazon S3 中指定。有关的 Amazon S3 存储桶的更多信息 CodePipeline，请参阅CodePipeline 概念 。

有关自定义作业工作人员的更复杂示例，请参见 GitHub。此示例是一个开源示例，按原样提供。

- Job Worker 示例 CodePipeline：从 GitHub 存储库下载示例。

向管道添加自定义操作

有了作业辅助工具后，您可以通过以下方式将您的自定义操作添加到管道中：创建一个新的管道并在使用 “创建管道” 向导时选择它，编辑现有管道并添加自定义操作，或者使用 AWS CLI、软件开发工具包或 API。

> **Note**
>
> 您可以在“Create Pipeline (创建管道)”向导中创建一个包含自定义操作（如果自定义操作是生成或部署操作）的管道。如果您的自定义操作位于测试类别中，则您必须通过编辑现有管道来添加它。

主题

- 向现有管道添加自定义操作 (CLI)

向现有管道添加自定义操作 (CLI)

您可以使用 AWS CLI 向现有管道添加自定义操作。

1. 打开终端会话（Linux、macOS 或 Unix）或命令提示符 (Windows)，然后运行 get-pipeline 命令，将要编辑的管道结构复制到 JSON 文件中。例如，对于名为 **MyFirstPipeline** 的管道，可以键入以下命令：

   ```
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```

   该命令不会返回任何结果，但您创建的文件将出现在您运行命令所在的目录中。

2. 在任何文本编辑器中打开 JSON 文件，修改文件结构，以便将您的自定义操作添加到一个现有阶段中。

   > **Note**
   >
   > 如果您希望您的操作与此阶段中的另一操作并行运行，请确保为其分配与另一操作相同的 `runOrder` 值。

   例如，要修改管道结构，以添加一个名为 Build 的阶段并将一个生成自定义操作添加到该阶段中，您可以将 JSON 修改为在部署阶段之前添加 Build 阶段，如下所示：

   ```
   ,
       {
         "name": "MyBuildStage",
         "actions":  [
                 {
                   "inputArtifacts": [
                   {
                      "name": "MyApp"
                    }
                      ],
                       "name": "MyBuildCustomAction",
                       "actionTypeId": {
                           "category": "Build",
                           "owner": "Custom",
                           "version": "1",
                           "provider": "My-Build-Provider-Name"
                       },
   ```

```
                    "outputArtifacts": [
                        {
                          "name": "MyBuiltApp"
                        }
                    ],
                    "configuration": {
                        "ProjectName": "MyBuildProject"
                    },
                    "runOrder": 1
                }
            ]
        },
        {
            "name": "Staging",
            "actions": [
                    {
                        "inputArtifacts": [
                            {
                                "name": "MyBuiltApp"
                            }
                        ],
                        "name": "Deploy-CodeDeploy-Application",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeDeploy"
                        },
                        "outputArtifacts": [],
                        "configuration": {
                            "ApplicationName": "CodePipelineDemoApplication",
                            "DeploymentGroupName": "CodePipelineDemoFleet"
                        },
                        "runOrder": 1
                    }
                ]
             }
    ]
}
```

3. 要应用更改，请运行 update-pipeline 命令并指定一个管道 JSON 文件，类似于以下内容：

> **Important**
>
> 务必在文件名前包含 `file://`。此命令中需要该项。

```
aws codepipeline update-pipeline --cli-input-json file://pipeline.json
```

该命令会返回编辑后的管道的整个结构。

4. 打开 CodePipeline 控制台，然后选择刚才编辑的管道的名称。

管道将显示您的更改。当您下一次更改源位置时，管道会通过管道修订后的结构运行该修订。

在中标记自定义操作 CodePipeline

标签是与资源关联的键值对。 AWS 您可以使用控制台或 CLI 将标签应用于中的自定义操作 CodePipeline。有关 CodePipeline 资源标记、用例、标签键和值限制以及支持的资源类型的信息，请参阅标记资源。

您可以添加、删除和更新自定义操作中的标签值。您可以为每个自定义操作添加最多 50 个标签。

主题

- 为自定义操作添加标签
- 查看自定义操作的标签
- 编辑自定义操作的标签
- 删除自定义操作的标签

为自定义操作添加标签

按照以下步骤使用 AWS CLI 向自定义操作添加标签。要在创建自定义操作的同时为其添加标签，请参阅在中创建和添加自定义操作 CodePipeline。

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。

在终端或命令行运行 tag-resource 命令，指定要为其添加标签的自定义操作的 Amazon 资源名称 (ARN)，以及要添加的标签的键/值。您可以为自定义操作添加多个标签。例如，要使用两个标

签为自定义操作添加标签，一个标签键*TestActionType*以标签值命名 *UnitTest*，一个标签键*ApplicationName*的标签值为 *MyApplication*：

```
aws codepipeline tag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:actiontype:Owner/Category/Provider/Version --tags key=TestActionType,value=UnitTest
 key=ApplicationName,value=MyApplication
```

如果成功，该命令不返回任何内容。

查看自定义操作的标签

按照以下步骤使用 AWS CLI 来查看自定义操作的 AWS 标签。如果尚未添加标签，则返回的列表为空。

在终端或命令行中，运行 list-tags-for-resource 命令。例如，要查看 ARN 为 arn:aws:codepipeline:*us-west-2*:*account-id*:actiontype:*Owner*/*Category*/*Provider*/*Version* 的自定义操作的标签键和标签值列表：

```
aws codepipeline list-tags-for-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:actiontype:Owner/Category/Provider/Version
```

如果成功，该命令返回类似以下内容的信息：

```
{
    "tags": {
        "TestActionType": "UnitTest",
        "ApplicationName": "MyApplication"
    }
}
```

编辑自定义操作的标签

按照以下步骤 AWS CLI 使用编辑自定义操作的标签。您可以更改现有键的值或添加另一个键。您还可以删除自定义操作的标签，如下一节所示。

在终端或命令行运行 tag-resource 命令，指定要为其更新标签的自定义操作的 Amazon 资源名称 (ARN) 并指定标签键和标签值：

```
aws codepipeline tag-resource --resource-arn arn:aws:codepipeline:us-
west-2:account-id:actiontype:Owner/Category/Provider/Version --tags
 key=TestActionType,value=IntegrationTest
```

删除自定义操作的标签

按照以下步骤使用 AWS CLI 从自定义操作中移除标签。当您移除关联资源的标签时，对应标签会被删除。

> **Note**
>
> 如果您删除自定义操作，会从已删除的自定义操作中移除关联的所有标签。您无需在删除该自定义操作之前移除标签。

在终端或命令行运行 untag-resource 命令，指定要从中删除标签的自定义操作的 ARN 以及要删除的标签的标签键。例如，要使用标签键删除自定义操作上的标签，请执行以下操作 *TestActionType*：

```
aws codepipeline untag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-
id:actiontype:Owner/Category/Provider/Version --tag-keys TestActionType
```

如果成功，该命令不返回任何内容。要验证与自定义操作关联的标签，运行 list-tags-for-resource 命令。

在中调用管道中的 AWS Lambda 函数 CodePipeline

AWS Lambda 是一项计算服务，可使您无需调配或管理服务器即可运行代码。您可以创建 Lambda 函数并将其作为操作添加到管道中。由于 Lambda 允许您编写函数来执行几乎任何任务，因此您可以自定义管道的工作方式。

> **Important**
>
> 不要记录 CodePipeline 发送到 Lambda 的 JSON 事件，因为这可能会导致用户凭证记录到日志中 CloudWatch 。该 CodePipeline 角色使用 JSON 事件将临时证书传递给该`artifactCredentials`领域的 Lambda。有关示例事件，请参阅JSON 事件示例。

以下是可在管道中使用 Lambda 函数的一些方法：

- 在管道的一个阶段使用 AWS CloudFormation 按需创建资源，然后在另一个阶段将其删除。
- 使用交换别名记录值的 Lambda 函数部署停机时间为零的应用程序版本。 AWS Elastic Beanstalk
- 部署到 Amazon ECS Docker 实例。
- 通过创建 AMI 快照在生成或部署之前备份资源。
- 将与第三方产品的集成添加到您的管道中，例如将消息发布到 IRC 客户端。

> **Note**
>
> 创建和运行 Lambda 函数可能会导致您的 AWS 账户产生费用。有关更多信息，请参阅定价。

本主题假设您熟悉 AWS CodePipeline AWS Lambda 并知道如何创建管道、函数以及它们所依赖的 IAM 策略和角色。此主题向您演示如何：

- 创建一个 Lambda 函数以测试网页是否已成功部署。
- 配置 CodePipeline 和 Lambda 执行角色以及在管道中运行函数所需的权限。
- 编辑管道，将 Lambda 函数作为一个操作添加到其中。
- 通过手动发布更改来测试操作。

> **Note**
>
> CodePipeline在中使用跨区域 Lambda 调用操作时，使用PutJobSuccessResultPutJobFailureResult和执行 lambda 的状态应发送到 AWS 存在 Lambda 函数的区域，而不是发送到存在的区域。 CodePipeline

本主题包含示例函数，用于演示如何灵活地使用 Lambda 函数： CodePipeline

- Basic Lambda function
  - 创建要与一起使用的基本 Lambda 函数。 CodePipeline
  - 将成功或失败 CodePipeline 的结果返回到操作的详细信息链接。
- 使用 AWS CloudFormation 模板的 Python 函数示例
  - 使用 JSON 编码的用户参数向函数 (`get_user_params`) 传递多个配置值。
  - 与项目存储桶 (`get_template`) 中的 .zip 项目交互。

- 使用延续令牌监控长期异步过程 (`continue_job_later`)。这将允许操作继续进行，使函数取得成功，即使已超过十五分钟的运行时（Lambda 中的限制）也不例外。

每个示例函数都包含有关您必须添加到角色的权限的信息。有关中限制的信息 AWS Lambda，请参阅《AWS Lambda 开发者指南》中的限制。

> **Important**
>
> 本主题中包含的示例代码、角色和策略仅作为示例，并按原样提供。

主题

步骤 1：创建管道

在此步骤中，您将创建一个管道，稍后您将向其中添加 Lambda 函数。这与您在CodePipeline 教程中创建的管道相同。如果该管道仍然配置用于您的账户，并且属于您计划创建 Lambda 函数的相同区域，则可以跳过此步骤。

创建管道

1. 按照中的教程：创建一个简单的管道（S3 存储桶）前三个步骤创建 Amazon S3 存储桶、CodeDeploy资源和两阶段管道。为您的实例类型选择 Amazon Linux 选项。你可以为管道使用任何你想要的名称，但本主题中的步骤使用 MyLambdaTestPipeline。
2. 在管道的状态页面的 CodeDeploy 操作中，选择详细信息。在部署组的部署详细信息页中，从列表中选择一个实例 ID。
3. 在 Amazon EC2 控制台上，在实例的详细信息选项卡中，复制公有 IPv4 地址中的 IP 地址（例如，**`192.0.2.4`**）。您应将此地址用作 AWS Lambda中的函数的目标。

> **Note**
>
> 的默认服务角色策略 CodePipeline 包括调用该函数所需的 Lambda 权限。但是，如果您修改了该默认服务角色或选择了另一个角色，请确保该角色的策略允许 `lambda:InvokeFunction` 和 `lambda:ListFunctions` 权限。否则，包含 Lambda 操作的管道将会失败。

第 2 步：创建 Lambda 函数

在此步骤中，您将创建一个 Lambda 函数，该函数会发出 HTTP 请求，并检查网页上的一行文本。作为此步骤的一部分，您还必须创建 IAM 策略和 IAM 执行角色。有关更多信息，请参阅 AWS Lambda 开发者指南 中的权限模型。

创建执行角色

1. 登录 AWS Management Console 并打开 IAM 控制台，网址为 https://console.aws.amazon.com/iam/。
2. 选择 Policies，然后选择 Create Policy。选择 JSON 选项卡，然后将以下策略粘贴到字段中。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:*"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Action": [
        "codepipeline:PutJobSuccessResult",
        "codepipeline:PutJobFailureResult"
        ],
        "Effect": "Allow",
        "Resource": "*"
     }
  ]
}
```

3. 选择查看策略。
4. 在 Review policy (查看策略) 页面上，在 Name (名称) 中，键入策略的名称（例如，**`CodePipelineLambdaExecPolicy`**）。在 Description (描述) 中，输入 **`Enables Lambda to execute code`**。

   选择创建策略。

   > **Note**
   >
   > 这些是 Lambda 函数与 Amazon 互操作所需的最低权限。 CodePipeline CloudWatch如果您想扩展此策略以允许函数与其他 AWS 资源交互，则应修改此策略以允许这些 Lambda 函数所需的操作。

5. 在策略控制面板页上，选择 Roles (角色)，然后选择 Create role (创建角色)。
6. 在创建角色页面上，选择 AWS 服务。选择 Lambda，然后选择 Next: Permissions (下一步: 权限)。
7. 在附加权限策略页面上，选中旁边的复选框 CodePipelineLambdaExecPolicy，然后选择下一步：标签。选择 下一步: 审核。
8. 在 Review (审核) 页面上的 Role name (角色名称) 中，输入名称，然后选择 Create role (创建角色)。

创建要与一起使用的示例 Lambda 函数 CodePipeline

1. 登录 AWS Management Console 并打开 AWS Lambda 控制台，网址为 https://console.aws.amazon.com/lambda/。
2. 在 Functions (LAM 函数) 页面上，选择 Create function (创建函数)。

   > **Note**
   >
   > 如果您看到欢迎页面而不是 Lambda 页面，请选择立即开始使用。

3. 在创建函数页面上，选择从头开始创作。在函数名称中，输入 Lambda 函数的名称（例如 **`MyLambdaFunctionForAWSCodePipeline`**）。在 Runtime 中，选择 Node.js 20.x。
4. 在 Role (角色) 下，选择 Choose an existing role (选择现有角色)。在 Existing role (现有角色) 中，选择角色，然后选择 Create function (创建函数)。

此时将打开供您创建函数的详细信息页面。

5. 在 Function code (函数代码) 框中，复制贴以下代码：

> **Note**
>
> CodePipeline.job 键下的事件对象包含任务详细信息。有关 CodePipeline 返回到 Lambda 的 JSON 事件的完整示例，请参阅。JSON 事件示例

```
import { CodePipelineClient, PutJobSuccessResultCommand,
 PutJobFailureResultCommand } from "@aws-sdk/client-codepipeline";
import http from 'http';
import assert from 'assert';

export const handler = (event, context) => {

    const codepipeline = new CodePipelineClient();

    // Retrieve the Job ID from the Lambda action
    const jobId = event["CodePipeline.job"].id;

    // Retrieve the value of UserParameters from the Lambda action configuration in
 CodePipeline, in this case a URL which will be
    // health checked by this function.
    const url =
 event["CodePipeline.job"].data.actionConfiguration.configuration.UserParameters;

    // Notify CodePipeline of a successful job
    const putJobSuccess = async function(message) {
        const command = new PutJobSuccessResultCommand({
            jobId: jobId
        });
         try {
             await codepipeline.send(command);
             context.succeed(message);
         } catch (err) {
             context.fail(err);
         }
    };

    // Notify CodePipeline of a failed job
```

```
    const putJobFailure = async function(message) {
        const command = new PutJobFailureResultCommand({
            jobId: jobId,
            failureDetails: {
                message: JSON.stringify(message),
                type: 'JobFailed',
                externalExecutionId: context.awsRequestId
            }
        });
        await codepipeline.send(command);
        context.fail(message);
    };

    // Validate the URL passed in UserParameters
    if(!url || url.indexOf('http://') === -1) {
        putJobFailure('The UserParameters field must contain a valid URL address to
test, including http:// or https://');
        return;
    }

    // Helper function to make a HTTP GET request to the page.
    // The helper will test the response and succeed or fail the job accordingly
    const getPage = function(url, callback) {
        var pageObject = {
            body: '',
            statusCode: 0,
            contains: function(search) {
                return this.body.indexOf(search) > -1;
            }
        };
        http.get(url, function(response) {
            pageObject.body = '';
            pageObject.statusCode = response.statusCode;

            response.on('data', function (chunk) {
                pageObject.body += chunk;
            });

            response.on('end', function () {
                callback(pageObject);
            });

            response.resume();
        }).on('error', function(error) {
```

```
            // Fail the job if our request failed
            putJobFailure(error);
        });
    };

    getPage(url, function(returnedPage) {
        try {
            // Check if the HTTP response has a 200 status
            assert(returnedPage.statusCode === 200);
            // Check if the page contains the text "Congratulations"
            // You can change this to check for different text, or add other tests
 as required
            assert(returnedPage.contains('Congratulations'));

            // Succeed the job
            putJobSuccess("Tests passed.");
        } catch (ex) {
            // If any of the assertions failed then fail the job
            putJobFailure(ex);
        }
    });
};
```

6. 保留 Handler (处理程序) 和 Role (角色) 的默认值 **CodePipelineLambdaExecRole**。
7. 在 Basic settings (基本设置) 中，对于 Timeout (超时)，输入 **20** 秒。
8. 选择保存。

步骤 3：在控制台中将 Lambda 函数添加到管道中 CodePipeline

在此步骤中，您将在管道中添加一个新阶段，然后向该阶段添加调用您的函数的 Lambda 操作。

添加阶段

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 在 Welcome (欢迎使用) 页面上，选择您创建的管道。
3. 在管道视图页中，选择 Edit。
4. 在 “编辑” 页面上，选择 + 添加阶段，在部署阶段之后添加一个带有 CodeDeploy 操作的阶段。输入阶段的名称（例如，**LambdaStage**），然后选择 Add stage (添加阶段)。

> **Note**
>
> 您还可以选择将您的 Lambda 操作添加到现有阶段。出于演示目的，我们将 Lambda 函数作为阶段中的唯一操作进行添加，以便您可以轻松地在构件通过管道时查看该操作的进度。

5. 选择 + 添加操作组。在编辑操作的操作名称中，键入 Lambda 操作的名称（例如，**`MyLambdaAction`**）。在 Provider (提供方) 中，选择 AWS Lambda。在函数名称中，选择或输入 Lambda 函数的名称（例如，**`MyLambdaFunctionForAWSCodePipeline`**）。在用户参数中，指定前面复制的 Amazon EC2 实例的 IP 地址（例如，**`http://192.0.2.4`**），然后选择完成。

> **Note**
>
> 本主题使用 IP 地址，但在真实场景中，您可以提供您注册的网站名称（例如，**`http://www.example.com`**）。有关事件数据和处理程序的更多信息 AWS Lambda，请参阅《AWS Lambda 开发人员指南》中的编程模型。

6. 在 Edit action (编辑操作) 页面上，选择 Save (保存)。

步骤 4：使用 Lambda 函数测试管道

要测试函数，请通过管道发布最近的更改。

使用控制台，通过管道运行最新版本的项目

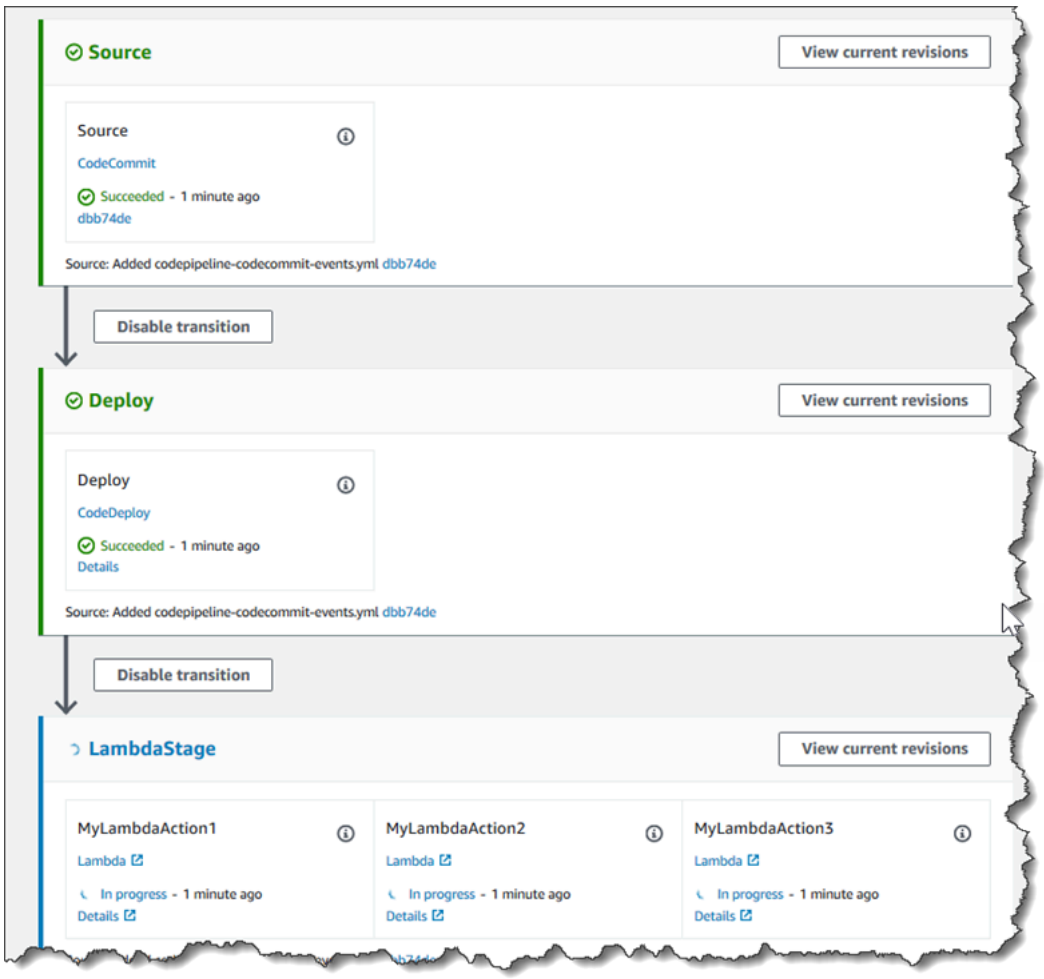
1. 在管道详细信息页中，选择发布更改。这会通过管道运行在源操作中指定的每个源位置中提供的最新修订。
2. Lambda 操作完成后，选择详细信息链接以在 Amazon 中查看该函数的日志流 CloudWatch，包括事件的计费时长。如果函数失败，则 CloudWatch 日志会提供有关原因的信息。

步骤 5：后续步骤

既然已成功创建 Lambda 函数并将其作为操作添加到管道中，接下来您便可以尝试以下操作：

- 向阶段中添加更多 Lambda 操作以检查其他网页。
- 修改 Lambda 函数以检查其他文本字符串。

- 探究 Lambda 函数并创建您自己的 Lambda 函数，然后将其添加到管道中。



在您完成 Lambda 函数的实验后，可以考虑将其从您的管道中删除、从 IAM 中删除并 AWS Lambda从 IAM 中删除该角色，以避免可能的费用。有关更多信息，请参阅在中编辑管道 CodePipeline、删除中的管道 CodePipeline和删除角色或实例配置文件。

JSON 事件示例

以下示例显示了发送给 Lambda 的 JSON 事件示例。 CodePipeline此事件的结构类似于对 `GetJobDetails API` 的响应，但没有 `actionTypeId` 和 `pipelineContext` 数据类型。JSON 事件和对 `GetJobDetails` API 的响应中都包含两个操作配置详细信息 `FunctionName` 和 `UserParameters`。`######` 中的值是示例或解释，而不是实际值。

```
{
    "CodePipeline.job": {
        "id": "11111111-abcd-1111-abcd-111111abcdef",
```

```
        "accountId": "111111111111",
        "data": {
            "actionConfiguration": {
                "configuration": {
                    "FunctionName": "MyLambdaFunctionForAWSCodePipeline",
                    "UserParameters": "some-input-such-as-a-URL"
                }
            },
            "inputArtifacts": [
                {
                    "location": {
                        "s3Location": {
                            "bucketName": "the name of the bucket configured as the
pipeline artifact store in Amazon S3, for example codepipeline-us-east-2-1234567890",
                            "objectKey": "the name of the application, for example
CodePipelineDemoApplication.zip"
                        },
                        "type": "S3"
                    },
                    "revision": null,
                    "name": "ArtifactName"
                }
            ],
            "outputArtifacts": [],
            "artifactCredentials": {
                "secretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
                "sessionToken": "MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
3rrszlaEXAMPLE=",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE"
            },
            "continuationToken": "A continuation token if continuing job",
```

```
            "encryptionKey": {
              "id": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
              "type": "KMS"
            }
        }
    }
}
```

其他示例函数

以下 Lambda 函数示例，演示了可用于中的管道的其他功能。 CodePipeline要使用这些函数，您可能必须修改 Lambda 执行角色的策略，如每个示例的介绍中所述。

主题

- 使用 AWS CloudFormation 模板的 Python 函数示例

使用 AWS CloudFormation 模板的 Python 函数示例

以下示例显示了一个基于提供的 AWS CloudFormation 模板创建或更新堆栈的函数。此模板将创建一个 Amazon S3 桶。它只用于演示用途，以将成本降至最低。理想情况下，您应该在向存储桶上传任何内容之前删除堆栈。如果您将文件上传到存储桶，那么在删除堆栈时将无法删除存储桶。您必须手动删除存储桶中的所有内容，然后才能删除存储桶本身。

该 Python 示例假设您有一个使用 Amazon S3 桶作为源操作的管道，或者您有权访问可与管道一起使用的版本控制型 Amazon S3 桶。您可以创建 AWS CloudFormation 模板，对其进行压缩，然后将其作为.zip 文件上传到该存储桶。然后，您必须向管道中添加一个源操作，以从存储桶中检索此 .zip 文件。

> **Note**
>
> 当 Amazon S3 是您的管道的源提供程序时，您可以将一个或多个源文件压缩到单个 .zip 文件中，然后将 .zip 文件上传到源桶。您也可以上传单个解压缩的文件；但是，需要 .zip 文件的下游操作将失败。

此示例演示：

- 使用 JSON 编码的用户参数向函数 (`get_user_params`) 传递多个配置值。
- 与项目存储桶 (`get_template`) 中的 .zip 项目交互。

- 使用延续令牌监控长期异步过程 (`continue_job_later`)。这将允许操作继续进行，使函数取得成功，即使已超过十五分钟的运行时（Lambda 中的限制）也不例外。

要使用此示例 Lambda 函数，Lambda 执行角色的策略必须具有 A AWS CloudFormation mazon S3 和 CodePipeline（如本示例策略所示）中的`Allow`权限：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "logs:*"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:logs:*:*:*"
        },
        {
            "Action": [
                "codepipeline:PutJobSuccessResult",
                "codepipeline:PutJobFailureResult"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "cloudformation:DescribeStacks",
                "cloudformation:CreateStack",
                "cloudformation:UpdateStack"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "s3:*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

要创建 AWS CloudFormation 模板，请打开任何纯文本编辑器并复制并粘贴以下代码：

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",
  "Description" : "CloudFormation template which creates an S3 bucket",
  "Resources" : {
    "MySampleBucket" : {
      "Type" : "AWS::S3::Bucket",
      "Properties" : {
      }
    }
  },
  "Outputs" : {
    "BucketName" : {
      "Value" : { "Ref" : "MySampleBucket" },
      "Description" : "The name of the S3 bucket"
    }
  }
}
```

使用名称 **template.json** 将此内容保存为名为 **template-package** 目录中的一个 JSON 文件。创建此目录和文件的压缩 (.zip) 文件并命名为 **template-package.zip**，然后将压缩文件上传到受版本控制的 Amazon S3 桶。如果您已经为管道配置了存储桶，则可以使用该存储桶。接下来，编辑您的管道以添加检索 .zip 文件的源操作。命名此操作的输出*MyTemplate*。有关更多信息，请参阅 在中编辑管道 CodePipeline。

> **Note**
>
> 示例 Lambda 函数期望使用这些文件名和压缩结构。但是，您可以用自己的 AWS CloudFormation 模板代替此示例。如果您使用自己的模板，请务必修改 Lambda 执行角色的策略，以允许模板所需的任何其他功能。 AWS CloudFormation

添加以下代码以作为 Lambda 中的函数

1. 打开 Lambda 控制台，选择创建函数。
2. 在创建函数页面上，选择从头开始创作。在函数名称中，为您的 Lambda 函数输入一个名称。
3. 在 Runtime (运行时) 中，选择 Python 2.7。
4. 在选择或创建执行角色下，选择使用现有角色。在 Existing role (现有角色) 中，选择角色，然后选择 Create function (创建函数)。

此时将打开供您创建函数的详细信息页面。

5. 在 Function code (函数代码) 框中，复制贴以下代码：

```
from __future__ import print_function
from boto3.session import Session

import json
import urllib
import boto3
import zipfile
import tempfile
import botocore
import traceback

print('Loading function')

cf = boto3.client('cloudformation')
code_pipeline = boto3.client('codepipeline')

def find_artifact(artifacts, name):
    """Finds the artifact 'name' among the 'artifacts'

    Args:
        artifacts: The list of artifacts available to the function
        name: The artifact we wish to use
    Returns:
        The artifact dictionary found
    Raises:
        Exception: If no matching artifact is found

    """
    for artifact in artifacts:
        if artifact['name'] == name:
            return artifact

    raise Exception('Input artifact named "{0}" not found in event'.format(name))

def get_template(s3, artifact, file_in_zip):
    """Gets the template artifact

    Downloads the artifact from the S3 artifact store to a temporary file
    then extracts the zip and returns the file containing the CloudFormation
```

```
    template.

    Args:
        artifact: The artifact to download
        file_in_zip: The path to the file within the zip containing the template

    Returns:
        The CloudFormation template as a string

    Raises:
        Exception: Any exception thrown while downloading the artifact or unzipping
 it

    """
    tmp_file = tempfile.NamedTemporaryFile()
    bucket = artifact['location']['s3Location']['bucketName']
    key = artifact['location']['s3Location']['objectKey']

    with tempfile.NamedTemporaryFile() as tmp_file:
        s3.download_file(bucket, key, tmp_file.name)
        with zipfile.ZipFile(tmp_file.name, 'r') as zip:
            return zip.read(file_in_zip)

def update_stack(stack, template):
    """Start a CloudFormation stack update

    Args:
        stack: The stack to update
        template: The template to apply

    Returns:
        True if an update was started, false if there were no changes
        to the template since the last update.

    Raises:
        Exception: Any exception besides "No updates are to be performed."

    """
    try:
        cf.update_stack(StackName=stack, TemplateBody=template)
        return True

    except botocore.exceptions.ClientError as e:
        if e.response['Error']['Message'] == 'No updates are to be performed.':
```

```
            return False
        else:
            raise Exception('Error updating CloudFormation stack
 "{0}"'.format(stack), e)

def stack_exists(stack):
    """Check if a stack exists or not

    Args:
        stack: The stack to check

    Returns:
        True or False depending on whether the stack exists

    Raises:
        Any exceptions raised .describe_stacks() besides that
        the stack doesn't exist.

    """
    try:
        cf.describe_stacks(StackName=stack)
        return True
    except botocore.exceptions.ClientError as e:
        if "does not exist" in e.response['Error']['Message']:
            return False
        else:
            raise e

def create_stack(stack, template):
    """Starts a new CloudFormation stack creation

    Args:
        stack: The stack to be created
        template: The template for the stack to be created with

    Throws:
        Exception: Any exception thrown by .create_stack()
    """
    cf.create_stack(StackName=stack, TemplateBody=template)

def get_stack_status(stack):
    """Get the status of an existing CloudFormation stack

    Args:
```

```
        stack: The name of the stack to check

    Returns:
        The CloudFormation status string of the stack such as CREATE_COMPLETE

    Raises:
        Exception: Any exception thrown by .describe_stacks()

    """
    stack_description = cf.describe_stacks(StackName=stack)
    return stack_description['Stacks'][0]['StackStatus']

def put_job_success(job, message):
    """Notify CodePipeline of a successful job

    Args:
        job: The CodePipeline job ID
        message: A message to be logged relating to the job status

    Raises:
        Exception: Any exception thrown by .put_job_success_result()

    """
    print('Putting job success')
    print(message)
    code_pipeline.put_job_success_result(jobId=job)

def put_job_failure(job, message):
    """Notify CodePipeline of a failed job

    Args:
        job: The CodePipeline job ID
        message: A message to be logged relating to the job status

    Raises:
        Exception: Any exception thrown by .put_job_failure_result()

    """
    print('Putting job failure')
    print(message)
    code_pipeline.put_job_failure_result(jobId=job, failureDetails={'message':
 message, 'type': 'JobFailed'})

def continue_job_later(job, message):
```

```
    """Notify CodePipeline of a continuing job

    This will cause CodePipeline to invoke the function again with the
    supplied continuation token.

    Args:
        job: The JobID
        message: A message to be logged relating to the job status
        continuation_token: The continuation token

    Raises:
        Exception: Any exception thrown by .put_job_success_result()

    """

    # Use the continuation token to keep track of any job execution state
    # This data will be available when a new job is scheduled to continue the
 current execution
    continuation_token = json.dumps({'previous_job_id': job})

    print('Putting job continuation')
    print(message)
    code_pipeline.put_job_success_result(jobId=job,
 continuationToken=continuation_token)

def start_update_or_create(job_id, stack, template):
    """Starts the stack update or create process

    If the stack exists then update, otherwise create.

    Args:
        job_id: The ID of the CodePipeline job
        stack: The stack to create or update
        template: The template to create/update the stack with

    """
    if stack_exists(stack):
        status = get_stack_status(stack)
        if status not in ['CREATE_COMPLETE', 'ROLLBACK_COMPLETE',
 'UPDATE_COMPLETE']:
            # If the CloudFormation stack is not in a state where
            # it can be updated again then fail the job right away.
            put_job_failure(job_id, 'Stack cannot be updated when status is: ' +
 status)
```

```
            return

        were_updates = update_stack(stack, template)

        if were_updates:
            # If there were updates then continue the job so it can monitor
            # the progress of the update.
            continue_job_later(job_id, 'Stack update started')

        else:
            # If there were no updates then succeed the job immediately
            put_job_success(job_id, 'There were no stack updates')
    else:
        # If the stack doesn't already exist then create it instead
        # of updating it.
        create_stack(stack, template)
        # Continue the job so the pipeline will wait for the CloudFormation
        # stack to be created.
        continue_job_later(job_id, 'Stack create started')

def check_stack_update_status(job_id, stack):
    """Monitor an already-running CloudFormation update/create

    Succeeds, fails or continues the job depending on the stack status.

    Args:
        job_id: The CodePipeline job ID
        stack: The stack to monitor

    """
    status = get_stack_status(stack)
    if status in ['UPDATE_COMPLETE', 'CREATE_COMPLETE']:
        # If the update/create finished successfully then
        # succeed the job and don't continue.
        put_job_success(job_id, 'Stack update complete')

    elif status in ['UPDATE_IN_PROGRESS', 'UPDATE_ROLLBACK_IN_PROGRESS',
    'UPDATE_ROLLBACK_COMPLETE_CLEANUP_IN_PROGRESS', 'CREATE_IN_PROGRESS',
    'ROLLBACK_IN_PROGRESS', 'UPDATE_COMPLETE_CLEANUP_IN_PROGRESS']:
        # If the job isn't finished yet then continue it
        continue_job_later(job_id, 'Stack update still in progress')

    else:
        # If the Stack is a state which isn't "in progress" or "complete"
```

```
        # then the stack update/create has failed so end the job with
        # a failed result.
        put_job_failure(job_id, 'Update failed: ' + status)

def get_user_params(job_data):
    """Decodes the JSON user parameters and validates the required properties.

    Args:
        job_data: The job data structure containing the UserParameters string which
 should be a valid JSON structure

    Returns:
        The JSON parameters decoded as a dictionary.

    Raises:
        Exception: The JSON can't be decoded or a property is missing.

    """
    try:
        # Get the user parameters which contain the stack, artifact and file
 settings
        user_parameters = job_data['actionConfiguration']['configuration']
['UserParameters']
        decoded_parameters = json.loads(user_parameters)

    except Exception as e:
        # We're expecting the user parameters to be encoded as JSON
        # so we can pass multiple values. If the JSON can't be decoded
        # then fail the job with a helpful message.
        raise Exception('UserParameters could not be decoded as JSON')

    if 'stack' not in decoded_parameters:
        # Validate that the stack is provided, otherwise fail the job
        # with a helpful message.
        raise Exception('Your UserParameters JSON must include the stack name')

    if 'artifact' not in decoded_parameters:
        # Validate that the artifact name is provided, otherwise fail the job
        # with a helpful message.
        raise Exception('Your UserParameters JSON must include the artifact name')

    if 'file' not in decoded_parameters:
        # Validate that the template file is provided, otherwise fail the job
        # with a helpful message.
```

```
        raise Exception('Your UserParameters JSON must include the template file
 name')

    return decoded_parameters

def setup_s3_client(job_data):
    """Creates an S3 client

    Uses the credentials passed in the event by CodePipeline. These
    credentials can be used to access the artifact bucket.

    Args:
        job_data: The job data structure

    Returns:
        An S3 client with the appropriate credentials

    """
    key_id = job_data['artifactCredentials']['accessKeyId']
    key_secret = job_data['artifactCredentials']['secretAccessKey']
    session_token = job_data['artifactCredentials']['sessionToken']

    session = Session(aws_access_key_id=key_id,
        aws_secret_access_key=key_secret,
        aws_session_token=session_token)
    return session.client('s3',
 config=botocore.client.Config(signature_version='s3v4'))

def lambda_handler(event, context):
    """The Lambda function handler

    If a continuing job then checks the CloudFormation stack status
    and updates the job accordingly.

    If a new job then kick of an update or creation of the target
    CloudFormation stack.

    Args:
        event: The event passed by Lambda
        context: The context passed by Lambda

    """
    try:
        # Extract the Job ID
```

```
        job_id = event['CodePipeline.job']['id']

        # Extract the Job Data
        job_data = event['CodePipeline.job']['data']

        # Extract the params
        params = get_user_params(job_data)

        # Get the list of artifacts passed to the function
        artifacts = job_data['inputArtifacts']

        stack = params['stack']
        artifact = params['artifact']
        template_file = params['file']

        if 'continuationToken' in job_data:
            # If we're continuing then the create/update has already been triggered
            # we just need to check if it has finished.
            check_stack_update_status(job_id, stack)
        else:
            # Get the artifact details
            artifact_data = find_artifact(artifacts, artifact)
            # Get S3 client to access artifact with
            s3 = setup_s3_client(job_data)
            # Get the JSON template file out of the artifact
            template = get_template(s3, artifact_data, template_file)
            # Kick off a stack update or create
            start_update_or_create(job_id, stack, template)

    except Exception as e:
        # If any other exceptions which we didn't expect are raised
        # then fail the job and log the exception message.
        print('Function failed due to exception.')
        print(e)
        traceback.print_exc()
        put_job_failure(job_id, 'Function exception: ' + str(e))

    print('Function complete.')
    return "Complete."
```

6. 将处理程序保留为默认值，将角色保留为您之前选择或创建的名称 **CodePipelineLambdaExecRole**。
7. 在 Basic settings (基本设置) 中，对于 Timeout (超时)，将默认值 3 秒替换为 **20** 秒。

8. 选择保存。
9. 在 CodePipeline 控制台中，编辑管道，将该函数添加为管道中某个阶段的操作。对要更改的管道选择编辑，然后选择添加操作组。在编辑操作页面的操作名称中，为您的操作输入一个名称。在操作提供程序中，选择 Lambda。

   在输入构件中，选择 `MyTemplate`。在中 UserParameters，您必须提供包含三个参数的 JSON 字符串：

   - 堆栈名称
   - AWS CloudFormation 模板名称和文件路径
   - 输入构件

   使用大括号 ({})，并使用逗号分隔各个参数。例如，要为带有输入工件的管道创建名为的堆栈 *`MyTestStack`*，请在中UserParameters输入：{“stack”:” “*`MyTemplate`*，“file”: “template-package/template.json”，*`MyTestStack`*“artifact”:” “}。*`MyTemplate`*

   > **Note**
   >
   > 即使您已在中指定了输入构件 UserParameters，也必须在 “输入构件” 中为操作指定此输入对象。

10. 保存您对管道所做的更改，然后手动发布更改，以便测试操作和 Lambda 函数。

重试阶段中失败的操作

您可以重试失败的阶段，而不必从头开始再次运行管道。为此，您可以重试阶段中失败的操作，也可以从阶段中的第一个操作开始重试所有操作。当重试阶段中失败的操作时，所有仍在进行的操作都会继续运行，而失败的操作将会重新触发。当从阶段中的第一个操作开始重试失败的阶段时，该阶段中不会有任何正在进行的操作。在重试某个阶段之前，它其中的操作必须全部失败，或者有些操作失败，有些操作成功。

> **Important**
>
> 重试失败的阶段时，会从其中第一个操作开始重试所有操作，而重试失败的操作则会重试阶段中所有失败的操作。这会覆盖同一执行中之前成功的操作的输出构件。
> 尽管构件可能会被覆盖，但之前成功的操作的执行历史记录仍会保留。

如果使用控制台查看管道，则可重试的阶段上将会出现重试阶段按钮或重试失败的操作按钮。

如果您使用的是 AWS CLI，则可以使用get-pipeline-state命令来确定是否有任何操作失败。

> **Note**
>
> 在以下情况下，您可能无法重试阶段：
>
> - 阶段中的所有操作都已成功，因此该阶段未处于失败状态。
> - 阶段失败后总体管道结构发生变化。
> - 阶段中的另一个重试尝试已在进行中。

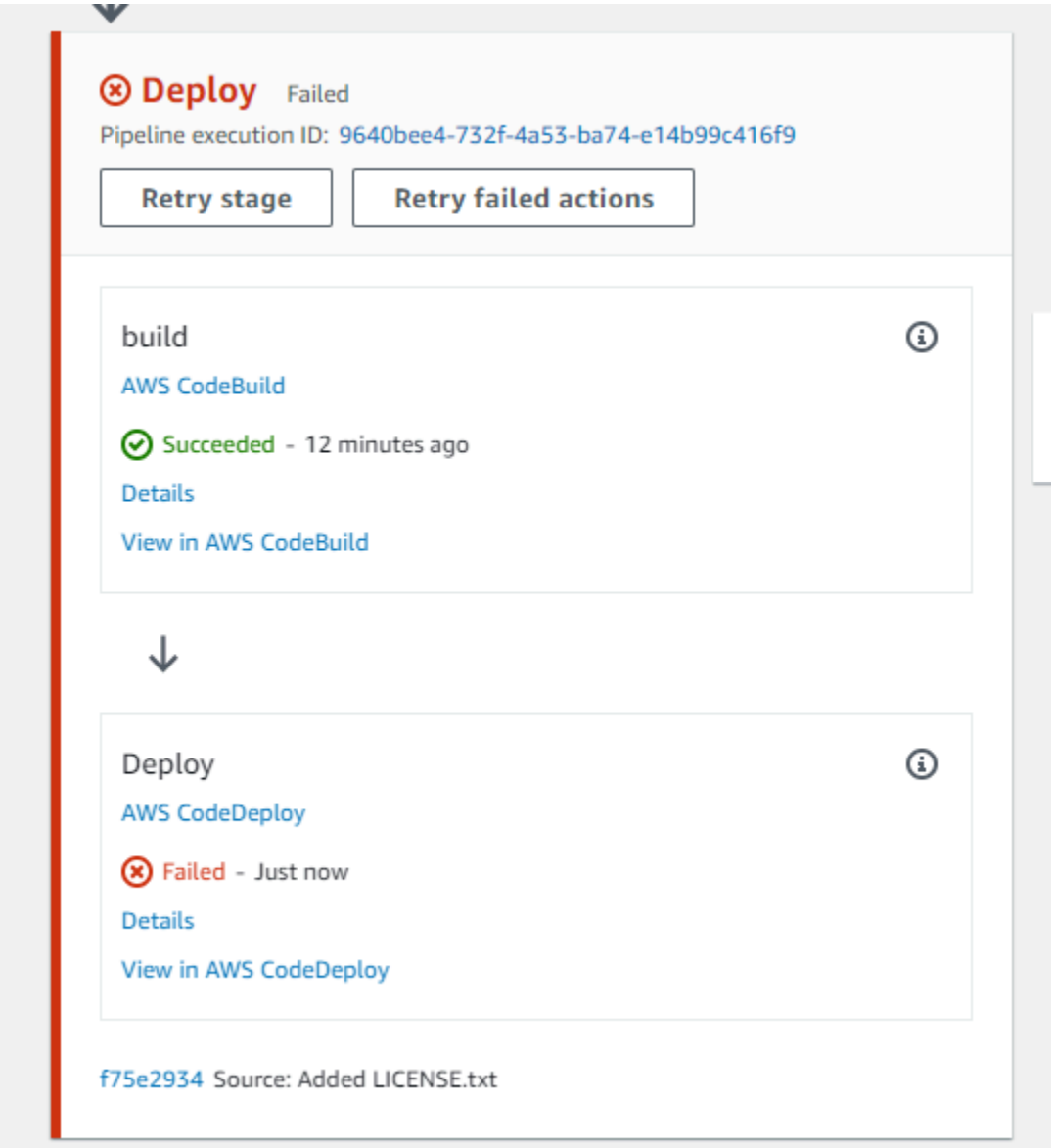
主题

重试失败的操作（控制台）

重试失败的阶段或阶段中失败的操作（控制台）

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   将显示与您的 AWS 账户关联的所有管道的名称。
2. 在名称中，选择管道的名称。
3. 找到包含失败操作的阶段，然后选择以下项之一：
   - 要重试阶段中的所有操作，请选择重试阶段。
   - 要仅重试阶段中失败的操作，请选择重试失败的操作。



如果该阶段中所有重试的操作都已成功完成，管道将继续运行。

重试失败的操作 (CLI)

重试失败的阶段或阶段中失败的操作 (CLI)

要使用重试所有操作或所有失败的操作，请使用以下参数运行retry-stage-execution命令： AWS CLI

```
--pipeline-name <value>
--stage-name <value>
```

```
--pipeline-execution-id <value>
--retry-mode ALL_ACTIONS/FAILED_ACTIONS
```

> **Note**
>
> 您可以对 `retry-mode` 使用的值是 `FAILED_ACTIONS` 和 `ALL_ACTIONS`。

1. 在终端（Linux、macOS 或 Unix）或命令提示符 (Windows) 下，运行 retry-stage-execution 命令，如以下名为 `MyPipeline` 的管道示例中所示。

   ```
   aws codepipeline retry-stage-execution --pipeline-name MyPipeline --stage-name
    Deploy --pipeline-execution-id b59babff-5f34-EXAMPLE --retry-mode FAILED_ACTIONS
   ```

   输出会返回执行 ID：

   ```
   {
       "pipelineExecutionId": "b59babff-5f34-EXAMPLE"
   }
   ```

2. 您还可以使用 JSON 输入文件运行该命令。您首先要创建一个 JSON 文件来标识管道、包含失败操作的阶段以及该阶段中的最新管道执行。然后，运行带 `--cli-input-json` 参数的 retry-stage-execution 命令。要检索您需要用于 JSON 文件的详细信息，最简单的方法是使用 get-pipeline-state 命令。

   a. 在终端（Linux、macOS 或 Unix）或命令提示符 (Windows) 处，对管道运行 get-pipeline-state 命令。例如，对于名为的管道 MyFirstPipeline，您可以键入类似于以下内容的内容：

      ```
      aws codepipeline get-pipeline-state --name MyFirstPipeline
      ```

      该命令的响应包括每个阶段的管道状态信息。在以下示例中，响应表示 Staging 阶段有一个或多个操作失败：

      ```
      {
          "updated": 1427245911.525,
          "created": 1427245911.525,
          "pipelineVersion": 1,
          "pipelineName": "MyFirstPipeline",
          "stageStates": [
      ```

```
        {
            "actionStates": [...],
            "stageName": "Source",
            "latestExecution": {
                "pipelineExecutionId": "9811f7cb-7cf7-SUCCESS",
                "status": "Succeeded"
            }
        },
        {
            "actionStates": [...],
            "stageName": "Staging",
            "latestExecution": {
                "pipelineExecutionId": "3137f7cb-7cf7-EXAMPLE",
                "status": "Failed"
            }
        }
    ]
}
```

b. 在纯文本编辑器中，创建一个 JSON 格式文件，您需要记录以下内容：

- 包含失败操作的管道的名称
- 包含失败操作的阶段的名称
- 该阶段中最新管道执行的 ID
- 重试模式。

对于前面的 MyFirstPipeline 示例，您的文件将如下所示：

```
{
    "pipelineName": "MyFirstPipeline",
    "stageName": "Staging",
    "pipelineExecutionId": "3137f7cb-7cf7-EXAMPLE",
    "retryMode": "FAILED_ACTIONS"
}
```

c. 使用类似于 **retry-failed-actions.json** 的名称保存文件。

d. 调用您运行 retry-stage-execution 命令时创建的文件。例如：

> **Important**
>
> 务必在文件名前包含 `file://`。此命令中需要该项。

```
aws codepipeline retry-stage-execution --cli-input-json file://retry-failed-actions.json
```

e. 要查看重试的结果，请打开 CodePipeline 控制台并选择包含失败操作的管道，或者再次使用该get-pipeline-state命令。有关更多信息，请参阅 在中查看管道和详细信息 CodePipeline。

在中管理批准操作 CodePipeline

在中 AWS CodePipeline，您可以将批准操作添加到管道中您希望停止管道执行的阶段，以便具有所需 AWS Identity and Access Management 权限的人员可以批准或拒绝该操作。

如果操作获得批准，管道执行将恢复。如果操作被拒绝（或者在管道到达该操作后停止的七天内无人批准或拒绝该操作），结果将与操作失败相同，并且管道执行不会继续。

您可能会出于以下原因使用手动审批：

- 您希望有人在修订获准进入管道的下一阶段之前，执行代码审核或变更管理审核。
- 您希望有人在应用程序的最新版本发布之前，对其执行手动质量保证测试，或者确认生成项目的完整性。
- 您希望有人在新文本或更新的文本发布到公司网站之前对其进行审核。

中手动批准操作的配置选项 CodePipeline

CodePipeline 提供了三个配置选项，可用来告知批准者有关批准操作的信息。

发布审批通知您可以配置一项审批操作，以便当管道在该操作停止时，向 Amazon Simple Notification Service 主题发布消息。Amazon SNS 会将消息传递给订阅该主题的每个端点。您必须使用在包含批准操作的管道所在 AWS 区域创建的主题。在创建主题时，我们建议您为主题指定一个标识其用途的名称，并采用 `MyFirstPipeline-us-east-2-approval` 这样的格式。

向 Amazon SNS 主题发布审批通知时，您可以从各种形式中做出选择，如电子邮件或 SMS 收件人、SQS 队列、HTTP/HTTPS 端点或您使用 Amazon SNS 调用的 AWS Lambda 函数。有关 Amazon SNS 主题通知的信息，请参阅以下主题：

- 什么是 Amazon Simple Notification Service？
- 在 Amazon SNS 中创建主题
- 将 Amazon SNS 消息发送至 Amazon SQS 队列
- 为队列订阅 Amazon SNS 主题
- 将 Amazon SNS 消息发送至 HTTP/HTTPS 终端节点
- 使用 Amazon SNS 通知调用 Lambda 函数

有关为审批操作通知生成的 JSON 数据的结构，请参阅手动批准通知的 JSON 数据格式 CodePipeline。

指定用于审阅的 URL 作为审批操作配置的一部分，您可以指定要审阅的 URL。该 URL 可能是一个指向以下内容的链接：指向您希望审批者进行测试的 Web 应用程序，或指向包含有关审批请求的更多信息的页面。该 URL 包含在发布到 Amazon SNS 主题的通知中。审批者可以使用控制台或 CLI 查看它。

输入审批者注释 在创建审批操作时，您还可以添加注释，这些注释将显示给接收通知的人员或者在控制台或 CLI 响应中查看操作的人员。

无配置选项 您也可以选择不配置这三个选项中的任何一个。您可能不需要它们，例如，您可直接通知某人该操作已可供他们审核，或者您只希望管道停止，直到您决定自行批准操作。

中批准操作的设置和工作流程概述 CodePipeline

下面概述了设置和使用手动审批的过程。

1. 您应向组织中的一个或多个 IAM 角色授予批准或拒绝审批操作所需的 IAM 权限。
2. （可选）如果您使用的是 Amazon SNS 通知，请确保您在 CodePipeline 操作中使用的服务角色有权访问 Amazon SNS 资源。
3. （可选）如果使用 Amazon SNS 通知，则应创建一个 Amazon SNS 主题并向其添加一个或多个订阅用户或端点。
4. 当您使用 AWS CLI 创建管道时，或者在使用 CLI 或控制台创建管道之后，您可以向管道中的某个阶段添加批准操作。

如果要使用通知，您应在操作的配置中包含 Amazon SNS 主题的 Amazon 资源名称 (ARN)。（ARN 是亚马逊资源的唯一标识符。 亚马逊 SNS 主题的 ARN 的结构*### arn: aws: sns: us-east-2:80398 ##:*。MyApprovalTopic 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

5. 管道在到达审批操作时停止。如果在操作的配置中包含 Amazon SNS 主题 ARN，则会向 Amazon SNS 主题发布通知，并向该主题的所有订阅用户或订阅的端点发送消息，其中包含用于在控制台中查看审批操作的链接。
6. 审批者检查目标 URL 并查看注释（如果有）。
7. 审批者使用控制台、CLI 或软件开发工具包提供摘要注释并提交响应：
   - 批准：管道执行将恢复。
   - 拒绝：阶段状态更改为“Failed”，并且管道执行不恢复。

   如果七天内未提交任何响应，操作将标记为“Failed”。

向中的 IAM 用户授予批准权限 CodePipeline

在组织内的 IAM 用户可以批准或拒绝审批操作之前，必须向他们授予访问管道和更新审批操作状态的权限。您可以通过将 `AWSCodePipelineApproverAccess` 托管式策略附加到 IAM 用户、角色或组，来授予其访问账户中的所有管道和审批操作的权限；或者您可以通过指定 IAM 用户、角色或组可以访问的各个资源来授予有限权限。

> **Note**
>
> 本主题中描述的权限授予非常有限的访问权限。要使用户、角色或组可以执行的不止是批准或拒绝审批操作，您可以附加其他托管策略。有关可用的托管策略的信息 CodePipeline，请参阅AWS 的托管策略 AWS CodePipeline。

授予对所有管道和审批操作的审批权限

对于需要在中执行批准操作的用户 CodePipeline，请使用`AWSCodePipelineApproverAccess`托管策略。

要提供访问权限，请为您的用户、组或角色添加权限：

- 中的用户和群组 AWS IAM Identity Center：

创建权限集合。按照《AWS IAM Identity Center 用户指南》中创建权限集的说明进行操作。

- 通过身份提供商在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中为第三方身份提供商创建角色（联合身份验证）的说明进行操作。

- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

指定对特定管道和审批操作的审批权限

对于需要在中执行批准操作的用户 CodePipeline，请使用以下自定义策略。在下面的策略中，指定用户可以访问的各项资源。例如，根据以下策略的授权，用户只能批准或拒绝美国东部（俄亥俄州）区域 (us-east-2) `MyFirstPipeline` 管道中名为 `MyApprovalAction` 的操作：

> **Note**
>
> 仅`codepipeline:ListPipelines`当 IAM 用户需要访问 CodePipeline 控制面板以查看此管道列表时，才需要该权限。如果不需要访问控制台，则可以忽略`codepipeline:ListPipelines`。

使用 JSON 策略编辑器创建策略

1. 登录 AWS Management Console 并打开 IAM 控制台，网址为 https://console.aws.amazon.com/iam/。
2. 在左侧的导航窗格中，选择策略。

   如果这是您首次选择策略，则会显示欢迎访问托管式策略页面。选择开始使用。
3. 在页面的顶部，选择创建策略。
4. 在策略编辑器部分，选择 JSON 选项。
5. 输入以下 JSON 策略文档：

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

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:ListPipelines"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:GetPipeline",
                "codepipeline:GetPipelineState",
                "codepipeline:GetPipelineExecution"
            ],
            "Resource": "arn:aws:codepipeline:us-
east-2:80398EXAMPLE:MyFirstPipeline"
        },
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:PutApprovalResult"
            ],
            "Resource": "arn:aws:codepipeline:us-
east-2:80398EXAMPLE:MyFirstPipeline/MyApprovalStage/MyApprovalAction"
        }
    ]
}
```

6. 选择下一步。

> **Note**
>
> 您可以随时在可视化和 JSON 编辑器选项卡之间切换。不过，如果您进行更改或在可视化编辑器中选择下一步，IAM 可能会调整策略结构以针对可视化编辑器进行优化。有关更多信息，请参阅《IAM 用户指南》中的调整策略结构。

7. 在查看并创建页面上，为您要创建的策略输入策略名称和描述（可选）。查看此策略中定义的权限以查看策略授予的权限。

8. 选择创建策略可保存新策略。

向服务角色授予 Amazon SNS 权限 CodePipeline

如果您计划在批准操作需要审核时使用 Amazon SNS 向主题发布通知，则必须向您在 CodePipeline 操作中使用的服务角色授予访问亚马逊 SNS 资源的权限。您可以使用 IAM 控制台将此权限添加到您的服务角色中。

在下面的策略中，指定 SNS 发布策略。对于以下策略，您可以将其命名为 `SNSPublish`。通过将以下策略附加到您的服务角色来使用该策略。

> **Important**
>
> 请确保您登录时使用的账户信息 AWS Management Console 与中使用的账户信息相同入门 CodePipeline。

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

使用 JSON 策略编辑器创建策略

1. 登录 AWS Management Console 并打开 IAM 控制台，网址为 https://console.aws.amazon.com/iam/。
2. 在左侧的导航窗格中，选择策略。

   如果这是您首次选择策略，则会显示欢迎访问托管式策略页面。选择开始使用。
3. 在页面的顶部，选择创建策略。
4. 在策略编辑器部分，选择 JSON 选项。
5. 输入或粘贴一个 JSON 策略文档。有关 IAM 策略语言的详细信息，请参阅 IAM JSON 策略参考。
6. 解决策略验证过程中生成的任何安全警告、错误或常规警告，然后选择下一步。

> **Note**
>
> 您可以随时在可视化和 JSON 编辑器选项卡之间切换。不过，如果您进行更改或在可视化编辑器中选择下一步，IAM 可能会调整策略结构以针对可视化编辑器进行优化。有关更多信息，请参阅《IAM 用户指南》中的调整策略结构。

7. （可选）在中创建或编辑策略时 AWS Management Console，可以生成可在模板中使用的 JSON 或 YAML 策略 AWS CloudFormation 模板。

   为此，请在策略编辑器中选择操作，然后选择生成 CloudFormation模板。要了解更多信息 AWS CloudFormation，请参阅AWS CloudFormation 用户指南中的AWS Identity and Access Management 资源类型参考。
8. 向策略添加完权限后，选择下一步。
9. 在查看并创建页面上，为您要创建的策略键入策略名称和描述（可选）。查看此策略中定义的权限以查看策略授予的权限。
10. （可选）通过以密钥值对的形式附加标签来向策略添加元数据。有关在 IAM 中使用标签的更多信息，请参阅 《IAM 用户指南》 中的标记 IAM 资源。
11. 选择创建策略可保存您的新策略。

向中的管道添加手动批准操作 CodePipeline

您可以将批准操作添加到管道中您希望停止 CodePipeline 管道的阶段，以便其他人可以手动批准或拒绝该操作。

> **Note**
>
> 不能将审批操作添加到源阶段。源阶段只能包含源操作。

如果要在审批操作准备好接受审核时使用 Amazon SNS 发送通知，您必须先完成以下先决条件：

- 向您的 CodePipeline 服务角色授予访问亚马逊 SNS 资源的权限。有关信息，请参阅 向服务角色授予 Amazon SNS 权限 CodePipeline。
- 向您组织中的一个或多个 IAM 身份授予更新审批操作状态的权限。有关信息，请参阅 向中的 IAM 用户授予批准权限 CodePipeline。

在此示例中，您将创建一个新的审批阶段，并向该阶段添加一个手动审批操作。您也可以向包含其他操作的现有阶段添加手动审批操作。

向 CodePipeline 管道添加手动批准操作（控制台）

您可以使用 CodePipeline 控制台向现有 CodePipeline管道添加批准操作。如果要在创建新管道时添加批准操作，则必须使用 AWS CLI。

1. 打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。
2. 在名称中，选择管道。
3. 在管道详细信息页中，选择编辑。
4. 如果您要在新的阶段添加审批操作，请在管道中您要添加审批请求的时间点选择 + 添加阶段，然后输入阶段名称。在添加阶段页面上的阶段名称中，输入新的阶段名称。例如，添加一个新阶段并将其命名为 `Manual_Approval`。

   如果您要在现有阶段中添加审批操作，请选择编辑阶段。
5. 在要在其中添加审批操作的阶段中，选择 + 添加操作组。
6. 在编辑操作页中，执行以下操作：

   1. 在操作名称中，输入用于识别操作的名称。
   2. 在操作提供程序中的审批下，选择手动审批。
   3. （可选）在 SNS 主题 ARN 中，选择用来发送审批操作通知的主题名称。
   4. （可选）在用于审阅的 URL 中，输入您希望审批者检查的页面或应用程序的 URL。审批者可以通过管道控制台视图中包含的链接访问此 URL。
   5. （可选）在注释中，输入您要与审批者共享的任何其他信息。
   6. 选择保存。

向 CodePipeline管道添加手动批准操作 (CLI)

您可以使用 CLI 将审批操作添加到现有管道，或者在创建管道时添加审批操作。您可以通过以下方式实现此目的：在创建或编辑的阶段中添加手动审批类型的审批操作。

有关创建和编辑管道的更多信息，请参阅在中创建管道 CodePipeline和在中编辑管道 CodePipeline。

要向管道中添加一个仅包含审批操作的阶段，那么在创建或更新管道时，您应包括类似于以下示例的内容。

> **Note**
>
> configuration 部分是可选的。这只是文件的一部分，而不是整个结构。有关更多信息，请参阅 CodePipeline 管道结构参考。

```
{
    "name": "MyApprovalStage",
    "actions": [
        {
            "name": "MyApprovalAction",
            "actionTypeId": {
                "category": "Approval",
                "owner": "AWS",
                "version": "1",
                "provider": "Manual"
            },
            "inputArtifacts": [],
            "outputArtifacts": [],
            "configuration": {
                "NotificationArn": "arn:aws:sns:us-
east-2:80398EXAMPLE:MyApprovalTopic",
                "ExternalEntityLink": "http://example.com",
                "CustomData": "The latest changes include feedback from Bob."},
            "runOrder": 1
        }
    ]
}
```

如果审批操作位于包含其他操作的阶段中，则 JSON 文件中包含该阶段的这一部分可能与以下示例类似。

> **Note**
>
> configuration 部分是可选的。这只是文件的一部分，而不是整个结构。有关更多信息，请参阅 CodePipeline 管道结构参考。

```
,
{
```

```
    "name": "Production",
    "actions": [
        {
            "inputArtifacts": [],
            "name": "MyApprovalAction",
            "actionTypeId": {
                "category": "Approval",
                "owner": "AWS",
                "version": "1",
                "provider": "Manual"
            },
            "outputArtifacts": [],
            "configuration": {
                "NotificationArn": "arn:aws:sns:us-
east-2:80398EXAMPLE:MyApprovalTopic",
                "ExternalEntityLink": "http://example.com",
                "CustomData": "The latest changes include feedback from Bob."
            },
            "runOrder": 1
        },
        {
            "inputArtifacts": [
                {
                    "name": "MyApp"
                }
            ],
            "name": "MyDeploymentAction",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "version": "1",
                "provider": "CodeDeploy"
            },
            "outputArtifacts": [],
            "configuration": {
                "ApplicationName": "MyDemoApplication",
                "DeploymentGroupName": "MyProductionFleet"
            },
            "runOrder": 2
        }
    ]
}
```

批准或拒绝中的批准操作 CodePipeline

当管道包含审批操作时，管道执行在添加操作的那一点停止。管道将不会恢复，除非有人手动批准该操作。如果审批者拒绝该操作，或者在管道因审批操作停止后的七天内未收到审批响应，则管道状态将变为“Failed”。

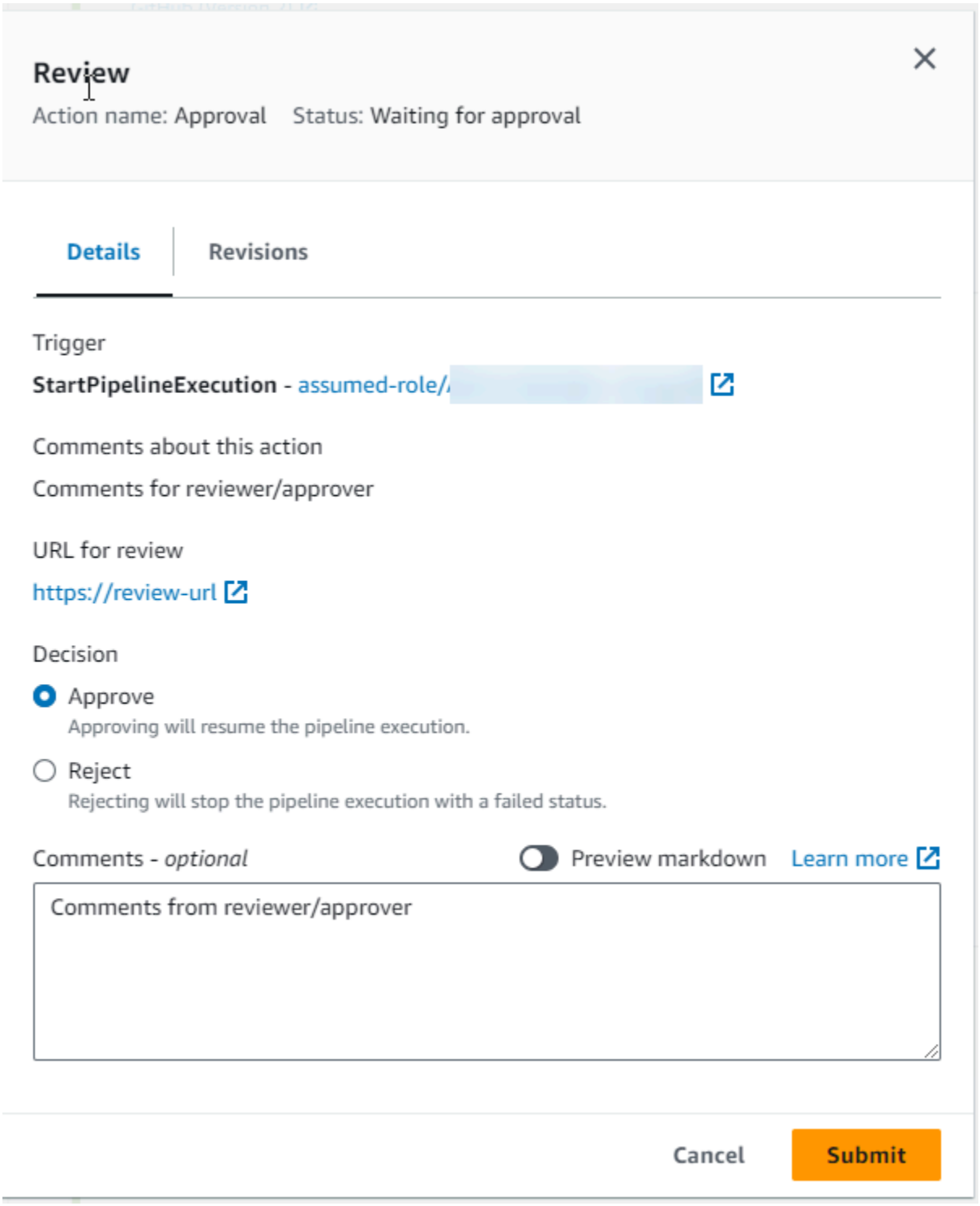
如果向管道中添加审批操作的人员配置了通知功能，则您可能会收到一封包含管道信息和批准状态的电子邮件。

批准或拒绝审批操作（控制台）

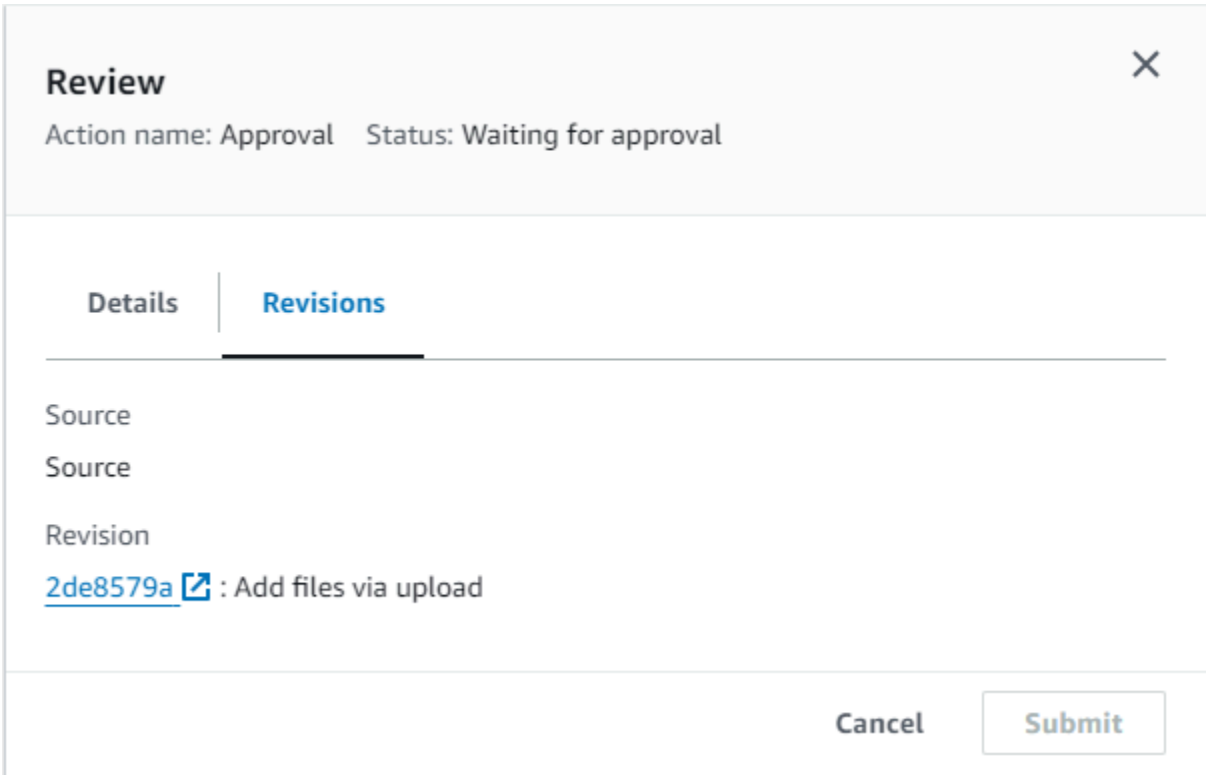
如果您收到包含审批操作的直接链接的通知，请选择批准或拒绝链接，登录控制台，然后继续执行此处的步骤 7。否则，请按照所有以下步骤操作。

1. 打开 CodePipeline 控制台，网址为 https://console.aws.amazon.com/codepipeline/。
2. 在 All Pipelines 页面上，选择管道名称。
3. 查找包含审批操作的阶段。选择审核。

   将显示审核对话框。详细信息选项卡显示审核内容和注释。



修订选项卡显示执行的源修订。



4. 在详细信息选项卡上，查看注释和 URL（如果有）。该消息还会显示内容的 URL 供您查看（如果已经包含）。
5. 如果提供了 URL，请选择操作中的用于审阅的 URL 链接以打开目标网页，然后审核内容。
6. 在审核窗口中，输入审核注释，例如您为何批准或拒绝操作，然后选择批准或拒绝。
7. 选择提交。

批准或拒绝审批请求（CLI）

要使用 CLI 来响应审批操作，您必须先使用 get-pipeline-state 命令来检索与最近一次执行审批操作相关联的令牌。

1. 在终端（Linux、macOS 或 Unix）或命令提示符（Windows）上，在包含批准操作的管道上运行get-pipeline-state命令。例如，对于名为的管道 *MyFirstPipeline*，输入以下内容：

```
aws codepipeline get-pipeline-state --name MyFirstPipeline
```

2. 在命令响应中，找到 `token` 值，该值显示在审批操作的 `actionStates` 部分的 `latestExecution` 中，如此处所示：

```
{
```

```
    "created": 1467929497.204,
    "pipelineName": "MyFirstPipeline",
    "pipelineVersion": 1,
    "stageStates": [
        {
            "actionStates": [
                {
                    "actionName": "MyApprovalAction",
                    "currentRevision": {
                        "created": 1467929497.204,
                        "revisionChangeId": "CEM7d6Tp7zfelUSLCPPwo234xEXAMPLE",
                        "revisionId": "HYGp7zmwbCPPwo23xCMdTeqIlEXAMPLE"
                    },
                    "latestExecution": {
                        "lastUpdatedBy": "identity",
                        "summary": "The new design needs to be reviewed before
 release.",
                        "token": "1a2b3c4d-573f-4ea7-a67E-XAMPLETOKEN"
                    }
                }
//More content might appear here
}
```

3. 在纯文本编辑器中，创建一个 JSON 格式文件，您需要添加以下内容：

   - 包含审批操作的管道的名称。
   - 包含审批操作的阶段的名称。
   - 审批操作的名称。
   - 您在上一步中收集的令牌值。
   - 您对操作的响应（“批准”或“拒绝”）。此响应必须大写。
   - 您的摘要注释。

   对于前面的*MyFirstPipeline*示例，您的文件应如下所示：

```
{
  "pipelineName": "MyFirstPipeline",
  "stageName": "MyApprovalStage",
  "actionName": "MyApprovalAction",
  "token": "1a2b3c4d-573f-4ea7-a67E-XAMPLETOKEN",
  "result": {
    "status": "Approved",
```

```
    "summary": "The new design looks good. Ready to release to customers."
  }
}
```

4. 使用类似于 **approvalstage-approved.json** 的名称保存文件。
5. 运行put-approval-result命令，指定批准 JSON 文件的名称，类似于以下内容：

> **Important**
>
> 务必在文件名前包含 `file://`。此命令中需要该项。

```
aws codepipeline put-approval-result --cli-input-json file://approvalstage-approved.json
```

手动批准通知的 JSON 数据格式 CodePipeline

对于使用 Amazon SNS 通知的审批操作，在管道停止时，将会创建与操作有关的 JSON 数据并将其发布到 Amazon SNS。您可以使用 JSON 输出向 Amazon SQS 队列发送消息或者在 AWS Lambda中调用函数。

> **Note**
>
> 本指南不解决如何使用 JSON 配置通知的问题。有关信息，请参阅 Amazon SNS 开发者指南中的向 Amazon SQS 队列发送 Amazon SNS 消息和使用 Amazon SNS 通知调用 Lambda 函数。

以下示例显示了经 CodePipeline批准后可用的 JSON 输出的结构。

```
{
    "region": "us-east-2",
    "consoleLink": "https://console.aws.amazon.com/codepipeline/home?region=us-east-2#/view/MyFirstPipeline",
    "approval": {
        "pipelineName": "MyFirstPipeline",
        "stageName": "MyApprovalStage",
        "actionName": "MyApprovalAction",
```

```
        "token": "1a2b3c4d-573f-4ea7-a67E-XAMPLETOKEN",
        "expires": "2016-07-07T20:22Z",
        "externalEntityLink": "http://example.com",
        "approvalReviewLink": "https://console.aws.amazon.com/codepipeline/
home?region=us-east-2#/view/MyFirstPipeline/MyApprovalStage/MyApprovalAction/
approve/1a2b3c4d-573f-4ea7-a67E-XAMPLETOKEN",
        "customData": "Review the latest changes and approve or reject within seven
 days."
    }
}
```

在中添加跨区域操作 CodePipeline

AWS CodePipeline 包括许多操作，可帮助您为自动发布流程配置构建、测试和部署资源。您可以向管道中添加与您的管道不同的 AWS 区域中的操作。如果 AWS 服务 是操作的提供者，并且此操作类型/提供者类型与您的管道位于不同的 AWS 区域，则这是跨区域操作。

> **Note**
>
> 支持跨区域操作，并且只能在支持跨 AWS 区域的 CodePipeline 区域中创建。有关支持的 AWS 区域列表 CodePipeline，请参阅中的配额 AWS CodePipeline。

您可以使用控制台 AWS CLI、或在管道中 AWS CloudFormation 添加跨区域操作。

> **Note**
>
> 中的某些操作类型 CodePipeline 可能仅在某些 AWS 地区可用。另请注意，有些 AWS 区域可能有操作类型可用，但该操作类型的特定 AWS 提供者不可用。

当您创建或编辑管道时，管道区域中必须有构件存储桶，然后每个您计划执行操作的区域必须有一个构件存储桶。有关 `ArtifactStores` 参数的更多信息，请参阅 CodePipeline 管道结构参考。

> **Note**
>
> CodePipeline 在执行跨区域操作时，处理将工件从一个 AWS 区域复制到其他区域的情况。

如果您使用控制台创建管道或跨区域操作，则默认对象存储桶由 CodePipeline 在您有操作的区域中进行配置。当您使用 AWS CLI AWS CloudFormation、或 SDK 创建管道或跨区域操作时，您需要为每个有操作的区域提供构件存储桶。

> **Note**
>
> 您必须在跨区域操作所在的 AWS 区域以及与您的管道相同的账户中创建构件存储桶和加密密钥。

无法创建以下操作类型的跨区域操作：

- 源操作
- 第三方操作
- 自定义操作

> **Note**
>
> CodePipeline在中使用跨区域 Lambda 调用操作时，使用PutJobSuccessResultPutJobFailureResult和执行 lambda 的状态应发送到 AWS 存在 Lambda 函数的区域，而不是发送到存在的区域。 CodePipeline

当管道将跨区域操作作为阶段的一部分时，仅 CodePipeline 将跨区域操作的输入项目从管道区域复制到操作的区域。

> **Note**
>
> 管道区域和维护您的 CloudWatch 事件变更检测资源的区域保持不变。托管管道的区域不会改变。

管理管道中的跨区域操作（控制台）

您可以使用 CodePipeline 控制台向现有管道添加跨区域操作。要使用“创建管道”向导创建具有跨区域操作的新管道，请参阅创建管道（控制台）。

在控制台中，您可以通过选择操作提供方和区域字段，在管道阶段创建跨区域操作，该字段列出您在该区域中为该提供方创建的资源。添加跨区域操作时，将在操作的区域中 CodePipeline使用单独的构件存储桶。有关跨区域构件存储桶的更多信息，请参阅CodePipeline 管道结构参考。

向管道阶段添加跨区域操作（控制台）

使用控制台向管道添加跨区域操作。

> **Note**
>
> 如果在保存更改时管道正在运行，则执行将无法执行。

添加跨区域操作

1. 在 http://console.aws.amazon.com/codesuite/codepipeline/home 登录控制台。
2. 选择您的管道，然后选择 Edit (编辑)。
3. 如果要添加新阶段，请在图表底部选择 + Add stage (+ 添加阶段)；如果要向现有阶段添加操作，请选择 Edit stage (编辑阶段)。
4. 在 Edit: <Stage> (编辑: <Stage>) 上，选择 + Add action group (+ 添加操作组) 以添加序列操作。或者，选择 + Add action (+ 添加操作) 以添加并行操作。
5. 在 Edit action (编辑操作) 页面上：
   a. 在操作名称中，输入跨区域操作的名称。
   b. 在 Action provider (操作提供方) 中，选择操作提供方。
   c. 在 “区域” 中，选择您已创建或计划为操作创建资源的 AWS 区域。选择区域后，将列出该区域的可用资源以供选择。Re g ion 字段指定为此操作类型和提供者类型创建 AWS 资源的位置。此字段仅在操作提供方是 AWS 服务的情况下对操作显示。区域字段默认为与您的管道相同的 AWS 区域 。
   d. 在 Input artifacts (输入构件) 中，选择上一阶段的相应输入。例如，如果前一个阶段是源阶段，请选择SourceArtifact。
   e. 完成您正在配置的操作提供方的所有必填字段。
   f. 在 Output artifacts (输出构件) 中，选择要输出到下一阶段的适当输出。例如，如果下一阶段是部署阶段，请选择BuildArtifact。
   g. 选择保存。
6. 在 Edit: <Stage> (编辑: <Stage>) 中，选择 Done (完成)。

7. 选择保存。

编辑管道阶段的跨区域操作（控制台）

使用控制台编辑管道中的现有跨区域操作。

> **Note**
> 如果在保存更改时管道正在运行，则执行将无法执行。

编辑跨区域操作

1. 在 https://console.aws.amazon.com/codesuite/codepipeline/home. 登录控制台。
2. 选择您的管道，然后选择 Edit (编辑)。
3. 选择编辑阶段。
4. 在 Edit: <Stage> (编辑: <Stage>) 上，选择图标以编辑现有操作。
5. 在 Edit action (编辑操作) 页面上，根据需要更改字段。
6. 在 Edit: <Stage> (编辑: <Stage>) 中，选择 Done (完成)。
7. 选择保存。

删除管道阶段的跨区域操作（控制台）

使用控制台删除管道中的现有跨区域操作。

> **Note**
> 如果在保存更改时管道正在运行，则执行将无法执行。

删除跨区域操作

1. 在 http://console.aws.amazon.com/codesuite/codepipeline/home 登录控制台。
2. 选择您的管道，然后选择 Edit (编辑)。
3. 选择编辑阶段。

4. 在 Edit: <Stage> (编辑: <Stage>) 上，选择图标以删除现有操作。
5. 在 Edit: <Stage> (编辑: <Stage>) 中，选择 Done (完成)。
6. 选择保存。

在管道中添加跨区域操作 (CLI)

您可以使用 AWS CLI 向现有管道添加跨区域操作。

要使用在管道阶段创建跨区域操作 AWS CLI，请添加配置操作和可选`region`字段。您还必须已在操作的区域中创建一个构件存储桶。您可以使用 `artifactStores` 参数来包含每个区域的构件存储桶的列表，而不是提供单区域管道的 `artifactStore` 参数。

> **Note**
>
> 在本演练及其示例中，*RegionA* 是创建管道的区域。它可以访问用于存储管道工件的 *RegionA* Amazon S3 存储桶以及使用的服务角色。 CodePipeline*RegionB* 是创建所使用的 CodeDeploy 应用程序、部署组和服务角色 CodeDeploy 的区域。

先决条件

您必须已经完成以下操作：

- 在 *RegionA* 中创建管道。
- *RegionB* 中的 Amazon S3 构件桶。
- 区域 *b* 中用于您的操作的资源，例如用于跨区域部署操作的 CodeDeploy 应用程序和部署组。

在管道中添加跨区域操作 (CLI)

使用 AWS CLI 向管道添加跨区域操作。

添加跨区域操作

1. 对于 *RegionA* 中的管道，运行 get-pipeline 命令以将管道结构复制到 JSON 文件中。例如，对于名为 `MyFirstPipeline` 的管道，运行以下命令：

```
aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
```

该命令不会返回任何结果，但您创建的文件将出现在您运行命令所在的目录中。

2. 添加 region 字段以添加一个包含您的跨区域操作的新阶段，其中包含您的操作的区域和资源。以下 JSON 示例在新区域中添加了一个带有跨区域部署操作的 Deploy 阶段 CodeDeploy，其中提供者位于新区域us-east-1。

```
{
                "name": "Deploy",
                "actions": [
                    {
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "name": "Deploy",
                        "region": "RegionB",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeDeploy"
                        },
                        "outputArtifacts": [],
                        "configuration": {
                            "ApplicationName": "name",
                            "DeploymentGroupName": "name"
                        },
                        "runOrder": 1
                    }
```

3. 在管道结构中，删除 artifactStore 字段并为新的跨区域操作添加 artifactStores 映射。映射必须包括您在其中执行操作的每个 AWS 区域的条目。对于映射中的每个条目，资源必须位于相应的 AWS 区域中。在以下示例中，ID-A 是 *RegionA* 的加密密钥 ID，ID-B 是 *RegionB* 的加密密钥 ID。

```
"artifactStores":{
   "RegionA":{
      "encryptionKey":{
         "id":"ID-A",
         "type":"KMS"
      },
```

```
      "location":"Location1",
      "type":"S3"
   },
   "RegionB":{
      "encryptionKey":{
         "id":"ID-B",
         "type":"KMS"
      },
      "location":"Location2",
      "type":"S3"
   }
}
```

下面的 JSON 示例将 us-west-2 存储桶显示为 `my-storage-bucket` 并添加名为 `my-storage-bucket-us-east-1` 的新 us-east-1 存储桶。

```
        "artifactStores": {
            "us-west-2": {
                "type": "S3",
                "location": "my-storage-bucket"
            },
            "us-east-1": {
                "type": "S3",
                "location": "my-storage-bucket-us-east-1"
            }
        },
```

4. 如果您要使用通过 get-pipeline 命令检索到的管道结构，请删除 JSON 文件中的 `metadata` 行。否则，update-pipeline 命令无法使用它。删除 `"metadata": { }` 行以及 `"created"`、`"pipelineARN"` 和 `"updated"` 字段。

   例如，从结构中删除以下各行：

```
"metadata": {
  "pipelineArn": "arn:aws:codepipeline:region:account-ID:pipeline-name",
  "created": "date",
  "updated": "date"
  }
```

   保存该文件。

5. 要应用更改，请运行 update-pipeline 命令，指定管道 JSON 文件：

> **Important**
>
> 务必在文件名前包含 `file://`。此命令中需要该项。

```
aws codepipeline update-pipeline --cli-input-json file://pipeline.json
```

该命令会返回编辑后的管道的整个结构。输出类似于以下内容。

```
{
    "pipeline": {
        "version": 4,
        "roleArn": "ARN",
        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "inputArtifacts": [],
                        "name": "Source",
                        "actionTypeId": {
                            "category": "Source",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeCommit"
                        },
                        "outputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "configuration": {
                            "PollForSourceChanges": "false",
                            "BranchName": "main",
                            "RepositoryName": "MyTestRepo"
                        },
                        "runOrder": 1
                    }
                ]
            },
            {
```

```
                "name": "Deploy",
                "actions": [
                    {
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "name": "Deploy",
                        "region": "us-east-1",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeDeploy"
                        },
                        "outputArtifacts": [],
                        "configuration": {
                            "ApplicationName": "name",
                            "DeploymentGroupName": "name"
                        },
                        "runOrder": 1
                    }
                ]
            }
        ],
        "name": "AnyCompanyPipeline",
        "artifactStores": {
            "us-west-2": {
                "type": "S3",
                "location": "my-storage-bucket"
            },
            "us-east-1": {
                "type": "S3",
                "location": "my-storage-bucket-us-east-1"
            }
        }
    }
}
```

> **Note**
>
> update-pipeline 命令会停止管道。如果在运行 update-pipeline 命令时正在通过管道运行修订，则该运行会被停止。您必须手动启动管道，通过升级后的管道运行此修订。使用 **start-pipeline-execution** 命令手动启动您的管道。

6. 更新您的管道后，跨区域操作将显示在控制台中。

在管道中添加跨区域操作 (AWS CloudFormation)

您可以使用 AWS CloudFormation 向现有管道添加跨区域操作。

使用添加跨区域操作 AWS CloudFormation

1. 将 `Region` 参数添加到模板中的 `ActionDeclaration` 资源，如以下示例所示：

```
ActionDeclaration:
              Type: Object
              Properties:
                ActionTypeId:
                  Type: ActionTypeId
                  Required: true
                Configuration:
                  Type: Map
                InputArtifacts:
                  Type: Array
                  ItemType:
```

```
                Type: InputArtifact
              Name:
                Type: String
                Required: true
              OutputArtifacts:
                Type: Array
                ItemType:
                  Type: OutputArtifact
              RoleArn:
                Type: String
              RunOrder:
                Type: Integer
              Region:
                Type: String
```

2. 在 Mappings 下，为映射键 RegionA 和 RegionB 的值的名为 SecondRegionMap 的映射添加区域映射，如此示例所示。在 Pipeline 资源下的 artifactStore 字段下，为您的新跨区域操作添加 artifactStores 映射，如下所示：

```
Mappings:
  SecondRegionMap:
    RegionA:
      SecondRegion: "RegionB"
    RegionB:
      SecondRegion: "RegionA"

...

          Properties:
            ArtifactStores:
              -
                Region: RegionB
                ArtifactStore:
                  Type: "S3"
                  Location: test-cross-region-artifact-store-bucket-RegionB
              -
                Region: RegionA
                ArtifactStore:
                  Type: "S3"
                  Location: test-cross-region-artifact-store-bucket-RegionA
```

以下 YAML 示例将 *RegionA* 存储桶显示为 us-west-2 并添加新的 *RegionB* 存储桶 eu-central-1：

```
Mappings:
  SecondRegionMap:
    us-west-2:
      SecondRegion: "eu-central-1"
    eu-central-1:
      SecondRegion: "us-west-2"

...

          Properties:
            ArtifactStores:
              -
                Region: eu-central-1
                ArtifactStore:
                  Type: "S3"
                  Location: test-cross-region-artifact-store-bucket-eu-central-1
              -
                Region: us-west-2
                ArtifactStore:
                  Type: "S3"
                  Location: test-cross-region-artifact-store-bucket-us-west-2
```

3. 将更新后的模板保存到本地计算机，然后打开 AWS CloudFormation 控制台。
4. 选择堆栈，然后选择为当前堆栈创建更改集。
5. 上传模板，然后查看 AWS CloudFormation中列出的更改。这些是要对堆栈进行的更改。您应在列表中看到新资源。
6. 选择执行。

使用变量

CodePipeline 生成变量中的一些操作。要使用变量，请执行以下操作：

- 将命名空间分配给某个操作，以使其生成的变量可用于下游操作配置。
- 配置下游操作以使用该操作生成的变量。

您可以查看每个操作执行的详细信息，以确定该操作在执行时生成的每个输出变量的值。

要查看使用变量的 step-by-step 示例，请执行以下操作：

- 有关使用上游操作 (CodeCommit) 中的变量并生成输出变量的 Lambda 操作的教程，请参阅。教程：将变量与 Lambda 调用操作一起使用
- 有关引用上游 AWS CloudFormation 操作堆栈输出变量的 CloudFormation 操作的教程，请参阅教程：创建使用 AWS CloudFormation 部署操作中的变量的管道。
- 有关手动批准操作的示例，其消息文本引用了解析为提 CodeCommit 交 ID 和提交消息的输出变量，请参阅示例：在手动审批中使用变量。
- 有关带有可解析为 GitHub 分支名称的环境变量的 CodeBuild 操作示例，请参阅示例：将 BranchName变量与 CodeBuild 环境变量一起使用。
- CodeBuild actions 将所有作为构建一部分导出的环境变量生成为变量。有关更多信息，请参阅 CodeBuild 操作输出变量。有关可在中使用的环境变量的列表 CodeBuild，请参阅《AWS CodeBuild 用户指南》中的构建环境中的环境变量。

主题

为操作配置变量

向管道添加操作时，您可以为其分配命名空间并将其配置为使用之前操作中的变量。

为操作配置变量（控制台）

此示例创建了一个包含 CodeCommit 源操作和 CodeBuild 生成操作的管道。该 CodeBuild 操作配置为使用该 CodeCommit 操作生成的变量。

如果未指定命名空间，则这些变量不可在操作配置中引用。使用控制台创建管道时，会自动生成每个操作的命名空间。

创建具有变量的管道

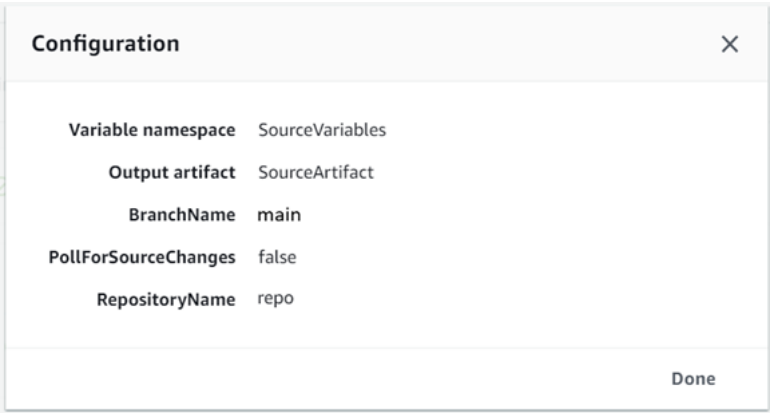
1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

2. 选择创建管道。为您的管道输入名称，然后选择 Next (下一步)。
3. 在源中，在提供者中，选择CodeCommit。为源操作选择 CodeCommit 存储库和分支，然后选择“下一步”。
4. 在“构建”中，在“提供者”中选择CodeBuild。选择现有的 CodeBuild 构建项目名称或选择“创建项目”。在“创建构建项目”上，创建生成项目，然后选择“返回到” CodePipeline。

   在 Environment variables (环境变量) 下，选择 Add environment variables (添加环境变量)。例如，使用变量语法 `#{codepipeline.PipelineExecutionId}` 输入执行 ID，使用变量语法 `#{SourceVariables.CommitId}` 输入提交 ID。

   > **Note**
   >
   > 您可以在向导的任何操作配置字段中输入变量语法。

5. 选择创建。
6. 创建管道后，您可以查看由向导创建的命名空间。在管道上，选择要查看其命名空间的阶段所对应的图标。在此示例中，将显示源操作自动生成的命名空间 `SourceVariables`。



编辑现有操作的命名空间

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 选择要编辑的管道，然后选择 Edit (编辑)。对于源阶段，选择 Edit stage (编辑阶段)。添加动 CodeCommit 作。
3. 在 Edit action (编辑操作) 中，查看 Variable namespace (变量命名空间) 字段。如果现有操作是以前创建或者未使用向导创建，则必须添加命名空间。在 Variable namespace (变量命名空间) 中，输入命名空间名称，然后选择 Save (保存)。

查看输出变量

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。
2. 成功创建管道并运行后，您可以在操作执行详细信息页面上查看变量。有关信息，请参阅 查看变量（控制台）。

为操作配置变量 (CLI)

在使用 create-pipeline 命令创建管道时，或者使用 update-pipeline 命令编辑管道时，您可以在操作配置中引用/使用变量。

如果未指定命名空间，则操作生成的变量不可在任何下游操作配置中引用。

使用命名空间配置操作

1. 按照中 在中创建管道 CodePipeline 的步骤使用 CLI 创建管道。启动输入文件以提供带有 `--cli-input-json` 参数的 create-pipeline 命令。在管道结构中，添加 `namespace` 参数并指定名称，例如 `SourceVariables`。

```
. . .
{
          "inputArtifacts": [],
          "name": "Source",
          "region": "us-west-2",
          "namespace": "SourceVariables",
          "actionTypeId": {
            "category": "Source",
            "owner": "AWS",
            "version": "1",
            "provider": "CodeCommit"
          },
          "outputArtifacts": [

. . .
```

2. 使用类似于 **`MyPipeline.json`** 的名称保存文件。
3. 在终端（Linux、macOS 或 Unix）或命令提示符 (Windows) 处，运行 create-pipeline 命令并创建管道。

   调用您运行 create-pipeline 命令时创建的文件。例如：

```
aws codepipeline create-pipeline --cli-input-json file://MyPipeline.json
```

配置下游操作以使用变量

1. 编辑输入文件以提供带有 --cli-input-json 参数的 update-pipeline 命令。在下游操作中，将变量添加到该操作的配置。变量由命名空间和键组成，以句点分隔。例如，要为管道执行 ID 和源提交 ID 添加变量，请为变量 #{codepipeline.PipelineExecutionId} 指定命名空间 codepipeline。为变量 #{SourceVariables.CommitId} 指定命名空间 SourceVariables。

```
{
    "name": "Build",
    "actions": [
        {
            "outputArtifacts": [
                {
                    "name": "BuildArtifacts"
                }
            ],
            "name": "Build",
            "configuration": {
                "EnvironmentVariables": "[{\"name\":\"Execution_ID\",\"value
\":\"#{codepipeline.PipelineExecutionId}\",\"type\":\"PLAINTEXT\"},{\"name\":
\"Commit_ID\",\"value\":\"#{SourceVariables.CommitId}\",\"type\":\"PLAINTEXT\"}]",
                "ProjectName": "env-var-test"
            },
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-west-2",
            "actionTypeId": {
                "provider": "CodeBuild",
                "category": "Build",
                "version": "1",
                "owner": "AWS"
            },
            "runOrder": 1
        }
```

```
    ]
},
```

2. 使用类似于 **MyPipeline.json** 的名称保存文件。
3. 在终端（Linux、macOS 或 Unix）或命令提示符 (Windows) 处，运行 create-pipeline 命令并创建管道。

   调用您运行 create-pipeline 命令时创建的文件。例如：

```
aws codepipeline create-pipeline --cli-input-json file://MyPipeline.json
```

查看输出变量

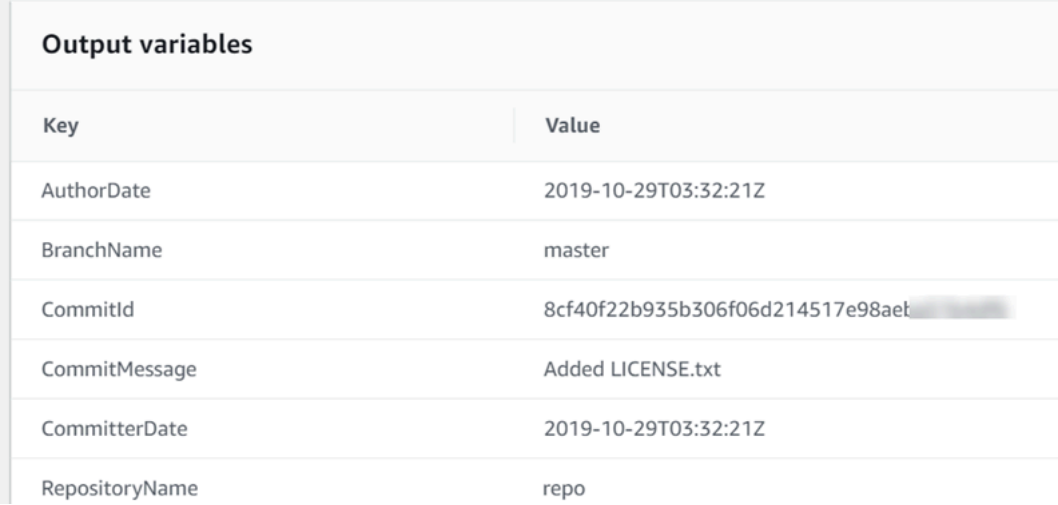
您可以查看操作执行详细信息，以查看该操作特定于每个执行的变量。

查看变量（控制台）

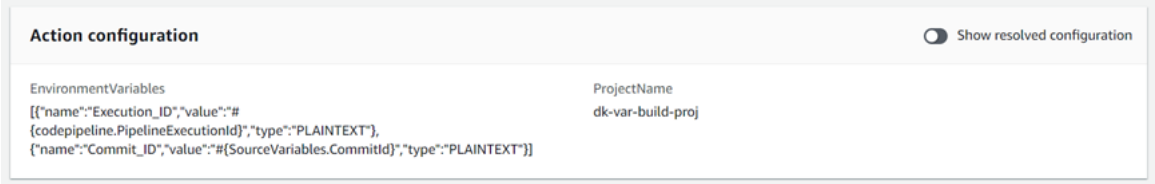
您可以使用控制台查看某个操作的变量。

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

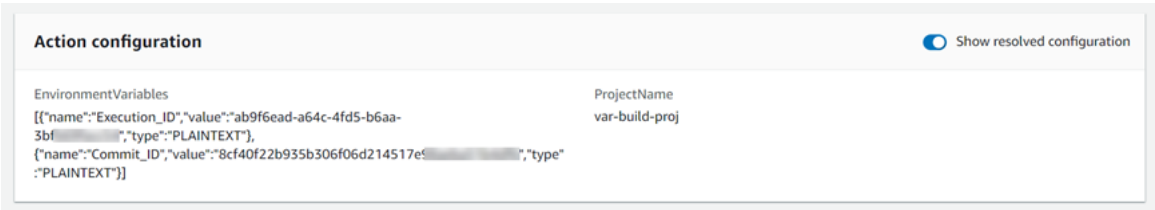
   将显示与您的 AWS 账户关联的所有管道的名称。
2. 在名称中，选择管道的名称。
3. 选择 View history (查看历史记录)。
4. 管道成功运行后，您可以查看由源操作生成的变量。选择 View history (查看历史记录)。在管道执行的操作列表中选择 S ourc e 以查看该操作的 CodeCommit 操作执行详细信息。在操作详细信息屏幕的 Output variables (输出变量) 下查看变量。



5. 管道成功运行后，您可以查看构建操作所使用的变量。选择 View history (查看历史记录)。在管道执行的操作列表中，选择 B uild 以查看该操作的 CodeBuild 操作执行详细信息。在操作详细信息页面上，查看 Action configuration (操作配置) 下的变量。此时将显示自动生成的命名空间。



默认情况下，Action configuration (操作配置) 显示变量语法。您可以选择 Show resolved configuration (显示已解析的配置) 以切换列表，显示在操作执行期间生成的值。



查看变量 (CLI)

您可以使用 list-action-executions 命令查看操作的变量。

1. 使用以下命令：

```
aws codepipeline list-action-executions
```

输出显示 `outputVariables` 参数，如此处所示。

```
"outputVariables": {
                    "BranchName": "main",
                    "CommitMessage": "Updated files for test",
                    "AuthorDate": "2019-11-08T22:24:34Z",
                    "CommitId": "d99b0083cc10EXAMPLE",
                    "CommitterDate": "2019-11-08T22:24:34Z",
                    "RepositoryName": "variables-repo"
                },
```

2. 使用以下命令：

```
aws codepipeline get-pipeline --name <pipeline-name>
```

在操作的操作配置中 CodeBuild ，您可以查看变量：

```
{
    "name": "Build",
    "actions": [
        {
            "outputArtifacts": [
                {
                    "name": "BuildArtifact"
                }
            ],
            "name": "Build",
            "configuration": {
                "EnvironmentVariables": "[{\"name\":\"Execution_ID\",\"value
\":\"#{codepipeline.PipelineExecutionId}\",\"type\":\"PLAINTEXT\"},{\"name\":
\"Commit_ID\",\"value\":\"#{SourceVariables.CommitId}\",\"type\":\"PLAINTEXT\"}]",
                "ProjectName": "env-var-test"
            },
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-west-2",
            "actionTypeId": {
                "provider": "CodeBuild",
                "category": "Build",
```

```
                "version": "1",
                "owner": "AWS"
            },
            "runOrder": 1
        }
    ]
},
```

示例：在手动审批中使用变量

当您为操作指定命名空间，并且该操作生成输出变量时，可以添加手动审批，该审批将在审批消息中显示变量。此示例演示如何向手动审批消息中添加变量语法。

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   将显示与您的 AWS 账户关联的所有管道的名称。选择要向其添加审批的管道。
2. 要编辑您的管道，请选择编辑。在源操作后添加手动审批。在操作名称中，输入审批操作的名称。
3. 在 Action provider (操作提供程序) 中，选择 Manual approval (手动审批)。
4. 在 “供审阅的网址” 中，将的变量语法`CommitId`添加到您 CodeCommit 的 URL 中。请确保您使用分配给源操作的命名空间。例如，具有默认命名空间的 CodeCommit操作的变量语法`SourceVariables`为`#{SourceVariables.CommitId}`。

   在评论的 `CommitMessage` 中，输入提交消息：

   ```
   Please approve this change. Commit message: #{SourceVariables.CommitMessage}
   ```
5. 管道成功运行后，您可以在审批消息中查看变量值。

示例：将 BranchName变量与 CodeBuild 环境变量一起使用

向管道添加 CodeBuild 操作时，可以使用 CodeBuild 环境变量来引用来自上游源操作的`BranchName`输出变量。通过中操作的输出变量 CodePipeline，您可以创建自己的 CodeBuild环境变量以在构建命令中使用。

此示例说明如何将 GitHub 源操作中的输出变量语法添加到 CodeBuild 环境变量。本示例中的输出变量语法表示的 GitHub 源操作输出变量`BranchName`。操作成功运行后，变量解析为显示 GitHub 分支名称。

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   将显示与您的 AWS 账户关联的所有管道的名称。选择要向其添加审批的管道。
2. 要编辑您的管道，请选择编辑。在包含您的 CodeBuild 操作的舞台上，选择编辑阶段。
3. 选择图标以编辑您的 CodeBuild 操作。
4. 在编辑操作页面的环境变量下，输入以下内容：

   - 在名称中，输入您的环境变量的名称。
   - 在值中，输入您的管道输出变量的变量语法，其中包括分配给您的源操作的命名空间。例如，具有默认命名空间的 GitHub操作的输出变量语法`SourceVariables`为`#{SourceVariables.BranchName}`。
   - 在类型中，选择纯文本。
5. 在管道成功运行后，您可以看到所解析的输出变量如何成为环境变量中的值。选择以下操作之一：

   - CodePipeline 控制台：选择您的管道，然后选择 “历史记录”。选择最近的管道执行。
     - 在时间轴下，选择源的选择器。这是生成 GitHub 输出变量的源操作。选择查看执行详细信息。在输出变量下，查看此操作生成的输出变量列表。
     - 在时间轴下，选择构建的选择器。这是为构建项目指定 CodeBuild 环境变量的生成操作。选择查看执行详细信息。在 “操作配置” 下，查看您的 CodeBuild环境变量。选择显示已解析配置。您的环境变量值是 GitHub 源操作中已解析的`BranchName`输出变量。在此示例中，解析的值为 `main`。

       有关更多信息，请参阅 查看变量（控制台）。
   - CodeBuild 控制台：选择您的构建项目并选择构建运行的链接。在环境变量下，已解析的输出变量是 CodeBuild 环境变量的值。在此示例中，环境变量 Name 为`BranchName`，Value 是 GitHub 源操作中已解析的`BranchName`输出变量。在此示例中，解析的值为 `main`。


Build logs
Phase details
Reports
Environment variables
Build details
Resource utilization
Name
Value
Type
BranchName
main
PLAINTEXT

在中处理舞台过渡 CodePipeline

过渡是管道阶段之间的链接，可以禁用或启用。默认处于启用状态。当您重新启用已禁用的过渡时，最新的修订将经历管道的其余阶段，除非已禁用超过 30 天。对于已禁用超过 30 天的过渡，将不会恢复管道执行，除非检测到新的更改或者您手动重新运行管道。

您可以使用 AWS CodePipeline 控制台或 AWS CLI 来禁用或启用管道中各个阶段之间的过渡。

> **Note**
>
> 您可以使用审批操作暂停管道的运行，直到手动批准它继续运行。有关更多信息，请参阅 在中管理批准操作 CodePipeline。

主题

禁用或启用过渡（控制台）

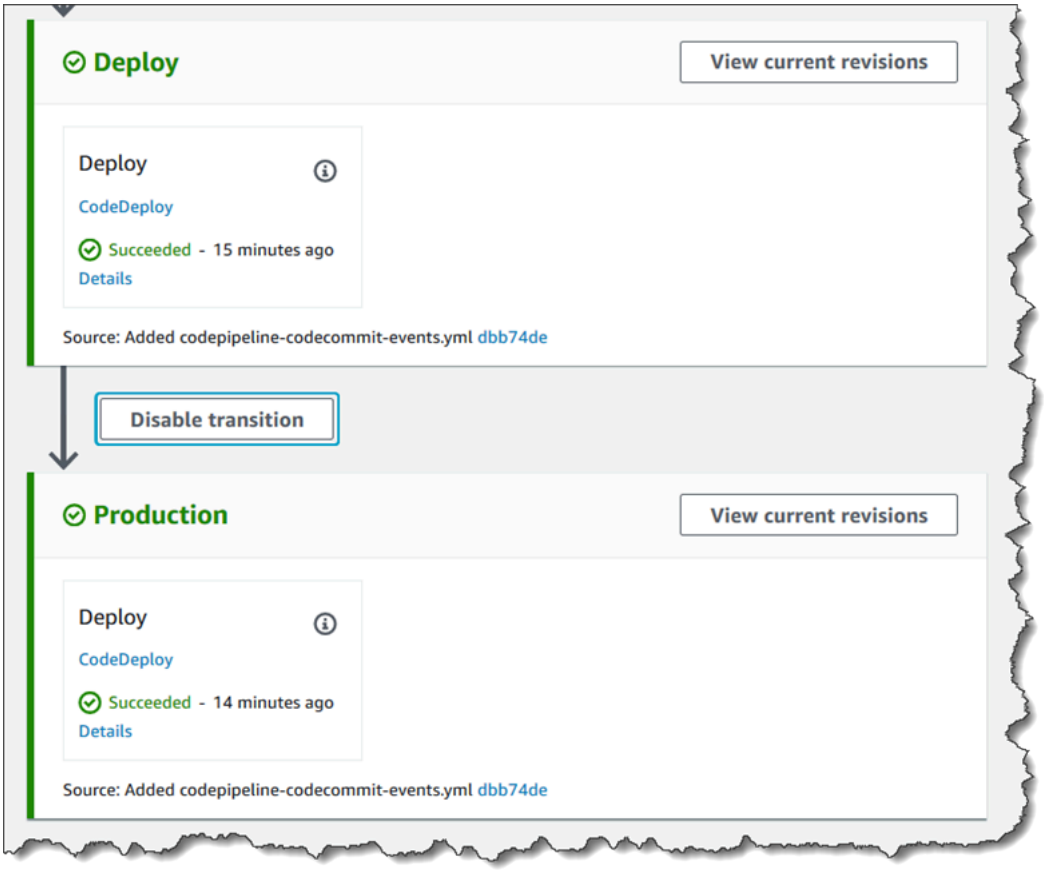
要在管道中禁用或启用过渡

1. 登录 AWS Management Console 并打开 CodePipeline 控制台，网址为 http://console.aws.amazon.com/codesuite/codepipeline/home。

   将会显示所有与您的 AWS 账户关联的管道的名称。

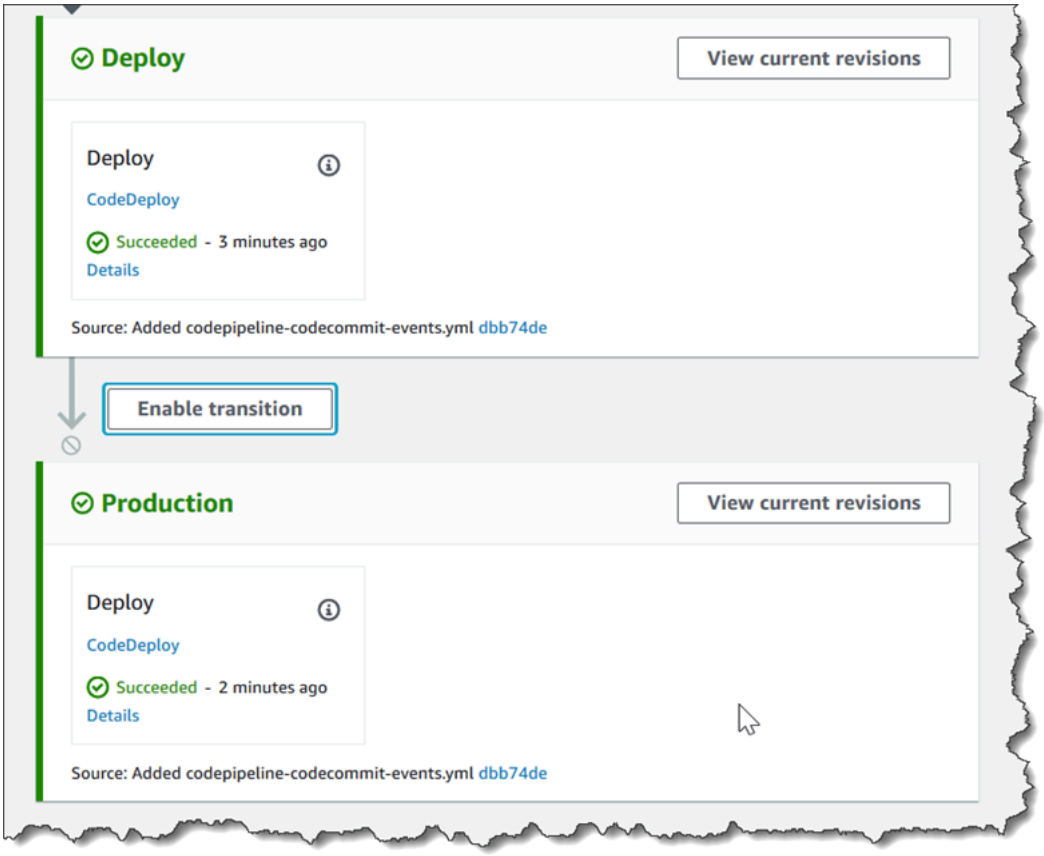
2. 在 Name 中，选择您要为其启用或禁用过渡的管道的名称。这将打开管道的详细视图，包括管道阶段之间的过渡。

3. 找到您要运行的最后一个阶段后面的箭头，然后选择它旁边的按钮。例如，在以下管道中，如果您希望 Staging 阶段中的操作运行，而名为 Production 的阶段中的操作不运行，则请选择这两个阶段之间的 Disable transition (禁用过渡) 按钮：



4. 在 Disable transition (禁用过渡) 对话框中，输入禁用过渡的原因，然后选择 Disable (禁用)。

   按钮会发生更改，显示箭头之前的阶段与箭头之后的阶段之间的过渡是禁用的。禁用过渡之后出现的阶段中已经运行的任何修订将继续通过管道，但任何后续修订不再继续通过禁用的过渡。



5. 选择箭头旁的 Enable transition (启用过渡) 按钮。在 Enable transition 对话框中，选择 Enable。管道将立即在两个阶段之间启用过渡。如果过渡禁用后，任何修订已经在之前的阶段中运行，则几分钟后，管道将开始在之前禁用的过渡之后出现的阶段中运行最新修订。管道将在管道的所有其余阶段中运行修订。

> **Note**
>
> 启用过渡后，更改可能需要几秒钟才能显示在 CodePipeline 控制台中。

禁用或启用过渡 (CLI)

要使用禁用阶段之间的过渡 AWS CLI，请运行disable-stage-transition命令。要启用已禁用的过渡，请运行 enable-stage-transition 命令。

要禁用过渡

1. 打开终端（Linux、macOS 或 Unix）或disable-stage-transition命令提示符 (Windows)，然后使用运行命令，指定管道名称、要禁用过渡的阶段名称、过渡类型以及禁用该阶段过渡的原因。 AWS

CLI 与使用控制台不同，您还必须指定您是禁用过渡到该阶段 (入站) 还是禁用在所有操作完成后从该阶段过渡出去 (出站)。

例如，要在名为的管道中禁用向名为 *Stag* ing 的阶段的过渡 *MyFirstPipeline*，可以键入类似于以下内容的命令：

```
aws codepipeline disable-stage-transition --pipeline-name MyFirstPipeline --stage-
name Staging --transition-type Inbound --reason "My Reason"
```

该命令不返回任何内容。

2. 要验证过渡是否已禁用，请在 CodePipeline 控制台中查看管道或运行get-pipeline-state命令。有关更多信息，请参阅 查看管道（控制台） 和 查看管道详细信息和历史记录（CLI）。

要启用过渡

1. 打开终端（Linux、macOS 或 Unix）或enable-stage-transition命令提示符 (Windows)，然后使用运行命令，指定管道名称、要启用过渡的阶段名称以及过渡类型。 AWS CLI

   例如，要允许过渡到名为的管道中名为 *St* aging 的阶段 *MyFirstPipeline*，可以键入类似于以下内容的命令：

   ```
   aws codepipeline enable-stage-transition --pipeline-name MyFirstPipeline --stage-
   name Staging  --transition-type Inbound
   ```

   该命令不返回任何内容。

2. 要验证过渡是否已禁用，请在 CodePipeline 控制台中查看管道或运行get-pipeline-state命令。有关更多信息，请参阅 查看管道（控制台） 和 查看管道详细信息和历史记录（CLI）。

监控管道

监控是保持 AWS CodePipeline可靠性、可用性和性能的重要环节。您应该从 AWS 解决方案的各个部分收集监控数据，以便在出现多点故障时可以更轻松地进行调试。在开始监控之前，您应该创建一个监控计划以回答以下问题：

- 监控目的是什么？
- 您将监控哪些资源？
- 监控这些资源的频率如何？
- 您可以使用哪些监控工具？
- 谁负责执行监控任务？
- 在出现错误时应通知谁？

您可以使用以下工具来监控您的 CodePipeline 管道及其资源：

- EventBridge 事件总线事件 — 您可以监控中的 CodePipeline事件 EventBridge，从而检测管道、阶段或操作执行状态的变化。 EventBridge 将该数据路由到目标，例如 AWS Lambda 和 Amazon 简单通知服务。 EventBridge 事件与 Amazon CloudWatch 活动中显示的事件相同。
- 开发者工具控制台中的管道事件通知 — 您可以使用在控制台中设置的通知来监控 CodePipeline 事件，然后创建 Amazon Simple Notification Service 主题并订阅。有关更多信息，请参阅开发人员工具控制台用户指南中的什么是通知。
- AWS CloudTrail— CloudTrail 用于捕获由您的账户或代表您的 AWS 账户进行的 API 调用，并将日志文件传输到 Amazon S3 存储桶。 CodePipeline 您可以选择在传送新日志文件时 CloudWatch 发布 Amazon SNS 通知，这样您就可以快速采取行动。
- 控制台和 CLI — 您可以使用 CodePipeline 控制台和 CLI 查看有关管道状态或特定管道执行的详细信息。

主题

- 监视 CodePipeline 事件
- 事件占位存储桶参考
- 使用记录 CodePipeline API 调用 AWS CloudTrail

监视 CodePipeline 事件

您可以在中监控 CodePipeline 事件 EventBridge，它会提供来自您自己的应用程序、 software-as-a-service (SaaS) 应用程序和的实时数据流 AWS 服务。 EventBridge 将该数据路由到目标，例如 AWS Lambda 和 Amazon 简单通知服务。这些事件与 Amazon Events 中显示 CloudWatch 的事件相同，Amazon Events 提供描述 AWS 资源变化的近乎实时的系统事件流。有关更多信息，请参阅什么是亚马逊 EventBridge？ 在《亚马逊 EventBridge 用户指南》中。

> **Note**
>
> Amazon EventBridge 是管理您的活动的首选方式。Amazon Ev CloudWatch ent EventBridge s 和 Amazon Events 是相同的底层服务和 API，但 EventBridge 提供了更多功能。您在 CloudWatch 活动中所做的更改或 EventBridge 将显示在每个控制台中。

事件由规则组成。通过选择以下选项来配置规则：

- 事件模式。每条规则都以事件模式表示，其中包含要监控的事件的来源和类型以及事件目标。要监控事件，您需要创建一条规则，将所监视的服务作为事件源，例如 CodePipeline。例如，您可以创建一个带有事件模式的规则，当管道、阶段或操作的状态发生变化时，该事件模式 CodePipeline 用作事件源来触发规则。
- 目标。新规则将选定的服务作为事件目标。您可能想设置目标服务以发送通知，捕获状态信息，采取纠正措施，启动事件或采取其他操作。添加目标时，还必须向授予权限 EventBridge 以允许其调用所选目标服务。

每种类型的执行状态更改事件发出包含特定消息内容的通知，其中：

- 初始 `version` 条目显示事件的版本号。
- `detail` 管道下面的 `version` 条目显示管道结构版本号。
- `detail` 管道下面的 `execution-id` 条目显示导致状态更改的管道执行的执行 ID。请参阅 AWS CodePipeline API 参考中的 GetPipelineExecution API 调用。
- 该 `pipeline-execution-attempt` 条目显示特定执行 ID 的尝试次数或重试次数。

CodePipeline EventBridge 每当您的资源状态发生 AWS 账户 变化时，都会向其报告事件。对于以下资源，事件将在保证 at-least-once 的基础上发出：

- 管道执行
- 阶段执行
- 操作执行

事件由 EventBridge 触发，事件模式和架构详述如上所述。对于已处理的事件，例如通过在开发人员工具控制台中配置的通知收到的事件，事件消息包含的事件模式字段会有些变化。例如，`detail-type` 字段转换为 `detailType`。有关更多信息，请参阅《亚马逊 PutEvents API 参考》中的 EventBridge API 调用。

以下示例显示了的事件 CodePipeline。在可能的情况下，每个示例都显示已发出事件的架构以及已处理事件的架构。

主题

详细信息类型

在设置要监控的事件时，可以选择事件的详细信息类型。

您可以配置在以下状态发生更改时发送的通知：

- 指定的管道或所有管道。您可以使用 `"detail-type": "CodePipeline Pipeline Execution State Change"` 控制该行为。
- 指定管道或所有管道中的指定阶段或所有阶段。您可以使用 `"detail-type": "CodePipeline Stage Execution State Change"` 控制该行为。
- 指定管道或所有管道的指定阶段或所有阶段中的指定操作或所有操作。您可以使用 `"detail-type": "CodePipeline Action Execution State Change"` 控制该行为。

> **Note**
>
> 由发出的事件 EventBridge 包含`detail-type`参数，该参数将在处理事件`detailType`时转换为该参数。

详细信息类型	状态	描述
CodePipeline 管道执行状态更改	CANCELED	已取消管道执行，因为已更新管道结构。
	FAILED	未成功完成管道执行。
	RESUMED	已重试失败的管道执行以响应 `RetryStageExecution` API 调用。
	STARTED	当前正在运行管道执行。
	STOPPED (已停止)	停止过程已完成，管道执行已停止。
	STOPPING (正在停止)	由于发出了停止并等待或停止并放弃管道执行的请求，管道执行正在停止。
	SUCCEEDED	已成功完成管道执行。
	SUPERSEDED	在该管道执行等待完成下一阶段时，启动了较新的管道执行并通过管道。
CodePipeline 阶段执行状态更改	CANCELED	已取消阶段，因为已更新管道结构。
	FAILED	阶段未成功完成。
	RESUMED	已重试失败的阶段以响应 `RetryStageExecution` API 调用。
	STARTED	当前正在运行阶段。
	STOPPED (已停止)	停止过程已完成，阶段执行已停止。

详细信息类型	状态	描述
	STOPPING (正在停止)	由于发出了停止并等待或停止并放弃管道执行的请求，阶段执行正在停止。
	SUCCEEDED	已成功完成阶段。
CodePipeline 动作执行状态更改	ABANDONED	由于发出了停止并放弃管道执行的请求，已放弃操作。
	CANCELED	已取消操作，因为已更新管道结构。
	FAILED	对于审批操作，FAILED (已失败) 状态表示审查者已拒绝操作，或者由于操作配置不正确而失败。
	STARTED	当前正在运行操作。
	SUCCEEDED	已成功完成操作。

管道级事件

当管道执行状态发生变化时，就会发出管道级事件。

主题

- 管道 STARTED 事件
- 管道 STOPPING 事件
- 管道 SUCCEEDED 事件
- 管道 SUCCEEDED（含 Git 标签的示例）
- 管道 FAILED 事件
- 管道 FAILED（含 Git 标签的示例）

管道 STARTED 事件

在启动管道执行时，它发出一个事件以发送包含以下内容的通知。此示例适用于 `us-east-1` 区域中名为 "`myPipeline`" 的管道。`id` 字段表示事件 ID，`account` 字段表示创建管道的账户 ID。

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:03:07Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "trigger-type": "StartPipelineExecution",
            "trigger-detail": "arn:aws:sts::123456789012:assumed-role/Admin/my-user"
        },
        "state": "STARTED",
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Pipeline Execution State Change",
    "region": "us-east-1",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:44:50Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
east-1:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "trigger-type": "StartPipelineExecution",
```

```
            "trigger-detail": "arn:aws:sts::123456789012:assumed-role/Admin/my-user"
        },
        "state": "STARTED",
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "additionalAttributes": {}
}
```

管道 STOPPING 事件

在停止管道执行时，它发出一个事件以发送包含以下内容的通知。此示例适用于 us-west-2 区域中名为 myPipeline 的管道。

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:02:20Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "STOPPING",
        "version": 3.0,
        "pipeline-execution-attempt": 1.0
        "stop-execution-comments": "Stopping the pipeline for an update"
    }
}
```

管道 SUCCEEDED 事件

在管道执行成功时，它发出一个事件以发送包含以下内容的通知。此示例适用于 us-east-1 区域中名为 myPipeline 的管道。

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:03:44Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "SUCCEEDED",
        "version": 3.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Pipeline Execution State Change",
    "region": "us-east-1",
    "source": "aws.codepipeline",
    "time": "2021-06-30T22:13:51Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "SUCCEEDED",
```

```
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {}
}
```

管道 SUCCEEDED（含 Git 标签的示例）

在管道执行有一个阶段已经重试并成功时，它发出一个事件以发送包含以下内容的通知。此示例适用于为 Git 标签配置了 `execution-trigger` 的 `eu-central-1` 区域中名为 `myPipeline` 的管道。

> **Note**
>
> `execution-trigger` 字段的值是 `tag-name` 或 `branch-name`，具体取决于触发管道的事件类型。

```
{
    "version": "0",
    "id": "b128b002-09fd-4574-4eba-27152726c777",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2023-10-26T13:50:53Z",
    "region": "eu-central-1",
    "resources": [
        "arn:aws:codepipeline:eu-central-1:123456789012:BuildFromTag"
    ],
    "detail": {
        "pipeline": "BuildFromTag",
        "execution-id": "e17b5773-cc0d-4db2-9ad7-594c73888de8",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "author-display-name": "Mary Major",
            "full-repository-name": "mmajor/sample-project",
            "provider-type": "GitLab",
            "author-email": "email_address",
            "commit-message": "Update file README.md",
```

```
            "author-date": "2023-08-16T21:08:08Z",
            "tag-name": "gitlab-v4.2.1",
            "commit-id": "commit_ID",
            "connection-arn": "arn:aws:codestar-connections:eu-
central-1:123456789012:connection/0f5b706a-1a1d-46c5-86b6-f177321bcfb2",
            "author-id": "Mary Major"
        },
        "state": "SUCCEEDED",
        "version": 32.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

管道 FAILED 事件

在管道执行失败时，它发出一个事件以发送包含以下内容的通知。此示例适用于 us-west-2 区域中名为 "myPipeline" 的管道。

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:55:43Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "FAILED",
        "version": 4.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Pipeline Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:46:16Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "FAILED",
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {
        "failedActionCount": 1,
        "failedActions": [
            {
                "action": "Deploy",
                "additionalInformation": "Deployment <ID> failed"
            }
        ],
        "failedStage": "Deploy"
    }
```

管道 FAILED（含 Git 标签的示例）

除非它在源阶段失败，否则对于配置了触发器的管道，它会发出一个事件，发送包含以下内容的通知。此示例适用于为 Git 标签配置了 `execution-trigger` 的 `eu-central-1` 区域中名为 `myPipeline` 的管道。

> **Note**
>
> `execution-trigger` 字段的值是 `tag-name` 或 `branch-name`，具体取决于触发管道的事件类型。

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:55:43Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "author-display-name": "Mary Major",
            "full-repository-name": "mmajor/sample-project",
            "provider-type": "GitLab",
            "author-email": "email_address",
            "commit-message": "Update file README.md",
            "author-date": "2023-08-16T21:08:08Z",
            "tag-name": "gitlab-v4.2.1",
            "commit-id": "commit_ID",
            "connection-arn": "arn:aws:codestar-connections:eu-
central-1:123456789012:connection/0f5b706a-1a1d-46c5-86b6-f177321bcfb2",
            "author-id": "Mary Major"
        },
        "state": "FAILED",
        "version": 4.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event


```
{
    "account": "123456789012",
    "detailType": "CodePipeline Pipeline Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:46:16Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "author-display-name": "Mary Major",
            "full-repository-name": "mmajor/sample-project",
            "provider-type": "GitLab",
            "author-email": "email_address",
            "commit-message": "Update file README.md",
            "author-date": "2023-08-16T21:08:08Z",
            "tag-name": "gitlab-v4.2.1",
            "commit-id": "commit_ID",
            "connection-arn": "arn:aws:codestar-connections:eu-
central-1:123456789012:connection/0f5b706a-1a1d-46c5-86b6-f177321bcfb2",
            "author-id": "Mary Major"
        },
        "state": "FAILED",
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {
        "failedActionCount": 1,
        "failedActions": [
            {
                "action": "Deploy",
                "additionalInformation": "Deployment <ID> failed"
            }
        ],
        "failedStage": "Deploy"
```

```
    }
```

阶段级事件

当阶段执行状态发生变化时，就会发出阶段级事件。

主题

阶段 STARTED 事件

在启动阶段执行时，它发出一个事件以发送包含以下内容的通知。此示例适用于 us-east-1 区域中名为 "myPipeline" 的管道的 Prod 阶段。

Emitted event

```
{
    "version": "0",
    "id": 01234567-EXAMPLE,
    "detail-type": "CodePipeline Stage Execution State Change",
    "source": "aws.codepipeline",
    "account": 123456789012,
    "time": "2020-01-24T22:03:07Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "version": 1.0,
        "execution-id": 12345678-1234-5678-abcd-12345678abcd,
        "start-time": "2023-10-26T13:49:39.208Z",
        "stage": "Prod",
        "state": "STARTED",
        "pipeline-execution-attempt": 1.0
    }
```

```
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Stage Execution State Change",
    "region": "us-east-1",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:45:40Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "stage": "Source",
        "state": "STARTED",
        "version": 1.0,
        "pipeline-execution-attempt": 0.0
    },
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "additionalAttributes": {
        "sourceActions": [
            {
                "sourceActionName": "Source",
                "sourceActionProvider": "CodeCommit",
                "sourceActionVariables": {
                    "BranchName": "main",
                    "CommitId": "<ID>",
                    "RepositoryName": "my-repo"
                }
            }
        ]
    }
}
```

阶段 STOPPING 事件

在停止阶段执行时，它发出一个事件以发送包含以下内容的通知。此示例适用于 us-west-2 区域中名为 myPipeline 的管道的 Deploy 阶段。

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Stage Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:02:20Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "stage": "Deploy",
        "state": "STOPPING",
        "version": 3.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

阶段 STOPPED 事件

在阶段执行已停止时，它发出一个事件以发送包含以下内容的通知。此示例适用于 us-west-2 区域中名为 myPipeline 的管道的 Deploy 阶段。

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Stage Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:21:39Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
```

```
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "stage": "Deploy",
        "state": "STOPPED",
        "version": 3.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

阶段重试后的阶段 RESUMED 事件

在阶段恢复执行而且有一个阶段已经重试时，它发出一个事件以发送包含以下内容的通知。

当阶段已重试时，将显示 `stage-last-retry-attempt-time` 字段，如本例所示。如果执行了重试，则该字段将显示在所有阶段事件上。

> **Note**
>
> 阶段重试后，`stage-last-retry-attempt-time` 字段将显示在所有后续阶段事件中。

```
{
    "version": "0",
    "id": "38656bcd-a798-5f92-c738-02a71be484e1",
    "detail-type": "CodePipeline Stage Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2023-10-26T14:14:56Z",
    "region": "eu-central-1",
    "resources": [
        "arn:aws:codepipeline:eu-central-1:123456789012:BuildFromTag"
    ],
    "detail": {
        "pipeline": "BuildFromTag",
        "execution-id": "05dafb6a-5a56-4951-a858-968795364846",
        "stage-last-retry-attempt-time": "2023-10-26T14:14:56.305Z",
        "stage": "Build",
        "state": "RESUMED",
        "version": 32.0,
        "pipeline-execution-attempt": 2.0
```

```
    }
}
```

操作级事件

当操作执行状态发生变化时，就会发出操作级事件。

主题

- 操作 STARTED 事件
- 操作 SUCCEEDED 事件
- 操作 FAILED 事件
- 操作 ABANDONED 事件

操作 STARTED 事件

在启动操作执行时，它发出一个事件以发送包含以下内容的通知。此示例适用于 us-east-1 区域中名为 myPipeline 的管道的部署操作 myAction。

Emitted event

```
{
    "version": "0",
    "id": 01234567-EXAMPLE,
    "detail-type": "CodePipeline Action Execution State Change",
    "source": "aws.codepipeline",
    "account": 123456789012,
    "time": "2020-01-24T22:03:07Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": 12345678-1234-5678-abcd-12345678abcd,
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Prod",
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "myAction",
        "state": "STARTED",
        "type": {
```

```
            "owner": "AWS",
            "category": "Deploy",
            "provider": "CodeDeploy",
            "version": "1"
        },
        "version": 2.0
        "pipeline-execution-attempt": 1.0
        "input-artifacts": [
            {
                "name": "SourceArtifact",
                "s3location": {
                    "bucket": "codepipeline-us-east-1-BUCKETEXAMPLE",
                    "key": "myPipeline/SourceArti/KEYEXAMPLE"
                }
            }
        ]
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Action Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:45:44Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Deploy",
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Deploy",
        "input-artifacts": [
            {
                "name": "SourceArtifact",
                "s3location": {
                    "bucket": "codepipeline-us-east-1-EXAMPLE",
                    "key": "myPipeline/SourceArti/EXAMPLE"
                }
```

```
            }
        ],
        "state": "STARTED",
        "region": "us-east-1",
        "type": {
            "owner": "AWS",
            "provider": "CodeDeploy",
            "category": "Deploy",
            "version": "1"
        },
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "additionalAttributes": {}
}
```

操作 SUCCEEDED 事件

在操作执行成功时，它发出一个事件以发送包含以下内容的通知。此示例适用于 us-west-2 区域中名为 "myPipeline" 的管道的源操作 "Source"。对于这种事件类型，有两个不同的 region 字段。事件 region 字段指定管道事件的区域。detail 部分下的 region 字段指定操作的区域。

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Action Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:03:11Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:51:09.981Z",
```

```
        "stage": "Source",
        "execution-result": {
            "external-execution-url": "https://us-west-2.console.aws.amazon.com/
codecommit/home#/repository/my-repo/commit/8cf40f2EXAMPLE",
            "external-execution-summary": "Added LICENSE.txt",
            "external-execution-id": "8cf40fEXAMPLE"
        },
        "output-artifacts": [
            {
                "name": "SourceArtifact",
                "s3location": {
                    "bucket": "codepipeline-us-west-2-BUCKETEXAMPLE",
                    "key": "myPipeline/SourceArti/KEYEXAMPLE"
                }
            }
        ],
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Source",
        "state": "SUCCEEDED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeCommit",
            "category": "Source",
            "version": "1"
        },
        "version": 3.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Action Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:45:44Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:ACCOUNT:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
```

```
        "execution-id": "arn:aws:codepipeline:us-west-2:123456789012:myPipeline",
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Source",
        "execution-result": {
            "external-execution-url": "https://us-west-2.console.aws.amazon.com/
codecommit/home#/repository/my-repo/commit/8cf40f2EXAMPLE",
            "external-execution-summary": "Edited index.html",
            "external-execution-id": "36ab3ab7EXAMPLE"
        },
        "output-artifacts": [
            {
                "name": "SourceArtifact",
                "s3location": {
                    "bucket": "codepipeline-us-west-2-EXAMPLE",
                    "key": "myPipeline/SourceArti/EXAMPLE"
                }
            }
        ],
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Source",
        "state": "SUCCEEDED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeCommit",
            "category": "Source",
            "version": "1"
        },
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {}
}
```

操作 FAILED 事件

在操作执行失败时，它发出一个事件以发送包含以下内容的通知。此示例适用于 us-west-2 区域中名为 "myPipeline" 的管道的 "Deploy" 操作。

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Action Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:55:43Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Deploy",
        "execution-result": {
            "external-execution-url": "https://us-west-2.console.aws.amazon.com/
codedeploy/home?#/deployments/<ID>",
            "external-execution-summary": "Deployment <ID> failed",
            "external-execution-id": "<ID>",
            "error-code": "JobFailed"
        },
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Deploy",
        "state": "FAILED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeDeploy",
            "category": "Deploy",
            "version": "1"
        },
        "version": 4.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
```

```
    "account": "123456789012",
    "detailType": "CodePipeline Action Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:46:16Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "stage": "Deploy",
        "execution-result": {
            "external-execution-url": "https://console.aws.amazon.com/codedeploy/
home?region=us-west-2#/deployments/<ID>",
            "external-execution-summary": "Deployment <ID> failed",
            "external-execution-id": "<ID>",
            "error-code": "JobFailed"
        },
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Deploy",
        "state": "FAILED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeDeploy",
            "category": "Deploy",
            "version": "1"
        },
        "version": 13.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {
        "additionalInformation": "Deployment <ID> failed"
    }
}
```

操作 ABANDONED 事件

在放弃操作执行时，它发出一个事件以发送包含以下内容的通知。此示例适用于 us-west-2 区域中名为 "myPipeline" 的管道的 "Deploy" 操作。

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Action Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:21:39Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "stage": "Deploy",
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Deploy",
        "state": "ABANDONED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeDeploy",
            "category": "Deploy",
            "version": "1"
        },
        "version": 3.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

创建发送管道事件通知的规则

规则监视某些事件，然后将其路由到您选择的 AWS 目标。您可以创建在发生其他 AWS 操作时自动执行 AWS 操作的规则，也可以创建按设定的时间表定期执行 AWS 操作的规则。

主题

- 当管道状态发生更改时发送通知（控制台）

- 当管道状态发生更改时发送通知 (CLI)

当管道状态发生更改时发送通知（控制台）

以下步骤说明如何使用 EventBridge 控制台创建规则，以发送更改通知 CodePipeline。

使用 Amazon S3 源创建针对您的管道的 EventBridge 规则

1. 打开亚马逊 EventBridge 控制台，网址为 https://console.aws.amazon.com/events/。
2. 在导航窗格中，选择规则。保留选中的默认总线或选择一个事件总线。选择创建规则。
3. 在名称中，输入规则的名称。
4. 对于规则类型，选择具有事件模式的规则。选择下一步。
5. 在事件模式下，选择 AWS 服务。
6. 从事件类型下拉列表中，选择通知的状态更改级别。
   - 要获取适用于管道级事件的规则，请选择CodePipeline管道执行状态更改。
   - 要获取适用于舞台级别事件的规则，请选择CodePipeline阶段执行状态更改。
   - 要获取适用于操作级事件的规则，请选择CodePipeline操作执行状态更改。
7. 指定规则适用的状态更改：
   - 对于适用于所有状态更改的规则，请选择任意状态。
   - 对于仅适用于某些状态更改的规则，请选择特定状态，然后从列表中选择一个或多个状态值。
8. 对于比选择器允许的级别更详细的事件模式，您也可以使用事件模式窗口中的编辑模式选项指定 JSON 格式的事件模式。

   > **Note**
   >
   > 如果未另行指定，则为所有管道/阶段/操作和状态创建事件模式。

   有关更详细的事件模式，您可以复制以下示例事件模式并粘贴到事件模式窗口中。

   - Example

     可以使用该示例事件模式在所有管道中捕获失败的部署和生成操作。

     ```
     {
     ```

```
"source": [
    "aws.codepipeline"
  ],
  "detail-type": [
    "CodePipeline Action Execution State Change"
  ],
  "detail": {
    "state": [
      "FAILED"
    ],
    "type": {
      "category": ["Deploy", "Build"]
    }
  }
}
```

- Example

  可以使用该示例事件模式在所有管道中捕获所有拒绝或失败的审批操作。

```
{
 "source": [
    "aws.codepipeline"
  ],
  "detail-type": [
    "CodePipeline Action Execution State Change"
  ],
  "detail": {
    "state": [
      "FAILED"
    ],
    "type": {
      "category": ["Approval"]
    }
  }
}
```

- Example

  可以使用该示例事件模式从指定的管道中捕获所有事件。

```
{
"source": [
```

```
    "aws.codepipeline"
  ],
  "detail-type": [
    "CodePipeline Pipeline Execution State Change",
    "CodePipeline Action Execution State Change",
    "CodePipeline Stage Execution State Change"
  ],
  "detail": {
    "pipeline": ["myPipeline", "my2ndPipeline"]
  }
}
```

9. 选择下一步。
10. 在目标类型中，选择 AWS 服务。
11. 在选择目标中，选择CodePipeline。在管道 ARN 中，输入该规则启动的管道的管道 ARN。

> **Note**
>
> 要获取管道 ARN，请运行 get-pipeline 命令。管道 ARN 将显示在输出中。它是使用以下格式构造的：
> arn:aws:codepipeline:*region*:*account*:*pipeline-name*
> 示例管道 ARN：
> arn: aws: codepipeline: us-east-2:80398 示例：MyFirstPipeline

12. 要创建或指定一个 IAM 服务角色来授予调用与您的 EventBridge 规则关联的目标的 EventBridge 权限（在本例中，目标是 CodePipeline），请执行以下操作：
    - 选择 “为此特定资源创建新角色” 以创建服务角色，该角色 EventBridge 授予您启动管道执行的权限。
    - 选择 “使用现有角色” 输入一个服务角色，该角色 EventBridge 授予您启动管道执行的权限。
13. 选择下一步。
14. 在标签页面上，选择下一步。
15. 在查看和创建页面上，检查规则配置。如果您对规则满意，请选择 Create rule。

当管道状态发生更改时发送通知 (CLI)

以下步骤说明如何使用 CLI 创建 CloudWatch 事件规则以发送更改通知 CodePipeline。

要使用创建规则，请调用put-rule命令，指定： AWS CLI

- 唯一地标识创建的规则的名称。在您创建的与 AWS 账户 CodePipeline 关联的所有管道中，此名称必须是唯一的。
- 规则使用的源事件模式和详细信息字段。有关更多信息，请参阅 Amazon EventBridge 和事件模式。

创建以事件源 CodePipeline 为对象的 EventBridge 规则

1. 调用 put-rule 命令以创建一个规则，从而指定事件模式。(请参阅上表以了解有效的状态。)

   以下示例命令用于创建名--event-pattern为的规则“MyPipelineStateChanges”，当名为“myPipeline” 的管道执行管道失败时，该规则会发出 CloudWatch 事件。

   ```
   aws events put-rule --name "MyPipelineStateChanges" --event-pattern "{\"source\":
   [\"aws.codepipeline\"],\"detail-type\":[\"CodePipeline Pipeline Execution State
    Change\"],\"detail\":{\"pipeline\":[\"myPipeline\"],\"state\":[\"FAILED\"]}}"
   ```

2. 调用 put-targets 命令并包含以下参数：

   - `--rule` 参数与您使用 put-rule 创建的 `rule_name` 结合使用。
   - `--targets` 参数与目标列表中该目标的列表 `Id` 以及 Amazon SNS 主题的 ARN 结合使用。

   以下示例命令为名为 `MyPipelineStateChanges` 的规则指定此内容，目标 `Id` 由数字 1 组成，这指示此内容位于规则的目标列表中，而这是目标 1。示例命令还为 Amazon SNS 主题指定一个示例 ARN。

   ```
   aws events put-targets --rule MyPipelineStateChanges --targets
    Id=1,Arn=arn:aws:sns:us-west-2:11111EXAMPLE:MyNotificationTopic
   ```

3. EventBridge 添加使用指定目标服务调用通知的权限。有关更多信息，请参阅使用适用于 Amazon EventBridge 的基于资源的政策。

事件占位存储桶参考

本部分仅供参考。有关使用事件检测资源创建管道的信息，请参阅 源操作和更改检测方法。

源操作由 Amazon S3 提供，并 CodeCommit 使用基于事件的更改检测资源在源存储桶或存储库中进行更改时触发您的管道。这些资源是 CloudWatch 事件规则，配置为响应管道源中的事件，例如对 CodeCommit 存储库的代码更改。当您对 Amazon S3 源使用 CloudWatch 事件时，必须开启才能记录

事件。 CloudTrail CloudTrail 需要一个 S3 存储桶，它可以在其中发送摘要。您可以从自定义存储桶访问 CloudWatch 事件资源的日志文件，但不能访问占位符存储桶中的数据。

- 如果您使用 CLI 或 AWS CloudFormation 设置了 CloudWatch 事件资源，则可以在设置管道时指定的存储桶中找到您的 CloudTrail 文件。
- 如果您使用控制台通过 S3 源设置管道，则控制台在为您创建 CloudWatch 事件资源时会使用 CloudTrail 占位符存储桶。 CloudTrail 摘要存储在创建管道的占位符存储桶 AWS 区域 中。

如果要使用除占位存储桶以外的存储桶，可以更改配置。

> **Note**
>
> 写入 CloudTrail 占位符存储桶的数据将在一天后自动过期，并且不会保留。

有关查找和管理 CloudTrail 日志文件的更多信息，请参阅获取和查看您的 CloudTrail 日志文件。

主题

按区域列出的事件占位存储桶名称

此表列出了 S3 占位符桶的名称，这些桶包含日志文件，可跟踪具有 Amazon S3 源操作的管道的更改检测事件。

区域名称	占位存储桶名称	区域标识符
美国东部（俄亥俄州）	codepipeline-cloudtrail-placeholder-bucket-us-east-2	us-east-2
美国东部（弗吉尼亚州北部）	codepipeline-cloudtrail-placeholder-bucket-us-east-1	us-east-1
美国西部（北加利福尼亚）	codepipeline-cloudtrail-placeholder-bucket-us-west-1	us-west-1
US West（Oregon）	codepipeline-cloudtrail-placeholder-bucket-us-west-2	us-west-2

区域名称	占位存储桶名称	区域标识符
加拿大（中部）	codepipeline-cloudtrail-placeholder-bucket-ca-central-1	ca-central-1
欧洲地区（法兰克福）	codepipeline-cloudtrail-placeholder-bucket-eu-central-1	eu-central-1
Europe（Ireland）	codepipeline-cloudtrail-placeholder-bucket-eu-west-1	eu-west-1
欧洲地区（伦敦）	codepipeline-cloudtrail-placeholder-bucket-eu-west-2	eu-west-2
欧洲地区（巴黎）	codepipeline-cloudtrail-placeholder-bucket-eu-west-3	eu-west-3
欧洲地区（斯德哥尔摩）	codepipeline-cloudtrail-placeholder-bucket-eu-north-1	eu-north-1
亚太地区（香港）	codepipeline-cloudtrail-placeholder-bucket-ap-east-1	ap-east-1
亚太地区（海得拉巴）	codepipeline-cloudtrail-placeholder-bucket-ap-south-2	ap-south-2
亚太地区（雅加达）	codepipeline-cloudtrail-placeholder-bucket-ap-southeast-3	ap-southeast-3
亚太地区（墨尔本）	codepipeline-cloudtrail-placeholder-bucket-ap-southeast-4	ap-southeast-4
亚太地区（孟买）	codepipeline-cloudtrail-placeholder-bucket-ap-south-1	ap-south-1

区域名称	占位存储桶名称	区域标识符
亚太地区（大阪）	codepipeline-cloudtrail-placeholder-bucket-ap-northeast-3-prod	ap-northeast-3
亚太地区（东京)	codepipeline-cloudtrail-placeholder-bucket-ap-northeast-1	ap-northeast-1
亚太地区（首尔）	codepipeline-cloudtrail-placeholder-bucket-ap-northeast-2	ap-northeast-2
亚太地区（新加坡）	codepipeline-cloudtrail-placeholder-bucket-ap-southeast-1	ap-southeast-1
Asia Pacific（Sydney）	codepipeline-cloudtrail-placeholder-bucket-ap-southeast-2	ap-southeast-2
亚太地区（东京)	codepipeline-cloudtrail-placeholder-bucket-ap-northeast-1	ap-northeast-1
Canada（Central）	codepipeline-cloudtrail-placeholder-bucket-ca-central-1	ca-central-1
欧洲地区（法兰克福）	codepipeline-cloudtrail-placeholder-bucket-eu-central-1	eu-central-1
Europe（Ireland）	codepipeline-cloudtrail-placeholder-bucket-eu-west-1	eu-west-1
欧洲地区（伦敦）	codepipeline-cloudtrail-placeholder-bucket-eu-west-2	eu-west-2

区域名称	占位存储桶名称	区域标识符
欧洲地区（米兰）	codepipeline-cloudtrail-placeholder-bucket-eu-south-1	eu-south-1
欧洲地区（巴黎）	codepipeline-cloudtrail-placeholder-bucket-eu-west-3	eu-west-3
欧洲（西班牙）	codepipeline-cloudtrail-placeholder-bucket-eu-south-2	eu-south-2
欧洲地区（斯德哥尔摩）	codepipeline-cloudtrail-placeholder-bucket-eu-north-1	eu-north-1
欧洲（苏黎世）*	codepipeline-cloudtrail-placeholder-bucket-eu-central-2	eu-central-2
以色列（特拉维夫）	codepipeline-cloudtrail-placeholder-bucket-il-central-1	il-central-1
中东（巴林）*	codepipeline-cloudtrail-placeholder-bucket-我-south-1	me-south-1
中东（阿联酋）	codepipeline-cloudtrail-placeholder-bucket-me-central-1	me-central-1
南美洲（圣保罗）	codepipeline-cloudtrail-placeholder-bucket-sa-east-1	sa-east-1

使用记录 CodePipeline API 调用 AWS CloudTrail

AWS CodePipeline 与 AWS CloudTrail一项服务集成，该服务提供用户、角色或 AWS 服务 中的用户所采取的操作的记录 CodePipeline；。 CloudTrail 将所有 API 调用捕获 CodePipeline 为事件。捕获的调用包括来自 CodePipeline 控制台的调用和对 CodePipeline API 操作的代码调用。如果您创建了跟踪，则可以允许将 CloudTrail 事件持续传输到 Amazon S3 存储桶，包括的事件 CodePipeline。如果您未配置跟踪，您仍然可以在 CloudTrail 控制台的 “事件历史记录” 中查看最新的事件。使用收集的信息 CloudTrail，您可以确定向哪个请求发出 CodePipeline、发出请求的 IP 地址、谁发出了请求、何时发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》。

CodePipeline 信息在 CloudTrail

CloudTrail 在您创建账户 AWS 账户 时已在您的账户上启用。当活动发生在中时 CodePipeline，该活动会与其他 CloudTrail 事件一起记录在 AWS 服务 事件历史记录中。您可以在自己的 AWS 账户中查看、搜索和下载最近发生的事件。有关更多信息，请参阅使用事件历史记录查看 CloudTrail 事件。

要持续记录您的 AWS 账户 事件（包括的事件） CodePipeline，请创建跟踪。跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。默认情况下，当您在控制台中创建跟踪时，该跟踪将应用于所有 AWS 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 Amazon S3 存储桶。此外，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，请参阅下列内容：

- 创建跟踪概述
- CloudTrail 支持的服务和集成
- 配置 Amazon 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 CodePipeline 操作均由《API 参考》记录 CloudTrail 并记录在《CodePipeline API 参考》中。例如，调用`GetPipelineExecution`和`UpdatePipeline`操作会在 CloudTrail 日志文件中生成条目。`CreatePipeline`

每个事件或日记账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：

- 请求是使用根证书还是 AWS Identity and Access Management (IAM) 凭证发出。
- 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 用户身份元素。

了解 CodePipeline 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输到您指定的 Amazon 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，因此它们不会按任何特定的顺序出现。

以下示例显示了更新管道事件的 CloudTrail 日志条目，在该事件中，名为的管道 MyFirstPipeline 已由名为 JaneDoe-CodePipeline 的用户编辑，账号为 80398EXAMPLE。该用户将管道的源阶段名称从 Source 更改为 MySourceStage。由于 CloudTrail日志中的requestParameters和responseElements元素都包含已编辑管道的整个结构，因此在以下示例中对这些元素进行了缩写。已着重强调发生更改的管道 requestParameters 部分、管道的先前版本号和 responseElements 部分（显示按 1 递增的版本号）。编辑的部分用省略号 (...) 标记，以说明实际日志条目中将会显示更多数据。

```
{
 "eventVersion":"1.03",
  "userIdentity": {
   "type":"IAMUser",
   "principalId":"AKIAI44QH8DHBEXAMPLE",
   "arn":"arn:aws:iam::80398EXAMPLE:user/JaneDoe-CodePipeline",
   "accountId":"80398EXAMPLE",
   "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
   "userName":"JaneDoe-CodePipeline",
   "sessionContext": {
   "attributes":{
    "mfaAuthenticated":"false",
    "creationDate":"2015-06-17T14:44:03Z"
    }
   },
 "invokedBy":"signin.amazonaws.com"},
 "eventTime":"2015-06-17T19:12:20Z",
 "eventSource":"codepipeline.amazonaws.com",
 "eventName":"UpdatePipeline",
 "awsRegion":"us-east-2",
 "sourceIPAddress":"192.0.2.64",
 "userAgent":"signin.amazonaws.com",
 "requestParameters":{
   "pipeline":{
   "version":1,
   "roleArn":"arn:aws:iam::80398EXAMPLE:role/CodePipeline_Service_Role",
   "name":"MyFirstPipeline",
   "stages":[
     {
     "actions":[
      {
          "name":"MySourceStage",
        "actionType":{
        "owner":"AWS",
```

```
        "version":"1",
        "category":"Source",
        "provider":"S3"
     },
    "inputArtifacts":[],
    "outputArtifacts":[
     {"name":"MyApp"}
     ],
    "runOrder":1,
    "configuration":{
     "S3Bucket":"awscodepipeline-demobucket-example-date",
     "S3ObjectKey":"sampleapp_linux.zip"
      }
      }
     ],
       "name":"Source"
     },
     (...)
                },
 "responseElements":{
   "pipeline":{
     "version":2,
     (...)
           },
     "requestID":"2c4af5c9-7ce8-EXAMPLE",
     "eventID":""c53dbd42-This-Is-An-Example"",
     "eventType":"AwsApiCall",
     "recipientAccountId":"80398EXAMPLE"
     }
      ]
}
```

故障排除 CodePipeline

以下信息可帮助您处理 AWS CodePipeline中的常见问题。

主题

- 当达到文件限制时，使用按文件路径进行触发器筛选的连接的管道可能无法启动
- CodeCommit 或者并行模式下的 S3 源版本可能与 EventBridge 事件不匹配
- 需要有关其他问题的帮助？

管道错误：使用 AWS Elastic Beanstalk 配置的管道返回错误消息："部署失败。提供的角色没有足够的权限：服务：AmazonElasticLoadBalancing"

问题：的服务角色对于 Elastic Load Balancing 中的某些操作 CodePipeline 没有足够的权限 AWS Elastic Beanstalk，包括但不限于 Elastic Load Balancing 中的某些操作。的服务角色已 CodePipeline 于 2015 年 8 月 6 日更新，以解决此问题。在此日期之前创建其服务角色的客户必须修改其服务角色的策略语句以添加所需的权限。

可能的修复措施：最简单的解决方案是编辑服务角色的策略声明，详情请参见向 CodePipeline 服务角色添加权限。

应用编辑过的策略后，按照手动启动管道中的步骤手动重新运行使用 Elastic Beanstalk 的任何管道。

根据您的安全需求，您也可以通过其他方式修改权限。

部署错误：如果缺少 "DescribeEvents" 权限，则配置了 AWS Elastic Beanstalk 部署操作的管道会挂起而不是失败

问题：的服务角色 CodePipeline 必须包括使用的所有管道的`"elasticbeanstalk:DescribeEvents"`操作 AWS Elastic Beanstalk。如果没有此权限，AWS Elastic Beanstalk 部署操作将挂起而不会失败或显示错误。如果您的服务角色中缺少此操作，CodePipeline 则无权 AWS Elastic Beanstalk 代表您运行管道部署阶段。

可能的修复方法：查看您的 CodePipeline 服务角色。如果缺少`"elasticbeanstalk:DescribeEvents"` 操作，请使用向 CodePipeline 服务角色添加权限中的步骤在 IAM 控制台中使用编辑策略功能添加它。

应用编辑过的策略后，按照手动启动管道中的步骤手动重新运行使用 Elastic Beanstalk 的任何管道。

管道错误：源操作返回权限不足消息：“无法访问 CodeCommit 存储库**repository-name**。确保管道 IAM 角色具有足够的权限来访问存储库。”

问题：的服务角色 CodePipeline 没有足够的权限，很可能是在 2016 年 4 月 18 日添加对使用 CodeCommit存储库的支持之前创建的。 CodeCommit 在此日期之前创建其服务角色的客户必须修改其服务角色的策略语句以添加所需的权限。

可能的修复方法：将所需的权限 CodeCommit 添加到您的 CodePipeline 服务角色的策略中。有关更多信息，请参阅 向 CodePipeline 服务角色添加权限。

管道错误：Jenkins 生成或测试操作运行很长时间后由于缺少凭证或权限而失败

问题：如果 Jenkins 服务器安装在 Amazon EC2 实例上，则该实例可能不是使用具有所需权限的实例角色创建的 CodePipeline。如果您在 Jenkins 服务器、本地实例或不是使用所需 IAM 角色创建的 Amazon EC2 实例上使用 IAM 用户，则 IAM 用户将不具有所需权限，或者 Jenkins 服务器将无法通过服务器上配置的配置文件访问这些凭证。

可能的修复措施：确保为 Amazon EC2 实例角色或 IAM 用户配置 `AWSCodePipelineCustomActionAccess` 托管策略或等效权限。有关更多信息，请参阅 AWS 的托管策略 AWS CodePipeline。

如果您使用的是 IAM 用户，请确保在实例上 AWS 配置的配置文件使用配置了正确权限的 IAM 用户。您可能需要提供您为在 Jenkins 之间集成和 CodePipeline 直接集成到 Jenkins 用户界面而配置的 IAM 用户证书。这不是推荐的最佳实践。如果必须这样做，请确保 Jenkins 服务器是安全的，并且使用 HTTPS 而不是 HTTP。

管道错误：使用在另一个 AWSAWS 区域创建的存储桶在一个区域创建的管道会返回一个 InternalError “”，代码为 “JobFailed”

问题：如果在不同的 AWS 区域创建管道和存储桶，则存储在 Amazon S3 存储桶中的项目下载将失败。

可能的修复方法：确保存储您的项目的 Amazon S3 存储桶与您创建的管道位于同一 AWS 区域。

部署错误：包含 WAR 文件的 ZIP 文件已成功部署到 AWS Elastic Beanstalk，但应用程序 URL 报告了 404 未找到错误

问题：WAR 文件已成功部署到 AWS Elastic Beanstalk 环境，但应用程序 URL 返回 404 Not Found 错误。

可能的修复方法： AWS Elastic Beanstalk 可以解压 ZIP 文件，但不能解压包含在 ZIP 文件中的 WAR 文件。不要在 `buildspec.yml` 文件中指定 WAR 文件，而是指定一个包含要部署的内容的文件夹。例如：

```
version: 0.2

phases:
  post_build:
    commands:
      - mvn package
      - mv target/my-web-app ./
artifacts:
  files:
    - my-web-app/**/*
 discard-paths: yes
```

有关示例，请参阅适用于 CodeBuild 的AWS Elastic Beanstalk 示例。

管道构件文件夹名称似乎被截断

问题：在中查看管道工件名称时 CodePipeline，名称似乎已被截断。这可能会使名称看似或似乎不再包含整个管道名称。

解释：在为作业人员 CodePipeline 生成临时证书时，会 CodePipeline 截断对象名称以确保完整的 Amazon S3 路径不会超过策略大小限制。

尽管工件名称似乎已被截断，但 CodePipeline 映射到工件存储桶的方式不会受到名称被截断的工件的影响。管道可以正常工作。这不是文件夹或构件的问题。管道名称不能超过 100 个字符。尽管构件文件夹名称似乎已缩短，但它对您的管道来说仍然是唯一的。

添加连接 Bitbucket、 GitHub、En GitHub terprise Server 或 GitLab .com 的 CodeBuild GitClone 权限

当您在源操作和操作中使用 AWS CodeStar 连接时，可以通过两种方式将输入构件传递给构建：CodeBuild

- 默认：源操作生成一个 zip 文件，其中包含要 CodeBuild下载的代码。
- Git 克隆：源代码可以直接下载到构建环境中。

  Git 克隆模式允许您将源代码作为工作 Git 存储库进行交互。要使用此模式，必须向您的 CodeBuild 环境授予使用连接的权限。

要向您的 CodeBuild 服务角色策略添加权限，您需要创建一个附加到您的 CodeBuild 服务角色的客户托管策略。以下步骤会创建一个策略，其中 `UseConnection` 权限在 `action` 字段中指定，而连接 ARN 在 `Resource` 字段中指定。

使用控制台添加 UseConnection 权限

1. 要查找管道的连接 ARN，请打开管道，然后在源操作中点击 (i) 图标。您将连接 ARN 添加到您的 CodeBuild 服务角色策略中。

   连接 ARN 的示例是：

   ```
   arn:aws:codeconnections:eu-central-1:123456789123:connection/
   sample-1908-4932-9ecc-2ddacee15095
   ```

2. 要查找您的 CodeBuild 服务角色，请打开管道中使用的构建项目，然后导航到构建详细信息选项卡。
3. 选择服务角色链接。此时将打开 IAM 控制台，您可以在其中添加新策略，以授予对连接的访问权限。
4. 在 IAM 控制台中，选择 Attach policies (附加策略)，然后选择 Create policy (创建策略)。

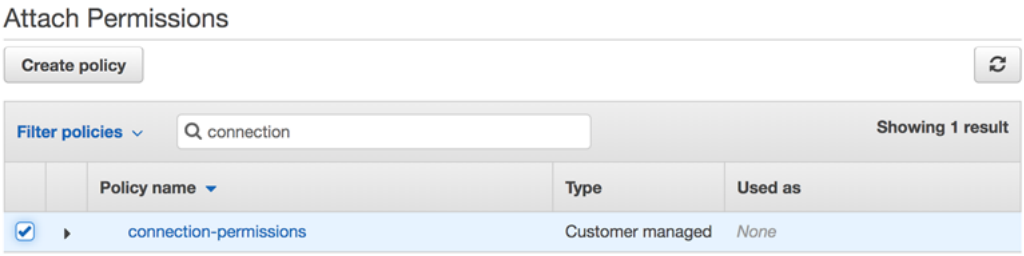
   使用以下示例策略模板。将您的连接 ARN 添加到 `Resource` 字段，如此示例中所示：

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

```
            "Effect": "Allow",
            "Action": "codestar-connections:UseConnection",
            "Resource": "insert connection ARN here"
        }
    ]
}
```

在存储库的 JSON 选项卡上，粘贴您的策略。

5. 选择查看策略。为策略输入名称（例如 **connection-permissions**），然后选择 Create policy (创建策略)。
6. 返回到附加权限的页面上，刷新策略列表，然后选择您刚才创建的策略。选择附加策略。



为 CodeCommit源操作添加 CodeBuild GitClone 权限

当你的管道有 CodeCommit 源代码操作时，有两种方法可以将输入构件传递给构建：

- 默认-源操作生成一个 zip 文件，其中包含要 CodeBuild 下载的代码。
- 完整克隆 – 源代码可以直接下载到构建环境中。

  完整克隆选项允许您将源代码作为工作 Git 存储库进行交互。要使用此模式，必须为 CodeBuild环境添加从存储库中提取数据的权限。

要向您的 CodeBuild 服务角色策略添加权限，您需要创建一个附加到您的 CodeBuild 服务角色的客户托管策略。以下步骤将创建一个在 `action` 字段中指定 `codecommit:GitPull` 权限的策略。

使用控制台添加 GitPull 权限

1. 要查找您的 CodeBuild 服务角色，请打开管道中使用的构建项目，然后导航到构建详细信息选项卡。

2. 选择服务角色链接。此时将打开 IAM 控制台，您可以在其中添加新策略，以授予对存储库的访问权限。
3. 在 IAM 控制台中，选择 Attach policies (附加策略)，然后选择 Create policy (创建策略)。
4. 在 JSON 选项卡上，粘贴以下示例策略。

```
{
    "Action": [
        "codecommit:GitPull"
    ],
    "Resource": "*",
    "Effect": "Allow"
},
```

5. 选择查看策略。为策略输入名称（例如 **codecommit-gitpull**），然后选择 Create policy (创建策略)。
6. 返回到附加权限的页面上，刷新策略列表，然后选择您刚才创建的策略。选择附加策略。

管道错误：使用 CodeDeployTo ECS 操作的部署会返回一条错误消息：“尝试从以下位置读取任务定义构件文件时出现异常：<source artifact name>”

问题：

任务定义文件是通过 CodeDeploy （操作） CodePipeline 部署到 Amazon ECS 的CodeDeployToECS操作所必需的对象。CodeDeployToECS 部署操作中的构件 ZIP 大小上限为 3 MB。如果找不到文件或构件大小超过 3 MB，则会返回以下错误消息：

尝试从以下位置读取任务定义构件文件时出现异常：<source artifact name>

可能的修复措施：确保包含任务定义文件构件。如果文件已经存在，请确保压缩后的大小小于 3 MB。

GitHub 版本 1 源操作：存储库列表显示不同的存储库

问题：

在 CodePipeline 控制台中成功授权 GitHub 版本 1 操作后，您可以从 GitHub 存储库列表中进行选择。如果列表中不包括您期望看到的存储库，则可以对用于授权的账户进行故障排除。

可能的修复方法： CodePipeline 控制台中提供的存储库列表基于授权账户所属的 GitHub 组织。验证您用于授权的账户是否 GitHub 是与创建仓库的 GitHub 组织关联的账户。

GitHub 版本 2 源操作：无法完成存储库的连接

问题：

由于与 GitHub 存储库的连接使用 AWS 连接器 GitHub，因此您需要组织所有者权限或仓库管理员权限才能创建连接。

可能的修复方法：有关 GitHub 存储库权限级别的信息，请参阅 https://docs.github.com/en/ free-pro-team @latest /github/ setting-up-and-managing-/-organizations-and-teams organization。permission-levels-for-an

Amazon S3 错误： CodePipeline 服务角色<ARN>对 S3 存储桶的 S3 访问被拒绝 < BucketName >

问题：

在进行过程中，中的 CodeCommit 操作 CodePipeline 会检查管道工件存储桶是否存在。如果操作无权检查，则 Amazon S3 中会出现`AccessDenied`错误，并在中显示以下错误消息 CodePipeline：

CodePipeline *#### “arn: aws: iam:: ac countID: role/service-role/roleId” #### # S3 ### “” BucketName*

该操作的 CloudTrail 日志也会记录`AccessDenied`错误。

可能的修复措施：执行以下操作：

- 对于附加到您的 CodePipeline 服务角色的策略，`s3:ListBucket`请添加到策略中的操作列表中。有关查看您的服务角色策略的说明，请参阅查看管道 ARN 和服务角色 ARN（控制台）。编辑您的服务角色的策略声明，详情请参见向 CodePipeline 服务角色添加权限。
- 对于附加到管道的 Amazon S3 项目存储桶的基于资源的策略（也称为项目存储桶策略），请添加一条语句以允许您的 CodePipeline 服务角色使用该`s3:ListBucket`权限。

  将您的策略添加到构件桶

  1. 按照查看管道 ARN 和服务角色 ARN（控制台）中的步骤在管道设置页面上选择您的构件桶，然后在 Amazon S3 控制台中进行查看。
  2. 选择权限。

3. 在 Bucket policy (存储桶策略) 下，请选择 Edit (编辑)。
4. 在策略文本字段中，输入新的桶策略，或编辑现有策略，如以下示例所示。桶策略是一个 JSON 文件，因此您必须输入有效的 JSON。

   以下示例显示了构件桶的桶策略声明，其中服务角色的示例角色 ID 为 *AROAEXAMPLEID*。

```
{
    "Effect": "Allow",
    "Principal": "*",
    "Action": "s3:ListBucket",
    "Resource": "arn:aws:s3:::BucketName",
    "Condition": {
        "StringLike": {
            "aws:userid": "AROAEXAMPLEID:*"
        }
    }
}
```

   以下示例显示了添加权限后的相同桶策略声明。

```
{
    "Version": "2012-10-17",
    "Id": "SSEAndSSLPolicy",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890",
            "Condition": {
                "StringLike": {
                    "aws:userid": "AROAEXAMPLEID:*"
                }
            }
        },
        {
            "Sid": "DenyUnEncryptedObjectUploads",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
            "Condition": {
```

```
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "aws:kms"
                }
            }
        },
        {
            "Sid": "DenyInsecureConnections",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": false
                }
            }
        }
    ]
}
```

有关更多信息，请参阅https://aws.amazon.com/blogs/security/writing-iam-policies-how-to-grant-access-to-an-amazon-s3-bucket/ 中的步骤。

5. 选择保存。

应用编辑过的策略后，按照手动启动管道中的步骤手动重新运行管道。

带有 Amazon S3、Amazon ECR 或 CodeCommit源的管道不再自动启动

问题：

更改使用事件规则（EventBridge或 CloudWatch 事件）进行更改检测的操作的配置设置后，控制台可能无法检测到源触发器标识符相似且初始字符相同的更改。由于新的事件规则不是由控制台创建的，因此管道不再自动启动。

的参数名称末尾稍作更改的一个例子是 CodeCommit 将您的 CodeCommit 分支名称更改`MyTestBranch-1`为`MyTestBranch-2`。由于更改位于分支名称的末尾，因此源操作的事件规则可能不会为新的源设置更新或创建规则。

这适用于使用 CWE 事件进行更改检测的源操作，如下所示：

源操作	参数/触发器标识符（控制台）
Amazon ECR	存储库名称 映像标签
Amazon S3	存储桶 S3 对象键
CodeCommit	存储库名称 分支名称

可能的修复措施：

请执行以下操作之一：

- 更改 CodeCommit /S3/ECR 配置设置，以便更改参数值的起始部分。

  示例：将分支名称 `release-branch` 更改为 `2nd-release-branch`。避免在名称末尾进行更改，例如 `release-branch-2`。
- 更改每个管 CodeCommit道的 /S3/ECR 配置设置。

  示例：将分支名称 `myRepo/myBranch` 更改为 `myDeployRepo/myDeployBranch`。避免在名称末尾进行更改，例如 `myRepo/myBranch2`。
- 不使用控制台，而是使用 CLI 或 AWS CloudFormation 来创建和更新您的变更检测事件规则。有关为 S3 源操作创建事件规则的说明，请参阅亚马逊 S3 源代码操作 EventBridge 以及 AWS CloudTrail。有关为 Amazon ECR 操作创建事件规则的说明，请参阅 Amazon ECR 源操作和 EventBridge 资源。有关为操作创建事件规则的 CodeCommit 说明，请参阅 CodeCommit 源操作和 EventBridge。

在控制台中编辑操作配置后，接受控制台创建的已更新的更改检测资源。

连接时出现连接错误 GitHub：“出现问题，请确保您的浏览器已启用 Cookie” 或 “组织所有者必须安装 GitHub 应用程序”

问题：

要在中为 GitHub 源操作创建连接 CodePipeline，您必须是 GitHub组织所有者。对于不属于组织的存储库，您必须是存储库拥有者。当连接由非组织拥有者的其他用户创建时，将针对组织拥有者创建请求，并显示以下错误之一：

出现问题，请确保在浏览器中启用 Cookie

或

组织所有者必须安装该 GitHub 应用程序

可能的修复方法：对于 GitHub组织中的仓库，组织所有者必须创建与 GitHub 存储库的连接。对于不属于组织的存储库，您必须是存储库拥有者。

当达到运行限制时，执行模式更改为 QUEUED 或 PARLEL 模式的管道会失败

问题：排队模式下管道的最大并发执行数为 50 个。当达到此限制时，管道将失败，且不显示状态消息。

可能的修复：在编辑执行模式的管道定义时，请将编辑与其他编辑操作分开进行。

有关排队或并行执行模式的更多信息，请参阅CodePipeline 概念 。

如果在更改为 QUEUED 或 SUPERSEDED 模式时进行编辑，则处于并行模式的管道定义会过时

问题：对于并行模式下的管道，当将管道执行模式编辑为 QUEUED 或 SUPERSEDED 时，不会更新并行模式的管道定义。更新并行模式时更新的管道定义不在 SUPERSEDED 或 QUEUED 模式下使用

可能的修复：对于并行模式下的管道，在将管道执行模式编辑为 QUEUED 或 SUPERSEDED 时，请避免同时更新管道定义。

有关排队或并行执行模式的更多信息，请参阅CodePipeline 概念 。

从 PARALLEL 模式更改的管道将显示之前的执行模式

问题：对于处于并行模式的管道，当将管道执行模式编辑为 QUEUED 或 SUPERSEDED 时，管道状态不会将更新的状态显示为 PARALLEL。如果管道从 PARALLEL 更改为 QUEUED 或 SUPERSEDED，则处于取代或队列模式的管道状态将是这两种模式下的最后一个已知状态。如果管道以前从未在该模式下运行，则状态将为空。

可能的修复：对于并行模式下的管道，当将管道执行模式编辑为 QUEUED 或 SUPERSEDED 时，请注意，执行模式显示不会显示 PARALLEL 状态。

有关排队或并行执行模式的更多信息，请参阅CodePipeline 概念 。

使用按文件路径进行触发器筛选的连接的管道可能不会在创建分支时启动

描述：对于具有使用连接的源操作的管道（例如 BitBucket 源操作），您可以使用 Git 配置设置触发器，允许您按文件路径筛选以启动管道。在某些情况下，对于具有按文件路径筛选的触发器的管道，当首次创建带有文件路径筛选器的分支时，管道可能无法启动，因为这不允许 CodeConnections 连接解析已更改的文件。当触发器的 Git 配置设置为筛选文件路径时，当在源存储库中刚刚创建了带有筛选器的分支时，管道将不会启动。有关筛选文件路径的更多信息，请参阅筛选代码推送或拉取请求的触发器。

结果：例如 CodePipeline ，在分支 “B” 上具有文件路径筛选器的管道在创建分支 “B” 时不会被触发。如果没有文件路径筛选器，管道仍将启动。

当达到文件限制时，使用按文件路径进行触发器筛选的连接的管道可能无法启动

描述：对于具有使用连接的源操作的管道（例如 BitBucket 源操作），您可以使用 Git 配置设置触发器，允许您按文件路径筛选以启动管道。 CodePipeline 最多检索前 100 个文件；因此，当触发器的 Git 配置设置为筛选文件路径时，如果文件超过 100 个，管道可能无法启动。有关筛选文件路径的更多信息，请参阅筛选代码推送或拉取请求的触发器。

结果：例如，如果差异包含 150 个文件，则 CodePipeline查看前 100 个文件（排名不分先后），根据指定的文件路径筛选器进行检查。如果与文件路径筛选器匹配的文件不在检索的 100 个文件中 CodePipeline，则不会调用管道。

CodeCommit 或者并行模式下的 S3 源版本可能与 EventBridge 事件不匹配

描述：对于并行模式下的管道执行，执行可能从最近的更改（例如 CodeCommit 存储库提交）开始，该更改可能与 EventBridge 事件的更改不同。在某些情况下，在启动管道的提交或图像标签之间可能有一瞬间，当 CodePipeline收到事件并开始执行时，另一个提交或图像标签已被推送 CodePipeline （例如， CodeCommit 操作）将在那时克隆 HEAD 提交。

结果：对于处于并行模式且源为 CodeCommit S3 的管道，无论触发管道执行的更改如何，源操作都将始终在启动时克隆 HEAD。例如，对于处于 PARALLEL 模式的管道，将推送提交，这将启动管道以执行 1，而第二次管道执行使用第二次提交。

需要有关其他问题的帮助？

尝试其他资源：

- 请联系 AWS Support。
- 在CodePipeline论坛中提问。
- 请求提高限额。有关更多信息，请参阅 中的配额 AWS CodePipeline。

> **Note**
>
> 最长可能需要两周的时间来处理提高限额的请求。

安全性 AWS CodePipeline

云安全 AWS 是重中之重。作为 AWS 客户，您可以从专为满足最安全敏感的组织的要求而构建的数据中心和网络架构中受益。

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 AWS 服务 中运行的基础架构 AWS Cloud。 AWS 还为您提供可以安全使用的服务。作为 AWS 合规性计划的一部分，第三方审核人员将定期测试和验证安全性的有效性。要了解适用于的合规计划 AWS CodePipeline，请参阅AWS 服务 按合规计划划分的范围。
- 云端安全 — 您的责任由您 AWS 服务 使用的内容决定。您还需要对其他因素负责，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 CodePipeline。以下主题向您介绍如何进行配置 CodePipeline 以满足您的安全和合规性目标。您还将学习如何使用其他方法 AWS 服务 来帮助您监控和保护您的 CodePipeline 资源。

主题

中的数据保护 AWS CodePipeline

分 AWS 担责任模型适用于中的数据保护 AWS CodePipeline。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设施上的内容的控制。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。有关欧洲数据保护的信息，请参阅 AWS 安全性博客 上的 AWS 责任共担模式和 GDPR 博客文章。

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。我们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)。
- 使用 SSL/TLS 与资源通信。 AWS 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 AWS CloudTrail。
- 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（例如 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 140-2 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 140-2 版》。

我们强烈建议您切勿将机密信息或敏感信息（如您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 CodePipeline 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网址，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。

以下安全最佳实践也涉及以下方面的数据保护 CodePipeline：

- 为存储在 Amazon S3 中的项目配置服务器端加密 CodePipeline
- AWS Secrets Manager 用于跟踪数据库密码或第三方 API 密钥

互联网络流量隐私保护

Amazon VPC 可用于在您定义的虚拟网络（虚拟私有云）中启动 AWS 资源。 AWS 服务 CodePipeline支持由 AWS PrivateLink提供支持的 Amazon VPC 终端节点，该 AWS 技术可促进 AWS 服务 使用弹性网络接口与私有 IP 地址之间的私密通信。这意味着您可以 CodePipeline 通过 VPC 中的私有终端节点直接连接，从而将所有流量保留在您的 VPC 和 AWS 网络内。以前，在 VPC 内运行的应用程序需要互联网访问才能连接 CodePipeline。利用 VPC，您可以控制网络设置，例如：

- IP 地址范围
- 子网
- 路由表

- 网络网关。

要将您的 VPC 连接到 CodePipeline，您需要为定义接口 VPC 终端节点 CodePipeline。这类端点使您能够将自己的 VPC 连接到 AWS 服务。该端点 CodePipeline 无需互联网网关、网络地址转换 (NAT) 实例或 VPN 连接即可提供可靠、可扩展的连接。有关设置 VPC 的信息，请参阅 VPC 用户指南。

静态加密

中的 CodePipeline 数据使用进行静态加密 AWS KMS keys。代码项目存储在客户拥有的 S3 存储桶中，并使用 AWS 托管式密钥 或客户托管密钥进行加密。有关更多信息，请参阅 为存储在 Amazon S3 中的项目配置服务器端加密 CodePipeline。

传输中加密

所有 service-to-service 通信在传输过程中均使用 SSL/TLS 进行加密。

加密密钥管理

如果您选择加密代码工件的默认选项，则 CodePipeline 使用。 AWS 托管式密钥您无法更改或删除此内容 AWS 托管式密钥。如果您使用客户托管密钥 AWS KMS 来加密或解密 S3 存储桶中的项目，则可以根据需要更改或轮换此客户托管密钥。

> **Important**
>
> CodePipeline 仅支持对称 KMS 密钥。请勿使用非对称 KMS 密钥对 S3 桶中的数据进行加密。

为存储在 Amazon S3 中的项目配置服务器端加密 CodePipeline

可以使用两种方法为 Amazon S3 构件配置服务器端加密：

- CodePipeline 创建 S3 工件存储桶， AWS 托管式密钥 当您使用创建管道向导创建管道时会创建默认存储桶。与对象数据一起加密，并由管理 AWS。 AWS 托管式密钥
- 您可以创建和管理自己的客户管理的密钥。

> **Important**
>
> CodePipeline 仅支持对称 KMS 密钥。请勿使用非对称 KMS 密钥对 S3 桶中的数据进行加密。

如果使用默认的 S3 密钥，则无法更改或删除此 AWS 托管式密钥。如果您使用客户托管密钥 AWS KMS 来加密或解密 S3 存储桶中的项目，则可以根据需要更改或轮换此客户托管密钥。

Amazon S3 支持存储桶策略，如果您要对所有存储在存储桶中的对象执行服务器端加密，则可以使用这些策略。例如，如果请求不包含用于请求服务器端加密（SSE-KMS）的 `s3:PutObject` 标头，则下面的存储桶策略将拒绝所有人的上传对象（`x-amz-server-side-encryption`）权限。

```
{
    "Version": "2012-10-17",
    "Id": "SSEAndSSLPolicy",
    "Statement": [
        {
            "Sid": "DenyUnEncryptedObjectUploads",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::codepipeline-us-west-2-89050EXAMPLE/*",
            "Condition": {
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "aws:kms"
                }
            }
        },
        {
            "Sid": "DenyInsecureConnections",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::codepipeline-us-west-2-89050EXAMPLE/*",
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": "false"
                }
            }
        }
    ]
}
```

有关服务器端加密的更多信息 AWS KMS，请参阅使用服务器端加密保护数据和使用存储在 AWS Key Management Service (SSE- KMS) 中的 KMS 密钥使用服务器端加密保护数据。

有关的更多信息 AWS KMS，请参阅《AWS Key Management Service 开发人员指南》。

主题

查看你的 AWS 托管式密钥

当您使用创建管道向导创建第一个管道时，系统将在您创建管道的同一区域为您创建一个 S3 存储桶。存储桶用于存储管道项目。当管道运行时，构件会放入 S3 存储桶并从中检索。默认情况下，CodePipeline 使用服务器端加密，并 AWS KMS 使用 AWS 托管式密钥 适用于 Amazon S3 的（aws/s3密钥）。 AWS 托管式密钥 这是创建并存储在您的 AWS 账户中。从 S3 存储桶检索项目时，CodePipeline 使用相同的 SSE-KMS 进程来解密该项目。

查看有关您的信息 AWS 托管式密钥

1. 登录 AWS Management Console 并打开 AWS KMS 控制台。
2. 如果出现欢迎页面，请选择立即开始使用。
3. 在服务导航窗格中，选择AWS 托管式密钥。
4. 选择您的管道所在的区域。例如，如果管道是在 us-east-2 中创建的，则确保将筛选条件设为美国东部（俄亥俄州）。

   有关可用区域和终端节点的更多信息 CodePipeline，请参阅终AWS CodePipeline 端节点和配额。
5. 在列表中，选择包含管道所用别名的密钥（默认情况下为 aws/s3）。有关该密钥的基本信息将会显示。

使用 AWS CloudFormation 或，为 S3 存储桶配置服务器端加密 AWS CLI

使用 AWS CloudFormation 或创建管道时，必须手动配置服务器端加密。 AWS CLI 使用上面的示例桶策略，然后创建您自己的客户管理的密钥。您也可以使用自己的密钥，而不是 AWS 托管式密钥。选择自己的密钥的一些理由包括：

- 您希望按时间表轮换密钥以满足贵组织的业务或安全要求。
- 您要创建一个使用与另一个 AWS 账户关联的资源的管道。这需要使用客户管理的密钥。有关更多信息，请参阅 在中 CodePipeline 创建使用其他 AWS 账户资源的管道。

加密最佳实践建议不要广泛重复使用加密密钥。作为最佳实践，请定期轮换您的密钥。要为您的 AWS KMS 密钥创建新的加密材料，您可以创建客户托管密钥，然后更改应用程序或别名以使用新的客户托管密钥。或者，您可以为现有客户管理的密钥启用自动密钥轮换。

要轮换客户管理的密钥，请参阅轮换密钥。

> **Important**
>
> CodePipeline 仅支持对称 KMS 密钥。请勿使用非对称 KMS 密钥对 S3 桶中的数据进行加密。

AWS Secrets Manager 用于跟踪数据库密码或第三方 API 密钥

我们建议您在数据库凭证、API 密钥和其他密钥的整个生命周期中使用 AWS Secrets Manager 轮换、管理和检索它们。Secrets Manager 使您可以将代码中的硬编码凭证（包括密码）替换为对 Secrets Manager 的 API 调用，以便以编程方式检索密钥。有关更多信息，请参阅什么是 S AWS ecrets Manager？ 在 S AWS ecrets Manager 用户指南中。

对于在模板中传递作为机密的参数（例如 OAuth 凭证）的管道，您应在 AWS CloudFormation 模板中包含用于访问存储在 Secrets Manager 中的密钥的动态引用。有关引用 ID 模式和示例，请参阅 AWS CloudFormation 用户指南 中的 Secrets Manager 密钥。有关在管道中 web GitHub hook 的模板片段中使用动态引用的示例，请参阅 Webhook 资源配置。

另请参阅

下列相关资源可以帮助您管理密钥。

- Secrets Manager 可以自动轮换数据库凭证，例如用于轮换 Amazon RDS 密钥。有关更多信息，请参阅 S AWS ecrets Manager 用户指南中的轮换 S AWS ecrets Manager 密钥。
- 要查看有关向 AWS CloudFormation 模板添加 Secrets Manager 动态引用的说明，请参阅 https://aws.amazon.com/blogs/security/how-to-create-and-retrieve-secrets-managed-in-aws-secrets-manager-using-aws-cloudformation-template/。

适用于 AWS CodePipeline的身份和访问管理

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制谁可以进行身份验证（登录）和授权（拥有权限）使用 CodePipeline 资源。您可以使用 IAM AWS 服务 ，无需支付额外费用。

主题

受众

您的使用方式 AWS Identity and Access Management (IAM) 会有所不同，具体取决于您所做的工作 CodePipeline。

服务用户-如果您使用 CodePipeline 服务完成工作，则管理员会为您提供所需的凭证和权限。当你使用更多 CodePipeline 功能来完成工作时，你可能需要额外的权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问中的功能 CodePipeline，请参阅对 AWS CodePipeline 身份和访问进行故障排除。

服务管理员-如果您负责公司的 CodePipeline 资源，则可能拥有完全访问权限 CodePipeline。您的工作是确定您的服务用户应访问哪些 CodePipeline 功能和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要详细了解您的公司如何将 IAM 与配合使用 CodePipeline，请参阅如何 AWS CodePipeline 与 IAM 配合使用。

IAM 管理员 — 如果您是 IAM 管理员，则可能需要详细了解如何编写策略来管理访问权限 CodePipeline。要查看您可以在 IAM 中使用的 CodePipeline 基于身份的策略示例，请参阅。AWS CodePipeline 基于身份的策略示例

使用身份进行身份验证

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 AWS 账户根用户任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。 AWS IAM Identity Center （ IAM Identity Center ）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。

当您以联合身份登录时，您的管理员以前使用 IAM 角色设置了身份联合验证。当你使用联合访问 AWS 时，你就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》中的如何登录到您 AWS 账户的。

如果您 AWS 以编程方式访问，则会 AWS 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 AWS 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅 IAM 用户指南中的签署 AWS API 请求。

无论使用何种身份验证方法，您可能需要提供其他安全信息。例如， AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的多重身份验证和《IAM 用户指南》中的在 AWS中使用多重身份验证（MFA）。

AWS 账户根用户

创建时 AWS 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 AWS 服务 和资源。此身份被称为 AWS 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关需要您以根用户身份登录的任务的完整列表，请参阅 IAM 用户指南 中的 需要根用户凭证的任务。

IAM 用户和群组

I AM 用户是您 AWS 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用场景需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的使用场景定期轮换访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可能具有一个名为 IAMAdmins 的组，并为该组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的何时创建 IAM 用户（而不是角色）。

IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。您可以 AWS Management Console 通过切换角色在中临时担任 IAM 角色。您可以通过调用 AWS CLI 或

AWS API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的 使用 IAM 角色。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问 – 要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅《IAM 用户指南》中的 为第三方身份提供商创建角色。如果您使用 IAM Identity Center，则需要配置权限集。为控制您的身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的权限集。
- 临时 IAM 用户权限 – IAM 用户可代入 IAM 用户或角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取 – 您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些资源 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。
- 跨服务访问 — 有些 AWS 服务 使用其他 AWS 服务服务中的功能。例如，当您在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 Amazon S3 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。
  - 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。
  - 服务相关角色-服务相关角色是一种与服务相关联的服务角色。 AWS 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 A@@ mazon EC2 上运行的应用程序 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时证书。这优先于在 EC2 实例中存储访问密钥。要向 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建附加到该实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的 使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。

要了解是使用 IAM 角色还是 IAM 用户，请参阅《IAM 用户指南》中的何时创建 IAM 角色（而不是用户）。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略是其中的一个对象 AWS ，当与身份或资源关联时，它会定义其权限。 AWS 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的结构和内容的更多信息，请参阅《IAM 用户指南》中的 JSON 策略概览。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体 可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM 策略，用户可以代入角色。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。拥有该策略的用户可以从 AWS Management Console AWS CLI、或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的创建 IAM 策略。

基于身份的策略可以进一步归类为内联策略或托管式策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 AWS 账户。托管策略包括 AWS 托管策略和客户托管策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管式策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Simple Storage Service（Amazon S3）存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用 IAM 中的 AWS 托管策略。

其他策略类型

AWS 支持其他不太常见的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界 - 权限边界是一个高级功能，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅《IAM 用户指南》中的 IAM 实体的权限边界。
- 服务控制策略 (SCP)-SCP 是 JSON 策略，用于指定组织或组织单位 (OU) 的最大权限。 AWS Organizations AWS Organizations 是一项用于对您的企业拥有的多 AWS 账户 项进行分组和集中管理的服务。如果在组织内启用了所有功能，则可对任意或全部账户应用服务控制策略 (SCP)。SCP 限制成员账户中的实体（包括每个 AWS 账户根用户实体）的权限。有关 Organizations 和 SCP 的更多信息，请参阅《AWS Organizations 用户指南》中的 SCP 的工作原理。
- 会话策略 – 会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅《IAM 用户指南》中的会话策略。

如何 AWS CodePipeline 与 IAM 配合使用

在使用 IAM 管理访问权限之前 CodePipeline，您应该了解哪些可用的 IAM 功能 CodePipeline。要全面了解如何使用 IAM CodePipeline 以及其他方法 AWS 服务 ，请参阅AWS 服务 IAM 用户指南中的如何与 I AM 配合使用。

主题

- CodePipeline 基于身份的策略
- CodePipeline 基于资源的政策
- 基于 CodePipeline 标签的授权
- CodePipeline IAM 角色

CodePipeline 基于身份的策略

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。CodePipeline 支持特定的操作、资源和条件键。要了解在 JSON 策略中使用的所有元素，请参阅《IAM 用户指南》 中的 IAM JSON 策略元素参考。

操作

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体 可以对什么资源 执行操作，以及在什么 条件 下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限 操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

正在执行的策略操作在操作前 CodePipeline 使用以下前缀:`codepipeline:`.

例如，要授予某人查看账户中现有管道的权限，您可以在其策略中包括 `codepipeline:GetPipeline` 操作。策略声明必须包含`Action`或`NotAction`元素。 CodePipeline 定义了它自己的一组操作，这些操作描述了您可以使用此服务执行的任务。

要在单个语句中指定多项操作，请使用逗号将它们隔开，如下所示：

```
"Action": [
      "codepipeline:action1",
      "codepipeline:action2"
```

您也可以使用通配符 （*) 指定多个操作。例如，要指定以单词 `Get` 开头的所有操作，包括以下操作：

```
"Action": "codepipeline:Get*"
```

有关 CodePipeline 操作列表，请参阅 IAM 用户指南 AWS CodePipeline中的由定义的操作。

资源

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体 可以对什么资源 执行操作，以及在什么条件 下执行。

Resource JSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。作为最佳实践，请使用其 Amazon 资源名称（ARN）指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

CodePipeline 资源和运营

在中 CodePipeline，主要资源是管道。在策略中，您可以使用 Amazon 资源名称 (ARN) 来标识该政策适用的资源。 CodePipeline 支持可与主资源一起使用的其他资源，例如阶段、操作和自定义操作。这些资源称作子资源。这些资源和子资源具有与其关联的唯一 Amazon 资源名称（ARN）。有关 ARN 的更多信息，请参阅中的 Amazon 资源名称 (ARN) AWS 服务 和命名空间。Amazon Web Services 一般参考要获取与您的管道关联的管道 ARN，您可以在控制台的设置下找到管道 ARN。有关更多信息，请参阅 查看管道 ARN 和服务角色 ARN（控制台）。

资源类型	ARN 格式
管道	arn:aws:codepipeline:*region*:*account*:*pipeline-name*
舞台	arn:aws:codepipeline:*region*:*account*:*pipeline-name* /*stage-name*
操作	arn:aws:codepipeline:*region*:*account*:*pipeline-name* /*stage-name* /*action-name*
自定义操作	arn:aws:codepipeline:*region*:*account*:actionty pe:*owner*/*category*/*provider*/*version*
所有 CodePipeline 资源	arn:aws:codepipeline:*
指定区域中指定账户拥有的所有 CodePipeline 资源	arn:aws:codepipeline:*region*:*account*:*

> **Note**
>
> 中的大多数服务都 AWS 将冒号 (:) 或正斜杠 (/) 视为 ARN 中的相同字符。但是，在事件模式和规则中 CodePipeline 使用精确匹配。请在创建事件模式时务必使用正确的 ARN 字符，以使其匹配需要匹配的管道中的 ARN 语法。

在中 CodePipeline，有支持资源级权限的 API 调用。资源级权限指示 API 调用是否可以指定资源 ARN，或者 API 调用是否只能使用通配符指定所有资源。CodePipeline 权限参考有关资源级权限的详细说明以及支持资源级权限的 CodePipeline API 调用的列表，请参阅。

例如，您可以使用某个特定管道（*myPipeline*）的 ARN 在语句中指定该管道，如下所示：

```
"Resource": "arn:aws:codepipeline:us-east-2:111222333444:myPipeline"
```

还可以使用通配符（*）指定属于特定账户的所有管道，如下所示：

```
"Resource": "arn:aws:codepipeline:us-east-2:111222333444:*"
```

要指定所有资源，或者，如果特定 API 操作不支持 ARN，请在 `Resource` 元素中使用（*）通配符，如下所示：

```
"Resource": "*"
```

> **Note**
>
> 在您创建 IAM 策略时，请遵循授予最低权限这一标准安全建议，即仅授予执行任务所需的权限。如果某个 API 调用支持 ARN，则它支持资源级权限，您无需使用通配符（*）。

某些 CodePipeline API 调用接受多个资源（例如`GetPipeline`）。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": ["arn1", "arn2"]
```

CodePipeline 提供了一组使用 CodePipeline 资源的操作。有关可用操作的列表，请参阅 CodePipeline 权限参考。

条件键

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

在 Condition 元素（或 Condition 块）中，可以指定语句生效的条件。Condition 元素是可选的。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果您在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则使用逻辑OR运算来 AWS 评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM 策略元素：变量和标签。

AWS 支持全局条件密钥和特定于服务的条件密钥。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

CodePipeline 定义自己的条件键集，还支持使用一些全局条件键。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

所有 Amazon EC2 操作都支持 aws:RequestedRegion 和 ec2:Region 条件键。有关更多信息，请参阅示例：限制对特定区域的访问。

要查看 CodePipeline 条件键列表，请参阅 IAM 用户指南 AWS CodePipeline中的条件密钥。要了解您可以使用条件键的操作和资源，请参阅操作定义者 AWS CodePipeline。

示例

要查看 CodePipeline 基于身份的策略的示例，请参阅。AWS CodePipeline 基于身份的策略示例

CodePipeline 基于资源的政策

CodePipeline 不支持基于资源的策略。但是，提供了与之相关的 S3 服务的基于资源的策略示例。CodePipeline

示例

要查看 CodePipeline 基于资源的策略的示例，请参阅 AWS CodePipeline 基于资源的策略示例

基于 CodePipeline 标签的授权

您可以为 CodePipeline 资源附加标签或在请求中传递标签 CodePipeline。要基于标签控制访问，您需要使用 `codepipeline:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。有关为 CodePipeline 资源添加标签的更多信息，请参阅标记资源。

要查看基于身份的策略（用于根据资源上的标签来限制对该资源的访问）的示例，请参阅使用标签控制对 CodePipeline资源的访问权限。

CodePipeline IAM 角色

I AM 角色是您的 AWS 账户中具有特定权限的实体。

将临时凭证与配合使用 CodePipeline

可以使用临时凭证进行联合身份验证登录，分派 IAM 角色或分派跨账户角色。您可以通过调用AssumeRole或之类的 AWS STS API 操作来获取临时安全证书GetFederationToken。

CodePipeline 支持使用临时证书。

服务角色

CodePipeline 允许服务代表您担任服务角色。此角色允许服务访问其他服务中的资源以代表您完成操作。服务角色显示在 IAM 账户中，并归该账户所有。这意味着，IAM 管理员可以更改该角色的权限。但是，这样做可能会中断服务的功能。

CodePipeline 支持服务角色。

AWS CodePipeline 基于身份的策略示例

默认情况下，IAM 用户和角色无权创建或修改 CodePipeline 资源。他们也无法使用 AWS Management Console AWS CLI、或 AWS API 执行任务。IAM 管理员必须创建 IAM 策略，以便为用户和角色授予权限以对所需的指定资源执行特定的 API 操作。然后，管理员必须将这些策略附加到需要这些权限的 IAM 用户或组。

要了解如何使用这些示例 JSON 策略文档创建 IAM 基于身份的策略，请参阅《IAM 用户指南》中的 在 JSON 选项卡上创建策略。

要了解如何创建使用其他账户资源的管道以及相关的示例策略，请参阅在中 CodePipeline 创建使用其他 AWS 账户资源的管道。

主题

策略最佳实践

基于身份的策略决定了某人是否可以在您的账户中创建、访问或删除 CodePipeline 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下准则和建议：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对您的用例的 AWS 客户托管策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的AWS 托管策略或工作职能的AWS 托管策略。
- 应用最低权限 – 在使用 IAM 策略设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM 策略中的条件进一步限制访问权限 – 您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定的方式使用的，则也可以使用条件来授予对服务操作的访问权限 AWS 服务，例如 AWS CloudFormation。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。
- 使用 IAM Access Analyzer 验证您的 IAM 策略，以确保权限的安全性和功能性 – IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM 策略语言 (JSON) 和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM Acess Analyzer 策略验证。
- 需要多重身份验证 (MFA)-如果 AWS 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的 配置受 MFA 保护的 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实操。

在控制台中查看资源

CodePipeline 控制台需要在您登录的 AWS 地区显示您的 AWS 账户存储库列表的`ListRepositories`权限。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。

允许用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包括在控制台上或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
```

```
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

基于身份的策略 (IAM) 示例

您可以向 IAM 身份附加策略。例如，您可以执行以下操作：

- 将@@ 权限策略附加到您账户中的用户或群组-要授予用户在 CodePipeline 控制台中查看管道的权限，您可以将权限策略附加到该用户所属的用户或群组。
- 向角色附加权限策略（授予跨账户权限）– 您可以向 IAM 角色附加基于身份的权限策略，以授予跨账户的权限。例如，账户 A 中的管理员可以创建一个角色来向另一个 AWS 账户（例如账户 B）或授予跨账户权限， AWS 服务 如下所示：

  1. 账户 A 管理员可以创建一个 IAM 角色，然后向该角色附加授予其访问账户 A 中资源的权限策略。
  2. 账户 A 管理员可以把信任策略附加至用来标识账户 B 的角色，账户 B 由此可以作为主体代入该角色。
  3. 然后，账户 B 管理员可以将代入该角色的权限委托给账户 B 中的任何用户。这样，账户 B 中的用户就可以创建或访问账户 A 中的资源。如果您想授予担任该角色的 AWS 服务 权限，则信任策略中的 AWS 服务 委托人也可以是委托人。

  有关使用 IAM 委托权限的更多信息，请参阅 IAM 用户指南中的访问权限管理。

下面显示了一个权限策略的示例，该策略可以授权启用和禁用 us-west-2 region 区域名为 MyFirstPipeline 的管道中所有阶段之间的转换：

```
{
  "Version": "2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "codepipeline:EnableStageTransition",
        "codepipeline:DisableStageTransition"
      ],
      "Resource" : [
        "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline/*"
      ]
    }
  ]
}
```

以下示例显示了 111222333444 账户中的一项策略，该策略允许用户查看但不能更改控制台中命名的管道。MyFirstPipeline CodePipeline 该策略基于 AWSCodePipeline_ReadOnlyAccess 托管策略，但由于它特定于 MyFirstPipeline 管道，因此无法直接使用托管策略。如果您不想将策略限制在特定的管道中，请考虑使用由创建和维护的托管策略之一 CodePipeline。有关更多信息，请参阅使用管理的策略。您必须将该策略附加到您为进行访问而创建的 IAM 角色，例如名为 CrossAccountPipelineViewers 的角色：

```
{
  "Statement": [
    {
      "Action": [
        "codepipeline:GetPipeline",
        "codepipeline:GetPipelineState",
        "codepipeline:GetPipelineExecution",
        "codepipeline:ListPipelineExecutions",
        "codepipeline:ListActionExecutions",
        "codepipeline:ListActionTypes",
        "codepipeline:ListPipelines",
        "codepipeline:ListTagsForResource",
        "iam:ListRoles",
        "s3:ListAllMyBuckets",
        "codecommit:ListRepositories",
        "codedeploy:ListApplications",
```

```
        "lambda:ListFunctions",
        "codestar-notifications:ListNotificationRules",
        "codestar-notifications:ListEventTypes",
        "codestar-notifications:ListTargets"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline"
    },
    {
      "Action": [
        "codepipeline:GetPipeline",
        "codepipeline:GetPipelineState",
        "codepipeline:GetPipelineExecution",
        "codepipeline:ListPipelineExecutions",
        "codepipeline:ListActionExecutions",
        "codepipeline:ListActionTypes",
        "codepipeline:ListPipelines",
        "codepipeline:ListTagsForResource",
        "iam:ListRoles",
        "s3:GetBucketPolicy",
        "s3:GetObject",
        "s3:ListBucket",
        "codecommit:ListBranches",
        "codedeploy:GetApplication",
        "codedeploy:GetDeploymentGroup",
        "codedeploy:ListDeploymentGroups",
        "elasticbeanstalk:DescribeApplications",
        "elasticbeanstalk:DescribeEnvironments",
        "lambda:GetFunctionConfiguration",
        "opsworks:DescribeApps",
        "opsworks:DescribeLayers",
        "opsworks:DescribeStacks"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "CodeStarNotificationsReadOnlyAccess",
      "Effect": "Allow",
      "Action": [
        "codestar-notifications:DescribeNotificationRule"
      ],
      "Resource": "*",
      "Condition": {
```

```
        "StringLike": {
          "codestar-notifications:NotificationsForResource": "arn:aws:codepipeline:*"
        }
      }
    }
  ],
  "Version": "2012-10-17"
}
```

创建此策略后，在 111222333444 账户中创建 IAM 角色，并将策略附加到该角色。在角色的信任关系中，您必须添加将担任此角色的 AWS 账户。以下示例显示了一项策略，该策略允许来自 111111111111 账户的用户担任在 *1112223334* AWS 44 账户中定义的角色：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111111111111:root"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

以下示例显示了在 *111111111111 ################ 1112* 22333444 AWS 账户中指定的*CrossAccountPipelineViewers*角色：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::111222333444:role/CrossAccountPipelineViewers"
        }
    ]
}
```

您可以创建 IAM 策略来限制您账户中的用户有权访问的调用和资源，然后将这些策略与管理用户相关联。有关如何创建 IAM 角色以及浏览的 IAM 策略声明示例 CodePipeline，请参阅客户管理型策略示例。

使用标签控制对 CodePipeline资源的访问权限

IAM 策略声明中的条件是您用来指定 CodePipeline 操作所需资源权限的语法的一部分。使用条件中的标签是控制对资源和请求的访问的一种方法。有关为 CodePipeline 资源添加标签的信息，请参阅标记资源。本主题讨论了基于标签的访问控制。

在设计 IAM 策略时，您可以通过授予对特定资源的访问权限来设置精细权限。但随着您管理的资源数量的增加，此任务会变得日益复杂。标记资源并在策略声明条件中使用标签可以简化这一任务。您可以向具有特定标签的任何资源批量授予访问权限。然后，在创建期间或之后，您可以将此标签反复应用到相关资源。

标签可以附加到资源，也可以从请求传入支持标签的服务。在中 CodePipeline，资源可以有标签，有些操作可以包含标签。在创建 IAM 策略时，您可以使用标签条件键来控制：

- 基于资源已有的标签，哪些用户可以对管道资源执行操作。
- 哪些标签可以在操作的请求中传递。
- 是否特定标签键可在请求中使用。

利用字符串条件运算符，您可以构建基于键与字符串值的对比来限制访问的 `Condition` 元素。除 Null 条件，您可在任何条件运算符名称的末尾添加 `IfExists`。如果您是指“如果请求的内容中存在策略键，则依照策略所述来处理键。如果该键不存在，则条件元素的计算结果将为 true。” 例如，您可以使用 `StringEqualsIfExists` 按条件键进行限制，这些条件键可能不存在于其他类型的资源上。

有关标签条件键的完整请求和语义，请参阅使用标签控制访问。有关条件键的更多信息，请参阅以下资源。本节中的 CodePipeline策略示例与以下有关条件键的信息保持一致，并通过 CodePipeline 诸如资源嵌套之类的细微差别示例对其进行扩展。

- 字符串条件运算符
- AWS 服务 可以与 IAM 配合使用
- SCP 语法
- IAM JSON 策略元素：Condition
- aws: RequestTag /tag-key
- 的条件密钥 CodePipeline

以下示例演示了如何在策略中为 CodePipeline 用户指定标签条件。

Example 1：基于请求中的标签限制操作

AWSCodePipeline_FullAccess托管用户策略为用户提供了对任何资源执行任何 CodePipeline 操作的无限权限。

以下策略将限制此权力，不允许未经授权的用户在请求中列出特定标签的情况下创建管道。为此，如果请求指定一个名为 Project 的带有值为 ProjectA 或 ProjectB 的标签，则它会拒绝 CreatePipeline 操作。（aws:RequestTag 条件键用于控制可以通过 IAM 请求传递哪些标签。）

在下面的示例中，该策略的目的是不允许未经授权的用户使用指定的标签值创建管道。但是，创建管道需要访问管道本身之外的资源（例如，管道操作和阶段）。由于策略中指定的 'Resource' 是 '*'，因此将针对每个具有 ARN 并且在创建管道时创建的资源，评估该策略。这些其他资源没有标签条件键，因此 StringEquals 检查会失败，并且不会向用户授予创建任何管道的能力。要解决此问题，请改用 StringEqualsIfExists 条件运算符。这样，仅当条件键存在时才会进行测试。

您可能会读到以下内容：“如果所检查的资源具有标签 "RequestTag/Project" 条件键，则仅当键值以 projectA 开头时允许操作。如果所检查的资源没有改条件键，则无需担心。”

此外，该策略使用 aws:TagKeys 条件键，不允许标签修改操作中包含这些相同的标签值，从而防止这些未经授权的用户篡改资源。除托管用户策略外，客户的管理员还必须将此 IAM 策略附加到未经授权的管理用户。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codepipeline:CreatePipeline",
        "codepipeline:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEqualsIfExists": {
          "aws:RequestTag/Project": ["ProjectA", "ProjectB"]
        }
      }
    },
    {
```

```
      "Effect": "Deny",
      "Action": [
        "codepipeline:UntagResource"
      ],
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["Project"]
        }
      }
    }
  ]
}
```

Example 2：基于资源标签限制标记操作

AWSCodePipeline_FullAccess托管用户策略为用户提供了对任何资源执行任何 CodePipeline 操作的无限权限。

以下策略限制此权力并拒绝未经授权的用户具有对指定项目管道执行操作的权限。为此，如果资源具有名为 Project 的带有值 ProjectA 或 ProjectB 之一的标签，那么它会拒绝某些操作。（使用 aws:ResourceTag 条件键，基于资源上的标签控制对这些资源的访问。）除托管用户策略外，客户的管理员还必须将此 IAM 策略附加到未经授权的 IAM 用户。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codepipeline:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Project": ["ProjectA", "ProjectB"]
        }
      }
    }
  ]
}
```

Example 3：基于请求中的标签允许操作

以下策略授予用户在中创建开发管道的权限 CodePipeline。

为此，如果请求指定一个名为 Project 的带有值为 ProjectA 的标签，则它允许 CreatePipeline 和 TagResource 操作。换句话说，唯一可以指定的标签键是 Project，它的值必须是 ProjectA。

aws:RequestTag 条件键用于控制可以通过 IAM 请求传递哪些标签。aws:TagKeys 条件确保标签键区分大小写。对于未附加 AWSCodePipeline_FullAccess 托管用户策略的用户或角色，此策略非常有用。托管策略为用户提供了对任何资源执行任何 CodePipeline 操作的无限权限。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codepipeline:CreatePipeline",
        "codepipeline:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/Project": "ProjectA"
        },
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["Project"]
        }
      }
    }
  ]
}
```

Example 4：基于资源标签限制取消标记操作

AWSCodePipeline_FullAccess托管用户策略为用户提供了对任何资源执行任何 CodePipeline 操作的无限权限。

以下策略限制此权力并拒绝未经授权的用户具有对指定项目管道执行操作的权限。为此，如果资源具有名为 Project 的带有值 ProjectA 或 ProjectB 之一的标签，那么它会拒绝某些操作。

此外，该策略使用 aws:TagKeys 条件键，不允许标签修改操作完全删除 Project 标签，从而防止这些未经授权的用户篡改资源。除托管用户策略外，客户的管理员还必须将此 IAM 策略附加到未经授权的用户或角色。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codepipeline:UntagResource"
      ],
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["Project"]
        }
      }
    }
  ]
}
```

使用 CodePipeline 控制台所需的权限

要 CodePipeline 在 CodePipeline 控制台中使用，您必须拥有来自以下服务的最低权限集：

- AWS Identity and Access Management
- Amazon Simple Storage Service

这些权限允许您描述 AWS 账户的其他 AWS 资源。

根据您纳入管道的其他服务，您可能需要来自以下一项或多项的权限：

- AWS CodeCommit
- CodeBuild
- AWS CloudFormation
- AWS CodeDeploy
- AWS Elastic Beanstalk
- AWS Lambda

- AWS OpsWorks

如果创建比必需的最低权限更为严格的 IAM policy，对于附加了该 IAM policy 的用户，控制台将无法按预期正常运行。为确保这些用户仍然可以使用 CodePipeline 控制台，还要将`AWSCodePipeline_ReadOnlyAccess`托管策略附加到该用户，如中所述AWS 的托管策略 AWS CodePipeline。

您无需为调用 AWS CLI 或 CodePipeline API 的用户设置最低控制台权限。

AWS 的托管策略 AWS CodePipeline

AWS 托管策略是由创建和管理的独立策略 AWS。 AWS 托管策略旨在为许多常见用例提供权限，以便您可以开始为用户、组和角色分配权限。

请记住， AWS 托管策略可能不会为您的特定用例授予最低权限权限，因为它们可供所有 AWS 客户使用。我们建议通过定义特定于您的使用场景的客户管理型策略来进一步减少权限。

您无法更改 AWS 托管策略中定义的权限。如果 AWS 更新 AWS 托管策略中定义的权限，则更新会影响该策略所关联的所有委托人身份（用户、组和角色）。 AWS 最有可能在启动新的 API 或现有服务可以使用新 AWS 服务 的 API 操作时更新 AWS 托管策略。

有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略。

> **Important**
>
> 亚马逊云科技托管式策略 `AWSCodePipelineFullAccess` 和 `AWSCodePipelineReadOnlyAccess` 已被替换。使用 `AWSCodePipeline_FullAccess` 和 `AWSCodePipeline_ReadOnlyAccess` 策略。

AWS 托管策略：**`AWSCodePipeline_FullAccess`**

这是一项授予完全访问权限的政策 CodePipeline。要在 IAM 控制台中查看 JSON 策略文档，请参阅AWSCodePipeline_FullAccess。

权限详细信息

该策略包含以下权限。

- codepipeline— 向授予权限 CodePipeline。
- chatbot— 授予权限以允许委托人管理中的资源。 AWS Chatbot
- cloudformation— 授予权限以允许委托人管理中的资源堆栈。 AWS CloudFormation
- cloudtrail— 授予权限以允许委托人管理中的日志资源。 CloudTrail
- codebuild— 授予权限以允许委托人访问中的生成资源。 CodeBuild
- codecommit— 授予权限以允许委托人访问中的源资源。 CodeCommit
- codedeploy— 授予权限以允许委托人访问中的部署资源。 CodeDeploy
- codestar-notifications— 授予权限以允许委托人访问 AWS CodeStar 通知中的资源。
- ec2— 授予权限以允许在中进行部署 CodeCatalyst 以管理 Amazon EC2 中的弹性负载平衡。
- ecr：授予权限以允许访问 Amazon ECR 中的资源。
- elasticbeanstalk：授予权限以允许主体访问 Elastic Beanstalk 中的资源。
- iam：授予权限以允许主体管理 IAM 中的角色和策略。
- lambda：授予权限以允许主体管理 Lambda 中的资源。
- events— 授予权限以允许委托人管理 Ev CloudWatch ents 中的资源。
- opsworks— 授予权限以允许委托人管理中的资源。 AWS OpsWorks
- s3：授予权限以允许主体管理 Amazon S3 中的资源。
- sns：授予权限以允许主体管理 Amazon SNS 中的通知资源。
- states— 授予权限以允许委托人查看中的状态机。 AWS Step Functions状态机由一组状态组成，这些状态用于管理任务和状态之间的转换。

```
{
    "Statement": [
        {
            "Action": [
                "codepipeline:*",
                "cloudformation:DescribeStacks",
                "cloudformation:ListStacks",
                "cloudformation:ListChangeSets",
                "cloudtrail:DescribeTrails",
```

```
                "codebuild:BatchGetProjects",
                "codebuild:CreateProject",
                "codebuild:ListCuratedEnvironmentImages",
                "codebuild:ListProjects",
                "codecommit:ListBranches",
                "codecommit:GetReferences",
                "codecommit:ListRepositories",
                "codedeploy:BatchGetDeploymentGroups",
                "codedeploy:ListApplications",
                "codedeploy:ListDeploymentGroups",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs",
                "ecr:DescribeRepositories",
                "ecr:ListImages",
                "ecs:ListClusters",
                "ecs:ListServices",
                "elasticbeanstalk:DescribeApplications",
                "elasticbeanstalk:DescribeEnvironments",
                "iam:ListRoles",
                "iam:GetRole",
                "lambda:ListFunctions",
                "events:ListRules",
                "events:ListTargetsByRule",
                "events:DescribeRule",
                "opsworks:DescribeApps",
                "opsworks:DescribeLayers",
                "opsworks:DescribeStacks",
                "s3:ListAllMyBuckets",
                "sns:ListTopics",
                "codestar-notifications:ListNotificationRules",
                "codestar-notifications:ListTargets",
                "codestar-notifications:ListTagsforResource",
                "codestar-notifications:ListEventTypes",
                "states:ListStateMachines"
            ],
            "Effect": "Allow",
            "Resource": "*",
            "Sid": "CodePipelineAuthoringAccess"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:ListBucket",
```

```
                "s3:GetBucketPolicy",
                "s3:GetBucketVersioning",
                "s3:GetObjectVersion",
                "s3:CreateBucket",
                "s3:PutBucketPolicy"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3::*:codepipeline-*",
            "Sid": "CodePipelineArtifactsReadWriteAccess"
        },
        {
            "Action": [
                "cloudtrail:PutEventSelectors",
                "cloudtrail:CreateTrail",
                "cloudtrail:GetEventSelectors",
                "cloudtrail:StartLogging"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:cloudtrail:*:*:trail/codepipeline-source-trail",
            "Sid": "CodePipelineSourceTrailReadWriteAccess"
        },
        {
            "Action": [
                "iam:PassRole"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iam::*:role/service-role/cwe-role-*"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "events.amazonaws.com"
                    ]
                }
            },
            "Sid": "EventsIAMPassRole"
        },
        {
            "Action": [
                "iam:PassRole"
            ],
            "Effect": "Allow",
            "Resource": "*",
```

```
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "codepipeline.amazonaws.com"
                    ]
                }
            },
            "Sid": "CodePipelineIAMPassRole"
        },
        {
            "Action": [
                "events:PutRule",
                "events:PutTargets",
                "events:DeleteRule",
                "events:DisableRule",
                "events:RemoveTargets"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:events:*:*:rule/codepipeline-*"
            ],
            "Sid": "CodePipelineEventsReadWriteAccess"
        },
        {
            "Sid": "CodeStarNotificationsReadWriteAccess",
            "Effect": "Allow",
            "Action": [
                "codestar-notifications:CreateNotificationRule",
                "codestar-notifications:DescribeNotificationRule",
                "codestar-notifications:UpdateNotificationRule",
                "codestar-notifications:DeleteNotificationRule",
                "codestar-notifications:Subscribe",
                "codestar-notifications:Unsubscribe"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "codestar-notifications:NotificationsForResource":
"arn:aws:codepipeline:*"
                }
            }
        },
        {
            "Sid": "CodeStarNotificationsSNSTopicCreateAccess",
```

```
            "Effect": "Allow",
            "Action": [
                "sns:CreateTopic",
                "sns:SetTopicAttributes"
            ],
            "Resource": "arn:aws:sns:*:*:codestar-notifications*"
        },
        {
            "Sid": "CodeStarNotificationsChatbotAccess",
            "Effect": "Allow",
            "Action": [
                "chatbot:DescribeSlackChannelConfigurations",
                "chatbot:ListMicrosoftTeamsChannelConfigurations"
            ],
            "Resource": "*"
        }
    ],
    "Version": "2012-10-17"
}
```

AWS 托管策略： AWSCodePipeline_ReadOnlyAccess

这是一项授予只读访问权限的策略 CodePipeline。要在 IAM 控制台中查看 JSON 策略文档，请参阅AWSCodePipeline_ReadOnlyAccess。

权限详细信息

该策略包含以下权限。

- codepipeline— 授予中操作的权限 CodePipeline。
- codestar-notifications— 授予权限以允许委托人访问 AWS CodeStar 通知中的资源。
- s3：授予权限以允许主体管理 Amazon S3 中的资源。
- sns：授予权限以允许主体管理 Amazon SNS 中的通知资源。

```
{
    "Statement": [
        {
            "Action": [
```

```
                "codepipeline:GetPipeline",
                "codepipeline:GetPipelineState",
                "codepipeline:GetPipelineExecution",
                "codepipeline:ListPipelineExecutions",
                "codepipeline:ListActionExecutions",
                "codepipeline:ListActionTypes",
                "codepipeline:ListPipelines",
                "codepipeline:ListTagsForResource",
                "s3:ListAllMyBuckets",
                "codestar-notifications:ListNotificationRules",
                "codestar-notifications:ListEventTypes",
                "codestar-notifications:ListTargets"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:ListBucket",
                "s3:GetBucketPolicy"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3::*:codepipeline-*"
        },
        {
            "Sid": "CodeStarNotificationsReadOnlyAccess",
            "Effect": "Allow",
            "Action": [
                "codestar-notifications:DescribeNotificationRule"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "codestar-notifications:NotificationsForResource":
 "arn:aws:codepipeline:*"
                }
            }
        }
    ],
    "Version": "2012-10-17"
}
```

AWS 托管策略：**AWSCodePipelineApproverAccess**

这是一项授予批准或拒绝手动审批操作的权限的策略。要在 IAM 控制台中查看 JSON 策略文档，请参阅 AWSCodePipelineApproverAccess...

权限详细信息

该策略包含以下权限。

- `codepipeline`— 授予中操作的权限 CodePipeline。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "codepipeline:GetPipeline",
                "codepipeline:GetPipelineState",
                "codepipeline:GetPipelineExecution",
                "codepipeline:ListPipelineExecutions",
                "codepipeline:ListPipelines",
                "codepipeline:PutApprovalResult"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

AWS 托管策略： AWSCodePipelineCustomActionAccess

此政策允许用户在生成或测试操作中创建自定义操作 CodePipeline或集成 Jenkins 资源。要在 IAM 控制台中查看 JSON 策略文档，请参阅AWSCodePipelineCustomActionAccess。

权限详细信息

该策略包含以下权限。

- `codepipeline`— 授予中操作的权限 CodePipeline。

```
{
    "Statement": [
        {
            "Action": [
                "codepipeline:AcknowledgeJob",
                "codepipeline:GetJobDetails",
                "codepipeline:PollForJobs",
                "codepipeline:PutJobFailureResult",
                "codepipeline:PutJobSuccessResult"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ],
    "Version": "2012-10-17"
}
```

CodePipeline 托管策略和通知

CodePipeline 支持通知，可以将管道的重要更改通知用户。的托管策略 CodePipeline 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？。

完全访问托管策略中的通知的相关权限

此托管策略授予相关服务 CodeCommit、 CodeBuild CodeDeploy、和 AWS CodeStar 通知的权限。CodePipeline 该策略还授予您使用与您的管道集成的其他服务所需的权限，例如 Amazon S3、Elastic B CloudTrail eanstalk、Amazon EC2 和。 AWS CloudFormation应用此托管策略的用户还可以创建和管理用于通知的 Amazon SNS 主题、为用户订阅和取消订阅主题、列出要选择作为通知规则目标的主题，以及列出为 Slack 配置的 AWS Chatbot 客户端。

`AWSCodePipeline_FullAccess` 托管策略包含以下语句，以允许对通知进行完全访问。

```
    {
        "Sid": "CodeStarNotificationsReadWriteAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:CreateNotificationRule",
            "codestar-notifications:DescribeNotificationRule",
```

```
            "codestar-notifications:UpdateNotificationRule",
            "codestar-notifications:DeleteNotificationRule",
            "codestar-notifications:Subscribe",
            "codestar-notifications:Unsubscribe"
        ],
        "Resource": "*",
        "Condition" : {
            "StringLike" : {"codestar-notifications:NotificationsForResource" :
"arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline"}
        }
    },
    {
        "Sid": "CodeStarNotificationsListAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:ListNotificationRules",
            "codestar-notifications:ListTargets",
            "codestar-notifications:ListTagsforResource",
            "codestar-notifications:ListEventTypes"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CodeStarNotificationsSNSTopicCreateAccess",
        "Effect": "Allow",
        "Action": [
            "sns:CreateTopic",
            "sns:SetTopicAttributes"
        ],
        "Resource": "arn:aws:sns:*:*:codestar-notifications*"
    },
    {
        "Sid": "SNSTopicListAccess",
        "Effect": "Allow",
        "Action": [
            "sns:ListTopics"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CodeStarNotificationsChatbotAccess",
        "Effect": "Allow",
        "Action": [
            "chatbot:DescribeSlackChannelConfigurations",
```

```
            "chatbot:ListMicrosoftTeamsChannelConfigurations"
          ],
       "Resource": "*"
    }
```

只读托管策略中的通知的相关权限

AWSCodePipeline_ReadOnlyAccess 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" : {
           "StringLike" : {"codestar-notifications:NotificationsForResource" :
"arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline"}
       }
   },
   {
       "Sid": "CodeStarNotificationsListAccess",
       "Effect": "Allow",
       "Action": [
           "codestar-notifications:ListNotificationRules",
           "codestar-notifications:ListEventTypes",
           "codestar-notifications:ListTargets"
       ],
       "Resource": "*"
   }
```

有关 IAM 和通知的更多信息，请参阅 AWS CodeStar 通知的身份和访问管理。

AWS CodePipelineAWS 托管策略的更新

查看 CodePipeline 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。要获得有关此页面变更的自动提醒，请订阅 “ CodePipeline 文档历史记录” 页面上的 RSS feed。

更改	描述	日期
AWSCodePipeline_FullAccess— 对现有政策的更新	CodePipeline 已向此策略添加了支持`ListStacks` 权限 AWS CloudFormation。	2024 年 3 月 15 日
AWSCodePipeline_FullAccess— 对现有政策的更新	此政策已更新，添加了的权限 AWS Chatbot。有关更多信息，请参阅 CodePipeline 托管策略和通知。	2023 年 6 月 21 日
AWSCodePipeline_FullAccess和AWSCodePipeline_ReadOnlyAccess托管策略-对现有策略的更新	CodePipeline 使用 AWS Chatbot、向这些策略添加了支持其他通知类型的权限 `chatbot:ListMicrosoftTeamsChannelConfigurations` 。	2023 年 5 月 16 日
AWSCodePipelineFullAccess— 已弃用	此策略已被 `AWSCodePipeline_FullAccess` 取代。 2022 年 11 月 17 日之后，该策略不能附加到任何新的用户、组或角色。有关更多信息，请参阅 AWS 的托管策略 AWS CodePipeline。	2022 年 11 月 17 日
AWSCodePipelineReadOnlyAccess— 已弃用	此策略已被 `AWSCodePipeline_ReadOnlyAccess` 取代。 2022 年 11 月 17 日之后，该策略不能附加到任何新的用户、组或角色。有关更多信息，请参阅 AWS 的托管策略 AWS CodePipeline。	2022 年 11 月 17 日

更改	描述	日期
CodePipeline 已开始跟踪更改	CodePipeline 开始跟踪其 AWS 托管策略的更改。	2021 年 3 月 12 日

客户管理型策略示例

在本节中，您可以找到授予各种 CodePipeline 操作权限的用户策略示例。这些策略在您使用 CodePipeline API、 AWS 软件开发工具包或时起 AWS CLI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。有关更多信息，请参阅 使用 CodePipeline 控制台所需的权限。

> **Note**
>
> 所有示例都使用美国西部（俄勒冈州）区域（us-west-2）并且包含虚构的账户 ID。

示例

- 示例 1：授予获取管道状态的权限
- 示例 2：授予启用和禁用阶段之间的过渡的权限
- 示例 3：授予获取所有可用操作类型列表的权限
- 示例 4：授予批准或拒绝手动审批操作的权限
- 示例 5：授予轮询作业以查找自定义操作的权限
- 示例 6：附加或编辑 Jenkins 与集成的策略 AWS CodePipeline
- 示例 7：配置对管道的跨账户访问
- 示例 8：在管道中使用与另一个账户相关联的 AWS 资源

示例 1：授予获取管道状态的权限

以下示例将授予获取名为 MyFirstPipeline 的管道状态的权限：

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

```
                "codepipeline:GetPipelineState"
            ],
            "Resource": "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline"
        }
    ]
}
```

示例 2：授予启用和禁用阶段之间的过渡的权限

以下示例授予禁用和启用名为 MyFirstPipeline 的管道中所有阶段之间的过渡的权限：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:DisableStageTransition",
                "codepipeline:EnableStageTransition"
            ],
            "Resource": "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline/*"
        }
    ]
}
```

要允许用户禁用和启用管道中单个阶段的过渡，您必须指定该阶段。例如，为了允许用户启用和禁用名为 MyFirstPipeline 的管道中 Staging 阶段的过渡：

```
"Resource": "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline/Staging"
```

示例 3：授予获取所有可用操作类型列表的权限

以下示例授予获取可用于 us-west-2 区域中的管道的所有操作类型列表的权限：

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

```
            "Resource": "arn:aws:codepipeline:us-west-2:111222333444:actiontype:*"
        }
    ]
}
```

示例 4：授予批准或拒绝手动审批操作的权限

以下示例授予批准或拒绝名为 MyFirstPipeline 的管道中 Staging 阶段的手动审批操作的权限：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:PutApprovalResult"
            ],
            "Resource": "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline/
Staging/*"
        }
    ]
}
```

示例 5：授予轮询作业以查找自定义操作的权限

以下示例授予轮询所有管道中的作业以查找名为 TestProvider 的自定义操作的权限，该操作是第一个版本中的 Test 操作类型：

> **Note**
>
> 自定义操作的作业辅助角色可以在不同的 AWS 账户下配置，或者需要特定的 IAM 角色才能运行。

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

```
            ],
            "Resource": [
                "arn:aws:codepipeline:us-
west-2:111222333444:actionType:Custom/Test/TestProvider/1"
            ]
        }
    ]
}
```

示例 6：附加或编辑 Jenkins 与集成的策略 AWS CodePipeline

如果您将管道配置为使用 Jenkins 进行构建或测试，请为该集成创建单独的身份，并附上具有 Jenkins 和之间集成所需的最低权限的 IAM 策略。 CodePipeline此策略与 AWSCodePipelineCustomActionAccess 托管策略相同。以下示例显示了一项 Jenkins 集成策略：

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:AcknowledgeJob",
                "codepipeline:GetJobDetails",
                "codepipeline:PollForJobs",
                "codepipeline:PutJobFailureResult",
                "codepipeline:PutJobSuccessResult"
            ],
            "Resource": "*"
        }
    ],
    "Version": "2012-10-17"
}
```

示例 7：配置对管道的跨账户访问

您可以为另一个 AWS 账户中的用户和组配置对管道的访问。建议的方法是在创建管道的账户中创建角色。该角色应允许其他 AWS 账户的用户担任该角色并访问管道。有关更多信息，请参阅演练：使用角色进行跨账户访问。

以下示例显示了 80398EXAMPLE 账户中的一项策略，该策略允许用户查看但不能更改控制台MyFirstPipeline中命名的管道。 CodePipeline 该策略基于 AWSCodePipeline_ReadOnlyAccess 托管策略，但由于它特定于 MyFirstPipeline 管道，因此

无法直接使用托管策略。如果您不想将策略限制在特定的管道中，请考虑使用由创建和维护的托管策略之一 CodePipeline。有关更多信息，请参阅使用管理的策略。您必须将该策略附加到您为进行访问而创建的 IAM 角色，例如名为 `CrossAccountPipelineViewers` 的角色：

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:GetPipeline",
                "codepipeline:GetPipelineState",
                "codepipeline:ListActionTypes",
                "codepipeline:ListPipelines",
                "iam:ListRoles",
                "s3:GetBucketPolicy",
                "s3:GetObject",
                "s3:ListAllMyBuckets",
                "s3:ListBucket",
                "codedeploy:GetApplication",
                "codedeploy:GetDeploymentGroup",
                "codedeploy:ListApplications",
                "codedeploy:ListDeploymentGroups",
                "elasticbeanstalk:DescribeApplications",
                "elasticbeanstalk:DescribeEnvironments",
                "lambda:GetFunctionConfiguration",
                "lambda:ListFunctions"
            ],
            "Resource": "arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline"
        }
    ],
    "Version": "2012-10-17"
}
```

创建该策略后，在 80398EXAMPLE 账户中创建 IAM 角色并将该策略附加到该角色。在角色的信任关系中，您必须添加担任此角色的 AWS 账户。以下示例显示了一项策略，该策略允许来自 *111111111111* AWS 账户的用户代入 80398EXAMPLE 账户中定义的角色：

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

```
            "Principal": {
                "AWS": "arn:aws:iam::111111111111:root"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

以下示例显示了在 *111111111111* AWS 账户中创建的策略，该策略允许用户代入 80398EXAMPLE 账户中指定的CrossAccountPipelineViewers角色：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::80398EXAMPLE:role/CrossAccountPipelineViewers"
        }
    ]
}
```

示例 8：在管道中使用与另一个账户相关联的 AWS 资源

您可以配置策略，允许用户创建使用其他 AWS 账户资源的管道。这需要在创建管道的账户（AccountA）和已创建要在管道中使用的资源的账户（AccountB）中同时配置策略和角色。您还必须在中创建客户托管密钥 AWS Key Management Service 以用于跨账户访问。有关更多信息和 step-by-step 示例，请参阅在中 CodePipeline 创建使用其他 AWS 账户资源的管道和为存储在 Amazon S3 中的项目配置服务器端加密 CodePipeline。

以下示例显示了 AccountA 为用于存储管道构件的 S3 存储桶配置的策略。该策略授予对 AccountB 的访问权限。在以下示例中，AccountB 的 ARN 是 012ID_ACCOUNT_B。S3 存储桶的 ARN 为 codepipeline-us-east-2-1234567890。将这些 ARN 分别替换为要允许访问的 S3 存储桶和账户的 ARN：

```
{
    "Version": "2012-10-17",
    "Id": "SSEAndSSLPolicy",
    "Statement": [
        {
            "Sid": "DenyUnEncryptedObjectUploads",
            "Effect": "Deny",
```

```
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
            "Condition": {
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "aws:kms"
                }
            }
        },
        {
            "Sid": "DenyInsecureConnections",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": false
                }
            }
        },
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::012ID_ACCOUNT_B:root"
            },
            "Action": [
                "s3:Get*",
                "s3:Put*"
            ],
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*"
        },
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::012ID_ACCOUNT_B:root"
            },
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890"
        }
    ]
```

```
}
```

以下示例显示了 AccountA 配置的策略，该策略允许 AccountB 担任某个角色。必须将此策略应用于 CodePipeline (CodePipeline_Service_Role) 的服务角色。有关如何将策略应用于 IAM 中的角色的更多信息，请参阅修改角色。在以下示例中，012ID_ACCOUNT_B 是 AccountB 的 ARN：

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource": [
            "arn:aws:iam::012ID_ACCOUNT_B:role/*"
        ]
    }
}
```

以下示例显示了由 AccountB 配置并应用于 EC2 实例角色的策略。 CodeDeploy此策略授予对 AccountA 用来存储管道工件的 S3 存储桶的访问权限 (*codepipeline-us-east-2-* 1234567890)：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:Get*"
            ],
            "Resource": [
                "arn:aws:s3:::codepipeline-us-east-2-1234567890/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::codepipeline-us-east-2-1234567890"
            ]
        }
    ]
```

```
}
```

以下示例显示了在 AccountA 中创建并配置为允许 AccountB 使用该密钥的客户托管密钥的 ARN 在 AWS KMS 何处***arn:aws:kms:us-east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE***的策略：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:DescribeKey",
                "kms:GenerateDataKey*",
                "kms:Encrypt",
                "kms:ReEncrypt*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-
east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE"
            ]
        }
    ]
}
```

以下示例显示了由 AccountB 创建的 IAM 角色 (`CrossAccount_Role`) 的内联策略，该策略允许访问 AccountA 中管道所需的 CodeDeploy 操作。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codedeploy:CreateDeployment",
                "codedeploy:GetDeployment",
                "codedeploy:GetDeploymentConfig",
                "codedeploy:GetApplicationRevision",
                "codedeploy:RegisterApplicationRevision"
            ],
            "Resource": "*"
```

```
        }
    ]
}
```

以下示例显示了由 AccountB 创建的 IAM 角色 (CrossAccount_Role) 的内联策略，该策略允许访问 S3 桶，以便下载输入构件和上传输出构件：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject*",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": [
                "arn:aws:s3:::codepipeline-us-east-2-1234567890/*"
            ]
        }
    ]
}
```

有关如何编辑管道以跨账户访问资源的更多信息，请参阅步骤 2：编辑管道 。

AWS CodePipeline 基于资源的策略示例

其他服务（如 Amazon S3）还支持基于资源的权限策略。例如，您可以将策略附加到 S3 存储桶以管理对该存储桶的访问权限。尽管它 CodePipeline 不支持基于资源的策略，但它确实将要在管道中使用的工件存储在版本控制的 S3 存储桶中。

Example 为 S3 存储桶创建策略以用作项目存储 CodePipeline

您可以使用任何版本控制的 S3 存储桶作为对象存储。 CodePipeline如果您使用创建管道向导创建第一个管道，则系统将会为您创建此 S3 存储桶，以确保上传到构件存储的所有对象都已加密，并且与存储桶的连接是安全的。如果您创建自己的 S3 存储桶，则作为最佳实践，请考虑向存储桶添加以下策略或其元素。在此策略中，S3 存储桶的 ARN 是 codepipeline-us-east-2-1234567890。将此 ARN 替换为您的 S3 存储桶的 ARN：

```
{
```

```
    "Version": "2012-10-17",
    "Id": "SSEAndSSLPolicy",
    "Statement": [
        {
            "Sid": "DenyUnEncryptedObjectUploads",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
            "Condition": {
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "aws:kms"
                }
            }
        },
        {
            "Sid": "DenyInsecureConnections",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": false
                }
            }
        }
    ]
}
```

对 AWS CodePipeline 身份和访问进行故障排除

使用以下信息来帮助您诊断和修复在使用 CodePipeline 和 IAM 时可能遇到的常见问题。

主题

- 我无权在以下位置执行操作 CodePipeline
- 我无权执行 iam：PassRole
- 我是一名管理员，想允许其他人访问 CodePipeline
- 我想允许 AWS 账户之外的人访问我的 CodePipeline资源

我无权在以下位置执行操作 CodePipeline

如果 AWS Management Console 告诉您您无权执行某项操作，则必须联系管理员寻求帮助。您的管理员是指为您提供用户名和密码的那个人。

当 `mateojackson` IAM 用户尝试使用控制台查看有关管道的详细信息但不具有 `codepipeline:GetPipeline` 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 codepipeline:GetPipeline on resource: my-pipeline
```

在这种情况下，Mateo 请求他的管理员更新其策略，以允许他使用 `codepipeline:GetPipeline` 操作访问 `my-pipeline` 资源。

我无权执行 iam：PassRole

如果您收到错误消息，提示您无权执行 `iam:PassRole` 操作，则必须联系您的管理员寻求帮助。您的管理员是指为您提供用户名和密码的那个人。要求该人更新您的政策，以允许您将角色传递给 CodePipeline。

有些 AWS 服务 允许您将现有角色传递给该服务，而不是创建新的服务角色或与服务相关的角色。为此，您必须具有将角色传递到服务的权限。

当名为的 IAM 用户`marymajor`尝试使用控制台在中执行操作时，会出现以下示例错误 CodePipeline。但是，服务必须具有服务角色所授予的权限才可执行操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，Mary 请求她的管理员来更新其策略，以允许她执行 `iam:PassRole` 操作。

我是一名管理员，想允许其他人访问 CodePipeline

要允许其他人访问 CodePipeline，您必须为需要访问的人员或应用程序创建 IAM 实体（用户或角色）。它们将使用该实体的凭证访问 AWS。然后，您必须将策略附加到授予他们在中的正确权限的实体 CodePipeline。

要立即开始使用，请参阅《IAM 用户指南》中的创建您的第一个 IAM 委派用户和组。

我想允许 AWS 账户之外的人访问我的 CodePipeline资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以担任角色。对于支持基于资源的策略或访问控制列表（ACL）的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解是否 CodePipeline 支持这些功能，请参阅如何 AWS CodePipeline 与 IAM 配合使用。
- 要了解如何提供对您拥有的资源的访问权限 AWS 账户 ，请参阅 IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问权限。
- 要了解如何向第三方提供对您的资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 AWS 账户
- 要了解如何通过身份联合验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（身份联合验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户存取之间的差别，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。

CodePipeline 权限参考

在设置访问控制以及编写可附加到 IAM 身份的权限策略（基于身份的策略）时，请将下表作为参考。该表列出了每个 CodePipeline API 操作以及您可以为其授予执行该操作的权限的相应操作。对于支持资源级权限的操作，该表列出了您可以为其授予权限的 AWS 资源。请在策略的 `Action` 字段中指定这些操作。

资源级权限是允许您指定允许用户对哪些资源执行操作的权限。 AWS CodePipeline 为资源级权限提供部分支持。这意味着，对于某些 AWS CodePipeline API 调用，您可以根据必须满足的条件或允许用户使用的资源来控制何时允许用户使用这些操作。例如，您可以授予用户列出管道执行信息的权限，但只针对一个或一些特定管道。

> **Note**
>
> 资源列中列出了支持资源级权限的 API 调用所需的资源。对于不支持资源级权限的 API 调用，您可以向用户授予使用调用的权限，但是必须为策略语句的“Resource”元素指定通配符（*）。

CodePipeline API 操作和操作所需的权限

AcknowledgeJob

操作：codepipeline:AcknowledgeJob

查看有关指定作业的信息以及作业辅助角色是否已收到该作业所必需的。仅用于自定义操作。

资源： 仅支持在策略的 Resource 元素中使用通配符（*）。

AcknowledgeThirdPartyJob

操作：codepipeline:AcknowledgeThirdPartyJob

确认作业辅助角色已收到指定作业所必需的。仅用于合作伙伴操作。

资源： 仅支持在策略的 Resource 元素中使用通配符（*）。

CreateCustomActionType

操作：codepipeline:CreateCustomActionType

创建可在与该 AWS 账户关联的所有管道中使用的新自定义操作所必需的。仅用于自定义操作。

资源：

操作类型

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

CreatePipeline

操作：codepipeline:CreatePipeline

创建管道所必需的。

资源：

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

DeleteCustomActionType

操作：codepipeline:DeleteCustomActionType

将自定义操作标记为已删除所必需的。自定义操作的 PollForJobs 在该操作标记为删除后失败。仅用于自定义操作。

资源：

操作类型

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

DeletePipeline

操作：codepipeline:DeletePipeline

删除管道所必需的。

资源：

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

DeleteWebhook

操作：codepipeline:DeleteWebhook

删除 Webhook 所必需的。

资源：

Webhook

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

DeregisterWebhookWithThirdParty

操作：codepipeline:DeregisterWebhookWithThirdParty

在删除 Webhook 之前，需要移除由 CodePipeline 创建的 Webhook 与带有待检测事件的外部工具之间的连接。目前仅支持以操作类型为为为的 Webhook。 GitHub

资源：

Webhook

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

DisableStageTransition

操作：codepipeline:DisableStageTransition

防止管道中的项目过渡到管道中的下一阶段所必需的。

资源：

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

EnableStageTransition

操作：codepipeline:EnableStageTransition

允许管道中的项目过渡到管道中的某一阶段所必需的。

资源：

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

GetJobDetails

操作：codepipeline:GetJobDetails

检索有关作业的信息所必需的。仅用于自定义操作。

资源：不需要资源。

GetPipeline

操作：codepipeline:GetPipeline

检索管道的结构、阶段、操作和元数据所必需的，包括管道 ARN。

资源：

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

GetPipelineExecution

操作：codepipeline:GetPipelineExecution

检索有关管道执行的信息，包括有关项目的详细信息、管道执行 ID 以及管道的名称、版本和状态所必需的。

资源：

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

GetPipelineState

操作：codepipeline:GetPipelineState

检索有关管道状态的信息，包括阶段和操作所必需的。

资源：

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

GetThirdPartyJobDetails

操作：codepipeline:GetThirdPartyJobDetails

请求第三方操作的作业详细信息所必需的。仅用于合作伙伴操作。

资源： 仅支持在策略的 Resource 元素中使用通配符（*）。

ListActionTypes

操作：codepipeline:ListActionTypes

需要生成与您的账户关联的所有 CodePipeline 操作类型的摘要。

资源：

操作类型

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

ListPipelineExecutions

操作：codepipeline:ListPipelineExecutions

生成最新管道执行的摘要所必需的。

资源：

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

ListPipelines

操作：codepipeline:ListPipelines

生成与您的账户相关联的所有管道的摘要所必需的。

资源：

带通配符的管道 ARN（不支持管道名称级别的资源级权限）

arn:aws:codepipeline:*region*:*account*:*

ListTagsForResource

操作：codepipeline:ListTagsForResource

列出指定资源的标签所必需的。

资源：

操作类型

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

Webhook

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

ListWebhooks

操作：codepipeline:ListWebhooks

列出该区域的账户中的所有 Webhook 所必需的。

资源：

Webhook

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

PollForJobs

操作： codepipeline:PollForJobs

需要检索有关任何 CodePipeline 要处理的任务的信息。

资源：

操作类型

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

PollForThirdPartyJobs

操作：codepipeline:PollForThirdPartyJobs

确定是否存在作业辅助角色要对其采取行动的任何第三方作业所必需的。仅用于合作伙伴操作。

资源： 仅支持在策略的 Resource 元素中使用通配符 (*) 。

PutActionRevision

操作：codepipeline:PutActionRevision

需要向来源报告 CodePipeline 有关新修订的信息。

资源：

操作

arn:aws:codepipeline:*region*:*account*:*pipeline-name*/*stage-name*/*action-name*

PutApprovalResult

操作：codepipeline:PutApprovalResult

需要向报告对手动批准请求的回应 CodePipeline。有效响应是 Approved 和 Rejected。

资源：

操作

arn:aws:codepipeline:*region*:*account*:*pipeline-name*/*stage-name*/*action-name*

> **Note**
>
> 此 API 调用支持资源级权限。但是，如果您使用 IAM 控制台或 Policy Generator 来通过 "codepipeline:PutApprovalResult" 创建指定资源 ARN 的策略，则可能会遇到错误。如果您遇到错误，则可以使用 IAM 控制台中的 JSON 选项卡或 CLI 来创建策略。

PutJobFailureResult

操作：codepipeline:PutJobFailureResult

报告作业辅助角色返回给管道的作业失败状态所必需的。仅用于自定义操作。

资源： 仅支持在策略的 Resource 元素中使用通配符 (*) 。

PutJobSuccessResult

操作：codepipeline:PutJobSuccessResult

报告作业辅助角色返回给管道的作业成功状态所必需的。仅用于自定义操作。

资源： 仅支持在策略的 Resource 元素中使用通配符 (*) 。

PutThirdPartyJobFailureResult

操作：codepipeline:PutThirdPartyJobFailureResult

报告作业辅助角色返回给管道的第三方作业失败状态所必需的。仅用于合作伙伴操作。

资源： 仅支持在策略的 Resource 元素中使用通配符 (*) 。

PutThirdPartyJobSuccessResult

操作：codepipeline:PutThirdPartyJobSuccessResult

报告作业辅助角色返回给管道的第三方作业成功状态所必需的。仅用于合作伙伴操作。

资源： 仅支持在策略的 Resource 元素中使用通配符 (*) 。

PutWebhook

操作：codepipeline:PutWebhook

创建 Webhook 所必需的。

资源：

Webhook

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

RegisterWebhookWithThirdParty

操作：codepipeline:RegisterWebhookWithThirdParty

资源：

创建 Webhook 之后，配置支持的第三方以调用所生成的 Webhook URL 时所必需的

Webhook

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

RetryStageExecution

操作：codepipeline:RetryStageExecution

通过重试阶段中最后一个失败的操作来恢复管道执行所必需的。

资源：

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

StartPipelineExecution

操作：codepipeline:StartPipelineExecution

启动指定管道（具体来说，开始处理指定为管道一部分的源位置的最新提交内容）所必需的。

资源：

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

TagResource

操作：codepipeline:TagResource

标记指定资源所必需的。

资源：

操作类型

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

Webhook

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

UntagResource

操作：codepipeline:UntagResource

标记指定资源所必需的。

资源：

操作类型

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

管道

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

Webhook

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

UpdatePipeline

操作：codepipeline:UpdatePipeline

使用对管道结构所做编辑或更改更新指定管道所必需的。

资源：

管道

```
arn:aws:codepipeline:region:account:pipeline-name
```

管理 CodePipeline 服务角色

CodePipeline 服务角色配置了一个或多个策略，用于控制对管道所用 AWS 资源的访问权限。您可能需要为该角色附加更多策略，编辑附加到该角色的策略，或者在中为其他服务角色配置策略 AWS。在配置对管道的跨账户访问时，您可能还需要将策略附加到角色。

Important

修改策略语句或向角色附加其他策略可能会导致您的管道无法运行。在以任何方式修改的服务角色之前，请务必了解 CodePipeline 其含义。对服务角色进行任何更改后，确保测试管道。

Note

在控制台中，使用名称 oneClick_AWS-CodePipeline-Service_*ID-Number* 创建 2018 年 9 月之前创建的服务角色。
2018 年 9 月之后创建的服务角色使用服务角色名称格式 AWSCodePipelineServiceRole-*Region*-*Pipeline_Name*。例如，对于 eu-west-2 区域中名为 MyFirstPipeline 的管道，控制台会将角色和策略命名为 AWSCodePipelineServiceRole-eu-west-2-MyFirstPipeline。

从 CodePipeline 服务角色删除权限

您可以编辑服务角色语句以删除对不使用的资源的访问权限。例如，如果您的所有管道均不包括 Elastic Beanstalk，您可以编辑策略语句以移除授权访问 Elastic Beanstalk 资源的部分。

同样，如果您的所有管道都不包含 CodeDeploy，则可以编辑策略声明以删除授予 CodeDeploy 资源访问权限的部分：

```
{
"Action": [
    "codedeploy:CreateDeployment",
    "codedeploy:GetApplicationRevision",
    "codedeploy:GetDeployment",
    "codedeploy:GetDeploymentConfig",
    "codedeploy:RegisterApplicationRevision"
```

```
    ],
    "Resource": "*",
    "Effect": "Allow"
},
```

向 CodePipeline 服务角色添加权限

您必须使用 AWS 服务 尚未包含在默认服务角色策略语句中的权限来更新服务角色策略语句，然后才能在管道中使用该声明。

如果您用于管道的服务角色是在向 CodePipeline 添加支持之前创建的，则这一点尤其重要 AWS 服务。

下表显示了添加对其他 AWS 服务的支持的时间。

AWS 服务	CodePipeline 支持日期
AWS CloudFormation StackSets 行动	2020 年 12 月 30 日
CodeCommit 完整克隆输出工件格式	2020 年 11 月 11 日
CodeBuild 批量构建	2020 年 7 月 30 日
AWS AppConfig	2020 年 6 月 22 日
AWS Step Functions	2020 年 5 月 27 日
AWS CodeStar 连接	2019 年 12 月 18 日
`CodeDeployToECS` 操作	2018 年 11 月 27 日
Amazon ECR	2018 年 11 月 27 日
服务目录	2018 年 10 月 16 日
AWS Device Farm	2018 年 7 月 19 日
Amazon ECS	2017 年 12 月 12 日/2017 年 7 月 21 日更新了选择加入标记授权的选项
CodeCommit	2016 年 4 月 18 日

AWS 服务	CodePipeline 支持日期
AWS OpsWorks	2016 年 6 月 2 日
AWS CloudFormation	2016 年 11 月 3 日
AWS CodeBuild	2016 年 12 月 1 日
Elastic Beanstalk	首次服务发布

请按照以下步骤添加受支持服务的权限：

1. 登录 AWS Management Console 并打开 IAM 控制台，网址为 https://console.aws.amazon.com/iam/。
2. 在 IAM 角色控制台的导航窗格中，选择角色，然后从角色列表中选择您的 `AWS-CodePipeline-Service` 角色。
3. 在权限选项卡上的内联策略中，选择您的服务角色策略所在行中的编辑策略。
4. 在策略文档框中添加所需权限。

> **Note**
>
> 在您创建 IAM 策略时，请遵循授予最低权限这一标准安全建议，即仅授予执行任务所需的权限。某些 API 调用支持基于资源的权限并允许限制访问。例如，在这种情况下，要在调用 `DescribeTasks` 和 `ListTasks` 时限制权限，您可以将通配符（*）替换为资源 ARN 或包含通配符（*）的资源 ARN。有关创建策略以授予最低权限访问权限的更多信息，请参阅 https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege。

例如，为了获得 CodeCommit 支持，请在您的政策声明中添加以下内容：

```
{
  "Effect": "Allow",
  "Action": [
      "codecommit:GetBranch",
      "codecommit:GetCommit",
      "codecommit:UploadArchive",
```

```
      "codecommit:GetUploadArchiveStatus",
      "codecommit:CancelUploadArchive"
           ],
  "Resource": "resource_ARN"
},
```

如需 AWS OpsWorks 支持，请在您的政策声明中添加以下内容：

```
{
    "Effect": "Allow",
    "Action": [
        "opsworks:CreateDeployment",
        "opsworks:DescribeApps",
        "opsworks:DescribeCommands",
        "opsworks:DescribeDeployments",
        "opsworks:DescribeInstances",
        "opsworks:DescribeStacks",
        "opsworks:UpdateApp",
        "opsworks:UpdateStack"
    ],
    "Resource": "resource_ARN"
},
```

如需 AWS CloudFormation 支持，请在您的政策声明中添加以下内容：

```
{
    "Effect": "Allow",
    "Action": [
        "cloudformation:CreateStack",
        "cloudformation:DeleteStack",
        "cloudformation:DescribeStackEvents",
        "cloudformation:DescribeStacks",
        "cloudformation:UpdateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation:DeleteChangeSet",
        "cloudformation:DescribeChangeSet",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:SetStackPolicy",
        "cloudformation:ValidateTemplate",
        "iam:PassRole"
    ],
    "Resource": "resource_ARN"
```

```
},
```

请注意，该 `cloudformation:DescribeStackEvents` 权限是可选的。它允许 AWS CloudFormation 操作显示更详细的错误消息。如果您不希望资源详细信息出现在管道错误消息中，可以撤销 IAM 角色的此权限。有关更多信息，请参阅 AWS CloudFormation。

如需 CodeBuild 支持，请在您的政策声明中添加以下内容：

```
{
    "Effect": "Allow",
    "Action": [
        "codebuild:BatchGetBuilds",
        "codebuild:StartBuild"
    ],
    "Resource": "resource_ARN"
},
```

> **Note**
>
> 后来又添加了对批量构建的支持。有关向服务角色添加的批量构建权限，请参阅步骤 11。

如需 AWS Device Farm 支持，请在您的政策声明中添加以下内容：

```
{
    "Effect": "Allow",
    "Action": [
        "devicefarm:ListProjects",
        "devicefarm:ListDevicePools",
        "devicefarm:GetRun",
        "devicefarm:GetUpload",
        "devicefarm:CreateUpload",
        "devicefarm:ScheduleRun"
    ],
    "Resource": "resource_ARN"
},
```

对于 Service Catalog 支持，请将以下内容添加到策略语句中：

```
{
```

```
    "Effect": "Allow",
    "Action": [
        "servicecatalog:ListProvisioningArtifacts",
        "servicecatalog:CreateProvisioningArtifact",
        "servicecatalog:DescribeProvisioningArtifact",
        "servicecatalog:DeleteProvisioningArtifact",
        "servicecatalog:UpdateProduct"
    ],
    "Resource": "resource_ARN"
},
{
    "Effect": "Allow",
    "Action": [
        "cloudformation:ValidateTemplate"
    ],
    "Resource": "resource_ARN"
}
```

5. 对于 Amazon ECR 支持，请将以下内容添加到策略语句中：

```
{
    "Effect": "Allow",
    "Action": [
        "ecr:DescribeImages"
    ],
    "Resource": "resource_ARN"
},
```

6. 对于 Amazon ECS，以下是创建具有 Amazon ECS 部署操作的管道所需的最低权限。

```
{
    "Effect": "Allow",
    "Action": [
        "ecs:DescribeServices",
        "ecs:DescribeTaskDefinition",
        "ecs:DescribeTasks",
        "ecs:ListTasks",
        "ecs:RegisterTaskDefinition",
        "ecs:TagResource",
        "ecs:UpdateService"
    ],
    "Resource": "resource_ARN"
},
```

您可以选择加入以使用 Amazon ECS 中的标记授权。选择加入后，您必须授予以下权限：`ecs:TagResource`。有关如何选择加入以及如何确定是否需要权限和是否强制执行标记授权的更多信息，请参阅《Amazon Elastic Container Service 开发者指南》中的标记授权时间表。

您还必须添加 `iam:PassRole` 权限以使用任务 IAM 角色。有关更多信息，请参阅 Amazon ECS 任务执行 IAM 角色和任务 IAM 角色。使用以下策略文本。

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

 "arn:aws:iam::aws_account_ID:role/ecsTaskExecutionRole_or_TaskRole_name"
            ]
        }
    ]
}
```

7. 对于CodeDeployToECS操作（蓝/绿部署），以下是使用到 CodeDeploy Amazon ECS 的蓝/绿部署操作创建管道所需的最低权限。

```
{
    "Effect": "Allow",
    "Action": [
        "codedeploy:CreateDeployment",
        "codedeploy:GetDeployment",
        "codedeploy:GetApplication",
        "codedeploy:GetApplicationRevision",
        "codedeploy:RegisterApplicationRevision",
        "codedeploy:GetDeploymentConfig",
        "ecs:RegisterTaskDefinition",
        "ecs:TagResource"
    ],
    "Resource": "resource_ARN"
},
```

您可以选择加入以使用 Amazon ECS 中的标记授权。选择加入后，您必须授予以下权限：ecs:TagResource。有关如何选择加入以及如何确定是否需要权限和是否强制执行标记授权的更多信息，请参阅《Amazon Elastic Container Service 开发者指南》中的标记授权时间表。

您还必须添加 iam:PassRole 权限以使用任务 IAM 角色。有关更多信息，请参阅 Amazon ECS 任务执行 IAM 角色和任务 IAM 角色。使用以下策略文本。

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

 "arn:aws:iam::aws_account_ID:role/ecsTaskExecutionRole_or_TaskRole_name"
            ]
        }
    ]
}
```

您还可以将 ecs-tasks.amazonaws.com 添加到 iam:PassedToService 条件下的服务列表中，如本示例所示。

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "resource_ARN",
            "Condition": {
                "StringEqualsIfExists": {
                    "iam:PassedToService": [
                        "cloudformation.amazonaws.com",
                        "elasticbeanstalk.amazonaws.com",
                        "ec2.amazonaws.com",
                        "ecs-tasks.amazonaws.com"
                    ]
                }
```

```
            }
        },
```

8. 对于 AWS CodeStar 连接，需要以下权限才能使用使用连接的源（例如 Bitbucket Cloud）创建管道。

```
{
    "Effect": "Allow",
    "Action": [
        "codestar-connections:UseConnection"
    ],
    "Resource": "resource_ARN"
},
```

有关连接的 IAM 权限的更多信息，请参阅连接权限参考。

9. 对于 StepFunctions 操作，需要以下最低权限才能创建具有 Step Functions 调用操作的管道。

```
{
    "Effect": "Allow",
    "Action": [
        "states:DescribeStateMachine",
        "states:DescribeExecution",
        "states:StartExecution"
    ],
    "Resource": "resource_ARN"
},
```

10对于该AppConfig操作，以下是使用 AWS AppConfig 调用操作创建管道所需的最低权限。

```
{
    "Effect": "Allow",
    "Action": [
        "appconfig:StartDeployment",
        "appconfig:GetDeployment",
        "appconfig:StopDeployment"
    ],
    "Resource": "resource_ARN"
},
```

11要 CodeBuild 支持批量构建，请在您的政策声明中添加以下内容：

```
{
```

```
    "Effect": "Allow",
    "Action": [
        "codebuild:BatchGetBuildBatches",
        "codebuild:StartBuildBatch"
    ],
    "Resource": "resource_ARN"
},
```

12对于 AWS CloudFormation StackSets 操作，需要以下最低权限。

- 对于 CloudFormationStackSet 操作，请将以下内容添加到策略语句中：

```
{
    "Effect": "Allow",
    "Action": [
        "cloudformation:CreateStackSet",
        "cloudformation:UpdateStackSet",
        "cloudformation:CreateStackInstances",
        "cloudformation:DescribeStackSetOperation",
        "cloudformation:DescribeStackSet",
        "cloudformation:ListStackInstances"
    ],
    "Resource": "resource_ARN"
},
```

- 对于 CloudFormationStackInstances 操作，请将以下内容添加到策略语句中：

```
{
    "Effect": "Allow",
    "Action": [
        "cloudformation:CreateStackInstances",
        "cloudformation:DescribeStackSetOperation"
    ],
    "Resource": "resource_ARN"
},
```

13要 CodeCommit 支持完整克隆选项，请在您的政策声明中添加以下内容：

```
{
    "Effect": "Allow",
    "Action": [
        "codecommit:GetRepository"
    ],
    "Resource": "resource_ARN"
```

```
},
```

> **Note**
>
> 为确保您的 CodeBuild 操作可以对 CodeCommit 源使用完全克隆选项，您还必须在策略声明中添加项目 CodeBuild 服务角色的`codecommit:GitPull`权限。

14对于 Elastic Beanstalk，需要以下最低权限才能创建具有 `ElasticBeanstalk` 部署操作的管道。

```
{
    "Effect": "Allow",
    "Action": [
        "elasticbeanstalk:*",
        "ec2:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "cloudformation:*",
        "rds:*",
        "sqs:*",
        "ecs:*"
    ],
    "Resource": "resource_ARN"
},
```

> **Note**
>
> 您应将资源策略中的通配符替换为要限制访问的账户资源。有关创建策略以授予最低权限访问权限的更多信息，请参阅 https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege。

15对于要为 CloudWatch 日志配置的管道，以下是您需要向 CodePipeline 服务角色添加的最低权限。

```
{
    "Effect": "Allow",
    "Action": [
        "logs:DescribeLogGroups",
        "logs:PutRetentionPolicy"
```

```
    ],
    "Resource": "resource_ARN"
},
```

> **Note**
>
> 您应将资源策略中的通配符替换为要限制访问的账户资源。有关创建策略以授予最低权限访问权限的更多信息，请参阅 https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege。

16选择查看策略以确保策略不包含错误。当策略正确无误时，选择应用策略。

登录和监控 CodePipeline

您可以使用登录功能 AWS 来确定用户在您的账户中执行的操作以及使用的资源。日志文件显示：

- 操作的时间和日期。
- 操作的源 IP 地址。
- 由于权限不足而失败的操作。

日志记录功能在以下 AWS 服务中可用：

- AWS CloudTrail 可用于记录由或代表发起 AWS 的 API 调用和相关事件 AWS 账户。有关更多信息，请参阅 使用记录 CodePipeline API 调用 AWS CloudTrail。
- Amazon CloudWatch Events 可用于监控您的 AWS Cloud 资源和您运行的应用程序 AWS。您可以根据自己定义的指标在 Amazon Ev CloudWatch ents 中创建提醒。有关更多信息，请参阅 监视 CodePipeline 事件。

合规性验证 AWS CodePipeline

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。 AWS 提供了以下资源来帮助实现合规性：

- 安全与合规性快速入门指南 — 这些部署指南讨论了架构注意事项，并提供了在这些基础上 AWS 部署以安全性和合规性为重点的基准环境的步骤。
- 在 A@@ mazon Web Services 上构建 HIPAA 安全与合规性 — 本白皮书描述了各公司如何使用 AWS 来创建符合 HIPAA 资格的应用程序。

> **Note**
>
> 并非所有 AWS 服务 人都符合 HIPAA 资格。有关更多信息，请参阅符合 HIPAA 要求的服务参考。

- AWS 合AWS 规资源 — 此工作簿和指南集可能适用于您的行业和所在地区。
- AWS 客户合规指南 — 从合规角度了解责任共担模式。这些指南总结了保护的最佳实践， AWS 服务并将指南映射到跨多个框架（包括美国国家标准与技术研究院 (NIST)、支付卡行业安全标准委员会 (PCI) 和国际标准化组织 (ISO)）的安全控制。
- 使用AWS Config 开发人员指南中的规则评估资源 — 该 AWS Config 服务评估您的资源配置在多大程度上符合内部实践、行业准则和法规。
- AWS Security Hub— 这 AWS 服务 提供了您内部安全状态的全面视图 AWS。Security Hub 通过安全控件评估您的 AWS 资源并检查其是否符合安全行业标准和最佳实践。有关受支持服务及控件的列表，请参阅 Security Hub 控件参考。
- Amazon GuardDuty — 它通过监控您的 AWS 账户环境中是否存在可疑和恶意活动，来 AWS 服务检测您的工作负载、容器和数据面临的潜在威胁。 GuardDuty 通过满足某些合规性框架规定的入侵检测要求，可以帮助您满足各种合规性要求，例如 PCI DSS。
- AWS Audit Manager— 这 AWS 服务 可以帮助您持续审计 AWS 使用情况，从而简化风险管理以及对法规和行业标准的合规性。

韧性在 AWS CodePipeline

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设施相比，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 CodePipeline

作为一项托管服务 AWS CodePipeline ，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 CodePipeline 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，建议使用 TLS 1.3。
- 具有完全向前保密 (PFS) 的密码套件，例如 DHE（临时 Diffie-Hellman）或 ECDHE（临时椭圆曲线 Diffie-Hellman）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 委托人关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service（AWS STS）生成临时安全凭证来对请求进行签名。

安全最佳实操

CodePipeline 提供了许多安全功能，供您在制定和实施自己的安全策略时考虑。以下最佳实践是一般指导原则，并不代表完整安全解决方案。这些最佳实践可能不适合环境或不满足环境要求，请将其视为有用的考虑因素而不是惯例。

您可以对连接到管道的源存储库使用加密和身份验证。以下是安全 CodePipeline 的最佳实践：

- 如果要创建需要包含密钥（如令牌或密码）的管道或操作配置，请不要直接在操作配置中输入密钥，也不要输入管道级别或 AWS CloudFormation 配置中定义的变量的默认值，因为这些信息将显示在日志中。请使用 Secrets Manager 设置和存储密钥，然后在管道和操作配置中使用引用的密钥，如AWS Secrets Manager 用于跟踪数据库密码或第三方 API 密钥中所述。
- 如果您创建使用 S3 源存储桶的管道，请 CodePipeline 通过管理为存储在 Amazon S3 中的项目配置服务器端加密 AWS KMS keys，如中所为存储在 Amazon S3 中的项目配置服务器端加密 CodePipeline述。
- 如果您使用 Jenkins 操作提供程序，则当您将 Jenkins 构建提供程序用于管道的构建或测试操作时，应在 EC2 实例上安装 Jenkins 并配置单独的 EC2 实例配置文件。确保实例配置文件仅授予 Jenkins 执行项目任务所需的 AWS 权限，例如从 Amazon S3 检索文件。要了解如何为您的 Jenkins 实例配置文件创建角色，请参阅创建 IAM 角色以用于 Jenkins 集成中的步骤。

AWS CodePipeline 命令行参考

使用 AWS CodePipeline 命令时，请使用此参考资料，并作为《AWS CLI 用户指南》和《AWS CLI 参考资料》中记录的信息的补充。

在使用之前 AWS CLI，请务必完成中的先决条件入门 CodePipeline。

要查看所有可用 CodePipeline 命令的列表，请运行以下命令：

```
aws codepipeline help
```

要查看有关特定 CodePipeline 命令的信息，请运行以下命令，其中 co *mmand-n* ame 是下面列出的命令之一的名称（例如）：create-pipeline

```
aws codepipeline command-name help
```

要开始学习如何使用 CodePipeline 扩展中的命令 AWS CLI，请转到以下一个或多个部分：

- 创建自定义操作
- 创建管道（CLI）
- 删除管道 (CLI)
- 禁用或启用过渡 (CLI)
- 查看管道详细信息和历史记录（CLI）
- 重试失败的操作 (CLI)
- 手动启动管道（CLI）
- 编辑管道（AWS CLI）

您还可以在 CodePipeline 教程 中查看有关如何使用大多数这些命令的示例。

CodePipeline 管道结构参考

默认情况下，您成功创建的任何管道都 AWS CodePipeline 具有有效的结构。但是，如果您手动创建或编辑 JSON 文件来创建管道或更新管道 AWS CLI，则可能会无意中创建了一个无效的结构。以下参考可帮助您更好地了解管道结构的要求以及如何排查问题。请参阅 中的配额 AWS CodePipeline 中适用于所有管道的约束。

主题

中的有效操作类型和提供者 CodePipeline

管道结构格式用于在管道中构建操作和阶段。操作类型由操作类别和提供方类型组成。

以下是中的有效操作类别 CodePipeline：

- 来源
- 构建
- 测试
- 部署
- 审批
- 调用

每个操作类别都有一组指定的提供方。每个操作提供方（比如 Amazon S3）都有一个提供方名称（比如 `S3`），您必须在管道结构中操作类别的 `Provider` 字段中使用该名称。

管道结构中的操作类别部分的 `Owner` 字段有三个有效值：`AWS`、`ThirdParty` 和 `Custom`。

要查找操作提供方的提供方名称和所有者信息，请参阅操作结构参考或每个操作类型的输入和输出构件数量。

此表按操作类型列出了有效的提供方。

> **Note**
>
> 有关 Bitbucket Cloud GitHub、、En GitHub terprise Server 或 GitLab .com 操作，请参阅CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、GitLab .com 和 GitLab 自我管理操作操作参考主题。

按操作类型列出的有效操作提供方

操作类别	有效操作提供方	操作参考
来源	Amazon S3	Amazon S3 源操作
	Amazon ECR	Amazon ECR
	CodeCommit	CodeCommit
	CodeStarSourceConnection（适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器或 GitLab .com 操作）	CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、 GitLab .com 和 GitLab 自我管理操作
构建	CodeBuild	AWS CodeBuild
	自定义 CloudBees	每个操作类型的输入和输出构件数量
	自定义 Jenkins	每个操作类型的输入和输出构件数量

操作类别	有效操作提供方	操作参考
	自定义 TeamCity	每个操作类型的输入和输出构件数量
测试	CodeBuild	AWS CodeBuild
	AWS Device Farm	每个操作类型的输入和输出构件数量
	ThirdParty GhostInspector	每个操作类型的输入和输出构件数量
	自定义 Jenkins	每个操作类型的输入和输出构件数量
	ThirdParty 微焦 StormRunner 负载	每个操作类型的输入和输出构件数量
	ThirdParty Nouvola	每个操作类型的输入和输出构件数量
部署	Amazon S3	Amazon S3 部署操作
	AWS CloudFormation	AWS CloudFormation
	AWS CloudFormation StackSets（包括`CloudFormationStackSet` 和`CloudFormationStackInstances` 操作）	AWS CloudFormation StackSets

操作类别	有效操作提供方	操作参考
	CodeDeploy	每个操作类型的输入和输出构件数量
	Amazon ECS	每个操作类型的输入和输出构件数量
	Amazon ECS（蓝色/绿色）（这是 CodeDeployToECS 操作）	每个操作类型的输入和输出构件数量
	Elastic Beanstalk	每个操作类型的输入和输出构件数量
	AWS AppConfig	AWS AppConfig
	AWS OpsWorks	每个操作类型的输入和输出构件数量
	服务目录	每个操作类型的输入和输出构件数量
	Amazon Alexa	每个操作类型的输入和输出构件数量
	自定义 XebiaLabs	每个操作类型的输入和输出构件数量
审批	手动	每个操作类型的输入和输出构件数量

操作类别	有效操作提供方	操作参考
调用	AWS Lambda	AWS Lambda
	AWS Step Functions	AWS Step Functions

中的 CodePipeline 某些操作类型仅在部分 AWS 区域可用。某一 AWS 地区可能有操作类型可用，但该操作类型的 AWS 提供者不可用。

有关各个操作提供方的更多信息，请参阅与动 CodePipeline 作类型的集成。

以下部分提供了每种操作类型的提供方信息和配置属性的示例。

中的管道和舞台结构要求 CodePipeline

两阶段管道具有以下基本结构：

```
{
    "roleArn": "An IAM ARN for a service role, such as arn:aws:iam::80398EXAMPLE:role/
CodePipeline_Service_Role",
    "stages": [
        {
            "name": "SourceStageName",
            "actions": [
            ... See ########  CodePipeline ...
            ]
        },
        {
            "name": "NextStageName",
            "actions": [
            ... See ########  CodePipeline ...
            ]
        }
    ],
    "artifactStore": {
        "type": "S3",
        "location": "The name of the Amazon S3 bucket automatically generated for you
 the first time you create a pipeline
            using the console, such as codepipeline-us-east-2-1234567890, or any Amazon
 S3 bucket you provision for this purpose"
```

```
    },
    "name": "YourPipelineName",
    "version": 1
}
```

管道结构具有以下要求：

- 一个管道必须包含至少两个阶段。
- 管道的第一阶段必须包含至少一个源操作。它只能包含源操作。
- 只有管道的第一阶段才可包含源操作。
- 每个管道中至少有一个阶段必须包含不是源操作的操作。
- 管道中的所有阶段名称都必须唯一。
- 无法在控制台中编辑舞 CodePipeline 台名称。如果您使用编辑阶段名称 AWS CLI，并且该阶段包含带有一个或多个秘密参数（例如 OAuth 令牌）的操作，则不会保留这些秘密参数的值。您必须手动输入参数的值（在返回的 JSON 中，这些值由四个星号掩盖 AWS CLI），并将其包含在 JSON 结构中。
- 该`artifactStore`字段包含管道的工件存储桶类型和位置，所有操作都在同一个 AWS 区域中。如果您在与您的管道不同的区域中添加操作，则该`artifactStores`映射将用于列出执行操作的每个 AWS 区域的构件存储桶。当您创建或编辑管道时，管道区域中必须有构件存储桶，然后每个您计划执行操作的区域必须有一个构件存储桶。

  以下示例显示了一个管道的基本结构，该管道具有跨区域操作并使用 `artifactStores` 参数：

```
    "pipeline": {
        "name": "YourPipelineName",
        "roleArn": "CodePipeline_Service_Role",
        "artifactStores": {
            "us-east-1": {
                "type": "S3",
                "location": "S3 artifact bucket name, such as codepipeline-us-
east-1-1234567890"
            },
            "us-west-2": {
                "type": "S3",
                "location": "S3 artifact bucket name, such as codepipeline-us-
west-2-1234567890"
            }
        },
        "stages": [
```

```
            {

...
```

- 管道元数据字段与管道结构截然不同，无法进行编辑。在更新管道时，将自动更改 updated 元数据字段中的日期。
- 在编辑或更新管道时，无法更改管道名称。

> **Note**
>
> 如果要重命名现有的管道，您可以使用 CLI `get-pipeline` 命令生成包含管道结构的 JSON 文件。然后，您可以使用 CLI `create-pipeline` 命令创建具有该结构的新管道，并为其指定新名称。

每次更新管道时，都会自动生成和更新管道的版本号。

中的操作结构要求 CodePipeline

操作具有以下高级结构：

```
[
                {
                   "inputArtifacts": [
                        An input artifact structure, if supported for the action
 category
                    ],
                   "name": "ActionName",
                   "region": "Region",
                   "namespace": "source_namespace",
                   "actionTypeId": {
                        "category": "An action category",
                        "owner": "AWS",
                        "version": "1"
                        "provider": "A provider type for the action category",
                   },
                   "outputArtifacts": [
                        An output artifact structure, if supported for the action
 category
                   ],
                   "configuration": {
```

```
                    Configuration details appropriate to the provider type
                },
                "runOrder": A positive integer that indicates the run order within
the stage,
            }
        ]
```

有关适用于提供方类型的示例 configuration 详细信息的列表，请参阅[按提供方类型列出的配置详细信息](#)。

操作结构具有以下要求：

- 阶段中的所有操作名称都必须是唯一的。
- 操作的输入构件必须与前一操作中声明的输出构件完全相符。例如，如果前一操作包含以下声明：

  ```
  "outputArtifacts": [
      {
      "MyApp"
      }
  ],
  ```

  并且没有其他输出项目，则后一操作的输入项目必须为：

  ```
  "inputArtifacts": [
      {
      "MyApp"
      }
  ],
  ```

  这适用于所有操作，无论它们处于同一阶段还是后续阶段，但输入项目不必是提供输出项目的操作的严格意义上的下一个操作。并行操作可以声明不同的输出构件包，这些包又由不同的后续操作使用。
- 输出构件名称在管道内必须唯一。例如，一个管道可以包括两个操作，一个具有名为 "MyApp" 的输出项目，另一个具有名为 "MyBuiltApp" 的输出项目。但是，管道不能包含两个都具有名为 "MyApp" 的输出项目的操作。
- 跨区域操作使用 Region 字段指定要创建操作的 AWS 区域。为此操作创建的 AWS 资源必须在region字段中提供的相同区域中创建。无法创建以下操作类型的跨区域操作：
  - 源操作
  - 按第三方提供方列出的操作

  - 按自定义提供方列出的操作
- 操作可以使用变量进行配置。您可以使用 namespace 字段为执行变量设置命名空间和变量信息。有关执行变量和操作输出变量的参考信息，请参阅Variables。
- 对于当前受支持的所有操作类型，唯一有效的所有者字符串为 AWS、ThirdParty 或 Custom。如需了解更多信息，请参阅 CodePipeline API 参考。
- 操作的默认 runOrder 值为 1。值必须是正整数 (自然数)。不能使用分数、小数、负数或零。要指定一个操作序列，请对序列中的第一个操作使用最小的数字，然后对其余的每个操作使用逐渐递增的数字。要指定并行操作，请对要并行运行的每个操作使用同一整数。在控制台中，您可以通过在要运行操作的阶段中选择该级别的添加操作组来指定操作串行序列，也可以通过选择添加操作来指定并行序列。操作组是指同一级别中一个或多个操作的运行顺序。

  例如，如果您希望一个阶段中的三个操作依次运行，则应将第一个操作的 runOrder 值指定为 1，将第二个操作的 runOrder 值指定为 2，将第三个操作的 runOrder 值指定为 3。但是，如果您希望第二个和第三个操作并行运行，则应将第一个操作的 runOrder 值指定为 1，将第二个和第三个操作的 runOrder 值均指定为 2。

  > **Note**
  >
  > 顺序操作的编号不必十分严格。例如，如果您的序列中有三个操作并决定删除第二个操作，则不需要对第三个操作的 runOrder 值重新编号。因为该操作的 runOrder 值 (3) 大于第一个操作的 runOrder 值 (1)，所以它将在此阶段中的第一个操作之后连续运行。

- 当您使用 Amazon S3 桶作为部署位置时，您还可以指定对象键。对象键可以是文件名（对象）或前缀（文件夹路径）和文件名的组合。您可以使用变量来指定您希望管道使用的位置名称。Amazon S3 部署操作支持在 Amazon S3 对象键中使用以下变量。

  使用 Amazon S3 中的变量

Variable	控制台输入的示例	输出
datetime	js-application/{datetime}.zip	此格式的 UTC 时间戳：<YYYY>-<MM>-DD>_<HH>-<MM>-<SS> 例如： js-application/2019-01-10_07-39-57.zip

Variable	控制台输入的示例	输出
uuid	js-application/{uuid}.zip	UUID 是全局唯一标识符，保证不同于任何其他标识符。此格式的 UUID（十六进制格式中的所有位数）：<8 位数> – <4 位数> –4 位数> – <4 位数> – <12 位数> 例如： js-application/54a60075-b96a-4bf3-9013-db3a9EXAMPLE.zip

- 以下是以下`actionTypeId`类别的有效类别 CodePipeline：
  - `Source`
  - `Build`
  - `Approval`
  - `Deploy`
  - `Test`
  - `Invoke`

  此处提供了一些提供方类型和配置选项。
- 操作类别的有效提供方类型因类别而异。例如，对于源操作类型，有效的提供方类型为 `S3`、`GitHub`、`CodeCommit` 或 `Amazon ECR`。此示例显示了使用 `S3` 提供方的源操作的结构：

```
"actionTypeId": {
  "category": "Source",
  "owner": "AWS",
  "version": "1",
  "provider": "S3"},
```

- 每个操作都必须具有有效的操作配置，这取决于该操作的提供方类型。下表列出了每个有效提供方类型所需的操作配置元素：

提供方类型的操作配置属性

提供方名称	操作类型中的提供方名称	配置属性	是否是必需的属性？
Amazon S3（部署操作提供方）	有关更多信息，包括与 Amazon S3 部署操作参数相关的示例，请参阅 Amazon S3 部署操作。		
Amazon S3（源操作提供方）	有关更多信息，包括与 Amazon S3 源操作参数相关的示例，请参阅 Amazon S3 源操作。		
Amazon ECR	有关更多信息，包括与 Amazon ECR 参数相关的示例，请参阅 Amazon ECR。		
CodeCommit	有关更多信息，包括与 CodeCommit 参数相关的示例，请参阅CodeCommit。		
GitHub	有关更多信息，包括与 GitHub 参数相关的示例，请参阅GitHub 版本 1 源操作结构参考。		
AWS CloudFormation	有关更多信息，包括与 AWS CloudFormation 参数相关的示例，请参阅AWS CloudFormation。		
CodeBuild	有关 CodeBuild 参数的更多说明和示例，请参阅AWS CodeBuild。		
CodeDeploy	有关 CodeDeploy 参数的更多说明和示例，请参阅AWS CodeDeploy。		
AWS Device Farm	有关 AWS Device Farm 参数的更多说明和示例，请参阅AWS Device Farm。		
AWS Elastic Beanstalk	`ElasticBeanstalk`	`ApplicationName`	必需
		`EnvironmentName`	必需

提供方名称	操作类型中的提供方名称	配置属性	是否是必需的属性？
AWS Lambda	有关更多信息，包括与 AWS Lambda 参数相关的示例，请参阅AWS Lambda。		
AWS OpsWorks Stacks	`OpsWorks`	`Stack`	必需
		`Layer`	可选
		`App`	必需
Amazon ECS	有关与 Amazon ECS 参数相关的更多说明和示例，请参阅 Amazon Elastic Container Service。		
Amazon ECS 和 CodeDeploy（蓝色/绿色）	有关 Amazon ECS 和 CodeDeploy 蓝/绿参数的更多描述和示例，请参阅。Amazon 弹性容器服务和 CodeDeploy 蓝绿色		
服务目录	`ServiceCatalog`	`TemplateFilePath`	必需
		`ProductVersionName`	必需
		`ProductType`	必需
		`ProductVersionDescription`	可选
		`ProductId`	必需
Alexa Skills Kit	`AlexaSkillsKit`	`ClientId`	必需
		`ClientSecret`	必需
		`RefreshToken`	必需
		`SkillId`	必需

提供方名称	操作类型中的提供方名称	配置属性	是否是必需的属性？
Jenkins	你在 Jenkins CodePipeline 插件中提供的操作名称（例如，*MyJenkinsProviderName*）	`ProjectName`	必需
手动审批	`Manual`	`CustomData`	可选
		`ExternalEntityLink`	可选
		`NotificationArn`	可选

主题

- 每个操作类型的输入和输出构件数量
- PollForSourceChanges 参数的默认设置
- 按提供方类型列出的配置详细信息

每个操作类型的输入和输出构件数量

根据操作类型，您可以具有以下数量的输入和输出项目：

构件的操作类型约束

所有者	操作的类型	提供商	有效的输入构件数	有效的输出构件数
`AWS`	来源	Amazon S3	0	1
`AWS`	来源	CodeCommit	0	1
`AWS`	来源	Amazon ECR	0	1
`ThirdParty`	来源	GitHub	0	1

所有者	操作的类型	提供商	有效的输入构件数	有效的输出构件数
AWS	构建	CodeBuild	1 到 5	0 到 5
AWS	测试	CodeBuild	1 到 5	0 到 5
AWS	测试	AWS Device Farm	1	0
AWS	审批	手动	0	0
AWS	部署	Amazon S3	1	0
AWS	部署	AWS CloudFormation	0 到 10	0 到 1
AWS	部署	CodeDeploy	1	0
AWS	部署	AWS Elastic Beanstalk	1	0
AWS	部署	AWS OpsWorks Stacks	1	0
AWS	部署	Amazon ECS	1	0
AWS	部署	服务目录	1	0
AWS	调用	AWS Lambda	0 到 5	0 到 5
ThirdParty	部署	Alexa Skills Kit	1 到 2	0
Custom	构建	Jenkins	0 到 5	0 到 5
Custom	测试	Jenkins	0 到 5	0 到 5
Custom	任意受支持的类别	与自定义操作中指定的一样	0 到 5	0 到 5

PollForSourceChanges 参数的默认设置

PollForSourceChanges 参数默认值由用于创建管道的方法确定，如下表所述。许多情况下，PollForSourceChanges 参数默认为 true，并且必须被禁用。

当 PollForSourceChanges 参数默认为 true 时，应执行以下操作：

- 将 PollForSourceChanges 参数添加到 JSON 文件或 AWS CloudFormation 模板。
- 创建变更检测资源（CloudWatch 事件规则，如果适用）。
- 将 PollForSourceChanges 参数设置为 false。

> **Note**
>
> 如果您创建了 CloudWatch 事件规则或 webhook，则必须将该参数设置为 false 以避免多次触发管道。

PollForSourceChanges 参数不可用于 Amazon ECR 源操作。

- PollForSourceChanges 参数默认值

来源	创建方法	示例“配置”JSON 结构输出
CodeCommit	使用控制台创建管道（更改检测资源也由控制台创建）。该参数显示在管道结构输出中，并默认为 false。	`BranchName": "main", "PollForSourceChanges": "false", "RepositoryName": "my-repo"`
	管道是使用 CLI 或创建的 AWS CloudFormation，PollForSourceChanges 参数不显示在 JSON 输出中，但它设置为 true .[2]	`BranchName": "main", "RepositoryName": "my-repo"`
Amazon S3	使用控制台创建管道（更改检测资源也由控制台创建）。该参数显示在管道结构输出中，并默认为 false。	`"S3Bucket": "my-bucket", "S3ObjectKey": "object.zip",`

来源	创建方法	示例“配置”JSON 结构输出
		`"PollForSourceChanges" : "false"`
	管道是使用 CLI 或创建的 AWS CloudFormation，`PollForSourceChanges` 参数不显示在 JSON 输出中，但它设置为 `true`.[2]	`"S3Bucket": "my-bucket", "S3ObjectKey": "object.zip"`
GitHub	使用控制台创建管道（更改检测资源也由控制台创建）。该参数显示在管道结构输出中，并默认为 `false`。	`"Owner": "MyGitHubAccountName ", "Repo": " MyGitHubRepositoryName " "PollForSourceChanges": "false", "Branch": " main" "OAuthToken": " ****"`
	管道是使用 CLI 或创建的 AWS CloudFormation，`PollForSourceChanges` 参数不显示在 JSON 输出中，但它设置为 `true`.[2]	`"Owner": "MyGitHubAccountName ", "Repo": "MyGitHubRepositoryName ", "Branch": " main", "OAuthToken": " ****"`
	[2] 如果已在某个时候将 `PollForSourceChanges` 添加到 JSON 结构或 AWS CloudFormation 模板中，它将如下所示： `"PollForSourceChanges": "true",` [3] 有关适用于每个源提供方的更改检测资源的信息，请参阅更改检测方法。	

按提供方类型列出的配置详细信息

本节列出每个操作提供方的有效 `configuration` 参数。

以下示例为使用 Service Catalog 的部署操作显示了一个有效的配置，用于在控制台中创建的管道（没有单独的配置文件）：

```
"configuration": {
  "TemplateFilePath": "S3_template.json",
  "ProductVersionName": "devops S3 v2",
  "ProductType": "CLOUD_FORMATION_TEMPLATE",
  "ProductVersionDescription": "Product version description",
  "ProductId": "prod-example123456"
}
```

以下示例为使用 Service Catalog 的部署操作显示了一个有效的配置，用于在控制台中创建的管道（有单独的 `sample_config.json` 配置文件）：

```
"configuration": {
  "ConfigurationFilePath": "sample_config.json",
  "ProductId": "prod-example123456"
}
```

以下示例显示使用 Alexa Skills Kit 的部署操作的有效配置：

```
"configuration": {
  "ClientId": "amzn1.application-oa2-client.aadEXAMPLE",
  "ClientSecret": "****",
  "RefreshToken": "****",
  "SkillId": "amzn1.ask.skill.22649d8f-0451-4b4b-9ed9-bfb6cEXAMPLE"
}
```

以下示例显示手动审批的有效配置：

```
"configuration": {
  "CustomData": "Comments on the manual approval",
  "ExternalEntityLink": "http://my-url.com",
  "NotificationArn": "arn:aws:sns:us-west-2:12345EXAMPLE:Notification"
}
```

操作结构参考

此部分仅为操作配置提供参考。有关管道结构的概念性介绍，请参阅 CodePipeline 管道结构参考。

中的每个操作提供者在工作流结构中 CodePipeline 使用一组必填和可选的配置字段。本节按操作提供方提供了以下参考信息：

- 管道结构操作块中包含的 `ActionType` 字段的有效值，如 `Owner` 和 `Provider`。
- 管道结构操作部分中包含的 `Configuration` 参数（必需和可选）的描述和其他参考信息。
- 有效的示例 JSON 和 YAML 操作字段。

此部分会定期进行更新，添加更多操作提供方。参考信息当前对下列操作提供方可用：

主题

- Amazon ECR
- Amazon 弹性容器服务和 CodeDeploy 蓝绿色
- Amazon Elastic Container Service
- Amazon S3 部署操作
- Amazon S3 源操作
- AWS AppConfig
- AWS CloudFormation
- AWS CloudFormation StackSets
- AWS CodeBuild
- CodeCommit
- AWS CodeDeploy
- CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、 GitLab .com 和 GitLab 自我管理操作
- AWS Device Farm
- AWS Lambda
- Snyk 操作结构参考
- AWS Step Functions

Amazon ECR

在新映像推送到 Amazon ECR 存储库时触发管道。此操作提供了一个映像定义文件，该文件引用推送到 Amazon ECR 的映像的 URI。此源操作通常与其他源操作（例如）结合使用 CodeCommit，以便为所有其他源对象提供源位置。有关更多信息，请参阅 教程：使用 Amazon ECR 源和 ECS 目标部署创建管道 CodeDeploy 。

当您使用控制台创建或编辑管道时， CodePipeline 会创建一个 CloudWatch 事件规则，该规则将在存储库发生更改时启动您的管道。

您必须事先创建 Amazon ECR 存储库并推送映像，然后才能通过 Amazon ECR 操作连接管道。

主题

操作类型

- 类别：`Source`
- 拥有者：`AWS`
- 提供方：`ECR`
- 版本：`1`

配置参数

RepositoryName

必需：是

向其中推送映像的 Amazon ECR 存储库的名称。

ImageTag

必需：否

为映像使用的标签。

> **Note**
>
> 如果未指定 `ImageTag` 的值，则该值默认为 `latest`。

输入构件

- 构件数：`0`
- 描述：输入构件不适用于此操作类型。

输出构件

- 构件数：`1`
- 描述：此操作生成包含 `imageDetail.json` 文件的构件，该文件包含触发管道执行的映像的 URI。有关 `imageDetail.json` 文件的信息，请参阅适用于 Amazon ECS 蓝绿部署的 imageDetail.json 文件。

输出变量

配置后，此操作会生成变量，该变量可由管道中下游操作的操作配置引用。此操作生成的变量可视为输出变量，即使操作没有命名空间也是如此。您可以使用命名空间配置操作，以使这些变量可用于下游操作的配置。

有关更多信息，请参阅 Variables。

RegistryId

与包含存储库的注册表关联的 AWS 账户 ID。

RepositoryName

向其中推送映像的 Amazon ECR 存储库的名称。

ImageTag

为映像使用的标签。

ImageDigest

映像清单的 sha256 摘要。

ImageURI

映像的 URI。

操作声明（Amazon ECR 示例）

YAML

```
Name: Source
Actions:
  - InputArtifacts: []
    ActionTypeId:
      Version: '1'
      Owner: AWS
      Category: Source
      Provider: ECR
    OutputArtifacts:
      - Name: SourceArtifact
    RunOrder: 1
    Configuration:
      ImageTag: latest
      RepositoryName: my-image-repo

    Name: ImageSource
```

JSON

```
{
    "Name": "Source",
    "Actions": [
        {
            "InputArtifacts": [],
            "ActionTypeId": {
                "Version": "1",
                "Owner": "AWS",
```

```
                "Category": "Source",
                "Provider": "ECR"
            },
            "OutputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "RunOrder": 1,
            "Configuration": {
                "ImageTag": "latest",
                "RepositoryName": "my-image-repo"
            },
            "Name": "ImageSource"
        }
    ]
},
```

另请参阅

下列相关资源在您使用此操作的过程中会有所帮助。

- 教程：使用 Amazon ECR 源和 ECS 目标部署创建管道 CodeDeploy — 本教程提供了一个示例应用程序规范文件以及示例 CodeDeploy 应用程序和部署组，用于创建管道，其中包含部署到 Amazon ECS 实例的 Amazon ECR 源。 CodeCommit

Amazon 弹性容器服务和 CodeDeploy 蓝绿色

您可以在中配置一个使用蓝/ AWS CodePipeline 绿部署部署容器应用程序的管道。在蓝绿部署中，您可以将应用程序新版本与旧版本一起启动，并在将流量重新路由到新版本之前对其进行测试。您还可以监控部署流程并且在存在问题的情况下迅速回滚。

已完成的管道会检测您的图像或任务定义文件的更改，并使用 CodeDeploy 这些更改将流量路由和部署到 Amazon ECS 集群和负载均衡器。 CodeDeploy 在您的负载均衡器上创建一个新的侦听器，该侦听器可以通过特殊端口定位您的新任务。您也可以将管道配置为使用存储您的 Amazon ECS 任务定义的源位置，例如存储 CodeCommit 库。

在创建管道之前，您必须已经创建 Amazon ECS CodeDeploy 资源、资源以及负载均衡器和目标组。您必须已将图像标记并存储在图像存储库中，并将任务定义和 AppSpec 文件上传到文件存储库。

> **Note**
>
> 本主题介绍了 Amazon ECS 到 CodeDeploy 蓝/绿部署操作。 CodePipeline有关中 Amazon ECS 标准部署操作的参考信息 CodePipeline，请参阅Amazon Elastic Container Service。

主题

操作类型

- 类别：`Deploy`
- 拥有者：`AWS`
- 提供方：`CodeDeployToECS`
- 版本：`1`

配置参数

ApplicationName

必需：是

中的应用程序名称 CodeDeploy。在创建管道之前，您必须已经在中创建了应用程序 CodeDeploy。

DeploymentGroupName

必需：是

为您为 CodeDeploy 应用程序创建的 Amazon ECS 任务集指定的部署组。在创建管道之前，您必须已经在中创建了部署组 CodeDeploy。

TaskDefinitionTemplateArtifact

必需：是

用于为部署操作提供任务定义文件的输入构件的名称。这通常为源操作的输出构件的名称。使用控制台时，源操作输出构件的默认名称为 SourceArtifact。

AppSpecTemplateArtifact

必需：是

为部署操作提供 AppSpec 文件的输入对象的名称。此值会在管道运行时进行更新。这通常为源操作的输出构件的名称。使用控制台时，源操作输出构件的默认名称为 SourceArtifact。对于TaskDefinition在 AppSpec 文件中，您可以保留<TASK_DEFINITION>占位符文本，如下所示。

AppSpecTemplatePath

必需：否

存储在管道源 AppSpec 文件位置（例如您的管道存储 CodeCommit 库）中的文件的文件名。默认文件名为 appspec.yaml。如果您的 AppSpec 文件名相同，并且存储在文件存储库的根级别，则无需提供文件名。如果路径不是默认路径，请输入路径和文件名。

TaskDefinitionTemplatePath

必需：否

存储在管道文件源位置（例如您的管道存储 CodeCommit 库）中的任务定义的文件名。默认文件名为 taskdef.json。如果您的任务定义文件同名且存储在文件存储库的根级别，则无需提供文件名。如果路径不是默认路径，请输入路径和文件名。

图片 <Number>ArtifactName

必需：否

用于为部署操作提供映像的输入构件的名称。这通常是映像存储库的输出构件，如 Amazon ECR 源操作的输出。

<Number> 的可用值为 1 到 4。

图片 <Number>ContainerName

必需：否

映像存储库（如 Amazon ECR 源存储库）中可用映像的名称。

<Number> 的可用值为 1 到 4。

输入构件

- 构件数：1 to 5
- 描述：该CodeDeployToECS操作首先在源文件存储库中查找任务定义文件和文件，然后在图像存储库中查找图像，然后动态生成任务定义的新修订版，最后运行 AppSpec 命令将任务集和容器部署到集群。 AppSpec

  CodeDeployToECS 操作会查找将映像 URI 映射到映像的 imageDetail.json 文件。当您向 Amazon ECR 映像存储库提交更改时，管道 ECR 源操作会为该提交创建 imageDetail.json 文件。在未自动执行操作的情况下，您还可以为管道手动添加 imageDetail.json 文件。有关 imageDetail.json 文件的信息，请参阅适用于 Amazon ECS 蓝绿部署的 imageDetail.json 文件。

  CodeDeployToECS 操作会动态生成任务定义的新修订版。在此阶段，该操作会将任务定义文件中的占位符替换为从 imageDetail.json 文件检索到的映像 URI。例如，如果您将 IMAGE1_NAM E 设置为 Image1 ContainerName 参数，则应在任务定义文件中将占位符指定<IMAGE1_NAME>为图像字段的值。在这种情况下， CodeDeployToECS 操作会将占位符替换<IMAGE1_NAME>为从您指定为 Image1 的构件中的 ImageDetail.json 中检索到的实际图像 URI。ArtifactName

  对于任务定义更新，该 CodeDeploy AppSpec.yaml文件包含TaskDefinition属性。

  ```
  TaskDefinition: <TASK_DEFINITION>
  ```

  创建新任务定义后，CodeDeployToECS 操作将会更新此属性。

  对于 TaskDefinition 字段的值，占位符文本必须为 <TASK_DEFINITION>。CodeDeployToECS 操作会将此占位符替换为动态生成的任务定义的实际 ARN。

输出构件

- 构件数：0
- 描述：输出构件不适用于此操作类型。

操作声明

YAML

```
Name: Deploy
Actions:
  - Name: Deploy
    ActionTypeId:
      Category: Deploy
      Owner: AWS
      Provider: CodeDeployToECS
      Version: '1'
    RunOrder: 1
    Configuration:
      AppSpecTemplateArtifact: SourceArtifact
      ApplicationName: ecs-cd-application
      DeploymentGroupName: ecs-deployment-group
      Image1ArtifactName: MyImage
      Image1ContainerName: IMAGE1_NAME
      TaskDefinitionTemplatePath: taskdef.json
      AppSpecTemplatePath: appspec.yaml
      TaskDefinitionTemplateArtifact: SourceArtifact
    OutputArtifacts: []
    InputArtifacts:
      - Name: SourceArtifact
      - Name: MyImage
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
        {
            "Name": "Deploy",
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "CodeDeployToECS",
                "Version": "1"
            },
            "RunOrder": 1,
```

```
            "Configuration": {
                "AppSpecTemplateArtifact": "SourceArtifact",
                "ApplicationName": "ecs-cd-application",
                "DeploymentGroupName": "ecs-deployment-group",
                "Image1ArtifactName": "MyImage",
                "Image1ContainerName": "IMAGE1_NAME",
                "TaskDefinitionTemplatePath": "taskdef.json",
                "AppSpecTemplatePath": "appspec.yaml",
                "TaskDefinitionTemplateArtifact": "SourceArtifact"
            },
            "OutputArtifacts": [],
            "InputArtifacts": [
                {
                    "Name": "SourceArtifact"
                },
                {
                    "Name": "MyImage"
                }
            ],
            "Region": "us-west-2",
            "Namespace": "DeployVariables"
        }
    ]
}
```

另请参阅

下列相关资源在您使用此操作的过程中会有所帮助。

- 教程：使用 Amazon ECR 源和 ECS 目标部署创建管道 CodeDeploy — 本教程将引导您创建蓝/绿部署所需的 CodeDeploy 和 Amazon ECS 资源。本教程展示了如何将 Docker 映像推送到 Amazon ECR，并创建 Amazon ECS 任务定义，其中列出您的 Docker 映像名称、容器名称、Amazon ECS 服务名称和负载均衡器配置。然后，本教程将引导您完成为部署创建 AppSpec 文件和管道的过程。

> **Note**
>
> 本主题和教程介绍了 CodeDeploy /ECS 的蓝/绿操作。 CodePipeline有关中 ECS 标准操作的信息 CodePipeline，请参阅教程：使用进行持续部署 CodePipeline。

- AWS CodeDeploy 用户指南 — 有关如何在蓝/绿部署中使用负载均衡器、生产侦听器、目标组和您的 Amazon ECS 应用程序的信息，请参阅教程：部署 Amazon ECS 服务。《AWS CodeDeploy 用户指南》中的此参考信息概述了 Amazon ECS 和的蓝/绿部署。 AWS CodeDeploy
- Amazon Elastic Container Service 开发者指南：有关使用 Docker 映像和容器、ECS 服务和集群以及 ECS 任务集的信息，请参阅什么是 Amazon ECS？

Amazon Elastic Container Service

您可以使用 Amazon ECS 操作来部署 Amazon ECS 服务和任务集。Amazon ECS 服务是部署到 Amazon ECS 集群的容器应用程序。Amazon ECS 集群是在云中托管容器应用程序的实例集。部署需要您在 Amazon ECS 中创建的任务定义和 CodePipeline 用于部署映像的图像定义文件。

> **Important**
>
> Amazon ECS 的标准部署操作会根据 Amazon ECS 服务使用的修订版 CodePipeline 创建自己的任务定义修订版。如果您在不更新 Amazon ECS 服务的情况下为任务定义创建新的修订，则部署操作将忽略这些修订。

在创建管道之前，您必须已创建 Amazon ECS 资源，在映像存储库中标记并存储映像，并将 BuildSpec 文件上传到文件存储库。

> **Note**
>
> 本参考主题介绍了 Amazon ECS 的标准部署操作 CodePipeline。有关中 Amazon ECS 到 CodeDeploy 蓝/绿部署操作的参考信息 CodePipeline，请参阅。Amazon 弹性容器服务和 CodeDeploy 蓝绿色

主题

操作类型

- 类别：Deploy
- 拥有者：AWS
- 提供方：ECS
- 版本：1

配置参数

ClusterName

必需：是

Amazon ECS 中的 Amazon ECS 集群。

ServiceName

必需：是

您在 Amazon ECS 中创建的 Amazon ECS 服务。

FileName

必需：否

您的映像定义文件，该 JSON 描述服务的容器名称以及映像和标签。您需要将此文件用于 ECS 标准部署。有关更多信息，请参阅 输入构件 和 适用于 Amazon ECS 标准部署操作的 imagedefinitions.json 文件。

DeploymentTimeout

必需：否

Amazon ECS 部署操作超时（以分钟为单位）。该超时可配置为此操作的最大默认超时。例如：

```
"DeploymentTimeout": "15"
```

输入构件

- 构件数：1

- 描述：该操作会在管道的源文件存储库中查找 `imagedefinitions.json` 文件。图像定义文档是一个 JSON 文件，用于描述您的 Amazon ECS 容器名称以及图像和标签。 CodePipeline 使用该文件从您的图像存储库（例如 Amazon ECR）中检索图像。在未自动执行操作的情况下，您可以为管道手动添加 `imagedefinitions.json` 文件。有关 `imagedefinitions.json` 文件的信息，请参阅适用于 Amazon ECS 标准部署操作的 imagedefinitions.json 文件。

  操作需要已推送到您的映像存储库的现有映像。由于映像映射由 `imagedefinitions.json` 文件提供，因此操作不要求包含 Amazon ECR 源以作为管道中的源操作。

输出构件

- 构件数：`0`
- 描述：输出构件不适用于此操作类型。

操作声明

YAML

```
Name: DeployECS
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: ECS
  Version: '1'
RunOrder: 2
Configuration:
  ClusterName: my-ecs-cluster
  ServiceName: sample-app-service
  FileName: imagedefinitions.json
  DeploymentTimeout: '15'
OutputArtifacts: []
InputArtifacts:
  - Name: my-image
```

JSON

```
{
    "Name": "DeployECS",
    "ActionTypeId": {
```

```
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "ECS",
        "Version": "1"
    },
    "RunOrder": 2,
    "Configuration": {
        "ClusterName": "my-ecs-cluster",
        "ServiceName": "sample-app-service",
        "FileName": "imagedefinitions.json",
        "DeploymentTimeout": "15"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "my-image"
        }
    ]
},
```

另请参阅

下列相关资源在您使用此操作的过程中会有所帮助。

- 教程：使用持续部署 CodePipeline — 本教程向您展示如何创建存储在源文件存储库中的 Dockerfile，例如。 CodeCommit接下来，本教程将向您展示如何合并一个 CodeBuild BuildSpec 文件，该文件用于生成您的 Docker 映像并将其推送到 Amazon ECR，并创建 imagedefinitions.json 文件。最后，您将创建 Amazon ECS 服务和任务定义，然后创建具有 Amazon ECS 部署操作的管道。

  > **Note**
  >
  > 本主题和教程介绍了 Amazon ECS 的标准部署操作 CodePipeline。有关中 Amazon ECS 到 CodeDeploy 蓝/绿部署操作的信息 CodePipeline，请参阅。教程：使用 Amazon ECR 源和 ECS 目标部署创建管道 CodeDeploy

- Amazon Elastic Container Service 开发者指南：有关使用 Docker 映像和容器、Amazon ECS 服务和集群以及 Amazon ECS 任务集的信息，请参阅什么是 Amazon ECS？

Amazon S3 部署操作

您可以使用 Amazon S3 部署操作，向用于静态网站托管或归档的 Amazon S3 桶部署文件。您可以指定是否在将部署文件上传到桶之前提取部署文件。

> **Note**
>
> 本参考主题介绍了 Amazon S3 部署操作， CodePipeline 其中部署平台是配置为托管的 Amazon S3 存储桶。有关中 Amazon S3 源操作的参考信息 CodePipeline，请参阅Amazon S3 源操作。

主题

操作类型

- 类别：`Deploy`
- 拥有者：`AWS`
- 提供方：`S3`
- 版本：`1`

配置参数

BucketName

必需：是

要用来存储源文件的 Amazon S3 桶的名称。

Extract

必需：是

如果为 true，则会规定在上传之前提取文件。否则，应用程序文件将保持压缩状态以供上传，如托管静态网站的情况。如果为 false，则 `ObjectKey` 为必填项。

ObjectKey

这是有条件的。如果 `Extract` = false，则为必需

以唯一方式标识 S3 桶中对象的 Amazon S3 对象密钥的名称。

KMS EncryptionKey ARN

必需：否

主机存储桶的 AWS KMS 加密密钥的 ARN。`KMSEncryptionKeyARN` 参数使用提供的 AWS KMS key对上传的构件进行加密。对于 KMS 密钥，您可以使用密钥 ID、密钥 ARN 或别名 ARN。

> **Note**
>
> 别名只能在创建 KMS 密钥的账户中识别。对于跨账户操作，您只能使用密钥 ID 或密钥 ARN 来标识密钥。跨账户操作涉及使用其他账户（AccountB）的角色，因此指定密钥 ID 将使用其他账户（AccountB）的密钥。

> **Important**
>
> CodePipeline 仅支持对称 KMS 密钥。请勿使用非对称 KMS 密钥对 S3 桶中的数据进行加密。

CannedACL

必需：否

`CannedACL` 参数将指定的标准 ACL 应用于部署到 Amazon S3 的对象。这将覆盖已应用于对象的任何现有 ACL。

CacheControl

必需：否

CacheControl 参数控制桶中对象的请求/响应的缓存行为。有关有效值的列表，请参阅 HTTP 操作的 Cache-Control 标头字段。要在 CacheControl 中输入多个值，请在每个值之间使用逗号。您可以为 CLI 在每个逗号后添加一个空格（可选），如本示例所示：

```
"CacheControl": "public, max-age=0, no-transform"
```

输入构件

- 构件数：1
- 描述：用于部署或存档的文件由 CodePipeline源存储库获取、压缩和上传。

输出构件

- 构件数：0
- 描述：输出构件不适用于此操作类型。

操作配置示例

下面展示了操作配置的示例。

当 **Extract** 设置为 **false** 时的示例配置

以下示例显示了在 Extract 字段设置为 false 的情况下，创建操作时的默认操作配置。

YAML

```
Name: Deploy
Actions:
  - Name: Deploy
    ActionTypeId:
      Category: Deploy
      Owner: AWS
      Provider: S3
      Version: '1'
    RunOrder: 1
    Configuration:
      BucketName: website-bucket
      Extract: 'false'
```

```
    OutputArtifacts: []
    InputArtifacts:
      - Name: SourceArtifact
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
        {
            "Name": "Deploy",
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "S3",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "BucketName": "website-bucket",
                "Extract": "false"
                },
            "OutputArtifacts": [],
            "InputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "Region": "us-west-2",
            "Namespace": "DeployVariables"
        }
    ]
},
```

当 **Extract** 设置为 **true** 时的示例配置

以下示例显示了在 Extract 字段设置为 true 的情况下，创建操作时的默认操作配置。

YAML

```
Name: Deploy
Actions:
  - Name: Deploy
    ActionTypeId:
      Category: Deploy
      Owner: AWS
      Provider: S3
      Version: '1'
    RunOrder: 1
    Configuration:
      BucketName: website-bucket
      Extract: 'true'
      ObjectKey: MyWebsite
    OutputArtifacts: []
    InputArtifacts:
      - Name: SourceArtifact
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
        {
            "Name": "Deploy",
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "S3",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "BucketName": "website-bucket",
                "Extract": "true",
                "ObjectKey": "MyWebsite"
                },
            "OutputArtifacts": [],
            "InputArtifacts": [
                {
```

```
                    "Name": "SourceArtifact"
                }
            ],
            "Region": "us-west-2",
            "Namespace": "DeployVariables"
        }
    ]
},
```

另请参阅

下列相关资源在您使用此操作的过程中会有所帮助。

- 教程：创建以 Amazon S3 作为部署提供程序的管道：本教程将演示两个示例，引导您创建具有 S3 部署操作的管道。您可以下载示例文件，将文件上传到 CodeCommit 存储库，创建 S3 存储桶，然后配置存储桶以进行托管。接下来，您可以使用 CodePipeline 控制台创建管道并指定 Amazon S3 部署配置。
- Amazon S3 源操作— 本操作参考提供了 Amazon S3 源操作的参考信息和示例 CodePipeline。

Amazon S3 源操作

将新对象上传到配置的存储桶和对象键时触发管道。

> **Note**
>
> 本参考主题介绍了 Amazon S3 源操作， CodePipeline 其中源位置是为版本控制配置的 Amazon S3 存储桶。有关中 Amazon S3 部署操作的参考信息 CodePipeline，请参阅Amazon S3 部署操作。

您可以创建一个 Amazon S3 桶，用作应用程序文件的源位置。

> **Note**
>
> 在您创建源存储桶时，请确保在存储桶上启用版本控制。如果您要使用现有 Amazon S3 桶，请参阅使用版本控制，以在现有的桶上启用版本控制。

如果您使用控制台创建或编辑管道，则 CodePipeline 会创建一个 CloudWatch 事件规则，该规则在 S3 源存储桶发生更改时启动您的管道。

您必须事先创建 Amazon S3 源桶并将源文件作为单个 ZIP 文件上传，然后才能通过 Amazon S3 操作连接管道。

> **Note**
>
> 当 Amazon S3 是您的管道的源提供程序时，您可以将一个或多个源文件压缩到单个 .zip 文件中，然后将 .zip 文件上传到源桶。您也可以上传单个解压缩的文件；但是，需要 .zip 文件的下游操作将失败。

主题

操作类型

- 类别：`Source`
- 拥有者：`AWS`
- 提供方：`S3`
- 版本：`1`

配置参数

S3Bucket

必需：是

要在其中检测源更改的 Amazon S3 桶的名称。

S3 ObjectKey

必需：是

要在其中检测源更改的 Amazon S3 对象键的名称。

AllowOverrideForS3 ObjectKey

必需：否

`AllowOverrideForS3ObjectKey`控制来自的源覆盖是否`StartPipelineExecution`可以覆盖源操作`S3ObjectKey`中已配置的。有关使用 S3 对象键替换源代码的更多信息，请参阅使用源修订覆盖启动管道。

> **Important**
>
> 如果省略`AllowOverrideForS3ObjectKey`，则 CodePipeline 默认能够通过将此参数设置为 `false`，ObjectKey 在源操作中覆盖 S3。

此参数的有效值：

- `true`: 如果已设置，则在管道执行期间，源版本覆盖可以覆盖预配置的 S3 对象密钥。

  > **Note**
  >
  > 如果您打算允许所有 CodePipeline 用户在开始新的管道执行时覆盖预先配置的 S3 对象密钥，则必须`AllowOverrideForS3ObjectKey`将`true`设置为。

- `false`:

  如果已设置，CodePipeline 则不允许使用源修订版本覆盖覆盖 S3 对象密钥。这也是此参数的默认值。

PollForSourceChanges

必需：否

`PollForSourceChanges`控制是否对 Amazon S3 源存储桶进行 CodePipeline 轮询以了解源代码更改。我们建议您改用 “ CloudWatch 事件” 和 CloudTrail “检测源更改”。有关配置 CloudWatch 事件的更多信息，请参阅使用 S3 源和跟 CloudTrail 踪 (CLI) 迁移轮询管道或使用 S3 源和 CloudTrail 跟踪（ AWS CloudFormation 模板）迁移轮询管道。

> **Important**
>
> 如果要配置 CloudWatch 事件，则必须将设置为`PollForSourceChangesfalse`以避免重复的管道执行。

此参数的有效值：

- `true`: 如果已设置，则会对您的来源位置进行 CodePipeline 轮询以了解源位置是否有更改。

  > **Note**
  >
  > 如果省略`PollForSourceChanges`，则 CodePipeline 默认为对源位置进行轮询以查看源更改。如果包括 `PollForSourceChanges` 并将其设置为 `true`，则此行为相同。

- `false`: 如果已设置，则 CodePipeline 不对源位置进行轮询以了解源位置是否有更改。如果您打算配置 CloudWatch 事件规则以检测源更改，请使用此设置。

输入构件

- 构件数：`0`
- 描述：输入构件不适用于此操作类型。

输出构件

- 构件数：`1`
- 描述：提供在源存储桶中配置为连接到管道的可用构件。从桶中生成的构件是 Amazon S3 操作的输出构件。Amazon S3 对象元数据（ETag 和版本 ID）显示为触发管道执行的源修订版。CodePipeline

输出变量

配置后，此操作会生成变量，该变量可由管道中下游操作的操作配置引用。此操作生成的变量可视为输出变量，即使操作没有命名空间也是如此。您可以使用命名空间配置操作，以使这些变量可用于下游操作的配置。

有关中变量的更多信息 CodePipeline，请参阅Variables。

BucketName

与触发管道的源代码更改相关的 Amazon S3 存储桶的名称。

ETag

与触发管道的源更改相关的对象的实体标签。ETag 是该对象的 MD5 哈希值。ETag 仅反映对象内容的更改，而不是其元数据的更改。

ObjectKey

与触发管道的源更改相关的 Amazon S3 对象密钥的名称。

VersionId

与触发管道的源更改相关的对象版本的版本 ID。

操作声明

YAML

```
Name: Source
Actions:
  - RunOrder: 1
    OutputArtifacts:
      - Name: SourceArtifact
    ActionTypeId:
      Provider: S3
      Owner: AWS
      Version: '1'
      Category: Source
    Region: us-west-2
    Name: Source
    Configuration:
      S3Bucket: my-bucket-oregon
      S3ObjectKey: my-application.zip
      PollForSourceChanges: 'false'
    InputArtifacts: []
```

JSON

```
{
    "Name": "Source",
```

```
    "Actions": [
        {
            "RunOrder": 1,
            "OutputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "ActionTypeId": {
                "Provider": "S3",
                "Owner": "AWS",
                "Version": "1",
                "Category": "Source"
            },
            "Region": "us-west-2",
            "Name": "Source",
            "Configuration": {
                "S3Bucket": "my-bucket-oregon",
                "S3ObjectKey": "my-application.zip",
                "PollForSourceChanges": "false"
            },
            "InputArtifacts": []
        }
    ]
},
```

另请参阅

下列相关资源在您使用此操作的过程中会有所帮助。

- 教程：创建一个简单的管道（S3 存储桶）— 本教程提供了示例应用程序规范文件以及示例 CodeDeploy应用程序和部署组。使用本教程可创建具有 Amazon S3 源的管道，用于向 Amazon EC2 实例进行部署。

AWS AppConfig

AWS AppConfig 是一种能力 AWS Systems Manager。 AppConfig 支持对任何规模的应用程序进行受控部署，并包括内置的验证检查和监控。您可以 AppConfig 与 Amazon EC2 实例上托管的应用程序 AWS Lambda、容器、移动应用程序或物联网设备一起使用。

AppConfig部署 AWS CodePipeline 操作是一种将存储在管道源位置的配置部署到指定的 AppConfig 应用程序、环境和配置文件中的操作。它使用 AppConfig 部署策略中定义的首选项。

操作类型

- 类别：Deploy
- 拥有者：AWS
- 提供方：AppConfig
- 版本：1

配置参数

应用程序

必需：是

AWS AppConfig 应用程序的 ID，其中包含您的配置和部署的详细信息。

环境

必需：是

部署配置的 AWS AppConfig 环境的 ID。

ConfigurationProfile

必需：是

要部署的 AWS AppConfig 配置文件的 ID。

InputArtifactConfigurationPath

必需：是

要部署的输入构件中的配置数据的文件路径。

DeploymentStrategy

必需：否

用于 AWS AppConfig 部署的部署策略。

输入构件

- 构件数：1
- 描述：部署操作的输入构件。

输出构件

不适用。

操作配置示例

YAML

```
name: Deploy
actions:
  - name: Deploy
    actionTypeId:
      category: Deploy
      owner: AWS
      provider: AppConfig
      version: '1'
    runOrder: 1
    configuration:
      Application: 2s2qv57
      ConfigurationProfile: PvjrpU
      DeploymentStrategy: frqt7ir
      Environment: 9tm27yd
      InputArtifactConfigurationPath: /
    outputArtifacts: []
    inputArtifacts:
      - name: SourceArtifact
    region: us-west-2
    namespace: DeployVariables
```

JSON

```
{
    "name": "Deploy",
    "actions": [
        {
```

```
            "name": "Deploy",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "provider": "AppConfig",
                "version": "1"
            },
            "runOrder": 1,
            "configuration": {
                "Application": "2s2qv57",
                "ConfigurationProfile": "PvjrpU",
                "DeploymentStrategy": "frqt7ir",
                "Environment": "9tm27yd",
                "InputArtifactConfigurationPath": "/"
            },
            "outputArtifacts": [],
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-west-2",
            "namespace": "DeployVariables"
        }
    ]
}
```

另请参阅

下列相关资源在您使用此操作的过程中会有所帮助。

- AWS AppConfig— 有关 AWS AppConfig 部署的信息，请参阅《AWS Systems Manager 用户指南》。
- 教程：创建 AWS AppConfig 用作部署提供者的管道— 本教程将帮助您开始设置简单的部署配置文件和 AppConfig 资源，并向您展示如何使用控制台创建带有 AWS AppConfig 部署操作的管道。

AWS CloudFormation

在 AWS CloudFormation 堆栈上执行操作。堆栈是您可以作为一个单元管理的 AWS 资源集合。堆栈中的资源均由堆栈的 AWS CloudFormation 模板定义。更改集会创建比较，您可以查看比较而不

会更改原始堆栈。有关可以对堆栈和更改集执行的 AWS CloudFormation 操作类型的信息，请参阅ActionMode参数。

要为堆栈操作失败的 AWS CloudFormation 操作构造错误消息， CodePipeline 请调用 AWS CloudFormation DescribeStackEvents API。如果操作 IAM 角色有权访问该 API，则有关第一个失败资源的详细信息将包含在 CodePipeline 错误消息中。否则，如果角色策略没有相应的权限，则 CodePipeline 会忽略访问该 API，而是显示一条通用的错误消息。为此，必须将 cloudformation:DescribeStackEvents 权限添加到管道的服务角色或其他 IAM 角色中。

如果您不希望资源详细信息出现在管道错误消息中，可以通过删除 cloudformation:DescribeStackEvents 权限为操作 IAM 角色撤销此权限。

主题

操作类型

- 类别：Deploy
- 拥有者：AWS
- 提供方：CloudFormation
- 版本：1

配置参数

ActionMode

必需：是

ActionMode是对堆栈或更改集 AWS CloudFormation 执行的操作的名称。提供以下操作模式：

- CHANGE_SET_EXECUTE 会基于一组指定资源更新执行资源堆栈的更改集。通过此操作， AWS CloudFormation 开始更改堆栈。
- CHANGE_SET_REPLACE 根据您提交的堆栈名称和模板创建更改集（如果更改集不存在）。如果更改集存在，则将其 AWS CloudFormation 删除，然后创建一个新的变更集。
- CREATE_UPDATE 会在堆栈不存在时创建堆栈。如果堆栈存在，则 AWS CloudFormation 更新堆栈。使用此操作更新现有堆栈。不同的是REPLACE_ON_FAILURE，如果堆栈存在且处于故障状态，则 CodePipeline 不会删除和替换堆栈。
- DELETE_ONLY 删除堆栈。如果您指定的堆栈不存在，操作将成功完成，而不会删除堆栈。
- REPLACE_ON_FAILURE 会在堆栈不存在时创建堆栈。如果堆栈存在且处于失败状态，则 AWS CloudFormation 删除该堆栈，然后创建一个新堆栈。如果堆栈未处于故障状态，则对其 AWS CloudFormation 进行更新。

  AWS CloudFormation中显示以下任意状态类型时，堆栈处于故障状态：

  - ROLLBACK_FAILED
  - CREATE_FAILED
  - DELETE_FAILED
  - UPDATE_ROLLBACK_FAILED

  使用自操作可自动替换出现故障的堆栈，而无需对其进行恢复或故障排除。

  > **Important**
  >
  > 我们建议您仅将 REPLACE_ON_FAILURE 用于测试目的，因为它可能会删除您的堆栈。

StackName

必需：是

StackName 是现有堆栈的名称或要创建的堆栈的名称。

功能

必需：条件

使用 Capabilities 即确认此模板可能具备创建和更新一些资源的功能，并且这些功能由模板资源的类型决定。

如果您的堆栈模板中有 IAM 资源或者您直接从包含宏的模板创建堆栈，则此属性是必需的。为了使操作成功地以这种方式运行，您必须明确确认您希望它使用以下功能之一执行操作： AWS CloudFormation

- CAPABILITY_IAM
- CAPABILITY_NAMED_IAM
- CAPABILITY_AUTO_EXPAND

您可以通过在功能之间添加逗号（无空格）的方式指定多个功能。操作声明中的示例显示了同时具有 CAPABILITY_IAM 和 CAPABILITY_AUTO_EXPAND 属性的条目。

有关更多信息Capabilities，请参阅《AWS CloudFormation API 参考》UpdateStack中下方的属性。

ChangeSetName

必需：条件

ChangeSetName 是现有更改集的名称或要为指定堆栈创建的新更改集的名称。

以下操作模式需要此属性：CHANGE_SET_REPLACE 和 CHANGE_SET_EXECUTE。对于所有其他操作模式，将忽略此属性。

RoleArn

必需：条件

RoleArn 是 AWS CloudFormation 在对指定堆栈中的资源执行操作时代入的 IAM 服务角色的 ARN。在执行更改集时，RoleArn 不适用。如果您不使用创建 CodePipeline 更改集，请确保更改集或堆栈具有关联的角色。

> **Note**
>
> 按照操作声明 RoleArn 中的配置，此角色必须与正在运行的操作的角色位于同一个账户。

以下操作模式需要此属性：

- CREATE_UPDATE
- REPLACE_ON_FAILURE
- DELETE_ONLY
- CHANGE_SET_REPLACE

> **Note**
>
> AWS CloudFormation 将获得模板的 S3 签名 URL；因此，这`RoleArn`不需要访问工件存储桶的权限。但是，为了生成签名 URL，操作 `RoleArn` 确实 需要用于访问构件桶的权限。

TemplatePath

必需：条件

`TemplatePath`表示 AWS CloudFormation 模板文件。您需在针对此操作的输入构件中包含该文件。文件名称遵循以下格式：

`Artifactname``::`*`TemplateFileName`*

`Artifactname`是显示在中的输入工件名称 CodePipeline。例如，源阶段利用构件名称 `SourceArtifact` 和文件名 `template-export.json` 创建 `TemplatePath` 名称，如以下示例所示：

```
"TemplatePath": "SourceArtifact::template-export.json"
```

以下操作模式需要此属性：

- CREATE_UPDATE
- REPLACE_ON_FAILURE
- CHANGE_SET_REPLACE

对于所有其他操作模式，将忽略此属性。

> **Note**
>
> 包含 AWS CloudFormation 模板正文的模板文件的最小长度为 1 字节，最大长度为 1 MB。对于中的 AWS CloudFormation 部署操作 CodePipeline，最大输入项目大小始终为 256 MB。有关更多信息，请参阅 中的配额 AWS CodePipeline和 AWS CloudFormation 限制。

OutputFileName

必需：否

`OutputFileName`用于指定输出文件名，例如`CreateStackOutput.json`，该文件名 CodePipeline 将添加到该操作的管道输出对象中。JSON 文件包含 AWS CloudFormation 堆栈中该`Outputs`部分的内容。

如果不指定名称，则 CodePipeline 不会生成输出文件或工件。

ParameterOverrides

必需：否

参数在堆栈模板中定义，并允许您在堆栈创建或更新时为这些参数提供值。您可以使用 JSON 对象设置模板中的参数值。（这些值会覆盖在模板配置文件中设置的值。）有关使用参数覆盖的更多信息，请参阅配置属性（JSON 对象）。

对于大多数参数值，建议您使用模板配置文件。仅对在管道运行前未知的值使用参数覆盖。有关更多信息，请参阅《AWS CloudFormation 用户指南》中的在 CodePipeline 管道中使用参数覆盖函数。

> **Note**
>
> 所有参数名称必须位于堆栈模板中。

TemplateConfiguration

必需：否

`TemplateConfiguration` 是模板配置文件。您需在针对此操作的输入构件中包含该文件。它可包含模板参数值和堆栈策略。有关模板配置文件格式的更多信息，请参阅 AWS CloudFormation 构件。

模板配置文件名遵循以下格式：

`Artifactname``::`*`TemplateConfigurationFileName`*

`Artifactname`是显示在中的输入工件名称 CodePipeline。例如，源阶段利用构件名称 `SourceArtifact` 和文件名 `test-configuration.json` 创建 `TemplateConfiguration` 名称，如以下示例所示：

```
"TemplateConfiguration": "SourceArtifact::test-configuration.json"
```

输入构件

- 构件数：`0 to 10`
- 描述：作为输入，该 AWS CloudFormation 操作可以选择接受用于以下目的的构件：
  - 提供要执行的堆栈模板文件。（请参阅 `TemplatePath` 参数。）
  - 提供要使用的模板配置文件。（请参阅 `TemplateConfiguration` 参数。）有关模板配置文件格式的更多信息，请参阅 AWS CloudFormation 构件。
  - 为要作为堆栈一部分部署的 Lambda 函数提供构件。 AWS CloudFormation

输出构件

- 构件数：`0 to 1`
- 描述：如果指定了 `OutputFileName` 参数，则此操作生成输出构件，其中包含具有指定名称的 JSON 文件。JSON 文件包含来自 AWS CloudFormation 堆栈的输出部分中的内容。

  有关您可为 AWS CloudFormation 操作创建的输出部分的更多信息，请参阅输出。

输出变量

配置后，此操作会生成变量，该变量可由管道中下游操作的操作配置引用。您可以使用命名空间配置操作，以使这些变量可用于下游操作的配置。

对于 AWS CloudFormation 操作，变量由堆栈模板`Outputs`部分中指定的任何值生成。请注意，生成输出的唯一 CloudFormation操作模式是那些导致创建或更新堆栈的操作模式，例如堆栈创建、堆栈更新和更改集执行。生成变量的相应操作模式包括：

- `CHANGE_SET_EXECUTE`
- `CHANGE_SET_REPLACE`
- `CREATE_UPDATE`
- `REPLACE_ON_FAILURE`

有关更多信息，请参阅 Variables。有关向您展示如何在使用 CloudFormation输出变量的管道中使用 CloudFormation 部署操作创建管道的教程，请参阅教程：创建使用 AWS CloudFormation 部署操作中的变量的管道。

操作声明

YAML

```
Name: ExecuteChangeSet
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormation
  Version: '1'
RunOrder: 2
Configuration:
  ActionMode: CHANGE_SET_EXECUTE
  Capabilities: CAPABILITY_NAMED_IAM,CAPABILITY_AUTO_EXPAND
  ChangeSetName: pipeline-changeset
  ParameterOverrides: '{"ProjectId": "my-project","CodeDeployRole":
 "CodeDeploy_Role_ARN"}'
  RoleArn: CloudFormation_Role_ARN
  StackName: my-project--lambda
  TemplateConfiguration: 'my-project--BuildArtifact::template-configuration.json'
  TemplatePath: 'my-project--BuildArtifact::template-export.yml'
OutputArtifacts: []
InputArtifacts:
  - Name: my-project-BuildArtifact
```

JSON

```
{
    "Name": "ExecuteChangeSet",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormation",
        "Version": "1"
    },
    "RunOrder": 2,
    "Configuration": {
        "ActionMode": "CHANGE_SET_EXECUTE",
        "Capabilities": "CAPABILITY_NAMED_IAM,CAPABILITY_AUTO_EXPAND",
        "ChangeSetName": "pipeline-changeset",
        "ParameterOverrides": "{\"ProjectId\": \"my-project\",\"CodeDeployRole\":
 \"CodeDeploy_Role_ARN\"}",
        "RoleArn": "CloudFormation_Role_ARN",
```

```
        "StackName": "my-project--lambda",
        "TemplateConfiguration": "my-project--BuildArtifact::template-
configuration.json",
        "TemplatePath": "my-project--BuildArtifact::template-export.yml"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
             "Name": "my-project-BuildArtifact"
        }
    ]
},
```

另请参阅

下列相关资源在您使用此操作的过程中会有所帮助。

- 配置属性参考 —《AWS CloudFormation 用户指南》中的本参考章节提供了有关这些 CodePipeline 参数的更多描述和示例。
- AWS CloudFormation API 参考 — AWS CloudFormation API 参考中的CreateStack参数描述了 AWS CloudFormation 模板的堆栈参数。

AWS CloudFormation StackSets

CodePipeline 提供了在 CI/CD 流程中执行 AWS CloudFormation StackSets 操作的能力。您可以使用堆栈集通过单个 AWS CloudFormation 模板在跨 AWS 区域的 AWS 账户中创建堆栈。每个堆栈中包含的所有资源均由堆栈集的 AWS CloudFormation 模板定义。在创建堆栈集时，您可指定要使用的模板以及模板所需的任何参数和功能。

有关概念的更多信息 AWS CloudFormation StackSets，请参阅《AWS CloudFormation 用户指南》中的StackSets 概念。

您可以 AWS CloudFormation StackSets 通过两种不同的操作类型将您的管道与之集成，这两种操作类型可以一起使用：

- `CloudFormationStackSet` 操作会通过存储在管道源位置的模板，创建或更新堆栈集或堆栈实例。每次创建或更新堆栈集时，它都会开始向指定的实例部署这些更改。在控制台中，您可以在创建或编辑管道时选择 CloudFormation Stack Set 操作提供者。

- `CloudFormationStackInstances` 操作会将 `CloudFormationStackSet` 操作的更改部署到指定的实例，创建新的堆栈实例，并定义对指定实例的参数覆盖。在控制台中，编辑现有管道时，您可以选择CloudFormation 堆栈实例操作提供者。

您可以使用这些操作来部署到目标 AWS 客户或目标 Organizations AWS 组织单位 ID。

> **Note**
>
> 要部署到目标 AWS 组织帐户或组织单位 ID 并使用服务管理的权限模型，您必须启用 AWS CloudFormation StackSets 和 AWS 组织之间的可信访问权限。有关更多信息，请参阅使用AWS CloudFormation 堆栈集启用可信访问。

主题

AWS CloudFormation StackSets 动作是如何运作的

`CloudFormationStackSet` 操作会根据操作是否是首次运行，创建或更新资源。

`CloudFormationStackSet` 操作会创建 或更新 堆栈集，并将这些更改部署到指定的实例。

> **Note**
>
> 如果您使用此操作进行更新，包括添加堆栈实例，则会先部署新实例，最后完成更新。新实例会先收到旧版本，然后更新将应用于所有实例。

- 创建：如果未指定实例且堆栈集不存在，则该CloudFormationStackSet操作将创建堆栈集而不创建任何实例。

- 更新：对已创建的堆栈集运行CloudFormationStackSet操作时，该操作会更新堆栈集。如果不指定实例且堆栈集已存在，则会更新所有实例。如果使用此操作来更新特定实例，则所有剩余的实例都将变为已过时状态。

  您可以使用该CloudFormationStackSet操作通过以下方式更新堆栈集。

  - 更新部分或全部实例上的模板。
  - 更新部分或全部实例上的参数。
  - 更新堆栈集的执行角色（这必须与管理员角色中指定的执行角色相匹配）。
  - 更改权限模型（仅在未创建实例的情况下）。
  - 启用/禁用 AutoDeployment（如果堆栈集权限模型为 Service Managed）。
  - 如果堆栈集权限模型是，则在成员账户中充当委托管理员Service Managed。
  - 更新管理员角色。
  - 更新堆栈集的描述。
  - 将部署目标添加到堆栈集更新中，以便创建新的堆栈实例。

CloudFormationStackInstances 操作会创建新的堆栈实例或更新已过时的堆栈实例。更新堆栈集后，实例将变为过时状态，但并非其中的所有实例都会更新。

- 创建：如果堆栈已经存在，则 CloudFormationStackInstances 操作仅更新实例，不会创建堆栈实例。
- 更新：执行 CloudFormationStackSet 操作后，如果仅部分实例中更新了模板或参数，则其余实例将被标记为 OUTDATED。在随后的管道阶段中，CloudFormationStackInstances 会分批更新堆栈集中的其余实例，从而将所有实例标记为 CURRENT。此操作还可用于在新实例或现有实例上添加其他实例或覆盖参数。

作为更新的一部分，CloudFormationStackSet 和 CloudFormationStackInstances 操作可以指定新的部署目标，从而创建新的堆栈实例。

作为更新的一部分，CloudFormationStackSet 和 CloudFormationStackInstances 操作不会删除堆栈集、实例或资源。当操作更新堆栈但不指定更新所有实例时，将从更新中移除未指定进行更新的实例，并将其设置为 OUTDATED 状态。

在部署期间，如果实例部署失败，则堆栈实例也可能显示 OUTDATED 状态。

如何在管道中构造 StackSets操作

作为最佳实践，您应对管道进行构建，以便创建堆栈集并且先向子集或单个实例进行部署。在测试您的部署并查看所生成的堆栈集后，再添加 CloudFormationStackInstances 操作以创建和更新剩余的实例。

使用控制台或 CLI 创建所建议的管道结构，如下所示：

1. 创建具有源操作（必需）和 CloudFormationStackSet 操作（作为部署操作）的管道。运行您的管道。
2. 当您的管道首次运行时，CloudFormationStackSet 操作会创建 您的堆栈集和至少一个初始实例。验证是否创建堆栈集，检查是否向您的初始实例进行部署。例如，为账户 Account-A 创建初始堆栈集，其中 us-east-1 为指定的区域，将会使用堆栈集创建以下堆栈实例：

堆栈实例	区域	Status
StackInstanceID-1	us-east-1	CURRENT

3. 编辑您的管道，将 CloudFormationStackInstances 添加为第二个部署操作，以便为您指定的目标创建/更新堆栈实例。例如，为账户 Account-A 创建堆栈实例，其中指定 us-east-2 和 eu-central-1 区域，将会创建剩余的堆栈实例，并且初始实例仍会进行更新，如下所示：

堆栈实例	区域	Status
StackInstanceID-1	us-east-1	CURRENT
StackInstanceID-2	us-east-2	CURRENT
StackInstanceID-3	eu-central-1	CURRENT

4. 根据需要运行管道以更新您的堆栈集，并更新或创建堆栈实例。

在已经从操作配置中移除部署目标的情况下，当您启动堆栈更新时，未指定进行更新的堆栈实例将从部署中移除，并且变为已过时状态。例如，为账户 Account-A 更新堆栈实例，其中 us-east-2 区域从操作配置中移除，则会创建剩余的堆栈实例，并将移除的实例设置为已过时，如下所示：

堆栈实例	区域	Status
StackInstanceID-1	us-east-1	CURRENT
StackInstanceID-2	us-east-2	OUTDATED
StackInstanceID-3	eu-central-1	CURRENT

有关部署堆栈集的最佳实践的更多信息，请参阅《AWS CloudFormation 用户指南》 StackSets 中的最佳实践。

`CloudFormationStackSet` 操作

此操作会通过存储在管道源位置的模板，创建或更新堆栈集。

在定义堆栈集后，您可以在配置参数中指定的目标账户和区域，创建、更新或删除堆栈。在创建、更新或删除堆栈时，您可以指定其他首选项，如执行操作的区域顺序、容错百分比（超过之后会导致堆栈操作停止），以及在其中对堆栈并行执行操作的账户的数量。

堆栈集是一种区域性资源。如果您在一个 AWS 区域中创建堆栈集，则无法从其他区域访问该堆栈集。

当此操作用作对堆栈集的更新操作时，如果不部署到至少一个堆栈实例，则不允许对堆栈进行更新。

主题

- 操作类型
- 配置参数
- 输入构件
- 输出构件
- 输出变量
- CloudFormationStackSet操作配置示例

操作类型

- 类别：`Deploy`
- 拥有者：`AWS`

- 提供方：CloudFormationStackSet
- 版本：1

配置参数

StackSetName

必需：是

要与堆栈集关联的名称。此名称在创建它的区域中必须唯一。

名称中只能包含字母数字字符和连字符。它必须以字母字符开头，且不得超过 128 个字符。

描述

必需：否

堆栈集的描述。您可以使用它来描述堆栈集的用途或其他相关信息。

TemplatePath

必需：是

定义堆栈集中资源的模板的位置。这必须指向最大大小为 460,800 字节的模板。

按照格式 "InputArtifactName::TemplateFileName" 输入源构件名称和模板文件的路径，如以下示例所示。

```
SourceArtifact::template.txt
```

参数

必需：否

部署期间会进行更新的堆栈集模板参数的列表。

您可以通过文本列表或文件路径的形式提供参数：

- 您可以使用以下简写语法格式输入参数：ParameterKey=string,ParameterValue=string,UsePreviousValue=boolean,Resolv
  ParameterKey=string,ParameterValue=string,UsePreviousValue=boolean,ResolvedV
  有关这些数据类型的更多信息，请参阅模板参数数据类型。

以下示例显示了一个名为 BucketName 且值为 my-bucket 的参数。

```
ParameterKey=BucketName,ParameterValue=my-bucket
```

以下示例显示了一个具有多个参数的条目：

```
ParameterKey=BucketName,ParameterValue=my-bucket
ParameterKey=Asset1,ParameterValue=true
ParameterKey=Asset2,ParameterValue=true
```

- 您可以按照 "InputArtifactName::ParametersFileName" 格式，输入包含模板参数覆盖列表的文件的位置，如以下示例所示。

```
SourceArtifact::parameters.txt
```

以下示例显示 parameters.txt 的文件内容。

```
[
    {
        "ParameterKey": "KeyName",
        "ParameterValue": "true"
    },
    {
        "ParameterKey": "KeyName",
        "ParameterValue": "true"
    }
]
```

功能

必需：否

表示模板可以创建和更新资源，具体取决于模板中的资源类型。

如果您的堆栈模板中有 IAM 资源或者您直接从包含宏的模板创建堆栈，则必须使用此属性。要使 AWS CloudFormation 操作以这种方式成功运行，必须使用以下功能之一：

- CAPABILITY_IAM
- CAPABILITY_NAMED_IAM

您可以使用逗号指定多项功能，各项功能之间不留空格。CloudFormationStackSet操作配置示例中的示例展示了一个具有多项功能的条目。

PermissionModel

必需：否

确定如何创建和管理 IAM 角色。如果未指定该字段，则使用默认值。有关信息，请参阅 堆栈集操作的权限模型。

有效值为：

- SELF_MANAGED（默认）：您必须创建管理员角色和执行角色才能向目标账户进行部署。
- SERVICE_MANAGED： AWS CloudFormation StackSets 自动创建部署到 Organizations 管理的账户所需的 AWS IAM 角色。这要求账户必须是组织的成员。

> **Note**
>
> 只有当堆栈集中不存在任何堆栈实例时，才能更改此参数。

AdministrationRoleArn

> **Note**
>
> 由于跨多个账户 AWS CloudFormation StackSets 执行操作，因此必须先在这些账户中定义必要的权限，然后才能创建堆栈集。

必需：否

> **Note**
>
> 此参数对于 SELF_MANAGED 权限模型来说是可选的，不用于 SERVICE_MANAGED 权限模型。

管理员账户中用于执行堆栈集操作的 IAM 角色的 ARN。

名称中可以包含字母数字字符，以及以下任意字符：_+=,.@-，不能含有空格。名称不区分大小写。此角色名称的最小长度必须为 20 个字符，最大长度为 2048 个字符。角色名称在账户中必

须是唯一的。此处指定的角色名称必须是现有的角色名称。如果未指定角色名称，则将其设置为 AWSCloudFormationStackSetAdministrationRole。如果指定 ServiceManaged，则不得定义角色名称。

ExecutionRoleName

> **Note**
>
> 由于跨多个账户 AWS CloudFormation StackSets 执行操作，因此必须先在这些账户中定义必要的权限，然后才能创建堆栈集。

必需：否

> **Note**
>
> 此参数对于 SELF_MANAGED 权限模型来说是可选的，不用于 SERVICE_MANAGED 权限模型。

目标账户中用于执行堆栈集操作的 IAM 角色的名称。名称中可以包含字母数字字符，以及以下任意字符：_+=,.@-，不能含有空格。名称不区分大小写。此角色名称的最小长度必须为 1 个字符，最大长度为 64 个字符。角色名称在账户中必须是唯一的。此处指定的角色名称必须是现有的角色名称。如果要使用自定义执行角色，请不要指定此角色。如果不指定角色名称，则它将被设置为 `AWSCloudFormationStackSetExecutionRole`。如果将 Service_Managed 设置为 true，则不得定义角色名称。

OrganizationsAutoDeployment

必需：否

> **Note**
>
> 此参数对于 SERVICE_MANAGED 权限模型来说是可选的，不用于 SELF_MANAGED 权限模型。

描述是否 AWS CloudFormation StackSets 自动部署到已添加到目标 AWS 组织或组织单位 (OU) 的 Organizations 帐户。如果指定 `OrganizationsAutoDeployment`，则请勿指定 `DeploymentTargets` 和 `Regions`。

> **Note**
>
> 如果不为 `OrganizationsAutoDeployment` 提供输入，则默认值为 `Disabled`。

有效值为：

- `Enabled`。必需：否。

  StackSets 自动将其他堆栈实例部署到指定区域的目标 AWS 组织或组织单位 (OU) 的 Organizations 账户。如果账户已从目标组织或 OU 中移除，则会从指定区域的账户中 AWS CloudFormation StackSets 删除堆栈实例。

- `Disabled`。必需：否。

  StackSets 不会自动将其他堆栈实例部署到指定区域的目标 AWS 组织或组织单位 (OU) 的 Organizations 账户。

- `EnabledWithStackRetention`。必需：否。

  从目标组织或 OU 中移除账户时，会保留堆栈资源。

DeploymentTargets

必需：否

> **Note**
>
> 对于 SERVICE_MANAGED 权限模型，您可以提供部署目标的组织根 ID 或组织单位 ID。对于 SELF_MANAGED 权限模型，您只能提供账户。

> **Note**
>
> 选择此参数后，还必须选择区域。

应在其中创建/更新堆栈集实例的 AWS 账户或组织单位 ID 列表。

- Accounts：

  您可以通过文本列表或文件路径的形式提供账户：

  - 文本：使用简写语法格式 `account_ID,account_ID` 输入参数，如以下示例所示。

```
111111222222,333333444444
```

- 文件路径：文件的位置，该文件包含应在其中创建/更新堆栈集实例的 AWS 帐户列表，格式为。`InputArtifactName::AccountsFileName`如果您使用文件路径来指定账户或 OrganizationalUnitIds，则文件格式必须采用 JSON 格式，如以下示例所示。

  ```
  SourceArtifact::accounts.txt
  ```

  以下示例显示 `accounts.txt` 的文件内容。

  ```
  [
      "111111222222"
  ]
  ```

  以下示例显示了列出多个账户时 `accounts.txt` 的文件内容：

  ```
  [
      "111111222222","333333444444"
  ]
  ```

- OrganizationalUnitIds:

  > **Note**
  >
  > 此参数对于 SERVICE_MANAGED 权限模型来说是可选的，不用于 SELF_MANAGED 权限模型。如果您选择，请不要使用此选项OrganizationsAutoDeployment。

  更新关联堆栈实例的 AWS 组织单位。

  您可以通过文本列表或文件路径的形式提供组织单位 ID：

  - 文本：输入用逗号分隔的字符串数组，如以下示例所示。

    ```
    ou-examplerootid111-exampleouid111,ou-examplerootid222-exampleouid222
    ```

  - 文件路径：文件的位置，其中包含要 OrganizationalUnitIds 在其中创建或更新堆栈集实例的列表。如果您使用文件路径来指定账户或 OrganizationalUnitIds，则文件格式必须采用 JSON 格式，如以下示例所示。

按照 InputArtifactName::OrganizationalUnitIdsFileName 格式输入文件路径。

```
SourceArtifact::OU-IDs.txt
```

以下示例显示 OU-IDs.txt 的文件内容：

```
[
    "ou-examplerootid111-exampleouid111","ou-examplerootid222-exampleouid222"
]
```

区域

必需：否

> **Note**
>
> 选择此参数后，还必须选择DeploymentTargets。

创建或更新堆栈集实例的 AWS 区域列表。区域将按照输入顺序进行更新。

按照以下示例所示的格式Region1,Region2输入有效 AWS 区域列表。

```
us-west-2,us-east-1
```

FailureTolerancePercentage

必需：否

在该区域 AWS CloudFormation 停止该堆栈操作之前，每个区域中该堆栈操作可能失败的账户百分比。如果操作在某个区域中停止，则 AWS CloudFormation 不会在后续区域中尝试该操作。根据指定的百分比计算账户数量时，向下 AWS CloudFormation 舍入到下一个整数。

MaxConcurrentPercentage

必需：否

一次执行此操作的账户数的最大百分比。根据指定的百分比计算账户数量时，向下 AWS CloudFormation 舍入到下一个整数。如果向下舍入结果为零，则改为将数字 AWS CloudFormation 设置为一。尽管您使用此设置来指定最大值，但对于大型部署，并行执行的实际账户的数量可能会因服务节流而减少。

RegionConcurrencyType

必需：否

您可以通过配置区域并发部署参数，指定堆栈集应该按照顺序方式还是并行方式跨 AWS 区域 进行部署。如果将区域并发性指定为 AWS 区域 并行部署多个堆栈，则可以缩短总体部署时间。

- 并行：只要区域的部署失败次数不超过指定的容错值，就会同时执行堆栈集部署。
- 顺序：只要区域的部署失败次数不超过指定的容错值，就会一次执行一个堆栈集部署。顺序部署是默认选项。

ConcurrencyMode

必需：否

并发模式允许您选择并发级别在堆栈集操作期间的行为方式，无论是严格容错还是软容错。严格容错会降低堆栈集操作失败时的部署速度，因为每次失败都会降低并发数。S@@ oft Fail ure Tolerance 优先考虑部署速度，同时仍能利用 AWS CloudFormation 安全功能。

- STRICT_FAILURE_TOLERANCE：此选项可动态降低并发级别，确保失败账户的数量永远不会超过特定的容错值。这是默认行为。
- SOFT_FAILURE_TOLERANCE：此选项将容错与实际并发数分离。这允许堆栈集操作在一个设定的并发级别上运行，而不管失败次数是多少。

CallAs

必需：否

> **Note**
>
> 此参数对于SERVICE_MANAGED权限模型是可选的，不用于SELF_MANAGED权限模型。

指定您是以组织的管理账户还是以成员账户的委托管理员身份行事。

> **Note**
>
> 如果此参数设置为DELEGATED_ADMIN，请确保管道 IAM 角色具有organizations:ListDelegatedAdministrators权限。否则，操作将在运行时失败，并显示类似于以下内容的错误：Account used is not a delegated administrator。

- SELF: 堆栈集部署将在登录管理账户时使用服务管理权限。
- DELEGATED_ADMIN: 堆栈集部署将在登录委托管理员账户时使用服务管理权限。

输入构件

您必须至少包括一个输入构件，其中包含 CloudFormationStackSet 操作中堆栈集的模板。对于部署目标、账户和参数列表，您可以包含更多输入构件。

- 构件数：1 to 3
- 描述：您可以包含构件以提供：
  - 堆栈模板文件。（请参阅 TemplatePath 参数。）
  - 参数文件。（请参阅 Parameters 参数。）
  - 账户文件。（请参阅 DeploymentTargets 参数。）

输出构件

- 构件数：0
- 描述：输出构件不适用于此操作类型。

输出变量

如果配置此操作，它会生成一些变量，可供管道中下游操作的操作配置引用。您可以使用命名空间配置操作，以使这些变量可用于下游操作的配置。

- StackSetId: 堆栈集的 ID。
- OperationId: 堆栈集操作的 ID。

有关更多信息，请参阅 Variables。

CloudFormationStackSet操作配置示例

以下示例显示了操作的CloudFormationStackSet操作配置。

自行管理权限模型的示例

以下示例显示了输入的部署目标为 AWS 账户 ID 的CloudFormationStackSet操作。

YAML

```
Name: CreateStackSet
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormationStackSet
  Version: '1'
RunOrder: 1
Configuration:
  DeploymentTargets: '111111222222'
  FailureTolerancePercentage: '20'
  MaxConcurrentPercentage: '25'
  PermissionModel: SELF_MANAGED
  Regions: us-east-1
  StackSetName: my-stackset
  TemplatePath: 'SourceArtifact::template.json'
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: us-west-2
Namespace: DeployVariables
```

JSON

```
{
    "Name": "CreateStackSet",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormationStackSet",
        "Version": "1"
    },
    "RunOrder": 1,
    "Configuration": {
        "DeploymentTargets": "111111222222",
        "FailureTolerancePercentage": "20",
        "MaxConcurrentPercentage": "25",
        "PermissionModel": "SELF_MANAGED",
        "Regions": "us-east-1",
        "StackSetName": "my-stackset",
        "TemplatePath": "SourceArtifact::template.json"
    },
```

```
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "us-west-2",
    "Namespace": "DeployVariables"
}
```

服务管理权限模型的示例

以下示例显示了服务托管权限模型的CloudFormationStackSet操作，其中启用了自动部署到 Organizations AWS 的选项，同时保留堆栈。

YAML

```
Name: Deploy
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormationStackSet
  Version: '1'
RunOrder: 1
Configuration:
  Capabilities: 'CAPABILITY_IAM,CAPABILITY_NAMED_IAM'
  OrganizationsAutoDeployment: EnabledWithStackRetention
  PermissionModel: SERVICE_MANAGED
  StackSetName: stacks-orgs
  TemplatePath: 'SourceArtifact::template.json'
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: eu-central-1
Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "ActionTypeId": {
        "Category": "Deploy",
```

```
        "Owner": "AWS",
        "Provider": "CloudFormationStackSet",
        "Version": "1"
    },
    "RunOrder": 1,
    "Configuration": {
        "Capabilities": "CAPABILITY_IAM,CAPABILITY_NAMED_IAM",
        "OrganizationsAutoDeployment": "EnabledWithStackRetention",
        "PermissionModel": "SERVICE_MANAGED",
        "StackSetName": "stacks-orgs",
        "TemplatePath": "SourceArtifact::template.json"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "eu-central-1",
    "Namespace": "DeployVariables"
}
```

动 CloudFormationStackInstances作

此操作会创建新实例并将堆栈集部署到指定的实例。堆栈实例 是对区域内的目标账户中的堆栈的引用。堆栈实例可在没有堆栈的情况下存在；例如，如果堆栈创建不成功，则堆栈实例将显示堆栈创建失败的原因。一个堆栈实例仅与一个堆栈集关联。

初始创建堆栈集后，您可以使用 CloudFormationStackInstances 添加新的堆栈实例。创建或更新堆栈集实例操作期间，可以在堆栈实例级别覆盖模板参数值。

每个堆栈集都有一个模板和一组模板参数。更新模板或模板参数时，您应对整个堆栈集更新它们。然后，所有实例状态都将被设置为 OUTDATED，直到将更改部署到该实例为止。

要覆盖特定实例上的参数值，例如，如果模板包含一个值为 prod 的 stage 参数，则可以将该参数的值覆盖为 beta 或 gamma。

主题

- 操作类型
- 配置参数

- 输入构件
- 输出构件
- 输出变量
- 操作配置示例

操作类型

- 类别：Deploy
- 拥有者：AWS
- 提供方：CloudFormationStackInstances
- 版本：1

配置参数

StackSetName

必需：是

要与堆栈集关联的名称。此名称在创建它的区域中必须唯一。

名称中只能包含字母数字字符和连字符。它必须以字母字符开头，且不得超过 128 个字符。

DeploymentTargets

必需：否

> **Note**
>
> 对于 SERVICE_MANAGED 权限模型，您可以提供部署目标的组织根 ID 或组织单位 ID。对于 SELF_MANAGED 权限模型，您只能提供账户。

> **Note**
>
> 选择此参数后，还必须选择区域。

应在其中创建/更新堆栈集实例的 AWS 账户或组织单位 ID 列表。

- Accounts：

  您可以通过文本列表或文件路径的形式提供账户：

  - 文本：使用简写语法格式 `account_ID,account_ID` 输入参数，如以下示例所示。

    ```
    111111222222,333333444444
    ```

  - 文件路径：文件的位置，该文件包含应在其中创建/更新堆栈集实例的 AWS 帐户列表，格式为。`InputArtifactName::AccountsFileName`如果您使用文件路径来指定账户或 OrganizationalUnitIds，则文件格式必须采用 JSON 格式，如以下示例所示。

    ```
    SourceArtifact::accounts.txt
    ```

    以下示例显示 `accounts.txt` 的文件内容：

    ```
    [
        "111111222222"
    ]
    ```

    以下示例显示了列出多个账户时 `accounts.txt` 的文件内容：

    ```
    [
        "111111222222","333333444444"
    ]
    ```

- OrganizationalUnitIds:

  > **Note**
  >
  > 此参数对于 SERVICE_MANAGED 权限模型来说是可选的，不用于 SELF_MANAGED 权限模型。如果您选择，请不要使用此选项OrganizationsAutoDeployment。

  更新关联堆栈实例的 AWS 组织单位。

  您可以通过文本列表或文件路径的形式提供组织单位 ID。

  - 文本：输入用逗号分隔的字符串数组，如以下示例所示。

```
ou-examplerootid111-exampleouid111,ou-examplerootid222-exampleouid222
```

- 文件路径：文件的位置，其中包含要 OrganizationalUnitIds 在其中创建或更新堆栈集实例的列表。如果您使用文件路径来指定账户或 OrganizationalUnitIds，则文件格式必须采用 JSON 格式，如以下示例所示。

  按照 `InputArtifactName::OrganizationalUnitIdsFileName` 格式输入文件路径。

  ```
  SourceArtifact::OU-IDs.txt
  ```

  以下示例显示 `OU-IDs.txt` 的文件内容：

  ```
  [
      "ou-examplerootid111-exampleouid111","ou-examplerootid222-exampleouid222"
  ]
  ```

区域

必需：是

> **Note**
>
> 选择此参数后，还必须选择DeploymentTargets。

创建或更新堆栈集实例的 AWS 区域列表。区域将按照输入顺序进行更新。

以:的格式输入有效 AWS 区域列表`Region1,Region2`，如以下示例所示。

```
us-west-2,us-east-1
```

ParameterOverrides

必需：否

要在所选堆栈实例中覆盖的堆栈集参数的列表。被覆盖的参数值会应用于指定账户和区域中的所有堆栈实例。

您可以通过文本列表或文件路径的形式提供参数：

- 您可以使用以下简写语法格式输入参数：ParameterKey=string,ParameterValue=string,UsePreviousValue=boolean,Resolv
  ParameterKey=string,ParameterValue=string,UsePreviousValue=boolean,ResolvedV
  有关这些数据类型的更多信息，请参阅模板参数数据类型。

  以下示例显示了一个名为 BucketName 且值为 my-bucket 的参数。

  ```
  ParameterKey=BucketName,ParameterValue=my-bucket
  ```

  以下示例显示了一个具有多个参数的条目。

  ```
  ParameterKey=BucketName,ParameterValue=my-bucket
  ParameterKey=Asset1,ParameterValue=true
  ParameterKey=Asset2,ParameterValue=true
  ```

- 您可以按照 InputArtifactName::ParameterOverridessFileName 格式，输入包含模板参数覆盖列表的文件的位置，如以下示例所示。

  ```
  SourceArtifact::parameter-overrides.txt
  ```

  以下示例显示 parameter-overrides.txt 的文件内容。

  ```
  [
      {
          "ParameterKey": "KeyName",
          "ParameterValue": "true"
      },
      {
          "ParameterKey": "KeyName",
          "ParameterValue": "true"
      }
  ]
  ```

FailureTolerancePercentage

必需：否

在该区域 AWS CloudFormation 停止该堆栈操作之前，每个区域中该堆栈操作可能失败的账户百分比。如果操作在某个区域中停止，则 AWS CloudFormation 不会在后续区域中尝试该操作。根据指定的百分比计算账户数量时，向下 AWS CloudFormation 舍入到下一个整数。

MaxConcurrentPercentage

必需：否

一次执行此操作的账户数的最大百分比。根据指定的百分比计算账户数量时，向下 AWS CloudFormation 舍入到下一个整数。如果向下舍入结果为零，则改为将数字 AWS CloudFormation 设置为一。尽管您指定最大值，但对于大型部署，并行执行的实际账户的数量可能会因服务节流而减少。

RegionConcurrencyType

必需：否

您可以通过配置区域并发部署参数，指定堆栈集应该按照顺序方式还是并行方式跨 AWS 区域 进行部署。如果将区域并发性指定为 AWS 区域 并行部署多个堆栈，则可以缩短总体部署时间。

- 并行：只要区域的部署失败次数不超过指定的容错值，就会同时执行堆栈集部署。
- 顺序：只要区域的部署失败次数不超过指定的容错值，就会一次执行一个堆栈集部署。顺序部署是默认选项。

ConcurrencyMode

必需：否

并发模式允许您选择并发级别在堆栈集操作期间的行为方式，无论是严格容错还是软容错。严格容错会降低堆栈集操作失败时的部署速度，因为每次失败都会降低并发数。S@@ oft Fail ure Tolerance 优先考虑部署速度，同时仍能利用 AWS CloudFormation 安全功能。

- `STRICT_FAILURE_TOLERANCE`：此选项可动态降低并发级别，确保失败账户的数量永远不会超过特定的容错值。这是默认行为。
- `SOFT_FAILURE_TOLERANCE`：此选项将容错与实际并发数分离。这允许堆栈集操作在一个设定的并发级别上运行，而不管失败次数是多少。

CallAs

必需：否

> **Note**
>
> 此参数对于`SERVICE_MANAGED`权限模型是可选的，不用于`SELF_MANAGED`权限模型。

指定您是以组织的管理账户还是以成员账户的委托管理员身份行事。

> **Note**
>
> 如果此参数设置为DELEGATED_ADMIN，请确保管道 IAM 角色具有organizations:ListDelegatedAdministrators权限。否则，操作将在运行时失败，并显示类似于以下内容的错误：Account used is not a delegated administrator。

- SELF: 堆栈集部署将在登录管理账户时使用服务管理权限。
- DELEGATED_ADMIN: 堆栈集部署将在登录委托管理员账户时使用服务管理权限。

输入构件

CloudFormationStackInstances 可以包含列出部署目标和参数的构件。

- 构件数：0 to 2
- 描述：作为输入，堆栈集操作可以选择接受构件以用于下列用途：
  - 提供参数文件以供使用。（请参阅 ParameterOverrides 参数。）
  - 提供目标账户文件以供使用。（请参阅 DeploymentTargets 参数。）

输出构件

- 构件数：0
- 描述：输出构件不适用于此操作类型。

输出变量

配置后，此操作会生成变量，该变量可由管道中下游操作的操作配置引用。您可以使用命名空间配置操作，以使这些变量可用于下游操作的配置。

- StackSetId: 堆栈集的 ID。
- OperationId: 堆栈集操作的 ID。

有关更多信息，请参阅 Variables。

操作配置示例

以下示例显示了操作的CloudFormationStackInstances操作配置。

自行管理权限模型的示例

以下示例显示了一个CloudFormationStackInstances操作，其中输入的部署目标是 AWS 账户 ID 111111222222。

YAML

```
Name: my-instances
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormationStackInstances
  Version: '1'
RunOrder: 2
Configuration:
  DeploymentTargets: '111111222222'
  Regions: 'us-east-1,us-east-2,us-west-1,us-west-2'
  StackSetName: my-stackset
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: us-west-2
```

JSON

```
{
    "Name": "my-instances",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormationStackInstances",
        "Version": "1"
    },
    "RunOrder": 2,
    "Configuration": {
        "DeploymentTargets": "111111222222",
        "Regions": "us-east-1,us-east-2,us-west-1,us-west-2",
        "StackSetName": "my-stackset"
    },
```

```
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "us-west-2"
}
```

服务管理权限模型的示例

以下示例显示了服务管理权限模型的CloudFormationStackInstances操作，其中部署目标是 Organizations AWS 组织单位 ID `ou-1111-1example`。

YAML

```
Name: Instances
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormationStackInstances
  Version: '1'
RunOrder: 2
Configuration:
  DeploymentTargets: ou-1111-1example
  Regions: us-east-1
  StackSetName: my-stackset
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: eu-central-1
```

JSON

```
{
    "Name": "Instances",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormationStackInstances",
        "Version": "1"
    },
```

```
    "RunOrder": 2,
    "Configuration": {
        "DeploymentTargets": "ou-1111-1example",
        "Regions": "us-east-1",
        "StackSetName": "my-stackset"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "eu-central-1"
}
```

堆栈集操作的权限模型

由于跨多个账户 AWS CloudFormation StackSets 执行操作，因此必须先在这些账户中定义必要的权限，然后才能创建堆栈集。您可以通过自管理权限或服务管理权限来定义权限。

使用自我管理权限，您可以创建所需的两个 IAM 角色：一个是管理员角色，例如定义堆栈集的账户 AWSCloudFormationStackSetAdministrationRole 中的，另一个是执行角色，例如部署堆栈集实例的每个账户 AWSCloudFormationStackSetExecutionRole 中的。 StackSets 使用此权限模型，StackSets 可以部署到用户有权创建 IAM 角色的任何 AWS 账户。有关更多信息，请参阅 AWS CloudFormation 用户指南 中的创建自管理权限。

> **Note**
>
> 由于跨多个账户 AWS CloudFormation StackSets 执行操作，因此必须先在这些账户中定义必要的权限，然后才能创建堆栈集。

借助服务管理权限，您可以将堆栈实例部署到由 Organizations 管理的账户 AWS 。使用此权限模型，您无需创建必要的 IAM 角色，因为您可以代表您 StackSets 创建 IAM 角色。使用此模型，您还可以启用向将来添加到组织的账户自动部署的功能。请参阅《AWS CloudFormation 用户指南》中的 “通过 Org AWS anizations 启用可信访问”。

模板参数数据类型

堆栈集操作中使用的模板参数包括以下数据类型。有关更多信息，请参阅DescribeStackSet。

ParameterKey

- 描述：与参数关联的键。如果您没有为特定参数指定键和值，则 AWS CloudFormation 使用模板中指定的默认值。
- 例如：

```
"ParameterKey=BucketName,ParameterValue=my-bucket"
```

ParameterValue

- 描述：与参数关联的输入值。
- 例如：

```
"ParameterKey=BucketName,ParameterValue=my-bucket"
```

UsePreviousValue

- 在堆栈更新期间，使用堆栈对给定参数键使用的现有参数值。如果指定 `true`，请勿指定参数值。
- 例如：

```
"ParameterKey=Asset1,UsePreviousValue=true"
```

每个堆栈集都有一个模板和一组模板参数。更新模板或模板参数时，您应对整个堆栈集更新它们。然后，所有实例状态都将被设置为 OUTDATED，直到将更改部署到该实例为止。

要覆盖特定实例上的参数值，例如，如果模板包含一个值为 `prod` 的 `stage` 参数，则可以将该参数的值覆盖为 `beta` 或 `gamma`。

另请参阅

下列相关资源在您使用此操作的过程中会有所帮助。

- 参数类型 —《AWS CloudFormation 用户指南》中的本参考章节提供了 CloudFormation 模板参数的更多描述和示例。
- 最佳实践：有关部署堆栈集的最佳实践的更多信息，请参阅 AWS CloudFormation 用户指南 中的 https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacksets-bestpractices.html。
- AWS CloudFormation API 参考 — 您可以参考 AWS CloudFormation API 参考中的以下 CloudFormation 操作，了解有关堆栈集操作中使用的参数的更多信息：

- 该CreateStackSet操作会创建堆栈集。
- 该UpdateStackSet操作更新了指定账户和区域中的堆栈集和关联堆栈实例。即使通过更新堆栈集创建的堆栈集操作失败（完全失败或部分失败，低于或高于指定的容错值），也会使用这些更改来更新堆栈集。对指定堆栈集的后续 CreateStackInstances 调用使用更新的堆栈集。
- 该CreateStackInstances操作在自行管理的权限模型上为所有指定账户中的所有指定区域创建堆栈实例，或者在服务管理的权限模型上为所有指定部署目标中的所有指定区域创建堆栈实例。您可以覆盖通过此操作创建的实例的参数。如果实例已经存在，则使用相同的输入参数 UpdateStackInstances 进行 CreateStackInstances 调用。当您使用此操作创建实例时，不会更改其他堆栈实例的状态。
- 该UpdateStackInstances操作使堆栈实例与在自管理权限模型上所有指定账户中的所有指定区域或服务托管权限模型上的所有指定部署目标内的所有指定部署目标中的堆栈集保持同步。您可以覆盖通过此操作更新的实例的参数。当您使用此操作更新实例子集时，不会更改其他堆栈实例的状态。
- 该DescribeStackSetOperation操作返回指定堆栈集操作的描述。
- 该DescribeStackSet操作返回指定堆栈集的描述。

AWS CodeBuild

允许您将构建和测试作为管道的一部分来运行。运行 CodeBuild 生成或测试操作时，构建规范中指定的命令将在 CodeBuild 容器内运行。所有被指定为 CodeBuild 操作输入工件的工件都可以在运行命令的容器中找到。 CodeBuild 可以提供生成或测试操作。有关更多信息，请参阅《AWS CodeBuild 用户指南》。

当您在控制台中使用 CodePipeline 向导创建构建项目时， CodeBuild 生成项目会显示源提供者是 CodePipeline。在 CodeBuild控制台中创建构建项目时，您无法指定 CodePipeline 为源代码提供者，但是向管道中添加生成操作会调整 CodeBuild 控制台中的源代码。有关更多信息，请参阅 AWS CodeBuild API 参考ProjectSource中的。

主题

- 操作类型
- 配置参数
- 输入构件
- 输出构件
- 输出变量
- 操作声明（CodeBuild 示例）

- 另请参阅

操作类型

- 类别：Build 或 Test
- 拥有者：AWS
- 提供方：CodeBuild
- 版本：1

配置参数

ProjectName

必需：是

ProjectName是中构建项目的名称 CodeBuild。

PrimarySource

必需：条件

PrimarySource参数的值必须是该操作的其中一个输入项目的名称。 CodeBuild 查找构建规范文件并在包含此工件的解压缩版本的目录中运行构建规范命令。

如果为一个 CodeBuild 操作指定了多个输入对象，则此参数为必填项。当操作仅有一个源构件时，PrimarySource 构件默认为该构件。

BatchEnabled

必需：否

BatchEnabled 参数的布尔值允许操作在同一个构建执行中运行多个构建。

启用此选项后，CombineArtifacts 选项将变为可用。

有关启用批量构建的管道示例，请参阅CodePipeline 集成 CodeBuild 和批量构建。

CombineArtifacts

必需：否

CombineArtifacts 参数的布尔值将来自一个批量构建的所有构建构件合并为单个构件文件，将其用于构建操作。

要使用此选项，必须启用 BatchEnabled 参数。

EnvironmentVariables

必需：否

此参数的值用于为管道中的 CodeBuild 操作设置环境变量。EnvironmentVariables 参数的值采用环境变量对象的 JSON 数组形式。请参阅操作声明（CodeBuild 示例）中的示例参数。

每个对象有三个均为字符串的部分：

- name：环境变量的名称或键。
- value：环境变量的值。使用PARAMETER_STORE或SECRETS_MANAGER类型时，此值必须分别是您已经存储在 Sy AWS stems Manager 参数存储中的参数的名称或已经存储在 Secrets Manager 中的 AWS 密钥。

> **Note**
>
> 我们强烈建议不要使用环境变量存储敏感值，尤其是 AWS 凭证。使用 CodeBuild 控制台或 AWS CLI 时，环境变量将以纯文本形式显示。对于敏感值，我们建议您改用 SECRETS_MANAGER 类型。

- type：（可选）环境变量的类型。有效值为 PARAMETER_STORE、SECRETS_MANAGER 或 PLAINTEXT。如果未指定，则此值默认为 PLAINTEXT。

> **Note**
>
> 当您type为环境变量配置输入namevalue、和时，尤其是在环境变量包含 CodePipeline 输出变量语法的情况下，请不要超过配置值字段的 1000 个字符限制。如果超过此限制，将会返回验证错误。

有关更多信息，请参阅 AWS CodeBuild API 参考 EnvironmentVariable中的。有关带有可解析为 GitHub 分支名称的环境变量的 CodeBuild 操作示例，请参阅示例：将 BranchName变量与 CodeBuild 环境变量一起使用。

输入构件

- 构件数：1 to 5

- 描述： CodeBuild 查找构建规范文件并从主源构件的目录中运行构建规范命令。如果为 CodeBuild 操作指定了多个输入源，则必须使用中的`PrimarySource`操作配置参数来设置此对象 CodePipeline。

  各个输入构件会解压缩到各自的目录，其位置存储在环境变量中。主源构件的目录通过 `$CODEBUILD_SRC_DIR` 提供。所有其他输入构件的目录通过 `$CODEBUILD_SRC_DIR_yourInputArtifactName` 提供。

  > **Note**
  > 在 CodeBuild 项目中配置的构件将成为管道中 CodeBuild 操作使用的输入构件。

输出构件

- 构件数：`0 to 5`
- 描述：它们可用于使 CodeBuild 构建规范文件中定义的工件可供管道中的后续操作使用。在仅定义了一个输出构件时，此构件可以直接在构建规范文件的 `artifacts` 部分下定义。在指定了多个输出构件时，引用的所有构件都必须在构建规范文件中定义为辅助构件。中输出工件的名称 CodePipeline 必须与构建规范文件中的构件标识符相匹配。

  > **Note**
  > 在 CodeBuild 项目中配置的构件将成为管道操作中的 CodePipeline 输入构件。

  如果为批量构建选择了 `CombineArtifacts` 参数，则输出构件位置将包含在同一执行中运行的多个构建的合并构件。

输出变量

此操作将生成在构建中导出的所有环境变量作为变量。有关如何导出环境变量的更多详细信息，请参阅 AWS CodeBuild API 指南 EnvironmentVariable中的。

有关在中使用 CodeBuild 环境变量的更多信息 CodePipeline，请参阅中的示例CodeBuild 操作输出变量。有关可在中使用的环境变量的列表 CodeBuild，请参阅《AWS CodeBuild 用户指南》中的构建环境中的环境变量。

操作声明（CodeBuild 示例）

YAML

```
Name: Build
Actions:
  - Name: PackageExport
    ActionTypeId:
      Category: Build
      Owner: AWS
      Provider: CodeBuild
      Version: '1'
    RunOrder: 1
    Configuration:
      BatchEnabled: 'true'
      CombineArtifacts: 'true'
      ProjectName: my-build-project
      PrimarySource: MyApplicationSource1
      EnvironmentVariables:
 '[{"name":"TEST_VARIABLE","value":"TEST_VALUE","type":"PLAINTEXT"},
{"name":"ParamStoreTest","value":"PARAMETER_NAME","type":"PARAMETER_STORE"}]'
    OutputArtifacts:
      - Name: MyPipeline-BuildArtifact
    InputArtifacts:
      - Name: MyApplicationSource1
      - Name: MyApplicationSource2
```

JSON

```
{
    "Name": "Build",
    "Actions": [
        {
            "Name": "PackageExport",
            "ActionTypeId": {
                "Category": "Build",
                "Owner": "AWS",
                "Provider": "CodeBuild",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
```

```
                "BatchEnabled": "true",
                "CombineArtifacts": "true",
                "ProjectName": "my-build-project",
                "PrimarySource": "MyApplicationSource1",
                "EnvironmentVariables": "[{\"name\":\"TEST_VARIABLE\",\"value\":
\"TEST_VALUE\",\"type\":\"PLAINTEXT\"},{\"name\":\"ParamStoreTest\",\"value\":
\"PARAMETER_NAME\",\"type\":\"PARAMETER_STORE\"}]"
            },
            "OutputArtifacts": [
                {
                    "Name": "MyPipeline-BuildArtifact"
                }
            ],
            "InputArtifacts": [
                {
                    "Name": "MyApplicationSource1"
                },
                {
                    "Name": "MyApplicationSource2"
                }
            ]
        }
    ]
}
```

另请参阅

下列相关资源在您使用此操作的过程中会有所帮助。

- AWS CodeBuild 用户指南-有关带有 CodeBuild 操作的管道示例，请参阅 CodePipeline 与一起使用 CodeBuild 来测试代码和运行构建。有关具有多个输入和输出 CodeBuild 构件的项目的示例，请参阅与 CodeBuild 多个输入源CodePipeline集成和输出构件示例以及多个输入源和输出构件示例。
- 教程：创建用于构建和测试您的 Android 应用程序的管道 AWS Device Farm— 本教程提供了示例构建规范文件和示例应用程序，用于创建管道，其 GitHub 源代码使用和构建和测试 Android 应用程序 AWS Device Farm。 CodeBuild
- 的编译规范参考 CodeBuild — 本参考主题为理解 CodeBuild 编译规范文件提供了定义和示例。有关可在中使用的环境变量的列表 CodeBuild，请参阅《AWS CodeBuild 用户指南》中的构建环境中的环境变量。

CodeCommit

在配置的 CodeCommit 存储库和分支上进行新提交时启动管道。

如果您使用控制台创建或编辑管道，则 CodePipeline 会创建一个 CodeCommit CloudWatch 事件规则，该规则将在存储库发生更改时启动您的管道。

在通过 CodeCommit 操作连接管道之前，您必须已经创建了 CodeCommit 存储库。

检测到代码更改后，您可以使用以下选项将代码传递给后续操作：

- 默认-将 CodeCommit 源操作配置为输出包含提交浅表副本的 ZIP 文件。
- 完整克隆：配置源操作，使之输出对仓库的 Git URL 引用，以供后续操作使用。

  目前，Git URL 引用只能由下游 CodeBuild 操作用于克隆存储库和关联的 Git 元数据。尝试将 Git URL 引用传递给非CodeBuild 操作会导致错误。

主题

操作类型

- 类别：`Source`
- 拥有者：`AWS`
- 提供方：`CodeCommit`
- 版本：`1`

配置参数

RepositoryName

必需：是

要在其中检测源更改的存储库的名称。

BranchName

必需：是

要在其中检测源更改的分支的名称。

PollForSourceChanges

必需：否

`PollForSourceChanges`控制是否 CodePipeline 轮询 CodeCommit存储库以获取源代码更改。我们建议您改用 “ CloudWatch 事件” 来检测源代码的更改。有关配置 CloudWatch 事件的更多信息，请参阅迁移轮询管道（CodeCommit 来源）(CLI)或迁移轮询管道（CodeCommit 来源）（AWS CloudFormation 模板）。

> **Important**
>
> 如果要配置 CloudWatch 事件规则，则必须将设置为`PollForSourceChanges``false`以避免重复的管道执行。

此参数的有效值：

- `true`: 如果已设置，则会 CodePipeline 轮询您的存储库以了解源代码更改。

  > **Note**
  >
  > 如果省略`PollForSourceChanges`，则 CodePipeline 默认为轮询存储库是否有源更改。如果包括 `PollForSourceChanges` 并将其设置为 `true`，则此行为相同。

- `false`: 如果已设置，则 CodePipeline 不会轮询您的存储库以了解源代码更改。如果您打算配置 CloudWatch 事件规则以检测源更改，请使用此设置。

OutputArtifactFormat

必需：否

输出构件格式。值可以是 CODEBUILD_CLONE_REF 或 CODE_ZIP。如果未指定，则默认为 CODE_ZIP。

> **Important**
>
> 该CODEBUILD_CLONE_REF选项只能由 CodeBuild下游操作使用。
> 如果选择此选项，则需要向您的 CodeBuild 服务角色添加codecommit:GitPull权限，如所示为 CodeCommit源操作添加 CodeBuild GitClone 权限。您还需要为 CodePipeline服务角色添加codecommit:GetRepository权限，如所示向 CodePipeline 服务角色添加权限。如需查看教程以了解如何使用完整克隆选项，请参阅教程：使用带有 CodeCommit 管道源的完整克隆。

输入构件

- 构件数：0
- 描述：输入构件不适用于此操作类型。

输出构件

- 构件数：1
- 描述：此操作的输出构件是一个 ZIP 文件，其中包含在提交时，指定作为管道执行的源修订的已配置存储库和分支的内容。从存储库生成的构件是 CodeCommit 操作的输出对象。源代码提交 ID 显示 CodePipeline 为触发管道执行的源修订版。

输出变量

配置后，此操作会生成变量，该变量可由管道中下游操作的操作配置引用。此操作生成的变量可视为输出变量，即使操作没有命名空间也是如此。您可以使用命名空间配置操作，以使这些变量可用于下游操作的配置。

有关更多信息，请参阅 Variables。

CommitId

触发管道执行的 CodeCommit 提交 ID。提交 ID 是提交的完整 SHA。

CommitMessage

与触发管道执行的提交相关联的描述消息（如果有）。

RepositoryName

触发管道的提交所在 CodeCommit 存储库的名称。

BranchName

进行源代码更改的 CodeCommit 存储库的分支名称。

AuthorDate

授权提交的日期，采用时间戳格式。

有关 Git 中作者和提交者之间区别的更多信息，请参阅 Pro Git 中由 Scott Chacon 和 Ben Straub 撰写的查看提交历史记录。

CommitterDate

进行提交的日期，采用时间戳格式。

有关 Git 中作者和提交者之间区别的更多信息，请参阅 Pro Git 中由 Scott Chacon 和 Ben Straub 撰写的查看提交历史记录。

操作配置示例

默认输出构件格式的示例

YAML

```
Actions:
  - OutputArtifacts:
      - Name: Artifact_MyWebsiteStack
    InputArtifacts: []
    Name: source
    Configuration:
      RepositoryName: MyWebsite
      BranchName: main
      PollForSourceChanges: 'false'
    RunOrder: 1
    ActionTypeId:
```

```
      Version: '1'
      Provider: CodeCommit
      Category: Source
      Owner: AWS
  Name: Source
```

JSON

```
{
    "Actions": [
        {
            "OutputArtifacts": [
                {
                    "Name": "Artifact_MyWebsiteStack"
                }
            ],
            "InputArtifacts": [],
            "Name": "source",
            "Configuration": {
                "RepositoryName": "MyWebsite",
                "BranchName": "main",
                "PollForSourceChanges": "false"
            },
            "RunOrder": 1,
            "ActionTypeId": {
                "Version": "1",
                "Provider": "CodeCommit",
                "Category": "Source",
                "Owner": "AWS"
            }
        }
    ],
    "Name": "Source"
},
```

完整克隆输出构件格式的示例

YAML

```
name: Source
actionTypeId:
  category: Source
```

```
  owner: AWS
  provider: CodeCommit
  version: '1'
runOrder: 1
configuration:
  BranchName: main
  OutputArtifactFormat: CODEBUILD_CLONE_REF
  PollForSourceChanges: 'false'
  RepositoryName: MyWebsite
outputArtifacts:
  - name: SourceArtifact
inputArtifacts: []
region: us-west-2
namespace: SourceVariables
```

JSON

```
{
    "name": "Source",
    "actionTypeId": {
        "category": "Source",
        "owner": "AWS",
        "provider": "CodeCommit",
        "version": "1"
    },
    "runOrder": 1,
    "configuration": {
        "BranchName": "main",
        "OutputArtifactFormat": "CODEBUILD_CLONE_REF",
        "PollForSourceChanges": "false",
        "RepositoryName": "MyWebsite"
    },
    "outputArtifacts": [
        {
            "name": "SourceArtifact"
        }
    ],
    "inputArtifacts": [],
    "region": "us-west-2",
    "namespace": "SourceVariables"
}
```

另请参阅

下列相关资源在您使用此操作的过程中会有所帮助。

- 教程：创建简单的管道（CodeCommit存储库）— 本教程提供了示例应用程序规范文件以及示例 CodeDeploy 应用程序和部署组。使用本教程创建具有部署到 Amazon EC2 实例的 CodeCommit 源的管道。

AWS CodeDeploy

您可以使用 AWS CodeDeploy 操作将应用程序代码部署到您的部署队列。您的部署实例集可以由 Amazon EC2 实例和/或本地实例组成。

> **Note**
>
> 本参考主题介绍 CodeDeploy 部署平台为 CodePipeline Amazon EC2 的部署操作。有关 CodeDeploy蓝/绿部署操作的 Amazon 弹性容器服务的参考信息 CodePipeline，请参阅。Amazon 弹性容器服务和 CodeDeploy 蓝绿色

主题

操作类型

- 类别：`Deploy`
- 拥有者：`AWS`
- 提供方：`CodeDeploy`
- 版本：`1`

配置参数

ApplicationName

必需：是

您在中创建的应用程序的名称 CodeDeploy。

DeploymentGroupName

必需：是

您在中创建的部署组 CodeDeploy。

输入构件

- 构件数：1
- 描述： CodeDeploy 用于确定以下内容的 AppSpec 文件：
  - 从 Amazon S3 中的应用程序修订版中将哪些安装到您的实例上，或者 GitHub。
  - 为响应部署生命周期事件而要运行的生命周期事件挂钩。

有关该 AppSpec 文件的更多信息，请参阅CodeDeploy AppSpec 文件参考。

输出构件

- 构件数：0
- 描述：输出构件不适用于此操作类型。

操作声明

YAML

```
Name: Deploy
Actions:
  - Name: Deploy
    ActionTypeId:
      Category: Deploy
```

```
      Owner: AWS
      Provider: CodeDeploy
      Version: '1'
    RunOrder: 1
    Configuration:
      ApplicationName: my-application
      DeploymentGroupName: my-deployment-group
    OutputArtifacts: []
    InputArtifacts:
      - Name: SourceArtifact
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
        {
            "Name": "Deploy",
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "CodeDeploy",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "ApplicationName": "my-application",
                "DeploymentGroupName": "my-deployment-group"
            },
            "OutputArtifacts": [],
            "InputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "Region": "us-west-2",
            "Namespace": "DeployVariables"
        }
    ]
},
```

另请参阅

下列相关资源在您使用此操作的过程中会有所帮助。

- 教程：创建一个简单的管道（S3 存储桶）— 本教程将引导您创建源存储桶、EC2 实例和 CodeDeploy 资源，以部署示例应用程序。然后，您可以使用 CodeDeploy 部署操作来构建您的管道，该操作会将 S3 存储桶中维护的代码部署到您的 Amazon EC2 实例。
- 教程：创建简单的管道（CodeCommit存储库）— 本教程将引导您创建 CodeCommit 源存储库、EC2 实例和部署示例应用程序 CodeDeploy 所需的资源。然后，您可以使用部署操作来构建管道，该操作将代码从您的 CodeCommit 存储库 CodeDeploy 部署到您的 Amazon EC2 实例。
- CodeDeploy AppSpec 文件引用 —《AWS CodeDeploy 用户指南》中的本参考章节提供了 CodeDeploy AppSpec 文件的参考信息和示例。

CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、GitHub 企业服务器、 GitLab .com 和 GitLab 自我管理操作

支持连接的源操作 AWS CodeConnections。 CodeConnections 允许您创建和管理 AWS 资源与第三方存储库（例如）之间的连接 GitHub。在第三方源代码存储库上进行新提交时启动管道。当手动执行管道或从源提供程序发送 Webhook 事件时，源操作会检索代码更改。

您可以将管道中的操作配置为使用 Git 配置，该配置允许您使用触发器启动管道。要将管道触发器配置配置配置为使用触发器进行筛选，请在中查看更多详细信息筛选代码推送或拉取请求的触发器。

> **Note**
>
> 此功能不适用于亚太地区（香港）、亚太地区（海得拉巴）、亚太地区（雅加达）、亚太地区（墨尔本）、亚太地区（墨尔本）、亚太地区（大阪）、非洲（开普敦）、中东（巴林）、中东（阿联酋）、欧洲（西班牙）、欧洲（苏黎世）、以色列（特拉维夫）或 AWS GovCloud（美国西部）地区。要参考其他可用操作，请参阅 产品和服务与 CodePipeline。有关在欧洲地区（米兰）区域使用此操作的注意事项，请参阅CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、 GitLab .com 和 GitLab 自我管理操作中的说明。

Connections 可以将您的 AWS 资源与以下第三方存储库相关联：

- Bitbucket Cloud（通过 CodePipeline 控制台中的 Bitbuck et 提供者选项或 CLI 中的`Bitbucket`提供者）

> **Note**
>
> 您可以创建到 Bitbucket Cloud 存储库的连接。不支持已安装的 Bitbucket 提供程序类型（如 Bitbucket 服务器）。

-

> **Note**
>
> 如果您使用的是 Bitbucket 工作区，则必须具有管理员访问权限才能创建连接。

- GitHub 和 GitHub 企业云（通过 CodePipeline 控制台中的 GitHub （版本 2）提供商选项或 CLI 中的`GitHub`提供商）

> **Note**
>
> 如果您的仓库位于 GitHub 组织中，则必须是组织所有者才能创建连接。如果使用不位于组织中的存储库，则您必须是存储库拥有者。

- GitHub 企业服务器（通过 CodePipeline 控制台中的GitHub 企业服务器提供者选项或 CLI 中的`GitHub Enterprise Server`提供者）
- GitLab.com（通过 CodePipeline控制台中的GitLab提供者选项或 CLI 中的`GitLab`提供者）

> **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 CodePipeline。对于群组中的仓库，您无需成为群组拥有者。

- GitLab （企业版或社区版）的自行管理安装（通过 CodePipeline控制台中的GitLab 自我管理提供商选项或 CLI 中的`GitLabSelfManaged`提供程序）

> **Note**
>
> 每个连接都支持您在该提供方处拥有的所有存储库。您只需为每个提供方类型创建一个新连接。

借助连接，您的管道可以通过第三方提供商的安装应用来检测源更改。例如，webhook 用于订阅 GitHub 事件类型，并且可以安装在组织、存储库或 GitHub 应用程序上。您的连接会在您的 GitHub 应用程序上安装一个订阅 GitHub 推送类型事件的存储库 webhook。

检测到代码更改后，您可以使用以下选项将代码传递给后续操作：

- 默认：与其他现有 CodePipeline 源操作一样，`CodeStarSourceConnection`可以输出包含提交浅表副本的 ZIP 文件。
- 完整克隆：也可以对 `CodeStarSourceConnection` 进行配置，以便为后续操作输出对存储库的 URL 引用。

  目前，Git URL 引用只能由下游 CodeBuild 操作用于克隆存储库和关联的 Git 元数据。尝试将 Git URL 引用传递给非CodeBuild 操作会导致错误。

CodePipeline 创建 AWS 连接时，会提示您将 Connector 安装应用程序添加到您的第三方帐户。您必须事先创建自己的第三方账户和存储库，然后才能通过 `CodeStarSourceConnection` 操作进行连接。

> **Note**
>
> 要为您的角色创建或附加一项包含使用 AWS CodeStar 连接所需权限的策略，请参阅连接权限参考。根据您的 CodePipeline 服务角色的创建时间，您可能需要更新其权限以支持 AWS CodeStar 连接。有关说明，请参阅向 CodePipeline 服务角色添加权限。

> **Note**
>
> 要在欧洲（米兰）使用连接 AWS 区域，您必须：
>
> 1. 安装区域特定的应用程序
> 2. 启用该区域
>
> 这一特定于区域的应用程序支持欧洲地区（米兰）区域中的连接。该应用程序在第三方提供商网站上发布，与支持其他区域的连接的现有应用程序是分开的。安装此应用程序，即表示您授权第三方提供商仅与该区域的服务共享您的数据，并且您可以随时通过卸载该应用程序来撤消权限。

除非您启用区域，否则该服务不会处理或存储您的数据。启用此区域，即表示您授予我们的服务处理和存储您的数据的权限。
即使未启用该区域，如果区域特定的应用程序仍保持安装状态，第三方提供商也仍可以与我们的服务共享您的数据，因此，请务必在禁用该区域后立即卸载该应用程序。有关更多信息，请参阅启用区域。

主题

操作类型

- 类别：Source
- 拥有者：AWS
- 提供方：CodeStarSourceConnection
- 版本：1

配置参数

ConnectionArn

必需：是

为源提供程序进行配置和身份验证的连接 ARN。

FullRepositoryId

必需：是

要在其中检测源更改的存储库的拥有者和名称。

例如：`some-user/my-repo`

> **Important**
>
> 您必须保持该FullRepositoryId值的正确大小写。例如，如果您的用户名为`some-user`，存储库名称为`My-Repo`，则建议的FullRepositoryId值为。`some-user/My-Repo`

BranchName

必需：是

要在其中检测源更改的分支的名称。

OutputArtifactFormat

必需：否

指定输出构件格式。可以是 `CODEBUILD_CLONE_REF` 或 `CODE_ZIP`。如果未指定，则默认为 `CODE_ZIP`。

> **Important**
>
> 该`CODEBUILD_CLONE_REF`选项只能由 CodeBuild下游操作使用。
> 如果选择此选项，则需要更新 CodeBuild 项目服务角色的权限，如所示添加连接 Bitbucket、 GitHub、En GitHub terprise Server 或 GitLab .com 的 CodeBuild GitClone 权限。如需查看教程以了解如何使用完整克隆选项，请参阅教程：使用带有 GitHub 管道源的完整克隆。

DetectChanges

必需：否

指在所配置的存储库和分支上进行新提交时，会自动启动管道的控制设置。如果未指定，则默认值为 `true`，并且默认情况下不显示该字段。此参数的有效值：

- `true`: 在新提交时 CodePipeline 自动启动您的管道。
- `false`: CodePipeline 不会在新提交时启动您的管道。

输入构件

- 构件数：0
- 描述：输入构件不适用于此操作类型。

输出构件

- 构件数：1
- 描述：从存储库生成的构件是 CodeStarSourceConnection 操作的输出构件。源代码提交 ID 显示 CodePipeline 为触发管道执行的源修订版。您可以在以下文件中配置此操作的输出构件：
  - ZIP 文件，其中包含在提交时，指定作为管道执行的源修订的已配置存储库和分支的内容。
  - JSON 文件，其中包含存储库的 URL 引用，以便下游操作可以直接执行 Git 命令。

> **Important**
>
> 此选项只能由 CodeBuild 下游操作使用。
> 如果选择此选项，则需要更新 CodeBuild 项目服务角色的权限，如所示故障排除 CodePipeline。如需查看教程以了解如何使用完整克隆选项，请参阅教程：使用带有 GitHub 管道源的完整克隆。

输出变量

配置后，此操作会生成变量，该变量可由管道中下游操作的操作配置引用。此操作生成的变量可视为输出变量，即使操作没有命名空间也是如此。您可以使用命名空间配置操作，以使这些变量可用于下游操作的配置。

有关更多信息，请参阅 Variables。

AuthorDate

授权提交的日期，采用时间戳格式。

BranchName

进行源更改的 存储库的分支名称。

CommitId

触发管道执行的 提交 ID。

CommitMessage

与触发管道执行的提交相关联的描述消息（如果有）。

ConnectionArn

为源提供程序进行配置和身份验证的连接 ARN。

FullRepositoryName

发出创建触发管道的提交的 存储库名称。

操作声明

在以下示例中，对于 ARN 为 arn:aws:codestar-connections:region:*account-id*:connection/*connection-id* 的连接，输出构件被设置为默认的 CODE_ZIP 压缩格式。

YAML

```
Name: Source
Actions:
  - InputArtifacts: []
    ActionTypeId:
      Version: '1'
      Owner: AWS
      Category: Source
      Provider: CodeStarSourceConnection
    OutputArtifacts:
      - Name: SourceArtifact
    RunOrder: 1
    Configuration:
      ConnectionArn: "arn:aws:codestar-connections:region:account-
id:connection/connection-id"
      FullRepositoryId: "some-user/my-repo"
      BranchName: "main"
      OutputArtifactFormat: "CODE_ZIP"
    Name: ApplicationSource
```

JSON

```
{
    "Name": "Source",
```

```
    "Actions": [
        {
            "InputArtifacts": [],
            "ActionTypeId": {
                "Version": "1",
                "Owner": "AWS",
                "Category": "Source",
                "Provider": "CodeStarSourceConnection"
            },
            "OutputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "RunOrder": 1,
            "Configuration": {
                "ConnectionArn": "arn:aws:codestar-connections:region:account-
id:connection/connection-id",
                "FullRepositoryId": "some-user/my-repo",
                "BranchName": "main",
                "OutputArtifactFormat": "CODE_ZIP"
            },
            "Name": "ApplicationSource"
        }
    ]
},
```

安装安装应用并创建连接

首次使用控制台向第三方存储库添加新连接时，必须授予对存储库的 CodePipeline 访问权限。您选择或创建一个安装应用程序，以帮助您连接到创建第三方代码存储库的账户。

使用 AWS CLI 或 AWS CloudFormation 模板时，必须提供已通过安装握手的连接的连接 ARN。否则，不会触发管道。

> **Note**
>
> 对于 `CodeStarSourceConnection` 源操作，您不必设置 Webhook 或默认进行轮询。连接操作会为您管理更改检测。

另请参阅

下列相关资源在您使用此操作的过程中会有所帮助。

- AWS::CodeStarConnections::Connection— Connect AWS CodeStar ions 资源的 AWS CloudFormation 模板参考提供了 AWS CloudFormation 模板中连接的参数和示例。
- AWS CodeStar连接 API 参考 —《AWS CodeStar 连接 API 参考》提供了可用连接操作的参考信息。
- 要查看使用连接支持的源操作来创建管道的步骤，请参阅以下内容：
  - 对于 Bitbucket Cloud，请使用控制台中的 Bitbucket 选项或 CLI 中的 `CodestarSourceConnection` 操作。请参阅 Bitbucket Cloud 连接。
  - 对于 GitHub 和 GitHub 企业云，请使用控制台中的GitHub提供者选项或 CLI 中的`CodestarSourceConnection`操作。请参阅 GitHub 连接。
  - 对于 GitHub 企业服务器，请使用控制台中的GitHub 企业服务器提供者选项或 CLI 中的`CodestarSourceConnection`操作。请参阅 GitHub 企业服务器连接。
  - 对于 GitLab .com，请使用控制台中的GitLab提供者选项或在 CLI 中使用`GitLab`提供程序的`CodestarSourceConnection`操作。请参阅 GitLab.com 连接。
- 要查看使用 Bitbucket 源和 CodeBuild 操作创建管道的入门教程，请参阅连接入门。
- 有关向您展示如何连接到 GitHub 存储库以及如何使用带有下游 CodeBuild 操作的 “完全克隆” 选项的教程，请参阅教程：使用带有 GitHub 管道源的完整克隆。

AWS Device Farm

在您的管道中，您可以配置用于 AWS Device Farm 在设备上运行和测试应用程序的测试操作。Device Farm 使用设备测试池和测试框架在特定设备上测试应用程序。有关 Device Farm 操作支持的测试框架类型的信息，请参阅在 Device Farm 中 AWS 使用测试类型。

主题

操作类型

- 类别：Test
- 拥有者：AWS
- 提供方：DeviceFarm
- 版本：1

配置参数

AppType

必需：是

您要测试的操作系统和应用程序类型。以下是有效值列表：

- iOS
- Android
- Web

ProjectId

必需：是

Device Farm 项目 ID。

要查找您的项目 ID，请在 Device Farm 控制台中选择您的项目。在浏览器中，复制新项目的 URL。URL 包含项目 ID。项目 ID 是 URL 中位于 projects/ 之后的值。在下面的示例中，项目 ID 为 eec4905f-98f8-40aa-9afc-4c1cfexample。

```
https://<region-URL>/devicefarm/home?region=us-west-2#/projects/
eec4905f-98f8-40aa-9afc-4c1cfexample/runs
```

App

必需：是

您的输入构件中应用程序文件的名称和位置。例如：s3-ios-test-1.ipa

TestSpec

条件：是

测试规范定义文件在输入构件中的位置。这是自定义模式测试所必需的。

DevicePoolArn

必需：是

Device Farm 设备池 ARN。

要获取项目可用的设备池 ARN，包括热门设备的 ARN，请 AWS 使用 CLI 输入以下命令：

```
aws devicefarm list-device-pools --arn arn:aws:devicefarm:us-
west-2:account_ID:project:project_ID
```

TestType

必需：是

为您的测试指定受支持的测试框架。以下是 TestType 的有效值列表：

- APPIUM_JAVA_JUNIT
- APPIUM_JAVA_TESTNG
- APPIUM_NODE
- APPIUM_RUBY
- APPIUM_PYTHON
- APPIUM_WEB_JAVA_JUNIT
- APPIUM_WEB_JAVA_TESTNG
- APPIUM_WEB_NODE
- APPIUM_WEB_RUBY
- APPIUM_WEB_PYTHON
- BUILTIN_FUZZ
- INSTRUMENTATION
- XCTEST
- XCTEST_UI

> **Note**
>
> CodePipeline: WEB_PERFORMANCE_PROFILE、REMOTE_ACCESS_RECORD和中的操作不支持以下测试类型REMOTE_ACCESS_REPLAY。

有关 Device Farm 测试类型的信息，请参阅使用 AWS Device Farm 中的测试类型。

RadioBluetoothEnabled

必需：否

用于指示是否在测试开始时启用蓝牙的布尔值。

RecordAppPerformanceData

必需：否

用于指示是否在测试期间记录设备性能数据（如 CPU、FPS 和内存性能）的布尔值。

RecordVideo

必需：否

用于指示是否在测试期间录制视频的布尔值。

RadioWifiEnabled

必需：否

用于指示是否在测试开始时启用 Wi-Fi 的布尔值。

RadioNfcEnabled

必需：否

用于指示是否在测试开始时启用 NFC 的布尔值。

RadioGpsEnabled

必需：否

用于指示是否在测试开始时启用 GPS 的布尔值。

测试

必需：否

您的源位置中测试定义文件的名称和路径。路径相对于测试的输入项目的根。

FuzzEventCount

必需：否

模糊测试要执行的用户接口事件数，介于 1 到 10,000 之间。

FuzzEventThrottle

必需：否

在执行下一个用户接口事件之前，模糊测试等待的毫秒数，介于 1 到 1,000 之间。

FuzzRandomizerSeed

必需：否

供模糊测试用来将用户接口事件随机化的种子。对后续模糊测试使用相同的数字会使事件序列相同。

CustomHostMachineArtifacts

必需：否

主机上存储自定义构件的位置。

CustomDeviceArtifacts

必需：否

设备上存储自定义构件的位置。

UnmeteredDevicesOnly

必需：否

用于指示在此步骤中运行测试时是否仅使用非计量设备的布尔值。

JobTimeoutMinutes

必需：否

测试在超时前将在每台设备上执行的分钟数。

Latitude

必需：否

以地理坐标系度数表示的设备的纬度。

Longitude

必需：否

以地理坐标系度数表示的设备的经度。

输入构件

- 构件数：1
- 描述：要提供给测试函数使用的一组构件。Device Farm 会查找要使用的已构建应用程序和测试定义。

输出构件

- 构件数：0
- 描述：输出构件不适用于此操作类型。

操作声明

YAML

```
Name: Test
Actions:
  - Name: TestDeviceFarm
    ActionTypeId: null
    category: Test
    owner: AWS
    provider: DeviceFarm
    version: '1'
RunOrder: 1
Configuration:
  App: s3-ios-test-1.ipa
  AppType: iOS
  DevicePoolArn: >-
    arn:aws:devicefarm:us-west-2::devicepool:0EXAMPLE-d7d7-48a5-ba5c-b33d66efa1f5
  ProjectId: eec4905f-98f8-40aa-9afc-4c1cfEXAMPLE
  TestType: APPIUM_PYTHON
  TestSpec: example-spec.yml
OutputArtifacts: []
```

```
InputArtifacts:
  - Name: SourceArtifact
Region: us-west-2
```

JSON

```
{
    "Name": "Test",
    "Actions": [
        {
            "Name": "TestDeviceFarm",
            "ActionTypeId": null,
            "category": "Test",
            "owner": "AWS",
            "provider": "DeviceFarm",
            "version": "1"
        }
    ],
    "RunOrder": 1,
    "Configuration": {
        "App": "s3-ios-test-1.ipa",
        "AppType": "iOS",
        "DevicePoolArn": "arn:aws:devicefarm:us-west-2::devicepool:0EXAMPLE-
d7d7-48a5-ba5c-b33d66efa1f5",
        "ProjectId": "eec4905f-98f8-40aa-9afc-4c1cfEXAMPLE",
        "TestType": "APPIUM_PYTHON",
        "TestSpec": "example-spec.yml"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "us-west-2"
},
```

另请参阅

下列相关资源在您使用此操作的过程中会有所帮助。

- 使用 Device Farm 中的测试类型：Device Farm 开发者指南 中的本参考章节详细描述了 Device Farm 支持的 Android、iOS 和 Web 应用程序测试框架。
- Device Farm 中的操作：Device Farm API 参考 中的 API 调用和参数可以帮助您处理 Device Farm 项目。
- 教程：创建用于构建和测试您的 Android 应用程序的管道 AWS Device Farm— 本教程提供了示例构建规范文件和示例应用程序，用于创建管道，其 GitHub 源代码用于使用和 Device Farm 构建 CodeBuild 和测试 Android 应用程序。
- 教程：创建用于测试 iOS 应用的管道 AWS Device Farm：本教程提供了一个示例应用程序，以创建具有 Amazon S3 源的管道，用于通过 Device Farm 测试所构建的一款 iOS 应用。

AWS Lambda

允许您执行 Lambda 函数以作为管道中的操作。使用作为此函数输入的事件对象，函数可以访问操作配置、输入构件位置、输出构件位置以及访问构件所需的其他信息。有关传递到 Lambda 调用函数的示例事件，请参阅JSON 事件示例。在 Lambda 函数实施中，必须有对 `PutJobSuccessResult API` 或 `PutJobFailureResult API` 的调用。否则，此操作的执行将挂起，直至操作超时。如果您为操作指定输出构件，则必须将其上传到 S3 存储桶作为函数实施的一部分。

> **Important**
>
> 不要记录 CodePipeline 发送到 Lambda 的 JSON 事件，因为这可能会导致用户凭证记录到日志中 CloudWatch 。该 CodePipeline 角色使用 JSON 事件将临时证书传递给该`artifactCredentials`领域的 Lambda。有关示例事件，请参阅JSON 事件示例。

操作类型

- 类别：`Invoke`
- 拥有者：`AWS`
- 提供方：`Lambda`
- 版本：`1`

配置参数

FunctionName

必需：是

`FunctionName` 是在 Lambda 中创建的函数的名称。

UserParameters

必需：否

一个供 Lambda 函数作为输入进行处理的字符串。

输入构件

- 构件数：`0 to 5`
- 描述：要提供给 Lambda 函数的一组构件。

输出构件

- 构件数：`0 to 5`
- 描述：Lambda 函数作为输出而生成的一组构件。

输出变量

此操作将生成包含在 PutJobSuccessResult API 请求`outputVariables`部分中的所有键值对作为变量。

有关中变量的更多信息 CodePipeline，请参阅Variables。

操作配置示例

YAML

```
Name: Lambda
Actions:
  - Name: Lambda
    ActionTypeId:
      Category: Invoke
```

```
      Owner: AWS
      Provider: Lambda
      Version: '1'
    RunOrder: 1
    Configuration:
      FunctionName: myLambdaFunction
      UserParameters: 'http://192.0.2.4'
    OutputArtifacts: []
    InputArtifacts: []
    Region: us-west-2
```

JSON

```
{
    "Name": "Lambda",
    "Actions": [
        {
            "Name": "Lambda",
            "ActionTypeId": {
                "Category": "Invoke",
                "Owner": "AWS",
                "Provider": "Lambda",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "FunctionName": "myLambdaFunction",
                "UserParameters": "http://192.0.2.4"
            },
            "OutputArtifacts": [],
            "InputArtifacts": [],
            "Region": "us-west-2"
        }
    ]
},
```

JSON 事件示例

Lambda 操作会发送一个 JSON 事件，其中包含作业 ID、管道操作配置、输入和输出构件位置，以及构件的任何加密信息。作业辅助角色将访问这些详细信息以完成 Lambda 操作。有关更多信息，请参阅：作业详细信息。以下是示例事件。

```
{
    "CodePipeline.job": {
        "id": "11111111-abcd-1111-abcd-111111abcdef",
        "accountId": "111111111111",
        "data": {
            "actionConfiguration": {
                "configuration": {
                    "FunctionName": "MyLambdaFunction",
                    "UserParameters": "input_parameter"
                }
            },
            "inputArtifacts": [
                {
                    "location": {
                        "s3Location": {
                            "bucketName": "bucket_name",
                            "objectKey": "filename"
                        },
                        "type": "S3"
                    },
                    "revision": null,
                    "name": "ArtifactName"
                }
            ],
            "outputArtifacts": [],
            "artifactCredentials": {
                "secretAccessKey": "secret_key",
                "sessionToken": "session_token",
                "accessKeyId": "access_key_ID"
            },
            "continuationToken": "token_ID",
            "encryptionKey": {
              "id": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
              "type": "KMS"
            }
        }
    }
}
```

JSON 事件为中的 Lambda 操作提供了以下任务详情： CodePipeline

- id：系统生成的唯一作业 ID。

- accountId：与任务关联的 AWS 账户 ID。
- data：作业辅助角色完成作业所需的其他信息。
  - actionConfiguration：Lambda 操作的操作参数。有关定义，请参阅配置参数 。
  - inputArtifacts：提供给操作的构件。
    - location：构件存储位置。
      - s3Location：操作的输入构件位置信息。
        - bucketName：操作的管道工件存储名称（例如，名为 codepipeline-us-east-2-1234567890 的 Amazon S3 存储桶）。
        - objectKey：应用程序的名称（例如，CodePipelineDemoApplication.zip）。
      - type：位置中构件的类型。目前，S3 是唯一有效的构件类型。
    - revision：构件的修订 ID。根据对象的类型，它可以是提交 ID (GitHub) 或修订版 ID（Amazon 简单存储服务）。有关更多信息，请参阅ArtifactRevision。
    - name：要处理的构件的名称，如 MyApp。
  - outputArtifacts：操作的输出。
    - location：构件存储位置。
      - s3Location：操作的输出构件位置信息。
        - bucketName：操作的管道工件存储名称（例如，名为 codepipeline-us-east-2-1234567890 的 Amazon S3 存储桶）。
        - objectKey：应用程序的名称（例如，CodePipelineDemoApplication.zip）。
      - type：位置中构件的类型。目前，S3 是唯一有效的构件类型。
    - revision：构件的修订 ID。根据对象的类型，它可以是提交 ID (GitHub) 或修订版 ID（Amazon 简单存储服务）。有关更多信息，请参阅ArtifactRevision。
    - name：构件的输出的名称，如 MyApp。
  - artifactCredentials：用于访问 Amazon S3 存储桶中输入和输出项目的 AWS 会话证书。这些凭证是临时凭证，由 AWS Security Token Service (AWS STS) 发布。
    - secretAccessKey：会话的秘密访问密钥。
    - sessionToken：令牌的会话。
    - accessKeyId：会话的秘密访问密钥。
  - continuationToken：操作生成的令牌。以后的操作使用此令牌来标识操作的正在运行的实例。操作完成时，不应提供延续令牌。

- `encryptionKey`：用于加密工件存储区中的数据的加密密钥，例如密 AWS KMS 钥。如果未定义此密钥，将使用 Amazon Simple Storage Service 的默认密钥。
  - `id`：用于标识密钥的 ID。对于 AWS KMS 密钥，您可以使用密钥 ID、密钥 ARN 或别名 ARN。
  - `type`：加密密钥的类型，如 AWS KMS 密钥。

另请参阅

下列相关资源在您使用此操作的过程中会有所帮助。

- AWS CloudFormation 用户指南 — 有关管道的 Lambda 操作和 AWS CloudFormation 工件的更多信息，请参阅在管道中使用参数覆盖函数、自动部署基于 Lambda 的应用程序和构件。 CodePipeline AWS CloudFormation
- 在中调用管道中的 AWS Lambda 函数 CodePipeline：此过程提供了一个示例 Lambda 函数，展示了如何使用控制台创建包含 Lambda 调用操作的管道。

Snyk 操作结构参考

中的 Snyk 操作 CodePipeline 可自动检测和修复开源代码中的安全漏洞。您可以将 Snyk 与第三方存储库（例如 GitHub 或 Bitbucket Cloud）中的应用程序源代码一起使用，也可以与容器应用程序的映像一起使用。您的操作将扫描并报告您配置的漏洞级别和警报。

> Note

主题

操作类型 ID

- 类别：`Invoke`

- 拥有者：ThirdParty
- 提供方：Snyk
- 版本：1

例如：

```
{
    "Category": "Invoke",
    "Owner": "ThirdParty",
    "Provider": "Snyk",
    "Version": "1"
},
```

输入构件

- 构件数：1
- 描述：构成调用操作输入构件的文件。

输出构件

- 构件数：1
- 描述：构成调用操作输出构件的文件。

另请参阅

下列相关资源在您使用此操作的过程中会有所帮助。

- 有关在中使用 Snyk 操作的更多信息 CodePipeline，请参阅使用 S nyk 自动 CodePipeline 进行漏洞扫描。

AWS Step Functions

执行以下 AWS CodePipeline 操作的操作：

- 从您的管道启动 AWS Step Functions 状态机执行。

- 通过操作配置中的属性或位于要作为输入传递的管道构件中的文件向状态机提供初始状态。
- （可选）设置执行 ID 前缀以便标识源自操作的执行。
- 支持标准和快速状态机。

> **Note**
>
> Step Functions 操作在 Lambda 上运行，因此它的工件大小配额与 Lambda 函数的工件大小配额相同。有关更多信息，请参阅 Lambda 开发人员指南中的 Lambda 配额。

操作类型

- 类别：`Invoke`
- 拥有者：`AWS`
- 提供方：`StepFunctions`
- 版本：`1`

配置参数

StateMachineArn

必需：是

要调用的状态机的 Amazon 资源名称 (ARN)。

ExecutionNamePrefix

必需：否

默认情况下，操作执行 ID 将用作状态机执行名称。如果提供了前缀，则该前缀将追加到带连字符的操作执行 ID 的前面，共同用作状态机执行名称。

```
myPrefix-1624a1d1-3699-43f0-8e1e-6bafd7fde791
```

对于快速状态机，名称应只包含 0-9、A-Z、a-z、- 和 _。

InputType

必需：否

- Literal (文本) (默认值) ：在指定此项时，Input (输入) 字段中的值将直接传递到状态机输入。

  选择文本时输入字段的输入项示例：

  ```
  {"action": "test"}
  ```

- FilePath：由 “输入” 字段指定的输入构件中的文件内容用作状态机执行的输入。如果设置为 ，InputType则需要输入工件FilePath。

  选中 “输入” 字段时输入FilePath的示例条目 ：

  ```
  assets/input.json
  ```

输入

必需：条件

- 文字：当设置InputType为 “文字” (默认) 时，此字段为可选字段。

  如果提供此项，则 Input (输入) 字段将直接用作状态机执行的输入。否则，将使用空 JSON 对象 {} 调用状态机。

- FilePath: 当设置InputType为时 FilePath，此字段为必填字段。

  当设置为时 InputType，还需要输入工件FilePath。

  指定的输入构件中文件的内容将用作状态机执行的输入。

输入构件

- 构件数：0 to 1
- 描述：如果设置InputType为 FilePath，则此构件是必需的，用于获取状态机执行的输入。

输出构件

- 构件数：0 to 1
- 描述：
  - 标准状态机：如果提供此项，则使用状态机的输出填充输出构件。这是在状态机执行成功完成后从 Step Fun DescribeExecution ctions API 响应的output属性中获取的。
  - 快速状态机：不受支持。

输出变量

此操作会生成输出变量，该变量可由管道中下游操作的操作配置引用。

有关更多信息，请参阅 Variables。

StateMachineArn

状态机的 ARN。

ExecutionArn

状态机的执行的 ARN。仅标准状态机。

操作配置示例

默认输入示例

YAML

```
Name: ActionName
ActionTypeId:
  Category: Invoke
  Owner: AWS
  Version: 1
  Provider: StepFunctions
OutputArtifacts:
  - Name: myOutputArtifact
Configuration:
  StateMachineArn: arn:aws:states:us-east-1:111122223333:stateMachine:HelloWorld-
StateMachine
  ExecutionNamePrefix: my-prefix
```

JSON

```
{
    "Name": "ActionName",
    "ActionTypeId": {
        "Category": "Invoke",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "StepFunctions"
    },
```

```
    "OutputArtifacts": [
        {
            "Name": "myOutputArtifact"
        }
    ],
    "Configuration": {
        "StateMachineArn": "arn:aws:states:us-
east-1:111122223333:stateMachine:HelloWorld-StateMachine",
        "ExecutionNamePrefix": "my-prefix"
    }
}
```

文本输入示例

YAML

```
Name: ActionName
ActionTypeId:
  Category: Invoke
  Owner: AWS
  Version: 1
  Provider: StepFunctions
OutputArtifacts:
  - Name: myOutputArtifact
Configuration:
  StateMachineArn: arn:aws:states:us-east-1:111122223333:stateMachine:HelloWorld-
StateMachine
  ExecutionNamePrefix: my-prefix
  Input: '{"action": "test"}'
```

JSON

```
{
    "Name": "ActionName",
    "ActionTypeId": {
        "Category": "Invoke",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "StepFunctions"
    },
    "OutputArtifacts": [
        {
```

```
            "Name": "myOutputArtifact"
        }
    ],
    "Configuration": {
        "StateMachineArn": "arn:aws:states:us-
east-1:111122223333:stateMachine:HelloWorld-StateMachine",
        "ExecutionNamePrefix": "my-prefix",
        "Input": "{\"action\": \"test\"}"
    }
}
```

输入文件示例

YAML

```
Name: ActionName
InputArtifacts:
  - Name: myInputArtifact
ActionTypeId:
  Category: Invoke
  Owner: AWS
  Version: 1
  Provider: StepFunctions
OutputArtifacts:
  - Name: myOutputArtifact
Configuration:
  StateMachineArn: 'arn:aws:states:us-east-1:111122223333:stateMachine:HelloWorld-
StateMachine'
  ExecutionNamePrefix: my-prefix
  InputType: FilePath
  Input: assets/input.json
```

JSON

```
{
    "Name": "ActionName",
    "InputArtifacts": [
        {
            "Name": "myInputArtifact"
        }
    ],
    "ActionTypeId": {
```

```
        "Category": "Invoke",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "StepFunctions"
    },
    "OutputArtifacts": [
        {
            "Name": "myOutputArtifact"
        }
    ],
    "Configuration": {
        "StateMachineArn": "arn:aws:states:us-
east-1:111122223333:stateMachine:HelloWorld-StateMachine",
        "ExecutionNamePrefix": "my-prefix",
        "InputType": "FilePath",
        "Input": "assets/input.json"
    }
}
```

行为

在发布期间，使用操作配置中指定的输入 CodePipeline 执行已配置的状态机。

当设置InputType为 Literal 时，输入操作配置字段的内容将用作状态机的输入。如果未提供文本输入，则状态机执行将使用空的 JSON 对象 {}。有关在没有输入的情况下运行状态机执行的更多信息，请参阅 Step Functions StartExecution API。

设置InputType为时 FilePath，该操作将解压缩输入对象，并使用在输入操作配置字段中指定的文件内容作为状态机的输入。指定后 FilePath，“输入” 字段为必填字段，并且必须存在输入对象；否则，操作将失败。

在成功启动执行后，两种状态机类型（标准 和快速）的行为将不同。

标准状态机

如果标准状态机执行成功启动，则会 CodePipeline 轮询 `DescribeExecution` API，直到执行达到终端状态。如果执行成功完成，则操作将成功；否则，操作将失败。

如果配置了输出构件，则该构件将包含状态机的返回值。这是在状态机执行成功完成后从 Step Fun DescribeExecution ctions API 响应的`output`属性中获取的。请注意，此 API 上已强制实施输出长度限制。

错误处理

- 如果操作无法启动状态机执行，则操作执行将失败。
- 如果在 Ste CodePipeline p Functions 操作达到超时时间（默认为 7 天）之前，状态机执行未能达到终端状态，则操作执行将失败。尽管发生此失败，状态机也可能会继续运行。有关 Step Functions 中的状态机执行超时的更多信息，请参阅标准和快速工作流。

> **Note**
>
> 您可以使用此操作请求增加账户的调用操作超时配额。但配额增加将应用到该账户的所有区域中此类型的所有操作。

- 如果状态机执行达到 FAILED、TIMED_OUT 或 ABORTED 终端状态，则操作执行失败。

快速状态机

如果快速状态机执行已成功启动，则表示调用操作执行已成功完成。

为快速状态机配置的操作的注意事项：

- 您无法指定输出构件。
- 操作不会等待状态机执行完成。
- 在中开始执行操作后 CodePipeline，即使状态机执行失败，操作也会成功执行。

错误处理

- 如果 CodePipeline 无法启动状态机执行，则操作执行失败。否则，操作将立即成功。 CodePipeline 无论状态机执行需要多长时间完成或其结果如何，操作都会成功。

另请参阅

下列相关资源在您使用此操作的过程中会有所帮助。

- AWS Step Functions 开发者指南 — 有关状态机、执行和状态机输入的信息，请参阅《AWS Step Functions 开发人员指南》。
- 教程：在管道中使用 AWS Step Functions 调用操作：本教程将帮助您开始使用示例标准状态机，并说明如何使用控制台通过添加 Step Functions 调用操作来更新管道。

集成模型参考

我们预先构建了若干第三方服务集成，有助于将现有的客户工具植入管道发布流程。合作伙伴或第三方服务提供商使用集成模型来实现操作类型以在中使用 CodePipeline。

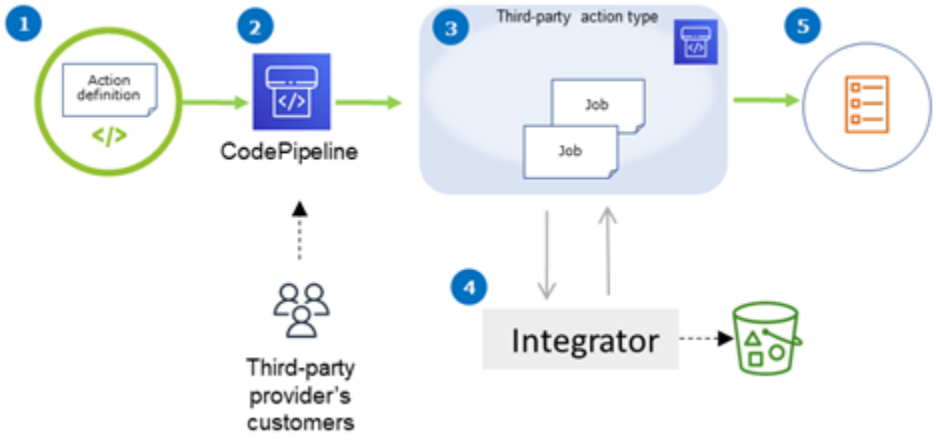
当您计划或使用中支持的集成模型管理的操作类型时，请使用此参考 CodePipeline。

要将您的第三方操作类型证明为合作伙伴集成 CodePipeline，请参阅 AWS 合作伙伴网络 (APN)。此信息是对 AWS CLI 参考的补充。

主题

第三方操作类型如何与集成商配合工作

您可以向客户管道中添加第三方操作类型，以完成基于客户资源的任务。集成商管理任务请求并使用CodePipeline运行操作。下图显示了为客户创建的要在其管道中使用的第三方操作类型。客户配置操作后，操作将运行并创建任务请求，这些请求将由集成商的操作引擎处理。



图中显示以下步骤：

1. 操作定义已在中注册并可用 CodePipeline。第三方提供商的客户可以使用第三方操作。

2. 提供商的客户在中选择并配置操作。 CodePipeline
3. 操作将运行，作业已进入队列。 CodePipeline当作业准备就绪时 CodePipeline，它会发送任务请求。
4. 集成商（第三方轮询 API 或 Lambda 函数的任务工作者）接收任务请求，返回确认信息，然后处理操作的构件。
5. 集成商返回包含任务结果和延续令牌的成功/失败输出（任务工作者使用成功/失败 API 或 Lambda 函数发送成功/失败输出）。

有关可用于请求、查看和更新操作类型的步骤的信息，请参阅使用操作类型。

概念

本节对第三方操作类型使用以下术语：

操作类型

一种可重复的流程，可在执行相同持续交付工作负载的管道中重复使用。操作类型由 `Owner`、`Category`、`Provider` 和 `Version` 标识。例如：

```
        {

            "Category": "Deploy",
            "Owner": "AWS",
            "Provider": "CodeDeploy",
            "Version": "1"
        },
```

相同类型的所有操作都采用相同的实现。

操作

操作类型的单个实例，即管道阶段内发生的非连续流程之一。这通常包括特定于运行此操作的管道的用户值。

操作定义

操作类型的架构，用于定义配置操作和输入/输出构件所需的属性。

操作执行

为确定客户管道上的操作是否成功而运行的一系列任务。

操作执行引擎

操作执行配置的一个属性，用于定义操作类型使用的集成类型。有效值为 `JobWorker` 和 `Lambda`。

集成

描述由集成商运行的一个软件，用于实现操作类型。 CodePipeline 支持两种集成类型，分别对应于两个支持的动作引擎`JobWorker`和`Lambda`。

集成商

负责实现操作类型的人。

作业

利用管道和客户背景来执行集成的一项工作。操作执行包含一个或多个任务。

任务工作者

处理客户输入并运行任务的服务。

支持的集成模型

CodePipeline 有两种集成模型：

- Lambda 集成模型：此集成模型是使用中的操作类型的首选方式。 CodePipeline当您的操作运行时，Lambda 集成模型使用 Lambda 函数来处理任务请求。
- 任务工作者集成模型：任务工作者集成模型是以前用于第三方集成的模型。作业辅助集成模型使用配置为在操作运行时与 CodePipeline API 联系以处理作业请求的作业工作者。

为了进行比较，下表描述了这两种模型的特征：

	Lambda 集成模型	任务工作者集成模型
描述	集成器将集成写为 Lambda 函数，只要有任务可用于操作，就会调用 CodePipeline 该函数。Lambda 函数不轮询可用的任务，而是等待，直到收到下一个任务请求。	集成商将集成写入为任务工作者，后者会持续轮询客户管道上的可用任务。然后，作业工作人员执行作业，并使用 CodePipel

	Lambda 集成模型	任务工作者集成模型
		ine API 将作业结果提交回给 CodePipeline 。
基础设施	AWS Lambda	将任务工作者代码部署到集成商的基础设施，例如 Amazon EC2 实例。
开发工作	集成仅包含业务逻辑。	除了包含业务逻辑外，集成还需要与 CodePipeline API 交互。
运营工作	由于基础设施只是 AWS 资源，因此运营工作量较少。	因为任务工作者需要独立的硬件，所以运营工作量较多。
最长任务运行时间	如果集成需要有效运行超过 15 分钟，则无法使用该模型。此操作适用于需要启动流程（例如，在客户的代码构件上启动构建）并在完成后返回结果的集成商。我们不建议集成商持续等待构建完成，相反，返回延续。 CodePipeline如果收到来自集成商代码的延续，则会在 30 秒内创建一个新作业，以检查该作业，直到任务完成。	使用该模型可以维持长时间运行的任务（数小时/数天）。

Lambda 集成模型

支持的 Lambda 集成模型包括创建 Lambda 函数和为第三方操作类型定义输出。

更新您的 Lambda 函数以处理来自的输入 CodePipeline

您可以创建新的 Lambda 函数。您可以向 Lambda 函数中添加业务逻辑，只要您的管道上有适用于操作类型的任务，该函数就会运行。例如，鉴于客户和管道的背景，您可能想在服务中为客户开始构建。

使用以下参数更新您的 Lambda 函数以处理来自的输入。 CodePipeline

格式：

- jobId:
  - 系统生成的唯一任务 ID。
  - 类型：字符串
  - 模式：[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}
- accountId:
  - 执行任务时要使用的客户 AWS 账户的 ID。
  - 类型：字符串
  - 模式：[0-9]{12}
- data:
  - 集成用于完成任务的其他任务信息。
  - 包含以下各项：
    - actionConfiguration:
      - 操作的配置数据。操作配置字段是键值对的映射，供您的客户输入值。设置操作时，键取决于操作类型定义文件中的键参数。在本例中，值取决于指定 Username 和 Password 字段信息的操作的用户。
      - 类型：字符串到字符串映射，目前可选

        例如：

        ```
        "configuration": {
            "Username": "MyUser",
            "Password": "MyPassword"
        },
        ```
    - encryptionKey:
      - 表示有关用于加密工件存储中数据的密钥的信息，例如密 AWS KMS 钥。
      - 内容：数据类型 encryptionKey 的类型，目前可选
    - inputArtifacts:
      - 有关要处理的构件的信息列表，例如测试或构建构件。
      - 内容：数据类型 Artifact 的列表，目前可选
    - outputArtifacts:
      - 有关操作输出的信息列表。
      - 内容：数据类型 Artifact 的列表，目前可选

- actionCredentials:
  - 表示会 AWS 话凭证对象。这些凭证是临时凭证，由 AWS STS发布。它们可用于访问用于存储管道工件的 S3 存储桶中的输入和输出项目 CodePipeline。

    这些凭证还具有与操作类型定义文件中指定的策略声明模板相同的权限。
  - 内容：数据类型 AWSSessionCredentials 的类型，目前可选
- actionExecutionId:
  - 操作运行的外部 ID。
  - 类型：字符串
- continuationToken:
  - 系统生成的令牌（例如部署 ID），任务需要它来异步延续任务。
  - 类型：字符串，目前可选

数据类型：

- encryptionKey:
  - id:
    - 用于标识密钥的 ID。对于密 AWS KMS 钥，您可以使用密钥 ID、密钥 ARN 或别名 ARN。
    - 类型：字符串
  - type:
    - 加密密钥的类型，例如密 AWS KMS 钥。
    - 类型：字符串
    - 有效值：KMS
- Artifact:
  - name:
    - 构件的名称。
    - 类型：字符串，目前可选
  - revision:
    - 构件的修订 ID。根据对象的类型，这可能是提交 ID (GitHub) 或修订版 ID (Amazon S3)。
    - 类型：字符串，目前可选
  - location:
    - 构件的位置。

    - 内容：数据类型 ArtifactLocation 的类型，目前可选
- ArtifactLocation:
  - type:
    - 位置中构件的类型。
    - 类型：字符串，目前可选
    - 有效值：S3
  - s3Location:
    - 包含修订的 S3 桶的位置。
    - 内容：数据类型 S3Location 的类型，目前可选
- S3Location:
  - bucketName:
    - S3 桶的名称。
    - 类型：字符串
  - objectKey:
    - S3 桶中对象的键，在桶中唯一标识该对象。
    - 类型：字符串
- AWSSessionCredentials:
  - accessKeyId:
    - 会话的访问密钥。
    - 类型：字符串
  - secretAccessKey:
    - 会话的秘密访问密钥。
    - 类型：字符串
  - sessionToken:
    - 会话的令牌。
    - 类型：字符串

例如：


```
{
    "jobId": "01234567-abcd-abcd-abcd-012345678910",
```

```
    "accountId": "012345678910",
    "data": {
        "actionConfiguration": {
            "key1": "value1",
            "key2": "value2"
        },
        "encryptionKey": {
            "id": "123-abc",
            "type": "KMS"
        },
        "inputArtifacts": [
            {
                "name": "input-art-name",
                "location": {
                    "type": "S3",
                    "s3Location": {
                        "bucketName": "inputBucket",
                        "objectKey": "inputKey"
                    }
                }
            }
        ],
        "outputArtifacts": [
            {
                "name": "output-art-name",
                "location": {
                    "type": "S3",
                    "s3Location": {
                        "bucketName": "outputBucket",
                        "objectKey": "outputKey"
                    }
                }
            }
        ],
        "actionExecutionId": "actionExecutionId",
        "actionCredentials": {
            "accessKeyId": "access-id",
            "secretAccessKey": "secret-id",
            "sessionToken": "session-id"
        },
        "continuationToken": "continueId-xxyyzz"
    }
}
```

将您的 Lambda 函数的结果返回到 CodePipeline

在成功、失败或延续的情况下，集成商的任务工作者资源必须返回有效的负载。

格式：

- `result`：任务的结果。
  - 必需
  - 有效值（不区分大小写）：
    - `Success`：表示任务成功且已终止。
    - `Continue`：表示任务成功且必须延续，例如，为执行相同操作而重新调用任务工作者时。
    - `Fail`：表示任务失败且已终止。
- `failureType`：要与失败的任务关联的失败类型。

  合作伙伴操作的 `failureType` 类别描述了运行任务时遇到的失败类型。将任务失败结果返回到 CodePipeline时，集成商会设置类型以及失败消息。
  - 可选。如果结果是 `Fail`，则为必填项。
  - 如果 `result` 为 `Success` 或 `Continue`，则必须为空
  - 有效值：
    - ConfigurationError
    - JobFailed
    - PermissionsError
    - RevisionOutOfSync
    - RevisionUnavailable
    - SystemUnavailable
- `continuation`：在当前操作执行中，要传递给下一个任务的延续状态。
  - 可选。如果结果是 `Continue`，则为必填项。
  - 如果 `result` 为 `Success` 或 `Fail`，则必须为空。
  - 属性：
    - `State`：要传递的状态的哈希值。
- `status`：操作执行的状态。
  - 可选。
  - 属性：

  - ExternalExecutionId：与任务关联的可选外部执行 ID 或提交 ID。
  - Summary：所发生情况的可选摘要。在失败情况下，这会成为用户看到的失败消息。
- outputVariables：要传递给下一个操作执行的一组键/值对。
  - 可选。
  - 如果 result 为 Continue 或 Fail，则必须为空。

例如：

```
{
    "result": "success",
    "failureType": null,
    "continuation": null,
    "status": {
        "externalExecutionId": "my-commit-id-123",
        "summary": "everything is dandy"
    },
    "outputVariables": {
        "FirstOne": "Nice",
        "SecondOne": "Nicest",
        ...
    }
}
```

使用延续令牌等待异步流程的结果

continuation 令牌是有效负载的一部分，也是 Lambda 函数的结果。这是一种将工作状态传递给工作状态 CodePipeline 并表明需要继续工作的方式。例如，集成商在其资源上为客户启动构建后，它不会等待构建完成，而是通过返回 as continue 并将版本的唯一 ID 返回 CodePipeline 到 a result s token 来 CodePipeline 表示它没有最终结果。continuation

> **Note**
>
> Lambda 函数只能运行不超过 15 分钟。如果任务需要运行更长的时间，可以使用延续令牌。

CodePipeline 队伍在 30 秒后调用积分器，其有效载荷中包含相同的continuation令牌，这样它就可以检查积分器是否完成。如果构建完成，集成商将返回终端成功/失败结果，否则继续。

提供 CodePipeline 在运行时调用集成商 Lambda 函数的权限

您可以向集成商 Lambda 函数添加权限，为服务提供 CodePipeline 使用服务委托人调用 CodePipeline 该函数的权限:。`codepipeline.amazonaws.com`您可以使用 AWS CloudFormation 或命令行添加权限。有关示例，请参阅使用操作类型。

任务工作者集成模型

设计完高层工作流程后，您可以创建任务工作者。尽管第三方操作的细节决定了任务工作者需要什么，但大多数第三方操作的任务工作者都包括以下功能：

- CodePipeline 正在使用轮询作业`PollForThirdPartyJobs`。
- 确认任务并将结果返回到 CodePipeline 使用`AcknowledgeThirdPartyJob`、`PutThirdPartyJobSuccessResult`和`PutThirdPartyJobFailureResult`。
- 从管道的 Amazon S3 桶中检索构件和/或将构件放入桶。要从 Amazon S3 桶下载构件，您必须创建一个使用签名版本 4 (Sig V4) 签名的 Amazon S3 客户端。需要 Sig V4 才能使用。 AWS KMS

  要将项目上传到 Amazon S3 存储桶，您还必须通过 AWS Key Management Service (AWS KMS) 将 Amazon S3 `PutObject` 请求配置为使用加密。 AWS KMS 使用 AWS KMS keys。为了知道是使用客户管理的 AWS 托管式密钥 密钥还是客户管理的密钥上传工件，您的作业工作人员必须查看作业数据并检查加密密钥属性。如果设置了该属性，则在配置时应使用该客户托管密钥 ID AWS KMS。如果键属性为空，则使用 AWS 托管式密钥。 CodePipeline AWS 托管式密钥 除非另行配置，否则使用。

  有关演示如何使用 Java 或.NET 创建 AWS KMS 参数的示例，请参阅使用 AWS 软件开发工具包 AWS Key Management Service 在 Amazon S3 中指定。有关的 Amazon S3 存储桶的更多信息 CodePipeline，请参阅CodePipeline 概念 。

为作业辅助角色选择和配置权限管理策略

要为你的第三方操作培养工作人员 CodePipeline，你需要一个整合用户和权限管理的策略。

最简单的策略是通过创建带有 AWS Identity and Access Management (IAM) 实例角色的 Amazon EC2 实例来添加作业人员所需的基础设施，这样您就可以轻松扩展集成所需的资源。您可以使用内置的集成 AWS 来简化作业人员与之间的互动 CodePipeline。

了解更多有关 Amazon EC2 的信息，并确定其是否为适合您的集成的正确选择。有关信息，请参阅 Amazon EC2 – 虚拟服务器托管。有关设置 Amazon EC2 实例的更多信息，请参阅 Amazon EC2 Linux 实例入门。

另一个需要考虑的策略是使用 IAM 的身份联合验证来集成您的现有身份提供程序系统和资源。如果您已经拥有企业身份提供程序，或已经配置为支持使用网络身份提供程序的用户，则此策略很有用。联合身份允许您授予对 AWS 资源（包括）的安全访问权限 CodePipeline，而无需创建或管理 IAM 用户。您可以利用针对密码安全性要求和凭证轮换的功能和策略。您可以使用示例应用程序作为自己设计的模板。有关信息，请参阅管理联合身份验证。

要提供访问权限，请为您的用户、组或角色添加权限：

- 中的用户和群组 AWS IAM Identity Center：

  创建权限集合。按照《AWS IAM Identity Center 用户指南》中创建权限集的说明进行操作。
- 通过身份提供商在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中为第三方身份提供商创建角色（联合身份验证）的说明进行操作。
- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

以下是您可能为搭配第三方任务工作者使用而创建的策略示例。此策略仅作为示例，按原样提供。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codepipeline:PollForThirdPartyJobs",
        "codepipeline:AcknowledgeThirdPartyJob",
        "codepipeline:GetThirdPartyJobDetails",
        "codepipeline:PutThirdPartyJobSuccessResult",
        "codepipeline:PutThirdPartyJobFailureResult"
      ],
      "Resource": [
        "arn:aws:codepipeline:us-east-2::actionType:ThirdParty/Build/Provider/1/"
```

```
        ]
      }
    ]
}
```

映像定义文件参考

本部分仅供参考。有关使用源或部署操作为容器创建管道的更多信息，请参阅在中创建管道 CodePipeline。

AWS CodePipeline 容器操作（例如 Amazon ECR 源操作或 Amazon ECS 部署操作）的作业工作人员使用定义文件将图像 URI 和容器名称映射到任务定义。每个定义文件是操作提供方使用的 JSON 格式文件，如下所示：

- Amazon ECS 标准部署需要 `imagedefinitions.json` 文件作为部署操作的输入。
- Amazon ECS 蓝绿部署需要 `imageDetail.json` 文件作为部署操作的输入。
  - Amazon ECR 源操作生成 `imageDetail.json` 文件，并作为源操作的输出提供。

主题

适用于 Amazon ECS 标准部署操作的 imagedefinitions.json 文件

映像定义文档是一个 JSON 文件，用于描述 Amazon ECS 容器名称以及映像和标签。如果您要部署基于容器的应用程序，则必须生成图像定义文件，为 CodePipeline 作业工作人员提供 Amazon ECS 容器和图像标识，以便从图像存储库（例如 Amazon ECR）中检索。

> **Note**
>
> 该文件的默认文件名是 `imagedefinitions.json`。如果您选择使用不同的文件名，必须在创建管道部署阶段时提供。

创建 `imagedefinitions.json` 文件时注意以下事项：

- 文件必须使用 UTF-8 编码。
- 映像定义文件的最大文件大小限制为 100 KB。
- 您必须创建 文件作为源或构建构件，以将其作为部署操作的输入构件。换句话说，请确保将文件上传到您的源位置（例如 CodeCommit 存储库），或者作为构建的输出构件生成。

imagedefinitions.json 文件提供容器名称和映像 URI。它必须采用以下一组键值对的结构。

键	值
name	*####*
imageUri	*imageUri*

此处为 JSON 结构，其中容器名称为 sample-app，映像 URI 为 ecs-repo，标签为 latest：

```
[
  {
    "name": "sample-app",
    "imageUri": "11111EXAMPLE.dkr.ecr.us-west-2.amazonaws.com/ecs-repo:latest"
  }
]
```

您还可以构造该文件以列出多个容器/映像对。

JSON 结构：

```
[
  {
    "name": "simple-app",
    "imageUri": "httpd:2.4"
  },
  {
    "name": "simple-app-1",
    "imageUri": "mysql"
  },
  {
    "name": "simple-app-2",
    "imageUri": "java1.8"
  }
]
```

在创建管道之前，请使用以下步骤设置 imagedefinitions.json 文件。

1. 在为管道计划基于容器的应用程序部署过程中，请计划源阶段和生成阶段 (如果适用)。
2. 选择以下操作之一：

a. 如果创建管道时跳过构件阶段，则您必须手动创建 JSON 文件并将其上传到源存储库，以便源操作可以提供构件。使用文本编辑器创建该文件，然后命名该文件或使用默认 `imagedefinitions.json` 文件名。将映像定义文件推送到源存储库。

> **Note**
>
> 如果源存储库为 Amazon S3 桶，请务必压缩 JSON 文件。

b. 如果管道具有生成阶段，请在生成规范文件中添加一个命令，以便在生成阶段在源存储库中输出映像定义文件。以下示例使用 printf 命令创建 `imagedefinitions.json` 文件。在 `buildspec.yml` 文件的 `post_build` 部分中列出此命令：

```
printf '[{"name":"container_name","imageUri":"image_URI"}]' >
imagedefinitions.json
```

您必须在 `buildspec.yml` 文件中包含映像定义文件以作为输出构件。

3. 在控制台中创建管道时，请在创建管道向导的部署页面上，在映像文件名字段中输入映像定义文件名。

有关创建使用 Amazon ECS 作为部署提供商的管道的教 step-by-step 程，请参阅教程：使用进行持续部署 CodePipeline。

适用于 Amazon ECS 蓝绿部署的 imageDetail.json 文件

`imageDetail.json` 文档是一个 JSON 文件，用于描述您的 Amazon ECS 映像 URI。如果您要为蓝/绿部署部署基于容器的应用程序，则必须生成`imageDetail.json`文件以向 Amazon ECS 和 CodeDeploy作业工作人员提供图像标识，以便从映像存储库（例如 Amazon ECR）中检索。

> **Note**
>
> 文件名称必须是 `imageDetail.json`。

有关操作的说明，请参阅 Amazon 弹性容器服务和 CodeDeploy 蓝绿色。

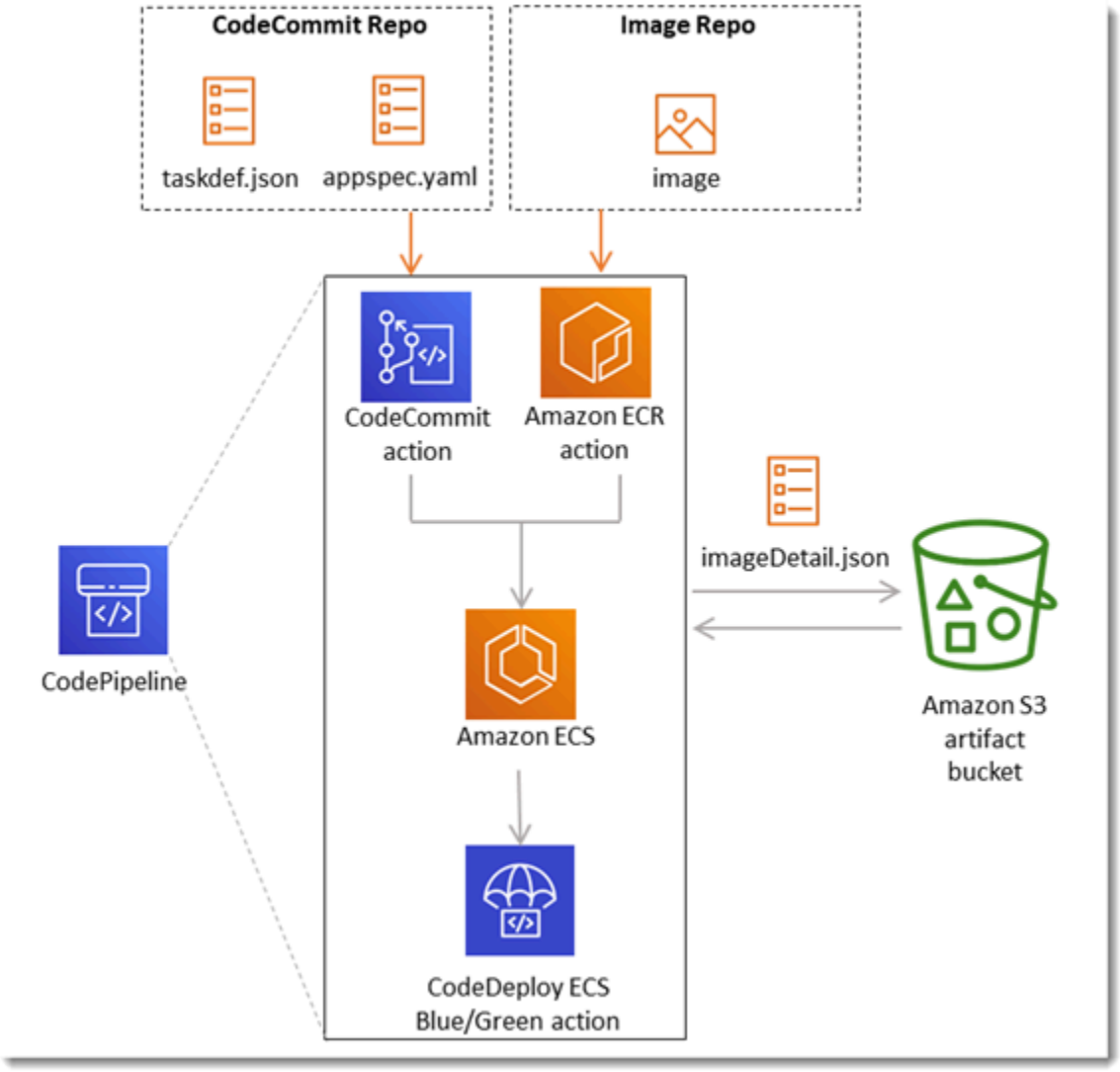
您必须创建 `imageDetail.json` 文件作为源或构建构件，以将其作为部署操作的输入构件。您可以使用以下方法之一在管道中提供 `imageDetail.json` 文件：

- 在您的源位置中包含 `imageDetail.json` 文件，使其在管道中作为输入提供给 Amazon ECS 蓝绿部署操作。

> **Note**
>
> 如果源存储库为 Amazon S3 桶，请务必压缩 JSON 文件。

- Amazon ECR 源操作会自动生成 `imageDetail.json` 文件，作为下一个操作的输入构件。

> **Note**
>
> 由于 Amazon ECR 源操作会创建此文件，因此具有 Amazon ECR 源操作的管道无需手动提供 `imageDetail.json` 文件。
> 有关创建包含 Amazon ECR 源阶段的管道的教程，请参阅教程：使用 Amazon ECR 源和 ECS 目标部署创建管道 CodeDeploy 。



`imageDetail.json` 文件提供映像 URI。它必须使用以下键值对进行构造。

键	值
ImageURI	*`image_URI`*

imageDetail.json

以下是 JSON 结构，其中映像 URI 为 `ACCOUNTID.dkr.ecr.us-west-2.amazonaws.com/dk-image-repo@sha256:example3`：

```
{
```

```
"ImageURI": "ACCOUNTID.dkr.ecr.us-west-2.amazonaws.com/dk-image-
repo@sha256:example3"
}
```

imageDetail.json (generated by ECR)

每次将更改推送到映像存储库时，Amazon ECR 源操作都会自动生成 imageDetail.json 文件。Amazon ECR 源操作生成的 imageDetail.json 将作为源操作的输出构件，提供给管道中的下一操作。

以下是 JSON 结构，其中存储库名称是 dk-image-repo、映像 URI 是 ecs-repo 且映像标签是 latest：

```
{
    "ImageSizeInBytes": "44728918",
    "ImageDigest":
 "sha256:EXAMPLE1122334455667788990bfea42ea2d3b8a1ee8329ba7e68694950afd3",
    "Version": "1.0",
    "ImagePushedAt": "Mon Jan 21 20:04:00 UTC 2019",
    "RegistryId": "EXAMPLE12233",
    "RepositoryName": "dk-image-repo",
    "ImageURI": "ACCOUNTID.dkr.ecr.us-west-2.amazonaws.com/dk-image-
repo@sha256:example3",
    "ImageTags": [
        "latest"
    ]
}
```

imageDetail.json 文件将映像 URI 和容器名称映射到 Amazon ECS 任务定义，如下所示：

- ImageSizeInBytes：存储库中映像的大小（以字节为单位）。
- ImageDigest：映像清单的 sha256 摘要。
- Version：映像版本。
- ImagePushedAt：将最新映像推送到存储库的日期和时间。
- RegistryId：与包含存储库的注册表关联的 AWS 账户 ID。
- RepositoryName：向其中推送映像的 Amazon ECR 存储库的名称。
- ImageURI：映像的 URI。
- ImageTags：为映像使用的标签。

在创建管道之前，请使用以下步骤设置 imageDetail.json 文件。

1. 在为管道计划基于容器的应用程序蓝绿部署过程中，请计划源阶段和生成阶段（如果适用）。
2. 选择以下操作之一：

   a. 如果您的管道跳过了构建阶段，则必须手动创建 JSON 文件并将其上传到源存储库（例如）CodeCommit，这样源操作才能提供构件。使用文本编辑器创建该文件，然后命名该文件或使用默认 imageDetail.json 文件名。将 imageDetail.json 文件推送至您的源存储库。

   b. 如果您的管道包括构建阶段，请执行以下操作：

      i. 在生成规范文件中添加一个命令，以便在生成阶段在源存储库中输出映像定义文件。以下示例使用 printf 命令创建 imageDetail.json 文件。可以在 buildspec.yml 文件的 post_build 部分中列出该命令：

      ```
      printf '{"ImageURI":"image_URI"}' > imageDetail.json
      ```

      您必须包含 imageDetail.json 文件作为 buildspec.yml 文件中的输出构件。

      ii. 添加 imageDetail.json 作为 buildspec.yml 文件中的构件文件。

      ```
      artifacts:
        files:
          - imageDetail.json
      ```

Variables

本部分仅供参考。有关创建变量的信息，请参阅使用变量。

变量允许您使用在管道执行或操作执行时确定的值来配置管道操作。

一些操作提供方会生成一组定义的变量。您可以从该操作提供方的默认变量键中进行选择，例如提交 ID。

> **Important**
>
> 传递密钥参数时，请勿直接输入值。该值以明文形式提供，因此是可读的。出于安全原因，请勿使用纯文本表示密钥。我们强烈建议您使用 AWS Secrets Manager 来存储机密。

要查看使用变量的 step-by-step 示例，请执行以下操作：

- 有关在管道执行时传递的管道级变量的教程，请参阅教程：使用管道级变量。
- 有关使用上游操作 (CodeCommit) 中的变量并生成输出变量的 Lambda 操作的教程，请参阅。教程：将变量与 Lambda 调用操作一起使用
- 有关引用上游 AWS CloudFormation 操作堆栈输出变量的 CloudFormation 操作的教程，请参阅教程：创建使用 AWS CloudFormation 部署操作中的变量的管道。
- 有关手动批准操作的示例，其消息文本引用了解析为提 CodeCommit 交 ID 和提交消息的输出变量，请参阅示例：在手动审批中使用变量。
- 有关带有可解析为 GitHub分支名称的环境变量的 CodeBuild 操作示例，请参阅示例：将 BranchName变量与 CodeBuild 环境变量一起使用。
- CodeBuild actions 将所有作为构建一部分导出的环境变量生成为变量。有关更多信息，请参阅 CodeBuild 操作输出变量。

变量限制

有关限制信息，请参阅 中的配额 AWS CodePipeline。

> **Note**
>
> 在操作配置字段中输入变量语法时，请不要超过配置字段的 1000 个字符的限制。如果超过此限制，将会返回验证错误。

主题

概念

此部分列出了与变量和命名空间相关的关键术语和概念。

Variables

变量是键/值对，可用于在管道中动态配置操作。目前，有三种方法提供这些变量：

- 在每个管道执行开始时，都有一组隐式可用的变量。这组变量当前包括 `PipelineExecutionId`，这是当前管道执行的 ID。
- 管道级变量是在管道运行时间创建和解析管道时定义的。

  您可以在创建管道时指定管道级变量，也可以在管道执行时提供值。
- 有的操作类型在执行时生成一组变量。您可以通过检查作为 ListActionExecutionsAPI 一部分的`outputVariables`字段来查看操作生成的变量。有关按操作提供方列出的可用键名称的列表，请参阅可用于管道操作的变量。要查看每种操作类型产生的变量，请参阅 CodePipeline 操作结构参考。

要在操作配置中引用这些变量，您必须使用带有正确命名空间的变量引用语法。

有关示例变量工作流，请参阅配置变量 。

命名空间

为了确保可以唯一地引用该变量，必须将其分配到命名空间。将一组变量分配到命名空间后，您可以通过以下语法，使用命名空间和变量键在操作配置中引用这些变量：

```
#{namespace.variable_key}
```

您可以在三种类型的命名空间中分配变量：

- Codepipeline 预留命名空间

  这是分配到在每个管道执行开始时可用的一组隐式变量的命名空间。此命名空间是 `codepipeline`。变量引用示例：

  ```
  #{codepipeline.PipelineExecutionId}
  ```

- 管道级变量命名空间

  这是分配给管道级变量的命名空间。所有管道级变量的命名空间是 `variables`。变量引用示例：

  ```
  #{variables.variable_name}
  ```

- 已分配命名空间的操作

  这是您分配给某个操作的命名空间。该操作生成的所有变量都属于此命名空间。要使某个操作生成的变量可在下游操作配置中使用，您必须使用命名空间配置生成操作。命名空间在整个管道定义中必须唯一，并且不能与任何构件名称冲突。以下是使用命名空间 `SourceVariables` 配置的操作的变量引用示例。

  ```
  #{SourceVariables.VersionId}
  ```

变量使用案例

以下是管道级变量的一些常见使用案例，有助于确定如何根据自己的特定需求使用变量。

- 管道级别的变量适用于希望每次都使用同一个管道但操作配置的输入略有差异的 CodePipeline 客户。任何启动管道的开发人员都会在管道启动时在 UI 中添加变量值。通过此配置，只为此次执行传递参数。
- 使用管道级变量，您可以将动态输入传递给管道中的操作。您可以将参数化管道迁移到，CodePipeline 而不必维护同一管道的不同版本，也不必创建复杂的管道。
- 您可以使用管道级变量传递输入参数，以便在每次执行时重复使用管道（例如当您要指定想要部署到生产环境的版本时），如此就不必复制管道了。
- 您可以使用单个管道将资源部署到多个构建和部署环境。例如，对于带有 CodeCommit 存储库的管道，可以从指定的分支和目标部署环境进行部署， CodeBuild 并在管道级别传递 CodeDeploy参数。

配置变量

您可以在管道级别或管道结构中的操作级别配置变量。

配置管道级变量

您可以添加一个或多个管道级变量。你可以在 CodePipeline 动作配置中引用这个值。您可以在创建管道时添加变量名称、默认值和描述。在执行时解析变量。

> **Note**
>
> 如果没有为管道级变量定义默认值，则认为该变量是必需的。启动管道时，必须为所有必需的变量指定覆盖，否则管道执行将因验证错误而失败。

您可以使用管道结构中的变量属性提供管道级变量。在以下示例中，变量 `Variable1` 的值为 `Value1`。

```
"variables": [
     {
         "name": "Variable1",
         "defaultValue": "Value1",
         "description": "description"
     }
 ]
```

有关管道 JSON 结构的示例，请参阅在中创建管道 CodePipeline。

有关在管道执行时传递的管道级变量的教程，请参阅教程：使用管道级变量。

请注意，不支持在任何类型的源操作中使用管道级变量。

> **Note**
>
> 如果管道中的某些操作中已经使用 `variables` 命名空间，则必须更新操作定义并为冲突的操作选择另一个命名空间。

配置操作级变量

您可以通过为操作声明命名空间来配置操作以生成变量。该操作必须已经是生成变量的操作提供方之一。否则，可用的变量将为管道级别的变量。

您可以通过以下方法声明命名空间：

- 在控制台的 Edit action (编辑操作) 页面上，在 Variable namespace (变量命名空间) 中输入命名空间。
- 在 JSON 管道结构的 `namespace` 参数字段中输入命名空间。

在此示例中，您将名称为的`namespace`参数添加到 CodeCommit 源操作中`SourceVariables`。这会将操作配置为生成可用于该操作提供方的变量，例如 `CommitId`。

```
{
    "name": "Source",
    "actions": [
        {
            "outputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "name": "Source",
            "namespace": "SourceVariables",
            "configuration": {
                "RepositoryName": "MyRepo",
                "BranchName": "mainline",
                "PollForSourceChanges": "false"
            },
            "inputArtifacts": [],
            "region": "us-west-2",
            "actionTypeId": {
                "provider": "CodeCommit",
                "category": "Source",
                "version": "1",
                "owner": "AWS"
            },
            "runOrder": 1
        }
    ]
```

```
},
```

接下来，您需要配置下游操作来使用由上一个操作生成的变量。您可以通过以下方法执行此操作：

- 在控制台的 Edit action (编辑操作) 页面上，在操作配置字段中输入变量语法（针对下游操作）。
- 在 JSON 管道结构的操作配置字段中输入变量语法（针对下游操作）

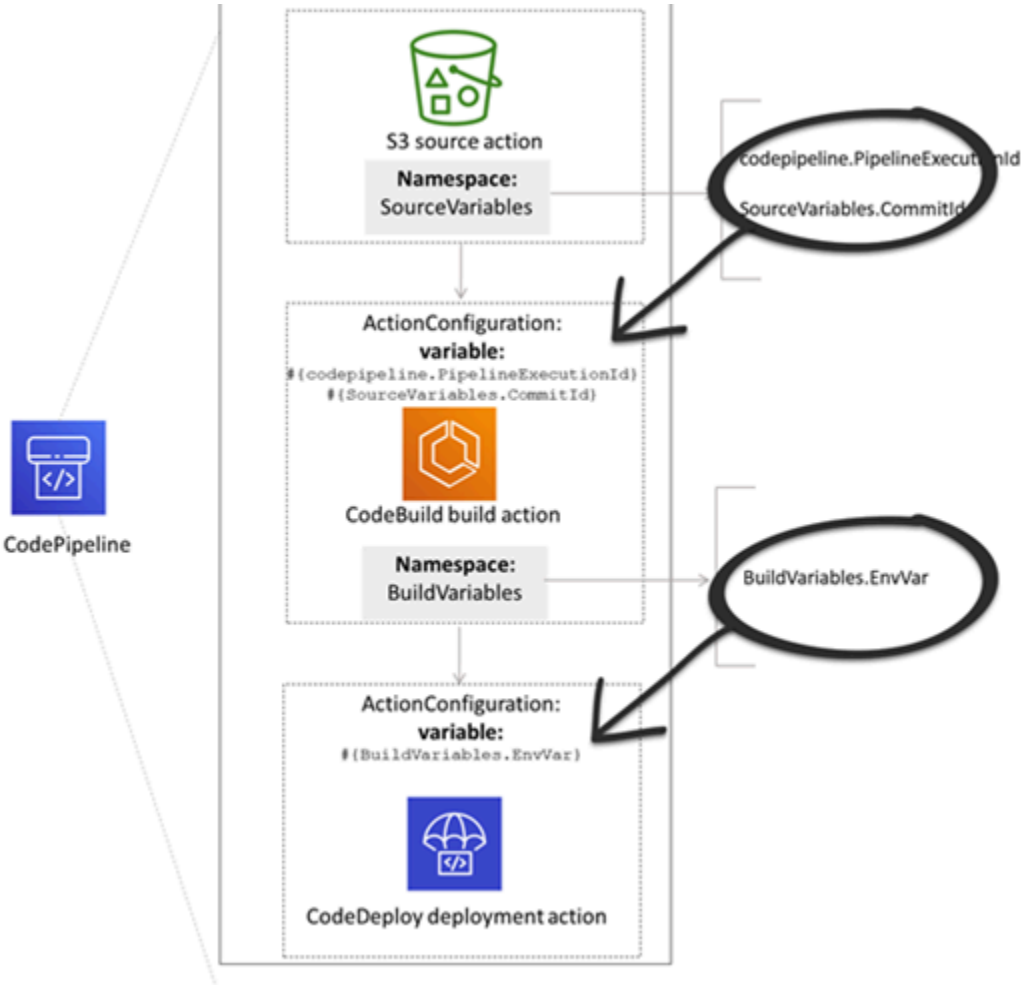
在此示例中，构建操作的配置字段显示在操作执行时更新的环境变量。该示例为执行 ID `#{codepipeline.PipelineExecutionId}` 指定命名空间和变量，并为提交 ID **`#{SourceVariables.CommitId}`** 指定命名空间和变量。

```
{
    "name": "Build",
    "actions": [
        {
            "outputArtifacts": [
                {
                    "name": "BuildArtifact"
                }
            ],
            "name": "Build",
            "configuration": {
                "EnvironmentVariables": "[{\"name\":\"Release_ID\",\"value\":
\"#{codepipeline.PipelineExecutionId}\",\"type\":\"PLAINTEXT\"},{\"name\":\"Commit_ID
\",\"value\":\"#{SourceVariables.CommitId}\",\"type\":\"PLAINTEXT\"}]",
                "ProjectName": "env-var-test"
            },
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-west-2",
            "actionTypeId": {
                "provider": "CodeBuild",
                "category": "Build",
                "version": "1",
                "owner": "AWS"
            },
            "runOrder": 1
        }
    ]
```

```
},
```

变量解析

每当在管道执行中执行某个操作时，该操作生成的变量均可用于确保在该操作之后运行的操作。要在使用操作中使用这些变量，您可以使用前面示例中显示的语法，将变量添加到使用操作的配置中。在它执行消耗操作之前，先 CodePipeline 解析配置中存在的所有变量引用，然后再启动动作执行。



变量规则

以下规则可帮助您进行变量的配置：

- 您可以通过新的操作属性或通过编辑操作，为操作指定命名空间和变量。
- 使用管道创建向导时，控制台会为该向导创建的每个操作生成一个命名空间。
- 如果未指定命名空间，则无法在任何操作配置中引用该操作生成的变量。

- 要引用操作生成的变量，引用操作必须发生在生成变量的操作之后。这意味着引用操作需要在生成变量的操作之后的阶段，或者在相同阶段中但具有较后的运行顺序。

可用于管道操作的变量

操作提供方确定操作可以生成哪些变量。

有关管理变量的 step-by-step 过程，请参见使用变量。

使用已定义变量键的操作

与可选择的命名空间不同，以下操作使用无法编辑的变量键。例如，对于 Amazon S3 操作提供方，只有 `ETag` 和 `VersionId` 变量键可用。

每次执行还具有一组 CodePipeline生成的管道变量，其中包含有关执行的数据，例如管道版本 ID。管道中的任何操作都可以使用这些变量。

主题

- CodePipeline执行 ID 变量
- Amazon ECR 操作输出变量
- AWS CloudFormation StackSets 操作输出变量
- CodeCommit 操作输出变量
- CodeStarSourceConnection 操作输出变量
- GitHub 操作输出变量（GitHub 操作版本 1）
- S3 操作输出变量

CodePipeline执行 ID 变量

CodePipeline执行 ID 变量

提供商	变量键	示例值	变量语法示例
codepipeline	`PipelineExecutionId`	8abc75f0-fbf8-4f4c-bfEXAMPLE	`#{codepipeline.PipelineExecutionId}`

Amazon ECR 操作输出变量

Amazon ECR 变量

变量键	示例值	变量语法示例
`ImageDigest`	sha256:EXAMPLE1122334455	`#{SourceVariables.ImageDigest}`
`ImageTag`	最新	`#{SourceVariables.ImageTag}`
`ImageURI`	11111EXAMPLE.dkr.ecr.us-west-2.amazonaws.com/ecs-repo:latest	`#{SourceVariables.ImageURI}`
`RegistryId`	EXAMPLE12233	`#{SourceVariables.RegistryId}`
`RepositoryName`	my-image-repo	`#{SourceVariables.RepositoryName}`

AWS CloudFormation StackSets 操作输出变量

AWS CloudFormation StackSets 变量

变量键	示例值	变量语法示例
`OperationId`	11111111-2bbb-111-2bbb-11111example	`#{DeployVariables.OperationId}`
`StackSetId`	my-stackset:1111aaaa-1111-2222-2bbb-11111example	`#{DeployVariables.StackSetId}`

CodeCommit 操作输出变量

CodeCommit 变量

变量键	示例值	变量语法示例
AuthorDate	2019-10-29T03:32:21Z	#{SourceVariables.AuthorDate}
BranchName	开发	#{SourceVariables.BranchName}
CommitId	exampleb01f91b31	#{SourceVariables.CommitId}
CommitMessage	修复了一个错误（最大大小为 100 KB）	#{SourceVariables.CommitMessage}
CommitterDate	2019-10-29T03:32:21Z	#{SourceVariables.CommitterDate}
RepositoryName	myCodeCommit回购	#{SourceVariables.RepositoryName}

CodeStarSourceConnection 操作输出变量

CodeStarSourceConnection变量（Bitbucket Cloud GitHub、、 GitHub企业存储库和 GitLab .com）

变量键	示例值	变量语法示例
AuthorDate	2019-10-29T03:32:21Z	#{SourceVariables.AuthorDate}
BranchName	开发	#{SourceVariables.BranchName}
CommitId	exampleb01f91b31	#{SourceVariables.CommitId}

变量键	示例值	变量语法示例
CommitMessage	修复了一个错误（最大大小为 100 KB）	#{SourceVariables.CommitMessage}
ConnectionArn	arn:aws:codestar-connections:region:*account-id* :connection/*connection-id*	#{SourceVariables.ConnectionArn}
FullRepositoryName	用户名/ GitHubRepo	#{SourceVariables.FullRepositoryName}

GitHub 操作输出变量（GitHub 操作版本 1）

GitHub 变量（GitHub 操作版本 1）

变量键	示例值	变量语法示例
AuthorDate	2019-10-29T03:32:21Z	#{SourceVariables.AuthorDate}
BranchName	main	#{SourceVariables.BranchName}
CommitId	exampleb01f91b31	#{SourceVariables.CommitId}
CommitMessage	修复了一个错误（最大大小为 100 KB）	#{SourceVariables.CommitMessage}
CommitterDate	2019-10-29T03:32:21Z	#{SourceVariables.CommitterDate}
CommitUrl		#{SourceVariables.CommitUrl}
RepositoryName	myGitHub回购	#{SourceVariables.RepositoryName}

S3 操作输出变量

S3 变量

变量键	示例值	变量语法示例
`ETag`	example28be1c3	`#{SourceVariables.ETag}`
`VersionId`	exampleta_IUQCv	`#{SourceVariables.VersionId}`

使用用户配置的变量键的操作

对于 CodeBuild AWS CloudFormation、和 Lambda 操作，变量键由用户配置。

主题

- CloudFormation 操作输出变量
- CodeBuild 操作输出变量
- Lambda 操作输出变量

CloudFormation 操作输出变量

AWS CloudFormation 变量

变量键	变量语法示例
对于 AWS CloudFormation 操作，变量由堆栈模板`Outputs`部分中指定的任何值生成。请注意，生成输出的唯一 CloudFormation 操作模式是那些导致创建或更新堆栈的操作模式，例如堆栈创建、堆栈更新和更改集执行。生成变量的相应操作模式包括： • CREATE_UPDATE • CHANGE_SET_EXECUTE • CHANGE_SET_REPLACE • REPLACE_ON_FAILURE	`#{DeployVariables.StackName}`

变量键	变量语法示例
有关这些操作模式的更多信息，请参阅AWS CloudFormation。有关介绍如何在使用 AWS CloudFormation 输出变量的管道中使用 AWS CloudFormation 部署操作创建管道的教程，请参阅教程：创建使用 AWS CloudFormation 部署操作中的变量的管道。	

CodeBuild 操作输出变量

CodeBuild 变量

变量键	变量语法示例
对于 CodeBuild 操作，变量由导出的环境变量生成的值生成。通过编辑您的 CodeBuild操作 CodePipeline 或将 CodeBuild 环境变量添加到构建规范来设置环境变量。 在 CodeBuild 构建规范中添加说明，以便在导出的变量部分下添加环境变量。请参阅 AWS CodeBuild 用户指南 中的 env/exported-variables。	`#{BuildVariables.EnvVar}`

Lambda 操作输出变量

Lambda 变量

变量键	变量语法示例
Lambda 操作将生成包含在 API 请求`outputVariables` 部分中的所有键值对作为变量。PutJobSuccessResult 有关使用上游操作 (CodeCommit) 中的变量并生成输出变量的 Lambda 操作的教程，请参阅。教程：将变量与 Lambda 调用操作一起使用	`#{TestVariables.testRunId}`

使用语法中的 glob 模式

在指定管道构件或源位置中使用的文件或路径时，可以根据操作类型指定构件。例如，对于 S3 操作，您可以指定 S3 对象键。

对于触发器，您可以指定筛选条件。您可以使用 glob 模式来指定筛选条件。示例如下。

当语法为“glob”时，使用有限的模式语言将路径的字符串表示与类似于正则表达式的语法匹配起来。例如：

- `*.java` 指定表示以 .java 结尾的代表文件名的路径
- `*.*` 指定包含圆点的文件名
- `*.{java,class}` 指定以 .java 或 .class 结尾的文件名
- `foo.?` 指定以 foo. 开头并有单字符扩展名的文件名

以下规则用于解释 glob 模式：

- 要在目录边界中指定名称组分的零个或多个字符，请使用 *。
- 要跨目录边界指定名称组分的零个或多个字符，请使用 **。
- 要指定名称组分的一个字符，请使用 ?。
- 要对可解释为特殊字符的字符进行转义，请使用反斜杠字符 (\)。
- 要指定一组字符中的单个字符，请使用 [ ]。
- 要指定位于构建位置或源存储库位置根目录中的单个文件，请使用 `my-file.jar`。
- 要在子目录中指定单个文件，请使用 `directory/my-file.jar` 或 `directory/subdirectory/my-file.jar`。
- 要指定所有文件，请使用 `"**"`。** glob 模式表示匹配任意数量的子目录。
- 要指定名为 `directory` 的目录中的所有文件和目录，请使用 `"directory/**"`。** glob 模式表示匹配任意数量的子目录。
- 要指定名为 `directory` 的目录中的所有文件，而非其任意子目录，请使用 `"directory/*"`。
- 在方括号表达式内，*、？和 \ 字符与自身匹配。如果连字符 (-) 是方括号内的第一个字符，或者如果它在您否定时是 ！ 之后的第一个字符，则它与自身匹配。
- { } 字符是一组子模式，如果组中的任何子模式匹配，则组也匹配。`","` 字符用于分隔子模式。不能对组进行嵌套。

将轮询管道更新为采用建议的更改检测方法

如果您的管道使用轮询来对源更改做出反应，则可以更新该管道以使用推荐的检测方法。有关说明如何更新轮询管道以使用推荐的基于事件的更改检测方法的迁移指南，请参阅迁移轮询管道以使用基于事件的更改检测。

将 GitHub 版本 1 的源操作更新为 GitHub 版本 2 的源操作

在中 AWS CodePipeline，有两个支持的 GitHub 源操作版本：

- 推荐： GitHub 版本 2 操作使用由资源支持的基于 Github 应用程序的身份验证。CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、GitLab .com 和 GitLab 自我管理操作它会在您的 GitHub 组织中安装 C AWS CodeStar onnections 应用程序，以便您可以管理中的访问权限 GitHub。
- 不推荐： GitHub 版本 1 操作使用 OAuth 令牌进行身份验证， GitHub 并使用单独的 webhook 来检测更改。这不再是建议方法。

> **Note**
>
> 亚太地区（香港）、亚太地区（海得拉巴）、亚太地区（雅加达）、亚太地区（墨尔本）、亚太地区（墨尔本）、亚太地区（大阪）、非洲（开普敦）、中东（巴林）、中东（阿联酋）、欧洲（西班牙）、欧洲（苏黎世）、以色列（特拉维夫）或 AWS GovCloud （美国西部）地区不可用。要参考其他可用操作，请参阅 产品和服务与 CodePipeline。有关在欧洲地区（米兰）区域使用此操作的注意事项，请参阅CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、 GitLab .com 和 GitLab 自我管理操作中的说明。

使用 GitHub 版本 2 操作而不是 GitHub 版本 1 操作有一些重要的优点：

- 通过连接， CodePipeline 不再需要 OAuth 应用程序或个人访问令牌来访问您的存储库。创建连接时，需要安装一个管理 GitHub 仓库身份验证并允许组织级别权限的 GitHub 应用程序。您必须以用户身份授权 OAuth 令牌访问存储库。有关基于 OAuth 的 GitHub 访问与基于应用程序 GitHub 的访问对比的更多信息，请参阅。https://docs.github.com/en/developers/apps/differences-between-github-apps-and-oauth-apps
- 当你在 CLI 或中管理 GitHub 版本 2 操作时 CloudFormation，你不必再将个人访问令牌作为密钥存储在 Secrets Manager 中。您不再需要在 CodePipeline 操作配置中动态引用存储的密钥。而是将连接 ARN 添加到操作配置中。有关示例操作配置，请参阅CodeStarSourceConnection 适用于 Bitbucket Cloud GitHub、、 GitHub 企业服务器、 GitLab .com 和 GitLab 自我管理操作。
- 在创建用于 GitHub 版本 2 操作的连接资源时 CodePipeline，您可以使用相同的连接资源将其他支持的服务（例如 CodeGuru Reviewer）与您的存储库关联。

- 在 Github 版本 2 中，你可以克隆仓库以在后续 CodeBuild 操作中访问 git 元数据，而在 Github 版本 1 中，你只能下载源代码。
- 管理员将为组织的存储库安装应用程序。您不再需要跟踪依赖于创建令牌的个人的 OAuth 令牌。

安装到组织的所有应用程序都可以访问同一组存储库。要更改谁可访问每个存储库，请修改每个连接的 IAM 策略。有关示例，请参阅示例：将连接用于指定存储库的范围缩小策略。

您可以使用本主题中的步骤从 CodePipeline 控制台中删除 GitHub 版本 1 源操作并添加 GitHub 版本 2 源操作。

主题

步骤 1：替换版本 1 的 GitHub 操作

使用管道编辑页面将您的版本 1 GitHub 操作替换为版本 2 GitHub 操作。

替换您的版本 1 GitHub 操作

1. 登录 CodePipeline 控制台。
2. 选择管道，然后选择编辑。在源阶段中，选择编辑阶段。将显示一条消息，建议您更新操作。
3. 在操作提供者中，选择 GitHub （版本 2）。
4. 请执行以下操作之一：
   - 在 “连接” 下，如果您尚未创建与提供商的连接，请选择 “连接到” GitHub。继续步骤 2：创建与的连接 GitHub。
   - 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 3：保存连接的源操作。

步骤 2：创建与的连接 GitHub

选择创建连接后，将显示 “Connect t GitHub o” 页面。

要创建与的连接 GitHub

1. 在 “GitHub 连接设置” 下，您的连接名称显示在 “连接名称” 中。

   在 “GitHub 应用程序” 下，选择应用程序安装或选择 “安装新应用程序” 来创建一个。

   > **Note**
   >
   > 您可以为与特定提供程序的所有连接安装一个应用程序。如果您已经安装了该 GitHub 应用程序，请选择它并跳过此步骤。

2. 如果 GitHub 显示的授权页面，请使用您的凭据登录，然后选择继续。
3. 在应用程序安装页面上，一条消息显示该 AWS CodeStar 应用程序正在尝试连接到您的 GitHub 帐户。

   > **Note**
   >
   > 您只需为每个 GitHub 账户安装一次该应用程序。如果您之前已安装了应用程序，则可以选择配置，继续进入应用程序安装的修改页面，也可以使用后退按钮返回到控制台。

4. 在安装 AWS CodeStar 页面上，选择安装。
5. 在 “Connect t o GitHub” 页面上，将显示新安装的连接 ID。选择连接。

第 3 步：保存您的 GitHub 源代码操作

在编辑操作页面上完成更新以保存新的源操作。

保存您的 GitHub 源代码操作

1. 在存储库中，输入第三方存储库的名称。在分支中，输入您希望管道在其中检测源更改的分支。

   > **Note**
   >
   > 在存储库中键入 `owner-name/repository-name`，如以下示例所示：
   >
   > ```
   > my-account/my-repository
   > ```

2. 在输出构件格式中，为构件选择格式。

- 要使用默认方法存储 GitHub 操作的输出对象，请选择CodePipeline 默认。该操作访问存储库中的文件，并将工件 GitHub 存储在管道工件存储区的 ZIP 文件中。
- 要存储包含存储库的 URL 引用的 JSON 文件，以便下游操作可以直接执行 Git 命令，请选择完全克隆。此选项只能由 CodeBuild 下游操作使用。

  如果选择此选项，则需要更新 CodeBuild 项目服务角色的权限，如所示添加连接 Bitbucket、GitHub、En GitHub terprise Server 或 GitLab .com 的 CodeBuild GitClone 权限。如需查看教程以了解如何使用完整克隆选项，请参阅教程：使用带有 GitHub 管道源的完整克隆。

3. 在输出构件中，可以保留此操作的输出对象的名称，例如 `SourceArtifact`。选择完成以关闭编辑操作页面。
4. 选择完成以关闭阶段编辑页面。选择保存以关闭管道编辑页面。

中的配额 AWS CodePipeline

CodePipeline 每个 AWS 账户在每个 AWS 区域中可以拥有的管道、阶段、操作和 webhook 的数量都有配额。这些配额按区域应用，并且可以提高。要请求提高限制，请使用支持中心控制台。

最长可能需要两周的时间来处理提高限额的请求。

资源	默认
操作超时前的时长 （这是可配置的超时。 有关不可配置的超时时间，请参见下表）	AWS CloudFormation 部署操作：3 天 CodeDeploy 和 CodeDeploy ECS（蓝/绿）部署操作：5 天 AWS Lambda 调用操作：24 小时 **Note** 操作正在运行时，请 CodePipeline 定期联系 Lambda 以获取状态。Lambda 函数会使用状态进行回复，其中操作执行为“成功”、“失败”或“进行中”。如果 Lambda 函数在 20 分钟后未发送任何回复，则操作将超时。如果在 20 分钟内，Lambda 函数回复说操作仍在进行中，则 CodePipeline重新启动 20 分钟计时器并重试。如果 24 小时后仍未成功，则将 Lambda 调用操作状态 CodePipeline 设置为失败。 Lambda 对与操作超时无关的 Lambda 函数有单独的超时时间。CodePipeline Amazon S3 部署操作：90 分钟

资源	默认
	Note 如果在部署大型 ZIP 文件期间上传到 S3 超时，则操作会因超时错误而失败。请尝试将 ZIP 文件分解成较小的文件。 手动批准操作账户级别默认超时：7 天 **Note** 对于管道中的特定操作，可以覆盖手动批准操作的默认超时，并且可将其配置为最长 86400 分钟（60 天），最小值为 5 分钟。有关更多信息，请参阅 CodePipeline API 参考ActionDeclaration中的。 配置后，此超时将应用于操作。否则，将使用账户级别的默认值。 所有其他操作：1 小时 **Note** Amazon ECS 部署操作超时可配置为最长一小时（默认超时设置）。

资源	默认
一个 AWS 账户中每个区域的最大管道总数	1000 **Note** 为轮询或基于事件的更改检测配置的管道将计入此配额。
每个 AWS 区域设置为轮询源更改的最大管道数	300 **Note** 这是固定限额，无法更改。如果您达到轮询管道的限制，您仍然可以配置使用基于事件的更改检测的其他管道。有关更多信息，请参阅 源操作和更改检测方法。[1]
一个 AWS 账户中每个区域的最大 Webhook 数量	300
一个 AWS 账户中每个区域的自定义操作数量	50

[1]根据源提供方，按照以下说明更新轮询管道，以使用基于事件的更改检测：

- 要更新 CodeCommit 源操作，请参阅迁移轮询管道（CodeCommit 或 Amazon S3 源代码）（控制台）。
- 要更新 Amazon S3 源操作，请参阅迁移轮询管道（CodeCommit 或 Amazon S3 源代码）（控制台）。
- 要更新 GitHub 源操作，请参阅将轮询管道迁移到 webhook（GitHub 版本 1 源操作）（控制台）。

以下配额 AWS CodePipeline 适用于区域可用性、命名限制和允许的对象大小。这些配额是固定的，不能更改。

有关每个区域的 CodePipeline 服务终端节点列表，请参阅《AWS 一般参考》中的AWS CodePipeline 终端节点和配额。

有关结构要求的信息，请参阅CodePipeline 管道结构参考。

AWS 可以在其中创建管道的区域	美国东部（俄亥俄）
	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）
	加拿大（中部）
	欧洲地区（法兰克福）
	欧洲（苏黎世）*
	以色列（特拉维夫）
	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）
	非洲（开普敦）*
	亚太地区（香港）*
	亚太地区（海得拉巴）
	亚太地区（孟买）
	亚太地区（东京)
	亚太地区 (首尔)
	亚太地区（大阪）

	亚太地区（新加坡） 亚太地区（悉尼） 亚太地区（雅加达） 亚太地区（墨尔本） 南美洲（圣保罗） 中东（巴林）* 中东（阿联酋） AWS GovCloud（美国西部） AWS GovCloud（美国东部）
操作名称中允许的字符	操作名称不能超过 100 个字符。允许的字符包括： 小写字母 a 到 z（含这两个字母）。 大写字母 A 到 Z（含这两个字母）。 数字 0 至 9（含这两个数字）。 特殊字符 .（句点）、@（at 符号）、-（减号）和 _（下划线）。 不允许任何其他字符，例如空格。

操作类型中允许的字符	操作类型名称不能超过 25 个字符。允许的字符包括： 小写字母 a 到 z（含这两个字母）。 大写字母 A 到 Z（含这两个字母）。 数字 0 到 9（含这两个数字）。 特殊字符 .（句点）、@（at 符号）、-（减号）和 _（下划线）。 不允许任何其他字符，例如空格。
构件名称中允许的字符	构件名称不能超过 100 个字符。允许的字符包括： 小写字母 a 到 z（含这两个字母）。 大写字母 A 到 Z（含这两个字母）。 数字 0 至 9（含这两个数字）。 特殊字符 -（减号）和 _（下划线）。 不允许任何其他字符，例如空格。
合作伙伴操作名称中允许的字符	合作伙伴操作名称必须遵循与中的其他操作名称相同的命名惯例和限制 CodePipeline。具体来说，不能超过 100 个字符。允许的字符包括： 小写字母 a 到 z（含这两个字母）。 大写字母 A 到 Z（含这两个字母）。 数字 0 到 9（含这两个数字）。 特殊字符 .（句点）、@（at 符号）、-（减号）和 _（下划线）。 不允许任何其他字符，例如空格。

管道名称中允许的字符	管道名称不能超过 100 个字符。允许的字符包括： 小写字母 a 到 z（含这两个字母）。 大写字母 A 到 Z（含这两个字母）。 数字 0 到 9（含这两个数字）。 特殊字符 .（句点）、@（at 符号）、-（减号）和 _（下划线）。 不允许任何其他字符，例如空格。
阶段名称中允许的字符	阶段名称不能超过 100 个字符。允许的字符包括： 小写字母 a 到 z（含这两个字母）。 大写字母 A 到 Z（含这两个字母）。 数字 0 到 9（含这两个数字）。 特殊字符 .（句点）、@（at 符号）、-（减号）和 _（下划线）。 不允许任何其他字符，例如空格。
操作超时前的时长	CodeBuild 生成操作和测试操作：8 小时 自定义操作：24 小时 Step Functions 调用操作：7 天
操作配置键的最大长度（例如，CodeBuild配置键为`ProjectName` `PrimarySource` 、和`EnvironmentVariables` ）	50 个字符

操作配置值的最大长度（例如， CodeCommit 操作RepositoryName 配置中的配置值应小于 1000 个字符： "RepositoryName": "my-repo-name-less-than-1000-characters")	1000 个字符
每个管道的最大操作数	500
每个管道并发管道执行的最大数量（QUEUED|并行模式）	50
每次并行模式管道执行的最大并发操作执行数	5
一个 Amazon S3 对象的最大文件数	100000
保留管道执行历史信息的最大月数	12
一个阶段中并行操作的最大数量	50
一个阶段中顺序操作的最大数量	50
源阶段中项目的最大大小	存储在 Amazon S3 桶中的构件：7 GB 存储在 CodeCommit 或存储 GitHub 库中的工件：1 GB 例外：如果您使用 AWS Elastic Beanstalk 部署应用程序，则最大构件大小始终为 512 MB。 例外：如果您使用 AWS CloudFormation 部署应用程序，则最大构件大小始终为 256 MB。 例外：如果您使用 CodeDeployToECS 部署应用程序，则最大构件大小始终为 3MB。
部署 Amazon ECS 容器和映像的管道中使用的映像定义 JSON 文件的最大大小	100KB
AWS CloudFormation 动作输入对象的最大大小	256 MB

`CodeDeployToECS` 操作的输入构件的最大大小	3 MB
`Step Functions` 操作的输入构件的最大大小	Step Functions 操作在 Lambda 上运行，因此它的工件大小配额与 Lambda 函数的工件大小配额相同。有关更多信息，请参阅 Lambda 开发人员指南中的 Lambda 配额。
可存储在 `ParameterOverrides` 属性中的 JSON 对象的最大大小	对于 AWS CloudFormation 作为提供者的 CodePipeline 部署操作，该`ParameterOverrides` 属性用于存储为 AWS CloudFormation 模板配置文件指定值的 JSON 对象。`ParameterOverrides` 属性中可存储的 JSON 对象的大小上限是 1 KB。
阶段中的操作数量	最小为 1，最大为 50
每个操作允许的构件数量	如需了解每个操作允许的输入和输出构件数量，请参阅每个操作类型的输入和输出构件数量
管道中的阶段数量	最小为 2，最大为 50
管道标签	标签区分大小写。每个资源最多 50 个。
管道标签键名称	UTF-8 格式的 Unicode 字母、数字、空格和允许使用的字符的任意组合，长度为 1 到 128 个字符。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。标签不能： • 开头 AWS： • 只包含空格 • 以空格结尾 • 包含表情符号或以下任何字符：? ^ * [\ ~ ! # $ % & * ( ) > < | " '

管道标签值	UTF-8 格式的 Unicode 字母、数字、空格和允许使用的字符的任意组合，长度为 1 到 256 个字符。允许使用的字符为 + - = . _ : / @ 一个键只能有一个值，但多个键可以具有相同的值。标签不能： • 开头 AWS： • 只包含空格 • 以空格结尾 • 包含表情符号或以下任何字符：? ^ * [\ ~ ! # $ % & * ( ) > < | " '
触发	在push和pull request配置中，管道定义中最多有 50 个触发器。 每个推送触发器和拉取请求触发器最多有三个过滤器。 Note 不允许在相同的事件类型数组中重复过滤器。 您最多可以为每种事件类型（推送、拉取请求）添加 8 个包含和 8 个排除模式、分支和文件路径。 模式值中允许的字符包括所有字符类型。 对于包含和排除模式，最大长度为 255 个字符。 标签名称最大长度为 255 个字符。 triggers阵列的最大大小不应超过 200 KB

触发过滤器	文件路径： • 模式数量：您最多可以添加 8 个包含模式和 8 个排除模式。 • 图案大小：每个包含或排除图案的大小最多可为 255 个字符。 分支机构： • 模式数量：您最多可以添加 8 个包含模式和 8 个排除模式。 • 图案大小：每个包含或排除图案的大小最多可为 255 个字符。 拉取请求： 分支机构： • 模式数量：您最多可以添加 8 个包含模式和 8 个排除模式。 • 图案大小：每个包含或排除图案的大小最多可为 255 个字符。
名称的唯一性	在单个 AWS 账户中，您在一个 AWS 区域中创建的每个管道都必须具有唯一的名称。您可以在不同 AWS 区域中重复使用管道名称。 管道中的阶段名称必须是唯一的。 阶段中的操作名称必须是唯一的。

输出变量和命名空间的限额	对于某个特定的操作，其所有输出变量的大小总和不得超过 122880 字节。 对于某个特定的操作，其已解析的操作配置的大小总和不得超过 100 KB。 输出变量名称区分大小写。 命名空间区分大小写。 允许的字符包括： • 小写字母 a 到 z（含这两个字母）。 • 大写字母 A 到 Z（含这两个字母）。 • 数字 0 到 9（含这两个数字）。 • 特殊字符 ^（插入符号）、@（at 符号）、-（减号）、_（下划线）、[（左括号）、]（右括号）、*（星号）、$（美元符号）。 不允许任何其他字符，例如空格。
管道级变量的限额	每个管道最多有 50 个管道级变量。 管道级变量的变量名称必须符合以下条件： • 最大长度为 128 个字符 • 小写字母 a 到 z（含这两个字母）。 • 大写字母 A 到 Z（含这两个字母）。 • 数字 0 到 9（含这两个数字）。 • 特殊字符 @\-_]+ 不允许任何其他字符，例如空格。 变量值的最大长度为 1000 个字符 变量值允许使用所有字符。 变量描述的最大长度为 200 个字符

* 您必须启用此区域，然后才能使用它。

附录 A： GitHub 版本 1 源代码操作

本附录提供有关中 GitHub 操作版本 1 的信息 CodePipeline。

> **Note**
>
> 虽然我们不建议使用 GitHub 版本 1 操作，但具有 GitHub 版本 1 操作的现有管道将继续运行而不会产生任何影响。对于具有 GitHub版本 1 操作的管道， CodePipeline 使用基于 OAuth 的令牌连接到您的 GitHub 存储库。相比之下， GitHub 操作（版本 2）使用连接资源将资源与您的 GitHub 存储库相关联 AWS 。该连接资源使用基于应用的令牌进行连接。有关将管道更新为使用连接的推荐 GitHub 操作的更多信息，请参阅将 GitHub 版本 1 的源操作更新为 GitHub 版本 2 的源操作。有关基于 OAuth 的 GitHub 访问与基于应用程序 GitHub 的访问对比的更多信息，请参阅。https://docs.github.com/en/developers/apps/differences-between-github-apps-and-oauth-apps

要与集成 GitHub，请在您的 GitHub 管道中 CodePipeline 使用 OAuth 应用程序。 CodePipeline使用 webhook 通过 GitHub 版本 1 源操作管理管道的变更检测。

> **Note**
>
> 在中配置 GitHub 版本 2 源操作时 AWS CloudFormation，无需包含任何 GitHub 令牌信息或添加 webhook 资源。您可以按照《 AWS CloudFormation 用户指南》AWS::CodeStarConnections::Connection中所示配置连接资源。

本参考包含 GitHub 版本 1 操作的以下章节：

- 有关如何向管道添加 GitHub 版本 1 源操作和 webhook 的信息，请参阅添加 GitHub 版本 1 源操作。
- 有关 GitHub 版本 1 源操作的配置参数和示例 YAML/JSON 片段的信息，请参阅。GitHub 版本 1 源操作结构参考

主题

- 添加 GitHub 版本 1 源操作
- GitHub 版本 1 源操作结构参考

添加 GitHub 版本 1 源操作

您可以 CodePipeline 通过以下方式向其中添加 GitHub 版本 1 源操作：

- 使用 CodePipeline 控制台的 “创建管道” 向导 (创建管道（控制台）) 或 “编辑” 操作页面选择GitHub提供者选项。控制台会创建一个 Webhook，用于在源发生变更时启动您的管道。
- 使用 CLI 添加 `GitHub` 操作的操作配置并创建其他资源，如下所示：
  - 使用 GitHub 版本 1 源操作结构参考中的 `GitHub` 示例操作配置来创建操作，如创建管道（CLI）中所示。
  - 禁用定期检查并手动创建更改检测，因为更改检测方法默认通过轮询源来启动管道。您可以将投票管道迁移到 webhook 以执行 GitHub 版本 1 操作。

GitHub 版本 1 源操作结构参考

> **Note**
>
> 虽然我们不建议使用 GitHub 版本 1 操作，但具有 GitHub 版本 1 操作的现有管道将继续运行而不会产生任何影响。对于具有 GitHub GitHub 版本 1 源操作的管道， CodePipeline 使用基于 OAuth 的令牌连接到您的 GitHub 存储库。相比之下，新 GitHub 操作（版本 2）使用连接资源将 AWS 资源与存储 GitHub 库相关联。该连接资源使用基于应用的令牌进行连接。有关将管道更新为使用连接的推荐 GitHub 操作的更多信息，请参阅将 GitHub 版本 1 的源操作更新为 GitHub 版本 2 的源操作。

在配置的 GitHub 存储库和分支上进行新提交时触发管道。

要与集成 GitHub，请对您的管道 CodePipeline 使用 OAuth 应用程序或个人访问令牌。如果您使用控制台创建或编辑管道，则 CodePipeline 会创建一个 GitHub webhook，当存储库发生更改时，该挂钩会启动您的管道。

在通过 GitHub 操作连接管道之前，您必须已经创建了 GitHub 账户和存储库。

如果您想限制对仓库的访问权限 CodePipeline ，请创建一个 GitHub 帐户，并仅向该账户授予您想要集成的存储库的访问权限 CodePipeline。当您配置 CodePipeline 为使用 GitHub 存储库作为管道中的源阶段时，请使用该帐户。

有关更多信息，请参阅 GitHub 网站上的GitHub 开发者文档。

主题

操作类型

- 类别：`Source`
- 拥有者：`ThirdParty`
- 提供方：`GitHub`
- 版本：`1`

配置参数

所有者

必需：是

拥有 GitHub存储库的 GitHub 用户或组织的名称。

存储库

必需：是

要在其中检测源更改的存储库的名称。

分支

必需：是

要在其中检测源更改的分支的名称。

O AuthToken

必需：是

表示允许对 GitHub 存储库执行操作 CodePipeline 的 GitHub 身份验证令牌。条目始终显示为四个星号的掩码。它表示以下值：

- 使用控制台创建管道时， CodePipeline 使用 OAuth 令牌注册连接。 GitHub
- 使用创建管道时，可以在此字段中传递您的 GitHub 个人访问令牌。 AWS CLI 将星号 (****) 替换为从中复制的个人访问令牌。 GitHub在运行 `get-pipeline` 查看操作配置时，将显示此值的四星号掩码。
- 使用 AWS CloudFormation 模板创建管道时，必须先将令牌作为密钥存储在中 AWS Secrets Manager。您应该动态引用 Secrets Manager 中存储的密钥，将其作为此字段的值包含在内，如 `{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}`。

有关 GitHub 范围的更多信息，请参阅 GitHub 网站上的GitHub 开发者 API 参考。

PollForSourceChanges

必需：否

`PollForSourceChanges`控制是否 CodePipeline 轮询 GitHub存储库以获取源代码更改。我们建议您改为使用 Webhook 来检测源更改。有关配置 Webhook 的更多信息，请参阅将轮询管道迁移到 Webhook（GitHub 版本 1 源操作）(CLI) 或更新推送事件的管道（GitHub 版本 1 源操作）（AWS CloudFormation 模板）。

> **Important**
>
> 如果您要配置 Webhook，则必须将 `PollForSourceChanges` 设置为 `false` 以避免重复的管道执行。

此参数的有效值：

- `True`: 如果已设置，则会 CodePipeline 轮询您的存储库以了解源代码更改。

> **Note**
>
> 如果省略`PollForSourceChanges`，则 CodePipeline 默认为轮询存储库是否有源更改。此行为与将 `PollForSourceChanges` 设置为 `true` 相同。

- `False`: 如果已设置，则 CodePipeline 不会轮询您的存储库以了解源代码更改。如果您打算配置 Webhook 来检测源更改，请使用此设置。

输入构件

- 构件数：`0`
- 描述：输入构件不适用于此操作类型。

输出构件

- 构件数：`1`
- 描述：此操作的输出构件是一个 ZIP 文件，其中包含在提交时，指定作为管道执行的源修订的已配置存储库和分支的内容。从存储库生成的构件是 GitHub 操作的输出对象。源代码提交 ID 显示 CodePipeline 为触发管道执行的源修订版。

输出变量

配置后，此操作会生成变量，该变量可由管道中下游操作的操作配置引用。此操作生成的变量可视为输出变量，即使操作没有命名空间也是如此。您可以使用命名空间配置操作，以使这些变量可用于下游操作的配置。

有关中变量的更多信息 CodePipeline，请参阅Variables。

CommitId

触发管道执行的 GitHub 提交 ID。提交 ID 是提交的完整 SHA。

CommitMessage

与触发管道执行的提交相关联的描述消息（如果有）。

CommitUrl

触发管道的提交的 URL 地址。

RepositoryName

触发管道的提交所在 GitHub 存储库的名称。

BranchName

进行源代码更改的 GitHub 存储库的分支名称。

AuthorDate

授权提交的日期，采用时间戳格式。

有关 Git 中作者和提交者之间区别的更多信息，请参阅 Pro Git 中由 Scott Chacon 和 Ben Straub 撰写的查看提交历史记录。

CommitterDate

进行提交的日期，采用时间戳格式。

有关 Git 中作者和提交者之间区别的更多信息，请参阅 Pro Git 中由 Scott Chacon 和 Ben Straub 撰写的查看提交历史记录。

操作声明（GitHub 示例）

YAML

```
Name: Source
Actions:
  - InputArtifacts: []
    ActionTypeId:
      Version: '1'
      Owner: ThirdParty
      Category: Source
      Provider: GitHub
    OutputArtifacts:
      - Name: SourceArtifact
    RunOrder: 1
    Configuration:
      Owner: MyGitHubAccountName
      Repo: MyGitHubRepositoryName
      PollForSourceChanges: 'false'
      Branch: main
      OAuthToken: '{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}'
    Name: ApplicationSource
```

JSON

```
{
    "Name": "Source",
    "Actions": [
```

```
        {
            "InputArtifacts": [],
            "ActionTypeId": {
                "Version": "1",
                "Owner": "ThirdParty",
                "Category": "Source",
                "Provider": "GitHub"
            },
            "OutputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "RunOrder": 1,
            "Configuration": {
                "Owner": "MyGitHubAccountName",
                "Repo": "MyGitHubRepositoryName",
                "PollForSourceChanges": "false",
                "Branch": "main",
                "OAuthToken":
 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}"
            },
            "Name": "ApplicationSource"
        }
    ]
},
```

正在连接到 GitHub (OAuth)

首次使用控制台向管道添加 GitHub 存储库时，系统会要求您授予对仓库的 CodePipeline 访问权限。该令牌需要以下 GitHub 范围：

- `repo` 范围，用于完全控制从公有和私有存储库读取项目并将项目提取到管道中的过程。
- `admin:repo_hook` 范围，用于完全控制存储库挂钩。

使用 CLI 或 AWS CloudFormation 模板时，必须提供已在中创建的个人访问令牌的值 GitHub。

另请参阅

下列相关资源在您使用此操作的过程中会有所帮助。

- 《AWS CloudFormation 用户指南》的资源参考 AWS::CodePipeline::Webhook — 其中包括中 AWS CloudFormation资源的字段定义、示例和片段。
- AWS CloudFormation 用户指南 AWS::CodeStar::GitHub存储库的资源参考 — 其中包括中 AWS CloudFormation资源的字段定义、示例和片段。
- 教程：创建用于构建和测试您的 Android 应用程序的管道 AWS Device Farm— 本教程提供了示例构建规范文件和示例应用程序，用于创建带有 GitHub 源代码的管道。它使用 CodeBuild 和构建和测试一款 Android 应用程序 AWS Device Farm。

AWS CodePipeline 用户指南文档历史记录

下表描述了每个版本的《 CodePipeline 用户指南》中的重要更改。要获得本文档的更新通知，您可以订阅 RSS 源。

- API 版本：2015-07-09
- 最新文档更新：2024 年 5 月 7 日

变更	说明	日期
对S3源操作进行了更新，为源重写添加了新选项	名为的源重写新选项可用S3_OBJECT_KEY 于 s S3 ource 操作。已AllowOverrideForS3ObjectKey 为 S3 源操作添加了一个新参数。请参阅 Amazon S3 源操作参考页面和使用源版本覆盖启动管道。	2024年6月7日
更新S3源操作以添加新的输出变量	名为BucketName 和的新输出变ObjectKey 量可用于S3源操作。请参阅 A mazon S3 源操作参考页面。	2024年6月5日
Support 支持将 V1 类型的管道移至 V2 类型的管道的成本分析	添加了该PipelineCostAnalyzer.py 脚本，用于分析将现有 V1 类型管道移至 V2 类型管道的运行成本。请参阅哪种类型的管道适合我？ 。	2024 年 5 月 30 日
CloudFormationStackSet 和 CloudFormationStackInstances 操作的更新	为 CloudFormationStackSet 和 CloudFormationStackInstances 操作添加了 CallAs 参数。请参阅操作参考页面。	2024年5月2日

Support 支持阶段级回滚	您可以手动或自动将阶段回滚到该阶段之前成功执行的管道中。请参阅配置阶段回滚和概念。	2024 年 4 月 26 日
更新了 Step Functi StackSets ons 操作的区域可用性	StackSets 和 Step Function CodePipeline s 操作现在可在所有可用的区域中使用。参见AWS CloudFormation StackSets 操作参考和 AWS Step Functions 操作参考。	2024 年 3 月 27 日
托管式策略的更新	AWS 托管策略AWSCodePipeline_FullAccess 已更新。请参阅适用于 AWS CodePipeline的AWS 托管策略。	2024 年 3 月 15 日
Support 支持手动批准操作的可配置超时	为手动批准操作的新可配置超时字段添加了配额信息。有关更多信息，请参阅 配额。	2024 年 2 月 15 日
Support 支持按分支和文件路径进行触发器筛选	增加了对触发器配置的支持，该配置允许筛选 V2 类型管道的拉取请求状态、分支和文件路径。有关更多信息，请参阅筛选代码推送或拉取请求上的触发器、触发器和筛选功能分支以启动管道以及配额。	2024 年 2 月 8 日
Support 支持新的管道执行模式	增加了对并行和排队管道执行模式的支持。有关更多信息，请参阅设置管道执行模式、如何在 QUEUED 模式下处理执行、如何在并行模式下处理执行以及配额。	2024 年 2 月 8 日

更新了用于查看操作详细信息、查看手动审批操作的控制台页面和“列出管道”页面	记录了控制台的更新，包括新的查看详细信息按钮和对话框、新的手动审批对话框以及“列出管道”页面上用于记录最近执行的新列。有关更多信息，请参阅查看管道（控制台）、查看管道中的操作详细信息和管理管道中的审批操作。	2024 年 1 月 10 日
Support 支持 GitLab 自我管理	添加了 Support，支持为 AWS 资源配置连接，以便与 GitLab 自我管理进行交互。有关更多信息，请参阅 GitLab自我管理连接。	2023 年 12 月 28 日
CloudFormationStackSet 和 CloudFormationStackInstances 操作的更新	为 CloudFormationStackSet 和 CloudFormationStackInstances 操作添加了 ConcurrencyMode 参数。请参阅操作参考页面。	2023 年 12 月 19 日
中对 AWS Device Farm 操作参数的更新 CodePipeline	中 AWS Device Farm 操作的参数 CodePipeline 已更新。有关更多信息，请参阅 AWS Device Farm 操作参考。	2023 年 12 月 18 日
为中 AWS CloudFormation 操作的详细错误消息添加了 Support CodePipeline	AWS CloudFormation 操作错误消息现在可以显示有关失败资源的详细信息。有关更多信息，请参阅 AWS CloudFormation 操作参考。	2023 年 12 月 15 日
用于启动带有源修订版本覆盖的管道的更新 CodePipeline	现在，您可以使用指定的源修订启动管道。有关更多信息，请参阅使用源修订覆盖启动管道。	2023 年 11 月 17 日

新的支持区域	CodePipeline 现已在亚太地区（海得拉巴）、亚太地区（雅加达）、亚太地区（墨尔本）、亚太地区（大阪）、中东（阿联酋）、欧洲（西班牙）和以色列（特拉维夫）地区推出。事件占位符桶参考主题和AWS 服务 端点主题已更新。	2023 年 11 月 13 日
Amazon 中事件字段的更新 EventBridge	现在，您可以在 Amazon 中查看更新的事件字段 EventBridge。有关更多信息，请参阅监控 CodePipeline事件。	2023 年 11 月 9 日
更新了新的管道类型 V2 管道、Git 标签上的触发器和中的管道变量 CodePipeline	现在，您可以在中选择管道类型 CodePipeline。对于 V2 类型的管道，您现在可以在 Git 标签上使用触发器配置启动管道。在 V2 类型的管道中，您还可以使用管道级的变量为管道执行传递输入参数。有关更多信息，请参阅变量、教程：使用管道级变量和教程：使用 Git 标签启动管道。有关管道类型的更多信息，请参阅管道类型。	2023 年 10 月 24 日
CodePipeline 允许在失败阶段重试所有操作	对于中失败的阶段 CodePipeline，您可以重试该阶段，而无需重新运行管道。为此，您可以重试阶段中失败的操作，也可以从阶段中的第一个操作开始重试所有操作。有关更多信息，请参阅重试失败的阶段或阶段中失败的操作。	2023 年 10 月 17 日

对 GitLab 群组的 Support	添加了 Support，支持配置 AWS 资源与 GitLab群组交互的连接。有关更多信息，请参阅GitLab 连接。	2023 年 9 月 15 日
CodePipeline 支持与 GitLab .com 的连接	您可以使用连接来配置 AWS 资源以与 GitLab .com 进行交互。您也可以选择完整克隆选项，将 Git 命令和元数据用于下游操作。有关更多信息，请参阅GitLab 连接和CodeStarSourceConnection 操作结构参考主题。	2023 年 8 月 10 日
CloudFormationStackInstances 操作的更新	为 CloudFormationStackInstances 操作添加了 RegionConcurrencyType 参数。请参阅 CloudFormationStackInstances 操作的操作参考页面。	2023 年 8 月 8 日
CloudFormationStackSet 操作的更新	为 CloudFormationStackSet 操作添加了 RegionConcurrencyType 参数。请参阅 CloudFormationStackSet 操作的操作参考页面。	2023 年 7 月 24 日
托管式策略的更新	AWS 托管策略AWSCodePipeline_FullAccess 已更新。请参阅适用于 AWS CodePipeline的AWS 托管策略。	2023 年 6 月 21 日

轮询管道迁移过程的更新	迁移（更新）轮询管道以使用基于事件的更改检测的过程已更新，其中包含使用启用通知的 Amazon S3 存储桶的管道的步骤。 EventBridge有关更多信息，请参阅迁移轮询管道以使用基于事件的更改检测。	2023 年 6 月 12 日
托管式策略的更新	AWS 托管策略AWSCodePipeline_FullAccess 和AWSCodePipeline_ReadOnlyAccess 已更新，增加了额外的权限。有关更多信息，请参阅AWS 托管策略的AWS CodePipeline 更新。	2023 年 5 月 16 日
托管式策略的更新	AWS 托管策略AWSCodePipelineFullAccess 和已AWSCodePipelineReadOnlyAccess 被弃用。使用 AWSCodePipeline_FullAccess 和 AWSCodePipeline_ReadOnlyAccess 策略。请参阅 AWS CodePipeline 对 AWS 托管式策略的更新。	2022 年 11 月 17 日

对使用的过程的更新 CloudTrail	所有控制台程序、示例 CLI 命令以及带有 S3 源的管道的示例 AWS CloudFormation 片段和模板均已更新，在管理事件中 CloudTrail可以选择 Write 并选择 false。请参阅启动管道、教程：使用创建管道 AWS CloudFormation、编辑管道以使用推送事件和更新轮询管道中更新的示例。	2022 年 4 月 27 日
新支持的 Snyk 集成	您可以使用中的 Snyk invoke 操作 CodePipeline 来自动对开源代码进行安全扫描。有关更多信息，请参阅 Snyk 操作参考和集成。	2021 年 6 月 10 日
新支持的欧洲地区（米兰）区域	CodePipeline 现已在欧洲（米兰）上市。限制主题和 AWS 服务 端点主题已更新。	2021 年 1 月 27 日
通过连接可关闭源操作的更改检测	您可以使用 CLI 或 SDK 更新 `CodeStarSourceConnection` 源操作，以关闭对源存储库的自动更改检测。CodeStarSourceConnection 操作结构参考主题已使用该`DetectChanges` 参数的描述进行了更新。	2021 年 1 月 8 日

CodePipeline 现在支持 AWS CloudFormation StackSets 部署操作	新教程 “教程：创建 AWS CloudFormation StackSets 用作部署提供程序的管道” 提供了使用 AWS CloudFormation StackSets 管道创建和更新堆栈集和堆栈实例的步骤。还添加了AWS CloudFormation StackSets 操作结构参考主题。	2020 年 12 月 30 日
新支持的亚太地区（香港）区域	CodePipeline 现已在亚太地区（香港）上市。限制主题和 AWS 服务 端点主题已更新。	2020 年 12 月 22 日
在中查看更新的 EventBridge 事件模式 CodePipeline	已在 “监控” 事件中添加了管道、阶段和操作级别事件的更新 CodePipeline 事件模式和状态。	2020 年 12 月 21 日
在中查看入站管道执行情况 CodePipeline	您可以使用控制台或 CLI 查看入站执行状态。有关更多信息，请参阅查看入站执行状态（控制台）和查看入站执行状态 (CLI)。	2020 年 11 月 16 日
中的 CodeCommit 源操作 CodePipeline 支持完全克隆选项	使用 CodeCommit 源操作时，可以选择完全克隆选项，以便将 Git 命令和元数据用于下游 CodeBuild 操作。有关更多信息，请参阅CodeCommit 操作参考和教程：使用带有 CodeCommit 管道源的完整克隆。	2020 年 11 月 11 日

CodePipeline 支持 GitHub 与 GitHub 企业服务器的连接	您可以使用连接来配置要与之交互的 AWS 资源 GitHub、GitHub企业云和 GitHub 企业服务器。您也可以选择完整克隆选项，将 Git 命令和元数据用于下游操作。有关更多信息，请参阅GitHub 连接、GitHub 企业服务器连接和教程：使用带有 GitHub 管道源的完整克隆。如果您有包含 GitHub源操作的现有管道，请参阅将 GitHub 版本 1 的源操作更新为 GitHub 版本 2 的源操作。	2020 年 9 月 30 日
该 CodeBuild 操作支持在中启用批量构建 AWS CodePipeline	对于管道中的 CodeBuild 操作，您可以启用批量构建，以便在一次执行中运行多个构建。有关更多信息，请参阅CodeBuild 操作结构参考和创建管道（控制台）。	2020 年 7 月 30 日
AWS CodePipeline 现在支持 AWS AppConfig 部署操作	新教程 “教程：创建 AWS AppConfig 用作部署提供程序的管道” 提供了使用 AWS AppConfig 管道部署配置文件的步骤。还添加了AWS AppConfig 操作结构参考主题。	2020 年 6 月 25 日
AWS CodePipeline 现在支持 AWS GovCloud（美国西部）的 Amazon VPC	现在，您可以 AWS CodePipeline 通过 AWS GovCloud（美国西部）的私有 Amazon VPC 终端节点直接连接。有关更多信息，请参阅使用亚马逊 Virtual CodePipeline Private Cloud。	2020 年 6 月 2 日

AWS CodePipeline 现在支持 AWS Step Functions 调用操作	现在，您可以在中创建 AWS Step Functions 用作调 CodePipeline 用操作提供者的管道。新教程 “教程：在管道中使用 AWS Step Functions 调用操作” 提供了从管道启动状态机执行的步骤。还添加了 AWS Step Functions 操作结构参考主题。	2020 年 5 月 28 日
查看、列出和更新连接	您可以在控制台中列出、删除和更新连接。请参阅中的列出连接 CodePipeline。	2020 年 5 月 21 日
Connections 支持在 CLI 中标记连接资源	连接资源现在支持在 AWS CLI 中进行标记。连接现在与集成 AWS CodeGuru。请参阅连接的 IAM 权限参考	2020 年 5 月 6 日
CodePipeline 现已在 AWS GovCloud （美国西部）上市	您现在可以 CodePipeline 在 AWS GovCloud （美国西部）中使用。有关更多信息，请参阅 配额。	2020 年 4 月 8 日
配额主题显示哪些 CodePipeline 服务配额是可配置的	CodePipeline 配额主题已重新格式化。该文档说明了哪些服务配额是可配置的，哪些配额是不可配置的。请参阅中的配额 AWS CodePipeline。	2020 年 3 月 12 日
Amazon ECS 部署操作超时是可配置的	Amazon ECS 部署操作超时可配置为最长一小时（默认超时设置）。请参阅 AWS 中的配额CodePipeline。	2020 年 2 月 5 日

新主题介绍了如何停止管道执行	您可以在中停止管道执行 CodePipeline。您可以指定在允许进行中的操作完成后停止执行，也可以指定立即停止执行并放弃进行中的操作。请参阅中的如何停止管道执行和停止管道执行 CodePipeline。	2020 年 1 月 21 日
CodePipeline 支持连接	您可以使用连接来配置 AWS 资源以与外部代码存储库进行交互。每个连接都是一种资源，可供诸如 CodePipeline 连接第三方存储库（例如 Bitbucket Cloud）之类的服务使用。有关更多信息，请参阅中的使用连接 CodePipeline。	2019 年 12 月 18 日
更新了安全、身份验证和访问控制主题	的安全、身份验证和访问控制信息 CodePipeline 已整理到新的安全章节中。关更多信息，请参阅安全性。	2019 年 12 月 17 日
新主题介绍如何在管道中使用变量	现在，您可以为操作配置命名空间，并在每次操作执行完成时生成变量。您可以设置下游操作来引用这些命名空间和变量。请参阅使用变量和变量。	2019 年 11 月 14 日

新主题介绍了管道执行的工作原理、为什么在执行期间锁定阶段以及何时取代管道执行	在“欢迎使用”部分中添加的多个主题描述了管道执行的工作原理，包括为什么在执行期间锁定阶段，以及当管道执行被取代时会发生什么情况。这些主题包括概念列表、 DevOps 工作流程示例以及有关如何构建管道的建议。添加了以下主题：管道术语、DevOps 管道示例和管道执行的工作原理。	2019 年 11 月 11 日
CodePipeline 支持通知规则	现在，您可以使用通知规则向用户通知管道中的重要更改。有关更多信息，请参阅创建通知规则。	2019 年 11 月 5 日
CodeBuild 中可用的环境变量 CodePipeline	您可以在管道的 CodeBuild 生成操作中设置 CodeBuild 环境变量。您可以使用控制台或 CLI 将 `EnvironmentVariables` 参数添加到管道结构。更新了创建管道（控制台）主题。的操作参考中的操作配置示例也CodeBuild已更新。	2019 年 10 月 14 日
新区域	CodePipeline 现已在欧洲（斯德哥尔摩）上市。限制主题和 AWS 服务 端点主题已更新。	2019 年 9 月 5 日

指定 Amazon S3 部署操作的标准 ACL 和缓存控制	现在，您可以在中创建 Amazon S3 部署操作时指定固定的 ACL 和缓存控制选项 CodePipeline。以下主题已更新：创建管道（控制台）、CodePipeline 管道结构参考和教程：创建使用 Amazon S3 作为部署提供商的管道。	2019 年 6 月 27 日
现在，您可以在中为资源添加标签 AWS CodePipeline	现在，您可以使用标记来跟踪和管理 AWS CodePipeline 资源，例如管道、自定义操作和 webhook。添加了以下新主题：标记资源、使用标签控制对资源的访问权限、在中标记管道、在中标记自定义操作以及在 CodePipeline webhook 中 CodePipeline标记 webhook。 CodePipeline CodePipeline以下主题已更新，显示如何使用 CLI 标记资源：创建管道 (CLI)、创建自定义操作 (CLI) 和为 GitHub源创建 Webhook。	2019 年 5 月 15 日
现在，您可以在中查看操作执行历史记录 AWS CodePipeline	现在，您可以查看有关管道中所有操作的过去执行情况的详细信息。这些详细信息包括开始和结束时间、持续时间、操作执行 ID、状态、输入和输出构件位置详细信息以及外部资源详细信息。更新了查看管道详细信息和历史记录主题以反映此支持。	2019 年 3 月 20 日

AWS CodePipeline 现在支持将应用程序发布到 AWS Serverless Application Repository	现在，您可以在中创建一个管道 CodePipeline ，将您的无服务器应用程序发布到。AWS Serverless Application Repository新教程 “教程：将应用程序发布到 AWS Serverless Application Repository上” 提供了创建和配置管道的步骤，以便将您的无服务器应用程序持续交付到 AWS Serverless Application Repository。	2019 年 3 月 8 日
AWS CodePipeline 现在支持控制台中的跨区域操作	现在，您可以在 AWS CodePipeline 控制台中管理跨区域操作。更新了添加跨区域操作，现在包含添加、编辑或删除与管道位于不同 AWS 区域中的操作的步骤。“创建管道”、“编辑管道” 和 “CodePipeline管道结构” 参考主题已更新。	2019 年 2 月 14 日
AWS CodePipeline 现在支持 Amazon S3 部署	现在，您可以在中 CodePipeline 创建一个使用 Amazon S3 作为部署操作提供者的管道。新教程 “教程：创建使用 Amazon S3 作为部署提供商的管道” 提供了使用将示例文件部署到 Amazon S3 存储桶的步骤 CodePipeline。CodePipeline管道结构参考主题也已更新。	2019 年 1 月 16 日

AWS CodePipeline 现在支持 Alexa 技能套件部署	现在，你可以使用 CodePipeline Alexa 技能套件来持续部署 Alexa 技能。新教程 “教程：创建部署 Amazon Alexa 技能的管道” 包含创建证书的步骤，这些证书允许 AWS CodePipeline 连接您的 Alexa Skills Kit 开发者账户，然后创建部署示例技能的管道。CodePipeline 管道结构参考主题已更新。	2018 年 12 月 19 日
AWS CodePipeline 现在支持由以下设备提供支持的 Amazon VPC 终端节点 AWS PrivateLink	现在，您可以 AWS CodePipeline 通过 VPC 中的私有终端节点直接连接，从而将所有流量保留在您的 VPC 和 AWS 网络内。有关更多信息，请参阅使用亚马逊 Virtual CodePipeline Private Cloud。	2018 年 12 月 6 日
AWS CodePipeline 现在支持 Amazon ECR 源代码操作和 ECS 到部署操作 CodeDeploy	现在，您可以将 CodePipeline 和 CodeDeploy 与 Amazon ECR 和 Amazon ECS 配合使用，持续部署基于容器的应用程序。新教程 “创建带有 Amazon ECR 源和 ECS 到 CodeDeploy 部署的管道”，其中包含使用控制台创建管道的步骤，该管道将存储在映像存储库中的容器应用程序部署到具有流量路由的 Amazon ECS 集群。 CodeDeploy 创建管道和CodePipeline 管道结构参考主题已更新。	2018 年 11 月 27 日

AWS CodePipeline 现在支持管道中的跨区域操作	新主题“添加跨区域操作”包含使用 AWS CLI 或 AWS CloudFormation 添加与您的管道位于不同区域的操作的步骤。“创建管道”、“编辑管道”和“CodePipeline 管道结构”参考主题已更新。	2018 年 11 月 12 日
AWS CodePipeline 现已与 Service Catalog 集成	现在可以将 Service Catalog 作为部署操作添加到管道中。这样，当您在源存储库中进行更改时，您就可以设置一个管道将产品更新发布到 Service Catalog。更新了主题集成，以反映对 Service Catalog 的这种支持。已向 AWS CodePipeline 教程部分添加两个 Service Catalog 教程。	2018 年 10 月 16 日
AWS CodePipeline 现在集成了 AWS Device Farm	现在，您可以将其 AWS Device Farm 作为测试操作添加到您的管道中。这样，您可以设置一个管道来测试移动应用程序。集成主题已更新，以反映对的支持。 AWS Device Farm已向 AWS CodePipeline 教程部分添加两个 AWS Device Farm 教程。	2018 年 7 月 19 日

AWS CodePipeline 《用户指南》更新通知现已通过 RSS 提供	HTML 版本的《 CodePipeline 用户指南》现在支持 RSS 提要，其中包含文档更新历史记录页面中记录的更新。RSS 源包括 2018 年 6 月 30 日及之后所做的更新。此前宣布的更新在“文档更新历史记录”页面中仍可用。使用顶部菜单面板中的 RSS 按钮，订阅此源。	2018 年 6 月 30 日

早期更新

下表描述了 2018 年 6 月 30 日及更早版本的《 CodePipeline 用户指南》中的重要更改。

更改	描述	更改日期
使用 webhook 检测管道中的 GitHub 源代码更改	CodePipeline 现在，当您在控制台中创建或编辑管道时，会创建一个 webhook 来检测 GitHub 源存储库的更改，然后启动您的管道。有关迁移管道的信息，请参阅将您的管道配置为使用 Webhook 进行更改检测。 GitHub有关更多信息，请参阅中的启动管道执行 CodePipeline。	2018 年 5 月 1 日
更新的主题	CodePipeline 现在，当您在控制台中创建或编辑管道时，会创建一个 Amazon Ev CloudWatch ents 规则和一条 AWS CloudTrail 跟踪，用于检测您的 Amazon S3 源存储桶的更改，然后启动您的管道。有关迁移管道的信息，请参阅 源操作和更改检测方法。 教程：创建一个简单的管道（S3 存储桶）已更新，以显示在您选择 Amazon CloudWatch S3 源时如何创建 Amazon Events 规则和跟踪。 在中创建管道 CodePipeline并且也在中编辑管道 CodePipeline已更新。 有关更多信息，请参阅 在中启动管道 CodePipeline。	2018 年 3 月 22 日
更新了主题	CodePipeline 现已在欧洲（巴黎）上市。更新了中的配额 AWS CodePipeline主题。	2018 年 2 月 21 日

更改	描述	更改日期
更新的主题	现在，您可以使用 CodePipeline 和 Amazon ECS 来持续部署基于容器的应用程序。在创建管道时，您可以选择 Amazon ECS 作为部署提供方。如果更改源控制存储库中的代码，则会触发管道生成新的 Docker 映像，将其推送到您的容器注册表，然后将更新的映像部署到 Amazon ECS 服务中。 更新了产品和服务与 CodePipeline、在中创建管道 CodePipeline和CodePipeline 管道结构参考主题以反映为 Amazon ECS 提供的这种支持。	2017 年 12 月 12 日
更新的主题	CodePipeline 现在，当您在控制台中创建或编辑管道时，会创建一个 Amazon Ev CloudWatch ents 规则，用于检测 CodeCommit 存储库的更改，然后自动启动您的管道。有关迁移现有管道的信息，请参阅 源操作和更改检测方法。 教程：创建简单的管道（CodeCommit存储库）已更新，以显示在您选择 CodeCommit 存储库和分支时如何创建 Amazon CloudWatch Events 规则和角色。 在中创建管道 CodePipeline并且也在中编辑管道 CodePipeline已更新。 有关更多信息，请参阅 在中启动管道 CodePipeline。	2017 年 10 月 11 日
新增和更新的主题	CodePipeline 现在通过亚马逊 CloudWatch 事件和亚马逊简单通知服务 (Amazon SNS) Simple Notification Service 为管道状态变更通知提供内置支持。增加了新教程教程：设置 CloudWatch 事件规则以接收有关管道状态更改的电子邮件通知。有关更多信息，请参阅 监视 CodePipeline 事件。	2017 年 9 月 8 日
新增和更新的主题	现在，您可以将其添加 CodePipeline 为 Amazon Ev CloudWatch ents 操作的目标。可以将 Amazon Ev CloudWatch ents 规则设置为检测源更改，以便在这些更改发生后立即启动管道，也可以将其设置为运行定期的管道执行。已为 PollForSourceChanges 源操作配置选项添加信息。有关更多信息，请参阅 在中启动管道 CodePipeline。	2017 年 9 月 5 日

更改	描述	更改日期
新的 区域	CodePipeline 现已在亚太地区（首尔）和亚太地区（孟买）上市。更新了中的配额 AWS CodePipeline主题及区域和终端节点主题。	2017 年 7 月 27 日
新的 区域	CodePipeline 现已在美国西部（加利福尼亚北部）、加拿大（中部）和欧洲（伦敦）上市。更新了中的配额 AWS CodePipeline主题及区域和终端节点主题。	2017 年 6 月 29 日
更新的主题	现在，您可以查看有关管道的以往执行的详细信息，而不仅仅是近期执行。这些详细信息包括开始和结束时间、持续时间及执行 ID。提供最近 12 个月内最多 100 次管道执行的详细信息。更新了主题 在中查看管道和详细信息 CodePipeline、CodePipeline 权限参考 和 中的配额 AWS CodePipeline 以反映这种支持。	2017 年 6 月 22 日
更新了主题	Nouvola 已添加到 测试操作集成 中的可用操作列表。	2017 年 5 月 18 日
更新的主题	在 AWS CodePipeline 向导中，“步骤 4：测试版” 页面已重命名为 “步骤 4：部署”。此步骤创建的阶段的默认名称已从“Beta”更改为“Staging”。许多主题和屏幕截图都已更新，以反映这些更改。	2017 年 4 月 7 日
更新的主题	现在，您可以 AWS CodeBuild 作为测试操作添加到管道的任何阶段。这使您可以更轻松地使用 AWS CodeBuild 对代码运行单元测试。在此版本之前，您只能 AWS CodeBuild 将单元测试作为生成操作的一部分来运行。生成操作需要生成输出构件，而单元测试通常不生成该项目。 主题产品和服务与 CodePipeline、在中编辑管道 CodePipeline、和CodePipeline 管道结构参考已更新，以反映对的支持 AWS CodeBuild。	2017 年 3 月 8 日

更改	描述	更改日期
新增和更新的主题	目录已经过重组，以包括有关管道、操作和阶段过渡的章节。为 CodePipeline 教程添加了一个新章节。为了方便使用，产品和服务与 CodePipeline已分割为更短的主题。 新增的 “授权和访问控制” 部分提供了有关使用 AWS Identity and Access Management (IAM) 的全面信息，以及 CodePipeline 如何通过使用证书来帮助安全访问您的资源。这些证书提供访问 AWS 资源所需的权限，例如从 Amazon S3 存储桶中放入和检索工件，以及将 AWS OpsWorks 堆栈集成到您的管道中。	2017 年 2 月 8 日
新的 区域	CodePipeline 现已在亚太地区（东京）上市。更新了中的配额 AWS CodePipeline主题及区域和终端节点主题。	2016 年 12 月 14 日
新的 区域	CodePipeline 现已在南美洲（圣保罗）上市。更新了中的配额 AWS CodePipeline主题及区域和终端节点主题。	2016 年 12 月 7 日
更新的主题	现在，您可以将生成操作 AWS CodeBuild 作为生成操作添加到管道的任何阶段。 AWS CodeBuild 是云端完全托管的构建服务，它可以编译您的源代码、运行单元测试并生成随时可以部署的工件。您可以使用现有的构建项目或在 CodePipeline 控制台中创建一个。然后，可以将生成项目的输出作为管道的一部分部署。 “身份验证和访问控制” 和 “主题产品和服务与 CodePipeline” CodePipeline 管道结构参考 已更新，以反映对的支持 AWS CodeBuild。在中创建管道 CodePipeline 现在，您可以使用 CodePipeline AWS CloudFormation 和 AWS 无服务器应用程序模型来持续交付您的无服务器应用程序。更新了主题产品和服务与 CodePipeline，以反映这一支持。 产品和服务与 CodePipeline已重组为按行动类型分组 AWS 和合作伙伴提供的产品。	2016 年 12 月 1 日

更改	描述	更改日期
新的 区域	CodePipeline 现已在欧洲（法兰克福）上市。更新了中的配额 AWS CodePipeline主题及区域和终端节点主题。	2016 年 11 月 16 日
更新的主题	AWS CloudFormation 现在可以被选为管道中的部署提供者，使您能够在管道执行过程中对 AWS CloudFormation 堆栈和更改集执行操作。“身份验证和访问控制” 和 “主题产品和服务与 CodePipeline” CodePipeline 管道结构参考 已更新，以反映对的支持 AWS CloudFormation。在中创建管道 CodePipeline	2016 年 11 月 3 日
新的 区域	CodePipeline 现已在亚太地区（悉尼）地区推出。更新了中的配额 AWS CodePipeline主题及区域和终端节点主题。	2016 年 10 月 26 日
新的 区域	CodePipeline 现已在亚太地区（新加坡）上市。更新了中的配额 AWS CodePipeline主题及区域和终端节点主题。	2016 年 10 月 20 日
新的 区域	CodePipeline 现已在美国东部（俄亥俄州）地区推出。更新了中的配额 AWS CodePipeline主题及区域和终端节点主题。	2016 年 10 月 17 日
更新了主题	更新了在中创建管道 CodePipeline，以反映对在源提供程序和构建提供程序列表中显示自定义操作的版本标识符的支持。	2016 年 9 月 22 日
更新了主题	更新了在中管理批准操作 CodePipeline一节，以反映允许审批操作审核者直接从电子邮件通知中打开批准或拒绝修订表单的增强功能。	2016 年 9 月 14 日
新增和更新的主题	新主题介绍了如何查看当前流经软件发布管道的代码更改的详细信息。在查看手动审批操作或排查管道中的故障时，能够快速访问此信息会非常有用。 新部分监控管道提供了与监控管道状态和进度相关的所有主题的中心位置。	2016 年 9 月 08 日

更改	描述	更改日期
新增和更新的主题	新部分在中管理批准操作 CodePipeline提供了有关在管道中配置和使用手动审批操作的信息。该部分中的主题提供有关审批过程的概念信息；设置所需 IAM 权限、创建审批操作及批准或拒绝审批操作的说明；以及在管道中到达审批操作时生成的 JSON 数据的示例。	2016 年 7 月 06 日
新的 区域	CodePipeline 现已在欧洲（爱尔兰）地区推出。更新了中的配额 AWS CodePipeline主题及区域和终端节点主题。	2016 年 6 月 23 日
新主题	添加了新主题重试阶段中失败的操作，以介绍如何重试阶段中某个失败的操作或一组失败的并行操作。	2016 年 6 月 22 日
更新的主题	许多主题，包括身份验证和访问控制在中创建管道 CodePipelineCodePipeline 管道结构参考、以及，都已更新产品和服务与 CodePipeline，以反映对配置管道以将代码与中创建的自定义 Chef 食谱和应用程序一起部署代码的 AWS OpsWorks支持。 CodePipeline 目前 AWS OpsWorks 仅在美国东部（弗吉尼亚北部）区域 (us-east-1) 提供支持。	2016 年 6 月 2 日
新增和更新的主题	增加了新主题教程：创建简单的管道（CodeCommit存储库）。本主题提供了一个演练示例，演示如何使用 CodeCommit 存储库和分支作为管道中源操作的源位置。为了反映与的这种集成，还更新了其他几个主题 CodeCommit，包括身份验证和访问控制产品和服务与 CodePipeline、教程：创建一个四阶段管道、和故障排除 CodePipeline。	2016 年 4 月 18 日
新主题	增加了新主题在中调用管道中的 AWS Lambda 函数 CodePipeline。本主题包含向管道添加 Lambda AWS Lambda 函数的示例函数和步骤。	2016 年 1 月 27 日
更新了主题	向“身份验证和访问控制”添加了一个新部分“基于资源的策略”。	2016 年 1 月 22 日

更改	描述	更改日期
新主题	增加了新主题产品和服务与 CodePipeline。有关与合作伙伴和其他人集成的信息 AWS 服务 已移至本主题。还添加了博客和视频链接。	2015 年 12 月 17 日
更新了主题	与 Solano CI 集成的详细信息已添加到产品和服务与 CodePipeline。	2015 年 11 月 17 日
更新了主题	Jenkins CodePipeline 插件现已通过 Jenkins 插件管理器提供，作为 Jenkins 插件库的一部分。教程：创建一个四阶段管道中安装插件的步骤已更新。	2015 年 11 月 9 日
新的 区域	CodePipeline 现已在美国西部（俄勒冈）地区推出。更新了中的配额 AWS CodePipeline主题。添加了区域和终端节点链接。	2015 年 10 月 22 日
新主题	增加了两个新主题为存储在 Amazon S3 中的项目配置服务器端加密 CodePipeline和在中 CodePipeline 创建使用其他 AWS 账户资源的管道。向“身份验证和访问控制”添加了一个新部分示例 8：在管道中使用与另一个账户相关联的 AWS 资源。	2015 年 8 月 25 日
更新了主题	更新了在中创建和添加自定义操作 CodePipeline主题，以反映结构更改，包括`inputArtifactDetails` 和`outputArtifactDetails` 。	2015 年 8 月 17 日
更新了主题	更新了故障排除 CodePipeline主题，修改了排查服务角色和 Elastic Beanstalk 问题的步骤。	2015 年 8 月 11 日
更新了主题	“身份验证和访问控制” 主题已更新，其中包含对的服务角色的最新更改 CodePipeline。	2015 年 8 月 6 日
新主题	增加了故障排除 CodePipeline主题。在教程：创建一个四阶段管道中增加了有关 IAM 角色和 Jenkins 的更新步骤。	2015 年 7 月 24 日
主题更新	在教程：创建一个简单的管道（S3 存储桶）和教程：创建一个四阶段管道中增加了有关下载示例文件的更新步骤。	2015 年 7 月 22 日

更改	描述	更改日期
主题更新	在教程：创建一个简单的管道（S3 存储桶）中增加了示例文件下载问题的临时解决方法。	2015 年 7 月 17 日
主题更新	在中的配额 AWS CodePipeline中增加了一个链接，指向有关哪些限制可以更改的信息。	2015 年 7 月 15 日
主题更新	更新了“身份验证和访问控制”中的托管策略部分。	2015 年 7 月 10 日
第一个公开发布版	这是《 CodePipeline 用户指南》的首次公开发布。	2015 年 7 月 9 日

AWS 词汇表

有关最新 AWS 术语，请参阅《AWS 词汇表 参考资料》中的AWS 词汇表。

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。