AWS CodePipeline: User Guide
Copyright © 2020 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

What is CodePipeline? ................................................................................................................................. 1
Continuous delivery and continuous integration .................................................................................................. 1
What can I do with CodePipeline? ...................................................................................................................... 1
Video introduction to AWS CodePipeline ............................................................................................................ 2
A quick look at CodePipeline .................................................................................................................................. 2
How do I get started with CodePipeline? ........................................................................................................... 3
We want to hear from you ..................................................................................................................................... 3
Concepts ............................................................................................................................................................. 3
Pipeline terms ...................................................................................................................................................... 3
DevOps pipeline example ....................................................................................................................................... 6
How pipeline executions work .............................................................................................................................. 8
How pipeline executions are started ..................................................................................................................... 8
How pipeline executions are stopped .................................................................................................................. 8
How executions are processed in a pipeline ........................................................................................................ 12
Managing pipeline executions ............................................................................................................................ 13
Input and output artifacts ....................................................................................................................................... 15
Getting started ..................................................................................................................................................... 19
Step 1: Create an AWS account ............................................................................................................................. 20
Step 2: Create or use an IAM user ......................................................................................................................... 20
Step 3: Use an IAM managed policy to assign CodePipeline permissions to the IAM user .................................. 20
Step 4: Install the AWS CLI ..................................................................................................................................... 21
Step 5: Open the console for CodePipeline ......................................................................................................... 21
Next steps ............................................................................................................................................................... 21
Product and service integrations .......................................................................................................................... 22
Integrations with CodePipeline action types .......................................................................................................... 22
Source action integrations ....................................................................................................................................... 22
Build action integrations ....................................................................................................................................... 25
Test action integrations ......................................................................................................................................... 26
Deploy action integrations ....................................................................................................................................... 27
Approval action integrations ..................................................................................................................................... 31
Invoke action integrations ....................................................................................................................................... 31
General integrations with CodePipeline ................................................................................................................ 31
Examples from the community ................................................................................................................................... 33
Blog posts ............................................................................................................................................................... 33
Videos ................................................................................................................................................................. 36
Tutorials ................................................................................................................................................................. 37
Tutorial: Create a simple pipeline (S3 bucket) ........................................................................................................ 38
Create an S3 bucket ................................................................................................................................................. 38
Create Windows Server Amazon EC2 instances and install the CodeDeploy agent .................................................. 39
Create an application in CodeDeploy .................................................................................................................. 41
Create your first pipeline ......................................................................................................................................... 42
Add another stage ...................................................................................................................................................... 44
Disable and enable transitions between stages ..................................................................................................... 51
Clean up resources .................................................................................................................................................. 51
Tutorial: Create a simple pipeline (CodeCommit repository) .................................................................................... 52
Create a CodeCommit repository .......................................................................................................................... 52
Download, commit, and push your code .................................................................................................................. 53
Create an EC2 Linux instance and install the CodeDeploy agent ........................................................................... 54
Create an application in CodeDeploy .................................................................................................................. 55
Create your first pipeline ......................................................................................................................................... 56
Update code in your CodeCommit repository ....................................................................................................... 60
Clean up resources .................................................................................................................................................. 62
Further reading ..................................................................................................................................................... 62
Tutorial: Create a four-stage pipeline .................................................................................................................... 63
View pipeline details and history (console) ................................................................. 237
View pipeline details and history (CLI) ................................................................. 244
Delete a pipeline ........................................................................................................... 250
Delete a pipeline (console) ......................................................................................... 251
Delete a pipeline (CLI) .............................................................................................. 251
Create a pipeline that uses resources from another account ........................................ 251
Prerequisite: Create an AWS KMS encryption key ..................................................... 253
Step 1: Set up account policies and roles ................................................................. 253
Step 2: Edit the pipeline .............................................................................................. 258
Edit a pipeline for push events .................................................................................... 261
Update pipelines for push events (console) ............................................................... 263
Update pipelines for push events (CLI) ..................................................................... 266
Update pipelines for push events (AWS CloudFormation template) ......................... 275
Create the CodePipeline service role ........................................................................... 312
Create the CodePipeline service role (console) ......................................................... 312
Create the CodePipeline service role (CLI) .............................................................. 312
Tag a pipeline ............................................................................................................... 313
Tag pipelines (console) ............................................................................................. 313
Tag pipelines (CLI) ..................................................................................................... 315
Create a connection to Bitbucket ................................................................................ 317
Create a connection (console) .................................................................................... 317
Create a connection (CLI) .......................................................................................... 321
Create a notification rule ............................................................................................ 321
Working with actions .................................................................................................. 324
Create a custom action for a pipeline ......................................................................... 324
Create a custom action ............................................................................................... 325
Create a job worker for your custom action ............................................................... 328
Add a custom action to a pipeline ............................................................................. 332
Tag a custom action in CodePipeline ......................................................................... 334
Add tags to a custom action ....................................................................................... 334
View tags for a custom action ..................................................................................... 335
Edit tags for a custom action ....................................................................................... 335
Remove tags from a custom action .............................................................................. 335
Invoke a Lambda function in a pipeline ...................................................................... 336
Step 1: Create a pipeline ............................................................................................. 337
Step 2: Create the Lambda function .......................................................................... 337
Step 3: Add the Lambda function to a pipeline in the CodePipeline console .............. 340
Step 4: Test the pipeline with the Lambda function .................................................. 341
Step 5: Next steps ....................................................................................................... 341
Example JSON event ................................................................................................. 342
Additional sample functions ....................................................................................... 343
Retry a failed action .................................................................................................... 352
Retry failed actions (console) ..................................................................................... 353
Retry failed actions (CLI) .......................................................................................... 353
Manage approval actions in pipelines .......................................................................... 355
Configuration options for manual approval actions .................................................... 355
Setup and workflow overview for approval actions .................................................... 356
Grant approval permissions to an IAM user in CodePipeline ..................................... 356
Grant Amazon SNS permissions to a service role ......................................................... 358
Add a manual approval action ..................................................................................... 359
Approve or reject an approval action .......................................................................... 362
JSON data format for manual approval notifications ................................................... 364
Add a cross-Region action to a pipeline ..................................................................... 364
Manage cross-Region actions in a pipeline (console) .................................................. 365
Add a cross-Region action to a pipeline (CLI) ............................................................ 367
Add a cross-Region action to a pipeline (AWS CloudFormation) .................................. 371
Working with variables ............................................................................................... 372
What is AWS CodePipeline?

AWS CodePipeline is a continuous delivery service you can use to model, visualize, and automate the steps required to release your software. You can quickly model and configure the different stages of a software release process. CodePipeline automates the steps required to release your software changes continuously. For information about pricing for CodePipeline, see Pricing.

Topics
- Continuous delivery and continuous integration (p. 1)
- What can I do with CodePipeline? (p. 1)
- A quick look at CodePipeline (p. 2)
- How do I get started with CodePipeline? (p. 3)
- We want to hear from you (p. 3)
- CodePipeline concepts (p. 3)
- DevOps pipeline example (p. 6)
- How pipeline executions work (p. 8)
- Input and output artifacts (p. 15)

Continuous delivery and continuous integration

CodePipeline is a continuous delivery service that automates the building, testing, and deployment of your software into production.

Continuous delivery is a software development methodology where the release process is automated. Every software change is automatically built, tested, and deployed to production. Before the final push to production, a person, an automated test, or a business rule decides when the final push should occur. Although every successful software change can be immediately released to production with continuous delivery, not all changes need to be released right away.

Continuous integration is a software development practice where members of a team use a version control system and frequently integrate their work to the same location, such as a master branch. Each change is built and verified to detect integration errors as quickly as possible. Continuous integration is focused on automatically building and testing code, as compared to continuous delivery, which automates the entire software release process up to production.

For more information, see Practicing Continuous Integration and Continuous Delivery on AWS: Accelerating Software Delivery with DevOps.

You can use the CodePipeline console, the AWS Command Line Interface (AWS CLI), the AWS SDKs, or any combination of these to create and manage your pipelines.

What can I do with CodePipeline?

You can use CodePipeline to help you automatically build, test, and deploy your applications in the cloud. Specifically, you can:
• **Automate your release processes**: CodePipeline fully automates your release process from end to end, starting from your source repository through build, test, and deployment. You can prevent changes from moving through a pipeline by including a manual approval action in any stage except a Source stage. You can release when you want, in the way you want, on the systems of your choice, across one instance or multiple instances.

• **Establish a consistent release process**: Define a consistent set of steps for every code change. CodePipeline runs each stage of your release according to your criteria.

• **Speed up delivery while improving quality**: You can automate your release process to allow your developers to test and release code incrementally and speed up the release of new features to your customers.

• **Use your favorite tools**: You can incorporate your existing source, build, and deployment tools into your pipeline. For a full list of AWS services and third-party tools currently supported by CodePipeline, see [Product and service integrations with CodePipeline](p. 22).

• **View progress at a glance**: You can review real-time status of your pipelines, check the details of any alerts, retry failed actions, view details about the source revisions used in the latest pipeline execution in each stage, and manually rerun any pipeline.

• **View pipeline history details**: You can view details about executions of a pipeline, including start and end times, run duration, and execution IDs.

**Video introduction to AWS CodePipeline**

This short video (3:06) describes how CodePipeline builds, tests, and deploys your code every time there is a code change, based on the release process models you define.

[Video Introduction to CodePipeline](#).

**A quick look at CodePipeline**

The following diagram shows an example release process using CodePipeline.

In this example, when developers commit changes to a source repository, CodePipeline automatically detects the changes. Those changes are built, and if any tests are configured, those tests are run. After the tests are complete, the built code is deployed to staging servers for testing. From the staging server,
CodePipeline runs more tests, such as integration or load tests. Upon the successful completion of those tests, and after a manual approval action that was added to the pipeline is approved, CodePipeline deploys the tested and approved code to production instances.

CodePipeline can deploy applications to EC2 instances by using CodeDeploy, AWS Elastic Beanstalk, or AWS OpsWorks Stacks. CodePipeline can also deploy container-based applications to services by using Amazon ECS. Developers can also use the integration points provided with CodePipeline to plug in other tools or services, including build services, test providers, or other deployment targets or systems.

A pipeline can be as simple or as complex as your release process requires.

How do I get started with CodePipeline?

To get started with CodePipeline:

1. **Learn** how CodePipeline works by reading the CodePipeline concepts (p. 3) section.
2. **Prepare** to use CodePipeline by following the steps in Getting started with CodePipeline (p. 19).
3. **Experiment** with CodePipeline by following the steps in the CodePipeline tutorials (p. 37) tutorials.
4. **Use** CodePipeline for your new or existing projects by following the steps in Create a pipeline in CodePipeline (p. 221).

We want to hear from you

We welcome your feedback. To contact us, visit the CodePipeline forum.

CodePipeline concepts

Modeling and configuring your automated release process is easier if you understand the concepts and terms used in AWS CodePipeline. Here are some concepts to know about as you use CodePipeline.

For an example of a DevOps pipeline, see DevOps pipeline example (p. 6).

Topics
- Pipeline terms (p. 3)

Pipeline terms

The following terms are used in CodePipeline:
- Pipelines (p. 4)
- Stages (p. 4)
- Actions (p. 4)
- Pipeline executions (p. 4)
- Stopped executions (p. 4)
- Failed executions (p. 5)
- Superseded executions (p. 5)
- Stage executions (p. 5)
Pipelines

A pipeline is a workflow construct that describes how software changes go through a release process. Each pipeline is made up of a series of stages.

Stages

A stage is a logical unit you can use to isolate an environment and to limit the number of concurrent changes in that environment. Each stage contains actions that are performed on the application artifacts. Your source code is an example of an artifact. A stage might be a build stage, where the source code is built and tests are run. It can also be a deployment stage, where code is deployed to runtime environments. Each stage is made up of a series of serial or parallel actions.

Actions

An action is a set of operations performed on application code and configured so that the actions run in the pipeline at a specified point. This can include things like a source action from a code change, an action for deploying the application to instances, and so on. For example, a deployment stage might contain a deployment action that deploys code to a compute service like Amazon EC2 or AWS Lambda.

Valid CodePipeline action types are source, build, test, deploy, approval, and invoke. For a list of action providers, see Valid action types and providers in CodePipeline (p. 454).

Actions can run in series or in parallel. For information about serial and parallel actions in a stage, see the runOrder information in action structure requirements.

Pipeline executions

An execution is a set of changes released by a pipeline. Each pipeline execution is unique and has its own ID. An execution corresponds to a set of changes, such as a merged commit or a manual release of the latest commit. Two executions can release the same set of changes at different times.

While a pipeline can process multiple executions at the same time, a pipeline stage processes only one execution at a time. To do this, a stage is locked while it processes an execution. Two pipeline executions can’t occupy the same stage at the same time.

Pipeline executions traverse pipeline stages in order.

Valid statuses for pipelines are InProgress, Stopping, Stopped, Succeeded, Superseded, and Failed. An execution with a Failed or Superseded status does not continue through the pipeline and cannot be retried. For more information, see PipelineExecution.

Stopped executions

The pipeline execution can be stopped manually so that the in-progress pipeline execution does not continue through the pipeline. If stopped manually, a pipeline execution shows a Stopping status until it is completely stopped. Then it shows a Stopped status. A Stopped pipeline execution can be retried.

There are two ways to stop a pipeline execution:

• Stop and wait
• **Stop and abandon**

For information about use cases for stopping an execution and sequence details for these options, see How pipeline executions are stopped (p. 8).

**Failed executions**

If an execution fails, it stops and does not completely traverse the pipeline. Its status is **FAILED** status and the stage is unlocked. A more recent execution can catch up and enter the unlocked stage and lock it. You can retry a failed execution unless the failed execution has been superseded or is not retryable.

**Superseded executions**

To deliver the latest set of changes through a pipeline, newer executions pass and replace less recent executions already running through the pipeline. When this occurs, the older execution is superseded by the newer execution. An execution can be superseded by a more recent execution at a certain point, which is the point between stages.

If an execution is waiting to enter a locked stage, a more recent execution might catch up and supersede it. The newer execution now waits for the stage to unlock, and the superseded execution stops with a **SUPERSEDED** status. When a pipeline execution is superseded, the execution is stopped and does not completely traverse the pipeline. You can no longer retry the superseded execution after it has been replaced at this stage.

For more information about superseded executions and locked stages, see How executions are processed in a pipeline (p. 12).

**Stage executions**

A **stage execution** is the process of completing all of the actions within a stage. For information about how a stage execution works and information about locked stages, see How executions are processed in a pipeline (p. 12).

Valid statuses for stages are **InProgress**, **Stopping**, **Stopped**, **Succeeded**, and **Failed**. For more information, see StageExecution.

**Action executions**

An **action execution** is the process of completing a configured action that operates on designated artifacts. These can be input artifacts, output artifacts, or both. For example, a build action might run build commands on an input artifact, such as compiling application source code. Action execution details include an action execution ID, the related pipeline execution source trigger, and the input and output artifacts for the action.

Valid statuses for actions are **InProgress**, **Abandoned**, **Succeeded**, or **Failed**. For more information, see ActionExecution.

**Transitions**

A **transition** is the point where a pipeline execution moves to the next stage in the pipeline. You can disable a stage's inbound transition to prevent executions from entering that stage, and then you can enable the transition to allow executions to continue. When more than one execution arrives at a disabled transition, only the latest execution continues to the next stage when the transition is enabled. This means that newer executions continue to supersede waiting executions while the transition is disabled, and then after the transition is enabled, the execution that continues is the superseding execution.
Artifacts

Artifacts refers to the collection of data, such as application source code, built applications, dependencies, definitions files, templates, and so on, that is worked on by pipeline actions. Artifacts are produced by some actions and consumed by others. In a pipeline, artifacts can be the set of files worked on by an action (input artifacts) or the updated output of a completed action (output artifacts).

Actions pass output to another action for further processing using the pipeline artifact bucket. CodePipeline copies artifacts to the artifact store, where the action picks them up. For more information about artifacts, see Input and output artifacts (p. 15).

Source revisions

When you make a source code change, a new version is created. A source revision is the version of a source change that triggers a pipeline execution. An execution processes that source revision only. For GitHub and CodeCommit repositories, this is the commit. For S3 buckets or actions, this is the object version.

DevOps pipeline example

As an example of a DevOps pipeline, a two-stage pipeline might have a source stage called Source and a second stage called Prod. In this example, the pipeline is updating the application with the latest changes and continuously deploying the latest result. Before it deploys the latest application, the pipeline builds and tests the web application. In this example, a group of developers have set up an infrastructure template and the source code for a web application in a GitHub repository called MyRepository.
For example, a developer pushes a fix to the web application's index page, and the following occurs:
1. The application source code is maintained in a repository configured as a GitHub source action in the pipeline. When developers push commits to the repository, CodePipeline detects the pushed change, and a pipeline execution starts from the Source Stage.

2. The GitHub source action completes successfully (that is, the latest changes have been downloaded and stored to the artifact bucket unique to that execution). The output artifacts produced by the GitHub source action, which are the application files from the repository, are then used as the input artifacts to be worked on by the actions in the next stage.

3. The pipeline execution transitions from the Source Stage to the Prod Stage. The first action in the Prod Stage runs a build project created in CodeBuild and configured as a build action in the pipeline. The build task pulls a build environment image and builds the web application in a virtual container.

4. The next action in the Prod Stage is a unit test project created in CodeBuild and configured as a test action in the pipeline.

5. The unit tested code is next worked on by a deploy action in the Prod Stage that deploys the application to a production environment. After the deploy action completes successfully, the final action in the stage is an integration testing project created in CodeBuild and configured as a test action in the pipeline. The test action calls to shell scripts that install and run a test tool, such as a link checker, on the web application. After successful completion, the output is a built web application and a set of test results.

Developers can add actions to the pipeline that deploy or further test the application after it is built and tested for each change.

For more information, see How pipeline executions work (p. 8).

How pipeline executions work

This section provides an overview of the way CodePipeline processes a set of changes. CodePipeline tracks each pipeline execution that starts when a change is made to the source code. CodePipeline also tracks the way each execution progresses through the pipeline, including whether it is superseded by another execution.

How pipeline executions are started

You can trigger an execution when you change your source code or manually start the pipeline. You can also trigger an execution through an Amazon CloudWatch Events rule that you schedule. For example, when a source code change is pushed to a repository configured as the pipeline's source action, the pipeline detects the change and starts an execution.

Note

If a pipeline contains multiple source actions, all of them run again, even if a change is detected for one source action only.

How pipeline executions are stopped

To use the console to stop a pipeline execution, you can choose Stop execution on the pipeline visualization page, on the execution history page, or on the detailed history page. To use the CLI to stop a pipeline execution, you use the stop-pipeline-execution command. For more information, see Stop a pipeline execution in CodePipeline (p. 218).

There are two ways to stop a pipeline execution:

- **Stop and wait:** All in-progress action executions are allowed to complete, and subsequent actions are not started. The pipeline execution does not continue to subsequent stages. You cannot use this option on an execution that is already in a Stopping state.
• **Stop and abandon**: All in-progress action executions are abandoned and do not complete, and subsequent actions are not started. The pipeline execution does not continue to subsequent stages. You can use this option on an execution that is already in a Stopping state.

  **Note**
  This option can lead to failed tasks or out of sequence tasks.

Each option results in a different sequence of pipeline and action execution phases, as follows.

**Option 1: Stop and wait**

When you choose to stop and wait, the selected execution continues until in-progress actions are completed. For example, the following pipeline execution was stopped while the build action was in progress.

1. In the pipeline view, the success message banner is displayed, and the build action continues until it is completed. The pipeline execution status is **Stopping**.

   ![Pipeline Execution Stopping](image)

   In the history view, the status for in-progress actions, such as the build action, is **In progress** until the build action is completed. While actions are in progress, the pipeline execution status is **Stopping**.

2. The execution stops when the stopping process is complete. If the build action is completed successfully, its status is **Succeeded**, and the pipeline execution shows a status of **Stopped**. Subsequent actions do not start. The **Retry** button is enabled.
In the history view, the execution status is **Stopped** after the in-progress action is completed.

**Option 2: Stop and abandon**

When you choose to stop and abandon, the selected execution does not wait for in-progress actions to complete. The actions are abandoned. For example, the following pipeline execution was stopped and abandoned while the build action was in progress.

1. In the pipeline view, the success banner message is displayed, the build action shows a status of **In progress**, and the pipeline execution shows a status of **Stopping**.
2. After the pipeline execution stops, the build action shows a status of Abandoned, and the pipeline execution shows a status of Stopped. Subsequent actions do not start. The Retry button is enabled.

3. In the history view, the execution status is Stopped.
Use cases for stopping a pipeline execution

We recommend that you use the stop and wait option to stop a pipeline execution. This option is safer because it avoids possible failed or out-of-sequence tasks in your pipeline. When an action is abandoned in CodePipeline, the action provider continues any tasks related to the action. In the case of an AWS CloudFormation action, the deployment action in the pipeline is abandoned, but the stack update might continue and result in a failed update.

As an example of abandoned actions that can result in out-of-sequence tasks, if you are deploying a large file (1GB) through an S3 deployment action, and you choose to stop and abandon the action while the deployment is already in progress, the action is abandoned in CodePipeline, but continues in Amazon S3. Amazon S3 does not encounter any instruction to cancel the upload. Next, if you start a new pipeline execution with a very small file, there are now two deployments in progress. Because the file size of the new execution is small, the new deployment completes while the old deployment is still uploading. When the old deployment completes, the new file is overwritten by the old file.

You might want to use the stop and abandon option in the case where you have a custom action. For example, you can abandon a custom action with work that does not need to finish before you stop the execution for a new execution with a bug fix.

How executions are processed in a pipeline

An execution consists of a set of changes picked up and processed by the execution. Pipelines can process multiple executions at the same time. Each execution is run through the pipeline separately. The pipeline processes each execution in order and might supersede an earlier execution with a later one. The following rules are used to process executions in a pipeline.

Rule 1: Stages are locked when an execution is being processed

Because each stage can process only one execution at a time, the stage is locked while in progress. When the execution completes a stage, it transitions to the next stage in the pipeline.

Before: Stage 1 is locked as Execution 1 enters. After: Stage 2 is locked as Execution 1 enters.
Rule 2: Subsequent executions wait for the stage to be unlocked

While a stage is locked, waiting executions are held in front of the locked stage. All actions configured for a stage must be completed successfully before the stage is considered complete. A failure releases the lock on the stage. When an execution is stopped, the execution does not continue in a stage and the stage is unlocked.

**Note**
Before you stop an execution, we recommend that you disable the transition in front of the stage. This way, when the stage is unlocked due to the stopped execution, the stage does not accept a subsequent pipeline execution.

![Diagram of Rule 2](image1)

Before: Stage 2 is locked as Execution 1 enters. After: Execution 2 exits Stage 1 and waits between stages.

Rule 3: Waiting executions are superseded by more recent executions

Executions are only superseded in between stages. A locked stage holds one execution at the front of the stage awaiting the stage to complete. A more recent execution overtakes a waiting execution and continues to the next stage as soon as the stage is unlocked. The superseded execution does not continue. In this example, Execution 2 has been superseded by Execution 3 while awaiting the locked stage. Execution 3 enters the stage next.

![Diagram of Rule 3](image2)

Before: execution 2 waits between stages while execution 3 enters stage 1. After: execution 3 exits stage 1. execution 2 is superseded by execution 3.

Managing pipeline executions

The flow of pipeline executions can be controlled by:

- A transition, which controls the flow of executions into the stage. Transitions can be enabled or disabled. After you enable the transition, any execution waiting to enter the stage moves into the stage and locks it. Similar to executions awaiting a locked stage, when a transition is disabled, the execution waiting to enter the stage can still be superseded by a new execution. When a disabled transition is re-enabled, the latest execution, including any that superseded older executions while the transition was disabled, enters the stage.

- An approval action, which prevents a pipeline from transitioning to the next action until permission is granted (for example, through manual approval from an authorized IAM user). You might use an approval action when you want to control the time at which a pipeline transitions to a final Production stage, for example.
Note
A stage with an approval action is locked until the approval action is approved or rejected or has timed out. A timed-out approval action is processed in the same way as a failed action.

- A failure, when an action in a stage does not complete successfully. The revision does not transition to the next action in the stage or the next stage in the pipeline. The following can occur:
  - You manually retry the stage that contains the failed actions. This resumes the execution (it retries failed actions and, if they succeed, continues in the stage/pipeline).
  - Another execution enters the failed stage and supersedes the failed execution. At this point, the failed execution cannot be retried.

Recommended pipeline structure

When deciding how a code change should flow through your pipeline, it is best to group related actions within a stage so that, when the stage locks, the actions all process the same execution. You might create a stage for each application environment, AWS Region, or Availability Zone, and so on. A pipeline with too many stages (that is, too granular) can allow too many concurrent changes, while a pipeline with many actions in a large stage (too coarse) can take too long to release a change.

As an example, a test action after a deployment action in the same stage is guaranteed to test the same change that was deployed. In this example, a change is deployed to a Test environment and then tested, and then the latest change from the test environment is deployed to a Production environment. In the recommended example, the Test environment and the Prod environment are separate stages.
Input and output artifacts

CodePipeline integrates with development tools to check for code changes and then build and deploy through all of the stages of the continuous delivery process.

Stages use input and output artifacts that are stored in the Amazon S3 artifact bucket you chose when you created the pipeline. CodePipeline zips and transfers the files for input or output artifacts as appropriate for the action type in the stage.

For example:

1. CodePipeline triggers your pipeline to run when there is a commit to the source repository, providing the output artifact (any files to be built) from the Source stage.
2. The output artifact (any files to be built) from the previous step is ingested as an input artifact to the Build stage. An output artifact (the built application) from the Build stage can be an updated application or an updated Docker image built to a container.
3. The output artifact from the previous step (the built application) is ingested as an input artifact to the **Deploy** stage, such as staging or production environments in the AWS Cloud. You can deploy applications to a deployment fleet, or you can deploy container-based applications to tasks running in ECS clusters.

When you create or edit an action, you designate the input and output artifact or artifacts for the action. In this example for a two-stage pipeline with a **Source** and **Deploy** stage, in **Edit Action**, you choose the artifact name of the source action for the input artifact for the deploy action.

The following diagram shows a high-level artifact workflow between stages in CodePipeline.
• When you use the console to create your first pipeline, CodePipeline creates an Amazon S3 bucket in the same AWS Region to store items for all pipelines. Every time you use the console to create another pipeline in that Region, CodePipeline creates a folder for that pipeline in the bucket. It uses that folder to store artifacts for your pipeline as the automated release process runs. This bucket is named codepipeline-region-12345EXAMPLE, where region is the AWS Region in which you created the pipeline, and 12345EXAMPLE is a 12-digit random number that ensures the bucket name is unique.

Note
CodePipeline truncates artifact names, which can cause some bucket names to appear similar. Even though the artifact name appears to be truncated, CodePipeline maps to the artifact bucket in a way that is not affected by artifacts with truncated names. The pipeline can function normally. This is not an issue with the folder or artifacts. There is a 100-character limit to pipeline names. Although the artifact folder name might appear to be shortened, it is still unique for your pipeline.
When you create or edit a pipeline, you must have an artifact bucket in the pipeline Region and you must have one artifact bucket per Region where you plan to execute an action. If you use the console to create a pipeline or cross-Region actions, default artifact buckets are configured by CodePipeline in the Regions where you have actions.

If you use the AWS CLI to create a pipeline, you can store the artifacts for that pipeline in any Amazon S3 bucket as long as that bucket is in the same AWS Region as the pipeline. You might do this if you are concerned about exceeding the limits of Amazon S3 buckets allowed for your account. If you use the AWS CLI to create or edit a pipeline, and you add a cross-Region action (an action with an AWS provider in a Region different from your pipeline), you must provide an artifact bucket for each additional Region where you plan to execute an action.

- Every action has a type. Depending on the type, the action might have one or both of the following:
  - An input artifact, which is the artifact it consumes or works on over the course of the action run.
  - An output artifact, which is the output of the action.

Every output artifact in the pipeline must have a unique name. Every input artifact for an action must match the output artifact of an action earlier in the pipeline, whether that action is immediately before the action in a stage or runs in a stage several stages earlier.

An artifact can be worked on by more than one action.

The following shows how input artifacts and output artifacts are produced and consumed in a pipeline:
Getting started with CodePipeline

If you are new to CodePipeline, you can follow the tutorials in this guide after following the steps in this chapter to get set up.

The CodePipeline console includes helpful information in a collapsible panel that you can open from the information icon or any Info link on the page. You can close this panel at any time.

The CodePipeline console also provides a way to quickly search for your resources, such as repositories, build projects, deployment applications, and pipelines. Choose Go to resource or press the / key, and then type the name of the resource. Any matches appear in the list. Searches are case insensitive. You only see resources that you have permissions to view. For more information, see Viewing resources in the console (p. 421).

Before you can use AWS CodePipeline for the first time, you must complete the following steps.

Topics

- Step 1: Create an AWS account (p. 20)
- Step 2: Create or use an IAM user (p. 20)
- Step 3: Use an IAM managed policy to assign CodePipeline permissions to the IAM user (p. 20)
- Step 4: Install the AWS CLI (p. 21)
- Step 5: Open the console for CodePipeline (p. 21)
Step 1: Create an AWS account

Create an AWS account, if you haven’t done so already, by going to https://aws.amazon.com/ and choosing Sign Up.

Step 2: Create or use an IAM user

Create an IAM user or use an existing one in your AWS account. Make sure that you have an AWS access key ID and an AWS secret access key associated with that IAM user. For more information, see Creating an IAM User in Your AWS Account.

Step 3: Use an IAM managed policy to assign CodePipeline permissions to the IAM user

You must give the IAM user permissions to interact with CodePipeline. The quickest way to do this is to apply the AWSCodePipelineFullAccess managed policy to the IAM user.

**Note**
The AWSCodePipelineFullAccess policy includes permissions that allow the console user to pass an IAM role to CodePipeline or other AWS services. This allows the service to assume the role and perform actions on your behalf. When you attach the policy to a user, role, or group, the iam:PassRole permissions are applied. Make sure the policy is only applied to trusted users. When users with these permissions use the console to create or edit a pipeline, the following choices are available:

- Create a CodePipeline service role or choose an existing one and pass the role to CodePipeline
- Might choose to create a CloudWatch Events rule for change detection and pass the CloudWatch Events service role to CloudWatch Events

For more information, see Granting a User Permissions to Pass a Role to an AWS Service.

To grant permissions to an IAM user using the AWS Management Console

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the IAM console, in the navigation pane, choose Policies, and then choose the AWSCodePipelineFullAccess managed policy from the list of policies.
3. On the Summary page, choose the Policy Usage tab, and then choose Attach.
4. On the Attach Policy page, select the check box next to the IAM users or groups, and then choose Attach Policy.

**Note**
The AWSCodePipelineFullAccess policy provides access to all CodePipeline actions and resources that the IAM user has access to, as well as all possible actions when creating stages in a pipeline, such as creating stages that include CodeDeploy, Elastic Beanstalk, or Amazon S3. As a best practice, you should grant individuals only the permissions they need to perform their duties. For more information about how to restrict IAM users to a limited API Version 2015-07-09

20
set of CodePipeline actions and resources, see Remove permissions from the CodePipeline service role (p. 446).

Step 4: Install the AWS CLI

To call CodePipeline commands from the AWS CLI on a local development machine, you must install the AWS CLI. This step is optional if you intend to get started using only the steps in this guide for the CodePipeline console.

To install and configure the AWS CLI

1. On your local machine, download and install the AWS CLI. This will enable you to interact with CodePipeline from the command line. For more information, see Getting Set Up with the AWS Command Line Interface.

   Note
   CodePipeline works only with AWS CLI versions 1.7.38 and later. To determine which version of the AWS CLI that you may have installed, run the command `aws --version`. To upgrade an older version of the AWS CLI to the latest version, follow the instructions in Uninstalling the AWS CLI, and then follow the instructions in Installing the AWS Command Line Interface.

2. Configure the AWS CLI with the `configure` command, as follows:

   ```
   aws configure
   ```

   When prompted, specify the AWS access key and AWS secret access key of the IAM user that you will use with CodePipeline. When prompted for the default region name, specify the region where you will create the pipeline, such as `us-east-2`. When prompted for the default output format, specify `json`. For example:

   ```
   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
   For more information about IAM, access keys, and secret keys, see Managing Access Keys for IAM Users and How Do I Get Credentials?.
   For more information about the Regions and endpoints available for CodePipeline, see AWS CodePipeline endpoints and quotas.

Step 5: Open the console for CodePipeline

- Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home.

Next steps

You have completed the prerequisites. You can begin using CodePipeline. To start working with CodePipeline, see the CodePipeline tutorials (p. 37).
AWS CodePipeline User Guide
Integrations with CodePipeline action types

Product and service integrations
with CodePipeline
By default, AWS CodePipeline is integrated with a number of AWS services and partner products and
services. Use the information in the following sections to help you conﬁgure CodePipeline to integrate
with the products and services you use.
Topics
• Integrations with CodePipeline action types (p. 22)
• General integrations with CodePipeline (p. 31)
• Examples from the community (p. 33)

Integrations with CodePipeline action types
The integrations information in this topic is organized by CodePipeline action type.
Topics
• Source action integrations (p. 22)
• Build action integrations (p. 25)
• Test action integrations (p. 26)
• Deploy action integrations (p. 27)
• Approval action integrations (p. 31)
• Invoke action integrations (p. 31)

Source action integrations
The following information is organized by CodePipeline action type and can help you conﬁgure
CodePipeline to integrate with the products and services you use.
Amazon Simple Storage
Service (Amazon S3)

Amazon S3 is storage for the internet. You can use Amazon S3 to store and
retrieve any amount of data at any time, from anywhere on the web. You
can conﬁgure CodePipeline to use a versioned Amazon S3 bucket as the
source stage for your code. Create the bucket and enable versioning on
it. Then you can create a pipeline that uses the bucket as part of a source
action in a stage.

Note

Amazon S3 can also be included in a pipeline as a deploy action.
Learn more:
• For action parameters and deﬁnitions, see the action structure reference
page for Amazon S3 (p. 496).
• Step 1: Create an S3 bucket for your application (p. 38)
• Create a pipeline (CLI) (p. 228)
• CodePipeline uses Amazon CloudWatch Events to detect changes in your
Amazon S3 source bucket. Each source action has a corresponding event
API Version 2015-07-09
22


source. See General integrations with CodePipeline (p. 31).

**AWS CodeCommit**

CodeCommit is a version control service that you can use to privately store and manage assets (such as documents, source code, and binary files) in the cloud. You can configure CodePipeline to use a branch in a CodeCommit repository as the source stage for your code. Create the repository and associate it with a working directory on your local machine. Then you can create a pipeline that uses the branch as part of a source action in a stage. You can connect to the CodeCommit repository by either creating a pipeline or editing an existing one.

Learn more:

- For action parameters and definitions, see the action structure reference page for CodeCommit (p. 479).
- Tutorial: Create a simple pipeline (CodeCommit repository) (p. 52)
- CodePipeline uses Amazon CloudWatch Events to detect changes in CodeCommit repositories used as a source for a pipeline. Each source action has a corresponding event rule. This event rule starts your pipeline when a change occurs in the repository. See General integrations with CodePipeline (p. 31).
GitHub

You can configure CodePipeline to use a GitHub repository as the source stage for your code. You must have previously created a GitHub account and at least one GitHub repository. You can connect to the GitHub repository by either creating a pipeline or editing an existing one.

**Note**

CodePipeline integration with GitHub Enterprise is not supported.

For action parameters and definitions, see the action structure reference page for GitHub (p. 486).

The first time you add a GitHub repository to a pipeline, you are prompted to authorize CodePipeline access to your repositories. To integrate with GitHub, CodePipeline creates an OAuth application for your pipeline. If you create or edit your pipeline in the console, CodePipeline creates a GitHub webhook that starts your pipeline when a change occurs in the repository. The token and webhook require the following GitHub scopes:

- The repo scope, which is used for full control to read and pull artifacts from public and private repositories into a pipeline.
- The admin:repo_hook scope, which is used for full control of repository hooks.

For more information about GitHub scopes, see the GitHub Developer API Reference.

Access for CodePipeline is configured for all repositories to which that GitHub account has access. It cannot currently be configured for individual repositories. To revoke this access from GitHub, choose Settings, and then choose Applications. Under Authorized applications, find CodePipeline in the list of authorized applications, and then choose Revoke. Revoking access immediately prevents CodePipeline from accessing any GitHub repositories previously configured for access with that GitHub account.

If you want to limit the access CodePipeline has to repositories, create a GitHub account, grant that account access only to those repositories you want to integrate with CodePipeline, and then use that account when you configure CodePipeline to use GitHub repositories for source stages in pipelines.

Learn more:

- For action parameters and definitions, see the action structure reference page for GitHub (p. 486).
- Tutorial: Create a four-stage pipeline (p. 63)
- Use webhooks to start a pipeline (GitHub source) (p. 199)
## Build action integrations

### Amazon ECR

**Amazon ECR** is an AWS Docker image repository service. You use Docker push and pull commands to upload Docker images to your repository. An Amazon ECR repository URI and image are used in Amazon ECS task definitions to reference source image information.

Learn more:
- For action parameters and definitions, see the action structure reference page for Amazon ECR (p. 490).
- Create a pipeline in CodePipeline (p. 221)
- Tutorial: Create a pipeline with an Amazon ECR source and ECS-to-CodeDeploy deployment (p. 116)

### AWS CodeStar Connections

(Connections to Bitbucket)

You can set up resources called **connections** to allow your pipelines to access third-party code repositories like Bitbucket. When you create a connection, you install the AWS CodeStar app with your third-party code repository, and then associate it with your connection.

Learn more:
- For action parameters and definitions, see the action structure reference page for CodeStarSourceConnection (p. 482).
- Create a Connection
- Create a connection to Bitbucket (p. 317)

---

### Build action integrations

| **AWS CodeBuild (p. 475)** | **CodeBuild** is a fully managed build service that compiles your source code, runs unit tests, and produces artifacts that are ready to deploy. You can add CodeBuild as a build action to the build stage of a pipeline.  
  
  **Note**  
  CodeBuild can also be included in a pipeline as a test action, with or without a build output.  

Learn more:
- For action parameters and definitions, see the action structure reference page for AWS CodeBuild (p. 475).
- What Is CodeBuild?  
  CodeBuild – Fully Managed Build Service  

| **CloudBees** | You can configure CodePipeline to use **CloudBees** to build or test your code in one or more actions in a pipeline.  

Learn more:
- Cloud First with AWS  

| **Jenkins** | You can configure CodePipeline to use **Jenkins CI** to build or test your code in one or more actions in a pipeline. You must have previously created a Jenkins project and installed and configured the CodePipeline Plugin for Jenkins for that project. You can connect to the Jenkins project by either creating a new pipeline or editing an existing one.  

---

API Version 2015-07-09

25
Access for Jenkins is configured on a per-project basis. You must install the CodePipeline Plugin for Jenkins on every Jenkins instance you want to use with CodePipeline. You must also configure CodePipeline access to the Jenkins project. Secure your Jenkins project by configuring it to accept HTTPS/SSL connections only. If your Jenkins project is installed on an Amazon EC2 instance, consider providing your AWS credentials by installing the AWS CLI on each instance. Then configure an AWS profile on those instances with the IAM user profile and AWS credentials you want to use for connections. This is an alternative to adding and storing them through the Jenkins web interface.

Learn more:
- Accessing Jenkins
- Tutorial: Create a four-stage pipeline (p. 63)

**TeamCity**

You can configure CodePipeline to use TeamCity to build and test your code in one or more actions in a pipeline.

Learn more:
- TeamCity Plugin for CodePipeline
- Building End-to-End Continuous Delivery and Deployment Pipelines in AWS and TeamCity

## Test action integrations

| **AWS CodeBuild (p. 475)** | CodeBuild is a fully managed build service in the cloud. CodeBuild compiles your source code, runs unit tests, and produces artifacts that are ready to deploy. You can add CodeBuild to a pipeline as a test action. For more information, see the CodePipeline Action Configuration Reference for AWS CodeBuild (p. 475).

**Note**

CodeBuild can also be included in a pipeline as a build action, with a mandatory build output artifact.

Learn more:
- For action parameters and definitions, see the action structure reference page for AWS CodeBuild (p. 475).
- What Is CodeBuild? |

| AWS Device Farm | AWS Device Farm is an app testing service that you can use to test and interact with your Android, iOS, and web applications on real, physical phones and tablets. You can configure CodePipeline to use AWS Device Farm to test your code in one or more actions in a pipeline. AWS Device Farm allows you to upload your own tests or use built-in, script-free compatibility tests. Because testing is performed in parallel, tests on multiple devices begin in minutes. A test report that contains high-level results, low-level logs, pixel-to-pixel screenshots, and performance data is updated as tests are completed. AWS Device Farm supports testing of native and hybrid Android, iOS, and Fire OS apps, including those created with PhoneGap, Titanium, Xamarin, Unity, and other frameworks. |

API Version 2015-07-09
26
It supports remote access of Android apps, which allows you to interact directly with test devices.

Learn more:
- What Is AWS Device Farm?
- Using AWS Device Farm in a CodePipeline Test Stage

BlazeMeter  You can configure CodePipeline to use BlazeMeter to test your code in one or more actions in a pipeline.
Learn more:
- BlazeMeter documentation for testing with CodePipeline

Ghost Inspector  You can configure CodePipeline to use Ghost Inspector to test your code in one or more actions in a pipeline.
Learn more:
- Ghost Inspector documentation for service integration with CodePipeline

Micro Focus StormRunner Load  You can configure CodePipeline to use Micro Focus StormRunner Load in one or more actions in a pipeline.
Learn more:
- Micro Focus StormRunner Load documentation for integrating with CodePipeline

Nouvola  You can configure CodePipeline to use Nouvola to test your code in one or more actions in a pipeline.
Learn more:
- Nouvola Plugin for CodePipeline

Runscope  You can configure CodePipeline to use Runscope to test your code in one or more actions in a pipeline.
Learn more:
- Runscope documentation for integrating with CodePipeline

---

**Deploy action integrations**

**Amazon Simple Storage Service (Amazon S3)**  Amazon S3 is storage for the internet. You can use Amazon S3 to store and retrieve any amount of data at any time, from anywhere on the web. You can add an action to a pipeline that uses Amazon S3 as a deployment provider.

**Note**  Amazon S3 can also be included in a pipeline as a source action.
Learn more:
- Create a pipeline in CodePipeline (p. 221)
### AWS AppConfig

AWS AppConfig is a capability of AWS Systems Manager to create, manage, and quickly deploy application configurations. You can use AppConfig with applications hosted on EC2 instances, AWS Lambda, containers, mobile applications, or IoT devices.

Learn more:
- CodePipeline Action Configuration Reference for AWS AppConfig (p. 504)
- Tutorial: Create a pipeline that uses AWS AppConfig as a deployment provider (p. 159)

### AWS CloudFormation (p. 470)

AWS CloudFormation gives developers and systems administrators an easy way to create and manage a collection of related AWS resources, using templates to provision and update those resources. You can use the service’s sample templates or create your own. Templates describe the AWS resources and any dependencies or runtime parameters required to run your application.

The AWS Serverless Application Model (AWS SAM) extends AWS CloudFormation to provide a simplified way to define and deploy serverless applications. AWS SAM supports Amazon API Gateway APIs, AWS Lambda functions, and Amazon DynamoDB tables. You can use CodePipeline with AWS CloudFormation and the AWS SAM to continuously deliver your serverless applications.

You can add an action to a pipeline that uses AWS CloudFormation as a deployment provider. When you use AWS CloudFormation as a deployment provider, you can take action on AWS CloudFormation stacks and change sets as part of a pipeline execution. AWS CloudFormation can create, update, replace, and delete stacks and change sets when a pipeline runs. As a result, AWS and custom resources can be created, provisioned, updated, or terminated during a pipeline execution according to the specifications you provide in AWS CloudFormation templates and parameter definitions.

Learn more:
- CodePipeline Action Configuration Reference for AWS CloudFormation (p. 470)
- Continuous Delivery with CodePipeline — Learn how to use CodePipeline to build a continuous delivery workflow for AWS CloudFormation.
- Automating Deployment of Lambda-based Applications — Learn how to use the AWS Serverless Application Model and AWS CloudFormation to build a continuous delivery workflow for your Lambda-based application.
<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
<th>Learn more</th>
</tr>
</thead>
</table>
| AWS CodeDeploy                | **CodeDeploy** coordinates application deployments to Amazon EC2 instances, on-premises instances, or both. You can configure CodePipeline to use CodeDeploy to deploy your code. You can create the CodeDeploy application, deployment, and deployment group to use in a deploy action in a stage either before you create the pipeline or when you use the **Create Pipeline** wizard. | • Step 3: Create an application in CodeDeploy (p. 41)  
• Tutorial: Create a simple pipeline (CodeCommit repository) (p. 52)                                                                                                                   |
| Amazon Elastic Container Service | Amazon ECS is a highly scalable, high performance container management service that allows you to run container-based applications in the AWS Cloud. When you create a pipeline, you can select Amazon ECS as a deployment provider. A change to code in your source control repository triggers your pipeline to build a new Docker image, push it to your container registry, and then deploy the updated image to Amazon ECS. You can also use the **ECS (Blue/Green)** provider action in CodePipeline to route and deploy traffic to Amazon ECS with CodeDeploy. | • What Is Amazon ECS?  
• Tutorial: Continuous Deployment with CodePipeline  
• Create a pipeline in CodePipeline (p. 221)  
• Tutorial: Create a pipeline with an Amazon ECR source and ECS-to-CodeDeploy deployment (p. 116)                                                                                     |
| AWS Elastic Beanstalk         | **Elastic Beanstalk** is a service for deploying and scaling web applications and services developed with Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker on familiar servers such as Apache, Nginx, Passenger, and IIS. You can configure CodePipeline to use Elastic Beanstalk to deploy your code. You can create the Elastic Beanstalk application and environment to use in a deploy action in a stage either before you create the pipeline or when you use the **Create Pipeline** wizard. | • Getting started using Elastic Beanstalk  
• Create a pipeline in CodePipeline (p. 221)                                                                                                                                         |
### AWS OpsWorks Stacks

AWS OpsWorks is a configuration management service that helps you configure and operate applications of all shapes and sizes using Chef. Using AWS OpsWorks Stacks, you can define the application's architecture and the specification of each component including package installation, software configuration and resources such as storage. You can configure CodePipeline to use AWS OpsWorks Stacks to deploy your code in conjunction with custom Chef cookbooks and applications in AWS OpsWorks.

- **Custom Chef Cookbooks** – AWS OpsWorks uses Chef Cookbooks to handle tasks such as installing and configuring packages and deploying applications.
- **Applications** – An AWS OpsWorks applications consists of code that you want to run on an application server. The application code is stored in a repository, such as an Amazon S3 bucket.

Before you create the pipeline, you create the AWS OpsWorks stack and layer. You can create the AWS OpsWorks application to use in a deploy action in a stage either before you create the pipeline or when you use the Create Pipeline wizard.

CodePipeline support for AWS OpsWorks is currently available in the US East (N. Virginia) Region (us-east-1) only.

Learn more:
- Using CodePipeline with AWS OpsWorks Stacks
- Cookbooks and Recipes
- AWS OpsWorks Apps

### AWS Service Catalog

AWS Service Catalog enables organizations to create and manage catalogs of products that are approved for use on AWS.

You can configure CodePipeline to deploy updates and versions of your product templates to AWS Service Catalog. You can create the AWS Service Catalog product to use in a deployment action and then use the Create Pipeline wizard to create the pipeline.

Learn more:
- Tutorial: Create a pipeline that deploys to AWS Service Catalog (p. 87)
- Create a pipeline in CodePipeline (p. 221)

### Alexa Skills Kit

Amazon Alexa Skills Kit lets you build and distribute cloud-based skills to users of Alexa-enabled devices.

You can add an action to a pipeline that uses Alexa Skills Kit as a deployment provider. Source changes are detected by your pipeline, and then your pipeline deploys updates to your Alexa skill in the Alexa service.

Learn more:
- Tutorial: Create a pipeline that deploys an Amazon Alexa skill (p. 131)
### Approval action integrations

<table>
<thead>
<tr>
<th><strong>XebiaLabs</strong></th>
<th>You can configure CodePipeline to use XebiaLabs to deploy your code in one or more actions in a pipeline. <strong>Learn more:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• XebiaLabs documentation for using XL Deploy with CodePipeline</td>
</tr>
</tbody>
</table>

### Amazon Simple Notification Service

Amazon SNS is a fast, flexible, fully managed push notification service that lets you send individual messages or to fan out messages to large numbers of recipients. Amazon SNS makes it simple and cost effective to send push notifications to mobile device users, email recipients or even send messages to other distributed services.

When you create a manual approval request in CodePipeline, you can optionally publish to a topic in Amazon SNS so that all IAM users subscribed to it are notified that the approval action is ready to be reviewed.

**Learn more:**

• What Is Amazon SNS?
• Grant Amazon SNS permissions to a CodePipeline service role (p. 358)

### Invoke action integrations

<table>
<thead>
<tr>
<th><strong>AWS Lambda</strong> <em>(p. 492)</em></th>
<th>Lambda lets you run code without provisioning or managing servers. You can configure CodePipeline to use Lambda functions to add flexibility and functionality to your pipelines. You can create the Lambda function to add as an action in a stage either before you create the pipeline or when you use the <strong>Create Pipeline</strong> wizard. <strong>Learn more:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• CodePipeline Action Configuration Reference for AWS Lambda <em>(p. 492)</em></td>
</tr>
<tr>
<td></td>
<td>• Invoke an AWS Lambda function in a pipeline in CodePipeline <em>(p. 336)</em></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>AWS Step Functions</strong> <em>(p. 499)</em></th>
<th>Step Functions lets you create and configure state machines. You can configure CodePipeline to use Step Functions invoke actions to trigger state machine executions. <strong>Learn more:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• CodePipeline Action Configuration Reference for AWS Step Functions <em>(p. 499)</em></td>
</tr>
<tr>
<td></td>
<td>• Tutorial: Use an AWS Step Functions invoke action in a pipeline <em>(p. 156)</em></td>
</tr>
</tbody>
</table>

### General integrations with CodePipeline

The following AWS service integrations are not based on CodePipeline action types.
<table>
<thead>
<tr>
<th>Integration</th>
<th>Description</th>
<th>Learn more</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AWS CloudTrail</strong></td>
<td>CloudTrail captures AWS API calls and related events made by or on behalf of an AWS account and delivers log files to an Amazon S3 bucket that you specify. You can configure CloudTrail to capture API calls from the CodePipeline console, CodePipeline commands from the AWS CLI, and from the CodePipeline API.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Learn more:</td>
<td>• Logging CodePipeline API calls with AWS CloudTrail (p. 394)</td>
</tr>
<tr>
<td><strong>Amazon CloudWatch</strong></td>
<td>Amazon CloudWatch monitors your AWS resources.</td>
<td>Learn more:</td>
</tr>
<tr>
<td></td>
<td>Learn more:</td>
<td>• What Is Amazon CloudWatch?</td>
</tr>
<tr>
<td><strong>Amazon CloudWatch Events</strong></td>
<td>Amazon CloudWatch Events is a web service that detects changes in your AWS services based on rules that you define and invokes an action in one or more specified AWS services when a change occurs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Learn more:</td>
<td>• Start a pipeline execution automatically when something changes — You can configure CodePipeline as a target in rules set up in Amazon CloudWatch Events. This sets up pipelines to start automatically when another service changes.</td>
</tr>
<tr>
<td></td>
<td>Learn more:</td>
<td>• What Is Amazon CloudWatch Events?</td>
</tr>
<tr>
<td></td>
<td>Learn more:</td>
<td>• Start a pipeline execution in CodePipeline (p. 171).</td>
</tr>
<tr>
<td></td>
<td>Learn more:</td>
<td>• Use CloudWatch Events to start a pipeline (CodeCommit source) (p. 174)</td>
</tr>
<tr>
<td></td>
<td>Learn more:</td>
<td>• Receive notifications when a pipeline state changes — You can set up Amazon CloudWatch Events rules to detect and react to changes in execution state for a pipeline, stage, or action.</td>
</tr>
<tr>
<td></td>
<td>Learn more:</td>
<td>• Detect and react to changes in pipeline state with Amazon CloudWatch Events (p. 384)</td>
</tr>
<tr>
<td></td>
<td>Learn more:</td>
<td>• Tutorial: Set up a CloudWatch Events rule to receive email notifications for pipeline state changes (p. 70)</td>
</tr>
<tr>
<td><strong>AWS CodeStar Notifications</strong></td>
<td>You can set up notifications to make users aware of important changes, such as when a pipeline starts execution. For more information, see Create a notification rule (p. 321).</td>
<td></td>
</tr>
<tr>
<td><strong>AWS Key Management Service</strong></td>
<td>AWS KMS is a managed service that makes it easy for you to create and control the encryption keys used to encrypt your data. By default, CodePipeline uses AWS KMS to encrypt artifacts for pipelines stored in Amazon S3 buckets.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Learn more:</td>
<td>• To create a pipeline that uses a source bucket, artifact bucket, and service role from one AWS account and CodeDeploy resources from a different AWS account, you must create a customer-managed KMS key, add the key to the pipeline, and set up account policies and roles to enable cross-account access. For more information, see Create a pipeline in CodePipeline that uses resources from another AWS account (p. 251).</td>
</tr>
</tbody>
</table>
• To create a pipeline from one AWS account that deploys an AWS CloudFormation stack to another AWS account, you must create a customer-managed KMS key, add the key to the pipeline, and set up account policies and roles to deploy the stack to another AWS account. For more information, see How do I use CodePipeline to deploy an AWS CloudFormation stack in a different account?

• To configure server-side encryption for your pipeline’s S3 artifact bucket, you can use the default AWS managed KMS key or create a customer-managed KMS key and set up the bucket policy to use the encryption key. For more information, see Configure server-side encryption for artifacts stored in Amazon S3 for CodePipeline (p. 405).

For an AWS KMS key, you can use the key ID, the key ARN, or the alias ARN.

**Note**
Aliases are recognized only in the account that created the customer master key (CMK). For cross-account actions, you can only use the key ID or key ARN to identify the key.

---

**Examples from the community**

The following sections provide links to blog posts, articles, and community-provided examples.

**Note**
These links are provided for informational purposes only, and should not be considered either a comprehensive list or an endorsement of the content of the examples. AWS is not responsible for the content or accuracy of external content.

**Topics**

- Integration examples: Blog posts (p. 33)
- Integration examples: Videos (p. 36)

**Integration examples: Blog posts**

- **How to deploy from GitHub to Amazon EC2 with CodePipeline**

  Learn how to set up CodePipeline from scratch to deploy dev, test, and prod branches to separate deployment groups. Learn how to use and configure IAM roles, the CodeDeploy agent, and CodeDeploy, along with CodePipeline.

  *Published April 2020*

- **Implementing DevSecOps Using CodePipeline**

  Learn how to use a CI/CD pipeline in CodePipeline to automate preventive and detective security controls. This post covers how to use a pipeline to create a simple security group and perform security checks during the source, test, and production stages to improve the security posture of your AWS accounts.

  *Published March 2017*

- **Continuous Deployment to Amazon ECS Using CodePipeline, CodeBuild, Amazon ECR, and AWS CloudFormation**
Learn how to create a continuous deployment pipeline to Amazon Elastic Container Service (Amazon ECS). Applications are delivered as Docker containers using CodePipeline, CodeBuild, Amazon ECR, and AWS CloudFormation.

- Download a sample AWS CloudFormation template and instructions for using it to create your own continuous deployment pipeline from the ECS Reference Architecture: Continuous Deployment repo on GitHub.

*Published January 2017*

- **Continuous Deployment for Serverless Applications**

Learn how to use a collection of AWS services to create a continuous deployment pipeline for your serverless applications. You'll use the Serverless Application Model (SAM) to define the application and its resources and CodePipeline to orchestrate your application deployment.

- View a sample application written in Go with the Gin framework and an API Gateway proxy shim.

*Published December 2016*

- **Integrating Git with CodePipeline**

Learn how to integrate CodePipeline with Git servers that support webhooks functionality, such as GitHub Enterprise, Bitbucket, and GitLab.

*Published November 2016*

- **Scaling DevOps Deployments with CodePipeline and Dynatrace**

Learn how use Dynatrace monitoring solutions to scale pipelines in CodePipeline, automatically analyze test executions before code is committed, and maintain optimal lead times.

*Published November 2016*

- **Create a Pipeline for AWS Elastic Beanstalk in CodePipeline Using AWS CloudFormation and CodeCommit**

Learn how to implement continuous delivery in a CodePipeline pipeline for an application in AWS Elastic Beanstalk. All AWS resources are provisioned automatically through the use of an AWS CloudFormation template. This walkthrough also incorporates CodeCommit and AWS Identity and Access Management (IAM).

*Published May 2016*

- **Automate CodeCommit and CodePipeline in AWS CloudFormation**

Use AWS CloudFormation to automate the provisioning of AWS resources for a continuous delivery pipeline that uses CodeCommit, CodePipeline, CodeDeploy, and AWS Identity and Access Management.

*Published April 2016*

- **Create a Cross-Account Pipeline in AWS CodePipeline**

Learn how to automate the provisioning of cross-account access to pipelines in AWS CodePipeline by using AWS Identity and Access Management. Includes examples in an AWS CloudFormation template.

*Published March 2016*

- **Exploring ASP.NET Core Part 2: Continuous Delivery**

Learn how to create a full continuous delivery system for an ASP.NET Core application using CodeDeploy and AWS CodePipeline.

*Published March 2016*

- **Create a Pipeline Using the AWS CodePipeline Console**
Learn how to use the AWS CodePipeline console to create a two-stage pipeline in a walkthrough based on the AWS CodePipeline Tutorial: Create a four-stage pipeline (p. 63).

Published March 2016

• Mocking AWS CodePipeline Pipelines with AWS Lambda

Learn how to invoke a Lambda function that lets you visualize the actions and stages in a CodePipeline software delivery process as you design it, before the pipeline is operational. As you design your pipeline structure, you can use the Lambda function to test whether your pipeline will complete successfully.

Published February 2016

• Running AWS Lambda Functions in CodePipeline Using AWS CloudFormation

Learn how to create an AWS CloudFormation stack that provisions all the AWS resources used in the user guide task Invoke an AWS Lambda function in a pipeline in CodePipeline (p. 336).

Published February 2016

• Provisioning Custom CodePipeline Actions in AWS CloudFormation

Learn how to use AWS CloudFormation to provision custom actions in CodePipeline.

Published January 2016

• Provisioning CodePipeline with AWS CloudFormation

Learn how to provision a basic continuous delivery pipeline in CodePipeline using AWS CloudFormation.

Published December 2015

• Building Continuous Deployment on AWS with CodePipeline, Jenkins, and Elastic Beanstalk

Learn how to use GitHub, CodePipeline, Jenkins, and Elastic Beanstalk to create a deployment pipeline for a web application that is updated automatically every time you change your code.

Published December 2015

• Performance Testing in Continuous Delivery Using CodePipeline and BlazeMeter

Learn how to inject automated load tests at the right places in the CodePipeline delivery workflow with BlazeMeter’s native CodePipeline integration.

Published September 2015

• Deploying from CodePipeline to AWS OpsWorks Using a Custom Action and AWS Lambda

Learn how to configure your pipeline and the AWS Lambda function to deploy to AWS OpsWorks using CodePipeline.

Published July 2015

• Automated Delivery Acceptance Test Nirvana: Powered by CodePipeline, CloudWatch, and BlazeMeter

Learn how to use CodePipeline, CloudWatch, and BlazeMeter to create a continuous delivery workflow that reduces time to release and increases test coverage for developers during the release.

Published July 2015
Integration examples: Videos

- Create a Pipeline Using the CodePipeline Console
  Learn how to use the CodePipeline console to create a pipeline that uses CodeDeploy and Amazon S3.
  
  Create a Pipeline Using the CodePipeline Console

  Published March 2016

  Duration: 8:53
CodePipeline tutorials

After you complete the steps in Getting started with CodePipeline (p. 19), you can try one of the AWS CodePipeline tutorials in this user guide:

<table>
<thead>
<tr>
<th>Task Description</th>
<th>Tutorial</th>
</tr>
</thead>
<tbody>
<tr>
<td>I want to use the wizard to create a pipeline that uses CodeDeploy to deploy a</td>
<td>See Tutorial: Create a simple pipeline (S3 bucket) (p. 38).</td>
</tr>
<tr>
<td>sample application from an Amazon S3 bucket to Amazon EC2 instances running</td>
<td></td>
</tr>
<tr>
<td>Amazon Linux. After using the wizard to create my two-stage pipeline, I want to</td>
<td></td>
</tr>
<tr>
<td>add a third stage.</td>
<td></td>
</tr>
<tr>
<td>I want to create a two-stage pipeline that uses CodeDeploy to deploy a sample</td>
<td>See Tutorial: Create a simple pipeline (CodeCommit repository) (p. 52).</td>
</tr>
<tr>
<td>application from a CodeCommit repository to an Amazon EC2 instance running</td>
<td></td>
</tr>
<tr>
<td>Amazon Linux.</td>
<td></td>
</tr>
<tr>
<td>I want to add a build stage to the three-stage pipeline I created in the first</td>
<td>See Tutorial: Create a four-stage pipeline (p. 63).</td>
</tr>
<tr>
<td>tutorial. The new stage uses Jenkins to build my application.</td>
<td></td>
</tr>
<tr>
<td>I want to set up a CloudWatch Events rule that sends notifications whenever there</td>
<td>See Tutorial: Set up a CloudWatch Events rule to receive email notifications for pipeline</td>
</tr>
<tr>
<td>are changes to the execution state of my pipeline, stage, or action.</td>
<td>state changes (p. 70).</td>
</tr>
<tr>
<td>I want to create a pipeline with a GitHub source that builds and tests an</td>
<td>See Tutorial: Create a pipeline that builds and tests your Android app when a commit is</td>
</tr>
<tr>
<td>Android app with CodeBuild and AWS Device Farm.</td>
<td>pushed to your GitHub repository (p. 73).</td>
</tr>
<tr>
<td>I want to create a pipeline with an Amazon S3 source that tests an iOS app with</td>
<td>See Tutorial: Create a pipeline that tests your iOS app after a change in your S3 bucket</td>
</tr>
<tr>
<td>AWS Device Farm.</td>
<td>(p. 80).</td>
</tr>
<tr>
<td>I want to create a pipeline that deploys my product template to AWS Service</td>
<td>See Tutorial: Create a pipeline that deploys to AWS Service Catalog (p. 87).</td>
</tr>
<tr>
<td>Catalog.</td>
<td></td>
</tr>
<tr>
<td>I want to use sample templates to create a simple pipeline (with an Amazon S3,</td>
<td>See Tutorial: Create a pipeline with AWS CloudFormation (p. 99).</td>
</tr>
<tr>
<td>CodeCommit, or GitHub source) using the AWS CloudFormation console.</td>
<td></td>
</tr>
<tr>
<td>I want to create a two-stage pipeline that uses CodeDeploy and Amazon ECS for</td>
<td>See Tutorial: Create a pipeline with an Amazon ECR source and ECS-to-CodeDeploy deployment</td>
</tr>
<tr>
<td>blue/green deployment of an image from an Amazon ECR repository to an Amazon ECS</td>
<td>(p. 116).</td>
</tr>
<tr>
<td>cluster and service.</td>
<td></td>
</tr>
<tr>
<td>I want to create a pipeline that continuously publishes my serverless application</td>
<td>See Tutorial: Create a pipeline that publishes your serverless application to the AWS</td>
</tr>
<tr>
<td>to the AWS Serverless Application Repository.</td>
<td>Serverless Application Repository (p. 146).</td>
</tr>
</tbody>
</table>

The following tutorials in other user guides provide guidance for integrating other AWS services into your pipelines:
Tutorial: Create a simple pipeline (S3 bucket)

The easiest way to create a pipeline is to use the Create pipeline wizard in the AWS CodePipeline console.

In this tutorial, you create a two-stage pipeline that uses a versioned S3 bucket and CodeDeploy to release a sample application.

Note
When Amazon S3 is the source provider for your pipeline, you may zip your source file or files into a single .zip and upload the .zip to your source bucket. You may also upload a single unzipped file; however, downstream actions that expect a .zip file will fail.

After you create this simple pipeline, you add another stage and then disable and enable the transition between stages.

Important
Many of the actions you add to your pipeline in this procedure involve AWS resources that you need to create before you create the pipeline. AWS resources for your source actions must always be created in the same AWS Region where you create your pipeline. For example, if you create your pipeline in the US East (Ohio) Region, your CodeCommit repository must be in the US East (Ohio) Region.

You can add cross-region actions when you create your pipeline. AWS resources for cross-region actions must be in the same AWS Region where you plan to execute the action. For more information, see Add a cross-Region action in CodePipeline (p. 364).

Before you begin, you should complete the prerequisites in Getting started with CodePipeline (p. 19).

Topics
- Step 1: Create an S3 bucket for your application (p. 38)
- Step 2: Create Amazon EC2 Windows instances and install the CodeDeploy agent (p. 39)
- Step 3: Create an application in CodeDeploy (p. 41)
- Step 4: Create your first pipeline in CodePipeline (p. 42)
- (Optional) Step 5: Add another stage to your pipeline (p. 44)
- (Optional) Step 6: Disable and enable transitions between stages in CodePipeline (p. 51)
- Step 7: Clean up resources (p. 51)

Step 1: Create an S3 bucket for your application

You can store your source files or applications in any versioned location. In this tutorial, you create an S3 bucket for the sample applications and enable versioning on that bucket. After you have enabled versioning, you copy the sample applications to that bucket.

To create an S3 bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. Choose Create bucket.

3. In Bucket name, enter a name for your bucket (for example, awscodelpipeline-demobucket-example-date).
   
   **Note**
   Because all bucket names in Amazon S3 must be unique, use one of your own, not the name shown in the example. You can change the example name just by adding the date to it. Make a note of this name because you need it for the rest of this tutorial.

   In Region, choose the Region where you intend to create your pipeline, such as US West (Oregon), and then choose Create bucket.

4. After the bucket is created, a success banner displays. Choose Go to bucket details.

5. On the Properties tab, choose Versioning. Choose Enable versioning, and then choose Save.
   
   When versioning is enabled, Amazon S3 saves every version of every object in the bucket.

6. On the Permissions tab, leave the defaults. For more information about S3 bucket and object permissions, see Specifying Permissions in a Policy.

7. Next, download a sample and save it into a folder or directory on your local computer.
   
   a. Choose one of the following. Choose SampleApp_Windows.zip if you want to follow the steps in this tutorial for Windows Server instances.
      
      • If you want to deploy to Amazon Linux instances using CodeDeploy, download the sample application here: SampleApp_Linux.zip.
      
      • If you want to deploy to Windows Server instances using CodeDeploy, download the sample application here: SampleApp_Windows.zip.

   b. Download the compressed (zipped) file. Do not unzip the file.

8. In the Amazon S3 console, for your bucket, upload the file:
   
   a. Choose Upload.
   
   b. Drag and drop the file or choose Add files and browse for the file.

Step 2: Create Amazon EC2 Windows instances and install the CodeDeploy agent

**Note**
This tutorial provides sample steps for creating Amazon EC2 Windows instances. For sample steps to create Amazon EC2 Linux instances, see Step 3: Create an EC2 Linux instance and install the CodeDeploy agent (p. 54). When prompted for the number of instances to create, specify 2 instances.

In this step, you create the Windows Server Amazon EC2 instances to which you will deploy a sample application. As part of this process, you install the CodeDeploy agent on the instances. The CodeDeploy agent is a software package that enables an instance to be used in CodeDeploy deployments.

To create an instance role

1. Open the IAM console at https://console.aws.amazon.com/iam/).
2. From the console dashboard, choose Roles.
3. Choose Create role.
4. Under Select type of trusted entity, select AWS service. Under Choose a use case, select EC2, and then choose Next: Permissions.
5. Search for and select the policy named AmazonEC2RoleforAWSCodeDeploy, and then choose Next: Tags.

6. Choose Next: Review. Enter a name for the role (for example, EC2InstanceRole).

   **Note**
   Make a note of your role name for the next step. You choose this role when you are creating your instance.

   Choose Create role.

**To launch instances**

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

2. From the console dashboard, choose Launch instance, and select Launch instance from the options that pop up.

3. On the Step 1: Choose an Amazon Machine Image (AMI) page, locate the Microsoft Windows Server 2019 Base option, and then choose Select. (This AMI is labeled "Free tier eligible" and can be found at the top of the list.)

4. On the Step 2: Choose an Instance Type page, choose the free tier eligible t2.micro type as the hardware configuration for your instance, and then choose Next: Configure Instance Details.

5. On the Step 3: Configure Instance Details page, do the following:

   - In Number of instances, enter 2.
   - In Auto-assign Public IP, choose Enable.
   - In IAM role, choose the IAM role you created in the previous procedure (for example, EC2InstanceRole).
   - Expand Advanced Details, and in User data, with As text selected, enter the following:

   ```powershell
   New-Item -Path c:\temp -ItemType "directory" -Force
   powershell.exe -Command Read-S3Object -BucketName bucket-name/latest -Key codedeploy-agent.msi -File c:\temp\codedeploy-agent.msi
   Start-Process -Wait -FilePath c:\temp\codedeploy-agent.msi -WindowStyle Hidden
   </powershell>
   
   bucket-name is the name of the S3 bucket that contains the CodeDeploy Resource Kit files for your Region. For example, for the US West (Oregon) Region, replace bucket-name with aws-codedeploy-us-west-2. For a list of bucket names, see Resource Kit Bucket Names by Region.

   This code installs the CodeDeploy agent on your instance as it is created. This script is written for Windows instances only.

   - Leave the rest of the items on the Step 3: Configure Instance Details page unchanged. Choose Next: Add Storage.

6. Leave the Step 4: Add Storage page unchanged, and then choose Next: Add Tags.

7. On the Add Tags page, choose Add Tag. Enter Name in the Key field, enter MyCodePipelineDemo in the Value field, and then choose Next: Configure Security Group.

   **Important**
   The Key and Value boxes are case sensitive.

8. On the Configure Security Group page, allow port 80 communication so you can access the public instance endpoint.


10. On the Review Instance Launch page, choose Launch. When prompted for a key pair, choose Proceed without a key pair.

API Version 2015-07-09
40
Create an application in CodeDeploy

**Step 3: Create an application in CodeDeploy**

In CodeDeploy, an *application* is an identifier, in the form of a name, for the code you want to deploy. CodeDeploy uses this name to ensure the correct combination of revision, deployment configuration, and deployment group are referenced during a deployment. You select the name of the CodeDeploy application you create in this step when you create your pipeline later in this tutorial.

**To create an application in CodeDeploy**

2. If the Applications page does not appear, on the AWS CodeDeploy menu, choose Applications.
3. Choose Create application.
4. In Application name, enter MyDemoApplication.
6. Choose Create application.

**To create a deployment group in CodeDeploy**

1. On the page that displays your application, choose Create deployment group.
2. In Deployment group name, enter MyDemoDeploymentGroup.
3. In Service Role, choose a service role that trusts AWS CodeDeploy with, at minimum, the trust and permissions described in Create a Service Role for CodeDeploy. To get the service role ARN, see Get the Service Role ARN (Console).
4. Under Deployment type, choose In-place.
5. Under Environment configuration, choose Amazon EC2 Instances. Choose Name in the Key field, and in the Value field, enter MyCodePipelineDemo.
   
   **Important**
   
   You must choose the same value for the Name key here that you assigned to your EC2 instances when you created them. If you tagged your instances with something other than MyCodePipelineDemo, be sure to use it here.
7. Under Load Balancer, clear Enable load balancing. You do not need to set up a load balancer or choose a target group for this example.
8. In the Advanced section, leave the defaults.
9. Choose Create deployment group.
Step 4: Create your first pipeline in CodePipeline

In this part of the tutorial, you create the pipeline. The sample runs automatically through the pipeline.

To create a CodePipeline automated release process

2. On the Welcome page, Getting started page, or the Pipelines page, choose Create pipeline.
3. In Step 1: Choose pipeline settings, in Pipeline name, enter MyFirstPipeline.

   Note
   If you choose another name for your pipeline, be sure to use that name instead of MyFirstPipeline for the rest of this tutorial. After you create a pipeline, you cannot change its name. Pipeline names are subject to some limitations. For more information, see Quotas in AWS CodePipeline (p. 521).

4. In Service role, do one of the following:
   - Choose New service role to allow CodePipeline to create a new service role in IAM. In Role name, the role and policy name both default to this format: AWSCodePipelineServiceRole-region-pipeline_name. For example, this is the service role created for this tutorial: AWSCodePipelineServiceRole-eu-west-2-MyFirstPipeline.
   - Choose Existing service role to use a service role already created in IAM. In Role name, choose your service role from the list.
5. Leave the settings under Advanced settings at their defaults, and then choose Next.
6. In Step 2: Add source stage, in Source provider, choose Amazon S3. In Bucket, enter the name of the S3 bucket you created in Step 1: Create an S3 bucket for your application (p. 38). In S3 object key, enter the object key with or without a file path, and remember to include the file extension. For example, for SampleApp_Windows.zip, enter the sample file name as shown in this example:

   SampleApp_Windows.zip

Choose Next step.
Under **Change detection options**, leave the defaults. This allows CodePipeline to use Amazon CloudWatch Events to detect changes in your source bucket.

Choose **Next**.

7. In **Step 3: Add build stage**, choose **Skip build stage**, and then accept the warning message by choosing **Skip** again. Choose **Next**.

8. In **Step 4: Add deploy stage**, in **Deploy provider**, choose **AWS CodeDeploy**. The **Region** field defaults to the same AWS Region as your pipeline. In **Application name**, enter **MyDemoApplication**, or choose the **Refresh** button, and then choose the application name from the list. In **Deployment group**, enter **MyDemoDeploymentGroup**, or choose it from the list, and then choose **Next**.

   ![](image)

   **Note**
   The name Deploy is the name given by default to the stage created in the **Step 4: Add deploy stage** step, just as Source is the name given to the first stage of the pipeline.

9. In **Step 5: Review**, review the information, and then choose **Create pipeline**.

10. The pipeline starts to run. You can view progress and success and failure messages as the CodePipeline sample deploys a webpage to each of the Amazon EC2 instances in the CodeDeploy deployment.

   Congratulations! You just created a simple pipeline in CodePipeline. The pipeline has two stages:

   - A source stage named **Source**, which detects changes in the versioned sample application stored in the S3 bucket and pulls those changes into the pipeline.
   - A **Deploy** stage that deploys those changes to EC2 instances with CodeDeploy.
Now, verify the results.

**To verify your pipeline ran successfully**

1. View the initial progress of the pipeline. The status of each stage changes from **No executions yet** to **In Progress**, and then to either **Succeeded** or **Failed**. The pipeline should complete the first run within a few minutes.

2. After **Succeeded** is displayed for the action status, in the status area for the **Deploy** stage, choose **Details**. This opens the AWS CodeDeploy console.

3. In the **Deployment group** tab, under **Deployment lifecycle events**, choose an instance ID. This opens the EC2 console.

4. On the **Description** tab, in **Public DNS**, copy the address, and then paste it into the address bar of your web browser. View the index page for the sample application you uploaded to your S3 bucket.

The following page is the sample application you uploaded to your S3 bucket.

Congratulations!

You have successfully created a pipeline that retrieved this source application from an Amazon S3 bucket and deployed it to two Amazon EC2 instances using AWS CodeDeploy.

For next steps, read the AWS CodePipeline Documentation.

For more information about stages, actions, and how pipelines work, see CodePipeline concepts (p. 3).

**(Optional) Step 5: Add another stage to your pipeline**

Now add another stage in the pipeline to deploy from staging servers to production servers using CodeDeploy. First, you create another deployment group in the CodePipelineDemoApplication in CodeDeploy. Then you add a stage that includes an action that uses this deployment group. To add another stage, you use the CodePipeline console or the AWS CLI to retrieve and manually edit the structure of the pipeline in a JSON file, and then run the `update-pipeline` command to update the pipeline with your changes.

**Topics**

- Create a second deployment group in CodeDeploy (p. 45)
- Add the deployment group as another stage in your pipeline (p. 45)
Create a second deployment group in CodeDeploy

**Note**
In this part of the tutorial, you create a second deployment group, but deploy to the same Amazon EC2 instances as before. This is for demonstration purposes only. It is purposely designed to fail to show you how errors are displayed in CodePipeline.

**To create a second deployment group in CodeDeploy**

2. Choose Applications, and in the list of applications, choose MyDemoApplication.
3. Choose the Deployment groups tab, and then choose Create deployment group.
4. On the Create deployment group page, in Deployment group name, enter a name for the second deployment group (for example, CodePipelineProductionFleet).
5. In Service Role, choose the same CodeDeploy service role you used for the initial deployment (not the CodePipeline service role).
6. Under Deployment type, choose In-place.
7. Under Environment configuration, choose Amazon EC2 Instances. Choose Name in the Key box, and in the Value box, choose MyCodePipelineDemo from the list. Leave the default configuration for Deployment settings.
10. Choose Create deployment group.

**Add the deployment group as another stage in your pipeline**

Now that you have another deployment group, you can add a stage that uses this deployment group to deploy to the same EC2 instances you used earlier. You can use the CodePipeline console or the AWS CLI to add this stage.

**Topics**
- Create a third stage (console) (p. 45)
- Create a third stage (CLI) (p. 48)

**Create a third stage (console)**

You can use the CodePipeline console to add a new stage that uses the new deployment group. Because this deployment group is deploying to the EC2 instances you’ve already used, the deploy action in this stage fails.

2. In Name, choose the name of the pipeline you created, MyFirstPipeline.
3. On the pipeline details page, choose Edit.
4. On the Edit page, choose + Add stage to add a stage immediately after the Deploy stage.
5. In **Add stage**, in **Stage name**, enter **Production**. Choose **Add stage**.

6. In the new stage, choose **+ Add action group**.

7. In **Edit action**, in **Action name**, enter **Deploy-Second-Deployment**. In **Action provider**, under **Deploy**, choose **AWS CodeDeploy**.

8. In the CodeDeploy section, in **Application name**, choose **MyDemoApplication** from the drop-down list, as you did when you created the pipeline. In **Deployment group**, choose the deployment group you just created, **CodePipelineProductionFleet**. In **Input artifacts**, choose the input artifact from the source action. Choose **Save**.

9. On the **Edit** page, choose **Save**. In **Save pipeline changes**, choose **Save**.

10. Although the new stage has been added to your pipeline, a status of **No executions yet** is displayed because no changes have triggered another run of the pipeline. You must manually rerun the last revision to see how the edited pipeline runs. On the pipeline details page, choose **Release change**, and then choose **Release** when prompted. This runs the most recent revision available in each source location specified in a source action through the pipeline.

    Alternatively, to use the AWS CLI to rerun the pipeline, from a terminal on your local Linux, macOS, or Unix machine, or a command prompt on your local Windows machine, run the **start-pipeline-execution** command, specifying the name of the pipeline. This runs the application in your source bucket through the pipeline for a second time.

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

This command returns a **pipelineExecutionId** object.

11. Return to the CodePipeline console and in the list of pipelines, choose **MyFirstPipeline** to open the view page.

    The pipeline shows three stages and the state of the artifact running through those three stages. It might take up to five minutes for the pipeline to run through all stages. You see the deployment
succeeds on the first two stages, just as before, but the Production stage shows the Deploy-Second-Deployment action failed.

12. In the Deploy-Second-Deployment action, choose Details. You are redirected to the page for the CodeDeploy deployment. In this case, the failure is the result of the first instance group deploying to all of the EC2 instances, leaving no instances for the second deployment group.

Note
This failure is by design, to demonstrate what happens when there is a failure in a pipeline stage.
**Create a third stage (CLI)**

Although using the AWS CLI to add a stage to your pipeline is more complex than using the console, it provides more visibility into the structure of the pipeline.

**To create a third stage for your pipeline**

1. Open a terminal session on your local Linux, macOS, or Unix machine, or a command prompt on your local Windows machine, and run the `get-pipeline` command to display the structure of the pipeline you just created. For `MyFirstPipeline`, you would type the following command:

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

   This command returns the structure of MyFirstPipeline. The first part of the output should look similar to the following:

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

   The final part of the output includes the pipeline metadata and should look similar to the following:

   ```json
   ...
   ],   
   "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. Copy and paste this structure into a plain-text editor, and save the file as `pipeline.json`. For convenience, save this file in the same directory where you run the `aws codepipeline` commands.

   **Note**
   You can pipe the JSON directly into a file with the `get-pipeline` command as follows:

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

3. Copy the `Deploy` stage section and paste it after the first two stages. Because it is a deploy stage, just like the `Deploy` stage, you use it as a template for the third stage.

4. Change the name of the stage and the deployment group details.

   The following example shows the JSON you add to the pipeline.json file after the `Deploy` stage. Edit the emphasized elements with new values. Remember to include a comma to separate the `Deploy` and `Production` stage definitions.

   ```json
   ,
   {   
     "name": "Production",
   }
   ```
5. If you are working with the pipeline structure retrieved using the `get-pipeline` command, you must remove the metadata lines from the JSON file. Otherwise, the `update-pipeline` command cannot use it. Remove the "metadata": { } lines and the "created", "pipelineARN", and "updated" fields.

For example, remove the following lines from the structure:

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

Save the file.

6. Run the `update-pipeline` command, specifying the pipeline JSON file, similar to the following:

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

This command returns the entire structure of the updated pipeline.

**Important**

Be sure to include `file://` before the file name. It is required in this command.

7. Run the `start-pipeline-execution` command, specifying the name of the pipeline. This runs the application in your source bucket through the pipeline for a second time.

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

This command returns a `pipelineExecutionId` object.

8. Open the CodePipeline console and choose MyFirstPipeline from the list of pipelines.

The pipeline shows three stages and the state of the artifact running through those three stages. It might take up to five minutes for the pipeline to run through all stages. Although the deployment succeeds on the first two stages, just as before, the Production stage shows that the Deploy-Second-Deployment action failed.
9. In the **Deploy-Second-Deployment** action, choose **Details** to see details of the failure. You are redirected to the details page for the CodeDeploy deployment. In this case, the failure is the result of the first instance group deploying to all of the EC2 instances, leaving no instances for the second deployment group.

**Note**
This failure is by design, to demonstrate what happens when there is a failure in a pipeline stage.
(Optional) Step 6: Disable and enable transitions between stages in CodePipeline

You can enable or disable the transition between stages in a pipeline. Disabling the transition between stages allows you to manually control transitions between one stage and another. For example, you might want to run the first two stages of a pipeline, but disable transitions to the third stage until you are ready to deploy to production, or while you troubleshoot a problem or failure with that stage.

To disable and enable transitions between stages in a CodePipeline pipeline

1. Open the CodePipeline console and choose MyFirstPipeline from the list of pipelines.
2. On the details page for the pipeline, choose the Disable transition button between the second stage (Deploy) and the third stage that you added in the previous section (Production).
3. In Disable transition, enter a reason for disabling the transition between the stages, and then choose Disable.

The arrow between stages displays an icon and color change, and the Enable transition button.

4. Upload your sample again to the S3 bucket. Because the bucket is versioned, this change starts the pipeline. For information, see Upload the sample application (p. 39).
5. Return to the details page for your pipeline and watch the status of the stages. The pipeline view changes to show progress and success on the first two stages, but no changes occur on the third stage. This process might take a few minutes.
6. Enable the transition by choosing the Enable transition button between the two stages. In the Enable transition dialog box, choose Enable. The stage starts running in a few minutes and attempts to process the artifact that has already been run through the first two stages of the pipeline.

   Note
   If you want this third stage to succeed, edit the CodePipelineProductionFleet deployment group before you enable the transition, and specify a different set of EC2 instances where the application is deployed. For more information about how to do this, see Change deployment group settings. If you create more EC2 instances, you might incur additional costs.

Step 7: Clean up resources

You can use some of the resources you created in this tutorial for the Tutorial: Create a four-stage pipeline (p. 63). For example, you can reuse the CodeDeploy application and deployment. You can configure a build action with a provider such as CodeBuild, which is a fully managed build service in the cloud. You can also configure a build action that uses a provider with a build server or system, such as Jenkins.
However, after you complete this and any other tutorials, you should delete the pipeline and the resources it uses, so that you are not charged for the continued use of those resources. First, delete the pipeline, then the CodeDeploy application and its associated EC2 instances, and finally, the S3 bucket.

To clean up the resources used in this tutorial

1. To clean up your CodePipeline resources, follow the instructions in Delete a pipeline in AWS CodePipeline (p. 250).
2. To clean up your CodeDeploy resources, follow the instructions in To clean up resources (console).
3. To delete the S3 bucket, follow the instructions in Deleting or emptying a bucket. If you do not intend to create more pipelines, delete the S3 bucket created for storing your pipeline artifacts. For more information about this bucket, see CodePipeline concepts (p. 3).

Tutorial: Create a simple pipeline (CodeCommit repository)

In this tutorial, you use CodePipeline to deploy code maintained in a CodeCommit repository to a single Amazon EC2 instance. Your pipeline is triggered when you push a change to the CodeCommit repository. The pipeline deploys your changes to an Amazon EC2 instance using CodeDeploy as the deployment service.

The pipeline has two stages:

- A source stage (Source) for your CodeCommit source action.
- A deployment stage (Deploy) for your CodeDeploy deployment action.

The easiest way to get started with AWS CodePipeline is to use the Create Pipeline wizard in the CodePipeline console.

Note
Before you begin, make sure you've set up your Git client to work with CodeCommit. For instructions, see Setting up for CodeCommit.

Step 1: Create a CodeCommit repository

First, you create a repository in CodeCommit. Your pipeline gets source code from this repository when it runs. You also create a local repository where you maintain and update code before you push it to the CodeCommit repository.

To create a CodeCommit repository

2. In the Region selector, choose the AWS Region where you want to create the repository and pipeline. For more information, see AWS Regions and Endpoints.
3. On the Repositories page, choose Create repository.
4. On the Create repository page, in Repository name, enter a name for your repository (for example, MyDemoRepo).
5. Choose Create.

Note
The remaining steps in this tutorial use MyDemoRepo for the name of your CodeCommit repository. If you choose a different name, be sure to use it throughout this tutorial.
To set up a local repository

In this step, you set up a local repository to connect to your remote CodeCommit repository.

1. With your new repository open in the console, choose **Clone URL** on the top right of the page, and then choose **Clone SSH**. The address to clone your Git repository is copied to your clipboard.
2. In your terminal or command line, navigate to a local directory where you’d like your local repository to be stored. In this tutorial, we use `/tmp`.
3. Run the following command to clone the repository, replacing the SSH address with the one you copied in the previous step. This command creates a directory called `MyDemoRepo`. You copy a sample application to this directory.

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

Step 2: Add sample code to your CodeCommit repository

In this step, you download code for a sample application that was created for a CodeDeploy sample walkthrough, and add it to your CodeCommit repository.

1. Download the following file: `SampleApp_LINUX.zip`
2. Unzip the files from `SampleApp_LINUX.zip` into the local directory you created earlier (for example, `/tmp/MyDemoRepo` or `c:\temp\MyDemoRepo`).

   Be sure to place the files directly into your local repository. Do not include a `SampleApp_LINUX` folder. On your local Linux, macOS, or Unix machine, for example, your directory and file hierarchy should look like this:

   ```
   /tmp
   #-- MyDemoRepo
   #-- appspec.yml
   #-- index.html
   #-- LICENSE.txt
   #-- scripts
   #-- install_dependencies
   #-- start_server
   #-- stop_server
   ```
3. Change directories to your local repo:

   ```
   (For Linux, macOS, or Unix) cd /tmp/MyDemoRepo
   (For Windows) cd c:\temp\MyDemoRepo
   ```
4. Run the following command to stage all of your files at once:

   ```
   git add -A
   ```
5. Run the following command to commit the files with a commit message:

   ```
   git commit -m "Add sample application files"
   ```
6. Run the following command to push the files from your local repo to your CodeCommit repository:

   ```
   git push
   ```
7. The files you downloaded and added to your local repo have now been added to the master branch in your CodeCommit MyDemoRepo repository and are ready to be included in a pipeline.

**Step 3: Create an EC2 Linux instance and install the CodeDeploy agent**

In this step, you create the EC2 instance where you deploy a sample application. As part of this process, you install the CodeDeploy agent on the EC2 instance. The CodeDeploy agent is a software package that enables an instance to be used in CodeDeploy deployments. You also attach an IAM role to the instance (known as an instance role) to allow it to fetch files that the CodeDeploy agent uses to deploy your application.

**To create an instance role**

1. Open the IAM console at https://console.aws.amazon.com/iam/).
2. From the console dashboard, choose Roles.
3. Choose Create role.
4. Under Select type of trusted entity, select AWS service. Under Choose a use case, select EC2, and then choose Next: Permissions.
5. Search for and select the policy named AmazonEC2RoleforAWSCodeDeploy, and then choose Next: Tags.
6. Choose Next: Review. Enter a name for the role (for example, EC2InstanceRole).
   
   **Note**
   
   Make a note of your role name for the next step. You choose this role when you are creating your instance.

   Choose Create role.

**To launch an instance**

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. From the console dashboard, choose Launch instance, and select Launch instance from the options that pop up.
3. On Step 1: Choose an Amazon Machine Image (AMI), locate Amazon Linux 2 AMI (HVM), SSD Volume Type, and then choose Select. (This AMI is labeled "Free tier eligible" and can be found at the top of the list.)
4. On the Step 2: Choose an Instance Type page, choose the free tier eligible t2.micro type as the hardware configuration for your instance, and then choose Next: Configure Instance Details.
5. On the Step 3: Configure Instance Details page, do the following:
   
   • In Number of instances, enter 1.
   • In Auto-assign Public IP, choose Enable.
   • In IAM role, choose the IAM role you created in the previous procedure (for example, EC2InstanceRole).
   • Expand Advanced Details, and in the User data field, enter the following:

   ```bash
   #!/bin/bash
   yum -y update
   yum install -y ruby
   yum install -y aws-cli
   cd /home/ec2-user
   ```

   API Version 2015-07-09
aws s3 cp s3://aws-codedeploy-us-east-2/latest/install . --region us-east-2
chmod +x ./install
./install auto

Note
For an example that runs these commands with elevated privileges (sudo commands), see the CodeDeploy agent reference in Install or reinstall the CodeDeploy agent for Amazon Linux or RHEL in the AWS CodeDeploy User Guide.

This code installs the CodeDeploy agent on your instance as it is created.

• Leave the rest of the items on the Step 3: Configure Instance Details page unchanged. Choose Next: Add Storage.

6. Leave the Step 4: Add Storage page unchanged, and then choose Next: Add Tags.

7. Choose Add Tag. In Key, enter Name, and in Value, enter MyCodePipelineDemo. Choose Next: Configure Security Group. Later, you create a CodeDeploy application that deploys the sample application to this instance. CodeDeploy selects instances to deploy based on the tags that are attached to instances.

8. On the Step 6: Configure Security Group page, do the following:

• Next to Assign a security group, choose Create a new security group.
• In the row for SSH, under Source, choose My IP.
• Choose Add Rule, choose HTTP, and then under Source, choose My IP.


10. On the Review Instance Launch page, choose Launch. When prompted for a key pair, choose Proceed without a key pair.

Note
For the purposes of this tutorial, you can proceed without a key pair. To use SSH to connect to your instances, create or use a key pair.

When you are ready, select the acknowledgment check box, and then choose Launch Instances.

11. Choose View Instances to close the confirmation page and return to the console.

12. You can view the status of the launch on the Instances page. When you launch an instance, its initial state is pending. After the instance starts, its state changes to running, and it receives a public DNS name. (If the Public DNS column is not displayed, choose the Show/Hide icon, and then select Public DNS.)

13. It can take a few minutes for the instance to be ready for you to connect to it. View the information in the Status Checks column to see if your instance has passed its status checks.

Step 4: Create an application in CodeDeploy

In CodeDeploy, an application is a resource that contains the software application you want to deploy. Later, you use this application with CodePipeline to automate deployments of the sample application to your Amazon EC2 instance.

First, you create a role that allows CodeDeploy to perform deployments. Then, you create a CodeDeploy application.

To create a CodeDeploy service role

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. From the console dashboard, choose Roles.
3. Choose Create role.
4. Under **Select type of trusted entity**, select **AWS service**. Under **Choose a use case**, select **CodeDeploy**, and then choose **Next: Permissions**. The AWSCodeDeployRole managed policy is already attached to the role.

5. Choose **Next: Tags**, and **Next: Review**.

6. Enter a name for the role (for example, **CodeDeployRole**), and then choose **Create role**.

**To create an application in CodeDeploy**

2. If the **Applications** page does not appear, on the AWS CodeDeploy menu, choose **Applications**.
3. Choose **Create application**.
4. In **Application name**, enter **MyDemoApplication**.
5. In **Compute Platform**, choose **EC2/On-premises**.
6. Choose **Create application**.

**To create a deployment group in CodeDeploy**

A **deployment group** is a resource that defines deployment-related settings like which instances to deploy to and how fast to deploy them.

1. On the page that displays your application, choose **Create deployment group**.
2. In **Deployment group name**, enter **MyDemoDeploymentGroup**.
3. In **Service Role**, choose the service role you created earlier (for example, **CodeDeployRole**).
4. Under **Deployment type**, choose **In-place**.
5. Under **Environment configuration**, choose **Amazon EC2 Instances**. In the **Key** field, enter the name you used to tag the instance (for example, **MyCodePipelineDemo**).
6. Under **Deployment configuration**, choose **CodeDeployDefault.OneAtaTime**.
7. Under **Load Balancer**, clear **Enable load balancing**. You do not need to set up a load balancer or choose a target group for this example.
8. Expand the **Advanced** section. Under **Alarms**, if any alarms are listed, choose **Ignore alarm configuration**.
9. Choose **Create deployment group**.

**Step 5: Create your first pipeline in CodePipeline**

You're now ready to create and run your first pipeline. In this step, you create a pipeline that runs automatically when code is pushed to your CodeCommit repository.

**To create a CodePipeline pipeline**

   
2. Choose **Create pipeline**.
3. In **Step 1: Choose pipeline settings**, in **Pipeline name**, enter **MyFirstPipeline**.
4. In **Service role**, choose **New service role** to allow CodePipeline to create a service role in IAM.
5. Leave the settings under **Advanced settings** at their defaults, and then choose **Next**.
6. In **Step 2: Add source stage**, in **Source provider**, choose **AWS CodeCommit**. In **Repository name**, choose the name of the CodeCommit repository you created in **Step 1: Create a CodeCommit repository** (p. 52). In **Branch name**, choose **master**, and then choose **Next step**.

After you select the repository name and branch, a message displays the Amazon CloudWatch Events rule to be created for this pipeline.

Under **Change detection options**, leave the defaults. This allows CodePipeline to use Amazon CloudWatch Events to detect changes in your source repository.

Choose **Next**.

7. In **Step 3: Add build stage**, choose **Skip build stage**, and then accept the warning message by choosing **Skip** again. Choose **Next**.

   **Note**
   In this tutorial, you are deploying code that requires no build service, so you can skip this step. However, if your source code needs to be built before it is deployed to instances, you can configure **CodeBuild** in this step.

8. In **Step 4: Add deploy stage**, in **Deploy provider**, choose **AWS CodeDeploy**. In **Application name**, choose **MyDemoApplication**. In **Deployment group**, choose **MyDemoDeploymentGroup**, and then choose **Next step**.
9. In Step 5: Review, review the information, and then choose Create pipeline.

10. The pipeline starts running after it is created. It downloads the code from your CodeCommit repository and creates a CodeDeploy deployment to your EC2 instance. You can view progress and success and failure messages as the CodePipeline sample deploys the webpage to the Amazon EC2 instance in the CodeDeploy deployment.
Congratulations! You just created a simple pipeline in CodePipeline.

Next, you verify the results.

**To verify that your pipeline ran successfully**

1. View the initial progress of the pipeline. The status of each stage changes from **No executions yet** to **In Progress**, and then to either **Succeeded** or **Failed**. The pipeline should complete the first run within a few minutes.

2. After **Succeeded** is displayed for the pipeline status, in the status area for the **Deploy** stage, choose **AWS CodeDeploy**. This opens the CodeDeploy console. If **Succeeded** is not displayed see Troubleshooting CodePipeline (p. 397).

3. On the **Deployments** tab, choose the deployment ID. On the page for the deployment, under **Deployment lifecycle events**, choose the instance ID. This opens the EC2 console.

4. On the **Description** tab, in **Public DNS**, copy the address (for example, ec2-192-0-2-1.us-west-2.compute.amazonaws.com), and then paste it into the address bar of your web browser.

   This is the sample application you downloaded and pushed to your CodeCommit repository.
Step 6: Modify code in your CodeCommit repository

Your pipeline is configured to run whenever code changes are made to your CodeCommit repository. In this step, you make changes to the HTML file that is part of the sample CodeDeploy application in the CodeCommit repository. When you push these changes, your pipeline runs again, and the changes you make are visible at the web address you accessed earlier.

1. Change directories to your local repo:

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

2. Use a text editor to modify the index.html file:

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

3. Revise the contents of the index.html file to change the background color and some of the text on the webpage, and then save the file.

   ```html
   <!DOCTYPE html>
   <html>
   <head>
   <title>Updated Sample Deployment</title>
   <style>
   body {
     color: #000000;
     background-color: #CCFFCC;
     font-family: Arial, sans-serif;
     font-size:14px;
   }
   ```
4. Commit and push your changes to your CodeCommit repository by running the following commands, one at a time:

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

```bash
git push
```

**To verify your pipeline ran successfully**

1. View the initial progress of the pipeline. The status of each stage changes from **No executions yet** to **In Progress**, and then to either **Succeeded** or **Failed**. The running of the pipeline should be complete within a few minutes.

2. After **Succeeded** is displayed for the action status, refresh the demo page you accessed earlier in your browser.

   The updated webpage is displayed:
Step 7: Clean up resources

You can use some of the resources you created in this tutorial for other tutorials in this guide. For example, you can reuse the CodeDeploy application and deployment. However, after you complete this and any other tutorials, you should delete the pipeline and the resources it uses so that you are not charged for the continued use of those resources. First, delete the pipeline, then the CodeDeploy application and its associated Amazon EC2 instance, and finally, the CodeCommit repository.

To clean up the resources used in this tutorial

1. To clean up your CodePipeline resources, follow the instructions in Delete a pipeline in AWS CodePipeline (p. 250).
2. To clean up your CodeDeploy resources, follow the instructions in Clean Up Deployment Walkthrough Resources.
3. To delete the CodeCommit repository, follow the instructions in Delete a CodeCommit repository.

Step 8: Further reading

Learn more about how CodePipeline works:

- For more information about stages, actions, and how pipelines work, see CodePipeline concepts (p. 3).
- For information about the actions you can perform using CodePipeline, see Integrations with CodePipeline action types (p. 22).
- Try this more advanced tutorial, Tutorial: Create a four-stage pipeline (p. 63). It creates a multi-stage pipeline that includes a step that builds code before it's deployed.
Tutorial: Create a four-stage pipeline

Now that you've created your first pipeline in Tutorial: Create a simple pipeline (S3 bucket) (p. 38) or Tutorial: Create a simple pipeline (CodeCommit repository) (p. 52), you can start creating more complex pipelines. This tutorial will walk you through the creation of a four-stage pipeline that uses a GitHub repository for your source, a Jenkins build server to build the project, and a CodeDeploy application to deploy the built code to a staging server. After the pipeline is created, you will edit it to add a stage with a test action to test the code, also using Jenkins.

Before you can create this pipeline, you must configure the required resources. For example, if you want to use a GitHub repository for your source code, you must create the repository before you can add it to a pipeline. As part of setting up, this tutorial walks you through setting up Jenkins on an EC2 instance for demonstration purposes.

**Important**
Many of the actions you add to your pipeline in this procedure involve AWS resources that you need to create before you create the pipeline. AWS resources for your source actions must always be created in the same AWS Region where you create your pipeline. For example, if you create your pipeline in the US East (Ohio) Region, your CodeCommit repository must be in the US East (Ohio) Region.
You can add cross-region actions when you create your pipeline. AWS resources for cross-region actions must be in the same AWS Region where you plan to execute the action. For more information, see Add a cross-Region action in CodePipeline (p. 364).

Before you begin this tutorial, you should have already completed the general prerequisites in Getting started with CodePipeline (p. 19).

**Topics**
- Step 1: Complete prerequisites (p. 63)
- Step 2: Create a pipeline in CodePipeline (p. 66)
- Step 3: Add another stage to your pipeline (p. 67)
- Step 4: Clean up resources (p. 69)

**Step 1: Complete prerequisites**

To integrate with Jenkins, AWS CodePipeline requires you to install the CodePipeline Plugin for Jenkins on any instance of Jenkins you want to use with CodePipeline. You should also configure a dedicated IAM user to use for permissions between your Jenkins project and CodePipeline. The easiest way to integrate Jenkins and CodePipeline is to install Jenkins on an EC2 instance that uses an IAM instance role that you create for Jenkins integration. In order for links in the pipeline for Jenkins actions to successfully connect, you must configure proxy and firewall settings on the server or EC2 instance to allow inbound connections to the port used by your Jenkins project. Make sure you have configured Jenkins to authenticate users and enforce access control before you allow connections on those ports (for example, 443 and 8443 if you have secured Jenkins to only use HTTPS connections, or 80 and 8080 if you allow HTTP connections). For more information, see Securing Jenkins.

**Note**
This tutorial uses a code sample and configures build steps that convert the sample from Haml to HTML. You can download the open-source sample code from the GitHub repository by following the steps in Copy or clone the sample into a GitHub repository (p. 64). You will need the entire sample in your GitHub repository, not just the .zip file. This tutorial also assumes that:

- You are familiar with installing and administering Jenkins and creating Jenkins projects.
• You have installed Rake and the Haml gem for Ruby on the same computer or instance that hosts your Jenkins project.
• You have set the required system environment variables so that Rake commands can be run from the terminal or command line (for example, on Windows systems, modifying the PATH variable to include the directory where you installed Rake).

Topics
• Copy or clone the sample into a GitHub repository (p. 64)
• Create an IAM role to use for Jenkins integration (p. 64)
• Install and configure Jenkins and the CodePipeline Plugin for Jenkins (p. 65)

Copy or clone the sample into a GitHub repository

To clone the sample and push to a GitHub repository
1. Download the sample code from the GitHub repository, or clone the repositories to your local computer. There are two sample packages:
   • If you will be deploying your sample to Amazon Linux, RHEL, or Ubuntu Server instances, choose `aws-codepipeline-jenkins-aws-codedeploy_linux.zip`.
   • If you will be deploying your sample to Windows Server instances, choose `AWSCodePipeline-Jenkins-AWSCodeDeploy_Windows.zip`.
2. From the repository, choose Fork to clone the sample repo into a repo in your Github account. For more information, see the GitHub documentation.

Create an IAM role to use for Jenkins integration

As a best practice, consider launching an EC2 instance to host your Jenkins server and using an IAM role to grant the instance the required permissions for interacting with CodePipeline.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the IAM console, in the navigation pane, choose Roles, and then choose Create role.
3. Under Select type of trusted entity, choose AWS service. Under Choose the service that will use this role, choose EC2. Under Select your use case, choose EC2.
5. On the Review page, in Role name, enter the name of the role to create specifically for Jenkins integration (for example, JenkinsAccess), and then choose Create role.

When you create the EC2 instance where you will install Jenkins, in Step 3: Configure Instance Details, make sure you choose the instance role (for example, JenkinsAccess).

For more information about instance roles and Amazon EC2, see IAM roles for Amazon EC2, Using IAM Roles to Grant Permissions to Applications Running on Amazon EC2 Instances, and Creating a Role to Delegate Permissions to an AWS Service.
Install and configure Jenkins and the CodePipeline Plugin for Jenkins

To install Jenkins and the CodePipeline Plugin for Jenkins

1. Create an EC2 instance where you will install Jenkins, and in Step 3: Configure Instance Details, make sure you choose the instance role you created (for example, JenkinsAccess). For more information about creating EC2 instances, see Launch an Amazon EC2 instance in the Amazon EC2 User Guide.

   **Note**
   If you already have Jenkins resources you want to use, you can do so, but you must create a special IAM user, apply the AWSCodePipelineCustomActionAccess managed policy to that user, and then configure and use the access credentials for that user on your Jenkins resource. If you want to use the Jenkins UI to supply the credentials, configure Jenkins to only allow HTTPS. For more information, see Troubleshooting CodePipeline (p. 397).

2. Install Jenkins on the EC2 instance. For more information, see the Jenkins documentation for installing Jenkins and starting and accessing Jenkins, as well as details of integration with Jenkins (p. 25) in Product and service integrations with CodePipeline (p. 22).

3. Launch Jenkins, and on the home page, choose Manage Jenkins.

4. On the Manage Jenkins page, choose Manage Plugins.

5. Choose the Available tab, and in the Filter search box, enter AWS CodePipeline. Choose CodePipeline Plugin for Jenkins from the list and choose Download now and install after restart.

6. On the Installing Plugins/Upgrades page, select Restart Jenkins when installation is complete and no jobs are running.

7. Choose Back to Dashboard.

8. On the main page, choose New Item.

9. In Item Name, enter a name for the Jenkins project (for example, MyDemoProject). Choose Freestyle project, and then choose OK.

   **Note**
   Make sure that the name for your project meets the requirements for CodePipeline. For more information, see Quotas in AWS CodePipeline (p. 521).

10. On the configuration page for the project, select the Execute concurrent builds if necessary check box. In Source Code Management, choose AWS CodePipeline. If you have installed Jenkins on an EC2 instance and configured the AWS CLI with the profile for the IAM user you created for integration between CodePipeline and Jenkins, leave all of the other fields empty.

11. Choose Advanced, and in Provider, enter a name for the provider of the action as it will appear in CodePipeline (for example, MyJenkinsProviderName). Make sure that this name is unique and easy to remember. You will use it when you add a build action to your pipeline later in this tutorial, and again when you add a test action.

   **Note**
   This action name must meet the naming requirements for actions in CodePipeline. For more information, see Quotas in AWS CodePipeline (p. 521).

12. In Build Triggers, clear any check boxes, and then select Poll SCM. In Schedule, enter five asterisks separated by spaces, as follows:

```
* * * * *
```

This polls CodePipeline every minute.

13. In Build, choose Add build step. Choose Execute shell (Amazon Linux, RHEL, or Ubuntu Server) Execute batch command (Windows Server), and then enter the following:
Note
Make sure that your environment is configured with the variables and settings required to run rake; otherwise, the build will fail.

14. Choose Add post-build action, and then choose AWS CodePipeline Publisher. Choose Add, and in Build Output Locations, leave the location blank. This configuration is the default. It will create a compressed file at the end of the build process.

15. Choose Save to save your Jenkins project.

Step 2: Create a pipeline in CodePipeline

In this part of the tutorial, you create the pipeline using the Create Pipeline wizard.

To create a CodePipeline automated release process


2. If necessary, use the Region selector to change the Region to the one where your pipeline resources are located. For example, if you created resources for the previous tutorial in us-east-2, make sure that the Region selector is set to US East (Ohio).

For more information about the Regions and endpoints available for CodePipeline, see AWS CodePipeline endpoints and quotas.

3. On the Welcome page, Getting started page, or the Pipelines page, choose Create pipeline.

4. On the Step 1: Choose pipeline settings page, in Pipeline name, enter the name for your pipeline.

5. In Service role, choose New service role to allow CodePipeline to create a service role in IAM.

6. Leave the settings under Advanced settings at their defaults, and choose Next.

7. In Step 2: Add source stage, in Source provider, choose GitHub, and then choose Connect to GitHub. This will open a new browser window that will connect you to GitHub. If prompted to sign in, provide your GitHub credentials.

Important
Do not provide your AWS credentials on the GitHub website.

After you have selected GitHub, a message displays advising that CodePipeline will create a webhook in GitHub for your pipeline.

After you have connected to GitHub, choose the repository and branch where you pushed the sample you want to use for this tutorial (aws-codepipeline-jenkins-aws-codedeploy_linux.zip or AWSCodePipeline-Jenkins-AWSCodeDeploy_Windows.zip), and then choose Next.

Note
In GitHub, there is a limit to the number of OAuth tokens you can use for an application, such as CodePipeline. If you exceed this limit, retry the connection to allow CodePipeline to reconnect by reusing existing tokens. For more information, see Pipeline error: I receive a pipeline error that says: "Could not access the GitHub repository" or "Unable to connect to the GitHub repository" (p. 399).

8. In Step 3: Add build stage, choose Add Jenkins. In Provider name, enter the name of the action you provided in the CodePipeline Plugin for Jenkins (for example MyJenkinsProviderName). This name must exactly match the name in the CodePipeline Plugin for Jenkins. In Server URL, enter the URL of the EC2 instance where Jenkins is installed. In Project name, enter the name of the project you created in Jenkins, such as MyDemoProject, and then choose Next.
9. In **Step 4: Add deploy stage**, reuse the CodeDeploy application and deployment group you created in Tutorial: Create a simple pipeline (S3 bucket) (p. 38). In **Deploy provider**, choose CodeDeploy. In **Application name**, enter `CodePipelineDemoApplication`, or choose the refresh button, and then choose the application name from the list. In **Deployment group**, enter `CodePipelineDemoFleet`, or choose it from the list, and then choose **Next**.

   **Note**
   You can use your own CodeDeploy resources or create new ones, but you might incur additional costs.

10. In **Step 5: Review**, review the information, and then choose **Create pipeline**.

11. The pipeline automatically starts and runs the sample through the pipeline. You can view progress and success and failure messages as the pipeline builds the Haml sample to HTML and deploys it a webpage to each of the Amazon EC2 instances in the CodeDeploy deployment.

---

### Step 3: Add another stage to your pipeline

Now you will add a test stage and then a test action to that stage that uses the Jenkins test included in the sample to determine whether the webpage has any content. This test is for demonstration purposes only.

   **Note**
   If you did not want to add another stage to your pipeline, you could add a test action to the Staging stage of the pipeline, before or after the deployment action.

#### Add a test stage to your pipeline

**Topics**
- Look up the IP address of an instance (p. 67)
- Create a Jenkins project for testing the deployment (p. 67)
- Create a fourth stage (p. 68)

#### Look up the IP address of an instance

**To verify the IP address of an instance where you deployed your code**

1. After **Succeeded** is displayed for the pipeline status, in the status area for the Staging stage, choose **Details**.
2. In the **Deployment Details** section, in **Instance ID**, choose the instance ID of one of the successfully deployed instances.
3. Copy the IP address of the instance (for example, `192.168.0.4`). You will use this IP address in your Jenkins test.

#### Create a Jenkins project for testing the deployment

**To create the Jenkins project**

1. On the instance where you installed Jenkins, open Jenkins and from the main page, choose **New Item**.
2. In **Item Name**, enter a name for the Jenkins project (for example, `MyTestProject`). Choose **Freestyle project**, and then choose **OK**.

   **Note**
   Make sure that the name for your project meets the CodePipeline requirements. For more information, see Quotas in AWS CodePipeline (p. 521).
3. On the configuration page for the project, select the **Execute concurrent builds if necessary** check box. In **Source Code Management**, choose **AWS CodePipeline**. If you have installed Jenkins on an EC2 instance and configured the AWS CLI with the profile for the IAM user you created for integration between CodePipeline and Jenkins, leave all the other fields empty.

**Important**
If you are configuring a Jenkins project and it is not installed on an Amazon EC2 instance, or it is installed on an EC2 instance that is running a Windows operating system, complete the fields as required by your proxy host and port settings, and provide the credentials of the IAM user you configured for integration between Jenkins and CodePipeline.

4. Choose **Advanced**, and in **Category**, choose **Test**.

5. In **Provider**, enter the same name you used for the build project (for example, *MyJenkinsProviderName*). You will use this name when you add the test action to your pipeline later in this tutorial.

**Note**
This name must meet the CodePipeline naming requirements for actions. For more information, see **Quotas in AWS CodePipeline** (p. 521).

6. In **Build Triggers**, clear any check boxes, and then select **Poll SCM**. In **Schedule**, enter five asterisks separated by spaces, as follows:

```
* * * *
```

This polls CodePipeline every minute.

7. In **Build**, choose **Add build step**. If you are deploying to Amazon Linux, RHEL, or Ubuntu Server instances, choose **Execute shell**. Then enter the following, where the IP address is the address of the EC2 instance you copied earlier:

```
TEST_IP_ADDRESS=192.168.0.4 rake test
```

If you are deploying to Windows Server instances, choose **Execute batch command**, and then enter the following, where the IP address is the address of the EC2 instance you copied earlier:

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

**Note**
The test assumes a default port of 80. If you want to specify a different port, add a test port statement, as follows:

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

8. Choose **Add post-build action**, and then choose **AWS CodePipeline Publisher**. Do not choose **Add**.

9. Choose **Save** to save your Jenkins project.

**Create a fourth stage**

**To add a stage to your pipeline that includes the Jenkins test action**

1. Sign in to the AWS Management Console and open the CodePipeline console at **http://console.aws.amazon.com/codesuite/codepipeline/home**.

2. In **Name**, choose the name of the pipeline you created, **MySecondPipeline**.

3. On the pipeline details page, choose **Edit**.

4. On the **Edit** page, choose **+ Stage** to add a stage immediately after the Build stage.
5. In the name field for the new stage, enter a name (for example, Testing), and then choose + Add action group.

6. In Action name, enter MyJenkinsTest-Action. In Test provider, choose the provider name you specified in Jenkins (for example, MyJenkinsProviderName). In Project name, enter the name of the project you created in Jenkins (for example, MyTestProject). In Input artifacts, choose the artifact from the Jenkins build whose default name is BuildArtifact, and then choose Done.

   **Note**
   Because the Jenkins test action operates on the application built in the Jenkins build step, use the build artifact for the input artifact to the test action.

   For more information about input and output artifacts and the structure of pipelines, see CodePipeline pipeline structure reference (p. 454).

7. On the Edit page, choose Save pipeline changes. In the Save pipeline changes dialog box, choose Save and continue.

8. Although the new stage has been added to your pipeline, a status of No executions yet is displayed for that stage because no changes have triggered another run of the pipeline. To run the sample through the revised pipeline, on the pipeline details page, choose Release change.

   The pipeline view shows the stages and actions in your pipeline and the state of the revision running through those four stages. The time it takes for the pipeline to run through all stages will depend on the size of the artifacts, the complexity of your build and test actions, and other factors.

### Step 4: Clean up resources

After you complete this tutorial, you should delete the pipeline and the resources it uses so you will not be charged for continued use of those resources. If you do not intend to keep using CodePipeline, delete the pipeline, then the CodeDeploy application and its associated Amazon EC2 instances, and finally, the Amazon S3 bucket used to store artifacts. You should also consider whether to delete other resources, such as the GitHub repository, if you do not intend to keep using them.

**To clean up the resources used in this tutorial**

1. Open a terminal session on your local Linux, macOS, or Unix machine, or a command prompt on your local Windows machine, and run the delete-pipeline command to delete the pipeline you created. For MySecondPipeline, you would enter the following command:

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

   This command returns nothing.

2. To clean up your CodeDeploy resources, follow the instructions in Cleaning Up.

3. To clean up your instance resources, delete the EC2 instance where you installed Jenkins. For more information, see Clean up your instance.

4. If you do not intend to create more pipelines or use CodePipeline again, delete the Amazon S3 bucket used to store artifacts for your pipeline. To delete the bucket, follow the instructions in Deleting a bucket.

5. If you do not intend to use the other resources for this pipeline again, consider deleting them by following the guidance for that particular resource. For example, if you want to delete the GitHub repository, follow the instructions in Deleting a repository on the GitHub website.
Tutorial: Set up a CloudWatch Events rule to receive email notifications for pipeline state changes

After you set up a pipeline in AWS CodePipeline, you can set up a CloudWatch Events rule to send notifications whenever there are changes to the execution state of your pipelines, or in the stages or actions in your pipelines. For more information on using CloudWatch Events to set up notifications for pipeline state changes, see Detect and react to changes in pipeline state with Amazon CloudWatch Events (p. 384).

In this tutorial, you configure a notification to send an email when a pipeline's state changes to FAILED. This tutorial uses an input transformer method when creating the CloudWatch Events rule. It transforms the message schema details to deliver the message in human-readable text.

**Note**
As you create the resources for this tutorial, such as the Amazon SNS notification and the CloudWatch Events rule, make sure the resources are created in the same AWS Region as your pipeline.

**Topics**
- Step 1: Set up an email notification using Amazon SNS (p. 70)
- Step 2: Create a rule and add the SNS topic as the target (p. 71)
- Step 3: Clean up resources (p. 73)

**Step 1: Set up an email notification using Amazon SNS**

Amazon SNS coordinates use of topics to deliver messages to subscribing endpoints or clients. Use Amazon SNS to create a notification topic and then subscribe to the topic using your email address. The Amazon SNS topic will be added as a target to your CloudWatch Events rule. For more information, see the Amazon Simple Notification Service Developer Guide.

Create or identify a topic in Amazon SNS. CodePipeline will use CloudWatch Events to send notifications to this topic through Amazon SNS. To create a topic:

1. Open the Amazon SNS console at https://console.aws.amazon.com/sns.
2. Choose **Create topic**.
3. In the **Create new topic** dialog box, for **Topic name**, type a name for the topic (for example, **PipelineNotificationTopic**).
4. Choose **Create topic**.
   
   For more information, see **Create a Topic** in the *Amazon SNS Developer Guide*.

Subscribe one or more recipients to the topic to receive email notifications. To subscribe a recipient to a topic:

1. In the Amazon SNS console, from the **Topics** list, select the check box next to your new topic. Choose **Actions, Subscribe to topic**.
2. In the **Create subscription** dialog box, verify that an ARN appears in **Topic ARN**.
3. For **Protocol**, choose **Email**.
4. For **Endpoint**, type the recipient’s full email address. Compare your results to the following:

![Create subscription](image)

5. Choose **Create Subscription**.
6. Amazon SNS sends a subscription confirmation email to the recipient. To receive email notifications, the recipient must choose the **Confirm subscription** link in this email. After the recipient clicks the link, if successfully subscribed, Amazon SNS displays a confirmation message in the recipient’s web browser.

   For more information, see **Subscribe to a Topic** in the *Amazon SNS Developer Guide*.

**Step 2: Create a rule and add the SNS topic as the target**

Create a CloudWatch Events notification rule with CodePipeline as the event source.

2. In the navigation pane, choose **Events**.
3. Choose **Create rule**. Under **Event source**, choose **AWS CodePipeline**. For **Event Type**, choose **Pipeline Execution State Change**.
4. Choose **Specific state(s)**, and choose **FAILED**.
5. Choose **Edit** to open the JSON editor for the **Event Pattern Preview** pane. Add the **pipeline** parameter with the name of your pipeline as shown in the following example for a pipeline named "myPipeline."
You can copy the event pattern here and paste it into the console:

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

6. For Targets, choose Add target.
7. In the list of targets, choose SNS topic. For Topic, enter the topic you created.
8. Expand Configure input, and then choose Input Transformer.
9. In the Input Path box, type the following key-value pairs.

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

In the Input Template box, type the following:
"The Pipeline <pipeline> has failed."

10. Choose Configure details.

11. On the Configure rule details page, type a name and an optional description. For State, leave the Enabled box selected.

12. Choose Create rule.

13. Confirm that CodePipeline is now sending build notifications. For example, check to see if the build notification emails are now in your inbox.

14. To change a rule's behavior, in the CloudWatch console, choose the rule, and then choose Actions, Edit. Edit the rule, choose Configure details, and then choose Update rule.

   To stop using a rule to send build notifications, in the CloudWatch console, choose the rule, and then choose Actions, Disable.

   To delete a rule, in the CloudWatch console, choose the rule, and then choose Actions, Delete.

Step 3: Clean up resources

After you complete this tutorial, you should delete the pipeline and the resources it uses so you will not be charged for continued use of those resources.

For information about how to clean up the SNS notification and delete the Amazon CloudWatch Events rule, see Clean Up (Unsubscribe from an Amazon SNS Topic) and reference DeleteRule in the Amazon CloudWatch Events API Reference.

Tutorial: Create a pipeline that builds and tests your Android app when a commit is pushed to your GitHub repository

You can use AWS CodePipeline to configure a continuous integration flow in which your app is built and tested each time a commit is pushed to its repository. This tutorial shows how to create and configure a pipeline to build and test your Android app with source code in a GitHub repository. The pipeline detects the arrival of a new commit through webhooks that CodePipeline configures for your GitHub repository, and then uses CodeBuild to build the app and Device Farm to test it.

Important

Many of the actions you add to your pipeline in this procedure involve AWS resources that you need to create before you create the pipeline. AWS resources for your source actions must always be created in the same AWS Region where you create your pipeline. For example, if you create your pipeline in the US East (Ohio) Region, your CodeCommit repository must be in the US East (Ohio) Region.

You can add cross-region actions when you create your pipeline. AWS resources for cross-region actions must be in the same AWS Region where you plan to execute the action. For more information, see Add a cross-Region action in CodePipeline (p. 364).

You can try this out using your existing Android app and test definitions, or you can use the sample app and test definitions provided by Device Farm.
Configure CodePipeline to use your Device Farm tests

1. Add and commit a file called `buildspec.yml` in the root of your app code, and push it to your repository. CodeBuild uses this file to perform commands and access artifacts required to build your app.

```yaml
version: 0.2
phases:
  build:
    commands:
      - chmod +x ./gradlew
      - ./gradlew assembleDebug
    artifacts:
      files:
        - ./android/app/build/outputs/**/*.apk
      discard-paths: yes
```

2. (Optional) If you use Calabash or Appium to test your app, add the test definition file to your repository. In a later step, you can configure Device Farm to use the definitions to carry out your test suite.
If you use Device Farm built-in tests, you can skip this step.

3. To create your pipeline and add a source stage, do the following:

   a. Sign in to the AWS Management Console and open the CodePipeline console at https://console.aws.amazon.com/codepipeline/.

   b. Choose Create pipeline. On the Step 1: Choose pipeline settings page, in Pipeline name, enter the name for your pipeline.

   c. In Service role, leave New service role selected, and leave Role name unchanged. You can also choose to use an existing service role, if you have one.

   Note
   If you use a CodePipeline service role that was created before July 2018, you need to add permissions for Device Farm. To do this, open the IAM console, find the role, and then add the following permissions to the role’s policy. For more information, see Add permissions to the CodePipeline service role (p. 446).

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

   d. Leave the settings under Advanced settings at their defaults, and then choose Next.

   e. On the Step 2: Add source stage page, in Source provider, choose GitHub, and then choose Connect to GitHub.

   f. In the browser window, choose Authorize aws-codesuite. This allows your pipeline to make your repository a source, and to use webhooks that detect when new code is pushed to the repository.

   g. In Repository, choose the source repository.

   h. In Branch, choose the branch that you want to use.
i. Choose Next.

4. In Add build stage, add a build stage:
   a. In Build provider, choose AWS CodeBuild. Allow Region to default to the pipeline Region.
   b. Choose Create project.
   c. In Project name, enter a name for this build project.
   d. In Environment image, choose Managed image. For Operating system, choose Ubuntu.
   e. For Runtime, choose Standard. For Image, choose aws/codebuild/standard:1.0.

   CodeBuild uses this OS image, which has Android Studio installed, to build your app.
   f. For Service role, choose your existing CodeBuild service role or create a new one.
   g. For Build specifications, choose Use a buildspec file.
   h. Choose Continue to CodePipeline. This returns to the CodePipeline console and creates a
      CodeBuild project that uses the buildspec.yml in your repository for configuration. The build
      project uses a service role to manage AWS service permissions. This step might take a couple of
      minutes.
   i. Choose Next.

5. On the Step 4: Add deploy stage page, choose Skip deploy stage, and then accept the warning
   message by choosing Skip again. Choose Next.

6. On Step 5: Review, choose Create pipeline. You should see a diagram that shows the source and
   build stages.
7. Add a Device Farm test action to your pipeline:

   a. In the upper right, choose **Edit**.
   
   b. At the bottom of the diagram, choose **+ Add stage**. In **Stage name**, enter a name, such as **Test**.
   
   c. Choose **+ Add action group**.
   
   d. In **Action name**, enter a name.
   
   e. In **Action provider**, choose **AWS Device Farm**. Allow **Region** to default to the pipeline Region.
   
   f. In **Input artifacts**, choose the input artifact that matches the output artifact of the stage that comes before the test stage, such as **BuildArtifact**.

   In the AWS CodePipeline console, you can find the name of the output artifact for each stage by hovering over the information icon in the pipeline diagram. If your pipeline tests your app directly from the **Source** stage, choose **SourceArtifact**. If the pipeline includes a **Build** stage, choose **BuildArtifact**.

   g. In **ProjectId**, enter your Device Farm project ID. Use the steps at the start of this tutorial to retrieve your project ID.
   
   h. In **DevicePoolArn**, enter the ARN for the device pool.
   
   i. In **AppType**, enter **Android**.

   API Version 2015-07-09
Configure CodePipeline to use your Device Farm tests

The following is a list of valid values for **AppType**:

- iOS
- Android
- Web

j. In **App**, enter the path of the compiled app package. The path is relative to the root of the input artifact for the test stage. Typically, this path is similar to `app-release.apk`.

k. In **TestType**, enter your type of test, and then in **Test**, enter the path of the test definition file. The path is relative to the root of the input artifact for your test.

The following is a list of valid values for **TestType**:

- **APPIUM_JAVA_JUNIT**
- **APPIUM_JAVA_TESTNG**
- **APPIUM_PYTHON**
- **APPIUM_WEB_JAVA_JUNIT**
• APPIUM_WEB_JAVA_TESTNG
• APPIUM_WEB_PYTHON
• BUILTIN_EXPLORER
• BUILTIN_FUZZ
• CALABASH
• INSTRUMENTATION
• UIAUTOMATION
• UIAUTOMATOR
• WEB_PERFORMANCE_PROFILE
• XCTEST
• XCTEST_UI

Note
Custom environment nodes are not supported.

l. In the remaining fields, provide the configuration that is appropriate for your test and application type.

m. (Optional) In Advanced, provide configuration information for your test run.

n. Choose Save.

o. On the stage you are editing, choose Done. In the AWS CodePipeline pane, choose Save, and then choose Save on the warning message.

p. To submit your changes and start a pipeline build, choose Release change, and then choose Release.
Tutorial: Create a pipeline that tests your iOS app after a change in your S3 bucket

You can use AWS CodePipeline to easily configure a continuous integration flow in which your app is tested each time the source bucket changes. This tutorial shows you how to create and configure a pipeline to test your built iOS app from an S3 bucket. The pipeline detects the arrival of a saved change through Amazon CloudWatch Events, and then uses Device Farm to test the built application.

**Important**

Many of the actions you add to your pipeline in this procedure involve AWS resources that you need to create before you create the pipeline. AWS resources for your source actions must always be created in the same AWS Region where you create your pipeline. For example, if you create your pipeline in the US East (Ohio) Region, your CodeCommit repository must be in the US East (Ohio) Region.
You can add cross-region actions when you create your pipeline. AWS resources for cross-region actions must be in the same AWS Region where you plan to execute the action. For more information, see Add a cross-Region action in CodePipeline (p. 364).

You can try this out using your existing iOS app, or you can use the sample iOS app.

Workflow to Set Up iOS Application Test

1. Configure CodePipeline resources
2. Add definitions
3. Upload
4. Test
5. Report

Configure
Add definitions
Upload
Test
Report

Configure pipeline resources
Add test definitions to your package
Upload .zip to your bucket
Test output artifact kicked off automatically
View test results

Before you begin

1. Sign in to the AWS Device Farm console and choose Create a new project.
2. Choose your project. In the browser, copy the URL of your new project. The URL contains the project ID.
3. Copy and retain this project ID. You use it when you create your pipeline in CodePipeline.

Here is an example URL for a project. To extract the project ID, copy the value after projects/. In this example, the project ID is eec4905f-98f8-40aa-9af-4c1cfexample.

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

Configure CodePipeline to use your Device Farm tests (Amazon S3 example)

1. Create or use an S3 bucket with versioning enabled. Follow the instructions in Step 1: Create an S3 bucket for your application (p. 38) to create an S3 bucket.
2. In the Amazon S3 console for your bucket, choose Upload, and follow the instructions to upload your .zip file.
   Your sample application must be packaged in a .zip file.
3. To create your pipeline and add a source stage, do the following:
   a. Sign in to the AWS Management Console and open the CodePipeline console at https://console.aws.amazon.com/codepipeline/.
   b. Choose Create pipeline. On the Step 1: Choose pipeline settings page, in Pipeline name, enter the name for your pipeline.
c. In **Service role**, leave **New service role** selected, and leave **Role name** unchanged. You can also choose to use an existing service role, if you have one.

**Note**

If you use a CodePipeline service role that was created before July 2018, you must add permissions for Device Farm. To do this, open the IAM console, find the role, and then add the following permissions to the role's policy. For more information, see Add permissions to the CodePipeline service role (p. 446).

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

d. Leave the settings under **Advanced settings** at their defaults, and then choose **Next**.

e. On the **Step 2: Add source stage** page, in **Source provider**, choose **Amazon S3**.

f. In **Amazon S3 location**, enter the bucket and object key for your .zip file.

![Add source stage](image)

g. Choose **Next**.

4. In **Build**, create a placeholder build stage for your pipeline. This allows you to create the pipeline in the wizard. After you use the wizard to create your two-stage pipeline, you no longer need this placeholder build stage. After the pipeline is completed, this second stage is deleted and the new test stage is added in step 5.

a. In **Build provider**, choose **Add Jenkins**. This build selection is a placeholder. It is not used.

b. In **Provider name**, enter a name. The name is a placeholder. It is not used.
c. In **Server URL**, enter text. The text is a placeholder. It is not used.

d. In **Project name**, enter a name. The name is a placeholder. It is not used.

e. Choose **Next**.

f. On the **Step 4: Add deploy stage** page, choose **Skip deploy stage**, and then accept the warning message by choosing **Skip** again.

g. On **Step 5: Review**, choose **Create pipeline**. You should see a diagram that shows the source and build stages.
5. Add a Device Farm test action to your pipeline as follows:

   a. In the upper right, choose **Edit**.
   b. Choose **Edit stage**. Choose **Delete**. This deletes the placeholder stage now that you no longer need it for pipeline creation.
   c. At the bottom of the diagram, choose **Add stage**.
   d. In Stage name, enter a name for the stage, such as Test, and then choose **Add stage**.
   e. Choose **Add action group**.
   f. In **Action name**, enter a name, such as DeviceFarmTest.
   g. In **Action provider**, choose **AWS Device Farm**. Allow **Region** to default to the pipeline Region.
   h. In **Input artifacts**, choose the input artifact that matches the output artifact of the stage that comes before the test stage, such as **SourceArtifact**.

   In the AWS CodePipeline console, you can find the name of the output artifact for each stage by hovering over the information icon in the pipeline diagram. If your pipeline tests your app directly from the **Source** stage, choose **SourceArtifact**. If the pipeline includes a **Build** stage, choose **BuildArtifact**.

   i. In **ProjectId**, choose your Device Farm project ID. Use the steps at the start of this tutorial to retrieve your project ID.
   j. In **DevicePoolArn**, enter the ARN for the device pool.
   k. In **AppType**, enter **iOS**.
The following is a list of valid values for **AppType:**

- iOS
- Android
- Web

1. In **App**, enter the path of the compiled app package. The path is relative to the root of the input artifact for the test stage. Typically, this path is similar to `ios-test.ipa`.

2. In **TestType**, enter your type of test, and then in **Test**, enter the path of the test definition file. The path is relative to the root of the input artifact for your test.

The following is a list of valid values for **TestType:**

- `APPIUM_JAVA_JUNIT`
- `APPIUM_JAVA_TESTNG`
- `APPIUM_PYTHON`
Configure CodePipeline to use your Device Farm tests (Amazon S3 example)

- APPIUM_WEB_JAVA_JUNIT
- APPIUM_WEB_JAVA_TESTNG
- APPIUM_WEB_PYTHON
- BUILTIN_EXPLORER
- BUILTIN_FUZZ
- CALABASH
- INSTRUMENTATION
- UIAUTOMATION
- UIAUTOMATOR
- WEB_PERFORMANCE_PROFILE
- XCTEST
- XCTEST_UI

**Note**
Custom environment nodes are not supported.

n. In the remaining fields, provide the configuration that is appropriate for your test and application type.

o. (Optional) In **Advanced**, provide configuration information for your test run.

p. Choose **Save**.

q. On the stage you are editing, choose **Done**. In the AWS CodePipeline pane, choose **Save**, and then choose **Save** on the warning message.

r. To submit your changes and start a pipeline execution, choose **Release change**, and then choose **Release**.
Tutorial: Create a pipeline that deploys to AWS Service Catalog

AWS Service Catalog enables you to create and provision products based on AWS CloudFormation templates. This tutorial shows you how to create and configure a pipeline to deploy your product template to AWS Service Catalog and deliver changes you have made in your source repository (already created in GitHub, CodeCommit, or Amazon S3).

**Note**
When Amazon S3 is the source provider for your pipeline, you must upload to your bucket all source files packaged as a single .zip file. Otherwise, the source action fails.

First, you create a product in AWS Service Catalog, and then you create a pipeline in AWS CodePipeline. This tutorial provides two options for setting up the deployment configuration:

- Create a product in AWS Service Catalog and upload a template file to your source repository. Provide product version and deployment configuration in the CodePipeline console (without a separate configuration file). See Option 1: Deploy to AWS Service Catalog without a configuration file (p. 88).

  **Note**
  The template file can be created in YAML or JSON format.
Option 1: Deploy to AWS Service Catalog without a configuration file

In this example, you upload the sample AWS CloudFormation template file for an S3 bucket, and then create your product in AWS Service Catalog. Next, you create your pipeline and specify deployment configuration in the CodePipeline console.

Step 1: Upload sample template file to source repository

1. Open a text editor. Create a sample template by pasting the following into the file. Save the file as S3_template.json.

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

This template allows AWS CloudFormation to create an S3 bucket that can be used by AWS Service Catalog.

2. Upload the S3_template.json file to your AWS CodeCommit repository.

Step 2: Create a product in AWS Service Catalog

1. As an IT administrator, sign in to the AWS Service Catalog console, go to the Products page, and then choose Upload new product.
2. On the Upload new product page, complete the following:
   a. In Product name, enter the name you want to use for your new product.
   b. In Description, enter the product catalog description. This description is shown in the product listing to help the user choose the correct product.
   c. In Provided by, enter the name of your IT department or administrator.
   d. Choose Next.
3. (Optional) In **Enter support details**, enter contact information for product support, and choose **Next**.

4. In **Version details**, complete the following:
   a. Choose **Upload a template file**. Browse for your S3_template.json file and upload it.
   b. In **Version title**, enter the name of the product version (for example, **devops S3 v2**).
   c. In **Description**, enter details that distinguish this version from other versions.
   d. Choose **Next**.

5. On the **Review** page, verify that the information is correct, and then choose **Create**.

6. On the **Products** page, in the browser, copy the URL of your new product. This contains the product ID. You use it when you create your pipeline in CodePipeline.

   Here is the URL for a product named **my-product**. To extract the product ID, copy the value between the equals sign (=) and the ampersand (&). In this example, the product ID is **prod-example123456**.

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

   **Note**
   Copy the URL for your product before you navigate away from the page. Once you navigate away from this page, you must use the CLI to obtain your product ID.

   After a few seconds, your product appears on the **Products** page. You might need to refresh your browser to see the product in the list.

---

**Step 3: Create your pipeline**

1. To name your pipeline and select parameters for your pipeline, do the following:
   b. Choose **Getting started**. Choose **Create pipeline**, and then enter a name for your pipeline.
   c. In **Service role**, choose **New service role** to allow CodePipeline to create a service role in IAM.
   d. Leave the settings under **Advanced settings** at their defaults, and then choose **Next**.

2. To add a source stage, do the following:
   a. In **Source provider**, choose **AWS CodeCommit**.
   b. In **Repository name** and **Branch name**, enter the repository and branch you want to use for your source action.
   c. Choose **Next**.

3. In **Add build stage**, choose **Skip build stage**, and then accept the warning message by choosing **Skip** again.

4. In **Add deploy stage**, complete the following:
   a. In **Deploy provider**, choose **AWS Service Catalog**.
   b. For deployment configuration, choose **Enter deployment configuration**.

---

**API Version 2015-07-09**

89
c. In **Product ID**, paste the product ID you copied from the AWS Service Catalog console.

d. In **Template file path**, enter the relative path where the template file is stored.

  e. In **Product type**, choose **AWS CloudFormation Template**.

  f. In **Product version name**, enter the name of the product version you specified in AWS Service Catalog. If you want to have the template change deployed to a new product version, enter a product version name that has not been used for any previous product version in the same product.

  g. For **Input artifact**, choose the source input artifact.

  h. Choose **Next**.

5. In **Review**, review your pipeline settings, and then choose **Create**.

6. After your pipeline runs successfully, on the deployment stage, choose **Details**. This opens your product in AWS Service Catalog.
7. Under your product information, choose your version name to open the product template. View the template deployment.
Step 4: Push a change and verify your product in AWS Service Catalog

1. View your pipeline in the CodePipeline console, and on your source stage, choose Details. Your source AWS CodeCommit repository opens in the console. Choose Edit, and make a change in the file (for example, to the description).

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

2. Commit and push your change. Your pipeline starts after you push the change. When the run of the pipeline is complete, on the deployment stage, choose Details to open your product in AWS Service Catalog.

3. Under your product information, choose the new version name to open the product template. View the deployed template change.
Option 2: Deploy to AWS Service Catalog using a configuration file

In this example, you upload the sample AWS CloudFormation template file for an S3 bucket, and then create your product in AWS Service Catalog. You also upload a separate configuration file that specifies your deployment configuration. Next, you create your pipeline and specify the location of your configuration file.

Step 1: Upload sample template file to source repository

1. Open a text editor. Create a sample template by pasting the following into the file. Save the file as S3_template.json.

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

2. Upload the S3_template.json file to your AWS CodeCommit repository.

Step 2: Create your product deployment configuration file

1. Open a text editor. Create the configuration file for your product. The configuration file is used to define your AWS Service Catalog deployment parameters/preferences. You use this file when you create your pipeline.

   This sample provides a ProductVersionName of "devops S3 v2" and a ProductVersionDescription of "MyProductVersionDescription". If you want to have the template change deployed to a new product version, just enter a product version name that has not been used for any previous product version in the same product.

   Save the file as sample_config.json.

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

This file creates the product version information for you each time your pipeline runs.

2. Upload the sample_config.json file to your AWS CodeCommit repository. Make sure you upload this file to your source repository.

Step 3: Create a product in AWS Service Catalog

1. As an IT administrator, sign in to the AWS Service Catalog console, go to the Products page, and then choose Upload new product.

2. On the Upload new product page, complete the following:
   a. In Product name, enter the name you want to use for your new product.
   b. In Description, enter the product catalog description. This description appears in the product listing to help the user choose the correct product.
   c. In Provided by, enter the name of your IT department or administrator.
   d. Choose Next.

3. (Optional) In Enter support details, enter product support contact information, and then choose Next.

4. In Version details, complete the following:
   a. Choose Upload a template file. Browse for your S3_template.json file and upload it.
   b. In Version title, enter the name of the product version (for example, "devops S3 v2").
   c. In Description, enter details that distinguish this version from other versions.
   d. Choose Next.
5. On the **Review** page, verify that the information is correct, and then choose **Confirm and upload**.

6. On the **Products** page, in the browser, copy the URL of your new product. This contains the product ID. Copy and retain this product ID. You use when you create your pipeline in CodePipeline.

Here is the URL for a product named *my-product*. To extract the product ID, copy the value between the equals sign (=) and the ampersand (&). In this example, the product ID is *prod-example123456*.

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

**Note**
Copy the URL for your product before you navigate away from the page. Once you navigate away from this page, you must use the CLI to obtain your product ID.

After a few seconds, your product appears on the **Products** page. You might need to refresh your browser to see the product in the list.

### Step 4: Create your pipeline

1. To name your pipeline and select parameters for your pipeline, do the following:
   a. Sign in to the AWS Management Console and open the CodePipeline console at https://console.aws.amazon.com/codepipeline/.
   b. Choose **Getting started**. Choose **Create pipeline**, and then enter a name for your pipeline.
   c. In **Service role**, choose **New service role** to allow CodePipeline to create a service role in IAM.
   d. Leave the settings under **Advanced settings** at their defaults, and then choose **Next**.

2. To add a source stage, do the following:
   a. In **Source provider**, choose **AWS CodeCommit**.
   b. In **Repository name** and **Branch name**, enter the repository and branch you want to use for your source action.
   c. Choose **Next**.

3. In **Add build stage**, choose **Skip build stage**, and then accept the warning message by choosing **Skip** again.

4. In **Add deploy stage**, complete the following:
   a. In **Deploy provider**, choose **AWS Service Catalog**.
   b. Choose **Use configuration file**.
c. In **Product ID**, paste the product ID you copied from the AWS Service Catalog console.

d. In **Configuration file path**, enter the file path of the configuration file in your repository.

e. Choose **Next**.

5. In **Review**, review your pipeline settings, and then choose **Create**.

6. After your pipeline runs successfully, on your deployment stage, choose **Details** to open your product in AWS Service Catalog.
7. Under your product information, choose your version name to open the product template. View the template deployment.
Option 2: Deploy to AWS Service Catalog using a configuration file

Step 5: Push a change and verify your product in Service Catalog

1. View your pipeline in the CodePipeline console, and on the source stage, choose Details. Your source AWS CodeCommit repository opens in the console. Choose Edit, and then make a change in the file (for example, to the description).

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

2. Commit and push your change. Your pipeline starts after you push the change. When the run of the pipeline is complete, on the deployment stage, choose Details to open your product in AWS Service Catalog.

3. Under your product information, choose the new version name to open the product template. View the deployed template change.
Tutorial: Create a pipeline with AWS CloudFormation

The examples provide sample templates that allow you to use AWS CloudFormation to create a pipeline that deploys your application to your instances each time the source code changes. The sample template creates a pipeline that you can view in AWS CodePipeline. The pipeline detects the arrival of a saved change through Amazon CloudWatch Events.

Topics

- Example 1: Create an AWS CodeCommit pipeline with AWS CloudFormation (p. 99)
- Example 2: Create an Amazon S3 pipeline with AWS CloudFormation (p. 101)
- Example 3: Create a GitHub pipeline with AWS CloudFormation (p. 103)

Example 1: Create an AWS CodeCommit pipeline with AWS CloudFormation

This walkthrough shows you how to use the AWS CloudFormation console to create infrastructure that includes a pipeline connected to a CodeCommit source repository. In this tutorial, you use the provided sample template file to create your resource stack, which includes your artifact store, pipeline, and change-detection resources, such as your Amazon CloudWatch Events rule. After you create your resource stack in AWS CloudFormation, you can view your pipeline in the AWS CodePipeline console. The pipeline is a two-stage pipeline with a CodeCommit source stage and a CodeDeploy deployment stage.

Prerequisites:

You must have created the following resources to use with the AWS CloudFormation sample template:

```
devops S3 v2

Description: Product version description

Guidance

Template

{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "AWS CloudFormation Sample Template S3_Bucket: Sample template showing how to create S3_Bucket",
  "Resources": {
    "S3Bucket": {
      "Type": "AWS::S3::Bucket",
      "Properties": {
        "BucketName": "devops-s3"
      }
    }
  },
  "Outputs": {
    "BucketName": {
      "Value": {
        "Ref": "S3Bucket"
      },
      "Description": "Name of Amazon S3 bucket to hold and version website content"
    }
  }
}
```
Example 1: Create an AWS CodeCommit pipeline with AWS CloudFormation

- You must have created a source repository. You can use the AWS CodeCommit repository you created in Tutorial: Create a simple pipeline (CodeCommit repository) (p. 52).

- You must have created a CodeDeploy application and deployment group. You can use the CodeDeploy resources you created in Tutorial: Create a simple pipeline (CodeCommit repository) (p. 52).

- Choose one of these links to download the sample AWS CloudFormation template file for creating a pipeline: YAML | JSON

Unzip the file and place it on your local computer.

- Download the SampleApp_Linux.zip sample application file.

1. Unzip the files from SampleApp_Linux.zip and upload the files to your AWS CodeCommit repository. You must upload the unzipped files to the root directory of your repository. You can follow the instructions in Step 2: Add sample code to your CodeCommit repository (p. 53) to push the files to your repository.

2. Open the AWS CloudFormation console and choose Create Stack.

3. In Choose a template, choose Upload a template to Amazon S3. Choose Browse and then select the template file from your local computer. Choose Next.

4. In Stack name, enter a name for your pipeline. Parameters specified by the sample template are displayed. Enter the following parameters:

   a. In ApplicationName, enter the name of your CodeDeploy application.
   b. In BetaFleet, enter the name of your CodeDeploy deployment group.
   c. In BranchName, enter the repository branch you want to use.
   d. In RepositoryName, enter the name of your CodeCommit source repository.

5. Choose Next. Accept the defaults on the following page, and then choose Next.

6. In Capabilities, select I acknowledge that AWS CloudFormation might create IAM resources, and then choose Create.
7. After your stack creation is complete, view the event list to check for any errors.

**Troubleshooting**

The IAM user who is creating the pipeline in AWS CloudFormation might require additional permissions to create resources for the pipeline. The following permissions are required in the IAM user’s policy to allow AWS CloudFormation to create the required Amazon CloudWatch Events resources for the CodeCommit pipeline:

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


Under **Pipelines**, choose your pipeline and choose **View**. The diagram shows your pipeline source and deployment stages.

9. In your source repository, commit and push a change. Your change-detection resources pick up the change, and your pipeline starts.

**Example 2: Create an Amazon S3 pipeline with AWS CloudFormation**

This walkthrough shows you how to use the AWS CloudFormation console to create infrastructure that includes a pipeline connected to an Amazon S3 source bucket. In this tutorial, you use the provided sample template file to create your resource stack, which includes your source bucket, artifact store, pipeline, and change-detection resources, such as your Amazon CloudWatch Events rule and CloudTrail trail. After you create your resource stack in AWS CloudFormation, you can view your pipeline in the AWS CodePipeline console. The pipeline is a two-stage pipeline with an Amazon S3 source stage and a CodeDeploy deployment stage.

**Prerequisites:**

You must have the following resources to use with the AWS CloudFormation sample template:

- You must have created the Amazon EC2 instances, where you installed the CodeDeploy agent on the instances. You must have created a CodeDeploy application and deployment group. Use the Amazon EC2 and CodeDeploy resources you created in Tutorial: Create a simple pipeline (CodeCommit repository) (p. 52).
- Choose the following links to download the sample AWS CloudFormation template files for creating a pipeline with an Amazon S3 source:
  - Download the sample template for your pipeline: YAML | JSON
  - Download the sample template for your CloudTrail bucket and trail: YAML | JSON
  - Unzip the files and place them on your local computer.
- Download the sample application from SampleApp_Linux.zip.
Create your pipeline in AWS CloudFormation

1. Open the AWS CloudFormation console, and choose Create Stack.
2. In Choose a template, choose Upload a template to Amazon S3. Choose Browse, and then select the template file from your local computer. Choose Next.
3. In Stack name, enter a name for your pipeline. Parameters specified by the sample template are displayed. Enter the following parameters:
   a. In ApplicationName, enter the name of your CodeDeploy application. You can replace the DemoApplication default name.
   b. In BetaFleet, enter the name of your CodeDeploy deployment group. You can replace the DemoFleet default name.
   c. In SourceObjectKey, enter SampleApp_Linux.zip. You upload this file to your bucket after the template creates the bucket and pipeline.
4. Choose Next. Accept the defaults on the following page, and then choose Next.
5. In Capabilities, select I acknowledge that AWS CloudFormation might create IAM resources, and then choose Create.
6. After your stack creation is complete, view the event list to check for any errors.

Troubleshooting

The IAM user who is creating the pipeline in AWS CloudFormation might require additional permissions to create resources for the pipeline. The following permissions are required in the IAM user’s policy to allow AWS CloudFormation to create the required Amazon CloudWatch Events resources for the Amazon S3 pipeline:

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

7. In AWS CloudFormation, in the Resources tab for your stack, view the resources that were created for your stack.

Choose the S3 bucket with a sourcebucket label in the name, such as s3-cfn-codepipeline-sourcebucket-y04EXAMPLE. Do not choose the pipeline artifact bucket.

The source bucket is empty because the resource is newly created by AWS CloudFormation. Open the Amazon S3 console and locate your sourcebucket bucket. Choose Upload, and follow the instructions to upload your SampleApp_Linux.zip .zip file.

Note

When Amazon S3 is the source provider for your pipeline, you must upload to your bucket all source files packaged as a single .zip file. Otherwise, the source action fails.

Under **Pipelines**, choose your pipeline, and then choose **View**. The diagram shows your pipeline source and deployment stages.

9. Complete the steps in the following procedure to create your AWS CloudTrail resources.

**Create your AWS CloudTrail resources in AWS CloudFormation**

1. Open the AWS CloudFormation console, and choose **Create Stack**.
2. In **Choose a template**, choose **Upload a template to Amazon S3**. Choose **Browse**, and then select the template file for the AWS CloudTrail resources from your local computer. Choose **Next**.
3. In **Stack name**, enter a name for your resource stack. Parameters specified by the sample template are displayed. Enter the following parameters:
   - In **SourceObjectKey**, accept the default for the sample application's zip file.
4. Choose **Next**. Accept the defaults on the following page, and then choose **Next**.
5. In **Capabilities**, select **I acknowledge that AWS CloudFormation might create IAM resources**, and then choose **Create**.
6. After your stack creation is complete, view the event list to check for any errors.

   The following permissions are required in the IAM user's policy to allow AWS CloudFormation to create the required CloudTrail resources for the Amazon S3 pipeline:

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

7. Sign in to the AWS Management Console and open the CodePipeline console at https://console.aws.amazon.com/codepipeline/.

   Under **Pipelines**, choose your pipeline, and then choose **View**. The diagram shows your pipeline source and deployment stages.

8. In your source bucket, commit and push a change. Your change-detection resources pick up the change and your pipeline starts.

**Example 3: Create a GitHub pipeline with AWS CloudFormation**

This walkthrough shows you how to use the AWS CloudFormation console to create infrastructure that includes a pipeline connected to a GitHub source repository. In this tutorial, you use the provided sample template file to create your resource stack, which includes your artifact store, pipeline, and change-detection resource (your webhook). After you create your resource stack in AWS CloudFormation, you can view your pipeline in the AWS CodePipeline console. The pipeline is a two-stage pipeline with a GitHub source stage and a CodeDeploy deployment stage.

We strongly recommend that you use AWS Secrets Manager to store your credentials. If you use Secrets Manager, you must have already configured and stored your secret parameters in Secrets Manager. This
example uses dynamic references to AWS Secrets Manager for the GitHub credentials for your webhook. For more information, see Using Dynamic References to Specify Template Values.

**Important**
When passing secret parameters, do not enter the value directly into the template. The value is rendered as plaintext and is therefore readable. For security reasons, do not use plaintext in your AWS CloudFormation template to store your credentials.

**Prerequisites:**

You must have created the following resources to use with the AWS CloudFormation sample template:

- A CodeDeploy application and deployment group. You can use the CodeDeploy resources you created in Tutorial: Create a simple pipeline (CodeCommit repository) (p. 52).
- Choose one of these links to download the sample AWS CloudFormation template file for creating a pipeline: YAML | JSON
  
  Unzip the file and place it on your local computer.
- The sample templates in the bullet above are configured to use a GitHub secret token with this dynamic reference to the stored token in AWS Secrets Manager: `{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}`. You must create a GitHub token and store it in Secrets Manager in order to use the dynamic reference in the template for your OAuthToken and SecretToken fields.
- Download the SampleApp_Linux.zip.
- The GitHub repository and branch you want to use for your source.
- A personal access key for your GitHub repository. This is used to provide an OAuth token for connection to your repository.

1. Unzip the files from SampleApp_Linux.zip and upload the files to your GitHub repository. You must upload the unzipped files to the root directory of your repository.
2. Open the AWS CloudFormation console and choose Create Stack.
3. In Choose a template, choose Upload a template to Amazon S3. Choose Browse, and then select the template file from your local computer. Choose Next.
4. In Stack name, enter a name for your pipeline. Parameters specified by the sample template are displayed. Enter the following parameters:
   a. In **ApplicationName**, enter the name of your CodeDeploy application.
   b. In **BetaFleet**, enter the name of your CodeDeploy deployment group.
   c. In **BranchName**, enter the repository branch you want to use.
   d. In **GitHubOAuthToken**, enter the personal access key for your GitHub repository.
   e. In **GitHubOwner**, enter the GitHub user name for the owner of the repository.
   f. In **GitHubSecret**, enter the secret you want to use for the webhook AWS CloudFormation creates.
   g. In **RepositoryName**, enter the name of your GitHub source repository.
5. Choose **Next**. Accept the defaults on the following page, and then choose **Next**.

6. In **Capabilities**, select **I acknowledge that AWS CloudFormation might create IAM resources**, and then choose **Create**.

7. After your stack creation is complete, view the event list to check for any errors.


   Under **Pipelines**, choose your pipeline, and then choose **View**. The diagram shows your pipeline source and deployment stages.

9. In your source repository, commit and push a change. Your change-detection resources pick up the change and your pipeline starts.

---

**Tutorial: Create a pipeline that uses variables from AWS CloudFormation deployment actions**

In this tutorial, you use the AWS CodePipeline console to create a pipeline with a deployment action. When the pipeline runs, the template creates a stack and also creates an `outputs` file. Outputs generated by the stack template are the variables generated by the AWS CloudFormation action in CodePipeline.

In the action where you create the stack from the template, you designate a variable namespace. The variables produced by the `outputs` file can then be consumed by subsequent actions. In this example,
you create a change set based on the `StackName` variable produced by the AWS CloudFormation action. After a manual approval, you execute the change set and then create a delete stack action that deletes the stack based on the `StackName` variable.

**Topics**

- Prerequisites: Create an AWS CloudFormation service role and a CodeCommit repository (p. 106)
- Step 1: Download, edit, and upload the sample AWS CloudFormation template (p. 106)
- Step 2: Create your pipeline (p. 107)
- Step 3: Add an AWS CloudFormation deployment action to create the change set (p. 109)
- Step 4: Add a manual approval action (p. 110)
- Step 5: Add a CloudFormation deployment action to execute the change set (p. 110)
- Step 6: Add a CloudFormation deployment action to delete the stack (p. 111)

**Prerequisites: Create an AWS CloudFormation service role and a CodeCommit repository**

You must already have the following:

- A CodeCommit repository. You can use the AWS CodeCommit repository you created in Tutorial: Create a simple pipeline (CodeCommit repository) (p. 52).
- This example creates an Amazon DocumentDB stack from a template. You must use AWS Identity and Access Management (IAM) to create an AWS CloudFormation service role with the following permissions for Amazon DocumentDB.

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

**Step 1: Download, edit, and upload the sample AWS CloudFormation template**

Download the sample AWS CloudFormation template file and upload it to your CodeCommit repository.

1. Navigate to the sample template page for your Region. For example, the page for us-west-2 is at https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/sample-templates-services-us-west-2.html. Download the template for `RDS_with_DBParameterGroup.template`. The file name is `documentdb_full_stack.yml`.
2. Unzip the `documentdb_full_stack.yml` file, and open it in a text editor. Make the following changes.
   a. For this example, add the following `Purpose:` parameter to your `Parameters` section in the template.

```
Purpose:
  Type: String
  Default: testing
  AllowedValues:
    - testing
    - production
```
Step 2: Create your pipeline

In this section, you create a pipeline with the following actions:

- A source stage with a CodeCommit action where the source artifact is your template file.
- A deployment stage with an AWS CloudFormation deployment action.

Each action in the source and deployment stages created by the wizard is assigned a variable namespace, SourceVariables and DeployVariables, respectively. Because the actions have a namespace assigned, the variables configured in this example are available to downstream actions. For more information, see Variables (p. 514).

To create a pipeline with the wizard

2. On the Welcome page, Getting started page, or Pipelines page, choose Create pipeline.
3. In Step 1: Choose pipeline settings, in Pipeline name, enter MyCFNDeployPipeline.
4. In Service role, do one of the following:
   - Choose New service role to allow CodePipeline to create a service role in IAM. In Role name, the role and policy name both default to this format: AWSCodePipelineServiceRole-region Pipeline name.
   - For example, the service role created for this tutorial is AWSCodePipelineServiceRole-us-west-2-MyS3DeployPipeline.
   - Choose Existing service role. In Role name, choose your service role from the list.
5. In Artifact store:
   - Choose Default location to use the default artifact store, such as the Amazon S3 artifact bucket designated as the default, for your pipeline in the Region you selected for your pipeline.
b. Choose **Custom location** if you already have an artifact store, such as an Amazon S3 artifact bucket, in the same Region as your pipeline.

**Note**
This is not the source bucket for your source code. This is the artifact store for your pipeline. A separate artifact store, such as an S3 bucket, is required for each pipeline. When you create or edit a pipeline, you must have an artifact bucket in the pipeline Region and one artifact bucket per AWS Region where you are running an action. For more information, see Input and output artifacts (p. 15) and CodePipeline pipeline structure reference (p. 454).

Choose **Next**.

6. **In Step 2: Add source stage:**
   a. In **Source provider**, choose **AWS CodeCommit**.
   b. In **Repository name**, choose the name of the CodeCommit repository that you created in **Step 1: Create a CodeCommit repository (p. 52)**.
   c. In **Branch name**, choose the name of the branch that contains your latest code update. Unless you created a different branch on your own, only **master** is available.

After you select the repository name and branch, the Amazon CloudWatch Events rule to be created for this pipeline is displayed.

Choose **Next**.

7. **In Step 3: Add build stage**, choose **Skip build stage**, and then accept the warning message by choosing **Skip** again.

Choose **Next**.

8. **In Step 4: Add deploy stage:**
   a. In **Action name**, choose **Deploy**. In **Deploy provider**, choose **CloudFormation**.
   b. In **Action mode**, choose **Create or update a stack**.
   c. In **Stack name**, enter a name for the stack. This is the name of the stack that the template will create.
   d. In **Output file name**, enter a name for the outputs file, such as **outputs**. This is the name of the file that will be created by the action after the stack is created.
   e. Expand **Advanced**. Under **Parameter overrides**, enter your template overrides as key-value pairs. For example, this template requires the following overrides.

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

If you don't enter overrides, the template creates a stack with default values.
Step 3: Add an AWS CloudFormation deployment action to create the change set

Create a next action in your pipeline that will allow AWS CloudFormation to create the change set before the manual approval action.


   Under Pipelines, choose your pipeline and choose View. The diagram shows your pipeline source and deployment stages.

2. Choose to edit the pipeline, or continue to display the pipeline in Edit mode.

3. Add a deployment action that will create a change set for the stack that was created in the previous action:

   a. In Action name, enter Change_Set. In Action provider, choose AWS CloudFormation.

   b. In Input artifact, choose SourceArtifact.

   c. In Action mode, choose Create or replace a change set.

   d. In Stack name, enter the variable syntax as shown. This is the name of the stack that the change set is created for, where the default namespace DeployVariables is assigned to the action.
Step 4: Add a manual approval action

Create a manual approval action in your pipeline.

1. Choose to edit the pipeline, or continue to display the pipeline in Edit mode.
2. Add a manual approval action after the deploy action that creates the change set. This action allows you to verify the created resource change set in AWS CloudFormation before the pipeline executes the change set.

Step 5: Add a CloudFormation deployment action to execute the change set

Create a next action in your pipeline that allows AWS CloudFormation to execute the change set after the manual approval action.


Under Pipelines, choose your pipeline and choose View. The diagram shows your pipeline source and deployment stages.

2. Choose to edit the pipeline, or continue to display the pipeline in Edit mode.

3. Add a deployment action that will execute the change set that was approved in the previous action:
   a. In Action name, enter Execute_Change_Set. In Action provider, choose AWS CloudFormation.
   b. In Input artifact, choose SourceArtifact.
   c. In Action mode, choose Execute a change set.
   d. In Stack name, enter the variable syntax as shown. This is the name of the stack that the change set is created for.

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

e. In Change set name, enter the name of the change set you created in the previous action.

   ```java
   my-changeset
   ```
f. Choose **Done** to save the action.

g. Continue the pipeline run.

**Step 6: Add a CloudFormation deployment action to delete the stack**

Create a final action in your pipeline that allows AWS CloudFormation to get the stack name from the variable in the outputs file and delete the stack.


   Under **Pipelines**, choose your pipeline and choose **View**. The diagram shows your pipeline source and deployment stages.

2. Choose to edit the pipeline.

3. Add a deployment action that will delete the stack:

   a. In **Action name**, choose **DeleteStack**. In **Deploy provider**, choose **CloudFormation**.

   b. In **Action mode**, choose **Delete a stack**.

   c. In **Stack name**, enter the variable syntax as shown. This is the name of the stack that the action will delete.

   ![CloudFormation deployment action settings](image)

   d. Choose **Done** to save the action.

   e. Choose **Save** to save the pipeline.
The pipeline runs when it is saved.

**Tutorial: Amazon ECS Standard Deployment with CodePipeline**

This tutorial helps you to create a complete, end-to-end continuous deployment (CD) pipeline with Amazon ECS with CodePipeline.

**Prerequisites**

There are a few resources that you must have in place before you can use this tutorial to create your CD pipeline. Here are the things you need to get started:

**Note**

All of these resources should be created within the same AWS Region.
Step 1: Add a Build Specification File to Your Source Repository

This tutorial uses CodeBuild to build your Docker image and push the image to Amazon ECR. Add a buildspec.yml file to your source code repository to tell CodeBuild how to do that. The example build specification below does the following:

- **Pre-build stage:**
  - Log in to Amazon ECR.
  - Set the repository URI to your ECR image and add an image tag with the first seven characters of the Git commit ID of the source.

- **Build stage:**
  - Build the Docker image and tag the image both as `latest` and with the Git commit ID.

- **Post-build stage:**
  - Push the image to your ECR repository with both tags.
  - Write a file called `imagedefinitions.json` in the build root that has your Amazon ECS service's container name and the image and tag. The deployment stage of your CD pipeline uses this information to create a new revision of your service's task definition, and then it updates the service to use the new task definition. The `imagedefinitions.json` file is required for the ECS job worker.

```yaml
version: 0.2
phases:
  install:
    runtime-versions:
      docker: 18
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - aws --version
      - $(aws ecr get-login --region $AWS_DEFAULT_REGION --no-include-email)
      - REPOSITORY_URI=012345678910.dkr.ecr.us-west-2.amazonaws.com/hello-world
      - COMMIT_HASH=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-7)
      - IMAGE_TAG=${COMMIT_HASH:=latest}
  build:
    commands:
      - echo Build started on `date`
```

After you have satisfied these prerequisites, you can proceed with the tutorial and create your CD pipeline.

Step 1: Add a Build Specification File to Your Source Repository

• A source control repository (this tutorial uses CodeCommit) with your Dockerfile and application source. For more information, see Create a CodeCommit Repository in the AWS CodeCommit User Guide.

• A Docker image repository (this tutorial uses Amazon ECR) that contains an image you have built from your Dockerfile and application source. For more information, see Creating a Repository and Pushing an Image in the Amazon Elastic Container Registry User Guide.

• An Amazon ECS task definition that references the Docker image hosted in your image repository. For more information, see Creating a Task Definition in the Amazon Elastic Container Service Developer Guide.

• An Amazon ECS cluster that is running a service that uses your previously mentioned task definition. For more information, see Creating a Cluster and Creating a Service in the Amazon Elastic Container Service Developer Guide.
Step 1: Add a Build Specification
File to Your Source Repository

- echo Building the Docker image...
- docker build -t $REPOSITORY_URI:latest .
- docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG

post_build:
  commands:
  - echo Build completed on `date`
  - echo Pushing the Docker images...
  - docker push $REPOSITORY_URI:latest
  - docker push $REPOSITORY_URI:$IMAGE_TAG
  - echo Writing image definitions file...
  - printf '[{"name":"hello-world","imageUri":%s}]' $REPOSITORY_URI:$IMAGE_TAG > imagedefinitions.json
  artifacts:
    files: imagedefinitions.json

The build specification was written for the following task definition, used by the Amazon ECS service for this tutorial. The $REPOSITORY_URI value corresponds to the image repository (without any image tag), and the hello-world value near the end of the file corresponds to the container name in the service's task definition.

```
{
  "taskDefinition": {
    "family": "hello-world",
    "containerDefinitions": [
      {
        "name": "hello-world",
        "image": "012345678910.dkr.ecr.us-west-2.amazonaws.com/hello-world:latest",
        "cpu": 100,
        "portMappings": [
          {
            "protocol": "tcp",
            "containerPort": 80,
            "hostPort": 80
          }
        ],
        "memory": 128,
        "essential": true
      }
    ]
  }
}
```

To add a buildspec.yml file to your source repository

1. Open a text editor and then copy and paste the build specification above into a new file.
2. Replace the $REPOSITORY_URI value (012345678910.dkr.ecr.us-west-2.amazonaws.com/hello-world) with your Amazon ECR repository URI (without any image tag) for your Docker image. Replace hello-world with the container name in your service's task definition that references your Docker image.
3. Commit and push your buildspec.yml file to your source repository.
   a. Add the file.
      
      `git add .`
   b. Commit the change.
      
      `git commit -m "Adding build specification."`
   c. Push the commit.
Step 2: Creating Your Continuous Deployment Pipeline

Use the CodePipeline wizard to create your pipeline stages and connect your source repository to your ECS service.

**To create your pipeline**

2. On the **Welcome** page, choose **Create pipeline**.
   
   If this is your first time using CodePipeline, an introductory page appears instead of **Welcome**. Choose **Get Started Now**.
3. On the **Step 1: Name** page, for Pipeline name, type the name for your pipeline and choose **Next**. For this tutorial, the pipeline name is **hello-world**.
4. On the **Step 2: Add source stage** page, for **Source provider**, choose **AWS CodeCommit**.
   a. For **Repository name**, choose the name of the CodeCommit repository to use as the source location for your pipeline.
   b. For **Branch name**, choose the branch to use and choose **Next**.
5. On the **Step 3: Add build stage** page, for **Build provider** choose **AWS CodeBuild**, and then choose **Create project**.
   a. For **Project name**, choose a unique name for your build project. For this tutorial, the project name is **hello-world**.
   b. For **Environment image**, choose **Managed image**.
   c. For **Operating system**, choose **Amazon Linux 2**.
   d. For **Runtime(s)**, choose **Standard**.
   e. For **Image**, choose **aws/codebuild/amazonlinux2-x86_64-standard:2.0**.
   f. For **Image version** and **Environment type**, use the default values.
   g. Select **Enable this flag if you want to build Docker images or want your builds to get elevated privileges**.
   h. Deselect **CloudWatch logs**.
   i. Choose **Continue to CodePipeline**.
   j. Choose **Next**.

   **Note**
   
   The wizard creates a CodeBuild service role for your build project, called **codebuild-build-project-name-service-role**. Note this role name, as you add Amazon ECR permissions to it later.
6. On the **Step 4: Add deploy stage** page, for Deployment provider, choose **Amazon ECS**.
   a. For **Cluster name**, choose the Amazon ECS cluster in which your service is running. For this tutorial, the cluster is **default**.
   b. For **Service name**, choose the service to update and choose **Next**. For this tutorial, the service name is **hello-world**.
7. On the **Step 5: Review** page, review your pipeline configuration and choose **Create pipeline** to create the pipeline.
Step 3: Add Amazon ECR Permissions to the CodeBuild Role

The CodePipeline wizard created an IAM role for the CodeBuild build project, called `code-build-build-project-name-service-role`. For this tutorial, the name is `code-build-hello-world-service-role`. Because the `buildspec.yml` file makes calls to Amazon ECR API operations, the role must have a policy that allows permissions to make these Amazon ECR calls. The following procedure helps you attach the proper permissions to the role.

To add Amazon ECR permissions to the CodeBuild role

2. In the left navigation pane, choose Roles.
3. In the search box, type `codebuild-` and choose the role that was created by the CodePipeline wizard. For this tutorial, the role name is `codebuild-hello-world-service-role`.
4. On the Summary page, choose Attach policies.
5. Select the box to the left of the `AmazonEC2ContainerRegistryPowerUser` policy, and choose Attach policy.

Step 4: Test Your Pipeline

Your pipeline should have everything for running an end-to-end native AWS continuous deployment. Now, test its functionality by pushing a code change to your source repository.

To test your pipeline

1. Make a code change to your configured source repository, commit, and push the change.
3. Choose your pipeline from the list.
4. Watch the pipeline progress through its stages. Your pipeline should complete and your Amazon ECS service runs the Docker image that was created from your code change.

Tutorial: Create a pipeline with an Amazon ECR source and ECS-to-CodeDeploy deployment

In this tutorial, you configure a pipeline in AWS CodePipeline that deploys container applications using a blue/green deployment that supports Docker images. In a blue/green deployment, you can launch the new version of your application alongside the old version and test the new version before you reroute traffic. You can also monitor the deployment process and rapidly rollback if there is an issue.

The completed pipeline detects changes to your image, which is stored in the Amazon ECR image repository, and uses CodeDeploy to route and deploy traffic to an Amazon ECS cluster and load balancer.
Prerequisites

CodeDeploy uses a listener to reroute traffic to the port of the updated container specified in the AppSpec file. For information about how the load balancer, production listener, target groups, and your Amazon ECS application are used in a blue/green deployment, see Tutorial: Deploy an Amazon ECS Service.

The pipeline is also configured to use a CodeCommit source location where your Amazon ECS task definition is stored. In this tutorial, you configure each of these AWS resources and then create your pipeline with stages that contain actions for each resource.

Your continuous delivery pipeline will automatically build and deploy container images whenever source code is changed or a new base image is uploaded to Amazon ECR.

This flow uses the following artifacts:

- A Docker image file that specifies the container name and repository URI of your Amazon ECR image repository.
- An Amazon ECS task definition that lists your Docker image name, container name, Amazon ECS service name, and load balancer configuration.
- A CodeDeploy AppSpec file that specifies the name of the Amazon ECS task definition file, the name of the updated application's container, and the container port where CodeDeploy reroutes production traffic. It can also specify optional network configuration and Lambda functions you can run during deployment lifecycle event hooks.

Note
When you commit a change to your Amazon ECR image repository, the pipeline source action creates an imageDetail.json file for that commit. For information about the imageDetail.json file, see imageDetail.json file for Amazon ECS blue/green deployment actions (p. 510).

When you create or edit your pipeline and update or specify source artifacts for your deployment stage, make sure to point to the source artifacts with the latest name and version you want to use. After you set up your pipeline, as you make changes to your image or task definition, you might need to update your source artifact files in your repositories and then edit the deployment stage in your pipeline.

Topics
- Prerequisites (p. 117)
- Step 1: Create image and push to an Amazon ECR repository (p. 118)
- Step 2: Create task definition and AppSpec source files and push to a CodeCommit repository (p. 119)
- Step 3: Create your Application Load Balancer and target groups (p. 121)
- Step 4: Create your Amazon ECS cluster and service (p. 123)
- Step 5: Create your CodeDeploy application and deployment group (ECS compute platform) (p. 124)
- Step 6: Create your pipeline (p. 125)
- Step 7: Make a change to your pipeline and verify deployment (p. 131)

Prerequisites

You must have already created the following resources:

- A CodeCommit repository. You can use the AWS CodeCommit repository you created in Tutorial: Create a simple pipeline (CodeCommit repository) (p. 52).
Step 1: Create image and push to an Amazon ECR repository

In this section, you use Docker to create an image and then use the AWS CLI to create an Amazon ECR repository and push the image to the repository.

Note
If you already have an image you want to use, you can skip this step.

To create an image

1. Sign in to your Linux instance where you have Docker installed.

   Pull down an image for nginx. This command provides the nginx:latest image from Docker Hub:

   ```
docker pull nginx
   ```

2. Run `docker images`. You should see the image in the list.

   ```
docker images
   ```

To create an Amazon ECR repository and push your image

1. Create an Amazon ECR repository to store your image. Make a note of the repositoryUri in the output.

   ```
aws ecr create-repository --repository-name nginx
   ```

   Output:

   ```
   {
   "repository": {
   "registryId": "aws_account_id",
   "repositoryName": "nginx",
   "repositoryArn": "arn:aws:ecr:us-east-1:aws_account_id:repository/nginx",
   "createdAt": 1505337806.0,
   "repositoryUri": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/nginx"
   }
   }
   ```

2. Tag the image with the repositoryUri value from the previous step.

   ```
docker tag nginx:latest aws_account_id.dkr.ecr.us-east-1.amazonaws.com/nginx:latest
   ```

3. Run the `aws ecr get-login-password` command, as shown in this example for the us-west-2 Region.

   ```
   aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin ACCOUNT_ID.dkr.ecr.us-west-2.amazonaws.com/nginx
   ```

4. Push the image to Amazon ECR using the repositoryUri from the earlier step.
Step 2: Create task definition and AppSpec source files and push to a CodeCommit repository

In this section, you create a task definition JSON file and register it with Amazon ECS. You then create an AppSpec file for CodeDeploy and use your Git client to push the files to your CodeCommit repository.

To create a task definition for your image

1. Create a file named taskdef.json with the following contents. For image, enter your image name, such as nginx. This value is updated when your pipeline runs.

   ```json
   {
   "executionRoleArn": "arn:aws:iam::account_ID:role/ecsTaskExecutionRole",
   "containerDefinitions": [
   {
   "name": "sample-website",
   "image": "nginx",
   "essential": true,
   "portMappings": [
   {
   "hostPort": 80,
   "protocol": "tcp",
   "containerPort": 80
   }
   ]
   },
   ],
   "requiresCompatibilities": [
   "FARGATE"
   ],
   "networkMode": "awsvpc",
   "cpu": "256",
   "memory": "512",
   "family": "ecs-demo"
   }
   }
   ``

   **Note**
   Make sure that the execution role specified in the task definition contains the AmazonECSTaskExecutionRolePolicy. For more information, see Amazon ECS Task Execution IAM Role in the Amazon ECS Developer Guide.

2. Register your task definition with the taskdef.json file.

   ```bash
   aws ecs register-task-definition --cli-input-json file://taskdef.json
   ``

3. After the task definition is registered, edit your file to remove the image name and include the <IMAGE1_NAME> placeholder text in the image field.

   ```json
   {
   "executionRoleArn": "arn:aws:iam::account_ID:role/ecsTaskExecutionRole",
   "containerDefinitions": [
   {
   "name": "sample-website",
   "image": "<IMAGE1_NAME>",
   "essential": true,
   ```

   **API Version 2015-07-09**

   119
Step 2: Create task definition and AppSpec source files and push to a CodeCommit repository

"portMappings": [
  {
    "hostPort": 80,
    "protocol": "tcp",
    "containerPort": 80
  }
],
"requiresCompatibilities": [
  "FARGATE"
],
"networkMode": "awsvpc",
"cpu": "256",
"memory": "512",
"family": "ecs-demo"
}

To create an AppSpec file

- The AppSpec file is used for CodeDeploy deployments. The file, which includes optional fields, uses this format:

```yaml
version: 0.0
Resources:
- TargetService:
  Type: AWS::ECS::Service
  Properties:
  TaskDefinition: "task-definition-ARN"
  LoadBalancerInfo:
    ContainerName: "container-name"
    ContainerPort: container-port-number
# Optional properties
NetworkConfiguration:
  AwsvpcConfiguration:
    Subnets: ["subnet-name-1", "subnet-name-2"]
    SecurityGroups: ["security-group"]
    AssignPublicIp: "ENABLED"
Hooks:
- BeforeInstall: "BeforeInstallHookFunctionName"
- AfterInstall: "AfterInstallHookFunctionName"
- BeforeAllowTestTraffic: "BeforeAllowTestTrafficHookFunctionName"
- AfterAllowTestTraffic: "AfterAllowTestTrafficHookFunctionName"
- BeforeAllowTraffic: "BeforeAllowTrafficHookFunctionName"
- AfterAllowTraffic: "AfterAllowTrafficHookFunctionName"
```

For more information about the AppSpec file, including examples, see CodeDeploy AppSpec File Reference.

Create a file named appspec.yaml with the following contents. For TaskDefinition, do not change the <TASK_DEFINITION> placeholder text. This value is updated when your pipeline runs.

```yaml
version: 0.0
Resources:
- TargetService:
  Type: AWS::ECS::Service
  Properties:
  TaskDefinition: <TASK_DEFINITION>
  LoadBalancerInfo:
    ContainerName: "sample-website"
    ContainerPort: 80
```

API Version 2015-07-09
To push files to your CodeCommit repository

1. Push or upload the files to your CodeCommit repository. These files are the source artifact created by the **Create pipeline** wizard for your deployment action in CodePipeline. Your files should look like this in your local directory:

```text
/tmp
  |my-demo-repo
  |-- appspec.yaml
  |-- taskdef.json
```

2. Choose the method you want to use to upload your files:
   a. To use your git command line from a cloned repository on your local computer:
      i. Change directories to your local repository:
         
         (For Linux, macOS, or Unix)  
         cd /tmp/my-demo-repo
         
         (For Windows)  
         cd c:\temp\my-demo-repo

      ii. Run the following command to stage all of your files at once:

         ```
         git add -A
         ```

      iii. Run the following command to commit the files with a commit message:

         ```
         git commit -m "Added task definition files"
         ```

      iv. Run the following command to push the files from your local repo to your CodeCommit repository:

         ```
         git push
         ```

   b. To use the CodeCommit console to upload your files:
      i. Open the CodeCommit console, and choose your repository from the **Repositories** list.
      ii. Choose **Add file**, and then choose **Upload file**.
      iii. Choose **Choose file**, and then browse for your file. Commit the change by entering your user name and email address. Choose **Commit changes**.
      iv. Repeat this step for each file you want to upload.

Step 3: Create your Application Load Balancer and target groups

In this section, you create an Amazon EC2 Application Load Balancer. You use the subnet names and target group values you create with your load balancer later, when you create your Amazon ECS service. You can create an Application Load Balancer or a Network Load Balancer. The load balancer must use a VPC with two public subnets in different Availability Zones. In these steps, you confirm your default VPC, create a load balancer, and then create two target groups for your load balancer. For more information, see [Target Groups for Your Network Load Balancers](https://docs.aws.amazon.com/elasticloadbalancing/latest/application/creating-target-groups.html).

To verify your default VPC and public subnets

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. Verify the default VPC to use. In the navigation pane, choose Your VPCs. Note which VPC shows Yes in the Default VPC column. This is the default VPC. It contains default subnets for you to select.

3. Choose Subnets. Choose two subnets that show Yes in the Default subnet column.

   Note
   Make a note of your subnet IDs. You need them later in this tutorial.

4. Choose the subnets, and then choose the Description tab. Verify that the subnets you want to use are in different Availability Zones.

5. Choose the subnets, and then choose the Route Table tab. To verify that each subnet you want to use is a public subnet, confirm that a gateway row is included in the route table.

To create an Amazon EC2 Application Load Balancer

1. Sign in to the AWS Management Console and open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

2. In the navigation pane, choose Load Balancers.

3. Choose Create Load Balancer.

4. Choose Application Load Balancer, and then choose Create.

5. In Name, enter the name of your load balancer.

6. In Scheme, choose internet-facing.

7. In IP address type, choose ipv4.

8. Configure two listener ports for your load balancer:

   b. Choose Add listener.

9. Under Availability Zones, in VPC, choose the default VPC. Next, choose the two default subnets you want to use.


12. Choose Select an existing security group, and make a note of the security group ID.


14. In Target group, choose New target group and configure your first target group:

   a. In Name, enter a target group name (for example, target-group-1).
   b. In Target type, choose IP.
   c. In Protocol choose HTTP. In Port, enter 80.
   d. Choose Next: Register Targets.

15. Choose Next: Review, and then choose Create.

To create a second target group for your load balancer

1. After your load balancer is provisioned, open the Amazon EC2 console. In the navigation pane, choose Target Groups.

2. Choose Create target group.

3. In Name, enter a target group name (for example, target-group-2).

4. In Target type, choose IP.


6. In VPC, choose the default VPC.
Step 4: Create your Amazon ECS cluster and service

In this section, you create an Amazon ECS cluster and service where CodeDeploy routes traffic during deployment (to an Amazon ECS cluster rather than EC2 instances). To create your Amazon ECS service, you must use the subnet names, security group, and target group value you created with your load balancer to create your service.

Note
When you use these steps to create your Amazon ECS cluster, you use the Networking only cluster template, which provisions AWS Fargate containers. AWS Fargate is a technology that manages your container instance infrastructure for you. You do not need to choose or manually create Amazon EC2 instances for your Amazon ECS cluster.

To create an Amazon ECS cluster

1. Open the Amazon ECS console at https://console.aws.amazon.com/ecs/.
2. In the navigation pane, choose Clusters.
3. Choose Create cluster.
4. Choose the Networking only cluster template that uses AWS Fargate, and then choose Next step.
5. Enter a cluster name on the Configure cluster page. You can add an optional tag for your resource. Choose Create.

To create an Amazon ECS service

Use the AWS CLI to create your service in Amazon ECS.

1. Create a JSON file and name it create-service.json. Paste the following into the JSON file.

   ```json
   {
   }
   ```

   For the taskDefinition field, when you register a task definition in Amazon ECS, you give it a family. This is similar to a name for multiple versions of the task definition, specified with a revision number. In this example, use "ecs-demo:1" for the family and revision number in your file. Use the subnet names, security group, and target group value you created with your load balancer in Step 3: Create your Application Load Balancer and target groups (p. 121).

   Note
   You need to include your target group ARN in this file. Open the Amazon EC2 console and from the navigation pane, under LOAD BALANCING, choose Target Groups. Choose your first target group. Copy your ARN from the Description tab.
Step 5: Create your CodeDeploy application and deployment group (ECS compute platform)

When you create a CodeDeploy application and deployment group for the Amazon ECS compute platform, the application is used during a deployment to reference the correct deployment group, target groups, listeners, and traffic rerouting behavior.

To create a CodeDeploy application

1. Open the CodeDeploy console and choose Create application.
2. In **Application name**, enter the name you want to use.
3. In **Compute platform**, choose **Amazon ECS**.
4. Choose **Create application**.

**To create a CodeDeploy deployment group**

1. On your application page's **Deployment groups** tab, choose **Create deployment group**.
2. In **Deployment group name**, enter a name that describes the deployment group.
3. In **Service role**, choose a service role that grants CodeDeploy access to Amazon ECS. To create a new service role, follow these steps:
   b. From the console dashboard, choose **Roles**.
   c. Choose **Create role**.
   d. Under **Select type of trusted entity**, select **AWS service**. Under **Choose a use case**, select **CodeDeploy**. Under **Select your use case**, select **CodeDeploy - ECS**. Choose **Next: Permissions**. The **AWSCodeDeployRoleForECS** managed policy is already attached to the role.
   e. Choose **Next: Tags**, and **Next: Review**.
   f. Enter a name for the role (for example, **CodeDeployECSRole**), and then choose **Create role**.
4. In **Environment configuration**, choose your Amazon ECS cluster name and service name.
5. From **Load balancers**, choose the name of the load balancer that serves traffic to your Amazon ECS service.
6. From **Production listener port**, choose the port and protocol for the listener that serves production traffic to your Amazon ECS service. From **Test listener port**, choose the port and protocol for the test listener.
7. From **Target group 1 name** and **Target group 2 name**, choose the target groups used to route traffic during your deployment. Make sure that these are the target groups you created for your load balancer.
8. Choose **Reroute traffic immediately** to determine how long after a successful deployment to reroute traffic to your updated Amazon ECS task.
9. Choose **Create deployment group**.

**Step 6: Create your pipeline**

In this section, you create a pipeline with the following actions:

- A CodeCommit action where the source artifacts are the task definition and the AppSpec file.
- A source stage with an Amazon ECR source action where the source artifact is the image file.
- A deployment stage with an Amazon ECS deploy action where the deployment runs with a CodeDeploy application and deployment group.

**To create a two-stage pipeline with the wizard**

2. On the **Welcome** page, **Getting started** page, or the **Pipelines** page, choose **Create pipeline**.
3. In **Step 1: Choose pipeline settings**, in **Pipeline name**, enter **MyImagePipeline**.
4. In **Service role**, choose **New service role** to allow CodePipeline to create a service role in IAM.
5. Leave the settings under **Advanced settings** at their defaults, and then choose **Next**.
6. In **Step 2: Add source stage**, in **Source provider**, choose **AWS CodeCommit**. In **Repository name**, choose the name of the CodeCommit repository you created in **Step 1: Create a CodeCommit repository** (p. 52). In **Branch name**, choose the name of the branch that contains your latest code update. Unless you created a different branch on your own, only **master** is available. Choose **Next**.

7. In **Step 3: Add build stage**, choose **Skip build stage**, and then accept the warning message by choosing **Skip** again. Choose **Next**.

8. In **Step 4: Add deploy stage**:
   a. In **Deploy provider**, choose **Amazon ECS (Blue/Green)**. In **Application name**, enter or choose the application name from the list, such as **codedeployapp**. In **Deployment group**, enter or choose the deployment group name from the list, such as **codedeploydeplgroup**.
**Note**
The name "Deploy" is the name given by default to the stage created in the Step 4: Deploy step, just as "Source" is the name given to the first stage of the pipeline.

b. Under **Amazon ECS task definition**, choose **SourceArtifact**. In the field, enter `taskdef.json`.
c. Under **AWS CodeDeploy AppSpec file**, choose **SourceArtifact**. In the field, enter `appspec.yaml`.

   **Note**
   At this point, do not fill in any information under **Dynamically update task definition image**.

d. Choose **Next**.

9. In **Step 5: Review**, review the information, and then choose **Create pipeline**.

### To add an Amazon ECR source action to your pipeline

View your pipeline and add an Amazon ECR source action to your pipeline.

1. Choose your pipeline. In the upper left, choose **Edit**.
2. In the source stage, choose **Edit stage**.
3. Add a parallel action by choosing **+ Add action** next to your CodeCommit source action.
4. In **Action name**, enter a name (for example, **Image**).
5. In **Action provider**, choose **Amazon ECR**.

![Edit action](image)

6. In **Repository name**, choose the name of your Amazon ECR repository.
7. In **Image tag**, specify the image name and version, if different from latest.
8. In **Output artifacts**, choose the output artifact default (for example, *MyImage*) that contains the image name and repository URI information you want the next stage to use.

9. Choose **Save** on the action screen. Choose **Done** on the stage screen. Choose **Save** on the pipeline. A message shows the Amazon CloudWatch Events rule to be created for the Amazon ECR source action.

**To wire your source artifacts to the deploy action**

1. Choose **Edit** on your Deploy stage and choose the icon to edit the **Amazon ECS (Blue/Green)** action.

2. Scroll to the bottom of the pane. In **Input artifacts**, choose **Add**. Add the source artifact from your new Amazon ECR repository (for example, *MyImage*).

3. In **Task Definition**, choose **SourceArtifact**, and then verify `taskdef.json` is entered.

4. In **AWS CodeDeploy AppSpec File**, choose **SourceArtifact**, and then verify `appspec.yaml` is entered.

5. In **Dynamically update task definition image**, in **Input Artifact with Image URI**, choose *MyImage*, and then enter the placeholder text that is used in the `taskdef.json` file: `IMAGE1_NAME`. Choose **Save**.
6. In the AWS CodePipeline pane, choose **Save pipeline change**, and then choose **Save change**. View your updated pipeline.

After this example pipeline is created, the action configuration for the console entries appears in the pipeline structure as follows:

```
"configuration": {
    "AppSpecTemplateArtifact": "SourceArtifact",
    "AppSpecTemplatePath": "appspec.yaml",
    "TaskDefinitionTemplateArtifact": "SourceArtifact",
    "TaskDefinitionTemplatePath": "taskdef.json",
    "ApplicationName": "codedeployapp",
    "DeploymentGroupName": "codedeploydeplgroup",
    "Image1ArtifactName": "MyImage",
    "Image1ContainerName": "IMAGE1_NAME"
},
```

7. To submit your changes and start a pipeline build, choose **Release change**, and then choose **Release**.
8. Choose the deployment action to view it in CodeDeploy and see the progress of the traffic shifting.

**Note**
You might see a deployment step that shows an optional wait time. By default, CodeDeploy waits one hour after a successful deployment before it terminates the original task set. You can use this time to roll back or terminate the task, but your deployment otherwise completes when the task set is terminated.

**Step 7: Make a change to your pipeline and verify deployment**

Make a change to your image and then push the change to your Amazon ECR repository. This triggers your pipeline to run. Verify that your image source change is deployed.

**Tutorial: Create a pipeline that deploys an Amazon Alexa skill**

In this tutorial, you configure a pipeline that continuously delivers your Alexa skill using the Alexa Skills Kit as the deployment provider in your deployment stage. The completed pipeline detects changes to your skill when you make a change to the source files in your source repository. The pipeline then uses the Alexa Skills Kit to deploy to the Alexa skill development stage.
Note
To create your custom skill as a Lambda function, see Host a Custom Skill as an AWS Lambda Function. You can also create a pipeline that uses Lambda source files and a CodeBuild project to deploy changes to Lambda for your skill. For example, to create a new skill and related Lambda function, you can create an AWS CodeStar project. See Create an Alexa Skill Project in AWS CodeStar. For that option, the pipeline includes a third build stage with an CodeBuild action and an action in the Deploy stage for AWS CloudFormation.

Prerequisites

You must already have the following:

- A CodeCommit repository. You can use the AWS CodeCommit repository you created in Tutorial: Create a simple pipeline (CodeCommit repository) on page 52.
- An Amazon developer account. This is the account that owns your Alexa skills. You can create an account for free at Alexa Skills Kit.
- An Alexa skill. You can create a sample skill using the Get Custom Skill Sample Code tutorial.
- Install the ASK CLI and configure it using `ask init` with your AWS credentials. See Install and initialize ASK CLI.

Step 1: Create an Alexa developer services LWA security profile

In this section, you create a security profile to use with Login with Amazon (LWA). If you already have a profile, you can skip this step.

- Use the steps in `generate-lwa-tokens` to create a Security Profile.
- After you create the profile, make a note of the Client ID and Client Secret.
- Make sure you enter the Allowed Return URLs as provided in the instructions. The URLs allow the ASK CLI command to redirect refresh token requests.

Step 2: Create Alexa skill source files and push to your CodeCommit repository

In this section, you create and push your Alexa skill source files to the repository that the pipeline uses for your source stage. For the skill you have created in the Amazon developer console, you produce and push the following:

- A skill.json file.
- An interactionModel/custom folder.

  Note
  This directory structure complies with Alexa Skills Kit skill package format requirements, as outlined in Skill package format. If your directory structure does not use the correct skill package format, changes do not successfully deploy to the Alexa Skills Kit console.

To create source files for your skill

1. Retrieve your skill ID from the Alexa Skills Kit developer console. Use this command:
Step 2: Create Alexa skill source files and push to your CodeCommit repository

2. Generate a `skill.json` file that contains your skill details. Use this command:

   ```
   ask api get-skill -s skill-ID > skill.json
   ```

3. (Optional) Create an `interactionModel/custom` folder.

   Use this command to generate the interaction model file within the folder. For locale, this tutorial uses en-US as the locale in the file name.

   ```
   ask api get-model --skill-id skill-ID --locale locale > ./interactionModel/custom/.locale.json
   ```

To push files to your CodeCommit repository

1. Push or upload the files to your CodeCommit repository. These files are the source artifact created by the Create Pipeline wizard for your deployment action in AWS CodePipeline. Your files should look like this in your local directory:

   ```
   skill.json
   /interactionModel
      /custom
         /en-US.json
   ```

2. Choose the method you want to use to upload your files:

   a. To use the Git command line from a cloned repository on your local computer:
      i. Run the following command to stage all of your files at once:

         ```
         git add -A
         ```
      ii. Run the following command to commit the files with a commit message:

         ```
         git commit -m "Added Alexa skill files"
         ```
      iii. Run the following command to push the files from your local repo to your CodeCommit repository:

         ```
         git push
         ```

   b. To use the CodeCommit console to upload your files:
      i. Open the CodeCommit console, and choose your repository from the Repositories list.
      ii. Choose Add file, and then choose Upload file.
      iii. Choose Choose file, and then browse for your file. Commit the change by entering your user name and email address. Choose Commit changes.
      iv. Repeat this step for each file you want to upload.
Step 3: Use ASK CLI commands to create a refresh token

CodePipeline uses a refresh token based on the client ID and secret in your Amazon developer account to authorize actions it performs on your behalf. In this section, you use the ASK CLI to create the token. You use these credentials when you use the Create Pipeline wizard.

To create a refresh token with your Amazon developer account credentials

1. Use the following command:
   
   ```
   ask util generate-lwa-tokens
   ```

2. When prompted, enter your client ID and secret as shown in this example:

   ```
   ? Please type in the client ID: amzn1.application-client.example112233445566
   ? Please type in the client secret: example112233445566
   ```

3. The sign-in browser page displays. Sign in with your Amazon developer account credentials.
4. Return to the command line screen. The access token and refresh token are generated in the output. Copy the refresh token returned in the output.

Step 4: Create your pipeline

In this section, you create a pipeline with the following actions:

- A source stage with a CodeCommit action where the source artifacts are the Alexa skill files that support your skill.
- A deployment stage with an Alexa Skills Kit deploy action.

To create a pipeline with the wizard

2. Choose the AWS Region where you want to create the project and its resources. The Alexa skill runtime is available only in the following Regions:
   - Asia Pacific (Tokyo)
   - Europe (Ireland)
   - US East (N. Virginia)
   - US West (Oregon)
3. On the Welcome page, Getting started page, or the Pipelines page, choose Create pipeline.
4. In Step 1: Choose pipeline settings, in Pipeline name, enter MyAlexaPipeline.
5. In Service role, choose New service role to allow CodePipeline to create a service role in IAM.
6. Leave the settings under Advanced settings at their defaults, and then choose Next.
7. In Step 2: Add source stage, in Source provider, choose AWS CodeCommit. In Repository name, choose the name of the CodeCommit repository you created in Step 1: Create a CodeCommit repository (p. 52). In Branch name, choose the name of the branch that contains your latest code update. Unless you created a different branch on your own, only master is available.
After you select the repository name and branch, a message shows the Amazon CloudWatch Events rule to be created for this pipeline.

Choose Next.

8. In Step 3: Add build stage, choose Skip build stage, and then accept the warning message by choosing Skip again.

Choose Next.

9. In Step 4: Add deploy stage:
   a. In Deploy provider, choose Alexa Skills Kit.
   b. In Alexa skill ID, enter the skill ID assigned to your skill in the Alexa Skills Kit developer console.
   c. In Client ID, enter the ID of the application you registered.
   d. In Client secret, enter the secret you chose when you registered.
   e. In Refresh token, enter the token you generated in step 3.
   f. Choose Next.

10. In Step 5: Review, review the information, and then choose Create pipeline.
Step 5: Make a change to any source file and verify deployment

Make a change to your skill and then push the change to your repository. This triggers your pipeline to run. Verify that your skill is updated in the Alexa Skills Kit developer console.

Tutorial: Create a pipeline that uses Amazon S3 as a deployment provider

In this tutorial, you configure a pipeline that continuously delivers files using Amazon S3 as the deployment action provider in your deployment stage. The completed pipeline detects changes when you make a change to the source files in your source repository. The pipeline then uses Amazon S3 to deploy the files to your bucket. Each time you modify, add, or delete your website files in your source location, the deployment creates the website with your latest files. This tutorial provides two options:

- Create a pipeline that deploys a static website to your S3 public bucket. This example creates a pipeline with an AWS CodeCommit source action and an Amazon S3 deployment action. See Option 1: Deploy static website files to Amazon S3 (p. 137).
Option 1: Deploy static website files to Amazon S3

In this example, you download the sample static website template file, upload the files to your AWS CodeCommit repository, create your bucket, and configure it for hosting. Next, you use the AWS CodePipeline console to create your pipeline and specify an Amazon S3 deployment configuration.

Prerequisites

You must already have the following:

- A CodeCommit repository. You can use the AWS CodeCommit repository you created in Tutorial: Create a simple pipeline (CodeCommit repository) (p. 52).
- Source files for your static website. Use this link to download a sample static website. The sample-website.zip download produces the following files:
  - An index.html file
  - A main.css file
  - A graphic.jpg file
- An S3 bucket configured for website hosting. See Hosting a static website on Amazon S3. Make sure you create your bucket in the same Region as the pipeline.

Note

To host a website, your bucket must have public read access, which gives everyone read access. With the exception of website hosting, you should keep the default access settings that block public access to S3 buckets.

Step 1: Push source files to your CodeCommit repository

In this section, push your source files to the repository that the pipeline uses for your source stage.

To push files to your CodeCommit repository

1. Extract the downloaded sample files. Do not upload the ZIP file to your repository.
2. Push or upload the files to your CodeCommit repository. These files are the source artifact created by the Create Pipeline wizard for your deployment action in CodePipeline. Your files should look like this in your local directory:

   index.html
   main.css
   graphic.jpg
3. You can use Git or the CodeCommit console to upload your files:
   a. To use the Git command line from a cloned repository on your local computer:
      i. Run the following command to stage all of your files at once:
         ```
         git add -A
         ```
      ii. Run the following command to commit the files with a commit message:
          ```
          git commit -m "Added static website files"
          ```
      iii. Run the following command to push the files from your local repo to your CodeCommit repository:
          ```
          git push
          ```
   b. To use the CodeCommit console to upload your files:
      i. Open the CodeCommit console, and choose your repository from the Repositories list.
      ii. Choose Add file, and then choose Upload file.
      iii. Select Choose file, and then browse for your file. Commit the change by entering your user name and email address. Choose Commit changes.
      iv. Repeat this step for each file you want to upload.

Step 2: Create your pipeline

In this section, you create a pipeline with the following actions:

- A source stage with a CodeCommit action where the source artifacts are the files for your website.
- A deployment stage with an Amazon S3 deployment action.

To create a pipeline with the wizard

2. On the Welcome page, Getting started page, or Pipelines page, choose Create pipeline.
3. In Step 1: Choose pipeline settings, in Pipeline name, enter MyS3DeployPipeline.
4. In Service role, choose New service role to allow CodePipeline to create a service role in IAM.
5. Leave the settings under Advanced settings at their defaults, and then choose Next.
6. In Step 2: Add source stage, in Source provider, choose AWS CodeCommit. In Repository name, choose the name of the CodeCommit repository you created in Step 1: Create a CodeCommit repository (p. 52). In Branch name, choose the name of the branch that contains your latest code update. Unless you created a different branch on your own, only master is available.

   After you select the repository name and branch, the Amazon CloudWatch Events rule to be created for this pipeline is displayed.

   Choose Next.
7. In Step 3: Add build stage, choose Skip build stage, and then accept the warning message by choosing Skip again.

   Choose Next.
8. In Step 4: Add deploy stage:
a. In **Deploy provider**, choose **Amazon S3**.

b. In **Bucket**, enter the name of your public bucket.

c. Select **Extract file before deploy**.

   **Note**
   The deployment fails if you do not select **Extract file before deploy**. This is because the AWS CodeCommit action in your pipeline zips source artifacts and your file is a ZIP file.

When **Extract file before deploy** is selected, **Deployment path** is displayed. Enter the name of the path you want to use. This creates a folder structure in Amazon S3 to which the files are extracted. For this tutorial, leave this field blank.

d. (Optional) In **Canned ACL**, you can apply a set of predefined grants, known as a **canned ACL**, to the uploaded artifacts.

e. (Optional) In **Cache control**, enter the caching parameters. You can set this to control caching behavior for requests/responses. For valid values, see the **Cache-Control** header field for HTTP operations.

f. Choose **Next**.

9. In **Step 5: Review**, review the information, and then choose **Create pipeline**.
10. After your pipeline runs successfully, open the Amazon S3 console and verify that your files appear in your public bucket as shown:

| index.html  
| main.css   
| graphic.jpg |

11. Access your endpoint to test the website. Your endpoint follows this format: http://bucket-name.s3-website-region.amazonaws.com/.


   The sample appears as shown here.
Step 3: Make a change to any source file and verify deployment

Make a change to your source files and then push the change to your repository. This triggers your pipeline to run. Verify that your website is updated.

Option 2: Deploy built archive files to Amazon S3 from an S3 source bucket

In this option, the build commands in your build stage compile TypeScript code into JavaScript code and deploy the output to your S3 target bucket under a separate timestamped folder. First, you create TypeScript code and a buildspec.yml file. After you combine the source files in a ZIP file, you upload the source ZIP file to your S3 source bucket, and use a CodeBuild stage to deploy a built application ZIP file to your S3 target bucket. The compiled code is retained as an archive in your target bucket.

Prerequisites

You must already have the following:

- An S3 source bucket. You can use the bucket you created in Tutorial: Create a simple pipeline (S3 bucket) (p. 38).
- An S3 target bucket. See Hosting a static website on Amazon S3. Make sure you create your bucket in the same AWS Region as the pipeline you want to create.

Note
This example demonstrates deploying files to a private bucket. Do not enable your target bucket for website hosting or attach any policies that make the bucket public.

Step 1: Create and upload source files to your S3 source bucket

In this section, you create and upload your source files to the bucket that the pipeline uses for your source stage. This section provides instructions for creating the following source files:

- A buildspec.yml file, which is used for CodeBuild build projects.
- An index.ts file.
To create a `buildspec.yml` file

- Create a file named `buildspec.yml` with the following contents. These build commands install TypeScript and use the TypeScript compiler to rewrite the code in `index.ts` to JavaScript code.

```
version: 0.2
phases:
  install:
    commands:
      - npm install -g typescript
  build:
    commands:
      - tsc index.ts
artifacts:
  files:
    - index.js
```

To create an `index.ts` file

- Create a file named `index.ts` with the following contents.

```
interface Greeting {
    message: string;
}

class HelloGreeting implements Greeting {
    message = "Hello!";
}

function greet(greeting: Greeting) {
    console.log(greeting.message);
}

let greeting = new HelloGreeting();
greet(greeting);
```

To upload files to your S3 source bucket

1. Your files should look like this in your local directory:

```
buildspec.yml
index.ts
```

Zip the files and name the file `source.zip`.

2. In the Amazon S3 console, for your source bucket, choose Upload. Choose Add files, and then browse for the ZIP file you created.

3. Choose Upload. These files are the source artifact created by the Create Pipeline wizard for your deployment action in CodePipeline. Your file should look like this in your bucket:

```
source.zip
```
Step 2: Create your pipeline

In this section, you create a pipeline with the following actions:

- A source stage with an Amazon S3 action where the source artifacts are the files for your downloadable application.
- A deployment stage with an Amazon S3 deployment action.

To create a pipeline with the wizard

2. On the Welcome page, Getting started page, or Pipelines page, choose Create pipeline.
3. In Step 1: Choose pipeline settings, in Pipeline name, enter MyS3DeployPipeline.
4. In Service role, choose New service role to allow CodePipeline to create a service role in IAM.
5. Leave the settings under Advanced settings at their defaults, and then choose Next.
6. In Step 2: Add source stage, in Source provider, choose Amazon S3. In Bucket, choose the name of your source bucket. In S3 object key, enter the name of your source ZIP file. Make sure you include the .zip file extension.

Choose Next.
7. In Step 3: Add build stage:
   a. In Build provider, choose AWS CodeBuild.
   b. Choose Create build project. On the Create project page:
      c. In Project name, enter a name for this build project.
   d. In Environment, choose Managed image. For Operating system, choose Ubuntu.
   e. For Runtime, choose Standard. For Runtime version, choose aws/codebuild/standard:1.0.
   f. In Image version, choose Always use the latest image for this runtime version.
   g. For Service role, choose your CodeBuild service role, or create one.
   h. For Build specifications, choose Use a buildspec file.
   i. Choose Continue to CodePipeline. A message is displayed if the project was created successfully.
   j. Choose Next.
8. In Step 4: Add deploy stage:
   a. In Deploy provider, choose Amazon S3.
   b. In Bucket, enter the name of your S3 target bucket.
   c. Make sure that Extract file before deploy is cleared.

When Extract file before deploy is cleared, S3 object key is displayed. Enter the name of the path you want to use: js-application/{datetime}.zip.

This creates a js-application folder in Amazon S3 to which the files are extracted. In this folder, the {datetime} variable creates a timestamp on each output file when your pipeline runs.
Option 2: Deploy built archive files to Amazon S3 from an S3 source bucket

- (Optional) In **Canned ACL**, you can apply a set of predefined grants, known as a **canned ACL**, to the uploaded artifacts.

- (Optional) In **Cache control**, enter the caching parameters. You can set this to control caching behavior for requests/responses. For valid values, see the **Cache-Control** header field for HTTP operations.

- Choose **Next**.

9. In **Step 5: Review**, review the information, and then choose **Create pipeline**.
Option 2: Deploy built archive files to Amazon S3 from an S3 source bucket

10. After your pipeline runs successfully, view your bucket in the Amazon S3 console. Verify that your deployed ZIP file is displayed in your target bucket under the `js-application` folder. The JavaScript file contained in the ZIP file should be `index.js`. The `index.js` file contains the following output:

```javascript
var HelloGreeting = /** @class */ (function () {
    function HelloGreeting() {
        this.message = "Hello!";
    }
    return HelloGreeting;
})();
function greet(greeting) {
    console.log(greeting.message);
}
```
Step 3: Make a change to any source file and verify deployment

Make a change to your source files and then upload them to your source bucket. This triggers your pipeline to run. View your target bucket and verify that the deployed output files are available in the `js-application` folder as shown:

![Image of Amazon S3 bucket with files]

Tutorial: Create a pipeline that publishes your serverless application to the AWS Serverless Application Repository

You can use AWS CodePipeline to continuously deliver your AWS SAM serverless application to the AWS Serverless Application Repository.

This tutorial shows how to create and configure a pipeline to build your serverless application that is hosted in GitHub and publish it to the AWS Serverless Application Repository automatically. The pipeline uses GitHub as the source provider and CodeBuild as the build provider. To publish your serverless application to the AWS Serverless Application Repository, you deploy an application (from the AWS Serverless Application Repository) and associate the Lambda function created by that application as an Invoke action provider in your pipeline. Then you can continuously deliver application updates to the AWS Serverless Application Repository, without writing any code.
Important
Many of the actions you add to your pipeline in this procedure involve AWS resources that you need to create before you create the pipeline. AWS resources for your source actions must always be created in the same AWS Region where you create your pipeline. For example, if you create your pipeline in the US East (Ohio) Region, your CodeCommit repository must be in the US East (Ohio) Region.
You can add cross-region actions when you create your pipeline. AWS resources for cross-region actions must be in the same AWS Region where you plan to execute the action. For more information, see Add a cross-Region action in CodePipeline (p. 364).

Before you begin
In this tutorial, we assume the following.

- You are familiar with AWS Serverless Application Model (AWS SAM) and the AWS Serverless Application Repository.
- You have a serverless application hosted in GitHub that you have published to the AWS Serverless Application Repository using the AWS SAM CLI. To publish an example application to the AWS Serverless Application Repository, see Quick Start: Publishing Applications in the AWS Serverless Application Repository Developer Guide. To publish your own application to the AWS Serverless Application Repository, see Publishing Applications Using the AWS SAM CLI in the AWS Serverless Application Model Developer Guide.

Step 1: Create a buildspec.yml file
Create a buildspec.yml file with the following contents, and add it to your serverless application's GitHub repository. Replace template.yml with your application's AWS SAM template and bucketname with the S3 bucket where your packaged application is stored.

```
version: 0.2
phases:
  install:
    runtime-versions:
      python: 3.8
  build:
    commands:
      - pip install --upgrade pip
      - pip install pipenv --user
      - pipenv install awscli aws-sam-cli
      - pipenv run sam package --template-file template.yml --s3-bucket bucketname --output-template-file packaged-template.yml
    artifacts:
      files:
        - packaged-template.yml
```

Step 2: Create and configure your pipeline
Follow these steps to create your pipeline in the AWS Region where you want to publish your serverless application.

1. Sign in to the AWS Management Console and open the CodePipeline console at https://console.aws.amazon.com/codepipeline/.
2. If necessary, switch to the AWS Region where you want to publish your serverless application.
3. Choose Create pipeline. On the Choose pipeline settings page, in Pipeline name, enter the name for your pipeline.
4. In Service role, choose New service role to allow CodePipeline to create a service role in IAM.
5. Leave the settings under Advanced settings at their defaults, and then choose Next.
6. On the Add source stage page, in Source provider, choose GitHub, and then choose Connect to GitHub.
7. In the browser window, choose Authorize aws-codesuite. This allows your pipeline to make your repository a source, and to use webhooks that detect when new code is pushed to the repository.
8. In Repository, choose your GitHub source repository.
9. In Branch, choose your GitHub branch.
10. Choose Next.
11. On the Add build stage page, add a build stage:
   a. In Build provider, choose AWS CodeBuild. For Region, use the pipeline Region.
   b. Choose Create project.
   c. In Project name, enter a name for this build project.
   d. In Environment image, choose Managed image. For Operating system, choose Ubuntu.
   e. For Runtime and Runtime version, choose the runtime and version required for your serverless application.
   f. For Service role, choose New service role.
   g. For Build specifications, choose Use a buildspec file.
   h. Choose Continue to CodePipeline. This opens the CodePipeline console and creates a CodeBuild project that uses the buildspec.yml in your repository for configuration. The build project uses a service role to manage AWS service permissions. This step might take a couple of minutes.
   i. Choose Next.
12. On the Add deploy stage page, choose Skip deploy stage, and then accept the warning message by choosing Skip again. Choose Next.
13. Choose Create pipeline. You should see a diagram that shows the source and build stages.
14. Grant the CodeBuild service role permission to access the S3 bucket where your packaged application is stored.
   a. In the Build stage of your new pipeline, choose CodeBuild.
   b. Choose the Build details tab.
   c. In Environment, choose the CodeBuild service role to open the IAM console.
   d. Expand the selection for CodeBuildBasePolicy, and choose Edit policy.
   e. Choose JSON.
   f. Add a new policy statement with the following contents. The statement allows CodeBuild to put objects into the S3 bucket where your packaged application is stored. Replace \textit{bucketname} with the name of your S3 bucket.

   ```json
   {
   "Effect": "Allow",
   "Resource": [
   "arn:aws:s3:::bucketname/**
   ],
   "Action": [
   "s3:PutObject"
   ]
   }
   ```
   g. Choose Review policy.
   h. Choose Save changes.
Step 3: Deploy the publish application

Follow these steps to deploy the application that contains the Lambda function that performs the publish to the AWS Serverless Application Repository. This application is `aws-serverless-codepipeline-serverlessrepo-publish`.

**Note**
You must deploy the application to the same AWS Region as your pipeline.

1. Go to the application page, and choose Deploy.
2. Select I acknowledge that this app creates custom IAM roles.
3. Choose Deploy.
4. Choose View AWS CloudFormation Stack to open the AWS CloudFormation console.
5. Expand the Resources section. You see `ServerlessRepoPublish`, which is of the type `AWS::Lambda::Function`. Make a note of the physical ID of this resource for the next step. You use this physical ID when you create the new publish action in CodePipeline.

Step 4: Create the publish action

Follow these steps to create the publish action in your pipeline.

2. In the left navigation section, choose the pipeline that you want to edit.
3. Choose Edit.
4. After the last stage of your current pipeline, choose + Add stage. In Stage name enter a name, such as Publish, and choose Add stage.
5. In the new stage, choose + Add action group.
6. Enter an action name. From Action provider, in Invoke, choose AWS Lambda.
7. From Input artifacts, choose BuildArtifact.
8. From Function name, choose the physical ID of the Lambda function that you noted in the previous step.
9. Choose Save for the action.
10. Choose Done for the stage.
11. In the upper right, choose Save.
12. To verify your pipeline, make a change to your application in GitHub. For example, change the application's description in the Metadata section of your AWS SAM template file. Commit the change and push it to your GitHub branch. This triggers your pipeline to run. When the pipeline is complete, check that your application has been updated with your change in the AWS Serverless Application Repository.

Tutorial: Using variables with Lambda invoke actions

A Lambda invoke action can use variables from another action as part of its input and return new variables along with its output. For information about variables for actions in CodePipeline, see Variables (p. 514).

At the end of this tutorial, you will have:

- A Lambda invoke action that:
• Consumes the CommitId variable from a CodeCommit source action
• Outputs three new variables: dateTime, testRunId, and region
• A manual approval action that consumes the new variables from your Lambda invoke action to provide a test URL and a test run ID
• A pipeline updated with the new actions

Topics
• Prerequisites (p. 150)
• Step 1: Create a Lambda function (p. 150)
• Step 2: Add a Lambda invoke action and manual approval action to your pipeline (p. 152)

Prerequisites
Before you begin, you must have the following:

• You can create or use the pipeline with the CodeCommit source in Tutorial: Create a simple pipeline (CodeCommit repository) (p. 52).
• Edit your existing pipeline so that the CodeCommit source action has a namespace. Assign the namespace SourceVariables to the action.

Step 1: Create a Lambda function
Use the following steps to create a Lambda function and a Lambda execution role. You add the Lambda action to your pipeline after you create the Lambda function.

To create a Lambda function and execution role
1. Sign in to the AWS Management Console and open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose Create function. Leave Author from scratch selected.
3. In Function name, enter the name of your function, such as myInvokeFunction. In Runtime, leave the default option selected.
4. Expand Choose or create an execution role. Choose Create a new role with basic Lambda permissions.
5. Choose Create function.
6. To use a variable from another action, it will have to be passed to the UserParameters in the Lambda invoke action configuration. You will be configuring the action in our pipeline later in the tutorial, but you will add the code assuming the variable will be passed.

```javascript
const commitId = event['CodePipeline.job'].data.actionConfiguration.configuration.UserParameters;
```

To produce new variables, set a property called outputVariables on the input to putJobSuccessResult. Note that you cannot produce variables as part of a putJobFailureResult.

```javascript
const successInput = {
  jobId: jobId,
  outputVariables: {
    testRunId: Math.floor(Math.random() * 1000).toString(),
    dateTime: Date(Date.now()).toString(),
```
In your new function, leave **Edit code inline** selected, and paste the following example code under `index.js`.

```javascript
var AWS = require('aws-sdk');
exports.handler = function(event, context) {
    var codepipeline = new AWS.CodePipeline();

    // Retrieve the Job ID from the Lambda action
    var jobId = event['CodePipeline.job'].id;

    // Retrieve the value of UserParameters from the Lambda action configuration in AWS CodePipeline,
    // in this case it is the Commit ID of the latest change of the pipeline.
    var commitId = event['CodePipeline.job'].data.actionConfiguration.configuration.UserParameters;

    // The region from where the lambda function is being executed.
    var lambdaRegion = process.env.AWS_REGION;

    // Notify AWS CodePipeline of a successful job
    var putJobSuccess = function(message) {
        var jobSuccessInput = {
            jobId: jobId,
            outputVariables: {
                testRunId: Math.floor(Math.random() * 1000).toString(),
                dateTime: Date(Date.now()).toString(),
                region: lambdaRegion
            }
        }
        codepipeline.putJobSuccessResult(params, function(err, data) {
            if(err) {
                context.fail(err);
            } else {
                context.succeed(message);
            }
        });
    }

    // Notify AWS CodePipeline of a failed job
    var putJobFailure = function(message) {
        var jobFailureInput = {
            jobId: jobId,
            failureDetails: {
                message: JSON.stringify(message),
                type: 'JobFailed',
                externalExecutionId: context.invokeid
            }
        }
        codepipeline.putJobFailureResult(params, function(err, data) {
            context.fail(message);
        });
    }

    var sendResult = function() {
        try {
            console.log("Testing commit - " + commitId);
            // Your tests here
        } catch (err) {
            context.fail(err);
        }
    }
};
```
Step 2: Add a Lambda invoke action and manual approval action to your pipeline

In this step, you add a Lambda invoke action to your pipeline. You add the action as part of a stage named Test. The action type is an invoke action. You then add a manual approval action after the invoke action.

To add a Lambda action and a manual approval action to the pipeline


   The names of all pipelines that are associated with your AWS account are displayed. Choose the pipeline where you want to add the action.

2. Add the Lambda test action to your pipeline.

   a. To edit your pipeline, choose Edit. Add a stage after the source action in the existing pipeline. Enter a name for the stage, such as Test.

   b. In the new stage, choose the icon to add an action. In Action name, enter the name of the invoke action, such as Test_Commit.

   c. In Action provider, choose AWS Lambda.

   d. In Input artifacts, choose the name of your source action's output artifact, such as SourceArtifact.

   e. In Function name, choose the name of the Lambda function that you created.

   f. In User parameters, enter the variable syntax for the CodeCommit commit ID. This creates the output variable that resolves to the commit to be reviewed and approved each time the pipeline is run.

      \#{SourceVariables.CommitId}

   g. In Variable namespace, add the namespace name, such as TestVariables.
Input artifacts
Choose an input artifact for this action. Learn more

SourceArtifact
Add
No more than 100 characters

Function name
Choose a function that you have already created in the AWS Lambda console. Or create a function in the Amazon Lambda console and then return to this task.

LambdaInvokeVariableFunction

User parameters - optional
This string will be used in the event data parameter passed to the handler in AWS Lambda.

#{SourceVariables.CommitId}

Variable namespace - optional
Choose a namespace for the output variables from this action. You must choose a namespace if you want to use the variables this action produces in your configuration. Learn more

TestVariables

Output artifacts
Choose a name for the output of this action.

h. Choose Done.

3. Add the manual approval action to your pipeline.

a. With your pipeline still in editing mode, add a stage after the invoke action. Enter a name for the stage, such as Approval.

b. In the new stage, choose the icon to add an action. In Action name, enter the name of the approval action, such as Change_Approval.


d. In URL for review, construct the URL by adding the variable syntax for the region variable and the CommitId variable. Make sure that you use the namespaces assigned to the actions that provide the output variables.

For this example, the URL with the variable syntax for a CodeCommit action has the default namespace SourceVariables. The Lambda region output variable has the TestVariables namespace. The URL looks like the following.

https://#{TestVariables.region}.console.aws.amazon.com/codesuite/codecommit/repositories/MyDemoRepo/commit/#{SourceVariables.CommitId}

In Comments, construct the approval message text by adding the variable syntax for the testRunId variable. For this example, the URL with the variable syntax for the Lambda testRunId output variable has the TestVariables namespace. Enter the following message.

API Version 2015-07-09
153
Step 2: Add a Lambda invoke action and manual approval action to your pipeline

Make sure to review the code before approving this action. Test Run ID: ${TestVariables.testRunId}

URL for review - optional
Type the URL you want to provide to the reviewer as part of the approval request. The URL must begin with 'http://' or 'https://'.

https://${TestVariables.region}.console.aws.amazon.com/codesuite/coderepo/repositories/lambda:

Comments - optional
Comments you type here display for the reviewer in email notifications or the console.

Variable namespace - optional
Choose a namespace for the output variables from this action. You must choose a namespace if you want to use the variables in the configuration. Learn more

4. Choose **Done** to close the edit screen for the action, and then choose **Done** to close the edit screen for the stage. To save the pipeline, choose **Done**. The completed pipeline now contains a structure with source, test, approval, and deploy stages.
Choose **Release change** to run the latest change through the pipeline structure.

5. When the pipeline reaches the manual approval stage, choose **Review**. The resolved variables appear as the URL for the commit ID. Your approver can choose the URL to view the commit.
6. After the pipeline runs successfully, you can also view the variable values on the action execution history page.

Tutorial: Use an AWS Step Functions invoke action in a pipeline

You can use AWS Step Functions to create and configure state machines. This tutorial shows you how to add an invoke action to a pipeline that activates state machine executions from your pipeline.

In this tutorial, you do the following tasks:

- Create a standard state machine in AWS Step Functions.
- Enter the state machine input JSON directly. You can also upload the state machine input file to an Amazon Simple Storage Service (Amazon S3) bucket.
- Update your pipeline by adding the state machine action.

Topics

- Prerequisite: Create or choose a simple pipeline (p. 157)
Prerequisite: Create or choose a simple pipeline

In this tutorial, you add an invoke action to an existing pipeline. You can use the pipeline you created in Tutorial: Create a simple pipeline (S3 bucket) (p. 38) or Tutorial: Create a simple pipeline (CodeCommit repository) (p. 52).

You use an existing pipeline with a source action and at least a two-stage structure, but you do not use source artifacts for this example.

Note
You might need to update the service role used by your pipeline with additional permissions required to run this action. To do this, open the AWS Identity and Access Management (IAM) console, find the role, and then add the permissions to the role's policy. For more information, see Add permissions to the CodePipeline service role (p. 446).

Step 1: Create the sample state machine

In the Step Functions console, create a state machine using the HelloWorld sample template. For instructions, see Create a State Machine in the AWS Step Functions Developer Guide.

Step 2: Add a Step Functions invoke action to your pipeline

Add a Step Functions invoke action to your pipeline as follows:

2. In Name, choose the name of the pipeline you want to edit. This opens a detailed view of the pipeline, including the state of each of the actions in each stage of the pipeline.
3. On the pipeline details page, choose Edit.
4. On the second stage of your simple pipeline, choose Edit stage. Choose Delete. This deletes the second stage now that you no longer need it.
5. At the bottom of the diagram, choose + Add stage.
6. In Stage name, enter a name for the stage, such as Invoke, and then choose Add stage.
7. Choose + Add action group.
8. In Action name, enter a name, such as Invoke.
9. In Action provider, choose AWS Step Functions. Allow Region to default to the pipeline Region.
10. In Input artifacts, choose SourceArtifact.
11. In State machine ARN, choose the Amazon Resource Name (ARN) for the state machine that you created earlier.
12. (Optional) In Execution name prefix, enter a prefix to be added to the state machine execution ID.
13. In Input type, choose Literal.
14. In Input, enter the input JSON that the HelloWorld sample state machine expects.
Note
The input to the state machine execution is different from the term used in CodePipeline to describe input artifacts for actions.

For this example, enter the following JSON:

```
{"IsHelloWorldExample": true}
```

15. Choose **Done**.

16. On the stage that you’re editing, choose **Done**. In the AWS CodePipeline pane, choose **Save**, and then choose **Save** on the warning message.

17. To submit your changes and start a pipeline execution, choose **Release change**, and then choose **Release**.
Tutorial: Create a pipeline that uses AWS AppConfig as a deployment provider

In this tutorial, you configure a pipeline that continuously delivers configuration files using AWS AppConfig as the deployment action provider in your deployment stage.

Topics

- Prerequisites (p. 160)
- Step 1: Create your AWS AppConfig resources (p. 160)
- Step 2: Upload files to your S3 source bucket (p. 160)
• Step 3: Create your pipeline (p. 161)
• Step 4: Make a change to any source file and verify deployment (p. 163)

Prerequisites

Before you begin, you must complete the following:

• Create or use an Amazon S3 bucket with versioning enabled. Follow the instructions in Step 1: Create an S3 bucket for your application (p. 38) to create an S3 bucket.

Step 1: Create your AWS AppConfig resources

In this section, you create the following resources:

• An application in AWS AppConfig is a logical unit of code that provides capabilities for your customers.
• An environment in AWS AppConfig is a logical deployment group of AppConfig targets, such as applications in a beta or production environment.
• A configuration profile is a collection of settings that influence the behavior of your application. The configuration profile enables AWS AppConfig to access your configuration in its stored location.
• (Optional) A deployment strategy in AWS AppConfig defines the behavior of a configuration deployment, such as what percentage of clients should receive the new deployed config at any given time during a deployment.

To create an application, environment, configuration profile, and deployment strategy

1. Sign in to the AWS Management Console.
2. Use the steps in the following topics to create your resources in AWS AppConfig.
   • Create an application.
   • Create an environment.
   • Create an AWS CodePipeline configuration profile.
   • (Optional) Choose a predefined deployment strategy or create your own.

Step 2: Upload files to your S3 source bucket

In this section, zip and push your source files to the bucket that the pipeline uses for your source stage.

To create configuration files

1. Create a configuration.json file for each configuration in each Region. Include the following contents:

   ```json
   Hello World!
   ```

2. In the Amazon S3 console for your bucket, choose Upload, and follow the instructions to upload your .zip file.

To zip and upload source files

1. Create a .zip file with your files and name the .zip file configuration-files.zip. As an example, your .zip file can use the following structure:
2. In the Amazon S3 console for your bucket, choose **Upload**, and follow the instructions to upload your .zip file.

### Step 3: Create your pipeline

In this section, you create a pipeline with the following actions:

- A source stage with a CodeCommit action where the source artifacts are the files for your website.
- A deployment stage with an Amazon S3 deployment action.

#### To create a pipeline with the wizard

2. On the **Welcome** page, **Getting started** page, or **Pipelines** page, choose **Create pipeline**.
3. In **Step 1: Choose pipeline settings**, in **Pipeline name**, enter **MyS3DeployPipeline**.
4. In **Service role**, choose **New service role** to allow CodePipeline to create a service role in IAM.
5. Leave the settings under **Advanced settings** at their defaults, and then choose **Next**.
6. In **Step 2: Add source stage**, in **Source provider**, choose **Amazon S3**. In **Bucket**, choose the name of your S3 source bucket.

   In **S3 object key**, enter the name of your .zip file: **configuration-files.zip**.

   Choose **Next**.

7. In **Step 3: Add build stage**, choose **Skip build stage**, and then accept the warning message by choosing **Skip** again.

   Choose **Next**.

8. In **Step 4: Add deploy stage**:
   a. In **Deploy provider**, choose **AWS AppConfig**.
   b. In **Application**, choose the name of the application you created in AWS AppConfig. The field shows the ID for your application.
   c. In **Environment**, choose the name of the environment you created in AWS AppConfig. The field shows the ID for your environment.
   d. In **Configuration profile**, choose the name of the configuration profile you created in AWS AppConfig. The field shows the ID for your configuration profile.
   e. In **Deployment strategy**, choose the name of your deployment strategy. This can be either a deployment strategy you created in AppConfig or one you have chosen from predefined deployment strategies in AppConfig. The field shows the ID for your deployment strategy.
   f. In **Input artifact configuration path**, enter the file path. Make sure that your input artifact configuration path matches the directory structure in your S3 bucket.
g. Choose Next.

9. In Step 5: Review, review the information, and then choose Create pipeline.
Step 4: Make a change to any source file and verify deployment

Make a change to your source files and upload the change to your bucket. This triggers your pipeline to run. Verify that your configuration is available by viewing the version.
CodePipeline best practices and use cases

AWS CodePipeline is integrated with a number of products and services. The following sections describe best practices and use cases for CodePipeline and these related products and services.

A simple business use case for CodePipeline can help you understand ways you might implement the service and control user access. The use cases are described in general terms. They do not prescribe the APIs to use to achieve the results you want.

Topics
• Best practices (p. 164)
• Use cases for CodePipeline (p. 165)

Best practices

Use the best practices outlined in these sections when using CodePipeline.

Security best practices for CodePipeline resources

You use encryption and authentication for the source repositories that connect to your pipelines. For CodePipeline best practices for security, see Security best practices (p. 452).

Monitoring and logging best practices for CodePipeline resources

You can use logging features in AWS to determine the actions users have taken in your account and the resources that were used. The log files show:

• The time and date of actions.
• The source IP address for an action.
• Which actions failed due to inadequate permissions.

Logging features are available in the following AWS services:

• AWS CloudTrail can be used to log AWS API calls and related events made by or on behalf of an AWS account. For more information, see Logging CodePipeline API calls with AWS CloudTrail (p. 394).
• Amazon CloudWatch Events can be used to monitor your AWS Cloud resources and the applications you run on AWS. You can create alerts in Amazon CloudWatch Events based on metrics that you define. For more information, see Detect and react to changes in pipeline state with Amazon CloudWatch Events (p. 384).

Best practices for the Jenkins plugin

Use the best practices provided in this section for pipelines with a Jenkins action provider.
Configure a separate Amazon EC2 instance and IAM role for your Jenkins build server

As a best practice, when you use a Jenkins build provider for your pipeline's build or test action, install Jenkins on an Amazon EC2 instance and configure a separate EC2 instance profile. Make sure the instance profile grants Jenkins only the AWS permissions required to perform tasks for your project, such as retrieving files from Amazon S3.

The instance profile provides applications running on an Amazon EC2 instance with the credentials to access other AWS services. As a result, you do not need to configure AWS credentials (AWS access key and secret key).

To learn how to create the role for your Jenkins instance profile, see the steps in Create an IAM role to use for Jenkins integration (p. 64).

Use cases for CodePipeline

You can create pipelines that integrate with other AWS services. These can be AWS services, such as Amazon S3, or third-party products, such as GitHub. This section provides examples for using CodePipeline to automate your code releases using different product integrations. For a full list of integrations with CodePipeline organized by action type, see CodePipeline pipeline structure reference (p. 454).

Topics

- Use CodePipeline with Amazon S3, AWS CodeCommit, and AWS CodeDeploy (p. 165)
- Use CodePipeline with third-party action providers (GitHub and Jenkins) (p. 166)
- Use CodePipeline with AWS CodeStar to build a pipeline in a code project (p. 166)
- Use CodePipeline to compile, build, and test code with CodeBuild (p. 166)
- Use CodePipeline with Amazon ECS for continuous delivery of container-based applications to the cloud (p. 167)
- Use CodePipeline with Elastic Beanstalk for continuous delivery of web applications to the cloud (p. 167)
- Use CodePipeline with AWS Lambda for continuous delivery of Lambda-based and serverless applications (p. 167)
- Use CodePipeline with AWS CloudFormation templates for continuous delivery to the cloud (p. 167)

Use CodePipeline with Amazon S3, AWS CodeCommit, and AWS CodeDeploy

When you create a pipeline, CodePipeline integrates with AWS products and services that act as action providers in each stage of your pipeline. When you choose stages in the wizard, you must choose a source stage and at least a build or deploy stage. The wizard creates the stages for you with default names that cannot be changed. These are the stage names created when you set up a full three-stage pipeline in the wizard:

- A source action stage with a default name of “Source.”
- A build action stage with a default name of “Build.”
- A deploy action stage with a default name of “Staging.”

You can use the tutorials in this guide to create pipelines and specify stages:

- The steps in Tutorial: Create a simple pipeline (S3 bucket) (p. 38) help you use the wizard to create a pipeline with two default stages: “Source” and “Staging”, where your Amazon S3 repository is the source provider. This tutorial creates a pipeline that uses AWS CodeDeploy to deploy a sample application from an Amazon S3 bucket to Amazon EC2 instances running Amazon Linux.
- The steps in Tutorial: Create a simple pipeline (CodeCommit repository) (p. 52) help you use the wizard to create a pipeline with a “Source” stage that uses your AWS CodeCommit repository as the source provider. This tutorial creates a pipeline that uses AWS CodeDeploy to deploy a sample application from an AWS CodeCommit repository to an Amazon EC2 instance running Amazon Linux.

Use CodePipeline with third-party action providers (GitHub and Jenkins)

You can create pipelines that integrate with third-party products such as GitHub and Jenkins. The steps in Tutorial: Create a four-stage pipeline (p. 63) show you how to create a pipeline that:

- Gets source code from a GitHub repository,
- Uses Jenkins to build and test the source code,
- Uses AWS CodeDeploy to deploy the built and tested source code to Amazon EC2 instances running Amazon Linux or Microsoft Windows Server.

Use CodePipeline with AWS CodeStar to build a pipeline in a code project

AWS CodeStar is a cloud-based service that provides a unified user interface for managing software development projects on AWS. AWS CodeStar works with CodePipeline to combine AWS resources into a project development toolchain. You can use your AWS CodeStar dashboard to automatically create the pipeline, repositories, source code, build spec files, deployment method, and hosting instances or serverless instances required for a complete code project.

To create your AWS CodeStar project, you choose your coding language and the type of application you want to deploy. You can create the following project types: a web application, a web service, or an Alexa skill.

At any time, you can integrate your preferred IDE into your AWS CodeStar dashboard. You can also add and remove team members and manage permissions for team members on your project. For a tutorial that shows you how to use AWS CodeStar to create a sample pipeline for a serverless application, see Tutorial: Creating and Managing a Serverless Project in AWS CodeStar.

Use CodePipeline to compile, build, and test code with CodeBuild

CodeBuild is a managed build service in the cloud that lets you build and test your code without a server or system. Use CodePipeline with CodeBuild to automate running revisions through the pipeline for continuous delivery of software builds whenever there is a change to the source code. For more information, see Use AWS CodePipeline with CodeBuild to test code and run builds.
Use CodePipeline with Amazon ECS for continuous delivery of container-based applications to the cloud

Amazon ECS is a container management service that lets you deploy container-based applications to Amazon ECS instances in the cloud. Use CodePipeline with Amazon ECS to automate running revisions through the pipeline for continuous deployment of container-based applications whenever there is a change to the source image repository. For more information, see Tutorial: Continuous Deployment with CodePipeline.

Use CodePipeline with Elastic Beanstalk for continuous delivery of web applications to the cloud

Elastic Beanstalk is a compute service that lets you deploy web applications and services to web servers. Use CodePipeline with Elastic Beanstalk for continuous deployment of web applications to your application environment. You can also use AWS CodeStar to create a pipeline with an Elastic Beanstalk deploy action.

Use CodePipeline with AWS Lambda for continuous delivery of Lambda-based and serverless applications

You can use AWS Lambda with CodePipeline for invoking an AWS Lambda function, as described in Deploying Serverless Applications. You can also use AWS Lambda and AWS CodeStar to create a pipeline for deploying serverless applications.

Use CodePipeline with AWS CloudFormation templates for continuous delivery to the cloud

You can use AWS CloudFormation with CodePipeline for continuous delivery and automation. For more information, see Continuous Delivery with CodePipeline. AWS CloudFormation is also used to create the templates for pipelines created in AWS CodeStar.
Tagging resources

A tag is a custom attribute label that you or AWS assigns to an AWS resource. Each AWS tag has two parts:

- A tag key (for example, CostCenter, Environment, Project, or Secret). Tag keys are case sensitive.
- An optional field known as a tag value (for example, 111122223333, Production, or a team name). Omitting the tag value is the same as using an empty string. Like tag keys, tag values are case sensitive.

Together these are known as key-value pairs.

Tags help you identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related. For example, you can assign the same tag to a pipeline that you assign to an Amazon S3 source bucket.

For tips on using tags, see the AWS Tagging Strategies post on the AWS Answers blog.

You can tag the following resource types in CodePipeline:

- Tag a pipeline in CodePipeline (p. 313)
- Tag a custom action in CodePipeline (p. 334)
- Tag a webhook in CodePipeline (p. 204)

You can use the AWS CLI, CodePipeline APIs, or AWS SDKs to:

- Add tags to a pipeline, custom action, or webhook when you create it.
- Add, manage, and remove tags for a pipeline, custom action, or webhook.

You can also use the console to add, manage, and remove tags for a pipeline.

In addition to identifying, organizing, and tracking your resource with tags, you can use tags in IAM policies to help control who can view and interact with your resource. For examples of tag-based access policies, see Using tags to control access to CodePipeline resources (p. 424).
Use CodePipeline with Amazon Virtual Private Cloud

AWS CodePipeline now supports Amazon Virtual Private Cloud (Amazon VPC) endpoints powered by AWS PrivateLink. This means you can connect directly to CodePipeline through a private endpoint in your VPC, keeping all traffic inside your VPC and the AWS network.

Amazon VPC is an AWS service that you can use to launch AWS resources in a virtual network that you define. With a VPC, you have control over your network settings, such as:

- IP address range
- Subnets
- Route tables
- Network gateways

Interface VPC endpoints are powered by AWS PrivateLink, an AWS technology that facilitates private communication between AWS services using an elastic network interface with private IP addresses. To connect your VPC to CodePipeline, you define an interface VPC endpoint for CodePipeline. This type of endpoint makes it possible for you to connect your VPC to AWS services. The endpoint provides reliable, scalable connectivity to CodePipeline without requiring an internet gateway, network address translation (NAT) instance, or VPN connection. For information about setting up a VPC, see the VPC User Guide.

Availability

CodePipeline currently supports VPC endpoints in the following AWS Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- Canada (Central)
- Europe (Stockholm)
- Europe (Ireland)
- Europe (London)
- Europe (Paris)
- Europe (Frankfurt)
- Asia Pacific (Mumbai)
- Asia Pacific (Tokyo)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- South America (São Paulo)
- AWS GovCloud (US-West)
Create a VPC endpoint for CodePipeline

You can use the Amazon VPC console to create the `com.amazonaws.region.codepipeline` VPC endpoint. In the console, `region` is the Region identifier for an AWS Region supported by CodePipeline, such as `us-east-2` for the US East (Ohio) Region. For more information, see Creating an Interface Endpoint in the Amazon VPC User Guide.

The endpoint is prepopulated with the Region you specified when you signed in to AWS. If you sign in to another Region, the VPC endpoint is updated with the new Region.

**Note**

Other AWS services that provide VPC support and integrate with CodePipeline, such as CodeCommit, might not support using Amazon VPC endpoints for that integration. For example, traffic between CodePipeline and CodeCommit cannot be restricted to the VPC subnet range.

Troubleshooting your VPC setup

When troubleshooting VPC issues, use the information that appears in internet connectivity error messages to help you identify, diagnose, and address issues.

1. Make sure that your internet gateway is attached to your VPC.
2. Make sure that the route table for your public subnet points to the internet gateway.
3. Make sure that your network ACLs allow traffic to flow.
4. Make sure that your security groups allow traffic to flow.
5. Troubleshoot your NAT gateway.
6. Make sure that the route table for private subnets points to the NAT gateway.
7. Make sure that the service role used by CodePipeline has the appropriate permissions. For example, if CodePipeline does not have the Amazon EC2 permissions required to work with an Amazon VPC, you might receive an error that says, "Unexpected EC2 error: UnauthorizedOperation."
Working with pipelines in CodePipeline

To define an automated release process in AWS CodePipeline, you create a pipeline, which is a workflow construct that describes how software changes go through a release process. A pipeline is composed of stages and actions that you configure.

**Note**
When you add Build, Deploy, Test, or Invoke stages, in addition to the default options provided with CodePipeline, you can choose custom actions that you have already created for use with your pipelines. Custom actions can be used for tasks such as running an internally developed build process or a test suite. Version identifiers are included to help you distinguish among different versions of a custom action in the provider lists. For more information, see Create and add a custom action in CodePipeline (p. 324).

Before you can create a pipeline, you must first complete the steps in Getting started with CodePipeline (p. 19).

For more information about pipelines, see CodePipeline concepts (p. 3), CodePipeline tutorials (p. 37), and, if you want to use the AWS CLI to create a pipeline, CodePipeline pipeline structure reference (p. 454). To view a list of pipelines, see View pipeline details and history in CodePipeline (p. 237).

**Topics**
- Start a pipeline execution in CodePipeline (p. 171)
- Stop a pipeline execution in CodePipeline (p. 218)
- Create a pipeline in CodePipeline (p. 221)
- Edit a pipeline in CodePipeline (p. 231)
- View pipeline details and history in CodePipeline (p. 237)
- Delete a pipeline in CodePipeline (p. 250)
- Create a pipeline in CodePipeline that uses resources from another AWS account (p. 251)
- Edit pipelines to use push events (p. 261)
- Create the CodePipeline service role (p. 312)
- Tag a pipeline in CodePipeline (p. 313)
- Create a connection to Bitbucket (p. 317)
- Create a notification rule (p. 321)

Start a pipeline execution in CodePipeline

When a pipeline execution starts, it runs a revision through every stage and action in the pipeline.

There are two ways to start a pipeline execution in AWS CodePipeline:

- **Automatically**: Using change detection methods that you specify, you can make your pipeline start when a change is made to a repository. You can also make your pipeline start on a schedule. The following are the automatic change detection methods:
  - When you use the console to create a pipeline that has a CodeCommit source repository or S3 source bucket, CodePipeline creates an Amazon CloudWatch Events rule that starts your pipeline when the source changes. This is the recommended change detection method. If you use the AWS CLI to create the pipeline, the change detection method defaults to starting the pipeline by periodically...
checking the source (CodeCommit, Amazon S3, and GitHub source providers only). We recommend that you disable periodic checks and create the rule manually. For more information, see Use CloudWatch Events to start a pipeline (CodeCommit source) (p. 174).

• When you use the console to create a pipeline that has a GitHub repository, CodePipeline creates a webhook that starts your pipeline when the source changes. This is the recommended change detection method. If you use the AWS CLI to create the pipeline, the change detection method defaults to starting the pipeline by periodically checking the source. We recommend that you disable periodic checks and create the webhook. For more information, see Use webhooks to start a pipeline (GitHub source) (p. 199).

• Most source actions in CodePipeline, such as GitHub, require either a configured change detection resource (such as a webhook or CloudWatch Events rule) or use the option to poll the repository for source changes. For pipelines with a Bitbucket Cloud source action, you do not have to set up a webhook or default to polling. The connections action manages your source change detection for you.

• **Manually:** You can use the console or the AWS CLI to start a pipeline manually. For information, see Start a pipeline manually in AWS CodePipeline (p. 215).

By default, pipelines are configured to start automatically using change detection methods.

**Note**
Your pipeline runs only when something changes in the source repository and branch that you have defined.

**Topics**
- Change detection methods to start pipelines (p. 172)
- Use CloudWatch Events to start a pipeline (CodeCommit source) (p. 174)
- Use CloudWatch Events to start a pipeline (Amazon S3 source) (p. 185)
- Use webhooks to start a pipeline (GitHub source) (p. 199)
- Use CloudWatch Events to start a pipeline (Amazon ECR source) (p. 209)
- Use periodic checks to start a pipeline (p. 215)
- Start a pipeline manually in AWS CodePipeline (p. 215)
- Use Amazon CloudWatch Events to start a pipeline on a schedule (p. 216)

## Change detection methods to start pipelines

When you create or update a pipeline, you specify the method to use to react to source repository changes. Your choice determines how your pipeline is started.

<table>
<thead>
<tr>
<th>Source</th>
<th>Detection method</th>
<th>Description</th>
<th>Requires additional resources?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon S3</td>
<td>Amazon CloudWatch Events (recommended) and an AWS CloudTrail trail. This is the</td>
<td>• Your pipeline is triggered as soon as a change is made to the repository.</td>
<td>Yes. See Create a pipeline in</td>
</tr>
<tr>
<td></td>
<td>default for pipelines with an Amazon S3 source created or edited in the console.</td>
<td>Events in your bucket are filtered by AWS CloudTrail, and then the Amazon</td>
<td>CodePipeline (p. 221), Use</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CloudWatch Events rule triggers your pipeline to start.</td>
<td>CloudWatch Events to start a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>pipeline (Amazon S3 source) (p. 185)</td>
</tr>
</tbody>
</table>
## Change detection methods to start pipelines

<table>
<thead>
<tr>
<th>Source</th>
<th>Detection method</th>
<th>Description</th>
<th>Requires additional resources?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• Faster and more configurable than periodic checks.</td>
<td>and Events placeholder bucket reference (p. 392).</td>
</tr>
<tr>
<td>Periodic checks</td>
<td>CodePipeline periodically contacts the source.</td>
<td></td>
<td>No.</td>
</tr>
<tr>
<td>AWS CodeCommit</td>
<td>Amazon CloudWatch Events (recommended). This is the default for pipelines with an CodeCommit source created or edited in the console.</td>
<td>• Your pipeline is triggered as soon as a change is made to the repository. • Faster and more configurable than periodic checks.</td>
<td>Yes. See Create a pipeline in CodePipeline (p. 221) and Use CloudWatch Events to start a pipeline (CodeCommit source) (p. 174).</td>
</tr>
<tr>
<td>Periodic checks</td>
<td>CodePipeline periodically contacts the source repository.</td>
<td></td>
<td>No.</td>
</tr>
<tr>
<td>GitHub</td>
<td>Webhooks (recommended). This is the default for pipelines with a GitHub source created or edited in the console.</td>
<td>• Your pipeline is triggered as soon as a change is made to the repository. • Faster and more configurable than periodic checks.</td>
<td>Yes. See Create a pipeline in CodePipeline (p. 221) and Use webhooks to start a pipeline (GitHub source) (p. 199).</td>
</tr>
<tr>
<td>Periodic checks</td>
<td>CodePipeline periodically contacts the source repository.</td>
<td></td>
<td>No.</td>
</tr>
<tr>
<td>Amazon ECR</td>
<td>Amazon CloudWatch Events. This is created by the wizard for pipelines with an Amazon ECR source created or edited in the console.</td>
<td>• Your pipeline is triggered as soon as a change is made to the repository. • Periodic checks are not applicable for this source provider.</td>
<td>Yes. See Create a pipeline in CodePipeline (p. 221) and Use CloudWatch Events to start a pipeline (Amazon ECR source) (p. 209).</td>
</tr>
</tbody>
</table>
If you have a pipeline that uses polling, you can update it to use the recommended detection method. For more information, see Update polling pipelines to the recommended change detection method (p. 520).

**Use CloudWatch Events to start a pipeline (CodeCommit source)**

You can use Amazon CloudWatch Events to trigger pipelines to start when rule or schedule criteria are met. For pipelines with an Amazon S3 or CodeCommit source, an Amazon CloudWatch Events rule detects source changes and then starts your pipeline. When you use the console to create or change a pipeline, the rule and all associated resources are created for you. If you create or change an Amazon S3 or CodeCommit pipeline in the AWS CLI or AWS CloudFormation, you must use these steps to create the Amazon CloudWatch Events rule and all associated resources manually.

In Amazon CloudWatch Events, you create a rule to detect and react to changes in the state of the pipeline's defined source.

**To create the rule**

1. Create an Amazon CloudWatch Events rule that uses the pipeline's source repository as the event source.
2. Add CodePipeline as the target.
3. Grant permissions to Amazon CloudWatch Events to start the pipeline.

As you build your rule, the **Event Pattern Preview** pane in the console (or the --event-pattern output in the AWS CLI) displays the event fields, in JSON format. The following sample CodeCommit event pattern uses this JSON structure:

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

The event pattern uses these fields:

- **source**: should contain `aws.codecommit` as the event source.
- **detail-type**: displays the available event type (`CodeCommit Repository State Change`).
- **resources**: contains the repository ARN.
- **detail**: contains the repository branch information, **referenceType** and **referenceName**.

**Topics**

- Create a CloudWatch Events rule for a CodeCommit source (console) (p. 175)
- Create a CloudWatch Events rule for a CodeCommit source (CLI) (p. 177)
- Create a CloudWatch Events rule for a CodeCommit source (AWS CloudFormation template) (p. 180)
Create a CloudWatch Events rule for a CodeCommit source (console)

**Important**
If you use the console to create or edit your pipeline, your CloudWatch Events rule is created for you.

**To create a CloudWatch Events rule for use in CodePipeline operations**

2. In the navigation pane, choose **Events**.
3. Choose **Create rule**, and then under **Event source**, from **Service Name**, choose **CodeCommit**.

   The service name that you choose owns the event resource. For example, choose CodeCommit to trigger a pipeline when there are changes to the CodeCommit repository associated with a pipeline.

4. From **Event Type**, choose **CodeCommit Repository State Change**.
5. To make a rule that applies to all repositories, choose Any resource.

To make a rule that applies to one or more repositories, choose Specific resource(s) by ARN, and then enter the ARN.

**Note**
You can find the ARN for a CodeCommit repository on the Settings page in the CodeCommit console.

To specify the branch to associate with the repository, choose Edit, and enter the resource type branch and branch name. Use the event pattern options for detail. The preceding example shows the detail options for a CodeCommit repository branch named master.
The following is a sample CodeCommit event pattern in the Event window for a MyTestRepo repository with a branch named master:

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

Choose Save.

In the Event Pattern Preview pane, view the rule.


7. Enter the pipeline ARN for the pipeline to be started when triggered by this rule.

   **Note**
   You can find the pipeline ARN in the metadata output after you run the `get-pipeline` command. The pipeline ARN is constructed in this format:

   `arn:aws:codepipeline:region:account:pipeline-name`

   Sample pipeline ARN:

   `arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline`

8. Create or specify an IAM service role that grants Amazon CloudWatch Events permissions to invoke the target associated with your Amazon CloudWatch Events rule (in this case, the target is CodePipeline).

   - Choose **Create a new role for this specific resource** to create a service role that gives Amazon CloudWatch Events permissions to your start your pipeline executions when triggered.
   - Choose **Use existing role** to enter a service role that gives Amazon CloudWatch Events permissions to your start your pipeline executions when triggered.

9. Review your rule setup to make sure it meets your requirements.

10. Choose Configure details.

11. On the **Configure rule details** page, enter a name and description for the rule, and then choose **State** to enable the rule.

12. If you're satisfied with the rule, choose **Create rule**.

**Create a CloudWatch Events rule for a CodeCommit source (CLI)**

Call the `put-rule` command, specifying:

- A name that uniquely identifies the rule you are creating. This name must be unique across all of the pipelines you create with CodePipeline associated with your AWS account.
Use a CloudWatch Events rule to start a pipeline (CodeCommit source)

- The event pattern for the source and detail fields used by the rule. For more information, see Amazon CloudWatch Events and Event Patterns.

To create a CloudWatch Events rule with CodeCommit as the event source and CodePipeline as the target

1. Add permissions for Amazon CloudWatch Events to use CodePipeline to invoke the rule. For more information, see Using Resource-Based Policies for Amazon CloudWatch Events.
   
a. Use the following sample to create the trust policy that allows CloudWatch Events to assume the service role. Name the trust policy trustpolicyforCWE.json.

   ```json
   {
   "Version": "2012-10-17",
   "Statement": [ 
       { 
           "Effect": "Allow",
           "Principal": {
               "Service": "events.amazonaws.com"
           },
           "Action": "sts:AssumeRole"
       }
   ]
   }
   ``

   b. Use the following command to create the Role-for-MyRule role and attach the trust policy.

   ```bash
   aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document file://trustpolicyforCWE.json
   ``

   c. Create the permissions policy JSON, as shown in this sample, for the pipeline named MyFirstPipeline. Name the permissions policy permissionspolicyforCWE.json.

   ```json
   {
   "Version": "2012-10-17",
   "Statement": [
       { 
           "Effect": "Allow",
           "Action": [ 
               "codepipeline:StartPipelineExecution"
           ],
           "Resource": [
               "arn:aws:codepipeline:us-west-2:80398EXAMPLE:MyFirstPipeline"
           ]
       }
   ]
   }
   ``

   d. Use the following command to attach the CodePipeline-Permissions-Policy-for-CWE permissions policy to the Role-for-MyRule role.

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

   Why am I making this change? Adding this policy to the role creates permissions for CloudWatch Events.

2. Call the put-rule command and include the --name, --event-pattern, and --role-arn parameters.
Why am I making this change? This command enables AWS CloudFormation to create the event.

The following sample command creates a rule called MyCodeCommitRepoRule.

```bash
aws events put-rule --name "MyCodeCommitRepoRule" --event-pattern "{\"source\": [\"aws.codecommit\"], \"detail-type\": [\"CodeCommit Repository State Change\"], \"resources\": [\"repository-ARN\"], \"detail\": {\"referenceType\": [\"branch\"], \"referenceName\": [\"master\"]} \" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-MyRule"
```

3. To add CodePipeline as a target, call the `put-targets` command and include the following parameters:

   - The `--rule` parameter is used with the `rule_name` you created by using `put-rule`.
   - The `--targets` parameter is used with the list ID of the target in the list of targets and the ARN of the target pipeline.

   The following sample command specifies that for the rule called MyCodeCommitRepoRule, the target ID is composed of the number one, indicating that in a list of targets for the rule, this is target 1. The sample command also specifies an example ARN for the pipeline. The pipeline starts when something changes in the repository.

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

To edit your pipeline’s PollForSourceChanges parameter

**Important**

When you create a pipeline with this method, the PollForSourceChanges parameter defaults to true if it is not explicitly set to false. When you add event-based change detection, you must add the parameter to your output and set it to false to disable polling. Otherwise, your pipeline starts twice for a single source change. For details, see Default settings for the PollForSourceChanges parameter (p. 466).

1. Run the `get-pipeline` command to copy the pipeline structure into a JSON file. For example, for a pipeline named MyFirstPipeline, run the following command:

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

   This command returns nothing, but the file you created should appear in the directory where you ran the command.

2. Open the JSON file in any plain-text editor and edit the source stage by changing the PollForSourceChanges parameter to false, as shown in this example.

   Why am I making this change? Changing this parameter to false turns off periodic checks so you can use event-based change detection only.

```json
"configuration": {
   "PollForSourceChanges": "false",
   "BranchName": "master",
   "RepositoryName": "MyTestRepo"
},
```
3. If you are working with the pipeline structure retrieved using the `get-pipeline` command, remove the metadata lines from the JSON file. Otherwise, the `update-pipeline` command cannot use it. Remove the "metadata": { } lines and the "created", "pipelineARN", and "updated" fields.

   For example, remove the following lines from the structure:

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

   Save the file.

4. To apply your changes, run the `update-pipeline` command, specifying the pipeline JSON file:

   ```bash
   Important
   Be sure to include file:// before the file name. It is required in this command.
   ```

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

   This command returns the entire structure of the edited pipeline.

   ```text
   Note
   The `update-pipeline` command stops the pipeline. If a revision is being run through the pipeline when you run the `update-pipeline` command, that run is stopped. You must manually start the pipeline to run that revision through the updated pipeline. Use the `start-pipeline-execution` command to manually start your pipeline.
   ```

Create a CloudWatch Events rule for a CodeCommit source (AWS CloudFormation template)

To use AWS CloudFormation to create a rule, update your template as shown here.

**To update your pipeline AWS CloudFormation template and create CloudWatch Events rule**

1. In the template, under Resources, use the `AWS::IAM::Role` AWS CloudFormation resource to configure the IAM role that allows your event to start your pipeline. This entry creates a role that uses two policies:
   - The first policy allows the role to be assumed.
   - The second policy provides permissions to start the pipeline.

   **Why am I making this change?** Adding the `AWS::IAM::Role` resource enables AWS CloudFormation to create permissions for CloudWatch Events. This resource is added to your AWS CloudFormation stack.

   YAML

   ```yaml
   AmazonCloudWatchEventRole:
   Type: AWS::IAM::Role
   Properties:
     AssumeRolePolicyDocument:
       Version: 2012-10-17
       Statement:
         Effect: Allow
   ```
Use a CloudWatch Events rule to start a pipeline (CodeCommit source)

Principal:
Service:  
- events.amazonaws.com

Action: sts:AssumeRole
Path: /
Policies:
- PolicyName: cwe-pipeline-execution
  PolicyDocument:
    Version: 2012-10-17
    Statement:
    - Effect: Allow
      Action: codepipeline:StartPipelineExecution

JSON

"AmazonCloudWatchEventRole": {  
  "Type": "AWS::IAM::Role",  
  "Properties": {  
    "AssumeRolePolicyDocument": {  
      "Version": "2012-10-17",  
      "Statement": [  
        {  
          "Effect": "Allow",  
          "Principal": {  
            "Service": [  
              "events.amazonaws.com"  
            ]  
          },  
          "Action": "sts:AssumeRole"  
        }  
      ]  
    },  
    "Path": "/",  
    "Policies": [  
      {  
        "PolicyName": "cwe-pipeline-execution",  
        "PolicyDocument": {  
          "Version": "2012-10-17",  
          "Statement": [  
            {  
              "Effect": "Allow",  
              "Action": "codepipeline:StartPipelineExecution",  
              "Resource": {  
                "Fn::Join": [  
                  "",  
                  [  
                    "arn:aws:codepipeline:",  
                    {  
                      "Ref": "AWS::Region"  
                    },  
                    "::",  
                    {  
                      "Ref": "AWS::AccountId"  
                    },  
                    "::",  
                    {  
                      "Ref": "AppPipeline"  
                    }  
                  ]  
                ]  
              }  
            }  
          ]  
        }  
      }  
    ]  
  }  
}
Use a CloudWatch Events rule to start a pipeline (CodeCommit source)

2. In the template, under Resources, use the AWS::Events::Rule AWS CloudFormation resource to add a CloudWatch Events rule. This event pattern creates an event that monitors push changes to your repository. When CloudWatch Events detects a repository state change, the rule invokes StartPipelineExecution on your target pipeline.

**Why am I making this change?** Adding the AWS::Events::Rule resource enables AWS CloudFormation to create the event. This resource is added to your AWS CloudFormation stack.

**YAML**

```yaml
AmazonCloudWatchEventRule:
  Type: AWS::Events::Rule
  Properties:
    EventPattern:
      source:
        - aws.codecommit
      detail-type:
        - 'CodeCommit Repository State Change'
      resources:
        - !Join [' ', [
          'arn:aws:codecommit:', !Ref 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref RepositoryName ]]
      detail:
        event:
          - referenceCreated
          - referenceUpdated
        referenceType:
          - branch
        referenceName:
          - master
      Targets:
        - Arn:
          !Join [' ', [
            'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ]]
          RoleArn: !GetAtt AmazonCloudWatchEventRole.Arn
          Id: codepipeline-AppPipeline
```

**JSON**

```json
"AmazonCloudWatchEventRule": {
  "Type": "AWS::Events::Rule",
  "Properties": {
    "EventPattern": {
      "source": ["aws.codecommit"],
      "detail-type": ["CodeCommit Repository State Change"],
      "resources": [
        {"Fn::Join": ["", [
          "arn:aws:codecommit:",
          {
            "Ref": "AWS::Region"
          },
          ":",
        ]}
      }
    }
  }
}
```

API Version 2015-07-09

182
Use a CloudWatch Events rule to start a pipeline (CodeCommit source)

```
"Ref": "AWS::AccountId"
},
"":",
{
  "Ref": "RepositoryName"
}
]
],
"detail": {
  "event": [
    "referenceCreated",
    "referenceUpdated"
  ],
  "referenceType": [
    "branch"
  ],
  "referenceName": [
    "master"
  ]
},
"Targets": [
{
  "Arn": {
    "Fn::Join": [
      ",
      [narn:aws:codepipeline:",
      {
        "Ref": "AWS::Region"
      },
      ",
      {
        "Ref": "AWS::AccountId"
      },
      ",
      {
        "Ref": "AppPipeline"
      }
    ]
  },
  "RoleArn": {
    "Fn::GetAtt": [
      "AmazonCloudWatchEventRole",
      "Arn"
    ],
    "Id": "codepipeline-AppPipeline"
  }
}]
},

3. Save the updated template to your local computer, and then open the AWS CloudFormation console.
4. Choose your stack, and then choose **Create Change Set for Current Stack**.
5. Upload the template, and then view the changes listed in AWS CloudFormation. These are the changes to be made to the stack. You should see your new resources in the list.
6. Choose **Execute**.
Use a CloudWatch Events rule to start a pipeline (CodeCommit source)

To edit your pipeline's PollForSourceChanges parameter

Important
In many cases, the PollForSourceChanges parameter defaults to true when you create a pipeline. When you add event-based change detection, you must add the parameter to your output and set it to false to disable polling. Otherwise, your pipeline starts twice for a single source change. For details, see Default settings for the PollForSourceChanges parameter (p. 466).

- In the template, change PollForSourceChanges to false. If you did not include PollForSourceChanges in your pipeline definition, add it and set it to false.

Why am I making this change? Changing this parameter to false turns off periodic checks so you can use event-based change detection only.

YAML

```yaml
Name: Source
Actions:
  - Name: SourceAction
    ActionTypeId:
      Category: Source
      Owner: AWS
      Version: 1
      Provider: CodeCommit
    OutputArtifacts:
      - Name: SourceOutput
    Configuration:
      BranchName: !Ref BranchName
      RepositoryName: !Ref RepositoryName
    PollForSourceChanges: false
    RunOrder: 1
```

JSON

```json
{
  "Name": "Source",
  "Actions": [
    {
      "Name": "SourceAction",
      "ActionTypeId": {
        "Category": "Source",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "CodeCommit"
      },
      "OutputArtifacts": [
        {
          "Name": "SourceOutput"
        }
      ],
      "Configuration": {
        "BranchName": {
          "Ref": "BranchName"
        },
        "RepositoryName": {
          "Ref": "RepositoryName"
        },
        "PollForSourceChanges": false
      },
      "RunOrder": 1
    }
  }
}
```
Use CloudWatch Events to start a pipeline (Amazon S3 source)

You should use Amazon CloudWatch Events to detect source code changes and trigger the start of your pipeline. If your pipeline has an Amazon S3 source, you must create an AWS CloudTrail trail to log write events to objects in your Amazon S3 source bucket.

AWS CloudTrail is a service that logs and filters events on your Amazon S3 source bucket. The trail sends the filtered source changes to the Amazon CloudWatch Events rule. The Amazon CloudWatch Events rule detects the source change and then starts your pipeline.

**Note**

For pipelines with an Amazon S3 source, an Amazon CloudWatch Events rule detects source changes and then starts your pipeline when changes occur. When you use the console to create or change a pipeline, the rule and all associated resources are created for you. If you create or change a pipeline with an Amazon S3 source in the CLI or AWS CloudFormation, you must create the Amazon CloudWatch Events rule, IAM role, and AWS CloudTrail trail manually.

**Requirements:**

- If you are not creating a trail, use an existing AWS CloudTrail trail for logging events in your Amazon S3 source bucket and sending filtered events to the Amazon CloudWatch Events rule.
- Create or use an existing S3 bucket where AWS CloudTrail can store its log files. AWS CloudTrail must have the permissions required to deliver log files to an Amazon S3 bucket. The bucket cannot be configured as a Requester Pays bucket. When you create an Amazon S3 bucket as part of creating or updating a trail in the console, AWS CloudTrail attaches the required permissions to a bucket for you. For more information, see Amazon S3 Bucket Policy for CloudTrail.

Create a CloudWatch Events rule for an Amazon S3 source (console)

Before you set up a rule in CloudWatch Events, you must create an AWS CloudTrail trail. For more information, see Creating a Trail in the Console.

**Important**

If you use the console to create or edit your pipeline, your CloudWatch Events rule and AWS CloudTrail trail are created for you.

**To create a trail**

1. Open the AWS CloudTrail console.
2. In the navigation pane, choose **Trails**.
3. Choose **Create Trail**. For **Trail name**, enter a name for your trail.
4. For **Apply trail to all regions**, choose **No**.
5. Under **Data events**, make sure **S3** is selected. Specify an Amazon S3 bucket and the object prefix (folder name) to log data events for all objects in the folder. For each trail, you can add up to 250 Amazon S3 objects.
6. For **Read/Write events**, choose **None**.
7. Choose **Write**. The trail records Amazon S3 object-level API activity (for example, **GetObject** and **PutObject**) on the specified bucket and prefix.
8. Under **Storage location**, create or specify the bucket to be used to store the log files. By default, Amazon S3 buckets and objects are private. Only the resource owner (the AWS account that created the bucket) can access the bucket and its objects. The bucket must have a resource policy that allows AWS CloudTrail permissions to access the objects in the bucket.

9. If you're satisfied with the trail, choose **Create**.

**To create a CloudWatch Events rule that targets your pipeline with an S3 source**

2. In the navigation pane, choose **Events**.
3. Choose **Event Pattern**, and then choose **Build event pattern to match events by service**.
4. Under **Event source**, from **Service Name**, choose **Simple Storage Service (S3)**.
5. From **Event Type**, choose **Object Level Operations**.
6. Choose **Specific operation(s)**, and then choose **CompleteMultipartUpload**, **CopyObject**, and **PutObject**.

Above the **Event Pattern Preview** pane, choose **Edit**. Edit the event pattern to add the bucket name and encryption key as **requestParameters**, as shown in this example for a bucket named `my-bucket`. When you use the **Edit** window to specify resources, your rule is updated to use a custom event pattern.
The following is a sample event pattern to copy and paste:

```json
{
  "source": [
    "aws.s3"
  ],
  "detail-type": [
    "AWS API Call via CloudTrail"
  ],
  "detail": {
    "eventSource": [
      "s3.amazonaws.com"
    ],
    "eventName": [
      "CopyObject",
      "CompleteMultipartUpload",
      "PutObject"
    ],
    "requestParameters": {
      "bucketName": [
        "my-bucket"
      ],
      "key": [
        "my-key"
      ]
    }
  }
}
```

7. In **Targets**, choose **CodePipeline**.
8. Enter the pipeline ARN for the pipeline to be started when triggered by this rule.

**Note**
To get the pipeline ARN, run the `get-pipeline` command. The pipeline ARN appears in the output. It is constructed in this format:

```
arn:aws:codepipeline::<region>:<account>:<pipeline-name>
```

Sample pipeline ARN:

```
arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline
```

9. To create or specify an IAM service role that grants Amazon CloudWatch Events permissions to invoke the target associated with your Amazon CloudWatch Events rule (in this case, the target is CodePipeline):

- Choose **Create a new role for this specific resource** to create a service role that gives Amazon CloudWatch Events permissions to your start your pipeline executions when triggered.
- Choose **Use existing role** to enter a service role that gives Amazon CloudWatch Events permissions to your start your pipeline executions when triggered.

10. Review your rule to make sure it meets your requirements, and then choose **Configure details**.
11. On the **Configure rule details** page, enter a name and description for the rule, and then choose **State** to enable the rule.
12. If you're satisfied with the rule, choose **Create rule**.

**Create a CloudWatch Events rule for an Amazon S3 source (CLI)**

**To create an AWS CloudTrail trail and enable logging**

To use the AWS CLI to create a trail, call the `create-trail` command, specifying:

- The trail name.
• The bucket to which you have already applied the bucket policy for AWS CloudTrail.

For more information, see Creating a Trail with the AWS Command Line Interface.

1. Call the `create-trail` command and include the `--name` and `--s3-bucket-name` parameters.

   **Why am I making this change?** This creates the CloudTrail trail required for your S3 source bucket.

   The following command uses `--name` and `--s3-bucket-name` to create a trail named `my-trail` and a bucket named `myBucket`.

   ```bash
   aws cloudtrail create-trail --name my-trail --s3-bucket-name myBucket
   ```

2. Call the `start-logging` command and include the `--name` parameter.

   **Why am I making this change?** This command starts the CloudTrail logging for your source bucket and sends events to CloudWatch Events.

   Example:

   The following command uses `--name` to start logging on a trail named `my-trail`.

   ```bash
   aws cloudtrail start-logging --name my-trail
   ```

3. Call the `put-event-selectors` command and include the `--trail-name` and `--event-selectors` parameters. Use event selectors to specify that you want your trail to log data events for your source bucket and send the events to the Amazon CloudWatch Events rule.

   **Why am I making this change?** This command filters events.

   Example:

   The following command uses `--trail-name` and `--event-selectors` to specify data events for a source bucket and prefix named `myBucket/myFolder`.

   ```bash
   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"]
      }']
    }
   ```

To create a CloudWatch Events rule with Amazon S3 as the event source and CodePipeline as the target and apply the permissions policy

1. Grant permissions for Amazon CloudWatch Events to use CodePipeline to invoke the rule. For more information, see Using Resource-Based Policies for Amazon CloudWatch Events.

   a. Use the following sample to create the trust policy to allow CloudWatch Events to assume the service role. Name it `trustpolicyforCWE.json`.

   ```json
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
           "Service": "events.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
       }
     ]
   }
   ```
Use CloudWatch Events to start a pipeline (Amazon S3 source)

b. Use the following command to create the Role-for-MyRule role and attach the trust policy.

```
Why am I making this change? Adding this trust policy to the role creates permissions for CloudWatch Events.
```

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document file://trustpolicyforCWE.json
```

c. Create the permissions policy JSON, as shown here for the pipeline named MyFirstPipeline. Name the permissions policy permissionspolicyforCWE.json.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codepipeline:StartPipelineExecution"
      ],
      "Resource": [
        "arn:aws:codepipeline:us-west-2:80398EXAMPLE:MyFirstPipeline"
      ]
    }
  ]
}
```

d. Use the following command to attach the new CodePipeline-Permissions-Policy-for-CWE permissions policy to the Role-for-MyRule role you created.

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

2. Call the put-rule command and include the --name, --event-pattern, and --role-arn parameters.

The following sample command creates a rule named MyS3SourceRule.

```
aws events put-rule --name "MyS3SourceRule" --event-pattern "{"source": ["aws.s3"], "detail-type": ["AWS API Call via CloudTrail"], "detail": {"eventSource": ["s3.amazonaws.com"], "eventName": ["CopyObject","PutObject","CompleteMultipartUpload"], "requestParameters": {"bucketName": ["my-bucket"], "key": ["my-key"])}}" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-MyRule"
```

3. To add CodePipeline as a target, call the put-targets command and include the --rule and --targets parameters.

The following command specifies that for the rule named MyS3SourceRule, the target Id is composed of the number one, indicating that in a list of targets for the rule, this is target 1. The command also specifies an example ARN for the pipeline. The pipeline starts when something changes in the repository.

```
aws events put-targets --rule MyS3SourceRule --targets Id=1,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
```
To edit your pipeline’s PollForSourceChanges parameter

Important
When you create a pipeline with this method, the PollForSourceChanges parameter defaults to true if it is not explicitly set to false. When you add event-based change detection, you must add the parameter to your output and set it to false to disable polling. Otherwise, your pipeline starts twice for a single source change. For details, see Default settings for the PollForSourceChanges parameter (p. 466).

1. Run the get-pipeline command to copy the pipeline structure into a JSON file. For example, for a pipeline named MyFirstPipeline, run the following command:

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

   This command returns nothing, but the file you created should appear in the directory where you ran the command.

2. Open the JSON file in any plain-text editor and edit the source stage by changing the PollForSourceChanges parameter for a bucket named storage-bucket to false, as shown in this example.

   **Why am I making this change?** Setting this parameter to false turns off periodic checks so you can use event-based change detection only.

   ```json
   "configuration": {
     "S3Bucket": "storage-bucket",
     "PollForSourceChanges": "false",
     "S3ObjectKey": "index.zip"
   },
   ```

3. If you are working with the pipeline structure retrieved using the get-pipeline command, you must remove the metadata lines from the JSON file. Otherwise, the update-pipeline command cannot use it. Remove the "metadata": { } lines and the "created", "pipelineARN", and "updated" fields.

   For example, remove the following lines from the structure:

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

   Save the file.

4. To apply your changes, run the update-pipeline command, specifying the pipeline JSON file:

   ```bash
   Important
   Be sure to include file:// before the file name. It is required in this command.
   ```

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

   This command returns the entire structure of the edited pipeline.

   **Note**
   The update-pipeline command stops the pipeline. If a revision is being run through the pipeline when you run the update-pipeline command, that run is stopped. You must manually start the pipeline to run that revision through the updated pipeline. Use the start-pipeline-execution command to manually start your pipeline.
Create a CloudWatch Events rule for an Amazon S3 source (AWS CloudFormation template)

To use AWS CloudFormation to create a rule, update your template as shown here.

**To create a CloudWatch Events rule with Amazon S3 as the event source and CodePipeline as the target and apply the permissions policy**

1. In the template, under `Resources`, use the `AWS::IAM::Role` AWS CloudFormation resource to configure the IAM role that allows your event to start your pipeline. This entry creates a role that uses two policies:
   - The first policy allows the role to be assumed.
   - The second policy provides permissions to start the pipeline.

**Why am I making this change?** Adding `AWS::IAM::Role` resource enables AWS CloudFormation to create permissions for Amazon CloudWatch Events. This resource is added to your AWS CloudFormation stack.

**YAML**

```yaml
AmazonCloudWatchEventRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: 2012-10-17
      Statement:
        -
          Effect: Allow
          Principal:
            Service:
              - events.amazonaws.com
          Action: sts:AssumeRole
          Path: /
          Policies:
            -
              PolicyName: cwe-pipeline-execution
              PolicyDocument:
                Version: 2012-10-17
                Statement:
                  -
                    Effect: Allow
                    Action: codepipeline:StartPipelineExecution
```

**JSON**

```json
"AmazonCloudWatchEventRole": {
  "Type": "AWS::IAM::Role",
  "Properties": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": "codepipeline:StartPipelineExecution",
        }
      ]
    }
  }
}
```
Use CloudWatch Events to start a pipeline (Amazon S3 source)

2. Use the AWS::Events::Rule AWS CloudFormation resource to add a CloudWatch Events rule. This event pattern creates an event that monitors CopyObject, PutObject and CompleteMultipartUpload on your Amazon S3 source bucket. In addition, include a target of your pipeline. When CopyObject, PutObject, or CompleteMultipartUpload occurs, this rule invokes StartPipelineExecution on your target pipeline.

Why am I making this change? Adding the AWS::Events::Rule resource enables AWS CloudFormation to create the event. This resource is added to your AWS CloudFormation stack.

YAML

```yaml
AWSCloudWatchEventRule:
  Type: AWS::Events::Rule
  Properties:
    EventPattern:
      source:
        - aws.s3
detail-type:
        - 'AWS API Call via CloudTrail'
detail:
  eventSource:
    - s3.amazonaws.com
  eventName:
    - CopyObject
```
Use CloudWatch Events to start a pipeline (Amazon S3 source)

- PutObject
- CompleteMultipartUpload
requestParameters:
  - !Ref SourceBucket
  key:
    - !Ref SourceObjectKey
Targets:
  - Arn:
    !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
  RoleArn: !GetAtt AmazonCloudWatchEventRole.Arn
  Id: codepipeline-AppPipeline

...
3. Add this snippet to your first template to allow cross-stack functionality:

**YAML**

```yaml
Outputs:
  SourceBucketARN:
    Description: "S3 bucket ARN that Cloudtrail will use"
    Value: !GetAtt SourceBucket.Arn
    Export:
      Name: SourceBucketARN

**JSON**

```json
"Outputs" : {
  "SourceBucketARN" : {
    "Description" : "S3 bucket ARN that Cloudtrail will use",
    "Value" : { "Fn::GetAtt": [ "SourceBucket", "Arn"] },
    "Export" : {
      "Name" : "SourceBucketARN"
    }
  }
}
```

4. Save your updated template to your local computer, and open the AWS CloudFormation console.
5. Choose your stack, and then choose **Create Change Set for Current Stack**.
6. Upload your updated template, and then view the changes listed in AWS CloudFormation. These are the changes that will be made to the stack. You should see your new resources in the list.
7. Choose **Execute**.

**To edit your pipeline's PollForSourceChanges parameter**

**Important**

When you create a pipeline with this method, the PollForSourceChanges parameter defaults to true if it is not explicitly set to false. When you add event-based change detection,
you must add the parameter to your output and set it to false to disable polling. Otherwise, your pipeline starts twice for a single source change. For details, see Default settings for the `PollForSourceChanges` parameter (p. 466).

- In the template, change `PollForSourceChanges` to `false`. If you did not include `PollForSourceChanges` in your pipeline definition, add it and set it to `false`.

**Why am I making this change?** Changing `PollForSourceChanges to false` turns off periodic checks so you can use event-based change detection only.

**YAML**

```yaml
Name: Source
Actions:
  - Name: SourceAction
    ActionType:
      Category: Source
      Owner: AWS
      Version: 1
      Provider: S3
    OutputArtifacts:
      - Name: SourceOutput
    Configuration:
      S3Bucket: !Ref SourceBucket
      S3ObjectKey: !Ref SourceObjectKey
      PollForSourceChanges: false
    RunOrder: 1
```

**JSON**

```json
{
  "Name": "SourceAction",
  "ActionTypeId": {
    "Category": "Source",
    "Owner": "AWS",
    "Version": 1,
    "Provider": "S3"
  },
  "OutputArtifacts": [
    {
      "Name": "SourceOutput"
    }
  ],
  "Configuration": {
    "S3Bucket": {
      "Ref": "SourceBucket"
    },
    "S3ObjectKey": {
      "Ref": "SourceObjectKey"
    },
    "PollForSourceChanges": false
  },
  "RunOrder": 1
}
```
To create a second template for your Amazon S3 pipeline's CloudTrail resources

- In a separate template, under Resources, use the AWS::S3::Bucket, AWS::S3::BucketPolicy, and AWS::CloudTrail::Trail AWS CloudFormation resources to provide a simple bucket definition and trail for CloudTrail.

**Why am I making this change?** Given the current limit of five trails per account, the CloudTrail trail must be created and managed separately. (See Limits in AWS CloudTrail.) However, you can include many Amazon S3 buckets on a single trail, so you can create the trail once and then add Amazon S3 buckets for other pipelines as necessary. Paste the following into your second sample template file.

YAML

```
# Prerequisites:
#   - S3 SourceBucket and SourceObjectKey must exist

Parameters:
  SourceObjectKey:
    Description: 'S3 source artifact'
    Type: String
    Default: SampleApp_Linux.zip

Resources:
  AWSCloudTrailBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref AWSCloudTrailBucket
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          - Sid: AWSCloudTrailAclCheck
            Effect: Allow
            Principal:
              Service:
                - cloudtrail.amazonaws.com
            Action: s3:GetBucketAcl
            Resource: !GetAtt AWSCloudTrailBucket.Arn
          - Sid: AWSCloudTrailWrite
            Effect: Allow
            Principal:
              Service:
                - cloudtrail.amazonaws.com
            Action: s3:PutObject
            Condition:
              StringEquals:
                s3:x-amz-acl: bucket-owner-full-control
  AWSCloudTrailBucket:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
  AwsCloudTrail:
    DependsOn:
      - AWSCloudTrailBucketPolicy
    Type: AWS::CloudTrail::Trail
    Properties:
      S3BucketName: !Ref AWSCloudTrailBucket
      EventSelectors:
        - DataResources:
```

API Version 2015-07-09
Use CloudWatch Events to start a pipeline (Amazon S3 source)

```json
{
    "Parameters": {
        "SourceObjectKey": {
            "Description": "S3 source artifact",
            "Type": "String",
            "Default": "SampleApp_Linux.zip"
        }
    },
    "Resources": {
        "AWSCloudTrailBucket": {
            "Type": "AWS::S3::Bucket",
            "DeletionPolicy": "Retain"
        },
        "AWSCloudTrailBucketPolicy": {
            "Type": "AWS::S3::BucketPolicy",
            "Properties": {
                "Bucket": {
                    "Ref": "AWSCloudTrailBucket"
                },
                "PolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Sid": "AWSCloudTrailAclCheck",
                            "Effect": "Allow",
                            "Principal": {
                                "Service": ["cloudtrail.amazonaws.com"
                            ],
                            "Action": "s3:GetBucketAcl",
                            "Resource": {
                                "Fn::GetAtt": ["AWSCloudTrailBucket", "Arn"
                            ]
                        },
                        {
                            "Sid": "AWSCloudTrailWrite",
                            "Effect": "Allow",
                            "Principal": {
                                "Service": ["cloudtrail.amazonaws.com"
                            ],
                            "Action": "s3:PutObject",
                            "Resource": {
                                "Fn::Join": [
                            
```

---

API Version 2015-07-09

197
Use CloudWatch Events to start a pipeline (Amazon S3 source)

```
"",
[
  {
    "Fn::GetAtt": [
      "AWSCloudTrailBucket",
      "Arn"
    ],
    "/AWSLogs/",
    {
      "Ref": "AWS::AccountId"
    },
    "/*"
  },
  "Condition": {
    "StringEquals": {
      "s3:x-amz-acl": "bucket-owner-full-control"
    }
  }
},
"AwsCloudTrail": {
  "DependsOn": [
    "AWSCloudTrailBucketPolicy"
  ],
  "Type": "AWS::CloudTrail::Trail",
  "Properties": {
    "S3BucketName": {
      "Ref": "AWSCloudTrailBucket"
    },
    "EventSelectors": [
      {
        "DataResources": [
          {
            "Type": "AWS::S3::Object",
            "Values": [
              {
                "Fn::Join": [
                  ",
                  [
                    {
                      "Fn::ImportValue": "SourceBucketARN"
                    },
                    "/*",
                    {
                      "Ref": "SourceObjectKey"
                    }
                  ]
                ]
              }
            ]
          }
        ],
        "ReadWriteType": "WriteOnly"
      }
    ],
    "IncludeGlobalServiceEvents": true,
    "IsLogging": true,
    "IsMultiRegionTrail": true
  }
}
```

API Version 2015-07-09
Use webhooks to start a pipeline (GitHub source)

A webhook is an HTTP notification that detects events in another tool, such as a GitHub repository, and connects those external events to a pipeline.

When you use the console to create or edit a pipeline that has a GitHub source, CodePipeline creates a webhook. CodePipeline deletes your webhook when you delete your pipeline. You do not need to manage it in GitHub. If you use the AWS CLI or AWS CloudFormation to create or edit a pipeline that has a GitHub source, you must use the information in these sections to manage webhooks yourself.

Topics
- Create a webhook for a GitHub source (p. 199)
- List webhooks in your account (p. 202)
- Edit the webhook for your GitHub source (p. 202)
- Delete the webhook for your GitHub source (p. 203)
- Tag a webhook in CodePipeline (p. 204)
- Create a webhook for a GitHub source (AWS CloudFormation template) (p. 205)

Create a webhook for a GitHub source

After you use the AWS CLI to manually create a webhook, you must register the webhook in GitHub. A designated AWS endpoint is used for the webhook and is supplied by the `put-webhook` command.

**Important**
If you use the console to create or edit your pipeline, your webhook is created for you.

To use the AWS CLI to create a webhook, call the `put-webhook` command and supply the following:

- A name that uniquely identifies the webhook. This name must be unique within the region of the account for the pipeline.
- A secret in the JSON file to be used for GitHub authorization.

To create and register your webhook

**Note**
When you use the CLI or AWS CloudFormation to create a pipeline and add a webhook, you must disable periodic checks. To disable periodic checks, you must explicitly add the `PollForSourceChanges` parameter and set it to false, as detailed in the final procedure below. Otherwise, the default for a CLI or AWS CloudFormation pipeline is that `PollForSourceChanges` defaults to true and does not display in the pipeline structure output. For more information about `PollForSourceChanges` defaults, see Default settings for the `PollForSourceChanges` parameter (p. 466).

1. In a text editor, create and save a JSON file for the webhook you want to create. Use this sample file for a webhook named `my-webhook`:

   ```json
   {"webhook":
    {"name": "my-webhook",
    "targetPipeline": "pipeline_name",
   }
   ```

   API Version 2015-07-09

   199
Use webhooks to start a pipeline (GitHub source)

2. Call the **put-webhook** command and include the **--cli-input** and **--region** parameters.

The following sample command creates a webhook with the **webhook_json** JSON file.

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

3. In the output shown in this example, the URL and ARN are returned for a webhook named **my-webhook**.

```json
{
  "webhook": {
    "url": "https://webhooks.domain.com/trigger1111111111EXAMPLE11111111111111111",
    "definition": {
      "authenticationConfiguration": {
        "SecretToken": "secret"
      },
      "name": "my-webhook",
      "authentication": "GITHUB_HMAC",
      "targetPipeline": "pipeline_name",
      "targetAction": "Source",
      "filters": [
        {
          "jsonPath": ".ref",
          "matchEquals": "refs/heads/{Branch}"n
        }
      ]
    },
  },
  "tags": [{
    "key": "Project",
    "value": "ProjectA"
  }]
}
```

This example adds tagging to the webhook by including the **Project** tag key and **ProjectA** value on the webhook. For more information about tagging resources in CodePipeline, see Tagging resources (p. 168).

4. Call the **register-webhook-with-third-party** command and include the **--webhook-name** parameter.

The following sample command registers a webhook named **my-webhook**.

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

If you are updating a pipeline to use webhooks, you must also use the following procedure to turn off periodic checks.
To edit your pipeline’s PollForSourceChanges parameter

Important
When you create a pipeline with this method, the PollForSourceChanges parameter defaults to true if it is not explicitly set to false. When you add event-based change detection, you must add the parameter to your output and set it to false to disable polling. Otherwise, your pipeline starts twice for a single source change. For details, see Default settings for the PollForSourceChanges parameter (p. 466).

1. Run the get-pipeline command to copy the pipeline structure into a JSON file. For example, for a pipeline named MyFirstPipeline, you would type the following command:

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

   This command returns nothing, but the file you created should appear in the directory where you ran the command.

2. Open the JSON file in any plain-text editor and edit the source stage by changing or adding the PollForSourceChanges parameter. In this example, for a repository named UserGitHubRepo, the parameter is set to false.

   Why am I making this change? Changing this parameter turns off periodic checks so you can use event-based change detection only.

   ```
   "configuration": {
   "Owner": "darlaker",
   "Repo": "UserGitHubRepo",
   "PollForSourceChanges": "false",
   "Branch": "master",
   "OAuthToken": "****"
   },
   ```

3. If you are working with the pipeline structure retrieved using the get-pipeline command, you must edit the structure in the JSON file by removing the metadata lines from the file. Otherwise, the update-pipeline command cannot use it. Remove the "metadata" section from the pipeline structure in the JSON file, including the :{} and the "created", "pipelineARN", and "updated" fields.

   For example, remove the following lines from the structure:

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

   Save the file.

4. To apply your changes, run the update-pipeline command, specifying the pipeline JSON file, similar to the following:

   ```
   Important
   Be sure to include file:// before the file name. It is required in this command.
   ```

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

   This command returns the entire structure of the edited pipeline.
Note

The update-pipeline command stops the pipeline. If a revision is being run through the pipeline when you run the update-pipeline command, that run is stopped. You must manually start the pipeline to run that revision through the updated pipeline. Use the start-pipeline-execution command to manually start your pipeline.

List webhooks in your account

You can use the AWS CLI to list webhooks in your account.

1. To list your webhooks, call the list-webhooks command and include the --endpoint-url and --region parameters.

   The following sample command lists webhooks for the "eu-central-1" endpoint URL.

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

2. Webhooks are listed, including the name and ARN for each webhook.

   ```
   
   "webhooks": [
   
   
   "url": "https://webhooks.domain.com/
   
   trigger111111111EXAMPLE111111111111111": {
   
   "authenticationConfiguration": {
   
   "SecretToken": "Secret"
   
   },
   
   "name": "my-webhook",

   "authentication": "GITHUB_HMAC",
   
   "targetPipeline": "my-Pipeline",
   
   "targetAction": "Source",

   "filters": [

   
   
   "jsonPath": "$ref",
   
   "matchEquals": "refs/heads/{Branch}"}
   
   ]}
   
   },
   
   
   ]}
   
   ]}
   ```

3. In GitHub, choose your repository.

4. Choose Settings, and then choose Webhooks.

   View the webhook information for your repository.

Edit the webhook for your GitHub source

You can use the AWS CLI to edit the webhook for your repository.

- If you use the console to edit the GitHub source action for your pipeline, the webhook is updated for you (and re-registered, if appropriate).

- If you are not updating the webhook name, and you are not changing the GitHub repository, you can use the AWS CLI to update the webhook. See Example 1.
• If you are changing the webhook name or GitHub repository name, you must edit the source action in the console or delete and recreate the webhook in the CLI. After you create the webhook, you also register it. See Example 2.

Example 1: To update a webhook secret

1. In a text editor, edit the JSON file for the webhook you want to update. This example modifies the sample file that was used to create the webhook in Create a webhook for a GitHub source (p. 199). This sample changes the secret token of the webhook named "my-webhook".

   ```json
   {"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. Call the put-webhook command and include the --cli-input and --region parameters.

   The following sample command updates a webhook with the modified "webhook_json" JSON file.

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

   3. The output returns the webhook details and the new secret.

   Note
   You can edit the GitHub source action in the console. This allows CodePipeline to manage webhooks for you.

Example 2: To update a webhook name or GitHub repository

1. Use the steps in Delete the webhook for your GitHub source (p. 203) to deregister and delete the existing webhook that is associated with the old webhook name or GitHub repository.
2. Use the steps in Create a webhook for a GitHub source (p. 199) to recreate the webhook.

   Note
   You can edit the GitHub source action in the console. This allows CodePipeline to manage webhooks for you.

Delete the webhook for your GitHub source

To use the AWS CLI to delete a webhook:

1. You must deregister the webhook before you delete it. Call the deregister-webhook-with-third-party command and include the --webhook-name parameter.

   The following sample command deregisters the webhook named "my-webhook".

   ```shell
   aws codepipeline deregister-webhook-with-third-party --webhook-name "my-webhook"
   ```
Use webhooks to start a pipeline (GitHub source)

1. Call the `aws codepipeline deregister-webhook-with-third-party` command and include the `--webhook-name` parameter.
   
   The following sample command deregisters the webhook named "my-webhook".
   
   ```
   aws codepipeline deregister-webhook-with-third-party --webhook-name my-webhook
   ```

2. Call the `delete-webhook` command and include the `--name` parameter.
   
   The following sample command deletes the webhook named "my-webhook".
   
   ```
   aws codepipeline delete-webhook --name my-webhook
   ```

Tag a webhook in CodePipeline

You can apply tags to your webhooks in CodePipeline. Tags are key-value pairs associated with AWS resources. For information about CodePipeline resource tagging, use cases, tag key and value constraints, and supported resource types, see Tagging resources (p. 168).

You can specify tags when you create a webhook. You can add, remove, and update the values of tags in a webhook. You can add up to 50 tags to each webhook.

Topics

- Add tags to an existing webhook (p. 204)
- View tags for a webhook (p. 204)
- Edit tags for a webhook (p. 205)
- Remove tags for a webhook (p. 205)

Add tags to an existing webhook

Follow these steps to use the AWS CLI to add a tag to a webhook. To add a tag to a webhook when you create it, see Create a webhook for a GitHub source (p. 199).

In these steps, we assume that you have already installed a recent version of the AWS CLI or updated to the current version. For more information, see Installing the AWS Command Line Interface.

At the terminal or command line, run the `tag-resource` command, specifying the Amazon Resource Name (ARN) of the webhook where you want to add tags and the key and value of the tag you want to add. You can add more than one tag to a webhook. For example, to tag a webhook named `MyWebhook` with two tags, a tag key named `Project` with the tag value of `NewProject`, and a tag key named `ApplicationName` with the tag value of `MyApplication`:

```
key=ApplicationName,value=MyApplication
```

If successful, this command returns nothing.

View tags for a webhook

Follow these steps to use the AWS CLI to view the AWS tags for a webhook. If no tags have been added, the returned list is empty.

At the terminal or command line, run the `list-tags-for-resource` command. For example, to view a list of tag keys and tag values for a webhook named `MyWebhook` with the ARN `arn:aws:codepipeline:us-west-2:account-id:webhook:MyWebhook`:

```
```
If successful, this command returns information similar to the following:

```json
{
  "tags": {
    "Project": "NewProject",
    "ApplicationName": "MyApplication"
  }
}
```

## Edit tags for a webhook

Follow these steps to use the AWS CLI to update a tag for a webhook. You can change the value for an existing key or add another key. You can also remove tags from a webhook, as shown in the next section.

At the terminal or command line, run the `tag-resource` command, specifying the ARN of the webhook where you want to update a tag and specify the tag key and tag value:

```bash
```

## Remove tags for a webhook

Follow these steps to use the AWS CLI to remove a tag from a webhook. When you remove tags from the associated resource, the tags are deleted.

**Note**

If you delete a webhook, all tag associations are removed from the webhook. You do not have to remove tags before you delete a webhook.

At the terminal or command line, run the `untag-resource` command, specifying the ARN of the webhook where you want to remove tags and the tag key of the tag you want to remove. For example, to remove a tag on a webhook named `MyWebhook` with the tag key `Project`:

```bash
```

If successful, this command returns nothing. To verify the tags associated with the webhook, run the `list-tags-for-resource` command.

## Create a webhook for a GitHub source (AWS CloudFormation template)

To use AWS CloudFormation to create a webhook, update your template as described here.

### To add parameters and create a webhook in your template

We strongly recommend that you use AWS Secrets Manager to store your credentials. If you use Secrets Manager, you must have already configured and stored your secret parameters in Secrets Manager. This example uses dynamic references to AWS Secrets Manager for the GitHub credentials for your webhook. For more information, see Using Dynamic References to Specify Template Values.

**Important**

When passing secret parameters, do not enter the value directly into the template. The value is rendered as plaintext and is therefore readable. For security reasons, do not use plaintext in your AWS CloudFormation template to store your credentials.

When you use the CLI or AWS CloudFormation to create a pipeline and add a webhook, you must disable periodic checks.
Use webhooks to start a pipeline (GitHub source)

**Note**
To disable periodic checks, you must explicitly add the `PollForSourceChanges` parameter and set it to false, as detailed in the final procedure below. Otherwise, the default for a CLI or AWS CloudFormation pipeline is that `PollForSourceChanges` defaults to true and does not display in the pipeline structure output. For more information about `PollForSourceChanges` defaults, see Default settings for the `PollForSourceChanges` parameter (p. 466).

1. **In the template, under Resources, add your parameters:**

   **YAML**
   ```yaml
   Parameters:
     GitHubOwner:
       Type: String
   ...
   ```

   **JSON**
   ```json
   { "Parameters": {
     "BranchName": {
       "Description": "GitHub branch name",
       "Type": "String",
       "Default": "master"
     },
     "GitHubOwner": {
       "Type": "String"
     },
   ...
   ```

2. **Use the `AWS::CodePipeline::Webhook` AWS CloudFormation resource to add a webhook.**

   **Note**
   The `TargetAction` you specify must match the `Name` property of the source action defined in the pipeline.

   If `RegisterWithThirdParty` is set to true, make sure the user associated to the OAuthToken can set the required scopes in GitHub. The token and webhook require the following GitHub scopes:

   - `repo` - used for full control to read and pull artifacts from public and private repositories into a pipeline.
   - `admin:repo_hook` - used for full control of repository hooks.

   Otherwise, GitHub returns a 404. For more information about the 404 returned, see https://help.github.com/articles/about-webhooks.

   **YAML**
   ```yaml
   AppPipelineWebhook:
     Type: AWS::CodePipeline::Webhook
     Properties:
       Authentication: GITHUB_HMAC
       AuthenticationConfiguration:
       Filters:
         - JsonPath: "$.ref"
   ```
Use webhooks to start a pipeline (GitHub source)

MatchEquals: refs/heads/{Branch}
TargetPipeline: !Ref AppPipeline
TargetAction: SourceAction
Name: AppPipelineWebhook
TargetPipelineVersion: !GetAtt AppPipeline.Version
RegisterWithThirdParty: true

```json
"AppPipelineWebhook": {
  "Type": "AWS::CodePipeline::Webhook",
  "Properties": {
    "Authentication": "GITHUB_HMAC",
    "AuthenticationConfiguration": {
      "SecretToken": "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}"
    },
    "Filters": [
      {"JsonPath": ".ref",
       "MatchEquals": "refs/heads/{Branch}"
      }
    ],
    "TargetPipeline": {
      "Ref": "AppPipeline"
    },
    "TargetAction": "SourceAction",
    "Name": "AppPipelineWebhook",
    "TargetPipelineVersion": {
      "Fn::GetAtt": [
        "AppPipeline",
        "Version"
      ]
    },
    "RegisterWithThirdParty": true
  }
},
```

3. Save the updated template to your local computer, and then open the AWS CloudFormation console.
4. Choose your stack, and then choose **Create Change Set for Current Stack**.
5. Upload the template, and then view the changes listed in AWS CloudFormation. These are the changes to be made to the stack. You should see your new resources in the list.
6. Choose **Execute**.

**To edit your pipeline’s PollForSourceChanges parameter**

**Important**
When you create a pipeline with this method, the PollForSourceChanges parameter defaults to true if it is not explicitly set to false. When you add event-based change detection, you must add the parameter to your output and set it to false to disable polling. Otherwise, your pipeline starts twice for a single source change. For details, see Default settings for the PollForSourceChanges parameter (p. 466).

- In the template, change PollForSourceChanges to false. If you did not include PollForSourceChanges in your pipeline definition, add it and set it to false.
Why am I making this change? Changing this parameter to `false` turns off periodic checks so you can use event-based change detection only.

**YAML**

```yaml
Name: Source
Actions:
  - Name: SourceAction
    ActionTypeId: 
      Category: Source
      Owner: ThirdParty
      Version: 1
      Provider: GitHub
    OutputArtifacts:
      - Name: SourceOutput
    Configuration:
      Owner: !Ref GitHubOwner
      Repo: !Ref RepositoryName
      Branch: !Ref BranchName
      OAuthToken: 
        {{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
      PollForSourceChanges: false
      RunOrder: 1
```

**JSON**

```json
{
  "Name": "Source",
  "Actions": [
    {
      "Name": "SourceAction",
      "ActionTypeId": {
        "Category": "Source",
        "Owner": "ThirdParty",
        "Version": 1,
        "Provider": "GitHub"
      },
      "OutputArtifacts": [
        {
          "Name": "SourceOutput"
        }
      ],
      "Configuration": {
        "Owner": {
          "Ref": "GitHubOwner"
        },
        "Repo": {
          "Ref": "RepositoryName"
        },
        "Branch": {
          "Ref": "BranchName"
        },
        "OAuthToken": 
          {{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}",
        "PollForSourceChanges": false
      },
      "RunOrder": 1
    }
  ]
}
```
Use CloudWatch Events to start a pipeline (Amazon ECR source)

You can use Amazon CloudWatch Events to trigger pipelines to start when rule or schedule criteria are met. For pipelines with an Amazon ECR source, an Amazon CloudWatch Events rule detects source changes and then starts your pipeline. When you use the console to create or change a pipeline, the rule and all associated resources are created for you. If you create or change a pipeline with an Amazon ECR source in the AWS CLI or AWS CloudFormation, you must use these steps to create the Amazon CloudWatch Events rule and all associated resources manually.

In Amazon CloudWatch Events, you create a rule to detect and react to changes in the state of the pipeline's defined source.

Topics

• Create a CloudWatch Events rule for an Amazon ECR source (console) (p. 209)
• Create a CloudWatch Events rule for an Amazon ECR source (CLI) (p. 210)
• Create a CloudWatch Events rule for an Amazon ECR source (AWS CloudFormation template) (p. 212)

Create a CloudWatch Events rule for an Amazon ECR source (console)

To create a CloudWatch Events rule for use in CodePipeline operations (Amazon ECR source)

2. In the navigation pane, choose Events.
3. Choose Create rule, and then under Event source, from Service Name, choose EC2 Container Registry.
4. In Event Source, choose Event Pattern.

Choose Edit, and then paste the following example event pattern in the Event Source window for a cwe-test repository with an image tag of cli-testing:

```json
{
  "detail-type": [
    "ECR Image Action"
  ],
  "source": [
    "aws.ecr"
  ],
  "detail": {
    "action-type": [
      "PUSH"
    ],
    "image-tag": [
      "latest"
    ],
    "repository-name": [
      "cwe-test"
    ],
    "result": [
      "SUCCESS"
    ]
  }
}
```
Use a CloudWatch Events rule to start a pipeline (Amazon ECR source)

5. Choose **Save**.

   In the **Event Pattern Preview** pane, view the rule.

6. In **Targets**, choose **CodePipeline**.

7. Enter the pipeline ARN for the pipeline to be started when triggered by this rule.

   **Note**
   
   You can find the pipeline ARN in the metadata output after you run the `get-pipeline` command. The pipeline ARN is constructed in this format:
   
   `arn:aws:codepipeline:region:account:pipeline-name`
   
   Sample pipeline ARN:
   
   `arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline`

8. Create or specify an IAM service role that grants Amazon CloudWatch Events permissions to invoke the target associated with your Amazon CloudWatch Events rule (in this case, the target is CodePipeline).

   - Choose **Create a new role for this specific resource** to create a service role that gives Amazon CloudWatch Events permissions to your start your pipeline executions when triggered.
   - Choose **Use existing role** to enter a service role that gives Amazon CloudWatch Events permissions to your start your pipeline executions when triggered.

9. Review your rule setup to make sure it meets your requirements.

10. Choose **Configure details**.

11. On the **Configure rule details** page, enter a name and description for the rule, and then choose **State** to enable the rule.

12. If you're satisfied with the rule, choose **Create rule**.

### Create a CloudWatch Events rule for an Amazon ECR source (CLI)

Call the `put-rule` command, specifying:

- A name that uniquely identifies the rule you are creating. This name must be unique across all of the pipelines you create with CodePipeline associated with your AWS account.
- The event pattern for the source and detail fields used by the rule. For more information, see Amazon CloudWatch Events and Event Patterns.

#### To create a CloudWatch Events rule with Amazon ECR as the event source and CodePipeline as the target

1. Add permissions for Amazon CloudWatch Events to use CodePipeline to invoke the rule. For more information, see **Using Resource-Based Policies for Amazon CloudWatch Events**.

   a. Use the following sample to create the trust policy that allows CloudWatch Events to assume the service role. Name the trust policy `trustpolicyforCWE.json`.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "events.amazonaws.com"
            }
        }
    ]
}
```
Use a CloudWatch Events rule to start a pipeline (Amazon ECR source)

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["codepipeline:StartPipelineExecution"],
        ]
    ]
}
```

b. Use the following command to create the Role-for-MyRule role and attach the trust policy.

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document file://trustpolicyforCWE.json
```

c. Create the permissions policy JSON, as shown in this sample, for the pipeline named MyFirstPipeline. Name the permissions policy permissionspolicyforCWE.json.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["codepipeline:StartPipelineExecution"],
        ]
    ]
}
```

d. Use the following command to attach the CodePipeline-Permissions-Policy-for-CWE permissions policy to the Role-for-MyRule role.

**Why am I making this change?** Adding this policy to the role creates permissions for CloudWatch Events.

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

2. Call the `put-rule` command and include the `--name`, `--event-pattern`, and `--role-arn` parameters.

**Why am I making this change?** We must create an event with a rule that specifies how an image push must be made, and a target that names the pipeline to be triggered by the event.

The following sample command creates a rule called MyECRRepoRule.

```
aws events put-rule --name "MyECRRepoRule" --event-pattern "{"detail-type": ["ECR Image Action"], "source": ["aws.ecr"], "detail": {"action-type": ["PUSH"], "image-tag": ["latest"], "repository-name": ["cwe-test"], "result": ["SUCCESS"]}}" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-MyRule"
```

**Note**
To view the full event pattern supported for Amazon ECR events, see Amazon ECR Events and EventBridge or Amazon Elastic Container Registry Events.

3. To add CodePipeline as a target, call the `put-targets` command and include the following parameters:

   - The `--rule` parameter is used with the `rule_name` you created by using `put-rule`.
   - The `--targets` parameter is used with the list Id of the target in the list of targets and the ARN of the target pipeline.
The following sample command specifies that for the rule called `MyECRRepoRule`, the target `Id` is composed of the number one, indicating that in a list of targets for the rule, this is target 1. The sample command also specifies an example `Arn` for the pipeline and the example `RoleArn` for the rule. The pipeline starts when something changes in the repository.

```
aws events put-targets --rule MyECRRepoRule --targets Id=1,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline,RoleArn=arn:aws:iam::80398EXAMPLE:role/Role-for-MyRule
```

### Create a CloudWatch Events rule for an Amazon ECR source (AWS CloudFormation template)

To use AWS CloudFormation to create a rule, use the template snippet as shown here.

**To update your pipeline AWS CloudFormation template and create CloudWatch Events rule**

1. In the template, under `Resources`, use the `AWS::IAM::Role` AWS CloudFormation resource to configure the IAM role that allows your event to start your pipeline. This entry creates a role that uses two policies:

   - The first policy allows the role to be assumed.
   - The second policy provides permissions to start the pipeline.

**Why am I making this change?** We must create a role that can be assumed by CloudWatch Events to start an execution in our pipeline.

**YAML**

```yaml
AmazonCloudWatchEventRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: 2012-10-17
      Statement:
        -
          Effect: Allow
          Principal:
            Service:
              - events.amazonaws.com
          Action: sts:AssumeRole
          Path: /
        -
          PolicyName: cwe-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
```

**JSON**

```
"AmazonCloudWatchEventRole": {
```
Use a CloudWatch Events rule to start a pipeline (Amazon ECR source)

2. In the template, under Resources, use the `AWS::Events::Rule` AWS CloudFormation resource to add a CloudWatch Events rule for the Amazon ECR source. This event pattern creates an event that monitors push changes to your repository. When CloudWatch Events detects a repository state change, the rule invokes `StartPipelineExecution` on your target pipeline.

**Why am I making this change?** We must create an event with a rule that specifies how an image push must be made, and a target that names the pipeline to be triggered by the event.

This snippet uses an image named `cwe-test` with a tag of `latest`.

YAML

```yaml
AmazonCloudWatchEventRule:
  Type: 'AWS::Events::Rule'
  Properties:
    EventPattern:
```
Use a CloudWatch Events rule to start a pipeline (Amazon ECR source)

**detail:**
- **action-type:** `[PUSH]`
- **image-tag:** `[latest]`
- **repository-name:** `[cwe-test]`
- **result:** `[SUCCESS]`
- **detail-type:** `[ECR Image Action]`
- **source:** `[aws.ecr]`

**Targets:**
- Arn: 
  - `!Join`
  - ''
  - 'arn:aws:codepipeline:'
  - !Ref 'AWS::Region'`
  - ':'
  - !Ref 'AWS::AccountId'
  - ':'
  - !Ref AppPipeline

**RoleArn:** 
- `!GetAtt`
  - `AmazonCloudWatchEventRole`
  - Arn

Id: codepipeline-AppPipeline

**JSON**

```json
{
  "AmazonCloudWatchEventRule": {
    "Type": "AWS::Events::Rule",
    "Properties": {
      "EventPattern": {
        "detail": {
          "action-type": ["PUSH"],
          "image-tag": ["latest"],
          "repository-name": ["cwe-test"],
          "result": ["SUCCESS"]
        },
        "detail-type": ["ECR Image Action"],
        "source": ["aws.ecr"]
      },
      "Targets": [
        {
          "Arn": {
            "Fn::Join": [
              "",
              ["arn:aws:codepipeline:",
               { "Ref": "AWS::Region" }
              ],
              ":",
              ["Ref": "AWS::AccountId" 
              ],
              ":",
              ["Ref": "AppPipeline"
              ]
            ]
          },
          "RoleArn": {
            "Fn::GetAtt": [
              "AmazonCloudWatchEventRole",
              "Arn"
            ]
          }
        }
      ]
    }
  }
}
```
Use periodic checks to start a pipeline

Pipelines start automatically when repository changes are detected. One change detection method is periodic checks. Periodic checks can be enabled or disabled using the PollForSourceChanges flag. If you use the CLI to create or edit a pipeline, this parameter defaults to true. This is not the recommended configuration. Instead, edit your pipeline to use the recommended change-detection method and then set this parameter to false.

Note
Most source actions in CodePipeline, such as GitHub, require either a configured change detection resource (such as a webhook or CloudWatch Events rule) or use the option to poll the repository for source changes. For pipelines with a Bitbucket Cloud source action, you do not have to set up a webhook or default to polling. The connections action manages your source change detection for you.

For more information about creating a pipeline with the recommended configuration, see Create a Pipeline (Console) (p. 222) and Create a Pipeline (CLI) (p. 228). For more information about updating an action or pipeline with the recommended configuration, see Edit a Pipeline (Console) (p. 232) and Edit a Pipeline (CLI) (p. 233).

For more information, see Change-Detection Methods Used to Start Pipelines Automatically (p. 172).

Start a pipeline manually in AWS CodePipeline

By default, a pipeline starts automatically when it is created and any time a change is made in a source repository. However, you might want to rerun the most recent revision through the pipeline a second time. You can use the CodePipeline console or the AWS CLI and start-pipeline-execution command to manually rerun the most recent revision through your pipeline.

Topics
- Start a pipeline manually (console) (p. 216)
- Start a pipeline manually (CLI) (p. 216)
Start a pipeline manually (console)

To manually start a pipeline and run the most recent revision through a pipeline

2. In Name, choose the name of the pipeline you want to start.
3. On the pipeline details page, choose Release change. This starts the most recent revision available in each source location specified in a source action through the pipeline.

Start a pipeline manually (CLI)

To manually start a pipeline and run the most recent version of an artifact through a pipeline

1. Open a terminal (Linux, macOS, or Unix) or command prompt (Windows) and use the AWS CLI to run the start-pipeline-execution command, specifying the name of the pipeline you want to start. For example, to start running the last change through a pipeline named MyFirstPipeline:

```bash
aws codepipeline start-pipeline-execution --name MyFirstPipeline
```
2. To verify success, view the returned object. This command returns an execution ID, similar to the following:

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

Note
After you have started the pipeline, you can monitor its progress in the CodePipeline console or by running the get-pipeline-state command. For more information, see View pipeline details and history (console) (p. 237) and View pipeline details and history (CLI) (p. 244).

Use Amazon CloudWatch Events to start a pipeline on a schedule

You can set up a rule in Amazon CloudWatch Events to start a pipeline on a schedule.

Create a CloudWatch Events rule that schedules your pipeline to start (console)

To create a CloudWatch Events rule with a schedule as the event source

2. In the navigation pane, choose Events.
3. Choose Create rule, and then under Event Source, choose Schedule.
4. Set up the schedule using a fixed rate or expression. For information, see Schedule Expression for Rules.
5. In Targets, choose CodePipeline.
6. Enter the pipeline ARN for the pipeline execution that starts when triggered by this schedule.
Note
You can find the pipeline ARN in the metadata output after you run the `get-pipeline` command.

7. Choose one of the following to create or specify an IAM service role that gives Amazon CloudWatch Events permissions to invoke the target associated with your Amazon CloudWatch Events rule (in this case, the target is CodePipeline).

- Choose **Create a new role for this specific resource** to create a service role that grants Amazon CloudWatch Events permissions to start your pipeline executions when triggered.
- Choose **Use existing role** to enter a service role that grants Amazon CloudWatch Events permissions to start your pipeline executions when triggered.

8. Choose **Configure details**.

9. On the **Configure rule details** page, enter a name and description for the rule, and then choose **State** to enable the rule.

10. If you're satisfied with the rule, choose **Create rule**.

### Create a CloudWatch Events rule that schedules your pipeline to start (CLI)

To use the AWS CLI to create a rule, call the `put-rule` command, specifying:

- A name that uniquely identifies the rule you are creating. This name must be unique across all of the pipelines you create with CodePipeline associated with your AWS account.
- The schedule expression for the rule.

**To create a CloudWatch Events rule with a schedule as the event source**

1. Call the `put-rule` command and include the `--name` and `--schedule-expression` parameters.

   Examples:

   ```
   The following sample command uses `--schedule-expression` to create a rule called MyRule2 that filters CloudWatch Events on a schedule.
   ```
   ```bash
   aws events put-rule --schedule-expression 'cron(15 10 ? * 6L 2002-2005)' --name MyRule2
   ```

2. Grant permissions for Amazon CloudWatch Events to use CodePipeline to invoke the rule. For more information, see Using Resource-Based Policies for Amazon CloudWatch Events.

   a. Use the following sample to create the trust policy to allow Amazon CloudWatch Events to assume the service role. Name it `trustpolicyforCWE.json`.

   ```json
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
           "Service": "events.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
       }
     ]
   }
   ```
b. Use the following command to create the Role-for-MyRule role and attach the trust policy.

```bash
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document file://trustpolicyforCWE.json
```

c. Create the permissions policy JSON as shown in this sample for the pipeline named MyFirstPipeline. Name the permissions policy permissionspolicyforCWE.json.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["codepipeline:StartPipelineExecution"],
    }
  ]
}
```

d. Use the following command to attach the new CodePipeline-Permissions-Policy-for-CWE permissions policy to the Role-for-MyRule role you created.

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

---

**Stop a pipeline execution in CodePipeline**

When a pipeline execution starts to run through a pipeline, it enters one stage at a time and locks the stage while all action executions in the stage are running. These in-progress actions must be handled in a way so that, when the pipeline execution is stopped, the actions are either allowed to complete or abandoned.

There are two ways to stop a pipeline execution:

- **Stop and wait**: AWS CodePipeline waits to stop the execution until all in-progress actions are completed (that is, the actions have a Succeeded or Failed status). This option preserves in-progress actions. The execution is in a Stopping state until the in-progress actions are complete. Then the execution is in a Stopped state. The stage unlocks after the actions are complete.

  If you choose to stop and wait, and you change your mind while your execution is still in a Stopping state, you can then choose to abandon.

- **Stop and abandon**: AWS CodePipeline stops the execution without waiting for in-progress actions to complete. The execution is in a Stopping state for a very short time while the in-progress actions are abandoned. After the execution is stopped, the action execution is in an Abandoned state while the pipeline execution is in a Stopped state. The stage unlocks.

  For a pipeline execution in a Stopped state, the actions in the stage where the execution stopped can be retried.

  **Warning**

  This option can lead to failed tasks or out of sequence tasks.

---

**Topics**
Stop a pipeline execution (console)


2. Do one of the following:
   - In Name, choose the name of the pipeline with the execution you want to stop. On the pipeline details page, choose Stop execution.
   - Choose View history. On the history page, choose Stop execution.

3. On the Stop execution page, under Select execution, choose the execution you want to stop.

   - Choose a stop mode for the execution
     - Stop and wait: Wait until all in-progress actions are complete.
     - Stop and abandon: Don’t wait until the in-progress actions are complete. Warning: This option can lead to failed actions.

   - Stop execution comments - optional

   - Cancel

   - Stop
4. Under **Select an action to apply to execution**, choose one of the following:

   • To make sure the execution does not stop until all in-progress actions are complete, choose **Stop and wait**.

     **Note**
     You cannot choose to stop and wait if the execution is already in a Stopping state, but you can choose to stop and abandon.

   • To stop without waiting for in-progress actions to complete, choose **Stop and abandon**.

     **Warning**
     This option can lead to failed tasks or out of sequence tasks.

5. (Optional) Enter comments. These comments, along with the execution status, are displayed on the history page for the execution.

6. Choose **Stop**.

   **Important**
   This action cannot be undone.

7. View the execution status in the pipeline visualization as follows:

   • If you chose to stop and wait, the selected execution continues until in-progress actions are completed.
   • The success banner message is displayed at the top of the console.
   • In the current stage, in-progress actions continue in an InProgress state. While the actions are in progress, the pipeline execution is in a Stopping state.

After the actions complete (that is, the action fails or succeeds), the pipeline execution changes to a Stopped state and the action changes to a Failed or Succeeded state. You can also view the action state on the execution details page. You can view the execution status on the execution history page or the execution details page.

   • The pipeline execution changes to a Stopping state briefly, and then it changes to a Stopped state. You can view the execution status on the execution history page or the execution details page.

   • If you chose to stop and abandon, the execution does not wait for in-progress actions to complete.
   • The success banner message is displayed at the top of the console.
   • In the current stage, in-progress actions change to a status of Abandoned. You can also view the action status on the execution details page.
   • The pipeline execution changes to a Stopping state briefly, and then it changes to a Stopped state. You can view the execution status on the execution history page or the execution details page.

You can view the pipeline execution status in the execution history view and the detailed history view.

---

**Stop a pipeline execution (CLI)**

To use the AWS CLI to manually stop a pipeline, use the `stop-pipeline-execution` command with the following parameters:

- Execution ID (required)
- Comments (optional)
- Pipeline name (required)
- Abandon flag (optional, the default is false)
Command format:

```bash
aws codepipeline stop-pipeline-execution --pipeline-name Pipeline_Name --pipeline-execution-id Execution_ID [--abandon | --no-abandon] [--reason STOP_EXECUTION_REASON]
```

1. Open a terminal (Linux, macOS, or Unix) or command prompt (Windows).
2. To stop a pipeline execution, choose one of the following:

   • To make sure the execution does not stop until all in-progress actions are complete, choose to stop and wait. You can do this by including the `no-abandon` parameter. If you do not specify the parameter, the command defaults to stop and wait. Use the AWS CLI to run the `stop-pipeline-execution` command, specifying the name of the pipeline and the execution ID. For example, to stop a pipeline named `MyFirstPipeline` with the stop and wait option specified:

   ```bash
   aws codepipeline stop-pipeline-execution --pipeline-name MyFirstPipeline --pipeline-execution-id d-EXAMPLE --no-abandon
   ```

   For example, to stop a pipeline named `MyFirstPipeline`, defaulting to the stop and wait option, and choosing to include comments:

   ```bash
   aws codepipeline stop-pipeline-execution --pipeline-name MyFirstPipeline --pipeline-execution-id d-EXAMPLE --reason "Stopping execution after the build action is done"
   ```

   **Note**
   You cannot choose to stop and wait if the execution is already in a **Stopping** state. You can choose to stop and abandon an execution that is already in a **Stopping** state.

   • To stop without waiting for in-progress actions to complete, choose to stop and abandon. Include the `abandon` parameter. Use the AWS CLI to run the `stop-pipeline-execution` command, specifying the name of the pipeline and the execution ID.

   For example, to stop a pipeline named `MyFirstPipeline`, specifying the abandon option, and choosing to include comments:

   ```bash
   aws codepipeline stop-pipeline-execution --pipeline-name MyFirstPipeline --pipeline-execution-id d-EXAMPLE --abandon --reason "Stopping execution for a bug fix"
   ```

### Create a pipeline in CodePipeline

You can use the AWS CodePipeline console or the AWS CLI to create a pipeline. Pipelines must have at least two stages. The first stage of a pipeline must be a source stage. The pipeline must have at least one other stage that is a build or deployment stage.

You can add actions to your pipeline that are in an AWS Region different from your pipeline. A cross-Region action is one in which an AWS service is the provider for an action and the action type or provider type are in an AWS Region different from your pipeline. For more information, see [Add a cross-Region action in CodePipeline](p. 364).

You can also create pipelines that build and deploy container-based applications by using Amazon ECS as the deployment provider. Before you create a pipeline that deploys container-based applications with Amazon ECS, you must create an image definitions file as described in [Image definitions file reference](p. 508).

CodePipeline uses change detection methods to start your pipeline when a source code change is pushed. These detection methods are based on source type:
Create a pipeline (console)

To create a pipeline in the console, you must provide the source file location and information about the providers you will use for your actions.

When you use the console to create a pipeline, you must include a source stage and one or both of the following:

- A build stage.
- A deployment stage.

When you use the pipeline wizard, CodePipeline creates the names of stages (source, build, staging). These names cannot be changed. You can use more specific names (for example, BuildToGamma or DeployToProd) to stages you add later.

Step 1: Create and name your pipeline

2. On the Welcome page, choose Create pipeline.
   
   If this is your first time using CodePipeline, choose Get Started.
3. On the Step 1: Choose pipeline settings page, in Pipeline name, enter the name for your pipeline.

   In a single AWS account, each pipeline you create in an AWS Region must have a unique name. Names can be reused for pipelines in different Regions.

   **Note**
   After you create a pipeline, you cannot change its name. For information about other limitations, see Quotas in AWS CodePipeline (p. 521).
4. In Service role, do one of the following:
   
   - Choose New service role to allow CodePipeline to create a new service role in IAM. In Role name, the role and policy name both default to this format: AWSCodePipelineServiceRole-<region>-<pipeline name>. For example, this is the service role created for a pipeline named MyPipeline: AWSCodePipelineServiceRole-eu-west-2-MyPipeline.
   
   - Choose Existing service role to use a service role already created in IAM. In Role ARN, choose your service role ARN from the list.
Note
Depending on when your service role was created, you might need to update its permissions to support additional AWS services. For information, see Add permissions to the CodePipeline service role (p. 446).

For more information about the service role and its policy statement, see Manage the CodePipeline service role (p. 445).

5. (Optional) Expand **Advanced settings**.

6. In **Artifact store**, do one of the following:
   a. Choose **Default location** to use the default artifact store, such as the S3 artifact bucket designated as the default, for your pipeline in the AWS Region you have selected for your pipeline.
   b. Choose **Custom location** if you already have an artifact store, such as an S3 artifact bucket, in the same Region as your pipeline. In **Bucket**, choose the bucket name.

Note
This is not the source bucket for your source code. This is the artifact store for your pipeline. A separate artifact store, such as an S3 bucket, is required for each pipeline. When you create or edit a pipeline, you must have an artifact bucket in the pipeline Region and one artifact bucket per AWS Region where you are running an action. For more information, see Input and output artifacts (p. 15) and CodePipeline pipeline structure reference (p. 454).

7. In **Encryption key**, do one of the following:
   a. To use the CodePipeline default AWS-managed AWS KMS customer master key (CMK) to encrypt the data in the pipeline artifact store (S3 bucket), choose **Default AWS Managed Key**.
   b. To use your CMK to encrypt the data in the pipeline artifact store (S3 bucket), choose **Customer Managed Key**. In **KMS customer master key**, choose the key ID, key ARN, or alias ARN.

8. Choose **Next**.

**Step 2: Create a source stage**

- On the **Step 2: Add source stage** page, in **Source provider**, choose the type of repository where your source code is stored, specify its required options, and then choose **Next step**.

  - **For GitHub**:
    1. Choose **Connect to GitHub**. If you are prompted to sign in, enter your GitHub credentials.

       Important
       Do not enter your AWS credentials.

    2. If this is your first time connecting to GitHub from CodePipeline for this Region, you are asked to authorize application access to your account. Review the permissions required for integration, and then, if you want to continue, choose **Authorize application**. When you connect to GitHub in the console, the following resources are created for you:
       - CodePipeline uses an OAuth token to create an authorized application that is managed by CodePipeline.

       Note
       In GitHub, there is a limit to the number of OAuth tokens you can use for an application, such as CodePipeline. If you exceed this limit, retry the connection to allow CodePipeline to reconnect by reusing existing tokens. For more information,
see Pipeline error: I receive a pipeline error that says: "Could not access the GitHub repository" or "Unable to connect to the GitHub repository" (p. 399).

- CodePipeline creates a webhook in GitHub to detect source changes and then start your pipeline when a change occurs. In addition to the webhook, CodePipeline:
  - Randomly generates a secret and uses it to authorize the connection to GitHub.
  - Generates the webhook URL using the public endpoint for the Region and registers it with GitHub. This subscribes the URL to receive repository events.

3. Choose the GitHub repository you want to use as the source location for your pipeline. In Branch, from the drop-down list, choose the branch you want to use.

- For Amazon S3:
  1. In Amazon S3 location, provide the S3 bucket name and path to the object in a bucket with versioning enabled. The format of the bucket name and path looks like this:

    \[s3://bucketName/folderName/objectName\]

    **Note**
    When Amazon S3 is the source provider for your pipeline, you may zip your source file or files into a single .zip and upload the .zip to your source bucket. You may also upload a single unzipped file; however, downstream actions that expect a .zip file will fail.

    2. After you choose the S3 source bucket, CodePipeline creates the Amazon CloudWatch Events rule and the AWS CloudTrail trail to be created for this pipeline. Accept the defaults under Change detection options. This allows CodePipeline to use Amazon CloudWatch Events and AWS CloudTrail to detect changes for your new pipeline. Choose Next.

- For AWS CodeCommit:
  - In Repository name, choose the name of the CodeCommit repository you want to use as the source location for your pipeline. In Branch name, from the drop-down list, choose the branch you want to use.
  
  After you choose the CodeCommit repository name and branch, a message is displayed in Change detection options showing the Amazon CloudWatch Events rule to be created for this pipeline. Accept the defaults under Change detection options. This allows CodePipeline to use Amazon CloudWatch Events to detect changes for your new pipeline.

- For Amazon ECR:
  - In Repository name, choose the name of your Amazon ECR repository.
  - In Image tag, specify the image name and version, if different from LATEST.
  - In Output artifacts, choose the output artifact default, such as MyApp, that contains the image name and repository URI information you want the next stage to use.

    For a tutorial about creating a pipeline for Amazon ECS with CodeDeploy blue-green deployments that includes an Amazon ECR source stage, see Tutorial: Create a Pipeline with an Amazon ECR Source and ECS-to-CodeDeploy Deployment.

    When you include an Amazon ECR source stage in your pipeline, the source action generates an `imageDetail.json` file as an output artifact when you commit a change. For information about the `imageDetail.json` file, see `imageDetail.json` file for Amazon ECS blue/green deployment actions (p. 510).

    **Note**
    The object and file type must be compatible with the deployment system you plan to use (for example, Elastic Beanstalk or CodeDeploy). Supported file types might include .zip, .tar, and .tgz files. For more information about the supported container types for Elastic Beanstalk, see Customizing and Configuring Elastic Beanstalk Environments and Supported API Version 2015-02-05. 224
Step 3: Create a build stage

This step is optional if you plan to create a deployment stage.

- On the **Step 3: Add build stage** page, do one of the following, and then choose **Next**:
  
  - Choose **Skip build stage** if you plan to create a deployment stage.
  - From **Build provider**, choose a custom action provider of build services, and provide the configuration details for that provider. For an example of how to add Jenkins as a build provider, see **Tutorial: Create a four-stage pipeline** (p. 63).
  - From **Build provider**, choose **AWS CodeBuild**.

    In **Region**, choose the AWS Region where the resource exists. The **Region** field designates where the AWS resources are created for this action type and provider type. This field is displayed only for actions where the action provider is an AWS service. The **Region** field defaults to the same AWS Region as your pipeline.

    In **Project name**, choose your build project. If you have already created a build project in CodeBuild, choose it. Or you can create a build project in CodeBuild and then return to this task. Follow the instructions in **Create a Pipeline That Uses CodeBuild** in the **CodeBuild User Guide**.

    In **Environment variables**, to add CodeBuild environment variables to your build action, choose **Add environment variable**. Each variable is made up of three entries:

    - In **Name**, enter the name or key of the environment variable.
    - In **Value**, enter the value of the environment variable. If you choose **Parameter** for the variable type, make sure this value is the name of a parameter you have already stored in AWS Systems Manager Parameter Store.

      **Note**
      
      We strongly discourage the use of environment variables to store sensitive values, especially AWS secret key IDs and secret access keys. When you use the CodeBuild console or AWS CLI, environment variables are displayed in plain text. For sensitive values, we recommend that you use the **Parameter** type instead.

    - (Optional) In **Type**, enter the type of environment variable. Valid values are **Plaintext** or **Parameter**. The default is **Plaintext**.

      (Optional) In **Build type**, choose one of the following:

      - To run each build in a single build action execution, choose **Single build**.
      - To run multiple builds in the same build action execution, choose **Batch build**.

      (Optional) If you chose to run batch builds, you can choose **Combine all artifacts from batch into a single location** to place all build artifacts into a single output artifact.

Step 4: Create a deployment stage

This step is optional if you have already created a build stage.

- On the **Step 4: Add deploy stage** page, do one of the following, and then choose **Next**:

  - Choose **Skip deploy stage** if you created a build stage in the previous step.

    **Note**
    
    This option does not appear if you have already skipped the build stage.

  - In **Deploy provider**, choose a custom action that you have created for a deployment provider.
In Region, for cross-Region actions only, choose the AWS Region where the resource is created. The Region field designates where the AWS resources are created for this action type and provider type. This field only displays for actions where the action provider is an AWS service. The Region field defaults to the same AWS Region as your pipeline.

- In Deploy provider, fields are available for default providers as follows:
  - CodeDeploy
    
    In Application name, enter or choose the name of an existing CodeDeploy application. In Deployment group, enter the name of a deployment group for the application. Choose Next. You can also create an application, deployment group, or both in the CodeDeploy console.
  
  - AWS Elastic Beanstalk
    
    In Application name, enter or choose the name of an existing Elastic Beanstalk application. In Environment name, enter an environment for the application. Choose Next. You can also create an application, environment, or both in the Elastic Beanstalk console.
  
  - AWS OpsWorks Stacks
    
    In Stack, enter or choose the name of the stack you want to use. In Layer, choose the layer that your target instances belong to. In App, choose the application that you want to update and deploy. If you need to create an app, choose Create a new one in AWS OpsWorks.

    For information about adding an application to a stack and layer in AWS OpsWorks, see Adding Apps in the AWS OpsWorks User Guide.

    For an end-to-end example of how to use a simple pipeline in CodePipeline as the source for code that you run on AWS OpsWorks layers, see Using CodePipeline with AWS OpsWorks Stacks.

  - AWS CloudFormation
    
    Do one of the following:

    - In Action mode, choose Create or update a stack, enter a stack name and template file name, and then choose the name of a role for AWS CloudFormation to assume. Optionally, enter the name of a configuration file and choose an IAM capability option.

    - In Action mode, choose Create or replace a change set, enter a stack name and change set name, and then choose the name of a role for AWS CloudFormation to assume. Optionally, enter the name of a configuration file and choose an IAM capability option.

    For information about integrating AWS CloudFormation capabilities into a pipeline in CodePipeline, see Continuous Delivery with CodePipeline in the AWS CloudFormation User Guide.

  - Amazon ECS
    
    In Cluster name, enter or choose the name of an existing Amazon ECS cluster. In Service name, enter or choose the name of the service running on the cluster. You can also create a cluster and service. In Image filename, enter the name of the image definitions file that describes your service's container and image.

    Note
    
    The Amazon ECS deployment action requires an imagedefinitions.json file as an input to the deployment action. The default file name for the file is imagedefinitions.json. If you choose to use a different file name, you must provide it when you create the pipeline deployment stage. For more information, see imagedefinitions.json file for Amazon ECS standard deployment actions (p. 508).

    Choose Next.
**Note**  
Make sure your Amazon ECS cluster is configured with two or more instances. Amazon ECS clusters must contain at least two instances so that one is maintained as the primary instance and another is used to accommodate new deployments.

For a tutorial about deploying container-based applications with your pipeline, see Tutorial: Continuous Deployment with CodePipeline.

- **Amazon ECS (Blue/Green)**

Enter the CodeDeploy application and deployment group, Amazon ECS task definition, and AppSpec file information, and then choose Next.

**Note**  
The Amazon ECS (Blue/Green) action requires an imageDetail.json file as an input artifact to the deploy action. Because the Amazon ECR source action creates this file, pipelines with an Amazon ECR source action do not need to provide an imageDetail.json file. For more information, see imageDetail.json file for Amazon ECS blue/green deployment actions (p. 510).

For a tutorial about creating a pipeline for blue-green deployments to an Amazon ECS cluster with CodeDeploy, see Tutorial: Create a pipeline with an Amazon ECR source and ECS-to-CodeDeploy deployment (p. 116).

- **AWS Service Catalog**

Choose Enter deployment configuration if you want to use fields in the console to specify your configuration, or choose Configuration file if you have a separate configuration file. Enter product and configuration information, and then choose Next.

For a tutorial about deploying product changes to AWS Service Catalog with your pipeline, see Tutorial: Create a pipeline that deploys to AWS Service Catalog (p. 87).

- **Alexa Skills Kit**

In Alexa Skill ID, enter the skill ID for your Alexa skill. In Client ID and Client secret, enter the credentials generated using a Login with Amazon (LWA) security profile. In Refresh token, enter the refresh token you generated using the ASK CLI command for retrieving a refresh token. Choose Next.

For a tutorial about deploying Alexa skills with your pipeline and generating the LWA credentials, see Tutorial: Create a pipeline that deploys an Amazon Alexa skill (p. 131).

- **Amazon S3**

In Bucket, enter the name of the S3 bucket you want to use. Choose Extract file before deploy if the input artifact to your deploy stage is a ZIP file. If Extract file before deploy is selected, you may optionally enter a value for Deployment path to which your ZIP file will be unzipped. If it is not selected, you are required to to enter a value in S3 object key.

**Note**  
Most source and build stage output artifacts are zipped. All pipeline source providers except Amazon S3 zip your source files before providing them as the input artifact to the next action.

(Optional) In Canned ACL, enter the canned ACL to apply to the object deployed to Amazon S3.

**Note**  
Applying a canned ACL overwrites any existing ACL applied to the object.

(Optional) In Cache control, specify the cache control parameters for requests to download objects from the bucket. For a list of valid values, see the Cache-Control header field for API Version 2015-07-09.
HTTP operations. To enter multiple values in Cache control, use a comma between each value. You can add a space after each comma (optional), as shown in this example:

```
Cache control - optional
Set cache control for objects requested from your Amazon S3 bucket.

public, max-age=0, no-transform
```

The preceding example entry is displayed in the CLI as follows:

```
"CacheControl": "public, max-age=0, no-transform"
```

Choose Next.

For a tutorial about creating a pipeline with an Amazon S3 deployment action provider, see Tutorial: Create a pipeline that uses Amazon S3 as a deployment provider (p. 136).

**Step 5: Review the pipeline**

- On the Step 5: Review page, review your pipeline configuration, and then choose Create pipeline to create the pipeline or Previous to go back and edit your choices. To exit the wizard without creating a pipeline, choose Cancel.

Now that you've created your pipeline, you can view it in the console. The pipeline starts to run after you create it. For more information, see View pipeline details and history in CodePipeline (p. 237). For more information about making changes to your pipeline, see Edit a pipeline in CodePipeline (p. 231).

**Create a pipeline (CLI)**

To use the AWS CLI to create a pipeline, you create a JSON file to define the pipeline structure, and then run the `create-pipeline` command with the `--cli-input-json` parameter.

**Important**

You cannot use the AWS CLI to create a pipeline that includes partner actions. You must use the CodePipeline console instead.

For more information about pipeline structure, see CodePipeline pipeline structure reference (p. 454) and `create-pipeline` in the CodePipeline API Reference.

To create a JSON file, use the sample pipeline JSON file, edit it, and then call that file when you run the `create-pipeline` command.

**Prerequisites:**

You need the ARN of the service role you created for CodePipeline in Getting started with CodePipeline (p. 19). You use the CodePipeline service role ARN in the pipeline JSON file when you run the `create-pipeline` command. For more information about creating a service role, see Create the CodePipeline service role (p. 312). Unlike the console, running the `create-pipeline` command in the AWS CLI does not have the option to create the CodePipeline service role for you. The service role must already exist.

You need the name of an S3 bucket where artifacts for the pipeline are stored. This bucket must be in the same Region as the pipeline. You use the bucket name in the pipeline JSON file when you run the `create-pipeline` command. Unlike the console, running the `create-pipeline` command in the AWS CLI does not create an S3 bucket for storing artifacts. The bucket must already exist.

**Note**

You can also use the `get-pipeline` command to get a copy of the JSON structure of that pipeline, and then modify that structure in a plain-text editor.
To create the JSON file

1. At a terminal (Linux, macOS, or Unix) or command prompt (Windows), create a new text file in a local directory.

2. Open the file in a plain-text editor and edit the values to reflect the structure you want to create. At a minimum, you must change the name of the pipeline. You should also consider whether you want to change:
   - The S3 bucket where artifacts for this pipeline are stored.
   - The source location for your code.
   - The deployment provider.
   - How you want your code deployed.
   - The tags for your pipeline.

The following two-stage sample pipeline structure highlights the values you should consider changing for your pipeline. Your pipeline likely contains more than two stages:

```json
{
    "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"
                        }
                    }
                ]
            }
        ]
    }
}
```
This example adds tagging to the pipeline by including the `Project` tag key and `ProjectA` value on the pipeline. For more information about tagging resources in CodePipeline, see Tagging resources (p. 168).

Make sure the `PollForSourceChanges` parameter in your JSON file is set as follows:

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

CodePipeline uses Amazon CloudWatch Events to detect changes in your CodeCommit source repository and branch or your S3 source bucket. CodePipeline uses webhooks to detect changes in your GitHub source repository and branch. The next step includes instructions to manually create these resources for your pipeline. Setting the flag to `false` disables periodic checks, which are not necessary when you are using the recommended change detection methods.

3. To create a build, test, or deploy action in a Region different from your pipeline, you must add the following to your pipeline structure. For instructions, see Add a cross-Region action in CodePipeline (p. 364).

- Add the `Region` parameter to your action's pipeline structure.
- Use the `artifactStores` parameter to specify an artifact bucket for each AWS Region where you have an action.

4. When you are satisfied with its structure, save your file with a name like `pipeline.json`.

To create a pipeline

1. Run the `create-pipeline` command and use the `--cli-input-json` parameter to specify the JSON file you created previously.
To create a pipeline named *MySecondPipeline* with a JSON file named `pipeline.json` that includes the name "*MySecondPipeline" as the value for `name` in the JSON, your command would look like the following:

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

**Important**

Be sure to include `file://` before the file name. It is required in this command.

This command returns the structure of the entire pipeline you created.

2. To view the pipeline, either open the CodePipeline console and choose it from the list of pipelines, or use the `get-pipeline-state` command. For more information, see View pipeline details and history in CodePipeline (p. 237).

3. If you use the CLI to create a pipeline, you must manually create the recommended change detection resources for your pipeline:

   - For a pipeline with a CodeCommit repository, you must manually create the CloudWatch Events rule, as described in Create a CloudWatch Events rule for a CodeCommit source (CLI) (p. 177).
   - For a pipeline with an Amazon S3 source, you must manually create the CloudWatch Events rule and AWS CloudTrail trail, as described in Use CloudWatch Events to start a pipeline (Amazon S3 source) (p. 185).
   - For a pipeline with a GitHub source, you must manually create the webhook, as described in Use webhooks to start a pipeline (GitHub source) (p. 199).

## Edit a pipeline in CodePipeline

A pipeline describes the release process that you want AWS CodePipeline to follow, including stages and actions that must be completed. You can edit a pipeline to add or remove these elements. However, when you edit a pipeline, values such as the pipeline name or pipeline metadata cannot be changed.

Unlike creating a pipeline, editing a pipeline does not rerun the most recent revision through the pipeline. If you want to run the most recent revision through a pipeline you've just edited, you must manually rerun it. Otherwise, the edited pipeline runs the next time you make a change to a source location configured in the source stage. For information, see Start a pipeline manually in AWS CodePipeline (p. 215).

You can add actions to your pipeline that are in an AWS Region different from your pipeline. When an AWS service is the provider for an action, and this action type/provider type are in a different AWS Region from your pipeline, this is a cross-Region action. For more information about cross-Region actions, see Add a cross-Region action in CodePipeline (p. 364).

CodePipeline uses change detection methods to start your pipeline when a source code change is pushed. These detection methods are based on source type:

- CodePipeline uses Amazon CloudWatch Events to detect changes in your CodeCommit source repository or your Amazon S3 source bucket.
- CodePipeline uses webhooks to detect changes in your GitHub source repository and branch.

**Note**

Change detection resources are created automatically when you use the console. When you use the console to create or edit a pipeline, the additional resources are created for you. If you use the AWS CLI to create the pipeline, you must create the additional resources yourself. For more information about creating or updating a CodeCommit pipeline, see Create a CloudWatch Events rule for a CodeCommit source (CLI) (p. 177). For more information about using the CLI
to create or update an Amazon S3 pipeline, see Create a CloudWatch Events rule for an Amazon S3 source (CLI) (p. 187). For more information about creating or updating a GitHub pipeline, see Use webhooks to start a pipeline (GitHub source) (p. 199).

Topics
• Edit a pipeline (console) (p. 232)
• Edit a pipeline (AWS CLI) (p. 233)

Edit a pipeline (console)

You can use the CodePipeline console to add, edit, or remove stages in a pipeline and to add, edit, or remove actions in a stage.

To edit a pipeline


   The names of all pipelines associated with your AWS account are displayed.
2. In Name, choose the name of the pipeline you want to edit. This opens a detailed view of the pipeline, including the state of each of the actions in each stage of the pipeline.
3. On the pipeline details page, choose Edit.
4. On the Edit page, do one of the following:

   • To edit a stage, choose Edit stage. You can add actions in serial and parallel with existing actions: You can also edit actions in this view by choosing the edit icon for those actions. To delete an action, choose the delete icon on that action.
   • To edit an action, choose the edit icon for that action, and then on Edit action, change the values. Items marked with an asterisk (*) are required.
     • For a CodeCommit repository name and branch, a message appears showing the Amazon CloudWatch Events rule to be created for this pipeline. If you remove the CodeCommit source, a message appears showing the Amazon CloudWatch Events rule to be deleted.
     • For an Amazon S3 source bucket, a message appears showing the Amazon CloudWatch Events rule and AWS CloudTrail trail to be created for this pipeline. If you remove the Amazon S3 source, a message appears showing the Amazon CloudWatch Events rule and AWS CloudTrail trail to be deleted. If the AWS CloudTrail trail is in use by other pipelines, the trail is not removed and the data event is deleted.
     • For a GitHub source, the following are added for the pipeline:
       • CodePipeline uses an OAuth token to create an authorized application that is managed by CodePipeline.
       
       Note
       In GitHub, there is a limit to the number of OAuth tokens you can use for an application, such as CodePipeline. If you exceed this limit, retry the connection to allow CodePipeline to reconnect by reusing existing tokens. For more information, see Pipeline error: I receive a pipeline error that says: "Could not access the GitHub repository" or "Unable to connect to the GitHub repository" (p. 399).
       
       • CodePipeline creates a webhook in GitHub to detect source changes and then start your pipeline when a change occurs. CodePipeline creates the following along with the webhook:
         • A secret is randomly generated and used to authorize the connection to GitHub.
         • The webhook URL is generated using the public endpoint for the Region.
         • The webhook is registered with GitHub. This subscribes the URL to receive repository events.
If you delete a GitHub source action, the webhook is deregistered and deleted for you.

- To add a stage, choose **Add stage** at the point in the pipeline where you want to add a stage. Provide a name for the stage, and then add at least one action to it. Items marked with an asterisk (*) are required.
- To delete a stage, choose the delete icon on that stage. The stage and all of its actions are deleted.

For example, if you wanted to add a serial action to a stage in a pipeline:

1. In the stage where you want to add your action, choose **Edit stage**, and then choose **Add action group**.
2. In **Edit action**, in **Action name**, enter the name of your action. The **Action provider** list displays provider options by category. Look for the category (for example, Deploy). Under the category, choose the provider (for example, **AWS CodeDeploy**). In **Region**, choose the AWS Region where the resource is created or where you plan to create it. The **Region** field designates where the AWS resources are created for this action type and provider type. This field only displays for actions where the action provider is an AWS service. The **Region** field defaults to the same AWS Region as your pipeline.

For more information about the requirements for actions in CodePipeline, including names for input and output artifacts and how they are used, see Action structure requirements in CodePipeline (p. 459). For examples of adding action providers and using the default fields for each provider, see Create a pipeline (console) (p. 222).

To add CodeBuild as a build action or test action to a stage, see Use CodePipeline with CodeBuild to Test Code and Run Builds in the **CodeBuild User Guide**.

**Note**

Some action providers, such as GitHub, require you to connect to the provider’s website before you can complete the configuration of the action. When you connect to a provider’s website, make sure you use the credentials for that website. Do not use your AWS credentials.

3. When you have finished configuring your action, choose **Save**.

**Note**

You cannot rename a stage in the console view. You can add a stage with the name you want to change, and then delete the old one. Make sure you have added all the actions you want in that stage before you delete the old one.

5. When you have finished editing your pipeline, choose **Save** to return to the summary page.

**Important**

After you save your changes, you cannot undo them. You must edit the pipeline again. If a revision is running through your pipeline when you save your changes, the run is not completed. If you want a specific commit or change to run through the edited pipeline, you must manually run it through the pipeline. Otherwise, the next commit or change runs automatically through the pipeline.

6. To test your action, choose **Release change** to process that commit through the pipeline and commit a change to the source specified in the source stage of the pipeline. Or follow the steps in Start a pipeline manually in AWS CodePipeline (p. 215) to use the AWS CLI to manually release a change.

**Edit a pipeline (AWS CLI)**

You can use the **update-pipeline** command to edit a pipeline.
Important
Although you can use the AWS CLI to edit pipelines that include partner actions, you must not manually edit the JSON of a partner action. If you do so, the partner action fails after you update the pipeline.

To edit a pipeline

1. Open a terminal session (Linux, macOS, or Unix) or command prompt (Windows) and run the `get-pipeline` command to copy the pipeline structure into a JSON file. For example, for a pipeline named `MyFirstPipeline`, enter the following command:

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

   This command returns nothing, but the file you created should appear in the directory where you ran the command.

2. Open the JSON file in any plain-text editor and modify the structure of the file to reflect the changes you want to make to the pipeline. For example, you can add or remove stages, or add another action to an existing stage.

   The following example shows how you would add another deployment stage in the pipeline.json file. This stage runs after the first deployment stage named `Staging`.

   **Note**
   This is just a portion of the file, not the entire structure. For more information, see [CodePipeline pipeline structure reference](p. 454).

   ```json
   {
     "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"
           }
         ]
       }
     ]
   }
   ```
The following example shows how you would add a source stage that uses a GitHub repository as its source action. For more information about how CodePipeline integrates with GitHub, see Source action integrations (p. 22).

**Note**
This is just a portion of the file, not the entire structure. For more information, see CodePipeline pipeline structure reference (p. 454).

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

The value for OAuthToken remains masked because CodePipeline uses it to access the GitHub repository. You can use a personal access token for this value. To create a personal access token, see Configure your pipeline to use a personal access token (GitHub and CLI) (p. 408).

**Note**
Some edits, such as moving an action from one stage to another stage, delete the last known state history for the action. If a pipeline contains one or more secret parameters,
such as an OAuth token for an action, that token is masked by a series of asterisks (****). These secret parameters are left unchanged unless you edit that portion of the pipeline (for example, if you change the name of the action that includes the OAuth token or the name of the stage that contains an action that uses an OAuth token). If you make a change that affects an action that includes an OAuth token, you must include the value of the token in the edited JSON. For more information, see Configure your pipeline to use a personal access token (GitHub and CLI) (p. 408). It is a security best practice to rotate your personal access token on a regular basis. For more information, see Use GitHub and the CodePipeline CLI to create and rotate your GitHub personal access token on a regular basis (p. 410).

For information about using the CLI to add an approval action to a pipeline, see Add a manual approval action to a pipeline in CodePipeline (p. 359).

Make sure the `PollForSourceChanges` parameter in your JSON file is set as follows:

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

CodePipeline uses Amazon CloudWatch Events to detect changes in your CodeCommit source repository and branch or your Amazon S3 source bucket. CodePipeline uses webhooks to detect changes in your GitHub source repository and branch. The next step includes instructions for creating these resources manually. Setting the flag to `false` disables periodic checks, which are not required when you use the recommended change detection methods.

3. To add a build, test, or deploy action in a Region different from your pipeline, you must add the following to your pipeline structure. For detailed instructions, see Add a cross-Region action in CodePipeline (p. 364).

- Add the `Region` parameter to your action's pipeline structure.
- Use the `artifactStores` parameter to specify an artifact bucket for each Region where you have an action.

4. If you are working with the pipeline structure retrieved using the `get-pipeline` command, you must modify the structure in the JSON file. You must remove the `metadata` lines from the file so the `update-pipeline` command can use it. Remove the section from the pipeline structure in the JSON file (the "metadata": { } lines and the "created", "pipelineARN", and "updated" fields).

For example, remove the following lines from the structure:

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

Save the file.

5. If you use the CLI to edit a pipeline, you must manually manage the recommended change detection resources for your pipeline:

- For a CodeCommit repository, you must create the CloudWatch Events rule, as described in Create a CloudWatch Events rule for a CodeCommit source (CLI) (p. 177).
- For an Amazon S3 source, you must create the CloudWatch Events rule and AWS CloudTrail trail, as described in Use CloudWatch Events to start a pipeline (Amazon S3 source) (p. 185).
- For a GitHub source, you must create the webhook, as described in Use webhooks to start a pipeline (GitHub source) (p. 199).

6. To apply your changes, run the `update-pipeline` command, specifying the pipeline JSON file:

   **Important**
   
   Be sure to include `file://` before the file name. It is required in this command.
This command returns the entire structure of the edited pipeline.

**Note**
The `update-pipeline` command stops the pipeline. If a revision is being run through the pipeline when you run the `update-pipeline` command, that run is stopped. You must start the pipeline manually to run that revision through the updated pipeline.

7. Open the CodePipeline console and choose the pipeline you just edited.

The pipeline shows your changes. The next time you make a change to the source location, the pipeline runs that revision through the revised structure of the pipeline.

8. To manually run the last revision through the revised structure of the pipeline, run the `start-pipeline-execution` command. For more information, see Start a pipeline manually in AWS CodePipeline (p. 215).

For more information about the structure of a pipeline and expected values, see CodePipeline pipeline structure reference (p. 454) and AWS CodePipeline API Reference.

### View pipeline details and history in CodePipeline

You can use the AWS CodePipeline console or the AWS CLI to view details about pipelines associated with your AWS account.

**Topics**
- View pipeline details and history (console) (p. 237)
- View pipeline details and history (CLI) (p. 244)

### View pipeline details and history (console)

You can use the CodePipeline console to view a list of all of the pipelines in your account. You can also view details for each pipeline, including when actions last ran in the pipeline, whether a transition between stages is enabled or disabled, whether any actions have failed, and other information. You can also view a history page that shows details for all pipeline executions for which history has been recorded. Execution history is retained for up to 12 months.

**Note**
Detailed execution history is available for executions run on or after February 21, 2019.

**Topics**
- View pipeline (console) (p. 237)
- View pipeline execution history (console) (p. 239)
- View execution status (console) (p. 240)
- View pipeline execution source revisions (console) (p. 241)
- View action executions (console) (p. 243)
- View action artifacts and artifact store information (console) (p. 243)

### View pipeline (console)

You can view status, transitions, and artifact updates for a pipeline.
Note
After an hour, the detailed view of a pipeline stops refreshing automatically in your browser. To view current information, refresh the page.

To view a pipeline


The names and creation date of all pipelines associated with your AWS account are displayed, along with links to view execution history.

2. To see details for a single pipeline, in Name, choose the pipeline. You can also select the pipeline, and then choose View pipeline. A detailed view of the pipeline, including the state of each action in each stage and the state of the transitions, is displayed.

The graphical view displays the following information for each stage:

- The stage name.
- Every action configured for the stage.
- The state of transitions between stages (enabled or disabled), as indicated by the state of the arrow between stages. An enabled transition is indicated by an arrow with a Disable transition button next to it. A disabled transition is indicated by an arrow with a strikeout under it and an Enable transition button next to it.
- A color bar to indicate the status of the stage:
  - Gray: No executions yet
  - Blue: In progress
  - Green: Succeeded
  - Red: Failed

The graphical view also displays the following information about actions in each stage:

- The name of the action.
- The provider of the action, such as CodeDeploy.
- When the action was last run.
• Whether the action succeeded or failed.
• Links to other details about the last run of the action, where available.
• Details about the source revisions that are running through the latest pipeline execution in the
  stage or, for CodeDeploy deployments, the latest source revisions that were deployed to target
  instances.

3. To see the configuration details for an action in a stage of a pipeline, choose the information icon
   next to the action.

4. To view the details of the provider of the action, choose the provider. For example, in the preceding
   example pipeline, if you choose CodeDeploy in either the Staging or Production stages the
   CodeDeploy console page for the deployment group configured for that stage is displayed.

5. To see the progress details for an action in a stage, choose Details when it is displayed next to an
   action in progress (indicated by an In Progress message). If the action is in progress, you see the
   incremental progress and the steps or actions as they occur.

   Note
   Details are available for source actions that retrieve content from GitHub repositories, but
   not those that retrieve content from Amazon S3 buckets or CodeCommit repositories.

6. To approve or reject actions that have been configured for manual approval, choose Review.

7. To retry actions in a stage that were not completed successfully, choose Retry.

8. To get more information about errors or failures for a completed action in a stage, choose Details.
   Details from the last time the action ran, including the results of that action (Succeeded or Failed)
   are displayed.

9. To view details about source artifacts (output artifact that originated in the first stage of a pipeline)
   that are used the latest pipeline execution for a stage, click in the details information area at the
   bottom of the stage. You can view details about identifiers, such as commit IDs, check-in comments,
   and the time since the artifact was created or updated.

10. To view details about the most recent executions for the pipeline, choose View history. For past
    executions, you can view revision details associated with source artifacts, such as execution IDs,
    status, start and end times, duration, and commit IDs and messages.

**View pipeline execution history (console)**

You can use the console to view the history of executions in a pipeline, including status, source revisions,
and timing details for each execution.

1. Sign in to the AWS Management Console and open the CodePipeline console at http://
   console.aws.amazon.com/codesuite/codepipeline/home.

   The names of all pipelines associated with your AWS account are displayed, along with their status.

2. In Name, choose the name of the pipeline.

3. Choose View history.

4. View the status, source revisions, change details, and triggers related to each execution for your
   pipeline.
5. Choose an execution. The detail view shows execution details, the Timeline tab, and the Visualization tab.

### View execution status (console)

You can view the pipeline status in **Status** on the execution history page. Choose an execution ID link, and then view the action status.

The following are valid states for pipelines, stages, and actions:

#### Pipeline-level states

<table>
<thead>
<tr>
<th>Pipeline state</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InProgress</td>
<td>The pipeline execution is currently running.</td>
</tr>
<tr>
<td>Stopping</td>
<td>The pipeline execution is stopping due to a request to either stop and wait or stop and abandon the pipeline execution.</td>
</tr>
<tr>
<td>Pipeline state</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Stopped</td>
<td>The stopping process is complete, and the pipeline execution is stopped.</td>
</tr>
<tr>
<td>Succeeded</td>
<td>The pipeline execution was completed successfully.</td>
</tr>
<tr>
<td>Superseded</td>
<td>While this pipeline execution was waiting for the next stage to be completed, a newer pipeline execution advanced and continued through the pipeline instead.</td>
</tr>
<tr>
<td>Failed</td>
<td>The pipeline execution was not completed successfully.</td>
</tr>
</tbody>
</table>

### Stage-level states

<table>
<thead>
<tr>
<th>Stage state</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InProgress</td>
<td>The stage is currently running.</td>
</tr>
<tr>
<td>Stopping</td>
<td>The stage execution is stopping due to a request to either stop and wait or stop and abandon the pipeline execution.</td>
</tr>
<tr>
<td>Stopped</td>
<td>The stopping process is complete, and the stage execution is stopped.</td>
</tr>
<tr>
<td>Succeeded</td>
<td>The stage was completed successfully.</td>
</tr>
<tr>
<td>Failed</td>
<td>The stage was not completed successfully.</td>
</tr>
</tbody>
</table>

### Action-level states

<table>
<thead>
<tr>
<th>Action state</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InProgress</td>
<td>The action is currently running.</td>
</tr>
<tr>
<td>Abandoned</td>
<td>The action is abandoned due to a request to stop and abandon the pipeline execution.</td>
</tr>
<tr>
<td>Succeeded</td>
<td>The action was completed successfully.</td>
</tr>
<tr>
<td>Failed</td>
<td>For approval actions, the FAILED state means the action was either rejected by the reviewer or failed due to an incorrect action configuration.</td>
</tr>
</tbody>
</table>

### View pipeline execution source revisions (console)

You can view details about source artifacts (output artifact that originated in the first stage of a pipeline) that are used in an execution of a pipeline. The details include identifiers, such as commit IDs, check-in comments, and, when you use the CLI, version numbers of pipeline build actions. For some revision types, you can view and open the URL of the commit. Source revisions are made up of the following:

- **Summary**: Summary information about the most recent revision of the artifact. For GitHub and AWS CodeCommit repositories, the commit message. For Amazon S3 buckets or actions, the user-provided content of a codepipeline-artifact-revision-summary key specified in the object metadata.
- **revisionUrl**: The revision URL for the artifact revision (for example, the external repository URL).
- **revisionId**: The revision ID for the artifact revision. For example, for a source change in a CodeCommit or GitHub repository, this is the commit ID. For artifacts stored in GitHub or CodeCommit repositories, the commit ID is linked to a commit details page.

   The names of all pipelines associated with your AWS account will be displayed.

2. Choose the name of the pipeline for which you want to view source revision details, Do one of the following:

   • Choose View history. In Source revisions, the source change for each execution is listed.

     ![Execution History Table]

   • Locate an action for which you want to view source revision details, and then find the revision information at the bottom of its stage:

     ![Source Details]

     Choose View current revisions to view source information. With the exception of artifacts stored in Amazon S3 buckets, identifiers such as commit IDs in this information detail view are linked to source information pages for the artifacts.
View action executions (console)

You can view action details for a pipeline, such as action execution ID, input artifacts, output artifacts, and status. You can view action details by choosing a pipeline in the console and then choosing an execution ID.

**Note**
Detailed execution history is available for executions run on or after February 21, 2019.

   The names of all pipelines associated with your AWS account are displayed.
2. Choose the name of the pipeline for which you want to view action details, and then choose View history.
3. In Execution ID, choose the execution ID for which you want to view action execution details.
4. You can view the following information on the Timeline tab:
   a. In Action name, choose the link to open a details page for the action where you can view status, stage name, action name, configuration data, and artifact information.
   b. In Provider, choose the link to view the action provider details. For example, in the preceding example pipeline, if you choose CodeDeploy in either the Staging or Production stages, the CodeDeploy console page for the CodeDeploy application configured for that stage is displayed.

View action artifacts and artifact store information (console)

You can view input and output artifact details for an action. You can also choose a link that takes you to the artifact information for that action. Because the artifact store uses versioning, each action execution has a unique input and output artifact location.

   The names of all pipelines associated with your AWS account are displayed.
2. Choose the name of the pipeline for which you want to view action details, and then choose View history.
3. In Execution ID, choose the execution ID for which you want to view action details.
4. On the Timeline tab, in Action name, choose the link to open a details page for the action.
5. On the details page, in Execution summary, view the status and timing of the action execution.
6. In **Action details**, view the action provider and AWS Region where the execution runs. In **Action configuration**, view the resource configuration for the action (for example, the CodeBuild build project name).

7. In **Artifacts**, view the artifact details in **Artifact type** and **Artifact provider**. Choose the link under **Artifact name** to view the artifacts in the artifact store.

---

**View pipeline details and history (CLI)**

You can run the following commands to view details about your pipelines and pipeline executions:

- **list-pipelines** command to view a summary of all of the pipelines associated with your AWS account.
- **get-pipeline** command to review details of a single pipeline.
- **list-pipeline-executions** to view summaries of the most recent executions for a pipeline.
- **get-pipeline-execution** to view information about an execution of a pipeline, including details about artifacts, the pipeline execution ID, and the name, version, and status of the pipeline.
- **get-pipeline-state** command to view pipeline, stage, and action status.
- **list-action-executions** to view action execution details for a pipeline.

**Topics**

- View pipeline (CLI) (p. 245)
- View execution history (CLI) (p. 245)
- View execution status (CLI) (p. 246)
- View source revisions (CLI) (p. 247)
- View action executions (CLI) (p. 249)
View pipeline (CLI)

You can view pipeline details.

1. Open a terminal (Linux, macOS, or Unix) or command prompt (Windows) and use the AWS CLI to run the `list-pipelines` command:

   ```bash
   aws codepipeline list-pipelines
   ```

   This command returns a list of all of the pipelines associated with your AWS account.

2. To view details about a pipeline, run the `get-pipeline` command, specifying the unique name of the pipeline. For example, to view details about a pipeline named `MyFirstPipeline`, enter the following:

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

   This command returns the structure of the pipeline.

View execution history (CLI)

You can view pipeline execution history.

- To view details about past executions of a pipeline, run the `list-pipeline-executions` command, specifying the unique name of the pipeline. For example, to view details about the current state of a pipeline named `MyFirstPipeline`, enter the following:

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

  This command returns summary information about all pipeline executions for which history has been recorded. The summary includes start and end times, duration, and status.

The following example shows the returned data for a pipeline named `MyFirstPipeline` that has had three executions:

```json
{
    "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"
        }
    ]
}
```
To view more details about a pipeline execution, run the `get-pipeline-execution` command, specifying the unique ID of the pipeline execution. For example, to view more details about the first execution in the previous example, enter the following:

```
aws codepipeline get-pipeline-execution --pipeline-name MyFirstPipeline --pipeline-execution-id 7cf7f7cb-3137-539g-j458-d7eu3EXAMPLE
```

This command returns summary information about an execution of a pipeline, including details about artifacts, the pipeline execution ID, and the name, version, and status of the pipeline.

The following example shows the returned data for a pipeline named `MyFirstPipeline`:

```
{
    "pipelineExecution": {
        "pipelineExecutionId": "3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE",
        "pipelineVersion": 2,
        "pipelineName": "MyFirstPipeline",
        "status": "Succeeded",
        "artifactRevisions": [
            {
                "created": 1496380678.648,
                "revisionChangeIdentifier": "1496380258.243",
                "revisionId": "7636d59f3c461cEXAMPLE8417dbc6371",
                "name": "MyApp",
                "revisionSummary": "Updating the application for feature 12-4820"
            }
        ]
    }
}
```

**View execution status (CLI)**

You can view pipeline, stage, and action status.

- To view details about the current state of a pipeline, run the `get-pipeline-state` command, specifying the unique name of the pipeline. For example, to view details about the current state of a pipeline named `MyFirstPipeline`, enter the following:

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

This command returns the current status of all stages of the pipeline and the status of the actions in those stages.

The following example shows the returned data for a three-stage pipeline named `MyFirstPipeline`, where the first two stages and actions show success, the third shows failure, and the transition between the second and third stages is disabled:

```
{
    "updated": 1427245911.525,
    "created": 1427245911.525,
    "pipelineVersion": 1,
    "pipelineName": "MyFirstPipeline",
    "stageStates": [
        {
            "actionStates": [
                {
                    "actionName": "Source",
                
```
View source revisions (CLI)

You can view details about source artifacts (output artifacts that originated in the first stage of a pipeline) that are used in an execution of a pipeline. The details include identifiers, such as commit IDs, check-in comments, time since the artifact was created or updated and, when you use the CLI, version numbers of build actions. For some revision types, you can view and open the URL of the commit for the artifact version. Source revisions are made up of the following:
• **Summary:** Summary information about the most recent revision of the artifact. For GitHub and AWS CodeCommit repositories, the commit message. For Amazon S3 buckets or actions, the user-provided content of a codepipeline-artifact-revision-summary key specified in the object metadata.

• **revisionUrl:** The commit ID for the artifact revision. For artifacts stored in GitHub or AWS CodeCommit repositories, the commit ID is linked to a commit details page.

You can run the `get-pipeline-execution` command to view information about the most recent source revisions that were included in a pipeline execution. After you first run the `get-pipeline-state` command to get details about all stages in a pipeline, you identify the execution ID that applies to a stage for which you want source revision details. Then you use the execution ID in the `get-pipeline-execution` command. (Because stages in a pipeline might have been last successfully completed during different pipeline runs, they can have different execution IDs.)

In other words, if you want to view details about artifacts currently in the Staging stage, run the `get-pipeline-state` command, identify the current execution ID of the Staging stage, and then run the `get-pipeline-execution` command using that execution ID.

To view source revisions in a pipeline

1. Open a terminal (Linux, macOS, or Unix) or command prompt (Windows) and use the AWS CLI to run the `get-pipeline-state` command. For a pipeline named `MyFirstPipeline`, you would enter:

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

   This command returns the most recent state of a pipeline, including the latest pipeline execution ID for each stage.

2. To view details about a pipeline execution, run the `get-pipeline-execution` command, specifying the unique name of the pipeline and the pipeline execution ID of the execution for which you want to view artifact details. For example, to view details about the execution of a pipeline named `MyFirstPipeline`, with the execution ID `3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE`, you would enter the following:

   ```bash
   aws codepipeline get-pipeline-execution --pipeline-name MyFirstPipeline --pipeline-execution-id 3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE
   ```

   This command returns information about each source revision that is part of the pipeline execution and identifying information about the pipeline. Only information about pipeline stages that were included in that execution are included. There might be other stages in the pipeline that were not part of that pipeline execution.

   The following example shows the returned data for a portion of pipeline named `MyFirstPipeline`, where an artifact named "MyApp" is stored in a GitHub repository:

3. ```json
   {
   "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",
   ```
View action executions (CLI)

You can view action execution details for a pipeline, such as action execution ID, input artifacts, output artifacts, execution result, and status. You provide the Execution ID filter to return a listing of actions in a pipeline execution:

**Note**

Detailed execution history is available for executions run on or after February 21, 2019.

- To view action executions for a pipeline, do one of the following:
  - To view details for all action executions in a pipeline, run the `list-action-executions` command, specifying the unique name of the pipeline. For example, to view action executions in a pipeline named `MyFirstPipeline`, enter the following:

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

The following shows a portion of sample output for this command:

```json
{
   "actionExecutionDetails": [
      {
         "actionExecutionId": "ID",
         "lastUpdateTime": 1552958312.034,
         "startTime": 1552958246.542,
         "pipelineExecutionId": "Execution_ID",
         "actionName": "Build",
         "status": "Failed",
         "output": {
            "executionResult": {
               "externalExecutionUrl": "Project_ID",
               "externalExecutionSummary": "Build terminated with state: FAILED",
               "externalExecutionId": "ID"
            },
            "outputArtifacts": []
         },
         "stageName": "Beta",
         "pipelineVersion": 8,
         "input": {
            "configuration": {
               "ProjectName": "java-project"
            },
            "region": "us-east-1",
            "inputArtifacts": [
               {
                  "s3location": {
                     "bucket": "codepipeline-us-east-1-ID",
                     "key": "MyFirstPipeline/MyApp/Object.zip"
                  },
                  "name": "MyApp"
               }
            ],
            "actionTypeId": {
               "version": "1",
```
To view all action executions in a pipeline execution, run the `list-action-executions` command, specifying the unique name of the pipeline and the execution ID. For example, to view action executions for an `Execution_ID`, enter the following:

```
aws codepipeline list-action-executions --pipeline-name MyFirstPipeline --filter pipelineExecutionId=Execution_ID
```

The following shows a portion of sample output for this command:

```
{
   "actionExecutionDetails": [ {
      "stageName": "Beta",
      "pipelineVersion": 8,
      "actionName": "Build",
      "status": "Failed",
      "lastUpdateTime": 1552958312.034,
      "input": {
         "configuration": {
            "ProjectName": "java-project"
         },
         "region": "us-east-1",
         "actionTypeId": {
            "owner": "AWS",
            "category": "Build",
            "provider": "CodeBuild",
            "version": "1"
         },
         "inputArtifacts": [ {
            "s3location": { "bucket": "codepipeline-us-east-1-ID",
                             "key": "MyFirstPipeline/MyApp/Object.zip" }
         } ],
         "name": "MyApp"
      }
   },
   ...
}
```

Delete a pipeline in CodePipeline

You can always edit a pipeline to change its functionality, but you might decide you want to delete it instead. You can use the AWS CodePipeline console or the `delete-pipeline` command in the AWS CLI to delete a pipeline.

**Topics**

- Delete a pipeline (console) (p. 251)
- Delete a pipeline (CLI) (p. 251)
Delete a pipeline (console)

To delete a pipeline

1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home. The names and status of all pipelines associated with your AWS account are displayed.
2. In Name, choose the name of the pipeline you want to delete.
3. On the pipeline details page, choose Edit.
4. On the Edit page, choose Delete.
5. Type delete in the field to confirm, and then choose Delete.

Important
This action cannot be undone.

Delete a pipeline (CLI)

To use the AWS CLI to manually delete a pipeline, use the delete-pipeline command.

Important
Deleting a pipeline is irreversible. There is no confirmation dialog box. After the command is run, the pipeline is deleted, but none of the resources used in the pipeline are deleted. This makes it easier to create a new pipeline that uses those resources to automate the release of your software.

To delete a pipeline

1. Open a terminal (Linux, macOS, or Unix) or command prompt (Windows) and use the AWS CLI to run the delete-pipeline command, specifying the name of the pipeline you want to delete. For example, to delete a pipeline named MyFirstPipeline:

   ```
   aws codepipeline delete-pipeline --name MyFirstPipeline
   ```

   This command returns nothing.
2. Delete any resources you no longer need.

Note
Deleting a pipeline does not delete the resources used in the pipeline, such as the CodeDeploy or Elastic Beanstalk application you used to deploy your code, or, if you created your pipeline from the CodePipeline console, the Amazon S3 bucket CodePipeline created to store the artifacts of your pipelines. Make sure that you delete resources that are no longer required so that you are not charged for them in the future. For example, when you use the console to create a pipeline for the first time, CodePipeline creates one Amazon S3 bucket to store all artifacts for all of your pipelines. If you have deleted all of your pipelines, follow the steps in Deleting a Bucket.

Create a pipeline in CodePipeline that uses resources from another AWS account

You might want to create a pipeline that uses resources created or managed by another AWS account. For example, you might want to use one account for your pipeline and another for your CodeDeploy resources.
Note
When you create a pipeline with actions from multiple accounts, you must configure your actions so that they can still access artifacts within the limitations of cross-account pipelines.
The following limitations apply to cross-account actions:

- In general, an action can only consume an artifact if:
  - The action is in the same account as the pipeline account OR
  - The artifact was created in the pipeline account for an action in another account OR
  - The artifact was produced by a previous action in the same account as the action

In other words, you cannot pass an artifact from one account to another if neither account is the pipeline account.

- Cross-account actions are not supported for the following action types:
  - Jenkins build actions

For this example, you must create an AWS Key Management Service (AWS KMS) key to use, add the key to the pipeline, and set up account policies and roles to enable cross-account access. For an AWS KMS key, you can use the key ID, the key ARN, or the alias ARN.

**Note**
To specify a customer master key (CMK) in a different AWS account, you must use the key ARN or alias ARN.

In this walkthrough and its examples, AccountA is the account originally used to create the pipeline. It has access to the Amazon S3 bucket used to store pipeline artifacts and the service role used by AWS CodePipeline. AccountB is the account originally used to create the CodeDeploy application, deployment group, and service role used by CodeDeploy.

For AccountA to edit a pipeline to use the CodeDeploy application created by AccountB, AccountA must:

- Request the ARN or account ID of AccountB (in this walkthrough, the AccountB ID is 012ID_ACCOUNT_B).
- Create or use an AWS KMS customer managed key in the Region for the pipeline, and grant permissions to use that key to the service role (AWS-CodePipeline-Service) and AccountB.
- Create an Amazon S3 bucket policy that grants AccountB access to the Amazon S3 bucket (for example, codepipeline-us-east-2-1234567890).
- Create a policy that allows AccountA to assume a role configured by AccountB, and attach that policy to the service role (AWS-CodePipeline-Service).
- Edit the pipeline to use the customer managed AWS KMS key instead of the default key.

For AccountB to allow access to its resources to a pipeline created in AccountA, AccountB must:

- Request the ARN or account ID of AccountA (in this walkthrough, the AccountA ID is 012ID_ACCOUNT_A).
- Create a policy applied to the Amazon EC2 instance role configured for CodeDeploy that allows access to the Amazon S3 bucket (codepipeline-us-east-2-1234567890).
- Create a policy applied to the Amazon EC2 instance role configured for CodeDeploy that allows access to the AWS KMS customer managed key used to encrypt the pipeline artifacts in AccountA.
- Configure and attach an IAM role (CrossAccount_Role) with a trust relationship policy that allows AccountA to assume the role.
- Create a policy that allows access to the deployment resources required by the pipeline and attach it to CrossAccount_Role.
Create a policy that allows access to the Amazon S3 bucket (codepipeline-us-east-2-1234567890) and attach it to CrossAccount_Role.

### Topics
- Prerequisite: Create an AWS KMS encryption key (p. 253)
- Step 1: Set up account policies and roles (p. 253)
- Step 2: Edit the pipeline (p. 258)

### Prerequisite: Create an AWS KMS encryption key

Customer-managed keys are specific to a Region, as are all AWS KMS keys. You must create your customer managed AWS KMS key in the same Region where the pipeline was created (for example, us-east-2).

**To create a customer managed key in AWS KMS**

1. Sign in to the AWS Management Console with `AccountA` and open the AWS KMS console.
2. On the left, choose Customer managed keys.
3. Choose Create key. In Configure key, leave the Symmetric default selected and choose Next.
4. In Alias, enter an alias to use for this key (for example, PipelineName-Key). Optionally, provide a description and tags for this key, and then choose Next.
5. In Define Key Administrative Permissions, choose your IAM user and any other users or groups you want to act as administrators for this key, and then choose Next.
6. In Define Key Usage Permissions, under This Account, select the name of the service role for the pipeline (for example, AWS-CodePipeline-Service). Under Other AWS accounts, choose Add another AWS account. Enter the account ID for `AccountB` to complete the ARN, and then choose Next.
7. In Review and edit key policy, review the policy, and then choose Finish.
8. From the list of keys, choose the alias of your key and copy its ARN (for example, arn:aws:kms:us-east-2:01234567890:alias/PipelineName-Key). You will need this when you edit your pipeline and configure policies.

### Step 1: Set up account policies and roles

After you create the AWS KMS key, you must create and attach policies that will enable the cross-account access. This requires actions from both `AccountA` and `AccountB`.

**Topics**
- Configure policies and roles in the account that will create the pipeline (AccountA) (p. 253)
- Configure policies and roles in the account that owns the AWS resource (AccountB) (p. 255)

### Configure policies and roles in the account that will create the pipeline (`AccountA`)

To create a pipeline that uses CodeDeploy resources associated with another AWS account, `AccountA` must configure policies for both the Amazon S3 bucket used to store artifacts and the service role for CodePipeline.
To create a policy for the Amazon S3 bucket that grants access to AccountB (console)

1. Sign in to the AWS Management Console with AccountA and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. In the list of Amazon S3 buckets, choose the Amazon S3 bucket where artifacts for your pipelines are stored. This bucket is named codepipeline-region-1234567890EXAMPLE, where region is the AWS Region in which you created the pipeline and 1234567890EXAMPLE is a ten-digit random number that ensures the bucket name is unique (for example, codepipeline-us-east-2-1234567890).
3. On the detail page for the Amazon S3 bucket, choose Properties.
4. In the properties pane, expand Permissions, and then choose Add bucket policy.

**Note**
If a policy is already attached to your Amazon S3 bucket, choose Edit bucket policy. You can then add the statements in the following example to the existing policy. To add a new policy, choose the link, and follow the instructions in the AWS Policy Generator. For more information, see Overview of IAM Policies.

5. In the Bucket Policy Editor window, type the following policy. This will allow AccountB access to the pipeline artifacts, and will give AccountB the ability to add output artifacts if an action, such as a custom source or build action, creates them.

In the following example, the ARN is for AccountB is 012ID_ACCOUNT_B. The ARN for the Amazon S3 bucket is codepipeline-us-east-2-1234567890. Replace these ARNs with the ARN for the account you want to allow access and the ARN for the Amazon S3 bucket:

```json
{
    "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*"
            ]
        }
    ]
}
```
Step 1: Set up account policies and roles

To create a policy for the service role for CodePipeline (console)

1. Sign in to the AWS Management Console with AccountA and open the IAM console at https://console.aws.amazon.com/iam/.
2. In Dashboard, choose Roles.
3. In the list of roles, under Role Name, choose the name of the service role for CodePipeline. By default, this is AWS-CodePipeline-Service. If you used a different name for your service role, be sure to choose it from the list.
4. On the Summary page, on the Permissions tab, expand Inline Policies, and then choose Create Role Policy.
   
   **Note**
   If you have not previously created any role policies, Create Role Policy will not appear. Choose the link to create a new policy instead.
5. In Set Permissions, choose Custom Policy, and then choose Select.
6. On the Review Policy page, type a name for the policy in Policy Name. In Policy Document, type the following policy to allow AccountB to assume the role. In the following example, 012ID_ACCOUNT_B is the ARN for AccountB:

   ```json
   {
     "Version": "2012-10-17",
     "Statement": {
       "Effect": "Allow",
       "Action": "sts:AssumeRole",
       "Resource": [ "arn:aws:iam::012ID_ACCOUNT_B:role/*"
     ]
   }
   }
   
   7. Choose Validate Policy.
7. After the policy is validated, choose Apply Policy.

Configure policies and roles in the account that owns the AWS resource (AccountB)

When you create an application, deployment, and deployment group in CodeDeploy, you also create an Amazon EC2 instance role. (This role is created for you if you use the Run Deployment Walkthrough API Version 2015-07-09

255
wizard, but you can also create it manually.) For a pipeline created in *AccountA* to use CodeDeploy resources created in *AccountB*, you must:

- Configure a policy for the instance role that allows it to access the Amazon S3 bucket where pipeline artifacts are stored.
- Create a second role in *AccountB* configured for cross-account access.

This second role must not only have access to the Amazon S3 bucket in *AccountA*, it must also contain a policy that allows access to the CodeDeploy resources and a trust relationship policy that allows *AccountA* to assume the role.

**Note**
These policies are specific to setting up CodeDeploy resources to be used in a pipeline created using a different AWS account. Other AWS resources will require policies specific to their resource requirements.

**To create a policy for the Amazon EC2 instance role configured for CodeDeploy (console)**

1. Sign in to the AWS Management Console with *AccountB* and open the IAM console at https://console.aws.amazon.com/iam/.
2. In **Dashboard**, choose **Roles**.
3. In the list of roles, under **Role Name**, choose the name of the service role used as the Amazon EC2 instance role for the CodeDeploy application. This role name can vary, and more than one instance role can be used by a deployment group. For more information, see Create an IAM Instance Profile for your Amazon EC2 Instances.
4. On the **Summary** page, on the **Permissions** tab, expand **Inline Policies**, and then choose **Create Role Policy**.
5. In **Set Permissions**, choose **Custom Policy**, and then choose **Select**.
6. On the **Review Policy** page, enter a name for the policy in **Policy Name**. In **Policy Document**, type the following policy to grant access to the Amazon S3 bucket used by *AccountA* to store artifacts for pipelines (in this example, *codepipeline-us-east-2-1234567890*):

```json
{
    "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. Choose **Validate Policy**.
8. After the policy is validated, choose **Apply Policy**.
9. Create a second policy for AWS KMS where `arn:aws:kms:us-east-1:0123456789:alias/my-key` is the ARN of the customer managed key created in `AccountA` and configured to allow `AccountB` to use it:

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "kms:DescribeKey",
            "kms:GenerateDataKey",
            "kms:Encrypt",
            "kms:ReEncrypt",
            "kms:Decrypt"
         ],
         "Resource": [
            "arn:aws:kms:us-east-1:0123456789:alias/my-key"
         ]
      }
   ]
}
```

**Important**

You must use the account ID of `AccountA` in this policy as part of the resource ARN for the AWS KMS key, as shown here, or the policy will not work.

10. Choose **Validate Policy**.

11. After the policy is validated, choose **Apply Policy**.

Now create an IAM role to use for cross-account access, and configure it so that `AccountA` can assume the role. This role must contain policies that allow access to the CodeDeploy resources and the Amazon S3 bucket used to store artifacts in `AccountA`.

**To configure the cross-account role in IAM**

2. In Dashboard, choose Roles, and then choose **Create New Role**.
3. On the **Set New Role** page, type a name for this role in **Role Name** (for example, `CrossAccount_Role`). You can name this role anything you want as long as it follows the naming conventions in IAM. Consider giving the role a name that clearly states its purpose.
4. On the **Select Role Type** page, choose **Role for Cross-Account Access**. Next to **Provide access between AWS accounts you own**, choose **Select**.
5. Type the AWS account ID for the account that will create the pipeline in CodePipeline (`AccountA`), and then choose **Next Step**.

   **Note**
   
   This step creates the trust relationship policy between `AccountB` and `AccountA`.

6. In **Attach Policy**, choose `AmazonS3ReadOnlyAccess`, and then choose **Next Step**.

   **Note**
   
   This is not the policy you will use. You must choose a policy to complete the wizard.

7. On the **Review** page, choose **Create Role**.
8. From the list of roles, choose the policy you just created (for example, `CrossAccount_Role`) to open the **Summary** page for that role.
9. Expand **Permissions**, and then expand **Inline Policies**. Choose the link to create an inline policy.
10. In **Set Permissions**, choose **Custom Policy**, and then choose **Select**.

11. On the **Review Policy** page, type a name for the policy in **Policy Name**. In **Policy Document**, type the following policy to allow access to CodeDeploy resources:

   ```json
   {
   "Version": "2012-10-17",
   "Statement": [
   {
   "Effect": "Allow",
   "Action": [
   "codedeploy:CreateDeployment",
   "codedeploy:GetDeployment",
   "codedeploy:GetDeploymentConfig",
   "codedeploy:GetApplicationRevision",
   "codedeploy:RegisterApplicationRevision"
   ],
   "Resource": "*
   ]
   }
   }
   ```

12. Choose **Validate Policy**.

13. After the policy is validated, choose **Apply Policy**.

14. In **Inline Policies**, choose **Create Role Policy**.

15. In **Set Permissions**, choose **Custom Policy**, and then choose **Select**.

16. On the **Review Policy** page, type a name for the policy in **Policy Name**. In **Policy Document**, type the following policy to allow this role to retrieve input artifacts from, and put output artifacts into, the Amazon S3 bucket in **AccountA**:

   ```json
   {
   "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. Choose **Validate Policy**.

18. When the policy is validated, choose **Apply Policy**.

19. In **Managed Policies**, find **AmazonS3ReadOnlyAccess** in the list of policies under **Policy Name**, and choose **Detach Policy**. When prompted, choose **Detach**.

### Step 2: Edit the pipeline

You cannot use the CodePipeline console to create or edit a pipeline that uses resources associated with another AWS account. However, you can use the console to create the general structure of the pipeline,
and then use the AWS CLI to edit the pipeline and add those resources. Alternatively, you can use the structure of an existing pipeline and manually add the resources to it.

To add the resources associated with another AWS account (AWS CLI)

1. At a terminal (Linux, macOS, or Unix) or command prompt (Windows), run the `get-pipeline` command on the pipeline to which you want to add resources. Copy the command output to a JSON file. For example, for a pipeline named `MyFirstPipeline`, you would type something similar to the following:

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

   The output is sent to the `pipeline.json` file.

2. Open the JSON file in any plain-text editor. After "type": "S3" in the artifact store, add the KMS encryptionKey, ID, and type information where `codepipeline-us-east-2-1234567890` is the name of the Amazon S3 bucket used to store artifacts for the pipeline and `arn:aws:kms:us-east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE` is the ARN of the customer-managed key you just created:

   ```
   {  
   "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. Add a deploy action in a stage to use the CodeDeploy resources associated with `AccountB`, including the `roleArn` values for the cross-account role you created (`CrossAccount_Role`).

   The following example shows JSON that adds a deploy action named `ExternalDeploy`. It uses the CodeDeploy resources created in `AccountB` in a stage named `Staging`. In the following example, the ARN for `AccountB` is `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,
   ```

API Version 2015-07-09

259
Note
This is not the JSON for the entire pipeline, just the structure for the action in a stage.

4. You must remove the `metadata` lines from the file so the `update-pipeline` command can use it. Remove the section from the pipeline structure in the JSON file (the "metadata": {} lines and the "created", "pipelineARN", and "updated" fields).

For example, remove the following lines from the structure:

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

Save the file.

5. To apply your changes, run the `update-pipeline` command, specifying the pipeline JSON file, similar to the following:

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

This command returns the entire structure of the edited pipeline.

To test the pipeline that uses resources associated with another AWS account

1. At a terminal (Linux, macOS, or Unix) or command prompt (Windows), run the `start-pipeline-execution` command, specifying the name of the pipeline, similar to the following:

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

For more information, see Start a pipeline manually in AWS CodePipeline (p. 215).


The names of all pipelines associated with your AWS account are displayed.

3. In Name, choose the name of the pipeline you just edited. This opens a detailed view of the pipeline, including the state of each action in each stage of the pipeline.

4. Watch the progress through the pipeline. Wait for a success message on the action that uses the resource associated with another AWS account.

   Note
   You will receive an error if you try to view details for the action while signed in with `AccountA`. Sign out, and then sign in with `AccountB` to view the deployment details in CodeDeploy.
Edit pipelines to use push events

AWS CodePipeline supports full, end-to-end continuous delivery, which includes starting your pipeline whenever there is a code change. There are two supported ways to start your pipeline upon a code change:

- Events (either Amazon CloudWatch Events or webhooks)
- Polling (checking periodically)

Initially, only polling was supported. Events are now the default and recommended way to start your pipeline when there's a code change.

**Important**

You must explicitly set the `PollForSourceChanges` parameter to `false` within your Source action's configuration to stop a pipeline from polling. As a result, it is possible to erroneously configure a pipeline with both event-based change detection and polling by, for example, configuring a CloudWatch Events rule and also omitting the `PollForSourceChanges` parameter. This results in duplicate pipeline executions, and the pipeline is counted toward the limit on total number of polling pipelines, which by default is much lower than event-based pipelines.

There are some important advantages to using push events instead of polling:

- On average, events are significantly faster. Events should start your pipeline almost immediately, as opposed to polling, which requires waiting for the next periodic check.
- Higher limits. Compared to pipelines that poll for changes, CodePipeline can support far more event-based pipelines.
- Better experience with many pipelines. Some customers might experience throttling or higher costs by having many pipelines continuously polling their repository for code changes. You can avoid this by using events.

When you use the CodePipeline console or AWS CodeStar to create a pipeline, events are enabled by default. For backward compatibility, new pipelines created through the API, AWS CLI, or AWS CloudFormation use the original polling functionality. We strongly recommend that you use events instead. To opt in, use the AWS CLI or AWS CloudFormation to create the CloudWatch event or webhook and disable polling. Use the instructions in the following table.

You should also use events on pipelines that were created before the new console was launched. To opt in, use the CodePipeline console to create the CloudWatch event or webhook and disable polling. Use the instructions in the following table.

**To update AWS CodeCommit source actions**

<table>
<thead>
<tr>
<th>Type</th>
<th>Action Required</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>If you create and manage your pipeline with the AWS CLI</td>
<td>Use the AWS CLI to disable periodic checks and create your Amazon CloudWatch Events resources</td>
<td>Update pipelines for push events (CodeCommit source) (CLI) (p. 267)</td>
</tr>
<tr>
<td>If you create and manage your pipeline with AWS CloudFormation</td>
<td>Use AWS CloudFormation to execute a change set that disables periodic checks and creates your Amazon CloudWatch Events resources</td>
<td>Update pipelines for push events (CodeCommit source) (AWS CloudFormation template) (p. 275)</td>
</tr>
<tr>
<td>Type</td>
<td>Action Required</td>
<td>Instructions</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>If you created your pipeline in the console before October 11, 2017</td>
<td>Use the CodePipeline console to let CodePipeline disable periodic checks and create your Amazon CloudWatch Events resources</td>
<td>Update pipelines for push events (CodeCommit or Amazon S3 source) (console) (p. 263)</td>
</tr>
<tr>
<td>If you created your pipeline in the console after October 11, 2017</td>
<td>No action required</td>
<td></td>
</tr>
</tbody>
</table>

**To update Amazon S3 source actions**

<table>
<thead>
<tr>
<th>Type</th>
<th>Action Required</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>If you create and manage your pipeline with the AWS CLI</td>
<td>Use the AWS CLI to disable periodic checks and create your Amazon CloudWatch Events and CloudTrail resources</td>
<td>Update pipelines for push events (Amazon S3 source) (CLI) (p. 269)</td>
</tr>
<tr>
<td>If you create and manage your pipeline with AWS CloudFormation</td>
<td>Use AWS CloudFormation to execute a change set that disables periodic checks and creates your Amazon CloudWatch Events and AWS CloudTrail resources</td>
<td>Update pipelines for push events (Amazon S3 source) (AWS CloudFormation template) (p. 285)</td>
</tr>
<tr>
<td>If you created your pipeline in the console before March 22, 2018</td>
<td>Use the CodePipeline console to let CodePipeline disable periodic checks and create your Amazon CloudWatch Events and AWS CloudTrail resources</td>
<td>Update pipelines for push events (CodeCommit or Amazon S3 source) (console) (p. 263)</td>
</tr>
<tr>
<td>If you create your pipeline in the console after March 22, 2018</td>
<td>No action required</td>
<td></td>
</tr>
</tbody>
</table>

**To update GitHub source actions**

<table>
<thead>
<tr>
<th>Type</th>
<th>Action Required</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>If you create and manage your pipeline with the AWS CLI</td>
<td>Use the AWS CLI to disable periodic checks and create and register your webhook</td>
<td>Update pipelines for push events (GitHub source) (CLI) (p. 273)</td>
</tr>
<tr>
<td>If you create and manage your pipeline AWS CloudFormation</td>
<td>Use AWS CloudFormation to execute a change set that disables periodic checks and creates and registers your webhook</td>
<td>Update pipelines for push events (GitHub source) (AWS CloudFormation template) (p. 304)</td>
</tr>
<tr>
<td>If you created your pipeline in the console before May 1, 2018</td>
<td>Use the CodePipeline console to let CodePipeline disable periodic checks and create and register your webhook</td>
<td>Update pipelines for push events (GitHub source) (console) (p. 265)</td>
</tr>
<tr>
<td>If you create your pipeline in the console after May 1, 2018</td>
<td>No action required</td>
<td></td>
</tr>
</tbody>
</table>
Update pipelines for push events (console)

You can use the CodePipeline console to update your pipeline to use the recommended change detection method.

Update pipelines for push events (CodeCommit or Amazon S3 source) (console)

You can use the CodePipeline console to update your pipeline to use Amazon CloudWatch Events to detect changes in your CodeCommit source repository or your Amazon S3 source bucket.

**Note**

When you use the console to edit a pipeline that has a CodeCommit source repository or an Amazon S3 source bucket, the rule and IAM role are created for you. If you use the AWS CLI to edit the pipeline, you must create the Amazon CloudWatch Events rule and IAM role yourself. For more information, see [Use CloudWatch Events to start a pipeline (CodeCommit source) (p. 174)](#).

Use these steps to edit a pipeline that is using periodic checks. If you want to create a pipeline, see [Create a pipeline in CodePipeline (p. 221)](#).

**To edit the pipeline source stage**


   The names of all pipelines associated with your AWS account are displayed.

2. In Name, choose the name of the pipeline you want to edit. This opens a detailed view of the pipeline, including the state of each of the actions in each stage of the pipeline.

3. On the pipeline details page, choose Edit.

4. In Edit stage, choose the edit icon on the source action.

5. Expand Change Detection Options and choose Use CloudWatch Events to automatically start my pipeline when a change occurs (recommended).

   A message appears showing the Amazon CloudWatch Events rule to be created for this pipeline. Choose Update.
If you are updating a pipeline that has an Amazon S3 source, you see the following message. Choose Update.
6. When you have finished editing your pipeline, choose **Save pipeline changes** to return to the summary page.

   A message displays the name of the Amazon CloudWatch Events rule to be created for your pipeline. Choose **Save and continue**.

7. To test your action, release a change by using the AWS CLI to commit a change to the source specified in the source stage of the pipeline.

**Update pipelines for push events (GitHub source) (console)**

You can use the CodePipeline console to update your pipeline to use webhooks to detect changes in your CodeCommit source repository.

Follow these steps to edit a pipeline that is using polling (periodic checks) to use Amazon CloudWatch Events instead. If you want to create a pipeline, see **Create a pipeline in CodePipeline** on page 221.

When you use the console, the **PollForSourceChanges** parameter for your pipelined is changed for you. The GitHub webhook is created and registered for you.
To edit the pipeline source stage

1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home. The names of all pipelines associated with your AWS account are displayed.

2. In Name, choose the name of the pipeline you want to edit. This opens a detailed view of the pipeline, including the state of each of the actions in each stage of the pipeline.

3. On the pipeline details page, choose Edit.

4. In Edit stage, choose the edit icon on the source action.

5. Expand Change detection options and choose Use Amazon CloudWatch Events to automatically start my pipeline when a change occurs (recommended).

   A message is displayed to advise that CodePipeline creates a webhook in GitHub to detect source changes. Choose Update. In addition to the webhook, CodePipeline creates the following:

   - A secret, randomly generated and used to authorize the connection to GitHub.
   - The webhook URL, generated using the public endpoint for the Region.

   CodePipeline registers the webhook with GitHub. This subscribes the URL to receive repository events.

6. When you have finished editing your pipeline, choose Save pipeline changes to return to the summary page.

   A message displays the name of the webhook to be created for your pipeline. Choose Save and continue.

7. To test your action, release a change by using the AWS CLI to commit a change to the source specified in the source stage of the pipeline.

Update pipelines for push events (CLI)

You can use the CLI to update your pipeline to use the recommended change detection method.
Update pipelines for push events (CodeCommit source) (CLI)

Follow these steps to edit a pipeline that is using polling (periodic checks) to use a CloudWatch Events rule to start the pipeline. If you want to create a pipeline, see Create a pipeline in CodePipeline (p. 221).

To build an event-driven pipeline with CodeCommit, you edit the PollForSourceChanges parameter of your pipeline and then create the following resources:

- Amazon CloudWatch Events event
- IAM role to allow this event to start your pipeline

To edit your pipeline's PollForSourceChanges parameter

**Important**
When you create a pipeline with this method, the PollForSourceChanges parameter defaults to true if it is not explicitly set to false. When you add event-based change detection, you must add the parameter to your output and set it to false to disable polling. Otherwise, your pipeline starts twice for a single source change. For details, see Default settings for the PollForSourceChanges parameter (p. 466).

1. Run the `get-pipeline` command to copy the pipeline structure into a JSON file. For example, for a pipeline named `MyFirstPipeline`, run the following command:

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

   This command returns nothing, but the file you created should appear in the directory where you ran the command.

2. Open the JSON file in any plain-text editor and edit the source stage by changing the PollForSourceChanges parameter to `false`, as shown in this example.

   **Why am I making this change?** Changing this parameter to `false` turns off periodic checks so you can use event-based change detection only.

   ```json
   "configuration": {
   "PollForSourceChanges": "false",
   "BranchName": "master",
   "RepositoryName": "MyTestRepo"
   },
   ```

3. If you are working with the pipeline structure retrieved using the `get-pipeline` command, remove the metadata lines from the JSON file. Otherwise, the `update-pipeline` command cannot use it. Remove the "metadata": {} lines and the "created", "pipelineARN", and "updated" fields.

   For example, remove the following lines from the structure:

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

   Save the file.

4. To apply your changes, run the `update-pipeline` command, specifying the pipeline JSON file:

   **Important**
   Be sure to include `file://` before the file name. It is required in this command.
aws codepipeline update-pipeline --cli-input-json file://pipeline.json

This command returns the entire structure of the edited pipeline.

Note
The `update-pipeline` command stops the pipeline. If a revision is being run through the pipeline when you run the `update-pipeline` command, that run is stopped. You must manually start the pipeline to run that revision through the updated pipeline. Use the `start-pipeline-execution` command to manually start your pipeline.

To create a CloudWatch Events rule with CodeCommit as the event source and CodePipeline as the target

1. Add permissions for Amazon CloudWatch Events to use CodePipeline to invoke the rule. For more information, see [Using Resource-Based Policies for Amazon CloudWatch Events](#).
   a. Use the following sample to create the trust policy that allows CloudWatch Events to assume the service role. Name the trust policy `trustpolicyforCWE.json`.

   ```json
   {   "Version": "2012-10-17",
       "Statement": [   {   "Effect": "Allow",
           "Principal": {   "Service": "events.amazonaws.com"
           },
           "Action": "sts:AssumeRole"
       }   ]
   }
   ```
   
   b. Use the following command to create the `Role-for-MyRule` role and attach the trust policy.

   ```bash
   aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document file://trustpolicyforCWE.json
   ```

   c. Create the permissions policy JSON, as shown in this sample, for the pipeline named `MyFirstPipeline`. Name the permissions policy `permissionspolicyforCWE.json`.

   ```json
   {   "Version": "2012-10-17",
       "Statement": [   {   "Effect": "Allow",
           "Action": [   "codepipeline:StartPipelineExecution"
           ],
           ]
       }   ]
   }
   ```
   
   d. Use the following command to attach the `CodePipeline-Permissions-Policy-for-CWE` permissions policy to the `Role-for-MyRule` role.
**Update pipelines for push events (CLI)**

Follow these steps to edit a pipeline that is using polling (periodic checks) to use an event in CloudWatch Events instead. If you want to create a pipeline, see Create a pipeline in CodePipeline (p. 221).

To build an event-driven pipeline with Amazon S3, you edit the PollForSourceChanges parameter of your pipeline and then create the following resources:

- AWS CloudTrail trail, bucket, and bucket policy that Amazon S3 can use to log the events.
- Amazon CloudWatch Events event
- IAM role to allow the CloudWatch event to start your pipeline

**To create an AWS CloudTrail trail and enable logging**

To use the AWS CLI to create a trail, call the create-trail command, specifying:

- The trail name.
- The bucket to which you have already applied the bucket policy for AWS CloudTrail.
For more information, see Creating a Trail with the AWS Command Line Interface.

1. Call the `create-trail` command and include the `--name` and `--s3-bucket-name` parameters.

   Why am I making this change? This creates the CloudTrail trail required for your S3 source bucket.

   The following command uses `--name` and `--s3-bucket-name` to create a trail named `my-trail` and a bucket named `myBucket`.

   ```bash
   aws cloudtrail create-trail --name my-trail --s3-bucket-name myBucket
   ```

2. Call the `start-logging` command and include the `--name` parameter.

   Why am I making this change? This command starts the CloudTrail logging for your source bucket and sends events to CloudWatch Events.

   Example:

   The following command uses `--name` to start logging on a trail named `my-trail`.

   ```bash
   aws cloudtrail start-logging --name my-trail
   ```

3. Call the `put-event-selectors` command and include the `--trail-name` and `--event-selectors` parameters. Use event selectors to specify that you want your trail to log data events for your source bucket and send the events to the Amazon CloudWatch Events rule.

   Why am I making this change? This command filters events.

   Example:

   The following command uses `--trail-name` and `--event-selectors` to specify data events for a source bucket and prefix named `myBucket/myFolder`.

   ```bash
   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"] }]]]'
   ```

To create a CloudWatch Events rule with Amazon S3 as the event source and CodePipeline as the target and apply the permissions policy

1. Grant permissions for Amazon CloudWatch Events to use CodePipeline to invoke the rule. For more information, see Using Resource-Based Policies for Amazon CloudWatch Events.

   a. Use the following sample to create the trust policy to allow CloudWatch Events to assume the service role. Name it `trustpolicyforCWE.json`.

   ```json
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
           "Service": "events.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
       }
     ]
   }
   ```
b. Use the following command to create the `Role-for-MyRule` role and attach the trust policy.

**Why am I making this change?** Adding this trust policy to the role creates permissions for CloudWatch Events.

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document file://trustpolicyforCWE.json
```

c. Create the permissions policy JSON, as shown here for the pipeline named `MyFirstPipeline`. Name the permissions policy `permissionspolicyforCWE.json`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codepipeline:StartPipelineExecution"
      ],
      "Resource": [
        "arn:aws:codepipeline:us-west-2:80398EXAMPLE:MyFirstPipeline"
      ]
    }
  ]
}
```

d. Use the following command to attach the new `CodePipeline-Permissions-Policy-for-CWE` permissions policy to the `Role-for-MyRule` role you created.

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

2. Call the `put-rule` command and include the `--name`, `--event-pattern`, and `--role-arn` parameters.

The following sample command creates a rule named `MyS3SourceRule`.

```
aws events put-rule --name "MyS3SourceRule" --event-pattern "{"source": ["aws.s3"], "detail-type": ["AWS API Call via CloudTrail"], "detail": {"eventSource": ["s3.amazonaws.com"], "eventName": ["CopyObject", "PutObject", "CompleteMultipartUpload"], "requestParameters": {"bucketName": ["my-bucket"], "key": ["my-key"]})}" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-MyRule"
```

3. To add CodePipeline as a target, call the `put-targets` command and include the `--rule` and `--targets` parameters.

The following command specifies that for the rule named `MyS3SourceRule`, the target ID is composed of the number one, indicating that in a list of targets for the rule, this is target 1. The command also specifies an example ARN for the pipeline. The pipeline starts when something changes in the repository.

```
aws events put-targets --rule MyS3SourceRule --targets Id=1,Arn=arn:aws:codepipeline:us-west-2:80398EXAMPLE:TestPipeline
```
To edit your pipeline’s PollForSourceChanges parameter

**Important**
When you create a pipeline with this method, the PollForSourceChanges parameter defaults to true if it is not explicitly set to false. When you add event-based change detection, you must add the parameter to your output and set it to false to disable polling. Otherwise, your pipeline starts twice for a single source change. For details, see Default settings for the PollForSourceChanges parameter (p. 466).

1. Run the `get-pipeline` command to copy the pipeline structure into a JSON file. For example, for a pipeline named `MyFirstPipeline`, run the following command:

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

   This command returns nothing, but the file you created should appear in the directory where you ran the command.

2. Open the JSON file in any plain-text editor and edit the source stage by changing the PollForSourceChanges parameter for a bucket named `storage-bucket` to `false`, as shown in this example.

   Why am I making this change? Setting this parameter to `false` turns off periodic checks so you can use event-based change detection only.

   ```json
   "configuration": {
     "S3Bucket": "storage-bucket",
     "PollForSourceChanges": "false",
     "S3ObjectKey": "index.zip"
   },
   ```

3. If you are working with the pipeline structure retrieved using the `get-pipeline` command, you must remove the metadata lines from the JSON file. Otherwise, the `update-pipeline` command cannot use it. Remove the "metadata": {} lines and the "created", "pipelineARN", and "updated" fields.

   For example, remove the following lines from the structure:

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

   Save the file.

4. To apply your changes, run the `update-pipeline` command, specifying the pipeline JSON file:

   **Important**
   Be sure to include `file://` before the file name. It is required in this command.

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

   This command returns the entire structure of the edited pipeline.

   **Note**
   The `update-pipeline` command stops the pipeline. If a revision is being run through the pipeline when you run the `update-pipeline` command, that run is stopped. You must manually start the pipeline to run that revision through the updated pipeline. Use the `start-pipeline-execution` command to manually start your pipeline.
Update pipelines for push events (GitHub source) (CLI)

Follow these steps to edit a pipeline that is using periodic checks to use a webhook instead. If you want to create a pipeline, see Create a pipeline in CodePipeline (p. 221).

To build an event-driven pipeline, you edit the PollForSourceChanges parameter of your pipeline and then create the following resources manually:

- GitHub webhook and authorization parameters

To create and register your webhook

**Note**
When you use the CLI or AWS CloudFormation to create a pipeline and add a webhook, you must disable periodic checks. To disable periodic checks, you must explicitly add the PollForSourceChanges parameter and set it to false, as detailed in the final procedure below. Otherwise, the default for a CLI or AWS CloudFormation pipeline is that PollForSourceChanges defaults to true and does not display in the pipeline structure output. For more information about PollForSourceChanges defaults, see Default settings for the PollForSourceChanges parameter (p. 466).

1. In a text editor, create and save a JSON file for the webhook you want to create. Use this sample file for a webhook named `my-webhook`:

   ```json
   {   "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. Call the `put-webhook` command and include the `--cli-input` and `--region` parameters.

   The following sample command creates a webhook with the `webhook_json` JSON file.

   ```bash
   aws codepipeline put-webhook --cli-input-json file://webhook_json.json --region "eu-central-1"
   
   3. In the output shown in this example, the URL and ARN are returned for a webhook named `my-webhook`.

   ```json
   {   "webhook": {   "url": "https://webhooks.domain.com/trigger111111111EXAMPLE11111111111111111",   "definition": {   "authenticationConfiguration": {   "SecretToken": "secret"   },   "name": "my-webhook",   "authentication": "GITHUB_HMAC",   "targetPipeline": "pipeline_name",   "targetAction": "Source",   }   }
   ```

API Version 2015-07-09

273
"filters": [
  {
    "jsonPath": "$.ref",
    "matchEquals": "refs/heads/{Branch}"
  }
],
"tags": [{
  "key": "Project",
  "value": "ProjectA"
}]
}

This example adds tagging to the webhook by including the Project tag key and ProjectA value on the webhook. For more information about tagging resources in CodePipeline, see Tagging resources (p. 168).

4. Call the register-webhook-with-third-party command and include the --webhook-name parameter.

   The following sample command registers a webhook named my-webhook.

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

   **To edit your pipeline's PollForSourceChanges parameter**

   **Important**
   When you create a pipeline with this method, the PollForSourceChanges parameter defaults to true if it is not explicitly set to false. When you add event-based change detection, you must add the parameter to your output and set it to false to disable polling. Otherwise, your pipeline starts twice for a single source change. For details, see Default settings for the PollForSourceChanges parameter (p. 466).

1. Run the get-pipeline command to copy the pipeline structure into a JSON file. For example, for a pipeline named MyFirstPipeline, you would type the following command:

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

   This command returns nothing, but the file you created should appear in the directory where you ran the command.

2. Open the JSON file in any plain-text editor and edit the source stage by changing or adding the PollForSourceChanges parameter. In this example, for a repository named UserGitHubRepo, the parameter is set to false.

   **Why am I making this change?** Changing this parameter turns off periodic checks so you can use event-based change detection only.

   ```json
   "configuration": {
      "Owner": "darlaker",
      "Repo": "UserGitHubRepo",
      "PollForSourceChanges": "false",
      "Branch": "master",
      "OAuthToken": "****"
   },
   ```
3. If you are working with the pipeline structure retrieved using the `get-pipeline` command, you must edit the structure in the JSON file by removing the `metadata` lines from the file. Otherwise, the `update-pipeline` command cannot use it. Remove the "metadata" section from the pipeline structure in the JSON file, including the :{} and the "created", "pipelineARN", and "updated" fields.

For example, remove the following lines from the structure:

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

Save the file.

4. To apply your changes, run the `update-pipeline` command, specifying the pipeline JSON file, similar to the following:

   **Important**
   Be sure to include `file://` before the file name. It is required in this command.

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

   This command returns the entire structure of the edited pipeline.

   **Note**
   The `update-pipeline` command stops the pipeline. If a revision is being run through the pipeline when you run the `update-pipeline` command, that run is stopped. You must manually start the pipeline to run that revision through the updated pipeline. Use the `start-pipeline-execution` command to manually start your pipeline.

**Update pipelines for push events (AWS CloudFormation template)**

You can use AWS CloudFormation to update your pipeline to use the recommended method to detect changes in your source.

Follow these steps to edit a pipeline with AWS CloudFormation that is using periodic checks. If you want to create a pipeline, see Continuous Delivery with CodePipeline.

**Update pipelines for push events (CodeCommit source) (AWS CloudFormation template)**

To build an event-driven pipeline with AWS CodeCommit, you edit the `PollForSourceChanges` parameter of your pipeline and then add the following resources to your template:

- An Amazon CloudWatch Events rule
- An IAM role for your CloudWatch Events rule

If you use AWS CloudFormation to create and manage your pipelines, your template includes content like the following.
**Note**
The Configuration property in the source stage called `PollForSourceChanges`. If that property isn't included in your template, then `PollForSourceChanges` is set to true by default.

**YAML**

```yaml
Resources:
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: codecommit-polling-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        - Name: Source
          Actions:
            - Name: SourceAction
              ActionType:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: CodeCommit
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                BranchName: !Ref BranchName
                RepositoryName: !Ref RepositoryName
                PollForSourceChanges: true
                RunOrder: 1
```

**JSON**

```json
"Stages": [
  {
    "Name": "Source",
    "Actions": [
      {
        "Name": "SourceAction",
        "ActionTypeId": {
          "Category": "Source",
          "Owner": "AWS",
          "Version": 1,
          "Provider": "CodeCommit"
        },
        "OutputArtifacts": [
          {"Name": "SourceOutput"}
        ],
        "Configuration": {
          "BranchName": {"Ref": "BranchName"},
          "RepositoryName": {"Ref": "RepositoryName"},
          "PollForSourceChanges": true,
          "RunOrder": 1
        }
      }
    ]
  }
```

---

API Version 2015-07-09

276
To update your pipeline AWS CloudFormation template and create CloudWatch Events rule

1. In the template, under Resources, use the AWS::IAM::Role AWS CloudFormation resource to configure the IAM role that allows your event to start your pipeline. This entry creates a role that uses two policies:
   - The first policy allows the role to be assumed.
   - The second policy provides permissions to start the pipeline.

Why am I making this change? Adding the AWS::IAM::Role resource enables AWS CloudFormation to create permissions for CloudWatch Events. This resource is added to your AWS CloudFormation stack.

YAML

```yaml
AmazonCloudWatchEventRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: 2012-10-17
      Statement:
      - Effect: Allow
        Principal:
          Service:
          - events.amazonaws.com
        Action: sts:AssumeRole
      Path: /
      Policies:
      - PolicyName: cwe-pipeline-execution
        PolicyDocument:
          Version: 2012-10-17
          Statement:
          - Effect: Allow
            Action: codepipeline:StartPipelineExecution
```

JSON

```json
"AmazonCloudWatchEventRole": {
  "Type": "AWS::IAM::Role",
  "Properties": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": [
              "events.amazonaws.com"
            ]
        },
        "Action": "sts:AssumeRole"
      ]
    }
  }
}
```
2. In the template, under Resources, use the AWS::Events::Rule AWS CloudFormation resource to add a CloudWatch Events rule. This event pattern creates an event that monitors push changes to your repository. When CloudWatch Events detects a repository state change, the rule invokes StartPipelineExecution on your target pipeline.

**Why am I making this change?** Adding the AWS::Events::Rule resource enables AWS CloudFormation to create the event. This resource is added to your AWS CloudFormation stack.

YAML

```yaml
AmazonCloudWatchEventRule:
  Type: AWS::Events::Rule
  Properties:
    EventPattern:
      source:
        - aws.codecommit
      detail-type:
        - CodeCommit Repository State Change
      resources:
        - !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref 'RepositoryName' ] ]
      detail:
        event:
          - referenceCreated
          - referenceUpdated
        referenceType:
          - branch
        referenceName:
          - master
    Targets:
    - Arn:
```

API Version 2015-07-09
Update pipelines for push events

(AWS CloudFormation template)

```json
"AmazonCloudWatchEventRule": {
  "Type": "AWS::Events::Rule",
  "Properties": {
    "EventPattern": {
      "source": ["aws.codecommit"],
      "detail-type": ["CodeCommit Repository State Change"],
      "resources": [
        { "Fn::Join": [
          ",
          ["arn:aws:codecommit:",
           { "Ref": "AWS::Region" },
           ":",
           { "Ref": "AWS::AccountId" },
           ":",
           { "Ref": "RepositoryName" }
        ]
      ],
      "detail": {
        "event": [
          "referenceCreated",
          "referenceUpdated"
        ],
        "referenceType": ["branch"],
        "referenceName": ["master"]
      }
    },
    "Targets": [
      { "Arn": {
        "Fn::Join": [
          ",
          ["arn:aws:codepipeline:",
           { "Ref": "AWS::Region" },
           ":",
           { "Ref": "AppPipeline" }
        ]
      }
    ],
    "RoleArn": !GetAtt AmazonCloudWatchEventRole.Arn
  }
}
```

API Version 2015-07-09
3. Save the updated template to your local computer, and then open the AWS CloudFormation console.

4. Choose your stack, and then choose **Create Change Set for Current Stack**.

5. Upload the template, and then view the changes listed in AWS CloudFormation. These are the changes to be made to the stack. You should see your new resources in the list.

6. Choose **Execute**.

**To edit your pipeline’s PollForSourceChanges parameter**

**Important**

In many cases, the PollForSourceChanges parameter defaults to true when you create a pipeline. When you add event-based change detection, you must add the parameter to your output and set it to false to disable polling. Otherwise, your pipeline starts twice for a single source change. For details, see Default settings for the PollForSourceChanges parameter (p. 466).

- In the template, change PollForSourceChanges to false. If you did not include PollForSourceChanges in your pipeline definition, add it and set it to false.

**Why am I making this change?** Changing this parameter to false turns off periodic checks so you can use event-based change detection only.

**YAML**

```yaml
Name: Source
Actions:
  - Name: SourceAction
    ActionTypeId:
      Category: Source
      Owner: AWS
      Version: 1
      Provider: CodeCommit
    OutputArtifacts:
      - Name: SourceOutput
    Configuration:
      BranchName: !Ref BranchName
      RepositoryName: !Ref RepositoryName
      PollForSourceChanges: false
    RunOrder: 1
```
Update pipelines for push events

(AWS CloudFormation template)

JSON

```json
{
  "Name": "Source",
  "Actions": [
    {
      "Name": "SourceAction",
      "ActionTypeId": {
        "Category": "Source",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "CodeCommit"
      },
      "OutputArtifacts": [
        {
          "Name": "SourceOutput"
        }
      ],
      "Configuration": {
        "BranchName": {
          "Ref": "BranchName"
        },
        "RepositoryName": {
          "Ref": "RepositoryName"
        },
        "PollForSourceChanges": false
      },
      "RunOrder": 1
    }
  ]
}
```

Example

When you create these resources with AWS CloudFormation, your pipeline is triggered when files in your repository are created or updated. Here is the final template snippet:

YAML

```yaml
Resources:
  AmazonCloudWatchEventRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
        Path: /
        Policies:
          - PolicyName: cwe-pipeline-execution
            PolicyDocument:
              Version: 2012-10-17
              Statement:
                - Effect: Allow
```

API Version 2015-07-09

281
Action: codepipeline:StartPipelineExecution
Resource: !Join [' ', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ]]

AmazonCloudWatchEventRule:
  Type: AWS::Events::Rule
  Properties:
    EventPattern:
      source:
        - aws.codecommit
detail-type:
        - 'CodeCommit Repository State Change'
      resources:
        - !Join [' ', [ 'arn:aws:codecommit:', !Ref 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref RepositoryName ]]
detail:
  event:
    - referenceCreated
    - referenceUpdated
  referenceType:
    - branch
  referenceName:
    - master
  Targets:
    - Arn:
      !Join [' ', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref 'AWS::CodePipeline::Pipeline' ]]
      RoleArn: !GetAtt AmazonCloudWatchEventRole.Arn
      Id: codepipeline-AppPipeline

AppPipeline:
  Type: AWS::CodePipeline::Pipeline
  Properties:
    Name: codecommit-events-pipeline
    RoleArn:
      !GetAtt CodePipelineServiceRole.Arn
  Stages:
    - Name: Source
      Actions:
        - Name: SourceAction
          ActionType:
            Category: Source
            Owner: AWS
            Version: 1
            Provider: CodeCommit
          OutputArtifacts:
            - Name: SourceOutput
              Configuration:
                BranchName: !Ref BranchName
                RepositoryName: !Ref RepositoryName
                PollForSourceChanges: false
                RunOrder: 1

...

JSON

"Resources": {
...

"AmazonCloudWatchEventRole": {
    "Type": "AWS::IAM::Role",
"Properties": {
    "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {
                    "Service": [
                        "events.amazonaws.com"
                    ]
                },
                "Action": "sts:AssumeRole"
            }
        ]
    },
    "Path": "/",
    "Policies": [
        {
            "PolicyName": "cwe-pipeline-execution",
            "PolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Action": "codepipeline:StartPipelineExecution",
                        "Resource": {
                            "Fn::Join": [
                                ",
                                [
                                    "arn:aws:codepipeline:",
                                    {
                                        "Ref": "AWS::Region"
                                    },
                                    "",
                                    {
                                        "Ref": "AWS::AccountId"
                                    },
                                    "",
                                    {
                                        "Ref": "AppPipeline"
                                    }
                                ]
                            ]
                        }
                    }
                ]
            }
        }
    ],
    "AmazonCloudWatchEventRule": {
        "Type": "AWS::Events::Rule",
        "Properties": {
            "EventPattern": {
                "source": ["aws.codecommit"],
                "detail-type": ["CodeCommit Repository State Change"],
                "resources": [
                    "Fn::Join": [
                        ",
                        [
                            "arn:aws:codecommit:",
                            {
                                "Ref": "AWS::Region"
                            },
                            "",
                            {
                                "Ref": "AWS::AccountId"
                            },
                            "",
                            {
                                "Ref": "AppPipeline"
                            }
                        ]
                    ]
                }
            }
        }
    }
},
"API Version 2015-07-09"
283
"arn:aws:codecommit:",
{
  "Ref": "AWS::Region"
},
";",
{
  "Ref": "AWS::AccountId"
},
";",
{
  "Ref": "RepositoryName"
}
]}
],
"detail": {
  "event": [
    "referenceCreated",
    "referenceUpdated"
  ],
  "referenceType": [
    "branch"
  ],
  "referenceName": [
    "master"
  ]
},
"Targets": [
  {
    "Arn": {
      "Fn::Join": [
        ",
        [
          "arn:aws:codepipeline:",
          {
            "Ref": "AWS::Region"
          },
          ":",
          {
            "Ref": "AWS::AccountId"
          },
          ":",
          {
            "Ref": "AppPipeline"
          }
        ]
      },
      "RoleArn": {
        "Fn::GetAtt": [
          "AmazonCloudWatchEventRole",
          "Arn"
        ],
        "Id": "codepipeline-AppPipeline"
      }
    }
  }
},
"AppPipeline": {
  "Type": "AWS::CodePipeline::Pipeline",
  "Properties": {
    "Name": "codecommit-events-pipeline",
    "RoleArn": {
      "Fn::Join": [
        ",
        [
          "arn:aws:codepipeline:",
          {"Ref": "AWS::Region"}
        ]
      },
      "RoleArn": {
        "Fn::GetAtt": [
          "AmazonCloudWatchEventRole",
          "Arn"
        ],
        "Id": "codepipeline-AppPipeline"
      }
    }
  }
}
Update pipelines for push events (Amazon S3 source) (AWS CloudFormation template)

Use these steps to edit your pipeline with an Amazon S3 source from polling to event-based change detection.

To build an event-driven pipeline with Amazon S3, you edit the `PollForSourceChanges` parameter of your pipeline and then add the following resources to your template:

- Amazon CloudWatch Events requires that all Amazon S3 events must be logged. You must create an AWS CloudTrail trail, bucket, and bucket policy that Amazon S3 can use to log the events that occur. For more information, see [Logging Management and Data Events with AWS CloudTrail](#).
- Amazon CloudWatch Events rule and IAM role to allow this event to start our pipeline.

If you use AWS CloudFormation to create and manage your pipelines, your template includes content like the following.

**Note**

The `Configuration` property in the source stage called `PollForSourceChanges`. If your template doesn't include that property, then `PollForSourceChanges` is set to `true` by default.
YAML

```
AppPipeline:
  Type: AWS::CodePipeline::Pipeline
  Properties:
    RoleArn: !GetAtt CodePipelineServiceRole.Arn
    Stages:
      - Name: Source
        Actions:
          - Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
              - Name: SourceOutput
            Configuration:
              S3Bucket: !Ref SourceBucket
              S3ObjectKey: !Ref S3SourceObjectKey
              PollForSourceChanges: true
              RunOrder: 1
```

JSON

```
"AppPipeline": {
  "Type": "AWS::CodePipeline::Pipeline",
  "Properties": {
    "RoleArn": {
      "Fn::GetAtt": ["CodePipelineServiceRole", "Arn"]
    },
    "Stages": [
      {
        "Name": "Source",
        "Actions": [
          {
            "Name": "SourceAction",
            "ActionTypeId": {
              "Category": "Source",
              "Owner": "AWS",
              "Version": 1,
              "Provider": "S3"
            },
            "OutputArtifacts": [
              {
                "Name": "SourceOutput"
              }
            ],
            "Configuration": {
              "S3Bucket": {
                "Ref": "SourceBucket"
              },
              "S3ObjectKey": {
                "Ref": "S3SourceObjectKey"
              },
              "PollForSourceChanges": true
            },
            "RunOrder": 1
          }
        ]
      }
    ]
  }
```

API Version 2015-07-09

286
To create a CloudWatch Events rule with Amazon S3 as the event source and CodePipeline as the target and apply the permissions policy

1. In the template, under Resources, use the AWS::IAM::Role AWS CloudFormation resource to configure the IAM role that allows your event to start your pipeline. This entry creates a role that uses two policies:

   - The first policy allows the role to be assumed.
   - The second policy provides permissions to start the pipeline.

Why am I making this change? Adding AWS::IAM::Role resource enables AWS CloudFormation to create permissions for Amazon CloudWatch Events. This resource is added to your AWS CloudFormation stack.

YAML

```
AmazonCloudWatchEventRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: 2012-10-17
      Statement:
        - Effect: Allow
          Principal:
            Service:
            - events.amazonaws.com
          Action: sts:AssumeRole
          Path: /
          Policies:
            - PolicyName: cwe-pipeline-execution
              PolicyDocument:
                Version: 2012-10-17
                Statement:
                  - Effect: Allow
                    Action: codepipeline:StartPipelineExecution
```

JSON

```
"AmazonCloudWatchEventRole": {
  "Type": "AWS::IAM::Role",
  "Properties": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [ ...
```
Update pipelines for push events

(AWS CloudFormation template)

```json
{
  "Effect": "Allow",
  "Principal": {
    "Service": [
      "events.amazonaws.com"
    ],
  },
  "Action": "sts:AssumeRole"
}

"Path": "/",
"Policies": [
  {
    "PolicyName": "cwe-pipeline-execution",
    "PolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Action": "codepipeline:StartPipelineExecution",
          "Resource": {
            "Fn::Join": [
              "",
              [
                "arn:aws:codepipeline:
                {
                  "Ref": "AWS::Region"
                }
              ],
              [
                "",
                {
                  "Ref": "AWS::AccountId"
                },
                ",",
                {
                  "Ref": "AppPipeline"
                }
              ],
              [
              ]
            ]
          }
        }
      ]
    }
  }
]
...
```

2. Use the `AWS::Events::Rule` AWS CloudFormation resource to add a CloudWatch Events rule. This event pattern creates an event that monitors `CopyObject`, `PutObject` and `CompleteMultipartUpload` on your Amazon S3 source bucket. In addition, include a target of your pipeline. When `CopyObject`, `PutObject`, or `CompleteMultipartUpload` occurs, this rule invokes `StartPipelineExecution` on your target pipeline.

**Why am I making this change?** Adding the `AWS::Events::Rule` resource enables AWS CloudFormation to create the event. This resource is added to your AWS CloudFormation stack.

**YAML**

```yaml
AmazonCloudWatchEventRule:
  Type: AWS::Events::Rule
  Properties:
    EventPattern:
      source: 
        - aws.s3
detail-type:
        - 'AWS API Call via CloudTrail'
detail:
  eventSource:
    - s3.amazonaws.com
```

API Version 2015-07-09
288
eventName:
  - CopyObject
  - PutObject
  - CompleteMultipartUpload
requestParameters:
  bucketName:
    - !Ref SourceBucket
  key:
    - !Ref SourceObjectKey
Targets:
  - Arn:
    !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
  RoleArn: !GetAtt AmazonCloudWatchEventRole.Arn
  Id: codepipeline-AppPipeline

JSON

"AmazonCloudWatchEventRule": {
  "Type": "AWS::Events::Rule",
  "Properties": {
    "EventPattern": {
      "source": [ "aws.s3" ],
      "detail-type": [ "AWS API Call via CloudTrail" ],
      "detail": {
        "eventSource": [ "s3.amazonaws.com" ],
        "eventName": [ "CopyObject", "PutObject", "CompleteMultipartUpload" ],
        "requestParameters": {
          "bucketName": [ {
            "Ref": "SourceBucket"
          } ],
          "key": [ {
            "Ref": "SourceObjectKey"
          } ]
        }
      }
    },
    "Targets": [
      { "Arn": {
        "Fn::Join": [ "", [ "arn:aws:codepipeline:" ] ]
      }]
    }
  }
}
3. Add this snippet to your first template to allow cross-stack functionality:

**YAML**

```yaml
Outputs:
  SourceBucketARN:
    Description: "S3 bucket ARN that Cloudtrail will use"
    Value: !GetAtt SourceBucket.Arn
    Export:
      Name: SourceBucketARN
```

**JSON**

```json
"Outputs" : {
  "SourceBucketARN" : {
    "Description" : "S3 bucket ARN that Cloudtrail will use",
    "Value" : { "Fn::GetAtt": ["SourceBucket", "Arn"] },
    "Export" : {
      "Name" : "SourceBucketARN"
    }
  }
}
```

4. Save your updated template to your local computer, and open the AWS CloudFormation console.
5. Choose your stack, and then choose **Create Change Set for Current Stack**.
6. Upload your updated template, and then view the changes listed in AWS CloudFormation. These are the changes that will be made to the stack. You should see your new resources in the list.
7. Choose **Execute**.
To edit your pipeline’s PollForSourceChanges parameter

**Important**
When you create a pipeline with this method, the PollForSourceChanges parameter defaults to true if it is not explicitly set to false. When you add event-based change detection, you must add the parameter to your output and set it to false to disable polling. Otherwise, your pipeline starts twice for a single source change. For details, see Default settings for the PollForSourceChanges parameter (p. 466).

- In the template, change PollForSourceChanges to false. If you did not include PollForSourceChanges in your pipeline definition, add it and set it to false.

**Why am I making this change?** Changing PollForSourceChanges to false turns off periodic checks so you can use event-based change detection only.

**YAML**

```yaml
Name: Source
Actions:
  - Name: SourceAction
    ActionTypeId:
      Category: Source
      Owner: AWS
      Version: 1
      Provider: S3
    OutputArtifacts:
      - Name: SourceOutput
    Configuration:
      S3Bucket: !Ref SourceBucket
      S3ObjectKey: !Ref SourceObjectKey
      PollForSourceChanges: false
    RunOrder: 1
```

**JSON**

```json
{
  "Name": "SourceAction",
  "ActionTypeId": {
    "Category": "Source",
    "Owner": "AWS",
    "Version": 1,
    "Provider": "S3"
  },
  "OutputArtifacts": [
    {
      "Name": "SourceOutput"
    }
  ],
  "Configuration": {
    "S3Bucket": {
      "Ref": "SourceBucket"
    },
    "S3ObjectKey": {
      "Ref": "SourceObjectKey"
    },
    "PollForSourceChanges": false
  },
  "RunOrder": 1
}
```
To create a second template for your Amazon S3 pipeline's CloudTrail resources

- In a separate template, under Resources, use the AWS::S3::Bucket, AWS::S3::BucketPolicy, and AWS::CloudTrail::Trail AWS CloudFormation resources to provide a simple bucket definition and trail for CloudTrail.

Why am I making this change? Given the current limit of five trails per account, the CloudTrail trail must be created and managed separately. (See Limits in AWS CloudTrail.) However, you can include many Amazon S3 buckets on a single trail, so you can create the trail once and then add Amazon S3 buckets for other pipelines as necessary. Paste the following into your second sample template file.

YAML

```yaml
# Prerequisites:
# - S3 SourceBucket and SourceObjectKey must exist

Parameters:
  SourceObjectKey:
    Description: 'S3 source artifact'
    Type: String
    Default: SampleApp_Linux.zip

Resources:
  AWSCloudTrailBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref AWSCloudTrailBucket
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          - Sid: AWSCloudTrailAclCheck
            Effect: Allow
            Principal:
              Service:
                - cloudtrail.amazonaws.com
            Action: s3:GetBucketAcl
            Resource: !GetAtt AWSCloudTrailBucket.Arn
          - Sid: AWSCloudTrailWrite
            Effect: Allow
            Principal:
              Service:
                - cloudtrail.amazonaws.com
            Action: s3:PutObject
            Condition:
              StringEquals:
                s3:x-amz-acl: bucket-owner-full-control
  AWSCloudTrailBucket:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
  AWS::CloudTrail:
    DependsOn:
      - AWSCloudTrailBucketPolicy
    Type: AWS::CloudTrail::Trail
    Properties:
      S3BucketName: !Ref AWSCloudTrailBucket
      EventSelectors:
      - DataResources:
```

API Version 2015-07-09

292
Update pipelines for push events

(AWS CloudFormation template)

```
- Type: AWS::S3::Object
  Values:
  - !Join [ ' ', [ !ImportValue SourceBucketARN, '/', !Ref SourceObjectKey ] ]
  ReadWriteType: WriteOnly
  IncludeGlobalServiceEvents: true
  IsLogging: true
  IsMultiRegionTrail: true

...
```

**JSON**

```json
{
  "Parameters": {
    "SourceObjectKey": {
      "Description": "S3 source artifact",
      "Type": "String",
      "Default": "SampleApp_Linux.zip"
    }
  },
  "Resources": {
    "AWSCloudTrailBucket": {
      "Type": "AWS::S3::Bucket",
      "DeletionPolicy": "Retain"
    },
    "AWSCloudTrailBucketPolicy": {
      "Type": "AWS::S3::BucketPolicy",
      "Properties": {
        "Bucket": {
          "Ref": "AWSCloudTrailBucket"
        },
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "AWSCloudTrailAclCheck",
              "Effect": "Allow",
              "Principal": {
                "Service": ["cloudtrail.amazonaws.com"
              ],
              "Action": "s3:GetBucketAcl",
              "Resource": {
                "Fn::GetAtt": [
                  "AWSCloudTrailBucket",
                  "Arn"
                ]
              }
            },
            {
              "Sid": "AWSCloudTrailWrite",
              "Effect": "Allow",
              "Principal": {
                "Service": ["cloudtrail.amazonaws.com"
              ],
              "Action": "s3:PutObject",
              "Resource": {
                "Fn::Join": [
                  
```
```
","[
{"Fn::GetAtt": [
  "AWSCloudTrailBucket",
  "Arn"
]
},
"/AWSLogs/",
{"Ref": "AWS::AccountId"
},
"/*"
]
],
"Condition": {
  "StringEquals": {
    "s3:x-amz-acl": "bucket-owner-full-control"
  }
}
]
}
"AwsCloudTrail": {
  "DependsOn": [
    "AWSCloudTrailBucketPolicy"
  ],
  "Type": "AWS::CloudTrail::Trail",
  "Properties": {
    "S3BucketName": {
      "Ref": "AWSCloudTrailBucket"
    },
    "EventSelectors": [
      {
        "DataResources": [
          {
            "Type": "AWS::S3::Object",
            "Values": [
              {
                "Fn::Join": [
                  "",
                  ["Fn::ImportValue": "SourceBucketARN",
                    "/",
                    {"Ref": "SourceObjectKey"}
                  ]
                ]
              }
            ]
          }
        ],
        "ReadWriteType": "WriteOnly"
      }
    ],
    "IncludeGlobalServiceEvents": true,
    "IsLogging": true,
    "IsMultiRegionTrail": true
  }
}
```
Example

When you use AWS CloudFormation to create these resources, your pipeline is triggered when files in your repository are created or updated.

**Note**

Do not stop here. Although your pipeline is created, you must create a second AWS CloudFormation template for your Amazon S3 pipeline. If you do not create the second template, your pipeline does not have any change detection functionality.

**YAML**

```
Resources:
  SourceBucket:
    Type: AWS::S3::Bucket
    Properties:
      VersioningConfiguration:
        Status: Enabled
  CodePipelineArtifactStoreBucket:
    Type: AWS::S3::Bucket
  CodePipelineArtifactStoreBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref CodePipelineArtifactStoreBucket
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          - Sid: DenyUnEncryptedObjectUploads
            Effect: Deny
            Principal: '*'
            Action: s3:PutObject
            Condition:
              StringNotEquals:
                s3:x-amz-server-side-encryption: aws:kms
          - Sid: DenyInsecureConnections
            Effect: Deny
            Principal: '*'
            Action: s3:*
            Condition:
              Bool:
                aws:SecureTransport: false
  CodePipelineServiceRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - codepipeline.amazonaws.com
```

**API Version 2015-07-09**
Action: sts:AssumeRole
Path: /
Policies:
  - PolicyName: AWS-CodePipeline-Service-3
    PolicyDocument:
      Version: 2012-10-17
      Statement:
        - Effect: Allow
          Action:
            - codecommit:CancelUploadArchive
            - codecommit:GetBranch
            - codecommit:GetCommit
            - codecommit:GetUploadArchiveStatus
            - codecommit:UploadArchive
          Resource: '*'
        - Effect: Allow
          Action:
            - codedeploy:CreateDeployment
            - codedeploy:GetApplicationRevision
            - codedeploy:GetDeployment
            - codedeploy:GetDeploymentConfig
            - codedeploy:RegisterApplicationRevision
          Resource: '*'
        - Effect: Allow
          Action:
            - codebuild:BatchGetBuilds
            - codebuild:StartBuild
          Resource: '*'
        - Effect: Allow
          Action:
            - devicefarm:ListProjects
            - devicefarm:ListDevicePools
            - devicefarm:GetRun
            - devicefarm:GetUpload
            - devicefarm:CreateUpload
            - devicefarm:ScheduleRun
          Resource: '*'
        - Effect: Allow
          Action:
            - lambda:InvokeFunction
            - lambda:ListFunctions
          Resource: '*'
        - Effect: Allow
          Action:
            - iam:PassRole
          Resource: '*'
        - Effect: Allow
          Action:
            - elasticbeanstalk:
            - ec2:
            - elasticloadbalancing:
            - autoscaling:
            - cloudwatch:
            - s3:
            - sns:
            - cloudformation:
            - rds:
            - sqs:

API Version 2015-07-09
296
AppPipeline:
Type: AWS::CodePipeline::Pipeline
Properties:
  Name: s3-events-pipeline
  RoleArn:
    !GetAtt CodePipelineServiceRole.Arn
  Stages:
  - Name: Source
    Actions:
    - Name: SourceAction
      ActionType: Source
      Configuration:
        S3Bucket: !Ref SourceBucket
        S3ObjectKey: !Ref SourceObjectKey
        PollForSourceChanges: false
      RunOrder: 1
  - Name: Beta
    Actions:
    - Name: BetaAction
      InputArtifacts:
        - Name: SourceOutput
      Configuration:
        ApplicationName: !Ref ApplicationName
        DeploymentGroupName: !Ref BetaFleet
      RunOrder: 1
  ArtifactStore:
    Type: S3
    Location: !Ref CodePipelineArtifactStoreBucket
AmazonCloudWatchEventRole:
Type: AWS::IAM::Role
Properties:
  AssumeRolePolicyDocument:
    Version: 2012-10-17
    Statement:
    - Effect: Allow
      Principal:
        Service:
        - events.amazonaws.com
      Action: sts:AssumeRole
Path: /
Policies:
- PolicyName: cwe-pipeline-execution
  PolicyDocument:
    Version: 2012-10-17
    Statement:
    - Effect: Allow
Update pipelines for push events

AWS CloudFormation template

```json
Action: codepipeline:StartPipelineExecution
AmazonCloudWatchEventRule:
    Type: AWS::Events::Rule
    Properties:
        EventPattern:
            source:
                - aws.s3
            detail-type:
                - 'AWS API Call via CloudTrail'
        detail:
            eventSource:
                - s3.amazonaws.com
            eventName:
                - PutObject
                - CompleteMultipartUpload
        resources:
            ARN:
        Targets:
            - Arn:
                !Join [ ' ', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
                RoleArn: !GetAtt AmazonCloudWatchEventRole.Arn
            Id: codepipeline-AppPipeline

Outputs:
SourceBucketARN:
    Description: "S3 bucket ARN that Cloudtrail will use"
    Value: !GetAtt SourceBucket.Arn
    Export:
        Name: SourceBucketARN
```

```
"Resources": {
    "SourceBucket": {
        "Type": "AWS::S3::Bucket",
        "Properties": {
            "VersioningConfiguration": {
                "Status": "Enabled"
            }
        }
    },
    "CodePipelineArtifactStoreBucket": {
        "Type": "AWS::S3::Bucket"
    },
    "CodePipelineArtifactStoreBucketPolicy": {
        "Type": "AWS::S3::BucketPolicy",
        "Properties": {
            "Bucket": {
                "Ref": "CodePipelineArtifactStoreBucket"
            },
            "PolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {"Sid": "DenyUnEncryptedObjectUploads","Effect": "Deny","Principal": "*","Action": "s3:PutObject","Resource": {"Fn::Join": ["",["arn:aws:s3:::","!GetAtt SourceBucket.Arn","/",!Ref SourceObjectKey]]}]
            }
        }
    }
}
```
Update pipelines for push events
(AWS CloudFormation template)

```
"",
[
{
    "Fn::GetAtt": [
        "CodePipelineArtifactStoreBucket",
        "Arn"
    ],
    "/*"
}
],
"/*"
]
"Condition": {
    "StringNotEquals": {
        "s3:x-amz-server-side-encryption": "aws:kms"
    }
},
{
    "Sid": "DenyInsecureConnections",
    "Effect": "Deny",
    "Principal": "*",
    "Action": "s3:*",
    "Resource": {
        "Fn::Join": [
            "",
            {
                "Fn::GetAtt": [
                    "CodePipelineArtifactStoreBucket",
                    "Arn"
                ],
                "/*"
            }
        ]
    }
},
"/*"
]
"Condition": {
    "Bool": {
        "aws:SecureTransport": false
    }
}
",
"Path": "/",
"Policies": [
", "CodePipelineServiceRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": [
                            "codepipeline.amazonaws.com"
                        ]
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        }
    }
}
```

API Version 2015-07-09
299
Update pipelines for push events

(AWS CloudFormation template)

```json
{
    "PolicyName": "AWS-CodePipeline-Service-3",
    "PolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "codecommit:CancelUploadArchive",
                    "codecommit:GetBranch",
                    "codecommit:GetCommit",
                    "codecommit:GetUploadArchiveStatus",
                    "codecommit:UploadArchive"
                ],
                "Resource": "*"
            },
            {
                "Effect": "Allow",
                "Action": [
                    "codedeploy:CreateDeployment",
                    "codedeploy:GetApplicationRevision",
                    "codedeploy:GetDeployment",
                    "codedeploy:GetDeploymentConfig",
                    "codedeploy:RegisterApplicationRevision"
                ],
                "Resource": "*"
            },
            {
                "Effect": "Allow",
                "Action": [
                    "codebuild:BatchGetBuilds",
                    "codebuild:StartBuild"
                ],
                "Resource": "*"
            },
            {
                "Effect": "Allow",
                "Action": [
                    "devicefarm:ListProjects",
                    "devicefarm:ListDevicePools",
                    "devicefarm:GetRun",
                    "devicefarm:GetUpload",
                    "devicefarm:CreateUpload",
                    "devicefarm:ScheduleRun"
                ],
                "Resource": "*"
            },
            {
                "Effect": "Allow",
                "Action": [
                    "lambda:InvokeFunction",
                    "lambda:ListFunctions"
                ],
                "Resource": "*"
            },
            {
                "Effect": "Allow",
                "Action": [
                    "iam:PassRole"
                ],
                "Resource": "*"
            },
            {
                "Effect": "Allow",
                "Action": [
                    "elasticbeanstalk:*"
                ],
                "Resource": "*"
            }
        ]
    }
}
```
Update pipelines for push events
(AWS CloudFormation template)

```
"ec2:*",
"elasticloadbalancing:*",
"autoscaling:*",
"cloudwatch:*",
"s3:*",
"sns:*",
"cloudformation:*",
"rds:*",
"sqs:*",
"ecs:*"
],
"Resource": "*
}
}
]
}
}
}
"AppPipeline": {
  "Type": "AWS::CodePipeline::Pipeline",
  "Properties": {
    "Name": "s3-events-pipeline",
    "RoleArn": {
      "Fn::GetAtt": [
        "CodePipelineServiceRole",
        "Arn"
      ],
    },
    "Stages": [
      {
        "Name": "Source",
        "Actions": [
          {
            "Name": "SourceAction",
            "ActionTypeId": {
              "Category": "Source",
              "Owner": "AWS",
              "Version": 1,
              "Provider": "S3"
            },
            "OutputArtifacts": [
              {
                "Name": "SourceOutput"
              }
            ],
            "Configuration": {
              "S3Bucket": {
                "Ref": "SourceBucket"
              },
              "S3ObjectKey": {
                "Ref": "SourceObjectKey"
              },
              "PollForSourceChanges": false
            },
            "RunOrder": 1
          }
        ]
      },
      {
        "Name": "Beta",
        "Actions": [
          {
            "Name": "BetaAction",
            "InputArtifacts": [
              {
            }
          ]
        ]
      }
    ]
  }
}
```

API Version 2015-07-09
301
"Name": "SourceOutput"
},
"ActionTypeId": {
  "Category": "Deploy",
  "Owner": "AWS",
  "Version": 1,
  "Provider": "CodeDeploy"
},
"Configuration": {
  "ApplicationName": {
    "Ref": "ApplicationName"
  },
  "DeploymentGroupName": {
    "Ref": "BetaFleet"
  },
  "RunOrder": 1
}
},
"ArtifactStore": {
  "Type": "S3",
  "Location": {
    "Ref": "CodePipelineArtifactStoreBucket"
  }
}
},
"AmazonCloudWatchEventRole": {
  "Type": "AWS::IAM::Role",
  "Properties": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": ["events.amazonaws.com"]
          },
          "Action": "sts:AssumeRole"
        }
      ]
    },
    "Path": "/",
    "Policies": [
      {
        "PolicyName": "cwe-pipeline-execution",
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Action": "codepipeline:StartPipelineExecution",
              "Resource": {
                "Fn::Join": [
                  "",
                  ["arn:aws:codepipeline:",
                   {"Ref": "AWS::Region"}],
                ""]
              }
            }
          ]
        }
      }
    ]
  }
}
Update pipelines for push events

(AWS CloudFormation template)

```json
{
  "Ref": "AWS::AccountId",
  ":",
  {
    "Ref": "AppPipeline"
  }
}

"AmazonCloudWatchEventRule": {
  "Type": "AWS::Events::Rule",
  "Properties": {
    "EventPattern": {
      "source": ["aws.s3"],
      "detail-type": ["AWS API Call via CloudTrail"],
      "detail": {
        "eventSource": ["s3.amazonaws.com"],
        "eventName": ["PutObject", "CompleteMultipartUpload"],
        "resources": {
          "ARN": [
            {"Fn::Join": ["", [
              {"Fn::GetAtt": [
                "SourceBucket",
                "Arn"
              }], "/", {
                "Ref": "SourceObjectKey"
              }
            ]}
          ]
        }
      }
    },
    "Targets": [
      {"Arn": {
        "Fn::Join": ["",
          {"Fn::GetAtt": [
            "SourceBucket",
            "arn:aws:codepipeline:"
          ]}
        ]}
      }
    ]
  }
}
```

API Version 2015-07-09

303
Update pipelines for push events (GitHub source) (AWS CloudFormation template)

Follow these steps to update your pipeline (with a GitHub source) from periodic checks (polling) to event-based change detection using webhooks.

To build an event-driven pipeline with AWS CodeCommit, you edit the PollForSourceChanges parameter of your pipeline and then add the following resources to your template:

- A GitHub webhook

If you use AWS CloudFormation to create and manage your pipelines, your template has content like the following.

**Note**

Note the PollForSourceChanges configuration property in the source stage. If your template doesn't include that property, then PollForSourceChanges is set to true by default.

**YAML**

```
Resources:
```

...
AppPipeline:
  Type: AWS::CodePipeline::Pipeline
  Properties:
    Name: github-polling-pipeline
    RoleArn:
      !GetAtt CodePipelineServiceRole.Arn
  Stages:
    - Name: Source
      Actions:
        - Name: SourceAction
          ActionType:
            Category: Source
            Owner: ThirdParty
            Version: 1
            Provider: GitHub
          OutputArtifacts:
            - Name: SourceOutput
          Configuration:
            Owner: !Ref GitHubOwner
            Repo: !Ref RepositoryName
            Branch: !Ref BranchName
            OAuthToken:
              {{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
  PollForSourceChanges: true
  RunOrder: 1

JSON

"AppPipeline": {
  "Type": "AWS::CodePipeline::Pipeline",
  "Properties": {
    "Name": "github-polling-pipeline",
    "RoleArn": {
      "Fn": "GetAtt": [
        "CodePipelineServiceRole",
        "Arn"
      ]
    },
    "Stages": [
      {
        "Name": "Source",
        "Actions": [
          {
            "Name": "SourceAction",
            "ActionTypeId": {
              "Category": "Source",
              "Owner": "ThirdParty",
              "Version": 1,
              "Provider": "GitHub"
            },
            "OutputArtifacts": [
              {
                "Name": "SourceOutput"
              }
            ],
            "Configuration": {
              "Owner": {
                "Ref": "GitHubOwner"
              },
              "Repo": {
                "Ref": "RepositoryName"
              }
            }
          }
        ]
      }
    ]
  }"
To add parameters and create a webhook in your template

We strongly recommend that you use AWS Secrets Manager to store your credentials. If you use Secrets Manager, you must have already configured and stored your secret parameters in Secrets Manager. This example uses dynamic references to AWS Secrets Manager for the GitHub credentials for your webhook. For more information, see Using Dynamic References to Specify Template Values.

**Important**

When passing secret parameters, do not enter the value directly into the template. The value is rendered as plaintext and is therefore readable. For security reasons, do not use plaintext in your AWS CloudFormation template to store your credentials.

When you use the CLI or AWS CloudFormation to create a pipeline and add a webhook, you must disable periodic checks.

**Note**

To disable periodic checks, you must explicitly add the `PollForSourceChanges` parameter and set it to false, as detailed in the final procedure below. Otherwise, the default for a CLI or AWS CloudFormation pipeline is that `PollForSourceChanges` defaults to true and does not display in the pipeline structure output. For more information about `PollForSourceChanges` defaults, see Default settings for the `PollForSourceChanges` parameter (p. 466).

1. In the template, under Resources, add your parameters:

   **YAML**
   
   ```yaml
   Parameters:
   GitHubOwner:
     Type: String
   ...
   ```

   **JSON**
   
   ```json
   {
   "Parameters": {
   "BranchName": {
   "Description": "GitHub branch name",
   "Type": "String",
   "Default": "master"
   },
   "GitHubOwner": {
   "Type": "String"
   },
   ...
   ```
2. Use the `AWS::CodePipeline::Webhook` AWS CloudFormation resource to add a webhook.

   **Note**
   The `TargetAction` you specify must match the `Name` property of the source action defined in the pipeline.

   If `RegisterWithThirdParty` is set to `true`, make sure the user associated to the OAuthToken can set the required scopes in GitHub. The token and webhook require the following GitHub scopes:

   - `repo` - used for full control to read and pull artifacts from public and private repositories into a pipeline.
   - `admin:repo_hook` - used for full control of repository hooks.

   Otherwise, GitHub returns a 404. For more information about the 404 returned, see https://help.github.com/articles/about-webhooks.

   **YAML**

   ```yaml
   AppPipelineWebhook:
     Type: AWS::CodePipeline::Webhook
     Properties:
       Authentication: GITHUB_HMAC
       AuthenticationConfiguration:
       Filters:
         - JsonPath: "$.ref"
         MatchEquals: refs/heads/{Branch}
       TargetPipeline: !Ref AppPipeline
       TargetAction: SourceAction
       Name: AppPipelineWebhook
       TargetPipelineVersion: !GetAtt AppPipeline.Version
       RegisterWithThirdParty: true
   ...
   
   **JSON**

   ```json
   "AppPipelineWebhook": {
     "Type": "AWS::CodePipeline::Webhook",
     "Properties": {
       "Authentication": "GITHUB_HMAC",
       "AuthenticationConfiguration": {
         "SecretToken": 
         "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}"
       },
       "Filters": [
         { "JsonPath": "$.ref",
           "MatchEquals": "refs/heads/{Branch}"}
       ],
       "TargetPipeline": {
         "Ref": "AppPipeline"
       }
   }
   ```
Update pipelines for push events
(AWS CloudFormation template)

"TargetAction": "SourceAction",
"Name": "AppPipelineWebhook",
"TargetPipelineVersion": {
  "Fn::GetAtt": [
    "AppPipeline",
    "Version"
  ]
},
"RegisterWithThirdParty": true
},

3. Save the updated template to your local computer, and then open the AWS CloudFormation console.
4. Choose your stack, and then choose **Create Change Set for Current Stack**.
5. Upload the template, and then view the changes listed in AWS CloudFormation. These are the changes to be made to the stack. You should see your new resources in the list.
6. Choose **Execute**.

**To edit your pipeline’s PollForSourceChanges parameter**

**Important**

When you create a pipeline with this method, the PollForSourceChanges parameter defaults to true if it is not explicitly set to false. When you add event-based change detection, you must add the parameter to your output and set it to false to disable polling. Otherwise, your pipeline starts twice for a single source change. For details, see Default settings for the PollForSourceChanges parameter (p. 466).

- In the template, change PollForSourceChanges to false. If you did not include PollForSourceChanges in your pipeline definition, add it and set it to false.

**Why am I making this change?** Changing this parameter to false turns off periodic checks so you can use event-based change detection only.

**YAML**

```yaml
Name: Source
Actions:
  - Name: SourceAction
    ActionTypeId:
      Category: Source
      Owner: ThirdParty
      Version: 1
      Provider: GitHub
    OutputArtifacts:
      - Name: SourceOutput
    Configuration:
      Owner: !Ref GitHubOwner
      Repo: !Ref RepositoryName
      Branch: !Ref BranchName
      OAuthToken:
        {{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
    PollForSourceChanges: false
    RunOrder: 1
```

**JSON**

```json
{
```

API Version 2015-07-09
308
"Name": "Source",
"Actions": [
{
   "Name": "SourceAction",
   "ActionTypeId": {
      "Category": "Source",
      "Owner": "ThirdParty",
      "Version": 1,
      "Provider": "GitHub"
   },
   "OutputArtifacts": [
   {
      "Name": "SourceOutput"
   }
   ],
   "Configuration": {
      "Owner": {
         "Ref": "GitHubOwner"
      },
      "Repo": {
         "Ref": "RepositoryName"
      },
      "Branch": {
         "Ref": "BranchName"
      },
      "OAuthToken": "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}",
      "PollForSourceChanges": false
   },
   "RunOrder": 1
}

Example
When you create these resources with AWS CloudFormation, the webhook defined is created in the specified GitHub repository. Your pipeline is triggered on commit.

YAML

Parameters:
GitHubOwner:
   Type: String

Resources:
AppPipelineWebhook:
   Type: AWS::CodePipeline::Webhook
   Properties:
      Authentication: GITHUB_HMAC
      AuthenticationConfiguration:
      Filters:
         - JsonPath: ".ref"
           MatchEquals: refs/heads/(Branch)
      TargetPipeline: !Ref AppPipeline
      TargetAction: SourceAction
      Name: AppPipelineWebhook
      TargetPipelineVersion: !GetAtt AppPipeline.Version
      RegisterWithThirdParty: true

AppPipeline:
   Type: AWS::CodePipeline::Pipeline
   Properties:
Name: github-events-pipeline
RoleArn: !GetAtt CodePipelineServiceRole.Arn
Stages:
  - Name: Source
    Actions:
      - Name: SourceAction
        ActionTypeId:
          Category: Source
          Owner: ThirdParty
          Version: 1
          Provider: GitHub
        OutputArtifacts:
          - Name: SourceOutput
            Configuration:
              Owner: !Ref GitHubOwner
              Repo: !Ref RepositoryName
              Branch: !Ref BranchName
              PollForSourceChanges: false
              RunOrder: 1

JSON

```json
{
  "Parameters": {
    "BranchName": {
      "Description": "GitHub branch name",
      "Type": "String",
      "Default": "master"
    },
    "RepositoryName": {
      "Description": "GitHub repository name",
      "Type": "String",
      "Default": "test"
    },
    "GitHubOwner": {
      "Type": "String"
    },
    "ApplicationName": {
      "Description": "CodeDeploy application name",
      "Type": "String",
      "Default": "DemoApplication"
    },
    "BetaFleet": {
      "Description": "Fleet configured in CodeDeploy",
      "Type": "String",
      "Default": "DemoFleet"
    }
  },
  "Resources": {
    "AppPipelineWebhook": {
      "Type": "AWS::CodePipeline::Webhook",
      "Properties": {
        "Authentication": "GITHUB_HMAC",
        "AuthenticationConfiguration": {
```
"SecretToken": {
  "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}"
},
"Filters": [
  {
    "JsonPath": ".ref",
    "MatchEquals": "refs/heads/{Branch}"  
  }
],
"TargetPipeline": {
  "Ref": "AppPipeline"
},
"TargetAction": "SourceAction",
"Name": "AppPipelineWebhook",
"TargetPipelineVersion": {
  "Fn::GetAtt": [
    "AppPipeline",
    "Version"
  ]
},
"RegisterWithThirdParty": true
},
"AppPipeline": {
  "Type": "AWS::CodePipeline::Pipeline",
  "Properties": {
    "Name": "github-events-pipeline",
    "RoleArn": {
      "Fn::GetAtt": [
        "CodePipelineServiceRole",
        "Arn"
      ]
    },
    "Stages": [
      {
        "Name": "Source",
        "Actions": [
          {
            "Name": "SourceAction",
            "ActionTypeId": {
              "Category": "Source",
              "Owner": "ThirdParty",
              "Version": 1,
              "Provider": "GitHub"
            },
            "OutputArtifacts": [
              {
                "Name": "SourceOutput"
              }
            ],
            "Configuration": {
              "Owner": {
                "Ref": "GitHubOwner"
              },
              "Repo": {
                "Ref": "RepositoryName"
              },
              "Branch": {
                "Ref": "BranchName"
              },
              "OAuthToken": {
                "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}",
                "PollForSourceChanges": false
              }
            }
          }
        ]
      }
    ]
  }
}
Create the CodePipeline service role

When you create a pipeline, you create a service role or use an existing service role.

You can use the CodePipeline console or the AWS CLI to create an CodePipeline service role. A service role is required to create a pipeline, and the pipeline is always associated to that service role.

The service role is not an AWS managed role but is created initially for pipeline creation, and then as new permissions are added to the service role policy, you may need to update the service role for your pipeline. Once your pipeline is created with a service role, you cannot apply a different service role to that pipeline. Attach the recommended policy to the service role.

For more information about the service role, see Manage the CodePipeline service role (p. 445).

Create the CodePipeline service role (console)


   Choose Create pipeline and complete the Step 1: Choose pipeline settings page in the pipeline creation wizard.

   Note
   After you create a pipeline, you cannot change its name. For information about other limitations, see Quotas in AWS CodePipeline (p. 521).

2. In Service role, do one of the following:

   • Choose New service role to allow CodePipeline to create a new service role in IAM. In Role name, the role and policy name both default to this format: AWSCodePipelineServiceRole-region-pipeline_name. For example, this is the service role created for this tutorial: AWSCodePipelineServiceRole-eu-west-2-MyFirstPipeline.

   • Choose Existing service role to use a service role already created in IAM. In Role name, choose your service role from the list.

   Note
   In the console, service roles created before September 2018 are created with the name "oneClick_AWS-CodePipeline-Service_ID-Number". Service roles created after September 2018 use the service role name format "AWSCodePipelineServiceRole-Region-Pipeline_Name". For example, for a pipeline named MyFirstPipeline created in the console in eu-west-2, the service role named "AWSCodePipelineServiceRole-eu-west-2-MyFirstPipeline" is created. The policy name format is the same as the role name format.

3. Complete the pipeline creation. Your pipeline service role is available to view in your list of IAM roles, and you can view the service role ARN associated to a pipeline by running the get-pipeline command with the AWS CLI.

Create the CodePipeline service role (CLI)

Before you create a pipeline with the AWS CLI or AWS CloudFormation, create a CodePipeline service role for your pipeline.
Tag a pipeline in CodePipeline

Tags are key-value pairs associated with AWS resources. You can apply tags to your pipelines in CodePipeline. For information about CodePipeline resource tagging, use cases, tag key and value constraints, and supported resource types, see Tagging resources (p. 168).

You can use the CLI to specify tags when you create a pipeline. You can use the console or CLI to add or remove tags, and update the values of tags in a pipeline. You can add up to 50 tags to each pipeline.

Topics
- Tag pipelines (console) (p. 313)
- Tag pipelines (CLI) (p. 315)

Add tags to a pipeline (console)

You can use the console to add tags to an existing pipeline.

2. On the Pipelines page, choose the pipeline where you want to add tags.
3. From the navigation pane, choose Settings.
4. Under Pipeline tags, choose Edit.
5. In the Key and Value fields, enter a key pair for each set of tags you want to add. (The Value field is optional.) For example, in Key, enter Project. In Value, enter ProjectA.
6. (Optional) Choose Add tag to add more rows and enter more tags.
7. Choose Submit. The tags are listed under pipeline settings.

View tags for a pipeline (console)

You can use the console to list tags for existing pipelines.

2. On the Pipelines page, choose the pipeline where you want to view tags.
3. From the navigation pane, choose Settings.
4. Under Pipeline tags, view the tags for the pipeline under the Key and Value columns.
Edit tags for a pipeline (console)

You can use the console to edit tags that have been added to pipelines.

2. On the Pipelines page, choose the pipeline where you want to update tags.
3. From the navigation pane, choose Settings.
4. Under Pipeline tags, choose Edit.
5. In the Key and Value fields, update the values in each field as needed. For example, for the Project key, in Value, change ProjectA to ProjectB.
6. Choose Submit.

Remove tags from a pipeline (console)

You can use the console to delete tags from pipelines. When you remove tags from the associated resource, the tags are deleted.

2. On the Pipelines page, choose the pipeline where you want to remove tags.
3. From the navigation pane, choose Settings.
4. Under Pipeline tags, choose Edit.
5. Next to the key and value for each tag you want to delete, choose Remove tag.
6. Choose Submit.

Tag pipelines (CLI)

You can use the CLI to tag resources. You must use the console to manage tags in pipelines.

Topics
- Add tags to a pipeline (CLI) (p. 315)
- View tags for a pipeline (CLI) (p. 316)
- Edit tags for a pipeline (CLI) (p. 316)
- Remove tags from a pipeline (CLI) (p. 316)

Add tags to a pipeline (CLI)

You can use the console or the AWS CLI to tag pipelines.

To add a tag to a pipeline when you create it, see Create a pipeline in CodePipeline (p. 221).

In these steps, we assume that you have already installed a recent version of the AWS CLI or updated to the current version. For more information, see Installing the AWS Command Line Interface.

At the terminal or command line, run the tag-resource command, specifying the Amazon Resource Name (ARN) of the pipeline where you want to add tags and the key and value of the tag you want to add. You can add more than one tag to a pipeline. For example, to tag a pipeline named MyPipeline
with two tags, a tag key named `DeploymentEnvironment` with the tag value of `Test`, and a tag key named `IsContainerBased` with the tag value of `true`:

```
aws codepipeline tag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:MyPipeline --tags key=Project,value=ProjectA key=IsContainerBased,value=true
```

If successful, this command returns nothing.

### View tags for a pipeline (CLI)

Follow these steps to use the AWS CLI to view the AWS tags for a pipeline. If no tags have been added, the returned list is empty.

At the terminal or command line, run the `list-tags-for-resource` command. For example, to view a list of tag keys and tag values for a pipeline named `MyPipeline` with the ARN value:

```
```

If successful, this command returns information similar to the following:

```
{
  "tags": {
    "Project": "ProjectA",
    "IsContainerBased": "true"
  }
}
```

### Edit tags for a pipeline (CLI)

Follow these steps to use the AWS CLI to edit a tag for a pipeline. You can change the value for an existing key or add another key. You can also remove tags from a pipeline, as shown in the next section.

At the terminal or command line, run the `tag-resource` command, specifying the ARN of the pipeline where you want to update a tag and specify the tag key and tag value:

```
```

If successful, this command returns nothing.

### Remove tags from a pipeline (CLI)

Follow these steps to use the AWS CLI to remove a tag from a pipeline. When you remove tags from the associated resource, the tags are deleted.

**Note**

If you delete a pipeline, all tag associations are removed from the deleted pipeline. You do not have to remove tags before you delete a pipeline.

At the terminal or command line, run the `untag-resource` command, specifying the ARN of the pipeline where you want to remove tags and the tag key of the tag you want to remove. For example, to remove multiple tags on a pipeline named `MyPipeline` with the tag keys `Project` and `IsContainerBased`:
Create a connection to Bitbucket

You can use the AWS CodePipeline console to create a connection to a third-party code repository. As an example, this topic shows you how to create a connection to a Bitbucket repository. Before you begin:

- You must have created an account with the provider of the third-party repository, such as Bitbucket.
- You must have already created a third-party code repository, such as a Bitbucket repository.

Note

Bitbucket connections only provide access to repositories owned by the Bitbucket account that was used to create the connection.

Topics

- Create a connection (console) (p. 317)
- Create a connection (CLI) (p. 321)

Create a connection (console)

You use the console to create a connection by using one of the following methods:

- Use the Create Pipeline wizard or the Edit action page in the CodePipeline console. Note that when you configure a CodePipeline action, you can choose existing connections or create a new connection.
- Associate your Bitbucket repository to your feedback and analysis tools in Amazon CodeGuru Reviewer.

This example shows the flow to create connections as part of the Create Pipeline wizard in CodePipeline. To create a connection to a third-party source provider in the console, you must provide the source file location and information about the repository.

Step 1: Create your pipeline and connection

1. Sign in to the CodePipeline console.
2. On the Welcome page, choose Create pipeline.

   If this is your first time using CodePipeline, choose Get Started.
3. On the Step 1: Choose pipeline settings page, in Pipeline name, enter the name for your pipeline.

   In a single AWS account, each pipeline you create in an AWS Region must have a unique name. Names can be reused for pipelines in different Regions.

   Note

   After you create a pipeline, you cannot change its name. For information about other limitations, see Quotas in AWS CodePipeline (p. 521).
4. In Service role, do one of the following:
To allow CodePipeline to create a service role in AWS Identity and Access Management (IAM), choose **New service role**. In **Role name**, the role and policy name both default to this format: AWSCodePipelineServiceRole-region-pipeline_name. For example, this is the service role for a pipeline named MyPipeline: AWSCodePipelineServiceRole-us-west-2-MyPipeline.

To use a service role that is already created in IAM, choose **Existing service role**. In **Role ARN**, choose your service role Amazon Resource Name (ARN) from the list.

**Note**
Depending on when your service role was created, you might need to update its permissions to support AWS CodeStar connections. For instructions for the CodePipeline service role, see Update the CodePipeline Service Role.

5. (Optional) Expand **Advanced settings**.

6. In **Artifact store**, do one of the following:
   a. To use the default artifact store, such as the Amazon Simple Storage Service (Amazon S3) artifact bucket designated as the default, for your pipeline in the AWS Region you selected for your pipeline, choose **Default location**.
   b. If you already have an artifact store, such as an S3 artifact bucket, in the same Region as your pipeline, choose **Custom location**. In **Bucket**, choose the bucket name.

7. In **Encryption key**, do one of the following:
   a. To use the CodePipeline default AWS managed AWS KMS customer master key (CMK) to encrypt the data in the pipeline artifact store (S3 bucket), choose **Default AWS Managed Key**.
   b. To use your CMK to encrypt the data in the pipeline artifact store (S3 bucket), choose **Customer Managed Key**. In **KMS customer master key**, choose the key ID, key ARN, or alias ARN.

8. Choose **Next**.

**Step 2: Create a source stage**

1. On the **Add source stage** page, in **Source provider**, choose Bitbucket.

2. Under **Connection**, choose an existing connection, or choose **Connect to Bitbucket** to create one.
3. On the Connect to Bitbucket page, in Connection name, enter the name for the connection that you want to create. The name helps you identify this connection later.

Under Bitbucket apps, choose an app installation or choose Install a new app to create one.

4. On the app installation page, a message shows that the AWS CodeStar app is trying to connect to your Bitbucket account. Choose Grant access.
5. The connection ID for your new installation is displayed. Choose **Complete connection.**

6. In **Repository name**, choose the name of your third-party repository. In **Branch name**, choose the branch where you want your pipeline to detect source changes.

7. In **Output artifact format**, you must choose the format for your artifacts.
   - To store output artifacts from the Bitbucket action using the default method, choose **CodePipeline default.** The action accesses the files from the Bitbucket repository and stores the artifacts in a ZIP file in the pipeline artifact store.
   - To store a JSON file that contains a URL reference to the repository so that downstream actions can perform Git commands directly, choose **Full clone.** This option can only be used by CodeBuild downstream actions.

8. Choose **Next.**

9. Finish creating your next stage with the wizard.

**To review the pipeline**

1. On the **Step 5: Review** page, review your pipeline configuration. Choose **Create pipeline** to create the pipeline, or choose **Previous** to go back and edit your choices.

2. To exit the wizard without creating a pipeline, choose **Cancel.**
Create a connection (CLI)

You can use the AWS Command Line Interface (AWS CLI) to create a connection.

To do this, use the `create-connection` command.

**Important**
A connection created through the AWS CLI or AWS CloudFormation is in `PENDING` status by default. After you create a connection with the CLI or AWS CloudFormation, use the console to edit the connection to make its status `AVAILABLE`.

**To create a connection**

1. Open a terminal (Linux, macOS, or Unix) or command prompt (Windows). Use the AWS CLI to run the `create-connection` command, specifying the `--provider-type` and `--connection-name` for your connection. In this example, the third-party provider name is `Bitbucket` and the specified connection name is `MyConnection`.

   ```bash
   aws codestar-connections create-connection --provider-type Bitbucket --connection-name MyConnection
   ```

   If successful, this command returns the connection ARN information similar to the following.

   ```json
   {
     "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/aEXAMPLE-8aad-4d5d-8878-dfcab0bc441f"
   }
   ```

2. Use the console to complete the connection. For more information, see Update a pending connection.

Create a notification rule

You can use notification rules to notify users of important changes, such as when a pipeline starts execution. Notification rules specify both the events and the Amazon SNS topic that is used to send notifications. For more information, see What are notifications?

You can use the console or the AWS CLI to create notification rules for AWS CodePipeline.

**To create a notification rule (console)**

1. Sign in to the AWS Management Console and open the CodePipeline console at https://console.aws.amazon.com/codepipeline/.
2. Choose Pipelines, and then choose a pipeline where you want to add notifications.
3. On the pipeline page, choose Notify, and then choose Create notification rule. You can also go to the Settings page for the pipeline and choose Create notification rule.
4. In Notification name, enter a name for the rule.
5. In Detail type, choose Basic if you want only the information provided to Amazon EventBridge included in the notification. Choose Full if you want to include information provided to Amazon EventBridge and information that might be supplied by the CodePipeline or the notification manager.

For more information, see Understanding Notification Contents and Security.
6. In **Events that trigger notifications**, select the events for which you want to send notifications. For more information, see [Events for Notification Rules on Pipelines](#).

7. In **Targets**, do one of the following:

   - If you have already configured a resource to use with notifications, in **Choose target type**, choose either **AWS Chatbot (Slack)** or **SNS topic**. In **Choose target**, choose the name of the client (for a Slack client configured in AWS Chatbot) or the Amazon Resource Name (ARN) of the Amazon SNS topic (for Amazon SNS topics already configured with the policy required for notifications).
   
   - If you have not configured a resource to use with notifications, choose **Create target**, and then choose **SNS topic**. Provide a name for the topic after `codestar-notifications-`, and then choose **Create**.

**Note**

- If you create the Amazon SNS topic as part of creating the notification rule, the policy that allows the notifications feature to publish events to the topic is applied for you. Using a topic created for notification rules helps ensure that you subscribe only those users that you want to receive notifications about this resource.

- You cannot create an AWS Chatbot client as part of creating a notification rule. If you choose AWS Chatbot (Slack), you will see a button directing you to configure a client in AWS Chatbot. Choosing that option opens the AWS Chatbot console. For more information, see [Configure Integrations Between Notifications and AWS Chatbot](#).

- If you want to use an existing Amazon SNS topic as a target, you must add the required policy for AWS CodeStar Notifications in addition to any other policies that might exist for that topic. For more information, see [Configure Amazon SNS Topics for Notifications](#) and [Understanding Notification Contents and Security](#).

8. To finish creating the rule, choose **Submit**.

9. You must subscribe users to the Amazon SNS topic for the rule before they can receive notifications. For more information, see [Subscribe Users to Amazon SNS Topics That Are Targets](#). You can also set up integration between notifications and AWS Chatbot to send notifications to Amazon Chime chatrooms or Slack channels. For more information, see [Configure Integration Between Notifications and AWS Chatbot](#).

**To create a notification rule (AWS CLI)**

1. At a terminal or command prompt, run the `create-notification-rule` command to generate the JSON skeleton:

   ```bash
   aws codestar-notifications create-notification-rule --generate-cli-skeleton > rule.json
   ```

   You can name the file anything you want. In this example, the file is named `rule.json`.

2. Open the JSON file in a plain-text editor and edit it to include the resource, event types, and target you want for the rule. The following example shows a notification rule named `MyNotificationRule` for a pipeline named `MyDemoPipeline` in an AWS account with the ID `123456789012`. Notifications are sent with the full detail type to an Amazon SNS topic named `codestar-notifications-MyNotificationTopic` when pipeline executions start:

   ```json
   {
   "Name": "MyNotificationRule",
   "EventTypeIds": [
   "codepipeline-pipeline-pipeline-execution-started"
   ],
   "Targets": [
   ```

API Version 2015-07-09

322
Create a notification rule

```json
{
   "TargetType": "SNS",
}
```

Save the file.

3. Using the file you just edited, at the terminal or command line, run the `create-notification-rule` command again to create the notification rule:

```bash
aws codestar-notifications create-notification-rule --cli-input-json file://rule.json
```

4. If successful, the command returns the ARN of the notification rule, similar to the following:

```json
{
   "Arn": "arn:aws:codestar-notifications:us-east-1:123456789012:notificationrule/dc82df7a-EXAMPLE"
}
```
Working with actions in CodePipeline

In AWS CodePipeline, an action is part of the sequence in a stage of a pipeline. It is a task performed on the artifact in that stage. Pipeline actions occur in a specified order, in sequence or in parallel, as determined in the configuration of the stage.

CodePipeline provides support for six types of actions:

- Source
- Build
- Test
- Deploy
- Approval
- Invoke

For information about the AWS services and partner products and services you can integrate into your pipeline based on action type, see Integrations with CodePipeline action types (p. 22).

Topics
- Create and add a custom action in CodePipeline (p. 324)
- Tag a custom action in CodePipeline (p. 334)
- Invoke an AWS Lambda function in a pipeline in CodePipeline (p. 336)
- Retry a failed action in CodePipeline (p. 352)
- Manage approval actions in CodePipeline (p. 355)
- Add a cross-Region action in CodePipeline (p. 364)
- Working with variables (p. 372)

Create and add a custom action in CodePipeline

AWS CodePipeline includes a number of actions that help you configure build, test, and deploy resources for your automated release process. If your release process includes activities that are not included in the default actions, such as an internally developed build process or a test suite, you can create a custom action for that purpose and include it in your pipeline. You can use the AWS CLI to create custom actions in pipelines associated with your AWS account.

You can create custom actions for the following AWS CodePipeline action categories:

- A custom build action that builds or transforms the items
- A custom deploy action that deploys items to one or more servers, websites, or repositories
- A custom test action that configures and runs automated tests
- A custom invoke action that runs functions

When you create a custom action, you must also create a job worker that will poll CodePipeline for job requests for this custom action, execute the job, and return the status result to CodePipeline. This job...
worker can be located on any computer or resource as long as it has access to the public endpoint for CodePipeline. To easily manage access and security, consider hosting your job worker on an Amazon EC2 instance.

The following diagram shows a high-level view of a pipeline that includes a custom build action:

![Pipeline Diagram](image)

When a pipeline includes a custom action as part of a stage, the pipeline will create a job request. A custom job worker detects that request and performs that job (in this example, a custom process using third-party build software). When the action is complete, the job worker returns either a success result or a failure result. If a success result is received, the pipeline will transition the revision and its artifacts to the next action. If a failure is returned, the pipeline will not transition the revision to the next action in the pipeline.

**Note**
These instructions assume that you have already completed the steps in Getting started with CodePipeline (p. 19).

**Topics**
- Create a custom action (p. 325)
- Create a job worker for your custom action (p. 328)
- Add a custom action to a pipeline (p. 332)

## Create a custom action

**To create a custom action with the AWS CLI**

1. Open a text editor and create a JSON file for your custom action that includes the action category, the action provider, and any settings required by your custom action. For example, to create a custom build action that requires only one property, your JSON file might look like this:

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

API Version 2015-07-09
325
"executionUrlTemplate": "https://my-build-instance/job/{Config:ProjectName}/lastSuccessfulBuild/{ExternalExecutionId}/",
"configurationProperties": [{
  "name": "ProjectName",
  "required": true,
  "key": true,
  "secret": false,
  "queryable": false,
  "description": "The name of the build project must be provided when this action is added to the pipeline.",
  "type": "String"
}]
},
"inputArtifactDetails": {
  "maximumCount": integer,
  "minimumCount": integer
},
"outputArtifactDetails": {
  "maximumCount": integer,
  "minimumCount": integer
},
"tags": [
  {
    "key": "Project",
    "value": "ProjectA"
  }
]
}

This example adds tagging to the custom action by including the Project tag key and ProjectA value on the custom action. For more information about tagging resources in CodePipeline, see Tagging resources (p. 168).

There are two properties included in the JSON file, entityUrlTemplate and executionUrlTemplate. You can refer to a name in the configuration properties of the custom action within the URL templates by following the format of {Config:name}, as long as the configuration property is both required and not secret. For example, in the sample above, the entityUrlTemplate value refers to the configuration property ProjectName.

- **entityUrlTemplate**: the static link that provides information about the service provider for the action. In the example, the build system includes a static link to each build project. The link format will vary, depending on your build provider (or, if you are creating a different action type, such as test, other service provider). You must provide this link format so that when the custom action is added, the user can choose this link to open a browser to a page on your website that provides the specifics for the build project (or test environment).

- **executionUrlTemplate**: the dynamic link that will be updated with information about the current or most recent run of the action. When your custom job worker updates the status of a job (for example, success, failure, or in progress), it will also provide an externalExecutionId that will be used to complete the link. This link can be used to provide details about the run of an action.

For example, when you view the action in the pipeline, you see the following two links:
Create a custom action

1. This static link appears after you add your custom action and points to the address in entityUrlTemplate, which you specify when you create your custom action.

2. This dynamic link is updated after every run of the action and points to the address in executionUrlTemplate, which you specify when you create your custom action.

For more information about these link types, as well as RevisionURLTemplate and ThirdPartyURL, see ActionTypeSettings and CreateCustomActionType in the CodePipeline API Reference. For more information about action structure requirements and how to create an action, see CodePipeline pipeline structure reference (p. 454).

2. Save the JSON file and give it a name you can easily remember (for example, MyCustomAction.json).

3. Open a terminal session (Linux, OS X, Unix) or command prompt (Windows) on a computer where you have installed the AWS CLI.

4. Use the AWS CLI to run the `aws codepipeline create-custom-action-type` command, specifying the name of the JSON file you just created.

   For example, to create a build custom action:

   ```shell
   Important
   Be sure to include file:// before the file name. It is required in this command.
   ```

   ```shell
   aws codepipeline create-custom-action-type --cli-input-json file://MyCustomAction.json
   ```

5. This command returns the entire structure of the custom action you created, as well as the JobList action configuration property, which is added for you. When you add the custom action to a pipeline, you can use JobList to specify which projects from the provider you can poll for jobs. If you do not configure this, all available jobs will be returned when your custom job worker polls for jobs.

   For example, the preceding command might return a structure similar to the following:

   ```json
   {
     "actionType": {
       "inputArtifactDetails": {
         "maximumCount": 1,
         "minimumCount": 1
       }
     }
   }"}
Create a job worker for your custom action

Custom actions require a job worker that will poll CodePipeline for job requests for the custom action, execute the job, and return the status result to CodePipeline. The job worker can be located on any computer or resource as long as it has access to the public endpoint for CodePipeline.

There are many ways to design your job worker. The following sections provide some practical guidance for developing your custom job worker for CodePipeline.

Topics
- Choose and configure a permissions management strategy for your job worker (p. 328)
- Develop a job worker for your custom action (p. 330)
- Custom job worker architecture and examples (p. 331)

Choose and configure a permissions management strategy for your job worker

To develop a custom job worker for your custom action in CodePipeline, you will need a strategy for the integration of user and permission management.

The simplest strategy is to add the infrastructure you need for your custom job worker by creating Amazon EC2 instances with an IAM instance role, which allow you to easily scale up the resources you
need for your integration. You can use the built-in integration with AWS to simplify the interaction between your custom job worker and CodePipeline.

To set up Amazon EC2 instances

1. Learn more about Amazon EC2 and determine whether it is the right choice for your integration. For information, see Amazon EC2 - Virtual Server Hosting.
2. Get started creating your Amazon EC2 instances. For information, see Getting Started with Amazon EC2 Linux Instances.

Another strategy to consider is using identity federation with IAM to integrate your existing identity provider system and resources. This strategy is particularly useful if you already have a corporate identity provider or are already configured to support users using web identity providers. Identity federation allows you to grant secure access to AWS resources, including CodePipeline, without having to create or manage IAM users. You can use features and policies for password security requirements and credential rotation. You can use sample applications as templates for your own design.

To set up identity federation

1. Learn more about IAM identity federation. For information, see Manage Federation.
2. Review the examples in Scenarios for Granting Temporary Access to identify the scenario for temporary access that best fits the needs of your custom action.
3. Review code examples of identity federation relevant to your infrastructure, such as:
   - Identity Federation Sample Application for an Active Directory Use Case
4. Get started configuring identity federation. For information, see Identity Providers and Federation in IAM User Guide.

A third strategy to consider is to create an IAM user to use under your AWS account when running your custom action and job worker.

To set up an IAM user

1. Learn more about IAM best practices and use cases in IAM Best Practices and Use Cases.
2. Get started creating IAM users by following the steps in Creating an IAM User in Your AWS Account.

The following is an example policy you might create for use with your custom job worker. This policy is meant as an example only and is provided as-is.

```json
{
  "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/
      ]
    }
  ]
}
```
Develop a job worker for your custom action

After you’ve chosen your permissions management strategy, you should consider how your job worker will interact with CodePipeline. The following high-level diagram shows the workflow of a custom action and job worker for a build process.

2. When a pipeline is triggered by a change in its source stage (for example, when a developer commits a change), the automated release process begins. The process continues until the stage at which your custom action has been configured. When it reaches your action in this stage, CodePipeline queues a job. This job will appear if your job worker calls `PollForJobs` again to get status. Take the job detail from `PollForJobs` and pass it back to your job worker.
3. The job worker calls `AcknowledgeJob` to send CodePipeline a job acknowledgment. CodePipeline returns an acknowledgment that indicates the job worker should continue the job (`InProgress`), or, if you have more than one job worker polling for jobs and another job worker has already claimed the job, an `InvalidNonceException` error response will be returned. After the `InProgress` acknowledgment, CodePipeline waits for results to be returned.
4. The job worker initiates your custom action on the revision, and then your action runs. Along with any other actions, your custom action returns a result to the job worker. In the example of a build custom action, the action pulls artifacts from the Amazon S3 bucket, builds them, and pushes successfully built artifacts back to the Amazon S3 bucket.
5. While the action is running, the job worker can call `PutJobSuccessResult` with a continuation token (the serialization of the state of the job generated by the job worker, for example a build identifier in JSON format, or an Amazon S3 object key), as well as the `ExternalExecutionId` information that will be used to populate the link in `executionUrlTemplate`. This will update the console view of the pipeline with a working link to specific action details while it is in progress. Although not required, it is a best practice because it enables users to view the status of your custom action while it runs.

   Once `PutJobSuccessResult` is called, the job is considered complete. A new job is created in CodePipeline that includes the continuation token. This job will appear if your job worker calls `PollForJobs` again. This new job can be used to check on the state of the action, and either returns with a continuation token, or returns without a continuation token once the action is complete.

   **Note**
   If your job worker performs all the work for a custom action, you should consider breaking your job worker processing into at least two steps. The first step establishes the details page for your action. Once you have created the details page, you can serialize the state of the job worker and return it as a continuation token, subject to size limits (see Quotas in AWS CodePipeline (p. 521)). For example, you could write the state of the action into the string you use as the continuation token. The second step (and subsequent steps) of your job worker processing perform the actual work of the action. The final step returns success or failure to CodePipeline, with no continuation token on the final step.

   For more information about using the continuation token, see the specifications for `PutJobSuccessResult` in the CodePipeline API Reference.

6. Once the custom action completes, the job worker returns the result of the custom action to CodePipeline by calling one of two APIs:

   - `PutJobSuccessResult` without a continuation token, which indicates the custom action ran successfully
   - `PutJobFailureResult`, which indicates the custom action did not run successfully

   Depending on the result, the pipeline will either continue on to the next action (success) or stop (failure).

### Custom job worker architecture and examples

After you have mapped out your high-level workflow, you can create your job worker. Although the specifics of your custom action will ultimately determine what is needed for your job worker, most job workers for custom actions include the following functionality:

- Polling for jobs from CodePipeline using `PollForJobs`.
- Acknowledging jobs and returning results to CodePipeline using `AcknowledgeJob`, `PutJobSuccessResult`, and `PutJobFailureResult`.
- Retrieving artifacts from and/or putting artifacts into the Amazon S3 bucket for the pipeline. To download artifacts from the Amazon S3 bucket, you must create an Amazon S3 client that uses Signature Version 4 signing (Sig V4). Sig V4 is required for SSE-KMS.

To upload artifacts to the Amazon S3 bucket, you must additionally configure the Amazon S3 `PutObject` request to use encryption. Currently only SSE-KMS is supported for encryption. In order to know whether to use the default key or a customer-managed key to upload artifacts, your custom job worker must look at the `job data` and check the `encryption key` property. If the encryption key property is set, you should use that encryption key ID when configuring SSE-KMS. If the key is null, you use the default master key. CodePipeline uses the default Amazon S3 master key unless otherwise configured.

The following sample shows how to create the KMS parameters in Java:
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();
}

For more samples, see Specifying the AWS Key Management Service in Amazon S3 Using the AWS SDKs. For more information about the Amazon S3 bucket for CodePipeline, see CodePipeline concepts (p. 3).

A more complex example of a custom job worker is available on GitHub. This sample is open source and provided as-is.

* Sample Job Worker for CodePipeline: Download the sample from the GitHub repository.

**Add a custom action to a pipeline**

After you have a job worker, you can add your custom action to a pipeline by creating a new one and choosing it when you use the Create Pipeline wizard, by editing an existing pipeline and adding the custom action, or by using the AWS CLI, the SDKs, or the APIs.

**Note**

You can create a pipeline in the Create Pipeline wizard that includes a custom action if it is a build or deploy action. If your custom action is in the test category, you must add it by editing an existing pipeline.

**Topics**

* Add a custom action to a pipeline (console) (p. 332)
* Add a custom action to an existing pipeline (CLI) (p. 332)

**Add a custom action to a pipeline (console)**

To create a pipeline with your custom action by using the CodePipeline console, follow the steps in Create a pipeline in CodePipeline (p. 221) and choose your custom action from as many stages as you would like to test. To add your custom action to an existing pipeline by using the CodePipeline console, follow the steps in Edit a pipeline in CodePipeline (p. 231) and add your custom action to one or more stages in the pipeline.

**Add a custom action to an existing pipeline (CLI)**

You can use the AWS CLI to add a custom action to an existing pipeline.

1. Open a terminal session (Linux, macOS, or Unix) or command prompt (Windows) and run the `get-pipeline` command to copy the pipeline structure you want to edit into a JSON file. For example, for a pipeline named `MyFirstPipeline`, you would type the following command:

   ```bash
   aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json
   ```
This command returns nothing, but the file you created should appear in the directory where you ran the command.

2. Open the JSON file in any text editor and modify the structure of the file to add your custom action to an existing stage.

   **Note**
   If you want your action to run in parallel with another action in that stage, make sure you assign it the same `runOrder` value as that action.

For example, to modify the structure of a pipeline to add a stage named Build and to add a build custom action to that stage, you might modify the JSON to add the Build stage before a deployment stage as follows:

```json
{
  "name": "MyBuildStage",
  "actions": [
    {
      "inputArtifacts": [{
        "name": "MyApp"
      },
      "name": "MyBuildCustomAction",
      "actionTypeId": {
        "category": "Build",
        "owner": "Custom",
        "version": "1",
        "provider": "My-Build-Provider-Name"
      },
      "outputArtifacts": [{
        "name": "MyBuiltApp"
      }]
    },
    {
      "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"
          }
        }
      ]
    }
  ]
}
```
3. To apply your changes, run the `update-pipeline` command, specifying the pipeline JSON file, similar to the following:

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

This command returns the entire structure of the edited pipeline.

4. Open the CodePipeline console and choose the name of the pipeline you just edited.

The pipeline shows your changes. The next time you make a change to the source location, the pipeline will run that revision through the revised structure of the pipeline.

---

**Tag a custom action in CodePipeline**

Tags are key-value pairs associated with AWS resources. You can use the console or the CLI to apply tags to your custom actions in CodePipeline. For information about CodePipeline resource tagging, use cases, tag key and value constraints, and supported resource types, see Tagging resources (p. 168).

You can add, remove, and update the values of tags in a custom action. You can add up to 50 tags to each custom action.

**Topics**

- Add tags to a custom action (p. 334)
- View tags for a custom action (p. 335)
- Edit tags for a custom action (p. 335)
- Remove tags from a custom action (p. 335)

---

**Add tags to a custom action**

Follow these steps to use the AWS CLI to add a tag to a custom action. To add a tag to a custom action when you create it, see Create and add a custom action in CodePipeline (p. 324).

In these steps, we assume that you have already installed a recent version of the AWS CLI or updated to the current version. For more information, see Installing the AWS Command Line Interface.

At the terminal or command line, run the `tag-resource` command, specifying the Amazon Resource Name (ARN) of the custom action where you want to add tags and the key and value of the tag you want to add. You can add more than one tag to a custom action. For example, to tag a custom action with two tags, a tag key named `TestActionType` with the tag value of `UnitTest`, and a tag key named `ApplicationName` with the tag value of `MyApplication`:

```bash
aws codepipeline tag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:actiontype:Owner/Category/Provider/Version --tags key=TestActionType,value=UnitTest
key=ApplicationName,value=MyApplication
```
View tags for a custom action

Follow these steps to use the AWS CLI to view the AWS tags for a custom action. If no tags have been added, the returned list is empty.

At the terminal or command line, run the `list-tags-for-resource` command. For example, to view a list of tag keys and tag values for a custom action with the ARN `arn:aws:codepipeline:us-west-2:account-id:actiontype:Owner/Category/Provider/Version`:

```bash
```

If successful, this command returns information similar to the following:

```json
{
  "tags": {
    "TestActionType": "UnitTest",
    "ApplicationName": "MyApplication"
  }
}
```

Edit tags for a custom action

Follow these steps to use the AWS CLI to edit a tag for a custom action. You can change the value for an existing key or add another key. You can also remove tags from a custom action, as shown in the next section.

At the terminal or command line, run the `tag-resource` command, specifying the Amazon Resource Name (ARN) of the custom action where you want to update a tag and specify the tag key and tag value:

```bash
```

Remove tags from a custom action

Follow these steps to use the AWS CLI to remove a tag from a custom action. When you remove tags from the associated resource, the tags are deleted.

**Note**

If you delete a custom action, all tag associations are removed from the deleted custom action. You do not have to remove tags before deleting a custom action.

At the terminal or command line, run the `untag-resource` command, specifying the ARN of the custom action where you want to remove tags and the tag key of the tag you want to remove. For example, to remove a tag on a custom action with the tag key `TestActionType`:

```bash
```

If successful, this command returns nothing. To verify the tags associated with the custom action, run the `list-tags-for-resource` command.
Invoke an AWS Lambda function in a pipeline in CodePipeline

**AWS Lambda** is a compute service that lets you run code without provisioning or managing servers. You can create Lambda functions and add them as actions in your pipelines. Because Lambda allows you to write functions to perform almost any task, you can customize the way your pipeline works.

**Note**
Creating and running Lambda functions might result in charges to your AWS account. For more information, see Pricing.

Here are some ways Lambda functions can be used in pipelines:

- To create resources on demand in one stage of a pipeline using AWS CloudFormation and delete them in another stage.
- To deploy application versions with zero downtime in AWS Elastic Beanstalk with a Lambda function that swaps CNAME values.
- To deploy to Amazon ECS Docker instances.
- To back up resources before building or deploying by creating an AMI snapshot.
- To add integration with third-party products to your pipeline, such as posting messages to an IRC client.

This topic assumes you are familiar with AWS CodePipeline and AWS Lambda and know how to create pipelines, functions, and the IAM policies and roles on which they depend. This topic shows you how to:

- Create a Lambda function that tests whether a webpage was deployed successfully.
- Configure the CodePipeline and Lambda execution roles and the permissions required to run the function as part of the pipeline.
- Edit a pipeline to add the Lambda function as an action.
- Test the action by manually releasing a change.

This topic includes sample functions to demonstrate the flexibility of working with Lambda functions in CodePipeline:

- **Basic Lambda function (p. 338)**
  - Creating a basic Lambda function to use with CodePipeline.
  - Returning success or failure results to CodePipeline in the Details link for the action.
- **Sample Python function that uses an AWS CloudFormation template (p. 344)**
  - Using JSON-encoded user parameters to pass multiple configuration values to the function (get_user_params).
  - Interacting with .zip artifacts in an artifact bucket (get_template).
  - Using a continuation token to monitor a long-running asynchronous process (continue_job_later). This allows the action to continue and the function to succeed even if it exceeds a fifteen-minute runtime (a limit in Lambda).

Each sample function includes information about the permissions you must add to the role. For information about limits in AWS Lambda, see Limits in the AWS Lambda Developer Guide.

**Important**
The sample code, roles, and policies included in this topic are examples only, and are provided as-is.
Step 1: Create a pipeline

In this step, you create a pipeline to which you later add the Lambda function. This is the same pipeline you created in CodePipeline tutorials (p. 37). If that pipeline is still configured for your account and is in the same Region where you plan to create the Lambda function, you can skip this step.

To create the pipeline

1. Follow the first three steps in Tutorial: Create a simple pipeline (S3 bucket) (p. 38) to create an Amazon S3 bucket, CodeDeploy resources, and a two-stage pipeline. Choose the Amazon Linux option for your instance types. You can use any name you want for the pipeline, but the steps in this topic use MyLambdaTestPipeline.
2. On the status page for your pipeline, in the CodeDeploy action, choose Details. On the deployment details page for the deployment group, choose an instance ID from the list.
3. In the Amazon EC2 console, on the Description tab for the instance, copy the IP address in Public IP (for example, 192.0.2.4). You use this address as the target of the function in AWS Lambda.

Note
The default service role policy for CodePipeline includes the Lambda permissions required to invoke the function. However, if you have modified the default service role or selected a different one, make sure the policy for the role allows the lambda:InvokeFunction and lambda:ListFunctions permissions. Otherwise, pipelines that include Lambda actions fail.

Step 2: Create the Lambda function

In this step, you create a Lambda function that makes an HTTP request and checks for a line of text on a webpage. As part of this step, you must also create an IAM policy and Lambda execution role. For more information, see Permissions Model in the AWS Lambda Developer Guide.

To create the execution role

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose Policies, and then choose Create Policy. Choose the JSON tab, and then paste the following policy into the field.

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

API Version 2015-07-09
337

4. On the Review policy page, in Name, type a name for the policy (for example, `CodePipelineLambdaExecPolicy`). In Description, enter Enables Lambda to execute code. Choose Create Policy.

    **Note**
    These are the minimum permissions required for a Lambda function to interoperate with CodePipeline and Amazon CloudWatch. If you want to expand this policy to allow functions that interact with other AWS resources, you should modify this policy to allow the actions required by those Lambda functions.

5. On the policy dashboard page, choose Roles, and then choose Create role.

6. On the Create role page, choose AWS service. Choose Lambda, and then choose Next: Permissions.

7. On the Attach permissions policies page, select the check box next to `CodePipelineLambdaExecPolicy`, and then choose Next: Tags. Choose Next: Review.

8. On the Review page, in Role name, enter the name, and then choose Create role.

To create the sample Lambda function to use with CodePipeline

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

2. On the Functions page, choose Create function.

    **Note**
    If you see a Welcome page instead of the Lambda page, choose Get Started Now.

3. On the Create function page, choose Author from scratch. In Function name, enter a name for your Lambda function (for example, `MyLambdaFunctionForAWSCodePipeline`). In Runtime, choose Node.js 10.x.

4. Under Role, select Choose an existing role. In Existing role, choose your role, and then choose Create function.

    The detail page for your created function opens.

5. Copy the following code into the Function code box:

    **Note**
    The event object, under the CodePipeline.job key, contains the job details. For a full example of the JSON event CodePipeline returns to Lambda, see Example JSON event (p. 342).

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

exports.handler = function(event, context) {
```

API Version 2015-07-09
338
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.awsRequestId
        }
    };
    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;
        });
    });
};
Step 3: Add the Lambda function to a pipeline in the CodePipeline console

In this step, you add a new stage to your pipeline, and then add a Lambda action that calls your function to that stage.

To add a stage

2. On the Welcome page, choose the pipeline you created.
3. On the pipeline view page, choose Edit.
4. On the Edit page, choose + Add stage to add a stage after the deployment stage with the CodeDeploy action. Enter a name for the stage (for example, LambdaStage), and choose Add stage.

   **Note**
   You can also choose to add your Lambda action to an existing stage. For demonstration purposes, we are adding the Lambda function as the only action in a stage to allow you to easily view its progress as artifacts progress through a pipeline.

5. Choose + Add action group. In Edit action, in Action name, enter a name for your Lambda action (for example, MyLambdaAction). In Provider, choose AWS Lambda. In Function name, choose or enter the name of your Lambda function (for example, MyLambdaFunctionForAWSCodePipeline). In User parameters, specify the IP address for the Amazon EC2 instance you copied earlier (for example, http://192.0.2.4), and then choose Done.

6. Leave Handler at the default value, and leave Role at the default, CodePipelineLambdaExecRole.
7. In Basic settings, for Timeout, enter 20 seconds.
8. Choose Save.
Step 4: Test the pipeline with the Lambda function

To test the function, release the most recent change through the pipeline.

**To use the console to run the most recent version of an artifact through a pipeline**

1. On the pipeline details page, choose Release change. This runs the most recent revision available in each source location specified in a source action through the pipeline.
2. When the Lambda action is complete, choose the Details link to view the log stream for the function in Amazon CloudWatch, including the billed duration of the event. If the function failed, the CloudWatch log provides information about the cause.

**Step 5: Next steps**

Now that you've successfully created a Lambda function and added it as an action in a pipeline, you can try the following:

- Add more Lambda actions to your stage to check other webpages.
- Modify the Lambda function to check for a different text string.
- Explore Lambda functions and create and add your own Lambda functions to pipelines.

After you have finished experimenting with the Lambda function, consider removing it from your pipeline, deleting it from AWS Lambda, and deleting the role from IAM to avoid possible charges. For more information, see Edit a pipeline in CodePipeline (p. 231), Delete a pipeline in CodePipeline (p. 250), and Deleting Roles or Instance Profiles.

Example JSON event

The following example shows a sample JSON event sent to Lambda by CodePipeline. The structure of this event is similar to the response to the GetJobDetails API, but without the actionTypeId and pipelineContext data types. Two action configuration details, FunctionName and UserParameters, are included in both the JSON event and the response to the GetJobDetails API. The values in red italic text are examples or explanations, not real values.

```
{
    "CodePipeline.job": {
        "id": "11111111-abcd-1111-abcd-111111abcdef",
        "accountId": "111111111111",
        "data": {
            "actionConfiguration": {
                "configuration": {
                    "FunctionName": "MyLambdaFunctionForAWSCodePipeline",
                    "UserParameters": "some-input-such-as-a-URL"
                }
            }
        }
    }
}
```
Additional sample functions

The following sample Lambda functions demonstrate additional functionality you can use for your pipelines in CodePipeline. To use these functions, you might have to modify the policy for the Lambda execution role, as noted in the introduction for each sample.

Topics

• Sample Python function that uses an AWS CloudFormation template (p. 344)
Sample Python function that uses an AWS CloudFormation template

The following sample shows a function that creates or updates a stack based on a supplied AWS CloudFormation template. The template creates an Amazon S3 bucket. It is for demonstration purposes only, to minimize costs. Ideally, you should delete the stack before you upload anything to the bucket. If you upload files to the bucket, you cannot delete the bucket when you delete the stack. You must manually delete everything in the bucket before you can delete the bucket itself.

This Python sample assumes you have a pipeline that uses an Amazon S3 bucket as a source action, or that you have access to a versioned Amazon S3 bucket you can use with the pipeline. You create the AWS CloudFormation template, compress it, and upload it to that bucket as a .zip file. You must then add a source action to your pipeline that retrieves this .zip file from the bucket.

Note
When Amazon S3 is the source provider for your pipeline, you may zip your source file or files into a single .zip and upload the .zip to your source bucket. You may also upload a single unzipped file; however, downstream actions that expect a .zip file will fail.

This sample demonstrates:

- The use of JSON-encoded user parameters to pass multiple configuration values to the function (get_user_params).
- The interaction with .zip artifacts in an artifact bucket (get_template).
- The use of a continuation token to monitor a long-running asynchronous process (continue_job_later). This allows the action to continue and the function to succeed even if it exceeds a fifteen-minute runtime (a limit in Lambda).

To use this sample Lambda function, the policy for the Lambda execution role must have Allow permissions in AWS CloudFormation, Amazon S3, and CodePipeline, as shown in this sample policy:

```json
{
  "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": [
        "cloudformation:DeleteStack"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```
To create the AWS CloudFormation template, open any plain-text editor and copy and paste the following code:

```json
{
  "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"
    }
  }
}
```

Save this as a JSON file with the name `template.json` in a directory named `template-package`. Create a compressed (.zip) file of this directory and file named `template-package.zip`, and upload the compressed file to a versioned Amazon S3 bucket. If you already have a bucket configured for your pipeline, you can use it. Next, edit your pipeline to add a source action that retrieves the .zip file. Name the output for this action `MyTemplate`. For more information, see Edit a pipeline in CodePipeline (p. 231).

**Note**

The sample Lambda function expects these file names and compressed structure. However, you can substitute your own AWS CloudFormation template for this sample. If you use your own template, make sure you modify the policy for the Lambda execution role to allow any additional functionality required by your AWS CloudFormation template.

To add the following code as a function in Lambda

1. Open the Lambda console and choose Create function.
2. On the Create function page, choose Author from scratch. In Function name, enter a name for your Lambda function.
3. In Runtime, choose Python 2.7.
4. Under Role, select Choose an existing role. In Existing role, choose your role, and then choose Create function.

The detail page for your created function opens.

5. Copy the following code into the Function code box:

```python
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 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."

```python
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 "{}"'.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
    """

    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.
    """

    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

```python
try:
    # Extract the Job ID
    job_id = event['CodePipeline.job']['id']

    # Extract the Job Data
    job_data = event['CodePipeline.job']['data']

    # Extract the params
    params = get_user_params(job_data)

    # Get the list of artifacts passed to the function
    artifacts = job_data['inputArtifacts']

    stack = params['stack']
    artifact = params['artifact']
    template_file = params['file']

    if 'continuationToken' in job_data:
        # If we're continuing then the create/update has already been triggered
        # we just need to check if it has finished.
        check_stack_update_status(job_id, stack)
    else:
        # Get the artifact details
        artifact_data = find_artifact(artifacts, artifact)
        # Get S3 client to access artifact with
        s3 = setup_s3_client(job_data)
        # Get the JSON template file out of the artifact
        template = get_template(s3, artifact_data, template_file)
        # Kick off a stack update or create
        start_update_or_create(job_id, stack, template)

except Exception as e:
    # If any other exceptions which we didn't expect are raised
    # then fail the job and log the exception message.
    print('Function failed due to exception.
    print(e)
    traceback.print_exc()
    put_job_failure(job_id, 'Function exception: ' + str(e))

    print('Function complete.')
    return "Complete."
```

6. Leave **Handler** at the default value, and leave **Role** at the default, **CodePipelineLambdaExecRole**.

7. In **Basic settings**, for **Timeout**, replace the default of 3 seconds with **20**.

8. Choose **Save**.

9. From the CodePipeline console, edit the pipeline to add the function as an action in a stage in your pipeline. In **UserParameters**, you must provide a JSON string with three parameters:
   - Stack name
   - AWS CloudFormation template name and path to the file
   - Application name.

   Use curly brackets ({{}}) and separate the parameters with commas. For example, to create a stack named **MyTestStack**, for a pipeline with the input artifact **MyTemplate**, in **UserParameters**, enter: 
   ```json
   {"stack":"MyTestStack","file":"template-package/template.json","artifact":"MyTemplate"}
   ```
Note
Even though you have specified the input artifact in UserParameters, you must also specify this input artifact for the action in Input artifacts.

10. Save your changes to the pipeline, and then manually release a change to test the action and Lambda function.

Retry a failed action in CodePipeline

In AWS CodePipeline, an action is a task performed on an artifact in a stage. A failure is an action in a stage that is not completed successfully. You can use the CLI to manually retry the failed action before the stage completes (while other actions are still in progress). If a stage completes with one or more failed actions, the stage fails, and the pipeline execution does not transition to the next stage in the pipeline.

You can retry the latest failed actions in a stage without having to run a pipeline again from the beginning. You do this by retrying the stage that contains the actions. You can retry a stage immediately after any of actions fail. All actions that are still in progress continue their work, and failed actions are triggered once again.

If you are using the console to view a pipeline, a Retry button appears on the stage where the failed actions can be retried.

If you are using the AWS CLI, you can use the get-pipeline-state command to determine whether any actions have failed.

Note
In the following cases, you may not be able to retry actions:

- The overall pipeline structure changed after an action failed.
- Another retry attempt in the stage is already in progress.

Topics
- Retry failed actions (console) (p. 353)
- Retry failed actions (CLI) (p. 353)
Retry failed actions (console)


   The names of all pipelines associated with your AWS account are displayed.

2. In **Name**, choose the name of the pipeline.

3. Locate the stage with the failed action, and then choose **Retry**.

   **Note**
   To identify which actions in the stage can be retried, hover over the **Retry** button.

   If all retried actions in the stage are completed successfully, the pipeline continues to run.

Retry failed actions (CLI)

Option 1: Retry failed actions using parameters for the CLI command

To use the AWS CLI to retry failed actions, you run the `retry-stage-execution` command with the following parameters:

```
--pipeline-name <value>
--stage-name <value>
--pipeline-execution-id <value>
--retry-mode <value>
```

**Note**

The only value you can use for `retry-mode` is `FAILED_ACTIONS`.

- At a terminal (Linux, macOS, or Unix) or command prompt (Windows), run the `retry-stage-execution` command, as shown in the following example for a pipeline named `MyPipeline`.

```
aws codepipeline retry-stage-execution --pipeline-name MyPipeline --stage-name Deploy --pipeline-execution-id b59babff-5f34-EXAMPLE --retry-mode FAILED_ACTIONS
```

The output returns the execution ID:

```
{
  "pipelineExecutionId": "b59babff-5f34-EXAMPLE"
}
```

Option 2: Retry failed actions using a JSON input file for the CLI command

You can also run the command with a JSON input file. You first create a JSON file that identifies the pipeline, the stage that contains the failed actions, and the latest pipeline execution in that stage. You then run the `retry-stage-execution` command with the `--cli-input-json` parameter. To retrieve the details you need for the JSON file, it's easiest to use the `get-pipeline-state` command.

1. At a terminal (Linux, macOS, or Unix) or command prompt (Windows), run the `get-pipeline-state` command on a pipeline. For example, for a pipeline named `MyFirstPipeline`, you would type something similar to the following:

```
aws codepipeline get-pipeline-state --name MyFirstPipeline
```
The response to the command includes pipeline state information for each stage. In the following example, the response indicates that one or more actions failed in the Staging stage:

```json
{
    "updated": 1427245911.525,
    "created": 1427245911.525,
    "pipelineVersion": 1,
    "pipelineName": "MyFirstPipeline",
    "stageStates": [
        
        "actionStates": [...],
        "stageName": "Source",
        "latestExecution": {
            "pipelineExecutionId": "9811f7cb-7cf7-SUCCESS",
            "status": "Succeeded"
        }
    ],
    "stageStates": [
        
        "actionStates": [...],
        "stageName": "Staging",
        "latestExecution": {
            "pipelineExecutionId": "3137f7cb-7cf7-EXAMPLE",
            "status": "Failed"
        }
    ]
}
```

2. In a plain-text editor, create a file where you will record the following, in JSON format:

- The name of the pipeline that contains the failed actions
- The name of the stage that contains the failed actions
- The ID of the latest pipeline execution in the stage
- The retry mode. (Currently, the only supported value is FAILED_ACTIONS)

For the preceding MyFirstPipeline example, your file would look something like this:

```json
{
    "pipelineName": "MyFirstPipeline",
    "stageName": "Staging",
    "pipelineExecutionId": "3137f7cb-7cf7-EXAMPLE",
    "retryMode": "FAILED_ACTIONS"
}
```

3. Save the file with a name like `retry-failed-actions.json`.

4. Call the file you created when you run the `retry-stage-execution` command. For example:

```bash
Important
Be sure to include file:// before the file name. It is required in this command.

aws codepipeline retry-stage-execution --cli-input-json file://retry-failed-actions.json
```

5. To view the results of the retry attempt, either open the CodePipeline console and choose the pipeline that contains the actions that failed, or use the `get-pipeline-state` command again. For more information, see View pipeline details and history in CodePipeline (p. 237).
Manage approval actions in AWS CodePipeline

In AWS CodePipeline, you can add an approval action to a stage in a pipeline at the point where you want the pipeline execution to stop so that someone with the required AWS Identity and Access Management permissions can approve or reject the action.

If the action is approved, the pipeline execution resumes. If the action is rejected—or if no one approves or rejects the action within seven days of the pipeline reaching the action and stopping—the result is the same as an action failing, and the pipeline execution does not continue.

You might use manual approvals for these reasons:

- You want someone to perform a code review or change management review before a revision is allowed into the next stage of a pipeline.
- You want someone to perform manual quality assurance testing on the latest version of an application, or to confirm the integrity of a build artifact, before it is released.
- You want someone to review new or updated text before it is published to a company website.

Configuration options for manual approval actions in CodePipeline

CodePipeline provides three configuration options you can use to tell approvers about the approval action.

**Publish Approval Notifications** You can configure an approval action to publish a message to an Amazon Simple Notification Service topic when the pipeline stops at the action. Amazon SNS delivers the message to every endpoint subscribed to the topic. You must use a topic created in the same AWS Region as the pipeline that will include the approval action. When you create a topic, we recommend you give it a name that will identify its purpose, in formats such as MyFirstPipeline-us-east-2-approval.

When you publish approval notifications to Amazon SNS topics, you can choose from formats such as email or SMS recipients, SQS queues, HTTP/HTTPS endpoints, or AWS Lambda functions you invoke using Amazon SNS. For information about Amazon SNS topic notifications, see the following topics:

- What Is Amazon Simple Notification Service?
- Create a Topic in Amazon SNS
- Sending Amazon SNS Messages to Amazon SQS Queues
- Subscribing a Queue to an Amazon SNS Topic
- Sending Amazon SNS Messages to HTTP/HTTPS Endpoints
- Invoking Lambda Functions Using Amazon SNS Notifications

For the structure of the JSON data generated for an approval action notification, see [JSON data format for manual approval notifications in CodePipeline](p. 364).

**Specify a URL for Review** As part of the configuration of the approval action, you can specify a URL to be reviewed. The URL might be a link to a web application you want approvers to test or a page with more information about your approval request. The URL is included in the notification that is published to the Amazon SNS topic. Approvers can use the console or CLI to view it.

**Enter Comments for Approvers** When you create an approval action, you can also add comments that are displayed to those who receive the notifications or those who view the action in the console or CLI response.
No Configuration Options You can also choose not to configure any of these three options. You might not need them if, for example, you can notify someone directly that the action is ready for their review, or you simply want the pipeline to stop until you decide to approve the action yourself.

Setup and workflow overview for approval actions in CodePipeline

The following is an overview for setting up and using manual approvals.

1. You grant the IAM permissions required for approving or rejecting approval actions to one or more IAM users in your organization.
2. (Optional) If you are using Amazon SNS notifications, you ensure that the service role you use in your CodePipeline operations has permission to access Amazon SNS resources.
3. (Optional) If you are using Amazon SNS notifications, you create an Amazon SNS topic and add one or more subscribers or endpoints to it.
4. When you use the AWS CLI to create the pipeline or after you have used the CLI or console to create the pipeline, you add an approval action to a stage in the pipeline.

If you are using notifications, you include the Amazon Resource Name (ARN) of the Amazon SNS topic in the configuration of the action. (An ARN is a unique identifier for an Amazon resource. ARNs for Amazon SNS topics are structured like `arn:aws:sns:us-east-2:80398EXAMPLE:MyApprovalTopic`. For more information, see Amazon Resource Names (ARNs) and AWS Service Namespaces in the Amazon Web Services General Reference.)

5. The pipeline stops when it reaches the approval action. If an Amazon SNS topic ARN was included in the configuration of the action, a notification is published to the Amazon SNS topic, and a message is delivered to any subscribers to the topic or subscribed endpoints, with a link to review the approval action in the console.
6. An approver examines the target URL and reviews comments, if any.
7. Using the console, CLI, or SDK, the approver provides a summary comment and submits a response:
   - Approved: The pipeline execution resumes.
   - Rejected: The stage status is changed to "Failed" and the pipeline execution does not resume.

   If no response is submitted within seven days, the action is marked as "Failed."

Grant approval permissions to an IAM user in CodePipeline

Before IAM users in your organization can approve or reject approval actions, they must be granted permissions to access pipelines and to update the status of approval actions. You can grant permission to access all pipelines and approval actions in your account by attaching the `AWSCodePipelineApproverAccess` managed policy to an IAM user, role, or group; or you can grant limited permissions by specifying the individual resources that can be accessed by an IAM user, role, or group.

Note
The permissions described in this topic grant very limited access. To enable a user, role, or group to do more than approve or reject approval actions, you can attach other managed policies. For information about the managed policies available for CodePipeline, see AWS managed (predefined) policies for CodePipeline (p. 428).
Grant approval permission to all pipelines and approval actions

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Groups, Roles, or Users.
3. Choose the group, role or IAM user to grant permissions to.
4. Choose the Permissions tab.
5. Choose Add permissions, and then choose Attach existing policies directly.
6. Select the check box next to AWSCodePipelineApproverAccess managed policy, and then choose Next: Review.
7. Choose Add permissions.

Specify approval permission for specific pipelines and approval actions

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

   Important
   Make sure you are signed in to the AWS Management Console with the same account information you used in Getting started with CodePipeline (p. 19).

2. In the navigation pane, choose Groups or Users, as appropriate.
3. Browse to and choose the user or group you want to change.
4. Do one of the following:
   - If you chose Groups, choose the Permissions tab, and expand Inline Policies. If no inline policies have been created yet, choose click here.
     Choose Custom Policy, and then choose Select.
     In Policy Name, enter a name for this policy. Continue to the next step to paste the policy in the Policy Document box.
   - If you chose Users, choose the Permissions tab, and choose Add inline policy. Choose the JSON tab. Continue to the next step to paste the policy.

5. Paste the policy into the policy box. Specify the individual resources an IAM user can access. For example, the following policy grants users the authority to approve or reject only the action named MyApprovalAction in the MyFirstPipeline pipeline in the US East (Ohio) Region (us-east-2):

```
{
   "Version": "2012-10-17",
   "Statement": [
   {
      "Action": ["codepipeline:ListPipelines"],
      "Resource": ["*"],
      "Effect": "Allow"
   },
   {
      "Action": [
      "codepipeline:GetPipeline",
      "codepipeline:GetPipelineState",
      "codepipeline:GetPipelineExecution"
```

API Version 2015-07-09
357
Grant Amazon SNS permissions to a service role

If you plan to use Amazon SNS to publish notifications to topics when approval actions require review, the service role you use in your CodePipeline operations must be granted permission to access the Amazon SNS resources. You can use the IAM console to add this permission to your service role.

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

   **Important**
   Make sure you are signed in to the AWS Management Console with the same account information you used in Getting started with CodePipeline (p. 19).

2. In the IAM console, in the navigation pane, choose Roles.
3. Choose the name of the service role you use in your CodePipeline operations.
4. On the Permissions tab, in the Inline Policies area, choose Create Role Policy.

   –or–

   If the Create Role Policy button is not available, expand the Inline Policies area, and then choose click here.

5. On the Set Permissions page, choose Custom Policy, and then choose Select.
6. On the Review Policy page, in the Policy Name field, type a name to identify this policy, such as SNSPublish.
7. Paste the following into the Policy Document field:

   ```json
   {
     "Statement": [
       {
         "Effect": "Allow",
       },
       {
         "Action": [
           "codepipeline:PutApprovalResult"
         ],
         "Effect": "Allow",
         "Resource": "arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline/MyApprovalStage/MyApprovalAction"
       }
     ]
   }
   ```

**Note**

The codepipeline:ListPipelines permission is required only if IAM users need to access the CodePipeline dashboard to view this list of pipelines. If console access is not required, you can omit codepipeline:ListPipelines.

6. Do one of the following:

   • If you chose Groups, choose Validate Policy. Correct any errors displayed in a red box at the top of the page. Choose Apply Policy.
   • If you chose Users, choose Review policy.

   In Name, enter a name for this policy. Choose Create policy.
Add a manual approval action to a pipeline in CodePipeline

You can add an approval action to a stage in a CodePipeline pipeline at the point where you want the pipeline to stop so someone can manually approve or reject the action.

Add a manual approval action to a CodePipeline pipeline (console)

You can use the CodePipeline console to add an approval action to an existing CodePipeline pipeline. You must use the AWS CLI if you want to add approval actions when you create a new pipeline.

2. In Name, choose the pipeline.
3. On the pipeline details page, choose Edit.
4. If you want to add an approval action to a new stage, choose + Add stage at the point in the pipeline where you want to add an approval request, and enter a name for the stage. On the Add stage page, in Stage name, enter your new stage name. For example, add a new stage and name it Manual_Approval.

If you want to add an approval action to an existing stage, choose Edit stage.
5. In the stage where you want to add the approval action, choose + Add action group.
6. On the Edit action page, do the following:
   1. In Action name, enter a name to identify the action.
   2. In Action provider, under Approval, choose Manual approval.
   3. (Optional) In SNS topic ARN, choose the name of the topic to be used to send notifications for the approval action.
4. (Optional) In **URL for review**, enter the URL of the page or application you want the approver to examine. Approvers can access this URL through a link included in the console view of the pipeline.

5. (Optional) In **Comments**, enter any other information you want to share with the reviewer.

Your completed page might look similar to the following:

6. Choose **Save**.

**Add a manual approval action to a CodePipeline pipeline (CLI)**

You can use the CLI to add an approval action to an existing pipeline or when you create a pipeline. You do this by including an approval action, with the Manual approval type, in a stage you are creating or editing.

For more information about creating and editing pipelines, see Create a pipeline in CodePipeline (p. 221) and Edit a pipeline in CodePipeline (p. 231).

To add a stage to a pipeline that includes only an approval action, you would include something similar to the following example when you create or update the pipeline.

**Note**

The configuration section is optional. This is just a portion, not the entire structure, of the file. For more information, see CodePipeline pipeline structure reference (p. 454).

```json
{
  "name": "MyApprovalStage",
  "actions": [
    {
      ...
    }
  ]
}
```
Add a manual approval action

```json
"name": "MyApprovalAction",
"actionTypeId": {
  "category": "Approval",
  "owner": "AWS",
  "version": "1",
  "provider": "Manual"
},
"inputArtifacts": [],
"outputArtifacts": [],
"configuration": {
  "ExternalEntityLink": "http://example.com",
  "CustomData": "The latest changes include feedback from Bob."
}
```

If the approval action is in a stage with other actions, the section of your JSON file that contains the stage might look similar instead to the following example.

**Note**
The configuration section is optional. This is just a portion, not the entire structure, of the file. For more information, see CodePipeline pipeline structure reference (p. 454).

```json
{
  "name": "Production",
  "actions": [
    {
      "inputArtifacts": [],
      "name": "MyApprovalAction",
      "actionTypeId": {
        "category": "Approval",
        "owner": "AWS",
        "version": "1",
        "provider": "Manual"
      },
      "outputArtifacts": [],
      "configuration": {
        "ExternalEntityLink": "http://example.com",
        "CustomData": "The latest changes include feedback from Bob."
      }
    },
    {
      "inputArtifacts": [
        {
          "name": "MyApp"
        }
      ],
      "name": "MyDeploymentAction",
      "actionTypeId": {
        "category": "Deploy",
        "owner": "AWS",
        "version": "1",
        "provider": "CodeDeploy"
      },
      "outputArtifacts": [],
      "configuration": {
        "ApplicationName": "MyDemoApplication",
        "DeploymentGroupName": "MyProductionFleet"
      }
    }
  ],
  "runOrder": 1
}
```
Approve or reject an approval action in CodePipeline

When a pipeline includes an approval action, the pipeline execution stops at the point where the action has been added. The pipeline won’t resume unless someone manually approves the action. If an approver rejects the action, or if no approval response is received within seven days of the pipeline stopping for the approval action, the pipeline status becomes "Failed."

If the person who added the approval action to the pipeline configured notifications, you might receive an email that looks similar to the following:

Hello,

The following Approval action is waiting for your response:

--Pipeline Details--

Pipeline name: MyFirstPipeline
Stage name: MyApprovalStage
Action name: MyApprovalAction
Region: us-east-1

--Approval Details--

Content to review: http://example.com
Approve or reject: https://console.aws.amazon.com/codepipeline/home?region=us-east-1#view/MyFirstPipeline/MyApprovalStage/MyApprovalAction/approve/1e2b3c4d-5e7f-4e7a-a67f-8b93485f8a01
Additional Information: The latest changes include feedback from Bob.
Deadline: This review request will expire on 2016-09-21T17:51Z

Sincerely,
Amazon Web Services

If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-east-1.amazonaws.com/unsubscribe.html?subscriptionArn=arn:aws:sns:us-east-1:890888EXAMPLE:MyApprovalTopic1a2b3c4d-5e7f-4e7a-a67f-8b93485f8a01&endpoint=john.doe@example.com

Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at https://aws.amazon.com/support

Approve or reject an approval action (console)

If you receive a notification that includes a direct link to an approval action, choose the Approve or reject link, sign in to the console, and then continue with step 7 here. Otherwise, follow all of these steps.

2. On the All Pipelines page, choose the name of the pipeline.
3. Locate the stage with the approval action.
4. Hover over the information icon to view the comments and URL, if any. The message also displays the URL of content for you to review, if one was included.
5. If a URL was provided, choose the Manual approval link in the action to open the target webpage, and then review the content.

6. Return to the pipeline details view, and then choose Review.

7. In the Approve or reject the revision window, enter review comments, such as why you are approving or rejecting the action, and then choose Approve or Reject.

Approve or reject an approval request (CLI)

To use the CLI to respond to an approval action, you must first use the get-pipeline-state command to retrieve the token associated with latest execution of the approval action.

1. At a terminal (Linux, macOS, or Unix) or command prompt (Windows), run the get-pipeline-state command on the pipeline that contains the approval action. For example, for a pipeline named MyFirstPipeline, enter the following:

   ```bash
codepipeline get-pipeline-state --name MyFirstPipeline
   ```

2. In the response to the command, locate the token value, which appears in latestExecution in the actionStates section for the approval action, as shown here:

   ```json
   { 
   "created": 1467929497.204,
   "pipelineName": "MyFirstPipeline",
   "pipelineVersion": 1,
   "stageStates": [ 
   { 
   "actionStates": [ 
   { 
   "actionName": "MyApprovalAction",
   "currentRevision": { 
   "created": 1467929497.204,
   "revisionChangeId": "CEM7d6Tp7zfe1USLCPPwo234xEXAMPLE",
   "revisionId": "HYGp7zmwCPPwo23xCMDTeqIlEXAMPLE"
   },
   "latestExecution": { 
   "lastUpdatedBy": "arn:aws:iam::123456789012:user/Bob",
   "summary": "The new design needs to be reviewed before release.",
   "token": "l2b3c4d-573f-4ea7-a67E-EXAMPLETOKEN"
   }
   } 
   ] 
   } 
   // More content might appear here
   }
   ```

3. In a plain-text editor, create a file where you add the following, in JSON format:

   - The name of the pipeline that contains the approval action.
   - The name of the stage that contains the approval action.
   - The name of the approval action.
   - The token value you collected in the previous step.
   - Your response to the action (Approved or Rejected). The response must be capitalized.
   - Your summary comments.

   For the preceding MyFirstPipeline example, your file should look like this:

   ```json
   { 
   "pipelineName": "MyFirstPipeline",
   ```
4. Save the file with a name like `approvalstage-approved.json`.

5. Run the `put-approval-result` command, specifying the name of the approval JSON file, similar to the following:

```
aws codepipeline put-approval-result --cli-input-json file://approvalstage-approved.json
```

## JSON data format for manual approval notifications in CodePipeline

For approval actions that use Amazon SNS notifications, JSON data about the action is created and published to Amazon SNS when the pipeline stops. You can use the JSON output to send messages to Amazon SQS queues or invoke functions in AWS Lambda.

**Note**

This guide does not address how to configure notifications using JSON. For information, see [Sending Amazon SNS Messages to Amazon SQS Queues](https://docs.aws.amazon.com/sns/latest/dg/sending-sqs.html) and [Invoking Lambda Functions Using Amazon SNS Notifications](https://docs.aws.amazon.com/lambda/latest/dg/sns-invoking-lambda.html) in the *Amazon SNS Developer Guide*.

The following example shows the structure of the JSON output available with CodePipeline approvals:

```
{
    "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."
    }
}
```

---

## Add a cross-Region action in CodePipeline

AWS CodePipeline includes a number of actions that help you configure build, test, and deploy resources for your automated release process. You can add actions to your pipeline that are in an AWS Region...
different from your pipeline. When an AWS service is the provider for an action, and this action type/provider type are in a different AWS Region from your pipeline, this is a cross-Region action.

**Note**
Certain action types in CodePipeline may only be available in certain AWS Regions. Also note that there may be AWS Regions where an action type is available, but a specific AWS provider for that action type is not available.

You can use the console, AWS CLI, or AWS CloudFormation to add cross-Region actions in pipelines. When you create or edit a pipeline, you must have an artifact bucket in the pipeline Region and then you must have one artifact bucket per Region where you plan to execute an action. For more information about the ArtifactStores parameter, see CodePipeline pipeline structure reference (p. 454).

**Note**
CodePipeline handles the copying of artifacts from one AWS Region to the other Regions when performing cross-region actions.

If you use the console to create a pipeline or cross-Region actions, default artifact buckets are configured by CodePipeline in the Regions where you have actions. When you use the AWS CLI, AWS CloudFormation, or an SDK to create a pipeline or cross-Region actions, you provide the artifact bucket for each Region where you have actions.

**Note**
You must create the artifact bucket and encryption key in the same AWS Region as the cross-Region action and in the same account as your pipeline.

You cannot create cross-Region actions for the following action types:

- Source actions
- Third-party actions
- Custom actions

When a pipeline includes a cross-Region action as part of a stage, CodePipeline replicates only the input artifacts of the cross-Region action from the pipeline Region to the action’s Region.

**Note**
The pipeline Region and the Region where your CloudWatch Events change detection resources are maintained remain the same. The Region where your pipeline is hosted does not change.

**Manage cross-Region actions in a pipeline (console)**

You can use the CodePipeline console to add a cross-Region action to an existing pipeline. To create a new pipeline with cross-Region actions using the Create pipeline wizard, see Create a pipeline (console) (p. 222).

In the console, you create a cross-Region action in a pipeline stage by choosing the action provider and the Region field, which lists the resources you have created in that region for that provider. When you add a cross-Region action, CodePipeline uses a separate artifact bucket in the action’s region. For more information about cross-Region artifact buckets, see CodePipeline pipeline structure reference (p. 454).

**Add a cross-Region action to a pipeline stage (console)**

Use the console to add a cross-Region action to a pipeline.

**Note**
If the pipeline is running when changes are saved, that execution does not complete.

**To add a cross-Region action**

2. Select your pipeline, and then choose **Edit**.
3. At the bottom of the diagram, choose *+ Add stage* if you are adding a new stage, or choose *Edit stage* if you want to add the action to an existing stage.
4. On **Edit: <Stage>**, choose *+ Add action group* to add a serial action. Or choose *+ Add action* to add a parallel action.
5. On the **Edit action** page:
   a. In **Action name**, enter a name for the cross-Region action.
   b. In **Action provider**, choose the action provider.
   c. In **Region**, choose the AWS Region where you have created or plan to create the resource for the action. When the Region is selected, the available resources for that Region are listed for selection. The **Region** field designates where the AWS resources are created for this action type and provider type. This field only displays for actions where the action provider is an AWS service. The **Region** field defaults to the same AWS Region as your pipeline.
   d. In **Input artifacts** choose the appropriate input from the previous stage. For example, if the previous stage is a source stage, choose **SourceArtifact**.
   e. Complete all the required fields for the action provider you are configuring.
   f. In **Output artifacts** choose the appropriate output to the next stage. For example, if the next stage is a deployment stage, choose **BuildArtifact**.
   g. Choose **Save**.
6. On **Edit: <Stage>**, choose **Done**.
7. Choose **Save**.

**Edit a cross-Region action in a pipeline stage (console)**

Use the console to edit an existing cross-Region action in a pipeline.

**Note**
If the pipeline is running when changes are saved, that execution does not complete.

**To edit a cross-Region action**
2. Select your pipeline, and then choose **Edit**.
3. Choose **Edit stage**.
4. On **Edit: <Stage>**, choose the icon to edit an existing action.
5. On the **Edit action** page, make changes to the fields, as appropriate.
6. On **Edit: <Stage>**, choose **Done**.
7. Choose **Save**.

### Delete a cross-Region action from a pipeline stage (console)

Use the console to delete an existing cross-Region action from a pipeline.

**Note**
If the pipeline is running when changes are saved, that execution does not complete.

**To delete a cross-Region action**

2. Select your pipeline, and then choose **Edit**.
3. Choose **Edit stage**.
4. On **Edit: <Stage>**, choose the icon to delete an existing action.
5. On **Edit: <Stage>**, choose **Done**.
6. Choose **Save**.

### Add a cross-Region action to a pipeline (CLI)

You can use the AWS CLI to add a cross-Region action to an existing pipeline.

To create a cross-Region action in a pipeline stage with the AWS CLI, you add the configuration action along with an optional `region` field. You must also have already created an artifact bucket in the action's region. Instead of providing the `artifactStore` parameter of the single-region pipeline, you use the `artifactStores` parameter to include a listing of each Region's artifact bucket.

**Note**
In this walkthrough and its examples, **RegionA** is the Region where the pipeline is created. It has access to the **RegionA** Amazon S3 bucket used to store pipeline artifacts and the service role used by CodePipeline. **RegionB** is the region where the CodeDeploy application, deployment group, and service role used by CodeDeploy are created.

**Prerequisites**

You must have created the following:

- A pipeline in **RegionA**.
- An Amazon S3 artifact bucket in **RegionB**.
- The resources for your action, such as your CodeDeploy application and deployment group for a cross-Region deploy action, in **RegionB**.

**Add a cross-Region action to a pipeline (CLI)**

Use the AWS CLI to add a cross-Region action to a pipeline.

**To add a cross-Region action**

1. For a pipeline in **RegionA**, run the `get-pipeline` command to copy the pipeline structure into a JSON file. For example, for a pipeline named **MyFirstPipeline**, run the following command:
aws codepipeline get-pipeline --name MyFirstPipeline >pipeline.json

This command returns nothing, but the file you created should appear in the directory where you ran the command.

2. Add the `region` field to add a new stage with your cross-Region action that includes the Region and resources for your action. The following JSON sample adds a Deploy stage with a cross-Region deploy action where the provider is CodeDeploy, in a new region `us-east-1`.

```json
{
    "name": "Deploy",
    "actions": [
        {
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "name": "Deploy",
            "region": "RegionB",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "version": "1",
                "provider": "CodeDeploy"
            },
            "outputArtifacts": [],
            "configuration": {
                "ApplicationName": "name",
                "DeploymentGroupName": "name"
            },
            "runOrder": 1
        }
    ]
}
```

3. In the pipeline structure, remove the `artifactStore` field and add the `artifactStores` map for your new cross-Region action. The mapping must include an entry for each AWS Region in which you have actions. For each entry in the mapping, the resources must be in the respective AWS Region. In the example below, ID–A is the encryption key ID for RegionA, and ID–B is the encryption key ID for RegionB.

```json
"artifactStores":{
    "RegionA":{
        "encryptionKey":{
            "id": "ID-A",
            "type": "KMS"
        },
        "location": "Location1",
        "type": "S3"
    },
    "RegionB":{
        "encryptionKey":{
            "id": "ID-B",
            "type": "KMS"
        },
        "location": "Location2",
        "type": "S3"
    }
}
```

The following JSON example shows the us-west-2 bucket as my-storage-bucket and adds the new us-east-1 bucket named my-storage-bucket-us-east-1.
Add a cross-Region action to a pipeline (CLI)

4. If you are working with the pipeline structure retrieved using the `get-pipeline` command, remove the metadata lines from the JSON file. Otherwise, the `update-pipeline` command cannot use it. Remove the "metadata": { } lines and the "created", "pipelineARN", and "updated" fields.

For example, remove the following lines from the structure:

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

Save the file.

5. To apply your changes, run the `update-pipeline` command, specifying the pipeline JSON file:

**Important**
Be sure to include `file://` before the file name. It is required in this command.

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

This command returns the entire structure of the edited pipeline. The output is similar to the following.

```json
{
  "pipeline": {
    "version": 4,
    "roleArn": "ARN",
    "stages": [
      {
        "name": "Source",
        "actions": [
          {
            "inputArtifacts": [],
            "name": "Source",
            "actionTypeId": {
              "category": "Source",
              "owner": "AWS",
              "version": "1",
              "provider": "CodeCommit"
            },
            "outputArtifacts": [
              {
                "name": "SourceArtifact"
              }
            ],
            "configuration": {
              "PollForSourceChanges": "false",
              "BranchName": "master",
              "RepositoryName": "MyTestRepo"
            }
          }
        ]
      }
    ]
  }
}
```
Add a cross-Region action to a pipeline (CLI)

```json
{
    "name": "AnyCompanyPipeline",
    "artifactStores": {
        "us-west-2": {
            "type": "S3",
            "location": "my-storage-bucket"
        },
        "us-east-1": {
            "type": "S3",
            "location": "my-storage-bucket-us-east-1"
        }
    },
    "stages": [
        {"name": "Source",
         "actions": [
            {"inputArtifacts": [
                {"name": "SourceArtifact"
            }],"name": "Source",
             "region": "us-west-2",
             "actionTypeId": {
                "category": "Source",
                "owner": "AWS",
                "version": "1",
                "provider": "CodePipeline"
            },
             "outputArtifacts": [],
             "configuration": {
                "ArtifactStoreLocation": "my-storage-bucket",
                "SourceArn": "arn:aws:s3:::my-storage-bucket"
            },
             "runOrder": 1
         }
        },
        {"name": "Deploy",
         "actions": [
            {"inputArtifacts": [
                {"name": "SourceArtifact"
            }],"name": "Deploy",
             "region": "us-east-1",
             "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "version": "1",
                "provider": "CodeDeploy"
            },
             "outputArtifacts": [],
             "configuration": {
                "ApplicationName": "name",
                "DeploymentGroupName": "name"
            },
             "runOrder": 1
         }
        }
    ],
    "note": "The update-pipeline command stops the pipeline. If a revision is being run through the pipeline when you run the update-pipeline command, that run is stopped. You must manually start the pipeline to run that revision through the updated pipeline. Use the start-pipeline-execution command to manually start your pipeline.

6. After you update your pipeline, the cross-Region actions is displayed in the console, as shown here.
```
Add a cross-Region action to a pipeline (AWS CloudFormation)

You can use AWS CloudFormation to add a cross-Region action to an existing pipeline.

**To add a cross-Region action with AWS CloudFormation**

1. Add the `Region` parameter to the `ActionDeclaration` resource in your template, as shown in this example:

```
ActionDeclaration:
  Type: Object
  Properties:
    ActionTypeId:
      Type: ActionTypeId
      Required: true
    Configuration:
      Type: Map
    InputArtifacts:
      Type: Array
      Item:
        Type: InputArtifact
        Name:
          Type: String
          Required: true
    OutputArtifacts:
      Type: Array
      Item:
        Type: OutputArtifact
    RoleArn:
      Type: String
    RunOrder:
      Type: Integer
    Region:
      Type: String
```

2. Under Mappings, add the region map as shown in this example for a mapping named `SecondRegionMap` that maps values for the keys `RegionA` and `RegionB`. Under the `Pipeline` resource, under the `artifactStore` field, add the `artifactStores` map for your new cross-Region action as follows:

```
Mappings:
```

API Version 2015-07-09
371
Working with variables

Some actions in CodePipeline generate variables. To use variables:

- You assign a namespace to an action to make the variables it produces available to a downstream action configuration.
Configure actions for variables

When you add an action to your pipeline, you can assign it a namespace and configure it to consume variables from previous actions.

Configure actions with variables (console)

This example creates a pipeline with a CodeCommit source action and a CodeBuild build action. The CodeBuild action is configured to consume the variables produced by the CodeCommit action.

If the namespace isn’t specified, the variables are not available for reference in the action configuration. When you use the console to create a pipeline, the namespace for each action is generated automatically.

To create a pipeline with variables

2. Choose Create pipeline. Enter a name for your pipeline, and then choose Next.
3. In Source, in Provider, choose CodeCommit. Choose the CodeCommit repository and branch for the source action, and then choose Next.
4. In Build, in Provider, choose CodeBuild. Choose an existing CodeBuild build project name or choose Create project. On Create build project, create a build project, and then choose Return to CodePipeline.

Under Environment variables, choose Add environment variables. In this example, enter the execution ID with the variable syntax #{codepipeline.PipelineExecutionId} and commit ID with the variable syntax #{SourceVariables.CommitId}.
Configure actions for variables

5. Choose **Create**.

6. After the pipeline is created, you can view the namespace that was created by the wizard. On the pipeline, choose the help pane icon for the stage you want to view the namespace for. In this example, the source action's auto-generated namespace, `SourceVariables`, is displayed.
To edit the namespace for an existing action

2. Choose the pipeline you want to edit, and then choose Edit. For the source stage, choose Edit stage. Add the CodeCommit action.
3. On Edit action, view the Variable namespace field. If the existing action was created previously or without using the wizard, you must add a namespace. In Variable namespace, enter a namespace name, and then choose Save.

To view output variables

2. After the pipeline is created and runs successfully, you can view the variables on the Action execution details page. For information, see View variables (console) (p. 377).

Configure actions for variables (CLI)

When you use the create-pipeline command to create a pipeline or the update-pipeline command to edit a pipeline, you can reference/use variables in the configuration of an action.

If the namespace isn't specified, the variables produced by the action are not available to be referenced in any downstream action configuration.

To configure an action with a namespace

1. Follow the steps in Create a pipeline in CodePipeline (p. 221) to create a pipeline using the CLI. Start an input file to provide the create-pipeline command with the --cli-input-json parameter. In the pipeline structure, add the namespace parameter and specify a name, such as SourceVariables.

```json
{
    "inputArtifacts": [],
    "name": "Source",
    "namespace": "SourceVariables",
}
```
Configure actions for variables

2. Save the file with a name like MyPipeline.json.
3. At a terminal (Linux, macOS, or Unix) or command prompt (Windows), run the create-pipeline command and create the pipeline.

Call the file you created when you run the create-pipeline command. For example:

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

To configure downstream actions to consume variables

1. Edit an input file to provide the update-pipeline command with the --cli-input-json parameter. In the downstream action, add the variable to the configuration for that action. A variable is made up of a namespace and key, separated by a period. For example, to add variables for the pipeline execution ID and the source commit ID, specify the namespace codepipeline for the variable `{codepipeline.PipelineExecutionId}`. Specify the namespace SourceVariables for the variable `{SourceVariables.CommitId}`.

```json
{
   "name": "Build",
   "actions": [
   {
   "outputArtifacts": [
   {
   "name": "BuildArtifacts"
   },
   "name": "Build",
   "configuration": {
   "EnvironmentVariables": "[{"name":"Execution_ID","value":
   "#(codepipeline.PipelineExecutionId)"},{"name":"Commit_ID
   "value":
   "#(SourceVariables.CommitId)"}],"ProjectName": "env-var-test"
   },
   "inputArtifacts": [
   {
   "name": "SourceArtifact"
   },
   "region": "us-west-2",
   "actionTypeId": {
   "provider": "CodeBuild",
   "category": "Build",
   "version": "1",
   "owner": "AWS"
   },
   "runOrder": 1
   }
   ]
   }
   ]
```
2. Save the file with a name like `MyPipeline.json`.
3. At a terminal (Linux, macOS, or Unix) or command prompt (Windows), run the `create-pipeline` command and create the pipeline.

   Call the file you created when you run the `create-pipeline` command. For example:

   ```bash
   aws codepipeline create-pipeline --cli-input-json file://MyPipeline.json
   ```

---

## View output variables

You can view the action execution details to view the variables for that action, specific to each execution.

### View variables (console)

You can use the console to view variables for an action.


   The names of all pipelines associated with your AWS account are displayed.

2. In **Name**, choose the name of the pipeline.

3. Choose **View history**.

4. After the pipeline runs successfully, you can view the variables produced by the source action. Choose **View history**. Choose **Source** in the action list for the pipeline execution to view the action execution details for the CodeCommit action. On the action detail screen, view the variables under **Output variables**.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthorDate</td>
<td>2019-10-29T03:32:21Z</td>
</tr>
<tr>
<td>BranchName</td>
<td>master</td>
</tr>
<tr>
<td>CommitId</td>
<td>8cf40f22b935b306f06d214517e98ae...</td>
</tr>
<tr>
<td>CommitMessage</td>
<td>Added LICENSE.txt</td>
</tr>
<tr>
<td>CommitterDate</td>
<td>2019-10-29T03:32:21Z</td>
</tr>
<tr>
<td>RepositoryName</td>
<td>repo</td>
</tr>
</tbody>
</table>

5. After the pipeline runs successfully, you can view the variables consumed by the build action. Choose **View history**. In the action list for the pipeline execution, choose **Build** to view the action execution details for the CodeBuild action. On the action detail page, view the variables under **Action configuration**. The auto-generated namespace is displayed.
By default, Action configuration displays the variable syntax. You can choose Show resolved configuration to toggle the list to display the values that were produced during the action execution.

View variables (CLI)

You can use the list-action-executions command to view variables for an action.

1. Use the following command:

   ```
   aws codepipeline list-action-executions
   ```

   The output shows the outputVariables parameter as shown here.

   ```
   "outputVariables": {
     "BranchName": "master",
     "CommitMessage": "Updated files for test",
     "AuthorDate": "2019-11-08T22:24:34Z",
     "CommitId": "d99b0083cc10EXAMPLE",
     "CommitterDate": "2019-11-08T22:24:34Z",
     "RepositoryName": "variables-repo"
   },
   ```

2. Use the following command:

   ```
   aws codepipeline get-pipeline --name <pipeline-name>
   ```

   In the action configuration for the CodeBuild action, you can view the variables:

   ```
   {  
     "name": "Build",
     "actions": [
       {  
         "outputArtifacts": [
           {  
             "name": "BuildArtifact"
           }
         ],
         "name": "Build",
   ```
Example: Use variables in manual approvals

When you specify a namespace for an action, and that action produces output variables, you can add a manual approval that displays variables in the approval message. This example shows you how to add variable syntax to a manual approval message.


   The names of all pipelines associated with your AWS account are displayed. Choose the pipeline you want to add the approval to.

2. To edit your pipeline, choose Edit. Add a manual approval after the source action. In Action name, enter the name of the approval action.


4. In URL for review and Comments, add the variable syntax for CommitId and CommitMessage as shown. Make sure you use the namespace assigned to your source action. For example, the variable syntax for a CodeCommit action with the default namespace SourceVariables is #{SourceVariables.CommitId}.

Example: Use variables in manual approvals

When you specify a namespace for an action, and that action produces output variables, you can add a manual approval that displays variables in the approval message. This example shows you how to add variable syntax to a manual approval message.
5. After the pipeline runs successfully, you can view the variable values in the approval message.
Working with stage transitions in CodePipeline

Transitions are links between pipeline stages that can be disabled or enabled. They are enabled by default. When you re-enable a disabled transition, the latest revision runs through the remaining stages of the pipeline unless more than 30 days have passed. Pipeline execution won’t resume for a transition that has been disabled more than 30 days unless a new change is detected or you manually rerun the pipeline.

You can use the AWS CodePipeline console or the AWS CLI to disable or enable transitions between stages in a pipeline.

**Note**

You can use an approval action to pause the run of a pipeline until it is manually approved to continue. For more information, see Manage approval actions in CodePipeline (p. 355).

**Topics**

- Disable or enable transitions (console) (p. 381)
- Disable or enable transitions (CLI) (p. 383)

Disable or enable transitions (console)

**To disable or enable transitions in a pipeline**


   The names of all pipelines associated with your AWS account are displayed.

2. In **Name**, choose the name of the pipeline for which you want to enable or disable transitions. This opens a detailed view of the pipeline, including the transitions between the stages of the pipeline.

3. Find the arrow after the last stage that you want to run, and then choose the button next to it. For example, in the following pipeline, if you want the actions in the **Staging** stage to run, but not the actions in the stage named **Production**, choose the **Disable transition** button between those two stages:
4. In the **Disable transition** dialog box, enter a reason for disabling the transition, and then choose **Disable**.

The button changes to show that transitions are disabled between the stage preceding the arrow and the stage following the arrow. Any revisions that were already running in the stages that come after the disabled transition continue through the pipeline, but any subsequent revisions do not continue past the disabled transition.

5. Choose the **Enable transition** button next to the arrow. In the **Enable transition** dialog box, choose **Enable**. The pipeline immediately enables the transition between the two stages. If any revisions have been run through the earlier stages after the transition was disabled, in a few moments, the
pipeline starts running the latest revision through the stages after the formerly disabled transition. The pipeline runs the revision through all remaining stages in the pipeline.

**Note**
It might take a few seconds for changes to appear in the CodePipeline console after you enable the transition.

## Disable or enable transitions (CLI)

To disable a transition between stages by using the AWS CLI, run the `disable-stage-transition` command. To enable a disabled transition, run the `enable-stage-transition` command.

### To disable a transition

1. Open a terminal (Linux, macOS, or Unix) or command prompt (Windows) and use the AWS CLI to run the `disable-stage-transition` command, specifying the name of the pipeline, the name of the stage to which you want to disable transitions, the transition type, and the reason you are disabling transitions to that stage. Unlike using the console, you must also specify whether you are disabling transitions into the stage (inbound) or transitions out of the stage after all actions complete (outbound).

For example, to disable the transition to a stage named `Staging` in a pipeline named `MyFirstPipeline`, you would type a command similar to the following:

```bash
aws codepipeline disable-stage-transition --pipeline-name MyFirstPipeline --stage-name Staging --transition-type Inbound --reason "My Reason"
```

The command returns nothing.

2. To verify the transition has been disabled, either view the pipeline in the CodePipeline console or run the `get-pipeline-state` command. For more information, see View pipeline details and history (console) (p. 237) and View pipeline details and history (CLI) (p. 244).

### To enable a transition

1. Open a terminal (Linux, macOS, or Unix) or command prompt (Windows) and use the AWS CLI to run the `enable-stage-transition` command, specifying the name of the pipeline, the name of the stage to which you want to enable transitions, and the transition type.

For example, to enable the transition to a stage named `Staging` in a pipeline named `MyFirstPipeline`, you would type a command similar to the following:

```bash
aws codepipeline enable-stage-transition --pipeline-name MyFirstPipeline --stage-name Staging --transition-type Inbound
```

The command returns nothing.

2. To verify the transition has been disabled, either view the pipeline in the CodePipeline console or run the `get-pipeline-state` command. For more information, see View pipeline details and history (console) (p. 237) and View pipeline details and history (CLI) (p. 244).
Monitoring pipelines with CodePipeline

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS CodePipeline. You should collect monitoring data from all parts of your AWS solution so that you can more easily debug a multi-point failure, if one occurs. Before you start monitoring, you should create a monitoring plan that answers the following questions:

- What are your monitoring goals?
- Which resources will you monitor?
- How often will you monitor these resources?
- Which monitoring tools are available for you to use?
- Who will perform the monitoring tasks?
- Who should be notified if something goes wrong?

You can use the following tools to monitor your CodePipeline pipelines and their resources:

- **Amazon CloudWatch Events** — Use Amazon CloudWatch Events to detect and react to pipeline execution state changes (for example, send an Amazon SNS notification or invoke a Lambda function).
- **AWS CloudTrail** — Use CloudTrail to capture API calls made by or on behalf of CodePipeline in your AWS account and deliver the log files to an Amazon S3 bucket. You can choose to have CloudWatch publish Amazon SNS notifications when new log files are delivered so you can take quick action.
- **Console and CLI** — You can use the CodePipeline console and CLI to view details about the status of a pipeline or a particular pipeline execution.

**Topics**

- Detect and react to changes in pipeline state with Amazon CloudWatch Events (p. 384)
- Events placeholder bucket reference (p. 392)
- Logging CodePipeline API calls with AWS CloudTrail (p. 394)

Detect and react to changes in pipeline state with Amazon CloudWatch Events

Amazon CloudWatch Events is a web service that monitors your AWS resources and the applications you run on AWS. You can use Amazon CloudWatch Events to detect and react to changes in the state of a pipeline, stage, or action. Then, based on rules you create, CloudWatch Events invokes one or more target actions when a pipeline, stage, or action enters the state you specify in a rule. Depending on the type of state change, you might want to send notifications, capture state information, take corrective action, initiate events, or take other actions.

Amazon CloudWatch Events are composed of:
• **Rules.** An event in Amazon CloudWatch Events is configured by first creating a rule with a selected service as the event source.

• **Targets.** The new rule receives a selected service as the event target. For a list of services available as Amazon CloudWatch Events targets, see What Is Amazon CloudWatch Events.

Examples of Amazon CloudWatch Events rules and targets:

• A rule that sends a notification when the instance state changes, where an EC2 instance is the event source and Amazon SNS is the event target.

• A rule that sends a notification when the build phase changes, where a CodeBuild configuration is the event source and Amazon SNS is the event target.

• A rule that detects pipeline changes and invokes an AWS Lambda function.

To configure AWS CodePipeline as an event source:

1. Create an Amazon CloudWatch Events rule that uses CodePipeline as an event source.
2. Create a target for your rule that uses one of the services available as targets in Amazon CloudWatch Events, such as AWS Lambda or Amazon SNS.
3. Grant permissions to Amazon CloudWatch Events to allow it to invoke the selected target service.

### Understand how a pipeline execution state change rule works

You build rules for detecting and reacting to pipeline state changes using the Events window in Amazon CloudWatch. As you build your rule, the Event Pattern Preview box in the console (or the --event-pattern output in the CLI) displays the event fields, in JSON format.

You can configure notifications to be sent when the state changes for:

• Specified pipelines or all your pipelines. You control this by using "detail-type": "CodePipeline Pipeline Execution State Change".

• Specified stages or all your stages, within a specified pipeline or all your pipelines. You control this by using "detail-type": "CodePipeline Stage Execution State Change".

• Specified actions or all actions, within a specified stage or all stages, within a specified pipeline or all your pipelines. You control this by using "detail-type": "CodePipeline Action Execution State Change".

Each type of execution state change event emits notifications with specific message content, where:

• The initial version entry shows the version number for the CloudWatch event.

• The version entry under pipeline detail shows the pipeline structure version number.

• The execution-id entry under pipeline detail shows the execution ID for the pipeline execution that caused the state change. Refer to the GetPipelineExecution API call in the AWS CodePipeline API Reference.

**Pipeline execution state change message content:** When a pipeline execution starts, it emits an event that sends notifications with the following content. This example is for the pipeline named "myPipeline" in the us-east-1 region.

```json
{
}
```
Stage execution state change message content: When a stage execution starts, it emits an event that sends notifications with the following content. This example is for the pipeline named "myPipeline" in the us-east-1 region, for the stage "Prod".

```json
{
    "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"
    }
}
```

Action execution state change message content: When an action execution starts, it emits an event that sends notifications with the following content. This example is for the pipeline named "myPipeline" in the us-east-1 region, for the action "myAction".

```json
{
    "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"
    }
}
```
Understand how a pipeline execution state change rule works

```
"state": "STARTED",
"type": {
  "owner": "AWS",
  "category": "Deploy",
  "provider": "CodeDeploy",
  "version": 1
}
```

Valid state values:

**Pipeline-level states in events**

<table>
<thead>
<tr>
<th>Pipeline state</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STARTED</td>
<td>The pipeline execution is currently running.</td>
</tr>
<tr>
<td>SUCCEEDED</td>
<td>The pipeline execution was completed successfully.</td>
</tr>
<tr>
<td>RESUMED</td>
<td>A failed pipeline execution has been retried in response to the <code>RetryStageExecution</code> API call.</td>
</tr>
<tr>
<td>FAILED</td>
<td>The pipeline execution was not completed successfully.</td>
</tr>
<tr>
<td>CANCELED</td>
<td>The pipeline execution was canceled because the pipeline structure was updated.</td>
</tr>
<tr>
<td>SUPERSEDED</td>
<td>While this pipeline execution was waiting for the next stage to be completed, a newer pipeline execution advanced and continued through the pipeline instead.</td>
</tr>
</tbody>
</table>

**Stage-level states in events**

<table>
<thead>
<tr>
<th>Stage state</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STARTED</td>
<td>The stage is currently running.</td>
</tr>
<tr>
<td>SUCCEEDED</td>
<td>The stage was completed successfully.</td>
</tr>
<tr>
<td>RESUMED</td>
<td>A failed stage has been retried in response to the <code>RetryStageExecution</code> API call.</td>
</tr>
<tr>
<td>FAILED</td>
<td>The stage was not completed successfully.</td>
</tr>
<tr>
<td>CANCELED</td>
<td>The stage was canceled because the pipeline structure was updated.</td>
</tr>
</tbody>
</table>

**Action-level states in events**

<table>
<thead>
<tr>
<th>Action state</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STARTED</td>
<td>The action is currently running.</td>
</tr>
<tr>
<td>SUCCEEDED</td>
<td>The action was completed successfully.</td>
</tr>
<tr>
<td>FAILED</td>
<td>For Approval actions, the FAILED state means the action was either rejected by the reviewer or failed due to an incorrect action configuration.</td>
</tr>
<tr>
<td>CANCELED</td>
<td>The action was canceled because the pipeline structure was updated.</td>
</tr>
</tbody>
</table>
Prerequisites

Before you create event rules for use in your CodePipeline operations, you should do the following:

- Complete the CloudWatch Events prerequisites. For this information, see Regional Endpoints.
- Familiarize yourself with events, rules, and targets in CloudWatch Events. For more information, see What Is Amazon CloudWatch Events.
- Create the target or targets you will use in your event rules, such as an Amazon SNS notification topic.

Send a notification whenever pipeline state changes (console)

These steps show how to use the CloudWatch console to create a rule to send notifications of changes in CodePipeline.

To create a CloudWatch Events rule with CodePipeline as the event source

2. In the navigation pane, choose Events.
3. Choose Create rule. Under Event source, from the Service Name drop-down list, choose CodePipeline.
4. From the Event Type drop-down list, choose the level of state change for the notification.
   - For a rule that applies to pipeline-level events, choose CodePipeline Pipeline Execution State Change.
   - For a rule that applies to stage-level events, choose CodePipeline Stage Execution State Change.
   - For a rule that applies to action-level events, choose CodePipeline Action Execution State Change.
5. Specify the state changes the rule applies to:
   - For a rule that applies to all state changes, choose Any state.
   - For a rule that applies to some state changes only, choose Specific state(s), and then choose one or more state values from the list.
6. For event patterns that are more detailed than the selectors allow, you can also use the **Edit** option in the **Event Pattern Preview** window to designate an event pattern in JSON format. The following example shows the JSON structure edited manually to specify a pipeline named "myPipeline."
**Note**  
If not otherwise specified, then the event pattern is created for all pipelines/stages/actions and states.

For more detailed event patterns, you can copy and paste the following example event patterns into the Edit window.

- **Example**

  Use this sample event pattern to capture failed deploy and build actions across all the pipelines.

```json
{
  "source": [
    "aws.codepipeline"
  ],
  "detail-type": [
    "CodePipeline Pipeline Execution State Change"
  ],
  "detail": {
    "state": [
      "FAILED"
    ],
    "pipeline": [
      "myPipeline"
    ]
  }
}
```
• **Example**

Use this sample event pattern to capture all rejected or failed approval actions across all the pipelines.

```json
{
  "source": [
    "aws.codepipeline"
  ],
  "detail-type": [
    "CodePipeline Action Execution State Change"
  ],
  "detail": {
    "state": [
      "FAILED"
    ],
    "type": {
      "category": ["Approval"]
    }
  }
}
```

• **Example**

Use this sample event pattern to capture all the events from the specified pipelines.

```json
{
  "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. In the **Targets** area, choose **Add target**.
8. In the **Select target type** list, choose the type of target for this rule, and then configure options required by that type.
9. Choose **Configure details**.
10. On the **Configure rule details** page, type a name and description for the rule, and then select the **State** box to enable to rule now.
11. Choose **Create rule**.

**Send a notification whenever pipeline state changes (CLI)**

These steps show how to use the CLI to create a CloudWatch Events rule to send notifications of changes in CodePipeline.

To use the AWS CLI to create a rule, call the **put-rule** command, specifying:

- A name that uniquely identifies the rule you are creating. This name must be unique across all of the pipelines you create with CodePipeline associated with your AWS account.
The event pattern for the source and detail fields used by the rule. For more information, see Amazon CloudWatch Events and Event Patterns.

To create a CloudWatch Events rule with CodePipeline as the event source

1. Call the put-rule command to create a rule specifying the event pattern. (See the preceding tables for valid states.)

   The following sample command uses --event-pattern to create a rule called "MyPipelineStateChanges" that emits the CloudWatch event when a pipeline execution fails for the pipeline named "myPipeline."

   ```
   aws events put-rule --name "MyPipelineStateChanges" --event-pattern "{"source": ["aws.codepipeline"], "detail-type": ["CodePipeline Pipeline Execution State Change"], "detail": {"pipeline": ["myPipeline"], "state": ["FAILED"]}}"
   ```

2. Call the put-targets command to add a target to your new rule, as shown in this example for an Amazon SNS topic:

   ```
   aws events put-targets --rule MyPipelineStateChanges --targets Id=1,Arn=arn:aws:sns:us-west-2:11111EXAMPLE:MyNotificationTopic
   ```

3. Add permissions for Amazon CloudWatch Events to use the designated target service to invoke the notification. For more information, see Using Resource-Based Policies for Amazon CloudWatch Events.

Events placeholder bucket reference

This section is a reference only. For information about creating a pipeline with event detection resources, see Change detection methods to start pipelines (p. 172).

Source actions provided by Amazon S3 and CodeCommit use event-based change detection resources to trigger your pipeline when a change is made in the source bucket or repository. These resources are the CloudWatch Events rules that are configured to respond to events in the pipeline source, such as a code change to the CodeCommit repository. When you use CloudWatch Events for an Amazon S3 source, you must turn on CloudTrail so the events are logged. CloudTrail requires an S3 bucket where it can send its digests. You can access the log files for your CloudWatch Events resources from the placeholder bucket or from a bucket you designated.

- If you used the CLI or AWS CloudFormation to set up the CloudWatch Events resources, you can find your CloudTrail files in the bucket that you specified when you set up your pipeline.
- If you used the console to set up your pipeline with an S3 source, the console uses a CloudTrail placeholder bucket when it creates your CloudWatch Events resources for you. CloudTrail digests are stored in the placeholder bucket in the AWS Region where the pipeline is created.

You can change the configuration if you want to use a bucket other than the placeholder bucket.

For more information about finding and managing your CloudTrail log files, see Getting and Viewing Your CloudTrail Log Files.

Topics

- Events placeholder bucket names by Region (p. 393)
Events placeholder bucket names by Region

This table lists the names of the S3 placeholder buckets that contain log files that track change detection events for pipelines with Amazon S3 source actions.

<table>
<thead>
<tr>
<th>Region name</th>
<th>Placeholder bucket name</th>
<th>Region identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>US East (Ohio)</td>
<td>codepipeline-cloudtrail-placeholder-bucket-us-east-2</td>
<td>us-east-2</td>
</tr>
<tr>
<td>US East (N. Virginia)</td>
<td>codepipeline-cloudtrail-placeholder-bucket-us-east-1</td>
<td>us-east-1</td>
</tr>
<tr>
<td>US West (N. California)</td>
<td>codepipeline-cloudtrail-placeholder-bucket-us-west-1</td>
<td>us-west-1</td>
</tr>
<tr>
<td>US West (Oregon)</td>
<td>codepipeline-cloudtrail-placeholder-bucket-us-west-2</td>
<td>us-west-2</td>
</tr>
<tr>
<td>Canada (Central)</td>
<td>codepipeline-cloudtrail-placeholder-bucket-ca-central-1</td>
<td>ca-central-1</td>
</tr>
<tr>
<td>Europe (Frankfurt)</td>
<td>codepipeline-cloudtrail-placeholder-bucket-eu-central-1</td>
<td>eu-central-1</td>
</tr>
<tr>
<td>Europe (Ireland)</td>
<td>codepipeline-cloudtrail-placeholder-bucket-eu-west-1</td>
<td>eu-west-1</td>
</tr>
<tr>
<td>Europe (London)</td>
<td>codepipeline-cloudtrail-placeholder-bucket-eu-west-2</td>
<td>eu-west-2</td>
</tr>
<tr>
<td>Europe (Paris)</td>
<td>codepipeline-cloudtrail-placeholder-bucket-eu-west-3</td>
<td>eu-west-3</td>
</tr>
<tr>
<td>Europe (Stockholm)</td>
<td>codepipeline-cloudtrail-placeholder-bucket-eu-north-1</td>
<td>eu-north-1</td>
</tr>
<tr>
<td>Asia Pacific (Tokyo)</td>
<td>codepipeline-cloudtrail-placeholder-bucket-ap-northeast-1</td>
<td>ap-northeast-1</td>
</tr>
<tr>
<td>Asia Pacific (Seoul)</td>
<td>codepipeline-cloudtrail-placeholder-bucket-ap-northeast-2</td>
<td>ap-northeast-2</td>
</tr>
<tr>
<td>Asia Pacific (Singapore)</td>
<td>codepipeline-cloudtrail-placeholder-bucket-ap-southeast-1</td>
<td>ap-southeast-1</td>
</tr>
<tr>
<td>Asia Pacific (Sydney)</td>
<td>codepipeline-cloudtrail-placeholder-bucket-ap-southeast-2</td>
<td>ap-southeast-2</td>
</tr>
<tr>
<td>Asia Pacific (Mumbai)</td>
<td>codepipeline-cloudtrail-placeholder-bucket-ap-south-1</td>
<td>ap-south-1</td>
</tr>
<tr>
<td>South America (São Paulo)</td>
<td>codepipeline-cloudtrail-placeholder-bucket-sa-east-1</td>
<td>sa-east-1</td>
</tr>
</tbody>
</table>
Logging CodePipeline API calls with AWS CloudTrail

AWS CodePipeline is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in CodePipeline; CloudTrail captures all API calls for CodePipeline as events. The calls captured include calls from the CodePipeline console and code calls to the CodePipeline API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for CodePipeline. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in Event history. Using the information collected by CloudTrail, you can determine the request that was made to CodePipeline, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

CodePipeline information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in CodePipeline, that activity is recorded in a CloudTrail event along with other AWS service events in Event history. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for CodePipeline, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All CodePipeline actions are logged by CloudTrail and are documented in the CodePipeline API Reference. For example, calls to the CreatePipeline, GetPipelineExecution and UpdatePipeline actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Understanding CodePipeline log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from
any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry for an update pipeline event, where a pipeline named MyFirstPipeline has been edited by the user named JaneDoe-CodePipeline with the account ID 80398EXAMPLE. The user changed the name of the source stage of a pipeline from Source to MySourceStage. Because both the requestParameters and the responseElements elements in the CloudTrail log contain the entire structure of the edited pipeline, those elements have been abbreviated in the following example. Emphasis has been added to the requestParameters portion of the pipeline where the change occurred, the previous version number of the pipeline, and the responseElements portion, which shows the version number incremented by 1. Edited portions are marked with ellipses (...) to illustrate where more data appears in a real log entry.

```
{  
  "eventVersion":"1.03",  
  "userIdentity": {  
    "type":"IAMUser",  
    "principalId":"AKIAI44QH8DHBEEXAMPLE",  
    "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"  
        }  
      ]  
    }  
  }  
```

API Version 2015-07-09
395
},
(...)
},
"responseElements":{
"pipeline":{
"version":2,
(...)
},
"requestID":"2c4af5c9-7ce8-EXAMPLE",
"eventID":"c53dbd42-This-Is-An-Example",
"eventType":"AwsApiCall",
"recipientAccountId":"80398EXAMPLE"
}
]}
}
Troubleshooting CodePipeline

The following information might help you troubleshoot common issues in AWS CodePipeline.

Topics

- Pipeline error: A pipeline configured with AWS Elastic Beanstalk returns an error message: "Deployment failed. The provided role does not have sufficient permissions: Service:AmazonElasticLoadBalancing" (p. 397)
- Deployment error: A pipeline configured with an AWS Elastic Beanstalk deploy action hangs instead of failing if the "DescribeEvents" permission is missing (p. 398)
- Pipeline error: A source action returns the insufficient permissions message: "Could not access the CodeCommit repository repository-name. Make sure that the pipeline IAM role has sufficient permissions to access the repository." (p. 398)
- Pipeline error: A Jenkins build or test action runs for a long time and then fails due to lack of credentials or permissions (p. 398)
- Pipeline error: My GitHub source stage contains Git submodules, but CodePipeline doesn't initialize them (p. 399)
- Pipeline error: I receive a pipeline error that says: "Could not access the GitHub repository" or "Unable to connect to the GitHub repository" (p. 399)
- Pipeline error: A pipeline created in one AWS Region using a bucket created in another AWS Region returns an "InternalError" with the code "JobFailed" (p. 401)
- Deployment error: A ZIP file that contains a WAR file is deployed successfully to AWS Elastic Beanstalk, but the application URL reports a 404 not found error (p. 398)
- Pipeline artifact folder names appear to be truncated (p. 401)
- Pipeline error: A deployment with the CodeDeployToECS action returns an error message: "Exception while trying to read the task definition artifact file from: <source artifact name>" (p. 402)
- Need help with a different issue? (p. 402)

Pipeline error: A pipeline configured with AWS Elastic Beanstalk returns an error message: "Deployment failed. The provided role does not have sufficient permissions: Service:AmazonElasticLoadBalancing"

**Problem:** The service role for CodePipeline does not have sufficient permissions for AWS Elastic Beanstalk, including, but not limited to, some operations in Elastic Load Balancing. The service role for CodePipeline was updated on August 6, 2015 to address this issue. Customers who created their service role before this date must modify the policy statement for their service role to add the required permissions.

**Possible fixes:** The easiest solution is to edit the policy statement for your service role as detailed in Add permissions to the CodePipeline service role (p. 446).

After you apply the edited policy, follow the steps in Start a pipeline manually in AWS CodePipeline (p. 215) to manually rerun any pipelines that use Elastic Beanstalk.

Depending on your security needs, you can modify the permissions in other ways, too.
Deployment error: A pipeline configured with an AWS Elastic Beanstalk deploy action hangs instead of failing if the "DescribeEvents" permission is missing

**Problem:** The service role for CodePipeline must include the "elasticbeanstalk:DescribeEvents" action for any pipelines that use AWS Elastic Beanstalk. Without this permission, AWS Elastic Beanstalk deploy actions hang without failing or indicating an error. If this action is missing from your service role, then CodePipeline does not have permissions to run the pipeline deployment stage in AWS Elastic Beanstalk on your behalf.

**Possible fixes:** Review your CodePipeline service role. If the "elasticbeanstalk:DescribeEvents" action is missing, use the steps in Add permissions to the CodePipeline service role (p. 446) to add it using the Edit Policy feature in the IAM console.

After you apply the edited policy, follow the steps in Start a pipeline manually in AWS CodePipeline (p. 215) to manually rerun any pipelines that use Elastic Beanstalk.

Pipeline error: A source action returns the insufficient permissions message: "Could not access the CodeCommit repository repository-name. Make sure that the pipeline IAM role has sufficient permissions to access the repository."

**Problem:** The service role for CodePipeline does not have sufficient permissions for CodeCommit and likely was created before support for using CodeCommit repositories was added on April 18, 2016. Customers who created their service role before this date must modify the policy statement for their service role to add the required permissions.

**Possible fixes:** Add the required permissions for CodeCommit to your CodePipeline service role's policy. For more information, see Add permissions to the CodePipeline service role (p. 446).

Pipeline error: A Jenkins build or test action runs for a long time and then fails due to lack of credentials or permissions

**Problem:** If the Jenkins server is installed on an Amazon EC2 instance, the instance might not have been created with an instance role that has the permissions required for CodePipeline. If you are using an IAM user on a Jenkins server, an on-premises instance, or an Amazon EC2 instance created without the required IAM role, the IAM user either does not have the required permissions, or the Jenkins server cannot access those credentials through the profile configured on the server.
Possible fixes: Make sure that Amazon EC2 instance role or IAM user is configured with the AWSCodePipelineCustomActionAccess managed policy or with the equivalent permissions. For more information, see AWS managed (predefined) policies for CodePipeline (p. 428).

If you are using an IAM user, make sure the AWS profile configured on the instance uses the IAM user configured with the correct permissions. You might have to provide the IAM user credentials you configured for integration between Jenkins and CodePipeline directly into the Jenkins UI. This is not a recommended best practice. If you must do so, be sure the Jenkins server is secured and uses HTTPS instead of HTTP.

Pipeline error: My GitHub source stage contains Git submodules, but CodePipeline doesn't initialize them

Problem: CodePipeline does not support git submodules. CodePipeline relies on the archive link API from GitHub, which does not support submodules.

Possible fixes: Consider cloning the GitHub repository directly as part of a separate script. For example, you could include a clone action in a Jenkins script.

Pipeline error: I receive a pipeline error that says: "Could not access the GitHub repository" or "Unable to connect to the GitHub repository"

Problem: CodePipeline uses OAuth tokens to integrate with GitHub. When you create a pipeline with a GitHub source provider, CodePipeline manages your GitHub credentials by creating a default OAuth token. When your pipeline connects to the repository, it uses GitHub credentials to connect to GitHub. The OAuth token credentials are managed by CodePipeline. You do not view or manage the token in any way. The other type of credentials you can use to connect to GitHub are personal access tokens, which are created by you instead of by OAuth apps. Personal access tokens are managed by you and not by CodePipeline.

If these permissions have been revoked or otherwise disabled, then the pipeline fails when it cannot use the GitHub token to connect to the repository.

It is a security best practice to rotate your personal access token on a regular basis. For more information, see Use GitHub and the CodePipeline CLI to create and rotate your GitHub personal access token on a regular basis (p. 410).

Possible fixes:

If CodePipeline is unable to connect to the GitHub repository, there are two troubleshooting options:

• You might simply need to reconnect your pipeline to the repository manually. You might have revoked the permissions of the OAuth token for CodePipeline and they just need to be restored. To do this, see the steps below.

• You might need to change your default OAuth token to a personal access token. The number of OAuth tokens is limited. For more information, see the GitHub documentation. If CodePipeline reaches that limit, older tokens stop working, and actions in pipelines that rely on that token fail.
Pipeline error: I receive a pipeline error that says: "Could not access the GitHub repository" or "Unable to connect to the GitHub repository"

1. Check to see if the permissions for CodePipeline have been revoked. For the steps to check the Authorized OAuth Apps list in GitHub, see View your authorized OAuth apps (p. 407). If you do not see CodePipeline in the list, you must use the console to reconnect your pipeline to GitHub.

   a. Open your pipeline in the console and choose Edit. On the source stage that contains your GitHub source action, choose Edit stage.
   b. On the GitHub source action, choose the edit icon.
   c. On the Edit action page, choose Connect to GitHub to restore the authorization.

   ![Edit action screenshot](image.png)

   If prompted, you might need to re-enter the Repository and Branch for your action. Choose Done. Choose Done on the stage editing page, and then choose Save on the pipeline editing page. Run the pipeline.

2. If this does not correct the error but you can see CodePipeline in the Authorized OAuth Apps list in GitHub, you might have exceeded the allowed number of tokens. To fix this issue, you can manually configure one OAuth token as a personal access token, and then configure all pipelines in your AWS account to use that token. For more information, see Configure your pipeline to use a personal access token (GitHub and CLI) (p. 408).
Pipeline error: A pipeline created in one AWS Region using a bucket created in another AWS Region returns an "InternalError" with the code "JobFailed"

**Problem:** The download of an artifact stored in an Amazon S3 bucket will fail if the pipeline and bucket are created in different AWS Regions.

**Possible fixes:** Make sure the Amazon S3 bucket where your artifact is stored is in the same AWS Region as the pipeline you have created.

Deployment error: A ZIP file that contains a WAR file is deployed successfully to AWS Elastic Beanstalk, but the application URL reports a 404 not found error

**Problem:** A WAR file is deployed successfully to an AWS Elastic Beanstalk environment, but the application URL returns a 404 Not Found error.

**Possible fixes:** AWS Elastic Beanstalk can unpack a ZIP file, but not a WAR file contained in a ZIP file. Instead of specifying a WAR file in your buildspec.yml file, specify a folder that contains the content to be deployed. For example:

```yaml
version: 0.2
phases:
  post_build:
    commands:
      - mvn package
      - mv target/my-web-app ./
artifacts:
  files:
    - my-web-app/**/*
discard-paths: yes
```

For an example, see [AWS Elastic Beanstalk Sample for CodeBuild](#).

Pipeline artifact folder names appear to be truncated

**Problem:** When you view pipeline artifact names in CodePipeline, the names appear to be truncated. This can make the names appear to be similar or seem to no longer contain the entire pipeline name.

**Explanation:** CodePipeline truncates artifact names to ensure that the full Amazon S3 path does not exceed policy size limits when CodePipeline generates temporary credentials for job workers.
Pipeline error: A deployment with the CodeDeployToECS action returns an error message: "Exception while trying to read the task definition artifact file from: <source artifact name>"

Problem:

The maximum artifact ZIP size in the CodePipeline deploy action to ECS through CodeDeploy (the CodeDeployToECS action) is 3 MB. The following error message is returned when artifact sizes exceed 3 MB:

Exception while trying to read the task definition artifact file from: <source artifact name>

Possible fixes: Create an artifact with a compressed size less than 3 MB.

Need help with a different issue?

Try these other resources:

- Contact AWS Support.
- Ask a question in the CodePipeline forum.
- Request a quota increase. For more information, see Quotas in AWS CodePipeline (p. 521).

Note

It can take up to two weeks to process requests for a quota increase.
Security in AWS CodePipeline

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS compliance programs. To learn about the compliance programs that apply to AWS CodePipeline, see AWS Services in Scope by Compliance Program.

- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors, including the sensitivity of your data, your company’s requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using CodePipeline. The following topics show you how to configure CodePipeline to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your CodePipeline resources.

**Topics**
- Data protection in AWS CodePipeline (p. 403)
- Identity and access management for AWS CodePipeline (p. 413)
- Logging and monitoring in CodePipeline (p. 450)
- Compliance validation for AWS CodePipeline (p. 451)
- Resilience in AWS CodePipeline (p. 451)
- Infrastructure security in AWS CodePipeline (p. 451)
- Security best practices (p. 452)

Data protection in AWS CodePipeline

AWS CodePipeline conforms to the AWS shared responsibility model, which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all of the AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and APN partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM) so that each user is given only the permissions required to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls in AWS services.
Internetwork traffic privacy

Amazon VPC is an AWS service that you can use to launch AWS resources in a virtual network (virtual private cloud) that you define. CodePipeline supports Amazon VPC endpoints powered by AWS PrivateLink, an AWS technology that facilitates private communication between AWS services using an elastic network interface with private IP addresses. This means you can connect directly to CodePipeline through a private endpoint in your VPC, keeping all traffic inside your VPC and the AWS network. Previously, applications running inside a VPC required internet access to connect to CodePipeline. With a VPC, you have control over your network settings, such as:

- IP address range.
- Subnets.
- Route tables.
- Network gateways.

To connect your VPC to CodePipeline, you define an interface VPC endpoint for CodePipeline. This type of endpoint makes it possible for you to connect your VPC to AWS services. The endpoint provides reliable, scalable connectivity to CodePipeline without requiring an internet gateway, network address translation (NAT) instance, or VPN connection. For information about setting up a VPC, see the VPC User Guide.

Encryption at rest

Data in CodePipeline is encrypted at rest using service-owned KMS keys. Code artifacts are stored in a customer-owned S3 bucket and encrypted with either the default AWS managed SSE-KMS encryption key or a customer managed SSE-KMS key. For more information, see Configure server-side encryption for artifacts stored in Amazon S3 for CodePipeline (p. 405).

Encryption in transit

All service to service communication is encrypted in transit using SSL/TLS.

Encryption key management

If the customer chooses the default option for encrypting code artifacts, CodePipeline uses the default AWS managed SSE-KMS encryption key. Customers cannot change or delete this AWS managed key. If customers are using a customer managed key (CMK) in AWS KMS to encrypt or decrypt artifacts in the S3 bucket, they can change or rotate this key as necessary.

Important

CodePipeline only supports symmetric customer master keys (CMKs). Do not use an asymmetric CMK to encrypt the data in your S3 bucket.
Data protection configuration

This section describes data protection configuration for the following:

- S3 artifacts server-side encryption (SSE).
- GitHub personal access tokens.
- Secret parameter tracking in AWS Secrets Manager.

Topics

- Configure server-side encryption for artifacts stored in Amazon S3 for CodePipeline (p. 405)
- Configure GitHub authentication (p. 407)
- Use AWS Secrets Manager to track database passwords or third-party API keys (p. 413)

Configure server-side encryption for artifacts stored in Amazon S3 for CodePipeline

There are two ways to configure server-side encryption for Amazon S3 artifacts:

- CodePipeline creates an S3 artifact bucket and default AWS managed SSE-KMS encryption keys when you create a pipeline using the Create Pipeline wizard. The master key is encrypted along with object data and managed by AWS.
- You can create and manage your own customer managed SSE-KMS keys.

**Important**

CodePipeline only supports symmetric customer master keys (CMKs). Do not use an asymmetric CMK to encrypt the data in your S3 bucket.

If you are using the default S3 key, you cannot change or delete this AWS managed key. If you are using a customer managed key in AWS KMS to encrypt or decrypt artifacts in the S3 bucket, you can change or rotate this key as necessary.

Amazon S3 supports bucket policies that you can use if you require server-side encryption for all objects that are stored in your bucket. For example, the following bucket policy denies upload object (s3:PutObject) permission to everyone if the request does not include the x-amz-server-side-encryption header requesting server-side encryption with SSE-KMS.

```json
{
  "Version": "2012-10-17",
  "Id": "SSEAndSSLPolicy",
  "Statement": [
    {
      "Sid": "DenyUnEncryptedObjectUploads",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::codepipeline-us-west-2-89050EXAMPLE/*",
      "Condition": {
        "StringNotEquals": {
          "s3:x-amz-server-side-encryption": "aws:kms"
        }
      }
    },
    {
      "Sid": "DenyInsecureConnections",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::codepipeline-us-west-2-89050EXAMPLE/*",
      "Condition": {
        "StringNotEquals": {
          "s3:x-amz-server-side-encryption": "aws:kms"
        }
      }
    }
  ]
}
```

API Version 2015-07-09

405
Data protection configuration

```json
"Effect": "Deny",
"Principal": "*",
"Action": "s3:*",
"Resource": "arn:aws:s3:::codepipeline-us-west-2-89050EXAMPLE/*",
"Condition": {
  "Bool": {
    "aws:SecureTransport": "false"
  }
}
```

For more information about server-side encryption and AWS KMS, see Protecting Data Using Server-Side Encryption and Protecting Data Using Server-Side Encryption with CMKs Stored in AWS Key Management Service (SSE-KMS).

For more information about AWS KMS, see the AWS Key Management Service Developer Guide.

Topics

- View your default Amazon S3 SSE-KMS encryption keys (p. 406)
- Configure server-side encryption for S3 buckets using AWS CloudFormation or the AWS CLI (p. 406)

View your default Amazon S3 SSE-KMS encryption keys

When you use the Create Pipeline wizard to create your first pipeline, an S3 bucket is created for you in the same Region you created the pipeline. The bucket is used to store pipeline artifacts. When a pipeline runs, artifacts are put into and retrieved from the S3 bucket. By default, CodePipeline uses server-side encryption with the AWS KMS-managed keys (SSE-KMS) using the default key for Amazon S3 (the `aws/s3` key). This key is created and stored in your AWS account. When artifacts are retrieved from the S3 bucket, CodePipeline uses the same SSE-KMS process to decrypt the artifact.

To view information about your default AWS KMS key

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the service navigation pane, choose Encryption Keys. (If a welcome page appears, choose Get Started Now.)
3. In Filter, choose the Region for your pipeline. For example, if the pipeline was created in us-east-2, make sure that the filter is set to US East (Ohio).

   For more information about the Regions and endpoints available for CodePipeline, see AWS CodePipeline endpoints and quotas.
4. In the list of encryption keys, choose the key with the alias used for your pipeline (by default, `aws/s3`). Basic information about the key is displayed.

Configure server-side encryption for S3 buckets using AWS CloudFormation or the AWS CLI

When you use AWS CloudFormation or the AWS CLI to create a pipeline, you must configure server-side encryption manually. Use the sample bucket policy above, and then create your own customer managed SSE-KMS encryption keys. You can also use your own keys instead of the default Amazon S3 key. Some advantages to using your own key include:

- You want to rotate the key on a schedule to meet business or security requirements for your organization.
• You want to create a pipeline that uses resources associated with another AWS account. This requires the use of a customer managed key. For more information, see Create a pipeline in CodePipeline that uses resources from another AWS account (p. 251).

Cryptographic best practices discourage extensive reuse of encryption keys. As a best practice, rotate your key on a regular basis. To create new cryptographic material for your AWS Key Management Service (AWS KMS) customer master keys (CMKs), you can create CMKs, and then change your applications or aliases to use the new CMKs. Or, you can enable automatic key rotation for an existing CMK.

To rotate your SSE-KMS customer master key, see Rotating Customer Master Keys.

**Important**
CodePipeline only supports symmetric customer master keys (CMKs). Do not use an asymmetric CMK to encrypt the data in your S3 bucket.

**Configure GitHub authentication**

CodePipeline uses GitHub OAuth tokens and personal access tokens to access your GitHub repositories and retrieve the latest changes. There are two ways to configure authentication in GitHub:

• AWS creates a default AWS managed OAuth token when you use the console to create or update pipelines.

• You can create and manage your own customer-generated personal access tokens. You need personal access tokens when you use the CLI, SDK, or AWS CloudFormation to create or update your pipeline.

**Topics**

• View your authorized OAuth apps (p. 407)

• Configure your pipeline to use a personal access token (GitHub and CLI) (p. 408)

• Use GitHub and the CodePipeline CLI to create and rotate your GitHub personal access token on a regular basis (p. 410)

**View your authorized OAuth apps**

CodePipeline uses OAuth tokens to integrate with GitHub. GitHub tracks the permissions of the OAuth token for CodePipeline.

**To view your authorized integrations in GitHub**

1. In GitHub, from the drop-down option on your profile photo, choose Settings.
2. Choose Applications, and then choose Authorized OAuth Apps.
3. Review your authorized apps.
Configure your pipeline to use a personal access token (GitHub and CLI)

You can configure your pipeline to use a personal access token to connect to GitHub. The advantage of using tokens instead of passwords in a script is that tokens can be revoked or rotated. You can also grant specific privileges and permissions to a personal access token. Each personal access token is associated at the pipeline, rather than account, level.

**Note**
You might have to update other applications if they are using the same personal access token. As a security best practice, do not share a single token across multiple applications. Create a personal access token for each application. For more information, see Creating a personal access token for the command line on the GitHub website.

**To create a GitHub personal access token and then update the pipeline structure with the new token**

1. In GitHub, from the drop-down option on your profile photo, choose **Settings**.
2. Choose **Developer settings**, and then choose **Personal access tokens**.
3. Choose **Generate new token**.
4. Under **Select scopes**, select `admin:repo_hook` and `repo`.
5. Choose **Generate token**.

6. Next to the generated token, choose the copy icon.

   **Note**
   Make sure you copy your generated token now. You cannot view the token after you close this page.

7. At a terminal (Linux, macOS, or Unix) or command prompt (Windows), run the `get-pipeline` command on the pipeline where you want to change the OAuth token, and then copy the output of the command to a JSON file. For example, for a pipeline named `MyFirstPipeline`, you would type something similar to the following:

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

   The output of the command is sent to the `pipeline.json` file.

8. Open the file in a plain-text editor and edit the value in the `OAuthTokenField` of your GitHub action.

   When you use the AWS CLI to create the pipeline, you can pass your GitHub personal access token in this field. Replace the asterisks (****) with the token you copied from GitHub. When you run `get-pipeline` to view the action configuration, the four-asterisk mask is displayed for this value. For example, for a personal access token with the value `111222333444555666777888EXAMPLE`:

   ```json
   "configuration": {
     "Owner": "MyGitHubUserName",
     "Repo": "test-repo",
     "Branch": "master",
     "OAuthToken": "111222333444555666777888EXAMPLE"
   }
   ```
Note
When you use an AWS CloudFormation template to create the pipeline, you must first store the token as a secret in AWS Secrets Manager. You include the value for this field as a dynamic reference to the stored secret in Secrets Manager. For an example, see GitHub (p. 486).

9. If you are working with the pipeline structure retrieved using the `get-pipeline` command, you must modify the structure in the JSON file by removing the metadata lines from the file. Otherwise, the `update-pipeline` command cannot use it. Remove the section from the pipeline structure in the JSON file (the "metadata": {} lines and the "created", "pipelineARN", and "updated" fields).

For example, remove the following lines from the structure:

```json
"metadata": {
    "created": "date",
    "updated": "date"
}
```

10. Save the file, and then run the `update-pipeline` with the `--cli-input-json` parameter to specify the JSON file you just edited.

For example, to update a pipeline named MyFirstPipeline, you would type something similar to the following:

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

Important
Be sure to include `file://` before the file name. It is required in this command.

11. Repeat steps 6 through 8 for every pipeline that contains a GitHub action.

12. When you are finished, delete the JSON files used to update those pipelines.

Use GitHub and the CodePipeline CLI to create and rotate your GitHub personal access token on a regular basis

The advantage of using tokens instead of passwords in a script is that tokens can be revoked or rotated. You can also grant specific privileges and permissions to a personal access token. Tokens should be stored securely and rotated or regenerated routinely. Token rotation is recommended by RFC-6819 (OAuth 2.0 Threat Model and Security Considerations), section 5.1.5.3.

For more information, see Creating a personal access token for the command line on the GitHub website.

After you have regenerated a new personal access token, you can rotate it by using the AWS CLI or API or by using AWS CloudFormation and calling `UpdatePipeline`.

Note
You might have to update other applications if they are using the same personal access token. As a security best practice, do not share a single token across multiple applications. Create a new personal access token for each application.

Use these steps to rotate your GitHub personal access token and then update the pipeline structure with the new token.

Note
After you rotate your personal access token, remember to update any AWS CLI scripts or AWS CloudFormation templates that contain the old token information.
1. In GitHub, from the drop-down option on your profile photo, choose **Settings**.
2. Choose **Developer settings**, and then choose **Personal access tokens**.
3. Next to your GitHub personal access token, choose **Edit**.

   ![GitHub Personal Access Tokens](image)

4. Choose **Regenerate token**.

   ![Regenerate Token](image)

5. Next to the regenerated token, choose the copy icon.
6. At a terminal (Linux, macOS, or Unix) or command prompt (Windows), run the `get-pipeline` command on the pipeline where you want to change the personal access token, and then copy the output of the command to a JSON file. For example, for a pipeline named MyFirstPipeline, you would type something similar to the following:

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

The output of the command is sent to the `pipeline.json` file.

7. Open the file in a plain-text editor and edit the value in the `OAuthTokenField` of your GitHub action.

When you use the AWS CLI to create the pipeline, you can pass your GitHub personal access token in this field. Replace the asterisks (****) with the token you copied from GitHub. When you run `get-pipeline` to view the action configuration, the four-asterisk mask is displayed for this value. For example, for a personal access token with the value `111222333444555666777888EXAMPLE`:

```json
"configuration": {
  "Owner": "MyGitHubUserName",
  "Repo": "test-repo",
  "Branch": "master",
  "OAuthToken": "111222333444555666777888EXAMPLE"
}
```

**Note**
When you use an AWS CloudFormation template to update the pipeline, you must first store the token as a secret in AWS Secrets Manager. You include the value for this field as a dynamic reference to the stored secret in Secrets Manager. For an example, see GitHub (p. 486).

8. If you are working with the pipeline structure retrieved using the `get-pipeline` command, you must modify the structure in the JSON file by removing the `metadata` lines from the file. Otherwise, the `update-pipeline` command cannot use it. Remove the section from the pipeline structure in the JSON file (the "metadata": `{ } lines and the "created", "pipelineARN", and "updated" fields).

For example, remove the following lines from the structure:

```json
"metadata": {
  "created": "date",
  "updated": "date"
}
```
9. Save the file, and then run `update-pipeline` with the `--cli-input-json` parameter to specify the JSON file you just edited. For example, to update a pipeline named MyFirstPipeline, you would type something similar to the following:

   **Important**
   Be sure to include `file://` before the file name. It is required in this command.

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

10. When you have finished updating your pipelines, delete the JSON files.

   For more information, see Pipeline error: I receive a pipeline error that says: "Could not access the GitHub repository" or "Unable to connect to the GitHub repository" (p. 399).

**Use AWS Secrets Manager to track database passwords or third-party API keys**

We recommend that you use AWS Secrets Manager to rotate, manage, and retrieve database credentials, API keys, and other secrets throughout their lifecycle. Secrets Manager enables you to replace hardcoded credentials in your code (including passwords) with an API call to Secrets Manager to retrieve the secret programmatically. For more information, see What Is AWS Secrets Manager? in the AWS Secrets Manager User Guide.

For pipelines where you pass parameters that are secrets (such as OAuth credentials) in an AWS CloudFormation template, you should include dynamic references in your template that access the secrets you have stored in Secrets Manager. For the reference ID pattern and examples, see Secrets Manager Secrets in the AWS CloudFormation User Guide. For an example that uses dynamic references in a template snippet for GitHub webhook in a pipeline, see Webhook Resource Configuration.

**See also**

The following related resources can help you as you work with managing secrets.

- Secrets Manager can rotate database credentials automatically, such as for rotation of Amazon RDS secrets. For more information, see Rotating Your AWS Secrets Manager Secrets in the AWS Secrets Manager User Guide.

Identity and access management for AWS CodePipeline

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be authenticated (signed in) and authorized (have permissions) to use CodePipeline resources. IAM is an AWS service that you can use with no additional charge.

**Topics**

- Audience (p. 414)
- Authenticating with identities (p. 414)
- Managing access using policies (p. 416)
- How AWS CodePipeline works with IAM (p. 417)
• AWS CodePipeline identity-based policy examples (p. 420)
• AWS CodePipeline resource-based policy examples (p. 436)
• Troubleshooting AWS CodePipeline identity and access (p. 437)
• CodePipeline permissions reference (p. 439)
• Manage the CodePipeline service role (p. 445)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work you do in CodePipeline.

Service user – If you use the CodePipeline service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more CodePipeline features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in CodePipeline, see Troubleshooting AWS CodePipeline identity and access (p. 437).

Service administrator – If you're in charge of CodePipeline resources at your company, you probably have full access to CodePipeline. It's your job to determine which CodePipeline features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with CodePipeline, see How AWS CodePipeline works with IAM (p. 417).

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to CodePipeline. To view example CodePipeline identity-based policies that you can use in IAM, see AWS CodePipeline identity-based policy examples (p. 420).

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see The IAM Console and Sign-in Page in the IAM User Guide.

You must be authenticated (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication, or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email or your IAM user name. You can access AWS programmatically using your root user or IAM user access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use AWS tools, you must sign the request yourself. Do this using Signature Version 4, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 Signing Process in the AWS General Reference.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using Multi-Factor Authentication (MFA) in AWS in the IAM User Guide.

AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account root user and
is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

**IAM users and groups**

An IAM user is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing Access Keys for IAM Users in the IAM User Guide. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An IAM group is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named IAMAdmins and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see **When to Create an IAM User (Instead of a Role)** in the IAM User Guide.

**IAM roles**

An IAM role is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see **Using IAM Roles** in the IAM User Guide.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as federated users. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated Users and Roles in the IAM User Guide.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the IAM User Guide.
- **AWS service access** – A service role is an IAM role that a service assumes to perform actions in your account on your behalf. When you set up some AWS service environments, you must define a role for the service to assume. This service role must include all the permissions that are required for the service to access the AWS resources that it needs. Service roles vary from service to service, but many allow you to choose your permissions as long as you meet the documented requirements for that service. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. You can create, modify, and delete a service role from within IAM. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the IAM User Guide.
• **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances in the *IAM User Guide*.

To learn whether to use IAM roles, see When to Create an IAM Role (Instead of a User) in the *IAM User Guide*.

## Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when an entity (root user, IAM user, or IAM role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON Policies in the *IAM User Guide*.

An IAM administrator can use policies to specify who has access to AWS resources, and what actions they can perform on those resources. Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

### Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, role, or group. These policies control what actions that identity can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM Policies in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing Between Managed Policies and Inline Policies in the *IAM User Guide*.

### Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource such as an Amazon S3 bucket. Service administrators can use these policies to define what actions a specified principal (account member, user, or role) can perform on that resource and under what conditions. Resource-based policies are inline policies. There are no managed resource-based policies.

### Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.
• **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity’s identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions Boundaries for IAM Entities in the *IAM User Guide*.

• **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs Work in the *AWS Organizations User Guide*.

• **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session’s permissions are the intersection of the user or role’s identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session Policies in the *IAM User Guide*.

### How AWS CodePipeline works with IAM

Before you use IAM to manage access to CodePipeline, you should understand what IAM features are available to use with CodePipeline. To get a high-level view of how CodePipeline and other AWS services work with IAM, see AWS Services That Work with IAM in the *IAM User Guide*.

**Topics**

- CodePipeline identity-based policies (p. 417)
- CodePipeline resource-based policies (p. 420)
- Authorization based on CodePipeline tags (p. 420)
- CodePipeline IAM roles (p. 420)

### CodePipeline identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. CodePipeline supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the *IAM User Guide*.

**Actions**

The `Action` element of an IAM identity-based policy describes the specific action or actions that will be allowed or denied by the policy. Policy actions usually have the same name as the associated AWS API operation. The action is used in a policy to grant permissions to perform the associated operation.

Policy actions in CodePipeline use the following prefix before the action: `codepipeline:`.

For example, to grant someone permission to view the existing pipelines in the account, you include the `codepipeline:GetPipeline` action in their policy. Policy statements must include either an `Action` or `NotAction` element. CodePipeline defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:
You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word "Get", include the following action:

"Action": "codepipeline:Get*"

For a list of CodePipeline actions, see Actions Defined by AWS CodePipeline in the IAM User Guide.

Resources

The Resource element specifies the object or objects to which the action applies. Statements must include either a Resource or a NotResource element. You specify a resource using an ARN or using the wildcard (*) to indicate that the statement applies to all resources.

CodePipeline resources and operations

In CodePipeline, the primary resource is a pipeline. In a policy, you use an Amazon Resource Name (ARN) to identify the resource that the policy applies to. CodePipeline supports other resources that can be used with the primary resource, such as stages, actions, and custom actions. These are referred to as subresources. These resources and subresources have unique Amazon Resource Names (ARNs) associated with them. For more information about ARNs, see Amazon Resource Names (ARN) and AWS Service Namespaces in the Amazon Web Services General Reference. To get the pipeline ARN associated with your pipeline, use the CLI to run the `get-pipeline` command. For more information, see GetPipeline in the CodePipeline API Reference.

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>ARN Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline</td>
<td>arn:aws:codepipeline:region:account:pipeline-name</td>
</tr>
<tr>
<td>Stage</td>
<td>arn:aws:codepipeline:region:account:pipeline-name/stage-name</td>
</tr>
<tr>
<td>Action</td>
<td>arn:aws:codepipeline:region:account:pipeline-name/stage-name/action-name</td>
</tr>
<tr>
<td>Custom action</td>
<td>arn:aws:codepipeline:region:account:actionType:owner/category/provider/version</td>
</tr>
<tr>
<td>All CodePipeline resources</td>
<td>arn:aws:codepipeline:*</td>
</tr>
<tr>
<td>All CodePipeline resources owned by the specified account in the specified Region</td>
<td>arn:aws:codepipeline:region:account:*</td>
</tr>
</tbody>
</table>

Note

Most services in AWS treat a colon (:) or a forward slash (/) as the same character in ARNs. However, CodePipeline uses an exact match in event patterns and rules. Be sure to use the correct ARN characters when creating event patterns so that they match the ARN syntax in the pipeline you want to match.

In CodePipeline, there are API calls that support resource-level permissions. Resource-level permissions indicate whether an API call can specify a resource ARN, or whether the API call can only specify all...
resources using the wildcard. See CodePipeline permissions reference (p. 439) for a detailed description of resource-level permissions and a listing of the CodePipeline API calls that support resource-level permissions.

For example, you can indicate a specific pipeline (myPipeline) in your statement using its ARN as follows:

"Resource": "arn:aws:codepipeline:us-east-2:111222333444:myPipeline"

You can also specify all pipelines that belong to a specific account by using the (*) wildcard character as follows:


To specify all resources, or if a specific API action does not support ARNs, use the (*) wildcard character in the Resource element as follows:

"Resource": "*

**Note**

When you create IAM policies, follow the standard security advice of granting least privilege—that is, granting only the permissions required to perform a task. If an API call supports ARNs, then it supports resource-level permissions, and you do not need to use the (*) wildcard character.

Some CodePipeline API calls accept multiple resources (for example, GetPipeline). To specify multiple resources in a single statement, separate their ARNs with commas, as follows:

"Resource": ["arn1", "arn2"]

CodePipeline provides a set of operations to work with the CodePipeline resources. For a list of available operations, see CodePipeline permissions reference (p. 439).

**Condition keys**

The Condition element (or Condition block) lets you specify conditions in which a statement is in effect. The Condition element is optional. You can build conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple Condition elements in a statement, or multiple keys in a single Condition element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM Policy Elements: Variables and Tags in the IAM User Guide.

CodePipeline defines its own set of condition keys and also supports using some global condition keys. To see all AWS global condition keys, see AWS Global Condition Context Keys in the IAM User Guide.

All Amazon EC2 actions support the aws:RequestedRegion and ec2:Region condition keys. For more information, see Example: Restricting Access to a Specific Region.

To see a list of CodePipeline condition keys, see Condition Keys for AWS CodePipeline in the IAM User Guide. To learn with which actions and resources you can use a condition key, see Actions Defined by AWS CodePipeline.
Examples

To view examples of CodePipeline identity-based policies, see AWS CodePipeline identity-based policy examples (p. 420).

CodePipeline resource-based policies

CodePipeline does not support resource-based policies. However, a resource-based policy example for the S3 service related to CodePipeline is provided.

Examples

To view examples of CodePipeline resource-based policies, see AWS CodePipeline resource-based policy examples (p. 436).

Authorization based on CodePipeline tags

You can attach tags to CodePipeline resources or pass tags in a request to CodePipeline. To control access based on tags, you provide tag information in the condition element of a policy using the codepipeline:ResourceTag/key-name, aws:RequestTag/key-name, or aws:TagKeys condition keys. For more information about tagging CodePipeline resources, see Tagging resources (p. 168).

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see Using tags to control access to CodePipeline resources (p. 424).

CodePipeline IAM roles

An IAM role is an entity in your AWS account that has specific permissions.

Using temporary credentials with CodePipeline

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

CodePipeline supports the use of temporary credentials.

Service roles

CodePipeline allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

CodePipeline supports service roles.

AWS CodePipeline identity-based policy examples

By default, IAM users and roles don't have permission to create or modify CodePipeline resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating Policies on the JSON Tab in the IAM User Guide.
Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete CodePipeline resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get Started Using AWS Managed Policies** – To start using CodePipeline quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see [Get Started Using Permissions With AWS Managed Policies](https://docs.aws.amazon.com/IAM/latest/UserGuide/idBarcode-AccessKeyGetStarted.html) in the IAM User Guide.

- **Grant Least Privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see [Grant Least Privilege](https://docs.aws.amazon.com/IAM/latest/UserGuide/idPolicies-least-privilege.html) in the IAM User Guide.

- **Enable MFA for Sensitive Operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see [Using Multi-Factor Authentication (MFA) in AWS](https://docs.aws.amazon.com/IAM/latest/UserGuide/idMFA.html) in the IAM User Guide.

- **Use Policy Conditions for Extra Security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see [IAM JSON Policy Elements: Condition](https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_elements_condition.html) in the IAM User Guide.

Viewing resources in the console

The CodePipeline console requires the `ListRepositories` permission to display a list of repositories for your AWS account in the AWS Region where you are signed in. The console also includes a **Go to resource** function to quickly perform a case insensitive search for resources. This search is performed in your AWS account in the AWS Region where you are signed in. The following resources are displayed across the following services:

- **AWS CodeBuild**: Build projects
- **AWS CodeCommit**: Repositories
- **AWS CodeDeploy**: Applications
- **AWS CodePipeline**: Pipelines

To perform this search across resources in all services, you must have the following permissions:

- **CodeBuild**: `ListProjects`
- **CodeCommit**: `ListRepositories`
Identity-based policy examples

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsForUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": ["arn:aws:iam::*:user/${aws:username}"].
    },
    {
      "Sid": "NavigateInConsole",
      "Effect": "Allow",
      "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:ListUsers"
      ],
      "Resource": "*"
    }
  ]
}
```

Identity-based policies (IAM) examples

You can attach policies to IAM identities. For example, you can do the following:

- Attach a permissions policy to a user or a group in your account — To grant a user permissions to view pipelines in the CodePipeline console, you can attach a permissions policy to a user or group that the user belongs to.

- Attach a permissions policy to a role (grant cross-account permissions) — You can attach an identity-based permissions policy to an IAM role to grant cross-account permissions. For example, the administrator in Account A can create a role to grant cross-account permissions to another AWS account (for example, Account B) or an AWS service as follows:
  1. Account A administrator creates an IAM role and attaches a permissions policy to the role that grants permissions on resources in Account A.
2. Account A administrator attaches a trust policy to the role identifying Account B as the principal who can assume the role.

3. Account B administrator can then delegate permissions to assume the role to any users in Account B. Doing this allows users in Account B to create or access resources in Account A. The principal in the trust policy can also be an AWS service principal if you want to grant an AWS service permissions to assume the role.

For more information about using IAM to delegate permissions, see Access Management in the IAM User Guide.

The following shows an example of a permissions policy that allows a user to enable and disable all stage transitions in the pipeline named `MyFirstPipeline` in the us-west-2 region:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codepipeline:EnableStageTransition",
        "codepipeline:DisableStageTransition"
      ],
      "Resource": [
        "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline"
      ]
    }
  ]
}
```

The following example shows a policy in the 111222333444 account that allows users to view, but not change, the pipeline named `MyFirstPipeline` in the CodePipeline console. This policy is based on the `AWSCodePipelineReadOnlyAccess` managed policy, but because it is specific to the `MyFirstPipeline` pipeline, it cannot use the managed policy directly. If you do not want to restrict the policy to a specific pipeline, consider using one of the managed policies created and maintained by CodePipeline. For more information, see Working with Managed Policies. You must attach this policy to an IAM role you create for access, for example, a role named `CrossAccountPipelineViewers`:

```
{
  "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",
```

API Version 2015-07-09
423
After you create this policy, create the IAM role in the 111222333444 account and attach the policy to that role. In the role's trust relationships, you must add the AWS account that will assume this role. The following example shows a policy that allows users from the 111111111111 AWS account to assume roles defined in the 111222333444 account:

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "AWS": "arn:aws:iam::111111111111:root"
         },
         "Action": "sts:AssumeRole"
      }
   ]
}
```

The following example shows a policy created in the 111111111111 AWS account that allows users to assume the role named CrossAccountPipelineViewers in the 111222333444 account:

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": "sts:AssumeRole",
         "Resource": "arn:aws:iam::111222333444:role/CrossAccountPipelineViewers"
      }
   ]
}
```

You can create IAM policies to restrict the calls and resources that users in your account have access to, and then attach those policies to IAM users. For more information about how to create IAM roles and to explore example IAM policy statements for CodePipeline, see Customer managed policy examples (p. 430).

### Using tags to control access to CodePipeline resources

Conditions in IAM user policy statements are part of the syntax that you use to specify permissions to resources required by CodePipeline actions. Using tags in conditions is one way to control access to resources and requests. For information about tagging CodePipeline resources, see Tagging resources (p. 168). This topic discusses tag-based access control.

When you design IAM policies, you might be setting granular permissions by granting access to specific resources. As the number of resources that you manage grows, this task becomes more difficult. Tagging resources and using tags in policy statement conditions can make this task easier. You grant access in...
bulk to any resource with a certain tag. Then you repeatedly apply this tag to relevant resources, during creation or later.

Tags can be attached to the resource or passed in the request to services that support tagging. In CodePipeline, resources can have tags, and some actions can include tags. When you create an IAM policy, you can use tag condition keys to control:

- Which users can perform actions on a pipeline resource, based on tags that it already has.
- Which tags can be passed in an action’s request.
- Whether specific tag keys can be used in a request.

For the complete syntax and semantics of tag condition keys, see Controlling Access Using Tags in the IAM User Guide.

The following examples demonstrate how to specify tag conditions in policies for CodePipeline users.

**Example 1: Limit actions based on tags in the request**

The CodePipelineFullAccess managed user policy gives users unlimited permission to perform any CodePipeline action on any resource.

The following policy limits this power and denies unauthorized users permission to create pipelines for specific projects. To do that, it denies the CreatePipeline action if the request specifies a tag named Project with one of the values ProjectA or ProjectB. (The aws:RequestTag condition key is used to control which tags can be passed in an IAM request.) In addition, the policy prevents these unauthorized users from tampering with the resources by using the aws:TagKeys condition key to not allow tag modification actions to include these same tag values or to completely remove the Project tag. A customer’s administrator must attach this IAM policy to unauthorized IAM users, in addition to the managed user policy.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Deny",
         "Action": [
            "codepipeline:CreatePipeline",
            "codepipeline:TagResource"
         ],
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "aws:RequestTag/Project": ["ProjectA", "ProjectB"]
            }
         }
      },
      {
         "Effect": "Deny",
         "Action": ["codepipeline:UntagResource"],
         "Resource": "*",
         "Condition": {
            "ForAllValues:StringEquals": {
               "aws:TagKeys": ["Project"]
            }
         }
      }
   ]
}
```

API Version 2015-07-09
Example 2: Limit actions based on resource tags

The CodePipelineFullAccess managed user policy gives users unlimited permission to perform any CodePipeline action on any resource.

The following policy limits this power and denies unauthorized users permission to perform actions on specified project pipelines. To do that, it denies some actions if the resource has a tag named Project with one of the values ProjectA or ProjectB. (The aws:ResourceTag condition key is used to control access to the resources based on the tags on those resources.) A customer's administrator must attach this IAM policy to unauthorized IAM users, in addition to the managed user policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codepipeline:TagResource",
        "codepipeline:UntagResource",
        "codepipeline:UpdatePipeline",
        "codepipeline:DeletePipeline",
        "codepipeline:ListTagsForResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Project": ["ProjectA", "ProjectB"]
        }
      }
    }
  ]
}
```

Example 3: Allow actions based on tags in the request

The following policy grants users permission to create development pipelines in CodePipeline.

To do that, it allows the CreatePipeline and TagResource actions if the request specifies a tag named Project with the value ProjectA. (The aws:RequestTag condition key is used to control which tags can be passed in an IAM request.) The aws:TagKeys condition ensures tag key case sensitivity. This policy is useful for IAM users who don't have the CodePipelineFullAccess managed user policy attached. The managed policy gives users unlimited permission to perform any CodePipeline action on any resource.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codepipeline:CreatePipeline",
        "codepipeline:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/Project": "ProjectA"
        },
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["Project"]
        }
      }
    }
  ]
}
```
Example 4: Allow actions based on resource tags

The following policy grants users permission to perform actions on, and get information about, project pipelines in CodePipeline.

To do that, it allows specific actions if the pipeline has a tag named `Project` with the value `ProjectA`. (The `aws:RequestTag` condition key is used to control which tags can be passed in an IAM request.) The `aws:TagKeys` condition ensures tag key case sensitivity. This policy is useful for IAM users who don’t have the `CodePipelineFullAccess` managed user policy attached. The managed policy gives users unlimited permission to perform any CodePipeline action on any resource.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codepipeline:UpdatePipeline",
        "codepipeline:DeletePipeline",
        "codepipeline:ListPipelines"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Project": "ProjectA"
        },
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["Project"]
        }
      }
    }
  ]
}
```

Permissions required to use the CodePipeline console

To use CodePipeline in the CodePipeline console, you must have a minimum set of permissions from the following services:

- AWS Identity and Access Management
- Amazon Simple Storage Service

These permissions allow you to describe other AWS resources for your AWS account.

Depending on the other services you incorporate into your pipelines, you might need permissions from one or more of the following:

- AWS CodeCommit
- CodeBuild
- AWS CloudFormation
- AWS CodeDeploy
- AWS Elastic Beanstalk
- AWS Lambda
- AWS OpsWorks
If you create an IAM policy that is more restrictive than the minimum required permissions, the console won't function as intended for users with that IAM policy. To ensure that those users can still use the CodePipeline console, also attach the `AWSCodePipelineReadOnlyAccess` managed policy to the user, as described in AWS managed (predefined) policies for CodePipeline (p. 428).

You don't need to allow minimum console permissions for users who are making calls to the AWS CLI or the CodePipeline API.

**AWS managed (predefined) policies for CodePipeline**

AWS addresses many common use cases by providing standalone IAM policies that are created and administered by AWS. Managed policies grant necessary permissions for common use cases so you can avoid having to investigate which permissions are needed. For more information, see AWS Managed Policies in the IAM User Guide.

The following AWS managed policies, which you can attach to users in your account, are specific to CodePipeline:

- **AWSCodePipelineFullAccess** – Grants full access to CodePipeline.
- **AWSCodePipelineCustomActionAccess** – Grants permission to an IAM user to create custom actions in CodePipeline or integrate Jenkins resources for build or test actions.
- **AWSCodePipelineReadOnlyAccess** – Grants read-only access to CodePipeline.
- **AWSCodePipelineApproverAccess** – Grants permission to an IAM user to approve or reject a manual approval action.

**CodePipeline managed policies and notifications**

CodePipeline supports notifications, which can notify users of important changes to pipelines. Managed policies for CodePipeline include policy statements for notification functionality. For more information, see What are notifications?

**Permissions related to notifications in full access managed policies**

The `AWSCodePipelineFullAccess` managed policy includes the following statements to allow full access to notifications. Users with this managed policy applied can also create and manage Amazon SNS topics for notifications, subscribe and unsubscribe users to topics, list topics to choose as targets for notification rules, and list AWS Chatbot clients configured for Slack.

```json
{
    "Sid": "CodeStarNotificationsReadWriteAccess",
    "Effect": "Allow",
    "Action": [
        "codestar-notifications:CreateNotificationRule",
        "codestar-notifications:DescribeNotificationRule",
        "codestar-notifications:UpdateNotificationRule",
        "codestar-notifications:DeleteNotificationRule",
        "codestar-notifications:Subscribe",
        "codestar-notifications:Unsubscribe"
    ],
    "Resource": "*",
    "Condition": {
        "StringLike": {
            "codestar-notifications:NotificationsForResource": "arn:aws:codepipeline:*"
        }
    }
},
{
    "Sid": "CodeStarNotificationsListAccess",
    "Effect": "Allow",
    "Action": [
        "codestar-notifications:DescribeNotificationRule",
        "codestar-notifications:ListNotificationRules"
    ],
    "Resource": "*",
    "Condition": {
        "StringLike": {
            "codestar-notifications:NotificationsForResource": "arn:aws:codepipeline:*"
        }
    }
}
```
Permissions related to notifications in read-only managed policies

The `AWSCodePipelineReadOnlyAccess` managed policy includes the following statements to allow read-only access to notifications. Users with this policy applied can view notifications for resources, but cannot create, manage, or subscribe to them.

```json

{  
    "Sid": "CodeStarNotificationsPowerUserAccess",
    "Effect": "Allow",
    "Action": [  
        "codestar-notifications:DescribeNotificationRule"
    ],
    "Resource": "*",
    "Condition": {  
        "StringLike": {"codestar-notifications:NotificationsForResource" :  
            "arn:aws:codepipeline:*"}
    }
},

{  
    "Sid": "CodeStarNotificationsListAccess",
    "Effect": "Allow",
    "Action": [  
        "codestar-notifications:ListNotificationRules",
        "codestar-notifications:ListEventTypes",
        "codestar-notifications:ListTagsForResource",
        "codestar-notifications:ListTargets"
    ],
    "Resource": "*"
}

```
Customer managed policy examples

In this section, you can find example user policies that grant permissions for various CodePipeline actions. These policies work when you are using the CodePipeline API, AWS SDKs, or the AWS CLI. When you are using the console, you must grant additional permissions specific to the console. For more information, see Permissions required to use the CodePipeline console (p. 427).

Note
All examples use the US West (Oregon) Region (us-west-2) and contain fictitious account IDs.

Examples

- Example 1: Grant permissions to get the state of a pipeline (p. 430)
- Example 2: Grant permissions to enable and disable transitions between stages (p. 430)
- Example 3: Grant permissions to get a list of all available action types (p. 431)
- Example 4: Grant permissions to approve or reject manual approval actions (p. 431)
- Example 5: Grant permissions to poll for jobs for a custom action (p. 431)
- Example 6: Attach or edit a policy for Jenkins integration with AWS CodePipeline (p. 432)
- Example 7: Configure cross-account access to a pipeline (p. 432)
- Example 8: Use AWS resources associated with another account in a pipeline (p. 433)

Example 1: Grant permissions to get the state of a pipeline

The following example grants permissions to get the state of the pipeline named MyFirstPipeline:

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "codepipeline:GetPipelineState"
         ],
      }
   ]
}
```

Example 2: Grant permissions to enable and disable transitions between stages

The following example grants permissions to disable and enable transitions between all stages in the pipeline named MyFirstPipeline:

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "codepipeline:DisableStageTransition",
            "codepipeline:EnableStageTransition"
         ],
      }
   ]
}
```
To allow the user to disable and enable transitions for a single stage in a pipeline, you must specify the stage. For example, to allow the user to enable and disable transitions for a stage named Staging in a pipeline named MyFirstPipeline:

```
```

**Example 3: Grant permissions to get a list of all available action types**

The following example grants permissions to get a list of all available action types available for pipelines in the us-west-2 Region:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["codepipeline:ListActionTypes"],
    }
  ]
}
```

**Example 4: Grant permissions to approve or reject manual approval actions**

The following example grants permissions to approve or reject manual approval actions in a stage named Staging in a pipeline named MyFirstPipeline:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["codepipeline:PutApprovalResult"],
    }
  ]
}
```

**Example 5: Grant permissions to poll for jobs for a custom action**

The following example grants permissions to poll for jobs for the custom action named TestProvider, which is a Test action type in its first version, across all pipelines:

**Note**

The job worker for a custom action might be configured under a different AWS account or require a specific IAM role in order to function.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["codepipeline:PollForJobs"],
    }
  ]
}
```
Example 6: Attach or edit a policy for Jenkins integration with AWS CodePipeline

If you configure a pipeline to use Jenkins for build or test, create a separate IAM user for that integration and attach an IAM policy that has the minimum permissions required for integration between Jenkins and CodePipeline. This policy is the same as the AWSCodePipelineCustomActionAccess managed policy. The following example shows a policy to attach to an IAM user for Jenkins integration:

```json
{
   "Statement": [
   {
      "Action": [
      "codepipeline:AcknowledgeJob",
      "codepipeline:GetJobDetails",
      "codepipeline:PollForJobs",
      "codepipeline:PutJobFailureResult",
      "codepipeline:PutJobSuccessResult"
      ],
      "Effect": "Allow",
      "Resource": "*"
   },
   
   "Version": "2012-10-17"
}]
}
```

Example 7: Configure cross-account access to a pipeline

You can configure access to pipelines for users and groups in another AWS account. The recommended way is to create a role in the account where the pipeline was created. The role should allow users from the other AWS account to assume that role and access the pipeline. For more information, see Walkthrough: Cross-Account Access Using Roles.

The following example shows a policy in the 80398EXAMPLE account that allows users to view, but not change, the pipeline named MyFirstPipeline in the CodePipeline console. This policy is based on the AWSCodePipelineReadOnlyAccess managed policy, but because it is specific to the MyFirstPipeline pipeline, it cannot use the managed policy directly. If you do not want to restrict the policy to a specific pipeline, consider using one of the managed policies created and maintained by CodePipeline. For more information, see Working with Managed Policies. You must attach this policy to an IAM role you create for access, for example, a role named CrossAccountPipelineViewers:

```json
{
   "Statement": [
   {
      "Action": [
      "codepipeline:GetPipeline",
      "codepipeline:GetPipelineState",
      "codepipeline:ListActionTypes",
      "codepipeline:ListPipelines",
      "iam:ListRoles",
      "s3:GetBucketPolicy",
      "s3:GetObject",
      "s3:ListBucket"
      ],
      "Effect": "Allow",
      "Resource": "*"
   }]
   
   "Version": "2012-10-17"
}]

---

API Version 2015-07-09
432
Identity-based policy examples

```json
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": "sts:AssumeRole",
"Resource": "arn:aws:iam::80398EXAMPLE:role/CrossAccountPipelineViewers"
}
]
}
```

After you create this policy, create the IAM role in the 80398EXAMPLE account and attach the policy to that role. In the role's trust relationships, you must add the AWS account that assumes this role. The following example shows a policy that allows users from the 111111111111 AWS account to assume roles defined in the 80398EXAMPLE account:

```json
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Principal": { "AWS": "arn:aws:iam::111111111111:root" },
"Action": "sts:AssumeRole"
}
]
}
```

The following example shows a policy created in the 111111111111 AWS account that allows users to assume the role named CrossAccountPipelineViewers in the 80398EXAMPLE account:

```json
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Action": "sts:AssumeRole",
"Resource": "arn:aws:iam::80398EXAMPLE:role/CrossAccountPipelineViewers"
}
]
}
```

Example 8: Use AWS resources associated with another account in a pipeline

You can configure policies that allow a user to create a pipeline that uses resources in another AWS account. This requires configuring policies and roles in both the account that creates the pipeline (AccountA) and the account that created the resources to be used in the pipeline (AccountB). You must also create a customer managed key in AWS Key Management Service to use for cross-account access. For more information and step-by-step examples, see Create a pipeline in CodePipeline that uses resources from another AWS account (p. 251) and Data protection configuration (p. 405).

The following example shows a policy configured by AccountA for an S3 bucket used to store pipeline artifacts. The policy grants access to AccountB. In the following example, the ARN for AccountB is
The ARN for the S3 bucket is codepipeline-us-east-2-1234567890. Replace these ARNs with the ARNs for the S3 bucket and the account you want to allow access:

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

The following example shows a policy configured by AccountA that allows AccountB to assume a role. This policy must be applied to the service role for CodePipeline (AWS-CodePipeline-Service). For more information about how to apply policies to roles in IAM, see Modifying a Role. In the following example, 012ID_ACCOUNT_B is the ARN for AccountB:

```
{
   "Version": "2012-10-17",
   "Statement": {
```
The following example shows a policy configured by AccountB and applied to the EC2 instance role for CodeDeploy. This policy grants access to the S3 bucket used by AccountA to store pipeline artifacts (codepipeline-us-east-2-1234567890):

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

The following example shows a policy for AWS KMS where arn:aws:kms:us-east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE is the ARN of the customer managed key created in AccountA and configured to allow AccountB to use it:

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

The following example shows an inline policy for an IAM role (CrossAccount_Role) created by AccountB that allows access to CodeDeploy actions required by the pipeline in AccountA.
Resource-based policy examples

The following example shows an inline policy for an IAM role (CrossAccount_Role) created by AccountB that allows access to the S3 bucket to download input artifacts and upload output artifacts:

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

For more information about how to edit a pipeline for cross-account access to resources, see Step 2: Edit the pipeline (p. 258).

AWS CodePipeline resource-based policy examples

Other services, such as Amazon S3, also support resource-based permissions policies. For example, you can attach a policy to an S3 bucket to manage access permissions to that bucket. Although CodePipeline doesn't support resource-based policies, it does store artifacts to be used in pipelines in versioned S3 buckets.

Example To create a policy for an S3 bucket to use as the artifact store for CodePipeline

You can use any versioned S3 bucket as the artifact store for CodePipeline. If you use the Create Pipeline wizard to create your first pipeline, this S3 bucket is created for you to ensure that all objects uploaded to the artifact store are encrypted and connections to the bucket are secure. If you create your own S3 bucket, as a best practice, consider adding the following policy or its elements to the bucket. In this policy, the ARN for the S3 bucket is codepipeline-us-east-2-1234567890. Replace this ARN with the ARN for your S3 bucket:

```
{
  "Version": "2012-10-17",
  "Id": "SSEAndSSLPolicy",
}
```
Troubleshooting AWS CodePipeline identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with CodePipeline and IAM.

**Topics**
- I am not authorized to perform an action in CodePipeline (p. 437)
- I am not authorized to perform iam:PassRole (p. 438)
- I want to view my access keys (p. 438)
- I’m an administrator and want to allow others to access CodePipeline (p. 438)
- I want to allow people outside of my AWS account to access my CodePipeline resources (p. 438)

**I am not authorized to perform an action in CodePipeline**

If the AWS Management Console tells you that you’re not authorized to perform an action, you must contact your administrator for assistance. Your administrator is the person who provided you with your user name and password.

The following example error occurs when the `mateo.jackson` IAM user tries to use the console to view details about a pipeline, but does not have `codepipeline:GetPipeline` permissions.

User: arn:aws:iam::123456789012:user/mateo.jackson is not authorized to perform: codepipeline:GetPipeline on resource: my-pipeline

In this case, Mateo asks his administrator to update his policies to allow him to access the `my-pipeline` resource using the `codepipeline:GetPipeline` action.
**I am not authorized to perform iam:PassRole**

If you receive an error that you’re not authorized to perform the `iam:PassRole` action, you must contact your administrator for assistance. Your administrator is the person who provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to CodePipeline.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in CodePipeline. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

**I want to view my access keys**

After you create your IAM user access keys, you can view your access key ID at any time. However, you can’t view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUttnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

> **Important**
> Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing Access Keys in the `IAM User Guide`.

**I’m an administrator and want to allow others to access CodePipeline**

To allow others to access CodePipeline, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in CodePipeline.

To get started right away, see Creating Your First IAM Delegated User and Group in the `IAM User Guide`.

**I want to allow people outside of my AWS account to access my CodePipeline resources**

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:
To learn whether CodePipeline supports these features, see How AWS CodePipeline works with IAM (p. 417).

To learn how to provide access to your resources across AWS accounts that you own, see Providing Access to an IAM User in Another AWS Account That You Own in the IAM User Guide.

To learn how to provide access to your resources to third-party AWS accounts, see Providing Access to AWS Accounts Owned by Third Parties in the IAM User Guide.

To learn how to provide access through identity federation, see Providing Access to Externally Authenticated Users (Identity Federation) in the IAM User Guide.

To learn the difference between using roles and resource-based policies for cross-account access, see How IAM Roles Differ from Resource-based Policies in the IAM User Guide.

## CodePipeline permissions reference

Use the following table as a reference when you are setting up access control and writing permissions policies that you can attach to an IAM identity (identity-based policies). The table lists each CodePipeline API operation and the corresponding actions for which you can grant permissions to perform the action. For operations that support resource-level permissions, the table lists the AWS resource for which you can grant the permissions. You specify the actions in the policy's `Action` field.

*Resource-level permissions* are those that allow you to specify which resources users are allowed to perform actions on. AWS CodePipeline provides partial support for resource-level permissions. This means that for some AWS CodePipeline API calls, you can control when users are allowed to use those actions based on conditions that must be met, or which resources users are allowed to use. For example, you can grant users permission to list pipeline execution information, but only for a specific pipeline or pipelines.

### Note

The `Resources` column lists the resource required for API calls that support resource-level permissions. For API calls that do not support resource-level permissions, you can grant users permission to use it, but you have to specify a wildcard (*) for the resource element of your policy statement.

### CodePipeline API Operations and Required Permissions for Actions

#### AcknowledgeJob

**Action:** codepipeline:AcknowledgeJob

Required to view information about a specified job and whether that job has been received by the job worker. Used for custom actions only.

**Resources:** Supports only a wildcard (*) in the policy `Resource` element.

#### AcknowledgeThirdPartyJob

**Action:** codepipeline:AcknowledgeThirdPartyJob

Required to confirms a job worker has received the specified job. Used for partner actions only.

**Resources:** Supports only a wildcard (*) in the policy `Resource` element.

#### CreateCustomActionType

**Action:** codepipeline:CreateCustomActionType

Required to create a new custom action that can be used in all pipelines associated with the AWS account. Used for custom actions only.

**Resources:**

Action Type
CreatePipeline

**Action:** codepipeline:CreatePipeline

Required to create a pipeline.

**Resources:**

Pipeline

arn:aws:codepipeline:region:account:actionType:owner/category/provider/version

DeleteCustomActionType

**Action:** codepipeline:DeleteCustomActionType

Required to mark a custom action as deleted. PollForJobs for the custom action fails after the action is marked for deletion. Used for custom actions only.

**Resources:**

Action Type

arn:aws:codepipeline:region:account:actionType:owner/category/provider/version

DeletePipeline

**Action:** codepipeline:DeletePipeline

Required to delete a pipeline.

**Resources:**

Pipeline

arn:aws:codepipeline:region:account:pipeline-name

DeleteWebhook

**Action:** codepipeline:DeleteWebhook

Required to delete a webhook.

**Resources:**

Webhook

arn:aws:codepipeline:region:account:webhook:webhook-name

DeregisterWebhookWithThirdParty

**Action:** codepipeline:DeregisterWebhookWithThirdParty

Before a webhook is deleted, required to remove the connection between the webhook that was created by CodePipeline and the external tool with events to be detected. Currently supported only for webhooks that target an action type of GitHub.

**Resources:**

Webhook

arn:aws:codepipeline:region:account:webhook:webhook-name

DisableStageTransition

**Action:** codepipeline:DisableStageTransition

Required to prevent artifacts in a pipeline from transitioning to the next stage in the pipeline.
Resources:

Pipeline

Arn: aws:codepipeline:region:account:pipeline-name

EnableStageTransition

Action: codepipeline:EnableStageTransition

Required to enable artifacts in a pipeline to transition to a stage in a pipeline.

Resources:

Pipeline

Arn: aws:codepipeline:region:account:pipeline-name

GetJobDetails

Action: codepipeline:GetJobDetails

Required to retrieve information about a job. Only used for custom actions.

Resources: No resource required.

GetPipeline

Action: codepipeline:GetPipeline

Required to retrieve the structure, stages, actions, and metadata of a pipeline, including the pipeline ARN.

Resources:

Pipeline

Arn: aws:codepipeline:region:account:pipeline-name

GetPipelineExecution

Action: codepipeline:GetPipelineExecution

Required to retrieve information about an execution of a pipeline, including details about artifacts, the pipeline execution ID, and the name, version, and status of the pipeline.

Resources:

Pipeline

Arn: aws:codepipeline:region:account:pipeline-name

GetPipelineState

Action: codepipeline:GetPipelineState

Required to retrieve information about the state of a pipeline, including the stages and actions.

Resources:

Pipeline

Arn: aws:codepipeline:region:account:pipeline-name

GetThirdPartyJobDetails

Action: codepipeline:GetThirdPartyJobDetails

Required to request the details of a job for a third-party action. Used for partner actions only.

Resources: Supports only a wildcard (*) in the policy Resource element.
ListActionTypes

**Action:** `codepipeline:ListActionTypes`

Required to generate a summary of all CodePipeline action types associated with your account.

**Resources:**

Action Type

```
arn:aws:codepipeline:region:account:actionType:owner/category/provider/version
```

ListPipelineExecutions

**Action:** `codepipeline:ListPipelineExecutions`

Required to generate a summary of the most recent executions for a pipeline.

**Resources:**

Pipeline

```
arn:aws:codepipeline:region:account:pipeline-name
```

ListPipelines

**Action:** `codepipeline:ListPipelines`

Required to generate a summary of all of the pipelines associated with your account.

**Resources:**

Pipeline ARN with wildcard (resource-level permissions at the pipeline name level are not supported)

```
arn:aws:codepipeline:region:account:*
```

ListTagsForResource

**Action:** `codepipeline:ListTagsForResource`

Required to list tags for a specified resource.

**Resources:**

Action Type

```
arn:aws:codepipeline:region:account:actionType:owner/category/provider/version
```

Pipeline

```
arn:aws:codepipeline:region:account:pipeline-name
```

Webhook

```
arn:aws:codepipeline:region:account:webhook:webhook-name
```

ListWebhooks

**Action:** `codepipeline:ListWebhooks`

Required to list all of the webhooks in the account for that Region.

**Resources:**

Webhook

```
arn:aws:codepipeline:region:account:webhook:webhook-name
```

PollForJobs

**Action(s):** `codepipeline:PollForJobs`
Required to retrieve information about any jobs for CodePipeline to act on.

**Resources:**

Action Type

```text
arn:aws:codepipeline:region:account:actionType:owner/category/provider/version
```

**PollForThirdPartyJobs**

**Action:** codepipeline:PollForThirdPartyJobs

Required to determine whether there are any third-party jobs for a job worker to act on. Used for partner actions only.

**Resources:** Supports only a wildcard (*) in the policy Resource element.

**PutActionRevision**

**Action:** codepipeline:PutActionRevision

Required to report information to CodePipeline about new revisions to a source.

**Resources:**

Action

```text
arn:aws:codepipeline:region:account:pipeline-name/stage-name/action-name
```

**PutApprovalResult**

**Action:** codepipeline:PutApprovalResult

Required to report the response to a manual approval request to CodePipeline. Valid responses are Approved and Rejected.

**Resources:**

Action

```text
arn:aws:codepipeline:region:account:pipeline-name/stage-name/action-name
```

**Note**

This API call supports resource-level permissions. However, you might encounter an error if you use the IAM console or Policy Generator to create policies with "codepipeline:PutApprovalResult" that specify a resource ARN. If you encounter an error, you can use the JSON tab in the IAM console or the CLI to create a policy.

**PutJobFailureResult**

**Action:** codepipeline:PutJobFailureResult

Required to report the failure of a job as returned to the pipeline by a job worker. Used for custom actions only.

**Resources:** Supports only a wildcard (*) in the policy Resource element.

**PutJobSuccessResult**

**Action:** codepipeline:PutJobSuccessResult

Required to report the success of a job as returned to the pipeline by a job worker. Used for custom actions only.

**Resources:** Supports only a wildcard (*) in the policy Resource element.

**PutThirdPartyJobFailureResult**

**Action:** codepipeline:PutThirdPartyJobFailureResult
Required to report the failure of a third-party job as returned to the pipeline by a job worker. Used for partner actions only.

**Resources**: Supports only a wildcard (*) in the policy Resource element.

**PutThirdPartyJobSuccessResult**

**Action**: codepipeline:PutThirdPartyJobSuccessResult

Required to report the success of a third-party job as returned to the pipeline by a job worker. Used for partner actions only.

**Resources**: Supports only a wildcard (*) in the policy Resource element.

**PutWebhook**

**Action**: codepipeline:PutWebhook

Required to create a webhook.

**Resources**:

Webhook

arn:aws:codepipeline:region:account:webhook:webhook-name

**RegisterWebhookWithThirdParty**

**Action**: codepipeline:RegisterWebhookWithThirdParty

Resources:

After a webhook is created, required to configure supported third parties to call the generated webhook URL.

Webhook

arn:aws:codepipeline:region:account:webhook:webhook-name

**RetryStageExecution**

**Action**: codepipeline:RetryStageExecution

Required to resume the pipeline execution by retrying the last failed actions in a stage.

**Resources**:

Pipeline

arn:aws:codepipeline:region:account:pipeline-name

**StartPipelineExecution**

**Action**: codepipeline:StartPipelineExecution

Required to start the specified pipeline (specifically, to start processing the latest commit to the source location specified as part of the pipeline).

**Resources**:

Pipeline

arn:aws:codepipeline:region:account:pipeline-name

**TagResource**

**Action**: codepipeline:TagResource

Required to tag the specified resource.
Resources:

Action Type

Pipeline

Webhook

UntagResource

Action: codepipeline:UntagResource

Required to tag the specified resource.

Resources:

Action Type

Pipeline

Webhook

UpdatePipeline

Action: codepipeline:UpdatePipeline

Required to update a specified pipeline with edits or changes to its structure.

Resources:

Pipeline

Manage the CodePipeline service role

The CodePipeline service role is configured with one or more policies that control access to the AWS resources used by the pipeline. You might want to attach more policies to this role, edit the policy attached to the role, or configure policies for other service roles in AWS. You might also want to attach a policy to a role when you configure cross-account access to your pipeline.

Important

Modifying a policy statement or attaching another policy to the role can prevent your pipelines from functioning. Be sure that you understand the implications before you modify the service role for CodePipeline in any way. Make sure you test your pipelines after you make any change to the service role.

Note

In the console, service roles created before September 2018 are created with the name oneClick_AWS-CodePipeline-Service_ID-Number. Service roles created after September 2018 use the service role name format AWSCodePipelineServiceRole-Region-Pipeline_Name. For example, for a pipeline...
named MyFirstPipeline created in the console in eu-west-2, the service role is named AWSCodePipelineServiceRole-eu-west-2-MyFirstPipeline.

Remove permissions from the CodePipeline service role

You can edit the service role statement to remove access to resources you do not use. For example, if none of your pipelines include Elastic Beanstalk, you can edit the policy statement to remove the section that grants access to Elastic Beanstalk resources.

Similarly, if none of your pipelines includes CodeDeploy, you can edit the policy statement to remove the section that grants access to CodeDeploy resources:

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

Add permissions to the CodePipeline service role

You must update your service role policy statement with permissions for an AWS service not already included in the default service role policy statement before you can use it in your pipelines.

This is especially important if the service role you use for your pipelines was created before support was added to CodePipeline for an AWS service.

The following table shows when support was added for other AWS services.
Follow these steps to add permissions for a supported service:

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the IAM console, in the navigation pane, choose Roles, and then choose your AWS-CodePipeline-Service role from the list of roles.
3. On the Permissions tab, in Inline Policies, in the row for your service role policy, choose Edit Policy.

   **Note**
   Your service role has a name in a format similar to oneClick_AWS-CodePipeline-1111222233334.

4. Add the required permissions in the Policy Document box.

   **Note**
   When you create IAM policies, follow the standard security advice of granting least privilege—that is, granting only the permissions required to perform a task. Some API calls support resource-based permissions and allow access to be limited. For example, in this case, to limit permissions when calling DescribeTasks and ListTasks, you can replace the wildcard character (*) with a resource ARN or with a resource ARN that contains a wildcard character (*).

   For example, for CodeCommit support, add the following to your policy statement:

   ```json
   {
   "Action": [
   "codecommit:GetBranch",
   "codecommit:GetCommit",
   "codecommit:UploadArchive",
   "codecommit:GetUploadArchiveStatus",
   "codecommit:CancelUploadArchive"
   ],
   "Resource": "*",
   "Effect": "Allow"
   },
   ```

   For AWS OpsWorks support, add the following to your policy statement:

   ```json
   {
   "Action": [
   "opsworks:CreateDeployment",
   "opsworks:DescribeApps",
   "opsworks:DescribeCommands",
   "opsworks:DescribeDeployments",
   "opsworks:DescribeInstances",
   "opsworks:DescribeStacks",
   "opsworks:UpdateApp",
   "opsworks:UpdateStack"
   ],
   "Resource": "*",
   "Effect": "Allow"
   },
   ```

   For AWS CloudFormation support, add the following to your policy statement:

   ```json
   {
   "Action": [
   "cloudformation:CreateStack",
   "cloudformation:DeleteStack",
   "cloudformation:DescribeStacks",
   "cloudformation:UpdateStack",
   ```
Manage the CodePipeline service role

For CodeBuild support, add the following to your policy statement:

```json
{
    "Action": [
        "codebuild:BatchGetBuilds",
        "codebuild:StartBuild"
    ],
    "Resource": "*",
    "Effect": "Allow"
}
```

**Note**
Support for batch builds was added at a later date. See step 11 for the permissions to add to the service role for batch builds.

For AWS Device Farm support, add the following to your policy statement:

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

For AWS Service Catalog support, add the following to your policy statement:

```json
{
    "Effect": "Allow",
    "Action": [
        "servicecatalog:ListProvisioningArtifacts",
        "servicecatalog:CreateProvisioningArtifact",
        "servicecatalog:DescribeProvisioningArtifact",
        "servicecatalog:DeleteProvisioningArtifact",
        "servicecatalog:UpdateProduct"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "cloudformation:ValidateTemplate"
    ],
    "Resource": "*"
}
```
5. For Amazon ECR support, add the following to your policy statement:

```json
{
  "Action": [
    "ecr:DescribeImages"
  ],
  "Resource": "*",
  "Effect": "Allow"
}

6. For Amazon ECS, the following are the minimum permissions needed to create pipelines with an Amazon ECS deploy action.

```json
{
  "Action": [
    "ecs:DescribeServices",
    "ecs:DescribeTaskDefinition",
    "ecs:DescribeTasks",
    "ecs:ListTasks",
    "ecs:RegisterTaskDefinition",
    "ecs:UpdateService"
  ],
  "Resource": "*",
  "Effect": "Allow"
}
```

7. For the CodeDeployToECS action (blue/green deployments), the following are the minimum permissions needed to create pipelines with a CodeDeploy to Amazon ECS blue/green deployment action.

```json
{
  "Action": [
    "codedeploy:CreateDeployment",
    "codedeploy:GetDeployment",
    "codedeploy:GetApplication",
    "codedeploy:GetApplicationRevision",
    "codedeploy:RegisterApplicationRevision",
    "codedeploy:GetDeploymentConfig",
    "ecs:RegisterTaskDefinition",
    "iam:PassRole"
  ],
  "Resource": "*",
  "Effect": "Allow"
}
```

8. For AWS CodeStar connections, the following permission is required to create pipelines with a source that uses a connection, such as Bitbucket.

```json
{
  "Action": [
    "codestar-connections:UseConnection"
  ],
  "Resource": "*",
  "Effect": "Allow"
}
```

For more information about the IAM permissions for connections, see Connections permissions reference.
9. For the StepFunctions action, the following are the minimum permissions needed to create pipelines with a Step Functions invoke action.

```json
{
    "Action": [
        "states:DescribeStateMachine",
        "states:DescribeExecution",
        "states:StartExecution"
    ],
    "Resource": "*",
    "Effect": "Allow"
}
```

10. For the AppConfig action, the following are the minimum permissions needed to create pipelines with an AWS AppConfig invoke action.

```json
{
    "Action": [
        "appconfig:StartDeployment",
        "appconfig:GetDeployment",
        "appconfig:StopDeployment"
    ],
    "Resource": "*",
    "Effect": "Allow"
}
```

11. For CodeBuild support for batch builds, add the following to your policy statement:

```json
{
    "Action": [
        "codebuild:BatchGetBuildBatches",
        "codebuild:StartBuildBatch"
    ],
    "Resource": "*",
    "Effect": "Allow"
}
```

12. Choose Validate Policy to ensure the policy contains no errors. When the policy is error-free, choose Apply Policy.

### Logging and monitoring in CodePipeline

You can use logging features in AWS to determine the actions users have taken in your account and the resources that were used. The log files show:

- The time and date of actions.
- The source IP address for an action.
- Which actions failed due to inadequate permissions.

Logging features are available in the following AWS services:

- AWS CloudTrail can be used to log AWS API calls and related events made by or on behalf of an AWS account. For more information, see [Logging CodePipeline API calls with AWS CloudTrail](p. 394).
- Amazon CloudWatch Events can be used to monitor your AWS Cloud resources and the applications you run on AWS. You can create alerts in Amazon CloudWatch Events based on metrics that you define. For more information, see [Detect and react to changes in pipeline state with Amazon CloudWatch Events](p. 384).
Compliance validation for AWS CodePipeline

Third-party auditors assess the security and compliance of AWS CodePipeline as part of multiple AWS compliance programs. These include PCI, ISO, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using CodePipeline is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- **Security and Compliance Quick Start Guides** – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- **Architecting for HIPAA Security and Compliance Whitepaper** – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- **AWS Compliance Resources** – This collection of workbooks and guides might apply to your industry and location.
- **Evaluating Resources with Rules in the AWS Config Developer Guide** – AWS Config assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- **AWS Security Hub** – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Resilience in AWS CodePipeline

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

Infrastructure security in AWS CodePipeline

As a managed service, AWS CodePipeline is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access CodePipeline through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.
Security best practices

CodePipeline provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don’t represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

You use encryption and authentication for the source repositories that connect to your pipelines. These are the CodePipeline best practices for security:

- If you create a pipeline that uses an S3 source bucket, configure server-side encryption for artifacts stored in Amazon S3 for CodePipeline by managing AWS KMS managed keys (SSE-KMS), as described in Configure server-side encryption for artifacts stored in Amazon S3 for CodePipeline (p. 405).
- If you create a pipeline that uses a GitHub source repository, configure GitHub authentication. You can use an AWS managed OAuth token or a customer managed personal access token, as described in Configure GitHub authentication (p. 407).
- If you are using the Jenkins action provider, when you use a Jenkins build provider for your pipeline's build or test action, install Jenkins on an EC2 instance and configure a separate EC2 instance profile. Make sure that the instance profile grants Jenkins only the AWS permissions required to perform tasks for your project, such as retrieving files from Amazon S3. To learn how to create the role for your Jenkins instance profile, see the steps in Create an IAM role to use for Jenkins integration (p. 64).
AWS CodePipeline command line reference

Use this reference when working with the AWS CodePipeline commands and as a supplement to information documented in the AWS CLI User Guide and the AWS CLI Reference.

Before you use the AWS CLI, make sure you complete the prerequisites in Getting started with CodePipeline (p. 19).

To view a list of all available CodePipeline commands, run the following command:

```
aws codepipeline help
```

To view information about a specific CodePipeline command, run the following command, where `command-name` is the name of one of the commands listed below (for example, create-pipeline):

```
aws codepipeline command-name help
```

To begin learning how to use the commands in the CodePipeline extension to the AWS CLI, go to one or more of the following sections:

- Create a custom action (p. 325)
- Create a pipeline (CLI) (p. 228)
- Delete a pipeline (CLI) (p. 251)
- Disable or enable transitions (CLI) (p. 383)
- View pipeline details and history (CLI) (p. 244)
- Retry failed actions (CLI) (p. 353)
- Start a pipeline manually (CLI) (p. 216)
- Edit a pipeline (AWS CLI) (p. 233)

You can also view examples of how to use most of these commands in CodePipeline tutorials (p. 37).
CodePipeline pipeline structure reference

By default, any pipeline you successfully create in AWS CodePipeline has a valid structure. However, if you manually create or edit a JSON file to create a pipeline or update a pipeline from the AWS CLI, you might inadvertently create a structure that is not valid. The following reference can help you better understand the requirements for your pipeline structure and how to troubleshoot issues. See the constraints in Quotas in AWS CodePipeline (p. 521), which apply to all pipelines.

Topics

- Valid action types and providers in CodePipeline (p. 454)
- Pipeline and stage structure requirements in CodePipeline (p. 457)
- Action structure requirements in CodePipeline (p. 459)

Valid action types and providers in CodePipeline

The pipeline structure format is used to build actions and stages in a pipeline. An action type consists of an action category and provider type.

The following are the valid action categories in CodePipeline:

- Source
- Build
- Test
- Deploy
- Approval
- Invoke

Each action category has a designated set of providers. Each action provider, such as Amazon S3, has a provider name, such as S3, that must be used in the Provider field in the action category in your pipeline structure.

There are three valid values for the Owner field in the action category section in your pipeline structure: AWS, ThirdParty, and Custom.

To find the provider name and owner information for your action provider, see Action structure reference (p. 470) or Number of input and output artifacts for each action type (p. 465).

This table lists valid providers by action type.

<table>
<thead>
<tr>
<th>Action category</th>
<th>Valid action providers</th>
<th>Action reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Amazon S3</td>
<td>Amazon S3 (p. 496)</td>
</tr>
<tr>
<td></td>
<td>Amazon ECR</td>
<td>Amazon ECR (p. 490)</td>
</tr>
<tr>
<td></td>
<td>CodeCommit</td>
<td>CodeCommit (p. 479)</td>
</tr>
<tr>
<td>Action category</td>
<td>Valid action providers</td>
<td>Action reference</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>CodeStarSourceConnection (Bitbucket)</td>
<td>[CodeStarSourceConnection](p. 482)</td>
</tr>
<tr>
<td></td>
<td>GitHub</td>
<td>[GitHub](p. 486)</td>
</tr>
<tr>
<td>Build</td>
<td>CodeBuild</td>
<td>[AWS CodeBuild](p. 475)</td>
</tr>
<tr>
<td></td>
<td>Custom CloudBees</td>
<td>Number of input and output artifacts for each action type (p. 465)</td>
</tr>
<tr>
<td></td>
<td>Custom Jenkins</td>
<td>Number of input and output artifacts for each action type (p. 465)</td>
</tr>
<tr>
<td></td>
<td>Custom TeamCity</td>
<td>Number of input and output artifacts for each action type (p. 465)</td>
</tr>
<tr>
<td>Test</td>
<td>CodeBuild</td>
<td>[AWS CodeBuild](p. 475)</td>
</tr>
<tr>
<td></td>
<td>AWS Device Farm</td>
<td>Number of input and output artifacts for each action type (p. 465)</td>
</tr>
<tr>
<td></td>
<td>Custom BlazeMeter</td>
<td>Number of input and output artifacts for each action type (p. 465)</td>
</tr>
<tr>
<td></td>
<td>ThirdParty GhostInspector</td>
<td>Number of input and output artifacts for each action type (p. 465)</td>
</tr>
<tr>
<td></td>
<td>Custom Jenkins</td>
<td>Number of input and output artifacts for each action type (p. 465)</td>
</tr>
<tr>
<td></td>
<td>ThirdParty Micro Focus StormRunner Load</td>
<td>Number of input and output artifacts for each action type (p. 465)</td>
</tr>
<tr>
<td>Action category</td>
<td>Valid action providers</td>
<td>Action reference</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td></td>
<td>ThirdParty Nouvola</td>
<td>Number of input and output artifacts for each action type (p. 465)</td>
</tr>
<tr>
<td></td>
<td>ThirdParty Runscope</td>
<td>Number of input and output artifacts for each action type (p. 465)</td>
</tr>
<tr>
<td>Deploy</td>
<td>Amazon S3</td>
<td>Number of input and output artifacts for each action type (p. 465)</td>
</tr>
<tr>
<td></td>
<td>AWS CloudFormation</td>
<td>AWS CloudFormation (p. 470)</td>
</tr>
<tr>
<td></td>
<td>CodeDeploy</td>
<td>Number of input and output artifacts for each action type (p. 465)</td>
</tr>
<tr>
<td></td>
<td>Amazon ECS</td>
<td>Number of input and output artifacts for each action type (p. 465)</td>
</tr>
<tr>
<td></td>
<td>Amazon ECS (Blue/Green) (this is the CodeDeployToECS action)</td>
<td>Number of input and output artifacts for each action type (p. 465)</td>
</tr>
<tr>
<td></td>
<td>Elastic Beanstalk</td>
<td>Number of input and output artifacts for each action type (p. 465)</td>
</tr>
<tr>
<td></td>
<td>AWS AppConfig</td>
<td>AWS AppConfig (p. 504)</td>
</tr>
<tr>
<td></td>
<td>AWS OpsWorks</td>
<td>Number of input and output artifacts for each action type (p. 465)</td>
</tr>
</tbody>
</table>
### Action category

<table>
<thead>
<tr>
<th>Valid action providers</th>
<th>Action reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Service Catalog</td>
<td>Number of input and output artifacts for each action type (p. 465)</td>
</tr>
<tr>
<td>Amazon Alexa</td>
<td>Number of input and output artifacts for each action type (p. 465)</td>
</tr>
<tr>
<td>Custom XebiaLabs</td>
<td>Number of input and output artifacts for each action type (p. 465)</td>
</tr>
<tr>
<td>Approval</td>
<td>Number of input and output artifacts for each action type (p. 465)</td>
</tr>
<tr>
<td>Manual</td>
<td></td>
</tr>
<tr>
<td>Invoke AWS Lambda</td>
<td>AWS Lambda (p. 492)</td>
</tr>
<tr>
<td>AWS Step Functions</td>
<td>AWS Step Functions (p. 499)</td>
</tr>
</tbody>
</table>

Some action types in CodePipeline are available in select AWS Regions only. It is possible that an action type is available in an AWS Region, but an AWS provider for that action type is not available.

For more information about each action provider, see Integrations with CodePipeline action types (p. 22).

The following sections provide examples for provider information and configuration properties for each action type.

## Pipeline and stage structure requirements in CodePipeline

A two-stage pipeline has the following basic structure:

```json
{
    "roleArn": "An IAM ARN for a service role, such as arn:aws:iam::80398EXAMPLE:role/AWS-CodePipeline-Service",
    "stages": [
        {
            "name": "SourceStageName",
            "actions": [
                "... See Action structure requirements in CodePipeline ...
            ]
        },
        {
```
The pipeline structure has the following requirements:

- A pipeline must contain at least two stages.
- The first stage of a pipeline must contain at least one source action. It can contain source actions only.
- Only the first stage of a pipeline can contain source actions.
- At least one stage in each pipeline must contain an action that is not a source action.
- All stage names in a pipeline must be unique.
- Stage names cannot be edited in the CodePipeline console. If you edit a stage name by using the AWS CLI, and the stage contains an action with one or more secret parameters (such as an OAuth token), the value of those secret parameters is not preserved. You must manually enter the value of the parameters (which are masked by four asterisks in the JSON returned by the AWS CLI) and include them in the JSON structure.
- The `artifactStore` field contains the artifact bucket type and location for a pipeline with all actions in the same AWS Region. If you add actions in a Region different from your pipeline, the `artifactStores` mapping is used to list the artifact bucket for each AWS Region where actions are executed. When you create or edit a pipeline, you must have an artifact bucket in the pipeline Region and then you must have one artifact bucket per Region where you plan to execute an action.

The following example shows the basic structure for a pipeline with cross-Region actions that uses the `artifactStores` parameter:

```json
"pipeline": {
   "name": "YourPipelineName",
   "roleArn": "ServiceRoleARN",
   "artifactStores": {
      "us-east-1": {
         "type": "S3",
         "location": "S3 artifact bucket name, such as codepipeline-us-east-1-1234567890"
      },
      "us-west-2": {
         "type": "S3",
         "location": "S3 artifact bucket name, such as codepipeline-us-west-2-1234567890"
      }
   },
   "stages": [
   ...
```

- The pipeline metadata fields are distinct from the pipeline structure and cannot be edited. When you update a pipeline, the date in the `updated` metadata field changes automatically.
• When you edit or update a pipeline, the pipeline name cannot be changed.

  **Note**
  If you want to rename an existing pipeline, you can use the CLI `get-pipeline` command to build a JSON file that contains your pipeline's structure. You can then use the CLI `create-pipeline` command to create a pipeline with that structure and give it a new name.

The version number of a pipeline is automatically generated and updated every time you update the pipeline.

### Action structure requirements in CodePipeline

An action has the following high-level structure:

```json
[
  {
    "inputArtifacts": [ An input artifact structure, if supported for the action category ],
    "name": "ActionName",
    "region": "Region",
    "namespace": "source_namespace",
    "actionTypeId": {
      "category": "An action category",
      "owner": "AWS",
      "version": "1",
      "provider": "A provider type for the action category",
    },
    "outputArtifacts": [ An output artifact structure, if supported for the action category ],
    "configuration": {
      Configuration details appropriate to the provider type
    },
    "runOrder": A positive integer that indicates the run order within the stage,
  }
]
```

For a list of example configuration details appropriate to the provider type, see [Configuration details by provider type (p. 467)](https://docs.aws.amazon.com/codepipeline/latest/userguide/).  

The action structure has the following requirements:

• All action names within a stage must be unique.

• The input artifact of an action must exactly match the output artifact declared in a preceding action.
  For example, if a preceding action includes the following declaration:

```json
"outputArtifacts": [
  {
    "MyApp"
  }
],
```

and there are no other output artifacts, then the input artifact of a following action must be:

```json
"inputArtifacts": [
  {
  
```

API Version 2015-07-09
459
This is true for all actions, whether they are in the same stage or in following stages, but the input artifact does not have to be the next action in strict sequence from the action that provided the output artifact. Actions in parallel can declare different output artifact bundles, which are, in turn, consumed by different following actions.

The following illustration provides an example of input and output artifacts in actions in a pipeline:

- Output artifact names must be unique in a pipeline. For example, a pipeline can include one action that has an output artifact named "MyApp" and another action that has an output artifact named "MyBuiltApp". However, a pipeline cannot include two actions that both have an output artifact named "MyApp".
- Cross-Region actions use the Region field to designate the AWS Region where the actions are to be created. The AWS resources created for this action must be created in the same Region provided in the region field. You cannot create cross-Region actions for the following action types:
  - Source actions
  - Actions by third-party providers
  - Actions by custom providers
- Actions can be configured with variables. You use the namespace field to set the namespace and variable information for execution variables. For reference information about execution variables and action output variables, see Variables (p. 514).
- For all currently supported action types, the only valid version string is "1".
- For all currently supported action types, the only valid owner string is "AWS", "ThirdParty", or "Custom". For more information, see the CodePipeline API Reference.
- The default runOrder value for an action is 1. The value must be a positive integer (natural number). You cannot use fractions, decimals, negative numbers, or zero. To specify a serial sequence of actions, use the smallest number for the first action and larger numbers for each of the rest of the actions in
sequence. To specify parallel actions, use the same integer for each action you want to run in parallel. In the console, you can specify a serial sequence for an action by choosing **Add action group** at the level in the stage where you want it to run, or you can specify a parallel sequence by choosing **Add action**. **Action group** refers to a run order of one or more actions at the same level.

For example, if you want three actions to run in sequence in a stage, you would give the first action the **runOrder** value of 1, the second action the **runOrder** value of 2, and the third the **runOrder** value of 3. However, if you want the second and third actions to run in parallel, you would give the first action the **runOrder** value of 1 and both the second and third actions the **runOrder** value of 2.

**Note**
The numbering of serial actions do not have to be in strict sequence. For example, if you have three actions in a sequence and decide to remove the second action, you do not need to renumber the **runOrder** value of the third action. Because the **runOrder** value of that action (3) is higher than the **runOrder** value of the first action (1), it runs serially after the first action in the stage.

• When you use an Amazon S3 bucket as a deployment location, you also specify an object key. An object key can be a file name (object) or a combination of a prefix (folder path) and file name. You can use variables to specify the location name you want the pipeline to use. Amazon S3 deployment actions support the use of the following variables in Amazon S3 object keys.

**Using variables in Amazon S3**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Example of console input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>uuid</td>
<td>js-application/(uuid).zip</td>
<td>The UUID is a globally unique identifier that is guaranteed to be different from any other identifier. The UUID is in this format (all digits in hexadecimal format): $&lt;8$-digits-$&lt;4$-digits-$&lt;4$-digits-$&lt;4$-digits-$&lt;12$-digits&lt;br&gt;Example:&lt;br&gt;js-application/54a60075-b96a-4bf3-9013-db3a9EXAMPLE.zip</td>
</tr>
</tbody>
</table>

• These are the valid **actionTypeId** categories for CodePipeline:
  • Source
  • Build
  • Approval
  • Deploy
  • Test
  • Invoke

Some provider types and configuration options are provided here.

• Valid provider types for an action category depend on the category. For example, for a source action type, a valid provider type is **S3**, **GitHub**, **CodeCommit**, or **Amazon ECR**. This example shows the structure for a source action with an **S3** provider:
Every action must have a valid action configuration, which depends on the provider type for that action. The following table lists the required action configuration elements for each valid provider type:

**Action configuration properties for provider types**

<table>
<thead>
<tr>
<th>Name of provider</th>
<th>Provider name in action type</th>
<th>Configuration properties</th>
<th>Required property?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon S3 (Deploy action provider)</td>
<td>S3</td>
<td>BucketName</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Extract</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ObjectKey</td>
<td>Required if Extract = false</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KMSEncryptionKeyARN</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CannedACL</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CacheControl</td>
<td>Optional</td>
</tr>
<tr>
<td>Amazon ECR</td>
<td>For more information, including examples related to Amazon ECR parameters, see Amazon ECR (p. 490).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CodeCommit</td>
<td>For more information, including examples related to CodeCommit parameters, see CodeCommit (p. 479).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GitHub</td>
<td>For more information, including examples related to GitHub parameters, see GitHub (p. 486).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amazon S3 (Source action provider)</td>
<td>For more information, including examples related to Amazon S3 source action parameters, see Amazon S3 (p. 496).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AWS CloudFormation</td>
<td>For more information, including examples related to AWS CloudFormation parameters, see AWS CloudFormation (p. 470).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CodeBuild</td>
<td>For more description and examples related to CodeBuild parameters, see AWS CodeBuild (p. 475).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CodeDeploy</td>
<td>CodeDeploy</td>
<td>ApplicationName</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DeploymentGroupName</td>
<td>Required</td>
</tr>
<tr>
<td>AWS Device Farm</td>
<td>DeviceFarm</td>
<td>RecordAppPerformanceData</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AppType</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ProjectId</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>App</td>
<td>Required</td>
</tr>
<tr>
<td>Name of provider</td>
<td>Provider name in action type</td>
<td>Configuration properties</td>
<td>Required property?</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------</td>
<td>------------------------------------------------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RadioBluetoothEnabled</td>
<td>Default=true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RecordVideo</td>
<td>Default=true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RadioWifiEnabled</td>
<td>Default=true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RadioNfcEnabled</td>
<td>Default=true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RadioGpsEnabled</td>
<td>Default=true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Test</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DevicePoolArn</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TestType</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AppiumVersion</td>
<td>Required</td>
</tr>
<tr>
<td>AWS Elastic Beanstalk</td>
<td>ElasticBeanstalk</td>
<td>ApplicationName</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EnvironmentName</td>
<td>Required</td>
</tr>
<tr>
<td>AWS Lambda</td>
<td></td>
<td>For more information, including examples related to AWS Lambda parameters, see AWS Lambda (p. 492).</td>
<td></td>
</tr>
<tr>
<td>AWS OpsWorks Stacks</td>
<td>OpsWorks</td>
<td>Stack</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Layer</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>App</td>
<td>Required</td>
</tr>
<tr>
<td>Amazon ECS</td>
<td>ECS</td>
<td>ClusterName</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ServiceName</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FileName</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DeploymentTimeout^4</td>
<td>Optional</td>
</tr>
<tr>
<td>Amazon ECS (Blue/Green)</td>
<td>CodeDeployToECS^5</td>
<td>ApplicationName</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DeploymentGroupName</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TaskDefinitionTemplateArtifact</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AppSpecTemplateArtifact</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AppSpecTemplatePath</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TaskDefinitionTemplatePath</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Image1ArtifactName</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Image1ContainerName</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Image2ArtifactName</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Image2ContainerName</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Image3ArtifactName</td>
<td>Optional</td>
</tr>
</tbody>
</table>

API Version 2015-07-09

463
<table>
<thead>
<tr>
<th>Name of provider</th>
<th>Provider name in action type</th>
<th>Configuration properties</th>
<th>Required property?</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Service Catalog</td>
<td>ServiceCatalog</td>
<td>TemplateFilePath, ProductVersionName, ProductType, ProductVersionDescription, ProductId</td>
<td>Required</td>
</tr>
<tr>
<td>Alexa Skills Kit</td>
<td>AlexaSkillsKit</td>
<td>ClientId, ClientSecret, RefreshToken, SkillId</td>
<td>Required</td>
</tr>
<tr>
<td>Jenkins</td>
<td>The name of the action you provided in the CodePipeline Plugin for Jenkins (for example, MyJenkinsProviderName)</td>
<td>ProjectName</td>
<td>Required</td>
</tr>
<tr>
<td>Manual Approval</td>
<td>Manual</td>
<td>CustomData, ExternalEntityLink, NotificationArn</td>
<td>Optional</td>
</tr>
</tbody>
</table>
The `KMSEncryptionKeyARN` parameter encrypts uploaded artifacts with the provided KMS key. For an AWS KMS key, you can use the key ID, the key ARN, or the alias ARN.

**Note**
To specify a customer master key (CMK) in a different AWS account, you must use the key ARN or alias ARN.

**Important**
CodePipeline only supports symmetric customer master keys (CMKs). Do not use an asymmetric CMK to encrypt the data in your S3 bucket.

The `CannedACL` parameter applies the specified canned ACL to objects deployed to Amazon S3. This overwrites any existing ACL that was applied to the object.

The `CacheControl` parameter controls caching behavior for requests/responses for objects in the bucket. For a list of valid values, see the `Cache-Control` header field for HTTP operations. To enter multiple values in `CacheControl`, use a comma between each value. You can add a space after each comma (optional), as shown in this example for the CLI:

```
"CacheControl": "public, max-age=0, no-transform"
```

You can use the `DeploymentTimeout` configuration field to set the Amazon ECS deployment action timeout (in minutes). The timeout is configurable up to the maximum default timeout for this action. For example: "DeploymentTimeout": "15"

For a tutorial that demonstrates creating a pipeline with several of these parameters, including `Image1ContainerName`, see Step 6: Create your pipeline (p. 125).

### Number of input and output artifacts for each action type

Depending on the action type, you can have the following number of input and output artifacts:

**Action type constraints for artifacts**

<table>
<thead>
<tr>
<th>Owner</th>
<th>Type of action</th>
<th>Provider</th>
<th>Valid number of input artifacts</th>
<th>Valid number of output artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS</td>
<td>Source</td>
<td>Amazon S3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>AWS</td>
<td>Source</td>
<td>CodeCommit</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>AWS</td>
<td>Source</td>
<td>Amazon ECR</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>ThirdParty</td>
<td>Source</td>
<td>GitHub</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>AWS</td>
<td>Build</td>
<td>CodeBuild</td>
<td>1 to 5</td>
<td>0 to 5</td>
</tr>
</tbody>
</table>
### Default settings for the PollForSourceChanges parameter

The `PollForSourceChanges` parameter default is determined by the method used to create the pipeline, as described in the following table. In many cases, the `PollForSourceChanges` parameter defaults to true and must be disabled. When the `PollForSourceChanges` parameter defaults to true, you should do the following:

- Add the `PollForSourceChanges` parameter to the JSON file or AWS CloudFormation template.
- Create change detection resources (CloudWatch Events rule or webhook, as applicable).
- Set the `PollForSourceChanges` parameter to false.

**Note**

If you create a CloudWatch Events rule or webhook, you must set the parameter to false to avoid triggering the pipeline more than once.

The `PollForSourceChanges` parameter is not used for Amazon ECR source actions.

<table>
<thead>
<tr>
<th>Owner</th>
<th>Type of action</th>
<th>Provider</th>
<th>Valid number of input artifacts</th>
<th>Valid number of output artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Test</td>
<td>Test</td>
<td>CodeBuild</td>
<td>1 to 5</td>
<td>0 to 5</td>
</tr>
<tr>
<td>AWS Test</td>
<td>Test</td>
<td>AWS Device Farm</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>AWS Approval</td>
<td>Approval</td>
<td>Manual</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>AWS Deploy</td>
<td>Deploy</td>
<td>Amazon S3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>AWS Deploy</td>
<td>Deploy</td>
<td>AWS CloudFormation</td>
<td>0 to 10</td>
<td>0 to 1</td>
</tr>
<tr>
<td>AWS Deploy</td>
<td>Deploy</td>
<td>CodeDeploy</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>AWS Deploy</td>
<td>Deploy</td>
<td>AWS Elastic Beanstalk</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>AWS Deploy</td>
<td>Deploy</td>
<td>AWS OpsWorks Stacks</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>AWS Deploy</td>
<td>Deploy</td>
<td>Amazon ECS</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>AWS Deploy</td>
<td>Deploy</td>
<td>AWS Service Catalog</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>AWS Invoke</td>
<td>Invoke</td>
<td>AWS Lambda</td>
<td>0 to 5</td>
<td>0 to 5</td>
</tr>
<tr>
<td>ThirdParty</td>
<td>Deploy</td>
<td>Alexa Skills Kit</td>
<td>1 to 2</td>
<td>0</td>
</tr>
<tr>
<td>Custom</td>
<td>Build</td>
<td>Jenkins</td>
<td>0 to 5</td>
<td>0 to 5</td>
</tr>
<tr>
<td>Custom</td>
<td>Test</td>
<td>Jenkins</td>
<td>0 to 5</td>
<td>0 to 5</td>
</tr>
<tr>
<td>Custom</td>
<td>Any supported category</td>
<td>As specified in the custom action</td>
<td>0 to 5</td>
<td>0 to 5</td>
</tr>
</tbody>
</table>
### PollForSourceChanges parameter defaults

<table>
<thead>
<tr>
<th>Source</th>
<th>Creation method</th>
<th>Example &quot;configuration&quot; JSON structure output</th>
</tr>
</thead>
</table>
| CodeCommit  | Pipeline is created with the console (and change detection resources are created by the console). The parameter is displayed in the pipeline structure output and defaults to false. | BranchName": "master",
"PollForSourceChanges": "false",
"RepositoryName": "my-repo" |
|             | Pipeline is created with the CLI or AWS CloudFormation, and the PollForSourceChanges parameter is not displayed in JSON output, but it sets to true.² | BranchName": "master",
"RepositoryName": "my-repo" |
| Amazon S3   | Pipeline is created with the console (and change detection resources are created by the console). The parameter is displayed in the pipeline structure output and defaults to false. | "S3Bucket": "my-bucket",
"S3ObjectKey": "object.zip",
"PollForSourceChanges": "false" |
|             | Pipeline is created with the CLI or AWS CloudFormation, and the PollForSourceChanges parameter is not displayed in JSON output, but it sets to true.² | "S3Bucket": "my-bucket",
"S3ObjectKey": "object.zip" |
| GitHub      | Pipeline is created with the console (and change detection resources are created by the console). The parameter is displayed in the pipeline structure output and defaults to false. | "Owner": 
"MyGitHubAccountName",
"Repo": 
"MyGitHubRepositoryName"
"PollForSourceChanges": "false",
"Branch": "master",
"OAuthToken": "****" |
|             | Pipeline is created with the CLI or AWS CloudFormation, and the PollForSourceChanges parameter is not displayed in JSON output, but it sets to true.² | "Owner": 
"MyGitHubAccountName",
"Repo": 
"MyGitHubRepositoryName",
"Branch": "master",
"OAuthToken": "****" |

² If PollForSourceChanges has been added at any point to the JSON structure or the AWS CloudFormation template, it is displayed as shown:

"PollForSourceChanges": "true",

³ For information about the change detection resources that apply to each source provider, see Change Detection Methods (p. 172).

---

**Configuration details by provider type**

This section lists valid configuration parameters for each action provider.

The following example shows a valid configuration for a deploy action that uses Amazon ECS:
"configuration": {
  "ClusterName": "my-ecs-cluster",
  "ServiceName": "sample-app-service",
  "FileName": "imagedefinitions.json"
}

The following example shows a valid configuration for a test action that uses AWS Device Farm:

"configuration": {
  "RecordAppPerformanceData": "true",
  "AppType": "Android",
  "ProjectId": "Project_ID",
  "App": "app-release.apk",
  "RadioBluetoothEnabled": "true",
  "RecordVideo": "true",
  "RadioWifiEnabled": "true",
  "RadioNfcEnabled": "true",
  "RadioGpsEnabled": "true",
  "Test": "tests.zip",
  "DevicePoolArn": "ARN",
  "TestType": "Calabash",
  "AppiumVersion": "1.7.2"
}

The following example shows a valid configuration for a deploy action that uses AWS Service Catalog, for a pipeline that was created in the console without a separate configuration file:

"configuration": {
  "TemplateFilePath": "S3_template.json",
  "ProductVersionName": "devops S3 v2",
  "ProductType": "CLOUD_FORMATION_TEMPLATE",
  "ProductVersionDescription": "Product version description",
  "ProductId": "prod-example123456"
}

The following example shows a valid configuration for a deploy action that uses AWS Service Catalog, for a pipeline that was created in the console with a separate sample_config.json configuration file:

"configuration": {
  "ConfigurationFilePath": "sample_config.json",
  "ProductId": "prod-example123456"
}

The following example shows a valid configuration for a deploy action that uses Alexa Skills Kit:

"configuration": {
  "ClientId": "amzn1.application-oa2-client.aadEXAMPLE",
  "ClientSecret": "****",
  "RefreshToken": "*****",
  "SkillId": "amzn1.ask.skill.22649d8f-0451-4b4b-9ed9-bfb6cEXAMPLE"
}

The following example shows a valid configuration for a deploy action that uses Amazon S3:

"configuration": {
  "BucketName": "website-bucket",
  "Extract": "true",
  "ObjectKey": "MyWebsite"
The following example shows a valid configuration for an Amazon ECS and CodeDeploy blue/green deployment:

```json
"configuration": {
    "ApplicationName": "codedeploy-ecs-application",
    "DeploymentGroupName": "ecs-codedeploy-deplgroup",
    "Image1ArtifactName": "MyImage",
    "TaskDefinitionTemplateArtifact": "SourceArtifact",
    "Image1ContainerName": "IMAGE1_NAME",
    "TaskDefinitionTemplatePath": "taskdef.json",
    "AppSpecTemplateArtifact": "SourceArtifact",
    "AppSpecTemplatePath": "appspec.yaml",
}
```

The following example shows a valid configuration for a manual approval:

```json
"configuration": {
    "CustomData": "Comments on the manual approval",
    "ExternalEntityLink": "http://my-url.com",
}
```
Action structure reference

This section is a reference for action configuration only. For a conceptual overview of the pipeline structure, see CodePipeline pipeline structure reference (p. 454).

Each action provider in CodePipeline uses a set of required and optional configuration fields in the pipeline structure. This section provides the following reference information by action provider:

- Valid values for the ActionType fields included in the pipeline structure action block, such as Owner and Provider.
- Descriptions and other reference information for the Configuration parameters (required and optional) included in the pipeline structure action section.
- Valid example JSON and YAML action fields.

This section is updated periodically with more action providers. Reference information is currently available for the following action providers:

Topics

- AWS CloudFormation (p. 470)
- AWS CodeBuild (p. 475)
- CodeCommit (p. 479)
- CodeStarSourceConnection (p. 482)
- GitHub (p. 486)
- Amazon ECR (p. 490)
- AWS Lambda (p. 492)
- Amazon S3 (p. 496)
- AWS Step Functions (p. 499)
- AWS AppConfig (p. 504)

AWS CloudFormation

Executes an operation on an AWS CloudFormation stack. A stack is a collection of AWS resources that you can manage as a single unit. The resources in a stack are defined by the stack’s AWS CloudFormation template. A change set creates a comparison that can be viewed without altering the original stack. For information about the types of AWS CloudFormation actions that can be performed on stacks and change sets, see the ActionMode parameter.

Topics

- Action type (p. 471)
- Configuration parameters (p. 471)
- Input artifacts (p. 473)
- Output artifacts (p. 474)
- Output variables (p. 474)
- Action declaration (p. 474)
- See also (p. 475)
Action type

- Category: Deploy
- Owner: AWS
- Provider: CloudFormation
- Version: 1

Configuration parameters

**ActionMode**

Required: Yes

ActionMode is the name of the action AWS CloudFormation performs on a stack or change set. The following action modes are available:

- **CHANGE_SET_EXECUTE** executes a change set for the resource stack that is based on a set of specified resource updates. With this action, AWS CloudFormation starts to alter the stack.
- **CHANGE_SET_REPLACE** creates the change set, if it doesn't exist, based on the stack name and template that you submit. If the change set exists, AWS CloudFormation deletes it, and then creates a new one.
- **CREATE_UPDATE** creates the stack if it doesn't exist. If the stack exists, AWS CloudFormation updates the stack. Use this action to update existing stacks. Unlike REPLACE_ON_FAILURE, if the stack exists and is in a failed state, CodePipeline won't delete and replace the stack.
- **DELETE_ONLY** deletes a stack. If you specify a stack that doesn't exist, the action is completed successfully without deleting a stack.
- **REPLACE_ON_FAILURE** creates a stack, if it doesn't exist. If the stack exists and is in a failed state, AWS CloudFormation deletes the stack, and then creates a new stack. If the stack isn't in a failed state, AWS CloudFormation updates it.

The stack is in a failed state when any of the following status types are displayed in AWS CloudFormation:

- ROLLBACK_FAILED
- CREATE_FAILED
- DELETE_FAILED
- UPDATE_ROLLBACK_FAILED

Use this action to automatically replace failed stacks without recovering or troubleshooting them.

**Important**

We recommend that you use REPLACE_ON_FAILURE for testing purposes only because it might delete your stack.

**StackName**

Required: Yes

StackName is the name of an existing stack or a stack that you want to create.

**Capabilities**

Required: Conditional

Use of **Capabilities** acknowledges that the template might have the capabilities to create and update some resources on its own, and that these capabilities are determined based on the types of resources in the template.
This property is required if you have IAM resources in your stack template or you create a stack directly from a template containing macros. In order for the AWS CloudFormation action to successfully operate in this way, you must explicitly acknowledge that you would like it to do so with one of the following capabilities:

- CAPABILITY_IAM
- CAPABILITY_NAMED_IAM
- CAPABILITY_AUTO_EXPAND

You can specify more than one capability by using a comma (no space) between capabilities. The example in Action declaration (p. 474) shows an entry with both the CAPABILITY_IAM and CAPABILITY_AUTO_EXPAND properties.

For more information about Capabilities, see the properties under UpdateStack in the AWS CloudFormation API Reference.

**ChangeSetName**

Required: Conditional

ChangeSetName is the name of an existing change set or a new change set that you want to create for the specified stack.

This property is required for the following action modes: CHANGE_SET_REPLACE and CHANGE_SET_EXECUTE. For all other action modes, this property is ignored.

**RoleArn**

Required: Conditional

The RoleArn is the ARN of the IAM service role that AWS CloudFormation assumes when it operates on resources in the specified stack. RoleArn is not applied when executing a change set. If you do not use CodePipeline to create the change set, make sure that the change set or stack has an associated role.

This property is required for the following action modes:

- CREATE_UPDATE
- REPLACE_ON_FAILURE
- DELETE_ONLY
- CHANGE_SET_REPLACE

**TemplatePath**

Required: Conditional

TemplatePath represents the AWS CloudFormation template file. You include the file in an input artifact to this action. The file name follows this format:

```
ArtifactName::TemplateName
```

ArtifactName is the input artifact name as it appears in CodePipeline. For example, a source stage with the artifact name of SourceArtifact and a template-export.json file name creates a TemplatePath name, as shown in this example:

```
"TemplatePath": "SourceArtifact::template-export.json"
```

This property is required for the following action modes:

- CREATE_UPDATE
- REPLACE_ON_FAILURE
- CHANGE_SET_REPLACE
For all other action modes, this property is ignored.

**Note**
The AWS CloudFormation template file containing the template body has a minimum length of 1 byte and a maximum length of 51,200 bytes. For AWS CloudFormation deployment actions in CodePipeline, the maximum input artifact size is always 256 MB. For more information, see Quotas in AWS CodePipeline (p. 521) and AWS CloudFormation Limits.

**OutputFileName**
Required: No

Use OutputFileName to specify an output file name, such as CreateStackOutput.json, that CodePipeline adds to the pipeline output artifact for this action. The JSON file contains the contents of the Outputs section from the AWS CloudFormation stack.

If you don't specify a name, CodePipeline doesn't generate an output file or artifact.

**ParameterOverrides**
Required: No

Parameters are defined in your stack template and allow you to provide values for them at the time of stack creation or update. You can use a JSON object to set parameter values in your template. (These values override those set in the template configuration file.) For more information about using parameter overrides, see Configuration Properties (JSON Object).

We recommend that you use the template configuration file for most of your parameter values. Use parameter overrides only for values that aren't known until the pipeline is running. For more information, see Using Parameter Override Functions with CodePipeline Pipelines in the AWS CloudFormation User Guide.

**Note**
All parameter names must be present in the stack template.

**TemplateConfiguration**
Required: No

TemplateConfiguration is the template configuration file. You include the file in an input artifact to this action. It can contain template parameter values and a stack policy. For more information about the template configuration file format, see AWS CloudFormation Artifacts.

The template configuration file name follows this format:

Artifactname::TemplateConfigurationFileName

Artifactname is the input artifact name as it appears in CodePipeline. For example, a source stage with the artifact name of SourceArtifact and a test-configuration.json file name creates a TemplateConfiguration name as shown in this example:

"TemplateConfiguration": "SourceArtifact::test-configuration.json"

**Input artifacts**

- **Number of Artifacts:** 0 to 10
- **Description:** As input, the AWS CloudFormation action optionally accepts artifacts for these purposes:
  - To provide the stack template file to execute. (See the TemplatePath parameter.)
• To provide the template configuration file to use. (See the TemplateConfiguration parameter.) For more information about the template configuration file format, see AWS CloudFormation Artifacts.
• To provide the artifact for a Lambda function to be deployed as part of the AWS CloudFormation stack.

Output artifacts

• **Number of Artifacts:** 0 to 1
• **Description:** If the OutputFileName parameter is specified, there is an output artifact produced by this action that contains a JSON file with the specified name. The JSON file contains the contents of the Outputs section from the AWS CloudFormation stack.

For more information about the outputs section you can create for your AWS CloudFormation action, see Outputs.

Output variables

When configured, this action produces variables that can be referenced by the action configuration of a downstream action in the pipeline. You configure an action with a namespace to make those variables available to the configuration of downstream actions.

For AWS CloudFormation actions, variables are produced from any values designated in the Outputs section of a stack template. Note that the only CloudFormation action modes that generate outputs are those that result in creating or updating a stack, such as stack creation, stack updates, and change set execution. The corresponding action modes that generate variables are:

• CREATE_UPDATE
• REPLACE_ON_FAILURE
• ALWAYS_REPLACE
• CHANGE_SET_EXECUTE

For more information, see Variables (p. 514). For a tutorial that shows you how to create a pipeline with a CloudFormation deployment action in a pipeline that uses CloudFormation output variables, see Tutorial: Create a pipeline that uses variables from AWS CloudFormation deployment actions (p. 105).

Action declaration

YAML

```yaml
Name: ExecuteChangeSet
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormation
  Version: '1'
RunOrder: 2
Configuration:
  ActionMode: CHANGE_SET_EXECUTE
  Capabilities: CAPABILITY_NAMED_IAM,CAPABILITY_AUTO_EXPAND
  ChangeSetName: pipeline-changeset
  ParameterOverrides: '{"ProjectId": "my-project","CodeDeployRole": "CodeDeploy_Role_ARN"}'
  RoleArn: CloudFormation_Role_ARN
```
AWS CodePipeline User Guide

See also

JSON

```json
{
    "Name": "ExecuteChangeSet",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormation",
        "Version": "1"
    },
    "RunOrder": 2,
    "Configuration": {
        "ActionMode": "CHANGE_SET_EXECUTE",
        "Capabilities": ["CAPABILITY_NAMED_IAM","CAPABILITY_AUTO_EXPAND"],
        "ChangeSetName": "pipeline-changeset",
        "ParameterOverrides": "{"ProjectId": "my-project","CodeDeployRole": "CodeDeploy_Role_ARN"}",
        "RoleArn": "CloudFormation_Role_ARN",
        "StackName": "my-project--lambda",
        "TemplateConfiguration": "my-project--BuildArtifact::template-configuration.json",
        "TemplatePath": "my-project--BuildArtifact::template-export.yml"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "my-project-BuildArtifact"
        }
    ]
}
```

See also

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

- **Configuration Properties Reference** – This reference chapter in the *AWS CloudFormation User Guide* provides more descriptions and examples for these CodePipeline parameters.
- **AWS CloudFormation API Reference** – The CreateStack parameter in the *AWS CloudFormation API Reference* describes stack parameters for AWS CloudFormation templates.

AWS CodeBuild

Allows you to run builds and tests as part of your pipeline. When you run a CodeBuild build or test action, commands specified in the build spec are run inside of a CodeBuild container. All artifacts that are specified as input artifacts to a CodeBuild action are available inside of the container running the commands. CodeBuild can provide either a build or test action. For more information, see the *AWS CodeBuild User Guide*.

When you use the CodePipeline wizard in the console to create a build project, the CodeBuild build project shows the source provider is CodePipeline. When you create a build project in the CodeBuild
console, you cannot specify CodePipeline as the source provider, but adding the build action to your pipeline adjusts the source in the CodeBuild console. For more information, see ProjectSource in the AWS CodeBuild API Reference.

Topics
- Action type (p. 476)
- Configuration parameters (p. 476)
- Input artifacts (p. 477)
- Output artifacts (p. 477)
- Output variables (p. 478)
- Action declaration (CodeBuild example) (p. 478)
- See also (p. 479)

Action type
- Category: Build or Test
- Owner: AWS
- Provider: CodeBuild
- Version: 1

Configuration parameters

ProjectName

  Required: Yes

  ProjectName is the name of the build project in CodeBuild.

PrimarySource

  Required: Conditional

  The value of the PrimarySource parameter must be the name of one of the input artifacts to the action. CodeBuild looks for the build spec file and runs the build spec commands in the directory that contains the unzipped version of this artifact.

  This parameter is required if multiple input artifacts are specified for a CodeBuild action. When there is only one source artifact for the action, the PrimarySource artifact defaults to that artifact.

BatchEnabled

  Required: No

  The Boolean value of the BatchEnabled parameter allows the action to run multiple builds in the same build execution.

  When this option is enabled, the CombineArtifacts option is available.

  For pipeline examples with batch builds enabled, see CodePipeline integration with CodeBuild and batch builds.

CombineArtifacts

  Required: No

  The Boolean value of the CombineArtifacts parameter combines all build artifacts from a batch build into a single artifact file for the build action.
To use this option, the `BatchEnabled` parameter must be enabled.

**EnvironmentVariables**

Required: No

The value of this parameter is used to set environment variables for the CodeBuild action in your pipeline. The value for the `EnvironmentVariables` parameter takes the form of a JSON array of environment variable objects. See the example parameter in Action declaration (CodeBuild example) (p. 478).

Each object has three parts, all of which are strings:

- **name**: The name or key of the environment variable.
- **value**: The value of the environment variable. When using the `PARAMETER_STORE` or `SECRETS_MANAGER` type, this value must be the name of a parameter you have already stored in AWS Systems Manager Parameter Store or a secret you have already stored in AWS Secrets Manager, respectively.

  **Note**
  We strongly discourage the use of environment variables to store sensitive values, especially AWS secret key IDs and secret access keys. When you use the CodeBuild console or AWS CLI, environment variables are displayed in plain text. For sensitive values, we recommend that you use the `SECRETS_MANAGER` type instead.

- **type**: (Optional) The type of environment variable. Valid values are `PARAMETER_STORE`, `SECRETS_MANAGER`, or `PLAINTEXT`. When not specified, this defaults to `PLAINTEXT`.

  **Note**
  When you enter the `name`, `value`, and `type` for your environment variables configuration, especially if the environment variable contains CodePipeline output variable syntax, do not exceed the 1000-character limit for the configuration's value field. A validation error is returned when this limit is exceeded.

For more information, see EnvironmentVariable.

**Input artifacts**

- **Number of Artifacts**: 1 to 5
- **Description**: CodeBuild looks for the build spec file and runs the build spec commands from the directory of the primary source artifact. When more than one input source is specified for the CodeBuild action, this artifact must be set using the `PrimarySource` action configuration parameter in CodePipeline.

Each input artifact is extracted to its own directory, the locations of which are stored in environment variables. The directory for the primary source artifact is made available with `$CODEBUILD_SRC_DIR`. The directories for all other input artifacts are made available with `$CODEBUILD_SRC_DIR_yourInputArtifactName`.

  **Note**
  The artifact configured in your CodeBuild project becomes the input artifact used by the CodeBuild action in your pipeline.

**Output artifacts**

- **Number of Artifacts**: 0 to 5
- **Description**: These can be used to make the artifacts that are defined in the CodeBuild build spec file available to subsequent actions in the pipeline. When only one output artifact is defined, this artifact
Output variables

This action will produce as variables all environment variables that were exported as part of the build. See [https://docs.aws.amazon.com/codebuild/latest/userguide/build-spec-ref.html#exported-variables-build-spec](https://docs.aws.amazon.com/codebuild/latest/userguide/build-spec-ref.html#exported-variables-build-spec) for more details on how to export environment variables.

For more information about variables in CodePipeline, see [Variables](p. 514).

**Action declaration (CodeBuild example)**

**YAML**

```yaml
Name: Build
Actions:
  - Name: PackageExport
    ActionTypeId:
      Category: Build
      Owner: AWS
      Provider: CodeBuild
      Version: '1'
      RunOrder: 1
    Configuration:
      BatchEnabled: 'true'
      CombineArtifacts: 'true'
      ProjectName: my-build-project
      PrimarySource: MyApplicationSource1
    EnvironmentVariables:
      
      "{{"name":"TEST_VARIABLE","value":"TEST_VALUE","type":"PLAINTEXT"},
      
      "name":"ParamStoreTest","value":"PARAMETER_NAME","type":"PARAMETER_STORE"}}
      
      OutputArtifacts:
        - Name: MyPipeline-BuildArtifact
        InputArtifacts:
          - Name: MyApplicationSource1
          - Name: MyApplicationSource2
```

**JSON**

```json
{
  "Name": "Build",
  "Actions": [
    {
      "Name": "PackageExport",
      "ActionTypeId": {
        "Category": "Build",
        "Owner": "AWS",
        "Provider": "CodeBuild",
```

Note

The artifact configured in your CodeBuild project becomes the CodePipeline input artifact in your pipeline action.

If the `CombineArtifacts` parameter is selected for batch builds, the output artifact location contains the combined artifacts from multiple builds that were run in the same execution.
See also

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

- **AWS CodeBuild User Guide** – For an example pipeline with a CodeBuild action, see Use CodePipeline with CodeBuild to Test Code and Run Builds. For examples of projects with multiple input and output CodeBuild artifacts, see CodePipeline Integration with CodeBuild and Multiple Input Sources and Output Artifacts Sample and Multiple Input Sources and Output Artifacts Sample.
- **Tutorial: Create a pipeline that builds and tests your Android app when a commit is pushed to your GitHub repository (p. 73)** – This tutorial provides a sample build spec file and sample application to create a pipeline with a GitHub source that builds and tests an Android app with CodeBuild and AWS Device Farm.
- **Build Specification Reference for CodeBuild** – This reference topic provides definitions and examples for understanding CodeBuild build spec files.

CodeCommit

Triggers the pipeline when a new commit is made on the configured CodeCommit repository and branch.

If you use the console to create or edit the pipeline, CodePipeline creates a CodeCommit CloudWatch Events rule that starts your pipeline when a change occurs in the repository.

You must have already created a CodeCommit repository before you connect the pipeline through a CodeCommit action.

Topics
- **Action type (p. 480)**
- **Configuration parameters (p. 480)**
Action type

- **Category:** Source
- **Owner:** AWS
- **Provider:** CodeCommit
- **Version:** 1

Configuration parameters

**RepositoryName**

- **Required:** Yes
- The name of the repository where source changes are to be detected.

**BranchName**

- **Required:** Yes
- The name of the branch where source changes are to be detected.

**PollForSourceChanges**

- **Required:** No

*PollForSourceChanges* controls whether CodePipeline polls the CodeCommit repository for source changes. We recommend that you use CloudWatch Events to detect source changes instead. For more information about configuring CloudWatch Events, see Update pipelines for push events (CodeCommit source) (CLI) (p. 267) or Update pipelines for push events (CodeCommit source) (AWS CloudFormation template) (p. 275).

**Important**

If you intend to configure a CloudWatch Events rule, you must set `PollForSourceChanges` to `false` to avoid duplicate pipeline executions.

**Valid values for this parameter:**

- **True:** If set, CodePipeline polls your repository for source changes.

**Note**

If you omit `PollForSourceChanges`, CodePipeline defaults to polling your repository for source changes. This behavior is the same as if `PollForSourceChanges` is included and set to `true`.

- **False:** If set, CodePipeline does not poll your repository for source changes. Use this setting if you intend to configure a CloudWatch Events rule to detect source changes.

Input artifacts

- **Number of Artifacts:** 0
- **Description:** Input artifacts do not apply for this action type.
Output artifacts

- **Number of Artifacts:** 1
- **Description:** The output artifact of this action is a ZIP file that contains the contents of the configured repository and branch at the commit specified as the source revision for the pipeline execution. The artifacts generated from the repository are the output artifacts for the CodeCommit action. The source code commit ID is displayed in CodePipeline as the source revision for the triggered pipeline execution.

Output variables

When configured, this action produces variables that can be referenced by the action configuration of a downstream action in the pipeline. This action produces variables which can be viewed as output variables, even if the action doesn't have a namespace. You configure an action with a namespace to make those variables available to the configuration of downstream actions.

For more information, see Variables (p. 514).

**CommitId**

The CodeCommit commit ID that triggered the pipeline execution. Commit IDs are the full SHA of the commit.

**CommitMessage**

The description message, if any, associated with the commit that triggered the pipeline execution.

**RepositoryName**

The name of the CodeCommit repository where the commit that triggered the pipeline was made.

**BranchName**

The name of the branch for the CodeCommit repository where the source change was made.

**AuthorDate**

The date when the commit was authored, in timestamp format.

For more information about the difference between an author and a committer in Git, see Viewing the Commit History in Pro Git by Scott Chacon and Ben Straub.

**CommitterDate**

The date when the commit was committed, in timestamp format.

For more information about the difference between an author and a committer in Git, see Viewing the Commit History in Pro Git by Scott Chacon and Ben Straub.

Action declaration (CodeCommit example)

**YAML**

```
Actions:
  - OutputArtifacts:
    - Name: Artifact_MyWebsiteStack
InputArtifacts: []
Name: source
Configuration:
  RepositoryName: MyWebsite
  BranchName: mainline
```
See also

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

- Tutorial: Create a simple pipeline (CodeCommit repository) (p. 52) – This tutorial provides a sample app spec file and sample CodeDeploy application and deployment group. Use this tutorial to create a pipeline with a CodeCommit source that deploys to Amazon EC2 instances.

**CodeStarSourceConnection**

Triggers a pipeline when a new commit is made on a third-party source code repository. The source action retrieves code changes when a pipeline is manually executed or when a webhook event is sent from the source provider. Currently, Bitbucket Cloud is the only connection type supported by the CodeStarSourceConnection action.

After a code change is detected, you have the following options for passing the code to subsequent actions:

- Like other existing CodePipeline source actions, CodeStarSourceConnection can output a ZIP file with a shallow copy of your commit.
• CodeStarSourceConnection can also be configured to output a URL reference to the repo for subsequent actions.

Currently, the git URL reference can only be used by downstream CodeBuild actions to clone the repo and associated Git metadata. Attempting to pass a Git URL reference to non-CodeBuild actions results in an error.

CodePipeline prompts you to add a Bitbucket Cloud app to your repo when you create a connection. If you use the console to create or edit your pipeline, CodePipeline creates a Bitbucket webhook that starts your pipeline when a change occurs in the repository. You must have already created your Bitbucket account and repository before you can connect through the CodeStarSourceConnection action. Use your Bitbucket account when you create a connection so that CodePipeline can use the Bitbucket repository for source stages in pipelines.

For more information, see Bitbucket Cloud apps in the Bitbucket developer documentation.

**Note**
To create or attach a policy to your IAM user or role with the permissions required to use AWS CodeStar connections, see Connections permissions reference. Depending on when your CodePipeline service role was created, you might need to update its permissions to support AWS CodeStar connections. For instructions, see Add permissions to the CodePipeline service role (p. 446).

**Topics**
• Action type (p. 483)
• Configuration parameters (p. 483)
• Input artifacts (p. 484)
• Output artifacts (p. 484)
• Action declaration (Bitbucket example) (p. 484)
• Installing the AWS CodeStar app on Bitbucket and creating a connection (p. 485)
• See also (p. 485)

**Action type**
• Category: Source
• Owner: AWS
• Provider: CodeStarSourceConnection
• Version: 1

**Configuration parameters**

**ConnectionArn**

Required: Yes

The connection ARN that is configured and authenticated for the source provider.

**FullRepositoryId**

Required: Yes

The owner and name of the repository where source changes are to be detected.

Example: `some-user/my-repo`
BranchName

Required: Yes

The name of the branch where source changes are to be detected.

OutputArtifactFormat

Required: No

Specifies the output artifact format. Can be either CODEBUILD_CLONE_REF or CODE_ZIP. If unspecified, the default is CODE_ZIP.

**Important**
The CODEBUILD_CLONE_REF option can only be used by CodeBuild downstream actions.

Input artifacts

- **Number of Artifacts:** 0
- **Description:** Input artifacts do not apply for this action type.

Output artifacts

- **Number of Artifacts:** 1
- **Description:** The artifacts generated from the repository are the output artifacts for the CodeStarSourceConnection action. The source code commit ID is displayed in CodePipeline as the source revision for the triggered pipeline execution. You can configure the output artifact of this action in:
  - A ZIP file that contains the contents of the configured repository and branch at the commit specified as the source revision for the pipeline execution.
  - A JSON file that contains a URL reference to the repository so that downstream actions can perform Git commands directly.

**Important**
This option can only be used by CodeBuild downstream actions.

Action declaration (Bitbucket example)

YAML

```yaml
Name: Source
Actions:
  - InputArtifacts: []
    ActionType:
      Version: '1'
      Owner: AWS
      Category: Source
      Provider: CodeStarSourceConnection
    OutputArtifacts:
      - Name: SourceArtifact
    RunOrder: 1
    Configuration:
      FullRepositoryId: "some-user/my-repo"
      BranchName: "master"
```
Installing the AWS CodeStar app on Bitbucket and creating a connection

The first time you use the console to add a new connection to a Bitbucket repository, you must authorize CodePipeline access to your repositories. You choose or create an installation app that helps you connect to the account where you have created your third-party code repository.

When you use the AWS CLI or an AWS CloudFormation template, you must provide the connection ARN of a Bitbucket connection that has already gone through the installation handshake. Otherwise, the pipeline is not triggered.

**Note**

Most source actions in CodePipeline, such as GitHub, require either a configured change detection resource (such as a webhook or CloudWatch Events rule) or use the option to poll the repository for source changes. For pipelines with a Bitbucket Cloud source action, you do not have to set up a webhook or default to polling. The connections action manages your source change detection for you.

**See also**

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

GitHub

Triggers the pipeline when a new commit is made on the configured GitHub repository and branch.

To integrate with GitHub, CodePipeline uses an OAuth application or a personal access token for your pipeline. If you use the console to create or edit your pipeline, CodePipeline creates a GitHub webhook that starts your pipeline when a change occurs in the repository.

You must have already created a GitHub account and repository before you connect the pipeline through a GitHub action.

If you want to limit the access CodePipeline has to repositories, create a GitHub account and grant the account access only to those repositories you want to integrate with CodePipeline. Use that account when you configure CodePipeline to use GitHub repositories for source stages in pipelines.

For more information, see the GitHub developer documentation on the GitHub website.

Topics

- Action type (p. 486)
- Configuration parameters (p. 486)
- Input artifacts (p. 487)
- Output artifacts (p. 487)
- Output variables (p. 488)
- Action declaration (GitHub example) (p. 488)
- Connecting to GitHub (OAuth) (p. 489)
- See also (p. 490)

Action type

- Category: Source
- Owner: ThirdParty
- Provider: GitHub
- Version: 1

Configuration parameters

Owner

Required: Yes

The name of the GitHub user or organization who owns the GitHub repository.

Repo

Required: Yes

The name of the repository where source changes are to be detected.

Branch

Required: Yes

The name of the branch where source changes are to be detected.
OAuthToken

Required: Yes

Represents the GitHub authentication token that allows CodePipeline to perform operations on your GitHub repository. The entry is always displayed as a mask of four asterisks. It represents one of the following values:

- When you use the console to create the pipeline, CodePipeline uses an OAuth token to register the GitHub connection.
- When you use the AWS CLI to create the pipeline, you can pass your GitHub personal access token in this field. Replace the asterisks (****) with your personal access token copied from GitHub. When you run `get-pipeline` to view the action configuration, the four-asterisk mask is displayed for this value.
- When you use an AWS CloudFormation template to create the pipeline, you must first store the token as a secret in AWS Secrets Manager. You include the value for this field as a dynamic reference to the stored secret in Secrets Manager, such as `{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}`.

For more information about GitHub authentication tokens for your pipeline, see Connecting to GitHub (OAuth) (p. 489). For more information about GitHub scopes, see the GitHub Developer API Reference on the GitHub website.

PollForSourceChanges

Required: No

PollForSourceChanges controls whether CodePipeline polls the GitHub repository for source changes. We recommend that you use webhooks to detect source changes instead. For more information about configuring webhooks, see Update pipelines for push events (GitHub source) (CLI) (p. 273) or Update pipelines for push events (GitHub source) (AWS CloudFormation template) (p. 304).

Important

If you intend to configure webhooks, you must set PollForSourceChanges to false to avoid duplicate pipeline executions.

Valid values for this parameter:

- True: If set, CodePipeline polls your repository for source changes.
- False: If set, CodePipeline does not poll your repository for source changes. Use this setting if you intend to configure a webhook to detect source changes.

Input artifacts

- Number of Artifacts: 0
- Description: Input artifacts do not apply for this action type.

Output artifacts

- Number of Artifacts: 1
- Description: The output artifact of this action is a ZIP file that contains the contents of the configured repository and branch at the commit specified as the source revision for the pipeline execution. The
artifacts generated from the repository are the output artifacts for the GitHub action. The source code commit ID is displayed in CodePipeline as the source revision for the triggered pipeline execution.

Output variables

When configured, this action produces variables that can be referenced by the action configuration of a downstream action in the pipeline. This action produces variables which can be viewed as output variables, even if the action doesn't have a namespace. You configure an action with a namespace to make those variables available to the configuration of downstream actions.

For more information about variables in CodePipeline, see Variables (p. 514).

CommitId

The GitHub commit ID that triggered the pipeline execution. Commit IDs are the full SHA of the commit.

CommitMessage

The description message, if any, associated with the commit that triggered the pipeline execution.

CommitUrl

The URL address for the commit that triggered the pipeline.

RepositoryName

The name of the GitHub repository where the commit that triggered the pipeline was made.

BranchName

The name of the branch for the GitHub repository where the source change was made.

AuthorDate

The date when the commit was authored, in timestamp format.

For more information about the difference between an author and a committer in Git, see Viewing the Commit History in Pro Git by Scott Chacon and Ben Straub.

CommitterDate

The date when the commit was committed, in timestamp format.

For more information about the difference between an author and a committer in Git, see Viewing the Commit History in Pro Git by Scott Chacon and Ben Straub.

Action declaration (GitHub example)

YAML

```yaml
Name: Source
Actions:
  - InputArtifacts: []
    ActionTypeId:
      Version: '1'
      Owner: ThirdParty
      Category: Source
      Provider: GitHub
    OutputArtifacts:
```

API Version 2015-07-09
488
Connecting to GitHub (OAuth)

The first time you use the console to add a GitHub repository to a pipeline, you are asked to authorize CodePipeline access to your repositories. The token requires the following GitHub scopes:

- The `repo` scope, which is used for full control to read and pull artifacts from public and private repositories into a pipeline.
- The `admin:repo_hook` scope, which is used for full control of repository hooks.

When you use the CLI or an AWS CloudFormation template, you must provide the value for a personal access token that you have already created in GitHub.

To view the CodePipeline OAuth applications for your pipeline, see View your authorized OAuth apps (p. 407).

To create and manage GitHub personal access tokens, see Configure your pipeline to use a personal access token (GitHub and CLI) (p. 408).
See also

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

- Resource reference for the AWS CloudFormation User Guide AWS::CodePipeline::Webhook – This includes field definitions, examples, and snippets for the resource in AWS CloudFormation.
- Resource reference for the AWS CloudFormation User Guide AWS::CodeStar::GitHubRepository – This includes field definitions, examples, and snippets for the resource in AWS CloudFormation.
- Tutorial: Create a pipeline that builds and tests your Android app when a commit is pushed to your GitHub repository (p. 73) – This tutorial provides a sample build spec file and sample application to create a pipeline with a GitHub source. It builds and tests an Android app with CodeBuild and AWS Device Farm.

Amazon ECR

Triggers the pipeline when a new image is pushed to the Amazon ECR repository. This action provides an image definitions file referencing the URI for the image that was pushed to Amazon ECR. This source action is often used in conjunction with another source action, such as CodeCommit, to allow a source location for all other source artifacts. For more information, see Tutorial: Create a pipeline with an Amazon ECR source and ECS-to-CodeDeploy deployment (p. 116).

When you use the console to create or edit your pipeline, CodePipeline creates a CloudWatch Events rule that starts your pipeline when a change occurs in the repository.

You must have already created an Amazon ECR repository and pushed an image before you connect the pipeline through an Amazon ECR action.

Topics

- Action type (p. 490)
- Configuration parameters (p. 490)
- Input artifacts (p. 491)
- Output artifacts (p. 491)
- Output variables (p. 491)
- Action declaration (Amazon ECR example) (p. 491)
- See also (p. 492)

Action type

- Category: Source
- Owner: AWS
- Provider: ECR
- Version: 1

Configuration parameters

RepositoryName

Required: Yes

The name of the Amazon ECR repository where the image was pushed.
ImageTag

Required: No
The tag used for the image.

**Note**
If a value for `ImageTag` is not specified, the value defaults to `latest`.

Input artifacts

- **Number of Artifacts:** 0
- **Description:** Input artifacts do not apply for this action type.

Output artifacts

- **Number of Artifacts:** 1
- **Description:** This action produces an artifact that contains an `imageDetail.json` file that contains the URI for the image that triggered the pipeline execution. For information about the `imageDetail.json` file, see [imageDetail.json file for Amazon ECS blue/green deployment actions](p. 510).

Output variables

When configured, this action produces variables that can be referenced by the action configuration of a downstream action in the pipeline. This action produces variables which can be viewed as output variables, even if the action doesn't have a namespace. You configure an action with a namespace to make those variables available to the configuration of downstream actions.

For more information, see [Variables](p. 514).

- **RegistryId**
  The AWS account ID associated with the registry that contains the repository.

- **RepositoryName**
  The name of the Amazon ECR repository where the image was pushed.

- **ImageTag**
  The tag used for the image.

- **ImageDigest**
  The `sha256` digest of the image manifest.

- **ImageURI**
  The URI for the image.

Action declaration (Amazon ECR example)

**YAML**

```
Name: Source
```
See also

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

- Tutorial: Create a pipeline with an Amazon ECR source and ECS-to-CodeDeploy deployment (p. 116)
  - This tutorial provides a sample app spec file and sample CodeDeploy application and deployment group to create a pipeline with a CodeCommit and Amazon ECR source that deploys to Amazon ECS instances.

AWS Lambda

Allows you to execute a Lambda function as an action in your pipeline. Using the event object that is an input to this function, the function has access to the action configuration, input artifact locations, output artifact locations, and other information required to access the artifacts. For an
example event passed to a Lambda invoke function, see Example JSON event (p. 494). As part of the implementation of the Lambda function, there must be a call to either the `PutJobSuccessResult API` or `PutJobFailureResult API`. Otherwise, the execution of this action hangs until the action times out. If you specify output artifacts for the action, they must be uploaded to the S3 bucket as part of the function implementation.

**Action type**

- **Category:** Invoke
- **Owner:** AWS
- **Provider:** Lambda
- **Version:** 1

**Configuration parameters**

**FunctionName**

Required: Yes

`FunctionName` is the name of the function created in Lambda.

**UserParameters**

Required: No

A string that can be processed as input by the Lambda function.

**Input artifacts**

- **Number of Artifacts:** 0 to 5
- **Description:** The set of artifacts to be made available to the Lambda function.

**Output artifacts**

- **Number of Artifacts:** 0 to 5
- **Description:** The set of artifacts produced as output by the Lambda function.

**Output variables**

This action will produce as variables all key-value pairs that are included in the `outputVariables` section of the `PutJobSuccessResult API` request.

For more information about variables in CodePipeline, see Variables (p. 514).

**Example action configuration**

**YAML**

```yaml
Name: Lambda
```
Example JSON event

The Lambda action sends a JSON event that contains the job ID, the pipeline action configuration, input and output artifact locations, and any encryption information for the artifacts. The job worker accesses these details to complete the Lambda action. For more information, see job details. The following is an example event.

```json
{
  "CodePipeline.job": {
    "id": "11111111-abcd-1111-abcd-111111abcdef",
    "accountId": "111111111111",
    "data": {
      "actionConfiguration": {
        "configuration": {
          "FunctionName": "MyLambdaFunction",
          "UserParameters": "input_parameter"
        }
      },
      "inputArtifacts": [],
      "location": {
        "uri": "http://192.0.2.4"
      },
      "outputArtifacts": []
    }
  }
}
```
Example JSON event

```
"s3Location": {
  "bucketName": "bucket_name",
  "objectKey": "filename"
},
"type": "S3"
},
"revision": null,
"name": "ArtifactName"
}

"outputArtifacts": [],
"artifactCredentials": {
  "secretAccessKey": "secret_key",
  "sessionToken": "session_token",
  "accessKeyId": "access_key_ID"
},
"continuationToken": "token_ID",
"encryptionKey": {
  "id": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
  "type": "KMS"
}
}
```

The JSON event provides the following job details for the Lambda action in CodePipeline:

- **id**: The unique system-generated ID of the job.
- **accountId**: The AWS account ID associated with the job.
- **data**: Other information required for a job worker to complete the job.
  - **actionConfiguration**: The action parameters for the Lambda action. For definitions, see [Configuration parameters](p. 493).
- **inputArtifacts**: The artifact supplied to the action.
  - **location**: The artifact store location.
    - **s3Location**: The input artifact location information for the action.
      - **bucketName**: The name of the pipeline artifact store for the action (for example, an Amazon S3 bucket named codepipeline-us-east-2-1234567890).
      - **objectKey**: The name of the application (for example, CodePipelineDemoApplication.zip).
      - **type**: The type of artifact in the location. Currently, S3 is the only valid artifact type.
      - **revision**: The artifact's revision ID. Depending on the type of object, this can be a commit ID (GitHub) or a revision ID (Amazon Simple Storage Service). For more information, see [ArtifactRevision](p. 493).
  - **name**: The name of the artifact to be worked on, such as MyApp.
- **outputArtifacts**: The output of the action.
  - **location**: The artifact store location.
    - **s3Location**: The output artifact location information for the action.
      - **bucketName**: The name of the pipeline artifact store for the action (for example, an Amazon S3 bucket named codepipeline-us-east-2-1234567890).
      - **objectKey**: The name of the application (for example, CodePipelineDemoApplication.zip).
      - **type**: The type of artifact in the location. Currently, S3 is the only valid artifact type.
      - **revision**: The artifact's revision ID. Depending on the type of object, this can be a commit ID (GitHub) or a revision ID (Amazon Simple Storage Service). For more information, see [ArtifactRevision](p. 493).
- name: The name of the output of an artifact, such as MyApp.
- artifactCredentials: The AWS session credentials used to access input and output artifacts in the Amazon S3 bucket. These credentials are temporary credentials that are issued by AWS Security Token Service (AWS STS).
  - secretAccessKey: The secret access key for the session.
  - sessionToken: The token for the session.
  - accessKeyId: The secret access key for the session.
- continuationToken: A token generated by the action. Future actions use this token to identify the running instance of the action. When the action is complete, no continuation token should be supplied.
- encryptionKey: The encryption key used to encrypt the data in the artifact store, such as an AWS KMS key. If this is undefined, the default key for Amazon Simple Storage Service is used.
  - id: The ID used to identify the key. For an AWS KMS key, you can use the key ID, the key ARN, or the alias ARN.
  - type: The type of encryption key, such as an AWS KMS key.

See also

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

- AWS Lambda Developer Guide – For an example pipeline with a Lambda invoke action, see Building a Continuous Delivery Pipeline for a Lambda Application with AWS CodePipeline in the AWS Lambda Developer Guide.
- Invoke an AWS Lambda function in a pipeline in CodePipeline (p. 336) – This procedure provides a sample Lambda function and shows you how to use the console to create a pipeline with a Lambda invoke action.

Amazon S3

Triggers the pipeline when a new object is uploaded to the configured bucket and object key.

Note
When you create your source bucket, make sure you enable versioning on the bucket. If you want to use an existing Amazon S3 bucket, see Using versioning to enable versioning on an existing bucket.

If you use the console to create or edit your pipeline, CodePipeline creates a CloudWatch Events rule that starts your pipeline when a change occurs in the S3 source bucket.

You must have already created an Amazon S3 source bucket and uploaded the source files as a single ZIP file before you connect the pipeline through an Amazon S3 action.

Note
When Amazon S3 is the source provider for your pipeline, you may zip your source file or files into a single .zip and upload the .zip to your source bucket. You may also upload a single unzipped file; however, downstream actions that expect a .zip file will fail.

Topics
- Action type (p. 497)
- Configuration parameters (p. 497)
Action type

- Category: Source
- Owner: AWS
- Provider: S3
- Version: 1

Configuration parameters

S3Bucket

Required: Yes

The name of the Amazon S3 bucket where source changes are to be detected.

S3ObjectKey

Required: Yes

The name of the Amazon S3 object key where source changes are to be detected.

PollForSourceChanges

Required: No

PollForSourceChanges controls whether CodePipeline polls the Amazon S3 source bucket for source changes. We recommend that you use CloudWatch Events and CloudTrail to detect source changes instead. For more information about configuring CloudWatch Events, see Update pipelines for push events (Amazon S3 source) (CLI) (p. 269) or Update pipelines for push events (Amazon S3 source) (AWS CloudFormation template) (p. 285).

Important

If you intend to configure CloudWatch Events, you must set PollForSourceChanges to false to avoid duplicate pipeline executions.

Valid values for this parameter:

- True: If set, CodePipeline polls your source location for source changes.

Note

If you omit PollForSourceChanges, CodePipeline defaults to polling your source location for source changes. This behavior is the same as if PollForSourceChanges is included and set to true.

- False: If set, CodePipeline does not poll your source location for source changes.

Use this setting if you intend to configure a CloudWatch Events rule to detect source changes.

Input artifacts

- Number of Artifacts: 0
- Description: Input artifacts do not apply for this action type.
Output artifacts

- **Number of Artifacts:** 1
- **Description:** Provides the artifacts that are available in the source bucket configured to connect to the pipeline. The artifacts generated from the bucket are the output artifacts for the Amazon S3 action. The Amazon S3 object metadata (ETag and version ID) is displayed in CodePipeline as the source revision for the triggered pipeline execution.

Output variables

When configured, this action produces variables that can be referenced by the action configuration of a downstream action in the pipeline. This action produces variables which can be viewed as output variables, even if the action doesn’t have a namespace. You configure an action with a namespace to make those variables available to the configuration of downstream actions.

For more information about variables in CodePipeline, see Variables (p. 514).

**ETag**

The entity tag for the object related to the source change that triggered the pipeline. The ETag is an MD5 hash of the object. ETag reflects only changes to the contents of an object, not its metadata.

**VersionId**

The version ID for the version of the object related to the source change that triggered the pipeline.

Action declaration (S3 example)

**YAML**

```yaml
Name: Source
Actions:
  - RunOrder: 1
    OutputArtifacts:
      - Name: SourceArtifact
    ActionTypeId:
      Provider: S3
      Owner: AWS
      Version: '1'
      Category: Source
      Region: us-west-2
      Name: Source
    Configuration:
      S3Bucket: my-bucket-oregon
      S3ObjectKey: my-application.zip
      PollForSourceChanges: 'false'
      InputArtifacts: []
```

**JSON**

```json
{
  "Name": "Source",
  "Actions": [
  {
    "RunOrder": 1,
    "OutputArtifacts": []
  }
```
"Name": "SourceArtifact"
}

"ActionTypeId": {
  "Provider": "S3",
  "Owner": "AWS",
  "Version": "1",
  "Category": "Source"
},

"Region": "us-west-2",
"Name": "Source",
"Configuration": {
  "S3Bucket": "my-bucket-oregon",
  "S3ObjectKey": "my-application.zip",
  "PollForSourceChanges": "false"
},
"InputArtifacts": []
],

See also

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

- Tutorial: Create a simple pipeline (S3 bucket) (p. 38) – This tutorial provides a sample app spec file and sample CodeDeploy application and deployment group. Use this tutorial to create a pipeline with an Amazon S3 source that deploys to Amazon EC2 instances.

AWS Step Functions

An AWS CodePipeline action that does the following:

- Starts an AWS Step Functions state machine execution from your pipeline.
- Provides an initial state to the state machine through either a property in the action configuration or a file located in a pipeline artifact to be passed as input.
- Optionally sets an execution ID prefix for identifying executions originating from the action.
- Supports Standard and Express state machines.

Action type

- Category: Invoke
- Owner: AWS
- Provider: StepFunctions
- Version: 1

Configuration parameters

StateMachineArn

Required: Yes
The Amazon Resource Name (ARN) for the state machine to be invoked.

**ExecutionNamePrefix**

Required: No

By default, the action execution ID is used as the state machine execution name. If a prefix is provided, it is prepended to the action execution ID with a hyphen and together used as the state machine execution name.

```
myPrefix-1624a1d1-3699-43f0-8e1e-6baf07fde791
```

For an express state machine, the name should only contain 0-9, A-Z, a-z, -, and _.

**InputType**

Required: No

- **Literal** (default): When specified, the value in the **Input** field is passed directly to the state machine input.

  Example entry for the **Input** field when **Literal** is selected:

  ```json
  {"action": "test"}
  ```

- **FilePath**: The contents of a file in the input artifact specified by the **Input** field is used as the input for the state machine execution. An input artifact is required when **InputType** is set to **FilePath**.

  Example entry for the **Input** field when **FilePath** is selected:

  ```
  assets/input.json
  ```

**Input**

Required: Conditional

- **Literal**: When **InputType** is set to **Literal** (default), this field is optional.

  If provided, the **Input** field is used directly as the input for the state machine execution. Otherwise, the state machine is invoked with an empty JSON object `{}`.

- **FilePath**: When **InputType** is set to **FilePath**, this field is required.

  An input artifact is also required when **InputType** is set to **FilePath**.

  The contents of the file in the input artifact specified are used as the input for the state machine execution.

**Input artifacts**

- **Number of Artifacts**: 0 to 1
  - **Description**: If **InputType** is set to **FilePath**, this artifact is required and is used to source the input for the state machine execution.

**Output artifacts**

- **Number of Artifacts**: 0 to 1
  - **Description**: API Version 2015-07-09
  500
• **Standard State Machines**: If provided, the output artifact is populated with the output of the state machine. This is obtained from the `output` property of the Step Functions DescribeExecution API response after the state machine execution completes successfully.

• **Express State Machines**: Not supported.

## Output variables

This action produces output variables that can be referenced by the action configuration of a downstream action in the pipeline.

For more information, see Variables (p. 514).

### StateMachineArn

The ARN of the state machine.

### ExecutionArn

The ARN of the execution of the state machine. Standard state machines only.

## Example action configuration

### Example for default input

**YAML**

```yaml
Name: ActionName
ActionTypeId:
  Category: Invoke
  Owner: AWS
  Version: 1
  Provider: StepFunctions
OutputArtifacts:
  - Name: myOutputArtifact
Configuration:
  ExecutionNamePrefix: my-prefix
```

**JSON**

```json
{
  "Name": "ActionName",
  "ActionTypeId": {
    "Category": "Invoke",
    "Owner": "AWS",
    "Version": 1,
    "Provider": "StepFunctions"
  },
  "OutputArtifacts": [
    {
      "Name": "myOutputArtifact"
    }
  ],
  "Configuration": {
    "ExecutionNamePrefix": "my-prefix"
  }
}
```
Example for literal input

**YAML**

```
Name: ActionName
ActionTypeId:
  Category: Invoke
  Owner: AWS
  Version: 1
  Provider: StepFunctions
OutputArtifacts:
  - Name: myOutputArtifact
Configuration:
  ExecutionNamePrefix: my-prefix
  Input: '{"action": "test"}'
```

**JSON**

```
{
  "Name": "ActionName",
  "ActionTypeId": {
    "Category": "Invoke",
    "Owner": "AWS",
    "Version": 1,
    "Provider": "StepFunctions"
  },
  "OutputArtifacts": [
    {
      "Name": "myOutputArtifact"
    }
  ],
  "Configuration": {
    "ExecutionNamePrefix": "my-prefix",
    "Input": '{"action": "test"}'
  }
}
```

Example for input file

**YAML**

```
Name: ActionName
InputArtifacts:
  - Name: myInputArtifact
ActionTypeId:
  Category: Invoke
  Owner: AWS
  Version: 1
  Provider: StepFunctions
OutputArtifacts:
  - Name: myOutputArtifact
Configuration:
```
Behavior

During a release, CodePipeline executes the configured state machine using the input as specified in the action configuration.

When `InputType` is set to `Literal`, the content of the `Input` action configuration field is used as the input for the state machine. When literal input is not provided, the state machine execution uses an empty JSON object `{}`. For more information about running a state machine execution without input, see the Step Functions StartExecution API.

When `InputType` is set to `FilePath`, the action unzips the input artifact and uses the content of the file specified in the `Input` action configuration field as the input for the state machine. When `FilePath` is specified, the `Input` field is required and an input artifact must exist; otherwise, the action fails.

After a successful start execution, behavior will diverge for the two state machine types, `standard` and `express`.

**Standard state machines**

If the standard state machine execution was successfully started, CodePipeline polls the DescribeExecution API until the execution reaches a terminal status. If the execution completes successfully, the action succeeds; otherwise, it fails.

If an output artifact is configured, the artifact will contain the return value of the state machine. This is obtained from the `output` property of the Step Functions DescribeExecution API response after the
state machine execution completes successfully. Note that there are output length constraints enforced on this API.

Error handling

- If the action fails to start a state machine execution, the action execution fails.
- If the state machine execution fails to reach a terminal status before the CodePipeline Step Functions action reaches its timeout (default of 7 days), the action execution fails. The state machine might continue despite this failure. For more information about state machine execution timeouts in Step Functions, see Standard vs. Express Workflows.

  Note
  You can request a quota increase for the invoke action timeout for the account with the action. However, the quota increase applies to all actions of this type in all Regions for that account.

- If the state machine execution reaches a terminal status of FAILED, TIMED_OUT, or ABORTED, the action execution fails.

Express state machines

If the express state machine execution was successfully started, the invoke action execution completes successfully.

Considerations for actions configured for express state machines:

- You cannot designate an output artifact.
- The action does not wait for the state machine execution to complete.
- After the action execution is started in CodePipeline, the action execution succeeds even if the state machine execution fails.

Error handling

- If CodePipeline fails to start a state machine execution, the action execution fails. Otherwise, the action succeeds immediately. The action succeeds in CodePipeline regardless of how long the state machine execution takes to complete or its outcome.

See also

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

- AWS Step Functions Developer Guide – For information about state machines, executions, and inputs for state machines, see the AWS Step Functions Developer Guide.
- Tutorial: Use an AWS Step Functions invoke action in a pipeline (p. 156) – This tutorial gets you started with a sample standard state machine and shows you how to use the console to update a pipeline by adding a Step Functions invoke action.

AWS AppConfig

AWS AppConfig is a capability of AWS Systems Manager. AppConfig supports controlled deployments to applications of any size and includes built-in validation checks and monitoring. You can use AppConfig with applications hosted on Amazon EC2 instances, AWS Lambda, containers, mobile applications, or IoT devices.
The `AppConfig` deploy action is an AWS CodePipeline action that deploys configurations stored in your pipeline source location to a specified AppConfig application, environment, and configuration profile. It uses the preferences defined in an AppConfig deployment strategy.

**Action type**

- Category: Deploy
- Owner: AWS
- Provider: AppConfig
- Version: 1

**Configuration parameters**

**Application**

Required: Yes

The ID of the AWS AppConfig application with the details for your configuration and deployment.

**Environment**

Required: Yes

The ID of the AWS AppConfig environment where the configuration is deployed.

**ConfigurationProfile**

Required: Yes

The ID of the AWS AppConfig configuration profile to deploy.

**InputArtifactConfigurationPath**

Required: Yes

The file path of the configuration data within the input artifact to deploy.

**DeploymentStrategy**

Required: No

The AWS AppConfig deployment strategy to use for deployment.

**Input artifacts**

- **Number of Artifacts**: 1
- **Description**: The input artifact for the deploy action.

**Output artifacts**

Not applicable.

**Example action configuration**

YAML

```yaml
name: Deploy
```
actions:
- name: Deploy
  actionTypeId:
    category: Deploy
    owner: AWS
    provider: AppConfig
    version: '1'
  runOrder: 1
  configuration:
    Application: 2s2qv57
    ConfigurationProfile: PvjrpU
    DeploymentStrategy: frqt7ir
    Environment: 9tm27yd
    InputArtifactConfigurationPath: /
  outputArtifacts: []
  inputArtifacts:
    - name: SourceArtifact
      region: us-west-2
      namespace: DeployVariables

See also

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

- AWS AppConfig – For information about AWS AppConfig deployments, see the AWS Systems Manager User Guide.
• **Tutorial: Create a pipeline that uses AWS AppConfig as a deployment provider (p. 159)** – This tutorial gets you started setting up simple deployment configuration files and AppConfig resources, and shows you how to use the console to create a pipeline with an AWS AppConfig deployment action.
Image definitions file reference

This section is a reference only. For information about creating a pipeline with source or deploy actions for containers, see Create a pipeline in CodePipeline (p. 221).

AWS CodePipeline job workers for container actions, such as an Amazon ECR source action or Amazon ECS deploy actions, use definitions files to map the image URI and container name to the task definition. Each definitions file is a JSON-formatted file used by the action provider as follows:

- Amazon ECS standard deployments require an `imagedefinitions.json` file as an input to the deploy action.
- Amazon ECS blue/green deployments require an `imageDetail.json` file as an input to the deploy action.
- Amazon ECR source actions generate an `imageDetail.json` file that is provided as an output from the source action.

Topics

- `imagedefinitions.json` file for Amazon ECS standard deployment actions (p. 508)
- `imageDetail.json` file for Amazon ECS blue/green deployment actions (p. 510)

imagedefinitions.json file for Amazon ECS standard deployment actions

An image definitions document is a JSON file that describes your Amazon ECS container name and the image and tag. If you are deploying container-based applications, you must generate an image definitions file to provide the CodePipeline job worker with the Amazon ECS container and image identification to retrieve from the repository, such as Docker Hub.

**Note**

The default file name for the file is `imagedefinitions.json`. If you choose to use a different file name, you must provide it when you create the pipeline deployment stage.

Create the `imagedefinitions.json` file with the following considerations:

- The file must use UTF-8 encoding.
- The maximum file size limit for the image definitions file is 100 KB.
- You must create the file as a source or build artifact so that it is an input artifact for the deploy action. In other words, make sure that the file is either uploaded to your source location, such as your CodeCommit repository, or generated as a built output artifact.

The `imagedefinitions.json` file provides the container name and image URI. It must be constructed with the following set of key-value pairs.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><code>container_name</code></td>
</tr>
</tbody>
</table>
Here is the JSON structure, where the container name is `sample-app`, the image URI is `ecs-repo`, and the tag is `latest`:

```json
[
  {
    "name": "sample-app",
    "imageUri": "11111EXAMPLE.dkr.ecr.us-west-2.amazonaws.com/ecs-repo:latest"
  }
]
```

You can also construct the file to list multiple container-image pairs.

JSON structure:

```json
[
  {
    "name": "simple-app",
    "imageUri": "httpd:2.4"
  },
  {
    "name": "simple-app-1",
    "imageUri": "mysql"
  },
  {
    "name": "simple-app-2",
    "imageUri": "java1.8"
  }
]
```

Before you create your pipeline, use the following steps to set up the `imagedefinitions.json` file.

1. As part of planning the container-based application deployment for your pipeline, plan the source stage and the build stage, if applicable.
2. Choose one of the following:
   a. If your pipeline has skipped the build stage, you must manually create the JSON file and upload it to your source repository so the source action can provide the artifact. Create the file using a text editor, and name the file or use the default `imagedefinitions.json` file name. Push the image definitions file to your source repository.
      
      **Note**
      
      If your source repository is an Amazon S3 bucket, remember to zip the JSON file.
   b. If your pipeline has a build stage, add a command to your build spec file that outputs the image definitions file in your source repository during the build phase. The following example uses the `printf` command to create an `imagedefinitions.json` file. List this command in the `post_build` section of the `buildspec.yml` file:

   ```
   printf '[["name":"container_name","imageUri":"image_URI"]]' > imagedefinitions.json
   ```

   You must include the image definitions file as an output artifact in the `buildspec.yml` file.
3. When you create your pipeline in the console, on the **Deploy** page of the **Create Pipeline** wizard, in **Image Filename**, enter the image definitions file name.
For a step-by-step tutorial for creating a pipeline that uses Amazon ECS as the deployment provider, see Tutorial: Continuous Deployment with CodePipeline.

imageDetail.json file for Amazon ECS blue/green deployment actions

An imageDetail.json document is a JSON file that describes your Amazon ECS image URI. If you are deploying container-based applications for a blue/green deployment, you must generate the imageDetail.json file to provide the Amazon ECS and CodeDeploy job worker with the image identification to retrieve from the repository, such as Docker Hub.

**Note**
The name of the file must be imageDetail.json.

You must create the imageDetail.json file as a source or build artifact so that it is an input artifact for the deploy action. You can use one of these methods to provide the imageDetail.json file in the pipeline:

- Include the imageDetail.json file in your source location so that it is provided in the pipeline as input to your Amazon ECS blue/green deployment action.
  
  **Note**
  If your source repository is an Amazon S3 bucket, remember to zip the JSON file.

- Amazon ECR source actions automatically generate an imageDetail.json file as an input artifact to the next action.
  
  **Note**
  Because the Amazon ECR source action creates this file, pipelines with an Amazon ECR source action do not need to manually provide an imageDetail.json file.
  For a tutorial about creating a pipeline that includes an Amazon ECR source stage, see Tutorial: Create a pipeline with an Amazon ECR source and ECS-to-CodeDeploy deployment (p. 116).
The `imageDetail.json` file provides the image URI. It must be constructed with the following key-value pair.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageURI</td>
<td><code>ACCOUNTID.dkr.ecr.us-west-2.amazonaws.com/dk-image-repo@sha256:example3</code></td>
</tr>
</tbody>
</table>

Here is the JSON structure, where the image URI is `ACCOUNTID.dkr.ecr.us-west-2.amazonaws.com/dk-image-repo@sha256:example3`:

```json
{
  "ImageURI": "ACCOUNTID.dkr.ecr.us-west-2.amazonaws.com/dk-image-repo@sha256:example3"
}
```
An imageDetail.json file is generated automatically by the Amazon ECR source action each time a change is pushed to the image repository. The imageDetail.json generated by Amazon ECR source actions is provided as an output artifact from the source action to the next action in the pipeline.

Here is the JSON structure, where the repository name is `dk-image-repo`, the image URI is `ecs-repo`, and the image tag is `latest`:

```json
{
    "ImageSizeInBytes": "44728918",
    "ImageDigest": "sha256:EXAMPLE11223344556677889900bfeaa42ea2d3b8a1ee8329ba7e68694950afd3",
    "Version": "1.0",
    "ImagePushedAt": "Mon Jan 21 20:04:00 UTC 2019",
    "RegistryId": "EXAMPLE12233",
    "RepositoryName": "dk-image-repo",
    "ImageURI": "ACCOUNTID.dkr.ecr.us-west-2.amazonaws.com/dk-image-repo@sha256:example3",
    "ImageTags": [ "latest"
    ]
}
```

The imageDetail.json file maps the image URI and container name to the Amazon ECS task definition as follows:

- **ImageSizeInBytes**: The size, in bytes, of the image in the repository.
- **ImageDigest**: The sha256 digest of the image manifest.
- **Version**: The image version.
- **ImagePushedAt**: The date and time when the latest image was pushed to the repository.
- **RegistryId**: The AWS account ID associated with the registry that contains the repository.
- **RepositoryName**: The name of the Amazon ECR repository where the image was pushed.
- **ImageURI**: The URI for the image.
- **ImageTags**: The tag used for the image.

Before you create your pipeline, use the following steps to set up the imageDetail.json file.

1. As part of planning the container-based application blue/green deployment for your pipeline, plan the source stage and the build stage, if applicable.
2. Choose one of the following:
   a. If your pipeline has skipped the build stage, you must manually create the JSON file and upload it to your source repository, such as CodeCommit, so the source action can provide the artifact. Create the file using a text editor, and name the file or use the default `imageDetail.json` file name. Push the imageDetail.json file to your source repository.
   b. If your pipeline has a build stage, perform the following:
      i. Add a command to your build spec file that outputs the image definitions file in your source repository during the build phase. The following example uses the `printf` command to create an imageDetail.json file. List this command in the `post_build` section of the `buildspec.yml` file:

```bash
printf '{"ImageURI":"image_URI"}' > imageDetail.json
```
You must include the `imageDetail.json` file as an output artifact in the `buildspec.yml` file.

ii. Add the `imageDetail.json` as an artifact file in the `buildspec.yml` file.

```yaml
artifacts:
  files:
    - imageDetail.json
```
Variables

This section is a reference only. For information about creating variables, see Working with variables (p. 372).

Variables allow you to configure your pipeline actions with values that are determined at the time of the action execution. Variables can be produced by an action execution or be implicitly available at the start of each pipeline execution.

Some action providers produce a defined set of variables. You choose from default variable keys for that action provider, such as commit ID.

To see step-by-step examples of using variables, see Tutorial: Using variables with Lambda invoke actions (p. 149), Example: Use variables in manual approvals (p. 379), and Tutorial: Create a pipeline that uses variables from AWS CloudFormation deployment actions (p. 105).

Variable Limits

For limit information, see Quotas in AWS CodePipeline (p. 521).

Note
When you enter output variable syntax in the action configuration fields, do not exceed the 1000-character limit for the configuration fields. A validation error is returned when this limit is exceeded.

Topics
- Concepts (p. 514)
- Configuring variables (p. 515)
- Variable resolution (p. 517)
- Rules for variables (p. 517)
- Variables available for pipeline actions (p. 518)

Concepts

This section lists key terms and concepts related to variables and namespaces.

Variables

Variables are key-value pairs that can be used to dynamically configure actions in your pipeline. There are currently two ways these variables are made available:

- There is a set of variables that are implicitly available at the start of each pipeline execution. This set currently includes PipelineExecutionId, the ID of the current pipeline execution.
- There are action types that produce sets of variables when they are executed. You can see the variables produced by an action by inspecting the outputVariables field that is part of the ListActionExecutions API. To see which variables each action type produces, see the CodePipeline Action structure reference (p. 470).

To reference these variables in your action configuration, you must use the variable reference syntax with the correct namespace.
Namespaces

To ensure that variables can be uniquely referenced, they must be assigned to a namespace. After you have a set of variables assigned to a namespace, they can be referenced in an action configuration by using the namespace and variable key with the following syntax:

```
#{namespace.variable_key}
```

There are two types of namespaces under which variables can be assigned:

- **The codepipeline reserved namespace**
  This is the namespace assigned to the set of implicit variables available at the start of each pipeline execution. This namespace is `codepipeline`. Example variable reference:

```
#{codepipeline.PipelineExecutionId}
```

- **Action assigned namespace**
  This is a namespace that you assign to an action. All variables produced by the action fall under this namespace. To make the variables produced by an action available for use in a downstream action configuration, you must configure the producing action with a namespace. Namespaces must be unique across the pipeline definition and cannot conflict with any artifact names. Here is an example variable reference for an action configured with a namespace of `SourceVariables`.

```
#{SourceVariables.VersionId}
```

Configuring variables

You configure an action to produce variables by declaring a namespace for the action. The action must already be one of the action providers that generates variables. Otherwise, the variables available are pipeline-level variables.

You declare the namespace either by:

- On the **Edit action** page of the console, entering a namespace in **Variable namespace**.
- Entering a namespace in the **namespace** parameter field in the JSON pipeline structure.

In this example, you add the **namespace** parameter to the CodeCommit source action with the name `SourceVariables`. This configures the action to produce the variables available for that action provider, such as `CommitId`.

```json
{
    "name": "Source",
    "actions": [
        {
            "outputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "name": "Source",
```
Next, you configure the downstream action to use the variables produced by the previous action. You do this by:

- On the Edit action page of the console, entering the variable syntax (for the downstream action) in the action configuration fields.
- Entering the variable syntax (for the downstream action) in the action configuration fields in the JSON pipeline structure

In this example, the build action's configuration field shows environment variables that are updated upon the action execution. The example specifies the namespace and variable for execution ID with `{codepipeline.PipelineExecutionId}` and the namespace and variable for commit ID with `{SourceVariables.CommitId}`.

```json
{
    "name": "Build",
    "actions": [
        {
            "outputArtifacts": [
                {
                    "name": "BuildArtifact"
                }
            ],
            "name": "Build",
            "configuration": {
                "EnvironmentVariables": 
                    "{name":"Release_ID","value":"{codepipeline.PipelineExecutionId}"},
                    
                    "{name":"Commit_ID","value":"{SourceVariables.CommitId}"}
            },
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-west-2",
            "actionTypeId": {
                "provider": "CodeBuild",
                "category": "Build",
                "version": "1",
                "owner": "AWS"
            },
            "runOrder": 1
        }
    ]
}
```
Variable resolution

Each time an action is executed as part of a pipeline execution, the variables it produces are available for use in any action that is guaranteed to occur after the producing action. To use these variables in a consuming action, you can add them to the consuming action's configuration using the syntax shown in the previous example. Before it performs a consuming action, CodePipeline resolves all of the variable references present in the configuration prior to initiating the action execution.

Rules for variables

The following rules help you with the configuration of variables:

- You specify the namespace and variable for an action through a new action property or by editing an action.
- When you use the pipeline creation wizard, the console generates a namespace for each action created with the wizard.
Variables available for pipeline actions

The action provider determines which variables can be generated by the action.

Unlike a namespace which you can choose, most variable keys cannot be edited. For example, for the Amazon S3 action provider, only the ETag and VersionId variable keys are available.

For CodeBuild, AWS CloudFormation, and Lambda actions, the variable keys are configured by the user.

Each execution also has a set of CodePipeline-generated pipeline variables that contain data about the execution, such as the pipeline release ID. These variables can be consumed by any action in the pipeline.

<table>
<thead>
<tr>
<th>Provider</th>
<th>Variable key</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td>codepipeline</td>
<td>PipelineExecutionId</td>
<td>8abc75f0-fb8-4f4c-bfEXAMPLE</td>
</tr>
<tr>
<td>Amazon ECR</td>
<td>ImageDigest</td>
<td>sha256:EXAMPLE1122334455</td>
</tr>
<tr>
<td></td>
<td>ImageTag</td>
<td>latest</td>
</tr>
<tr>
<td></td>
<td>ImageURI</td>
<td>11111EXAMPLE.dkr.ecr.us-west-2.amazonaws.com/ecs-repo:latest</td>
</tr>
<tr>
<td></td>
<td>RegistryId</td>
<td>EXAMPLE12233</td>
</tr>
<tr>
<td></td>
<td>RepositoryName</td>
<td>my-image-repo</td>
</tr>
<tr>
<td>CodeCommit</td>
<td>AuthorDate</td>
<td>2019-10-29T03:32:21Z</td>
</tr>
<tr>
<td></td>
<td>BranchName</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td>CommitId</td>
<td>exampleb01f91b31</td>
</tr>
<tr>
<td></td>
<td>CommitMessage</td>
<td>Fixed a bug (100 KB maximum size)</td>
</tr>
<tr>
<td></td>
<td>CommitterDate</td>
<td>2019-10-29T03:32:21Z</td>
</tr>
<tr>
<td></td>
<td>RepositoryName</td>
<td>myCodeCommitRepo</td>
</tr>
<tr>
<td>GitHub</td>
<td>AuthorDate</td>
<td>2019-10-29T03:32:21Z</td>
</tr>
<tr>
<td></td>
<td>BranchName</td>
<td>master</td>
</tr>
<tr>
<td></td>
<td>CommitId</td>
<td>exampleb01f91b31</td>
</tr>
<tr>
<td></td>
<td>CommitMessage</td>
<td>Fixed a bug (100 KB maximum size)</td>
</tr>
<tr>
<td></td>
<td>CommitterDate</td>
<td>2019-10-29T03:32:21Z</td>
</tr>
<tr>
<td></td>
<td>CommitUrl</td>
<td></td>
</tr>
</tbody>
</table>
Variables available for pipeline actions

<table>
<thead>
<tr>
<th>Provider</th>
<th>Variable key</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RepositoryName</td>
<td>myGitHubRepo</td>
</tr>
<tr>
<td>S3</td>
<td>ETag</td>
<td>example28be1c3</td>
</tr>
<tr>
<td></td>
<td>VersionId</td>
<td>exampletaIUQCv</td>
</tr>
</tbody>
</table>

For step-by-step procedures for managing variables, see Working with variables (p. 372).
Update polling pipelines to the recommended change detection method

If you have a pipeline that uses polling to react to source changes, you can update it to use the recommended detection method (webhooks for pipelines with a GitHub source and Amazon CloudWatch Events for pipelines with a CodeCommit or Amazon S3 source).

<table>
<thead>
<tr>
<th>How to migrate pipelines to the recommended change detection method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Amazon S3</strong></td>
</tr>
<tr>
<td><strong>AWS CodeCommit</strong></td>
</tr>
</tbody>
</table>
| **GitHub** | Webhooks (recommended).  
  • Allow the console to create your webhook. | See Update pipelines for push events (GitHub source) (console) (p. 265). | See Update pipelines for push events (GitHub source) (CLI) (p. 273). | See Update pipelines for push events (GitHub source) (AWS CloudFormation template) (p. 304). |
Quotas in AWS CodePipeline

CodePipeline has quotas for the number of pipelines, stages, actions, and webhooks that an AWS account can have in each AWS Region. These quotas apply per Region and can be increased. To request an increase, use the Support Center console.

It can take up to two weeks to process requests for a quota increase.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length of time before an action times out</td>
<td>Approval action: 7 days</td>
</tr>
<tr>
<td></td>
<td>AWS CloudFormation deployment action: 3 days</td>
</tr>
<tr>
<td></td>
<td>CodeBuild build action and test action: 8 hours</td>
</tr>
<tr>
<td></td>
<td>CodeDeploy and CodeDeploy ECS (blue/green) deployment actions: 5 days</td>
</tr>
<tr>
<td></td>
<td>AWS Lambda invoke action: 24 hours</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>While the action is running,</td>
</tr>
<tr>
<td></td>
<td>CodePipeline periodically contacts</td>
</tr>
<tr>
<td></td>
<td>Lambda for a status. The Lambda function</td>
</tr>
<tr>
<td></td>
<td>replies with a status, where the action</td>
</tr>
<tr>
<td></td>
<td>execution is either successful, failed, or</td>
</tr>
<tr>
<td></td>
<td>still in progress. After 24 hours, if the</td>
</tr>
<tr>
<td></td>
<td>Lambda function has either sent no reply</td>
</tr>
<tr>
<td></td>
<td>or replied that the action execution is</td>
</tr>
<tr>
<td></td>
<td>still in progress, the CodePipeline</td>
</tr>
<tr>
<td></td>
<td>action times out. If the action times</td>
</tr>
<tr>
<td></td>
<td>out, CodePipeline sets the Lambda</td>
</tr>
<tr>
<td></td>
<td>invoke action state to failed. Lambda has</td>
</tr>
<tr>
<td></td>
<td>a separate timeout for Lambda functions</td>
</tr>
<tr>
<td></td>
<td>that is not related to the CodePipeline</td>
</tr>
<tr>
<td></td>
<td>action timeout.</td>
</tr>
<tr>
<td></td>
<td>Amazon S3 deployment action: 20 minutes</td>
</tr>
<tr>
<td><strong>Note</strong></td>
<td>If the upload to S3 times out during</td>
</tr>
<tr>
<td></td>
<td>deployment of a large ZIP file, the</td>
</tr>
<tr>
<td></td>
<td>action fails with a timeout error. Try</td>
</tr>
<tr>
<td></td>
<td>breaking up the ZIP file into smaller</td>
</tr>
<tr>
<td></td>
<td>files.</td>
</tr>
<tr>
<td></td>
<td>Step Functions invoke action: 7 days</td>
</tr>
<tr>
<td></td>
<td>Custom actions: 24 hours</td>
</tr>
<tr>
<td></td>
<td>All other actions: 1 hour</td>
</tr>
</tbody>
</table>
The Amazon ECS deployment action timeout is configurable up to one hour (the default timeout).

Pipelines configured for either polling or event-based change detection are counted toward this quota.

We recommend that you configure your pipeline to use event-based change detection, such as webhooks or Amazon CloudWatch Events. For more information, see Change detection methods to start pipelines (p. 172).

Based on your source provider, use the following instructions to update your polling pipelines to use event-based change detection:

- To update a CodeCommit source action, see Update pipelines for push events (CodeCommit or Amazon S3 source) (console) (p. 263).
- To update an Amazon S3 source action, see Update pipelines for push events (CodeCommit or Amazon S3 source) (console) (p. 263).
- To update a GitHub source action, see Update pipelines for push events (GitHub source) (console) (p. 265).

The following quotas in AWS CodePipeline apply to Region availability, naming constraints, and allowed artifact sizes. These quotas are fixed and cannot be changed.

For information about structural requirements, see CodePipeline pipeline structure reference (p. 454).
<table>
<thead>
<tr>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Europe (London)</td>
</tr>
<tr>
<td>Europe (Paris)</td>
</tr>
<tr>
<td>Europe (Frankfurt)</td>
</tr>
<tr>
<td>Asia Pacific (Mumbai)</td>
</tr>
<tr>
<td>Asia Pacific (Tokyo)</td>
</tr>
<tr>
<td>Asia Pacific (Seoul)</td>
</tr>
<tr>
<td>Asia Pacific (Singapore)</td>
</tr>
<tr>
<td>Asia Pacific (Sydney)</td>
</tr>
<tr>
<td>South America (São Paulo)</td>
</tr>
<tr>
<td>AWS GovCloud (US-West)</td>
</tr>
</tbody>
</table>

### Characters allowed in an action name
Action names cannot exceed 100 characters. Allowed characters include:
- Lowercase letters a through z, inclusive.
- Uppercase letters A through Z, inclusive.
- Numbers 0 through 9, inclusive.
- Special characters . (period), @ (at sign), - (minus sign), and _ (underscore).
- Any other characters, such as spaces, are not allowed.

### Characters allowed in action types
Action type names cannot exceed 25 characters. Allowed characters include:
- Lowercase letters a through z, inclusive.
- Uppercase letters A through Z, inclusive.
- Numbers 0 through 9, inclusive.
- Special characters . (period), @ (at sign), - (minus sign), and _ (underscore).
- Any other characters, such as spaces, are not allowed.
<table>
<thead>
<tr>
<th>Characters allowed in partner action names</th>
<th>Partner action names must follow the same naming conventions and restrictions as other action names in CodePipeline. Specifically, they cannot exceed 100 characters. Allowed characters include:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Lowercase letters a through z, inclusive.</td>
</tr>
<tr>
<td></td>
<td>Uppercase letters A through Z, inclusive.</td>
</tr>
<tr>
<td></td>
<td>Numbers 0 through 9, inclusive.</td>
</tr>
<tr>
<td></td>
<td>Special characters . (period), @ (at sign), - (minus sign), and _ (underscore).</td>
</tr>
<tr>
<td></td>
<td>Any other characters, such as spaces, are not allowed.</td>
</tr>
<tr>
<td>Characters allowed in a pipeline name</td>
<td>Pipeline names cannot exceed 100 characters. Allowed characters include:</td>
</tr>
<tr>
<td></td>
<td>Lowercase letters a through z, inclusive.</td>
</tr>
<tr>
<td></td>
<td>Uppercase letters A through Z, inclusive.</td>
</tr>
<tr>
<td></td>
<td>Numbers 0 through 9, inclusive.</td>
</tr>
<tr>
<td></td>
<td>Special characters . (period), @ (at sign), - (minus sign), and _ (underscore).</td>
</tr>
<tr>
<td></td>
<td>Any other characters, such as spaces, are not allowed.</td>
</tr>
<tr>
<td>Characters allowed in a stage name</td>
<td>Stage names cannot exceed 100 characters. Allowed characters include:</td>
</tr>
<tr>
<td></td>
<td>Lowercase letters a through z, inclusive.</td>
</tr>
<tr>
<td></td>
<td>Uppercase letters A through Z, inclusive.</td>
</tr>
<tr>
<td></td>
<td>Numbers 0 through 9, inclusive.</td>
</tr>
<tr>
<td></td>
<td>Special characters . (period), @ (at sign), - (minus sign), and _ (underscore).</td>
</tr>
<tr>
<td></td>
<td>Any other characters, such as spaces, are not allowed.</td>
</tr>
<tr>
<td>Maximum length of the action configuration key (for example, the CodeBuild configuration keys are ProjectName, PrimarySource, and EnvironmentVariables)</td>
<td>50 characters</td>
</tr>
<tr>
<td>Maximum length of the action configuration value (for example, the value of the RepositoryName configuration in the CodeCommit action configuration should be less than 1000 characters:</td>
<td>1000 characters</td>
</tr>
</tbody>
</table>

"RepositoryName": "my-repo-name-less-than-1000-characters"
| **Maximum number of actions per pipeline** | 500 |
| **Maximum number of months that pipeline execution history information is retained** | 12 |
| **Maximum number of parallel actions in a stage** | 50 |
| **Maximum number of sequential actions in a stage** | 50 |
| **Maximum size of artifacts in a source stage** | Artifacts stored in Amazon S3 buckets: 5 GB  
Artifacts stored in CodeCommit or GitHub repositories: 1 GB  
Exception: If you are using AWS Elastic Beanstalk to deploy applications, the maximum artifact size is always 512 MB.  
Exception: If you are using AWS CloudFormation to deploy applications, the maximum artifact size is always 256 MB.  
Exception: If you are using the CodeDeployToECS action to deploy applications, the maximum artifact size is always 3 MB. |
<p>| <strong>Maximum size of the image definitions JSON file used in pipelines deploying Amazon ECS containers and images</strong> | 100 KB |
| <strong>Maximum size of input artifacts for AWS CloudFormation actions</strong> | 256 MB |
| <strong>Maximum size of input artifacts for the CodeDeployToECS action</strong> | 3 MB |
| <strong>Maximum size of the JSON object that can be stored in the <code>ParameterOverrides</code> property</strong> | For a CodePipeline deploy action with AWS CloudFormation as the provider, the <code>ParameterOverrides</code> property is used to store a JSON object that specifies values for the AWS CloudFormation template configuration file. There is a maximum size limit of 1 kilobyte for the JSON object that can be stored in the <code>ParameterOverrides</code> property. |
| <strong>Number of actions in a stage</strong> | Minimum of 1, maximum of 50 |
| <strong>Number of artifacts allowed for each action</strong> | For the number of input and output artifacts allowed for each action, see the <a href="#">Number of input and output artifacts for each action type (p. 465)</a> |
| <strong>Number of stages in a pipeline</strong> | Minimum of 2, maximum of 50 |
| <strong>Pipeline tags</strong> | Tags are case sensitive. Maximum of 50 per resource. |</p>
<table>
<thead>
<tr>
<th>Pipeline tag key names</th>
<th>Any combination of Unicode letters, numbers, spaces, and allowed characters in UTF-8 between 1 and 128 characters in length. Allowed characters are + - = . _ : / @</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tag key names must be unique, and each key can have only one value. A tag cannot:</td>
</tr>
<tr>
<td></td>
<td>• begin with aws:</td>
</tr>
<tr>
<td></td>
<td>• consist only of spaces</td>
</tr>
<tr>
<td></td>
<td>• end with a space</td>
</tr>
<tr>
<td></td>
<td>• contain emojis or any of the following characters: ? ^ * \ ~ ! # $ &amp; * ( ) &gt; &lt;</td>
</tr>
<tr>
<td>Pipeline tag values</td>
<td>Any combination of Unicode letters, numbers, spaces, and allowed characters in UTF-8 between 1 and 256 characters in length. Allowed characters are + - = . _ : / @</td>
</tr>
<tr>
<td></td>
<td>A key can have only one value, but many keys can have the same value. A tag cannot:</td>
</tr>
<tr>
<td></td>
<td>• begin with aws:</td>
</tr>
<tr>
<td></td>
<td>• consist only of spaces</td>
</tr>
<tr>
<td></td>
<td>• end with a space</td>
</tr>
<tr>
<td></td>
<td>• contain emojis or any of the following characters: ? ^ * \ ~ ! # $ &amp; * ( ) &gt; &lt;</td>
</tr>
<tr>
<td>Uniqueness of names</td>
<td>Within a single AWS account, each pipeline you create in an AWS Region must have a unique name. You can reuse names for pipelines in different AWS Regions.</td>
</tr>
<tr>
<td></td>
<td>Stage names must be unique within a pipeline.</td>
</tr>
<tr>
<td></td>
<td>Action names must be unique within a stage.</td>
</tr>
</tbody>
</table>
Variable quotas

<table>
<thead>
<tr>
<th>AWS CodePipeline User Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>There is a maximum size limit of 122880 bytes for all output variables combined for a particular action.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AWS CodePipeline User Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>There is a maximum size limit of 100 KB for the total resolved action configuration for a particular action.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AWS CodePipeline User Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output variable names are case sensitive.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AWS CodePipeline User Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespaces are case sensitive.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AWS CodePipeline User Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allowed characters include:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AWS CodePipeline User Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Lowercase letters a through z, inclusive.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AWS CodePipeline User Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Uppercase letters A through Z, inclusive.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AWS CodePipeline User Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Numbers 0 through 9, inclusive.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AWS CodePipeline User Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Special characters ^ (caret), @ (at sign), - (minus sign), _ (underscore), [ (left bracket), ] (right bracket), * (asterisk), $ (dollar sign).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AWS CodePipeline User Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any other characters, such as spaces, are not allowed.</td>
</tr>
</tbody>
</table>
## AWS CodePipeline User Guide document history

The following table describes the important changes in each release of the CodePipeline User Guide. For notification about updates to this documentation, you can subscribe to an RSS feed.

- **API version:** 2015-07-09
- **Latest documentation update:** July 30, 2020

<table>
<thead>
<tr>
<th>update-history-change</th>
<th>update-history-description</th>
<th>update-history-date</th>
</tr>
</thead>
<tbody>
<tr>
<td>The CodeBuild action supports enabling batch builds in AWS CodePipeline (p. 528)</td>
<td>For CodeBuild actions in your pipeline, you can enable batch builds to run multiple builds in a single execution. For more information, see CodeBuild action structure reference and Create a pipeline (console).</td>
<td>July 30, 2020</td>
</tr>
<tr>
<td>AWS CodePipeline now supports AWS AppConfig deployment actions (p. 528)</td>
<td>A new tutorial, Tutorial: Create a pipeline that uses AWS AppConfig as a deployment provider, provides steps to use AWS AppConfig to deploy configuration files with your pipeline. The AWS AppConfig action structure reference topic has also been added.</td>
<td>June 25, 2020</td>
</tr>
<tr>
<td>AWS CodePipeline now supports Amazon VPC in AWS GovCloud (US-West) (p. 528)</td>
<td>You can now connect directly to AWS CodePipeline through a private Amazon VPC endpoint in AWS GovCloud (US-West). For more information, see Use CodePipeline with Amazon Virtual Private Cloud.</td>
<td>June 2, 2020</td>
</tr>
<tr>
<td>AWS CodePipeline now supports AWS Step Functions invoke actions (p. 528)</td>
<td>You can now create a pipeline in CodePipeline that uses AWS Step Functions as the invoke action provider. A new tutorial, Tutorial: Use an AWS Step Functions invoke action in a pipeline, provides steps for starting a state machine execution from your pipeline. The AWS Step Functions Action Structure Reference topic has also been added.</td>
<td>May 28, 2020</td>
</tr>
<tr>
<td>Change Description</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------------------------------------------------------------------</td>
<td>------------</td>
<td></td>
</tr>
<tr>
<td>View, list, and update connections (p. 528)</td>
<td>May 21, 2020</td>
<td></td>
</tr>
<tr>
<td>Connections support tagging connections resources in the CLI (p. 528)</td>
<td>May 6, 2020</td>
<td></td>
</tr>
<tr>
<td>CodePipeline is now available in AWS GovCloud (US-West) (p. 528)</td>
<td>April 8, 2020</td>
<td></td>
</tr>
<tr>
<td>The quotas topic shows which CodePipeline service quotas are configurable (p. 528)</td>
<td>March 12, 2020</td>
<td></td>
</tr>
<tr>
<td>The Amazon ECS deployment action timeout is configurable (p. 528)</td>
<td>February 5, 2020</td>
<td></td>
</tr>
<tr>
<td>New topics describe how you can stop a pipeline execution (p. 528)</td>
<td>January 21, 2020</td>
<td></td>
</tr>
<tr>
<td>CodePipeline supports connections (p. 528)</td>
<td>December 18, 2019</td>
<td></td>
</tr>
<tr>
<td>Updated security, authentication, and access control topics (p. 528)</td>
<td>December 17, 2019</td>
<td></td>
</tr>
<tr>
<td>New topics describe how you can use variables in your pipelines (p. 528)</td>
<td>You can now configure namespaces for actions and generate variables each time the action execution is complete. You can set up downstream actions to reference these namespaces and variables. See Working with variables and Variables.</td>
<td>November 14, 2019</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>New topics describe how pipeline executions work, why stages are locked during an execution, and when pipeline executions are superseded (p. 528)</td>
<td>A number of topics have been added to the Welcome section to describe how pipeline executions work, including why stages are locked during an execution and what happens when pipeline executions are superseded. These topics include a list of concepts, a DevOps workflow example, and recommendations for how a pipeline should be structured. The following topics have been added: Pipeline terms, DevOps pipeline example, and How pipeline executions work.</td>
<td>November 11, 2019</td>
</tr>
<tr>
<td>CodePipeline supports notification rules (p. 528)</td>
<td>You can now use notification rules to notify users of important changes in pipelines. For more information, see Create a notification rule.</td>
<td>November 5, 2019</td>
</tr>
<tr>
<td>CodeBuild environment variables available in CodePipeline (p. 528)</td>
<td>You can set CodeBuild environment variables in the CodeBuild build action for your pipeline. You can use the console or CLI to add the EnvironmentVariables parameter to the pipeline structure. The Create a pipeline (console) topic has been updated. The action configuration examples in the action reference for CodeBuild have also been updated.</td>
<td>October 14, 2019</td>
</tr>
<tr>
<td>New Region (p. 528)</td>
<td>CodePipeline is now available in Europe (Stockholm). The Limits topic and AWS service endpoints topic have been updated.</td>
<td>September 5, 2019</td>
</tr>
<tr>
<td>Feature Description</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>Specify canned ACLs and cache control for Amazon S3 deployment actions (p. 528)</td>
<td>You can now specify canned ACL and cache control options when you create an Amazon S3 deployment action in CodePipeline. The following topics have been updated: Create a pipeline (console), CodePipeline Pipeline structure reference, and Tutorial: Create a pipeline that uses Amazon S3 as a deployment provider.</td>
<td>June 27, 2019</td>
</tr>
<tr>
<td>You can now add tags to resources in AWS CodePipeline (p. 528)</td>
<td>You can now use tagging to track and manage AWS CodePipeline resources such as pipelines, custom actions, and webhooks. The following new topics have been added: Tagging resources, Using tags to control access to CodePipeline resources, Tag a pipeline in CodePipeline, Tag a custom action in CodePipeline, and Tag a webhook in CodePipeline. The following topics have been updated to show how to use the CLI to tag resources: Create a pipeline (CLI), Create a custom action (CLI), and Create a webhook for a GitHub source.</td>
<td>May 15, 2019</td>
</tr>
<tr>
<td>You can now view action execution history in AWS CodePipeline (p. 528)</td>
<td>You can now view details about past executions of all actions in a pipeline. These details include start and end times, duration, action execution ID, status, input and output artifact location details, and external resource details. The View pipeline details and history topic has been updated to reflect this support.</td>
<td>March 20, 2019</td>
</tr>
<tr>
<td>AWS CodePipeline now supports publishing applications to the AWS Serverless Application Repository (p. 528)</td>
<td>You can now create a pipeline in CodePipeline that publishes your serverless application to the AWS Serverless Application Repository. A new tutorial, Tutorial: Publish applications to the AWS Serverless Application Repository, provides steps for creating and configuring a pipeline to continuously deliver your serverless application to the AWS Serverless Application Repository.</td>
<td>March 8, 2019</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>AWS CodePipeline now supports cross-region actions in the console (p. 528)</td>
<td>You can now manage cross-region actions in the AWS CodePipeline console. Add a cross-Region action has been updated with the steps to add, edit, or delete an action that is in a different AWS Region from your pipeline. The Create a pipeline, Edit a pipeline, and CodePipeline pipeline structure reference topics have been updated.</td>
<td>February 14, 2019</td>
</tr>
<tr>
<td>AWS CodePipeline now supports Amazon S3 deployments (p. 528)</td>
<td>You can now create a pipeline in CodePipeline that uses Amazon S3 as the deployment action provider. A new tutorial, Tutorial: Create a pipeline that uses Amazon S3 as a deployment provider, provides steps for deploying sample files to your Amazon S3 bucket with CodePipeline. The CodePipeline pipeline structure reference topic has also been updated.</td>
<td>January 16, 2019</td>
</tr>
<tr>
<td>AWS CodePipeline now supports Alexa Skills Kit deployments (p. 528)</td>
<td>You can now use CodePipeline and Alexa Skills Kit for continuous deployment of Alexa skills. A new tutorial, Tutorial: Create a pipeline that deploys an Amazon Alexa skill, contains steps for creating credentials that allow AWS CodePipeline to connect to your Alexa Skills Kit developer account and then creating a pipeline that deploys a sample skill. The CodePipeline pipeline structure reference topic has been updated.</td>
<td>December 19, 2018</td>
</tr>
<tr>
<td>AWS CodePipeline now supports Amazon VPC endpoints powered by AWS PrivateLink (p. 528)</td>
<td>You can now connect directly to AWS CodePipeline through a private endpoint in your VPC, keeping all traffic inside your VPC and the AWS network. For more information, see Use CodePipeline with Amazon Virtual Private Cloud.</td>
<td>December 6, 2018</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>AWS CodePipeline now supports Amazon ECR source actions and</td>
<td>You can now use CodePipeline and CodeDeploy with Amazon ECR and Amazon ECS for continuous deployment of container-based applications. A new tutorial, Create a pipeline with an Amazon ECR source and ECS-to-CodeDeploy deployment, contains steps for using the console to create a pipeline that deploys container applications stored in an image repository to an Amazon ECS cluster with CodeDeploy traffic routing. The Create a pipeline and CodePipeline pipeline structure reference topics have been updated.</td>
<td>November 27, 2018</td>
</tr>
<tr>
<td>ECS-to-CodeDeploy deployment actions (p. 528)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AWS CodePipeline now supports cross-region actions in a pipeline (p. 528)</td>
<td>A new topic, Add a Cross-region Action, contains steps for using the AWS CLI or AWS CloudFormation to add an action that is in a different region from your pipeline. The Create a pipeline, Edit a pipeline, and CodePipeline pipeline structure reference topics have been updated.</td>
<td>November 12, 2018</td>
</tr>
<tr>
<td>AWS CodePipeline now integrates with AWS Service Catalog (p. 528)</td>
<td>You can now add AWS Service Catalog as a deployment action to your pipeline. This allows you to set up a pipeline to publish product updates to AWS Service Catalog when you make a change in your source repository. The Integrations topic has been updated to reflect this support for AWS Service Catalog. Two AWS Service Catalog tutorials have been added to the AWS CodePipeline tutorials section.</td>
<td>October 16, 2018</td>
</tr>
<tr>
<td>AWS CodePipeline now integrates with AWS Device Farm (p. 528)</td>
<td>You can now add AWS Device Farm as a test action to your pipeline. This allows you to set up a pipeline to test mobile applications. The Integrations topic has been updated to reflect this support for AWS Device Farm. Two AWS Device Farm tutorials have been added to the AWS CodePipeline tutorials section.</td>
<td>July 19, 2018</td>
</tr>
</tbody>
</table>
Earlier updates

The following table describes important changes in each release of the CodePipeline User Guide on June 30, 2018 and earlier.

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use webhooks to detect source changes in GitHub pipelines</td>
<td>When you create or edit a pipeline in the console, CodePipeline now creates a webhook that detects changes to your GitHub source repository and then starts your pipeline. For information about migrating your pipeline, see Configure Your GitHub Pipelines to Use Webhooks for Change Detection. For more information, see Start a Pipeline Execution in CodePipeline.</td>
<td>May 1, 2018</td>
</tr>
<tr>
<td>Updated topics</td>
<td>When you create or edit a pipeline in the console, CodePipeline now creates an Amazon CloudWatch Events rule and an AWS CloudTrail trail that detects changes to your Amazon S3 source bucket and then starts your pipeline. For information about migrating your pipeline, see Change detection methods to start pipelines (p. 172). The Tutorial: Create a simple pipeline (S3 bucket) (p. 38) has been updated to show how the Amazon CloudWatch Events rule and trail are created when you select an Amazon S3 source. Create a pipeline in CodePipeline (p. 221) and Edit a pipeline in CodePipeline (p. 231) have also been updated. For more information, see Start a pipeline execution in CodePipeline (p. 171).</td>
<td>March 22, 2018</td>
</tr>
<tr>
<td>Updated topic</td>
<td>CodePipeline is now available in Europe (Paris). The Quotas in AWS CodePipeline (p. 521) topic has been updated.</td>
<td>February 21, 2018</td>
</tr>
<tr>
<td>Updated topics</td>
<td>You can now use CodePipeline and Amazon ECS for continuous deployment of container-based applications. When you create a pipeline, you can select Amazon ECS as a deployment provider. A change to code in your source control repository triggers your pipeline to build a new Docker</td>
<td>December 12, 2017</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date changed</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td>New and updated topics</td>
<td>CodePipeline now provides built-in support for pipeline state change notifications through Amazon CloudWatch Events and Amazon Simple Notification Service (Amazon SNS). A new tutorial Tutorial: Set up a CloudWatch Events rule to receive email notifications for pipeline state changes (p. 70) has been added. For more information, see Detect and react to changes in pipeline state with Amazon CloudWatch Events (p. 384).</td>
<td>September 8, 2017</td>
</tr>
<tr>
<td>New and updated topics</td>
<td>You can now add CodePipeline as a target for Amazon CloudWatch Events actions. Amazon CloudWatch Events rules can be set up to detect source changes so that the pipeline starts as soon as those changes occur, or they can be set up to run scheduled pipeline executions. Information has been added for the PollForSourceChanges source action configuration option. For more information, see Start a pipeline execution in CodePipeline (p. 171).</td>
<td>September 5, 2017</td>
</tr>
<tr>
<td>New Regions</td>
<td>CodePipeline is now available in Asia Pacific (Seoul) and Asia Pacific (Mumbai). The Quotas in AWS CodePipeline (p. 521) topic and Regions and Endpoints topic have been updated.</td>
<td>July 27, 2017</td>
</tr>
<tr>
<td>New Regions</td>
<td>CodePipeline is now available in US West (N. California), Canada (Central), and Europe (London). The Quotas in AWS CodePipeline (p. 521) topic and Regions and Endpoints topic have been updated.</td>
<td>June 29, 2017</td>
</tr>
</tbody>
</table>

When you create or edit a pipeline in the console, CodePipeline now creates an Amazon CloudWatch Events rule that detects changes to your CodeCommit repository and then automatically starts your pipeline. For information about migrating your existing pipeline, see Change detection methods to start pipelines (p. 172).

The Tutorial: Create a simple pipeline (CodeCommit repository) (p. 52) has been updated to show how the Amazon CloudWatch Events rule and role are created when you select a CodeCommit repository and branch. Create a pipeline in CodePipeline (p. 221) and Edit a pipeline in CodePipeline (p. 231) have also been updated.

For more information, see Start a pipeline execution in CodePipeline (p. 171).

The topics Product and service integrations with CodePipeline (p. 22), Create a pipeline in CodePipeline (p. 221), and CodePipeline pipeline structure reference (p. 454) have been updated to reflect this support for Amazon ECS.

When you create or edit a pipeline in the console, image, push it to your container registry, and then deploy the updated image to an Amazon ECS service.

For more information, see Start a pipeline execution in CodePipeline (p. 171).
## Earlier updates

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updated topics</td>
<td>You can now view details about past executions of a pipeline, not just the most recent execution. These details include start and end times, duration, and execution ID. Details are available for a maximum of 100 pipeline executions during the most recent 12-month period. The topics View pipeline details and history in CodePipeline (p. 237), CodePipeline permissions reference (p. 439), and Quotas in AWS CodePipeline (p. 521) have been updated to reflect this support.</td>
<td>June 22, 2017</td>
</tr>
<tr>
<td>Updated topic</td>
<td>Nouvola has been added to the list of available actions in Test action integrations (p. 26).</td>
<td>May 18, 2017</td>
</tr>
<tr>
<td>Updated topics</td>
<td>In the AWS CodePipeline wizard, the page Step 4: Beta has been renamed Step 4: Deploy. The default name of the stage created by this step has been changed from “Beta” to “Staging”. Numerous topics and screenshots have been updated to reflect these changes.</td>
<td>April 7, 2017</td>
</tr>
<tr>
<td>Updated topics</td>
<td>You can now add AWS CodeBuild as a test action to any stage of a pipeline. This allows you to more easily use AWS CodeBuild to run unit tests against your code. Prior to this release, you could use AWS CodeBuild to run unit tests only as part of a build action. A build action requires a build output artifact, which unit tests typically do not produce. The topics Product and service integrations with CodePipeline (p. 22), Edit a pipeline in CodePipeline (p. 231), and CodePipeline pipeline structure reference (p. 454) have been updated to reflect this support for AWS CodeBuild.</td>
<td>March 8, 2017</td>
</tr>
<tr>
<td>New and updated topics</td>
<td>The table of contents has been reorganized to include sections for pipelines, actions, and stage transitions. A new section has been added for CodePipeline tutorials. For better usability, Product and service integrations with CodePipeline (p. 22) has been divided into shorter topics. A new section, Authorization and Access Control, provides comprehensive information about using AWS Identity and Access Management (IAM) and CodePipeline to help secure access to your resources through the use of credentials. These credentials provide the permissions required to access AWS resources, such as putting and retrieving artifacts from Amazon S3 buckets and integrating AWS OpsWorks stacks into your pipelines.</td>
<td>February 8, 2017</td>
</tr>
<tr>
<td>New Region</td>
<td>CodePipeline is now available in Asia Pacific (Tokyo). The Quotas in AWS CodePipeline (p. 521) topic and Regions and Endpoints topic have been updated.</td>
<td>December 14, 2016</td>
</tr>
<tr>
<td>New Region</td>
<td>CodePipeline is now available in South America (São Paulo). The Quotas in AWS CodePipeline (p. 521) topic and Regions and Endpoints topic have been updated.</td>
<td>December 7, 2016</td>
</tr>
</tbody>
</table>
### Updated topics
You can now add AWS CodeBuild as a build action to any stage of a pipeline. AWS CodeBuild is a fully managed build service in the cloud that compiles your source code, runs unit tests, and produces artifacts that are ready to deploy. You can use an existing build project or create one in the CodePipeline console. The output of the build project can then be deployed as part of a pipeline.

The topics Product and service integrations with CodePipeline (p. 22), Create a pipeline in CodePipeline (p. 221), Authentication and Access Control, and CodePipeline pipeline structure reference (p. 454) have been updated to reflect this support for AWS CodeBuild.

You can now use CodePipeline with AWS CloudFormation and the AWS Serverless Application Model to continuously deliver your serverless applications. The topic Product and service integrations with CodePipeline (p. 22) has been updated to reflect this support.

Product and service integrations with CodePipeline (p. 22) has been reorganized to group AWS and partner offerings by action type.

#### New Region
CodePipeline is now available in Europe (Frankfurt). The Quotas in AWS CodePipeline (p. 521) topic and Regions and Endpoints topic have been updated.

#### Updated topics
AWS CloudFormation can now be selected as a deployment provider in pipelines, enabling you to take action on AWS CloudFormation stacks and change sets as part of a pipeline execution. The topics Product and service integrations with CodePipeline (p. 22), Create a pipeline in CodePipeline (p. 221), Authentication and Access Control, and CodePipeline pipeline structure reference (p. 454) have been updated to reflect this support for AWS CloudFormation.

#### New Region
CodePipeline is now available in the Asia Pacific (Sydney) Region. The Quotas in AWS CodePipeline (p. 521) topic and Regions and Endpoints topic have been updated.

#### New Region
CodePipeline is now available in Asia Pacific (Singapore). The Quotas in AWS CodePipeline (p. 521) topic and Regions and Endpoints topic have been updated.

#### New Region
CodePipeline is now available in the US East (Ohio) Region. The Quotas in AWS CodePipeline (p. 521) topic and Regions and Endpoints topic have been updated.

#### Updated topic
Create a pipeline in CodePipeline (p. 221) has been updated to reflect support for displaying version identifiers of custom actions in the **Source provider** and **Build provider** lists.
<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date changed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updated topic</td>
<td>The Manage approval actions in CodePipeline (p. 355) section has been updated to reflect an enhancement that lets Approval action reviewers open the Approve or reject the revision form directly from an email notification.</td>
<td>September 14, 2016</td>
</tr>
<tr>
<td>New and updated topics</td>
<td>A new topic, View pipeline execution source revisions (console) (p. 241), describes how to view details about code changes currently flowing through your software release pipeline. Quick access to this information can be useful when reviewing manual approval actions or troubleshooting failures in your pipeline. A new section, Monitoring pipelines with CodePipeline (p. 384), provides a central location for all topics related to monitoring the status and progress of your pipelines.</td>
<td>September 08, 2016</td>
</tr>
<tr>
<td>New and updated topics</td>
<td>A new section, Manage approval actions in CodePipeline (p. 355), provides information about configuring and using manual approval actions in pipelines. Topics in this section provide conceptual information about the approval process; instructions for setting up required IAM permissions, creating approval actions, and approving or rejecting approval actions; and samples of the JSON data generated when an approval action is reached in a pipeline.</td>
<td>July 06, 2016</td>
</tr>
<tr>
<td>New Region</td>
<td>CodePipeline is now available in the Europe (Ireland) Region. The Quotas in AWS CodePipeline (p. 521) topic and Regions and endpoints topic have been updated.</td>
<td>June 23, 2016</td>
</tr>
<tr>
<td>New topic</td>
<td>A new topic, Retry a failed action in CodePipeline (p. 352), has been added to describe how to retry a failed action or a group of parallel failed actions in stage.</td>
<td>June 22, 2016</td>
</tr>
<tr>
<td>Updated topics</td>
<td>A number of topics, including Create a pipeline in CodePipeline (p. 221), Authentication and Access Control, CodePipeline pipeline structure reference (p. 454), and Product and service integrations with CodePipeline (p. 22), have been updated to reflect support for configuring a pipeline to deploy code in conjunction with custom Chef cookbooks and applications created in AWS OpsWorks. CodePipeline support for AWS OpsWorks is currently available in the US East (N. Virginia) Region (us-east-1) only.</td>
<td>June 2, 2016</td>
</tr>
<tr>
<td>New and updated topics</td>
<td>A new topic, Tutorial: Create a simple pipeline (CodeCommit repository) (p. 52), has been added. This topic provides a sample walkthrough showing how to use a CodeCommit repository and branch as the source location for a source action in a pipeline. Several other topics have been updated to reflect this integration with CodeCommit, including Authentication and Access Control, Product and service integrations with CodePipeline (p. 22), Tutorial: Create a four-stage pipeline (p. 63), and Troubleshooting CodePipeline (p. 397).</td>
<td>April 18, 2016</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date changed</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>New topic</td>
<td>A new topic, <em>Invoke an AWS Lambda function in a pipeline in CodePipeline</em> (p. 336), has been added. This topic contains sample AWS Lambda functions and steps for adding Lambda functions to pipelines.</td>
<td>January 27, 2016</td>
</tr>
<tr>
<td>Updated topic</td>
<td>A new section has been added to Authentication and Access Control, Resource-based Policies.</td>
<td>January 22, 2016</td>
</tr>
<tr>
<td>New topic</td>
<td>A new topic, <em>Product and service integrations with CodePipeline</em> (p. 22), has been added. Information about integrations with partners and other AWS services has been moved to this topic. Links to blogs and videos have also been added.</td>
<td>December 17, 2015</td>
</tr>
<tr>
<td>Updated topic</td>
<td>Details of integration with Solano CI have been added to <em>Product and service integrations with CodePipeline</em> (p. 22).</td>
<td>November 17, 2015</td>
</tr>
<tr>
<td>Updated topic</td>
<td>The CodePipeline Plugin for Jenkins is now available through the Jenkins Plugin Manager as part of the library of plugins for Jenkins. The steps for installing the plugin have been updated in <em>Tutorial: Create a four-stage pipeline</em> (p. 63).</td>
<td>November 9, 2015</td>
</tr>
<tr>
<td>New Region</td>
<td>CodePipeline is now available in the US West (Oregon) Region. The <em>Quotas in AWS CodePipeline</em> (p. 521) topic has been updated. Links have been added to <em>Regions and Endpoints</em>.</td>
<td>October 22, 2015</td>
</tr>
<tr>
<td>New topic</td>
<td>Two new topics, <em>Configure server-side encryption for artifacts stored in Amazon S3 for CodePipeline</em> (p. 405) and <em>Create a pipeline in CodePipeline that uses resources from another AWS account</em> (p. 251), have been added. A new section has been added to Authentication and Access Control, <em>Example 8: Use AWS resources associated with another account in a pipeline</em> (p. 433).</td>
<td>August 25, 2015</td>
</tr>
<tr>
<td>Updated topic</td>
<td>The <em>Create and add a custom action in CodePipeline</em> (p. 324) topic has been updated to reflect changes in the structure, including inputArtifactDetails and outputArtifactDetails.</td>
<td>August 17, 2015</td>
</tr>
<tr>
<td>Updated topic</td>
<td>The <em>Troubleshooting CodePipeline</em> (p. 397) topic has been updated with revised steps for troubleshooting problems with the service role and Elastic Beanstalk.</td>
<td>August 11, 2015</td>
</tr>
<tr>
<td>Updated topic</td>
<td>The <em>Authentication and Access Control</em> topic has been updated with the latest changes to the <em>service role for CodePipeline</em> (p. 445).</td>
<td>August 6, 2015</td>
</tr>
<tr>
<td>New topic</td>
<td>A <em>Troubleshooting CodePipeline</em> (p. 397) topic has been added. Updated steps have been added for IAM roles and Jenkins in <em>Tutorial: Create a four-stage pipeline</em> (p. 63).</td>
<td>July 24, 2015</td>
</tr>
<tr>
<td>Topic update</td>
<td>Updated steps have been added for downloading the sample files in <em>Tutorial: Create a simple pipeline (S3 bucket)</em> (p. 38) and <em>Tutorial: Create a four-stage pipeline</em> (p. 63).</td>
<td>July 22, 2015</td>
</tr>
<tr>
<td>Change</td>
<td>Description</td>
<td>Date changed</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Topic update</td>
<td>A temporary workaround for download issues with the sample files was added in Tutorial: Create a simple pipeline (S3 bucket) (p. 38).</td>
<td>July 17, 2015</td>
</tr>
<tr>
<td>Topic update</td>
<td>A link was added in Quotas in AWS CodePipeline (p. 521) to point to information about which limits can be changed.</td>
<td>July 15, 2015</td>
</tr>
<tr>
<td>Topic update</td>
<td>The managed policies section in Authentication and Access Control was updated.</td>
<td>July 10, 2015</td>
</tr>
<tr>
<td>Initial Public Release</td>
<td>This is the initial public release of the CodePipeline User Guide.</td>
<td>July 9, 2015</td>
</tr>
</tbody>
</table>
AWS glossary

For the latest AWS terminology, see the AWS glossary in the AWS General Reference.