

---

# AWS CodePipeline

用户指南

API 版本 2015-07-09



## AWS CodePipeline: 用户指南

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

## Table of Contents

什么是 AWS CodePipeline ? .....	1
AWS CodePipeline 视频简介 .....	1
AWS CodePipeline 可以用来做什么 ? .....	1
快速了解 AWS CodePipeline .....	1
快速查看输入和输出项目 .....	2
如何开始使用 AWS CodePipeline ? .....	3
我们希望听到您的意见和建议 .....	3
概念 .....	3
持续交付和集成 .....	3
AWS CodePipeline 如何工作 .....	4
入门 .....	8
步骤 1 : 创建 AWS 账户 .....	8
步骤 2 : 创建或使用 IAM 用户 .....	8
步骤 3 : 使用 IAM 托管策略向 IAM 用户分配 AWS CodePipeline 权限 .....	8
步骤 4 : 安装 AWS CLI .....	9
步骤 5 : 打开 AWS CodePipeline 控制台 .....	9
后续步骤 .....	9
产品和服务集成 .....	10
与 AWS CodePipeline 操作类型集成 .....	10
源操作集成 .....	10
生成操作集成 .....	11
测试操作集成 .....	13
部署操作集成 .....	14
审批操作集成 .....	15
调用操作集成 .....	16
与 AWS CodePipeline 的一般集成 .....	16
来自社区的示例 .....	19
博客文章 .....	19
视频 .....	21
教程 .....	22
教程 : 创建一个简单的管道 (Amazon S3 存储桶) .....	22
创建 Amazon S3 存储桶 .....	23
创建 AWS CodeDeploy 资源 .....	24
创建您的第一个管道 .....	26
添加另一个阶段 .....	28
禁用和启用阶段之间的过渡 .....	34
清理资源 .....	35
教程 : 创建一个简单的管道 (AWS CodeCommit 存储库) .....	36
创建 AWS CodeCommit 存储库 .....	36
下载、提交并推送您的代码 .....	37
创建 Amazon EC2 实例并安装 AWS CodeDeploy 代理 .....	38
在 AWS CodeDeploy 中创建应用程序 .....	40
创建您的第一个管道 .....	40
更新 AWS CodeCommit 存储库中的代码 .....	45
可选的阶段管理任务 .....	47
清理资源 .....	47
教程 : 创建一个四阶段管道 .....	47
设置先决条件 .....	47
创建管道 .....	50
添加更多阶段 .....	51
清理资源 .....	53
教程 : 设置 CloudWatch Events 规则以接收管道状态更改的电子邮件通知 .....	54
使用 Amazon SNS 设置电子邮件通知 .....	54
为 AWS CodePipeline 创建 CloudWatch Events 通知规则 .....	55

清理资源 .....	57
最佳实践和用户案例 .....	58
最佳实践 .....	58
AWS CodePipeline 资源的安全最佳实践 .....	58
AWS CodePipeline 资源的监控和日志记录最佳实践 .....	58
Jenkins 插件的最佳实践 .....	59
如何使用 AWS CodePipeline 的示例 .....	59
将 AWS CodePipeline 与 Amazon S3、AWS CodeCommit 和 AWS CodeDeploy 一起使用 .....	59
将 AWS CodePipeline 与第三方行动提供程序 (GitHub 和 Jenkins) 一起使用 .....	60
将 AWS CodePipeline 与 AWS CodeStar 一起使用在代码项目中生成管道 .....	60
使用 AWS CodePipeline 通过 AWS CodeBuild 编译、生成和测试代码 .....	60
将 AWS CodePipeline 与 Amazon ECS 一起使用以便将基于容器的应用程序连续交付到云 .....	60
将 AWS CodePipeline 与 Elastic Beanstalk 一起使用以便将 Web 应用程序连续交付到云 .....	61
将 AWS CodePipeline 与 AWS Lambda 一起使用可连续交付基于 Lambda 的应用程序和无服务器应用程序 .....	61
将 AWS CodePipeline 与 AWS CloudFormation 模板一起使用以便连续交付到云 .....	61
使用管道 .....	62
在 AWS CodePipeline 中启动管道执行 .....	62
用于自动启动管道的更改检测方法 .....	63
使用 CloudWatch Events 规则自动启动 AWS CodeCommit 管道 .....	64
使用 CloudWatch Events 规则自动启动 Amazon S3 管道 .....	69
使用 Webhook 自动启动 GitHub 管道 .....	75
使用定期检查自动启动管道 .....	79
在 AWS CodePipeline 中手动启动管道 .....	80
使用 Amazon CloudWatch Events 按计划自动启动管道 .....	80
创建管道 .....	82
创建映像定义文件以部署基于容器的应用程序 .....	83
创建管道 (控制台) .....	84
创建管道 (CLI) .....	88
编辑管道 .....	90
编辑管道 (控制台) .....	91
编辑管道 (AWS CLI) .....	93
查看管道详细信息和历史记录 .....	96
查看管道详细信息和历史记录 (控制台) .....	96
查看管道详细信息和历史记录 (CLI) .....	98
删除管道 .....	101
删除管道 (控制台) .....	101
删除管道 (CLI) .....	101
创建使用另一个账户的资源的管道 .....	102
先决条件：创建 AWS KMS 加密密钥 .....	103
步骤 1：设置账户策略和角色 .....	103
步骤 2：编辑管道 .....	108
使用操作 .....	111
为管道创建自定义操作 .....	111
创建自定义操作 (CLI) .....	112
为您的自定义操作创建作业辅助角色 .....	115
向管道添加自定义操作 .....	118
在管道中调用 Lambda 函数 .....	120
步骤 1：创建管道 .....	121
步骤 2：创建 Lambda 函数 .....	121
步骤 3：在 AWS CodePipeline 控制台中向管道添加 Lambda 函数 .....	124
步骤 4：使用 Lambda 函数测试管道 .....	126
步骤 5：后续步骤 .....	126
示例 JSON 事件 .....	128
其他示例函数 .....	128
重试失败的操作 .....	136
重试失败的操作 (控制台) .....	137

重试失败的操作 (CLI) .....	137
管理管道中的审批操作 .....	139
手动审批操作的配置选项 .....	139
审批操作的设置和工作流程概述 .....	139
在 AWS CodePipeline 中向 IAM 用户授予审批权限 .....	140
向服务角色授予 Amazon SNS 权限 .....	142
添加手动审批操作 .....	142
批准或拒绝审批操作 .....	146
手动审批通知的 JSON 数据格式 .....	148
使用阶段过渡 .....	149
禁用或启用过渡 (控制台) .....	149
禁用或启用过渡 (CLI) .....	151
监控管道 .....	153
使用 CloudWatch Events 检测和响应管道状态更改 .....	153
了解管道执行状态更改规则的工作方式 .....	154
使用 AWS CloudTrail 记录 API 调用 .....	160
CloudTrail 中的 AWS CodePipeline 信息 .....	160
了解 AWS CodePipeline 日志文件条目 .....	160
查看管道中的当前源修订详细信息 .....	162
查看管道中的当前源修订详细信息 (控制台) .....	162
查看管道中的当前源修订详细信息 (CLI) .....	163
故障排除 .....	165
管道错误：使用 AWS Elastic Beanstalk 配置的管道返回错误消息：“Deployment failed. The provided role does not have sufficient permissions: Service:AmazonElasticLoadBalancing” .....	165
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 .....	166
管道错误：源操作返回权限不足消息：“Could not access the AWS CodeCommit repository repository-name. Make sure that the pipeline IAM role has sufficient permissions to access the repository.” .....	166
管道错误：Jenkins 生成或测试操作运行很长时间，然后由于缺少凭证或权限而失败 .....	167
管道错误：我的 GitHub 源阶段包含 Git 子模块，但 AWS CodePipeline 不对它们进行初始化 .....	167
管道错误：我收到一个管道错误，其中包含以下消息：“PermissionError: Could not access the GitHub repository” .....	167
管道错误：在一个 AWS 区域中使用在另一个 AWS 区域中创建的存储桶创建的管道返回“InternalError”，代码为“JobFailed” .....	168
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 404 Not Found 错误 .....	166
在 ZIP 不保留外部属性时，在 GitHub 的源文件上不保留文件权限 .....	169
需要有关其他问题的帮助？ .....	170
身份验证、访问控制和安全配置 .....	171
身份验证 .....	171
使用 IAM 策略控制访问 .....	172
访问管理概述 .....	172
使用基于身份的策略 (IAM 策略) .....	179
使用 AWS CodePipeline 控制台所需的权限 .....	180
适用于 AWS CodePipeline 的 AWS 托管 (预定义) 策略 .....	180
管理 AWS CodePipeline 服务角色 .....	180
客户托管策略示例 .....	185
AWS CodePipeline 权限参考 .....	192
安全配置 .....	195
为适用于 AWS CodePipeline 的 Amazon S3 中存储的项目配置服务器端加密 .....	195
配置 GitHub 身份验证 .....	197
使用 Parameter Store 跟踪数据库密码或第三方 API 密钥 .....	200
命令行参考 .....	201
管道结构参考 .....	202
AWS CodePipeline 中的管道和阶段结构要求 .....	202
AWS CodePipeline 中的操作结构要求 .....	203

限制 .....	208
文档历史记录 .....	212
AWS 词汇表 .....	218

# 什么是 AWS CodePipeline ?

AWS CodePipeline 是一种持续交付服务，可用于建模、可视化和自动执行发布软件所需的步骤。您可以快速对软件发布过程的不同阶段进行建模和配置。AWS CodePipeline 自动执行持续发布软件更改所需的步骤。有关 AWS CodePipeline 定价的信息，请参阅[定价](#)。

## 主题

- [AWS CodePipeline 视频简介 \(p. 1\)](#)
- [AWS CodePipeline 可以用来做什么？ \(p. 1\)](#)
- [快速了解 AWS CodePipeline \(p. 1\)](#)
- [如何开始使用 AWS CodePipeline？ \(p. 3\)](#)
- [我们希望听到您的意见和建议 \(p. 3\)](#)
- [AWS CodePipeline 概念 \(p. 3\)](#)

## AWS CodePipeline 视频简介

这个简短的视频 (3:06) 描述了 AWS CodePipeline 如何根据您定义的发布过程模型，在每次代码发生变化时生成、测试和部署代码。

[AWS CodePipeline 视频简介](#)。

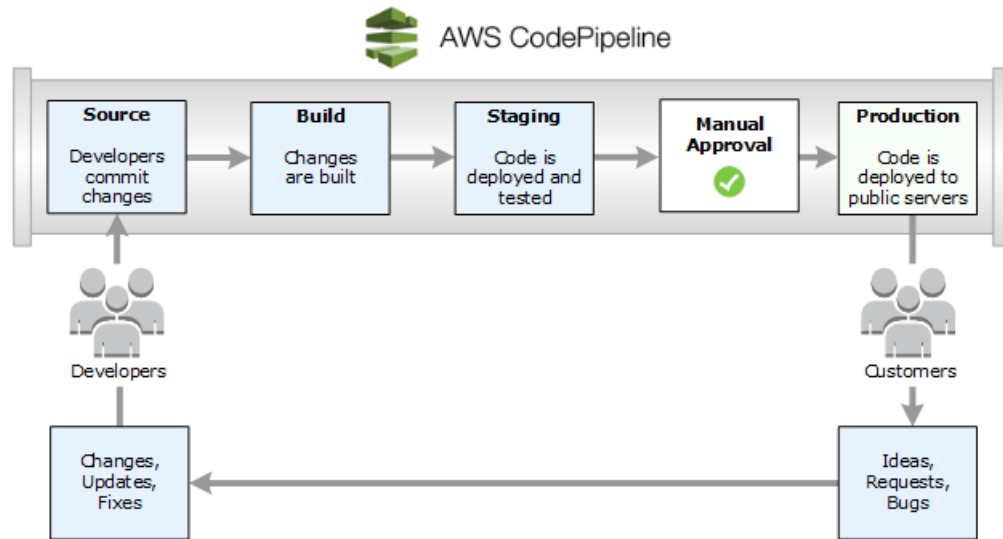
## AWS CodePipeline 可以用来做什么？

您可以使用 AWS CodePipeline 帮助您自动生成、测试并在云中部署应用程序。具体来说，您可以：

- 实现发布过程自动化：从源存储库开始，一直到生成、测试和部署，AWS CodePipeline 可端到端地完全自动执行您的发布过程。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自动发布。
- 建立一致的发布过程：为每个代码更改定义一系列一致的步骤。AWS 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 AWS CodePipeline 目前支持的 AWS 服务和第三方工具的完整列表，请参阅[AWS CodePipeline 产品和服务集成 \(p. 10\)](#)。
- 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何管道。
- 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。

## 快速了解 AWS CodePipeline

下图显示了使用 AWS CodePipeline 的示例发布过程。



在此示例中，当开发人员将更改提交到源存储库时，AWS 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。AWS CodePipeline 会从暂存服务器运行其他测试，例如集成或负载测试。在成功完成这些测试，并且添加到管道中的手动审批操作获得批准后，AWS CodePipeline 将经过测试并获得批准的代码部署到生产实例中。

AWS CodePipeline 可以通过使用 AWS CodeDeploy、AWS Elastic Beanstalk 或 AWS OpsWorks Stacks 将应用程序部署到 Amazon EC2 实例。AWS 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 AWS CodePipeline 提供的集成点来插入其他工具或服务，包括生成服务、测试提供商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。

## 快速查看输入和输出项目

AWS CodePipeline 与开发工具集成在一起以自动检查代码更改，然后通过持续交付过程的所有阶段生成并部署。

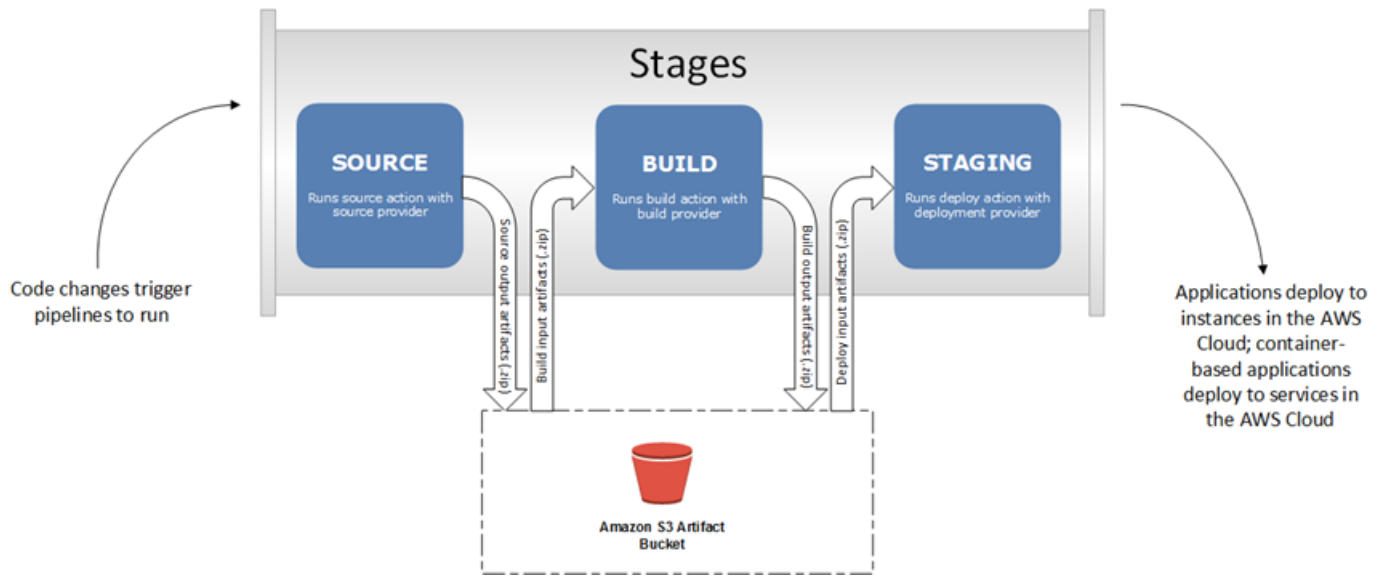
阶段使用在创建管道时选择的 Amazon S3 项目存储桶中存储的输入和输出项目。根据阶段中的操作类型，AWS CodePipeline 压缩并传输输入或输出项目的文件。例如，在开始执行生成操作时，AWS CodePipeline 检索输入项目（要生成的任何文件），并为生成操作提供该项目。在该操作完成后，AWS CodePipeline 使用输出项目（生成的应用程序），并将其保存到输出项目存储桶以在下一阶段中使用。

在使用创建管道向导配置或选择阶段时：

1. 在提交到源存储库时，AWS CodePipeline 自动触发以运行管道，从而从源阶段提供输出项目。
2. 上一步骤中的输出项目将作为生成阶段的输入项目。生成阶段的输出项目可能是为容器生成的更新的应用程序或更新的 Docker 映像。
3. 上一步骤中的输出项目将作为部署阶段的输入项目，例如，AWS 云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。

下图显示了 AWS CodePipeline 中的阶段之间的简要项目工作流程。





## 如何开始使用 AWS CodePipeline ?

开始使用 AWS CodePipeline :

1. 阅读[AWS CodePipeline 概念 \(p. 3\)](#)部分，了解 AWS CodePipeline 如何工作。
2. 按照[AWS CodePipeline 入门 \(p. 8\)](#)中的步骤操作，准备使用 AWS CodePipeline。
3. 按照[AWS CodePipeline 教程 \(p. 22\)](#)教程中的步骤操作，试用 AWS CodePipeline。
4. 按照在 [AWS CodePipeline 中创建管道 \(p. 82\)](#)中的步骤操作，对您的新项目或现有项目使用 AWS CodePipeline。

## 我们希望听到您的意见和建议

我们欢迎您提供反馈。要与我们联系，请访问 [AWS CodePipeline 论坛](#)。

## AWS CodePipeline 概念

如果您了解 AWS CodePipeline 中使用的概念和术语以及发布自动化的一些基本概念，您会发现建模和配置自动化发布过程更容易一些。以下是使用 AWS CodePipeline 时要了解的一些概念。

主题

- [持续交付以及与 AWS CodePipeline 集成 \(p. 3\)](#)
- [AWS CodePipeline 如何工作 \(p. 4\)](#)

## 持续交付以及与 AWS CodePipeline 集成

AWS CodePipeline 是一种持续交付服务，可自动生成和测试您的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。

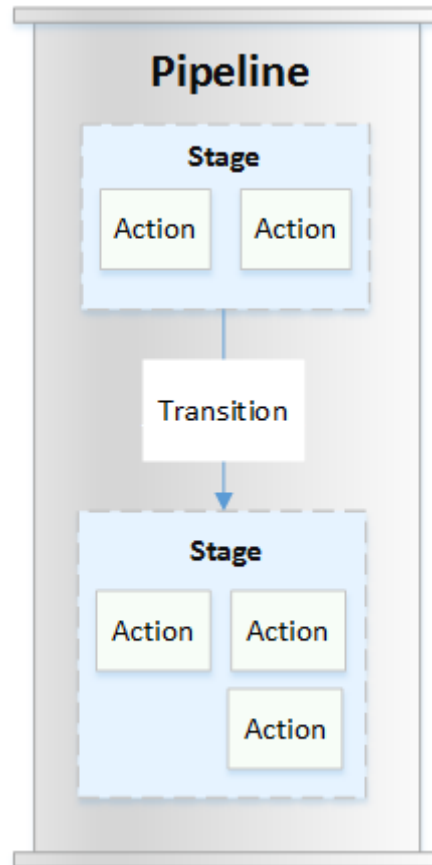
**持续集成**是一种软件开发实践，其中团队成员使用版本控制系统，并将其工作频繁地集成到同一位置，如主分支。每项更改都经过生成和验证，以尽可能快地检测到集成错误。持续交付会自动执行整个软件发布过程，一直到最后的生产部署，而持续集成重点关注自动生成和测试代码。

有关更多信息，请参阅在 [AWS 上实现持续集成和持续交付：使用 DevOps 加速软件交付](#)。

## AWS CodePipeline 如何工作

AWS CodePipeline 可帮助您使用管道创建和管理您的发布工作流程。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。您可以在 AWS 和 AWS CodePipeline 的限制内根据需要创建任意数量的管道，如[AWS CodePipeline 中的限制 \(p. 208\)](#)中所述。

下图和随附的说明介绍了 AWS CodePipeline 特有的术语以及这些概念之间的关系：



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

当您使用控制台创建第一个管道时，AWS CodePipeline 将在管道所在的同一区域中创建一个 Amazon S3 存储桶，以存储该区域中与该账户相关联的所有管道的项目。每次使用控制台在该区域中创建另一个管道时，AWS CodePipeline 都将在存储桶中为该管道创建一个文件夹。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。该存储桶被命名为 `codepipeline-region-1234567EXAMPLE`，其中 *region* 是您创建管道的 AWS 区域，`1234567EXAMPLE` 是一个十位随机数，可确保存储桶名称是唯一的。

如果使用 AWS CLI 创建管道，则可以将该管道的项目存储在任意 Amazon S3 存储桶中，只要该存储桶与管道位于同一 AWS 区域即可。如果您担心超出您的账户允许的 Amazon S3 存储桶限制，则可以这么做。

- 修订是对在 AWS CodePipeline 的源操作中配置的源进行的更改，例如将提交内容推送到 GitHub 存储库或 AWS CodeCommit 存储库，或更新受版本控制的 Amazon S3 存储桶中的文件。每个修订都通过管道

单独运行。可以在同一管道中处理多个修订，但每个阶段一次只能处理一个修订。只要在管道的源阶段中指定的位置进行更改，修订就会通过管道运行。

#### 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。

- AWS CodePipeline 将您的工作流程分解为一系列阶段。例如，可能有一个生成阶段，用于生成代码和运行测试。还有部署阶段，用于将代码更新部署到运行时环境。您可以在同一部署阶段配置部署到不同环境的多个并行部署。您可以在发布过程中标记每个阶段，以便更好地跟踪、控制和报告 (例如“Source”、“Build”和“Staging”)。

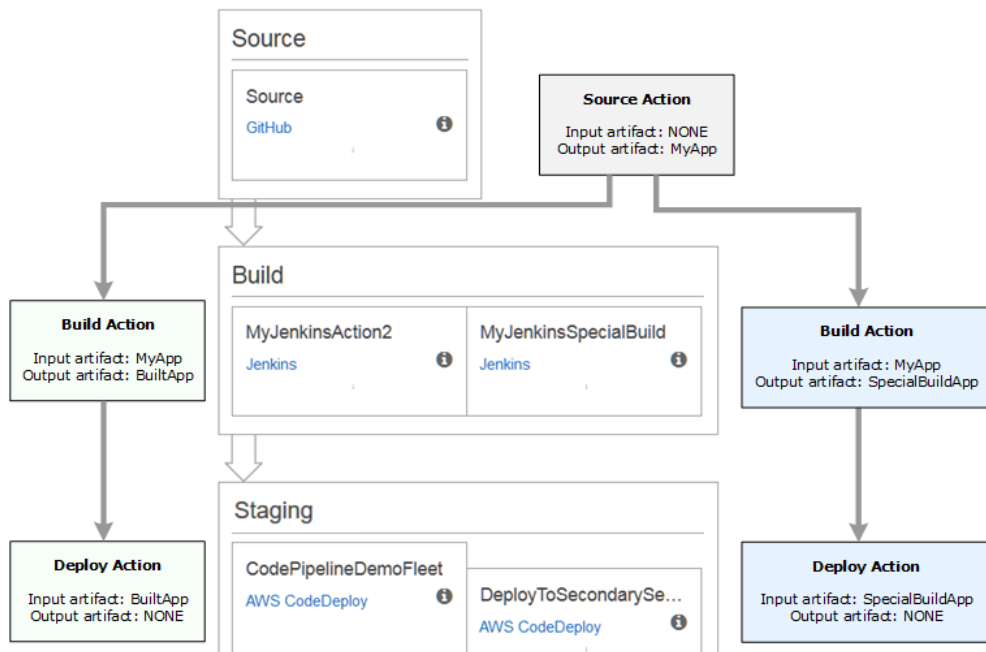
管道中的每个阶段都有一个唯一的名称，并且包含一系列操作作为其工作流程的一部分。一个阶段一次只能处理一个修订。修订必须先经过一个阶段，然后下一个修订才能经过该阶段。在阶段被视为完成之前，为该阶段配置的所有操作必须都已成功完成。在阶段完成之后，管道会将该阶段中的操作创建的修订及其项目过渡到管道中的下一阶段。您可以手动禁用和启用这些过渡。有关阶段要求和结构的更多信息，请参阅 [AWS CodePipeline 中的管道和阶段结构要求 \(p. 202\)](#)。

- 每个阶段至少包含一个操作，它是对项目执行的某种任务。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。例如，部署阶段可能包含部署操作，该操作将代码部署到一个或多个暂存服务器。一开始您可以为阶段配置单个操作，以后再根据需要向该阶段添加操作。有关更多信息，请参阅 [在 AWS CodePipeline 中编辑管道 \(p. 90\)](#) 和 [AWS CodePipeline 中的操作结构要求 \(p. 203\)](#)。

在修订开始通过管道运行后，AWS CodePipeline 会将管道中的操作和阶段将要处理的文件或更改复制到 Amazon S3 存储桶。这些对象被称为项目，可能是操作的源 (input artifacts) 或操作的输出 (output artifacts)。项目可以由多个操作处理。

- 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。
  - 输出项目，它是操作的输出。

管道中的每个输出项目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项目匹配，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。下图演示了如何在管道中生成和使用输入项目和输出项目：



- 过渡是修订在管道中从工作流程的一个阶段继续执行下一阶段的操作。在 AWS CodePipeline 控制台中，过渡箭头将各个阶段连接在一起以显示阶段的执行顺序。当阶段完成时，默认情况下，修订将过渡到管道的下一阶段。您可以禁用从一个阶段过渡到下一个阶段。这样做时，您的管道将运行阶段中该过渡之前的所有操作，但不会运行任何其他阶段或该阶段之后的操作，直到您启用该过渡。这是防止更改在整个管道中运行的一种简单方法。启用过渡后，已成功运行完之前阶段的最新修订将在该过渡之后的阶段中运行。如果启用所有过渡，则管道将连续运行。每个修订都作为成功的管道运行 (持续部署) 的一部分部署。
- 由于一次只能有一个修订在阶段中运行，因此 AWS CodePipeline 会分批处理已完成上一阶段的任何修订，直到下一阶段可用。如果更新的修订完成该阶段，则分批修订将被最新修订取代。
- 审批操作可防止管道过渡到下一个操作，直到授予权限 (例如，收到授权的 IAM 用户的手动批准)。例如，如果您希望管道只在成功的代码审查后才能继续，或者想要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。在这种情况下，您可以向“生产”前面的阶段添加手动审批操作，并在准备好向公众发布更改时亲自批准它。
- 失败是阶段中一个未成功完成的操作。如果一个操作在某个阶段失败，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。如果失败，则管道中不会再生针对该修订的过渡。AWS CodePipeline 暂停管道，直到出现以下情况之一：
  - 您手动重试包含失败操作的阶段。
  - 您再次为该修订启动管道。
  - 在源阶段操作中创建了另一个修订。
- 在源位置 (在管道的源操作中定义) 中进行更改或手动启动管道时，将自动启动管道。您也可以 Amazon CloudWatch 中设置一个规则，以便在发生指定的事件时自动启动管道。在管道开始之后，修订将经过管道中的每个阶段和每个操作。您可以在管道视图页面上查看管道中每个操作的上一次运行的详细信息。

#### Note

如果在控制台中创建或编辑具有 AWS CodeCommit 源存储库的管道，则 AWS CodePipeline 使用 Amazon CloudWatch Events 检测存储库中的更改，并在发生更改时启动管道。

下图显示了 AWS CodePipeline 控制台中的示例管道的两个阶段。它包括每个阶段的操作以及两个阶段之间已启用的过渡。

## AnyCompanyPipeline [View pipeline history](#)

View progress and manage your pipeline.

Edit

Release change



# AWS CodePipeline 入门

在首次使用 AWS CodePipeline 之前，必须完成以下步骤。

## 主题

- [步骤 1：创建 AWS 账户 \(p. 8\)](#)
- [步骤 2：创建或使用 IAM 用户 \(p. 8\)](#)
- [步骤 3：使用 IAM 托管策略向 IAM 用户分配 AWS CodePipeline 权限 \(p. 8\)](#)
- [步骤 4：安装 AWS CLI \(p. 9\)](#)
- [步骤 5：打开 AWS CodePipeline 控制台 \(p. 9\)](#)
- [后续步骤 \(p. 9\)](#)

## 步骤 1：创建 AWS 账户

如果您还没有创建 AWS 账号，请转到 <https://aws.amazon.com/> 并选择 Sign Up，创建一个账号。

## 步骤 2：创建或使用 IAM 用户

创建 IAM 用户或使用您的 AWS 账户中现有的 IAM 用户。确保您具有与该 IAM 用户关联的 AWS 访问密钥 ID 和 AWS 秘密访问密钥。有关更多信息，请参阅[在您的 AWS 账户中创建 IAM 用户](#)。

## 步骤 3：使用 IAM 托管策略向 IAM 用户分配 AWS CodePipeline 权限

您必须向 IAM 用户授予与 AWS CodePipeline 交互的权限。执行此操作的最快方法是向 IAM 用户应用 `AWSCodePipelineFullAccess` 托管策略。

要使用 AWS 管理控制台 向 IAM 用户授予权限

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
2. 在 IAM 控制台的导航窗格中，选择 Policies，然后从策略列表中选择 `AWSCodePipelineFullAccess` 托管策略。
3. 在 Policy Details 页面上，选择 Attached Entities 选项卡，然后选择 Attach。
4. 在 Attach Policy 页面上，选中 IAM 用户或组旁边的复选框，然后选择 Attach Policy。

### Note

`AWSCodePipelineFullAccess` 策略提供对 IAM 用户有权访问的所有 AWS CodePipeline 操作和资源以及在管道中创建阶段中时所有可能的操作（比如创建包含 AWS CodeDeploy、Elastic Beanstalk 或 Amazon S3 的阶段）的访问。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。有关如何仅允许 IAM 用户使用一组有限的 AWS CodePipeline 操作和资源的信息，请参阅 [删除未使用的 AWS 服务的权限 \(p. 184\)](#)。

## 步骤 4：安装 AWS CLI

### 安装和配置 AWS CLI

1. 在本地计算机上，下载并安装 AWS CLI。这将允许您从命令行与 AWS CodePipeline 交互。有关更多信息，请参阅[使用 AWS 命令行界面进行设置](#)。

#### Note

AWS CodePipeline 只能配合 AWS CLI versions 1.7.38 及更高版本使用。要确定您安装的 AWS CLI 版本，请运行 `aws --version` 命令。要将旧版本的 AWS CLI 升级到最新版本，请按照[卸载 AWS CLI](#) 中的说明操作，然后按照[安装 AWS Command Line Interface](#) 中的说明操作。

2. 使用 `configure` 命令配置 AWS CLI，如下所述：

```
aws configure
```

出现提示时，指定将用于 AWS CodePipeline 的 IAM 用户的 AWS 访问密钥和 AWS 私有访问密钥。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、访问密钥和私有密钥的更多信息，请参阅[管理 IAM 用户的访问密钥和如何获得证书？](#)。

有关 AWS CodePipeline 可用的区域和终端节点的更多信息，请参阅[区域和终端节点](#)。

## 步骤 5：打开 AWS CodePipeline 控制台

- 登录 AWS 管理控制台并通过以下网址打开 AWS CodePipeline 控制台：<http://console.aws.amazon.com/codepipeline>。

## 后续步骤

您已满足先决条件。您可以开始使用 AWS CodePipeline。要开始使用 AWS CodePipeline，请参阅[AWS CodePipeline 教程 \(p. 22\)](#)。

# AWS CodePipeline 产品和服务集成

默认情况下，AWS CodePipeline 与许多 AWS 服务以及合作伙伴的产品和服务集成。使用以下部分中的信息来帮助您将 AWS CodePipeline 配置为与您使用的产品和服务集成。

## 主题

- [与 AWS CodePipeline 操作类型集成 \(p. 10\)](#)
- [与 AWS CodePipeline 的一般集成 \(p. 16\)](#)
- [来自社区的示例 \(p. 19\)](#)

## 与 AWS CodePipeline 操作类型集成

本主题中的集成信息按 AWS CodePipeline 操作类型组织。

## 主题

- [源操作集成 \(p. 10\)](#)
- [生成操作集成 \(p. 11\)](#)
- [测试操作集成 \(p. 13\)](#)
- [部署操作集成 \(p. 14\)](#)
- [审批操作集成 \(p. 15\)](#)
- [调用操作集成 \(p. 16\)](#)

## 源操作集成

以下信息按 AWS CodePipeline 操作类型组织，可帮助您配置 AWS CodePipeline 以便与您使用的产品和服务集成。

Amazon Simple Storage Service (Amazon S3)	<p><a href="#">Amazon S3</a> 是一项面向 Internet 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。您可以将 AWS CodePipeline 配置为使用受版本控制的 Amazon S3 存储桶作为代码的源阶段。您必须先创建存储桶，再对其启用版本控制，然后才能创建一个使用该存储桶作为阶段源操作的一部分的管道。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">步骤 1：为您的应用程序创建一个 Amazon S3 存储桶 (p. 23)</a></li><li>• <a href="#">创建管道 (CLI) (p. 88)</a></li><li>• AWS CodePipeline 使用 Amazon CloudWatch Events 检测 Amazon S3 源存储桶中的更改。每个源操作具有相应的事件规则。在源中发生更改时，该事件规则将自动启动您的管道。请参阅 <a href="#">与 AWS CodePipeline 的一般集成 (p. 16)</a>。</li></ul>
AWS CodeCommit	<p><a href="#">AWS CodeCommit</a> 是由 AWS 托管的版本控制服务，可让您在云中私下存储和管理资产 (如文档、源代码和二进制文件)。您可以将 AWS CodePipeline 配置为使用 AWS CodeCommit 存储库中的分支作为代码的源阶段。您必须先创建存储库并将其与本地计算机上的工作目录相关联，然后才能创建使用该分支</p>



	<p>作为阶段源操作的一部分的管道。您可以通过创建新管道或编辑现有管道来连接到 AWS CodeCommit 存储库。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">教程：创建一个简单的管道 (AWS CodeCommit 存储库) (p. 36)</a></li><li>• <a href="#">AWS for DevOps 入门指南</a> — 了解如何使用 AWS CodePipeline 将 AWS CodeCommit 存储库中的源代码持续交付和部署到 AWS CodeDeploy、Elastic Beanstalk 和 AWS OpsWorks 中的部署目标。</li><li>• AWS CodePipeline 使用 Amazon CloudWatch Events 检测用作管道源的 AWS CodeCommit 存储库中的更改。每个源操作具有相应的事件规则。在存储库中发生更改时，该事件规则将自动启动管道。请参阅 <a href="#">与 AWS CodePipeline 的一般集成 (p. 16)</a>。</li></ul>
GitHub	<p>您可以将 AWS CodePipeline 配置为使用 <a href="#">GitHub</a> 存储库作为代码的源阶段。您先前必须已经创建了一个 GitHub 账户和至少一个 GitHub 存储库。您可以通过创建新管道或编辑现有管道来连接到 GitHub 存储库。</p> <p><b>Note</b></p> <p>不支持 AWS CodePipeline 与 GitHub Enterprise 集成。</p> <p>第一次将 GitHub 存储库添加到管道时，您需要授权 AWS CodePipeline 访问您的存储库。要与 GitHub 集成，AWS CodePipeline 会为您的管道创建 OAuth 应用程序；如果在控制台中创建或更新了管道，AWS CodePipeline 将创建 GitHub Webhook，以便当存储库中发生更改时启动您的管道。令牌和 Webhook 需要以下 GitHub 范围：</p> <ul style="list-style-type: none"><li>• <code>repo</code> 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。</li><li>• <code>admin:repo_hook</code> 范围，用于完全控制存储库挂钩。</li></ul> <p>有关 GitHub 范围的更多信息，请参阅 <a href="#">GitHub 开发人员 API 参考</a>。</p> <p>将为该 GitHub 账户可以访问的所有存储库配置 AWS CodePipeline 访问权限；目前无法为单个存储库配置该访问权限。您可以通过选择 Settings，选择 Applications，然后在 Authorized applications 下，从授权应用程序列表中选择 AWS CodePipeline，然后选择 Revoke，撤销 GitHub 的此访问权限。撤销访问权限将立即阻止 AWS CodePipeline 访问以前配置为使用该 GitHub 账户访问的任何 GitHub 存储库。</p> <p>如果要限制 AWS CodePipeline 对一组特定存储库的访问，请创建一个 GitHub 账户，仅授予该账户对要与 AWS CodePipeline 集成的存储库的访问权限，然后在配置 AWS CodePipeline 时使用该账户将 GitHub 存储库用于管道中的源阶段。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">教程：创建一个四阶段管道 (p. 47)</a></li><li>• <a href="#">使用 Webhook 自动启动 GitHub 管道 (p. 75)</a></li></ul>

## 生成操作集成

AWS CodeBuild	<p><a href="#">AWS CodeBuild</a> 是一项在云中完全托管的生成服务。AWS CodeBuild 可编译源代码，运行单元测试，并生成可供部署的项目。</p>
---------------	--

	<p>您可以将 AWS CodeBuild 作为生成操作添加到管道的生成阶段中。您可以使用现有的生成项目或在 AWS 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。</p> <p><b>Note</b></p> <p>AWS CodeBuild 也可以作为测试操作包含在管道中，有或没有生成输出均可。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">什么是 AWS CodeBuild？</a></li><li>• <a href="#">将 AWS CodeBuild 生成操作添加到管道中 (在将 AWS CodePipeline 与 AWS CodeBuild 结合使用以测试代码和运行生成中)</a></li><li>• <a href="#">使用 AWS CodeBuild 中的生成项目</a></li><li>• <a href="#">AWS CodeBuild – 完全托管的生成服务</a></li></ul>
CloudBees	<p>您可以将 AWS CodePipeline 配置为使用 <a href="#">CloudBees</a> 来生成或测试管道中一个或多个操作的代码。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">有关将 CloudBees Jenkins 平台与 AWS CodePipeline 集成以运行您的生成和测试作业的 CloudBees 文档</a></li></ul>
Jenkins	<p>您可以将 AWS CodePipeline 配置为使用 <a href="#">Jenkins CI</a> 来生成或测试管道中一个或多个操作的代码。您先前必须已经创建了 Jenkins 项目并为该项目安装和配置了 AWS CodePipeline Plugin for Jenkins。您可以通过创建新管道或编辑现有管道来连接到 Jenkins 项目。</p> <p>对 Jenkins 的访问权限基于每个项目配置。您必须在要用于 AWS CodePipeline 的每个 Jenkins 实例上安装 AWS CodePipeline Plugin for Jenkins。此外，您还必须配置 AWS CodePipeline 对 Jenkins 项目的访问权限。您应该通过将 Jenkins 项目配置为仅接受 HTTPS/SSL 连接来保护 Jenkins 项目。如果您的 Jenkins 项目安装在 Amazon EC2 实例上，请考虑通过在每个实例上安装 AWS CLI，并在具有要用于 AWS CodePipeline 和 Jenkins 之间的连接的 IAM 用户配置文件和 AWS 凭证的这些实例上配置 AWS 配置文件，来提供您的 AWS 凭证，而不是通过 Jenkins Web 界面添加或存储它们。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">访问 Jenkins</a></li><li>• <a href="#">教程：创建一个四阶段管道 (p. 47)</a></li></ul>
Solano CI	<p>您可以将 AWS CodePipeline 配置为使用 <a href="#">Solano Labs</a> 来生成和测试管道中一个或多个操作的代码。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">有关将 Solano CI 与 AWS CodePipeline 一起使用的 Solano Labs 文档</a></li></ul>
TeamCity	<p>您可以将 AWS CodePipeline 配置为使用 <a href="#">TeamCity</a> 来生成和测试管道中一个或多个操作的代码。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">适用于 AWS CodePipeline 的 TeamCity 插件</a></li><li>• <a href="#">在 AWS 和 TeamCity 中构建端到端持续交付和部署管道</a></li></ul>

## 测试操作集成

AWS CodeBuild	<p><a href="#">AWS CodeBuild</a> 是一项在云中完全托管的生成服务。AWS CodeBuild 可编译源代码，运行单元测试，并生成可供部署的项目。</p> <p>您可以将 AWS CodeBuild 作为测试操作添加到管道中，以便对您的代码运行单元测试，有或没有生成输出项目均可。如果为测试操作生成输出项目，则可以将其部署为管道的一部分。您可以使用现有的生成项目或在 AWS CodePipeline 控制台中创建一个。</p> <p><b>Note</b></p> <p>AWS CodeBuild 也可以作为生成操作包含在管道中，并具有强制性的生成输出项目。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">什么是 AWS CodeBuild ?</a></li><li>• <a href="#">将 AWS CodeBuild 测试操作添加到管道中 (在将 AWS CodePipeline 与 AWS CodeBuild 结合使用以测试代码和运行生成中)</a></li></ul>
BlazeMeter	<p>您可以将 AWS CodePipeline 配置为使用 <a href="#">BlazeMeter</a> 来测试管道中一个或多个操作的代码。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">有关使用 AWS CodePipeline 进行测试的 BlazeMeter 文档</a></li></ul>
Ghost Inspector	<p>您可以将 AWS CodePipeline 配置为使用 <a href="#">Ghost Inspector</a> 来测试管道中一个或多个操作的代码。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">有关与 AWS CodePipeline 的服务集成的 Ghost Inspector 文档</a></li></ul>
HPE StormRunner Load	<p>您可以将 AWS CodePipeline 配置为使用 <a href="#">HPE StormRunner Load</a> 来测试管道中一个或多个操作的代码。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">有关与 AWS CodePipeline 集成的 HPE StormRunner Load 文档</a></li></ul>
Nouvola	<p>您可以将 AWS CodePipeline 配置为使用 <a href="#">Nouvola</a> 来测试管道中一个或多个操作的代码。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">适用于 AWS CodePipeline 的 Nouvola 插件</a></li></ul>
Runscope	<p>您可以将 AWS CodePipeline 配置为使用 <a href="#">Runscope</a> 来测试管道中一个或多个操作的代码。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">有关与 AWS CodePipeline 集成的 Runscope 文档</a></li></ul>

## 部署操作集成

AWS CloudFormation	<p><a href="#">AWS CloudFormation</a> 让开发人员和系统管理员可轻松创建和管理相关 AWS 资源集合，并使用模板来预配置和更新这些资源。您可以使用 AWS CloudFormation 的示例模板或自己创建的模板来描述 AWS 资源以及应用程序运行时所需的任何相关依赖项或运行时参数。</p> <p>AWS 无服务器应用程序模型 (AWS SAM) 可扩展 AWS CloudFormation，以提供一种简化的方法来定义和部署无服务器应用程序。AWS SAM 支持 Amazon API Gateway API、AWS Lambda 函数和 Amazon DynamoDB 表。您可以将 AWS CodePipeline 与 AWS CloudFormation 和 AWS 无服务器应用程序模型结合使用来持续提供无服务器应用程序。</p> <p>您可以向管道中添加使用 AWS CloudFormation 作为部署提供商的操作。AWS CloudFormation 作为部署提供商的独特角色使您能够在管道执行过程中对 AWS CloudFormation 堆栈和更改集采取行动。在管道运行时，AWS CloudFormation 可以创建、更新、替换和删除堆栈和更改集。因此，根据您在 AWS CloudFormation 模板和参数定义中提供的规范，可以在管道执行期间自动创建、预配置、更新或终止 AWS 和自定义资源。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">利用 AWS CodePipeline 进行持续交付</a> — 了解如何使用 AWS CodePipeline 为 AWS CloudFormation 构建持续交付工作流程。</li><li>• <a href="#">基于 Lambda 应用程序的自动化部署</a> – 了解如何使用 AWS 无服务器应用程序模型和 AWS CloudFormation 为基于 Lambda 的应用程序构建持续交付工作流程。</li></ul>
AWS CodeDeploy	<p><a href="#">AWS CodeDeploy</a> 协调将应用程序部署到 Amazon EC2 实例和/或本地实例的过程。您可以将 AWS CodePipeline 配置为使用 AWS CodeDeploy 来部署您的代码。在创建管道之前或使用 Create Pipeline 向导时，您可以创建 AWS CodeDeploy 应用程序、部署和部署组以在阶段的部署操作中使用。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">步骤 2：创建 AWS CodeDeploy 资源以部署示例应用程序 (p. 24)</a></li><li>• <a href="#">使用管道初学者工具包探究 AWS 中的持续交付</a></li></ul>
Amazon Elastic Container Service	<p>Amazon ECS 是一种可高度扩展的高性能容器管理服务，可用于在 AWS 云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 以作为部署提供程序。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 中。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">什么是 Amazon ECS？</a></li><li>• <a href="#">教程：使用 AWS CodePipeline 进行持续部署</a></li><li>• <a href="#">在 AWS CodePipeline 中创建管道 (p. 82)</a></li></ul>
AWS Elastic Beanstalk	<p><a href="#">Elastic Beanstalk</a> 是一项易于使用的服务，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。您可以将 AWS CodePipeline 配置为使用 Elastic Beanstalk 来部署您的代码。在创建管道之前或使用 Create Pipeline 向导时，您可以创建 Elastic Beanstalk 应用程序和环境以在阶段的部署操作中使用。</p>

	<p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">Elastic Beanstalk 演练</a></li><li>• <a href="#">在 AWS CodePipeline 中创建管道 (p. 82)</a></li></ul>
AWS OpsWorks Stacks	<p>AWS OpsWorks 是一项配置管理服务，可帮助您利用 Chef 配置和操作各种类型和规模的应用程序。通过使用 AWS OpsWorks Stacks，您可以定义应用程序的架构和每个组件的规范 (包括软件包安装、软件配置和存储等资源)。您可以将 AWS CodePipeline 配置为使用 AWS OpsWorks Stacks 在 AWS OpsWorks 中部署您的代码与自定义 Chef 说明书和应用程序。</p> <ul style="list-style-type: none"><li>• 自定义 Chef 说明书 – AWS OpsWorks 使用 Chef 说明书来处理诸如安装和配置程序包以及部署应用程序等任务。</li><li>• 应用程序 – AWS OpsWorks 应用程序包含要在应用程序服务器上运行的代码。应用程序代码存储在存储库 (如 Amazon S3 存储桶) 中。</li></ul> <p>您应在创建管道之前创建要使用的 AWS OpsWorks 堆栈和层。在创建管道之前或使用 Create Pipeline 向导时，您可以创建 AWS OpsWorks 应用程序以在阶段的部署操作中使用。</p> <p>AWS CodePipeline 对 AWS OpsWorks 的支持目前仅在 美国东部 (弗吉尼亚北部) 区域 (us-east-1) 可用。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">将 AWS CodePipeline 与 AWS OpsWorks Stacks 结合使用</a></li><li>• <a href="#">说明书和诀窍</a></li><li>• <a href="#">AWS OpsWorks 应用程序</a></li></ul>
XebiaLabs	<p>您可以将 AWS CodePipeline 配置为使用 <a href="#">XebiaLabs</a> 来部署管道中一个或多个操作的代码。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">介绍将 XL Deploy 与 AWS CodePipeline 一起使用的 XebiaLabs 文档</a></li></ul>

## 审批操作集成

Amazon Simple Notification Service	<p><a href="#">Amazon SNS</a> 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 让您以简单且经济高效的方式来将推送通知发送给移动设备用户、电子邮件收件人，甚至可以将消息发送给其他分布式服务。</p> <p>当您在 AWS CodePipeline 中创建手动审批请求时，可以选择将其发布到 Amazon SNS 中的主题，以便通知所有订阅该主题的 IAM 用户审批操作已可供批准或拒绝。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">什么是 Amazon SNS？</a></li><li>• <a href="#">向 AWS CodePipeline 服务角色授予 Amazon SNS 权限 (p. 142)</a></li></ul>
------------------------------------	---

## 调用操作集成

AWS Lambda	<p>利用 <a href="#">Lambda</a>，您可以运行代码而无需预配置或管理服务器。您可以将 AWS CodePipeline 配置为使用 Lambda 函数为您的管道增加灵活性和功能。在创建管道之前或使用 Create Pipeline 向导时，您可以创建 Lambda 函数以作为操作添加到阶段中。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">在 AWS CodePipeline 管道中调用 AWS Lambda 函数 (p. 120)</a></li></ul>
------------	---

## 与 AWS CodePipeline 的一般集成

以下 AWS 服务集成不基于 AWS CodePipeline 操作类型。

AWS CloudTrail	<p><a href="#">CloudTrail</a> 捕获由 AWS 账户或代表该账户发起的 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。您可以将 CloudTrail 配置为从 AWS CodePipeline 控制台、AWS CLI 中的 AWS CodePipeline 命令和 AWS CodePipeline API 捕获 API 调用。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">使用 AWS CloudTrail 记录 AWS CodePipeline API 调用 (p. 160)</a></li></ul>
Amazon CloudWatch	<p><a href="#">Amazon CloudWatch</a> 监控您的 AWS 资源。</p> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">什么是 Amazon CloudWatch？</a></li></ul>
Amazon CloudWatch Events	<p><a href="#">Amazon CloudWatch Events</a> 是一项 Web 服务，它根据定义的规则检测您的 AWS 服务中的更改，并在发生更改时在一个或多个指定的 AWS 服务中调用操作。</p> <ul style="list-style-type: none"><li>• 在发生更改时自动启动管道执行 — 您可以在 Amazon CloudWatch Events 上设置的规则中将 AWS CodePipeline 配置为目标。这会将管道设置为在其他服务发生更改时自动启动。</li></ul> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">什么是 Amazon CloudWatch Events？</a></li><li>• <a href="#">在 AWS CodePipeline 中启动管道执行 (p. 62)</a>。</li><li>• <a href="#">使用 CloudWatch Events 规则自动启动 AWS CodeCommit 管道 (p. 64)</a></li><li>• 在管道状态发生更改时收到通知 — 您可以设置 Amazon CloudWatch Events 规则以检测和响应管道、阶段或操作的执行状态更改。</li></ul> <p>了解更多：</p> <ul style="list-style-type: none"><li>• <a href="#">使用 Amazon CloudWatch Events 检测和响应管道状态更改 (p. 153)</a></li><li>• <a href="#">教程：设置 CloudWatch Events 规则以接收管道状态更改的电子邮件通知 (p. 54)</a></li></ul>
AWS Key Management Service	<p><a href="#">AWS KMS</a> 是一项托管服务，可让您轻松创建和控制用于加密数据的加密密钥。默认情况下，AWS CodePipeline 使用 AWS KMS 对存储在 Amazon S3 存储桶中的管道项目进行加密。</p>

| 了解更多 : |



- 可以使用两种方法为 Amazon S3 项目配置服务器端加密：
  - 在使用创建管道向导创建管道时，AWS CodePipeline 创建 Amazon S3 项目存储桶和默认 AWS 托管 SSE-KMS 加密密钥。主密钥是与数据数据一起加密的，并由 AWS 进行管理。

- 您可以创建和管理自己的客户托管 SSE-KMS 密钥。如果使用默认的 Amazon S3 密钥，则无法更改或删除此 AWS 管理的密钥。如果您在 AWS KMS 中使用客户管理的密钥来加密或解密 Amazon S3 存储桶中的项目，则可以根据需要更改或轮换该密钥。

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-2-890506445442/*",
"Condition": {
"StringNotEquals": {
"s3:x-amz-server-side-encryption":
"aws:kms"
}
}
},
{
"Sid": "DenyInsecureConnections",

API 版本 2015-07-09	"Deny",
18	"Principal": "*",
	"Action": "s3:*",



## 来自社区的示例

以下各部分提供的链接指向博客帖子、文章和社区提供的示例。

### Note

提供的这些链接仅供参考，不应视为全面列表或支持示例内容。AWS 对这些内容或外部内容的准确性不承担责任。

### 主题

- [集成示例：博客文章 \(p. 19\)](#)
- [集成示例：视频 \(p. 21\)](#)

## 集成示例：博客文章

- [使用 AWS CodePipeline 实施 DevSecOps](#)

了解如何在 AWS CodePipeline 中使用 CI/CD 管道来实现预防和探测安全控制自动化。本文介绍如何使用管道创建简单的安全组，并在源、测试和生产阶段执行安全检查，以改善 AWS 账户的安全状况。

发布时间：2017 年 3 月

- [使用 AWS CodePipeline、AWS CodeBuild、Amazon ECR 和 AWS CloudFormation 持续部署到 Amazon ECS](#)

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

- 下载一个示例 AWS CloudFormation 模板和使用说明，从 GitHub 上的 [ECS 参考架构：持续部署存储库](#) 创建您自己的持续部署管道。

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 AWS 服务集成为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) 定义应用程序及其资源，并使用 AWS CodePipeline 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

- [将 Git 与 AWS CodePipeline 集成](#)

了解如何将 AWS CodePipeline 与支持 Webhooks 功能的 Git 服务器 (如 GitHub Enterprise、Bitbucket 和 GitLab) 集成。

发布时间：2016 年 11 月

- [使用 AWS CodePipeline 和 Dynatrace 扩展 DevOps 部署](#)

了解如何使用 Dynatrace 监控解决方案在 AWS CodePipeline 中扩展管道，在提交代码之前自动分析测试执行，以及保持最佳的准备时间。

发布时间：2016 年 11 月

- [使用 CloudFormation 和 CodeCommit 在 AWS CodePipeline 中为 AWS Elastic Beanstalk 创建管道](#)

了解如何在 AWS CodePipeline 管道中为 AWS Elastic Beanstalk 应用程序实现持续交付。所有 AWS 资源都通过使用 AWS CloudFormation 模板自动预配置。此演练还纳入了 AWS CodeCommit 和 AWS Identity and Access Management (IAM)。

发布时间：2016 年 5 月

- [在 AWS CloudFormation 中实现 AWS CodeCommit 和 AWS CodePipeline 自动化](#)

使用 AWS CloudFormation 自动预配置 AWS 资源以实现使用 AWS CodeCommit、AWS CodePipeline、AWS CodeDeploy 和 AWS Identity and Access Management 的持续交付管道。

发布时间：2016 年 4 月

- [在 AWS CodePipeline 中创建跨账户管道](#)

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。包括 AWS CloudFormation 模板中的示例。

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 AWS CodeDeploy 和 AWS CodePipeline 为 ASP.NET 内核应用程序创建完整的持续交付系统。

发布时间：2016 年 3 月

- [使用 AWS CodePipeline 控制台创建管道](#)

了解如何使用 AWS CodePipeline 控制台在基于 AWS CodePipeline [教程：创建一个四阶段管道 \(p. 47\)](#) 的演练中创建一个两阶段管道。

发布时间：2016 年 3 月

- [使用 AWS Lambda 模拟 AWS CodePipeline 管道](#)

了解如何调用 Lambda 函数，让您可以在管道运行之前，在设计过程中查看 AWS 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

- [使用 AWS CloudFormation 在 AWS CodePipeline 中运行 AWS Lambda 函数](#)

了解如何创建一个 AWS CloudFormation 堆栈，以预配置用户指南任务在 [AWS CodePipeline 管道中调用 AWS Lambda 函数 \(p. 120\)](#) 中使用的所有 AWS 资源。

发布时间：2016 年 2 月

- [在 AWS CloudFormation 中预配置自定义 AWS CodePipeline 操作](#)

了解如何使用 AWS CloudFormation 在 AWS CodePipeline 中预配置自定义操作。

发布时间：2016 年 1 月

- [使用 AWS CloudFormation 预配置 AWS CodePipeline](#)

了解如何使用 AWS CloudFormation 在 AWS CodePipeline 中预配置基本持续交付管道。

发布时间：2015 年 12 月

- [使用 AWS CodePipeline、Jenkins 和 Elastic Beanstalk 在 AWS 上构建持续部署](#)

了解如何使用 GitHub、AWS CodePipeline、Jenkins 和 Elastic Beanstalk 为您每次更改代码时自动更新的 Web 应用程序创建一个部署管道。

发布时间：2015 年 12 月

- [使用 AWS CodePipeline、Elastic Beanstalk 和 Solano Labs 持续交付 PHP 应用程序](#)

了解如何将 Solano CI 与 AWS CodePipeline 结合使用来测试使用 PHPUnit 的 PHP 应用程序，然后将应用程序部署到 Elastic Beanstalk。

发布时间：2015 年 12 月

- [使用 AWS CodePipeline 和 BlazeMeter 对持续交付进行性能测试](#)

了解如何使用 BlazeMeter 的原生 AWS CodePipeline 集成在 AWS CodePipeline 交付工作流程中的正确位置注入自动负载测试。

发布时间：2015 年 9 月

- [使用自定义操作和 AWS Lambda 从 AWS CodePipeline 部署到 AWS OpsWorks](#)

了解如何配置管道和 AWS Lambda 函数以使用 AWS CodePipeline 部署到 AWS OpsWorks。

发布时间：2015 年 7 月

- [自动交付验收测试 Nirvana：由 AWS CodePipeline、CloudWatch 和 BlazeMeter 提供支持](#)

了解如何使用 AWS CodePipeline、CloudWatch 和 BlazeMeter 创建一个持续交付工作流程，从而缩短发布时间，并增加开发人员在发布过程中的测试覆盖面。

发布时间：2015 年 7 月

## 集成示例：视频

- [使用 AWS CodePipeline 控制台创建管道](#)

了解如何使用 AWS CodePipeline 控制台创建使用 AWS CodeDeploy 和 Amazon S3 的管道。

[使用 AWS CodePipeline 控制台创建管道](#)

发布时间：2016 年 3 月

持续时间：8 分 53 秒

- [Solano CI 和 AWS CodePipeline](#)

查看使用 Solano CI 设置并成功运行 AWS CodePipeline 的演示。

[Solano CI 和 AWS CodePipeline](#)

发布时间：2015 年 11 月

持续时间：3 分 07 秒

# AWS CodePipeline 教程

完成[AWS CodePipeline 入门 \(p. 8\)](#)中的步骤后，您可以尝试本用户指南中的其中一个教程：

我想创建一个使用 AWS CodeDeploy 的管道，以将示例应用程序从 Amazon S3 存储桶部署到运行 Amazon Linux 的 Amazon EC2 实例。	请参阅教程： <a href="#">创建一个简单的管道 (Amazon S3 存储桶) (p. 22)</a> 。
我想创建一个使用 AWS CodeDeploy 的管道，以将示例应用程序从 AWS CodeCommit 存储库部署到运行 Amazon Linux 的 Amazon EC2 实例。	请参阅教程： <a href="#">创建一个简单的管道 (AWS CodeCommit 存储库) (p. 36)</a> 。

## Note

[教程：创建一个四阶段管道 \(p. 47\)](#)显示了如何创建一个执行以下操作的管道：从 GitHub 存储库获取源代码，使用 Jenkins 生成和测试源代码，然后使用 AWS CodeDeploy 将生成和测试的源代码部署到运行 Amazon Linux 或 Microsoft Windows Server 的 Amazon EC2 实例。由于本教程以演练中涵盖的概念为基础，因此我们建议您先完成其中至少一个演练。

如果您想尝试对 [AWS CodeDeploy](#) 使用 AWS CodePipeline，而不设置所有必需的资源，请尝试使用[管道初学者工具包](#)探究 [AWS 中的持续交付](#)。初学者工具包会建立一个只需几步即可生成和部署示例应用程序的完整管道，并使用 AWS CloudFormation 模板在美国东部（弗吉尼亚北部）区域创建管道及其所有资源。

其他用户指南中的以下教程和演练提供了将其他 AWS 服务集成到管道中的指导：

- [AWS CodeBuild 用户指南](#) 中的 [创建使用 AWS CodeBuild 的管道](#)
- [AWS OpsWorks 用户指南](#) 中的 [将 AWS CodePipeline 与 AWS OpsWorks Stacks 结合使用](#)
- [AWS CloudFormation 用户指南](#) 中的 [利用 AWS CodePipeline 进行持续交付](#)
- [AWS Elastic Beanstalk 开发人员指南](#) 中的 [Elastic Beanstalk 演练](#)
- [AWS for DevOps 入门指南](#)
- [使用管道初学者工具包探究 AWS 中的持续交付](#)
- [使用 AWS CodePipeline 设置持续部署管道](#)

## 教程：创建一个简单的管道 (Amazon S3 存储桶)

创建管道的最简单方法是使用 AWS CodePipeline 控制台中的创建管道向导。

在本演练中，您将创建一个两阶段管道，该管道使用受版本控制的 Amazon S3 存储桶和 AWS CodeDeploy 来发布示例应用程序。

## Note

对于具有 Amazon S3 源的管道，Amazon CloudWatch Events 规则将检测源更改，然后在发生更改时启动您的管道。在使用控制台创建或更改管道时，将为您创建规则和相关联资源。如果您在 CLI 或 AWS CloudFormation 中创建或更改 Amazon S3 管道，则必须手动创建 Amazon CloudWatch Events 规则、IAM 角色和 AWS CloudTrail 跟踪。

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

除了完成[AWS CodePipeline 入门 \(p. 8\)](#)中的步骤外，您还应该创建一个 Amazon EC2 实例密钥对，可使用该密钥对在 Amazon EC2 实例启动后连接到这些实例。要创建 Amazon EC2 实例密钥对，请按照[使用 Amazon EC2 创建密钥对](#)。

不是您要找的内容？要使用 AWS CodeCommit 分支作为代码存储库来创建一个简单的管道，请参阅[教程：创建一个简单的管道 \(AWS CodeCommit 存储库\) \(p. 36\)](#)。

在开始之前，您应完成[AWS CodePipeline 入门 \(p. 8\)](#)中的先决条件。

### 主题

- [步骤 1：为您的应用程序创建一个 Amazon S3 存储桶 \(p. 23\)](#)
- [步骤 2：创建 AWS CodeDeploy 资源以部署示例应用程序 \(p. 24\)](#)
- [步骤 3：在 AWS CodePipeline 中创建您的第一个管道 \(p. 26\)](#)
- [步骤 4：向管道中添加另一个阶段 \(p. 28\)](#)
- [步骤 5：在 AWS CodePipeline 中禁用和启用阶段之间的过渡 \(p. 34\)](#)
- [步骤 6：清理资源 \(p. 35\)](#)

## 步骤 1：为您的应用程序创建一个 Amazon S3 存储桶

您可以将源文件或应用程序存储在任意受版本控制的位置。在本演练中，您为示例应用程序创建一个 Amazon S3 存储桶，并对该存储桶启用版本控制。启用版本控制后，您可将示例应用程序复制到该存储桶。

如果要使用现有 Amazon S3 存储桶，请参阅[为存储桶启用版本控制](#)，将示例应用程序复制到该存储桶，然后跳到前面的[步骤 2：创建 AWS CodeDeploy 资源以部署示例应用程序 \(p. 24\)](#)。

如果要使用 GitHub 存储库而不是 Amazon S3 存储桶，请将示例应用程序复制到该存储库，然后向前跳转到[步骤 2：创建 AWS CodeDeploy 资源以部署示例应用程序 \(p. 24\)](#)。

### 要创建 Amazon S3 存储段

1. 登录 AWS 管理控制台并通过以下网址打开 Amazon S3 控制台：<https://console.aws.amazon.com/s3/>。
2. 选择 Create bucket (创建存储桶)。
3. 在 Bucket name 中，输入您的存储桶的名称，例如 **awscodepipeline-demobucket-example-date**。

### Note

由于 Amazon S3 中的所有存储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。请记住该名称，因为您需要在演练的其余部分中使用该名称。

在区域下拉框中，选择您将创建管道的区域，例如 美国西部（俄勒冈），然后选择下一步。

4. 选择 Versioning，选择 Enable versioning，然后选择 Save。

启用版本控制后，Amazon S3 会在存储桶中存储每个对象的每个版本。

5. 选择 Next。接受默认权限以允许您的账户对对象进行读/写访问。有关 Amazon 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](#)。
6. 选择 Next，然后选择 Create bucket。

7. 接下来，您将从 GitHub 存储库下载示例，并将其保存到本地计算机上的文件夹或目录中。

#### Important

请勿在 GitHub 存储库中使用 Clone or download 或 Download ZIP 按钮。这会创建无法与 AWS CodeDeploy 配合使用的嵌套文件夹结构。

- a. 打开托管示例的 GitHub 存储库。
    - 如果您想使用 AWS CodeDeploy 部署到 Amazon Linux 实例：[https://github.com/aws-labs/aws-codepipeline-s3-aws-codedeploy\\_linux](https://github.com/aws-labs/aws-codepipeline-s3-aws-codedeploy_linux)
    - 如果您想使用 AWS CodeDeploy 部署到 Windows Server 实例：[https://github.com/aws-labs/AWSCodePipeline-S3-AWSCodeDeploy\\_Windows](https://github.com/aws-labs/AWSCodePipeline-S3-AWSCodeDeploy_Windows)
  - b. 选择 dist 文件夹。
  - c. 选择文件名称。
    - 如果您想部署到 Amazon Linux 实例：aws-codepipeline-s3-aws-codedeploy\_linux.zip
    - 如果您想部署到 Windows Server 实例：AWSCodePipeline-S3-AWSCodeDeploy\_Windows.zip
  - d. 选择 View Raw，然后将该示例文件保存到本地计算机。
8. 在存储桶的 Amazon S3 控制台中，选择 Upload，然后按照说明将您的 .zip 文件上传到存储桶。

## 步骤 2：创建 AWS CodeDeploy 资源以部署示例应用程序

可使用 Elastic Beanstalk 或 AWS CodeDeploy 部署代码进行暂存或部署到生产环境中。在本演练中，您将使用 AWS CodeDeploy 中的示例部署向导创建部署资源。

### 创建 AWS CodeDeploy 自动部署

1. 在您打算创建管道的区域打开 AWS CodeDeploy 控制台：<https://console.aws.amazon.com/codedeploy>。

有关 AWS CodePipeline 可用的区域和终端节点的更多信息，请参阅[区域和终端节点](#)。

如果您看到 Applications 页面，而不是 Welcome 页面，则在 More info 部分中选择 Sample deployment wizard。

2. 在 Get started with AWS CodeDeploy 页中，选择 Sample deployment，然后选择 Next。
3. 在 Choose a deployment type 页中，选择 In-place deployment，然后选择 Next。
4. 在 Configure instances 页中，执行以下操作：
  1. 选择您要使用的操作系统和 Amazon EC2 实例密钥对。您选择的操作系统应与从 GitHub 中下载的示例应用程序相符。

#### Important

如果您已下载 aws-codepipeline-s3-aws-codedeploy\_linux.zip，请选择 Amazon Linux。

如果您已下载 AWSCodePipeline-S3-AWSCodeDeploy\_Windows.zip，请选择 Windows Server。

2. 从密钥对名称下拉列表中，选择将用于连接到已启动的 Amazon EC2 实例的 Amazon EC2 实例密钥对的名称。
3. 在 Tag key and value 中，保留 Key 名称不变。在 Value 中，键入 **CodePipelineDemo**。



#### 4. 选择 Launch instances。

### Configure instances ?

To supply the resources for the sample application deployment, AWS CodeDeploy will use AWS CloudFormation to launch three Amazon EC2 instances for you using the following configuration. After the instances launch, they can be used in a deployment.

**Operating system\*** ☒ Amazon Linux ☐ Windows Server

**Instance type\*** t1.micro ?

**Key pair name\***  ?

**Tag key and value\***

<input type="text" value="Name"/>	<input type="text" value="CodeDeployDemo"/>	<span>?</span>
Key	Value	

Launch the instances in AWS CloudFormation. This may take a few minutes.

Choose **Skip** if you have existing Amazon EC2 instances that you want to deploy to.

**Launch instances**

---

\*Required Cancel Previous Skip Next

有关这些选项的更多信息，请参阅 AWS CodeDeploy User Guide 中的 [Step 3: Configure instances](#)。

5. 创建您的实例后，选择 Next。
6. 在 Name your application 页中，在 Application name 中键入 **CodePipelineDemoApplication**，然后选择 Next。
7. 在 Select a revision 页中，选择 Next。
8. 在 Create a deployment group 页中，在 Deployment group name 中键入 **CodePipelineDemoFleet**，然后选择 Next。

#### Note

在 Step 3: Configure instances 中为您创建的实例列于 Add instances 区域。

9. 在 Select a service role 页中，从 Service role 下拉框中选择 Use an existing service role。在 Role name 列表中，选择您要使用的服务角色，然后选择 Next。

#### Note

如果 Role name 中没有显示任何服务角色，或者没有您想使用的服务角色，则选择 Create a service role 或遵循 [创建服务角色](#) 中的步骤。

为 AWS CodeDeploy 创建或使用的服务角色不同于为 AWS CodePipeline 创建的服务角色。

10. 在 Choose a deployment configuration 页中，选择 Next。
11. 在 Review deployment details 页中，选择 Deploy。AWS CodeDeploy 的示例应用程序将部署到每个 Amazon EC2 实例。
12. 成功部署示例应用程序之后，转到部署中的每个 Amazon EC2 实例的 <http://PublicDNS>，在 Web 浏览器中验证部署。要获取 Amazon EC2 实例的公有 DNS 值，请在 Amazon EC2 控制台中选择实例，然后在 Description 选项卡上查找 Public DNS 的值。

网页显示指向 AWS CodeDeploy 文档的链接。

#### Note

此网页不会与您下一部分中创建的管道一起部署和发布。

有关 AWS CodeDeploy 的更多信息，请参阅 [AWS CodeDeploy 入门](#)。

## 步骤 3：在 AWS CodePipeline 中创建您的第一个管道

在这部分的演练中，您将创建管道。示例将自动通过管道运行。

### 创建 AWS CodePipeline 自动发布流程

1. 登录 AWS 管理控制台并通过以下网址打开 AWS CodePipeline 控制台：<http://console.aws.amazon.com/codepipeline>。
2. 在介绍页面上，选择 Get started。

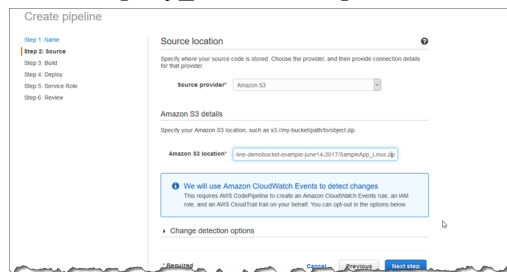
如果您看到 Welcome 页面，则选择 Create pipeline。

3. 在 Step 1: Name 中，在 Pipeline name 中键入 **MyFirstPipeline**，然后选择 Next step。

#### Note

如果您为管道选择另一个名称，请务必确保用此名称替换此演练其他部分的 **MyFirstPipeline**。当您创建管道后，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 [AWS CodePipeline 中的限制 \(p. 208\)](#)。

4. 在 Step 2: Source 中，在 Source provider 中选择 Amazon S3。在 Amazon S3 location 中，键入您在 [步骤 1：为您的应用程序创建一个 Amazon S3 存储桶 \(p. 23\)](#) 中创建的 Amazon S3 存储桶和您复制到该存储桶的示例文件的名称，即 `aws-codepipeline-s3-aws-codedeploy_linux.zip` 或 `AWSCodePipeline-S3-AWSCodeDeploy_Windows.zip`。选择下一步。



例如，如果您的存储桶名为 **awscodepipeline-demobucket-example-date** 且您为 AWS CodeDeploy 中的 Amazon EC2 实例选择 Amazon Linux，则您需要键入：

```
s3://awscodepipeline-demobucket-example-date/aws-codepipeline-s3-aws-codedeploy_linux.zip
```

如果您的存储桶名为 **awscodepipeline-demobucket-example-date** 且您为 AWS CodeDeploy 中的 Amazon EC2 实例选择 Windows，则您需要键入：

```
s3://awscodepipeline-demobucket-example-date/AWSCodePipeline-S3-AWSCodeDeploy_Windows.zip
```

#### Note

如果您已将示例应用程序复制到 GitHub 存储库，而不是 Amazon S3 存储桶，则从源提供程序列表中选择 GitHub，然后按照说明执行操作。有关更多信息，请参阅 [创建管道 \(控制台\) \(p. 84\)](#)。

选择源提供程序后，将显示一条消息，指明将为此管道创建 Amazon CloudWatch Events 规则。该消息还以如下格式显示要添加到 AWS CloudTrail 跟踪的数据事件：`my-bucket/path/to/object.zip`。



在更改检测选项下面，保留默认值。这样，AWS CodePipeline 就可以使用 Amazon CloudWatch Events 检测源存储桶中的更改。

- 在 Step 3: Build 中，选择 No Build，然后选择 Next step。

#### Note

您可以配置一个具有提供程序的生成操作，例如 AWS CodeBuild，它是云中的完全托管生成服务。您还可以配置一个将提供程序与生成服务器或系统结合使用的生成操作，例如 Jenkins。在下一个教程中，您可以查看设置构建资源和创建使用这些资源的管道的步骤：[教程：创建一个四阶段管道 \(p. 47\)](#)。

- 在 Step 4: Deploy 中，在 Deployment provider 中选择 AWS CodeDeploy。在 Application name 中，键入 **CodePipelineDemoApplication**，或选择 Refresh 按钮，然后从列表中选择应用程序名称。在 Deployment group 中，键入 **CodePipelineDemoFleet** 或从列表中选择，然后选择 Next step。

### Create pipeline

Step 1: Name  
Step 2: Source  
Step 3: Build  
**Step 4: Deploy**  
Step 5: Service Role  
Step 6: Review

#### Deploy

Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.

Deployment provider\* AWS CodeDeploy

AWS CodeDeploy ⓘ

Choose one of your existing applications, or [create a new one in AWS CodeDeploy](#).

Application name\* MyDemoApplication

Choose one of your existing deployment groups, or [create a new one in AWS CodeDeploy](#).

Deployment group\* MyDemoDeploymentGroup

\* Required Cancel Previous Next step

#### Note

名称“Staging”是在 Step 4: Deploy 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。

- 在 Step 5: Service Role 中，选择 Create role。

在描述将为您创建的 AWS-CodePipeline-Service 角色的 IAM 控制台页中，选择允许。

在 Step 5: Service Role 页中，AWS-CodePipeline-Service 显示在 Role name 中，选择 Next step。

#### Note

只有第一次在 AWS CodePipeline 中创建管道时才需要创建服务角色。如果已创建服务角色，您可以从角色下拉列表中选择服务角色。下拉列表显示所有与您的账户关联的 IAM 服务角色。如果您选择一个与默认名称不同的名称，请确保该名称可识别为 AWS CodePipeline 的服务角色。

- 在 Step 6: Review 中，查看信息，然后选择 Create pipeline。
- 管道会自动开始运行。在 AWS CodePipeline 示例为 AWS CodeDeploy 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。

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

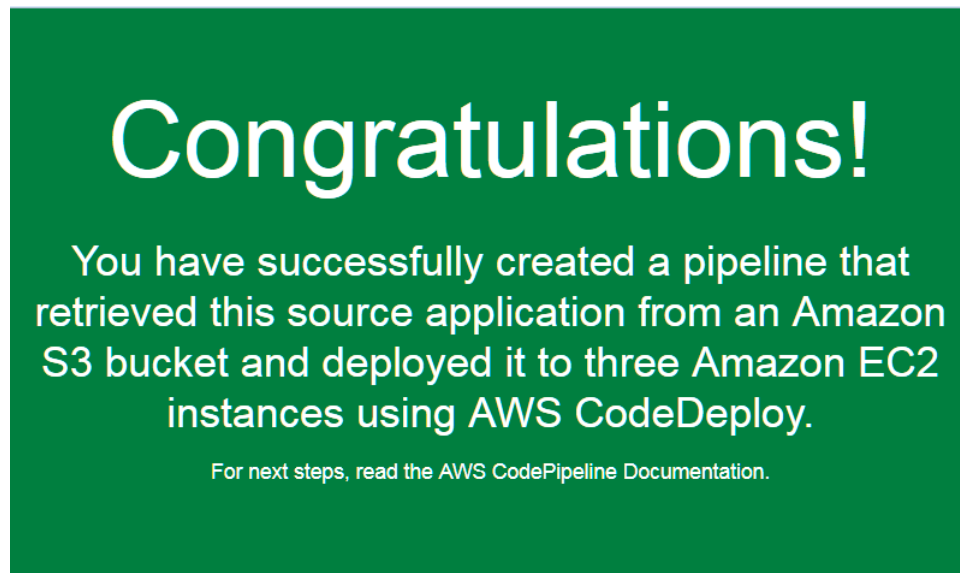
- 一个名为 Source 的源阶段，此阶段将自动探测存储在 Amazon S3 存储桶中的受版本控制的示例应用程序中的更改，并将这些更改推入管道中。
- 一个 Staging 阶段，该阶段使用 AWS CodeDeploy 将这些更改部署到 Amazon EC2 实例。

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 Succeeded 之后，在 Staging 阶段的状态区域，选择 Details。
3. 在 Deployment details 部分，在 Instance ID 中，选择其中一个成功部署的实例的实例 ID。
4. 在 Description 选项卡上，在 Public DNS 中复制地址，然后将其转帖到 Web 浏览器的地址栏中。

以下页面是您上传到 Amazon S3 存储桶的示例应用程序。



有关阶段和操作以及管道如何工作的更多信息，请参阅[AWS CodePipeline 概念 \(p. 3\)](#)。

## 步骤 4：向管道中添加另一个阶段

现在，使用 AWS CodeDeploy 在管道中添加另一个阶段，以便从暂存服务器部署到生产服务器。首先，您在 AWS CodeDeploy 的 CodePipelineDemoApplication 中创建另一个部署组。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，您可使用 AWS CodePipeline 控制台或 AWS CLI 来检索并手动编辑 JSON 文件中的管道结构，然后运行 update-pipeline 命令使用所做更改更新管道。

主题

- [在 AWS CodeDeploy 中创建第二个部署组 \(p. 29\)](#)
- [将部署组作为管道中的另一个阶段添加 \(p. 29\)](#)

## 在 AWS CodeDeploy 中创建第二个部署组

### Note

在这部分的演练中，您创建第二个部署组，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。它故意设计为失败，以便向您展示 AWS CodePipeline 中显示错误的方式。

### 在 AWS CodeDeploy 中创建第二个部署组

1. 打开 AWS CodeDeploy 控制台：<https://console.aws.amazon.com/codedeploy>。
2. 选择 Applications，然后在应用程序列表中选择 CodePipelineDemoApplication。
3. 在 Deployment groups 中，选择 Create deployment group。
4. 在 Create deployment group 页中，在 Deployment group name 中键入第二个部署组的名称，例如 **CodePipelineProductionFleet**。
5. 选择 In-place deployment。
6. 在 Key 中输入 **Name**，但在 Value 中，从列表中选择 CodePipelineDemo。保留 Deployment configuration 的默认配置。在 Service role ARN 中，选择用于初始部署的相同 AWS CodeDeploy 服务角色（而不是 AWS CodePipeline 服务角色），然后选择 Create deployment group。

## 将部署组作为管道中的另一个阶段添加

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以便部署到您之前使用的相同 Amazon EC2 实例。您可以使用 AWS CodePipeline 控制台或 AWS CLI 添加此阶段。

### 主题

- [创建第三个阶段 \(控制台\) \(p. 29\)](#)
- [创建第三个阶段 \(CLI\) \(p. 32\)](#)

### 创建第三个阶段 (控制台)

您可以使用 AWS CodePipeline 控制台添加一个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 Amazon EC2 实例，因此该阶段的部署操作将失败。

1. 登录 AWS 管理控制台并通过以下网址打开 AWS CodePipeline 控制台：<http://console.aws.amazon.com/codepipeline>。
2. 在 Name 中，选择您创建的管道名称 MyFirstPipeline。
3. 在管道详细信息页中，选择 Edit。
4. 在 Edit 页中，选择 + Stage 以紧随 Staging 阶段之后添加一个阶段。

## Edit: MyFirstPipeline

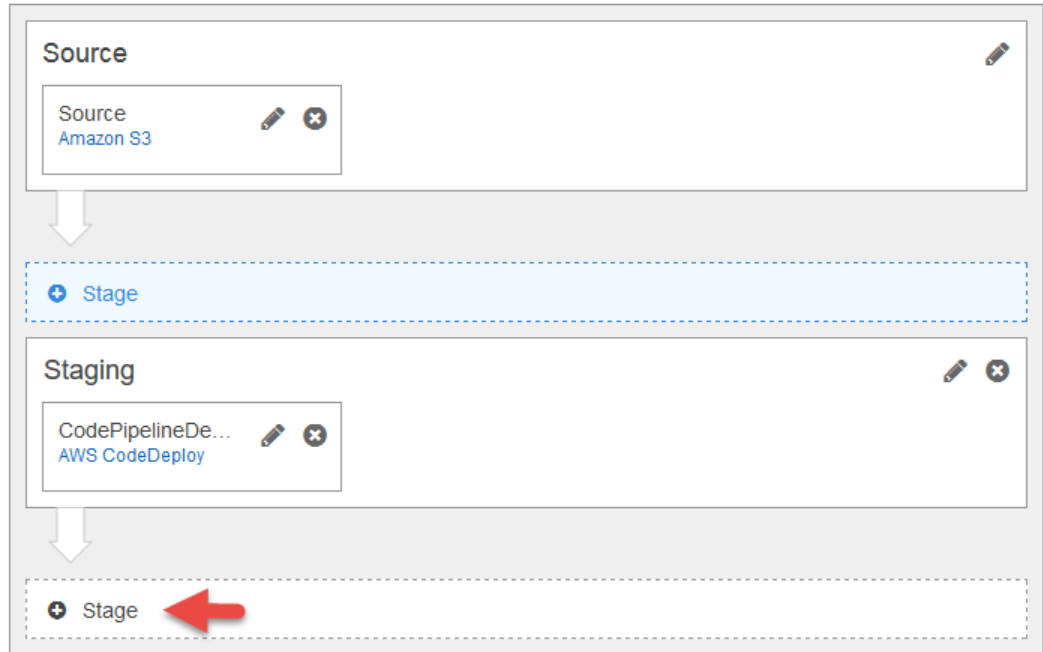


Add or edit a stage in a pipeline or actions in a stage. [Learn more](#)

Cancel

Delete

Save pipeline changes



5. 在新阶段的名称字段中，键入 **Production**，然后选择 + Action：



6. 在 Action category 下拉列表中，选择 Deploy。

在 Action name 中，键入 **Deploy-Second-Deployment**。

在 Deployment provider 中，从下拉列表中选择 AWS CodeDeploy。

在 AWS CodeDeploy 部分，在 Application name 中，从下拉列表中选择 CodePipelineDemoApplication，正如您在创建管道时一样。在 Deployment group 中，选择您刚刚创建的部署组 **CodePipelineProductionFleet**。在 Input artifacts 部分，在 **Input artifacts #1** 中键入 MyApp，然后选择 Add action。

### Note

作为源操作的输出项目，Create pipeline 向导中将自动为您创建输入项目的名称 **MyApp**。每个操作具有输入项目（操作处理的项目）和/或输出项目（操作的产物或结果），具体取决于操作类型。在此示例中，部署操作将源操作的输出输入到源阶段 MyApp，然后进行部署。由于针对上一阶段 (Staging) 配置的操作已将应用程序部署到相同的 Amazon EC2 实例，因此操作将会失败。有关输入和输出项目以及管道结构的更多信息，请参阅 [AWS CodePipeline 管道结构参考](#) (p. 202)。

7. 在 Edit 页中，选择 Save pipeline changes。在 Save pipeline changes 对话框中，选择 Save and continue。
8. 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。在管道详细信息页中，

选择 发布更改，然后在系统提示时选择 Release。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。

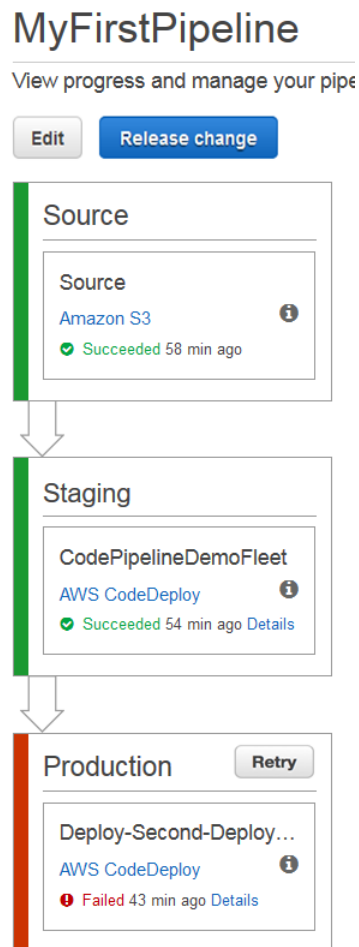
或者，要使用 AWS CLI 从本地 Linux, macOS, or Unix 计算机的终端或本地 Windows 计算机的命令提示符重新运行管道，请运行 start-pipeline-execution 命令，在命令中指定管道名称。这将第二次通过管道运行您的源存储桶中的应用程序。

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

该命令返回 pipelineExecutionId 对象。

9. 返回到 AWS CodePipeline 控制台，然后在管道列表中选择 MyFirstPipeline 以打开视图页面。

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。



10. 在 Deploy-Second-Deployment 操作中，选择 Details。在 Action execution failed 对话框中，选择 Link to execution details。您将被重定向到 AWS 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 Amazon EC2 实例，第二个部署组没有任何实例。

#### 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，但它提供了对管道结构的更多可见性。

为管道创建第三个阶段

1. 在本地 Linux, macOS, or Unix 计算机上打开终端会话，或在本地 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"
```

该命令会返回 MyFirstPipeline 的结构。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 Staging 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 Staging 阶段一样，您可以将其用作第三个阶段的模板。
4. 更改阶段名称和部署组详细信息，然后保存该文件。

以下示例显示将在 Staging 阶段结束后添加到 `pipeline.json` 文件中的 JSON。使用新值编辑突出显示的元素。请记住，使用逗号分隔 Staging 和 Production 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. 运行 `update-pipeline` 命令并指定一个管道 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。

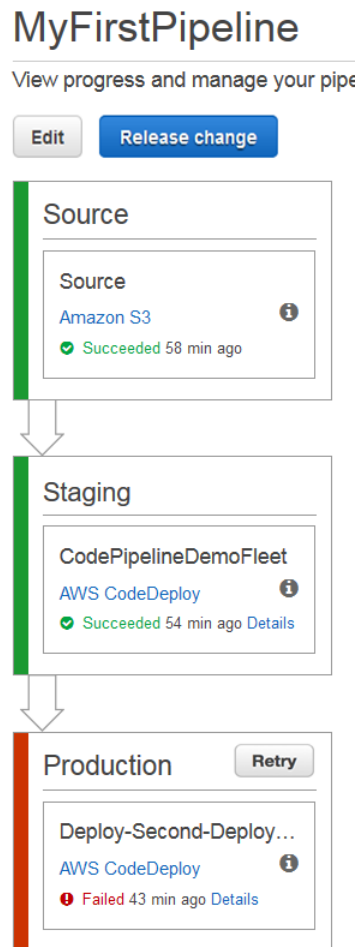
6. 运行 `start-pipeline-execution` 命令，在命令中指定管道名称。这将第二次通过管道运行您的源存储桶中的应用程序。

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

该命令返回 `pipelineExecutionId` 对象。

7. 打开 AWS CodePipeline 控制台，并从管道列表中选择 `MyFirstPipeline`。

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。



8. 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 AWS 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 Amazon EC2 实例，第二个部署组没有任何实例。

#### 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。

## 步骤 5：在 AWS CodePipeline 中禁用和启用阶段之间的过渡

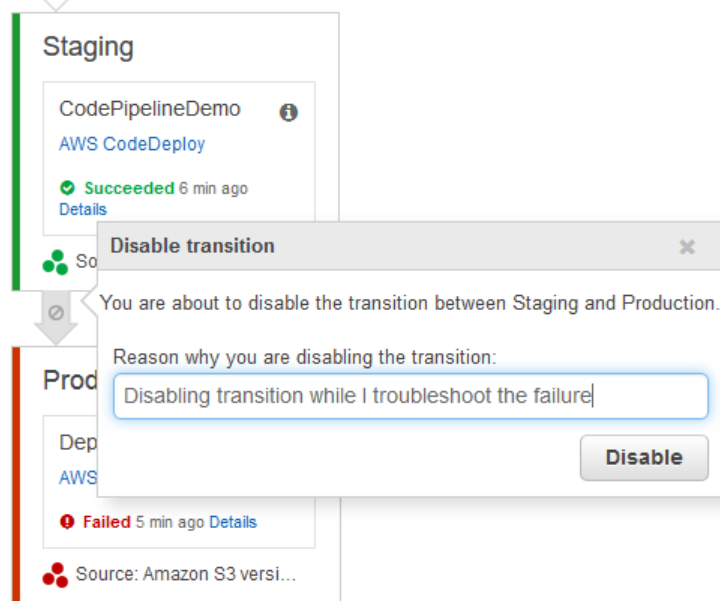
您可以启用或禁用管道中阶段之间的过渡。禁用阶段之间的过渡允许您手动控制一个阶段和另一个阶段之间的过渡。例如，您可能希望运行管道的前两个阶段，但是禁用向第三阶段的过渡，直到您准备好部署到生产环境，或者您要排查该阶段的问题或故障。

### 禁用和启用 AWS CodePipeline 管道中的各个阶段之间的过渡

1. 打开 AWS CodePipeline 控制台，并从管道列表中选择 MyFirstPipeline。
2. 在管道详细信息页中，选择指示第二个阶段 Staging 与您在上一部分中添加的第三个阶段 Production 之间的过渡的箭头。
3. 在 Disable transition 对话框中，键入禁用阶段过渡的原因，然后选择 Disable。



各个阶段之间的箭头显示一个图标和颜色变化，表示过渡已禁用。



4. 将示例再次上传到 Amazon 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。有关信息，请参阅 [Upload the sample application \(p. 24\)](#)。
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. 通过选择表示两个阶段之间的过渡已禁用的箭头，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

如果您希望第三个阶段成功，请在启用过渡之前编辑 `CodePipelineProductionFleet` 部署组，并指定一组不同的 Amazon EC2 实例 (用于部署应用程序)。有关如何执行此操作的更多信息，请参阅 [更改部署组设置](#)。如果您创建多个 Amazon EC2 实例，则可能会产生额外成本。

## 步骤 6：清理资源

您可以将在本演练中创建的一些资源用于教程 [教程：创建一个四阶段管道 \(p. 47\)](#)。例如，您可以重用 AWS 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 AWS CodeDeploy 应用程序及其关联的 Amazon EC2 实例，最后删除 Amazon S3 存储桶。

#### 清除本教程中使用的资源

1. 要清除您的 AWS CodePipeline 资源，请遵循 [删除 AWS CodePipeline 中的资源 \(p. 101\)](#) 中的说明。
2. 要清除您的 AWS CodeDeploy 资源，请遵循 [清除部署演练资源](#) 中的说明。
3. 要删除 Amazon S3 存储桶，请遵循 [删除或清空 Amazon S3 存储桶](#) 中的说明。如果您不打算创建更多管道，请删除为了存储管道项目而创建的 Amazon S3 存储桶。有关此存储桶的更多信息，请参阅 [AWS CodePipeline 概念 \(p. 3\)](#)。

## 教程：创建一个简单的管道 (AWS CodeCommit 存储库)

开始使用 AWS CodePipeline 最简单的方法是使用 AWS CodePipeline 控制台中的 Create Pipeline 向导创建一个简单的管道。

在本教程中，您将使用 AWS CodePipeline 将 AWS CodeCommit 存储库中维护的代码部署到单个 Amazon EC2 实例。您可将 AWS CodeDeploy 用作部署服务。

不是您要找的内容？要将受版本控制的 Amazon S3 存储桶作为代码存储库来创建一个简单的管道，请参阅[教程：创建一个简单的管道 \(Amazon S3 存储桶\)](#) (p. 22)。

完成本教程后，您应该对将 AWS CodeCommit 用作管道中的存储库这一概念有了足够多的实践。

AWS CodePipeline 使用 Amazon CloudWatch Events 检测 AWS CodeCommit 源存储库和分支中的更改。该源类型的默认行为是，在发生更改时使用 Amazon CloudWatch Events 自动启动管道。在控制台中使用该向导创建管道时，将创建规则。

在开始之前，请确保您已完成以下任务：

- [配置 IAM 用户](#)
- [安装和配置 AWS CLI](#)
- [使用 Amazon EC2 创建密钥对](#)

此外，请确保完成以下特定于服务的任务：

- [AWS CodeCommit](#)：安装 Git 并配置凭证
- [AWS CodeDeploy](#)：创建 IAM 实例配置文件和 AWS CodeDeploy 服务角色
- [AWS CodePipeline](#)：将 AWS CodePipeline 权限分配给 IAM 用户角色

### Note

如果您已经完成[教程：创建一个简单的管道 \(Amazon S3 存储桶\)](#) (p. 22)教程，但尚未清理其资源，则需要为在该教程中使用的许多资源创建不同的名称。例如，您可以将管道命名为 **MySecondPipeline**，而不是使用 **MyFirstPipeline** 名称。

### 主题

- [步骤 1：创建 AWS CodeCommit 存储库和本地存储库](#) (p. 36)
- [步骤 2：向 AWS CodeCommit 存储库添加示例代码](#) (p. 37)
- [步骤 3：创建 Amazon EC2 实例并安装 AWS CodeDeploy 代理](#) (p. 38)
- [步骤 4：在 AWS CodeDeploy 中创建应用程序](#) (p. 40)
- [步骤 5：在 AWS CodePipeline 中创建您的第一个管道](#) (p. 40)
- [步骤 6：修改 AWS CodeCommit 存储库中的代码](#) (p. 45)
- [步骤 7：可选的阶段管理任务](#) (p. 47)
- [步骤 8：清理资源](#) (p. 47)

## 步骤 1：创建 AWS CodeCommit 存储库和本地存储库

要开始本教程，您应在 AWS CodeCommit 中创建一个存储库。您的管道在运行时将从该存储库获取源代码。您还应创建一个本地存储库，您可以先在该存储库中维护和更新代码，然后再将其推送到 AWS CodeCommit 存储库。

## Important

目前，只有以下区域的管道支持 AWS CodeCommit：

- 美国东部（俄亥俄州）区域 (us-east-2)
- 美国东部（弗吉尼亚北部）区域 (us-east-1)
- 美国西部（俄勒冈）区域 (us-west-2)
- 欧洲（爱尔兰）区域 (eu-west-1)
- 南美洲（圣保罗）区域 (sa-east-1)

请确保选择其中一个 AWS 区域完成本教程中的所有步骤。

按照 AWS CodeCommit 用户指南的 [Git 与 AWS CodeCommit 教程](#) 中的前两个步骤操作：

- [步骤 1：创建 AWS CodeCommit 存储库](#)
- [步骤 2：创建本地存储库](#)

## Note

有关连接到您创建的本地存储库的信息，请参阅[连接到 AWS CodeCommit 存储库](#)。

完成这两个步骤后，返回到此页面并继续执行下一步。不要继续执行 AWS CodeCommit 教程中的第三步，而是应先完成本教程中的其他步骤。

## 步骤 2：向 AWS CodeCommit 存储库添加示例代码

在此步骤中，您将下载为 AWS CodeDeploy 示例演练创建的示例应用程序的代码，并将其添加到您的 AWS CodeCommit 库。

1. 下载以下文件：

- [SampleApp\\_Linux.zip](#)。

2. 将文件从 [SampleApp\\_Linux.zip](#) 解压缩到您在上一步骤中创建的本地目录中（例如，`/tmp/my-demo-repo` 或 `c:\temp\my-demo-repo`）。

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, or Unix 计算机上，您的目录和文件层次结构应如下所示：

```
/tmp
├── my-demo-repo
│   ├── appspec.yml
│   ├── index.html
│   ├── LICENSE.txt
│   └── scripts
│       ├── install_dependencies
│       ├── start_server
│       └── stop_server
```

3. 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
```

4. 运行以下命令以立即暂存您的所有文件：

```
git add -A
```

5. 运行以下命令以提交带有提交消息的文件：

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

6. 运行以下命令以将文件从您的本地存储库推送到您的 AWS CodeCommit 存储库：

```
git push
```

7. 您下载并添加到您的本地存储库的文件现已被添加到 AWS CodeCommit MyDemoRepo 存储库中的 master 分支，并准备要包括在您的管道中。

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

在此步骤中，您将创建要向其部署示例应用程序的 Amazon EC2 实例。在此过程中，您将在实例上安装 AWS CodeDeploy 代理。AWS CodeDeploy 代理是一个支持在 AWS CodeDeploy 部署中使用实例的软件包。

### 启动实例

1. 打开 Amazon EC2 控制台 <https://console.aws.amazon.com/ec2/>。
2. 从控制台控制面板中，选择 Launch Instance。
3. 在 Step 1: Choose an Amazon Machine Image (AMI) 页面上，找到针对 Amazon Linux AMI 的 HVM 版本的行，然后选择 Select。（此 AMI 被标记为“Free tier eligible”，可在列表顶部找到。）

#### Note

这些称为 Amazon 系统映像 (AMI) 的基本配置充当您的实例的模板。可以使用任何符合免费套餐条件的 AMI 完成本教程。为简单起见，我们将使用 Amazon Linux AMI 的 HVM 版本。

4. 在 Step 2: Choose an Instance Type 页面上，选择符合免费套餐条件的 t2.micro 类型作为您的实例的硬件配置，然后选择 Next: Configure Instance Details。
5. 在 Step 3: Configure Instance Details 页面中，执行以下操作：
  - 在 Number of instances 中，输入 1。
  - 在 Auto-assign Public IP 中，选择 Enable。
  - 在 IAM role 中，选择一个 IAM 角色，该角色已被配置为用作与 AWS CodeDeploy 一起使用的 IAM 实例配置文件。如果您没有 IAM 实例配置文件，请选择 Create new IAM role，然后按照[为 Amazon EC2 实例创建 IAM 实例配置文件](#)中的说明操作。

#### Note

在本教程中，您可以为 AWS CodeDeploy 使用您的 IAM 实例配置文件中的以下无限制策略。对于您的开发工作流程中使用的管道，您可以创建一个限制性更强的存储桶策略。

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

```
}  
}
```

- 在 Step 3: Configure Instance Details 页面上，展开 Advanced Details，然后在 User data 字段中，键入以下内容：

```
#!/bin/bash  
yum -y update  
yum install -y ruby  
yum install -y aws-cli  
cd /home/ec2-user  
aws s3 cp s3://aws-coddeploy-us-east-2/latest/install . --region us-east-2  
chmod +x ./install  
./install auto
```

此代码将在创建实例时在实例上安装 AWS CodeDeploy 代理。如果您愿意，您可以在创建实例后[使用 SSH 连接到您的 Linux 实例并手动安装 AWS CodeDeploy 代理](#)。

- 将 Step 3: Configure Instance Details 页面上的剩余项目保留不变。选择 Next: Add Storage，将 Step 4: Add Storage 页面保留不变，然后选择 Next: Add Tags。
- 在 Add Tags 页面 (Name 显示在 Key 框中) 中，在 Value 框中键入 MyCodePipelineDemo，然后选择 Next: Configure Security Group。

#### Important

Key 和 Value 框区分大小写。

- 在 Step 6: Configure Security Group 页面中，执行以下操作：
  - 在 Assign a security group 旁边，选择 Create a new security group。
  - 在针对 SSH 的行中，在 Source 下，选择 My IP。
  - 选择 Add Rule，选择 HTTP，然后在 Source 下选择 My IP。
- 选择 Review and Launch。
- 在 Review Instance Launch 页面中，选择 Launch，然后当提示提供密钥对时执行以下操作：
  - 如果您已经拥有要与 Amazon EC2 实例一起使用的密钥对，请选择 Choose an existing key pair，然后选择您的密钥对。
  - 如果您尚未创建密钥对，请选择 Create a new key pair，为密钥对输入名称，然后选择 Download Key Pair。这是您保存私有密钥文件的唯一机会。请务必下载它。将私有密钥文件保存在安全位置。当您启动实例时，您将需要提供密钥对的名称；当您每次连接到实例时，您将需要提供相应的私有密钥。有关更多信息，请参阅[Amazon EC2 密钥对](#)。

#### Warning

请勿选择 Proceed without a key pair (在没有密钥对的情况下继续) 选项。如果您需要对有关 AWS CodeDeploy 代理出现的问题进行故障排除，而您在没有密钥对的情况下启动实例，则您无法连接到实例。

准备好后，选中确认复选框，然后选择 Launch Instances。

- 选择 View Instances 以关闭确认页面并返回控制台。
- 您可以在 Instances 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 Public DNS 列不显示，请选择 Show/Hide 图标，然后选择 Public DNS。)
- 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 Status Checks 列中查看此信息。

如果要确认 AWS CodeDeploy 代理已正确配置，可以[使用 SSH 连接到您的 Linux 实例](#)，然后验证 AWS CodeDeploy 代理是否正在运行。

## 步骤 4：在 AWS CodeDeploy 中创建应用程序

在 AWS CodeDeploy 中，应用程序是您要部署的代码的标识符，采用名称形式。AWS CodeDeploy 使用此名称来确保在部署期间引用修订、部署配置和部署组的正确组合。在本教程后面创建管道时，您应选择在这一步中创建的 AWS CodeDeploy 应用程序的名称。

在 AWS CodeDeploy 中创建应用程序

1. 打开 AWS CodeDeploy 控制台，网址为：<https://console.aws.amazon.com/codedeploy>。
2. 如果未显示 Applications 页，请在 AWS CodeDeploy 菜单上选择 Applications。
3. 选择 Create application。
4. 在 Application name 框中键入 MyDemoApplication。
5. 在 Deployment group name 框中键入 MyDemoDeploymentGroup。
6. 在 Search by tags 列表中，选择 Amazon EC2 标签类型，选择 Key 框中的 Name，然后在 Value 框中键入 MyCodePipelineDemo。

### Important

在此处，您必须为 Name 密钥选择您创建 Amazon EC2 实例时为该实例分配的那个值。如果您使用 MyCodePipelineDemo 以外的内容标记实例，那么，在这里，请务必使用该内容。

7. 在 Deployment configuration 列表中选择 CodeDeployDefault.OneAtATime。
8. 在 Service role ARN 框中，选择信任 AWS CodeDeploy 的服务角色（至少具有为 [AWS CodeDeploy 创建服务角色](#) 中描述的信任和权限）的 Amazon 资源名称 (ARN)。要获得服务角色 ARN，请参阅 [获得服务角色 ARN \(控制台\)](#)。
9. 选择 Create application。

## 步骤 5：在 AWS CodePipeline 中创建您的第一个管道

现在您已经可以创建并运行您的第一个管道。

创建 AWS CodePipeline 自动发布流程

1. 登录 AWS 管理控制台并通过以下网址打开 AWS CodePipeline 控制台：<http://console.aws.amazon.com/codepipeline>。
2. 在介绍页面上，选择 Get started。

如果您看到 All pipelines 页面，请选择 Create pipeline。

3. 在 Step 1: Name 的 Pipeline name 中，键入 **MyFirstPipeline**，然后选择 Next step。

### Note

如果您为管道选择其他名称，请务必在本教程的剩余步骤中使用该名称代替 **MyFirstPipeline**。当您创建管道后，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 [AWS CodePipeline 中的限制 \(p. 208\)](#)。

4. 在 Step 2: Source 的 Source provider 中，选择 AWS CodeCommit。在 Repository name 中，选择您在 [步骤 1：创建 AWS CodeCommit 存储库和本地存储库 \(p. 36\)](#) 中创建的 AWS CodeCommit 存储库的名称。在 Branch name 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 master 可用。

选择下一步。



**Create pipeline**

Step 1: Name  
**Step 2: Source**  
Step 3: Build  
Step 4: Deploy  
Step 5: Service Role  
Step 6: Review

### Source location ?

Specify where your source code is stored. Choose the provider, and then provide connection details for that provider.

Source provider\* AWS CodeCommit

#### AWS CodeCommit i

Choose a repository and a branch to use as the source location.

Repository name\*  ↺

Branch name\* Choose a repository before choosing a branch. ↺

**i** We will use Amazon CloudWatch Events to detect changes  
This requires AWS CodePipeline to create an Amazon CloudWatch Events rule and an IAM role on your behalf. You can opt-out in the options below.

#### ▼ Change detection options

Using Amazon CloudWatch Events to automatically start your pipeline when a change occurs in the source repository and branch is the default for this source type. When your pipeline is saved via the AWS CodePipeline Console, the corresponding Amazon CloudWatch Events rule and associated IAM role will be created automatically.

If your pipeline is created or updated via other means such as the CLI or AWS CloudFormation, then you will need to create the Amazon CloudWatch Events rule separately.

You can choose to revert to the original method of using AWS CodePipeline to check periodically for changes (not recommended). [Learn More.](#)

☒ Use Amazon CloudWatch Events to automatically start my pipeline when a change occurs (recommended)  
☐ Use AWS CodePipeline to check periodically for changes

\* Required Cancel Previous Next step

在选择存储库名称和分支后，将看到一条消息以显示将为该管道创建的 Amazon CloudWatch Events 规则。

在更改检测选项下面，保留默认值。这样，AWS CodePipeline 就可以使用 Amazon CloudWatch Events 检测源存储库中的更改。

- 在 Step 3: Build 中，选择 No Build，然后选择 Next step。

#### Note

在本教程中，您将部署不需要生成服务的代码。

- 在 Step 4: Deploy 的 Deployment provider 中，选择 AWS CodeDeploy。在 Application name 中，键入 **MyDemoApplication**，或选择 Refresh 按钮，然后从列表中选择应用程序名称。在 Deployment group 中，键入 **MyDemoDeploymentGroup** 或从列表中选择它，然后选择 Next step。

## Create pipeline

- Step 1: Name
- Step 2: Source
- Step 3: Build
- Step 4: Deploy**
- Step 5: Service Role
- Step 6: Review

### Deploy

Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.

Deployment provider\* AWS CodeDeploy

#### AWS CodeDeploy

Choose one of your existing applications, or [create a new one in AWS CodeDeploy](#).

Application name\* MyDemoApplication

Choose one of your existing deployment groups, or [create a new one in AWS CodeDeploy](#).

Deployment group\* MyDemoDeploymentGroup

\* Required

Cancel

Previous

Next step

### Note

名称“Staging”是默认情况下为在 Step 4: Deploy 步骤中创建的阶段提供的名称，正如“Source”是为管道的第一阶段提供的名称一样。

- 在 Step 5: Service Role 中，您将选择用于授予使用您账户中的资源的 AWS CodePipeline 权限的 IAM 角色。只有第一次在 AWS CodePipeline 中创建管道时才需要创建服务角色。

如果您还没有为 AWS CodePipeline 创建服务角色：

- 选择 Create role。
- 在 IAM 控制台页面 (描述将要为您创建的 AWS-CodePipeline-Service 角色) 上，选择 Allow。当您创建角色后，AWS-CodePipeline-Service 将显示在 Step 5: Service Role 页面上的 Role name 中。
- 选择下一步。

如果您已经拥有 AWS CodePipeline 的服务角色，您必须确保它包含与 AWS CodeCommit 一起运行所需的权限。如果您的服务角色是在 2016 年 4 月 18 日之后创建的，则它包含必要的权限。如果它是在 2016 年 4 月 18 日或之前创建的，您可能需要执行以下步骤：

- 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
- 在 IAM 控制台的导航窗格中，选择 Roles，然后从角色列表中选择您的 AWS-CodePipeline-Service 角色。
- 在 Permissions 选项卡上的 Inline Policies 中，选择您的服务角色策略所在行中的 Edit Policy。

### Note

您的服务角色的名称格式类似于 oneClick\_AWS-CodePipeline-1111222233334。

- 在 Policy Document 框中，将以下内容添加到您的策略语句中：

```
{
  "Action": [
    "codecommit:GetBranch",
    "codecommit:GetCommit",
    "codecommit:UploadArchive",
    "codecommit:UploadArchiveToRepository"
  ]
}
```

API 版本: 2015-07-09



```
    "codecommit:CancelUploadArchive"
  ],
  "Resource": "*",
  "Effect": "Allow"
},
```

在您添加这些内容之后，权限可能会显示在您的策略文档中。

```
18     "arn:aws:s3:::elasticbeanstalk*",
19   ],
20   "Effect": "Allow"
21 },
22 {
23   "Action": [
24     "codecommit:GetBranch",
25     "codecommit:GetCommit",
26     "codecommit:UploadArchive",
27     "codecommit:GetUploadArchiveStatus",
28     "codecommit:CancelUploadArchive"
29   ],
30   "Resource": "*",
31   "Effect": "Allow"
32 },
33 {
34   "Action": [
35     "codedeploy:CreateDeployment",
```

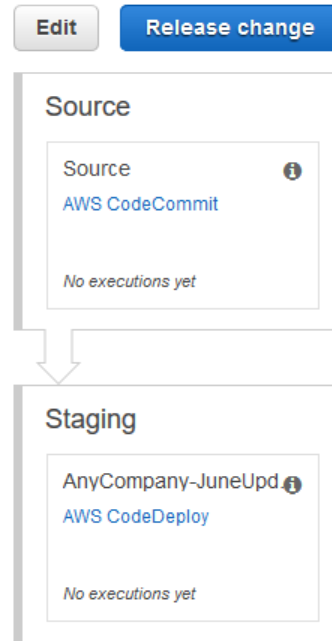
5. 选择 Validate Policy 确保策略不包含错误。当策略正确无误时，选择 Apply Policy。
6. 在 Step 5: Service Role 页面的 Role name 中，为 AWS CodePipeline 选择您的服务角色名称。

由于下拉列表会显示所有与您的账户相关联的 IAM 服务角色，如果您选择一个非默认的名称，请确保该名称可识别为 AWS CodePipeline 的服务角色。

7. 选择下一步。
8. 在 Step 6: Review 中，查看信息，然后选择 Create pipeline。
9. 管道会自动开始运行。在 AWS CodePipeline 示例将网页部署至 AWS CodeDeploy 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。

## AnyCompanyPipeline No history available

View progress and manage your pipeline.

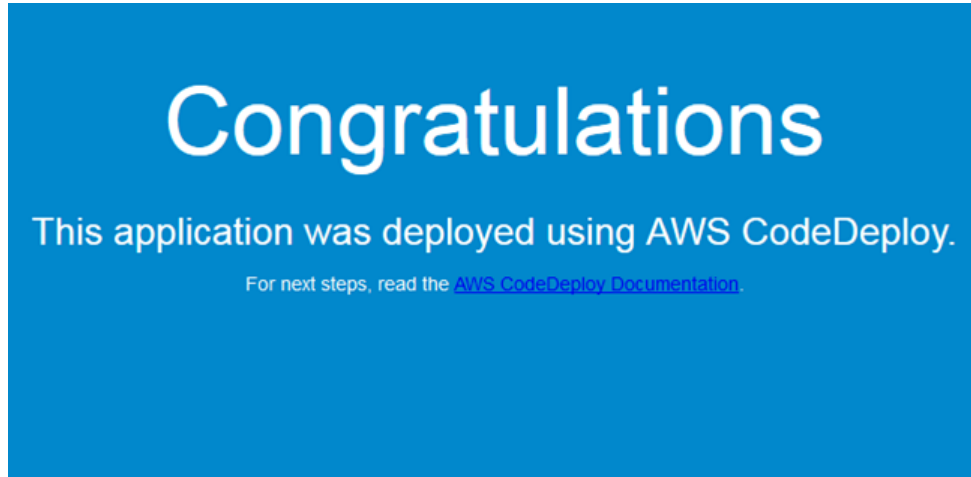


恭喜您！您刚刚在 AWS CodePipeline 中创建了一个简单的管道。该管道有两个阶段：名为 Source 的源阶段，它检测存储在 AWS CodeCommit 存储库中的示例应用程序的更改，并将这些更改提取到管道中，以及 Staging 阶段，它使用 AWS CodeDeploy 将这些更改部署到 Amazon EC2 实例。接下来，您将验证结果。

### 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 No executions yet 变为 In Progress，然后变为 Succeeded 或 Failed。管道将在几分钟内完成首次运行。
2. 当管道状态显示 Succeeded 后，在 Staging 阶段的状态区域中，选择 Details。
3. 在 AWS CodeDeploy 控制台中的 Deployment Details 中，在 Instance ID 中，选择已成功部署的实例的实例 ID。
4. 在 Description 选项卡上的 Public DNS 中，复制地址，然后将其粘贴到 Web 浏览器的地址栏中。

这是您下载并推送到您的 AWS CodeCommit 存储库的示例应用程序。



有关阶段和操作以及管道如何工作的更多信息，请参阅[AWS CodePipeline 概念](#) (p. 3)。

## 步骤 6：修改 AWS CodeCommit 存储库中的代码

在此步骤中，您将更改作为部署到 Amazon EC2 实例的示例 AWS CodeDeploy 应用程序的一部分的 HTML 文件。当您的管道在本教程后面再次运行时，您所做的更改将在 <http://PublicDNS> URL 中可见。

1. 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
```

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 AWS CodePipeline, AWS
CodeCommit, and AWS CodeDeploy.</h2></div>
  <div align="center">
    <p>Learn more:</p>
    <p><a href="http://docs.aws.amazon.com/codepipeline/latest/userguide/">AWS
CodePipeline User Guide</a></p>
    <p><a href="http://docs.aws.amazon.com/codecommit/latest/userguide/">AWS CodeCommit
User Guide</a></p>
    <p><a href="http://docs.aws.amazon.com/codedeploy/latest/userguide/">AWS CodeDeploy
User Guide</a></p>
  </div>
</body>
</html>
```

4. 通过一次运行以下一条命令的方式，将您的更改提交并推送到您的 AWS CodeCommit 存储库中。

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

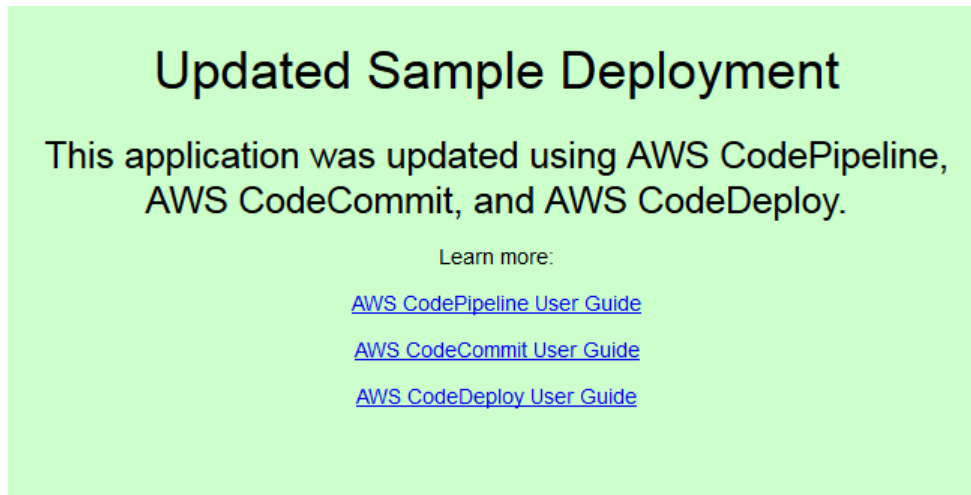
```
git push
```

您的管道配置为每次对 AWS CodeCommit 存储库中的代码进行更改时运行。

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 No executions yet 变为 In Progress，然后变为 Succeeded 或 Failed。管道将在几分钟内完成。
2. 当操作状态显示 Succeeded 后，在 Staging 阶段的状态区域中，选择 Details。
3. 在 Deployment Details 部分，在 Instance ID 中，选择实例的实例 ID。
4. 在 Description 选项卡上的 Public DNS 中，复制地址，然后将其粘贴到 Web 浏览器的地址栏中。

此时将显示更新后的网页：



有关阶段和操作以及管道如何工作的更多信息，请参阅[AWS CodePipeline 概念 \(p. 3\)](#)。

## 步骤 7：可选的阶段管理任务

如果您想在结束本教程之前获得更多阶段使用经验，则可以执行[教程：创建一个简单的管道 \(Amazon S3 存储桶\)](#) (p. 22)中的两个附加步骤。

- [步骤 4：向管道中添加另一个阶段](#) (p. 28)
- [在 AWS CodePipeline 中禁用和启用阶段之间的过渡](#) (p. 34)

### Note

在第二个过程的第 4 步中，不要按照上述说明再次将您的示例上传到 Amazon S3 存储桶，而是对本地存储库中的示例应用程序进行更改，然后再将其推送到 AWS CodeCommit 存储库。

## 步骤 8：清理资源

您可以将在本教程中创建的一些资源用于下一个教程，[教程：创建一个四阶段管道](#) (p. 47)。例如，您可以重用 AWS 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 AWS CodeDeploy 应用程序及其关联的 Amazon EC2 实例，最后删除 AWS CodeCommit 存储库。

清除本教程中使用的资源

1. 要清除您的 AWS CodePipeline 资源，请遵循[删除 AWS CodePipeline 中的资源](#) (p. 101)中的说明。
2. 要清除您的 AWS CodeDeploy 资源，请按照[清除部署演练资源](#)中的说明操作。
3. 要删除 AWS CodeCommit 存储库，请按照[删除 AWS CodeCommit 存储库](#)中的说明操作。

## 教程：创建一个四阶段管道

在[教程：创建一个简单的管道 \(Amazon S3 存储桶\)](#) (p. 22)或[教程：创建一个简单的管道 \(AWS CodeCommit 存储库\)](#) (p. 36)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。本教程将引导您创建一个四阶段管道，该管道使用 GitHub 存储库作为源位置，使用 Jenkins 生成服务器来生成项目，并使用 AWS CodeDeploy 应用程序将内置代码部署到暂存服务器。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，如果要对源代码使用 GitHub 存储库，则必须先创建该存储库，然后才能将其添加到管道中。作为设置的一部分，本教程将引导您在 Amazon EC2 实例上设置 Jenkins 以进行演示。

在开始本教程之前，您应该已经完成了[AWS CodePipeline 入门](#) (p. 8)中的一般先决条件。

### 主题

- [步骤 1：设置先决条件](#) (p. 47)
- [步骤 2：在 AWS CodePipeline 中创建管道](#) (p. 50)
- [步骤 3：向管道中添加另一个阶段](#) (p. 51)
- [步骤 4：清理资源](#) (p. 53)

## 步骤 1：设置先决条件

要与 Jenkins 集成，AWS CodePipeline 需要您在要与 AWS CodePipeline 一起使用的任何 Jenkins 实例上安装 AWS CodePipeline Plugin for Jenkins。您还应该配置一个专用 IAM 用户，以用于 Jenkins 项目和

AWS CodePipeline 之间的权限。集成 Jenkins 和 AWS CodePipeline 的最简单方法是将 Jenkins 安装在您为 Jenkins 集成创建的 IAM 实例角色的 Amazon EC2 实例上。为了使管道中 Jenkins 操作的链接成功连接，您必须在服务器或 Amazon EC2 实例上配置代理和防火墙设置，以允许 Jenkins 项目使用的端口的入站连接。确保您已将 Jenkins 配置为对用户进行身份验证并强制执行访问控制，然后再允许这些端口上的连接 (例如，如果您已对 Jenkins 进行了保护以便仅使用 HTTPS 连接，则为 443 和 8443，如果允许使用 HTTP 连接，则为 80 和 8080)。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。您可以按照本主题中的步骤从 GitHub 存储库下载开源示例代码。您需要 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 \(p. 48\)](#)
- [创建 IAM 角色以用于 Jenkins 集成 \(p. 48\)](#)
- [安装并配置 Jenkins 和 AWS CodePipeline Plugin for Jenkins \(p. 49\)](#)

## 将示例复制或克隆到 GitHub 存储库

克隆示例，并将其推送到 GitHub 存储库

1. 从 GitHub 存储库中下载示例代码，或者将存储库克隆到您本地的电脑。有两个示例包：
  - 如果您要将示例部署到 Amazon Linux、RHEL 或 Ubuntu Server 实例，请选择 [aws-codepipeline-jenkins-aws-codedeploy\\_linux.zip](#)。
  - 如果您要将示例部署到 Windows Server 实例，请选择 [AWSCodePipeline-Jenkins-AWSCodeDeploy\\_Windows.zip](#)。
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，请考虑启动 Amazon EC2 实例来托管 Jenkins 服务器，并使用 IAM 角色向实例授予与 AWS CodePipeline 交互所需的权限。

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
2. 在 IAM 控制台中，在导航窗格中，选择 Roles，然后选择 Create new role。
3. 在 Select role type 页中，在选中 AWS Service Role 的情况下，在 Amazon EC2 旁边选择 Select。
4. 在 Attach Policy 页中，选择 AWSCodePipelineCustomActionAccess 托管策略，然后选择 Next Step。
5. 在 Set role name and review 页中，在 Role name 框中，键入您专为 Jenkins 集成创建的角色名称 (例如 `JenkinsAccess`)，然后选择 Create role。

当您创建将在其中安装 Jenkins 的 Amazon EC2 实例时，请在步骤 3：配置实例详细信息中确保选择该实例角色 (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 角色为在 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](#)。

## 安装并配置 Jenkins 和 AWS CodePipeline Plugin for Jenkins

安装 Jenkins 和 AWS CodePipeline Plugin for Jenkins

1. 创建将在其中安装 Jenkins 的 Amazon EC2 实例，在 Step 3: Configure Instance Details 中确保您选择自己所创建的实例角色 (例如 **JenkinsAccess**)。有关创建 Amazon EC2 实例的更多信息，请参阅 [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 [AWS CodePipeline 故障排除](#) (p. 165)。

2. 在 Amazon 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[AWS CodePipeline 产品和服务集成](#) (p. 10)中的 [details of integration with Jenkins](#) (p. 12)。
3. 启动 Jenkins，然后在主页上，选择 Manage Jenkins。
4. 在 Manage Jenkins 页中，选择 Manage Plugins。
5. 选择 Available 选项卡，然后在 Filter 搜索框中键入 **AWS CodePipeline**。从列表中选择 AWS CodePipeline Plugin for Jenkins，然后选择 Download now and install after restart。
6. 在 Installing Plugins/Upgrades 页中，选择 Restart Jenkins when installation is complete and no jobs are running。
7. 选择 Back to Dashboard。
8. 在主页中，选择 New Item。
9. 在 Item Name 上，键入 Jenkins 项目的名称 (例如，**MyDemoProject**)。选择 Freestyle project，然后选择 OK。

### Note

确保您的项目的名称符合 AWS CodePipeline 要求。有关更多信息，请参阅 [AWS CodePipeline 中的限制](#) (p. 208)。

10. 在项目的配置页中，选择 Execute concurrent builds if necessary 复选框。在 Source Code Management 中，选择 AWS CodePipeline。如果您在 Amazon EC2 实例上安装了 Jenkins，并且使用您为 AWS CodePipeline 与 Jenkins 之间的集成创建的 IAM 用户的配置文件配置了 AWS CLI，请将其其他所有字段留空。
11. 选择 Advanced，然后在 Provider 中，按照 AWS CodePipeline 中显示的内容键入操作的提供程序名称 (例如，**MyJenkinsProviderName**)。确保此名称是唯一的，且是容易记住的。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 AWS CodePipeline 中的操作的命名要求。有关更多信息，请参阅 [AWS CodePipeline 中的限制](#) (p. 208)。

12. 在 Build Triggers 中，清除所有复选框，然后选择 Poll SCM。在 Schedule 中，键入五个以空格分隔的星号，如下所示：

```
* * * * *
```

这会每分钟轮询一次 AWS CodePipeline。



13. 在 Build 中，选择 Add build step。选择 Execute shell (Amazon Linux、RHEL 或 Ubuntu Server) 或 Execute batch command (Windows Server)，然后键入以下内容：

```
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 Add post-build action，然后选择 AWS CodePipeline Publisher。选择 Add，然后在 Build Output Locations 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 Save 以保存您的 Jenkins 项目。

## 步骤 2：在 AWS CodePipeline 中创建管道

在本教程中，您将使用 Create Pipeline 向导创建管道。

### 创建 AWS CodePipeline 自动发布流程

1. 登录 AWS 管理控制台并通过以下网址打开 AWS CodePipeline 控制台：<http://console.aws.amazon.com/codepipeline>。
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 AWS CodePipeline 可用的区域和终端节点的更多信息，请参阅[区域和终端节点](#)。

3. 在 All Pipelines 页中，选择 Create pipeline。

#### Note

如果您看到 AWS CodePipeline 起始页，请选择 Get started。

4. 在 Step 1: Name 的 Pipeline name 中，键入 **MySecondPipeline**，然后选择 Next step。

#### Note

如果您为管道选择另一个名称，请务必使用此名称替换此教程其他部分的 **MySecondPipeline**。当您创建管道后，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[限制](#) (p. 208)。

5. 在 Step 2: Source 的 Source provider 中，选择 GitHub，然后选择 Connect to GitHub。这将打开一个新的浏览器窗口，让您连接到 GitHub。如果系统提示您登录，请提供您的 GitHub 凭证。

#### Important

不要在 GitHub 网站上提供您的 AWS 凭证。

选择 GitHub 后，系统将显示一条消息，建议 AWS CodePipeline 在 GitHub 中为您的管道创建 Webhook。

连接到 GitHub 后，请选择您要用于此教程的示例推送到的存储库和分支 (aws-codepipeline-jenkins-aws-codedeploy\_linux.zip 或 AWSCodePipeline-Jenkins-AWSCodeDeploy\_Windows.zip)，然后选择 Next step。

#### Note

在 GitHub 中，可用于应用程序 (如 AWS CodePipeline) 的 OAuth 令牌的数量有限。如果您超过此限制，请重试连接以允许 AWS CodePipeline 通过重用现有令牌来重新连接。有关更多信息，请参阅[配置管道以使用 GitHub 的个人访问令牌](#) (p. 167)。

6. 在 Step 3: Build 中，选择 Add Jenkins。在 Provider name 中，键入您在 AWS CodePipeline Plugin for Jenkins 中提供的操作名称 (例如，**MyJenkinsProviderName**)。此名称必须与 AWS CodePipeline



Plugin for Jenkins 中的名称完全匹配。在 Server URL 中，键入 Jenkins 安装到的 Amazon EC2 实例的 URL。在 Project name 中，键入您在 Jenkins 中创建的项目名称，如 *MyDemoProject*，然后选择 Next step。

- 在 Step 4: Deploy 中，再次使用您在 [教程：创建一个简单的管道 \(Amazon S3 存储桶\) \(p. 22\)](#) 中创建的 AWS CodeDeploy 应用程序和部署组。在 Deployment provider 中，选择 AWS CodeDeploy。在 Application name 中，键入 **CodePipelineDemoApplication**，或选择刷新按钮，然后从列表中选择应用程序名称。在 Deployment group 中，键入 **CodePipelineDemoFleet** 或从列表中选择，然后选择 Next step。

#### Note

您可以使用自己的 AWS CodeDeploy 资源或者创建新的资源，但您可能会产生额外的成本。

- 在 Step 5: Service Role 中，从 Role name 中，选择您为 AWS CodePipeline 创建的服务角色 (例如，AWS-CodePipeline-Service)，然后选择 Next step。

#### Note

只有第一次在 AWS CodePipeline 中创建管道时才需要创建服务角色。如果您按照简单管道教程之一中的步骤操作，则您已经创建了此服务角色，您将能够从下拉列表中选择此服务角色。由于下拉列表中会显示与您的账户相关联的所有 IAM 服务角色，如果您已经选定了一个非默认的名称，则请选择该名称。如果您未创建服务角色，请选择 Create role。  
如果您使用的是 AWS CodeCommit 存储库，而不是 GitHub 存储库，并且使用的服务角色是 2016 年 4 月 18 日前创建的，请确定其包括访问 AWS CodeCommit 所需的权限。有关更多信息，请参阅 [添加其他 AWS 服务的权限 \(p. 182\)](#)。

- 在 Step 6: Review 中，查看信息，然后选择 Create pipeline。
- 管道会自动启动示例并通过管道运行示例。当管道将 HamI 示例生成为 HTML，并以网页形式将其部署至 AWS CodeDeploy 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。

## 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 \(p. 51\)](#)
- [创建一个 Jenkins 项目以测试部署 \(p. 52\)](#)
- [创建第四个阶段 \(p. 53\)](#)

### 查找实例的 IP 地址

验证您将代码部署到的实例的 IP 地址

- 当管道状态显示为 Succeeded 后，在 Staging 阶段的状态区域中，选择 Details。
- 在 Deployment Details 部分，在 Instance ID 中，选择成功部署的一个实例的实例 ID。
- 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 项目以测试部署

### 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 New Item。
2. 在 Item Name 中，键入 Jenkins 项目的名称 (例如，*MyTestProject*)。选择 Freestyle project，然后选择 OK。

#### Note

确保您的项目命名符合 AWS CodePipeline 要求。有关更多信息，请参阅 [AWS CodePipeline 中的限制 \(p. 208\)](#)。

3. 在项目的配置页中，选择 Execute concurrent builds if necessary 复选框。在 Source Code Management 中，选择 AWS CodePipeline。如果您在 Amazon EC2 实例上安装了 Jenkins，并且使用您为 AWS CodePipeline 与 Jenkins 之间的集成创建的 IAM 用户的配置文件配置了 AWS CLI，请将其所有字段留空。

#### Important

如果您要配置 Jenkins 项目，但 Amazon EC2 实例上未安装此 Jenkins 项目，或者该项目安装在了运行 Windows 操作系统的 Amazon EC2 实例上，请按照您的代理主机和端口设置的要求填写字段，并提供您为 Jenkins 和 AWS CodePipeline 之间的集成而配置的 IAM 用户凭证。

4. 选择 Advanced，然后在 Category 中，选择 Test。
5. 在 Provider 中，键入您为生成项目所使用的相同名称 (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 AWS CodePipeline 命名要求。有关更多信息，请参阅 [AWS CodePipeline 中的限制 \(p. 208\)](#)。

6. 在 Build Triggers 中，清除所有复选框，然后选择 Poll SCM。在 Schedule 中，键入五个以空格分隔的星号，如下所示：

```
* * * * *
```

这会每分钟轮询一次 AWS CodePipeline。

7. 在 Build 中，选择 Add build step。如果您要部署到 Amazon Linux、RHEL 或 Ubuntu Server 实例，请选择 Execute shell，然后键入以下内容，其中 IP 地址是指您前面复制的 Amazon 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 Execute batch command，然后键入以下内容，其中 IP 地址是指您前面复制的 Amazon 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 Add post-build action，然后选择 AWS CodePipeline Publisher。不要选择 Add。
9. 选择 Save 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 管理控制台并通过以下网址打开 AWS CodePipeline 控制台：<http://console.aws.amazon.com/codepipeline>。
2. 在 Name 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 Edit。
4. 在 Edit 页中，选择 + Stage 以紧随 Staging 阶段之后添加一个阶段。
5. 在新阶段的名称字段中，键入名称 (例如，**Testing**)，然后选择 + Action。
6. 在 Action category 下拉列表中，选择 Test。在 Action name 中，键入 **MyJenkinsTest-Action**。在 Test provider 中，选择您在 Jenkins 中指定的提供程序名称 (例如，**MyJenkinsProviderName**)。在 Project name 中，键入您在 Jenkins 中创建的项目名称 (例如，**MyTestProject**)。在 Input artifacts 中，从默认名称为 **MyBuiltApp** 的 Jenkins 生成中选择项目，然后选择 Add action。

有关输入和输出项目以及管道结构的更多信息，请参阅[AWS CodePipeline 管道结构参考 \(p. 202\)](#)。

7. 在 Edit 页中，选择 Save pipeline changes。在 Save pipeline changes 对话框中，选择 Save and continue。
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 AWS CodePipeline，请删除管道，然后删除 AWS 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。您还应考虑是否删除其他不打算继续使用的资源，例如 GitHub 存储库。

清除本教程中使用的资源

1. 在您本地的 Linux, macOS, or 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**，您可以键入以下命令：

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

该命令不返回任何内容。

2. 要清理您的 AWS CodeDeploy 资源，请按照[清理](#)中的说明进行操作。
3. 要清理您的实例资源，请删除您将 Jenkins 安装到的 Amazon EC2 实例。有关更多信息，请参阅[清理您的实例和卷](#)。
4. 如果您不打算创建更多的管道或再次使用 AWS CodePipeline，请删除用于为您的管道存储项目的 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，如果您想删除 GitHub 存储库，请按照 GitHub 网站上 [Deleting a repository](#) 中的说明进行操作。

# 教程：设置 CloudWatch Events 规则以接收管道状态更改的电子邮件通知

在设置管道后，您可以设置一个 CloudWatch Events 规则，以便在管道或管道中的阶段或操作的执行状态发生更改时发送通知。有关使用 CloudWatch Events 设置管道状态更改通知的更多信息，请参阅[使用 Amazon CloudWatch Events 检测和响应管道状态更改](#) (p. 153)。

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。在创建 CloudWatch Events 规则时，本教程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。

## 主题

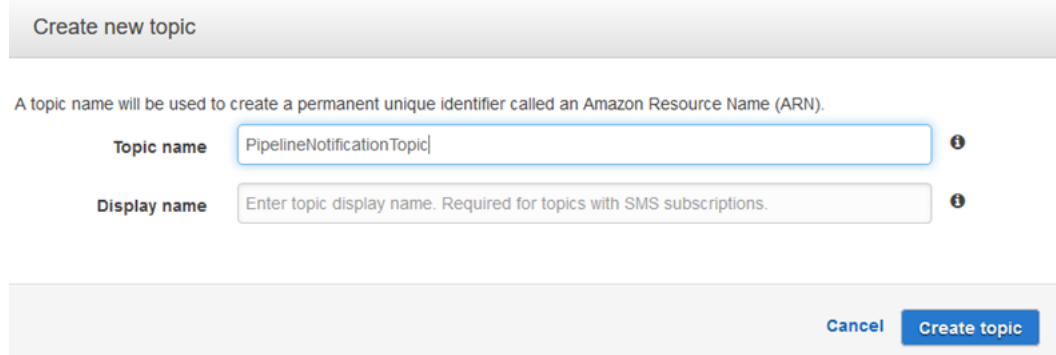
- [步骤 1：使用 Amazon SNS 设置电子邮件通知](#) (p. 54)
- [步骤 2：创建规则并将 SNS 主题添加为目标](#) (p. 55)
- [步骤 3：清除资源](#) (p. 57)

## 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终端节点或客户端。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 CloudWatch Events 规则中。有关更多信息，请参阅[Amazon Simple Notification Service 开发人员指南](#)。

1. 在 Amazon SNS 中创建或标识主题。AWS CodePipeline 将使用 CloudWatch Events 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 Amazon SNS 控制台：<https://console.aws.amazon.com/sns>。
2. 选择 Create topic。
3. 在创建新主题对话框中，为主题名称键入主题名（例如，**PipelineNotificationTopic**）。



Create new topic

A topic name will be used to create a permanent unique identifier called an Amazon Resource Name (ARN).

Topic name

Display name

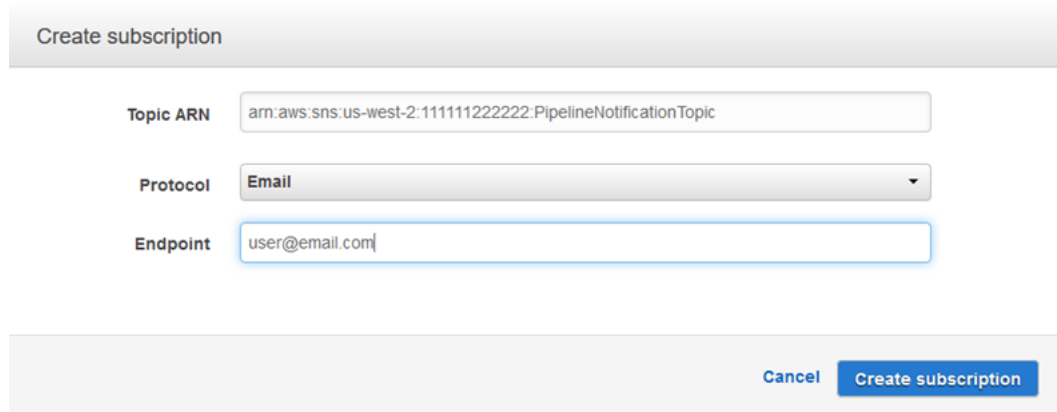
Cancel Create topic

4. 选择 Create topic。

有关更多信息，请参阅 Amazon SNS 开发人员指南中的[创建主题](#)。

2. 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 Protocol，选择 Email。
4. 对于终端节点，请键入收件人的完整电子邮件地址。将结果与以下内容进行比较：



Create subscription

Topic ARN

Protocol

Endpoint

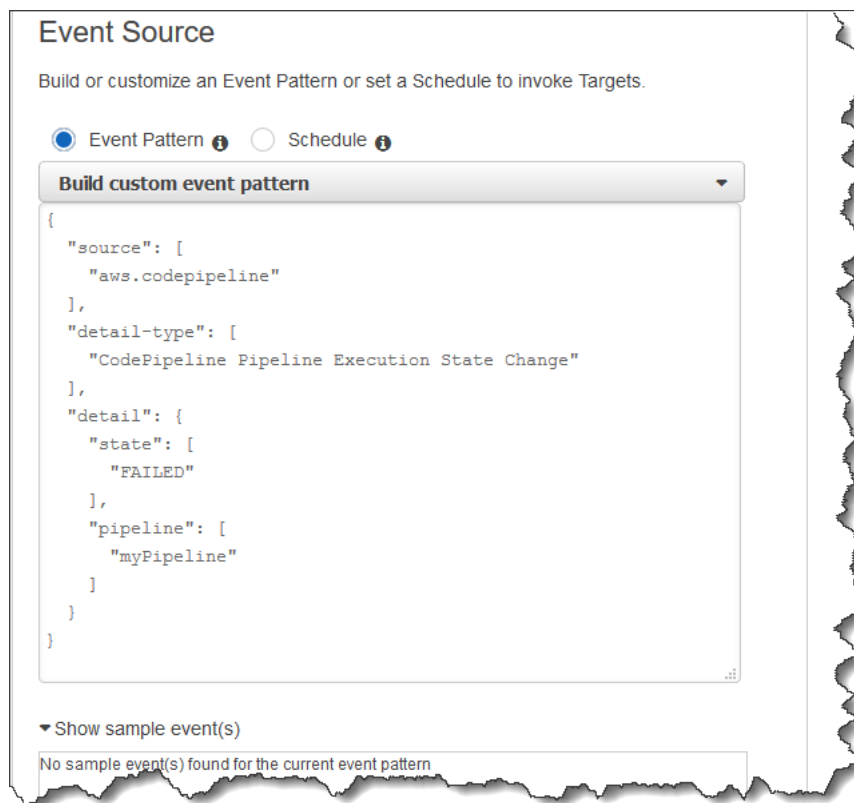
5. 选择 Create Subscription。
6. Amazon SNS 向收件人发送订阅确认电子邮件。要接收电子邮件通知，收件人必须在该电子邮件中选择确认订阅链接。在收件人单击该链接后，如果成功订阅，Amazon SNS 将在收件人的 Web 浏览器中显示一条确认消息。

有关更多信息，请参阅 Amazon SNS 开发人员指南中的[订阅主题](#)。

## 步骤 2：创建规则并将 SNS 主题添加为目标

创建 CloudWatch Events 通知规则并将 AWS CodePipeline 作为事件源。

1. 通过以下网址打开 CloudWatch 控制台：<https://console.aws.amazon.com/cloudwatch/>。
2. 在导航窗格中，选择 Events。
3. 选择 Create rule。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。



6. 对于 Targets，选择 Add target。
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。
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 Create rule。
13. 确认 AWS CodePipeline 立即发送生成通知。例如，检查在您的收件箱中现在是否具有生成通知电子邮件。
14. 要更改规则的行为，请在 CloudWatch 控制台中选择该规则，选择操作，然后选择编辑。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 控制台中选择该规则，选择操作，然后选择禁用。

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

## 步骤 3：清除资源

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。

有关如何清除 SNS 通知并删除 Amazon CloudWatch Events 规则的信息，请参阅[清除 \(取消订阅 Amazon SNS 主题\)](#) 和 [Amazon CloudWatch Events API 参考](#) 中的 DeleteRule 参考。



# AWS CodePipeline 最佳实践和使用案例

AWS CodePipeline 与多种产品和服务集成在一起。以下几节介绍了 AWS CodePipeline 以及相关产品和服务的最佳实践和使用案例。

AWS CodePipeline 的简单商用案例可以帮助您了解实施服务和控制用户访问的方法。这些使用案例是使用通用术语描述的。它们并未规定用于获得所需结果的 API。

主题

- [最佳实践 \(p. 58\)](#)
- [AWS CodePipeline 使用案例 \(p. 59\)](#)

## 最佳实践

在使用 AWS CodePipeline 时使用这些部分中介绍的最佳实践。

### AWS CodePipeline 资源的安全最佳实践

您可以对连接到管道的源存储库使用加密和身份验证。这些是 AWS CodePipeline 的安全性最佳实践：

- 如果您创建使用 Amazon S3 源存储桶的管道，请通过管理 AWS KMS 托管密钥 (SSE-KMS)，为适用于 AWS CodePipeline 的 Amazon S3 中存储的项目配置服务器端加密，如[为适用于 AWS CodePipeline 的 Amazon S3 中存储的项目配置服务器端加密 \(p. 195\)](#)中所述。
- 如果创建一个使用 GitHub 源存储库的管道，请配置 GitHub 身份验证。您可以使用 AWS 托管的 OAuth 令牌或客户管理的个人访问令牌，如[配置 GitHub 身份验证 \(p. 197\)](#)中所述。

### AWS CodePipeline 资源的监控和日志记录最佳实践

您可以使用 AWS 中的日志记录功能来确定用户在您的账户中进行了哪些操作，以及使用了哪些资源。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 可以用于记录由某个 AWS 账户发出或代表该账户发出的 AWS API 调用和相关事件。有关更多信息，请参阅[使用 AWS CloudTrail 记录 AWS CodePipeline API 调用 \(p. 160\)](#)。
- Amazon CloudWatch Events 可以用于监控您的 AWS 云资源以及您在 AWS 上运行的应用程序。您可以在 Amazon CloudWatch Events 中基于您定义的指标创建警报。有关更多信息，请参阅[使用 Amazon CloudWatch Events 检测和响应管道状态更改 \(p. 153\)](#)。



## Jenkins 插件的最佳实践

将本节中提供的关于管道的最佳实践与 Jenkins 操作提供程序结合使用。

### 为您的 Jenkins 构建服务器配置单独的 Amazon EC2 实例和 IAM 角色

作为最佳实践，当您使用 Jenkins 构建提供程序用于管道的构建或测试操作时，应在 Amazon EC2 实例上安装 Jenkins 并配置单独的 EC2 实例配置文件。确保该实例配置文件仅授予 Jenkins 为您的项目执行任务所需的 AWS 权限，如从 Amazon S3 检索文件。

该实例配置文件提供在 Amazon EC2 实例上运行并具有访问其他 AWS 服务的凭证的应用程序。因此，您不需要配置 AWS 凭证（AWS 访问密钥和私有密钥）。

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](#) (p. 48) 中的步骤。

## AWS CodePipeline 使用案例

您可以创建与其他 AWS 服务集成的管道。它们可以是 AWS 服务 (如 Amazon S3) 或第三方产品 (如 GitHub)。此部分提供一些示例，说明如何使用 AWS CodePipeline 来使用不同的产品集成自动执行代码发布。有关按操作类型组织的与 AWS CodePipeline 集成的完整列表，请参阅[AWS CodePipeline 管道结构参考](#) (p. 202)。

#### 主题

- 将 AWS CodePipeline 与 Amazon S3、AWS CodeCommit 和 AWS CodeDeploy 一起使用 (p. 59)
- 将 AWS CodePipeline 与第三方行动提供程序 (GitHub 和 Jenkins) 一起使用 (p. 60)
- 将 AWS CodePipeline 与 AWS CodeStar 一起使用在代码项目中生成管道 (p. 60)
- 使用 AWS CodePipeline 通过 AWS CodeBuild 编译、生成和测试代码 (p. 60)
- 将 AWS CodePipeline 与 Amazon ECS 一起使用以便将基于容器的应用程序连续交付到云 (p. 60)
- 将 AWS CodePipeline 与 Elastic Beanstalk 一起使用以便将 Web 应用程序连续交付到云 (p. 61)
- 将 AWS CodePipeline 与 AWS Lambda 一起使用可连续交付基于 Lambda 的应用程序和无服务器应用程序 (p. 61)
- 将 AWS CodePipeline 与 AWS CloudFormation 模板一起使用以便连续交付到云 (p. 61)

### 将 AWS CodePipeline 与 Amazon S3、AWS CodeCommit 和 AWS CodeDeploy 一起使用

当您创建管道时，AWS CodePipeline 与在管道的每个阶段充当操作提供程序的 AWS 产品和服务集成。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 \(Amazon S3 存储桶\) \(p. 22\)](#) 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。此教程创建一个使用 AWS CodeDeploy 的管道，以将示例应用程序从 Amazon S3 存储桶部署到运行 Amazon Linux 的 Amazon EC2 实例。
- [教程：创建一个简单的管道 \(AWS CodeCommit 存储库\) \(p. 36\)](#) 中的步骤可帮助您使用向导创建一个具有“Source”阶段的管道，它使用您的 AWS CodeCommit 存储库作为源提供程序。此教程创建一个使用 AWS CodeDeploy 的管道，以将示例应用程序从 AWS CodeCommit 存储库部署到运行 Amazon Linux 的 Amazon EC2 实例。

## 将 AWS CodePipeline 与第三方行动提供程序 (GitHub 和 Jenkins) 一起使用

您可以创建与第三方产品 (如 GitHub 和 Jenkins) 集成的管道。[教程：创建一个四阶段管道 \(p. 47\)](#) 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，
- 使用 AWS CodeDeploy 将生成和测试过的源代码部署到运行 Amazon Linux 或 Microsoft Windows Server 的 Amazon EC2 实例。

## 将 AWS CodePipeline 与 AWS CodeStar 一起使用在代码项目中生成管道

AWS CodeStar 是基于云的服务，为管理 AWS 上的软件开发项目提供统一用户界面。AWS CodeStar 与 AWS CodePipeline 协作将 AWS 资源合并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、生成规范文件、部署方法以及托管实例或无服务器实例。

要创建 AWS CodeStar 项目，请选择您的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。

您可以随时将您的首选 IDE 集成到 AWS CodeStar 控制面板。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。有关如何使用 AWS CodeStar 为无服务器应用程序创建示例管道的教程，请参阅[教程：在 AWS CodeStar 中创建和管理无服务器项目](#)。

## 使用 AWS CodePipeline 通过 AWS CodeBuild 编译、生成和测试代码

AWS CodeBuild 是一项在云中的托管生成服务，可让您在无需服务器或系统的情况下构建生成和测试代码。将 AWS CodePipeline 与 AWS CodeBuild 一起使用可在源代码发生更改时，通过管道自动运行修订，以便连续交付软件生成。有关更多信息，请参阅[将 AWS CodePipeline 与 AWS CodeBuild 一起使用来测试代码和运行生成任务](#)。

## 将 AWS CodePipeline 与 Amazon ECS 一起使用以便将基于容器的应用程序连续交付到云

Amazon ECS 是一种容器管理服务，可让您在 ECS 云中将基于容器的应用程序部署到 Amazon ECS 实例。将 AWS CodePipeline 与 Amazon ECS 一起使用可在源映像存储库发生更改时，通过管道自动运行修订，以便连续交付部署基于容器的应用程序。有关更多信息，请参阅[教程：使用 AWS CodePipeline 进行持续部署](#)。

## 将 AWS CodePipeline 与 Elastic Beanstalk 一起使用以便将 Web 应用程序连续交付到云

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。将 AWS CodePipeline 与 Elastic Beanstalk 一起使用可将 Web 应用程序连续部署到您的应用程序环境。您还可以使用 AWS CodeStar 通过 Elastic Beanstalk 部署操作创建管道。

## 将 AWS CodePipeline 与 AWS Lambda 一起使用可连续交付基于 Lambda 的应用程序和无服务器应用程序

您可以将 AWS Lambda 与 AWS CodePipeline 一起使用来调用 AWS Lambda 函数，如[基于 Lambda 应用程序的自动化部署](#)中所述。您还可以使用 AWS Lambda 和 AWS CodeStar 创建一个用于部署无服务器应用程序的管道。

## 将 AWS CodePipeline 与 AWS CloudFormation 模板一起使用以便连续交付到云

您可以将 AWS CloudFormation 与 AWS CodePipeline 一起使用以进行持续交付和自动化。有关更多信息，请参阅[使用 AWS CodePipeline 进行持续交付](#)。AWS CloudFormation 还用于为在 AWS CodeStar 中创建的管道创建模板。

# 使用 AWS CodePipeline 中的管道

要定义自动发布流程，您需要创建一个管道，它是一种工作流程结构，描述软件更改如何经过发布过程。管道由您配置的阶段和操作组成。

## Note

当您添加生成、部署、测试或调用阶段以补充 AWS CodePipeline 提供的默认选项时，可以选择已创建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 AWS CodePipeline 中创建和添加自定义操作 \(p. 111\)](#)。

在创建管道之前，您必须先完成[AWS CodePipeline 入门 \(p. 8\)](#)中的步骤。

有关管道的更多信息，请参阅[AWS CodePipeline 概念 \(p. 3\)](#)、[AWS CodePipeline 教程 \(p. 22\)](#)；如果您想使用 AWS CLI 创建管道，请参阅[AWS CodePipeline 管道结构参考 \(p. 202\)](#)。要查看管道列表，请参阅[在 AWS CodePipeline 中查看管道详细信息和历史记录 \(p. 96\)](#)。

## 主题

- [在 AWS CodePipeline 中启动管道执行 \(p. 62\)](#)
- [在 AWS CodePipeline 中创建管道 \(p. 82\)](#)
- [在 AWS CodePipeline 中编辑管道 \(p. 90\)](#)
- [在 AWS CodePipeline 中查看管道详细信息和历史记录 \(p. 96\)](#)
- [在 AWS CodePipeline 中删除管道 \(p. 101\)](#)
- [在 AWS CodePipeline 中创建使用另一个 AWS 账户的资源的管道 \(p. 102\)](#)

## 在 AWS CodePipeline 中启动管道执行

在启动管道执行时，修订将经过管道中的每个阶段和操作。

可以使用两种方法启动管道执行：

- 自动：通过使用指定的更改检测方法，您可以将管道配置为在对存储库进行更改时自动启动。您还可以将管道配置为按计划自动启动。下面是自动更改检测方法：
  - 当您使用控制台创建具有 AWS CodeCommit 源存储库或 Amazon S3 源存储桶的管道时，AWS CodePipeline 将创建 Amazon CloudWatch Events 规则，此规则将在源发生更改时启动您的管道。这是推荐的更改检测方法。如果您使用 AWS CLI 创建管道，更改检测方法默认为通过定期检查源来自动启动管道。建议禁用定期检查并手动创建其他资源。有关更多信息，请参阅 [使用 CloudWatch Events 规则自动启动 AWS CodeCommit 管道 \(p. 64\)](#)。
  - 当您使用控制台创建具有 GitHub 存储库的管道时，AWS CodePipeline 会创建 Webhook，以便在源发生更改时启动您的管道。这是推荐的更改检测方法。如果您使用 AWS CLI 创建管道，更改检测方法默认为通过定期检查源来自动启动管道。建议禁用定期检查并手动创建其他资源。有关更多信息，请参阅 [使用 Webhook 自动启动 GitHub 管道 \(p. 75\)](#)。
- 手动：您可以使用控制台或 AWS CLI 手动启动管道，如[在 AWS CodePipeline 中手动启动管道 \(p. 80\)](#)中所述。

默认情况下，管道配置为使用更改检测方法自动启动。

## Note

只有在您已定义的源存储库和分支发生更改时，才会运行您的管道。

### 主题

- [用于自动启动管道的更改检测方法 \(p. 63\)](#)
- [使用 CloudWatch Events 规则自动启动 AWS CodeCommit 管道 \(p. 64\)](#)
- [使用 CloudWatch Events 规则自动启动 Amazon S3 管道 \(p. 69\)](#)
- [使用 Webhook 自动启动 GitHub 管道 \(p. 75\)](#)
- [使用定期检查自动启动管道 \(p. 79\)](#)
- [在 AWS CodePipeline 中手动启动管道 \(p. 80\)](#)
- [使用 Amazon CloudWatch Events 按计划自动启动管道 \(p. 80\)](#)

## 用于自动启动管道的更改检测方法

在创建或更新管道时，您可以指定用于响应源存储库更改的更改检测方法。您选择的方法决定了如何启动管道。

来源	检测方法	描述	需要其他资源？	设置管道的方法
Amazon S3	Amazon CloudWatch Events (建议). 对于在控制台中创建或编辑的 S3 管道，这是默认值。	<ul style="list-style-type: none"><li>在对存储库进行更改时，将立即触发管道。存储桶中的事件由 AWS CloudTrail 进行筛选，之后 Amazon CloudWatch Events 规则将触发您的管道以启动该管道。</li><li>比定期检查更快，并且更易于配置。</li></ul>	是。必须应用 Amazon CloudWatch Events 规则和 AWS CloudTrail 跟踪。	<a href="#">控制台 (p. 91)</a> 、 <a href="#">CLI (p. 93)</a> 、 <a href="#">CloudForm</a>
	定期检查	AWS CodePipeline 定期与源联系。	否	
AWS CodeCommit	Amazon CloudWatch Events (建议). 对于在控制台中创建或编辑的 AWS CodeCommit 管道，这是默认值。	<ul style="list-style-type: none"><li>在对存储库进行更改时，将立即触发管道。</li><li>比定期检查更快，并且更易于配置。</li></ul>	是。必须应用 Amazon CloudWatch Events 规则。	<a href="#">控制台 (p. 91)</a> 、 <a href="#">CLI (p. 93)</a> 、 <a href="#">CloudForm</a>
	定期检查	AWS CodePipeline 定期连接到源存储库。	否	
GitHub	Webhook (推荐). 对于在控制台中创建或编辑的 GitHub 管道，这是默认值。	<ul style="list-style-type: none"><li>在对存储库进行更改时，将立即触发管道。</li><li>比定期检查更快，并且更易于配置。</li></ul>	是	<a href="#">控制台 (p. 91)</a> 、 <a href="#">CLI (p. 93)</a> 、 <a href="#">CloudForm</a>  <b>Note</b>  有关 AWS CloudFormation 中 webhook 资源的信
	定期检查	AWS CodePipeline 定期连接到源存储库。	否	

来源	检测方法	描述	需要其他资源？	设置管道的方法
				息，请参阅 <a href="#">AWS::CodePipeline::Webhook</a>

## 使用 CloudWatch Events 规则自动启动 AWS CodeCommit 管道

您可以使用 Amazon CloudWatch Events 触发管道以在符合规则或计划条件时自动启动。对于具有 Amazon S3 或 AWS CodeCommit 源的管道，Amazon CloudWatch Events 规则将检测源更改，然后启动您的管道。在使用控制台创建或更改管道时，将为您创建规则和相关联资源。如果您在 CLI 或 AWS CloudFormation 中创建或更改 Amazon S3 或 AWS CodeCommit 管道，则必须使用以下步骤手动创建 Amazon CloudWatch Events 规则和相关联资源。

在 Amazon CloudWatch Events 中，您创建规则以检测和响应定义的管道源中的状态更改。

### 创建规则

1. 创建一个 Amazon CloudWatch Events 规则，并将管道的源存储库作为事件源。
2. 将 AWS CodePipeline 添加为目标。
3. 向 Amazon CloudWatch Events 授予启动管道的权限。

在生成规则时，控制台中的事件模式预览窗格 (或 AWS CLI 中的 `--event-pattern` 输出) 以 JSON 格式显示事件字段。以下示例 AWS CodeCommit 事件模式使用该 JSON 结构：

```
{
  "source": [ "aws.codecommit" ],
  "detail-type": [ "CodeCommit Repository State Change" ],
  "resources": [ "CodeCommitRepo_ARN" ],
  "detail": {
    "event": [
      "referenceCreated",
      "referenceUpdated",
      "referenceType":["branch"],
      "referenceName": ["branch_name"]
    ]
  }
}
```

下面是具有名为“master”的分支的“MyTestRepo”存储库的事件窗口中的示例 AWS CodeCommit 事件模式：

```
{
  "source": [
    "aws.codecommit"
  ],
  "detail-type": [
    "CodeCommit Repository State Change"
  ],
  "resources": [
    "arn:aws:codecommit:us-west-2:80398EXAMPLE:MyTestRepo"
  ],
  "detail": {
    "referenceType": [
      "branch"
    ]
  }
}
```



```
    ],  
    "referenceName": [  
        "master"  
    ]  
  }  
}
```

该事件模式使用以下字段：

- **source**：应包含 `aws.codecommit` 以作为事件源。
- **detail-type**：显示可用的事件类型 (CodeCommit Repository State Change)。
- **resources**：包含存储库 ARN。
- **detail**：包含存储库分支信息 `referenceType` 和 `referenceName`。

#### 主题

- [创建启动您的 AWS CodeCommit 管道的 CloudWatch Events 规则 \(控制台\) \(p. 65\)](#)
- [创建启动 AWS CodeCommit 管道的 CloudWatch Events 规则 \(CLI\) \(p. 67\)](#)
- [将 AWS CodeCommit 管道配置为使用 Amazon CloudWatch Events 检测更改 \(p. 69\)](#)

## 创建启动您的 AWS CodeCommit 管道的 CloudWatch Events 规则 (控制台)

创建在 AWS CodePipeline 操作中使用的 CloudWatch Events 规则

1. 通过以下网址打开 CloudWatch 控制台：<https://console.aws.amazon.com/cloudwatch/>。
2. 在导航窗格中，选择 Events。
3. 选择创建规则，然后从事件源下面的服务名称下拉列表中选择 CodeCommit。
4. 服务名称指定拥有事件资源的服务。例如，选择 AWS CodeCommit 以在与管道关联的 AWS CodeCommit 存储库发生更改时触发管道。
5. 从事件类型下拉列表中，选择 CodeCommit 存储库状态更改。

## Step 1: Create rule

Create rules to invoke Targets based on Events happening in your AWS environment.

### Event Source

Build or customize an Event Pattern or set a Schedule to invoke Targets.

☒ Event Pattern ⓘ ☐ Schedule ⓘ

Build custom event pattern

```
{
  "source": [
    "aws.codecommit"
  ],
  "detail-type": [
    "CodeCommit Repository State Change"
  ],
  "resources": [
    "arn:aws:codecommit:us-west-2:80398EXAMPLE:MyTestRepo"
  ],
  "detail": {
    "referenceType": [
      "branch"
    ],
    "referenceName": [
      "master"
    ]
  }
}
```

### Targets

Select Target to schedule is triggered

**CodePipeline**

Pipeline ARN

[Learn more](#)

► [Configure](#)

**CloudWatch Events**

AWS CodePipeline provide these

☒ Create

☐ Use existing

[Learn more about](#)

+ Add target

6. 要创建适用于所有存储库的规则，请选择任意资源。

要创建适用于一个或多个存储库的规则，请选择按 ARN 排列的特定资源，然后输入 ARN。

#### Note

您可以在 AWS CodeCommit 控制台的设置页中查找 AWS CodeCommit 存储库 ARN。

要指定与存储库关联的分支，请选择编辑，然后键入资源类型分支和分支名称。请使用 detail 的事件模式选项。以上示例显示名为“master”的 AWS CodeCommit 存储库分支的 detail 选项。选择 Save (保存)。

在事件模式预览窗格中，查看该规则。



7. 在目标区域中，选择 CodePipeline。
8. 输入在该规则触发时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`

9. 创建或指定一个 IAM 服务角色，该角色向 Amazon CloudWatch Events 授予权限以调用与 Amazon CloudWatch Events 规则关联的目标 (在此示例中，目标为 AWS CodePipeline)。
  - 选择为此特定资源创建新角色以创建一个服务角色，该角色为 Amazon CloudWatch Events 授予权限以在触发时启动管道执行。
  - 选择使用现有角色以输入一个服务角色，该角色为 Amazon CloudWatch Events 授予权限以在触发时启动管道执行。
10. 检查规则设置以确保它符合您的要求。
11. 选择 `Configure details`。
12. 在配置规则详细信息页中，键入规则的名称和描述，然后选择状态以启用该规则。
13. 如果您对规则满意，请选择 `Create rule`。

## 创建启动 AWS CodeCommit 管道的 CloudWatch Events 规则 (CLI)

要使用 AWS CLI 创建规则，请调用 `put-rule` 命令并指定以下内容：

- 唯一地标识创建的规则的名称。在使用与您的 AWS 账户关联的 AWS CodePipeline 创建的所有管道中，该名称必须是唯一的。
- 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 CloudWatch Events 和事件模式](#)。

创建 CloudWatch Events 规则并将 AWS CodeCommit 作为事件源，而将 AWS CodePipeline 作为目标

1. 为 Amazon CloudWatch Events 添加权限以使用 AWS CodePipeline 调用规则。有关更多信息，请参阅 [将基于资源的策略用于 Amazon CloudWatch Events](#)。
  - a. 使用以下示例创建信任策略以允许 CloudWatch Events 代入服务角色。将信任策略命名为“`trustpolicyforCWE.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://trustpolicyforCWE.json
```

- c. 创建权限策略 JSON，如下示例中名为“MyFirstPipeline”的管道所示。将权限策略命名为“permissionspolicyforCWE.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-CWE”权限策略附加到“Role-for-MyRule”角色。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-CWE --policy-document file://permissionspolicyforCWE.json
```

2. 调用 put-rule 命令并包含 --name 和 --event-pattern 参数。

使用下面的语法：

```
aws events put-rule
--name "rule_name"
--event-pattern '{"source":["aws.service_name"], "detail-type":["event_type"],
"resources":["repository_ARN"]}'
```

示例：

以下示例命令使用 --event-pattern 创建名为 MyCodeCommitRepoRule 的规则。

```
aws events put-rule --name "MyCodeCommitRepoRule" --event-pattern '{"source":
["aws.codecommit"], "detail-type":["CodeCommit Repository State Change"], "detail
\":"referenceType":["branch"], "referenceName":["master"]}]'
```

3. 要将 AWS CodePipeline 添加为目标，请调用 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 以及目标管道的 ARN 结合使用。

使用下面的语法：

```
aws events put-targets
--rule rule_name
--targets Id,ARN
```

例如：

下面的示例命令为名为 MyCodeCommitRepoRule 的规则指定此内容，目标 Id 由数字 1 组成，这指示此内容可能是规则的目标列表，而这是目标 1。示例命令还为在存储库中发生更改时启动的管道指定示例 ARN。

```
aws events put-targets --rule MyCodeCommitRepoRule --targets  
Id=1,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
```

## 将 AWS CodeCommit 管道配置为使用 Amazon CloudWatch Events 检测更改

您现在可以将基于事件的 Amazon CloudWatch Events 规则作为具有 AWS CodeCommit 存储库的管道的更改检测方法。Amazon CloudWatch Events 将实时检测更改，然后启动您的管道。您必须更新现有管道以使用此功能。在控制台中创建或编辑管道时，AWS CodePipeline 将创建基于事件的 Amazon CloudWatch Events 规则。

此表包含适用于 AWS CodeCommit 管道的过程。

方法	说明	需要进一步操作？
在控制台中配置管道	<ol style="list-style-type: none"><li>在控制台中打开管道。</li><li>选择 Edit。</li><li>选择 AWS CodeCommit 源操作旁边的铅笔图标。</li><li>选择 Update。</li><li>选择保存管道更改。</li></ol> <p>请参阅 <a href="#">编辑管道 (控制台) (p. 91)</a>。</p>	<p>这将为您创建 Amazon CloudWatch Events 规则。</p> <p>无需进一步操作。</p>
在 CLI 中配置管道	<p>使用 update-pipeline 命令将 PollForSourceChanges 参数设置为 false。</p> <p>请参阅 <a href="#">编辑管道 (AWS CLI) (p. 93)</a>。</p>	<p>您必须创建 Amazon CloudWatch Events 规则。</p> <p>请参阅 <a href="#">使用 CloudWatch Events 规则自动启动 AWS CodeCommit 管道 (p. 64)</a>。</p>
在 AWS CloudFormation 中配置管道	<p>更新 AWS CloudFormation 资源堆栈。</p> <p>请参阅 <a href="#">什么是 Amazon CloudWatch Events</a>。</p>	<p>您必须创建 Amazon CloudWatch Events 规则。</p> <p>请参阅 <a href="#">使用 CloudWatch Events 规则自动启动 AWS CodeCommit 管道 (p. 64)</a>。</p>

## 使用 CloudWatch Events 规则自动启动 Amazon S3 管道

您应使用 Amazon CloudWatch Events 检测源代码更改并触发您的管道以自动启动。如果您的管道具有 Amazon S3 源，则您必须创建相应的 AWS CloudTrail 跟踪来将写入事件记录到 Amazon S3 源存储桶中的对象。

AWS CloudTrail 是一项服务，可记录和筛选 Amazon S3 源存储桶上的事件。跟踪将筛选存储桶上的事件，然后将筛选出的源更改发送到 Amazon CloudWatch Events 规则。Amazon CloudWatch Events 规则将检测源更改，然后启动您的管道。

#### Note

对于具有 Amazon S3 源的管道，Amazon CloudWatch Events 规则将检测源更改，然后在发生更改时启动您的管道。在使用控制台创建或更改管道时，将为您创建规则和相关联资源。如果您在 CLI 或 AWS CloudFormation 中创建或更改 Amazon S3 管道，则必须手动创建 Amazon CloudWatch Events 规则、IAM 角色和 AWS CloudTrail 跟踪。

要求：

- 如果您未创建跟踪，则需要使用现有 AWS CloudTrail 跟踪以在 Amazon S3 源存储桶中记录事件并将筛选出的事件发送到 Amazon CloudWatch Events 规则。
- 您需要创建或使用 AWS CloudTrail 可将其日志文件存储到的现有 S3 存储桶。要将日志文件传输到 Amazon S3 存储桶，AWS CloudTrail 必须具有所需的权限，且不得配置为 [申请方付款](#) 存储桶。在控制台中创建或更新跟踪的过程中创建 Amazon S3 存储桶时，AWS CloudTrail 会自动将所需权限附加到存储桶。有关更多信息，请参阅[适用于 CloudTrail 的 Amazon S3 存储桶策略](#)。

## 创建启动 Amazon S3 管道的 CloudWatch Events 规则 (控制台)

在 CloudWatch Events 中设置规则之前，您必须创建 AWS CloudTrail 跟踪。有关更多信息，请参阅[在控制台中创建跟踪](#)。

#### 创建跟踪

1. 打开 AWS CloudTrail 控制台。
2. 在导航窗格中，选择 Trails。
3. 选择 Create Trail (创建跟踪)。对于 Trail name，键入跟踪的名称。
4. 对于 Apply trail to all regions (对所有区域应用跟踪)，选择 No (否)。
5. 在 Data events (数据事件) 下，确保 S3 处于选中状态。指定 Amazon S3 存储桶和对象前缀 (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。
6. 对于 Read/Write events (读取/写入事件)，选择 None (无)。
7. 选择 Write (写入) 选项。跟踪将记录指定存储桶和前缀的 Amazon S3 对象级别 API 活动 (例如，GetObject 和 PutObject)。
8. 在 Storage location (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。仅资源所有者 (创建该存储桶的 AWS 账户) 能够访问存储桶及其对象。该存储桶必须具有一个允许 AWS CloudTrail 访问存储桶中的对象的资源策略。
9. 如果您对跟踪满意，请选择 Create (创建)。

#### 创建针对具有 S3 源的管道的 CloudWatch Events 规则

1. 通过以下网址打开 CloudWatch 控制台：<https://console.aws.amazon.com/cloudwatch/>。
2. 在导航窗格中，选择 Events。
3. 选择 Event Pattern (事件模式)，然后选择 Build event pattern to match events by service (生成事件模式以按服务匹配事件)。
4. 在 Event source (事件源) 下，从 Service Name (服务名称) 下拉列表中，选择 Simple Storage Service (S3)。
5. 从 Event Type (事件类型) 下拉列表中，选择 Object Level Operations (对象级别操作)。
6. 选择 Specific operation(s) (特定操作)，然后选择 PutObject。

在 Event Pattern Preview (事件模式预览) 窗格上方, 选择 Edit (编辑)。编辑事件模式以指定 resources 参数, 后跟存储桶的名称、前缀 (文件夹名称) 和对象, 如名为 myObject.zip 的对象的此示例中所示。在使用 Edit (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。



7. 在目标区域中, 选择 CodePipeline。
8. 输入在该规则触发时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`

9. 选择以下选项之一以创建或指定一个 IAM 服务角色, 该角色向 Amazon CloudWatch Events 授予权限以调用与 Amazon CloudWatch Events 规则关联的目标 (在此示例中, 目标为 AWS CodePipeline)。
  - 选择为此特定资源创建新角色以创建一个服务角色, 该角色为 Amazon CloudWatch Events 授予权限以在触发时启动管道执行。
  - 选择使用现有角色以输入一个服务角色, 该角色为 Amazon CloudWatch Events 授予权限以在触发时启动管道执行。
10. 检查规则以确保它符合您的要求。
11. 选择 Configure details。
12. 在配置规则详细信息页中, 键入规则的名称和描述, 然后选择状态以启用该规则。
13. 如果您对规则满意, 请选择 Create rule。

## 创建启动您的 Amazon S3 管道的 CloudWatch Events 规则 (CLI)

创建 AWS CloudTrail 跟踪并启用日志记录

要使用 AWS CLI 创建跟踪, 请调用 `create-trail` 命令, 并指定:

- 跟踪名称。
- 已将 AWS CloudTrail 的存储桶策略应用于的存储桶。

有关在 CLI 中创建跟踪的更多信息, 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。

使用下面的语法：

```
aws cloudtrail create-trail
--name trail_name
--s3-bucket-name bucket_name
```

例如：

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。

例如：

以下示例命令使用 `--name` 对名为 `my-trail` 的跟踪启动日志记录。

```
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跟踪记录源存储桶的数据事件并将这些事件发送到 Amazon CloudWatch Events 规则。

例如：

以下示例命令使用 `--trail-name` 和 `--event-selectors` 指定源存储桶的管理数据事件以及名为 `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 CloudWatch Events 规则，并将 Amazon S3 和 AWS CodePipeline 分别用作事件源和目标，同时应用权限策略

1. 为 Amazon CloudWatch Events 授予权限以使用 AWS CodePipeline 调用规则。有关更多信息，请参阅[将基于资源的策略用于 Amazon CloudWatch Events](#)。
  - a. 使用以下示例创建信任策略以允许 CloudWatch Events 代入服务角色。将它命名为 `trustpolicyforCWE.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://trustpolicyforCWE.json
```

- c. 创建权限策略 JSON，如以下示例中名为“MyFirstPipeline”的管道所示。将权限策略命名为“permissionspolicyforCWE.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-CWE”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-CWE --policy-document file://permissionspolicyforCWE.json
```

2. 调用 put-rule 命令并包含 --name 和 --event-pattern 参数。

使用下面的语法：

```
aws events put-rule
--name "rule_name"
--event-pattern "{\"source\":\"aws.service_name\", \"detail-type\":\"event_type\",
\"resources\":\"repository_ARN\"}"
```

以下示例命令使用 --event-pattern 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\":[\"PutObject\"],\"resources\":[\"ARN\":
[\"arn:aws:s3::myBucket/myFolder/file.zip\"]}]}"
```

3. 要将 AWS CodePipeline 添加为目标，请调用 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 以及目标管道的 ARN 结合使用。

使用下面的语法：

```
aws events put-targets
--rule rule_name
--targets Id,ARN
```

下面的示例命令为名为 MyS3SourceRule 的规则指定此内容，目标 Id 由数字 1 组成，这指示此内容可能是规则的目标列表，而这是目标 1。示例命令还为在存储库中发生更改时启动的管道指定示例 ARN。

```
aws events put-targets --rule MyS3SourceRule --targets  
Id=1,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
```

## 将您的 Amazon S3 管道配置为使用 Amazon CloudWatch Events 检测更改

您现在可以将基于事件的 Amazon CloudWatch Events 规则作为具有 Amazon S3 源的管道的更改检测方法。Amazon CloudWatch Events 将实时检测更改，然后启动您的管道。您必须更新现有管道以使用此功能。在控制台中创建或编辑管道时，AWS CodePipeline 将创建基于 Amazon CloudWatch Events 事件的规则和 AWS CloudTrail 跟踪。

此表包含适用于 Amazon S3 管道的过程。

方法	说明	需要进一步操作？
在控制台中配置管道	<ol style="list-style-type: none"><li>在控制台中打开管道。</li><li>选择 Edit。</li><li>选择 Amazon S3 源操作旁边的铅笔图标。</li><li>选择 Use Amazon CloudWatch Events to automatically start my pipeline when a change occurs (在发生更改时，使用 Amazon CloudWatch Events 自动启动我的管道)。</li><li>选择 Update。</li><li>选择保存管道更改。</li></ol> <p>请参阅 <a href="#">编辑管道 (控制台) (p. 91)</a>。</p>	<p>这将为您创建 Amazon CloudWatch Events 规则和 AWS CloudTrail 跟踪。</p> <p>无需进一步操作。</p>
在 CLI 中配置管道	<p>使用 update-pipeline 命令将 PollForSourceChanges 参数设置为 false。</p> <p>请参阅 <a href="#">编辑管道 (AWS CLI) (p. 93)</a>。</p>	<p>您必须创建：</p> <ul style="list-style-type: none"><li>• AWS CloudTrail 跟踪。</li><li>• Amazon CloudWatch Events 规则。</li></ul> <p>请参阅 <a href="#">使用 CloudWatch Events 规则自动启动 Amazon S3 管道 (p. 69)</a>。</p>
在 AWS CloudFormation 中配置管道	<p>更新 AWS CloudFormation 资源堆栈。</p> <p>请参阅<a href="#">什么是 Amazon CloudWatch Events</a>。</p>	<p>您必须创建：</p> <ul style="list-style-type: none"><li>• AWS CloudTrail 跟踪。</li><li>• Amazon CloudWatch Events 规则。</li></ul> <p>请参阅 <a href="#">使用 CloudWatch Events 规则自动启动 Amazon S3 管道 (p. 69)</a>。</p>



## 使用 Webhook 自动启动 GitHub 管道

Webhook 是一个 HTTP 通知，用于检测其他工具（如 GitHub 存储库）中的事件并将那些外部事件连接到管道。

当您使用控制台创建或编辑您的 GitHub 管道时，AWS CodePipeline 会创建一个 Webhook；而当您删除您的管道时，它将删除您的 Webhook。您不需要手动在 GitHub 中管理它。如果您使用 AWS CLI 或 AWS CloudFormation 创建或编辑 GitHub 管道，则必须使用下面各部分中的信息自行管理 Webhook。

### 主题

- [为 GitHub 管道创建 Webhook \(p. 75\)](#)
- [列出您账户中的 Webhook \(p. 76\)](#)
- [更新 GitHub 管道的 Webhook \(p. 77\)](#)
- [删除 GitHub 管道的 Webhook \(p. 78\)](#)
- [将 GitHub 管道配置为使用 Webhook 检测更改 \(p. 79\)](#)

## 为 GitHub 管道创建 Webhook

您可以使用 AWS CLI 手动创建 Webhook。接下来，您必须在 GitHub 中注册该 Webhook。指定的 AWS 终端节点将用于此 Webhook，并由 put-webhook 命令提供。

要使用 AWS CLI 创建 Webhook，请调用 put-webhook 命令并提供以下内容：

- 唯一标识 Webhook 的名称。此名称在此管道的账户所在的区域必须具有唯一性。
- JSON 文件中要用于 GitHub 授权的密钥。

1. 在文本编辑器中，创建并保存您要创建的 Webhook 的 JSON 文件。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。

使用下面的语法：

```
aws codepipeline put-webhook  
--cli-input-json file:"file_name"  
--region region
```

示例：

下面的示例命令使用“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. 在本例中针对名为“my-webhook”的 Webhook 显示的输出中，返回了 URL 和 ARN。

```
{
  "webhook": {
    "url": "https://webhooks.domain.com/trigger11111111EXAMPLE1111111111111111",
    "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"
  }
}
```

4. 调用 register-webhook-with-third-party 命令并包含 --webhook-name 参数。

使用下面的语法：

```
aws codepipeline register-webhook-with-third-party
--webhook-name webhook_name
```

示例：

下面的示例命令使用名称“my-webhook”注册一个 Webhook。

```
aws codepipeline register-webhook-with-third-party --webhook-name my-webhook
```

## 列出您账户中的 Webhook

您可以使用 AWS CLI 列出您账户中的 Webhook。

1. 要列出您的 Webhook，请调用 list-webhooks 命令并包括 --endpoint-url 和 --region 参数。

使用下面的语法：

```
aws codepipeline list-webhooks
--endpoint-url "endpoint_URL"
--region region
```

示例：

以下示例命令列出“eu-central-1”终端节点 URL 的 Webhook。

```
aws codepipeline list-webhooks --endpoint-url "https://codepipeline.eu-central-1.amazonaws.com" --region "eu-central-1"
```

2. 列出了 Webhook，包括每个 Webhook 的名称和 ARN。

```
{
  "webhooks": [
    {
      "url": "https://webhooks.domain.com/trigger11111111EXAMPLE1111111111111111": {
        "authenticationConfiguration": {
          "SecretToken": "Secret"
        },
        "name": "my-webhook",
        "authentication": "GITHUB_HMAC",
        "targetPipeline": "my-Pipeline",
        "targetAction": "Source",
        "filters": [
          {
            "jsonPath": "$.ref",
            "matchEquals": "refs/heads/{Branch}"
          }
        ]
      },
      "arn": "arn:aws:codepipeline:eu-central-1:ACCOUNT_ID:webhook:my-webhook"
    }
  ]
}
```

3. 在 GitHub 中，选择您的存储库。
4. 选择 Settings。
5. 选择 Webhooks。查看存储库的 Webhook 信息。

## 更新 GitHub 管道的 Webhook

您可以使用 AWS CLI 编辑存储库的 Webhook。根据您的更新 Webhook 的方式，您可以使用下列方法之一：

- 如果您使用控制台来编辑您管道的 GitHub 源操作，则将为您更新 (和重新注册，如果适用) Webhook。
- 如果您不更新 Webhook 名称且不更改 GitHub 存储库，则可以使用 AWS CLI 更新 Webhook。具体示例在下面用于更新 Webhook 密钥的过程中提供。请参阅示例 1。
- 如果您正在更改 Webhook 名称或 GitHub 存储库名称，您必须在控制台中编辑源操作，或者在 CLI 中删除并重新创建 Webhook。在创建新的 Webhook 后，您还注册它。请参阅示例 2。

### 示例 1：使用 CLI 更新 Webhook 密钥

1. 在文本编辑器中，修改您要更新的 Webhook 的 JSON 文件。本示例修改用于在[GitHub 管道创建 Webhook \(p. 75\)](#)中创建 Webhook 的示例文件。此示例更新名为“my-webhook”的 Webhook 以更改密钥令牌。

```
{ "webhook":
  { "name": "my-webhook",
    "targetPipeline": "pipeline_name",
    "targetAction": "source_action_name",
    "filters": [
      {
        "jsonPath": "$.ref",
        "matchEquals": "refs/heads/{Branch}"
      }
    ]
  }
}
```

```
    },  
    ],  
    "authentication": "GITHUB_HMAC",  
    "authenticationConfiguration": {"SecretToken": "new_secret" }  
  }  
}
```

2. 调用 put-webhook 命令并包含 --cli-input 和 --region 参数。

使用下面的语法：

```
aws codepipeline put-webhook  
--cli-input-json file:"file_name"  
--region region
```

示例：

下面的示例命令使用修改后的“webhook\_json”JSON 文件更新 Webhook。

```
aws codepipeline put-webhook --cli-input-json file://webhook_json.json --region "eu-central-1"
```

3. 输出将返回 Webhook 详细信息并显示新密钥。

#### Note

您可以在控制台中编辑 GitHub 源操作。这允许 AWS CodePipeline 为您管理 Webhook。

#### 示例 2：使用 CLI 更新 Webhook 名称或 GitHub 存储库

1. 使用[删除 GitHub 管道的 Webhook \(p. 78\)](#)中的步骤取消注册并删除与旧 Webhook 名称或 GitHub 存储库关联的现有 Webhook。
2. 使用[为 GitHub 管道创建 Webhook \(p. 75\)](#)中的步骤，通过创建和注册新的 Webhook 来重新创建 Webhook。

#### Note

您可以在控制台中编辑 GitHub 源操作。这允许 AWS CodePipeline 为您管理 Webhook。

## 删除 GitHub 管道的 Webhook

使用 AWS CLI 删除 Webhook：

1. 在删除 Webhook 之前，您必须将其取消注册。调用 deregister-webhook-with-third-party 命令并包含 --webhook-name 参数。

使用下面的语法：

```
aws codepipeline deregister-webhook-with-third-party  
--webhook-name webhook_name
```

示例：

下面的示例命令取消注册名为“my-webhook”的 Webhook。

```
aws codepipeline deregister-webhook-with-third-party --webhook-name my-webhook
```

2. 调用 `delete-webhook` 命令并包含 `--name` 参数。

使用下面的语法：

```
aws codepipeline delete-webhook
--name "webhook_name"
```

示例：

下面的示例命令删除名为“my-webhook”的 Webhook。

```
aws codepipeline delete-webhook --name my-webhook
```

## 将 GitHub 管道配置为使用 Webhook 检测更改

您现在可以将 Webhook 作为具有 GitHub 存储库的管道的更改检测方法。您必须更新现有管道以使用此功能。当您使用控制台创建或编辑管道时，AWS CodePipeline 将创建 Webhook。

此表包含适用于 GitHub 管道的过程。

方法	说明	需要进一步操作？
在控制台中配置管道	<ol style="list-style-type: none"><li>1. 在控制台中打开管道。</li><li>2. 选择 Edit。</li><li>3. 选择 GitHub 源操作旁边的铅笔图标。</li><li>4. 选择 Connect to repository (连接到存储库)，然后选择 Update (更新)。</li><li>5. 选择保存管道更改。</li></ol> <p>请参阅 <a href="#">编辑管道 (控制台)</a> (p. 91)。</p>	<p>将为您创建 Webhook 并向 GitHub 注册它。</p> <p>无需进一步操作。</p>
在 CLI 中配置管道	<p>使用 <code>update-pipeline</code> 命令将 <code>PollForSourceChanges</code> 参数设置为 <code>false</code>。</p> <p>请参阅 <a href="#">编辑管道 (AWS CLI)</a> (p. 93)。</p>	<p>您必须创建 Webhook 并向 GitHub 注册它。</p> <p>请参阅 <a href="#">为 GitHub 管道创建 Webhook</a> (p. 75)。</p>
在 AWS CloudFormation 中配置管道	<p>为您的 GitHub 源存储库保留定期检查。</p>	<p>有关 AWS CloudFormation 中 webhook 资源的信息，请参阅 <a href="#">AWS::CodePipeline::Webhook</a>。</p>

## 使用定期检查自动启动管道

当检测到存储库更改时管道会自动启动，而一种更改检测方法是定期检查。可使用 `PollForSourceChanges` 标志启用或禁用定期检查。对于使用 CLI 创建或更新的管道，此参数默认设置为 `true`，但这不是推荐的配置。而是应更新您的管道以使用建议的更改检测方法，然后将此参数设置为 `false`。

有关使用推荐的配置创建管道的更多信息，请参阅[创建管道 \(控制台\)](#) (p. 84) 和[创建管道 \(CLI\)](#) (p. 88)。有关使用推荐的配置更新操作或管道的更多信息，请参阅[编辑管道 \(控制台\)](#) (p. 91) 和[编辑管道 \(CLI\)](#) (p. 93)。

有关更多信息，请参阅[用于自动启动管道的变更检测方法](#) (p. 63)。

## 在 AWS CodePipeline 中手动启动管道

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 AWS CodePipeline 控制台或 AWS CLI 以及 `start-pipeline-execution` 命令，通过管道手动重新运行最新的修订。

### 主题

- [手动启动管道 \(控制台\) \(p. 80\)](#)
- [手动启动管道 \(CLI\) \(p. 80\)](#)

## 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 管理控制台并通过以下网址打开 AWS CodePipeline 控制台：<http://console.aws.amazon.com/codepipeline>。
2. 在名称中，选择要启动的管道的名称。
3. 在管道详细信息页面上，选择 发布更改。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, or Unix) 或命令提示符 (Windows)，然后使用 AWS CLI 运行 `start-pipeline-execution` 命令，并指定要手动启动的管道的名称。例如，通过名为 *MyFirstPipeline* 的管道开始运行上次的更改：

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

2. 要验证是否成功，请查看返回的对象。该命令会返回 `ExecutionID` 对象，如下所示：

```
{
  "pipelineExecutionId": "c53dbd42-This-Is-An-Example"
}
```

### Note

在启动管道后，您可以在 AWS CodePipeline 控制台中监控其进度，或者运行 `get-pipeline-state` 命令以监控其进度。有关更多信息，请参阅 [查看管道详细信息和历史记录 \(控制台\) \(p. 96\)](#) 和 [查看管道详细信息和历史记录 \(CLI\) \(p. 98\)](#)。

## 使用 Amazon CloudWatch Events 按计划自动启动管道

您可以在 Amazon CloudWatch Events 中设置一个规则以按计划启动管道。

## 创建用于按计划启动您的管道的 CloudWatch Events 规则 (控制台)

创建 CloudWatch Events 规则并将计划作为事件源

1. 通过以下网址打开 CloudWatch 控制台：<https://console.aws.amazon.com/cloudwatch/>。
2. 在导航窗格中，选择 Events。
3. 选择创建规则，然后在事件源下面选择计划。
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. 在目标区域中，选择 CodePipeline。
6. 输入在该计划触发时启动的管道执行的管道 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。

7. 选择以下选项之一以创建或指定一个 IAM 服务角色，以便为 Amazon CloudWatch Events 授予权限以调用与 Amazon CloudWatch Events 规则关联的目标 (在此示例中，目标为 AWS CodePipeline)。
  - 选择为此特定资源创建新角色以创建一个服务角色，该角色为 Amazon CloudWatch Events 授予权限以在触发时启动管道执行。
  - 选择使用现有角色以输入一个服务角色，该角色为 Amazon CloudWatch Events 授予权限以在触发时启动管道执行。
8. 选择 Configure details。
9. 在配置规则详细信息页中，键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。

## 创建用于按计划启动您的管道的 CloudWatch Events 规则 (CLI)

要使用 AWS CLI 创建规则，请调用 `put-rule` 命令并指定以下内容：

- 唯一地标识创建的规则的名称。在使用与您的 AWS 账户关联的 AWS CodePipeline 创建的所有管道中，该名称必须是唯一的。
- 规则的计划表达式。

创建 CloudWatch Events 规则并将计划作为事件源

1. 调用 `put-rule` 命令并包含 `--name` 和 `--schedule-expression` 参数。

示例：

以下示例命令使用 `--schedule-expression` 创建名为 `MyRule2` 的规则以按计划筛选 CloudWatch Events。

```
aws events put-rule --schedule-expression 'cron(15 10 ? * 6L 2002-2005)' --name MyRule2
```

2. 为 Amazon CloudWatch Events 授予权限以使用 AWS CodePipeline 调用规则。有关更多信息，请参阅[将基于资源的策略用于 Amazon CloudWatch Events](#)。
  - a. 使用以下示例创建信任策略以允许 CloudWatch Events 代入服务角色。将它命名为 `"trustpolicyforCWE.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://trustpolicyforCWE.json
```

- c. 创建权限策略 JSON，如下示例中名为“MyFirstPipeline”的管道所示。将权限策略命名为“permissionspolicyforCWE.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-CWE”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-CWE --policy-document file://permissionspolicyforCWE.json
```

## 在 AWS CodePipeline 中创建管道

您可以使用 AWS CodePipeline 控制台或 AWS CLI 创建管道。

您还可以创建管道以生成基于容器的应用程序，并将 Amazon ECS 作为部署提供程序以进行部署。在创建管道以使用 Amazon ECS 部署基于容器的应用程序之前，您必须准备一个映像定义文件。

当推送源代码更改时，AWS CodePipeline 使用更改检测方法以启动您的管道。这些检测方法基于源类型：

- AWS CodePipeline 使用 Amazon CloudWatch Events 检测 AWS CodeCommit 源存储库和分支或 Amazon S3 源存储桶中的更改。
- AWS CodePipeline 使用 Webhook 检测 GitHub 源存储库和分支中的更改。Webhook 是一个 HTTP 通知，用于检测外部工具中的事件并将那些外部事件连接到管道。



## 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。如果使用 AWS CLI 创建管道，您必须自行创建其他资源。有关更多信息，请参阅 [使用 CloudWatch Events 规则自动启动 AWS CodeCommit 管道 \(p. 64\)](#)。

## 主题

- [创建映像定义文件以部署基于容器的应用程序 \(p. 83\)](#)
- [创建管道 \(控制台\) \(p. 84\)](#)
- [创建管道 \(CLI\) \(p. 88\)](#)

# 创建映像定义文件以部署基于容器的应用程序

映像定义文档是一个 JSON 文件，它描述 Amazon ECS 服务的容器名称以及映像和标签。如果部署基于容器的应用程序，您必须生成一个映像定义文件，以便为 AWS CodePipeline 作业辅助角色提供 Amazon ECS 容器和映像标识以用于管道的部署阶段。

- 映像定义文件的最大文件大小限制为 100 KB。
- 您必须生成映像定义文件，以将其作为部署操作的输入项目。

映像定义文件提供容器名称和映像 URI，必须使用以下键值对集合进行构造。

创建映像定义文件以作为基于容器的部署的源或生成项目

键	值
name	####
imageURI	## URI

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"
  }
]
```

您也可以构造映像定义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映像定义文件。

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. 选择以下选项之一：
  - a. 如果管道不包含生成阶段，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项目。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 存储桶，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。可以在 `buildspec.yml` 文件的 `post_build` 部分中列出该命令：

```
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项目。

3. 在控制台中创建管道时，您必须在创建管道向导的部署页面上，在映像文件名字段中输入确切的映像定义文件名。

有关创建将 Amazon ECS 作为部署提供程序的管道的分步教程，请参阅[教程：使用 AWS CodePipeline 进行持续部署](#)。

#### 主题

- [创建管道 \(控制台\) \(p. 84\)](#)
- [创建管道 \(CLI\) \(p. 88\)](#)

## 创建管道 (控制台)

要在控制台中创建管道，您需要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。

当您使用管道向导时，AWS CodePipeline 将创建阶段的名称 (source、build、staging)。这些名称不能更改。您可以为以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 管理控制台并通过以下网址打开 AWS CodePipeline 控制台：<http://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 AWS CodePipeline，请选择立即开始试用。

3. 在步骤 1: 名称页面上的管道名称中，键入您的管道的名称，然后选择下一步。

在单个 AWS 账户中，您在一个区域中创建的每个管道都必须具有唯一名称。名称可重用于不同区域的管道。

#### Note

当您创建管道后，便无法再更改其名称。有关其他限制的信息，请参阅[AWS CodePipeline 中的限制 \(p. 208\)](#)。

### 步骤 2：创建源阶段

- 在 Step 2: Source (步骤 2: 源) 页面上，在 Source provider (源提供商) 下拉列表中，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 Next step (下一步)。

- 对于 GitHub：

- 选择 Connect to GitHub。如果系统提示您登录，请提供您的 GitHub 凭证。

#### Important

不要提供您的 AWS 凭证。

- 如果这是您第一次针对此区域从 AWS CodePipeline 连接到 GitHub，您将需要授予应用程序访问您的账户的权限。查看集成所需的权限，然后，如果您想继续操作，请选择 Authorize application。当您在控制台中连接到 GitHub 时，将为您创建以下资源：

- AWS CodePipeline 使用 OAuth 令牌创建由 AWS CodePipeline 管理的授权应用程序。

#### Note

在 GitHub 中，可用于应用程序 (如 AWS CodePipeline) 的 OAuth 令牌的数量有限制。如果您超过此限制，请重试连接以允许 AWS CodePipeline 通过重用现有令牌来重新连接。有关更多信息，请参阅 [配置管道以使用 GitHub 的个人访问令牌 \(p. 167\)](#)。

- AWS CodePipeline 在 GitHub 中创建一个 Webhook 以检测源更改，然后在发生更改时启动您的管道。除了 Webhook 外，AWS CodePipeline 还：

- 随机生成一个密钥，并使用它来授权连接到 GitHub。
- 使用此区域的公有终端节点生成 Webhook URL 并将其注册到 GitHub。这将订阅 URL 以接收存储库事件。

- 选择要用作管道的源位置的 GitHub 存储库。在 Branch (分支名称) 中，从下拉列表中选择要使用的分支。

- 对于 Amazon S3：

- 在 Amazon S3 location (Amazon S3 位置) 中，提供 Amazon 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
```

- 在选择 Amazon S3 源存储桶后，Change detection options (更改检测选项) 中将显示一条消息，该消息显示将为该管道创建的 Amazon CloudWatch Events 规则和 AWS CloudTrail 跟踪。该消息还以如下格式显示要添加到 AWS CloudTrail 跟踪的数据事件：my-bucket/path/to/object.zip。接受更改检测选项下面的默认值。这样，AWS CodePipeline 就可以使用 Amazon CloudWatch Events 和 AWS CloudTrail 检测新管道的更改。选择下一步。

- 对于 AWS CodeCommit：

- 在 Repository name (存储库名称) 中，选择您要用作管道的源位置的 AWS 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。
- 在选择 AWS CodeCommit 存储库名称和分支后，将在更改检测选项中显示一条消息以显示将为该管道创建的 Amazon CloudWatch Events 规则。接受更改检测选项下面的默认值。这样，AWS CodePipeline 就可以使用 Amazon CloudWatch Events 检测新管道的更改。

#### Note

对象和文件类型必须与您计划使用的部署系统 (例如 Elastic Beanstalk 或 AWS CodeDeploy) 兼容。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 AWS CodeDeploy 部署修订的更多信息, 请参阅[上传应用程序修订](#)和[准备修订](#)。

### 步骤 3 : 创建生成阶段

在 Step 3: Build 页面上, 执行以下操作之一, 然后选择 Next step :

- 选择 No Build 跳过生成阶段的配置操作。
- 选择您要使用的生成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。

#### Note

添加生成提供程序的步骤因提供程序而异。有关如何添加 Jenkins 作为生成提供程序的示例, 请参阅[教程 : 创建一个四阶段管道 \(p. 47\)](#)。

- 选择 AWS CodeBuild, 然后选择您的构建选项。您可以按照 [AWS CodeBuild 用户指南](#) 中的[创建使用 AWS CodeBuild 的管道](#)中的说明操作。
  1. 对于 Project (项目), 选择下列操作之一 :
    - 如果您已在 AWS CodeBuild 中创建项目, 请选择选择现有生成项目。在项目名称中, 选择构建项目的名称, 然后选择保存构建项目。
    - 要使用 AWS CodePipeline 向导创建您的 AWS CodeBuild 项目, 请选择创建新生成项目。确保您已完成[规划构建](#)中的步骤。在项目名称中, 键入您的构建项目的名称。
  2. 为您的构建项目选择映像和操作系统。
  3. 选择以下构建规范选项之一 :
    - 如果您已在管道的源位置中包括构建规范文件, 请选择在源代码根目录中使用 buildspec.yml。
    - 要在向导中指定构建命令和输出文件名称, 请选择插入构建命令。
  4. 在 Cache 中, 执行下列操作之一 :
    - 如果您不想对构建依赖项使用缓存, 请选择无缓存。
    - 如果您想对构建依赖项使用缓存, 请选择 Amazon S3, 然后键入您的 Amazon S3 缓存存储桶的位置。使用缓存可节省大量构建时间, 因为构建环境的一些可重用部分被存储在缓存中, 并且可跨构建使用。

有效的缓存存储桶位置必须包括存储桶名称和前缀 (文件夹名称), 如下示例所示 :

```
s3://mybucket/cacheBucket
```

#### Note

确保您的存储桶与您的管道位于同一区域中。  
确保 AWS CodePipeline 的服务角色拥有针对 AWS CodeBuild 的适当权限。有关更多信息, 请参阅[添加其他 AWS 服务的权限 \(p. 182\)](#)。

### 步骤 4 : 创建部署阶段

在 Step 4: Deploy 页面上, 执行以下操作之一, 然后选择 Next step :

- 选择以下选项之一 :

- 要跳过部署阶段的配置, 请选择 [无部署](#)。

2015-07-09

## Note

只有您在上一个步骤中选择了生成提供程序，您才可以跳过添加部署提供程序这个操作。

- 选择您为部署提供程序创建的自定义操作。
- 从部署提供商下拉列表中，选择以下默认提供程序之一：
  - AWS CodeDeploy

在应用程序名称中，键入或选择现有 AWS CodeDeploy 应用程序的名称。在部署组中，键入应用程序的部署组的名称。选择下一步。您还可以在 AWS CodeDeploy 控制台中创建应用程序和/或部署组。

- AWS Elastic Beanstalk

在应用程序名称中，键入或选择现有 Elastic Beanstalk 应用程序的名称。在环境名称中，键入应用程序的环境。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，键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 create a new one in AWS OpsWorks。

有关向 AWS OpsWorks 中的堆栈和层添加应用程序的信息，请参阅 AWS OpsWorks 用户指南 中的[添加应用程序](#)。

有关如何将 AWS CodePipeline 中的简单管道用作您要在 AWS OpsWorks 层上运行的代码源的端到端示例，请参阅[将 AWS CodePipeline 与 AWS OpsWorks Stacks 结合使用](#)。

- AWS CloudFormation

执行以下任一操作：

- 在操作模式中，选择创建或更新堆栈，键入堆栈名称和模板文件名，然后选择要代入的 AWS CloudFormation 角色的名称。(可选) 键入配置文件的名称，然后选择 IAM 功能选项。
- 在操作模式中，选择创建或替换更改集，键入堆栈名称和更改集名称，然后选择要代入的 AWS CloudFormation 角色的名称。(可选) 键入配置文件的名称，然后选择 IAM 功能选项。

有关将 AWS CloudFormation 功能集成到 AWS CodePipeline 中的管道的信息，请参阅 AWS CloudFormation 用户指南 中的[利用 AWS CodePipeline 进行持续交付](#)。

- Amazon ECS

在集群名称中，键入或选择现有 Amazon ECS 集群的名称。在服务名称中，键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。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。

有关部署基于容器的应用程序的教程，请参阅[教程：使用 AWS CodePipeline 进行持续部署](#)。

## 步骤 5：创建服务角色和查看管道

1. 在 Step 5: Service Role 页面上，执行以下操作之一，然后选择 Next step：

- 在服务角色下拉列表中，选择您已为 AWS CodePipeline 设置的 IAM 服务角色。
- 如果您没有服务角色，请选择创建角色。在介绍要为您创建的角色 IAM 控制台页面上，选择 Allow。在步骤 5: 服务角色页面上，[AWS CodePipeline-Service](#) 将显示在下拉框中。

#### 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更多的 AWS 服务。有关信息，请参阅 [添加其他 AWS 服务的权限 \(p. 182\)](#)。

有关服务角色及其策略语句的更多信息，请参阅[管理 AWS CodePipeline 服务角色 \(p. 180\)](#)。

2. 在 Step 6: Review 页面上，查看您的管道配置，然后选择 Create Pipeline 创建管道，或选择 Previous 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。管道将在创建之后自动开始运行。有关更多信息，请参阅 [在 AWS CodePipeline 中查看管道详细信息和历史记录 \(p. 96\)](#)。有关对管道进行更改的更多信息，请参阅 [在 AWS CodePipeline 中编辑管道 \(p. 90\)](#)。

## 创建管道 (CLI)

要使用 AWS CLI 创建管道，可以创建一个 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。

#### Important

您不能使用 AWS CLI 创建包含合作伙伴操作的管道。您必须使用 AWS CodePipeline 控制台。

有关管道结构的更多信息，请参阅[AWS CodePipeline 管道结构参考 \(p. 202\)](#)和 AWS CodePipeline [API 参考](#)中的 `create-pipeline`。

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。

您需要在[AWS CodePipeline 入门 \(p. 8\)](#)中为 AWS CodePipeline 创建的服务角色的 ARN 以及将存储管道项目的 Amazon S3 存储桶的名称。该存储桶必须与管道位于同一区域。在运行 `create-pipeline` 命令时，您将使用 ARN 和存储桶名称。与控制台不同，在 AWS CLI 中运行 `create-pipeline` 命令不会创建用于存储项目的 Amazon 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, or Unix) 或命令提示符 (Windows) 处，在本地目录中创建新的文本文件。
2. 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：
  - 用于存储此管道的项目的 Amazon 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。

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
},
"metadata": {
  "pipelineArn": "arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline",
  "updated": 1501626591.112,
  "created": 1501626591.112
}
```



```
}  
}
```

确保按如下所示设置 JSON 文件中的 `PollForSourceChanges` 参数：

```
"PollForSourceChanges": "false",
```

AWS CodePipeline 使用 Amazon CloudWatch Events 检测 AWS CodeCommit 源存储库和分支或 Amazon S3 源存储桶中的更改。AWS CodePipeline 使用 Webhook 检测 GitHub 源存储库和分支中的更改。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，当使用建议的更改检测方法时不需要定期检查。

3. 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。

## 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。

要使用名为 `pipeline.json` 的 JSON 文件 (包含名称 `"MySecondPipeline"` 以作为 JSON 文件中的 `name` 值) 创建名为 `MySecondPipeline` 的管道，命令应类似于以下内容：

```
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. 要查看您刚创建的管道，可以打开 AWS CodePipeline 控制台并从管道列表中选择该管道，也可以使用 `get-pipeline-state` 命令。有关更多信息，请参阅 [在 AWS CodePipeline 中查看管道详细信息和历史记录 \(p. 96\)](#)。
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：
  - 对于带 AWS CodeCommit 存储库的管道，必须手动创建 CloudWatch Events 规则，如 [创建启动 AWS CodeCommit 管道的 CloudWatch Events 规则 \(CLI\) \(p. 67\)](#) 中所述。
  - 对于带 Amazon S3 源的管道，必须手动创建 CloudWatch Events 规则和 AWS CloudTrail 跟踪，如 [使用 CloudWatch Events 规则自动启动 Amazon S3 管道 \(p. 69\)](#) 中所述。
  - 对于带 GitHub 源的管道，必须手动创建 Webhook，如 [使用 Webhook 自动启动 GitHub 管道 \(p. 75\)](#) 中所述。

## 在 AWS CodePipeline 中编辑管道

管道描述您希望 AWS CodePipeline 遵循的发布过程，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。

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 [在 AWS CodePipeline 中手动启动管道 \(p. 80\)](#)。

当推送源代码更改时，AWS CodePipeline 使用更改检测方法以启动您的管道。这些检测方法基于源类型：

- AWS CodePipeline 使用 Amazon CloudWatch Events 检测 AWS CodeCommit 源存储库和分支或 Amazon S3 源存储桶中的更改。
- AWS CodePipeline 使用 Webhook 检测 GitHub 源存储库和分支中的更改。



## 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。如果使用 AWS CLI 创建管道，您必须自行创建其他资源。有关创建或更新 AWS CodeCommit 管道的更多信息，请参阅[创建启动 AWS CodeCommit 管道的 CloudWatch Events 规则 \(CLI\)](#) (p. 67)。有关使用 CLI 创建或更新 Amazon S3 管道的更多信息，请参阅[创建启动您的 Amazon S3 管道的 CloudWatch Events 规则 \(CLI\)](#) (p. 71)。有关创建或更新 GitHub 管道的更多信息，请参阅[使用 Webhook 自动启动 GitHub 管道](#) (p. 75)。

## 主题

- [编辑管道 \(控制台\)](#) (p. 91)
- [编辑管道 \(AWS CLI\)](#) (p. 93)

# 编辑管道 (控制台)

您可以使用 AWS CodePipeline 控制台来添加、编辑或删除管道中的阶段，以及添加、编辑或删除某个阶段中的操作。

AWS CodePipeline 使用 Amazon CloudWatch Events 检测 AWS CodeCommit 源存储库和分支或 Amazon S3 源存储桶中的更改。

## Note

在使用控制台编辑具有 AWS 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。如果使用 AWS CLI 编辑管道，您必须自行创建 Amazon CloudWatch Events 规则和 IAM 角色。有关更多信息，请参阅[使用 CloudWatch Events 规则自动启动 AWS CodeCommit 管道](#) (p. 64)。

## 编辑管道

1. 登录 AWS 管理控制台并通过以下网址打开 AWS CodePipeline 控制台：<http://console.aws.amazon.com/codepipeline>。

所有与您的 AWS 账户关联的管道的名称将会显示。


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 Edit。
4. 在 Edit 页面上，执行以下操作之一：

•

要编辑阶段，请选择该阶段上的编辑图标 ()。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 ()。

要向阶段中添加作为生成操作或测试操作的 AWS CodeBuild，请参阅[AWS CodeBuild 用户指南](#)中的[将 AWS CodePipeline 与 AWS CodeBuild 结合使用以测试代码和运行生成](#)。

- 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。
  - 如果您通过添加 AWS CodeCommit 存储库名称和分支来编辑源操作，则将显示一条消息，指明将为此管道创建 Amazon CloudWatch Events 规则。如果您删除 AWS CodeCommit 源，则会显示一条消息，指明将删除 Amazon CloudWatch Events 规则。

- 如果您通过添加 Amazon S3 源存储桶来编辑源操作，则将显示一条消息，指明将为此管道创建 Amazon CloudWatch Events 规则和 AWS CloudTrail 跟踪。如果您删除 Amazon S3 源，则会显示一条消息，指明将删除 Amazon CloudWatch Events 规则和 AWS CloudTrail 跟踪。如果 AWS CloudTrail 跟踪正由其他管道使用，则不会删除跟踪，但将删除数据事件。
- 如果您通过添加 GitHub 源来编辑源操作，则将为管道添加以下内容：
  - AWS CodePipeline 使用 OAuth 令牌创建由 AWS CodePipeline 管理的授权应用程序。

#### Note

在 GitHub 中，可用于应用程序 (如 AWS CodePipeline) 的 OAuth 令牌的数量有限制。如果您超过此限制，请重试连接以允许 AWS CodePipeline 通过重用现有令牌来重新连接。有关更多信息，请参阅 [配置管道以使用 GitHub 的个人访问令牌 \(p. 167\)](#)。


- AWS CodePipeline 在 GitHub 中创建一个 Webhook 以检测源更改，然后在发生更改时启动您的管道。AWS CodePipeline 将创建以下内容以及 Webhook：
  - 随机生成一个密钥，并使用它来授权连接到 GitHub。
  - 使用该区域的公有终端节点生成 Webhook URL。
  - 该 Webhook 注册到 GitHub。这将订阅 URL 以接收存储库事件。

如果您删除 GitHub 源操作，则会取消注册并删除 Webhook。

- 要添加阶段，请在管道中您要添加阶段的时间点选择 + Stage。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。

例如，如果您想要向管道中的阶段添加操作：

1.

在您要添加操作的阶段中，选择编辑图标 ()，然后选择 + Action 来添加您的操作。

#### Note

要添加与另一个操作并行运行的操作，请选择该操作旁边的添加操作图标。要在另一个操作之前运行您的操作，请选择该操作上方的添加操作图标。要在该阶段中的另一个操作之后运行您的操作，请选择该操作下方的添加操作图标。

2. 在 Add action (添加操作) 面板的 Action category (操作类别) 中，选择操作的类别，如生成或部署。在系统下拉列表中，选择系统，然后提供操作配置的所有必需的详细信息 (如操作名称、提供程序和输入或输出项目)，具体取决于操作类型。标记为星号 (\*) 的项目都是必填的。有关针对 AWS CodePipeline 中的操作的要求的更多信息 (包括输入和输出项目的名称以及如何使用这些名称)，请参阅 [AWS CodePipeline 中的操作结构要求 \(p. 203\)](#)。

#### Note

有些操作提供程序 (如 GitHub) 要求您连接到该提供程序的网站，然后您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。不要使用您的 AWS 凭证。

3. 配置完操作后，请选择 Add action。

#### Note

您无法在控制台视图中对操作或阶段进行重命名。您可以添加具有您要更改的名称的阶段或操作，然后删除旧阶段或操作。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。

5. 编辑完您的管道后，请选择 Save pipeline changes 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。

6. 要测试您的操作，请选择 **发布更改** 来处理通过管道的提交，将更改提交至管道的源阶段中指定的源，或按照在 [AWS CodePipeline 中手动启动管道 \(p. 80\)](#) 中的步骤使用 AWS CLI 手动发布更改。

## 编辑管道 (AWS CLI)

您可以使用 `update-pipeline` 命令来编辑管道。

### Important

虽然您可以使用 AWS CLI 来编辑包含合作伙伴操作的管道，但不得手动编辑合作伙伴操作本身的 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, or Unix) 或命令提示符 (Windows)，然后运行 `get-pipeline` 命令，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，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 [AWS CodePipeline 管道结构参考 \(p. 202\)](#)。

```
{
  "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
    }
  ]
}
]
}

```

以下示例介绍如何添加将 GitHub 存储库用作源操作的源阶段。有关 AWS CodePipeline 如何与 GitHub 集成的更多信息，请参阅[源操作集成](#) (p. 10)。

#### 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 [AWS CodePipeline 管道结构参考](#) (p. 202)。

```

{
  "name": "Source",
  "actions": [
    {
      "inputArtifacts": [],
      "name": "Source",
      "actionTypeId": {
        "category": "Source",
        "owner": "ThirdParty",
        "provider": "GitHub",
        "version": "1"
      },
      "outputArtifacts": [
        {
          "name": "MyApp"
        }
      ],
      "configuration": {
        "Owner": "MyGitHubAccountName",
        "Repo": "MyGitHubRepositoryName",
        "PollForSourceChanges": "false",
        "Branch": "master",
        "OAuthToken": "****"
      },
      "runOrder": 1
    }
  ]
}

```

```
    }  
  ],  
},
```

OAuthToken 的值保持遮蔽状态，因为 AWS CodePipeline 使用它来访问 GitHub 存储库。您可以将个人访问令牌用于此值。要创建个人访问令牌，请参阅[管道错误：我收到一个管道错误，其中包含以下消息：“PermissionError: Could not access the GitHub repository”](#) (p. 167)。

#### Note

某些编辑 (例如，将操作从一个阶段移动到另一个阶段) 删除操作的上次已知状态历史记录。此外，如果管道包含一个或多个密钥参数 (如操作的 OAuth 令牌)，将使用一系列星号 (\*\*\*\*) 遮盖该令牌。这些密钥参数保持不变，除非您编辑管道的该部分 (例如，如果您更改包含 OAuth 令牌的操作名称，或者更改包含使用 OAuth 令牌的操作的阶段名称)。如果您所做的更改影响了包含 OAuth 令牌的操作，您必须将令牌值包含在编辑后的 JSON 中。有关更多信息，请参阅[AWS CodePipeline 管道结构参考](#) (p. 202)。安全最佳实践是定期轮换您的个人访问令牌。有关更多信息，请参阅[使用 GitHub 和 AWS CodePipeline CLI 创建并定期轮换您的 GitHub 个人访问令牌](#) (p. 198)。

有关使用 CLI 在管道中添加审批操作的信息，请参阅[向 AWS CodePipeline 中的管道添加手动审批操作](#) (p. 142)。

确保按如下所示设置 JSON 文件中的 PollForSourceChanges 参数：

```
"PollForSourceChanges": "false",
```

AWS CodePipeline 使用 Amazon CloudWatch Events 检测 AWS CodeCommit 源存储库和分支或 Amazon S3 源存储桶中的更改。AWS CodePipeline 使用 Webhook 检测 GitHub 源存储库和分支中的更改。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。

- 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 行来修改 JSON 文件中的结构，否则 update-pipeline 命令将无法使用管道结构。从 JSON 文件中的管道结构中删除该部分 ( "metadata": { } 行以及其中的“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"  
}
```

保存文件。

- 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：
  - 如果您使用 CLI 通过添加 AWS CodeCommit 存储库来编辑管道，则必须创建 CloudWatch Events 规则，如[创建启动 AWS CodeCommit 管道的 CloudWatch Events 规则 \(CLI\)](#) (p. 67) 中所述。
  - 如果您使用 CLI 通过添加 Amazon S3 源来编辑管道，则必须创建 CloudWatch Events 规则和 AWS CloudTrail 跟踪，如[使用 CloudWatch Events 规则自动启动 Amazon S3 管道](#) (p. 69) 中所述。
  - 如果您使用 CLI 通过添加 GitHub 源来编辑管道，则必须创建 Webhook，如[使用 Webhook 自动启动 GitHub 管道](#) (p. 75) 中所述。
- 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:/API 版本命令需要该项。

```
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。

6. 打开 AWS CodePipeline 控制台，然后从列表中选择您刚刚编辑的管道。

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。

7. 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 [在 AWS CodePipeline 中手动启动管道 \(p. 80\)](#)。

有关管道结构和预期值的更多信息，请参阅[AWS CodePipeline 管道结构参考 \(p. 202\)](#)和 [AWS CodePipeline API 参考](#)。

## 在 AWS CodePipeline 中查看管道详细信息和历史记录

可以使用 AWS CodePipeline 控制台或 AWS CLI 查看与您的 AWS 账户关联的管道的详细信息。

#### 主题

- [查看管道详细信息和历史记录 \(控制台\) \(p. 96\)](#)
- [查看管道详细信息和历史记录 \(CLI\) \(p. 98\)](#)

### 查看管道详细信息和历史记录 (控制台)

您可以使用 AWS CodePipeline 控制台来查看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。执行历史记录限制为最近 12 个月。

#### 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

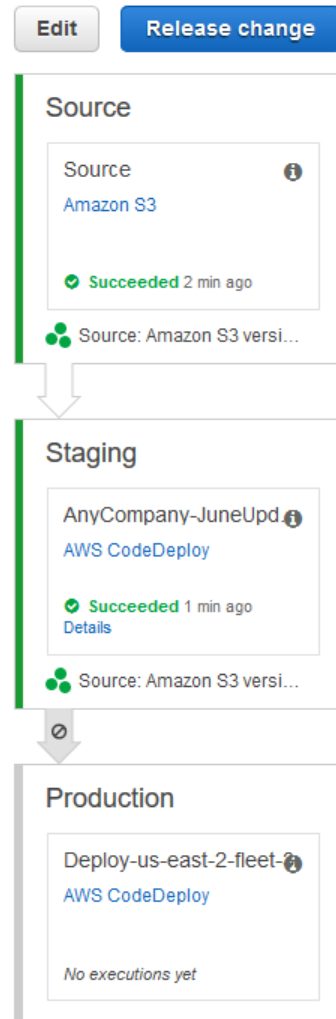
1. 登录 AWS 管理控制台并通过以下网址打开 AWS CodePipeline 控制台：<http://console.aws.amazon.com/codepipeline>。

此时会显示与您的 AWS 账户相关联的所有管道的名称和创建日期，以及一个可用于查看管道执行历史记录的链接。

2. 要查看单个管道的详细信息，在 (Name) 中，选择该管道。您将看到管道的详细视图，包括每个阶段中每个操作的状态和过渡的状态。

## AnyCompanyPipeline [View pipeline history](#)

View progress and manage your pipeline.



图形视图会显示每个阶段的以下信息：

- 阶段的名称。
- 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。已启用的过渡由单色箭头表示。已禁用的过渡由灰色箭头和图标表示。
- 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。
  - 操作的提供程序，如 AWS 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)。
  - 关于贯穿阶段中最新管道执行的源修订的详细信息，或者关于 (对于 AWS CodeDeploy 部署) 部署到目标实例的最新源修订的详细信息。
3. 要查看管道的某个阶段的操作的配置详细信息，请选择或将鼠标悬停在操作旁边的信息图标上。
  4. 要查看操作提供程序的详细信息，请选择提供程序。例如，在前面的示例管道中，如果您在 Staging 或 Production 阶段选择 AWS CodeDeploy，将显示为该阶段配置的部署组的 AWS CodeDeploy 控制台页面。
  5. 要查看阶段中某个操作的进度详细信息，请在正在进行的操作旁边显示 Details 时选择该选项 (由 In Progress 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。

#### Note

提供可从 GitHub 存储库检索内容的源操作的详细信息，但不提供可从 Amazon S3 存储桶或 AWS CodeCommit 存储库中检索内容的源操作的详细信息。

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. 要获得有关阶段中已完成操作的错误或失败的更多信息，请选择 Details。您将看到操作最后一次运行的详细信息，包括操作的结果：Succeeded 或 Failed。
9. 要查看用于阶段中最新管道执行的源项目的详细信息 (管道第一阶段中出现的输出项目)，请单击阶段底部的详细信息区域。您可以查看关于标识符的详细信息，如提交 ID、签入注释、自创建或更新项目以来的时间。有关更多信息，请参阅 [查看 AWS CodePipeline 管道中的当前源修订详细信息 \(p. 162\)](#)。
10. 要查看管道的最近执行的详细信息，请选择管道名称旁边的 View pipeline history。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。

## 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，用于查看与您的 AWS 账户相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，包括有关项目的详细信息、管道执行 ID 以及管道的名称、版本和状态。

有关使用 CLI 查看某阶段的最新管道执行中使用的源修订的详细信息，请参阅[查看管道中的当前源修订详细信息 \(CLI\) \(p. 163\)](#)。

### 查看管道

1. 打开终端 (Linux, macOS, or Unix) 或命令提示符 (Windows)，并使用 AWS CLI 运行 `list-pipelines` 命令，如下所示：

```
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 *MyFirstPipeline* 的管道的详细信息，您可以键入以下内容：

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

该命令会返回管道结构。

3. 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 *MyFirstPipeline* 的管道的当前状态的详细信息，您可以键入以下内容：

```
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。

以下示例显示名为 *MyFirstPipeline* 的三个阶段管道的返回数据，其中前两个阶段和操作显示成功，第三个阶段显示失败，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",
            "code": "JobFailed",
            "lastStatusChange": 1427246155.648
        },
        "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"
}
]
}
```

4. 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 *MyFirstPipeline* 的管道的当前状态的详细信息，您可以键入以下内容：

```
aws codepipeline list-pipeline-executions --pipeline-name MyFirstPipeline
```

该命令返回最近 12 个月已记录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。

以下示例显示拥有三次执行的名 *MyFirstPipeline* 管道的返回数据。

```
{
  "pipelineExecutionSummaries": [
    {
      "lastUpdateTime": 1496380678.648,
      "pipelineExecutionId": "7cf7f7cb-3137-539g-j458-d7eu3EXAMPLE",
      "startTime": 1496380258.243,
      "status": "Succeeded"
    },
    {
      "lastUpdateTime": 1496591045.634,
      "pipelineExecutionId": "3137f7cb-8d494hj4-039j-d84l-d7eu3EXAMPLE",
      "startTime": 1496590401.222,
      "status": "Succeeded"
    },
    {
      "lastUpdateTime": 1496946071.6456,
      "pipelineExecutionId": "4992f7jf-7cf7-913k-k334-d7eu3EXAMPLE",
      "startTime": 1496945471.5645,
      "status": "Succeeded"
    }
  ]
}
```

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，您可以键入以下内容：

```
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。

以下示例显示名为 *MyFirstPipeline* 的管道的返回数据：

```
{
  "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"
      }
    ]
  }
}
```

## 在 AWS CodePipeline 中删除管道

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。可以使用 AWS CodePipeline 控制台或使用 AWS CLI 与 `delete-pipeline` 命令删除管道。

### 主题

- [删除管道 \(控制台\) \(p. 101\)](#)
- [删除管道 \(CLI\) \(p. 101\)](#)

## 删除管道 (控制台)

要在 AWS CodePipeline 控制台中删除管道

1. 登录 AWS 管理控制台并通过以下网址打开 AWS CodePipeline 控制台：<http://console.aws.amazon.com/codepipeline>。
2. 所有与您的 AWS 账户关联的管道的名称及其状态将会显示。
3. 在 Name 中，选择您要删除的管道的名称。这将打开管道的详细视图。
4. 在管道详细信息页面上，选择 Edit。这会打开管道的编辑页面。
5. 在 Edit 页面上，选择 Delete。
6. 键入管道的名称，然后选择 Delete。

### Important

此操作无法撤销。

## 删除管道 (CLI)

要使用 AWS CLI 手动删除管道，请使用 `delete-pipeline` 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。

## 要使用 AWS CLI 删除管道

1. 打开终端 (Linux, macOS, or Unix) 或命令提示符 (Windows), 并使用 AWS CLI 运行 delete-pipeline 命令, 指定您要删除的管道的名称。例如, 要删除名为 *MyFirstPipeline* 的管道:

```
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所使用的资源, 比如您用于部署代码的 AWS CodeDeploy 或 Elastic Beanstalk 应用程序, 或者如果您从 AWS CodePipeline 控制台创建了管道, 包括创建用于存储您的管道项目的 Amazon S3 存储桶 AWS CodePipeline。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, AWS CodePipeline 会创建一个 Amazon S3 存储桶, 用于存储您的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 AWS CodePipeline 中创建使用另一个 AWS 账户的资源的管道

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, 您可能希望将一个账户用于您的管道, 将另一个账户用于您的 AWS CodeDeploy 资源。要这样做, 您必须创建一个要使用的 AWS Key Management Service (AWS KMS) 密钥, 将该密钥添加到管道, 并设置账户策略和角色以启用跨账户访问。

### Note

源操作不能使用来自其他 AWS 账户的 Amazon S3 存储桶。

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 存储桶以及 AWS CodePipeline 使用的服务角色。*AccountB* 是最初用于创建 AWS CodeDeploy 应用程序、部署组以及 AWS CodeDeploy 使用的服务角色的账户。

要使 *AccountA* 能够编辑管道以使用 *AccountB* 创建的 AWS CodeDeploy 应用程序, *AccountA* 必须:

- 请求 *AccountB* 的 ARN 或账户 ID (在本演练中, *AccountB* ID 为 *012ID\_ACCOUNT\_B*)。
- 在管道的区域中创建或使用客户管理的 AWS KMS 密钥, 并向服务角色 (*AWS-CodePipeline-Service*) 和 *AccountB* 授予使用该密钥的权限。
- 创建一个 Amazon S3 存储桶策略, 以授予 *AccountB* 访问该 Amazon S3 存储桶的权限 (例如 *codepipeline-us-east-2-1234567890*)。
- 创建一个允许 *AccountA* 担任由 *AccountB* 配置的角色策略, 并将该策略附加到服务角色 (*AWS-CodePipeline-Service*)。
- 编辑管道以使用客户管理的 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*)。
- 创建一个应用于为 AWS CodeDeploy 配置的 *Amazon EC2 实例角色* 的策略, 以允许访问 Amazon S3 存储桶 (*codepipeline-us-east-2-1234567890*)。
- 创建一个应用于为 AWS CodeDeploy 配置的 *Amazon EC2 实例角色* 的策略, 以允许访问用于加密 *AccountA* 中的管道项目的客户管理的 AWS KMS 密钥。

- 配置一个 IAM 角色 (`CrossAccount_Role`)，并将允许 `AccountA` 担任该角色的信任关系策略附加到该角色。
- 创建一个允许访问管道所需的部署资源的策略，并将其附加到 `CrossAccount_Role`。
- 创建一个允许访问 Amazon S3 存储桶 (`codepipeline-us-east-2-1234567890`) 的策略，并将其附加到 `CrossAccount_Role`。

#### 主题

- 先决条件：创建 AWS KMS 加密密钥 (p. 103)
- 步骤 1：设置账户策略和角色 (p. 103)
- 步骤 2：编辑管道 (p. 108)

## 先决条件：创建 AWS KMS 加密密钥

与所有 AWS KMS 密钥一样，客户管理的密钥也是特定于区域的。您必须在创建管道的同一区域 (例如 `us-east-2`) 创建客户管理的 AWS KMS 密钥。

#### Note

有关 AWS CodePipeline 可用的区域和终端节点的更多信息，请参阅[区域和终端节点](#)。

#### 要在 AWS KMS 中创建客户托管密钥

1. 使用 `AccountA` 登录 AWS 管理控制台，并通过以下网址打开 IAM 控制台：<https://console.aws.amazon.com/iam/>。
2. 在 Dashboard 中，选择 Encryption keys。
3. 在 Encryption keys 中的 Filter 中，确保所选区域与创建管道的区域相同，然后选择 Create key。

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 )。

4. 在 Alias 中，键入要为该密钥使用的别名 (例如 `PipelineName-Key`)。或者提供有关该密钥的描述，然后选择 Next Step。
5. 在 Define Key Administrative Permissions 中，选择您的 IAM 用户和任何您希望其担任此密钥管理员的用户或组，然后选择 Next Step。
6. 在 Define Key Usage Permissions 中的 This Account 下，选择管道服务角色的名称 (例如 `AWS-CodePipeline-Service`)。在 External Accounts 下，选择 Add an External Account。键入 `AccountB` 的账户 ID 以完成 ARN，然后选择 Next Step。
7. 在 Preview Key Policy 中查看此策略，然后选择 Finish。
8. 从密钥列表中，选择您的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的策略。这需要 `AccountA` 和 `AccountB` 均采取操作。

#### 主题

- 在将要创建管道的账户中配置策略和角色 (`AccountA`) (p. 104)
- 在拥有 AWS 资源的账户中配置策略和角色 (`AccountB`) (p. 105)

## 在将要创建管道的账户中配置策略和角色 (*AccountA*)

要创建一个使用与另一个 AWS 账户相关联的 AWS CodeDeploy 资源的管道，*AccountA* 必须为用于存储项目的 Amazon S3 存储桶和 AWS CodePipeline 服务角色配置策略。

要为授予 *AccountB* (控制台) 访问权限的 Amazon S3 存储桶创建策略

1. 使用 *AccountA* 登录 AWS 管理控制台，并通过以下网址打开 Amazon S3 控制台：<https://console.aws.amazon.com/s3/>。
2. 在 Amazon S3 存储桶列表中，选择用于存储您的管道项目的 Amazon S3 存储桶。该存储桶被命名为 `codepipeline-region-1234567EXAMPLE`，其中 *region* 是您创建管道的 AWS 区域，`1234567EXAMPLE` 是一个十位随机数，可确保存储桶名称是唯一的 (例如 `codepipeline-us-east-2-1234567890`)。
3. 在 Amazon S3 存储桶的详细信息页面上，选择 Properties。
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 存储桶，请选择 Edit bucket policy。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操作。有关更多信息，请参阅 [IAM 策略概述](#)。

5. 在 Bucket Policy Editor 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`。Amazon 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账户的 ARN 和 Amazon 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
        }
      }
    }
  ],
  {
    "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 Save 以保存 Amazon S3 存储桶的权限。

### 要为 AWS CodePipeline (控制台) 的服务角色创建策略

1. 使用 **AccountA** 登录 AWS 管理控制台，并通过以下网址打开 IAM 控制台：<https://console.aws.amazon.com/iam/>。
2. 在 Dashboard 中，选择 Roles。
3. 在角色列表中，在 Role Name 下，选择 AWS CodePipeline 的服务角色的名称。默认情况下，该名称为 AWS-CodePipeline-Service。如果您为您的服务角色使用了其他名称，请确保从列表中选择它。
4. 在 Summary 页面的 Permissions 选项卡中，展开 Inline Policies，然后选择 Create Role Policy。

#### Note

如果您之前未创建任何角色策略，Create Role Policy 将不会显示。请选择链接以创建一个新的策略。

5. 在 Set Permissions 中，选择 Custom Policy，然后选择 Select。
6. 在 Review Policy 页面上，在 Policy Name 中键入策略的名称。在 Policy Document 中，键入以下策略以允许 **AccountB** 担任该角色。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/*"
    ]
  }
}
```

7. 选择 Validate Policy。
8. 验证完策略后，请选择 Apply Policy。

## 在拥有 AWS 资源的账户中配置策略和角色 (**AccountB**)

在 AWS CodeDeploy 中创建应用程序、部署和部署组时，还应创建一个 **Amazon EC2 实例角色**。(如果您使用“Run Deployment Walkthrough”向导，系统将为您创建此角色，但您也可以手动创建此角色。)要使在 **AccountA** 中创建的管道能够使用在 **AccountB** 中创建的 AWS CodeDeploy 资源，您必须：

- 为实例角色配置策略，允许它访问存储管道项目的 Amazon S3 存储桶。
- 在 **AccountB** 中创建第二个角色，并配置为跨账户访问。

第二个角色必须不仅可以访问 **AccountA** 中的 Amazon S3 存储桶，而且还包含允许访问 AWS CodeDeploy 资源的策略和允许 **AccountA** 担任该角色的信任关系策略。

#### Note

这些策略专门用于设置 AWS CodeDeploy 资源以便在使用不同 AWS 账户创建的管道中使用。其他 AWS 资源将需要特定于其资源需求的策略。

### 要为针对 AWS CodeDeploy (控制台) 配置的 Amazon EC2 实例创建策略

1. 使用 **AccountB** 登录 AWS 管理控制台，并通过以下网址打开 IAM 控制台：<https://console.aws.amazon.com/iam/>。
2. 在 Dashboard 中，选择 Roles。
3. 在角色列表中，在 Role Name 下，选择用作 AWS 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 [Amazon EC2 实例创建 IAM 实例配置文件](#)。
4. 在 Summary 页面的 Permissions 选项卡中，展开 Inline Policies，然后选择 Create Role Policy。
5. 在 Set Permissions 中，选择 Custom Policy，然后选择 Select。
6. 在 Review Policy 页面上，在 Policy Name 中键入策略的名称。在 Policy Document 中，键入以下策略授予对 **AccountA** 用于存储管道项目的 Amazon S3 存储桶的访问权限 (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"
      ]
    }
  ]
}
```

7. 选择 Validate Policy。
8. 验证完策略后，请选择 Apply Policy。
9. 为 AWS KMS 创建第二个策略，其中 **arn:aws:kms:us-east-1:012ID\_ACCOUNT\_A:key/2222222-3333333-4444-556677EXAMPLE** 是在 **AccountA** 中创建并配置为允许 **AccountB** 使用它的客户托管密钥的 ARN：

```
{
  "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

您必须在此策略中使用 **AccountA** 的账户 ID 作为 AWS KMS 密钥的资源 ARN 的一部分，如此处所示，否则此策略将不起作用。

10. 选择 Validate Policy。
11. 验证完策略后，请选择 Apply Policy。

现在创建一个 IAM 角色用于跨账户访问，并进行配置，以便 **AccountA** 可以担任该角色。此角色必须包含允许访问 **AccountA** 中的 AWS CodeDeploy 资源和用于存储项目的 Amazon S3 存储桶的策略。

### 要在 IAM 中配置跨账户角色

1. 使用 **AccountB** 登录 AWS 管理控制台，并通过以下网址打开 IAM 控制台：<https://console.aws.amazon.com/iam/>。
2. 在 Dashboard 中，选择 Roles，然后选择 Create New Role。
3. 在 Set New Role 页面上，在 Role Name 中键入该角色的名称 (例如 **CrossAccount\_Role**)。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。
4. 在 Select Role Type 页面上，选择 Role for Cross-Account Access。在 Provide access between AWS accounts you own 旁边，选择 Select。
5. 键入要在 AWS CodePipeline 中创建管道的账户 (**AccountA**) 的 AWS 账户 ID，然后选择 Next Step。

### Note

此步骤将在 **AccountB** 与 **AccountA** 之间创建信任关系策略。

6. 在 Attach Policy 中，选择 AmazonS3ReadOnlyAccess，然后选择 Next Step。

### 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。

7. 在 Review 页面上，选择 Create Role。
8. 从角色列表中，选择您刚刚创建的策略 (例如 **CrossAccount\_Role**) 以打开该角色的 Summary 页面。
9. 展开 Permissions，然后展开 Inline Policies。选择链接以创建一个内联策略。
10. 在 Set Permissions 中，选择 Custom Policy，然后选择 Select。
11. 在 Review Policy 页面上，在 Policy Name 中键入策略的名称。在 Policy Document 中，键入以下策略以允许访问 AWS CodeDeploy 资源：

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

```
{
  "Effect": "Allow",
  "Action": [
    "codedeploy:CreateDeployment",
    "codedeploy:GetDeployment",
    "codedeploy:GetDeploymentConfig",
    "codedeploy:GetApplicationRevision",
    "codedeploy:RegisterApplicationRevision"
  ],
  "Resource": "*"
}
```

12. 选择 Validate Policy。
13. 验证完策略后，请选择 Apply Policy。
14. 在 Inline Policies 中，选择 Create Role Policy。
15. 在 Set Permissions 中，选择 Custom Policy，然后选择 Select。
16. 在 Review Policy 页面上，在 Policy Name 中键入策略的名称。在 Policy Document 中，键入以下策略以允许该角色从 **AccountA** 的 Amazon S3 存储桶中检索输入项目，或者将输出项目放入该存储桶中。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject*",
        "s3:PutObject",
        "s3:PutObjectAcl",
        "codecommit:ListBranches",
        "codecommit:ListRepositories"
      ],
      "Resource": [
        "arn:aws:s3:::codepipeline-us-east-2-1234567890/*"
      ]
    }
  ]
}
```

17. 选择 Validate Policy。
18. 验证完策略后，请选择 Apply Policy。
19. 在 Managed Policies 中，在 Policy Name 下的策略列表中找到 AmazonS3ReadOnlyAccess，然后选择 Detach Policy。系统提示时，选择 Detach。

## 步骤 2：编辑管道

您不能使用 AWS CodePipeline 控制台来创建或编辑使用与另一个 AWS 账户相关联的资源的管道。但是，您可以使用控制台创建管道的一般结构，然后使用 AWS CLI 编辑管道并添加这些资源。或者，您也可以使用现有管道的结构，并手动添加资源。

要添加与另一个 AWS 账户 (AWS CLI) 关联的资源

1. 在终端 (Linux, macOS, or 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 MyFirstPipeline 的管道，您应键入类似以下的内容：

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

输出将会发送到 `pipeline.json` 文件。

2. 在任何纯文本编辑器中打开 JSON 文件。在项目存储中的 `"type": "S3"` 之后，添加 KMS encryptionKey、ID 并键入信息，其中 `codepipeline-us-east-2-1234567890` 是用于存储管道项目的 Amazon S3 存储桶的名称，`arn:aws:kms:us-east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE` 是您刚刚创建的客户托管密钥的 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. 在一个阶段中添加部署操作，以使用与 `AccountB` 关联的 AWS CodeDeploy 资源，包括您创建的跨账户角色 (`CrossAccount_Role`) 的 `roleArn` 值。

以下示例显示添加了一个名为 `ExternalDeploy` 的部署操作的 JSON。它使用在名为 `Staging` 的阶段中在 `AccountB` 中创建的 AWS CodeDeploy 资源。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. 保存文件。
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。

### 要测试使用与另一个 AWS 账户关联的资源的管道

1. 在终端 (Linux, macOS, or 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 在 [AWS CodePipeline 中手动启动管道](#) (p. 80)。

2. 使用 **AccountA** 登录 AWS 管理控制台，并通过以下网址打开 AWS CodePipeline 控制台：<http://console.aws.amazon.com/codepipeline>。

所有与您的 AWS 账户关联的管道的名称将会显示。

3. 在 Name 中，选择您刚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。等待使用与另一个 AWS 账户关联的资源的操作显示成功消息。

### 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 **AccountB** 登录以查看 AWS CodeDeploy 中的部署详细信息。

# 使用 AWS CodePipeline 中的操作

在 AWS CodePipeline 中，操作是管道某个阶段中序列的一部分。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。

AWS CodePipeline 支持六种操作类型：

- Source
- 生成
- 测试
- 部署
- 审批
- Invoke

有关可基于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 AWS CodePipeline 操作类型集成 \(p. 10\)](#)。

主题

- [在 AWS CodePipeline 中创建和添加自定义操作 \(p. 111\)](#)
- [在 AWS CodePipeline 管道中调用 AWS Lambda 函数 \(p. 120\)](#)
- [在 AWS CodePipeline 中重试失败的操作 \(p. 136\)](#)
- [管理 AWS CodePipeline 中的审批操作 \(p. 139\)](#)

## 在 AWS CodePipeline 中创建和添加自定义操作

AWS CodePipeline 包括一些帮助您配置生成、测试和部署资源以实现自动化发布过程的操作。如果您的发布过程包括默认操作中不包含的活动，例如内部开发的生成过程或测试套件，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 AWS CLI 在与 AWS 账户关联的管道中创建自定义操作。

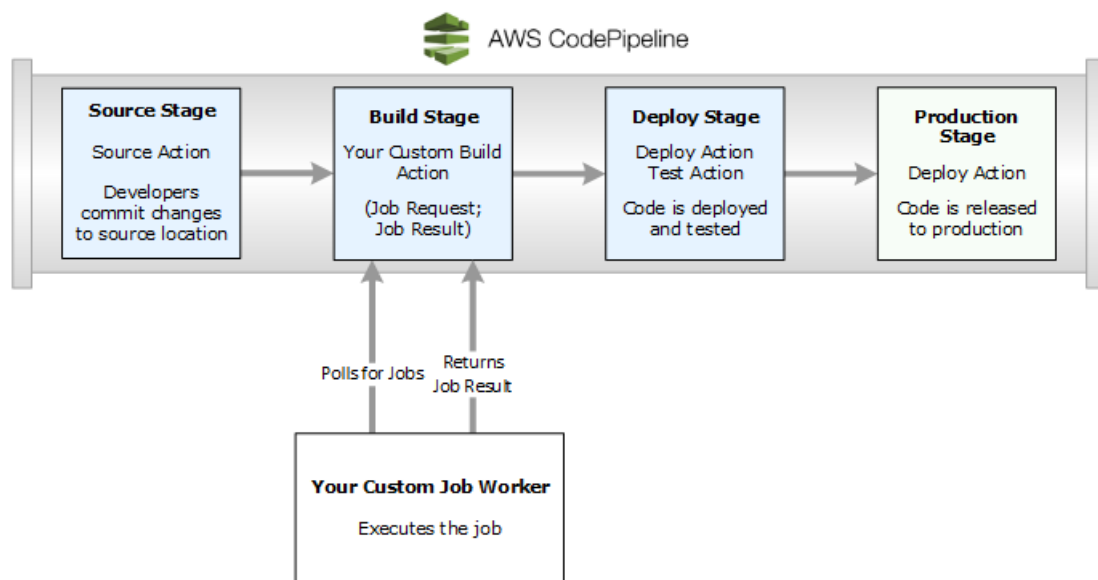
自定义操作分为以下几类：

- 生成或转换项目的生成操作
- 将项目部署到一个或多个服务器、网站或存储库的部署操作
- 配置和运行自动化测试的测试操作
- 运行函数的调用操作

当您创建自定义操作时，还必须创建一个作业辅助角色，它将轮询 AWS CodePipeline 以获取该自定义操作的作业请求，执行该作业，并将状态结果返回给 AWS CodePipeline。该作业辅助角色可以位于任何计算机或资源上，只要它能够访问 AWS CodePipeline 的公有终端节点即可。要轻松管理访问和安全性，请考虑在 Amazon EC2 实例上托管您的作业辅助角色。

下图显示了包含自定义生成操作的管道的概要视图：





当某个管道包含自定义操作作为阶段的一部分时，该管道将创建一个作业请求。自定义作业辅助角色将检测到该请求并执行该作业 (在此示例中为使用第三方生成软件的自定义进程)。操作完成时，作业辅助角色将返回成功或失败结果。如果收到成功结果，管道会将修订及其构件转换到下一项操作。如果返回失败结果，管道则不会将修订转换到管道中的下一项操作。

#### Note

这些说明假设您已经完成了[AWS CodePipeline 入门 \(p. 8\)](#)中的步骤。

#### 主题

- [创建自定义操作 \(CLI\) \(p. 112\)](#)
- [为您的自定义操作创建作业辅助角色 \(p. 115\)](#)
- [向管道添加自定义操作 \(p. 118\)](#)

## 创建自定义操作 (CLI)

### 使用 AWS CLI 创建自定义操作

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：

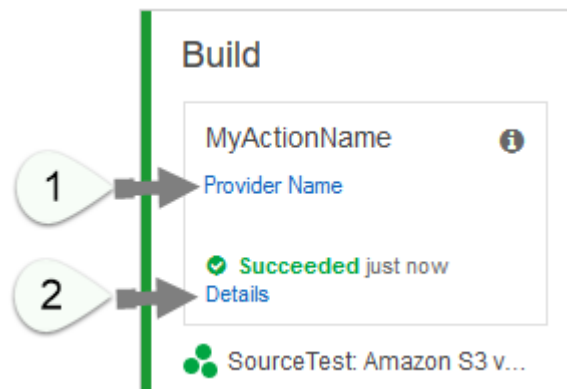
```
{
  "category": "Build",
  "provider": "My-Build-Provider-Name",
  "version": "1",
  "settings": {
    "entityUrlTemplate": "https://my-build-instance/job/{Config:ProjectName}/",
    "executionUrlTemplate": "https://my-build-instance/job/{Config:ProjectName}/lastSuccessfulBuild/{ExternalExecutionId}/"
  },
  "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
}
}
```

您将会注意到 JSON 文件中包含两个属性：`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，在以上示例中，`entityUrlTemplate` 值是指配置属性 `ProjectName`。

- `entityUrlTemplate`：提供有关操作的服务提供程序的信息的静态链接。在示例中，生成系统包含一个指向每个生成项目的静态链接。链接格式可能不同，具体取决于您的生成提供程序（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程序）。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 `ThirdPartyURL` 的更多信息，请参阅 [AWS CodePipeline API 参考](#) 中的 [ActionTypeSettings](#) 和 [CreateCustomActionType](#)。有关操作结构要求以及如何创建操作的更多信息，请参阅 [AWS CodePipeline 管道结构参考](#) (p. 202)。

2. 保存 JSON 文件，并为其指定一个方便您记忆的名称 (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 AWS CLI 运行 `aws codepipeline create-custom-action-type` 命令，指定您刚刚创建的 JSON 文件的名称。

例如，要创建 build 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，`id` 部分包括 `"owner"`：`"Custom"`。AWS CodePipeline 会自动分配 `Custom` 作为自定义操作类型的拥有者。当您使用 `create-custom-action-type` 命令或 `update-pipeline` 命令时，无法分配或更改此值。

## 为您的自定义操作创建作业辅助角色

自定义操作需要一个作业辅助角色来轮询 AWS CodePipeline 以获取该自定义操作的作业请求，执行该作业，并将状态结果返回给 AWS CodePipeline。作业辅助角色可以位于任何计算机或资源上，只要它能够访问 AWS CodePipeline 的公有终端节点即可。

设计作业辅助角色的方法有很多。以下各节针对为 AWS CodePipeline 开发自定义作业辅助角色提供了一些实用的指导。

### 主题

- [为您的作业辅助角色选择和配置权限管理策略 \(p. 115\)](#)
- [为您的自定义操作开发作业辅助角色 \(p. 116\)](#)
- [自定义作业辅助角色架构和示例 \(p. 117\)](#)

## 为您的作业辅助角色选择和配置权限管理策略

要在 AWS CodePipeline 中为您的自定义操作开发自定义作业辅助角色，您需要一个用于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，从而允许您轻松扩展集成所需的资源。您可以使用与 AWS 的内置集成来简化您的自定义作业辅助角色与 AWS CodePipeline 之间的交互。

### 要设置 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 [Amazon EC2 Linux 实例入门](#)。

另一个需要考虑的策略是使用 IAM 的联合身份来集成您的现有身份提供商系统和资源。如果您已经拥有企业身份提供商，或已经配置为支持使用网络身份提供商的用户，则此策略特别有用。联合身份允许您授予对 AWS 资源 (包括 AWS CodePipeline) 的安全访问权限，而不必创建或管理 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。可以使用示例应用程序作为自己的设计的模板。

### 要设置联合身份验证

1. 了解更多有关 IAM 联合身份验证的信息。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](#)
  - [使用 Amazon Cognito 的移动应用程序联合身份验证](#)
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](#)。

要考虑的第三个策略是创建一个 IAM 用户，以便在运行您的自定义操作和作业辅助角色时在您的 AWS 账户下使用。

### 要设置 IAM 用户

1. 在 [IAM 最佳实践和用户案例](#) 中了解更多有关 IAM 最佳实践和用户案例的信息。
2. 按照在您的 [AWS 账户中创建 IAM 用户](#) 中的步骤开始创建 IAM 用户。

以下是可创建用于您的自定义作业辅助角色的示例策略。此策略仅用作示例，并按原样提供。

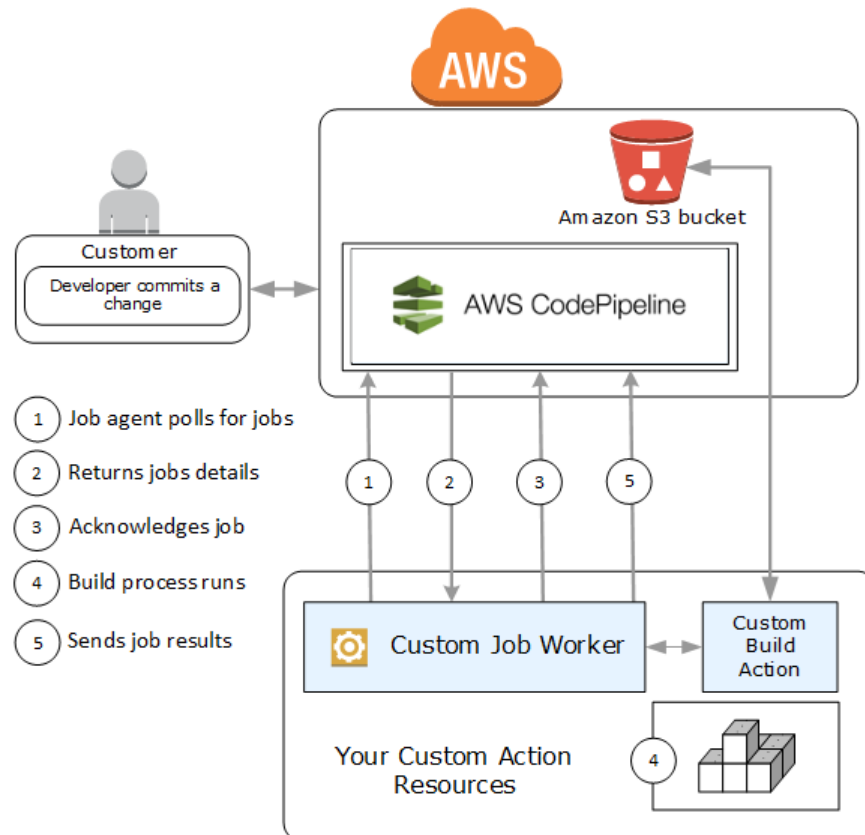
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codepipeline:PollForJobs",
        "codepipeline:AcknowledgeJob",
        "codepipeline:GetJobDetails",
        "codepipeline:PutJobSuccessResult",
        "codepipeline:PutJobFailureResult"
      ],
      "Resource": [
        "arn:aws:codepipeline:us-east-2::actionType:custom/Build/MyBuildProject/1/"
      ]
    }
  ]
}
```

#### Note

考虑对 IAM 用户使用 AWSCodePipelineCustomActionAccess 托管策略。

## 为您的自定义操作开发作业辅助角色

选择权限管理策略后，您应该考虑作业辅助角色如何与 AWS CodePipeline 交互。以下概要图表显示了生成过程的自定义操作和作业辅助角色的工作流程。



1. 作业辅助角色使用 PollForJobs 轮询 AWS CodePipeline 以获取作业。

2. 当管道由其源阶段中的更改触发时 (例如, 当开发人员提交更改时), 自动发布过程即会开始。该过程将继续执行, 直到到达配置您的自定义操作的阶段。在它到达此阶段的操作时, AWS CodePipeline 会将作业排队。如果您的作业辅助角色再次调用 `PollForJobs` 以获取状态, 此作业将会出现。从 `PollForJobs` 获取作业详细信息, 并将其传递回您的作业辅助角色。
3. 作业辅助角色调用 `AcknowledgeJob` 以便向 AWS CodePipeline 发送作业确认。AWS CodePipeline 返回一个确认, 指示作业辅助角色应继续处理该作业 (`InProgress`), 如果您有多个作业辅助角色轮询作业, 而另一个作业辅助角色已经认领了该作业, 则将返回一个 `InvalidNonceException` 错误响应。收到 `InProgress` 确认后, AWS CodePipeline 等待返回结果。
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。在生成自定义操作的示例中, 操作从 Amazon S3 存储桶提取项目, 生成它们, 然后将成功生成的项目推送回 Amazon S3 存储桶。
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的状态的序列化, 例如 JSON 格式的生成标识符或一个 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。

一旦调用 `PutJobSuccessResult`, 作业即视为已完成。将在 AWS CodePipeline 中创建一个包括延续令牌的新作业。如果您的作业辅助角色再次调用 `PollForJobs`, 此作业将会出现。该新作业可用于检查操作的状态, 并随延续令牌一起返回状态, 如果操作已完成, 则返回不带延续令牌的状态。

#### Note

如果您的作业辅助角色执行自定义操作的所有工作, 则您应考虑将您的作业辅助角色处理分为至少两个步骤。第一步为您的操作建立详细信息页面。创建详细信息页面后, 您可以序列化作业辅助角色的状态, 并将其作为延续令牌返回, 但须符合大小限制 (请参阅[AWS CodePipeline 中的限制 \(p. 208\)](#))。例如, 您可以将操作的状态写入用作延续令牌的字符串。您的作业辅助角色处理的第二步 (以及后续步骤) 完成操作的实际工作。最后一步向 AWS CodePipeline 返回成功或失败结果, 其中最后一步中没有延续令牌。

有关使用延续令牌的更多信息, 请参阅 [AWS CodePipeline API 参考](#) 中的 `PutJobSuccessResult` 规范。

6. 自定义操作完成后, 作业辅助角色通过调用两个 API 之一将自定义操作的结果返回给 AWS CodePipeline :
  - `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操作 (成功) 或停止 (失败)。

## 自定义作业辅助角色架构和示例

在制定概括性工作流程后, 即可创建您的作业辅助角色。虽然您的自定义操作的具体细节最终确定您的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:

- 使用 `PollForJobs` 从 AWS CodePipeline 轮询作业。
- 使用 `AcknowledgeJob`、`PutJobSuccessResult` 和 `PutJobFailureResult` 确认作业并将结果返回 AWS CodePipeline。
- 从管道的 Amazon S3 存储桶检索项目和/或将项目放入存储桶。要从 Amazon S3 存储桶下载项目, 您必须创建一个使用签名版本 4 (Sig V4) 签名的 Amazon S3 客户端。Sig V4 是 SSE-KMS 所必需的。

要将项目上传到 Amazon S3 存储桶, 您还必须另外配置 Amazon S3 `PutObject` 请求以使用加密。目前仅支持使用 SSE-KMS 进行加密。为了了解是使用默认密钥还是使用客户管理的密钥来上传项目, 您的自定义作业辅助角色必须查看[作业数据](#)并检查[加密密钥](#)属性。如果设置了加密密钥属性, 在配置 SSE-KMS 时应使用加密密钥 ID。如果密钥为空, 则应使用默认主密钥。AWS CodePipeline 使用默认 Amazon S3 主密钥, 除非另有配置。



以下示例显示如何在 Java 中创建 KMS 参数：

```
private static SSEAwsKeyManagementParams createSSEAwsKeyManagementParams(final
EncryptionKey encryptionKey) {
    if (encryptionKey != null
        && encryptionKey.getId() != null
        && EncryptionKeyType.KMS.toString().equals(encryptionKey.getType())) {
        // Use a customer-managed encryption key
        return new SSEAwsKeyManagementParams(encryptionKey.getId());
    }
    // Use the default master key
    return new SSEAwsKeyManagementParams();
}
```

有关更多示例，请参阅[使用 AWS 软件开发工具包在 Amazon S3 中指定 AWS Key Management Service](#)。有关 AWS CodePipeline 的 Amazon S3 存储桶的更多信息，请参阅[AWS CodePipeline 概念 \(p. 3\)](#)。

GitHub 上提供有更复杂的自定义作业辅助角色示例。该示例是一个开源示例，按原样提供。

- [AWS CodePipeline 的示例作业辅助角色](#)：从 GitHub 存储库下载示例。

## 向管道添加自定义操作

在拥有作业辅助角色后，您可以通过以下方法将自定义操作添加到管道中：创建一个新管道并在使用创建管道向导时选择该管道，编辑现有管道并添加自定义操作，或使用 AWS CLI、软件开发工具包或 API。

### Note

如果自定义操作是生成或部署操作，则可以在创建管道向导中创建一个包含该操作的管道。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

- [向管道添加自定义操作 \(控制台\) \(p. 118\)](#)
- [向现有管道添加自定义操作 \(CLI\) \(p. 118\)](#)

## 向管道添加自定义操作 (控制台)

要使用 AWS CodePipeline 控制台创建包含自定义操作的管道，请按照在[AWS CodePipeline 中创建管道 \(p. 82\)](#)中的步骤操作，并从您想要测试的任意数量的阶段中选择自定义操作。要使用 AWS CodePipeline 控制台将自定义操作添加到现有管道，请按照在[AWS CodePipeline 中编辑管道 \(p. 90\)](#)中的步骤操作，并将自定义操作添加到管道中的一个或多个阶段。

## 向现有管道添加自定义操作 (CLI)

您可以使用 AWS CLI 将自定义操作添加到现有管道。

1. 打开终端对话 (Linux, macOS, or Unix) 或命令提示符 (Windows)，并运行 get-pipeline 命令，将您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 AWS CodePipeline 控制台，并选择您刚刚编辑的管道的名称。

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 AWS CodePipeline 管道中调用 AWS Lambda 函数

[AWS Lambda](#) 是一项计算服务，可使您无需预配置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。

#### Note

创建并运行 Lambda 函数可能会导致向您的 AWS 账户收取费用。要了解更多信息，请参阅[定价](#)。

以下是 Lambda 函数可用于管道的一些方法：

- 通过应用或更新 AWS CloudFormation 模板来实施对环境所做的更改。
- 使用 AWS CloudFormation 在管道的一个阶段按需创建资源，并在另一个阶段将其删除。
- 使用将交换别名记录值的 Lambda 函数，在 AWS Elastic Beanstalk 中部署零停机时间的应用程序版本。
- 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。

本主题假定您熟悉 AWS CodePipeline 和 AWS Lambda 并且知道如何创建管道、函数和 IAM 策略以及它们所依赖的角色。在本主题中，我们将演练以下步骤：

- 创建一个 Lambda 函数以测试网页是否已成功部署。
- 配置 AWS CodePipeline 和 Lambda 执行角色以及运行函数所需的权限作为管道的一部分。
- 编辑管道以便将 Lambda 函数作为操作添加。
- 通过手动发布更改来测试操作。

本主题包含示例函数以演示在 AWS CodePipeline 中使用 Lambda 函数的灵活性：

- [Basic Lambda function \(p. 122\)](#)
  - 创建基本 Lambda 函数以便与 AWS CodePipeline 一起使用。
  - 在操作的 Details 链接中向 AWS CodePipeline 返回成功或失败结果。
- [使用 AWS CloudFormation 模板的示例 Python 函数 \(p. 129\)](#)

- 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`)。这将允许操作继续并让函数取得成功，即使已超过五分钟的运行时间 (Lambda 中的限制) 也是如此。

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 AWS Lambda 中的限制的信息，请参阅“AWS Lambda 开发人员指南”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 \(p. 121\)](#)
- [步骤 2：创建 Lambda 函数 \(p. 121\)](#)
- [步骤 3：在 AWS CodePipeline 控制台中向管道添加 Lambda 函数 \(p. 124\)](#)
- [步骤 4：使用 Lambda 函数测试管道 \(p. 126\)](#)
- [步骤 5：后续步骤 \(p. 126\)](#)
- [示例 JSON 事件 \(p. 128\)](#)
- [其他示例函数 \(p. 128\)](#)

## 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[AWS CodePipeline 教程 \(p. 22\)](#)中创建的管道相同。如果该管道仍然配置用于您的账户，并且属于将创建 Lambda 函数的相同区域，则可以跳过此步骤。

#### Important

您必须在将创建 Lambda 函数的同一区域创建管道及其所有资源。

#### 创建管道

1. 按照[教程：创建一个简单的管道 \(Amazon S3 存储桶\) \(p. 22\)](#)中的前三步骤来创建 Amazon S3 存储桶、AWS CodeDeploy 资源和两阶段管道。为您的实例类型选择 Amazon Linux 选项。您可以使用要对管道使用的任何名称，但此主题中的步骤会使用 `MyLambdaTestPipeline`。
2. 在您的管道的状态页中，在 AWS CodeDeploy 操作中，选择 Details。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 Description 选项卡中，复制 Public IP 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

AWS CodePipeline 的默认服务角色 `AWS-CodePipeline-Service` 包括调用该函数所需的 Lambda 权限，因此您不必创建其他调用策略或角色。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，包括 Lambda 操作的管道将会失败。

## 步骤 2：创建 Lambda 函数

在此步骤中，您将创建一个 Lambda 函数，该函数将发出 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 Lambda 执行角色。有关 Lambda、执行角色以及为什么需要这些角色的详细信息，请参阅“AWS Lambda 开发人员指南”中的[权限模型](#)。

## 创建执行角色

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
2. 选择 Policies，然后选择 Create Policy。
3. 在 Create Policy 页上，选择 Create Your Own Policy 按钮旁边的 Select。
4. 在 Review Policy 页中，在 Policy Name 中，键入策略名称 (例如，**CodePipelineLambdaExecPolicy**)。在 Description 中，键入 **Enables Lambda to execute code**。在 Policy Document 中，将以下策略复制并粘贴到策略框中，然后选择 Validate Policy。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:*"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Action": [
        "codepipeline:PutJobSuccessResult",
        "codepipeline:PutJobFailureResult"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

验证完策略后，请选择 Create Policy。

### Note

这些是 Lambda 函数与 AWS CodePipeline 和 Amazon CloudWatch 交互操作所需的最低权限。如果您想展开此策略以允许执行与其他 AWS 资源交互的函数，您应修改此策略允许执行这些 Lambda 函数请求的操作。

5. 在策略控制面板页中，选择 Roles，然后选择 Create New Role。
6. 在 Set Role Name 页中，为角色键入名称 (例如，**CodePipelineLambdaExecRole**)，然后选择 Next Step。
7. 在 Select Role Type 页中，在 AWS Service Roles 下的角色列表中，选择 AWS Lambda 旁边的 Select 按钮。
8. 在 Attach Policy 页中，选择 CodePipelineLambdaExecPolicy 旁边的复选框，然后选择 Next Step。
9. 在 Review 页面上，选择 Create Role。

## 创建示例 Lambda 函数以便与 AWS CodePipeline 一起使用

1. 通过以下网址登录 AWS 管理控制台并打开 AWS Lambda 控制台：<https://console.aws.amazon.com/lambda/>。
2. 在 Lambda: Function list 页中，选择 Create a Lambda function。

### Note

如果您看到的是 Welcome 页面，而不是 Lambda: Function list 页面，请选择 Get Started Now。

3. 在 Select blueprint 页面上，选择 Skip。
4. 在 Configure function 页中，在 Name 中，为您的 Lambda 函数键入名称 (例如，**MyLambdaFunctionForAWSCodePipeline**)。或者，在 Description 中，为函数键入描述 (例如，**A sample test to check whether the website responds with a 200 (OK) and contains a specific word on the page**)。在 Runtime 列表中，选择 Node.js，然后将以下代码复制到 Lambda function code 框中：

Note

CodePipeline.job 密钥下的事件对象包含 [job details](#)。有关 AWS CodePipeline 返回到 Lambda 的 JSON 事件的完整示例，请参阅[示例 JSON 事件 \(p. 128\)](#)。

```
var assert = require('assert');
var AWS = require('aws-sdk');
var http = require('http');

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 AWS
    // CodePipeline, in this case a URL which will be
    // health checked by this function.
    var url =
event["CodePipeline.job"].data.actionConfiguration.configuration.UserParameters;

    // Notify AWS CodePipeline of a successful job
    var putJobSuccess = function(message) {
        var params = {
            jobId: jobId
        };
        codepipeline.putJobSuccessResult(params, function(err, data) {
            if(err) {
                context.fail(err);
            } else {
                context.succeed(message);
            }
        });
    };

    // Notify AWS 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);
        });
    };

    // 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
var 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);
    }
});
};
```

5. 将 Handler name 的值保留为默认值，但要将其 Role 更改为 **CodePipelineLambdaExecRole**。
6. 在 Advanced settings 中，对于 Timeout (s)，键入 **20**。
7. 配置完这些详细信息后，选择 Next。
8. 在 Review 页面上，选择 Create function。

## 步骤 3：在 AWS CodePipeline 控制台中向管道添加 Lambda 函数

在此步骤中，您将向管道中添加一个新阶段，然后向该阶段添加一个操作 — 一个调用您的函数的 Lambda 操作。

## 添加阶段

1. 登录 AWS 管理控制台并通过以下网址打开 AWS CodePipeline 控制台：<http://console.aws.amazon.com/codepipeline>。
2. 在 Welcome 页中，从管道列表中选择您创建的管道。
3. 在管道视图页中，选择 Edit。
4. 在 Edit 页中，选择用于在部署阶段之后使用 AWS CodeDeploy 操作添加阶段的选项。为阶段键入名称 (例如，**LambdaStage**)，然后选择用于为阶段添加操作的选项。

### 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，以便您可以轻松地查看该操作的进度。

5. 在 Add action 面板的 Action category 中，选择 Invoke。在 Invoke actions 的 Action name 中，为您的 Lambda 操作键入名称 (例如，**MyLambdaAction**)。在 Provider 中，选择 AWS Lambda。在 Function name 中，选择或键入您的 Lambda 函数的名称 (例如，**MyLambdaFunctionForAWSCodePipeline**)。在 User parameters 中，指定您前面复制的 Amazon EC2 实例的 IP 地址 (例如，**http://192.0.2.4**)，然后选择 Add action。

**Add action**

Choose a serial action from the action category list.

**Action category\*** Invoke

Configure how your function runs in the pipeline.

**Invoke actions**

Choose how to invoke an action.

**Action name\*** MyLambdaAction

**Provider\*** AWS Lambda

**AWS Lambda**

Choose one of your existing functions, or [create a new one in AWS Lambda](#).

**Function name\*** MyLambdaFunctionforAWSCodePipeline

**User parameters** http://192.0.2.4

This string will be used in the event data parameter passed to the handler in AWS Lambda. [Learn more](#)

\* Required

Cancel Add action



#### 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 AWS Lambda 中事件数据和处理程序的更多信息，请参阅 AWS Lambda 开发人员指南中的[编程模型](#)。

6. 在 Edit 页中，选择 Save pipeline changes。

## 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 Details 链接，以查看 Amazon CloudWatch 中函数的日志流，包括事件的计费持续时间。如果函数失败，CloudWatch 日志将提供有关原因的信息。

## 步骤 5：后续步骤

成功创建 Lambda 函数并将其作为操作添加到管道中后，您可以尝试以下操作：

- 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。

## AnyCompanyPipeline [View pipeline history](#)

View progress and manage your pipeline.

Edit

Release change

### Source

Source

Amazon S3

✓ Succeeded 1 min ago

Source: Amazon S3 versi...

### Staging

AnyCompany-JuneUpd.

AWS CodeDeploy

✓ Succeeded just now

[Details](#)

Source: Amazon S3 versi...

### LambdaStage

MyLambdaAction

AWS Lambda

⌚ In Progress just now

MyLambdaAction2

AWS Lambda

⌚ In Progress just now

MyLambdaAction3

AWS Lambda

⌚ In Progress just now

Source: Amazon S3 version id: aKT2AkgNpaPHpVVx.nPuJbSeDK1XfKQt

试验完 Lambda 函数后，请考虑将其从管道中移除，并从 AWS Lambda 中删除，然后从 IAM 中删除角色，以避免可能产生的费用。有关更多信息，请参阅在 [AWS CodePipeline 中编辑管道](#) (p. 90)、在 [AWS CodePipeline 中删除管道](#) (p. 101) 和 [删除角色或实例配置文件](#)。

## 示例 JSON 事件

以下示例显示了 AWS CodePipeline 发送给 Lambda 的示例 JSON 事件。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": "wJa1rXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
        "sessionToken": "MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
0BAQUFADCBiDELMAKGA1UEBhMCVVMxCzAJBgNVBAGTAldBMRAdG9YDVQQHEwdTZ
WF0dGx1MQ8wDQYDVQQKEWZBbWF6b24xZDASBgNVBAsTC01BTSBDb25zb2x1MRIw
EAYDVQQDEw1UZXXNQ21sYWMxH2AdBgkqhkiG9w0BCQEWEG5vb25lQGFTYXpvi5
jb20wHhcNMTEwNDI1MjAONTIwHhcNMTEwNDI1MjAONTIwYjCBiDELMAKGA1UEBh
MCVVMxCzAJBgNVBAGTAldBMRAdG9YDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEWZBb
WF6b24xZDASBgNVBAsTC01BTSBDb25zb2x1MRIwEAYDVQQDEw1UZXXNQ21sYWMx
H2AdBgkqhkiG9w0BCQEWEG5vb25lQGFTYXpvi5jb20wZ8wDQYJKoZIhvcNAQEE
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITxOUSQv7c7ugFFDZQGBzZswY6786m86gpEibb3OhjZnzcqvQAARHhdLQWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntned9+h8Mg9q6q+auN
KyExzyLwaxLAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJILJ00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp378OD8uTs7fLvJx79LjSTbNYiYtVbZPQUQ5Yaxu2jXnImvw
3rrszlaEXAMPLE=",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE"
      },
      "continuationToken": "A continuation token if continuing job"
    }
  }
}
```

## 其他示例函数

以下示例 Lambda 函数演示了您可以在 AWS CodePipeline 中用于您的管道的其他功能。要使用这些函数，您可能必须对 Lambda 执行角色的策略进行修改，如每个示例的介绍中所述。

## 主题

- [使用 AWS CloudFormation 模板的示例 Python 函数 \(p. 129\)](#)

## 使用 AWS CloudFormation 模板的示例 Python 函数

以下示例显示了基于提供的 AWS CloudFormation 模板创建或更新堆栈的函数。该模板将创建一个 Amazon S3 存储桶。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 存储桶作为源操作的管道，或者您可以访问可与管道一起使用的受版本控制的 Amazon S3 存储桶。您将创建 AWS CloudFormation 模板，压缩它，并将其作为 .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。

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)。这将允许操作继续并让函数取得成功，即使已超过五分钟的运行时间 (Lambda 中的限制) 也是如此。

要使用此示例 Lambda 函数，Lambda 执行角色的策略在 AWS CloudFormation、Amazon S3 和 AWS CodePipeline 中必须具有 Allow 权限，如此示例策略中所示：

```
{
  "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，然后复制并粘贴以下代码：

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",
  "Description" : "AWS 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 存储桶。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。将此操作的输出命名为 **MyTemplate**。有关更多信息，请参阅 [在 AWS CodePipeline 中编辑管道 \(p. 90\)](#)。

#### Note

示例 Lambda 函数需要这些文件名和压缩结构。但是，您可以使用您自己的 AWS CloudFormation 模板替换此示例。如果您选择使用自己的模板，请确保修改 Lambda 执行角色的策略，以允许您的 AWS CloudFormation 模板所需的任何额外功能。

添加以下代码作为 Lambda 中的函数

1. 打开 Lambda 控制台，选择 Create a Lambda function。
2. 在 Select blueprint 页面上，选择 Skip。
3. 在 Configure function 页中，在 Name 中，为您的 Lambda 函数键入名称。或者，在 Description 中，为函数键入描述。
4. 在 Runtime 列表中，选择 Python 2.7。
5. 将 Handler name 的值保留为默认值，但将 Role 更改为您的 Lambda 执行角色 (例如，**CodePipelineLambdaExecRole**)。
6. 在 Advanced settings 中，对于 Timeout (s)，将默认值 3 秒替换为 **20** 秒。
7. 将以下代码复制到 Lambda function code 代码框：

```
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']:
        # 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 off 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."
```

8. 保存函数。
9. 从 AWS CodePipeline 控制台，编辑管道以在您的管道中添加该函数作为阶段中的操作。在 UserParameters 中，您必须为包含大括号的 JSON 字符串提供由逗号隔开的三个参数：堆栈名称、AWS CloudFormation 模板名称和文件的路径以及应用程序名称。

例如，要创建名为 *MyTestStack* 的堆栈，对于输入项目 *MyTemplate* 的管道，在 UserParameters 中，您需要键入：{"stack": "*MyTestStack*", "file": "template-package/template.json", "artifact": "*MyTemplate*"}

#### Note

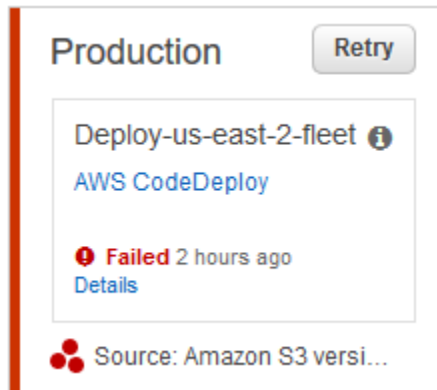
即使您在 UserParameters 中指定了输入项目，您也必须在 Input artifacts 中为操作指定此输入项目。

10. 将您的更改保存至管道，然后手动释放更改，以便测试操作和 Lambda 函数。

## 在 AWS CodePipeline 中重试失败的操作

在 AWS CodePipeline 中，操作是在某阶段对项目执行的任务。如果某个操作或一组并行操作未成功完成，则管道将停止运行。

您可以重试某个阶段最新的失败操作，而不必从头开始再次运行管道。如果您正在使用控制台查看管道，在可以重试失败操作的阶段，将会出现 **Retry** 按钮。



如果使用 AWS CLI，则可以通过 `get-pipeline-state` 命令确定是否有任何操作失败。

#### Note

在以下情况下，您可能无法重试操作：

- 操作失败后总体管道结构发生变化。
- 阶段中的一个或多个操作正在进行中。
- 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 \(控制台\) \(p. 137\)](#)
- [重试失败的操作 \(CLI\) \(p. 137\)](#)

## 重试失败的操作 (控制台)

1. 登录 AWS 管理控制台并通过以下网址打开 AWS CodePipeline 控制台：<http://console.aws.amazon.com/codepipeline>。

所有与您的 AWS 账户关联的管道的名称将会显示。

2. 在 Name 中，选择管道的名称。
3. 查找包含失败操作的阶段，然后选择 **Retry**。

#### Note

要确定该阶段中的哪些操作可以重试，请将鼠标指针悬停在 **Retry** 按钮上。

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)

要使用 AWS CLI 重试失败的操作，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。然后您可以使用 `--cli-input-json` 参数运行 `retry-stage-execution` 命令。要检索您需要用于 JSON 文件的详细信息，最简单的是使用 `get-pipeline-state` 命令。

1. 在终端 (Linux, macOS, or Unix) 或命令提示符 (Windows) 中，对一个管道运行 `get-pipeline-state` 命令。例如，对于名为 `MyFirstPipeline` 的管道，您应键入类似以下内容：

```
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"
      }
    }
  ]
}
```

2. 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。(目前唯一受支持的值是 `FAILED_ACTIONS`)

对于之前的 `MyFirstPipeline` 示例，您的文件应类似如下所示：

```
{
  "pipelineName": "MyFirstPipeline",
  "stageName": "Staging",
  "pipelineExecutionId": "3137f7cb-7cf7-EXAMPLE",
  "retryMode": "FAILED_ACTIONS"
}
```

3. 保存名称类似于 `retry-failed-actions.json` 的文件。
4. 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
```

5. 要查看重试尝试的结果，请打开 AWS CodePipeline 控制台并选择包含失败操作的管道，或者再次使用 `get-pipeline-state` 命令。有关更多信息，请参阅 [在 AWS CodePipeline 中查看管道详细信息和历史记录 \(p. 96\)](#)。

## 管理 AWS CodePipeline 中的审批操作

在 AWS CodePipeline 中，您可以在管道内某个阶段中您希望管道执行停止的位置添加审批操作，以便拥有所需 AWS Identity and Access Management 权限的人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。如果该操作被拒绝，或者没有人在管道到达该操作并停止的七天内批准或拒绝该操作，结果将与操作失败的结果相同，并且管道执行不会继续。

您可以出于各种目的使用手动审批：

- 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。

## AWS CodePipeline 中手动审批操作的配置选项

AWS CodePipeline 提供了三个配置选项，您可使用这些选项将审批操作事宜告知审批者。

发布审批通知 您可以配置审批操作，以便管道在审批操作处停止时，向 Amazon Simple Notification Service 主题发布消息。Amazon SNS 将消息发送给订阅该主题的每个终端节点。所使用的主题必须是在某个 AWS 区域中创建的，而该区域必须与将包含此审批操作的管道相同。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，您可以从电子邮件或 SMS 收件人、SQS 队列、HTTP/HTTPS 终端节点或您使用 Amazon SNS 调用的 AWS Lambda 函数等格式中选择。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](#)

要查看为审批操作通知生成的 JSON 数据的结构，请参阅[AWS CodePipeline 中的手动审批通知的 JSON 数据格式 \(p. 148\)](#)。

指定供审核的 URL 作为审批操作配置的一部分，您可以指定要审核的 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 AWS 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。

## AWS CodePipeline 中审批操作的设置和 workflows 概述

下面概述了设置和使用手动审批的过程。



1. 您向组织中的一个或多个 IAM 用户授予批准或拒绝审批操作所需的 IAM 权限。
2. (可选) 如果使用 Amazon SNS 通知，则确保在 AWS CodePipeline 操作中使用的服务角色具有访问 Amazon SNS 资源的权限。
3. (可选) 如果使用 Amazon SNS 通知，则创建一个 Amazon SNS 主题并向其添加一个或多个订阅者或终端节点。
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，可以向管道的某个阶段添加审批操作。

如果您正在使用通知，请在操作配置中包括 Amazon SNS 主题的 Amazon 资源名称 (ARN)。(ARN 是 Amazon 资源的唯一标识符。Amazon SNS 主题的 ARN 结构类似于 `arn:aws:sns:us-east-2:80398EXAMPLE:MyApprovalTopic`。有关更多信息，请参阅 Amazon Web Services 一般参考中的 [Amazon 资源名称 \(ARN\)](#) 和 [AWS 服务命名空间](#)。)

5. 管道在到达审批操作时停止。如果操作配置中包含 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者或订阅的终端节点发送一条消息，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”。

## 在 AWS CodePipeline 中向 IAM 用户授予审批权限

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。有关可用于 AWS CodePipeline 的托管策略的信息，请参阅 [适用于 AWS CodePipeline 的 AWS 托管 \(预定义\) 策略 \(p. 180\)](#)。

## 授予对所有管道和审批操作的审批权限

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
2. 在导航窗格中，选择 Groups、Users 或 Roles。
3. 选择要授予权限的组、角色或 IAM 用户。
4. 选择 Permissions 选项卡。
5. 选择 Add permissions，然后选择 Attach existing policies directly。
6. 选择 `AWSCodePipelineApproverAccess` 托管策略旁边的复选框，然后选择 Next: Review。
7. 选择 Add permissions。

## 指定对特定管道和审批操作的审批权限

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。

### Important

确保您已使用 [AWS CodePipeline 入门 \(p. 8\)](#) 中所用的相同账户信息登录 AWS 管理控制台。

2. 在导航窗格中，根据需要选择 Groups 或 Users。
3. 浏览并选择您要更改的用户或组。
4. 在摘要页面上，选择 Permissions 选项卡，然后展开 Inline Policies 部分。
5. 执行以下任一操作：
  - 如果您选择了 Groups，则现在选择 Create Group Policy。如果您选择了 Users，则现在选择 Create User Policy。
  - 如果尚未创建任何内联策略，请选择 [click here](#)。
6. 选择 Custom Policy，然后选择 Select。
7. 在 Policy Name 中键入此策略的名称。
8. 指定 IAM 用户可访问的单个资源。例如，以下策略授权用户仅批准或拒绝 美国东部 ( 俄亥俄 ) 区域 (us-east-2) 中的 MyFirstPipeline 管道中名为 MyApprovalAction 的操作：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "codepipeline:ListPipelines"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "codepipeline:GetPipeline",
        "codepipeline:GetPipelineState",
        "codepipeline:GetPipelineExecution"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline"
    },
    {
      "Action": [
        "codepipeline:PutApprovalResult"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline/MyApprovalStage/MyApprovalAction"
    }
  ]
}
```

#### Note

仅当 IAM 用户需要访问 AWS CodePipeline 控制面板来查看该管道列表时才需要 codepipeline:ListPipelines 权限。如果不需要访问控制台，则可以忽略 codepipeline:ListPipelines。

9. 选择 Validate Policy。更正显示在页面顶部的红框中的任何错误。
10. 若您满意所创建的策略，请选择 Apply Policy。

## 向 AWS CodePipeline 服务角色授予 Amazon SNS 权限

如果您计划在审批操作需要审核时使用 Amazon SNS 向主题发布通知，则必须向您的 AWS CodePipeline 操作中使用到的服务角色授予访问 Amazon 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。

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。

### Important

确保您已使用[AWS CodePipeline 入门 \(p. 8\)](#)中所用的相同账户信息登录 AWS 管理控制台。

2. 在 IAM 控制台的导航窗格中，选择 Roles。
3. 选择您在 AWS CodePipeline 操作中使用到的服务角色的名称。
4. 在 Permissions 选项卡上的 Inline Policies 区域中，选择 Create Role Policy。

–或–

如果 Create Role Policy 按钮不可用，展开 Inline Policies 区域，然后选择 [click here](#)。

5. 在 Set Permissions 页面上，选择 Custom Policy，然后选择 Select。
6. 在 Review Policy 页面上的 Policy Name 字段中，键入一个名称以标识此策略，例如 SNSPublish。
7. 将以下内容粘贴到 Policy Document 字段中：

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

8. 选择 Apply Policy。

## 向 AWS CodePipeline 中的管道添加手动审批操作

您可以向 AWS CodePipeline 管道内的某个阶段中您希望管道停止的位置添加审批操作，以便有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。


如果要在审批操作已可供审核时使用 Amazon SNS 发送通知，则必须先完成以下先决条件：


- 授予您的 AWS CodePipeline 服务角色访问 Amazon SNS 资源的权限。有关信息，请参阅 [向 AWS CodePipeline 服务角色授予 Amazon SNS 权限 \(p. 142\)](#)。
- 授予组织中的一个或多个 IAM 用户更新审批操作状态的权限。有关信息，请参阅 [在 AWS CodePipeline 中向 IAM 用户授予审批权限 \(p. 140\)](#)。

## 向 AWS CodePipeline 管道添加手动审批操作 (控制台)

您可以使用 AWS CodePipeline 控制台向现有 AWS CodePipeline 管道添加审批操作。如果要在创建新管道时添加审批操作，则必须使用 AWS CLI。

1. 通过以下网址打开 AWS CodePipeline 控制台：<https://console.aws.amazon.com/codepipeline/>。
2. 在 Name 中，选择管道。
3. 在管道详细信息页面上，选择 Edit。
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 + Stage，然后键入阶段名称。

如果您要在现有阶段中添加审批操作，请选择该阶段的编辑图标 (  )。

5. 选择操作图标 (  )。
6. 在 Add action 页面中，执行以下操作：
  1. 在 Action category 中，选择 Approval。
  2. 在 Action name 中，键入用于识别操作的名称。
  3. 在 Approval type 中，选择 Manual approval。
  4. (可选) 在 SNS topic ARN 中，选择用来发送审批操作通知的主题名称。
  5. (可选) 在 URL for review 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。
  6. (可选) 在 Comments 中，键入您要与审批者共享的任何其他信息。

完成后的页面可能与下文类似：

Add action

Choose a serial action from the action category list.

Action category\*

Approval

Action name\*

MyApprovalAction

Approval type\*

Manual approval

Manual approval configuration ⓘ

Configure the approval request.

SNS topic ARN

arn:aws:sns:us-east-1:80398EXAMPLE:MyAp

URL for review

http://example.com

The URL you enter here will be provided to the reviewer as part of the approval request. It should begin with 'http://' or 'https://'.

Comments

he latest changes include feedback from Bob.

The information you provide will be displayed to the approver in email notifications or the console.

\* Required

Cancel

Add action

7. 选择 Add Action。

## 向 AWS CodePipeline 管道添加手动审批操作 (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。您可以通过在创建或编辑的阶段中包含审批类型为手动审批的审批操作来实现此目的。

有关创建和编辑管道的更多信息，请参阅在 [AWS CodePipeline 中创建管道 \(p. 82\)](#)和在 [AWS CodePipeline 中编辑管道 \(p. 90\)](#)。

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 [AWS CodePipeline 管道结构参考 \(p. 202\)](#)。

```
{
```

API 版本 2015-07-09  
144

```

"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 [AWS CodePipeline 管道结构参考 \(p. 202\)](#)。

```

{
  "name": "Production",
  "actions": [
    {
      "inputArtifacts": [],
      "name": "MyApprovalStage",
      "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": "MyDeploymentStage",
      "actionTypeId": {
        "category": "Deploy",
        "owner": "AWS",
        "version": "1",
        "provider": "CodeDeploy"
      },
      "outputArtifacts": [],
      "configuration": {
        "ApplicationName": "MyDemoApplication",

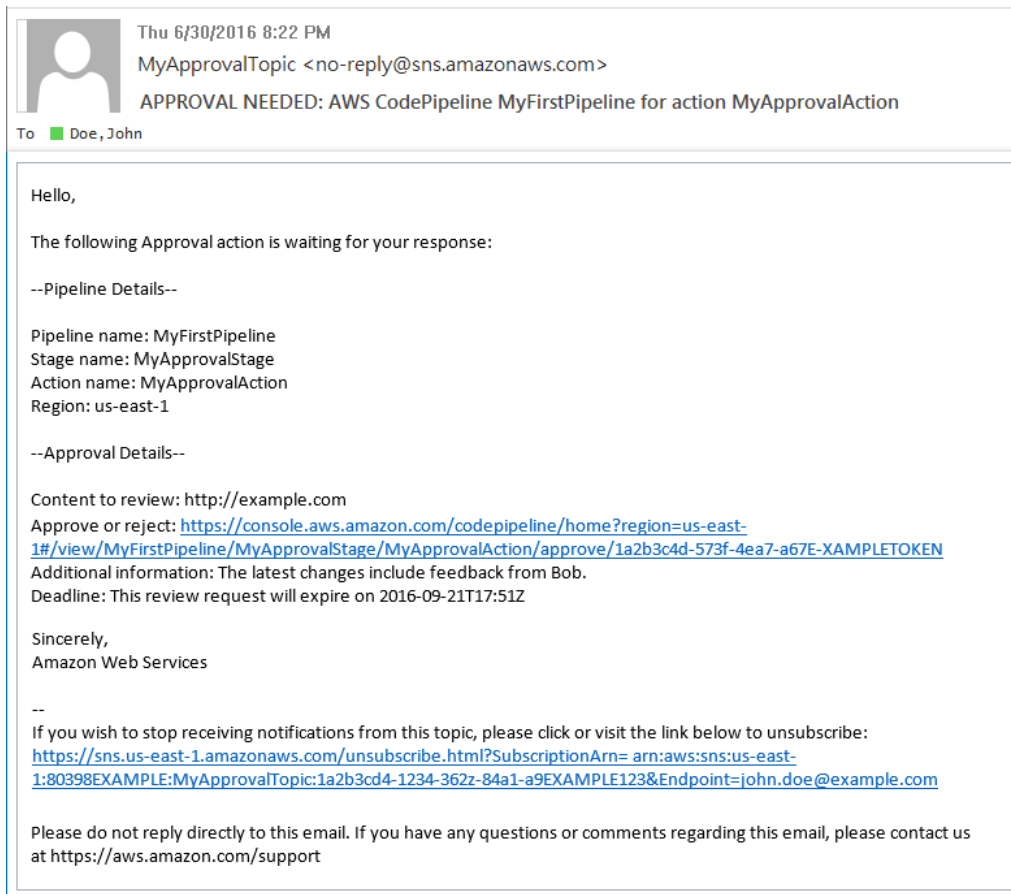
```

```
        "DeploymentGroupName": "MyProductionFleet"
      },
      "runOrder": 2
    }
  ]
}
```

## 批准或拒绝 AWS CodePipeline 中的审批操作

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”。

如果将审批操作添加到管道中的人员配置了通知，则您可能会收到类似于以下内容的电子邮件：



## 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 Approve or reject 链接，如有必要，请登录控制台，然后继续执行下面的步骤 7。否则，请完成以下所有步骤。

1. 通过以下网址打开 AWS CodePipeline 控制台：<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。
4. 将鼠标指针悬停在信息图标上方，以查看注释和 URL (如果有)。该信息弹出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 Manual approval 链接以打开目标网页，然后查看内容。
6. 返回管道详细信息视图，然后选择 Review 按钮。
7. 在 Approve or reject the revision 窗口中，键入审查注释，例如您为何批准或拒绝操作，然后选择 Approve 或 Reject 按钮。

## 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, or Unix) 或命令提示符 (Windows) 中，对包含审批操作的管道运行 `get-pipeline-state` 命令。例如，对于名为 `MyFirstPipeline` 的管道，可以键入以下内容：

```
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。例如：

```
{
  "created": 1467929497.204,
  "pipelineName": "MyFirstPipeline",
  "pipelineVersion": 1,
  "stageStates": [
    {
      "actionStates": [
        {
          "actionName": "MyApprovalAction",
          "currentRevision": {
            "created": 1467929497.204,
            "revisionChangeId": "CEM7d6Tp7zfelUSLCPwo234xEXAMPLE",
            "revisionId": "HYGp7zmwbCPPwo23xCmdTeqIleXAMPLE"
          },
          "latestExecution": {
            "lastUpdatedBy": "arn:aws:iam::123456789012:user/Bob",
            "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，您需要记录以下内容：
- 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，即“Approved”或“Rejected”。(此响应必须大写。)
  - 您的摘要注释。

对于之前的 `MyFirstPipeline` 示例，您的文件应类似如下所示：

```
{
  "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，并指定审批 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
```

## AWS CodePipeline 中的手动审批通知的 JSON 数据格式

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。有关使用 Amazon SNS 将消息发送到 Amazon SQS 队列的信息，请参阅[将 Amazon SNS 消息发送到 Amazon SQS 队列](#)。有关使用 Amazon SNS 调用 Lambda 函数的信息，请参阅[使用 Amazon SNS 通知调用 Lambda 函数](#)。

以下示例显示适用于 AWS CodePipeline 审批的 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."
  }
}
```

# 使用 AWS CodePipeline 中的阶段过渡

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。

## 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 [管理 AWS CodePipeline 中的审批操作 \(p. 139\)](#)。

## 主题

- [禁用或启用过渡 \(控制台\) \(p. 149\)](#)
- [禁用或启用过渡 \(CLI\) \(p. 151\)](#)

## 禁用或启用过渡 (控制台)

### 要在管道中禁用或启用过渡

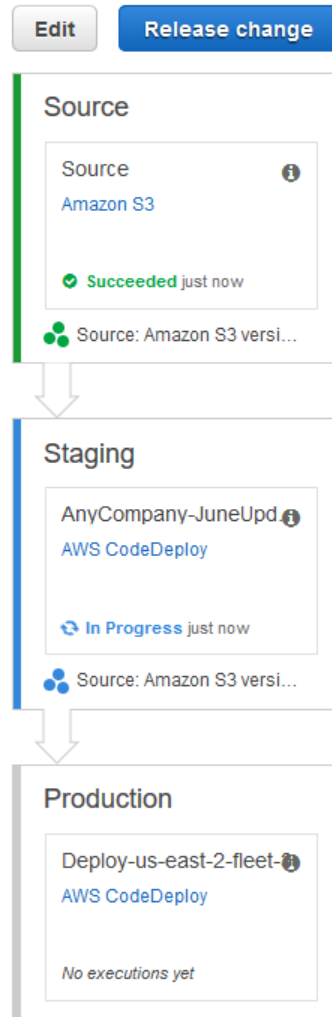
1. 登录 AWS 管理控制台并通过以下网址打开 AWS CodePipeline 控制台：<http://console.aws.amazon.com/codepipeline>。

所有与您的 AWS 账户关联的管道的名称将会显示。

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，然后选中它。例如，在以下示例管道中，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箭头：

## AnyCompanyPipeline [View pipeline history](#)

View progress and manage your pipeline.

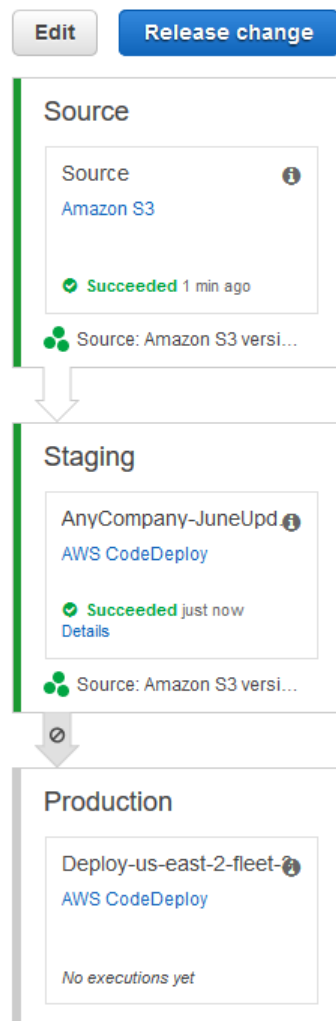


4. 在 Disable transition 对话框中，键入禁用过渡的原因，然后选择 Disable。

箭头变化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。

## AnyCompanyPipeline [View pipeline history](#)

View progress and manage your pipeline.



5. 要启用过渡，请选择禁用的过渡箭头。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，管道将开始在此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

在您启用过渡后，可能需要几秒钟的时间更改才会显示在 AWS CodePipeline 控制台中。

## 禁用或启用过渡 (CLI)

要使用 AWS CLI 禁用阶段之间的过渡，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. 请打开终端 (Linux, macOS, or Unix) 或命令提示符 (Windows) 并使用 AWS CLI 运行 `disable-stage-transition` 命令，指定管道的名称、您要禁用过渡的阶段的名称、过渡类型以及您要禁用向该阶段过渡的原因。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，要禁用向名为 `MyFirstPipeline` 的管道中名为 `Staging` 的阶段过渡，您将键入类似以下的命令：

```
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. 要确认已禁用过渡，请在 AWS CodePipeline 控制台中查看管道，或者运行 `get-pipeline-state` 命令。有关更多信息，请参阅 [查看管道详细信息和历史记录 \(控制台\)](#) (p. 96) 和 [查看管道详细信息和历史记录 \(CLI\)](#) (p. 98)。

## 要启用过渡

1. 请打开终端 (Linux, macOS, or Unix) 或命令提示符 (Windows) 并使用 AWS CLI 运行 `enable-stage-transition` 命令，指定管道的名称、您要启用过渡的阶段的名称以及过渡类型。

例如，要启用向名为 `MyFirstPipeline` 的管道中名为 `Staging` 的阶段过渡，您将键入类似以下的命令：

```
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. 要确认已禁用过渡，请在 AWS CodePipeline 控制台中查看管道，或者运行 `get-pipeline-state` 命令。有关更多信息，请参阅 [查看管道详细信息和历史记录 \(控制台\)](#) (p. 96) 和 [查看管道详细信息和历史记录 \(CLI\)](#) (p. 98)。

# 使用 AWS CodePipeline 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。您应从 AWS 解决方案的各个部分收集监控数据，以便更轻松地了解出现的多点故障。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：

- 您的监控目标是什么？
- 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监控 AWS CodePipeline 管道及其资源：

- Amazon CloudWatch Events — 使用 Amazon CloudWatch Events 检测和响应管道执行状态更改 (例如，发送 Amazon SNS 通知或调用 Lambda 函数)。
- AWS CloudTrail — 使用 CloudTrail 捕获由 AWS CodePipeline 自身或代表其在您的 AWS 账户中进行的 API 调用，并将日志文件传送到一个 Amazon S3 存储桶。您可以选择让 CloudWatch 在传送新日志文件时发布 Amazon SNS 通知，以便快速采取措施。
- 控制台和 CLI — 您可以使用 AWS CodePipeline 控制台和 CLI 查看有关管道或特定管道执行的状态的详细信息。

## 主题

- [使用 Amazon CloudWatch Events 检测和响应管道状态更改 \(p. 153\)](#)
- [使用 AWS CloudTrail 记录 AWS CodePipeline API 调用 \(p. 160\)](#)
- [查看 AWS CodePipeline 管道中的当前源修订详细信息 \(p. 162\)](#)

## 使用 Amazon CloudWatch Events 检测和响应管道状态更改

Amazon CloudWatch Events 是一项 Web 服务，可以监控您的 AWS 资源以及在 AWS 上运行的应用程序。您可以使用 Amazon CloudWatch Events 检测和响应管道、阶段或操作状态更改。然后，在管道、阶段或操作进入在创建的规则中指定的状态时，CloudWatch Events 根据该规则调用一个或多个目标操作。根据状态更改的类型，您可能想发送通知，捕获状态信息，采取纠正措施，启动事件或采取其他操作。

Amazon CloudWatch Events 由以下内容组成：

- 规则.在配置 Amazon CloudWatch Events 中的事件时，将先创建一个规则并将选定的服务作为事件源。
- 目标.新规则将选定的服务作为事件目标。有关可作为 Amazon CloudWatch Events 目标的服务列表，请参阅[什么是 Amazon CloudWatch Events](#)。

Amazon CloudWatch Events 规则和目标示例：



- 在实例状态发生更改时发送通知的规则，其中 EC2 实例是事件源，Amazon SNS 是事件目标。
- 在生成阶段发生更改时发送通知的规则，其中 AWS CodeBuild 配置是事件源，Amazon SNS 是事件目标。
- 检测管道更改并调用 AWS Lambda 函数的规则。

将 AWS CodePipeline 配置为事件源：

1. 创建一个 Amazon CloudWatch Events 规则并将 AWS CodePipeline 作为事件源。
2. 为您的规则创建一个目标，并将某个可用服务 (如 AWS Lambda 或 Amazon SNS) 作为 Amazon CloudWatch Events 中的目标。
3. 为 Amazon CloudWatch Events 授予权限，以允许其调用选定的目标服务。

## 了解管道执行状态更改规则的工作方式

您可以使用 Amazon CloudWatch 中的事件窗口生成规则以检测和响应管道状态更改。在生成规则时，控制台中的事件模式预览框 (或 CLI 中的 `--event-pattern` 输出) 以 JSON 格式显示事件字段。

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。

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 CloudWatch 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。

管道执行状态更改消息内容：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。

```
{
  "version": "0",
  "id": event_Id,
  "detail-type": "CodePipeline Pipeline Execution State Change",
  "source": "aws.codepipeline",
  "account": Pipeline_Account,
  "time": TimeStamp,
  "region": "us-east-1",
  "resources": [
    "arn:aws:codepipeline:us-east-1:account_ID:myPipeline"
  ],
  "detail": {
    "pipeline": "myPipeline",
    "version": "1",
    "state": "STARTED",
    "execution-id": execution_Id
  }
}
```

```
}
```

阶段执行状态更改消息内容：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。

```
{
  "version": "0",
  "id": event_Id,
  "detail-type": "CodePipeline Stage Execution State Change",
  "source": "aws.codepipeline",
  "account": Pipeline_Account,
  "time": TimeStamp,
  "region": "us-east-1",
  "resources": [
    "arn:aws:codepipeline:us-east-1:account_ID:myPipeline"
  ],
  "detail": {
    "pipeline": "myPipeline",
    "version": "1",
    "execution-id": execution_Id,
    "stage": "Prod",
    "state": "STARTED"
  }
}
```

操作执行状态更改消息内容：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 "myAction" 操作。

```
{
  "version": "0",
  "id": event_Id,
  "detail-type": "CodePipeline Action Execution State Change",
  "source": "aws.codepipeline",
  "account": Pipeline_Account,
  "time": TimeStamp,
  "region": "us-east-1",
  "resources": [
    "arn:aws:codepipeline:us-east-1:account_ID:myPipeline"
  ],
  "detail": {
    "pipeline": "myPipeline",
    "version": "1",
    "execution-id": execution_Id,
    "stage": "Prod",
    "action": "myAction",
    "state": "STARTED",
    "type": {
      "owner": "AWS",
      "category": "Deploy",
      "provider": "CodeDeploy",
      "version": 1
    }
  }
}
```

有效的状态值：

管道级别状态

管道状态	描述
STARTED	当前正在运行管道执行。

管道状态	描述
SUCCEEDED	已成功完成管道执行。
RESUMED	已重试失败的管道执行以响应 <code>RetryStageExecution</code> API 调用。
FAILED	未成功完成管道执行。
CANCELED	已取消管道执行，因为已更新管道结构。
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。

#### 阶段级别状态

阶段状态	描述
STARTED	当前正在运行阶段。
SUCCEEDED	已成功完成阶段。
RESUMED	已重试失败的阶段以响应 <code>RetryStageExecution</code> API 调用。
FAILED	未成功完成阶段。
CANCELED	已取消阶段，因为已更新管道结构。

#### 操作级别权限

操作状态	描述
STARTED	当前正在运行操作。
SUCCEEDED	已成功完成操作。
FAILED	对于审批操作，FAILED 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。
CANCELED	已取消操作，因为已更新管道结构。

## 先决条件

在创建事件规则以用于 AWS CodePipeline 操作之前，您应该执行以下操作：

- 完成 CloudWatch Events 先决条件。有关此信息，请参阅[区域终端节点](#)。
- 熟悉 CloudWatch Events 中的事件、规则和目标。有关更多信息，请参阅[什么是 Amazon CloudWatch Events](#)。
- 创建一个或多个将在事件规则中使用的目标，如 Amazon SNS 通知主题。

## 每次管道状态发生更改时发送通知 (控制台)

这些步骤说明了如何使用 CloudWatch 控制台创建规则以发送 AWS CodePipeline 中的更改通知。

创建 CloudWatch Events 规则并将 AWS CodePipeline 作为事件源

1. 通过以下网址打开 CloudWatch 控制台：<https://console.aws.amazon.com/cloudwatch/>。

2. 在导航窗格中，选择 Events。
3. 选择 Create rule。在事件源下面，从服务名称下拉列表中选择 CodePipeline。
4. 从事件类型下拉列表中，选择通知的状态更改级别。
  - 对于适用于管道级别事件的规则，请选择 CodePipeline 管道执行状态更改。
  - 对于适用于阶段级别事件的规则，请选择 CodePipeline 阶段执行状态更改。
  - 对于适用于操作级别事件的规则，请选择 CodePipeline 操作执行状态更改。
5. 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，然后从列表选择一个或多个状态值。

**Event Source**

Build or customize an Event Pattern or set a Schedule to invoke Targets.

☒ Event Pattern ⓘ ☐ Schedule ⓘ

**Build event pattern to match events by service**

Service Name: CodePipeline

Event Type: CodePipeline Pipeline Execution State Change

☐ Any state ☒ Specific state(s)

FAILED

**Event Pattern Preview** [Copy to clipboard](#) [Edit](#)

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

**Targets**

Select Target to invoke when this rule is triggered.

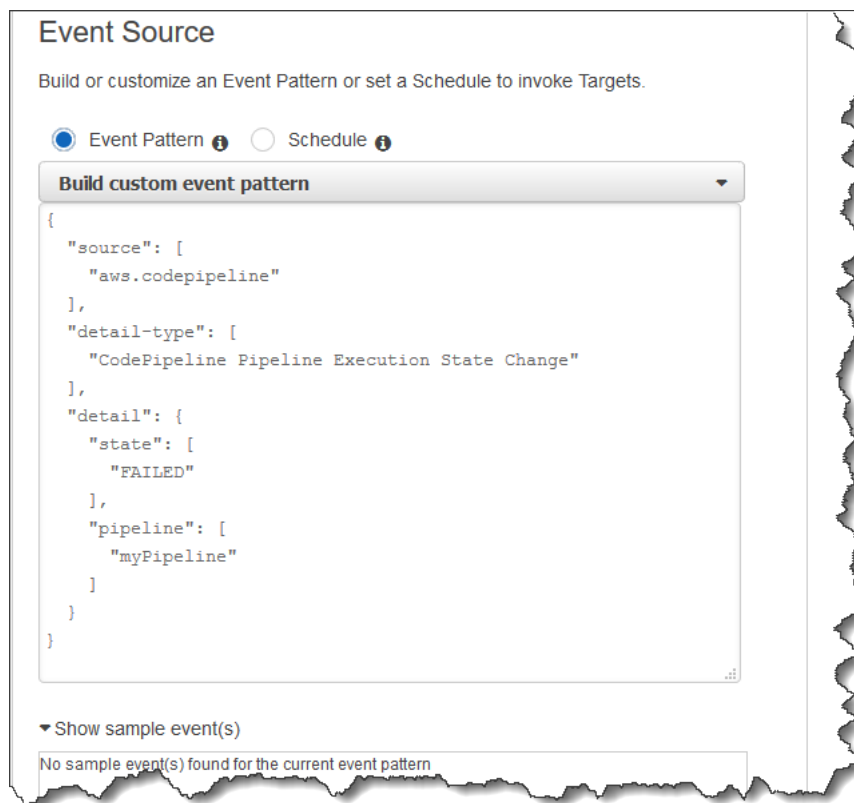
**SNS topic**

Topic\* No

[Configure input](#)

[Add target\\*](#)

6. 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预览窗口中的编辑选项指定 JSON 格式的事件模式。以下示例显示手动编辑以指定名为“myPipeline”的管道的 JSON 结构。



#### 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编辑窗口中。

- 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"]
  }
}
```

7. 在 Targets 区域，选择 Add target\*。
8. 在选择目标类型列表中，为该规则选择目标类型，然后配置该类型所需的选项。
9. 选择 Configure details。
10. 在配置规则详细信息页中，为该规则键入名称和描述，然后选中状态框以立即启用该规则。
11. 选择 Create rule。

## 每次管道状态发生更改时发送通知 (CLI)

这些步骤说明了如何使用 CLI 创建 CloudWatch Events 规则以发送 AWS CodePipeline 中的更改通知。

要使用 AWS CLI 创建规则，请调用 put-rule 命令并指定以下内容：

- 唯一地标识创建的规则的名称。在使用与您的 AWS 账户关联的 AWS CodePipeline 创建的所有管道中，该名称必须是唯一的。
- 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 CloudWatch Events 和事件模式](#)。

创建 CloudWatch Events 规则并将 AWS CodePipeline 作为事件源

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。)

以下示例命令使用 `--event-pattern` 创建名为 “MyPipelineStateChanges” 的规则，以便在名为 “myPipeline” 的管道的管道执行失败时发出 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` 命令以在新规则中添加一个目标，如下示例中的 Amazon SNS 主题所示：

```
aws events put-targets --rule MyPipelineStateChanges --targets Id=1,Arn=arn:aws:sns:us-west-2:11111EXAMPLE:MyNotificationTopic
```

3. 为 Amazon CloudWatch Events 添加权限以使用指定的目标服务调用通知。有关更多信息，请参阅[将基于资源的策略用于 Amazon CloudWatch Events](#)。

## 使用 AWS CloudTrail 记录 AWS CodePipeline API 调用

AWS CodePipeline 与 CloudTrail 集成，后者是一种服务，它在 AWS 账户中捕获由 AWS CodePipeline 自身或代表其发出的 API 调用，并将日志文件传输到您指定的 Amazon S3 存储桶。CloudTrail 从 AWS CodePipeline 控制台、通过 AWS CLI 从 AWS CodePipeline 命令或直接从 AWS CodePipeline API 捕获 API 调用。通过 CloudTrail 收集的信息，您可以确定向 AWS CodePipeline 发出了什么请求、发出请求的源 IP 地址、何人发出的请求以及发出请求的时间等。要了解有关 CloudTrail 的更多信息，包括如何对其进行配置和启用，请参阅 [AWS CloudTrail User Guide](#)

## CloudTrail 中的 AWS CodePipeline 信息

在您的 AWS 账户中启用 CloudTrail 日志记录后，将在日志文件中跟踪对 AWS CodePipeline 执行的 API 调用。AWS CodePipeline 记录与其他 AWS 服务记录一起写入日志文件。CloudTrail 基于时间段和文件大小来确定何时创建新文件并向其写入内容。

所有 AWS CodePipeline 操作都会被记录，并正式记载到 [AWS CodePipeline API 参考](#) 和 [AWS CodePipeline 命令行参考](#) 中。例如，创建、删除和编辑管道以及创建自定义操作的调用会在 CloudTrail 日志文件中生成条目。

每个日志条目都包含有关生成请求的人员的信息。日志中的用户身份信息有助于确定发出的请求是否具有根或 IAM 用户证书，是否具有角色或联合用户的临时安全证书，或者是否是由其他 AWS 服务发出的。有关更多信息，请参阅 [CloudTrail 事件参考](#) 中的 `userIdentity` 字段。

日志文件可以在存储桶中存储任意长时间，不过您也可以定义 Amazon S3 生命周期规则以自动存档或删除日志文件。默认情况下，将使用 Amazon S3 服务器端加密 (SSE) 对日志文件进行加密。

如果需要针对日志文件传输快速采取措施，可选择让 CloudTrail 在传输新日志文件时发布 Amazon SNS 通知。有关更多信息，请参阅 [配置 Amazon SNS 通知](#)。

您也可以将多个 AWS 区域和多个 AWS 账户中的 AWS CodePipeline 日志文件聚合到单个 Amazon S3 存储桶中。更多信息，请参阅[将 CloudTrail 日志文件聚合到单个 Amazon S3 存储桶中](#)。

## 了解 AWS CodePipeline 日志文件条目

CloudTrail 日志文件可包含一个或多个日志条目，每个条目由多个 JSON 格式的事件组成。一个日志条目表示来自任何源的一个请求，包括有关所请求的操作、所有参数以及操作的日期和时间等信息。不能保证日志条目具有任何特定顺序（即，它们不是公共 API 调用的有序堆栈跟踪）。

以下示例显示某更新管道事件的 CloudTrail 日志条目，其中名为 MyFirstPipeline 的管道已被名为 JaneDoe-CodePipeline、账户 ID 为 80398EXAMPLE 的用户编辑。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 元素同时包含已编辑管道的整个结构，因此这些元素在下面的示例中已被简化。对发生更改的管道的 requestParameters 部分和 responseElements 部分进行了强调，前一部分表示管道以前的版本号，后一部分显示加 1 后的版本号。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/AWS-CodePipeline-Service",
        "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"  
  }  
]
```

## 查看 AWS CodePipeline 管道中的当前源修订详细信息

您可以使用 AWS CodePipeline 控制台或 AWS CLI 查看在管道执行过程中使用的源项目 (在管道的第一阶段产生的输出项目) 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, 您可以查看并打开项目版本对应的提交内容的 URL。

### 主题

- [查看管道中的当前源修订详细信息 \(控制台\) \(p. 162\)](#)
- [查看管道中的当前源修订详细信息 \(CLI\) \(p. 163\)](#)

## 查看管道中的当前源修订详细信息 (控制台)

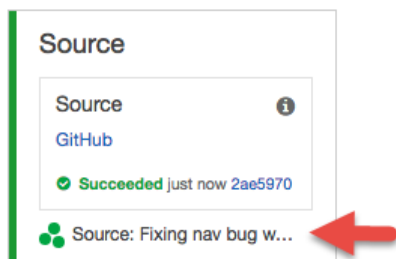
您可以使用 AWS CodePipeline 控制台查看有关管道执行中包含的最新源修订的信息。

### 要查看管道中的源修订

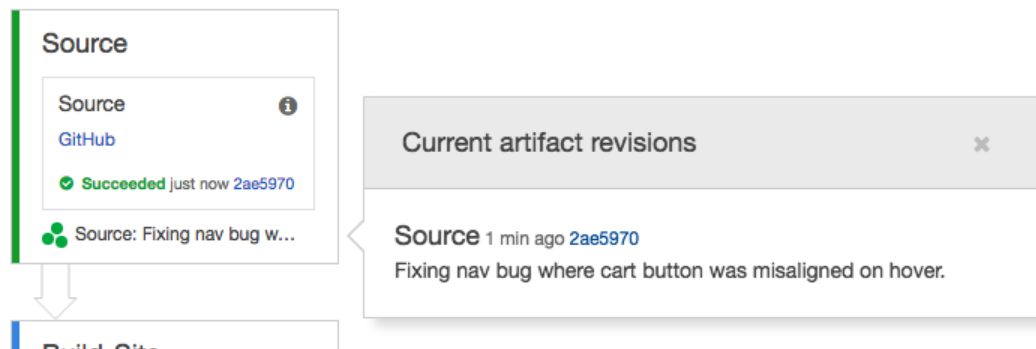
1. 登录 AWS 管理控制台并通过以下网址打开 AWS CodePipeline 控制台 : <http://console.aws.amazon.com/codepipeline>。

所有与您的 AWS 账户关联的管道的名称将会显示。

2. 选择您要查看源修订详细信息的管道的名称。
3. 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 :



4. 单击详细信息区域查看有关项目的更多信息, 包括自项目提交以来的时间长度。除存储在 Amazon S3 存储桶中的项目之外, 在该详细信息视图中, 提交 ID 等标识符与项目的源信息页面关联。



## 查看管道中的当前源修订详细信息 (CLI)

可以运行 `get-pipeline-execution` 命令查看有关管道执行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特定阶段的执行 ID，然后在 `get-pipeline-execution` 命令中使用该执行 ID。（管道中的各阶段之前可能是在不同的管道运行期间成功完成的，因此可能具有不同的执行 ID。）

换句话说，如果要查看当前处于 Staging 阶段的项目的详细信息，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。

要查看管道中的源修订

1. 打开终端 (Linux, macOS, or Unix) 或命令提示符 (Windows)，并使用 AWS CLI 运行 `get-pipeline-state` 命令。对于名为 `MyFirstPipeline` 的管道，请键入：

```
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项目详细信息的特定执行的管道执行 ID。例如，要查看有关名为 `MyFirstPipeline`、执行 ID 为 `3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE` 的管道执行的详细信息，请键入以下内容：

```
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。

以下示例显示名为 `MyFirstPipeline` 的管道的某个部分返回的数据，其中一个名为“MyApp”的项目存储在 GitHub 存储库中：

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"
      }
    ]
  }
}
```

//More revisions might be listed here

```
    ],  
    "pipelineExecutionId": "3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE",  
    "pipelineName": "MyFirstPipeline",  
    "pipelineVersion": 2,  
    "status": "Succeeded"  
  }  
}
```

# AWS CodePipeline 故障排除

以下信息可帮助您排查 AWS CodePipeline 中的常见问题。

## 主题

- 管道错误：使用 AWS Elastic Beanstalk 配置的管道返回错误消息：“Deployment failed. The provided role does not have sufficient permissions: Service:AmazonElasticLoadBalancing” (p. 165)
- 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 (p. 166)
- 管道错误：源操作返回权限不足消息：“Could not access the AWS CodeCommit repository repository-name. Make sure that the pipeline IAM role has sufficient permissions to access the repository.” (p. 166)
- 管道错误：Jenkins 生成或测试操作运行很长时间，然后由于缺少凭证或权限而失败 (p. 167)
- 管道错误：我的 GitHub 源阶段包含 Git 子模块，但 AWS CodePipeline 不对它们进行初始化 (p. 167)
- 管道错误：我收到一个管道错误，其中包含以下消息：“PermissionError: Could not access the GitHub repository” (p. 167)
- 管道错误：在一个 AWS 区域中使用在另一个 AWS 区域中创建的存储桶创建的管道返回“InternalError”，代码为“JobFailed” (p. 168)
- 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 404 Not Found 错误 (p. 166)
- 在 ZIP 不保留外部属性时，在 GitHub 的源文件上不保留文件权限 (p. 169)
- 需要有关其他问题的帮助？ (p. 170)

## 管道错误：使用 AWS Elastic Beanstalk 配置的管道返回错误消息：“Deployment failed. The provided role does not have sufficient permissions: Service:AmazonElasticLoadBalancing”

问题：AWS CodePipeline 的服务角色对 AWS Elastic Beanstalk 不具有足够的权限，包括但不限于 Elastic Load Balancing 中的一些操作。2015 年 8 月 6 日已更新 AWS CodePipeline 的服务角色来解决此问题。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从[查看默认 AWS CodePipeline 服务角色策略 \(p. 181\)](#)复制更新后的服务角色策略语句，编辑服务角色，并使用当前策略覆盖旧策略语句。策略语句的这一部分适用于 Elastic Beanstalk：

```
{
  "Action": [
    "elasticbeanstalk:*",
    "ec2:*",
    "elasticloadbalancing:*"
  ]
}
```

```
"autoscaling:*",
"cloudwatch:*",
"s3:*",
"sns:*",
"cloudformation:*",
"rds:*",
"sqs:*",
"ecs:*",
"iam:PassRole"
],
"Resource": "*",
"Effect": "Allow"
},
```

应用编辑过的策略后，按照在 [AWS CodePipeline 中手动启动管道 \(p. 80\)](#) 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：对于使用 AWS Elastic Beanstalk 的任何管道，AWS CodePipeline 的服务角色必须包含 "elasticbeanstalk:DescribeEvents" 操作。没有此权限，AWS Elastic Beanstalk 部署操作会挂起但不会失败或指示错误。如果您的服务角色缺少此操作，则 AWS CodePipeline 将无权代表您在 AWS Elastic Beanstalk 中运行管道部署阶段。

可能的修复措施：检查您的 AWS CodePipeline 服务角色。如果缺少 "elasticbeanstalk:DescribeEvents" 操作，请使用 [添加其他 AWS 服务的权限 \(p. 182\)](#) 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，按照在 [AWS CodePipeline 中手动启动管道 \(p. 80\)](#) 中的步骤手动重新运行使用 Elastic Beanstalk 的任何管道。

有关默认服务角色的更多信息，请参阅 [查看默认 AWS CodePipeline 服务角色策略 \(p. 181\)](#)。

## 管道错误：源操作返回权限不足消息：“Could not access the AWS CodeCommit repository repository-name. Make sure that the pipeline IAM role has sufficient permissions to access the repository.”

问题：AWS CodePipeline 的服务角色不具有对 AWS CodeCommit 的足够权限，可能是在 2016 年 4 月 18 日添加对使用 AWS CodeCommit 存储库的支持之前创建的。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：将 AWS CodeCommit 所需的权限添加到 AWS CodePipeline 服务角色的策略中。有关更多信息，请参阅 [添加其他 AWS 服务的权限 \(p. 182\)](#)。

## 管道错误：Jenkins 生成或测试操作运行很长时间，然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 AWS CodePipeline 所需权限的实例角色创建的。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 [适用于 AWS CodePipeline 的 AWS 托管 \(预定义\) 策略 \(p. 180\)](#)。

如果您正在使用 IAM 用户，请确保实例上配置的 AWS 配置文件使用配置了正确权限的 IAM 用户。您可能必须直接在 Jenkins UI 中提供您配置的用于在 Jenkins 和 AWS CodePipeline 之间进行集成的 IAM 用户凭证。这不是推荐的最佳实践。如果必须这么做，请确保 Jenkins 服务器已受到安全保护，并使用 HTTPS 而不是 HTTP。

## 管道错误：我的 GitHub 源阶段包含 Git 子模块，但 AWS CodePipeline 不对它们进行初始化

问题：AWS CodePipeline 不支持 Git 子模块。AWS CodePipeline 依赖于 GitHub 的存档链接 API，该 API 不支持子模块。

可能的修复措施：考虑将 GitHub 存储库直接克隆为单独脚本的一部分。例如，您可以在 Jenkins 脚本中包含克隆操作。

## 管道错误：我收到一个管道错误，其中包含以下消息：“PermissionError: Could not access the GitHub repository”

问题：AWS CodePipeline 使用 OAuth 令牌与 GitHub 集成。您可能已经撤销了 OAuth 令牌对 AWS CodePipeline 的权限。此外，令牌数是有限的 (请参阅 [GitHub 文档](#)，获取详细信息)。如果 AWS CodePipeline 达到该限制，旧令牌将停止工作，管道中依赖该令牌的操作将失败。

可能的修复措施：检查 AWS CodePipeline 的权限是否已撤销。登录到 GitHub，转到应用程序，然后选择授权的 OAuth 应用程序。如果未在列表中看到 AWS CodePipeline，请打开 AWS CodePipeline 控制台，编辑管道，然后选择 Connect to GitHub 以恢复授权。

如果 AWS CodePipeline 存在于 GitHub 的授权应用程序列表中，则您可能已超过允许的令牌数。要解决此问题，您可以手动将一个 OAuth 令牌配置为个人访问令牌，然后将您的 AWS 账户中的所有管道配置为使用该令牌。

安全最佳实践是定期轮换您的个人访问令牌。有关更多信息，请参阅 [使用 GitHub 和 AWS CodePipeline CLI 创建并定期轮换您的 GitHub 个人访问令牌 \(p. 198\)](#)。

## 配置管道以使用 GitHub 的个人访问令牌

1. 登录您的 GitHub 账户，并按照[关于创建个人访问令牌的 GitHub 文档](#)中的说明进行操作。确保配置令牌以包含以下 GitHub 范围：admin:repo\_hook 和 repo。复制令牌。
2. 在终端 (Linux, macOS, or Unix) 或命令提示符 (Windows) 中，对您要更改 OAuth 令牌的管道运行 get-pipeline 命令，然后将命令输出复制到 JSON 文件。例如，对于名为 MyFirstPipeline 的管道，您应键入类似以下的内容：

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

命令输出将会发送到 *pipeline.json* 文件。

3. 在纯文本编辑器中打开该文件，并在您的 GitHub 操作的 OAuthTokenField 中编辑值。使用您从 GitHub 中复制的令牌替换星号 (\*\*\*\*)。例如：

```
"configuration": {  
    "Owner": "MyGitHubUserName",  
    "Repo": "test-repo",  
    "Branch": "master",  
    "OAuthToken": "Replace the **** with your personal access  
token"  
},
```

4. 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 行来修改 JSON 文件中的结构，否则 update-pipeline 命令将无法使用管道结构。从 JSON 文件中的管道结构中删除该部分 ("metadata": {} 行以及其中的“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"  
}
```

5. 保存文件，然后运行带有 --cli-input-json 参数的 update-pipeline，以指定您刚刚编辑的 JSON 文件。例如，要更新名为 MyFirstPipeline 的管道，您应该键入类似下面的内容：

### 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
```

6. 对每个包含 GitHub 操作的管道都重复第 2 步到第 4 步。
7. 更新完您的管道后，请删除用于更新这些管道的 .json 文件。

## 管道错误：在一个 AWS 区域中使用在另一个 AWS 区域中创建的存储桶创建的管道返回“InternalError”，代码为“JobFailed”

问题：如果管道和 Amazon S3 存储桶是在不同的 AWS 区域创建的，那么存储在存储桶中的项目的下载将失败。

可能的修复措施：确保持存项目的 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 404 Not Found 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。

可能的修复措施：AWS Elastic Beanstalk 可以解压缩 ZIP 文件，但不能解压缩包含在 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: 0.1

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 AWS CodeBuild 的 AWS Elastic Beanstalk 示例](#)。

## 在 ZIP 不保留外部属性时，在 GitHub 的源文件上不保留文件权限

问题：使用 GitHub 的源文件的用户可能会遇到以下问题：在存储在管道的 Amazon S3 项目存储桶时，输入项目丢失文件权限。压缩 GitHub 文件的 AWS CodePipeline 源操作使用的 ZIP 文件进程不保留外部属性，因此，这些文件不保留文件权限。

可能的修复措施：您必须通过 `chmod` 命令修改使用项目所需的文件权限。请在生成提供程序 (如 AWS CodeBuild) 中更新生成规范文件，以便在每次运行生成阶段时恢复文件权限。以下示例显示了 AWS CodeBuild 生成规范，并在 `build` 部分中包含 `chmod` 命令：

```
version: 0.2

phases:
  build:
    commands:
      - dotnet restore
      - dotnet build
      - chmod a+x bin/Debug/myTests
      - bin/Debug/myTests
artifacts:
  files:
    - bin/Debug/myApplication
```

### Note

要使用 AWS CodePipeline 控制台确认生成输入项目的名称，请显示管道，然后在生成操作中将鼠标指针悬停在工具提示上。记下输入项目的值 (例如，MyApp)。要使用 AWS CLI 获取 S3 项目存储桶的名称，请运行 `AWS CodePipeline get-pipeline` 命令。输出包含一个 `artifactStore` 对象，其中的 `location` 字段显示存储桶的名称。



## 需要有关其他问题的帮助？

请尝试这些其他资源：

- 请联系 [AWS Support](#)。
- 在 [AWS CodePipeline 论坛](#)中提问。
- [请求提高限制](#)。有关更多信息，请参阅 [AWS CodePipeline 中的限制 \(p. 208\)](#)。

### Note

最多可能需要两周的时间来处理提高限制的请求。

# AWS CodePipeline 身份验证、访问控制和安全配置

访问 AWS CodePipeline 需要凭证。这些凭证必须具有访问 AWS 资源的权限，例如从 Amazon S3 存储桶检索项目；访问有关 AWS CodeDeploy 中的应用程序和部署组的信息；以及授予批准或拒绝手动审批操作的权限。下面几节提供详细的信息来说明如何使用 [AWS Identity and Access Management \(IAM\)](#) 和 AWS CodePipeline 来帮助保护对您的资源的访问：

- [身份验证 \(p. 171\)](#)
- [使用 IAM 策略控制访问 \(p. 172\)](#)
- [安全配置 \(p. 195\)](#)

## 身份验证

您可以下面任一类型的身份访问 AWS：

- **AWS 账户根用户** – 注册 AWS 时，您需要提供与您的 AWS 账户关联的电子邮件地址和密码。这些是您的根凭证，它们提供对您所有 AWS 资源的完全访问权限。

### Important

出于安全考虑，我们建议您仅使用根凭证创建管理员用户，此类用户是对您的 AWS 账户具有完全访问权限的 IAM 用户。随后，您可以使用此管理员用户来创建具有有限权限的其他 IAM 用户和角色。有关更多信息，请参阅 IAM 用户指南中的 [IAM 最佳实践](#)和[创建管理员用户和组](#)。

- **IAM 用户** - [IAM 用户](#)就是您的 AWS 账户中的一种身份，它具有特定的自定义权限（例如，用于在 AWS CodePipeline 中向目标发送事件数据的权限）。您可以使用 IAM 用户名和密码来登录以保护 AWS 网页，如 [AWS 管理控制台](#)、[AWS 开发论坛](#)或 [AWS Support Center](#)。

除了用户名和密码之外，您还可以为每个用户生成[访问密钥](#)。在通过[多个软件开发工具包之一](#)或使用 [AWS Command Line Interface \(AWS CLI\)](#) 以编程方式访问 AWS 服务时，可以使用这些密钥。SDK 和 CLI 工具使用访问密钥对您的请求进行加密签名。如果您不使用 AWS 工具，则必须自行对请求签名。AWS CodePipeline supports 签名版本 4，后者是一种用于对入站 API 请求进行身份验证的协议。有关验证请求的更多信息，请参阅 AWS General Reference 中的[签名版本 4 签名流程](#)。

- **IAM 角色** – [IAM 角色](#)是可在账户中创建的另一种具有特定权限的 IAM 身份。它类似于 IAM 用户，但未与特定人员关联。利用 IAM 角色，您可以获得可用于访问 AWS 服务和资源的临时访问密钥。具有临时凭证的 IAM 角色在以下情况下很有用：
  - **联合身份用户访问** – 您可以不创建 IAM 用户，而是使用来自 AWS Directory Service、您的企业用户目录或 Web 身份提供商的既有用户身份。他们被称为联合身份用户。在通过[身份提供商](#)请求访问权限时，AWS 将为联合身份用户分配角色。有关联合身份用户的更多信息，请参阅 IAM 用户指南中的[联合身份用户和角色](#)。
  - **跨账户访问** – 可以使用您账户中的 IAM 角色向另一个 AWS 账户授予对您账户的资源的访问权限。有关示例，请参阅 IAM 用户指南中的[教程：使用 IAM 角色委派跨 AWS 账户的访问权限](#)。

- **AWS 服务访问** – 可以使用您账户中的 IAM 角色向 AWS 服务授予对您账户的资源的访问权限。例如，您可以创建一个角色，此角色允许 Amazon Redshift 代表您访问 Amazon S3 存储桶，然后将存储在该存储桶中的数据加载到 Amazon Redshift 群集中。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](#)。
- **在 Amazon EC2 上运行的应用程序** – 您不用将访问密钥存储在 EC2 实例中以供实例上运行的应用程序使用并发出 AWS API 请求，而是可以使用 IAM 角色管理这些应用程序的临时凭证。要将 AWS 角色分配给 EC2 实例并使其对该实例的所有应用程序可用，您可以创建一个附加到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 [对 Amazon EC2 上的应用程序使用角色](#)。

## 使用 IAM 策略控制访问

您可以使用有效的凭证来对自己的请求进行身份验证，但您还必须拥有权限才能创建或访问 AWS CodePipeline 资源。例如，您必须具有以下权限：创建、查看或删除管道；检索有关 AWS CodeDeploy 中的应用程序和部署组的信息，以及有关 AWS OpsWorks Stacks 中的堆栈、应用程序和层的信息；在 Amazon Simple Storage Service 存储桶中添加或检索项目，等等。

下面几节介绍如何管理 AWS CodePipeline 的权限。我们建议您先阅读概述。

- [管理您的 AWS CodePipeline 资源的访问权限概述 \(p. 172\)](#)
- [为 AWS CodePipeline 使用基于身份的策略 \(IAM 策略\) \(p. 179\)](#)
- [AWS CodePipeline 权限参考 \(p. 192\)](#)

## 管理您的 AWS CodePipeline 资源的访问权限概述

每个 AWS 资源都归某个 AWS 账户所有，创建和访问资源的权限由权限策略进行管理。账户管理员可以向 IAM 身份（即：用户、组和角色）挂载权限策略，某些服务（如 AWS Lambda）也支持向资源挂载权限策略。

### Note

账户管理员（或管理员 IAM 用户）是具有管理员权限的用户。有关更多信息，请参阅 IAM 用户指南 中的 [IAM 最佳实践](#)。

在授予权限时，您要决定谁获得权限，获得对哪些资源的权限，以及您允许对这些资源执行的具体操作。

### 主题

- [AWS CodePipeline 资源和操作 \(p. 172\)](#)
- [AWS CodePipeline API 调用支持的资源级权限 \(p. 174\)](#)
- [了解资源所有权 \(p. 176\)](#)
- [管理对资源的访问 \(p. 176\)](#)
- [指定策略元素：操作、效果和委托人 \(p. 178\)](#)
- [在策略中指定条件 \(p. 179\)](#)

## AWS CodePipeline 资源和操作

在 AWS CodePipeline 中，主要资源为管道。在策略中，您可以使用 Amazon 资源名称 (ARN) 标识策略应用到的资源。AWS CodePipeline 支持可以与主要资源一起使用的其他资源，例如阶段、操作和自定义操作。这些资源称为子资源。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 的详细信息

信息，请参阅 Amazon Web Services 一般参考 中的 [Amazon 资源名称 \(ARN\)](#) 和 [AWS 服务命名空间](#)。要获取与管道关联的管道 ARN，请使用 CLI 运行 `get-pipeline` 命令。有关更多信息，请参阅 [AWS CodePipeline API 参考](#) 中的 [GetPipeline](#)。

资源类型	ARN 格式
管道	<code>arn:aws:codepipeline:<i>region</i>:<i>account</i>:<i>pipeline-name</i></code>
阶段	<code>arn:aws:codepipeline:<i>region</i>:<i>account</i>:<i>pipeline-name</i>/<i>stage-name</i></code>
操作	<code>arn:aws:codepipeline:<i>region</i>:<i>account</i>:<i>pipeline-name</i>/<i>stage-name</i>/<i>action-name</i></code>
自定义操作	<code>arn:aws:codepipeline:<i>region</i>:<i>account</i>:actionType:<i>owner</i>/<i>category</i>/<i>provider</i>/<i>version</i></code>
所有 AWS CodePipeline 资源	<code>arn:aws:codepipeline:*</code>
指定账户在指定区域拥有的所有 AWS CodePipeline 资源	<code>arn:aws:codepipeline:<i>region</i>:<i>account</i>:*</code>

#### Note

AWS 中的大多数服务将 ARN 中的冒号 (:) 或正斜杠 (/) 视为相同的字符。不过，AWS CodePipeline 在事件模式和规则中使用精确匹配。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 AWS CodePipeline 中，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。请参阅 [AWS CodePipeline API 调用支持的资源级权限](#) (p. 174)，以了解资源级权限的详细描述以及支持资源级权限的 AWS CodePipeline API 调用的列表。

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，请遵循授予最小权限这一标准的安全建议，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 (\*)。

有些 AWS 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"]
```

AWS CodePipeline 提供一组操作用来处理 AWS CodePipeline 资源。有关可用操作的列表，请参阅 [AWS CodePipeline 权限参考](#) (p. 192)。

## AWS CodePipeline API 调用支持的资源级权限

资源级权限是指能让您指定允许用户对哪些资源执行操作的权限。AWS CodePipeline 对资源级权限提供部分支持。这意味着，对于某些 AWS CodePipeline API 调用，您可以控制何时允许用户执行那些操作（基于必须满足的条件）或是允许用户使用哪些资源。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。

下表介绍当前支持资源级权限的 AWS CodePipeline API 调用，以及每个操作支持的资源、资源 ARN 和条件密钥。

### Note

在此表中没有列出的 AWS CodePipeline API 调用不支持资源级权限。如果 AWS CodePipeline API 调用不支持资源级权限，那么，您可以向用户授予使用该调用的权限，但是必须为策略语句的资源元素指定 \*。

API 调用	资源类型
PutActionRevision	管道 arn:aws:codepipeline: <i>region</i> : <i>account</i> : <i>pipeline-name</i>
DeleteCustomActionType	操作类型 arn:aws:codepipeline: <i>region</i> : <i>account</i> :actionType: <i>owner</i> / <i>category</i> / <i>provider</i> / <i>version</i>
CreatePipeline	管道 arn:aws:codepipeline: <i>region</i> : <i>account</i> : <i>pipeline-name</i>
ListPipelines	管道 arn:aws:codepipeline: <i>region</i> : <i>account</i> : <i>pipeline-name</i>
PollForJobs	操作类型 arn:aws:codepipeline: <i>region</i> : <i>account</i> :actionType: <i>owner</i> / <i>category</i> / <i>provider</i> / <i>version</i>
DisableStageTransition	管道 arn:aws:codepipeline: <i>region</i> : <i>account</i> : <i>pipeline-name</i>
StartPipelineExecution	管道 arn:aws:codepipeline: <i>region</i> : <i>account</i> : <i>pipeline-name</i>
UpdatePipeline	管道 arn:aws:codepipeline: <i>region</i> : <i>account</i> : <i>pipeline-name</i>
GetPipelineState	管道 arn:aws:codepipeline: <i>region</i> : <i>account</i> : <i>pipeline-name</i>
RetryStageExecution	管道 arn:aws:codepipeline: <i>region</i> : <i>account</i> : <i>pipeline-name</i>
ListActionTypes	操作类型

API 调用	资源类型
	arn:aws:codepipeline: <i>region</i> : <i>account</i> :actionType: <i>owner</i> / <i>category</i> / <i>provider</i> / <i>version</i>
CreateCustomActionType	操作类型  arn:aws:codepipeline: <i>region</i> : <i>account</i> :actionType: <i>owner</i> / <i>category</i> / <i>provider</i> / <i>version</i>
GetPipelineExecution	管道  arn:aws:codepipeline: <i>region</i> : <i>account</i> : <i>pipeline-name</i>
GetPipeline	管道  arn:aws:codepipeline: <i>region</i> : <i>account</i> : <i>pipeline-name</i>
ListPipelineExecutions	管道  arn:aws:codepipeline: <i>region</i> : <i>account</i> : <i>pipeline-name</i>
DeletePipeline	管道  arn:aws:codepipeline: <i>region</i> : <i>account</i> : <i>pipeline-name</i>
EnableStageTransition	管道  arn:aws:codepipeline: <i>region</i> : <i>account</i> : <i>pipeline-name</i>
PutApprovalResult	操作  arn:aws:codepipeline: <i>region</i> : <i>account</i> : <i>pipeline-name</i> / <i>stage-name</i> / <i>action-name</i>  Note  此 API 调用支持资源级权限。但是，如果您使用 IAM 控制台或策略生成器来通过指定资源 ARN 的 "codepipeline:PutApprovalResult" 创建策略，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。
DeleteWebhook	Webhook  arn:aws:codepipeline: <i>region</i> : <i>account</i> :webhook: <i>webhook-name</i>
DeregisterWebhookWithThirdParty	Webhook  arn:aws:codepipeline: <i>region</i> : <i>account</i> :webhook: <i>webhook-name</i>
PutWebhook	管道  arn:aws:codepipeline: <i>region</i> : <i>account</i> : <i>pipeline-name</i>
	Webhook  arn:aws:codepipeline: <i>region</i> : <i>account</i> :webhook: <i>webhook-name</i>
RegisterWebhookWithThirdParty	Webhook  arn:aws:codepipeline: <i>region</i> : <i>account</i> :webhook: <i>webhook-name</i>

## 了解资源所有权

AWS 账户对在该账户下创建的资源具有所有权，而无论创建资源的人员是谁。具体而言，资源所有者是对资源创建请求进行身份验证的 **委托人实体** (即根账户、IAM 用户或 IAM 角色) 的 AWS 账户。以下示例说明了它的工作原理：

- 如果您使用 AWS 账户的根账户凭证创建规则，则您的 AWS 账户即为该 AWS CodePipeline 资源的所有者。
- 如果您在您的 AWS 账户中创建 IAM 用户并对该用户授予创建 AWS CodePipeline 资源的权限，则该用户可以创建 AWS CodePipeline 资源。但是，该用户所属的 AWS 账户拥有这些 AWS CodePipeline 资源。
- 如果您在您的 AWS 账户中创建具有创建 AWS CodePipeline 资源的权限的 IAM 角色，则能够担任该角色的任何人都可以创建 AWS CodePipeline 资源。该角色所属的 AWS 账户拥有这些 AWS CodePipeline 资源。

## 管理对资源的访问

权限策略 规定谁可以访问哪些内容。下一节介绍创建权限策略时的可用选项。

### Note

本节讨论如何在 AWS CodePipeline 范围内使用 IAM。它不提供有关 IAM 服务的详细信息。有关完整的 IAM 文档，请参阅 [什么是 IAM？](#) (在 IAM 用户指南 中)。有关 IAM 策略语法和说明的信息，请参阅 IAM 用户指南 中的 [IAM 策略参考](#)。

附加到 IAM 身份的策略称为基于身份的策略 (IAM 策略)，而附加到资源的策略称为基于资源的策略。AWS CodePipeline 只支持基于身份的策略 (IAM 策略)。

### 主题

- [基于身份的策略 \(IAM 策略\) \(p. 176\)](#)
- [基于资源的策略 \(p. 178\)](#)

## 基于身份的策略 (IAM 策略)

您可以向 IAM 身份挂载策略。例如，您可以执行以下操作：

- 将权限策略附加到您的账户中的用户或组 - 要授予用户查看 AWS CodePipeline 控制台中的管道的权限，您可以将权限策略附加到用户或用户所属的组。
- 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，以向其他 AWS 账户 (如账户 B) 或某项 AWS 服务授予跨账户权限，如下所述：
  1. 账户 A 管理员创建一个 IAM 角色，向该角色挂载授权其访问账户 A 中资源的权限策略。
  2. 账户 A 管理员可以向将账户 B 标识为能够担任该角色的委托人的角色挂载信任策略。
  3. 之后，账户 B 管理员可以委派权限，指定账户 B 中的任何用户担任该角色。这样，账户 B 中的用户就可以创建或访问账户 A 中的资源了。如果您需要授予 AWS 服务权限来担任该角色，则信任策略中的委托人也可以是 AWS 服务委托人。

有关使用 IAM 委派权限的更多信息，请参阅 IAM 用户指南 中的 [访问权限管理](#)。

以下示例显示 111222333444 账户中的策略，该策略允许用户在 AWS CodePipeline 控制台中查看但不能更改名为 **MyFirstPipeline** 的管道。该策略基于 AWSCodePipelineReadOnlyAccess 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于某特定管道，则绝对应该考虑使用 AWS CodePipeline 创建和维护的托管策略之一。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:ListActionTypes",
        "codepipeline:ListPipelines",
        "iam:ListRoles",
        "s3:GetBucketPolicy",
        "s3:GetObject",
        "s3:ListAllMyBuckets",
        "s3:ListBucket",
        "codecommit:ListBranches",
        "codecommit:ListRepositories",
        "codedeploy:GetApplication",
        "codedeploy:GetDeploymentGroup",
        "codedeploy:ListApplications",
        "codedeploy:ListDeploymentGroups",
        "elasticbeanstalk:DescribeApplications",
        "elasticbeanstalk:DescribeEnvironments",
        "lambda:GetFunctionConfiguration",
        "lambda:ListFunctions",
        "opsworks:DescribeApps",
        "opsworks:DescribeLayers",
        "opsworks:DescribeStacks"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ],
  "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。以下示例显示了允许 **111111111111** AWS 账户中的用户担任 111222333444 账户中定义的角色策略：

```
{
  "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，该策略允许用户担任 111222333444 账户中名为 **CrossAccountPipelineViewers** 的角色：

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



```
        "Resource": "arn:aws:iam::111222333444:role/CrossAccountPipelineViewers"
    }
  ]
}
```

可以创建特定的 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 IAM 用户关联。有关如何创建 IAM 角色和探究 AWS CodePipeline 的 IAM 策略语句示例的更多信息，请参阅[管理您的 AWS CodePipeline 资源的访问权限概述](#) (p. 172)。

## 基于资源的策略

其他服务 (如 Amazon S3) 也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。虽然 AWS CodePipeline 不支持基于资源的策略，但它会将管道中要使用的项目存储在受版本控制的 S3 存储桶中。

Example 为要用作 AWS CodePipeline 的项目存储的 Amazon S3 存储桶创建策略

您可以将任何受版本控制的 Amazon S3 存储桶用作 AWS CodePipeline 的项目存储。如果您使用 Create Pipeline 向导创建第一个管道，则系统将会自动为您创建此 Amazon S3 存储桶，以确保上传到项目存储的所有对象都已加密，并且与存储桶的连接是安全的。作为最佳实践，如果您创建自己的 Amazon S3 存储桶，请考虑向存储桶添加以下策略或其元素。在此策略中，Amazon S3 存储桶的 ARN 是 *codepipeline-us-east-2-1234567890*。将此 ARN 替换为您的 Amazon 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
        }
      }
    }
  ]
}
```

## 指定策略元素：操作、效果和委托人

对于每个 AWS CodePipeline 资源，该服务都定义了一组 API 操作。为授予这些 API 操作的权限，AWS CodePipeline 定义了一组您可以在策略中指定的操作。某些 API 操作可能需要多个操作的权限才能执行 API 操作。有关资源和 API 操作的更多信息，请参阅 [AWS CodePipeline 资源和操作](#) (p. 172) 和 [AWS CodePipeline 权限参考](#) (p. 192)。

以下是基本的策略元素：

- Resource - 您使用 Amazon 资源名称 (ARN) 来标识策略应用到的资源。有关更多信息，请参阅 [AWS CodePipeline 资源和操作 \(p. 172\)](#)。
- Action - 您可以使用操作关键字标识要允许或拒绝的资源操作。例如，codepipeline:GetPipeline 权限允许执行 GetPipeline 操作的用户权限。
- Effect - 用于指定当用户请求特定操作时的效果 (可以是允许或拒绝)。如果没有显式授予 (允许) 对资源的访问权限，则隐式拒绝访问。您也可显式拒绝对资源的访问，这样可确保用户无法访问该资源，即使有其他策略授予了访问权限的情况下也是如此。
- Principal - 在基于身份的策略 (IAM 策略) 中，附加了策略的用户是隐式委托人。对于基于资源的策略，您可以指定要接收权限的用户、账户、服务或其他实体 (仅适用于基于资源的策略)。

要了解有关 IAM 策略语法和说明的更多信息，请参阅 IAM 用户指南 中的 [AWS IAM 策略参考](#)。

有关显示所有 AWS CodePipeline API 操作及其适用资源的表，请参阅 [AWS CodePipeline 权限参考 \(p. 192\)](#)。

## 在策略中指定条件

当您授予权限时，可使用访问策略语言来指定规定策略何时生效的条件。例如，您可能希望策略仅在特定日期后应用。有关使用策略语言指定条件的更多信息，请参阅 IAM 用户指南 中的 [条件](#)。

要表示条件，您可以使用预定义的条件键。没有特定于 AWS CodePipeline 的条件键。但有 AWS 范围内的条件密钥，您可以根据需要使用。有关 AWS 范围内的键的完整列表，请参阅 IAM 用户指南 中的 [条件的可用键](#)。

## 为 AWS CodePipeline 使用基于身份的策略 (IAM 策略)

本主题提供了基于身份的策略的示例，这些示例演示了账户管理员如何向 IAM 身份 (即，用户、组和角色) 附加权限策略。

### Important

我们建议您首先阅读以下介绍性主题，这些主题讲解了管理 AWS CodePipeline 资源访问的基本概念和选项。有关更多信息，请参阅 [管理您的 AWS CodePipeline 资源的访问权限概述 \(p. 172\)](#)。

以下几节提供了使用 AWS CodePipeline 特定的 IAM 策略的说明。

下面显示了一个权限策略的示例，该策略允许用户启用和禁用 us-west-2 区域中名为 **MyFirstPipeline** 的管道中的所有阶段过渡：

```
{
  "Version": "2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "codepipeline:EnableStageTransition",
        "codepipeline:DisableStageTransition"
      ],
      "Resource" : [
        "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline"
      ]
    }
  ]
}
```

## 使用 AWS CodePipeline 控制台所需的权限

用户若要能够使用 AWS CodePipeline 控制台中的 AWS CodePipeline，则必须拥有一组为其 AWS 账户描述其他 AWS 资源的最低权限。要使用 AWS CodePipeline 控制台中的 AWS CodePipeline，您必须拥有来自以下服务的权限：

- AWS Identity and Access Management
- Amazon Simple Storage Service

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
- AWS 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 策略，对于附加了该 IAM 策略的用户，控制台将无法按预期正常运行。为确保这些用户仍可使用 AWS CodePipeline 控制台，同时向用户附加 `AWSCodePipelineReadOnlyAccess` 托管策略，请参阅[适用于 AWS CodePipeline 的 AWS 托管 \(预定义\) 策略 \(p. 180\)](#)。

对于只需要调用 AWS CLI 或 AWS CodePipeline API 的用户，您无需为其提供最低控制台权限。

## 适用于 AWS CodePipeline 的 AWS 托管 (预定义) 策略

AWS 通过提供由 AWS 创建和管理的独立 IAM 策略来解决很多常用案例。托管策略可授予常用案例的必要权限，因此，您可以免去调查都需要哪些权限的工作。有关更多信息，请参阅 IAM 用户指南 中的 [AWS 托管策略](#)。

以下 AWS 托管策略 (您可以将它们附加到您账户中的用户) 是特定于 AWS CodePipeline 的：

- `AWSCodePipelineFullAccess` – 授予对 AWS CodePipeline 的完全访问权限。
- `AWSCodePipelineCustomActionAccess` – 授予 IAM 用户在 AWS CodePipeline 中创建自定义操作或将 Jenkins 资源集成到生成或测试操作中的权限。
- `AWSCodePipelineReadOnlyAccess` – 授予对 AWS CodePipeline 的只读访问权限。
- `AWSCodePipelineApproverAccess` – 授予 IAM 用户批准或拒绝手动审批操作的权限。

## 管理 AWS CodePipeline 服务角色

管道执行过程某些方面的权限已授予另一个代表 AWS CodePipeline 进行操作的角色类型，而不是授予 IAM 用户。服务角色 是一个 IAM 角色，可以向 AWS CodePipeline 授予使用您账户中的资源的权限。只有第一次在 AWS CodePipeline 中创建管道时才需要创建服务角色。

通过管道中的阶段和操作处理修订时，AWS CodePipeline 将使用此服务角色。该角色配置有一个或多个策略，以控制对管道使用的 AWS 资源的访问。您可能希望将其他策略附加到该角色，编辑附加到该角色的策略，或者为 AWS 中的其他服务角色配置策略。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。在您以任何方式修改 AWS CodePipeline 的服务角色之前，请务必了解其影响。对服务角色进行任何更改后，确保测试管道。

### 主题

- [查看默认 AWS CodePipeline 服务角色策略 \(p. 181\)](#)
- [添加其他 AWS 服务的权限 \(p. 182\)](#)
- [删除未使用的 AWS 服务的权限 \(p. 184\)](#)

## 查看默认 AWS CodePipeline 服务角色策略

默认情况下，AWS CodePipeline 的 IAM 服务角色 (AWS-CodePipeline-Service) 的策略语句包括 AWS CodePipeline 使用您账户中的其他资源所需的权限。

AWS-CodePipeline-Service 目前包括以下策略语句：

```
{
  "Statement": [
    {
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:GetBucketVersioning"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::codepipeline*",
        "arn:aws:s3:::elasticbeanstalk*"
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "codedeploy:CreateDeployment",
        "codedeploy:GetApplicationRevision",
        "codedeploy:GetDeployment",
        "codedeploy:GetDeploymentConfig",
        "codedeploy:RegisterApplicationRevision"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "elasticbeanstalk:CreateApplicationVersion",
        "elasticbeanstalk:DescribeApplicationVersions",
        "elasticbeanstalk:DescribeEnvironments",
        "elasticbeanstalk:DescribeEvents",
        "elasticbeanstalk:UpdateEnvironment",
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeLaunchConfigurations",
        "autoscaling:DescribeScalingActivities",
        "autoscaling:ResumeProcesses",
        "autoscaling:SuspendProcesses",
        "cloudformation:GetTemplate",

```

```

        "cloudformation:DescribeStackResource",
        "cloudformation:DescribeStackResources",
        "cloudformation:DescribeStackEvents",
        "cloudformation:DescribeStacks",
        "cloudformation:UpdateStack",
        "ec2:DescribeInstances",
        "ec2:DescribeImages",
        "ec2:DescribeAddresses",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeKeyPairs",
        "elasticloadbalancing:DescribeLoadBalancers",
        "rds:DescribeDBInstances",
        "rds:DescribeOrderableDBInstanceOptions",
        "sns:ListSubscriptionsByTopic"
    ],
    "Resource": "*",
    "Effect": "Allow"
  },
  {
    "Action": [
      "lambda:invokefunction",
      "lambda:listfunctions"
    ],
    "Resource": "*",
    "Effect": "Allow"
  },
  {
    "Action": [
      "s3:ListBucket",
      "s3:GetBucketPolicy",
      "s3:GetObjectAcl",
      "s3:PutObjectAcl",
      "s3:DeleteObject"
    ],
    "Resource": "arn:aws:s3:::elasticbeanstalk*",
    "Effect": "Allow"
  }
],
"Version": "2012-10-17"
}

```

#### Note

对于使用 AWS Elastic Beanstalk 的任何管道，确保 AWS CodePipeline 的服务角色包含 "elasticbeanstalk:DescribeEvents" 操作。如果没有此权限，AWS Elastic Beanstalk 部署操作会挂起但不会失败或指示错误。

## 添加其他 AWS 服务的权限

您必须使用尚未包含在默认服务角色策略语句中的 AWS 服务的权限更新服务角色策略语句，然后才能在管道中使用它。

如果要用于管道的服务角色是在向 AWS CodePipeline 中添加对 AWS 服务的支持之前创建的，则这一点尤为重要。

下表显示了添加对其他 AWS 服务的支持的时间。

AWS 服务	AWS CodePipeline 支持日期
Amazon ECS	2017 年 12 月 12 日

AWS 服务	AWS CodePipeline 支持日期
AWS CodeCommit	2016 年 4 月 18 日
AWS OpsWorks	2016 年 6 月 2 日
AWS CloudFormation	2016 年 11 月 3 日
AWS CodeBuild	2016 年 12 月 1 日

要添加对其他受支持服务的权限，请按照下列步骤操作：

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
2. 在 IAM 控制台的导航窗格中，选择 Roles，然后从角色列表中选择您的 AWS-CodePipeline-Service 角色。
3. 在 Permissions 选项卡上的 Inline Policies 中，选择您的服务角色策略所在行中的 Edit Policy。

#### Note

您的服务角色的名称格式类似于 oneClick\_AWS-CodePipeline-1111222233334。

4. 在 Policy Document 框中添加所需权限。例如，对于 AWS CodeCommit 支持，将以下内容添加到策略语句中：

```
{
  "Action": [
    "codecommit:GetBranch",
    "codecommit:GetCommit",
    "codecommit:UploadArchive",
    "codecommit:GetUploadArchiveStatus",
    "codecommit:CancelUploadArchive"
  ],
  "Resource": "*",
  "Effect": "Allow"
},
```

对于 AWS OpsWorks 支持，将以下内容添加到策略语句中：

```
{
  "Action": [
    "opsworks:CreateDeployment",
    "opsworks:DescribeApps",
    "opsworks:DescribeCommands",
    "opsworks:DescribeDeployments",
    "opsworks:DescribeInstances",
    "opsworks:DescribeStacks",
    "opsworks:UpdateApp",
    "opsworks:UpdateStack"
  ],
  "Resource": "*",
  "Effect": "Allow"
},
```

对于 AWS CloudFormation 支持，将以下内容添加到策略语句中：

```
{
  "Action": [
    "cloudformation:CreateStack",
    "cloudformation>DeleteStack",
    "cloudformation:DescribeStacks",
```

```
        "cloudformation:UpdateStack",
        "cloudformation:CreateChangeSet",
        "cloudformation>DeleteChangeSet",
        "cloudformation:DescribeChangeSet",
        "cloudformation:ExecuteChangeSet",
        "cloudformation:SetStackPolicy",
        "cloudformation:ValidateTemplate",
        "iam:PassRole"
    ],
    "Resource": "*",
    "Effect": "Allow"
},
```

对于 AWS CodeBuild 支持，将以下内容添加到策略语句中：

```
{
  "Action": [
    "codebuild:BatchGetBuilds",
    "codebuild:StartBuild"
  ],
  "Resource": "*",
  "Effect": "Allow"
},
```

5. 对于 Amazon ECS，以下是使用 Amazon ECS 部署操作创建管道所需的最小权限。

```
{
  "Action": [
    "ecs:DescribeServices",
    "ecs:DescribeTaskDefinition",
    "ecs:DescribeTasks",
    "ecs:ListTasks",
    "ecs:RegisterTaskDefinition",
    "ecs:UpdateService"
  ],
  "Resource": "*",
  "Effect": "Allow"
},
```

#### Note

在您创建 IAM 策略时，请遵循授予最小权限这一标准的安全建议，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 (\*) 替换为特定的资源 ARN 或资源 ARN 中的通配符 (\*)。

6. 选择 `Validate Policy` 确保策略不包含错误。当策略正确无误时，选择 `Apply Policy`。

## 删除未使用的 AWS 服务的权限

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删除授予 Elastic Beanstalk 资源权限的部分。例如，您可以删除策略语句中的以下部分：

```
{
  "Action": [
    "elasticbeanstalk:*",
    "ec2:*",
    "elasticloadbalancing:*",
    "autoscaling:*",
```

```
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "cloudformation:*",
        "rds:*",
        "sqs:*",
        "ecs:*",
        "iam:PassRole"
    ],
    "Resource": "*",
    "Effect": "Allow"
},
```

类似地，如果您的所有管道均不包括 AWS CodeDeploy，您可以编辑策略语句以删除授予 AWS CodeDeploy 资源权限的部分。

```
{
  "Action": [
    "codedeploy:CreateDeployment",
    "codedeploy:GetApplicationRevision",
    "codedeploy:GetDeployment",
    "codedeploy:GetDeploymentConfig",
    "codedeploy:RegisterApplicationRevision"
  ],
  "Resource": "*",
  "Effect": "Allow"
},
```

有关 IAM 角色的更多信息，请参阅 [IAM 角色](#)。

## 客户托管策略示例

本节的用户策略示例介绍如何授予执行各个 AWS CodePipeline 操作的权限。当您使用 AWS CodePipeline API、AWS 软件开发工具包或 AWS CLI 时，这些策略将会发挥作用。当您使用控制台时，您需要授予特定于控制台的其他权限，[使用 AWS CodePipeline 控制台所需的权限 \(p. 180\)](#)中对此进行了讨论。

### Note

所有示例都使用 美国西部 ( 俄勒冈 ) 区域 (us-west-2) 和虚构的账户 ID。

### 示例

- [示例 1：授予获取管道状态的权限 \(p. 185\)](#)
- [示例 2：授予启用和禁用阶段之间的过渡的权限 \(p. 186\)](#)
- [示例 3：授予获取所有可用操作类型列表的权限 \(p. 186\)](#)
- [示例 4：授予批准或拒绝手动审批操作的权限 \(p. 186\)](#)
- [示例 5：授予轮询作业以查找自定义操作的权限 \(p. 187\)](#)
- [示例 6：附加或编辑 Jenkins 与 AWS CodePipeline 集成的策略 \(p. 187\)](#)
- [示例 7：配置对管道的跨账户访问 \(p. 188\)](#)
- [示例 8：在管道中使用与另一个账户相关联的 AWS 资源 \(p. 189\)](#)

## 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 AWS CodePipeline 集成的策略

如果配置管道以使用 Jenkins 进行生成或测试，请为该集成创建单独的 IAM 用户，并附加具有 Jenkins 和 AWS CodePipeline 之间的集成所需的最低权限的 IAM 策略。此策略与 AWSCodePipelineCustomActionAccess 托管策略相同。以下示例显示了要附加到将用于 Jenkins 集成的 IAM 用户的策略：

```

{
  "Statement": [
    {
      "Action": [
        "codepipeline:AcknowledgeJob",
        "codepipeline:GetJobDetails",
        "codepipeline:PollForJobs",
        "codepipeline:PutJobFailureResult",
        "codepipeline:PutJobSuccessResult"
      ],
      "Effect": "Allow",
      "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。推荐的方法是在创建管道的账户中创建一个角色，以允许来自其他 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 80398EXAMPLE 账户中的策略，该策略允许用户在 AWS CodePipeline 控制台中查看但不能更改名为 *MyFirstPipeline* 的管道。该策略基于 `AWSCodePipelineReadOnlyAccess` 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于某特定管道，则绝对应该考虑使用 AWS CodePipeline 创建和维护的托管策略之一。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: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"
      ],
      "Effect": "Allow",
      "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，您必须添加将担任此角色的 AWS 账户。以下示例显示允许来自 *111111111111* AWS 账户的用户担任在 80398EXAMPLE 账户中定义的角色策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111111111111:root"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

以下示例显示在 `111111111111` AWS 账户中创建的策略，该策略允许用户担任 80398EXAMPLE 账户中名为 `CrossAccountPipelineViewers` 的角色：

```
{
  "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

您可以配置策略以允许用户创建使用另一个 AWS 账户中的资源的管道。这需要在将创建管道的账户 (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 AWS Key Management Service 中创建一个客户管理的密钥，以用于跨账户访问。有关更多信息和分步示例，请参阅在 [AWS CodePipeline 中创建使用另一个 AWS 账户的资源的管道](#) (p. 102) 和 [安全配置](#) (p. 195)。

以下示例显示了 AccountA 为用于存储管道项目的 Amazon S3 存储桶配置的策略，该策略授予对 AccountB 的访问权限。在该示例中，*AccountB* 的 ARN 为 `012ID_ACCOUNT_B`。Amazon 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账户和 Amazon 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
        }
      }
    },
    {
      "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。此策略必须应用于 AWS CodePipeline 的服务角色 (**AWS-CodePipeline-Service**)。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 AWS CodeDeploy 的 **Amazon EC2 实例角色** 的策略。此策略授予对 AccountA 使用的 Amazon S3 存储桶的访问权限以存储管道项目 (**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"
      ]
    }
  ]
}

```

以下示例显示了一个 AWS KMS 策略，其中 **arn:aws:kms:us-east-1:012ID\_ACCOUNT\_A:key/22222222-3333333-4444-556677EXAMPLE** 是在 **AccountA** 中创建并配置为允许 **AccountB** 使用它的客户管理密钥的 ARN：

```
{
  "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 中的管道所需的 AWS 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 Amazon S3 存储桶，以便下载输入项目和上传输出项目：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject*",
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": [
        "arn:aws:s3:::codepipeline-us-east-2-1234567890/*"
      ]
    }
  ]
}
```

有关在创建必需的策略、角色和 AWS Key Management Service 客户管理密钥之后，如何编辑管道以便对资源进行跨账户访问的更多信息，请参阅[步骤 2：编辑管道](#) (p. 108)。

## AWS CodePipeline 权限参考

在设置 [使用 IAM 策略控制访问](#) (p. 172) 和编写您可挂载到 IAM 身份的权限策略 (基于身份的策略) 时，可以使用下表作为参考。此表列出每个 AWS CodePipeline API 操作及您可授予执行该操作的权限的对应操作。可在策略的 `Action` 字段中指定操作，在策略的 `Resource` 字段中指定通配符 (\*) 作为资源值。

您可以在 AWS CodePipeline 策略中使用 AWS 范围的条件键来表达条件。有关 AWS 范围内的键的完整列表，请参阅 IAM 用户指南 中的 [可用键](#)。

### Note

要指定操作，请在 API 操作名称之前使用 `codepipeline:` 前缀。例  
如：`codepipeline:GetPipeline`、`codepipeline:CreatePipeline` 或 `codepipeline:*`  
(针对所有 AWS CodePipeline 操作)。

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项操作。例如，您可以指定名称以单词“Get”开头的所有操作，如下所示：

```
"Action": "codepipeline:Get*"
```

要指定所有 AWS CodePipeline API 操作，请使用 \* 通配符，如下所示：

```
"Action": "codepipeline:*
```

下面列出了 IAM 策略中可指定用于 AWS CodePipeline 的操作。

AWS CodePipeline API 操作和必需的操作权限

### [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。

### [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。

### [CreateCustomActionType](#)

操作：`codepipeline:CreateCustomActionType`

创建可用于与 AWS 账户相关联的所有管道的新自定义操作所必需的。仅用于自定义操作。

### [CreatePipeline](#)

操作：`codepipeline:CreatePipeline`

创建管道所必需的。

### [DeleteCustomActionType](#)

操作：`codepipeline>DeleteCustomActionType`

将自定义操作标记为已删除所必需的。将自定义操作标记为删除后，该操作的 PollForJobs 将失败。仅用于自定义操作。

#### DeletePipeline

操作 : `codepipeline:DeletePipeline`

删除管道所必需的。

#### DeleteWebhook

操作 : `codepipeline:DeleteWebhook`

删除 Webhook 所必需的。

#### DeregisterWebhookWithThirdParty

操作 : `codepipeline:DeregisterWebhookWithThirdParty`

在删除 Webhook 之前，需要使用它来删除通过 CodePipeline 创建的 Webhook 与用于检测事件的外部工具之间的连接。目前仅支持目标是 GitHub 的操作类型的 Webhook。

#### 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。

#### 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。

#### 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。

#### 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。

#### 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。

#### 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。

#### 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。



### ListActionTypes

操作 : `codepipeline:ListActionTypes`

生成与您的账户相关联的所有 AWS CodePipeline 操作类型的摘要所必需的。

### ListPipelineExecutions

操作 : `codepipeline:ListPipelineExecutions`

生成最新管道执行的摘要所必需的。

### 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。

### ListWebhooks

操作 : `codepipeline:ListWebhooks`

列出与该区域的账户中的所有 Webhook 时所必需的。

### PollForJobs

操作 : `codepipeline:PollForJobs`

检索有关 AWS CodePipeline 要对其采取行动的任何作业的信息所必需的。

### 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。

### PutActionRevision

操作 : `codepipeline:PutActionRevision`

向 AWS CodePipeline 报告有关源的新修订的信息所必需的。

### PutApprovalResult

操作 : `codepipeline:PutApprovalResult`

向 AWS CodePipeline 报告对手动审批请求的响应所必需的。有效的响应包括“Approved”和“Rejected”。

### 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。

### 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。

### 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。

#### 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。

#### PutWebhook

操作 : `codepipeline:PutWebhook`

创建 Webhook 所必需的。

#### RegisterWebhookWithThirdParty

操作 : `codepipeline:RegisterWebhookWithThirdParty`

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

#### 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。

#### StartPipelineExecution

操作 : `codepipeline:StartPipelineExecution`

启动指定的管道所必需的。具体来说就是，它开始处理指定为管道一部分的源位置的最新提交内容。

#### 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。

## 安全配置

此部分介绍了基于以下最佳实践的安全配置：

- S3 项目服务器端加密 (SSE)
- GitHub 个人访问令牌
- Parameter Store 中的配置参数跟踪

#### 主题

- 为适用于 AWS CodePipeline 的 Amazon S3 中存储的项目配置服务器端加密 (p. 195)
- 配置 GitHub 身份验证 (p. 197)
- 使用 Parameter Store 跟踪数据库密码或第三方 API 密钥 (p. 200)

## 为适用于 AWS CodePipeline 的 Amazon S3 中存储的项目配置服务器端加密

可以使用两种方法为 Amazon S3 项目配置服务器端加密：

- 在使用创建管道向导创建管道时，AWS CodePipeline 创建 Amazon S3 项目存储桶和默认 AWS 托管 SSE-KMS 加密密钥。主密钥是与数据一起加密的，并由 AWS 进行管理。
- 您可以创建和管理自己的客户托管 SSE-KMS 密钥。

如果使用默认的 Amazon S3 密钥，则无法更改或删除此 AWS 管理的密钥。如果您在 AWS KMS 中使用客户管理的密钥来加密或解密 Amazon S3 存储桶中的项目，则可以根据需要更改或轮换该密钥。

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-2-890506445442/*",
      "Condition": {
        "StringNotEquals": {
          "s3:x-amz-server-side-encryption": "aws:kms"
        }
      }
    },
    {
      "Sid": "DenyInsecureConnections",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": "arn:aws:s3:::codepipeline-us-west-2-890506445442/*",
      "Condition": {
        "Bool": {
          "aws:SecureTransport": "false"
        }
      }
    }
  ]
}
```

有关服务器端加密和 AWS KMS 的更多信息，请参阅[使用服务器端加密保护数据](http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingKMSEncryption.html)和<http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingKMSEncryption.html>。

有关 AWS KMS 的更多信息，请参阅 [AWS Key Management Service 开发人员指南](http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingKMSEncryption.html)、和<http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingKMSEncryption.html>。

#### 主题

- [查看默认 Amazon S3 SSE-KMS 加密密钥 \(p. 196\)](#)
- [在使用 AWS CloudFormation 或 CLI 时为 S3 存储桶配置服务器端加密 \(p. 197\)](#)

## 查看默认 Amazon S3 SSE-KMS 加密密钥

当您使用 Create Pipeline 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 Amazon S3 存储桶。存储桶用于存储管道项目。当管道运行时，项目将被放入 Amazon S3 存储桶并从中检索。默认情况下，AWS CodePipeline 使用具有 AWS KMS 托管密钥 (SSE-KMS) 的服务器端加密，该加密方法使用 Amazon S3 的默认密钥 (`aws/s3` 密钥)。该密钥创建并存储在您的 AWS 账户中。从 Amazon S3 存储桶中检索项目时，AWS CodePipeline 使用相同的 SSE-KMS 过程解密项目。

要查看有关默认 AWS KMS 密钥的信息，请执行以下操作：

1. 登录 AWS 管理控制台 并通过以下网址打开 IAM 控制台 <https://console.aws.amazon.com/iam/>。
2. 在服务导航窗格中，选择 Encryption Keys。(如果出现欢迎页面，请选择 Get Started Now。)
3. 在 Filter 中，选择您的管道的区域。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)。

有关 AWS CodePipeline 可用的区域和终端节点的更多信息，请参阅 [区域和终端节点](#)。

4. 在加密密钥列表中，选择包含用于您的管道的别名的密钥 (默认情况下为 aws/s3)。有关该密钥的基本信息将会显示。

## 在使用 AWS CloudFormation 或 CLI 时为 S3 存储桶配置服务器端加密

在使用 AWS CloudFormation 或 CLI 创建管道时，您必须手动配置服务器端加密。可以使用上面的示例存储桶策略，然后创建您自己的客户托管 SSE-KMS 加密密钥。您也可以选择使用自己的密钥，而不是通常的默认 Amazon S3 密钥。这样做的一些原因包括：

- 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。这需要客户管理的密钥。有关更多信息，请参阅 [在 AWS CodePipeline 中创建使用另一个 AWS 账户的资源的管道 \(p. 102\)](#)。

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 Key Management Service (AWS KMS) 客户主密钥 (CMK) 创建新的加密材料，您可以创建新的 CMK，然后更改您的应用程序或别名以使用新的 CMK。或者，您可以为现有 CMK 启用自动密钥轮换。

要轮换您的 SSE-KMS 客户主密钥，请参阅 [轮换客户主密钥](#)。

## 配置 GitHub 身份验证

AWS CodePipeline 使用 GitHub OAuth 令牌和个人访问令牌访问您的 GitHub 存储库并检索最新的更改。可以使用两种方法在 GitHub 中配置身份验证：

- 在使用控制台创建或更新管道时，AWS 创建一个默认 AWS 托管 OAuth 令牌。
- 您可以创建并管理自己的客户生成的个人访问令牌。在使用 CLI、SDK 或 AWS CloudFormation 创建或更新管道时，您需要使用个人访问令牌。

### 主题

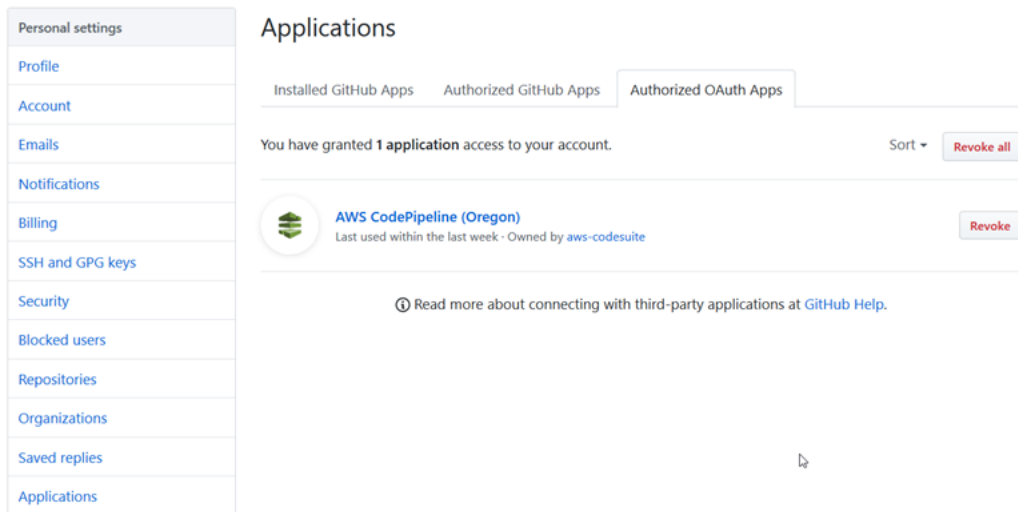
- [查看授权的 OAuth 应用程序 \(p. 197\)](#)
- [使用 GitHub 和 AWS CodePipeline CLI 创建并定期轮换您的 GitHub 个人访问令牌 \(p. 198\)](#)

## 查看授权的 OAuth 应用程序

AWS CodePipeline 使用 OAuth 令牌与 GitHub 集成在一起。GitHub 跟踪 AWS CodePipeline 的 OAuth 令牌的权限。

可以使用以下步骤在 GitHub 中检查授权的集成。

1. 在 GitHub 中，从您的资料照片上的下拉列表选项中选择设置。
2. 选择应用程序，然后选择授权的 OAuth 应用程序。
3. 查看授权的应用程序。



## 使用 GitHub 和 AWS CodePipeline CLI 创建并定期轮换您的 GitHub 个人访问令牌

在脚本中使用令牌而不是密码的优点是，可以吊销或轮换令牌。您也可以为个人访问令牌授予特定的权限。应安全地存储令牌，并定期轮换或重新生成令牌。RFC-6819 (OAuth 2.0 威胁模型和安全注意事项) 第 5.1.5.3 条建议进行令牌轮换。

有关更多信息，请参阅 GitHub 网站上的[为命令行创建个人访问令牌](#)。

在重新生成新的个人访问令牌后，您可以使用 CLI、API 或使用 AWS CloudFormation 并调用 UpdatePipeline 以进行轮换。

### Note

如果使用相同的个人访问令牌，您可能需要更新其他应用程序。作为安全最佳实践，请不要在多个应用程序之间共享单个令牌。请为每个应用程序创建新的个人访问令牌。

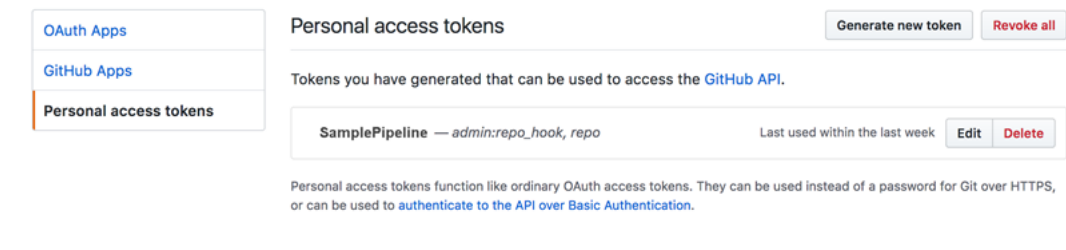
可以使用以下步骤轮换您的 GitHub 个人访问令牌，然后使用新令牌更新管道结构。

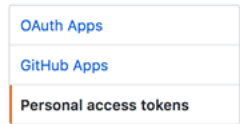
### Note

在轮换您的个人访问令牌后，请务必更新包含旧令牌信息的任何 CLI 脚本或 AWS CloudFormation 模板。

1. 在 GitHub 中，从您的资料照片上的下拉列表选项中选择设置。
2. 选择开发人员设置，然后选择个人访问令牌。
3. 在您的 GitHub 个人访问令牌旁边，选择编辑。

[Settings](#) / [Developer settings](#)



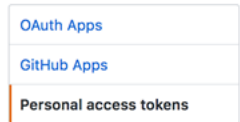


**Regenerate token**

## SamplePipeline

### Select scopes

<input checked="" type="checkbox"/> <b>repo</b>	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<hr/>	
<input type="checkbox"/> <b>admin:org</b>	Full control of orgs and teams
<input type="checkbox"/> write:org	Read and write org and team membership
<input type="checkbox"/> read:org	Read org and team membership
<hr/>	
<input type="checkbox"/> <b>admin:public_key</b>	Full control of user public keys
<input type="checkbox"/> write:public_key	Write user public keys
<input type="checkbox"/> read:public_key	Read user public keys



Generate new token

Make sure to copy your new personal access token now. You won't be able to see it again!

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

```
"Branch": "master",  
"OAuthToken": "Replace the **** with your personal access token"  
},
```

- 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` 行来修改 JSON 文件中的结构，否则 `update-pipeline` 命令将无法使用管道结构。从 JSON 文件中的管道结构中删除该部分 ( `"metadata": {}` 行以及其中的“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"  
}
```

- 保存文件，然后带 `--cli-input-json` 参数运行 `update-pipeline`，以指定您刚刚编辑的 JSON 文件。例如，要更新名为 `MyFirstPipeline` 的管道，您应该键入类似下面的内容：

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
```

- 在更新完管道时，删除 JSON 文件。

有关更多信息，请参阅 [管道错误：我收到一个管道错误，其中包含以下消息：“PermissionError: Could not access the GitHub repository”](#) (p. 167)。

## 使用 Parameter Store 跟踪数据库密码或第三方 API 密钥

您可以使用 Parameter Store 跟踪和更新配置密钥，如数据库密码。该过程介绍了如何使用 Parameter Store 手动创建密钥参数。您还可以创建自动化脚本以使用 Parameter Store 安全地自动设置和管理您的密码和密钥。请参阅此 [博客文章](#) 中的 AWS CodeDeploy 生成规范自动化示例。

在 Parameter Store 中手动创建参数

- 登录到您的 AWS 账户，然后转到 Amazon EC2 控制台。
- 在 Systems Manager 共享资源部分中，单击 Parameter Store。
- 单击立即开始使用或创建参数，然后输入以下信息：
  - 在名称字段中，键入参数的名称。
  - 在类型下面，选择安全字符串。这会使用您的默认 AWS KMS 密钥加密敏感数据。
  - 在值字段中粘贴该参数。
- 单击创建参数，将转到 Parameter Store 控制台，您可以在其中看到新创建的参数。

# AWS CodePipeline 命令行参考

在使用 AWS CodePipeline 命令时可利用此参考来补充 [AWS CLI 用户指南](#) 和 [AWS CLI 参考](#) 中正式记载的信息。

使用 AWS CLI 之前，请确保先完成 [AWS CodePipeline 入门 \(p. 8\)](#) 中的先决条件。

要查看所有可用 AWS CodePipeline 命令的列表，请运行以下命令：

```
aws codepipeline help
```

要查看有关某个特定 AWS CodePipeline 命令的信息，请运行以下命令，其中 *command-name* 是下面列出的其中一个命令的名称 (例如，create-pipeline)：

```
aws codepipeline command-name help
```

要开始学习如何使用 AWS CLI 的 AWS CodePipeline 扩展中的命令，请转到以下一节或多节：

- [创建自定义操作 \(CLI\) \(p. 112\)](#)
- [创建管道 \(CLI\) \(p. 88\)](#)
- [删除管道 \(CLI\) \(p. 101\)](#)
- [禁用或启用过渡 \(CLI\) \(p. 151\)](#)
- [查看管道详细信息和历史记录 \(CLI\) \(p. 98\)](#)
- [重试失败的操作 \(CLI\) \(p. 137\)](#)
- [手动启动管道 \(CLI\) \(p. 80\)](#)>
- [编辑管道 \(AWS CLI\) \(p. 93\)](#)

您还可以在 [AWS CodePipeline 教程 \(p. 22\)](#) 中查看有关如何使用大多数这些命令的示例。



# AWS CodePipeline 管道结构参考

默认情况下，您在 AWS CodePipeline 中成功创建的任何管道都将具有有效结构。但是，如果手动创建或编辑 JSON 文件以创建管道或从 AWS CLI 更新管道，则可能会无意中创建无效的结构。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。另请参阅[AWS CodePipeline 中的限制 \(p. 208\)](#)中记载的约束，这些约束适用于所有管道。

## 主题

- [AWS CodePipeline 中的管道和阶段结构要求 \(p. 202\)](#)
- [AWS CodePipeline 中的操作结构要求 \(p. 203\)](#)

## AWS CodePipeline 中的管道和阶段结构要求

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/AWS-CodePipeline-Service",
  "stages": [
    {
      "name": "SourceStageName",
      "actions": [
        ... See AWS CodePipeline ##### ...
      ]
    },
    {
      "name": "NextStageName",
      "actions": [
        ... See AWS 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，并且只能包含源操作。
- 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必须是唯一的。
- 阶段名称不能在 AWS CodePipeline 控制台内编辑。如果您使用 AWS CLI 编辑阶段名称，并且该阶段包含具有一个或多个秘密参数 (例如 OAuth 令牌) 的操作，则这些秘密参数的值将不会保留下来。您必须手动键入参数的值 (在 AWS CLI 返回的 JSON 中由四个星号遮替)，并将其包含在 JSON 结构中。

- 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。

## AWS CodePipeline 中的操作结构要求

操作具有以下高级结构：

```
[
  {
    "inputArtifacts": [
      An input artifact structure, if supported for the action category
    ],
    "name": "ActionName",
    "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
  }
]
```

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。
- 操作的输入项目必须与前一操作中声明的输出项目完全相符。例如，如果前一操作包含以下声明：

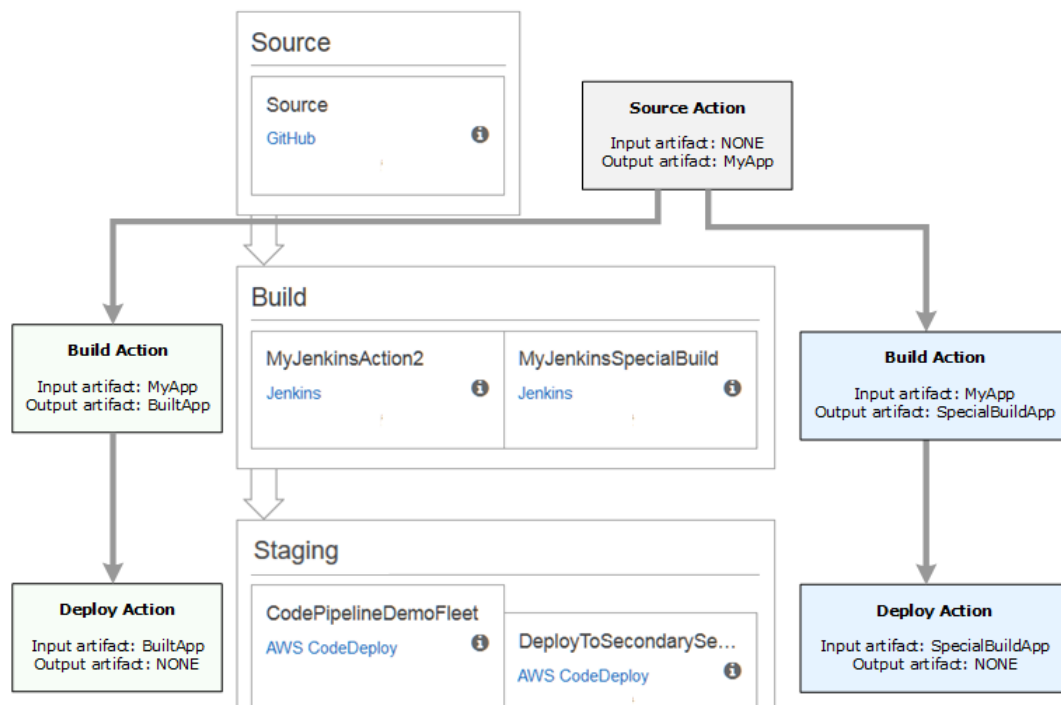
```
"outputArtifacts": [
  {
    "MyApp"
  }
],
```

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项目包，这些包又由不同的后续操作使用。

下图提供了管道操作中的输入和输出项目的示例：



- 输出项目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。
- 如果某个操作包含具有秘密值的参数 (例如 GitHub 源操作的 OAuth 令牌)，那么该参数的值在 JSON 中将被四个连续星号 (\*\*\*\*) 遮替。系统将会存储实际值，只要您不编辑该值，或者更改操作名称或运行该操作的阶段的名称，那么在使用 AWS CLI 或 AWS CodePipeline API 编辑 JSON 时就不必提供该值。但是，如果您更改操作的名称或运行操作的阶段的名称，则任何秘密参数的值都将丢失。您必须在 JSON 中手动键入任何秘密参数的值，否则下一次管道运行时，该操作将失败。
- 对于当前受支持的所有操作类型，唯一有效的版本字符串为 "1"。
- 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是 "AWS"、"ThirdParty" 或 "Custom"。有关更多信息，请参阅 [AWS 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。

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。

- 根据操作类型，您可以具有以下数量的输入和输出项目：

#### 项目的操作类型约束

所有者	操作类型	提供商	有效的输入项目数	有效的输出项目数
AWS	Source	Amazon S3	0	1
AWS	Source	AWS CodeCommit	0	1
ThirdParty	Source	GitHub	0	1
AWS	Build	AWS CodeBuild	1	0-1
AWS	Test	AWS CodeBuild	1	0-1
AWS	Approval	Manual	0	0
AWS	Deploy	AWS CloudFormation	0-10	0-1
AWS	Deploy	AWS CodeDeploy	1	0
AWS	Deploy	AWS Elastic Beanstalk	1	0
AWS	Deploy	AWS OpsWorks Stacks	1	0
AWS	Deploy	Amazon ECS	1	0
AWS	Invoke	AWS Lambda	0 到 5	0 到 5
Custom	Build	Jenkins	0 到 5	0 到 5
Custom	Test	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

- 这些是 AWS CodePipeline 的有效 `actionTypeId` 类别：

- Source
- Build
- Approval
- Deploy
- Test
- Invoke

下面提供了一些提供程序类型和配置选项。

- 操作类别的有效提供商类型因类别而异。例如，对于源操作类型，有效的提供程序类型为 S3、GitHub 或 CodeCommit。此示例显示了 S3 源操作提供程序的结构：

```
"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商类型。下表列出了每个有效提供商类型所需的操作配置元素：

提供商类型的操作配置属性

提供商名称	操作类型中的提供商名称	配置属性	必需属性？
Amazon S3	S3	S3Bucket	必需
		PollForSourceChanges <sup>1</sup>	可选
		S3ObjectKey	必需
AWS CodeCommit	CodeCommit	PollForSourceChanges <sup>1</sup>	可选
		RepositoryName	必需
		BranchName	必需
GitHub	GitHub	Owner	必需
		Repo	必需
		PollForSourceChanges <sup>1</sup>	可选
		Branch	必需
		OAuthToken	必需
AWS CloudFormation <sup>2</sup>	CloudFormation	ActionMode	必需
		StackName	必需
AWS CodeBuild	CodeBuild	ProjectName	必需
AWS CodeDeploy	CodeDeploy	ApplicationName	必需
		DeploymentGroupName	必需
AWS Elastic Beanstalk	ElasticBeanstalk	ApplicationName	必需
		EnvironmentName	必需
AWS Lambda	Lambda	FunctionName	必需
		UserParameters	可选
AWS OpsWorks Stacks	OpsWorks	Stack	必需
		Layer	可选
		App	必需
Amazon ECS	ECS	ClusterName	必需
		ServiceName	必需
		FileName	可选
Jenkins	在 AWS CodePipeline Plugin for Jenkins 中提供的操作名称 (例如, <i>MyJenkinsProviderName</i> )	ProjectName	必需

提供商名称	操作类型中的提供商名称	配置属性	必需属性？
<p><sup>1</sup></p> <p>对于在以下日期后在控制台中创建或更新的管道或源操作，此参数默认为 false：</p> <ul style="list-style-type: none"><li>• 2017 年 10 月 11 日之后在控制台中创建或更新的 AWS CodeCommit 管道。</li><li>• 2018 年 3 月 22 日之后在控制台中创建或更新的 Amazon S3 管道。</li><li>• 2018 年 5 月 1 日之后在控制台中创建或更新的 GitHub 管道。</li></ul> <p>对于使用 CLI 创建/更新的管道，此参数默认设置为 true，但这不是推荐的配置。而是应更新您的管道以使用建议的更改检测方法，然后将此参数设置为 false。有关更多信息，请参阅<a href="#">用于自动启动管道的变更检测方法 (p. 63)</a>。如果从未明确设置，则不会显示该参数。</p> <p><sup>2</sup> AWS CloudFormation 的第三和第四个必需属性取决于所选的 <code>ActionMode</code> 属性。有关更多信息，请参阅 AWS CloudFormation 用户指南 中的<a href="#">利用 AWS CodePipeline 进行持续交付</a>。</p>			

以下示例显示了使用 Amazon S3 的源操作的有效操作配置：

```
"configuration": {
  "S3Bucket": "awscodepipeline-demobucket-example-date",
  "S3ObjectKey": "CodePipelineDemoApplication.zip",
  "PollForSourceChanges": "false"
}
```

以下示例显示针对使用 GitHub 的源操作返回的操作配置：

```
"configuration": {
  "Owner": "MyGitHubAccountName",
  "Repo": "MyGitHubRepositoryName",
  "PollForSourceChanges": "false",
  "Branch": "master",
  "OAuthToken": "****"
},
```

以下示例显示使用 Amazon ECS 的部署操作的有效配置：

```
"configuration": {
  "ClusterName": "my-ecs-cluster",
  "ServiceName": "sample-app-service",
  "FileName": "imagedefinitions.json",
}
```

# AWS CodePipeline 中的限制

下表列出了 AWS CodePipeline 中的当前限制。[AWS CodePipeline 管道结构参考 \(p. 202\)](#)中讨论了结构要求。有关可更改的限制的信息，请参阅 [AWS 服务限制](#)。

AWS 账户中每个区域的最大管道数量	美国东部 ( 弗吉尼亚北部 ) (us-east-1) : 40 美国西部 ( 俄勒冈 ) (us-west-2) : 60 欧洲 ( 爱尔兰 ) (eu-west-1) : 60 所有其他支持的区域 : 20
管道中的阶段数量	最小为 2，最大为 10
阶段中的操作数量	最小为 1，最大为 20
AWS 账户中每个区域自定义操作的最大数量	最大 50
一个阶段中并行操作的最大数量	最大 10
一个阶段中顺序操作的最大数量	最大 10
在一个 AWS 账户中每个区域的最大 webhook 数量	最大 300
操作超时前的时长	审批操作 : 7 天 AWS CloudFormation 部署操作 : 3 天 AWS CodeBuild 生成操作和测试操作 : 8 小时 AWS CodeDeploy 部署操作 : 7 天 AWS Lambda 调用操作 : 1 小时 Note AWS CodePipeline 等待一小时，以确定 Lambda 函数是否失败。如果操作超时，AWS CodePipeline 将调用操作状态设置为失败。AWS Lambda 具有不同的限制。AWS Lambda 函数超时的默认值为 3 秒，但您可以将此限制设置为最多 300 秒。 所有其他操作 : 1 小时
阶段执行超时前的天数	40 Note 您添加到阶段中的连续操作的超时时间总和不能超过 40 天。
可查看管道执行历史信息的以前月份的最大数量	12
可创建区域的管道	美国东部 ( 俄亥俄州 ) (us-east-2) 美国东部 ( 弗吉尼亚北部 ) (us-east-1)

	<p>美国西部 ( 加利福尼亚北部 ) (us-west-1)</p> <p>美国西部 ( 俄勒冈 ) (us-west-2)</p> <p>加拿大 (中部) (ca-central-1)</p> <p>欧洲 ( 爱尔兰 ) (eu-west-1)</p> <p>欧洲 (伦敦)(eu-west-2)</p> <p>欧洲 (巴黎)(eu-west-3)</p> <p>欧洲 ( 法兰克福 ) (eu-central-1)</p> <p>亚太地区 ( 孟买 ) (ap-south-1)</p> <p>亚太区域 ( 东京 ) (ap-northeast-1)</p> <p>亚太区域 ( 首尔 ) (ap-northeast-2)</p> <p>亚太区域 ( 新加坡 ) (ap-southeast-1)</p> <p>亚太区域 ( 悉尼 ) (ap-southeast-2)</p> <p>南美洲 ( 圣保罗 ) (sa-east-1)</p>
源阶段中项目的最大大小	<p>存储在 Amazon S3 存储桶中的项目 : 2 GB</p> <p>存储在 AWS CodeCommit 或 GitHub 存储库中的项目 : 1 GB</p> <p>例外 : 如果您使用 Amazon EBS 部署应用程序, 则最大项目大小始终为 512 MB。</p>
AWS CloudFormation 操作的输入项目的最大大小	<p>如果您使用 AWS CloudFormation 部署 Lambda 函数, 则 Lambda 代码存档文件大小不应超过 256 MB。请参阅 <a href="#">AWS Lambda 限制</a>。</p>
名称的唯一性	<p>在单个 AWS 账户中, 您在一个区域中创建的每个管道都必须具有唯一名称。名称可重用于不同区域的管道。</p> <p>管道中的阶段名称必须是唯一的。</p> <p>阶段中的操作名称必须是唯一的。</p>
管道名称中允许的字符	<p>管道名称不能超过 100 个字符。允许的字符包括 :</p> <p>字母字符 a 至 z (含这两个字符)。</p> <p>字母字符 A 至 Z (含这两个字符)。</p> <p>数字字符 0 至 9 (含这两个字符)。</p> <p>特殊字符 . (句点)、@ (at 符号)、- (减号) 以及 _ (下划线)。</p> <p>不允许任何其他字符, 例如空格。</p>



阶段名称中允许的字符	<p>阶段名称不能超过 100 个字符。允许的字符包括：</p> <p>字母字符 a 至 z (含这两个字符)。</p> <p>字母字符 A 至 Z (含这两个字符)。</p> <p>数字字符 0 至 9 (含这两个字符)。</p> <p>特殊字符 . (句点)、@ (at 符号)、- (减号) 以及 _ (下划线)。</p> <p>不允许任何其他字符，例如空格。</p>
操作名称中允许的字符	<p>操作名称不能超过 100 个字符。允许的字符包括：</p> <p>字母字符 a 至 z (含这两个字符)。</p> <p>字母字符 A 至 Z (含这两个字符)。</p> <p>数字字符 0 至 9 (含这两个字符)。</p> <p>特殊字符 . (句点)、@ (at 符号)、- (减号) 以及 _ (下划线)。</p> <p>不允许任何其他字符，例如空格。</p>
操作类型中允许的字符	<p>操作类型名称不能超过 25 个字符。允许的字符包括：</p> <p>字母字符 a 至 z (含这两个字符)。</p> <p>字母字符 A 至 Z (含这两个字符)。</p> <p>数字字符 0 至 9 (含这两个字符)。</p> <p>特殊字符 . (句点)、@ (at 符号)、- (减号) 以及 _ (下划线)。</p> <p>不允许任何其他字符，例如空格。</p>
合作伙伴操作名称中允许的字符	<p>合作伙伴操作名称必须遵循与 AWS CodePipeline 中其他操作名称相同的命名约定和限制。具体来说，它们不能超过 100 个字符，并允许使用以下字符：</p> <p>字母字符 a 至 z (含这两个字符)。</p> <p>字母字符 A 至 Z (含这两个字符)。</p> <p>数字字符 0 至 9 (含这两个字符)。</p> <p>特殊字符 . (句点)、@ (at 符号)、- (减号) 以及 _ (下划线)。</p> <p>不允许任何其他字符，例如空格。</p>

可存储在 <code>ParameterOverrides</code> 属性中的 JSON 对象的最大大小	对于将 AWS CloudFormation 作为提供程序的 AWS CodePipeline 部署操作， <code>ParameterOverrides</code> 属性用于存储一个指定 AWS CloudFormation 模板配置文件值的 JSON 对象。 <code>ParameterOverrides</code> 属性中可存储的 JSON 对象的大小上限是 1 KB。
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	100 KB

# AWS CodePipeline 用户指南文档历史记录

下表描述了 AWS CodePipeline 用户指南的每次发布中所做的重要更改。如需对此文档更新的通知，您可以订阅 RSS 源。

update-history-change	update-history-description	update-history-date
现在可通过 <a href="#">RSS</a> 获得 <a href="#">AWS CodePipeline 用户指南更新通知</a> (p. 212)	HTML 版本的 AWS CodePipeline 用户指南现在支持更新的 RSS 源，此类更新记载在“文档更新历史记录”页面上。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。	June 30, 2018

下表描述了 2018 年 6 月 30 日之前 AWS CodePipeline 用户指南的每次发布中所做的重要更改。

- API 版本：2015-07-09
- 文档最新更新时间：2018 年 6 月 30 日

更改	描述	更改日期
使用 Webhook 检测 GitHub 管道中的源更改	在控制台中创建或编辑管道时，AWS CodePipeline 现在创建 Webhook 以检测对 GitHub 源存储库的更改，然后启动管道。有关迁移管道的信息，请参阅 <a href="#">将 GitHub 管道配置为使用 Webhook 检测更改</a> 。有关更多信息，请参阅 <a href="#">在 AWS CodePipeline 中启动管道执行</a> 。	2018 年 5 月 1 日
对主题进行了更新	在控制台中创建或编辑管道时，AWS CodePipeline 现在会创建 Amazon CloudWatch Events 规则和 AWS CloudTrail 跟踪，后者将检测对 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 <a href="#">将您的 Amazon S3 管道配置为使用 Amazon CloudWatch Events 检测更改</a> (p. 74)。  更新了 <a href="#">教程：创建一个简单的管道 (Amazon S3 存储桶)</a> (p. 22)以说明在选择 Amazon S3 源时如何创建 Amazon CloudWatch Events 规则和跟踪。还更新了 <a href="#">在 AWS CodePipeline 中创建管道</a> (p. 82)和 <a href="#">在 AWS CodePipeline 中编辑管道</a> (p. 90)。  有关更多信息，请参阅 <a href="#">在 AWS CodePipeline 中启动管道执行</a> (p. 62)。	2018 年 3 月 22 日
对主题进行了更新	AWS CodePipeline 目前已在欧洲 (巴黎) 中可用。更新了 <a href="#">AWS CodePipeline 中的限制</a> (p. 208)主题。	2018 年 2 月 21 日

更改	描述	更改日期
对主题进行了更新	<p>您现在可以使用 AWS CodePipeline 和 Amazon ECS 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 以作为部署提供程序。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。</p> <p>更新了<a href="#">AWS CodePipeline 产品和服务集成 (p. 10)</a>、在 <a href="#">AWS CodePipeline 中创建管道 (p. 82)</a>和<a href="#">AWS CodePipeline 管道结构参考 (p. 202)</a>主题以反映为 Amazon ECS 提供的这种支持。</p>	2017 年 12 月 12 日
对主题进行了更新	<p>在控制台中创建或编辑管道时，AWS CodePipeline 现在创建 Amazon CloudWatch Events 规则以检测对 AWS CodeCommit 存储库的更改，然后自动启动管道。有关迁移现有管道的信息，请参阅<a href="#">将 AWS CodeCommit 管道配置为使用 Amazon CloudWatch Events 检测更改 (p. 69)</a>。</p> <p>更新了教程：<a href="#">创建一个简单的管道 (AWS CodeCommit 存储库) (p. 36)</a>以说明在选择 AWS CodeCommit 存储库和分支时如何创建 Amazon CloudWatch Events 规则和角色。还更新了在 <a href="#">AWS CodePipeline 中创建管道 (p. 82)</a>和在 <a href="#">AWS CodePipeline 中编辑管道 (p. 90)</a>。</p> <p>有关更多信息，请参阅 <a href="#">在 AWS CodePipeline 中启动管道执行 (p. 62)</a>。</p>	2017 年 10 月 11 日
新增和更新的主体	<p>AWS CodePipeline 现在通过 Amazon CloudWatch Events 和 Amazon Simple Notification Service (Amazon SNS) 为管道状态更改通知提供内置支持。增加了新教程<a href="#">教程：设置 CloudWatch Events 规则以接收管道状态更改的电子邮件通知 (p. 54)</a>。有关更多信息，请参阅 <a href="#">使用 Amazon CloudWatch Events 检测和响应管道状态更改 (p. 153)</a>。</p>	2017 年 9 月 8 日
新增和更新的主体	<p>您现在可以将 AWS CodePipeline 添加为 Amazon CloudWatch Events 操作目标。可以设置 Amazon CloudWatch Events 规则以检测源更改，以便在发生这些更改时立即启动管道，也可以将其设置为运行计划的管道执行。已添加 PollForSourceChanges 源操作配置选项的信息。有关更多信息，请参阅 <a href="#">在 AWS CodePipeline 中启动管道执行 (p. 62)</a>。</p>	2017 年 9 月 5 日
新区域	<p>目前，在 亚太区域（首尔）和 亚太地区（孟买）中提供了 AWS CodePipeline。更新了<a href="#">AWS CodePipeline 中的限制 (p. 208)</a>主题及<a href="#">区域和终端节点</a>主题。</p>	2017 年 27 月 7 日
新区域	<p>AWS CodePipeline 现在提供 美国西部（加利福尼亚北部）、加拿大（中部）和 欧洲（伦敦）。更新了<a href="#">AWS CodePipeline 中的限制 (p. 208)</a>主题及<a href="#">区域和终端节点</a>主题。</p>	2017 年 6 月 29 日
对主题进行了更新	<p>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 <a href="#">在 AWS CodePipeline 中查看管道详细信息和历史记录 (p. 96)</a>、<a href="#">AWS CodePipeline 权限参考 (p. 192)</a> 和 <a href="#">AWS CodePipeline 中的限制 (p. 208)</a> 以反映这种支持。</p>	2017 年 6 月 22 日
对主题进行了更新	<p>向 <a href="#">测试操作集成 (p. 13)</a> 中的可用操作列表添加了 <a href="#">Nuvola</a>。</p>	2017 年 5 月 18 日

更改	描述	更改日期
对主题进行了更新	在 AWS CodePipeline 向导中，页面 Step 4: Beta 已重命名为 Step 4: Deploy。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
对主题进行了更新	<p>现在您可以将 AWS CodeBuild 作为测试操作添加到管道的任何阶段。这让您能够更轻松地使用 AWS CodeBuild 对您的代码运行单元测试。在此版本之前，您只能使用 AWS CodeBuild 作为生成操作的一部分运行单元测试。生成操作需要生成输出项目，而单元测试通常不生成该项目。</p> <p>更新了主题<a href="#">AWS CodePipeline 产品和服务集成 (p. 10)</a>、在<a href="#">AWS CodePipeline 中编辑管道 (p. 90)</a>和<a href="#">AWS CodePipeline 管道结构参考 (p. 202)</a>以反映对 AWS CodeBuild 的这种支持。</p>	2017 年 3 月 8 日
新增和更新的主题	<p>目录已经过重组，以包括有关管道、操作和阶段过渡的章节。已为 AWS CodePipeline 教程添加新的部分。为了方便使用，<a href="#">AWS CodePipeline 产品和服务集成 (p. 10)</a>已分割为更短的主题。</p> <p>新的章节 <a href="#">AWS CodePipeline 身份验证、访问控制和安全配置 (p. 171)</a>提供了有关使用 <a href="#">AWS Identity and Access Management (IAM)</a> 和 AWS CodePipeline 通过使用凭证帮助保护对您的资源的访问的全面地信息。这些凭证提供访问 AWS 资源所需的权限，例如在 Amazon S3 存储桶中放置和检索项目，以及将 AWS OpsWorks 堆栈集成到管道中。</p>	2017 年 2 月 8 日
新区域	AWS CodePipeline 目前已在亚太区域（东京）可用。更新了 <a href="#">AWS CodePipeline 中的限制 (p. 208)</a> 主题及 <a href="#">区域和终端节点</a> 主题。	2016 年 12 月 14 日
新区域	AWS CodePipeline 目前已在南美洲（圣保罗）可用。更新了 <a href="#">AWS CodePipeline 中的限制 (p. 208)</a> 主题及 <a href="#">区域和终端节点</a> 主题。	2016 年 12 月 7 日
对主题进行了更新	<p>现在您可以将 AWS CodeBuild 作为生成操作添加到管道的任何阶段。AWS CodeBuild 是一项在云中完全托管的生成服务，可编译源代码、运行单元测试以及生成可供部署的项目。您可以使用现有的生成项目或在 AWS 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。</p> <p>更新了主题<a href="#">AWS CodePipeline 产品和服务集成 (p. 10)</a>、在<a href="#">AWS CodePipeline 中创建管道 (p. 82)</a>、<a href="#">AWS CodePipeline 身份验证、访问控制和安全配置 (p. 171)</a>和<a href="#">AWS CodePipeline 管道结构参考 (p. 202)</a>以反映对 AWS CodeBuild 的这种支持。</p> <p>现在，您可以将 AWS CodePipeline 与 AWS CloudFormation 和 AWS 无服务器应用程序模型结合使用来持续提供无服务器应用程序。更新了主题<a href="#">AWS CodePipeline 产品和服务集成 (p. 10)</a>，以反映这一支持。</p> <p><a href="#">AWS CodePipeline 产品和服务集成 (p. 10)</a>已经过重组，以便按操作类型组合 AWS 和合作伙伴产品。</p>	2016 年 12 月 1 日
新区域	AWS CodePipeline 目前已在欧洲（法兰克福）可用。更新了 <a href="#">AWS CodePipeline 中的限制 (p. 208)</a> 主题及 <a href="#">区域和终端节点</a> 主题。	2016 年 11 月 16 日

更改	描述	更改日期
对主题进行了更新	现在，可以选择 AWS CloudFormation 作为管道中的部署提供商，这使您能够在管道执行过程中对 AWS CloudFormation 堆栈和更改集采取措施。更新了主题 <a href="#">AWS CodePipeline 产品和服务集成 (p. 10)</a> 、在 <a href="#">AWS CodePipeline 中创建管道 (p. 82)</a> 、 <a href="#">AWS CodePipeline 身份验证、访问控制和安全配置 (p. 171)</a> 和 <a href="#">AWS CodePipeline 管道结构参考 (p. 202)</a> 以反映对 AWS CloudFormation 的这种支持。	2016 年 11 月 3 日
新区域	AWS CodePipeline 目前已在亚太区域（悉尼）区域可用。更新了 <a href="#">AWS CodePipeline 中的限制 (p. 208)</a> 主题及 <a href="#">区域和终端节点</a> 主题。	2016 年 10 月 26 日
新区域	AWS CodePipeline 目前已在亚太区域（新加坡）可用。更新了 <a href="#">AWS CodePipeline 中的限制 (p. 208)</a> 主题及 <a href="#">区域和终端节点</a> 主题。	2016 年 10 月 20 日
新区域	AWS CodePipeline 现已在美国东部（俄亥俄州）区域可用。更新了 <a href="#">AWS CodePipeline 中的限制 (p. 208)</a> 主题及 <a href="#">区域和终端节点</a> 主题。	2016 年 10 月 17 日
对主题进行了更新	更新了在 <a href="#">AWS CodePipeline 中创建管道 (p. 82)</a> ，以反映对在源提供商和生成提供商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
对主题进行了更新	更新了 <a href="#">管理 AWS CodePipeline 中的审批操作 (p. 139)</a> 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 <a href="#">查看 AWS CodePipeline 管道中的当前源修订详细信息 (p. 162)</a> 介绍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 <a href="#">使用 AWS CodePipeline 监控管道 (p. 153)</a> 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
新增和更新的主题	新部分 <a href="#">管理 AWS CodePipeline 中的审批操作 (p. 139)</a> 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
新区域	AWS CodePipeline 在欧洲（爱尔兰）区域中现已可用。更新了 <a href="#">AWS CodePipeline 中的限制 (p. 208)</a> 主题及 <a href="#">区域和终端节点</a> 主题。	2016 年 6 月 23 日
新主题	添加了新主题在 <a href="#">AWS CodePipeline 中重试失败的操作 (p. 136)</a> ，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
对主题进行了更新	更新了在 <a href="#">AWS CodePipeline 中创建管道 (p. 82)</a> 、 <a href="#">AWS CodePipeline 身份验证、访问控制和安全配置 (p. 171)</a> 、 <a href="#">AWS CodePipeline 管道结构参考 (p. 202)</a> 和 <a href="#">AWS CodePipeline 产品和服务集成 (p. 10)</a> 等许多主题，以反映支持配置管道以便将代码与自定义 Chef 说明书和 AWS OpsWorks 中创建的应用程序一起部署。AWS CodePipeline 对 AWS OpsWorks 的支持目前仅在美国东部（弗吉尼亚北部）区域 (us-east-1) 可用。	2016 年 6 月 2 日



更改	描述	更改日期
新增和更新的主体	增加了新主题教程：创建一个简单的管道 (AWS CodeCommit 存储库) (p. 36)。该主题提供了一个示例演练，演示如何使用 AWS CodeCommit 存储库和分支作为管道中源操作的源位置。更新了一些其他几个主题，以反映与 AWS CodeCommit 的集成，包括 AWS CodePipeline 身份验证、访问控制和安全配置 (p. 171)、AWS CodePipeline 产品和服务集成 (p. 10)、教程：创建一个四阶段管道 (p. 47) 和 AWS CodePipeline 故障排除 (p. 165)。	2016 年 4 月 18 日
新主题	增加了新主题在 AWS CodePipeline 管道中调用 AWS Lambda 函数 (p. 120)。该主题包含示例 AWS Lambda 函数以及向管道添加 Lambda 函数的步骤。	2016 年 1 月 27 日
对主题进行了更新	AWS CodePipeline 身份验证、访问控制和安全配置 (p. 171) 中增加了一个新的部分基于资源的策略 (p. 178)。	2016 年 1 月 22 日
新主题	增加了新主题 AWS CodePipeline 产品和服务集成 (p. 10)。有关与合作伙伴和其他 AWS 服务集成的信息已移到该主题。还添加了博客和视频链接。	2015 年 12 月 17 日
对主题进行了更新	向 AWS CodePipeline 产品和服务集成 (p. 10) 中添加了 Details of integration with Solano CI (p. 12)。	2015 年 11 月 17 日
对主题进行了更新	AWS CodePipeline Plugin for Jenkins 现在可通过 Jenkins 插件管理器作为 Jenkins 插件库的一部分提供。教程：创建一个四阶段管道 (p. 47) 中安装插件的步骤已更新。	2015 年 11 月 9 日
新区域	AWS CodePipeline 目前已在美国西部（俄勒冈）区域可用。更新了 AWS CodePipeline 中的限制 (p. 208) 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适用于 AWS CodePipeline 的 Amazon S3 中存储的项目配置服务器端加密 (p. 195) 和在 AWS CodePipeline 中创建使用另一个 AWS 账户的资源的管道 (p. 102)。AWS CodePipeline 身份验证、访问控制和安全配置 (p. 171) 中增加了一个新的部分示例 8：在管道中使用与另一个账户相关联的 AWS 资源 (p. 189)。	2015 年 8 月 25 日
对主题进行了更新	更新了在 AWS CodePipeline 中创建和添加自定义操作 (p. 111) 主题，以反映结构更改，包括 inputArtifactDetails 和 outputArtifactDetails。	2015 年 8 月 17 日
对主题进行了更新	更新了 AWS CodePipeline 故障排除 (p. 165) 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
对主题进行了更新	更新了 AWS CodePipeline 身份验证、访问控制和安全配置 (p. 171) 主题以反映 AWS CodePipeline 的服务角色的最新更新 (p. 180)。	2015 年 8 月 6 日
新主题	增加了 AWS CodePipeline 故障排除 (p. 165) 主题。在教程：创建一个四阶段管道 (p. 47) 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 (Amazon S3 存储桶) (p. 22) 和教程：创建一个四阶段管道 (p. 47) 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 <a href="#">教程：创建一个简单的管道 (Amazon S3 存储桶)</a> (p. 22)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 <a href="#">AWS CodePipeline 中的限制</a> (p. 208)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 <a href="#">AWS CodePipeline 身份验证、访问控制和安全配置</a> (p. 171)中的托管策略部分。	2015 年 7 月 10 日
第一个公开发布版	这是 AWS CodePipeline 用户指南的第一个公开发布版。	2015 年 7 月 9 日



# AWS 词汇表

有关最新 AWS 术语，请参阅 AWS General Reference 中的 [AWS 词汇表](#)。