aws

User Guide

AWS CodePipeline

API Version 2015-07-09

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

AWS CodePipeline: User Guide

Copyright © 2026 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.

Services or capabilities described in AWS documentation might vary by Region. To see the differences applicable to the AWS European Sovereign Cloud Region, see the AWS European Sovereign Cloud User Guide.

Table of Contents

What is CodePipeline? 1
Continuous delivery and continuous integration 1
What can I do with CodePipeline? 2
A quick look at CodePipeline 2
How do I get started with CodePipeline? 3
Concepts 4
Pipelines 4
Pipeline executions 6
Stage operations 7
Action executions 8
Execution types 8
Action types 8
Artifacts 8
Source revisions 9
Triggers 9
Variables 10
Conditions 10
Rules 11
DevOps pipeline example 11
How pipeline executions work 13
How pipeline executions are started 14
How source revisions are processed in pipeline executions 14
How source overrides work with the EventBridge input transformer 15
How pipeline executions are stopped 16
How executions are processed in SUPERSEDED mode 20
How executions are processed in QUEUED mode 21
How executions are processed in PARALLEL mode 23
Managing Pipeline Flow 23
Input and output artifacts 26
How do stage conditions work? 29
Rules for stage conditions 32
Pipeline types 33
What type of pipeline is right for me? 33
Getting started 39

Sign up for an AWS account ... 39
Apply a managed policy for administrative access to CodePipeline ... 40
Install the AWS CLI ... 41
Open the console for CodePipeline ... 42
Next steps ... 42
Product and service integrations ... 43
Integrations with CodePipeline action types ... 43
Source action integrations ... 43
Build action integrations ... 50
Test action integrations ... 52
Deploy action integrations ... 54
Approval action integration with Amazon Simple Notification Service ... 60
Invoke action integrations ... 61
General integrations with CodePipeline ... 62
Examples from the community ... 65
Blog posts ... 65
Tutorials ... 69
Tutorial: Deploy to Amazon EC2 instances with CodePipeline ... 70
Prerequisites ... 70
Step 1: Create Amazon EC2 Linux instances ... 71
Step 2: Add artifact bucket permissions to the EC2 instance role ... 73
Step 3: Add a script file to your repository ... 74
Step 4: Creating your pipeline ... 74
Step 5: Test Your Pipeline ... 79
Tutorial: Build and push a Docker image to Amazon ECR with CodePipeline (V2 type) ... 80
Prerequisites ... 81
Step 1: Add a Dockerfile to your source repository ... 81
Step 2: Add an imagedefinitions.json file to your source repository ... 83
Step 3: Creating your pipeline ... 84
Step 4: Test Your Pipeline ... 89
Tutorial: Deploy to Amazon EKS with CodePipeline ... 90
Prerequisites ... 90
Step 1: (Optional) Create a cluster in Amazon EKS ... 91
Step 2: Configure your private cluster in Amazon EKS ... 93
Step 3: Update the CodePipeline service role policy in IAM ... 94
Step 4: Create an access entry for the CodePipeline service role ... 97

Step 5: Create a source repository and add the `Helm chart` config files ... 97
Step 6: Creating your pipeline ... 98
Tutorial: Create a pipeline that runs commands with compute (V2 type) ... 100
Prerequisites ... 101
Step 1: Create source files and push to your GitHub repository ... 101
Step 2: Create your pipeline ... 102
Step 3: Run your pipeline and verify build commands ... 105
Tutorial: Use Git tags to start your pipeline ... 106
Prerequisites ... 107
Step 1: Open CloudShell and clone your repository ... 107
Step 2: Create a pipeline to trigger on Git tags ... 108
Step 3: Tag your commits for release ... 112
Step 4: Release change and view logs ... 113
Tutorial: Filter on branch names for pull requests to start your pipeline (V2 type) ... 113
Prerequisites ... 114
Step 1: Create a pipeline to start on pull request for specified branches ... 114
Step 2: Create and merge a pull request in GitHub.com to start your pipeline executions ... 117
Tutorial: Use pipeline-level variables ... 118
Prerequisites ... 119
Step 1: Create your pipeline and build project ... 119
Step 2: Release change and view logs ... 122
Tutorial: Create a simple pipeline (S3 bucket) ... 123
Create an S3 bucket ... 124
Create Windows Server Amazon EC2 instances and install the CodeDeploy agent ... 126
Create an application in CodeDeploy ... 128
Create your first pipeline ... 130
Add another stage ... 132
Disable and enable transitions between stages ... 139
Clean up resources ... 140
Tutorial: Create a simple pipeline (CodeCommit repository) ... 141
Create a CodeCommit repository ... 142
Download, commit, and push your code ... 143
Create an Amazon EC2 Linux instance and install the CodeDeploy agent ... 146
Create an application in CodeDeploy ... 147
Create your first pipeline ... 149
Update code in your CodeCommit repository ... 152

Clean up resources ... 154
Further reading ... 154
Tutorial: Create a four-stage pipeline ... 155
Complete prerequisites ... 156
Create a pipeline ... 160
Add more stages ... 162
Clean up resources ... 166
Tutorial: Set up a CloudWatch Events rule to receive email notifications for pipeline state changes ... 166
Set up an email notification using Amazon SNS ... 167
Create a CloudWatch Events notification rule for CodePipeline ... 168
Clean up resources ... 170
Tutorial: Build and test an Android app with AWS Device Farm ... 170
Configure CodePipeline to use your Device Farm tests ... 171
Tutorial: Test an iOS app with AWS Device Farm ... 176
Configure CodePipeline to use your Device Farm tests (Amazon S3 example) ... 177
Tutorial: Create a pipeline that deploys to Service Catalog ... 182
Option 1: Deploy to Service Catalog without a configuration file ... 183
Option 2: Deploy to Service Catalog using a configuration file ... 188
Tutorial: Create a pipeline with AWS CloudFormation ... 193
Example 1: Create an AWS CodeCommit pipeline with AWS CloudFormation ... 193
Example 2: Create an Amazon S3 pipeline with AWS CloudFormation ... 195
Tutorial: Create a pipeline that uses variables from AWS CloudFormation deployment actions ... 199
Prerequisites: Create an AWS CloudFormation service role and a CodeCommit repository ... 200
Step 1: Download, edit, and upload the sample AWS CloudFormation template ... 200
Step 2: Create your pipeline ... 202
Step 3: Add a CloudFormation deployment action to create the change set ... 204
Step 4: Add a manual approval action ... 205
Step 5: Add a CloudFormation deployment action to execute the change set ... 206
Step 6: Add a CloudFormation deployment action to delete the stack ... 206
Tutorial: Amazon ECS Standard Deployment with CodePipeline ... 207
Prerequisites ... 208
Step 1: Add a Build Specification File to Your Source Repository ... 211
Step 2: Creating Your Continuous Deployment Pipeline ... 213

Step 3: Add Amazon ECR Permissions to the CodeBuild Role 215
Step 4: Test Your Pipeline 215
Tutorial: Create a pipeline with an Amazon ECR source and ECS-to-CodeDeploy deployment 216
Prerequisites 218
Step 1: Create image and push to an Amazon ECR repository 218
Step 2: Create task definition and AppSpec source files and push to a CodeCommit repository 219
Step 3: Create your Application Load Balancer and target groups 223
Step 4: Create your Amazon ECS cluster and service 226
Step 5: Create your CodeDeploy application and deployment group (ECS compute platform) 229
Step 6: Create your pipeline 230
Step 7: Make a change to your pipeline and verify deployment 234
Tutorial: Create a pipeline that deploys an Amazon Alexa skill 234
Prerequisites 235
Step 1: Create an Alexa developer services LWA security profile 235
Step 2: Create Alexa skill source files and push to your CodeCommit repository 235
Step 3: Use ASK CLI commands to create a refresh token 237
Step 4: Create your pipeline 238
Step 5: Make a change to any source file and verify deployment 240
Tutorial: Create a pipeline that uses Amazon S3 as a deployment provider 240
Option 1: Deploy static website files to Amazon S3 242
Option 2: Deploy built archive files to Amazon S3 from an S3 source bucket 246
Tutorial: Publish applications to the AWS Serverless Application Repository 252
Before you begin 253
Step 1: Create a buildspec.yml file 253
Step 2: Create and configure your pipeline 254
Step 3: Deploy the publish application 256
Step 4: Create the publish action 256
Tutorial: Lambda function deployments with CodePipeline 257
Prerequisites 258
Step 1: Create your sample Lambda function 258
Step 2: Upload the function file to your repository 259
Step 3: Creating your pipeline 259
Step 4: Test Your Pipeline 262

Learn more ... 263
Tutorial: Using variables with Lambda invoke actions ... 263
Prerequisites ... 264
Step 1: Create a Lambda function ... 264
Step 2: Add a Lambda invoke action and manual approval action to your pipeline ... 267
Tutorial: Use an AWS Step Functions invoke action ... 268
Prerequisite: Create or choose a simple pipeline ... 269
Step 1: Create the sample state machine ... 270
Step 2: Add a Step Functions invoke action to your pipeline ... 270
Tutorial: Create a pipeline that uses AppConfig as a deployment provider ... 271
Prerequisites ... 272
Step 1: Create your AWS AppConfig resources ... 272
Step 2: Upload files to your S3 source bucket ... 273
Step 3: Create your pipeline ... 274
Step 4: Make a change to any source file and verify deployment ... 275
Tutorial: Use full clone with a GitHub pipeline source ... 275
Prerequisites ... 276
Step 1: Create a README file ... 276
Step 2: Create your pipeline and build project ... 277
Step 3: Update the CodeBuild service role policy to use connections ... 280
Step 4: View repository commands in build output ... 281
Tutorial: Use full clone with a CodeCommit pipeline source ... 281
Prerequisites ... 282
Step 1: Create a README file ... 282
Step 2: Create your pipeline and build project ... 282
Step 3: Update the CodeBuild service role policy to clone the repository ... 285
Step 4: View repository commands in build output ... 285
Tutorial: Create a pipeline with AWS CloudFormation StackSets deployment actions ... 286
Prerequisites ... 287
Step 1: Upload the sample AWS CloudFormation template and parameter file ... 287
Step 2: Create your pipeline ... 289
Step 3: View initial deployment ... 292
Step 4: Add a CloudFormationStackInstances action ... 292
Step 5: View stack set resources for your deployment ... 293
Step 6: Make an update to your stack set ... 294
Create a variable check rule for a pipeline as an entry condition ... 294

Prerequisites ... 295
Step 1: Create sample source file and add to your GitHub repository ... 296
Step 2: Create your pipeline ... 296
Step 2: Edit the build stage to add the condition and rule ... 299
Step 3: Run the pipeline and view resolved variables ... 301
Best practices and use cases ... 304
Examples of how to use CodePipeline ... 304
Use CodePipeline with Amazon S3, AWS CodeCommit, and AWS CodeDeploy ... 304
Use CodePipeline with third-party action providers (GitHub and Jenkins) ... 305
Use CodePipeline to compile, build, and test code with CodeBuild ... 305
Use CodePipeline with Amazon ECS for continuous delivery of container-based applications to the cloud ... 306
Use CodePipeline with Elastic Beanstalk for continuous delivery of web applications to the cloud ... 306
Use CodePipeline with AWS Lambda for continuous delivery of Lambda-based and serverless applications ... 306
Use CodePipeline with CloudFormation templates for continuous delivery to the cloud ... 306
Use CodePipeline with Amazon VPC ... 307
Availability ... 307
Create a VPC endpoint for CodePipeline ... 308
Troubleshooting your VPC setup ... 308
Define CI/CD pipelines with stages and actions ... 310
Create a pipeline, stages, and actions ... 311
Create a custom pipeline (console) ... 312
Create a pipeline (CLI) ... 324
Create a pipeline from static templates ... 330
Edit a pipeline ... 337
Edit a pipeline (console) ... 338
Edit a pipeline (AWS CLI) ... 341
View pipelines and details ... 345
View pipelines (console) ... 346
View action details in a pipeline (console) ... 350
View the pipeline ARN and service role ARN (console) ... 353
View pipeline details and history (CLI) ... 354
View rule results for stage conditions in execution history ... 355
Delete a pipeline ... 358

Delete a pipeline (console) ... 358
Delete a pipeline (CLI) ... 358
Create a pipeline that uses resources from another account ... 360
Prerequisite: Create an AWS KMS encryption key ... 362
Step 1: Set up account policies and roles ... 362
Step 2: Edit the pipeline ... 369
Migrate polling pipelines to use event-based change detection ... 373
How to migrate polling pipelines ... 373
Viewing polling pipelines in your account ... 374
Migrate polling pipelines with a CodeCommit source ... 380
Migrate polling pipelines with an S3 source enabled for events ... 401
Migrate polling pipelines with an S3 source and CloudTrail trail ... 421
Migrate polling pipelines for a GitHub (via OAuth app) source action to connections ... 456
Migrate polling pipelines for a GitHub (via OAuth app) source action to webhooks ... 460
Create the CodePipeline service role ... 477
Create the CodePipeline service role (console) ... 477
Create the CodePipeline service role (CLI) ... 478
Tagging resources ... 480
Tag a pipeline ... 481
Tag pipelines (console) ... 481
Tag pipelines (CLI) ... 483
Create a notification rule ... 485
... 489
Connect to first-party source providers using source actions ... 492
Amazon ECR source actions and EventBridge ... 492
Create an EventBridge rule for an Amazon ECR source (console) ... 493
Create an EventBridge rule for an Amazon ECR source (CLI) ... 495
Create an EventBridge rule for an Amazon ECR source (CloudFormation template) ... 499
Connecting to Amazon S3 source actions that use EventBridge events ... 504
Create pipelines with an S3 source enabled for events (CLI) ... 505
Create pipelines with an S3 source enabled for events (CloudFormation template) ... 509
Connecting to Amazon S3 source actions that use EventBridge and AWS CloudTrail ... 523
Create an EventBridge rule for an Amazon S3 source (console) ... 525
Create an EventBridge rule for an Amazon S3 source (CLI) ... 528
Create an EventBridge rule for an Amazon S3 source (CloudFormation template) ... 534
CodeCommit source actions and EventBridge ... 547

Create an EventBridge rule for a CodeCommit source (console) ... 548
Create an EventBridge rule for a CodeCommit source (CLI) ... 550
Create an EventBridge rule for a CodeCommit source (CloudFormation template) ... 555
Add third-party source providers to pipelines using AWS CodeConnections ... 564
Azure DevOps connections ... 564
Create a connection to Azure DevOps (console) ... 566
Create a connection to Azure DevOps (CLI) ... 569
Bitbucket Cloud connections ... 570
Create a connection to Bitbucket Cloud (console) ... 572
Create a connection to Bitbucket Cloud (CLI) ... 575
GitHub connections ... 577
Create a connection to GitHub (console) ... 579
Create a connection to GitHub (CLI) ... 582
GitHub Enterprise Server connections ... 584
Create a connection to GitHub Enterprise Server (console) ... 585
Create a host and connection to GitHub Enterprise Server (CLI) ... 589
GitLab.com connections ... 592
Create a connection to GitLab.com (console) ... 594
Create a connection to GitLab.com (CLI) ... 598
Connections for GitLab self-managed ... 600
Create a connection to GitLab self-managed (console) ... 601
Create a host and connection to GitLab self-managed (CLI) ... 605
Use a connection shared with another account ... 607
Automate starting pipelines using triggers and filtering ... 608
Considerations for trigger filters ... 610
Pull request events for triggers by provider ... 611
Examples for trigger filters ... 612
Add trigger on code push with no filter ... 620
Add trigger with code push or pull request event types ... 621
Add filters for push and pull request event types (console) ... 622
Add filters for push and pull request event types (CLI) ... 624
Add filters for push and pull request event types (CloudFormation templates) ... 627
Add trigger to turn off change detection ... 628
Start and stop pipelines manually ... 630
Start a pipeline in CodePipeline ... 630
Start a pipeline manually ... 632

Start a pipeline on a schedule ... 633
Start a pipeline with a source revision override ... 637
Stop a pipeline execution ... 639
Stop a pipeline execution (console) ... 640
Stop an Inbound Execution (Console) ... 644
Stop a pipeline execution (CLI) ... 644
Stop an Inbound Execution (CLI) ... 646
View history and set the mode for pipeline executions ... 648
View executions ... 648
View pipeline execution history (console) ... 648
View execution status (console) ... 650
View an inbound execution (Console) ... 652
View pipeline execution source revisions (console) ... 653
View action executions (console) ... 655
View action artifacts and artifact store information (console) ... 655
View pipeline details and history (CLI) ... 656
Set or change the pipeline execution mode ... 668
Considerations for viewing execution modes ... 668
Considerations for switching between execution modes ... 671
Set or change the pipeline execution mode (console) ... 672
Set the pipeline execution mode (CLI) ... 673
Roll back or retry stages ... 677
Configuring stage retry for a failed stage or failed actions ... 677
Considerations for stage retry ... 678
Retry a failed stage manually ... 678
Configure a stage for automatic retry on failure ... 682
Configuring stage rollback ... 687
Considerations for rollbacks ... 688
Roll back a stage manually ... 688
Configure a stage for automatic rollback ... 693
View rollback status in execution listing ... 697
View rollback status details ... 700
Configure conditions for a stage ... 705
Use cases for stage conditions ... 706
Considerations for results configured for stage conditions ... 706
Considerations for rules configured for stage conditions ... 707

Creating Entry conditions ... 707
Creating Entry conditions - CloudWatchAlarm rule example (Console) ... 708
Creating Entry conditions with Skip result and `VariableCheck` rule (console) ... 709
Creating Entry conditions (CLI) ... 711
Creating Entry conditions (CFN) ... 713
Creating On Failure conditions ... 714
Creating On Failure conditions (Console) ... 714
Creating onFailure conditions with a Retry result example (Console) ... 715
Creating On Failure conditions (CLI) ... 716
Creating On Failure conditions (CFN) ... 718
Creating On Success conditions ... 719
Creating On Success conditions (Console) ... 719
Creating On Success conditions (CLI) ... 721
Create an On Success condition (CFN) ... 723
Deleting stage conditions ... 724
Overriding stage conditions ... 725
Use action types, custom actions, and approval actions ... 727
Working with action types ... 727
Request an action type ... 729
Add an available action type to a pipeline (console) ... 735
View an action type ... 737
Update an action type ... 738
Create a custom action for a pipeline ... 740
Create a custom action ... 742
Create a job worker for your custom action ... 746
Add a custom action to a pipeline ... 754
Tag a custom action in CodePipeline ... 757
Add tags to a custom action ... 757
View tags for a custom action ... 758
Edit tags for a custom action ... 758
Remove tags from a custom action ... 758
Invoke a Lambda function in a pipeline ... 759
Step 1: Create a pipeline ... 761
Step 2: Create the Lambda function ... 762
Step 3: Add the Lambda function to a pipeline in the CodePipeline console ... 767
Step 4: Test the pipeline with the Lambda function ... 768

Step 5: Next steps ... 768
Example JSON event ... 769
Additional sample functions ... 771
Add a manual approval action to a stage ... 784
Configuration options for manual approval actions ... 784
Setup and workflow overview for approval actions ... 785
Grant approval permissions to an IAM user in CodePipeline ... 786
Grant Amazon SNS permissions to a service role ... 789
Add a manual approval action ... 791
Approve or reject an approval action ... 795
JSON data format for manual approval notifications ... 799
Add a cross-region action to a pipeline ... 800
Manage cross-region actions in a pipeline (console) ... 802
Add a cross-region action to a pipeline (CLI) ... 804
Add a cross-region action to a pipeline (CloudFormation) ... 810
Working with variables ... 812
Configure actions for variables ... 813
View output variables ... 818
Example: Use variables in manual approvals ... 820
Example: Use a BranchName variable with CodeBuild environment variables ... 821
Working with stage transitions ... 824
Disable or enable transitions (console) ... 824
Disable or enable transitions (CLI) ... 826
Monitoring pipelines ... 828
Monitoring CodePipeline events ... 829
Detail types ... 830
Pipeline-level events ... 833
Stage-level events ... 841
Action-level events ... 845
Create a Rule That Sends a Notification on a Pipeline Event ... 853
Events placeholder bucket reference ... 858
Events placeholder bucket names by Region ... 858
Logging API calls with AWS CloudTrail ... 859
CodePipeline information in CloudTrail ... 859
Understanding CodePipeline log file entries ... 860
CodePipeline CloudWatch metrics ... 862

PipelineDuration 862
FailedPipelineExecutions 863
Troubleshooting 864
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" 865
Deployment error: A pipeline configured with an AWS Elastic Beanstalk deploy action hangs instead of failing if the "DescribeEvents" permission is missing 866
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." 866
Pipeline error: A Jenkins build or test action runs for a long time and then fails due to lack of credentials or permissions 866
Pipeline error: A pipeline created in one AWS Region using a bucket created in another AWS Region returns an "InternalError" with the code "JobFailed" 867
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 866
Pipeline artifact folder names appear to be truncated 868
Add CodeBuild GitClone permissions for connections to Bitbucket, GitHub, GitHub Enterprise Server, or GitLab.com 868
Add CodeBuild GitClone permissions for CodeCommit source actions 870
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>" 871
GitHub (via OAuth app) source action: Repository list shows different repositories 872
GitHub (via GitHub App) source action: Unable to complete the connection for a repository . 872
Amazon S3 error: CodePipeline service role <ARN> is getting S3 access denied for the S3 bucket <BucketName> 872
Pipelines with an Amazon S3, Amazon ECR, or CodeCommit source no longer start automatically 875
Connections error when connecting to GitHub: "A problem occurred, make sure cookies are enabled in your browser" or "An organization owner must install the GitHub app" 876
Pipelines with execution mode changed to QUEUED or PARALLEL mode fail when run limit reached 877
Pipelines in PARALLEL mode have an outdated pipeline definition if edited when changing to QUEUED or SUPERSEDED mode 877

Pipelines changed from PARALLEL mode will display a previous execution mode 878
Pipelines with connections that use trigger filtering by file paths might not start at branch creation 878
Pipelines with connections that use trigger filtering by file paths might not start when file limit is reached 878
CodeCommit or S3 source revisions in PARALLEL mode might not match EventBridge event 879
EC2 Deploy action fails with an error message `No such file` 879
EKS Deploy action fails with a `cluster unreachable` error message 880
Pipeline does not trigger for all branches when multiple source actions reference the same repository 881
Need help with a different issue? 881
Security 883
Data protection 883
Internetwork traffic privacy 885
Encryption at rest 885
Encryption in transit 885
Configure server-side encryption for artifacts stored in Amazon S3 for CodePipeline 885
Use AWS Secrets Manager to track database passwords or third-party API keys 889
Identity and access management 889
Audience 890
Authenticating with identities 890
Managing access using policies 891
How AWS CodePipeline works with IAM 893
Identity-based policy examples 898
Resource-based policy examples 926
Troubleshooting 927
permissions reference 929
Manage the CodePipeline service role 939
Incident response 945
Compliance validation 946
Resilience 946
Infrastructure security 946
Security best practices 947
Pipeline structure reference 948
Pipeline declaration 952

name ... 954
roleArn ... 955
artifactStore OR artifactStores ... 955
stages ... 956
version ... 957
executionMode ... 957
pipelineType ... 957
variables ... 958
triggers ... 958
metadata ... 962
Stage declaration ... 963
name ... 966
actions ... 966
conditions ... 966
rules ... 967
Action declaration ... 967
name ... 971
region ... 971
roleArn ... 971
namespace ... 972
actionTypeId ... 972
InputArtifacts ... 973
outputArtifacts ... 974
configuration (by action provider) ... 975
runOrder ... 976
Valid action providers in CodePipeline ... 977
Valid settings for the PollForSourceChanges parameter ... 982
Valid input and output artifacts for each action type ... 984
Valid configuration parameters for each provider type ... 986
Action structure reference ... 990
Amazon EC2 action reference ... 991
Action type ... 992
Configuration parameters ... 992
Input artifacts ... 995
Output artifacts ... 996
Service role policy permissions for the EC2 deploy action ... 996

Deploy spec file reference ... 998
Action declaration ... 1000
Action declaration with Deploy spec example ... 1001
See also ... 1003
Amazon ECR source action reference ... 1003
Action type ... 1004
Configuration parameters ... 1004
Input artifacts ... 1004
Output artifacts ... 1005
Output variables ... 1005
Service role permissions: Amazon ECR action ... 1006
Action declaration (Amazon ECR example) ... 1006
See also ... 1007
ECRBuildAndPublish build action reference ... 1007
Action type ... 1008
Configuration parameters ... 1009
Input artifacts ... 1010
Output artifacts ... 1010
Output variables ... 1010
Service role permissions: ECRBuildAndPublish action ... 1010
Action declaration ... 1012
See also ... 1013
Amazon ECS and CodeDeploy blue-green deploy action reference ... 1013
Action type ... 1014
Configuration parameters ... 1015
Input artifacts ... 1016
Output artifacts ... 1017
Service role permissions: CodeDeployToECS action ... 1017
Action declaration ... 1019
See also ... 1021
Amazon Elastic Container Service deploy action reference ... 1022
Action type ... 1023
Configuration parameters ... 1023
Input artifacts ... 1024
Output artifacts ... 1024
Service role permissions: Amazon ECS standard action ... 1024

Action declaration ... 1026
See also ... 1027
Amazon Elastic Kubernetes Service EKS deploy action reference ... 1028
Action type ... 1029
Configuration parameters ... 1029
Input artifacts ... 1031
Output artifacts ... 1031
Environment variables ... 1031
Output variables ... 1031
Service role policy permissions ... 1032
Action declaration ... 1034
See also ... 1035
AWS Lambda deploy action reference ... 1035
Action type ... 1036
Configuration parameters ... 1036
Input artifacts ... 1040
Output artifacts ... 1040
Output variables ... 1040
Service role policy permissions for the Lambda deploy action ... 1040
Action declaration ... 1042
See also ... 1043
Amazon S3 deploy action reference ... 1043
Action type ... 1044
Configuration parameters ... 1044
Input artifacts ... 1045
Output artifacts ... 1046
Service role permissions: S3 deploy action ... 1046
Example action configuration ... 1047
See also ... 1049
Amazon S3 source action reference ... 1050
Action type ... 1051
Configuration parameters ... 1051
Input artifacts ... 1053
Output artifacts ... 1053
Output variables ... 1053
Service role permissions: S3 source action ... 1054

Action declaration ... 1055
See also ... 1056
AWS AppConfig deploy action reference ... 1056
Action type ... 1056
Configuration parameters ... 1057
Input artifacts ... 1057
Output artifacts ... 1057
Service role permissions: AppConfig action ... 1058
Example action configuration ... 1058
See also ... 1060
CloudFormation deploy action reference ... 1060
Action type ... 1061
Configuration parameters ... 1061
Input artifacts ... 1066
Output artifacts ... 1066
Output variables ... 1066
Service role permissions: CloudFormation action ... 1067
Action declaration ... 1068
See also ... 1070
CloudFormation StackSets ... 1070
How CloudFormation StackSets actions work ... 1071
How to structure StackSets actions in a pipeline ... 1073
The CloudFormationStackSet action ... 1074
The CloudFormationStackInstances action ... 1088
Service role permissions: CloudFormationStackSet action ... 1098
Service role permissions: CloudFormationStackInstances action ... 1098
Permissions models for stack set operations ... 1099
Template parameter data types ... 1099
See also ... 1070
AWS CodeBuild build and test action reference ... 1101
Action type ... 1102
Configuration parameters ... 1102
Input artifacts ... 1105
Output artifacts ... 1106
Output variables ... 1107
Service role permissions: CodeBuild action ... 1107

Action declaration (CodeBuild example) ... 1108
See also ... 1109
AWS CodePipeline invoke action reference ... 1110
Action type ... 1110
Configuration parameters ... 1111
Input artifacts ... 1113
Output artifacts ... 1114
Service role policy permissions for the CodePipeline invoke action ... 1114
Action declaration ... 1114
See also ... 1115
AWS CodeCommit source action reference ... 1115
Action type ... 1116
Configuration parameters ... 1117
Input artifacts ... 1118
Output artifacts ... 1118
Output variables ... 1119
Service role permissions: CodeCommit action ... 1119
Example action configuration ... 1120
See also ... 1122
AWS CodeDeploy deploy action reference ... 1123
Action type ... 1123
Configuration parameters ... 1124
Input artifacts ... 1124
Output artifacts ... 1124
Service role permissions: AWS CodeDeploy action ... 1124
Action declaration ... 1126
See also ... 1127
CodeStarSourceConnection for Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed actions ... 1127
Action type ... 1131
Configuration parameters ... 1131
Input artifacts ... 1132
Output artifacts ... 1132
Output variables ... 1133
Service role permissions: AWS CodeConnections action ... 1134
Action declaration ... 1134

Installing the installation app and creating a connection ... 1136
See also ... 1136
Commands action reference ... 1137
Considerations for the Commands action ... 1138
Service role policy permissions ... 1139
Action type ... 1140
Configuration parameters ... 1140
Input artifacts ... 1145
Output artifacts ... 1145
Environment variables ... 1145
Service role permissions: Commands action ... 1147
Action declaration (example) ... 1147
See also ... 1149
AWS Device Farm test action reference ... 1149
Action type ... 1149
Configuration parameters ... 1150
Input artifacts ... 1154
Output artifacts ... 1154
Service role permissions: AWS Device Farm action ... 1154
Action declaration ... 1155
See also ... 1156
Elastic Beanstalk deploy action reference ... 1156
Action type ... 1157
Configuration parameters ... 1157
Input artifacts ... 1157
Output artifacts ... 1158
Service role permissions: `ElasticBeanstalk` deploy action ... 1158
Action declaration ... 1159
See also ... 1160
Amazon Inspector `InspectorScan` invoke action reference ... 1160
Action type ID ... 1161
Configuration parameters ... 1161
Input artifacts ... 1164
Output artifacts ... 1164
Output variables ... 1164
Service role permissions: `InspectorScan` action ... 1164

Action declaration ... 1165
See also ... 1166
AWS Lambda invoke action reference ... 1166
Action type ... 1167
Configuration parameters ... 1167
Input artifacts ... 1168
Output artifacts ... 1168
Output variables ... 1168
Example action configuration ... 1168
Example JSON event ... 1169
See also ... 1172
AWS OpsWorks deploy action reference ... 1172
Action type ... 1172
Configuration parameters ... 1172
Input artifacts ... 1173
Output artifacts ... 1173
Service role permissions: AWS OpsWorks action ... 1173
Example action configuration ... 1174
See also ... 1174
AWS Service Catalog deploy action reference ... 1175
Action type ... 1175
Configuration parameters ... 1175
Input artifacts ... 1176
Output artifacts ... 1176
Service role permissions: Service Catalog action ... 1176
Example action configurations by type of configuration file ... 1177
See also ... 1178
AWS Step Functions ... 1178
Action type ... 1172
Configuration parameters ... 1179
Input artifacts ... 1180
Output artifacts ... 1181
Output variables ... 1181
Service role permissions: `StepFunctions` action ... 1181
Example action configuration ... 1182
Behavior ... 1185

See also ... 1060
Rule structure reference ... 1188
CloudWatchAlarm ... 1188
Rule type ... 1189
Configuration parameters ... 1189
Example rule configuration ... 1189
See also ... 1190
CodeBuild rule ... 1191
Service role policy permissions ... 1191
Rule type ... 1192
Configuration parameters ... 1192
Example rule configuration ... 1194
See also ... 1195
Commands ... 1195
Considerations for the Commands rule ... 1195
Service role policy permissions ... 1191
Rule type ... 1192
Configuration parameters ... 1192
Example rule configuration ... 1194
See also ... 1195
DeploymentWindow ... 1200
Rule type ... 1201
Configuration parameters ... 1201
Example rule configuration ... 1203
See also ... 1204
LambdaInvoke ... 1204
Rule type ... 1189
Configuration parameters ... 1205
Example rule configuration ... 1205
See also ... 1206
VariableCheck ... 1206
Rule type ... 1207
Configuration parameters ... 1207
Example rule configuration ... 1210
See also ... 1210
Integration model reference ... 1212

How third-party action types work with the integrator ... 1212
Concepts ... 1213
Supported integration models ... 1214
Lambda integration model ... 1216
Update your Lambda function to handle the input from CodePipeline ... 1216
Return the results from your Lambda function to CodePipeline ... 1221
Use continuation tokens to wait for results from an asynchronous process ... 1222
Provide CodePipeline the permissions to invoke the integrator Lambda function at runtime ... 1223
Job worker integration model ... 1223
Choose and configure a permissions management strategy for your job worker ... 1224
Image definitions file reference ... 1226
imagedefinitions.json file for Amazon ECS standard deployment actions ... 1226
imageDetail.json file for Amazon ECS blue/green deployment actions ... 1229
Variables reference ... 1234
Concepts ... 1235
Variables ... 1235
Namespaces ... 1236
Use cases for variables ... 1237
Configuring variables ... 1238
Configuring variables at the pipeline level ... 1238
Configuring variables at the action level ... 1239
Variable resolution ... 1241
Rules for variables ... 1242
Variables available for pipeline actions ... 1242
Actions with defined variable keys ... 1242
Actions with user-configured variable keys ... 1248
Working with glob patterns in syntax ... 1251
Update polling pipelines to the recommended change detection method ... 1253
Update a GitHub (via OAuth app) source action to a GitHub (via GitHub App) source action 1254
Step 1: Replace your (via OAuth app) GitHub action ... 1255
Step 2: Create a connection to GitHub ... 1255
Step 3: Save your GitHub source action ... 1256
Quotas ... 1258
Appendix A: GitHub (via OAuth app) source actions ... 1274
Adding a GitHub (via OAuth app) source action ... 1275

GitHub (via OAuth app) source action reference 1276
Action type 1277
Configuration parameters 1277
Input artifacts 1279
Output artifacts 1279
Output variables 1279
Action declaration (GitHub example) 1280
Connecting to GitHub (OAuth) 1281
See also 1281
Document history 1283
Earlier updates 1314
CodePipeline feature reference 1325

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
- What can I do with CodePipeline?
- A quick look at CodePipeline
- How do I get started with CodePipeline?
- CodePipeline concepts
- DevOps pipeline example
- How pipeline executions work
- Input and output artifacts
- How do stage conditions work?
- Pipeline types
- What type of pipeline is right for me?

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 main 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.

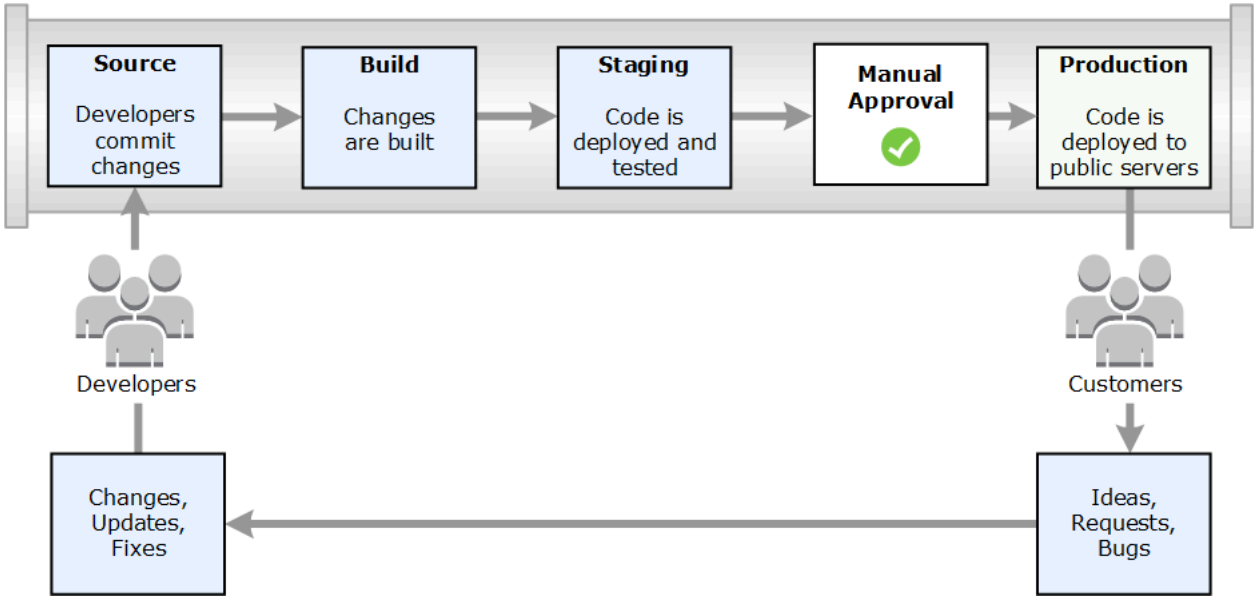
What can I do with CodePipeline?

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.
- **View progress at a glance**: You can review real-time status of your pipelines, check the details of any alerts, retry failed stages or 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.

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 section.
2. **Prepare** to use CodePipeline by following the steps in Getting started with CodePipeline.
3. **Experiment** with CodePipeline by following the steps in the CodePipeline tutorials tutorials.

4. **Use** CodePipeline for your new or existing projects by following the steps in Create a pipeline, stages, and actions.

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.

The following terms are used in CodePipeline:

Topics

- Pipelines
- Pipeline executions
- Stage operations
- Action executions
- Execution types
- Action types
- Artifacts
- Source revisions
- Triggers
- Variables
- Conditions
- Rules

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*.

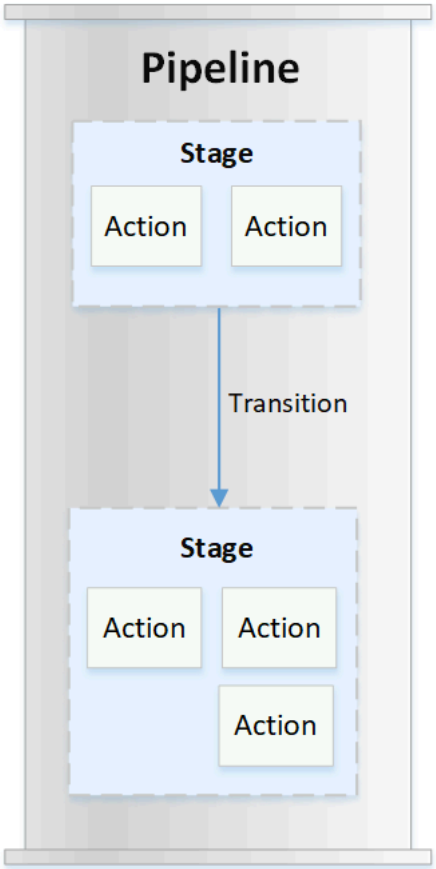
Stages

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*.

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.



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 providers in CodePipeline.

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. The execution waiting to enter the occupied stage is an *inbound execution*. An inbound execution can still fail, be superseded, or be manually stopped. For more information about how inbound executions work, see How Inbound Executions Work.

Pipeline executions traverse pipeline stages in order. Valid statuses for pipelines are `InProgress`, `Stopping`, `Stopped`, `Succeeded`, `Superseded`, and `Failed`.

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.

Failed executions

If an execution fails, it stops and does not completely traverse the pipeline. Its status changes to `FAILED` and CodePipeline unlocks the stage. 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. You can roll back a failed stage to a previous successful execution.

Execution modes

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. SUPERSEDED is the default execution mode.

In SUPERSEDED mode, 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. Other available execution modes are PARALLEL or QUEUED mode.

For more information about execution modes and locked stages, see How executions are processed in SUPERSEDED mode.

Stage operations

When a pipeline execution runs through a stage, the stage is in the process of completing all of the actions within it. For information about how stage operations work and information about locked stages, see How executions are processed in SUPERSEDED mode.

Valid statuses for stages are `InProgress`, `Stopping`, `Stopped`, `Succeeded`, and `Failed`. You can retry a failed stage unless the failed stage is not retryable. For more information, see StageExecution. You can roll back a stage to a specified previous successful execution. A stage can be configured to roll back automatically on failure as detailed in Configuring stage rollback. For more information, see RollbackStage.

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.

Execution types

A pipeline or stage execution can be either a standard or a rolled-back execution.

For standard types, the execution has a unique ID and is a full pipeline run. A pipeline rollback has a stage to be rolled back and a successful execution for the stage as the target execution to which to roll back. The target pipeline execution is used to retrieve source revisions and variables for the stage to rerun.

Action types

Action types are preconfigured actions that are available for selection in CodePipeline. The action type is defined by its owner, provider, version, and category. The action type provides customized parameters that are used to complete the action tasks in a pipeline.

For information about the AWS services and third-party products and services you can integrate into your pipeline based on action type, see Integrations with CodePipeline action types.

For information about the integration models supported for action types in CodePipeline, see Integration model reference.

For information about how third-party providers can set up and manage action types in CodePipeline, see Working with action types.

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.

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 source revisions. For GitHub and CodeCommit repositories, this is the commit. For S3 buckets or actions, this is the object version.

You can start a pipeline execution with a source revision, such as a commit, that you specify. The execution will process the specified revision and override what would have been the revision used for the execution. For more information, see Start a pipeline with a source revision override.

Triggers

Triggers are events that start your pipeline. Some triggers, such as starting a pipeline manually, are available for all source action providers in a pipeline. Certain triggers depend on the source provider for a pipeline. For example, CloudWatch events must be configured with event resources from Amazon CloudWatch that have the pipeline ARN added as a target in the event rule. Amazon CloudWatch Events is the recommended trigger for automatic change detection for pipelines with a CodeCommit or S3 source action. Webhooks are a type of trigger configured for third-party repository events. For example, WebhookV2 is a trigger type that allows Git tags to be used to start pipelines with third-party source providers such as GitHub.com, GitHub Enterprise Server, GitLab.com, GitLab self-managed, or Bitbucket Cloud. In the pipeline configuration, you can specify a filter for triggers, such as push or pull request. You can filter code push events on Git tags, branches, or file paths. You can filter pull request events on event (opened, updated, closed), branches, or file paths.

For more information about triggers, see Start a pipeline in CodePipeline. For a tutorial that walks you through using Git tags as triggers for your pipeline, see Tutorial: Use Git tags to start your pipeline.

Important

Pipelines that are inactive for longer than 30 days will have polling disabled for the pipeline. For more information, see pollingDisabledAt in the pipeline structure reference. For the steps to migrate your pipeline from polling to event-based change detection, see Change Detection Methods.

Variables

A *variable* is a value that can be used to dynamically configure actions in your pipeline. Variables can be either declared on the pipeline level, or emitted by actions in the pipeline. Variable values are resolved at the time of pipeline execution and can be viewed in the execution history. For variables declared at the pipeline level, you can either define default values in the pipeline configuration, or override them for a given execution. For variables emitted by an action, the value is available after an action successfully completes. For more information, see Variables reference.

Conditions

A *condition* contains a set of rules that are evaluated. If all rules in a condition succeed, then the condition is met. You can configure conditions so that when the criteria are not met, the specified result, such as failing the stage, engages. Conditions are also referred to as gates because they allow you to specify when an execution will enter and run through a stage or exit the stage after running through it. This is analogous to allowing a line of traffic on a roadway to gather at a closed gate, and then specifying the opening of the gate to allow flow of traffic into an area. Results for condition types include failing the stage or rolling back the stage. Conditions help you to specify when these actions happen in a pipeline stage. You can override conditions at runtime.

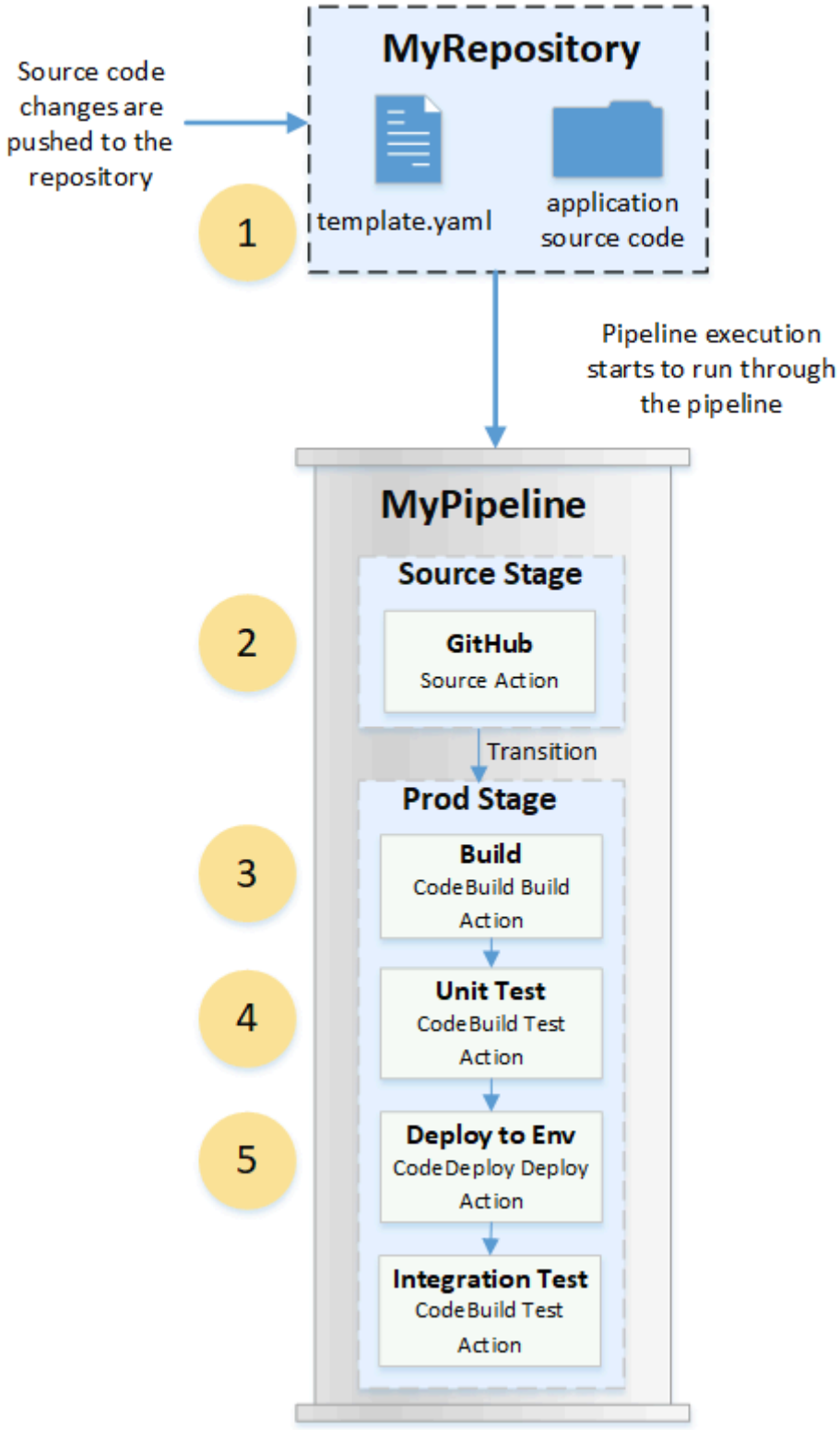
There are three types of conditions. Entry conditions answer the question “If the rules for the condition are met, then enter the stage.” The stage is locked when the execution enters the stage, and then the rules are run. For On Failure conditions, the rule engages when a stage has failed, with a result of rolling back a failed stage. For On Success conditions, the rule engages when a stage is successful, such as checking a successful run for alarms before proceeding. For example, an On Success condition would result in a rollback of a successful stage if the CloudWatchAlarm rule finds that there are alarms in the deployment environment. For more information, see How do stage conditions work?.

Rules

Conditions use one or more preconfigured *rules* that run and perform checks that will then engage the configured result for when the condition is not met. For example, meeting all rules for an Entry condition rule that checks for alarm status and deployment window times will deploy a successful stage after all checks pass. For more information, see How do stage conditions work?.

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.

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 pipeline is started manually or a change is made to the source code. CodePipeline uses the following execution modes to handle the way each execution progresses through the pipeline. For more information, see Set or change the pipeline execution mode.

- SUPERSEDED mode: A more recent execution can overtake an older one. This is the default.
- QUEUED mode: Executions are processed one by one in the order that they are queued. This requires pipeline type V2.

- PARALLEL mode: In PARALLEL mode, executions run simultaneously and independently of one another. Executions don't wait for other runs to complete before starting or finishing. This requires pipeline type V2.

> **Important**
>
> For pipelines in PARALLEL mode, stage rollback is not available. Similarly, failure conditions with a rollback result type cannot be added to a PARALLEL mode pipeline.

How pipeline executions are started

You can start 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 source revisions are processed in pipeline executions

For each pipeline execution that starts with source code changes (source revisions), source revisions are determined as follows.

- For pipelines with a CodeCommit source, the HEAD is cloned by CodePipeline at the moment that the commit is pushed. For example, a commit is pushed, which starts the pipeline for execution 1. At the moment a second commit is pushed, this starts the pipeline for execution 2.

> **Note**
>
> For pipelines in PARALLEL mode with a CodeCommit source, regardless of the commit that triggered the pipeline execution, the source action will always clone the HEAD at the time it is started. For more information, see CodeCommit or S3 source revisions in PARALLEL mode might not match EventBridge event.

- For pipelines with an S3 source, the EventBridge event for the S3 bucket update is used. For example, the event is generated when a file is updated in the source bucket, which starts the pipeline for execution 1. At the moment that the event for a second bucket update is made, this starts the pipeline for execution 2.

  > **Note**
  >
  > For pipelines in PARALLEL mode with an S3 source, regardless of the image tag that triggered the execution, the source action will always start with the latest image tag. For more information, see CodeCommit or S3 source revisions in PARALLEL mode might not match EventBridge event.

- For pipelines with a connections source, such as to Bitbucket, the HEAD is cloned by CodePipeline at the moment that the commit is pushed. For example, for a pipeline in PARALLEL mode, a commit is pushed, which starts the pipeline for execution 1, and the second pipeline execution uses the second commit.

How source overrides work with the EventBridge input transformer

You can use overrides to start a pipeline with a specific source revision ID that you provide for the pipeline execution. For example, if you want to start a pipeline that will process a specific commit ID from your CodeCommit source, you can add the commit ID as an override when you start your pipeline.

There are four types of source revision for `revisionType`:

- `COMMIT_ID`
- `IMAGE_DIGEST`
- `S3_OBJECT_VERSION_ID`
- `S3_OBJECT_KEY`

> **Note**
>
> For the `COMMIT_ID` and `IMAGE_DIGEST` types of source revisions, the source revision ID applies to all content in the repository, across all branches.

> **Note**
>
> For the `S3_OBJECT_VERSION_ID` and `S3_OBJECT_KEY` types of source revisions, either of the types can be used independently, or they can be used together to override the source with a specific ObjectKey and VersionID. For `S3_OBJECT_KEY`, the configuration parameter `AllowOverrideForS3ObjectKey` needs to be set to `true`. For more information on S3 source configuration parameters, see Configuration parameters.

You can specify source overrides using the input transformer in EventBridge. Use the input transformer to pass the data as one of the following:

- You can use the input transformer to pass the data as JSON parameters.
- You can use the input transformer to pass pipeline variables.

For examples of passing the data as JSON parameters, see Amazon ECR source actions and EventBridge resources, Connecting to Amazon S3 source actions that use EventBridge and AWS CloudTrail for S3, and CodeCommit source actions and EventBridge for CodeCommit.

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.

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.

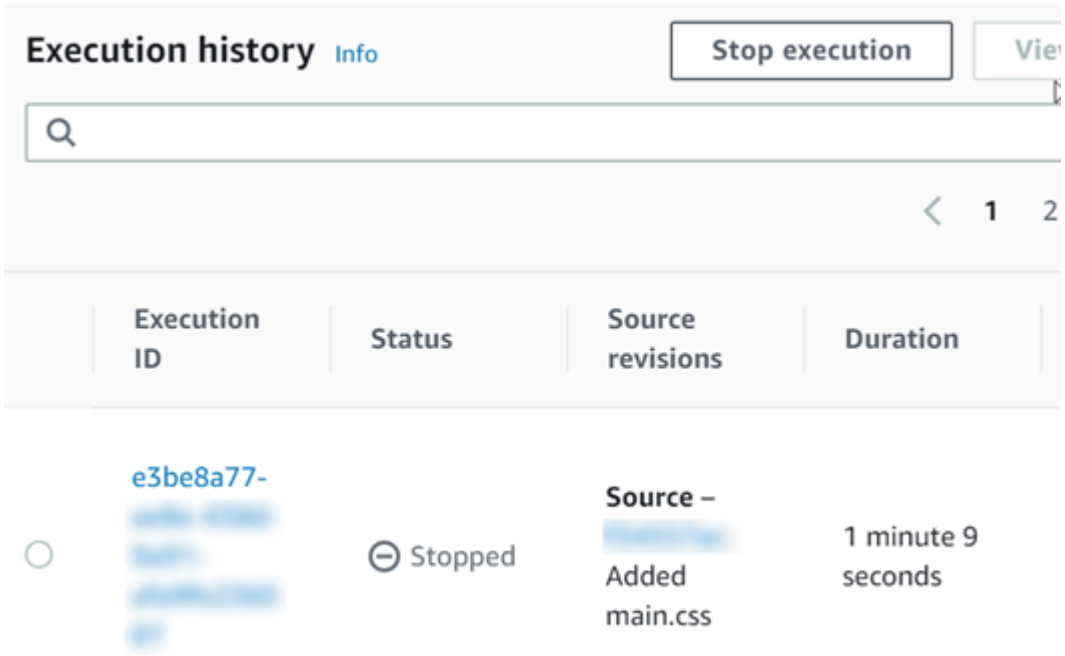
Option 1: Stop and wait

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**.

   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.

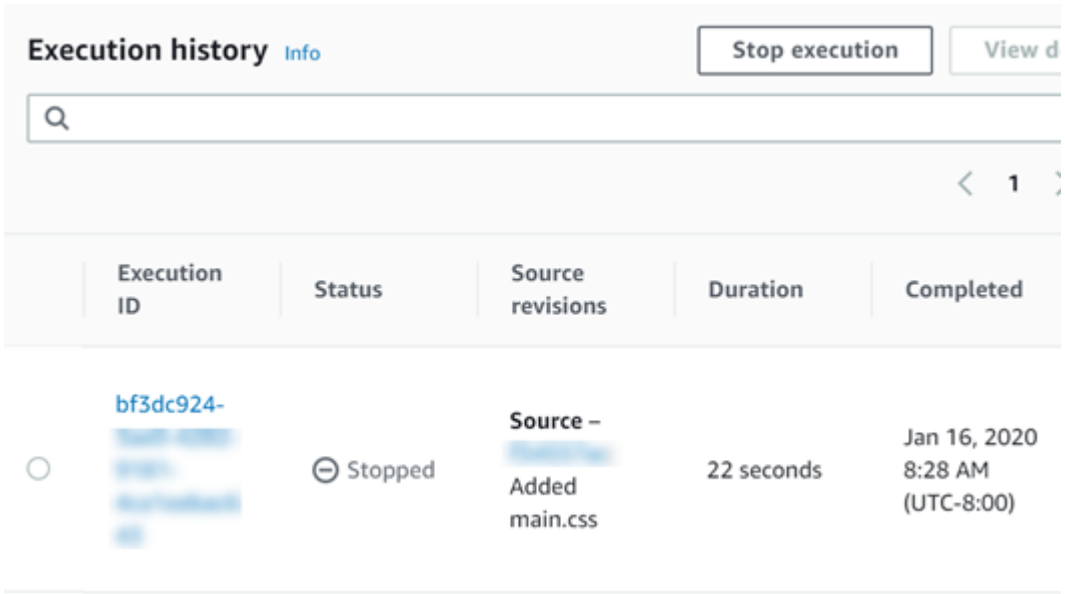
   In the history view, the execution status is **Stopped** after the in-progress action is completed.



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 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.

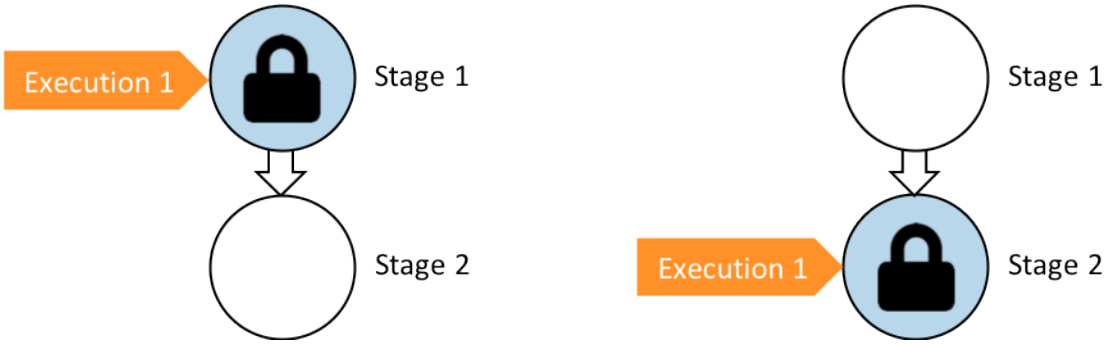
You might want to use the stop and abandon option in the case where you have a custom action. For example, you can abandon a custom action with work that does not need to finish before starting a new execution for a bug fix.

How executions are processed in SUPERSEDED mode

The default mode for processing executions is SUPERSEDED mode. 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 for SUPERSEDED mode.

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.

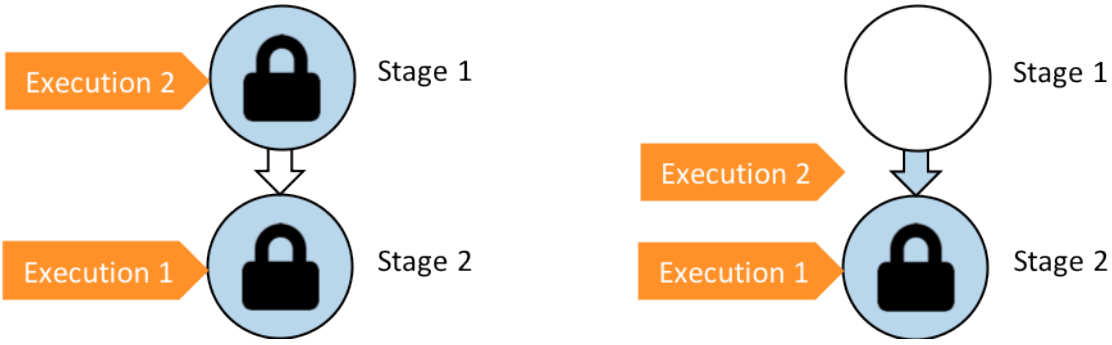


Before: Stage 1 is locked as Execution 1 enters. **After:** Stage 2 is locked as Execution 1 enters.

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.

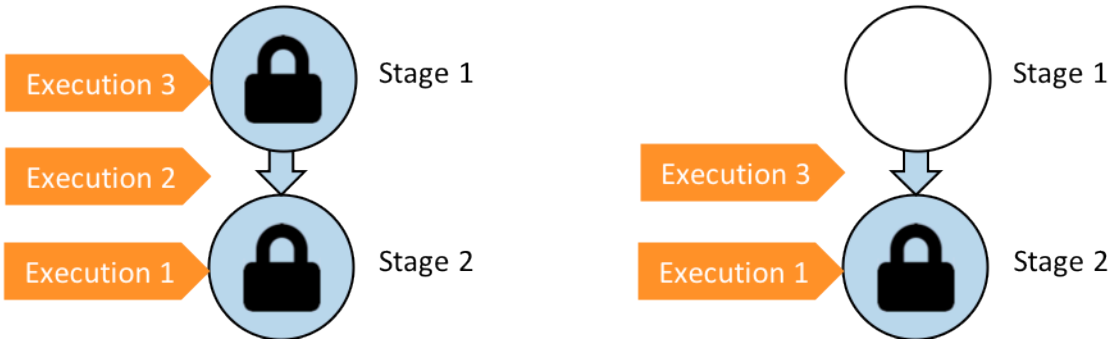
> **Note**
>
> Before you stop an execution, we recommend that you disable the transition in front of the stage. This way, when the stage is unlocked due to the stopped execution, the stage does not accept a subsequent pipeline execution.



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.



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.

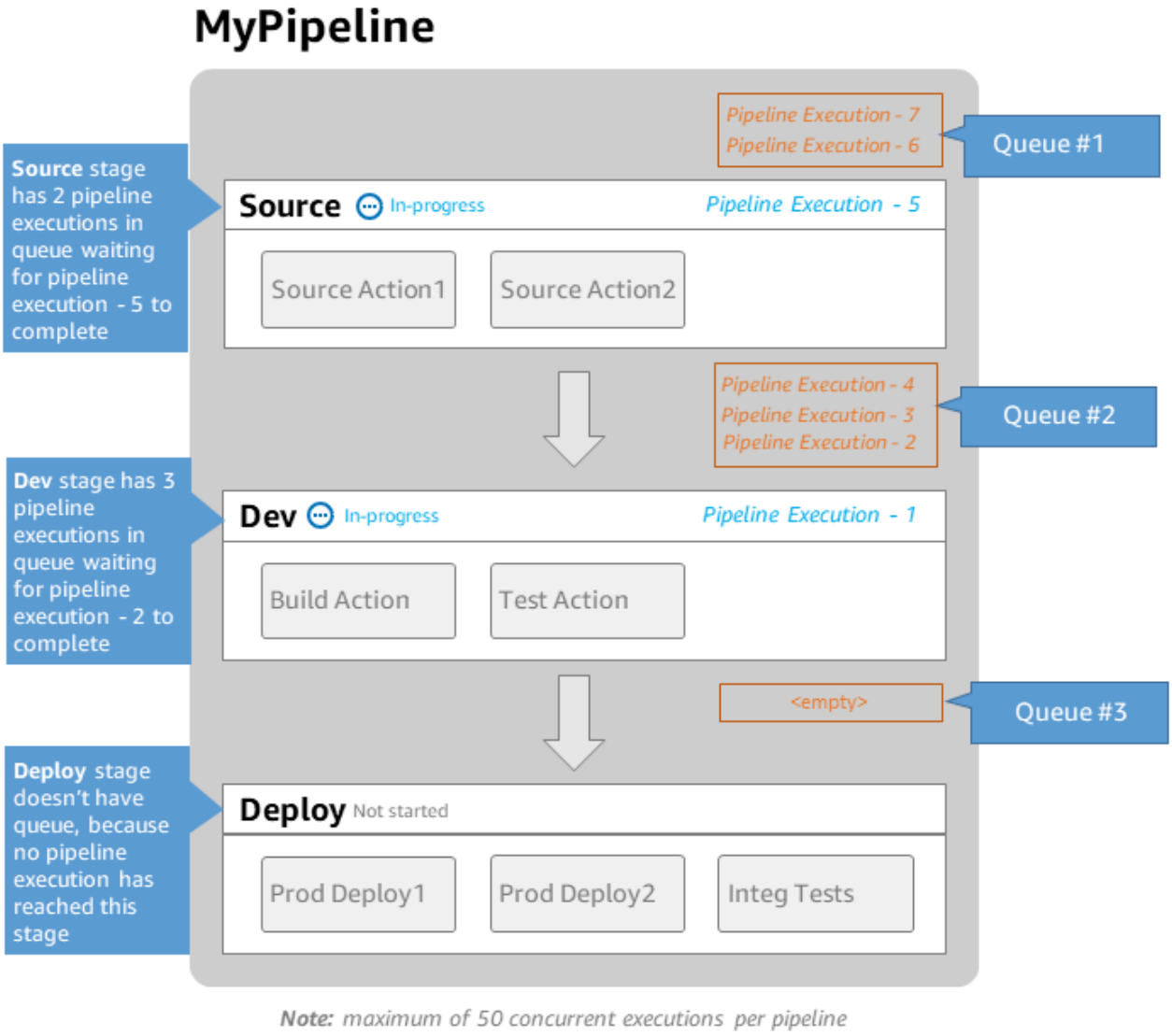
For more information about considerations for viewing and switching between execution modes, see Set or change the pipeline execution mode. For more information about quotas with execution modes, see Quotas in AWS CodePipeline.

How executions are processed in QUEUED mode

For pipelines in QUEUED mode, stages are locked when an execution is being processed; however, waiting executions do not overtake executions that have already started.

Waiting executions gather at the entry points to locked stages in the order that they reach the stage, forming a queue of waiting executions. With QUEUED mode, you can have multiple queues in the same pipeline. When a queued execution enters a stage, the stage is locked and no other executions can enter. This behavior remains the same as SUPERSEDED mode. When the execution finishes the stage, the stage becomes unlocked and ready for the next execution.

The following diagram shows how stages in a QUEUED mode pipeline process executions. For example, while the Source stage processes execution 5, the executions for 6 and 7 form Queue #1 and wait at the stage entry point. The next execution in the queue will be processed after the stage unlocks.



Note: maximum of 50 concurrent executions per pipeline

For more information about considerations for viewing and switching between execution modes, see Set or change the pipeline execution mode. For more information about quotas with execution modes, see Quotas in AWS CodePipeline.

How executions are processed in PARALLEL mode

For pipelines in PARALLEL mode, executions are independent of one another and don't wait for other executions to complete before starting. There are no queues. To view parallel executions in the pipeline, use the execution history view.

Use PARALLEL mode in development environments where each feature has its own feature branch and deploys to targets that are not shared by other users.

For more information about considerations for viewing and switching between execution modes, see Set or change the pipeline execution mode. For more information about quotas with execution modes, see Quotas in AWS CodePipeline.

Managing Pipeline Flow

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. When a transition is disabled, pipeline executions cannot enter the stage. The pipeline execution waiting to enter a stage where the transition is disabled is called the inbound execution. After you enable the transition, an inbound execution 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 identity). 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.

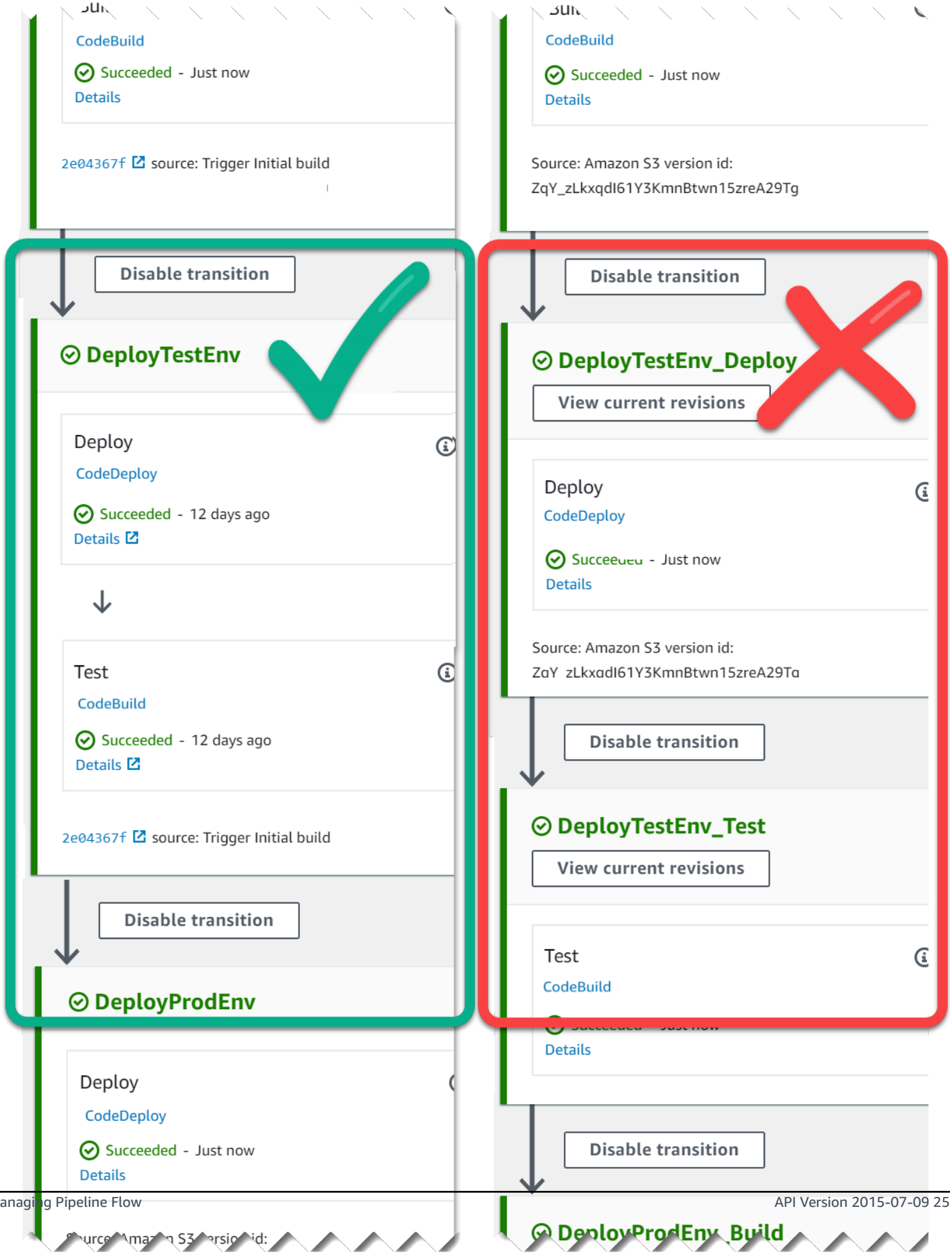
> **Note**
>
> A stage with an approval action is locked until the approval action is approved or rejected or has timed out. A timed-out approval action is processed in the same way as a failed action.

- 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.


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

Left: related test, deploy, and approval actions grouped together (recommended). **Right:** related actions in separate stages (not recommended).

How Inbound Executions Work

An inbound execution is an execution that is waiting for an unavailable stage, transition, or action to become available before it moves forward. The next stage, transition, or action might be unavailable because:

- Another execution has already entered the next stage and locked it.
- The transition to enter the next stage is disabled.

You might disable a transition to hold an inbound execution if you want to control whether a current execution has time to complete in subsequent stages, or if you want to stop all actions at a certain point. To determine if you have an inbound execution, you can view the pipeline in the console or view the output from the **get-pipeline-state** command.

Inbound executions operate with the following considerations:

- As soon as the action, transition, or locked stage becomes available, the in-progress inbound execution enters the stage and continues through the pipeline.
- While the inbound execution is waiting, it can be manually stopped. An inbound execution can have an `InProgress`, `Stopped`, or `Failed` state.
- When an inbound execution has been stopped or has failed, it cannot be retried because there are no failed actions to retry. When an inbound execution has been stopped, and the transition is enabled, the stopped inbound execution does not continue into the stage.

You can view or stop an inbound execution.

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. Artifacts are the files that are worked on by actions in the pipeline, such as files or folders with application code, index page files, scripts, and so on. For example, the Amazon S3 source action artifact is a file name (or file path) where the application source code files are provided for the pipeline source action. The files are provided as a ZIP file, such as the following example artifact name: `SampleApp_Windows.zip`.

The output artifact for the source action, the application source code files, is the output artifact from that action and is the input artifact for the next action, such as a build action. As another example, a build action might run build commands that compile application source code for an input artifact, which is the application source code files. See the action configuration reference page for a specific action for details about artifact parameters, such as AWS CodeBuild build and test action reference for the CodeBuild action.

Actions 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.

> **Note**
>
> The artifact bucket is not the same bucket as the bucket used as the source file location for a pipeline where the chosen source action is S3.

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. For 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.

- When you use the console to create your first pipeline, CodePipeline creates an Amazon S3 bucket in the same AWS account and 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-1a2b-3c4d-67890EXAMPLE*, where *region* is the AWS Region in which you created the pipeline, and *12345EXAMPLE-1a2b-3c4d-67890EXAMPLE* is a random value based on a universally unique identifier (UUID) that ensures the bucket name is unique.

> **Note**
>
> If you already have a bucket starting with codepipeline-*region*- in the Region where you are creating the pipeline, CodePipeline uses that as the default bucket. It also follows lexicographical order; for example, codepipeline-*region-abcexample* is chosen before codepipeline-*region-defexample*.

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 AWS account and AWS 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 account and 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.

How do stage conditions work?

For each condition that specifies a rule, the rule is run. If the condition fails, the stage performs the specified result. Optionally, as part of the rule, you also specify which resources CodePipeline should use for certain cases. For example, the `CloudWatchAlarm` rule will use a CloudWatch alarm resource to run checks for the condition.

A condition might match multiple rules, and each rule can specify one of three providers.

The high-level flow for creating conditions is as follows:

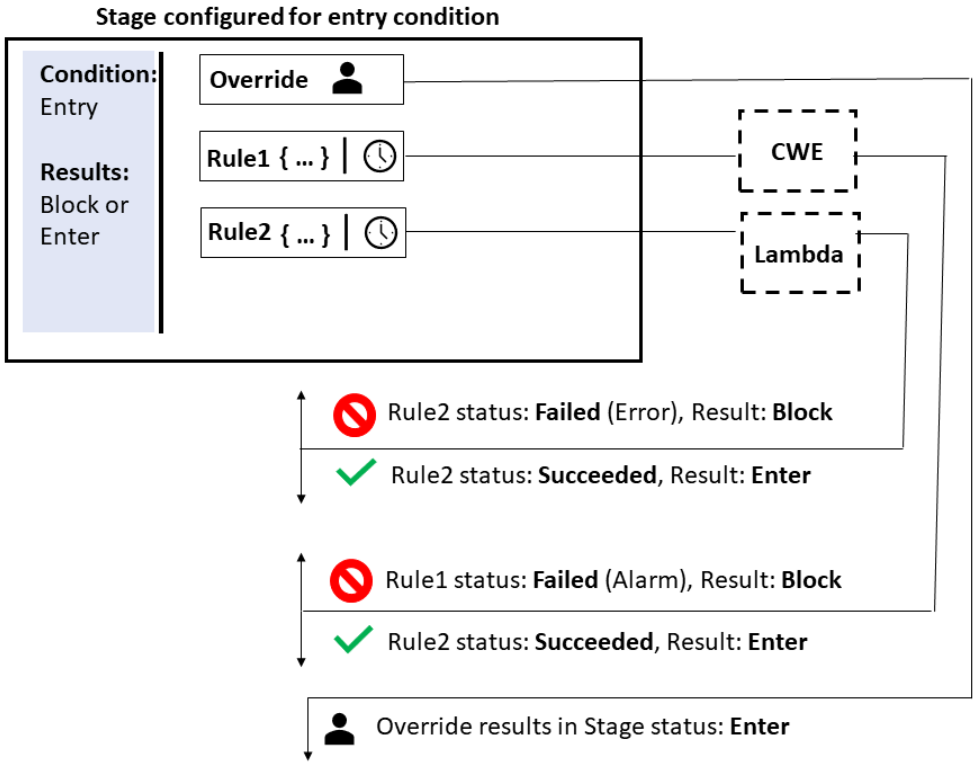
1. Choose the type of condition from the available condition types in CodePipeline. For example, use an On Success condition type to set up a stage so that after the stage succeeds, a set of rules can be used to run checks before proceeding.
2. Choose the rule. For example, the `CloudWatchAlarm` rule will check for alarms and uses EB to check for a preconfigured alarm threshold. If the check is successful, and the alarm is below the threshold, the stage can proceed.
3. Configure the result, such as a rollback that would be used if the rule fails.

Conditions are used for specific types of expressions and each has specific options for results available as follows:

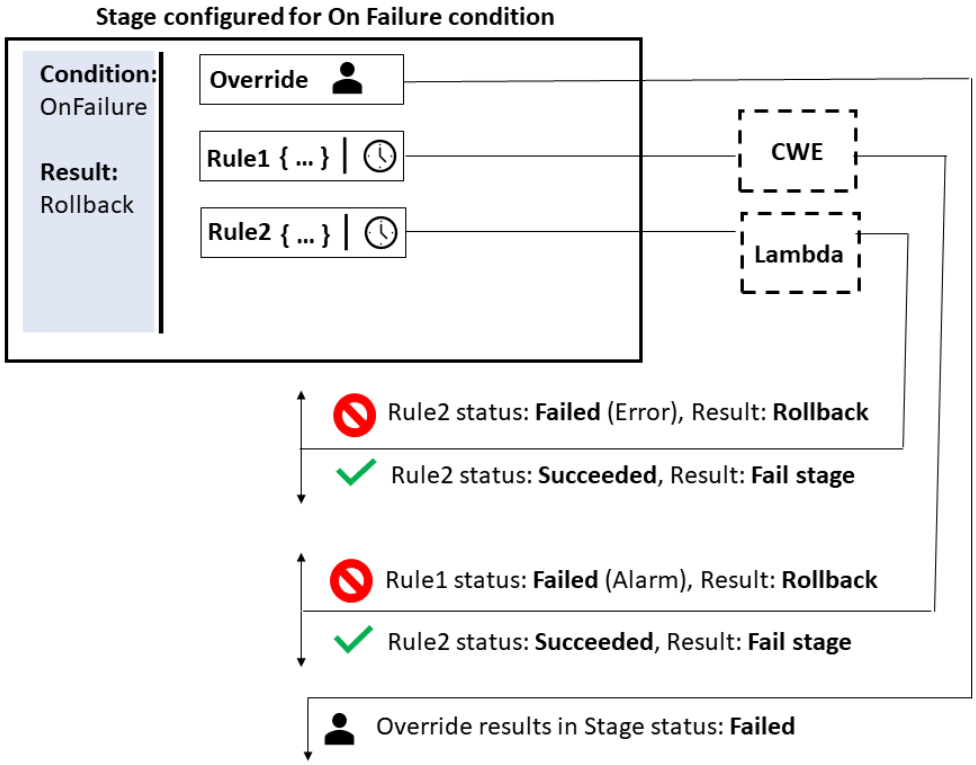
- **Entry** - The conditions for making checks that, if met, allow entry to a stage. Rules are engaged with the following result options: **Fail** or **Skip**
- **On Failure** - The conditions for making checks for the stage when it fails. Rules are engaged with the following result option: **Rollback**

- **On Success** - The conditions for making checks for the stage when it succeeds. Rules are engaged with the following result options: **Rollback** or **Fail**

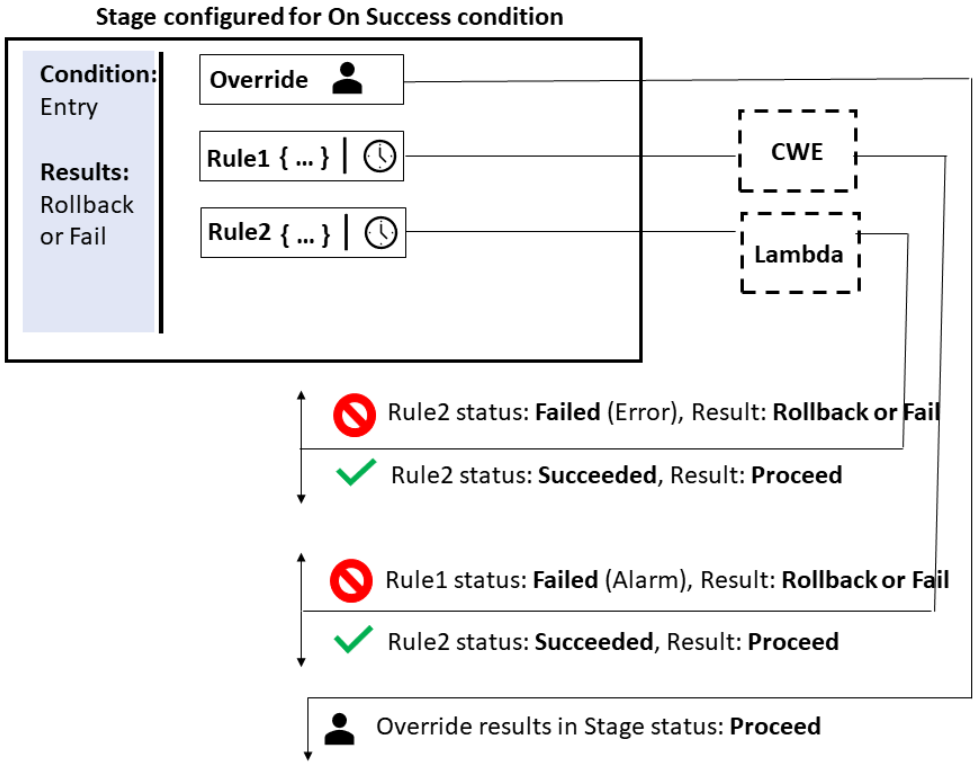
The following diagram shows an example flow for the Entry condition type in CodePipeline. Conditions answer the question, What should happen if the condition is not met, meaning any rule fails? In the following flow, an Entry condition is configured with a LambdaInvoke rule and a `CloudWatchAlarm` rule. If the rule fails, then the configured result, such as Fail, is engaged.



The following diagram shows an example flow for the On Failure condition type in CodePipeline. Conditions answer the question, What should happen if the condition is met, meaning the rules all succeed their checks? In the following flow, an On Failure condition is configured with a LambdaInvoke rule and a `CloudWatchAlarm` rule. If the rule succeeds, then the configured result, such as Fail, is engaged.



The following diagram shows an example flow for the On Success condition type in CodePipeline. Conditions answer the question, What should happen if the condition is met, meaning the rules all succeed their checks? In the following flow, an On Success condition is configured with a `LambdaInvoke` rule and a `CloudWatchAlarm` rule. If the rule succeeds, then the configured result, such as Fail, is engaged.



Rules for stage conditions

When you configure stage conditions, you select from pre-defined rules and specify the results for the rule. A condition state will be Failed if any of the rules in the condition are failed and Succeeded if all the rules succeed. How criteria are met for conditions for On Failure and On Success depends on the type of rule.

The following are managed rules that you can add to stage conditions.

- Conditions can use the **Commands** rule to specify commands to meet the rule criteria for conditions. For more information about this rule, see Commands.
- Conditions can use the **AWS DeploymentWindow** rule to specify approved deployment times for allowing a deployment. The criteria for the rule will be measured with a provided cron expression for a deployment window. The rule succeeds when the date and time in the deployment window meet the criteria in the cron expression for the rule. For more information about this rule, see DeploymentWindow.
- Conditions can use the **AWS Lambda** rule to check for error states returned from configured Lambda functions. The rule is met when the check receives the Lambda function result. An error

from the Lambda function meets the criteria for On Failure conditions. For more information about this rule, see LambdaInvoke.

- Conditions can use the **AWS CloudWatchAlarm** rule to check for alarms configured from CloudWatch events. The rule is met when the check returns an alarm state of OK, ALARM, or INSUFFICIENT_DATA. For On Success conditions, OK and INSUFFICIENT_DATA meet the criteria. ALARM meets the criteria for On Failure conditions. For more information about this rule, see CloudWatchAlarm.
- Conditions can use the **VariableCheck** rule to create a condition where the output variable is checked against a provided expression. The rule passes the check when the variable value meets the rule criteria, such as the value being equal or greater than a specified output variable. For more information about this rule, see VariableCheck.

Pipeline types

CodePipeline provides the following pipeline types, which differ in characteristics and price, so that you can tailor your pipeline features and cost to the needs of your applications.

- V1 type pipelines have a JSON structure that contains standard pipeline, stage, and action-level parameters.
- V2 type pipelines have the same structure as a V1 type, along with additional parameters for release safety and trigger configuration.

For information about pricing for CodePipeline, see Pricing.

See the CodePipeline pipeline structure reference page for details about parameters in each pipeline type. For information about which type of pipeline to choose, see What type of pipeline is right for me?.

What type of pipeline is right for me?

The pipeline type is determined by the set of characteristics and features supported by each pipeline version.

The following is a summary of the use cases and characteristics available for each type of pipeline.

	V1 type	V2 type
Characteristics		
Use cases	• Standard deployments	• Deployments with configuration from passing pipeline-level variables at runtime • Deployments where pipelines are configured to start on Git tags
Action-level variables	Supported	Supported
PARALLEL execution mode	Not supported	Supported
Pipeline-level variables	Not supported	Supported
QUEUED execution mode	Not supported	Supported
Rollback for pipeline stages	Not supported	Supported
Source revision overrides	Not supported	Supported
Stage conditions	Not supported	Supported
Triggers and filtering Git tags, pull requests, branches, or file paths	Not supported	Supported
The Commands action	Not supported	Supported
Creating Entry conditions with Skip result	Not supported	Supported
Configure a stage for automatic retry on failure	Not supported	Supported

For information about pricing for CodePipeline, see Pricing.

You can create and run a Python script to help you analyze the potential cost of moving a V1 type pipeline to a V2 type pipeline.

> **Note**
>
> The sample script below is intended for demonstration and evaluation purposes only. It is not a quote tool and does not guarantee the cost for your actual use of a V2 type pipeline, and it does not include any taxes that might apply. For information about pricing for CodePipeline, see Pricing.

To create and run a script to help you evaluate the cost of moving a V1 type pipeline to a V2 type pipeline

1. Download and install python.
2. Open a terminal window. Run the following command to create a new python script named **PipelineCostAnalyzer.py**.

```
vi PipelineCostAnalyzer.py
```

3. Copy and paste the following code into the **PipelineCostAnalyzer.py** script.

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

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

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

```
    total_action_execution_minutes = 0
    cost = 0.0
    hasNextToken = True
    nextToken = ""

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

                total_blling_action_executions += 1
                action_execution_minutes = (end_time -
 start_time).total_seconds()/60
                action_execution_cost = math.ceil(action_execution_minutes) * 0.002
                total_action_execution_minutes += action_execution_minutes
                cost = round(cost + action_execution_cost, 2)

    print ("{:<40}".format('Activity in last 30 days:'))
    print ("| {:<40} | {:<10}".format('_____________________________________',
 '____________________'))
    print ("| {:<40} | {:<10}".format('Total action executions:',
 total_action_executions))
```

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

analyze_cost_in_v2(pipeline)
```

4. From the terminal or command prompt, change directories to where you created the analyzer script.

   From that directory, run the following command, where *region* is the AWS Region where you created the V1 pipelines you want to analyze. You can also optionally evaluate a specific pipeline by providing its name:

```
python3 PipelineCostAnalyzer.py region --pipelineName
```

   For example, run the following command to run the python script named **PipelineCostAnalyzer.py**. In this example, the Region is `us-west-2`.

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

> **Note**
>
> This script will analyze all V1 pipelines in the specified AWS Region unless you specify a specific pipeline name.

5. In the following sample output from the script, we can see the list of action executions, the list of action executions that were eligible for billing, the total runtime of these action executions, and the estimated cost of these actions as performed in a V2 pipeline.

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

In this example, the negative value in the last row represents the estimated amount that might be saved by moving to V2 type pipelines.

> **Note**
>
> The script output and related examples that show costs and other information are estimates only. They are intended for demonstration and evaluation purposes only and do not guarantee any actual savings. For information about pricing for CodePipeline, see Pricing.

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.

Before you can use AWS CodePipeline for the first time, you must create your AWS account and create your first administrative user.

Topics

- Sign up for an AWS account
- Apply a managed policy for administrative access to CodePipeline
- Install the AWS CLI
- Open the console for CodePipeline
- Next steps

Sign up for an AWS account

To get started with AWS, you need an AWS account. For information about creating an AWS account, see Getting started with an AWS account in the *AWS Account Management Reference Guide*.

Apply a managed policy for administrative access to CodePipeline

You must grant permissions to interact with CodePipeline. The quickest way to do this is to apply the `AWSCodePipeline_FullAccess` managed policy to the administrative user.

Note

The `AWSCodePipeline_FullAccess` 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.

Note

The `AWSCodePipeline_FullAccess` 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 set of CodePipeline actions and resources, see Remove permissions from the CodePipeline service role.

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

Create a permission set. Follow the instructions in Create a permission set in the *AWS IAM Identity Center User Guide*.

- Users managed in IAM through an identity provider:

  Create a role for identity federation. Follow the instructions in Create a role for a third-party identity provider (federation) in the *IAM User Guide*.

- IAM users:
  - Create a role that your user can assume. Follow the instructions in Create a role for an IAM user in the *IAM User Guide*.
  - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in Adding permissions to a user (console) in the *IAM User Guide*.

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.

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.

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.

The following related resources can help you as you work with this service.

Topics

- Integrations with CodePipeline action types
- General integrations with CodePipeline
- Examples from the community

Integrations with CodePipeline action types

The integrations information in this topic is organized by CodePipeline action type.

Topics

- Source action integrations
- Build action integrations
- Test action integrations
- Deploy action integrations
- Approval action integration with Amazon Simple Notification Service
- Invoke action integrations

Source action integrations

The following information is organized by CodePipeline action type and can help you configure CodePipeline to integrate with the following source action providers.

Topics

- Amazon ECR source actions
- Amazon S3 source actions

- Connections to Bitbucket Cloud, GitHub (via GitHub App), GitHub Enterprise Server, GitLab.com, and GitLab self-managed
- CodeCommit source actions
- GitHub (via OAuth app) source actions

Amazon ECR source actions

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:

- To view configuration parameters and an example JSON/YAML snippet, see Amazon ECR source action reference
- Create a pipeline, stages, and actions
- Tutorial: Create a pipeline with an Amazon ECR source and ECS-to-CodeDeploy deployment

Amazon S3 source actions

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 action for your code.

> **Note**
>
> Amazon S3 can also be included in a pipeline as a deploy action.

Learn more:

- To view configuration parameters and an example JSON/YAML snippet, see Amazon S3 source action reference
- Step 1: Create an S3 source bucket for your application
- Create a pipeline (CLI)
- CodePipeline uses Amazon EventBridge (previously Amazon CloudWatch Events) to detect changes in your Amazon S3 source bucket. See General integrations with CodePipeline.

Connections to Bitbucket Cloud, GitHub (via GitHub App), GitHub Enterprise Server, GitLab.com, and GitLab self-managed

Connections (`CodeStarSourceConnection` actions) are used to access your third-party Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, or GitLab self-managed repository.

Bitbucket Cloud

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

> **Note**
>
> You can create connections to a Bitbucket Cloud repository. Installed Bitbucket provider types, such as Bitbucket Server, are not supported .

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

For Bitbucket Cloud, use the **Bitbucket** option in the console or the `CodestarSourceConnection` action in the CLI. See Bitbucket Cloud connections.

You can use the **Full clone** option for this action to reference the repositor y Git metadata so that downstream actions can perform Git commands directly. This option can only be used by CodeBuild downstream actions.

Learn more:

- To view configuration parameters and an example JSON/YAML snippet, see CodeStarSourceConnection for Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed actions.

- To view a Getting Started tutorial that creates a pipeline with a Bitbucket Cloud source, see Getting started with connections.

GitHub or GitHub Enterprise Cloud

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

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

Use the **GitHub (via GitHub App)** provider option in the console or the `CodestarSourceConnection` action in the CLI. See GitHub connections.

You can use the **Full clone** option for this action to reference the repository Git metadata so that downstream actions can perform Git commands directly. This option can only be used by CodeBuild downstream actions.

Learn more:

- To view configuration parameters and an example JSON/YAML snippet, see CodeStarSourceConnection for Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed actions
- For a tutorial that shows you how to connect to a GitHub repository and use the **Full clone** option, see Tutorial: Use full clone with a GitHub pipeline source.
- The current GitHub (via GitHub App) action is the version 2 source action for GitHub. The GitHub (via OAuth app) action is the version 1 GitHub action managed with OAuth token authentication. While we don't recommend using the GitHub (via OAuth app) action, existing pipelines with the GitHub (via OAuth app) action will continue to work without any impact. You can now use a CodeStarSourceConnection for Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-

	managed actions source action in your pipeline that manages your GitHub source action with GitHub apps. If you have a pipeline that uses the GitHub (via OAuth app) action, see the steps to update it to use a GitHub (via GitHub App) action in Update a GitHub (via OAuth app) source action to a GitHub (via GitHub App) source action.
GitHub Enterprise Server	You can configure CodePipeline to use a GitHub Enterprise Server repository as the source for your code. You must have previously created a GitHub account and at least one GitHub repository. You can add a source action for your GitHub Enterprise Server repository by either creating a pipeline or editing an existing one. You can set up resources called *connections* to allow your pipelines to access third-party code repositories. When you create a connection, you install the connector app with your third-party code repository, and then associate it with your connection. Use the **GitHub Enterprise Server** provider option in the console or the `CodestarSourceConnection` action in the CLI. See GitHub Enterprise Server connections. You can use the **Full clone** option for this action to reference the repository Git metadata so that downstream actions can perform Git commands directly. This option can only be used by CodeBuild downstream actions. Learn more: • To view configuration parameters and an example JSON/YAML snippet, see CodeStarSourceConnection for Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed actions • For a tutorial that shows you how to connect to a GitHub repository and use the **Full clone** option, see Tutorial: Use full clone with a GitHub pipeline source.

GitLab.com	You can configure CodePipeline to use a GitLab.com repository as the source for your code. You must have previously created a GitLab.com account and at least one GitLab.com repository. You can add a source action for your GitLab.com repository by either creating a pipeline or editing an existing one. Use the **GitLab** provider option in the console or the `CodestarSourceConnection` action with the `GitLab` provider in the CLI. See GitLab.com connections. Learn more: • To view configuration parameters and an example JSON/YAML snippet, see CodeStarSourceConnection for Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed actions
GitLab self-managed	You can configure CodePipeline to use a GitLab self-managed installation as the source for your code. You must have previously created a GitLab account and have a subscription for self-managed GitLab (Enterprise Edition or Community Edition). You can add a source action for your GitLab self-managed repository by either creating a pipeline or editing an existing one. You can set up resources called *connections* to allow your pipelines to access third-party code repositories. When you create a connection, you install the connector app with your third-party code repository, and then associate it with your connection. Use the **GitLab self-managed** provider option in the console or the `CodestarSourceConnection` action in the CLI. See Connections for GitLab self-managed. You can use the **Full clone** option for this action to reference the repository Git metadata so that downstream actions can perform Git commands directly. This option can only be used by CodeBuild downstream actions. Learn more:

- To view configuration parameters and an example JSON/YAML snippet, see CodeStarSourceConnection for Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed actions
- For the steps to create a connection with this provider type, see Connections for GitLab self-managed.

CodeCommit source actions

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 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.

You can use the **Full clone** option for this action to reference the repository Git metadata so that downstream actions can perform Git commands directly. This option can only be used by CodeBuild downstream actions.

Learn more:

- To view configuration parameters and an example JSON/YAML snippet, see CodeCommit source action reference.
- Tutorial: Create a simple pipeline (CodeCommit repository)
- 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.

GitHub (via OAuth app) source actions

The GitHub (via OAuth app) action is the version 1 GitHub action managed with OAuth Apps. In available Regions, you can also use a CodeStarSourceConnection for Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed actions source action in your pipeline that manages your GitHub source action with GitHub Apps. If you have a pipeline that uses the GitHub (via OAuth app) action, see the steps to update it to use a GitHub (via GitHub

app) action in Update a GitHub (via OAuth app) source action to a GitHub (via GitHub App) source action.

> **Note**
>
> While we don't recommend using the GitHub (via OAuth app) action, existing pipelines with the GitHub (via OAuth app) action will continue to work without any impact.

Learn more:

- For more information about OAuth-based GitHub (via OAuth app) access in contrast to app-based GitHub access, see https://docs.github.com/en/developers/apps/differences-between-github-apps-and-oauth-apps.
- To view an appendix that contains the GitHub (via OAuth app) action details, see Appendix A: GitHub (via OAuth app) source actions.

Build action integrations

The following information is organized by CodePipeline action type and can help you configure CodePipeline to integrate with the following build action providers.

Topics

- CodeBuild build actions
- CloudBees build actions
- Amazon ECR build and publish actions
- Jenkins build actions
- TeamCity build actions

CodeBuild build actions

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. For more information, see the CodePipeline Action Configuration Reference for AWS CodeBuild build and test action reference.

> **Note**
>
> CodeBuild can also be included in a pipeline as a test action, with or without a build output.

Learn more:

- To view configuration parameters and an example JSON/YAML snippet, see AWS CodeBuild build and test action reference.
- What Is CodeBuild?
- CodeBuild – Fully Managed Build Service

CloudBees build actions

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

Learn more:

- re:INVENT 2017: Cloud First with AWS

Amazon ECR build and publish actions

Amazon ECR is an AWS Docker image repository service. You use Docker push and pull commands to upload Docker images to your repository.

You can add the `ECRBuildAndPublish` action to your pipeline to automate building and pushing an image. For more information, see the CodePipeline Action configuration reference for `ECRBuildAndPublish` build action reference.

Jenkins build actions

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.

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 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

TeamCity build actions

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

Test action integrations

The following information is organized by CodePipeline action type and can help you configure CodePipeline to integrate with the following test action providers.

Topics

- CodeBuild test actions
- AWS Device Farm test actions
- Ghost Inspector test actions
- OpenText LoadRunner Cloud test actions
- Reflect test automation

CodeBuild test actions

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 build and test action reference.

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

Learn more:

- To view configuration parameters and an example JSON/YAML snippet, see AWS CodeBuild build and test action reference.
- What Is CodeBuild?

AWS Device Farm test actions

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. It supports remote access of Android apps, which allows you to interact directly with test devices.

Learn more:

- To view configuration parameters and an example JSON/YAML snippet, see AWS Device Farm test action reference.
- What Is AWS Device Farm?
- Using AWS Device Farm in a CodePipeline Test Stage

Ghost Inspector test actions

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

OpenText LoadRunner Cloud test actions

You can configure CodePipeline to use OpenText LoadRunner Cloud in one or more actions in a pipeline.

Learn more:

- LoadRunner Cloud documentation for integrating with CodePipeline

Reflect test automation

Reflect is the AI-powered test automation solution that empowers you to simplify testing and overcome the challenges of manual processes. With no-code test automation, Reflect streamlines test creation, execution, and maintenance, enabling you to create robust, repeatable tests without requiring any technical knowledge. By eliminating complexity and ensuring minimal disruption to your workflows, it helps you accelerate testing and confidently deliver high-quality applications every time.

Learn more:

- AWS CodePipeline test integration with Reflect

Deploy action integrations

The following information is organized by CodePipeline action type and can help you configure CodePipeline to integrate with the following deploy action providers.

Topics

- Amazon EC2 deploy actions
- Amazon Elastic Kubernetes Service EKS deploy actions
- Amazon S3 deploy actions
- AWS AppConfig deploy actions
- CloudFormation deploy actions

- CloudFormation StackSets deploy actions
- Amazon ECS deploy actions
- Elastic Beanstalk deploy actions
- OpsWorks deploy actions
- Service Catalog deploy actions
- Amazon Alexa deploy actions
- CodeDeploy deploy actions
- XebiaLabs deploy actions

Amazon EC2 deploy actions

Amazon EC2 allows you to create and manage compute in the cloud. You can add an action to a pipeline that uses Amazon EC2 as a deployment provider that deploys your application to your instances.

Learn more:

- See the action reference page at Amazon EC2 action reference.
- For a tutorial, see Tutorial: Deploy to Amazon EC2 instances with CodePipeline.

Amazon Elastic Kubernetes Service EKS deploy actions

Amazon EKS allows you to create and manage Kubernetes clusters. You can add an action to a pipeline that uses Amazon EKS as a deployment provider that deploys your image to your cluster. You can use Helm templates or Kubernetes manifest files.

Learn more:

- See the action reference page at Amazon Elastic Kubernetes Service EKS deploy action reference.
- For a tutorial, see Tutorial: Deploy to Amazon EKS with CodePipeline.

Amazon S3 deploy actions

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, stages, and actions
- Tutorial: Create a pipeline that uses Amazon S3 as a deployment provider

AWS AppConfig deploy actions

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 deploy action reference
- Tutorial: Create a pipeline that uses AWS AppConfig as a deployment provider

CloudFormation deploy actions

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 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 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 CloudFormation stacks and change sets as part of a pipeline execution. 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 CloudFormation templates and parameter definitions.

Learn more:

- CodePipeline Action Configuration Reference for CloudFormation deploy action reference
- Continuous Delivery with CodePipeline — Learn how to use CodePipeline to build a continuous delivery workflow for CloudFormation.
- Automating Deployment of Lambda-based Applications — Learn how to use the AWS Serverless Application Model and CloudFormation to build a continuous delivery workflow for your Lambda-based application.

CloudFormation StackSets deploy actions

CloudFormation gives you a way to deploy resources across multiple accounts and AWS Regions.

You can use CodePipeline with CloudFormation to update your stack set definition and deploy updates to your instances.

You can add the following actions to a pipeline to use AWS CloudFormation StackSets as a deployment provider.

- CloudFormationStackSet
- CloudFormationStackInstances

Learn more:

- CodePipeline Action Configuration Reference for CloudFormation StackSets deploy action reference
- Tutorial: Create a pipeline with AWS CloudFormation StackSets deployment actions

Amazon ECS deploy actions

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.

Learn more:

- What Is Amazon ECS?
- Tutorial: Continuous Deployment with CodePipeline
- Create a pipeline, stages, and actions
- Tutorial: Create a pipeline with an Amazon ECR source and ECS-to-CodeDeploy deployment

Elastic Beanstalk deploy actions

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.

Learn more:

- Getting started using Elastic Beanstalk
- Create a pipeline, stages, and actions

OpsWorks deploy actions

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 OpsWorks.

- **Custom Chef Cookbooks** – OpsWorks uses Chef Cookbooks to handle tasks such as installing and configuring packages and deploying applications.
- **Applications** – An 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 OpsWorks stack and layer. You can create the 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 OpsWorks is currently available in the US East (N. Virginia) Region (eusc-de-east-1) only.

Learn more:

- Using CodePipeline with AWS OpsWorks Stacks
- Cookbooks and Recipes
- AWS OpsWorks Apps

Service Catalog deploy actions

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 Service Catalog. You can create the 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 Service Catalog
- Create a pipeline, stages, and actions

Amazon Alexa deploy actions

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

> **Note**
>
> This feature is not available in the Asia Pacific (Hong Kong) or Europe (Milan) Region. To use other deploy actions available in that Region, see Deploy action integrations.

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

CodeDeploy deploy actions

CodeDeploy coordinates application deployments to Amazon EC2/on-premises instances, Amazon Elastic Container Service compute platforms, and serverless AWS Lambda compute platforms. 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.

Learn more:

- Step 3: Create an application in CodeDeploy
- Tutorial: Create a simple pipeline (CodeCommit repository)

XebiaLabs deploy actions

You can configure CodePipeline to use XebiaLabs to deploy your code in one or more actions in a pipeline.

Learn more:

- Using XL Deploy with CodePipeline

Approval action integration with 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

Invoke action integrations

The following information is organized by CodePipeline action type and can help you configure CodePipeline to integrate with the following invoke action providers.

Topics

- Amazon Inspector invoke actions
- Lambda invoke actions
- Step Functions invoke actions

Amazon Inspector invoke actions

Amazon Inspector is a vulnerability management service that automatically discovers workloads and continually scans them for software vulnerabilities and unintended network exposure. Amazon Inspector supports multiple archive formats including tar and war, and Amazon Inspector supports binaries including Rust and Go binaries.

You can configure the CodePipeline `InspectorScan` action to automate scanning your source code or Amazon ECR image repository for vulnerabilities.

Learn more:

- CodePipeline action configuration reference for Amazon Inspector `InspectorScan` invoke action reference

Lambda invoke actions

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 **Create Pipeline** wizard.

Learn more:

- CodePipeline Action configuration reference for AWS Lambda invoke action reference
- Invoke an AWS Lambda function in a pipeline in CodePipeline

Step Functions invoke actions

Step Functions lets you create and configure state machines. You can configure CodePipeline to use Step Functions invoke actions to trigger state machine executions.

Learn more:

- CodePipeline Action Configuration Reference for AWS Step Functions invoke action reference
- Tutorial: Use an AWS Step Functions invoke action in a pipeline

General integrations with CodePipeline

The following AWS service integrations are not based on CodePipeline action types.

Amazon CloudWatch	Amazon CloudWatch monitors your AWS resources. Learn more: • What Is Amazon CloudWatch?
Amazon EventBridge	Amazon EventBridge 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. • **Start a pipeline execution automatically when something changes** — You can configure CodePipeline as a target in rules set up in Amazon EventBridge. This sets up pipelines to start automatic ally when another service changes. Learn more:

	• What Is Amazon EventBridge? • Start a pipeline in CodePipeline. • CodeCommit source actions and EventBridge • **Receive notifications when a pipeline state changes** — You can set up EventBridge rules to detect and react to changes in execution state for a pipeline, stage, or action. Learn more: • Monitoring CodePipeline events • Tutorial: Set up a CloudWatch Events rule to receive email notifications for pipeline state changes
AWS Cloud9	AWS Cloud9 is an online IDE, which you access through your web browser. The IDE offers a rich code editing experience with support for several programming languages and runtime debuggers, as well as a built-in terminal. In the background, an Amazon EC2 instance hosts an AWS Cloud9 development environment. For more information, see the *AWS Cloud9 User Guide*. Learn more: • Setting up AWS Cloud9
AWS CloudTrail	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. Learn more: • Logging CodePipeline API calls with AWS CloudTrail
AWS CodeStar Notifications	You can set up *notifications* to make users aware of important changes, such as when a pipeline starts execution. For more informati on, see Create a notification rule.

AWS Key Management Service	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. Learn more: • 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. • To create a pipeline from one AWS account that deploys an 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. 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 KMS key. For cross-account actions, you can only use the key ID or key ARN to identify the key. Cross-account actions involve using the role from the other account (AccountB), so

specifying the key ID will use the key from the other account (AccountB).

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

Integration examples: Blog posts

- Tracking the AWS CodePipeline build status from the third-party Git repository

  Learn how to set up resources that will display your pipeline and build action status in your third-party repository, making it easy for the developer to track status without switching context.

  Published March 2021

- Complete CI/CD with AWS CodeCommit, AWS CodeBuild, AWS CodeDeploy, and AWS CodePipeline

  Learn how to set up a pipeline that uses the CodeCommit, CodePipeline, CodeBuild, and CodeDeploy services to compile, build, and install a version-controlled Java application onto a set of Amazon EC2 Linux instances.

  Published September 2020

- Testing and creating CI/CD pipelines for AWS Step Functions

Learn how to set up resources that will coordinate your Step Functions state machine and your pipeline.

Published March 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 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 CloudFormation.

  - Download a sample 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

- 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 CloudFormation template. This walkthrough also incorporates CodeCommit and AWS Identity and Access Management (IAM).

  Published May 2016

- Automate CodeCommit and CodePipeline in CloudFormation

  Use 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 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.

  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.

  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

- Deploying from CodePipeline to 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

CodePipeline tutorials

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

Topics

- Tutorial: Deploy to Amazon EC2 instances with CodePipeline
- Tutorial: Build and push a Docker image to Amazon ECR with CodePipeline (V2 type)
- Tutorial: Deploy to Amazon EKS with CodePipeline
- Tutorial: Create a pipeline that runs commands with compute (V2 type)
- Tutorial: Use Git tags to start your pipeline
- Tutorial: Filter on branch names for pull requests to start your pipeline (V2 type)
- Tutorial: Use pipeline-level variables
- Tutorial: Create a simple pipeline (S3 bucket)
- Tutorial: Create a simple pipeline (CodeCommit repository)
- Tutorial: Create a four-stage pipeline
- Tutorial: Set up a CloudWatch Events rule to receive email notifications for pipeline state changes
- Tutorial: Create a pipeline that builds and tests your Android app with AWS Device Farm
- Tutorial: Create a pipeline that tests your iOS app with AWS Device Farm
- Tutorial: Create a pipeline that deploys to Service Catalog
- Tutorial: Create a pipeline with AWS CloudFormation
- Tutorial: Create a pipeline that uses variables from AWS CloudFormation deployment actions
- Tutorial: Amazon ECS Standard Deployment with CodePipeline
- Tutorial: Create a pipeline with an Amazon ECR source and ECS-to-CodeDeploy deployment
- Tutorial: Create a pipeline that deploys an Amazon Alexa skill
- Tutorial: Create a pipeline that uses Amazon S3 as a deployment provider
- Tutorial: Create a pipeline that publishes your serverless application to the AWS Serverless Application Repository
- Tutorial: Lambda function deployments with CodePipeline

- Tutorial: Using variables with Lambda invoke actions
- Tutorial: Use an AWS Step Functions invoke action in a pipeline
- Tutorial: Create a pipeline that uses AWS AppConfig as a deployment provider
- Tutorial: Use full clone with a GitHub pipeline source
- Tutorial: Use full clone with a CodeCommit pipeline source
- Tutorial: Create a pipeline with AWS CloudFormation StackSets deployment actions
- Tutorial: Create a variable check rule for a pipeline as an entry condition

Tutorial: Deploy to Amazon EC2 instances with CodePipeline

This tutorial helps you to create a deploy action in CodePipeline that deploys your code to instances you have configured in Amazon EC2.

> **Note**
>
> As part of creating a pipeline in the console, an S3 artifact bucket will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

> **Note**
>
> The EC2 deploy action is only available for V2 type pipelines.

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.

- A source control repository (this tutorial uses GitHub) where you will add a sample `script.sh` file.
- You must use an existing CodePipeline service role that has been updated with the permissions for this action. To update your service role, see Service role policy permissions for the EC2 deploy action.

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

Step 1: Create Amazon EC2 Linux instances

In this step, you create the Amazon EC2 instances where you will deploy a sample application. As part of this process, create an instance role in IAM, if you have not already created an instance role in the Region where you want to create resources.

To create an instance role

1. Open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/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**. Under **Select your use case**, choose **EC2**. Choose **Next**.
5. Search for and select the policy named **`AmazonSSMManagedEC2InstanceDefaultPolicy`**.
6. Search for and select the policy named **`AmazonSSMManagedInstanceCore`**. Choose **Next: Tags**.
7. 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.

> **Note**
>
> You will add permissions to this role to allow access to the S3 artifact bucket for your pipeline after pipeline creation.

Choose **Create role**.

To launch instances

1. Open the Amazon EC2 console at https://eusc-de-east-1.console.amazonaws-eusc.eu/ec2/.
2. From the side navigation, choose **Instances**, and select **Launch instances** from the top of the page.
3. In **Name**, enter **`MyInstances`**. This assigns the instance a tag **Key** of **`Name`** and a tag **Value** of **`MyInstances`**.
4. Under **Application and OS Images (Amazon Machine Image)**, locate the **Amazon Linux** AMI option with the AWS logo, and make sure it is selected. (This AMI is described as the Amazon Linux 2 AMI (HVM) and is labeled "Free tier eligible".)
5. Under **Instance type**, choose the free tier eligible `t2.micro` type as the hardware configuration for your instance.
6. Under **Key pair (login)**, choose a key pair or create one.
7. Under **Network settings**, make sure the status is **Enable**.
8. Expand **Advanced details**. In **IAM instance profile**, choose the IAM role you created in the previous procedure (for example, **`EC2InstanceRole`**).

   > **Note**
   >
   > Do not leave the instance role blank as this creates a default role and does not select the role you created.

9. Under **Summary**, under **Number of instances**, enter 2.
10. Choose **Launch instance**.
11. 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**.)

Step 2: Add artifact bucket permissions to the EC2 instance role

You must update the EC2 instance role you created for your instance to allow it access to your pipeline's artifact bucket.

> **Note**
>
> When you create the instance, you create or use an existing EC2 instance role. To avoid `Access Denied` errors, you must add S3 bucket permissions to the instance role to give the instance permissions to the CodePipeline artifact bucket. Create a default role or update your existing role with the `s3:GetObject` permission scoped down to the artifact bucket for your pipeline's Region.

1. Navigate to your pipeline in the CodePipeline console. Choose **Settings**. View the name and location of the artifact store for an existing pipeline. Make a note of the artifact bucket Amazon Resource Name (ARN) and copy it.
2. Navigate to the IAM console and choose **Roles**. Choose the instance role you created in Step 1 of this tutorial.
3. On the **Permissions** tab, choose **Add inline policy**.
4. Add the following JSON to the policy document, replacing the value in the `Resource` field with the bucket ARN.

```
{
    "Effect": "Allow",
    "Principal": "*",
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::BucketName"
}
```

5. Choose **Update**.

Step 3: Add a script file to your repository

Paste this sample text to create your `script.sh` file for the post-script step in the deployment.

```
echo "Hello World!"
```

To add a `script.sh` file to your source repository

1. Open a text editor and then copy and paste the file above into a new file.
2. Commit and push your `script.sh` file to your source repository.

   a. Add the file.

   ```
   git add .
   ```

   b. Commit the change.

   ```
   git commit -m "Adding script.sh."
   ```

   c. Push the commit.

   ```
   git push
   ```

   Make a note of the path in your repository.

   ```
   /MyDemoRepo/test/script.sh
   ```

Step 4: Creating your pipeline

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

To create your pipeline

1. Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
2. On the **Welcome** page, **Getting started** page, or the **Pipelines** page, choose **Create pipeline**.

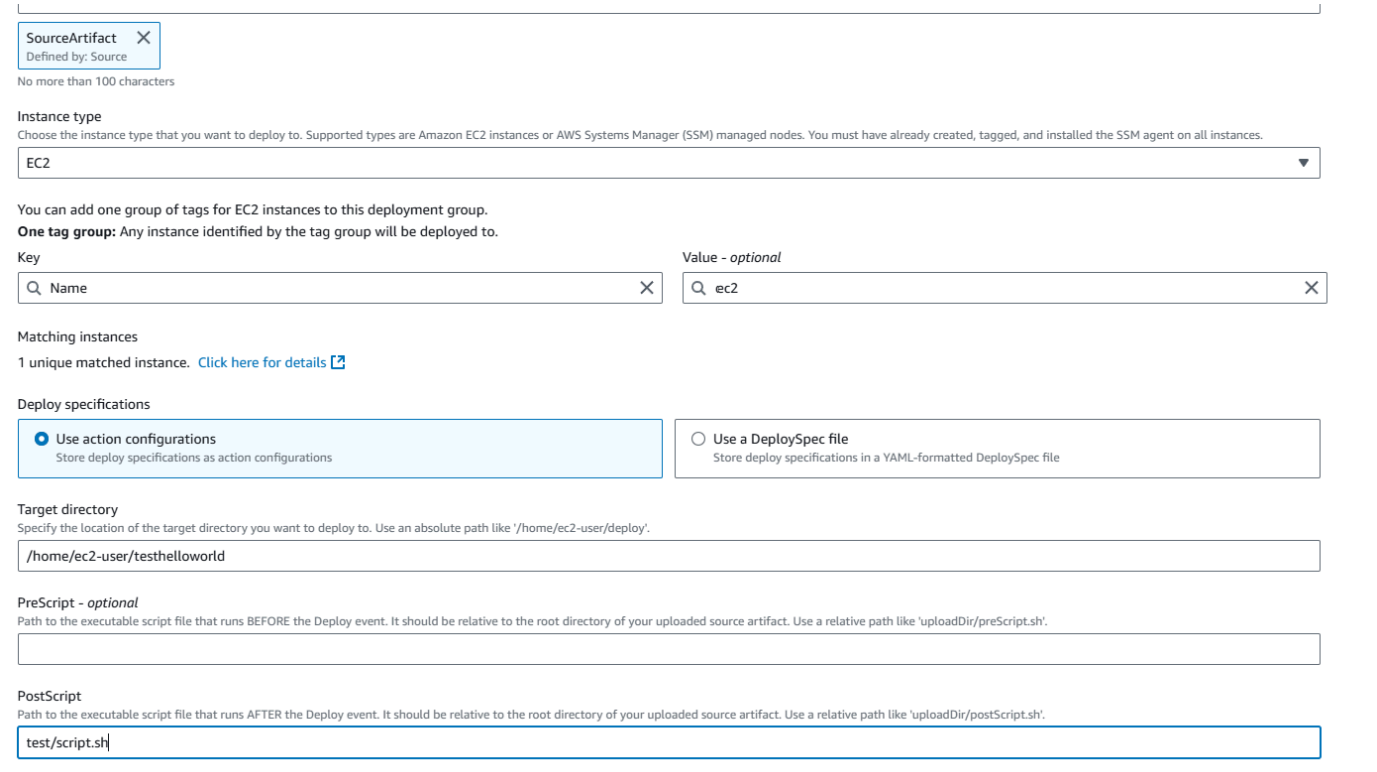
3. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. In **Step 2: Choose pipeline settings**, in **Pipeline name**, enter **`MyPipeline`**.
5. CodePipeline provides V1 and V2 type pipelines, which differ in characteristics and price. The V2 type is the only type you can choose in the console. For more information, see pipeline types. For information about pricing for CodePipeline, see Pricing.
6. In **Service role**, choose **Use existing service role**, and then choose the CodePipeline service role that has been updated with the required permissions for this action. To configure your CodePipeline service role for this action, see Service role policy permissions for the EC2 deploy action.
7. Leave the settings under **Advanced settings** at their defaults, and then choose **Next**.
8. On the **Step 3: Add source stage** page, add a source stage:

   a. In **Source provider**, choose **GitHub (via GitHub App)**.

   b. Under **Connection**, choose an existing connection or create a new one. To create or manage a connection for your GitHub source action, see GitHub connections.

   c. In **Repository name**, choose the name of your GitHub repository.

   Choose **Next**.
9. On the **Step 4: Add build stage** page, choose **Skip**.
10. On the **Step 5: Add deploy stage** page, choose **EC2**.



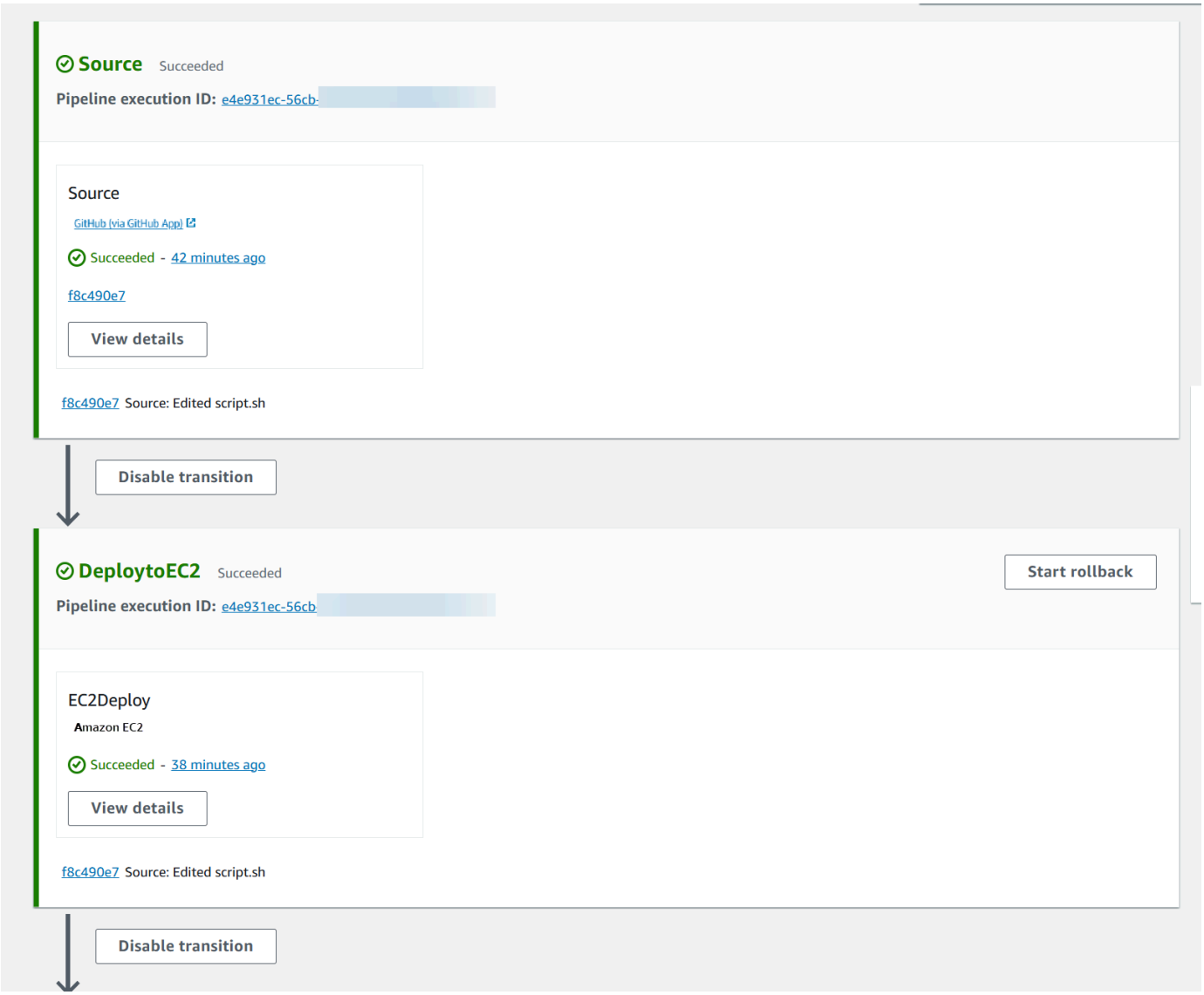
a. For **Target directory**, enter the directory on the instance that you want to deploy to, such as `/home/ec2-user/testhelloworld`.

> **Note**
>
> Specify the deployment directory that you want the action to use on the instance. The action will automate creating the specified directory on the instance as part of the deployment.

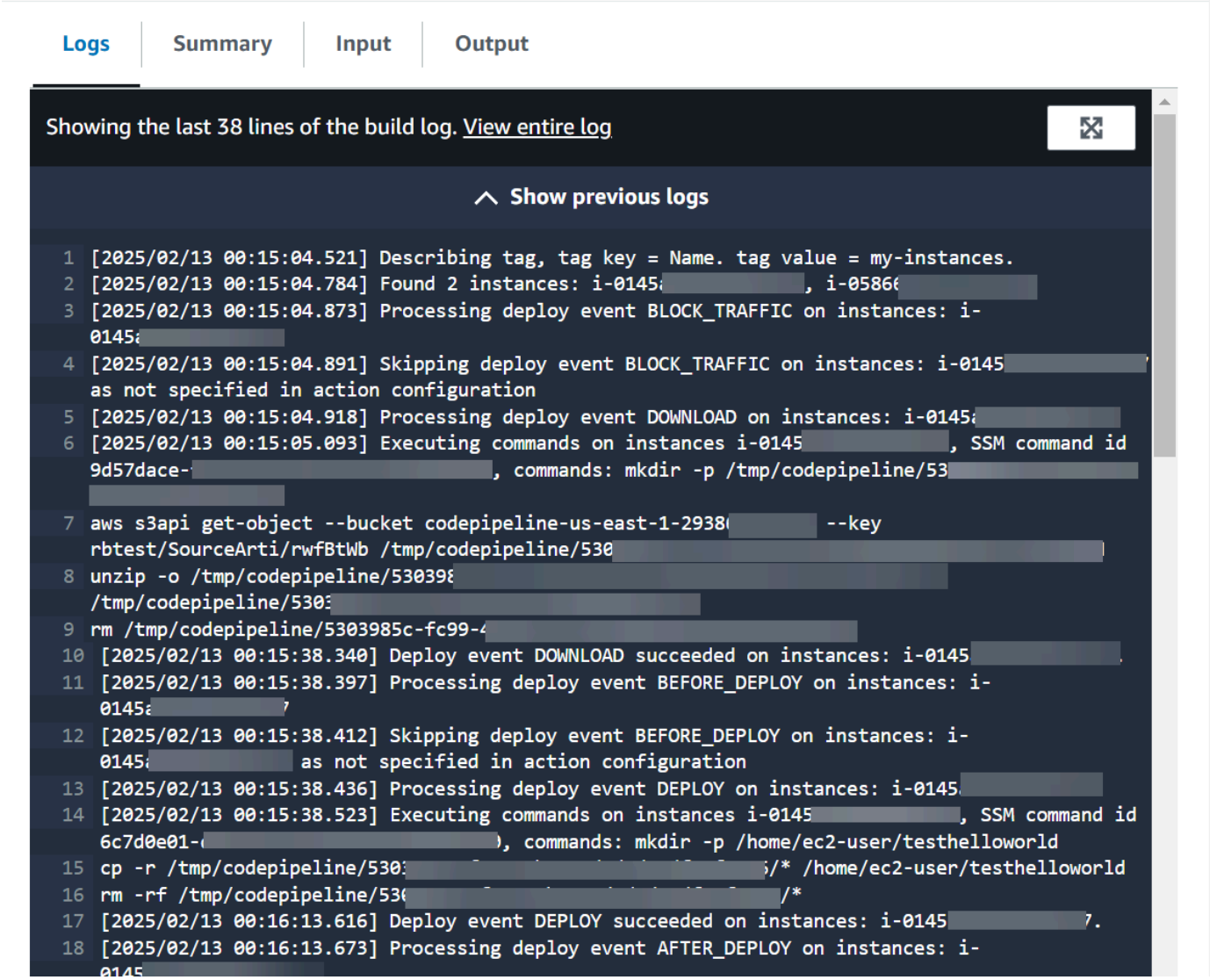
b. For **PostScript**, enter the path and file name for your script, such as `test/script.sh`.

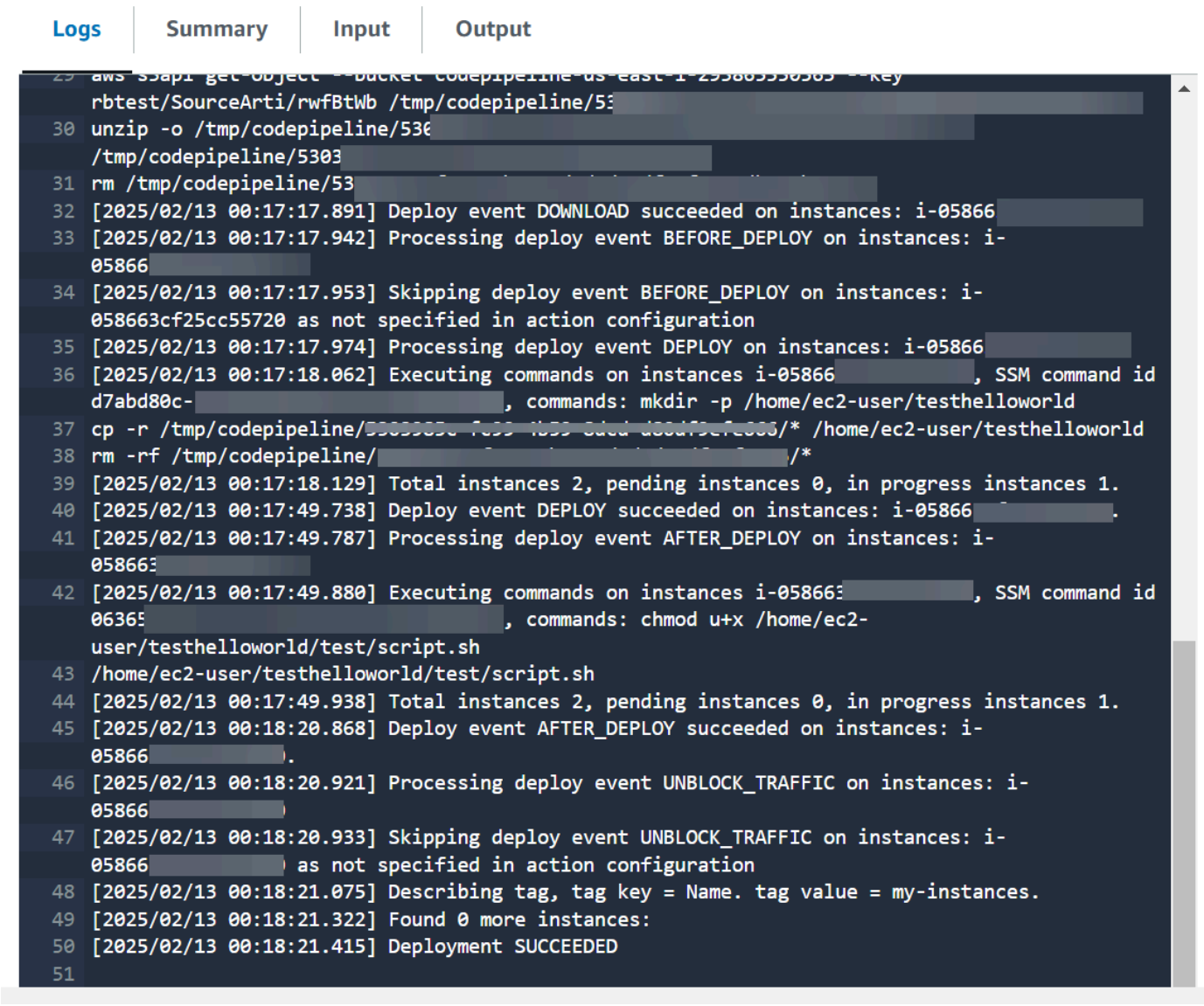
c. Choose **Next**.

11. On the **Step 6: Review** page, review your pipeline configuration and choose **Create pipeline** to create the pipeline.



12. After the pipeline runs successfully, choose **View details** to view the logs on the action to view the managed compute action output.


Logs
Summary
Input
Output
Showing the last 38 lines of the build log. View entire log
Show previous logs
1 [2025/02/13 00:15:04.521] Describing tag, tag key = Name. tag value = my-instances.
2 [2025/02/13 00:15:04.784] Found 2 instances: i-0145 , i-0586
3 [2025/02/13 00:15:04.873] Processing deploy event BLOCK_TRAFFIC on instances: i-
0145
4 [2025/02/13 00:15:04.891] Skipping deploy event BLOCK_TRAFFIC on instances: i-0145
as not specified in action configuration
5 [2025/02/13 00:15:04.918] Processing deploy event DOWNLOAD on instances: i-0145
6 [2025/02/13 00:15:05.093] Executing commands on instances i-0145 , SSM command id
9d57dace- , commands: mkdir -p /tmp/codepipeline/53
7 aws s3api get-object --bucket codepipeline-us-east-1-2938 --key
rbtest/SourceArti/rwfBtWb /tmp/codepipeline/530
8 unzip -o /tmp/codepipeline/530398
/tmp/codepipeline/530
9 rm /tmp/codepipeline/5303985c-fc99-4
10 [2025/02/13 00:15:38.340] Deploy event DOWNLOAD succeeded on instances: i-0145
11 [2025/02/13 00:15:38.397] Processing deploy event BEFORE_DEPLOY on instances: i-
0145
12 [2025/02/13 00:15:38.412] Skipping deploy event BEFORE_DEPLOY on instances: i-
0145 as not specified in action configuration
13 [2025/02/13 00:15:38.436] Processing deploy event DEPLOY on instances: i-0145
14 [2025/02/13 00:15:38.523] Executing commands on instances i-0145 , SSM command id
6c7d0e01- , commands: mkdir -p /home/ec2-user/testhelloworld
15 cp -r /tmp/codepipeline/530 /* /home/ec2-user/testhelloworld
16 rm -rf /tmp/codepipeline/53 /*
17 [2025/02/13 00:16:13.616] Deploy event DEPLOY succeeded on instances: i-0145 .
18 [2025/02/13 00:16:13.673] Processing deploy event AFTER_DEPLOY on instances: i-
0145



Step 5: 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.
2. Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
3. Choose your pipeline from the list.

4. Watch the pipeline progress through its stages. Your pipeline should complete and your action deploys the script on your instances.
5. For more troubleshooting information, see [EC2 Deploy action fails with an error message No such file](#).

Tutorial: Build and push a Docker image to Amazon ECR with CodePipeline (V2 type)

This tutorial helps you to create a build action in CodePipeline that runs and pushes your Docker image to Amazon ECR after a change to your source code. This tutorial also shows you how to add an Amazon ECS deploy action that deploys your pushed image.

Important

As part of creating a pipeline in the console, an S3 artifact bucket will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

Note

This tutorial is for the ECRBuildAndPublish build action for a CodePipeline pipeline with a GitHub source repository and an Amazon ECS standard action for deploying to an Amazon ECS cluster. For a tutorial that uses a pipeline with an ECR image repository as the source for an Amazon ECS to CodeDeploy blue/green deployment action in CodePipeline, see [Tutorial: Create a pipeline with an Amazon ECR source and ECS-to-CodeDeploy deployment](#).

Important

This action uses CodePipeline managed CodeBuild compute to run commands in a build environment. Running the commands action will incur separate charges in AWS CodeBuild.

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.

- A source control repository (this tutorial uses GitHub) where you will add the following for this tutorial:
  - In Step 1, you will add a sample Dockerfile to your source repository as the input artifact for the ECRBuildAndPublish build action in CodePipeline.
  - In Step 2, you will add a sample imagedefinitions.json file to your source repository as a requirement for the Amazon ECS standard deploy action in CodePipeline.
- An Amazon ECR image repository that contains an image you have built from your Dockerfile. For more information, see Creating a Repository and Pushing an Image in the *Amazon Elastic Container Registry User Guide*.
- An Amazon ECS cluster and service created in the same Region as the image repository. For more information, see Creating a Cluster and Creating a Service in the *Amazon Elastic Container Service Developer Guide*.

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

Step 1: Add a Dockerfile to your source repository

This tutorial uses the ECRBuildAndPublish action to build your Docker image and push the image to Amazon ECR. The managed compute action in CodePipeline uses CodeBuild to run the commands for the ECR login and image push. You do not need to add a `buildspec.yml` file to your source code repository to tell CodeBuild how to do that. You only provide the Dockerfile in your repository as follows for this example.

Paste this sample text to create your `Dockerfile` file. This sample Dockerfile is the same as the sample used in the ECR image instructions in the prerequisites.

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

# Install dependencies
RUN yum update -y && \
 yum install -y httpd

# Install apache and write hello world message
RUN echo 'Hello World!' > /var/www/html/index.html

# Configure apache
RUN echo 'mkdir -p /var/run/httpd' >> /root/run_apache.sh && \
 echo 'mkdir -p /var/lock/httpd' >> /root/run_apache.sh && \
 echo '/usr/sbin/httpd -D FOREGROUND' >> /root/run_apache.sh && \
 chmod 755 /root/run_apache.sh

EXPOSE 80

CMD /root/run_apache.sh
```

To add a `Dockerfile` file to your source repository

1. Open a text editor and then copy and paste the Dockerfile above into a new file.
2. Commit and push your `Dockerfile` file to your source repository.

   a. Add the file.

   ```
   git add .
   ```

   b. Commit the change.

   ```
   git commit -m "Adding Dockerfile."
   ```

   c. Push the commit.

   ```
   git push
   ```

   Be sure to place the file at the root level of your repository.

   ```
   / Dockerfile
   ```

Step 2: Add an imagedefinitions.json file to your source repository

This tutorial uses the Amazon ECS standard deploy action in CodePipeline to deploy your container to your Amazon ECS cluster. The Amazon ECS standard deploy action requires an imagedefinitions.json file containing your image name and URI. For more information about the imagedefinitions.json file, see imagedefinitions.json file for Amazon ECS standard deployment actions.

Paste this sample text to create your `imagedefinitions.json` file. Use the name in your Dockerfile, such as `hello-world`, and use the URI from your Amazon ECR repository where the image is stored.

```
[
  {
    "name": "hello-world",
    "imageUri": "ACCOUNT-ID.dkr.ecr.us-east-1.amazonaws.com/actions/image-repo"
  }
]
```

To add an `imagedefinitions.json` file to your source repository

1. Open a text editor and then copy and paste the example above into a new file.
2. Commit and push your `imagedefinitions.json` file to your source repository.

   a. Add the file.

   ```
   git add .
   ```

   b. Commit the change.

   ```
   git commit -m "Adding imagedefinitions.json."
   ```

   c. Push the commit.

   ```
   git push
   ```

   Be sure to place the file at the root level of your repository.

   ```
   / imagedefinitions.json
   ```

Step 3: Creating your pipeline

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

To create your pipeline

1. Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
2. On the **Welcome** page, **Getting started** page, or the **Pipelines** page, choose **Create pipeline**.
3. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. In **Step 2: Choose pipeline settings**, in **Pipeline name**, enter **`MyPipeline`**.
5. CodePipeline provides V1 and V2 type pipelines, which differ in characteristics and price. The V2 type is the only type you can choose in the console. For more information, see pipeline types. For information about pricing for CodePipeline, see Pricing.
6. In **Service role**, choose **New service role** to allow CodePipeline to create a service role in IAM.
7. Leave the settings under **Advanced settings** at their defaults, and then choose **Next**.
8. On the **Step 3: Add source stage** page, add a source stage:

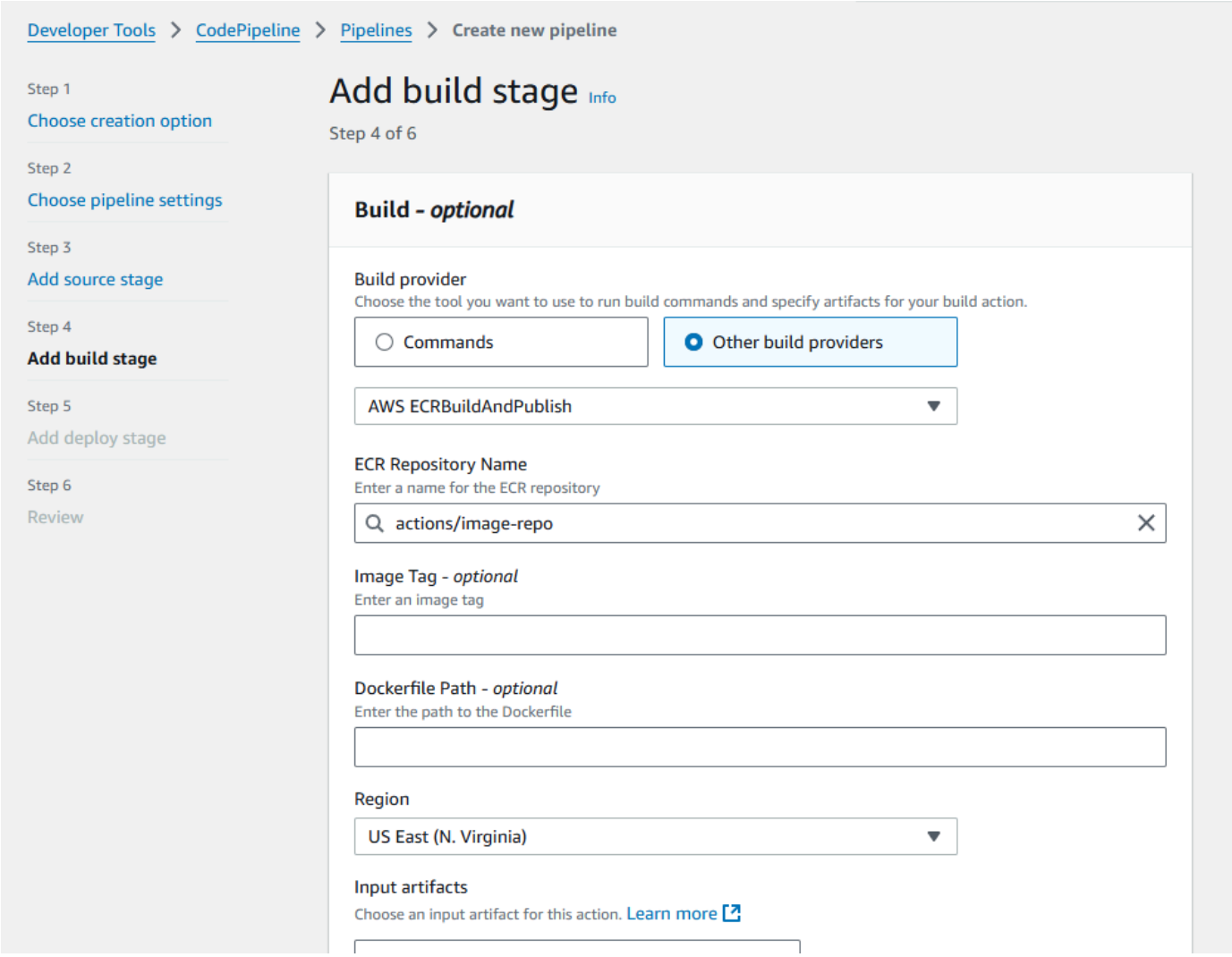
   a. In **Source provider**, choose **GitHub (via GitHub App)**.

   b. Under **Connection**, choose an existing connection or create a new one. To create or manage a connection for your GitHub source action, see GitHub connections.

   c. In **Repository name**, choose the name of your GitHub repository.

   d. In **Default branch**, choose the branch that you want to specify when the pipeline is started manually or with a source event that is not a Git tag. If the source of the change is not the trigger or if a pipeline execution was started manually, then the change used will be the HEAD commit from the default branch.

   Choose **Next**.
9. On the **Step 4: Add build stage** page, choose **Other build providers**, and then choose **ECRBuildAndPublish**.



a. For **ECR repository name**, choose your image repository.

b. Choose **Next**.

10. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

    Choose **Next**.

11. On the **Step 6: Add deploy stage** page, choose **Skip deploy stage**. You will add the ECS action in the following step.

12. On the **Step 7: Review** page, review your pipeline configuration and choose **Create pipeline** to create the pipeline.

13. Edit your pipeline to add the Amazon ECS deploy 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 **`Deploy`**.

c. Choose **+ Add action group**.

d. In **Action name**, enter a name.

e. In **Action provider**, choose Amazon ECS. Allow **Region** to default to the pipeline Region.

f. In **Input artifacts**, choose the input artifact from the source stage, such as `SourceArtifact`.

g. For **Cluster name**, choose the Amazon ECS cluster in which your service is running.

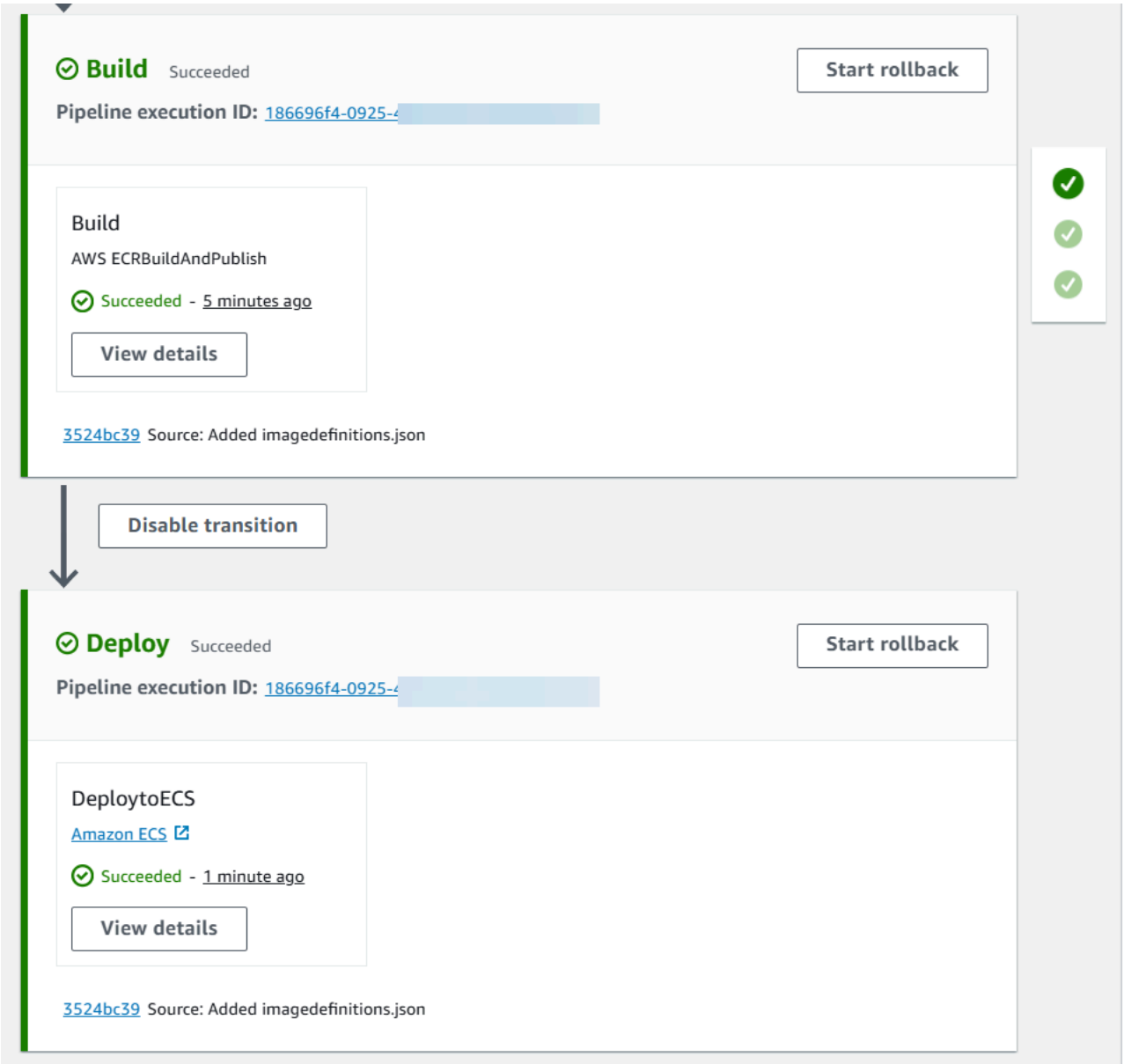
h. For **Service name**, choose the service to update.

i. Choose **Save**.

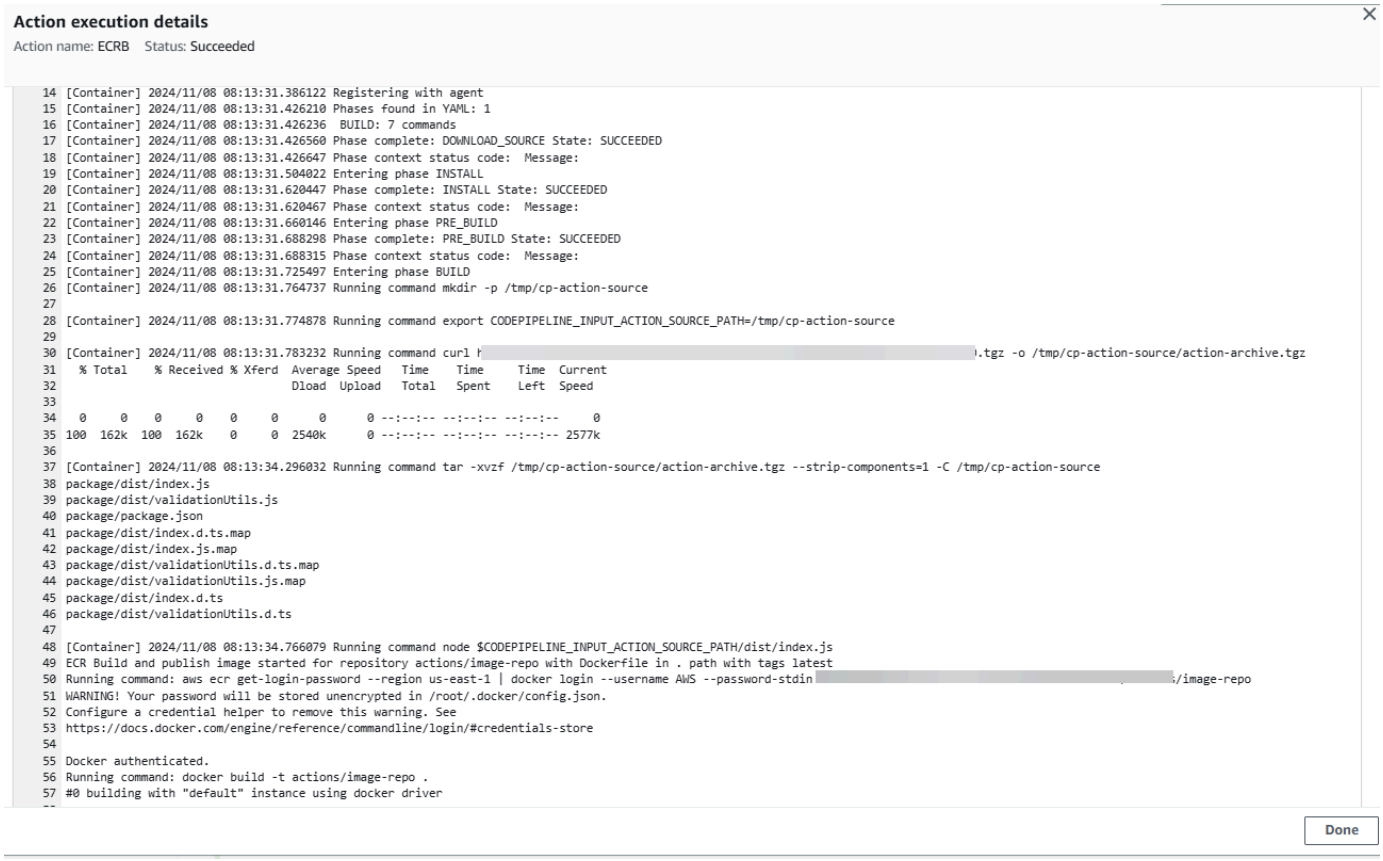
j. On the stage you are editing, choose **Done**. In the AWS CodePipeline pane, choose **Save**, and then choose **Save** on the warning message.

k. To submit your changes and start a pipeline build, choose **Release change**, and then choose **Release**.

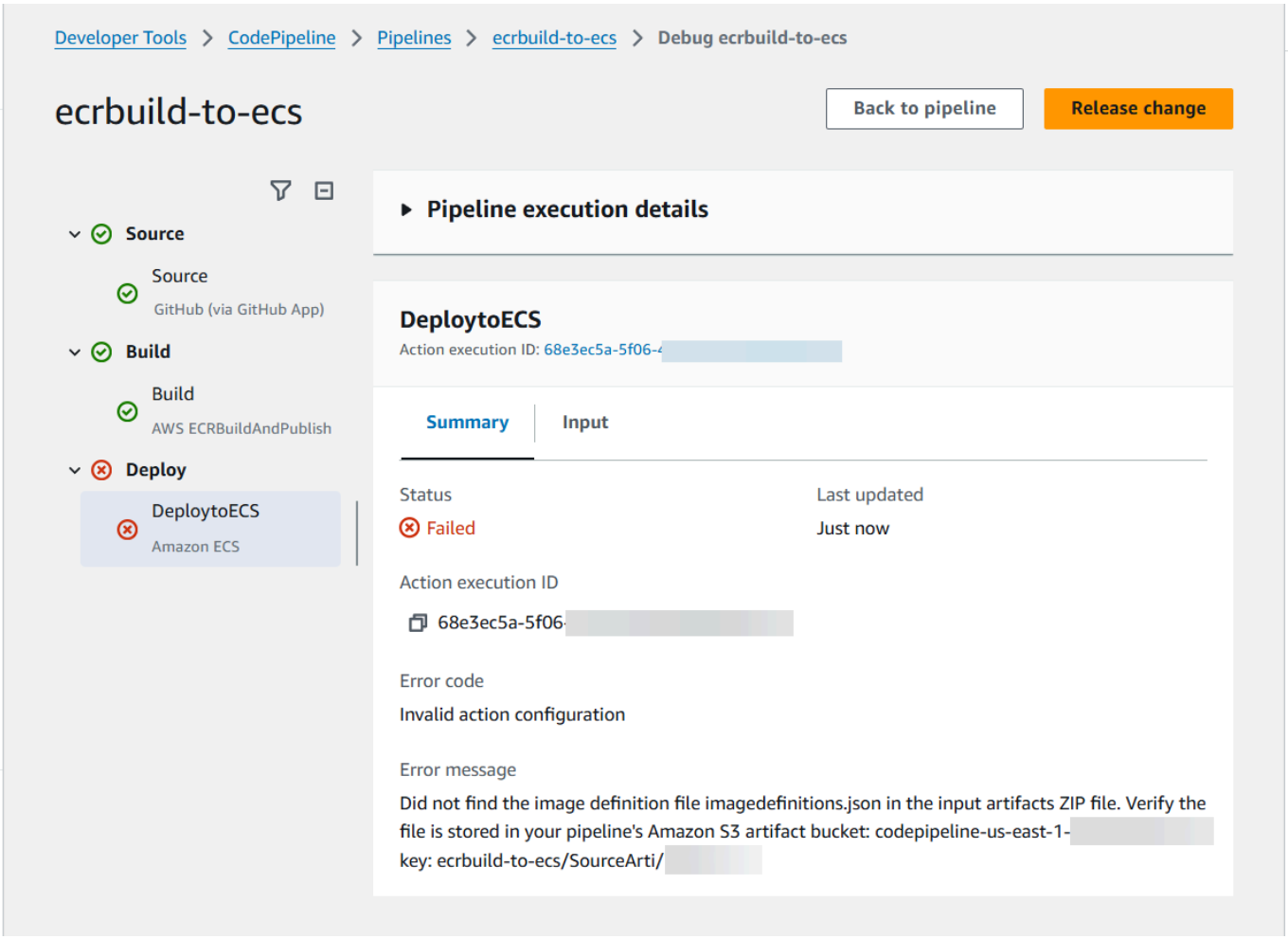
14. After the pipeline runs, view the pipeline structure and status.



15. After the pipeline runs successfully, choose **View details** to view the logs on the action to view the managed compute action output.



16. Troubleshoot any failed actions. For example, the ECS deploy action can fail if the imagedefinitions.json file is not in the source repository. The following is an example of the error message that displays when the imagedefinitions.json file is missing.



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.
2. Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
3. Choose your pipeline from the list.
4. Watch the pipeline progress through its stages. Your pipeline should complete and your action pushes the Docker image to ECR that was created from your code change.

Tutorial: Deploy to Amazon EKS with CodePipeline

This tutorial helps you to create a deploy action in CodePipeline that deploys your code to a cluster you have configured in Amazon EKS.

The EKS action supports both public and private EKS clusters. Private clusters are the type recommended by EKS; however, both types are supported.

> **Note**
>
> As part of creating a pipeline in the console, an S3 artifact bucket will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

> **Note**
>
> This action uses CodePipeline managed CodeBuild compute to run commands in a build environment. Running the commands action will incur separate charges in AWS CodeBuild.

> **Note**
>
> The EKS deploy action is only available for V2 type pipelines.

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.

- A source control repository (this tutorial uses GitHub) where you will add a sample `deployment.yaml` file.
- You must use an existing CodePipeline service role that you will update with the permissions for this action using Step 3: Update the CodePipeline service role policy in IAM below. The permissions needed are based on the type of cluster you create. For more information, see Service role policy permissions.
- A working image and repository tag that you have pushed to ECR or your image repository.

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

Step 1: (Optional) Create a cluster in Amazon EKS

You can choose to create an EKS cluster with a public or private endpoint.

In the following steps, you create a public or a private cluster in EKS. This step is optional if you have already created your cluster.

Create a public cluster in Amazon EKS

In this step, you create a cluster in EKS.

Create a public cluster

1. Open the EKS console, and then choose **Create cluster**.
2. In **Name**, name your cluster. Choose **Next**.
3. Choose **Create**.

Create a private cluster in Amazon EKS

In case you choose to create a cluster with a private endpoint, make sure to attach the private subnets only, and make sure they have an internet connection.

Follow the next five sub-steps for creating a cluster with a private endpoint.

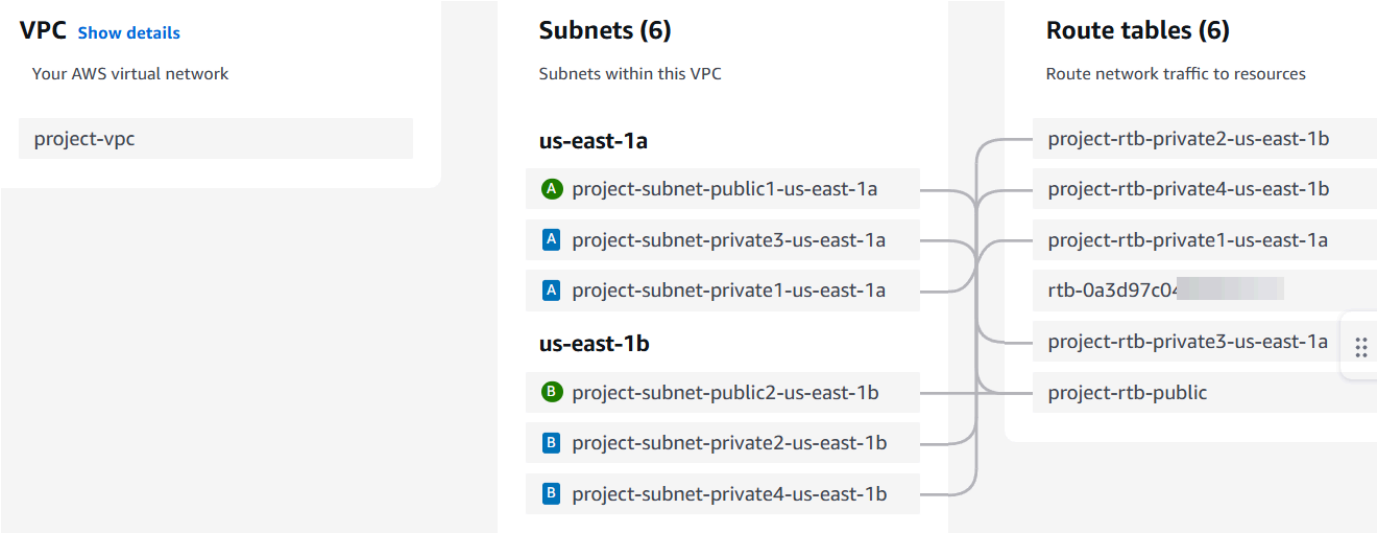
Create a VPC in the console

1. Open the VPC console, and then choose **Create VPC**.
2. Under **VPC settings**, choose **VPC and more**.

3. Choose to create one public and 4 private subnets. Choose **Create VPC**.
4. On the subnets page, choose **Private**.

Determine the private subnets in your VPC

1. Navigate to your VPC and choose the VPC ID to open the VPC details page.
2. On the VPC details page, choose the **Resource map** tab.
3. View the diagram and make a note of your private subnets. The subnets display with labels to indicate public or private status, and each subnet is mapped to a route table.



   Note that a private cluster will have all private subnets.
4. Create a public subnet to host the NAT gateway. You can attach only one internet gateway to a VPC at a time.

Create a NAT gateway in the public subnet

1. In the public subnet, create a NAT gateway. Navigate to the VPC console, and then choose **Internet gateways**. Choose **Create internet gateway**.
2. In Name, enter a name for your internet gateway. Choose **Create internet gateway**.

Update the route table for the private subnet to direct traffic to the NAT gateway.

Add the NAT gateway to your route tables for your private subnets

1. Navigate to the VPC console, and then choose **Subnets**.

2. For each private subnet, choose it and then choose the route table for that subnet on the details page. Choose **Edit route table**.
3. Update the route table for the private subnet to direct internet traffic to the NAT gateway. Choose **Add route**. Choose **NAT gateway** from the options to add. Choose the internet gateway you created.
4. For the public subnet, create a custom route table. Verify that the network access control list (ACL) for your public subnet allows inbound traffic from the private subnet.
5. Choose **Save changes**.

In this step, you create a cluster in EKS.

Create a private cluster

1. Open the EKS console, and then choose **Create cluster**.
2. In **Name**, name your cluster. Choose **Next**.
3. Specify your VPC and other configuration information. Choose **Create**.

Your EKS cluster can be a public or a private cluster. This step is for clusters that have ONLY a private endpoint. Make sure that if your cluster is private.

Step 2: Configure your private cluster in Amazon EKS

This step is applicable only if you have created a private cluster. This step is for clusters that have ONLY a private endpoint.

Configure your cluster

1. Attach private subnets only in the EKS cluster under the **Networking** tab. Attach the private subnets captured in the **Determine the private subnets in your VPC** section under Step 1: (Optional) Create a cluster in Amazon EKS.
2. Make sure that the private subnets have access to the internet since CodePipeline stores and retrieves artifacts from the S3 artifact bucket for your pipeline.

Step 3: Update the CodePipeline service role policy in IAM

In this step, you will update an existing CodePipeline service role, such as **`cp-service-role`**, with permissions required by CodePipeline to connect with your cluster. If you do not have an existing role, create a new one.

Update your CodePipeline service role with the following steps.

To update your CodePipeline service role policy

1. Open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. From the console dashboard, choose **Roles**.
3. Look up your CodePipeline service role, such as **`cp-service-role`**.
4. Add a new inline policy.
5. In the **Policy editor**, enter the following.

   - For a public cluster, add the following permissions.

     JSON

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

    "Statement": [
        {
            "Sid": "EksClusterPolicy",
            "Effect": "Allow",
            "Action": "eks:DescribeCluster",
            "Resource": "arn:aws-eusc:eks:eusc-de-
east-1:111122223333:cluster/my-cluster"
        },
        {
            "Sid": "EksVpcClusterPolicy",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeVpcs"
```

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

- For a private cluster, add the following permissions. Private clusters will require additional permissions for your VPC, if applicable.

  JSON

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

    "Statement": [
        {
            "Sid": "EksClusterPolicy",
            "Effect": "Allow",
            "Action": "eks:DescribeCluster",
            "Resource": "arn:aws-eusc:eks:eusc-de-
east-1:111122223333:cluster/my-cluster"
        },
        {
            "Sid": "EksVpcClusterPolicy",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeRouteTables",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeVpcs"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "ec2:CreateNetworkInterface",
            "Resource": "*",
            "Condition": {
```

```
                "StringEqualsIfExists": {
                    "ec2:Subnet": [
                        "arn:aws-eusc:ec2:eusc-de-east-1:ACCOUNT-ID:subnet/
subnet-03ebd65daeEXAMPLE",
                        "arn:aws-eusc:ec2:eusc-de-east-1:ACCOUNT-ID:subnet/
subnet-0e377f6036EXAMPLE",
                        "arn:aws-eusc:ec2:eusc-de-east-1:ACCOUNT-ID:subnet/
subnet-0db658ba1cEXAMPLE",
                        "arn:aws-eusc:ec2:eusc-de-east-1:ACCOUNT-ID:subnet/
subnet-0db658ba1cEXAMPLE"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "ec2:CreateNetworkInterfacePermission",
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "ec2:Subnet": [
                        "arn:aws-eusc:ec2:eusc-de-
east-1:111122223333:subnet/subnet-03ebd65daeEXAMPLE",
                        "arn:aws-eusc:ec2:eusc-de-
east-1:111122223333:subnet/subnet-0e377f6036EXAMPLE",
                        "arn:aws-eusc:ec2:eusc-de-
east-1:111122223333:subnet/subnet-0db658ba1cEXAMPLE",
                        "arn:aws-eusc:ec2:eusc-de-
east-1:111122223333:subnet/subnet-0db658ba1cEXAMPLE"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "ec2:DeleteNetworkInterface",
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "ec2:Subnet": [
                        "arn:aws-eusc:ec2:eusc-de-east-1:ACCOUNT-ID:subnet/
subnet-03ebd65daeEXAMPLE",
                        "arn:aws-eusc:ec2:eusc-de-east-1:ACCOUNT-ID:subnet/
subnet-0e377f6036EXAMPLE",
```

```
                        "arn:aws-eusc:ec2:eusc-de-east-1:ACCOUNT-ID:subnet/
subnet-0db658ba1cEXAMPLE",
                        "arn:aws-eusc:ec2:eusc-de-east-1:ACCOUNT-ID:subnet/
subnet-0db658ba1cEXAMPLE"
                    ]
                }
            }
        }
    ]
}
```

6. Choose **Update policy**.

Step 4: Create an access entry for the CodePipeline service role

In this step, you create an access entry on your cluster that will add the CodePipeline service role you updated in Step 3, along with a managed access policy.

1. Open the EKS console and navigate to your cluster.
2. Choose the **Access** tab.
3. Under **IAM access entries**, choose **Create access entry**.
4. In **IAM principal ARN**, enter the role you just updated for the action, such as **`cp-service-role`**. Choose **Next**.
5. On the **Step 2: Add access policy** page, in **Policy name**, choose the managed policy for access, such as `AmazonEKSClusterAdminPolicy`. Choose **Add policy**. Choose **Next**.

   > **Note**
   >
   > This is the policy that the CodePipeline action uses to talk to Kubernetes. As a best practice, to scope down permissions in your policy with least privilege rather than the administrative policy, attach a custom policy instead.

6. On the review page, choose **Create**.

Step 5: Create a source repository and add the `Helm chart` config files

In this step, you create a config file that is appropriate for your action (Kubernetes manifest files or Helm chart) and store the config file in your source repository. Use the appropriate file for your

configuration. For more information, see https://kubernetes.io/docs/reference/kubectl/quick-reference/ or https://helm.sh/docs/topics/charts/.

- For Kubernetes, use a manifest file.
- For Helm, use a Helm chart.

1. Create or use an existing GitHub repository.
2. Create a new structure in your repository for your Helm chart files as shown in the example below.

```
mychart
|-- Chart.yaml
|-- charts
|-- templates
|   |-- NOTES.txt
|   |-- _helpers.tpl
|   |-- deployment.yaml
|   |-- ingress.yaml
|   `-- service.yaml
`-- values.yaml
```

3. Add the file to the root level of your repository.

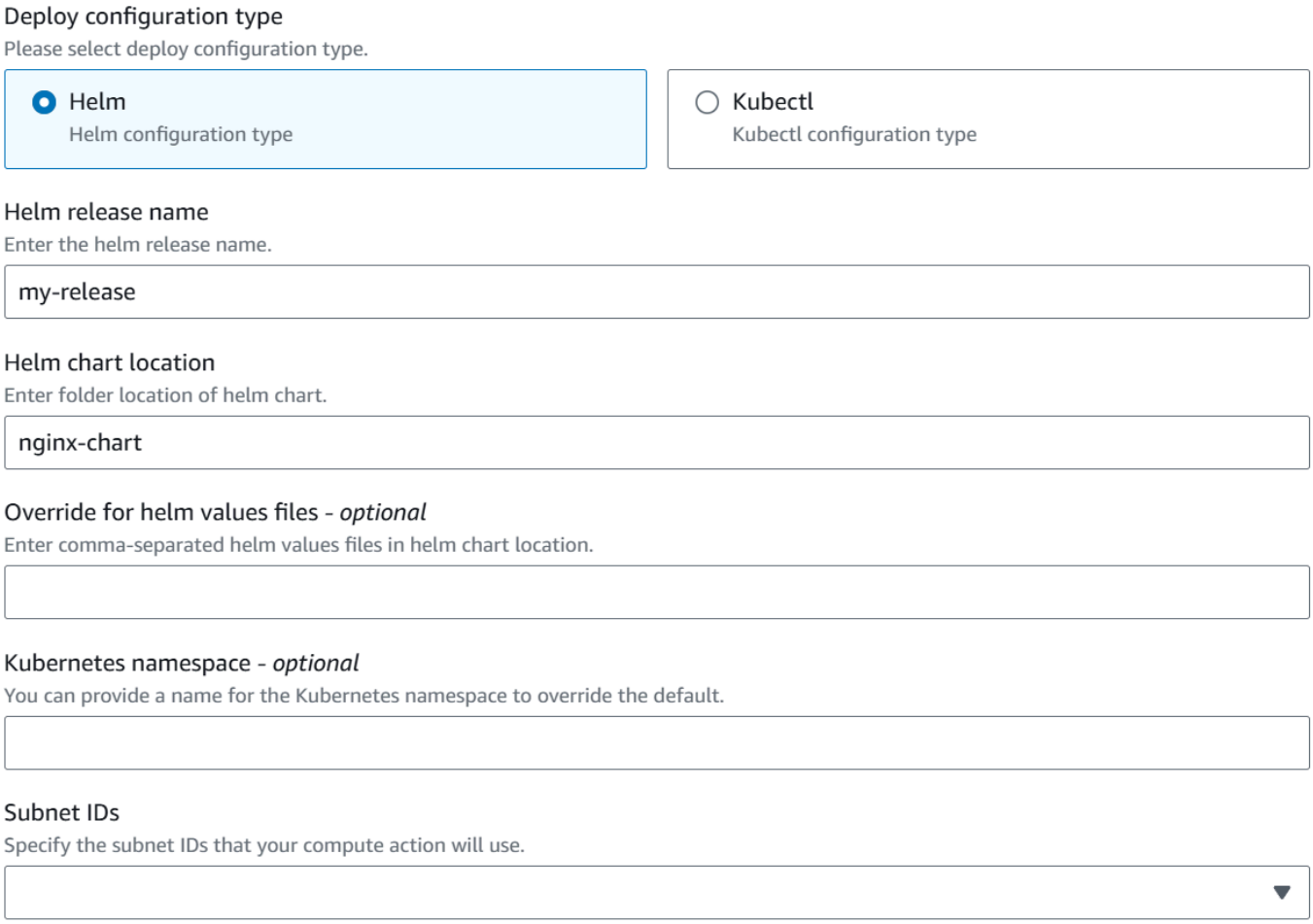
Step 6: Creating your pipeline

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

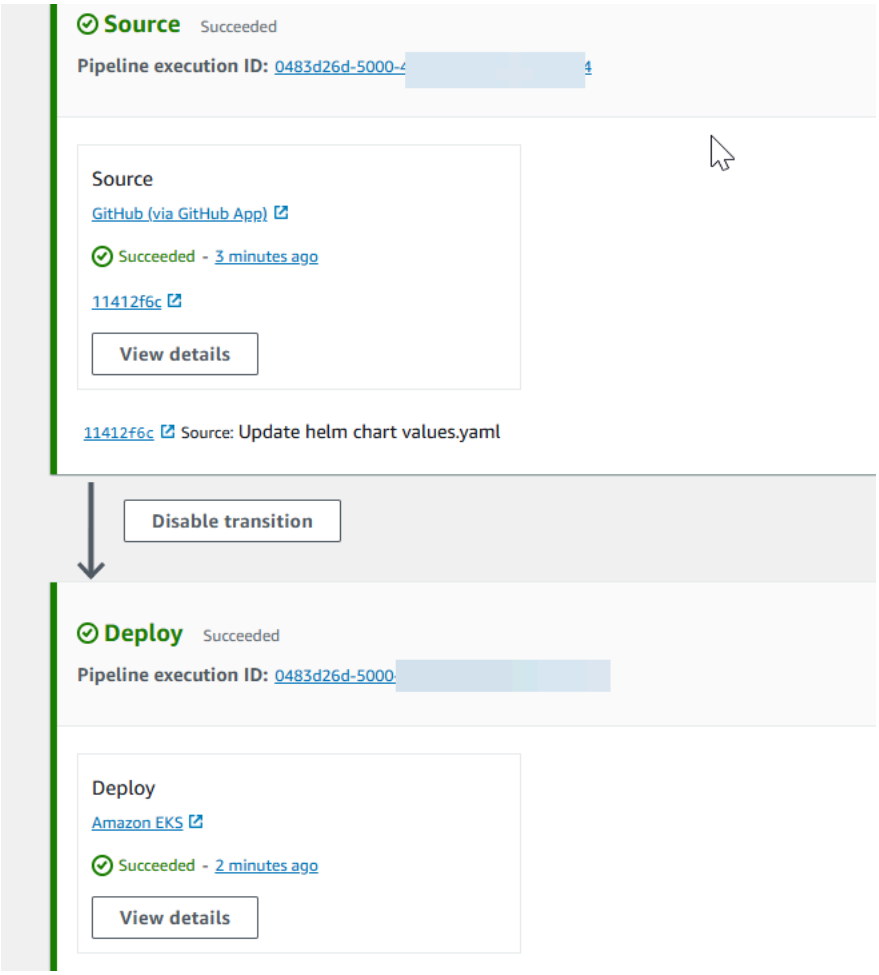
To create your pipeline

1. Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
2. On the **Welcome** page, **Getting started** page, or the **Pipelines** page, choose **Create pipeline**.
3. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. In **Step 2: Choose pipeline settings**, in **Pipeline name**, enter **`MyEKSPipeline`**.
5. CodePipeline provides V1 and V2 type pipelines, which differ in characteristics and price. The V2 type is the only type you can choose in the console. For more information, see pipeline types. For information about pricing for CodePipeline, see Pricing.

6. In **Service role**, choose the service role that you updated in Step 3.
7. Leave the settings under **Advanced settings** at their defaults, and then choose **Next**.
8. On the **Step 3: Add source stage** page, for **Source provider**, choose **to create a connection to your GitHub repository**.
9. On the **Step 4: Add build stage** page, choose **Skip**.
10. On the **Step 5: Add deploy stage** page, choose **Amazon EKS**.



    a. Under **Deploy configuration type**, choose **Helm**.
    b. In **Helm chart location**, enter the release name, such as `my-release`. For **Helm chart location**, enter the path for your Helm chart files, such as `mychart`.
    c. Choose **Next**.
11. On the **Step 6: Review** page, review your pipeline configuration and choose **Create pipeline** to create the pipeline.



12. After the pipeline runs successfully, choose **View details** to view the logs on the action to view the action output.

Tutorial: Create a pipeline that runs commands with compute (V2 type)

In this tutorial, you configure a pipeline that continuously runs provided build commands using the Commands action in a build stage. For more information about the Commands action, see Commands action reference.

> **Important**
>
> As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline,

make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

Prerequisites

You must already have the following:

- A GitHub repository. You can use the GitHub repository you created in Tutorial: Use full clone with a GitHub pipeline source.

Step 1: Create source files and push to your GitHub repository

In this section, you create and push your example source files to the repository that the pipeline uses for your source stage. For this example, you produce and push the following:

- A `README.txt` file.

To create source files

1. Create a file with the following text:

   ```
   Sample readme file
   ```

2. Save the file as `README.txt`.

To push files to your GitHub repository

1. Push or upload the files to your 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:

   ```
   README.txt
   ```

2. To use the Git command line from a cloned repository on your local computer:

   a. Run the following command to stage all of your files at once:

```
git add -A
```

b. Run the following command to commit the files with a commit message.

```
git commit -m "Added source files"
```

c. Run the following command to push the files from your local repo to your repository:

```
git push
```

Step 2: Create your pipeline

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

- A source stage with a GitHub (via GitHub App) action for the repository where the source files are stored.
- A build stage with the Commands action.

To create a pipeline with the wizard

1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home.
2. On the **Welcome** page, **Getting started** page, or the **Pipelines** page, choose **Create pipeline**.
3. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. In **Step 2: Choose pipeline settings**, in **Pipeline name**, enter **`MyCommandsPipeline`**.
5. CodePipeline provides V1 and V2 type pipelines, which differ in characteristics and price. The V2 type is the only type you can choose in the console. For more information, see pipeline types. For information about pricing for CodePipeline, see Pricing.
6. In **Service role**, choose **New service role** to allow CodePipeline to create a service role in IAM.

> **Note**
>
> If you are using an existing service role, to use the Commands action, you will need to add the following permissions for the service role. Scope down the permissions

to the pipeline resource level by using resource-based permissions in the service role policy statement. For more information, see the policy example in Service role policy permissions.

- logs:CreateLogGroup
- logs:CreateLogStream
- logs:PutLogEvents

7. Leave the settings under **Advanced settings** at their defaults, and then choose **Next**.
8. On the **Step 3: Add source stage** page, add a source stage:

   a. In **Source provider**, choose **GitHub (via GitHub App)**.

   b. Under **Connection**, choose an existing connection or create a new one. To create or manage a connection for your GitHub source action, see GitHub connections.

   c. In **Repository name**, choose the name of your GitHub.com repository.

   d. In **Default branch**, choose the branch that you want to specify when the pipeline is started manually or with a source event that is not a Git tag. If the source of the change is not the trigger or if a pipeline execution was started manually, then the change used will be the HEAD commit from the default branch. Optionally, you can also specify webhooks with filtering (triggers). For more information, see Automate starting pipelines using triggers and filtering.

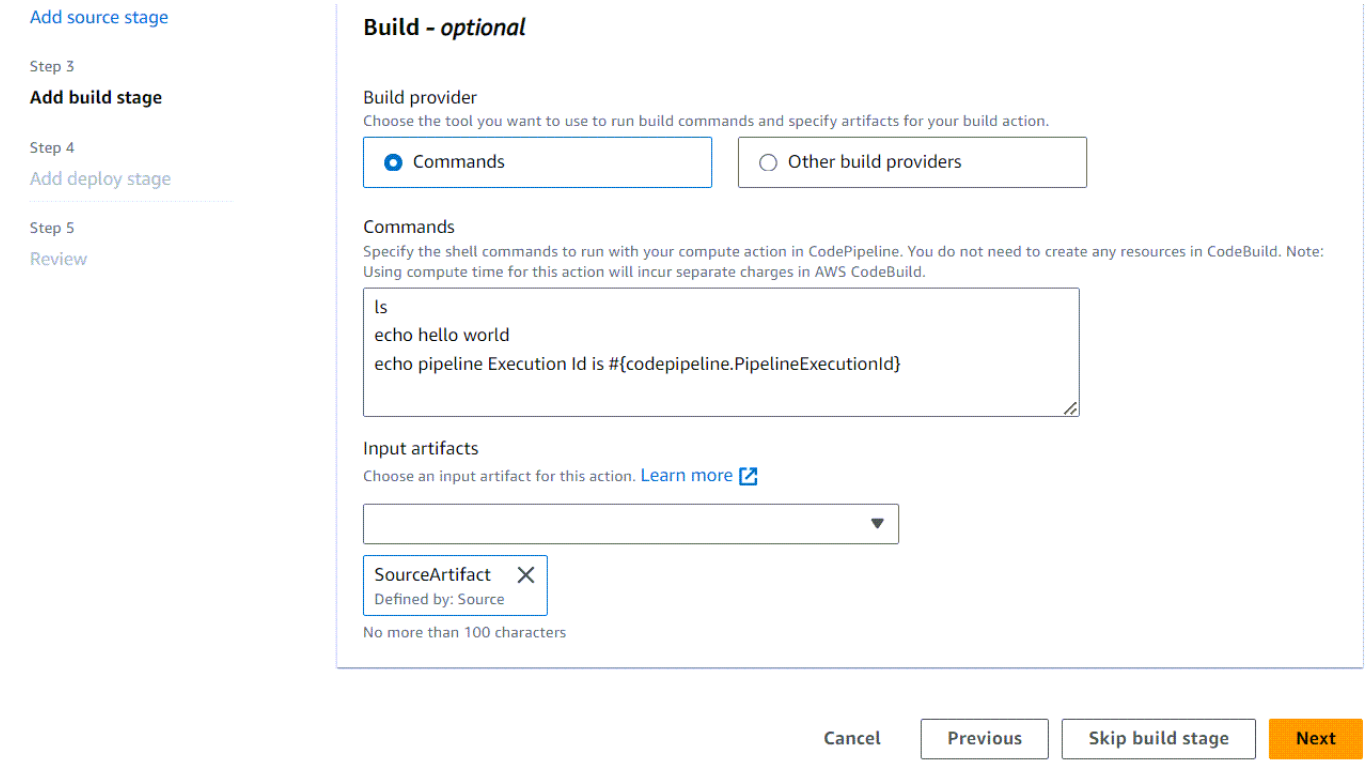
   Choose **Next**.
9. In **Step 4: Add build stage**, choose **Commands**.

> **Note**
>
> Running the Commands action will incur separate charges in AWS CodeBuild.

Enter the following commands:

```
ls
echo hello world
cat README.txt
echo pipeline Execution Id is #{codepipeline.PipelineExecutionId}
```

Choose **Next**.



10. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

    Choose **Next**.

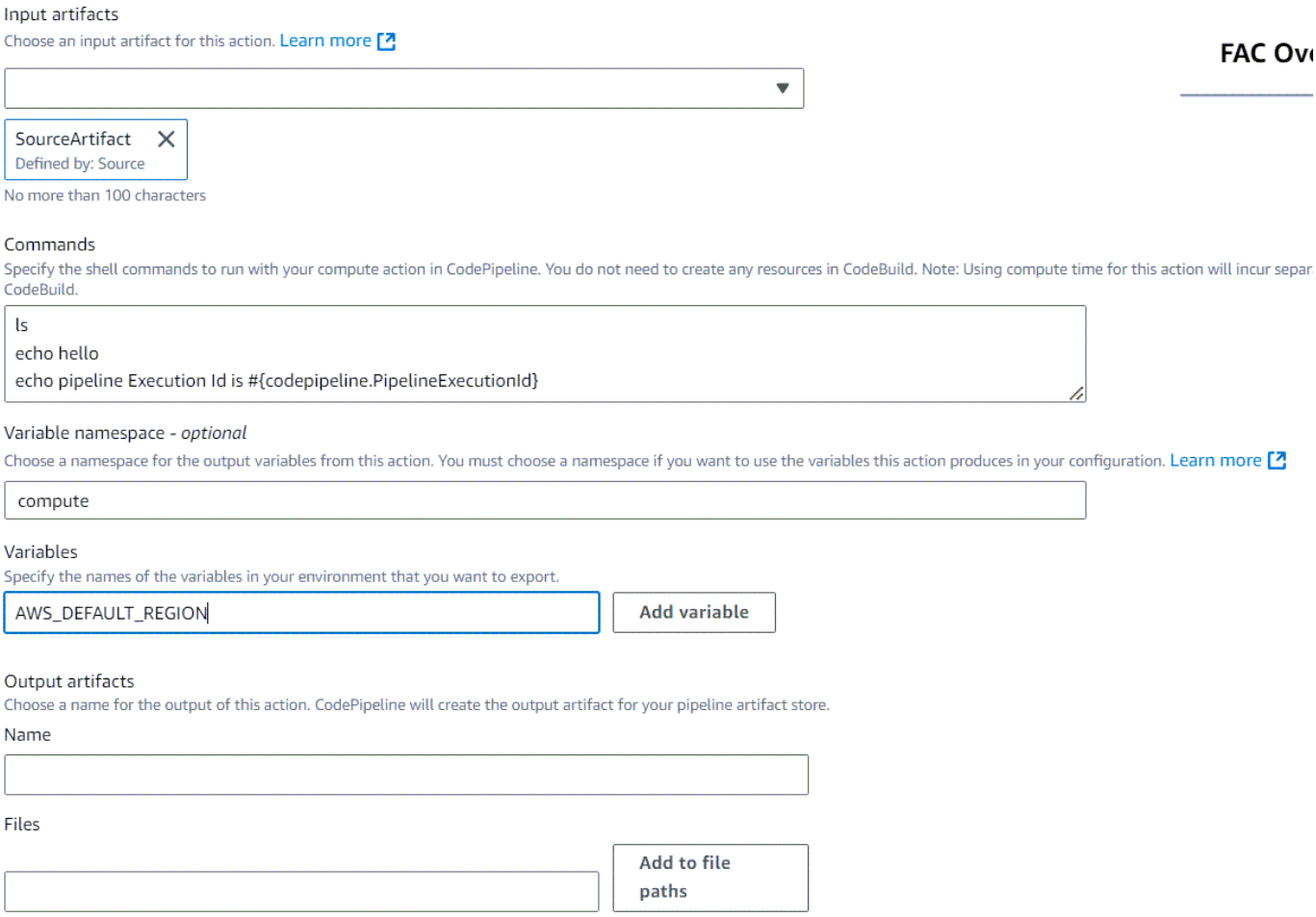
11. In **Step 6: Add deploy stage**, choose **Skip deploy stage**, and then accept the warning message by choosing **Skip** again.

    Choose **Next**.

12. In **Step 7: Review**, review the information, and then choose **Create pipeline**.

13. As a final step for creating your action, add an environment variable to the action that will result in an output variable for the action. On the Commands action, choose **Edit**. On the **Edit** screen, specify a variable namespace for your action by entering `compute` in the **Variable namespace** field.

    Add the CodeBuild output variable `AWS_Default_Region`, and then choose **Add variable**.



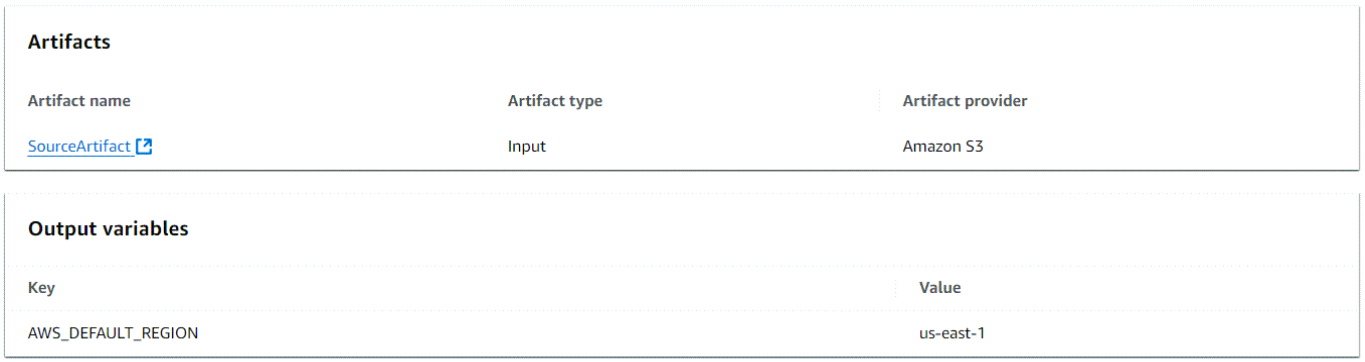
Step 3: Run your pipeline and verify build commands

Release a change to run your pipeline. Verify that the build commands ran by viewing the execution history, the build logs, and the output variables.

To view action logs and output variables

1. After the pipeline runs successfully, you can view the logs and output for the action.
2. To view the output variables for the action, choose **History**, and then choose **Timeline**.

   View the output variable that was added to the action. The output for the Commands action shows the output variable resolved to the action Region.



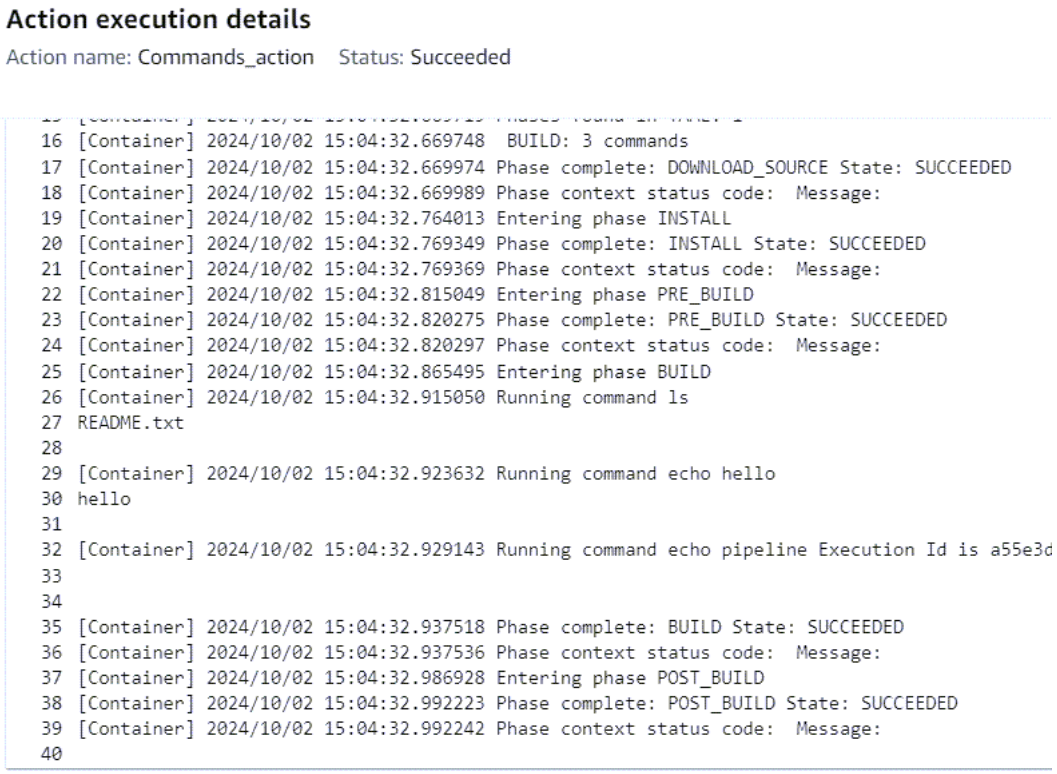
Artifacts

Artifact name	Artifact type	Artifact provider
SourceArtifact	Input	Amazon S3

Output variables

Key	Value
AWS_DEFAULT_REGION	us-east-1

3. To view the logs for the action, choose **View details** on the successful Commands action. View the logs for the Commands action.



Action execution details

Action name: Commands_action Status: Succeeded

```
16 [Container] 2024/10/02 15:04:32.669748  BUILD: 3 commands
17 [Container] 2024/10/02 15:04:32.669974 Phase complete: DOWNLOAD_SOURCE State: SUCCEEDED
18 [Container] 2024/10/02 15:04:32.669989 Phase context status code:  Message:
19 [Container] 2024/10/02 15:04:32.764013 Entering phase INSTALL
20 [Container] 2024/10/02 15:04:32.769349 Phase complete: INSTALL State: SUCCEEDED
21 [Container] 2024/10/02 15:04:32.769369 Phase context status code:  Message:
22 [Container] 2024/10/02 15:04:32.815049 Entering phase PRE_BUILD
23 [Container] 2024/10/02 15:04:32.820275 Phase complete: PRE_BUILD State: SUCCEEDED
24 [Container] 2024/10/02 15:04:32.820297 Phase context status code:  Message:
25 [Container] 2024/10/02 15:04:32.865495 Entering phase BUILD
26 [Container] 2024/10/02 15:04:32.915050 Running command ls
27 README.txt
28
29 [Container] 2024/10/02 15:04:32.923632 Running command echo hello
30 hello
31
32 [Container] 2024/10/02 15:04:32.929143 Running command echo pipeline Execution Id is a55e3d
33
34
35 [Container] 2024/10/02 15:04:32.937518 Phase complete: BUILD State: SUCCEEDED
36 [Container] 2024/10/02 15:04:32.937536 Phase context status code:  Message:
37 [Container] 2024/10/02 15:04:32.986928 Entering phase POST_BUILD
38 [Container] 2024/10/02 15:04:32.992223 Phase complete: POST_BUILD State: SUCCEEDED
39 [Container] 2024/10/02 15:04:32.992242 Phase context status code:  Message:
40
```

Tutorial: Use Git tags to start your pipeline

In this tutorial, you will create a pipeline that connects to your GitHub repository where the source action is configured for the Git tags trigger type. When a Git tag is created on a commit, your pipeline starts. This example shows you how to create a pipeline that allows filtering for tags

based on the syntax of the tag name. For more information about filtering with glob patterns, see Working with glob patterns in syntax.

> **Important**
>
> As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

This tutorial connects to GitHub through the `CodeStarSourceConnection` action type.

Topics

- Prerequisites
- Step 1: Open CloudShell and clone your repository
- Step 2: Create a pipeline to trigger on Git tags
- Step 3: Tag your commits for release
- Step 4: Release change and view logs

Prerequisites

Before you begin, you must do the following:

- Create a GitHub repository with your GitHub account.
- Have your GitHub credentials ready. When you use the AWS Management Console to set up a connection, you are asked to sign in with your GitHub credentials.

Step 1: Open CloudShell and clone your repository

You can use a command line interface to clone your repository, make commits, and add tags. This tutorial launches a CloudShell instance for the command line interface.

1. Sign in to the AWS Management Console.

2. In the top navigation bar, choose the AWS icon. The main page of the AWS Management Console displays.
3. In the top navigation bar, choose the AWS CloudShell icon. CloudShell opens. Wait while the CloudShell environment is created.

> **Note**
>
> If you don't see the CloudShell icon, make sure that you're in a Region supported by CloudShell. This tutorial assumes you are in the US West (Oregon) Region.

4. In GitHub, navigate to your repository. Choose **Code**, and then choose **HTTPS**. Copy the path. The address to clone your Git repository is copied to your clipboard.
5. Run the following command to clone the repository.

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

6. Enter your GitHub account `Username` and `Password` when prompted. For the `Password` entry, you must use a user-created token rather than your account password.

Step 2: Create a pipeline to trigger on Git tags

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

- A source stage with a connection to your GitHub repository and action.
- A build stage with an AWS CodeBuild build action.

To create a pipeline with the wizard

1. Sign in to the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
2. On the **Welcome** page, **Getting started** page, or **Pipelines** page, choose **Create pipeline**.
3. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. In **Step 2: Choose pipeline settings**, in **Pipeline name**, enter **`MyGitHubTagsPipeline`**.
5. In **Pipeline type**, keep the default selection at **V2**. Pipeline types differ in characteristics and price. For more information, see Pipeline types.

6. In **Service role**, choose **New service role**.

   > **Note**
   >
   > If you choose instead to use your existing CodePipeline service role, make sure that you have added the `codestar-connections:UseConnection` IAM permission to your service role policy. For instructions for the CodePipeline service role, see Add permissions to the CodePipeline service role.

7. Under **Advanced settings**, leave the defaults. 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.

   > **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.

   Choose **Next**.

8. On the **Step 3: Add source stage** page, add a source stage:

   a. In **Source provider**, choose **GitHub (via GitHub App)**.

   b. Under **Connection**, choose an existing connection or create a new one. To create or manage a connection for your GitHub source action, see GitHub connections.

   c. In **Repository name**, choose the name of your GitHub repository.

   d. In **Default branch**, choose the branch that you want to specify when the pipeline is started manually or with a source event that is not a Git tag. If the source of the change is not the trigger or if a pipeline execution was started manually, then the change used will be the HEAD commit from the default branch.

   e. Under **Webhook events**, in **Filter type**, choose **Tags**.

      In the **Tags or patterns** field, enter `release*`.

> **Important**
>
> Pipelines that start with a trigger type of Git tags will be configured for WebhookV2 events and will not use the Webhook event (change detection on all push events) to start the pipeline.

Choose **Next**.

9. 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:5.0**.

   f. For **Service role**, choose **New service role**.

   > **Note**
   >
   > Note the name of your CodeBuild service role. You will need the role name for the final step in this tutorial.

   g. Under **Buildspec**, for **Build specifications**, choose **Insert build commands**. Choose **Switch to editor**, and paste the following under **Build commands**.

```
version: 0.2
#env:
  #variables:
     # key: "value"
     # key: "value"
  #parameter-store:
     # key: "value"
     # key: "value"
  #git-credential-helper: yes
phases:
  install:
    #If you use the Ubuntu standard image 2.0 or later, you must specify
 runtime-versions.
```

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

h. Choose **Continue to CodePipeline**. This returns to the CodePipeline console and creates a CodeBuild project that uses your build commands 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**.

10. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

    Choose **Next**.

11. On the **Step 6: Add deploy stage** page, choose **Skip deploy stage**, and then accept the warning message by choosing **Skip** again. Choose **Next**.

12. On **Step 7: Review**, choose **Create pipeline**.

Step 3: Tag your commits for release

After you create your pipeline and specify Git tags, you can tag commits in your GitHub repository. In these steps, you will tag a commit with the `release-1` tag. Each commit in a Git repository must have a unique Git tag. When you choose the commit and tag it, this allows you to incorporate changes from different branches into your pipeline deployment. Note that the tag name release does not apply to the concept of a release in GitHub.

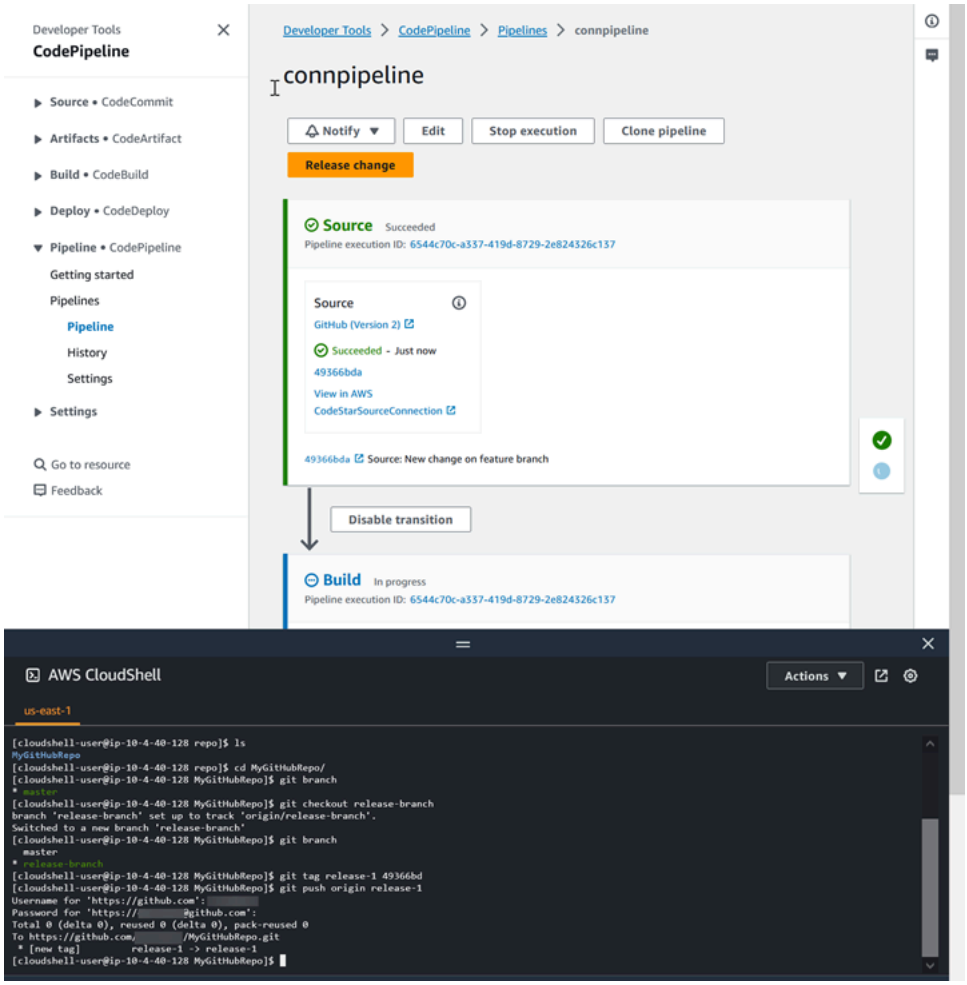
1. Reference the copied commit IDs you want to tag. To view commits in each branch, in the CloudShell terminal, enter the following command to capture the commit IDs you want to tag:

   ```
   git log
   ```

2. In the CloudShell terminal, enter the command to tag your commit and push it to origin. After you tag your commit, you use the git push command to push the tag to origin. In the following example, enter the following command to use the `release-1` tag for the second commit with ID 49366bd. This tag will be filtered by the pipeline `release*` tag filter and will start the pipeline.

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

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



Step 4: Release change and view logs

1. After the pipeline runs successfully, on your successful build stage, choose **View log**.

   Under **Logs**, view the CodeBuild build output. The commands output the value of the entered variable.

2. In the **History** page, view the **Triggers** column. View the trigger type **GitTag : release-1**.

Tutorial: Filter on branch names for pull requests to start your pipeline (V2 type)

In this tutorial, you will create a pipeline that connects to your GitHub.com repository where the source action is configured to start your pipeline with a trigger configuration that filters on pull

requests. When a specified pull request event occurs for a specified branch, your pipeline starts. This example shows you how to create a pipeline that allows filtering for branch names. For more information about working with triggers, see Add filters for push and pull request event types (CLI). For more information about filtering with regex patterns in glob format, see Working with glob patterns in syntax.

> **Important**
>
> As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

This tutorial connects to GitHub.com through the `CodeStarSourceConnection` action type.

Topics

- Prerequisites
- Step 1: Create a pipeline to start on pull request for specified branches
- Step 2: Create and merge a pull request in GitHub.com to start your pipeline executions

Prerequisites

Before you begin, you must do the following:

- Create a GitHub.com repository with your GitHub.com account.
- Have your GitHub credentials ready. When you use the AWS Management Console to set up a connection, you are asked to sign in with your GitHub credentials.

Step 1: Create a pipeline to start on pull request for specified branches

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

- A source stage with a connection to your GitHub.com repository and action.
- A build stage with an AWS CodeBuild build action.

To create a pipeline with the wizard

1. Sign in to the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
2. On the **Welcome** page, **Getting started** page, or **Pipelines** page, choose **Create pipeline**.
3. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. In **Step 2: Choose pipeline settings**, in **Pipeline name**, enter **`MyFilterBranchesPipeline`**.
5. In **Pipeline type**, keep the default selection at **V2**. Pipeline types differ in characteristics and price. For more information, see Pipeline types.
6. In **Service role**, choose **New service role**.

   > **Note**
   >
   > If you choose instead to use your existing CodePipeline service role, make sure that you have added the `codeconnections:UseConnection` IAM permission to your service role policy. For instructions for the CodePipeline service role, see Add permissions to the CodePipeline service role.

7. Under **Advanced settings**, leave the defaults. 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.

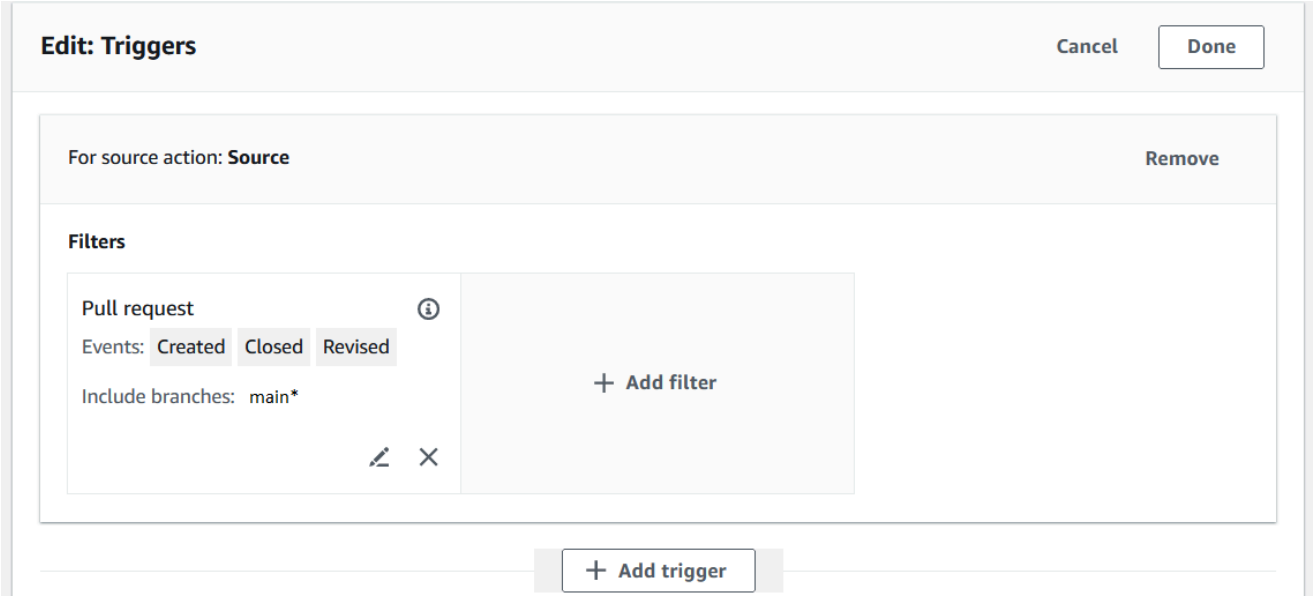
   > **Note**
   >
   > This is not the source bucket for your source code. This is the artifact store for your pipeline. A separate artifact store, such as an S3 bucket, is required for each pipeline.

   Choose **Next**.
8. On the **Step 3: Add source stage** page, add a source stage:
   a. In **Source provider**, choose **GitHub (via GitHub App)**.
   b. Under **Connection**, choose an existing connection or create a new one. To create or manage a connection for your GitHub source action, see GitHub connections.
   c. In **Repository name**, choose the name of your GitHub.com repository.

d. Under **Trigger type**, choose **Specify filter**.

Under **Event type**, choose **Pull request**. Select all of the events under pull request so that the event occurs for created, updated, or closed pull requests.

Under **Branches**, in the **Include** field, enter `main*`.



> **Important**
>
> Pipelines that start with this trigger type will be configured for WebhookV2 events and will not use the Webhook event (change detection on all push events) to start the pipeline.

Choose **Next**.

9. In **Step 4: Add build stage**, in **Build provider**, choose **AWS CodeBuild**. Allow **Region** to default to the pipeline Region. Choose or create the build project as instructed in Tutorial: Use Git tags to start your pipeline. This action will only be used in this tutorial as the second stage needed to create your pipeline.
10. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

    Choose **Next**.

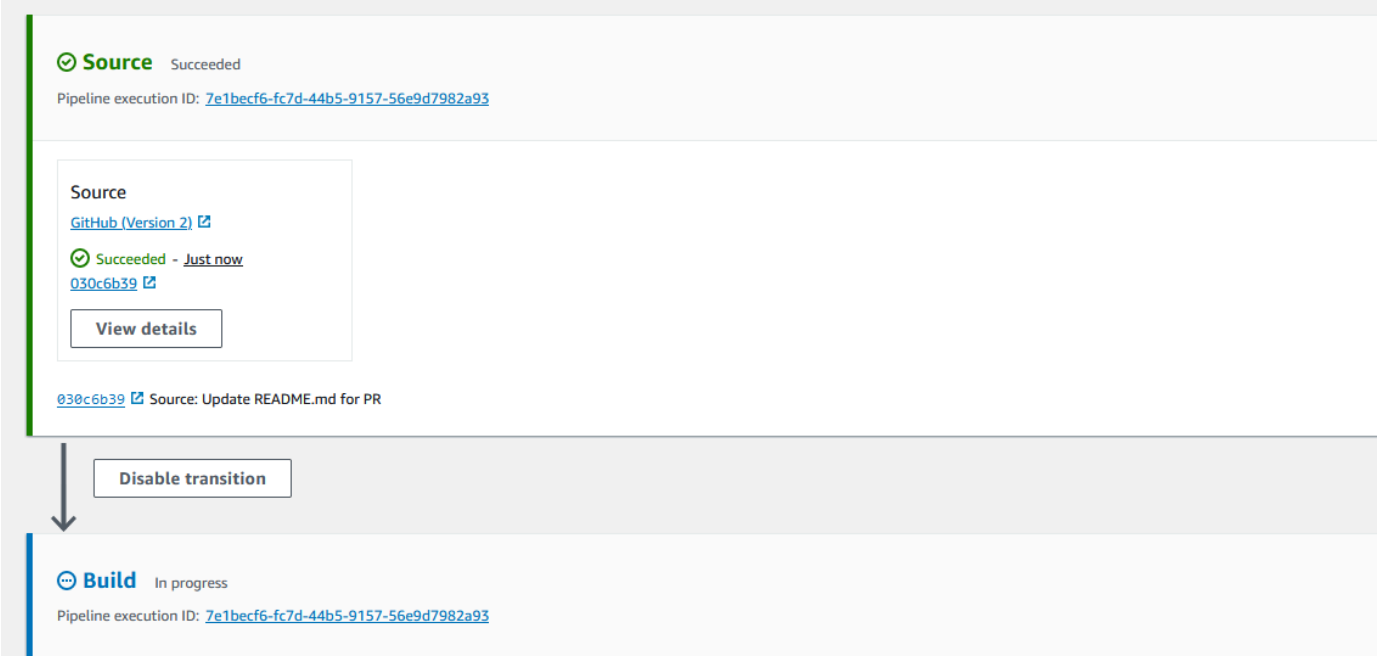
11. On the **Step 6: Add deploy stage** page, choose **Skip deploy stage**, and then accept the warning message by choosing **Skip** again. Choose **Next**.
12. On **Step 7: Review**, choose **Create pipeline**.

Step 2: Create and merge a pull request in GitHub.com to start your pipeline executions

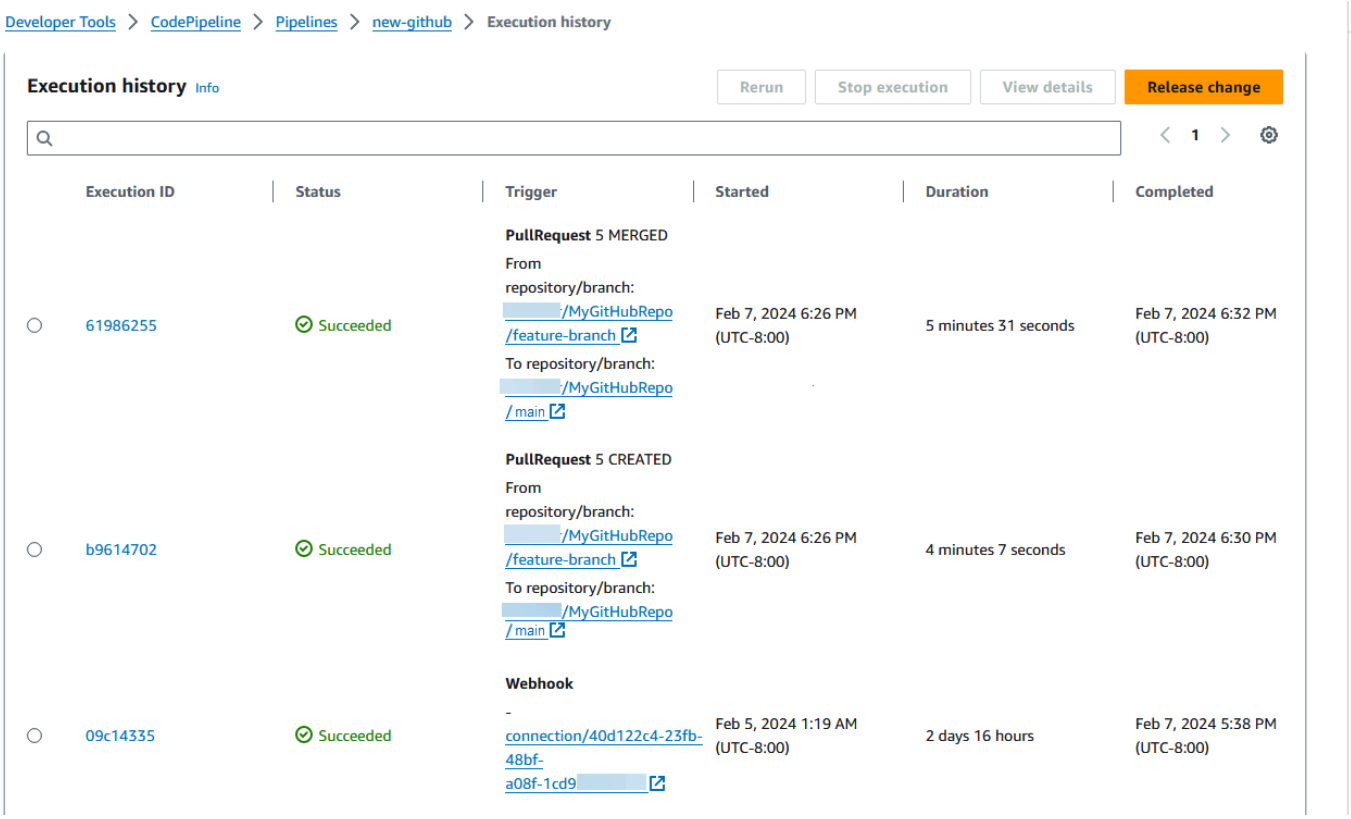
In this section, you create and merge a pull request. This starts your pipeline, with one execution for the opened pull request and one execution for the closed pull request.

To create a pull request and start your pipeline

1. In GitHub.com, create a pull request by making a change to the README.md on a feature branch and raising a pull request to the `main` branch. Commit the change with a message like `Update README.md for PR`.
2. The pipeline starts with the source revision showing the **Source** message for the pull request as **Update README.md for PR**.



3. Choose **History**. In the pipeline execution history, view the CREATED and MERGED pull request status events that started the pipeline executions.



Tutorial: Use pipeline-level variables

In this tutorial, you will create a pipeline where you add a variable at the pipeline level and run a CodeBuild build action that outputs your variable value.

> **Important**
>
> As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

Topics

- Prerequisites
- Step 1: Create your pipeline and build project
- Step 2: Release change and view logs

Prerequisites

Before you begin, you must do the following:

- Create a CodeCommit repository.
- Add a .txt file to the repository.

Step 1: Create your pipeline and build project

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

- A source stage with a connection to your CodeCommit repository.
- A build stage with an AWS CodeBuild build action.

To create a pipeline with the wizard

1. Sign in to the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
2. On the **Welcome** page, **Getting started** page, or **Pipelines** page, choose **Create pipeline**.
3. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. In **Step 2: Choose pipeline settings**, in **Pipeline name**, enter **`MyVariablesPipeline`**.
5. In **Pipeline type**, keep the default selection at **V2**. Pipeline types differ in characteristics and price. For more information, see Pipeline types.
6. In **Service role**, choose **New service role**.

> **Note**
>
> If you choose instead to use your existing CodePipeline service role, make sure that you have added the `codeconnections:UseConnection` IAM permission to your service role policy. For instructions for the CodePipeline service role, see Add permissions to the CodePipeline service role.

7. Under **Variables**, choose **Add variable**. In **Name**, enter `timeout`. In **Default**, enter 1000. In description, enter the following description: **`Timeout`**.

This will create a variable where you can declare the value when the pipeline execution starts. Variable names must match `[A-Za-z0-9@\-_]+` and can be anything except an empty string.

8. Under **Advanced settings**, leave the defaults. 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.

   > **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.

   Choose **Next**.

9. On the **Step 3: Add source stage** page, add a source stage:

   a. In **Source provider**, choose **AWS CodeCommit**.

   b. In **Repository name** and **Branch name**, choose your repository and branch.

   Choose **Next**.

10. In **Step 4: 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:5.0**.

    f. For **Service role**, choose **New service role**.

    > **Note**
    >
    > Note the name of your CodeBuild service role. You will need the role name for the final step in this tutorial.

    g. Under **Buildspec**, for **Build specifications**, choose **Insert build commands**. Choose **Switch to editor**, and paste the following under **Build commands**. In the buildspec, the customer

variable $CUSTOM_VAR1 will be used to output the pipeline variable in the build log. You will create the $CUSTOM_VAR1 output variable as an environment variable in the following step.

```
version: 0.2
#env:
  #variables:
     # key: "value"
     # key: "value"
  #parameter-store:
     # key: "value"
     # key: "value"
  #git-credential-helper: yes
phases:
  install:
    #If you use the Ubuntu standard image 2.0 or later, you must specify
 runtime-versions.
    #If you specify runtime-versions and use an image other than Ubuntu
 standard image 2.0, the build fails.
    runtime-versions:
      nodejs: 12
    #commands:
      # - command
      # - command
  #pre_build:
    #commands:
      # - command
      # - command
  build:
    commands:
      - echo $CUSTOM_VAR1
  #post_build:
    #commands:
      # - command
      # - command
artifacts:
  files:
     - '*'
    # - location
  name: $(date +%Y-%m-%d)
  #discard-paths: yes
  #base-directory: location
#cache:
```

```
#paths:
  # - paths
```

h. Choose **Continue to CodePipeline**. This returns to the CodePipeline console and creates a CodeBuild project that uses your build commands for configuration. The build project uses a service role to manage AWS service permissions. This step might take a couple of minutes.

i. Under **Environment variables - *optional***, to create an environment variable as an input variable for the build action that will be resolved by the pipeline-level variable, choose **Add environment variable**. This will create the variable specified in the buildspec as `$CUSTOM_VAR1`. In **Name**, enter `CUSTOM_VAR1`. In **Value**, enter `#{variables.timeout}`. In **Type**, choose `Plaintext`.

The `#{variables.timeout}` value for the environment variable is based on the pipeline-level variable namespace `variables` and the pipeline-level variable `timeout` created for the pipeline in step 7.

j. Choose **Next**.

11. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

Choose **Next**.

12. On the **Step 6: Add deploy stage** page, choose **Skip deploy stage**, and then accept the warning message by choosing **Skip** again. Choose **Next**.

13. On **Step 7: Review**, choose **Create pipeline**.

Step 2: Release change and view logs

1. After the pipeline runs successfully, on your successful build stage, choose **View details**.

On the details page, choose the **Logs** tab. View the CodeBuild build output. The commands output the value of the entered variable.

2. In the left-hand nav, choose **History**.

Choose the recent execution, and then choose the **Variables** tab. View the resolved value for the pipeline variable.

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 source 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.

> **Important**
>
> As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

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.

Before you begin, you should complete the prerequisites in Getting started with CodePipeline.

Topics

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

Step 1: Create an S3 source 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 application files 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 console at AWS Management Console. Open the S3 console.
2. Choose **Create bucket**.
3. In **Bucket name**, enter a name for your bucket (for example, **`awscodepipeline-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.

   The sample application contains the following files for deploying with CodeDeploy:

   - `appspec.yml` – The application specification file (AppSpec file) is a YAML-formatted file used by CodeDeploy to manage a deployment. For more information about the AppSpec file, see CodeDeploy AppSpec File reference in the *AWS CodeDeploy User Guide*.
   - `index.html` – The index file contains the home page for the deployed sample application.
   - `LICENSE.txt` – The license file contains license information for the sample application.
   - Files for scripts – The sample application uses scripts to write text files to a location on your instance. One file is written for each of several CodeDeploy deployment lifecycle events as follows:
     - (Linux sample only) `scripts` folder – The folder contains the following shell scripts to install dependencies and start and stop the sample application for the automated deployment: `install_dependencies`, `start_server`, and `stop_server`.
     - (Windows sample only) `before-install.bat` – This is a batch script for the `BeforeInstall` deployment lifecycle event, which will run to remove old files written during previous deployments of this sample and create a location on your instance to which to write the new files.

   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.

c. Choose **Upload**.

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 Amazon EC2 Linux instance and install the CodeDeploy agent. 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 create an instance role with policies that allow install and management of the CodeDeploy agent on the instances. The CodeDeploy agent is a software package that enables an instance to be used in CodeDeploy deployments. You also attach policies that allow the instance to fetch files that the CodeDeploy agent uses to deploy your application and to allow the instance to be managed by SSM.

To create an instance role

1. Open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/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`**.
6. Search for and select the policy named **`AmazonSSMManagedInstanceCore`**. Choose **Next: Tags**.
7. 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://eusc-de-east-1.console.amazonaws-eusc.eu/ec2/.
2. From the side navigation, choose **Instances**, and select **Launch instances** from the top of the page.
3. Under **Name and tags**, in **Name**, enter **`MyCodePipelineDemo`**. This assigns the instances a tag **Key** of **`Name`** and a tag **Value** of **`MyCodePipelineDemo`**. Later, you create a CodeDeploy application that deploys the sample application to the instances. CodeDeploy selects instances to deploy based on the tags.
4. Under **Application and OS Images (Amazon Machine Image)**, choose the **Windows** option. (This AMI is described as the **Microsoft Windows Server 2019 Base** and is labeled "Free tier eligible" and can be found under **Quick Start**.)
5. Under **Instance type**, choose the free tier eligible `t2.micro` type as the hardware configuration for your instance.
6. Under **Key pair (login)**, choose a key pair or create one.

   You can also 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.

7. Under **Network settings**, do the following.

   In **Auto-assign Public IP**, make sure the status is **Enable**.

   - Next to **Assign a security group**, choose **Create a new security group**.

- In the row for **SSH**, under **Source type**, choose **My IP**.
- Choose **Add security group**, choose **HTTP**, and then under **Source type**, choose **My IP**.

8. Expand **Advanced details**. In **IAM instance profile**, choose the IAM role you created in the previous procedure (for example, **`EC2InstanceRole`**).
9. Under **Summary**, under **Number of instances**, enter 2.
10. Choose **Launch instance**.
11. Choose **View all 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. Check that your instance has passed its status checks. You can view this information in the **Status Checks** column.

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.

You first create a service role for CodeDeploy to use. If you have already created a service role, you do not need to create another one.

To create a CodeDeploy service role

1. Open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. From the console dashboard, choose **Roles**.
3. Choose **Create role**.
4. Under **Select trusted entity**, choose **AWS service**. Under **Use case**, choose **CodeDeploy**. Choose **CodeDeploy** from the options listed. Choose **Next**. The `AWSCodeDeployRole` managed policy is already attached to the role.
5. Choose **Next**.

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

To create an application in CodeDeploy

1. Open the CodeDeploy console at https://console.aws.amazon.com/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

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. You must use 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.

6. Under **Agent configuration with AWS Systems Manager**, choose **Now and schedule updates**. This installs the agent on the instance. The Windows instance is already configured with the SSM agent and will now be updated with the CodeDeploy agent.
7. Under **Deployment settings**, choose `CodeDeployDefault.OneAtaTime`.

8. Under **Load Balancer**, make sure the **Enable load balancing** box is not selected. You do not need to set up a load balancer or choose a target group for this example. After you de-select the checkbox, the load balancer options do not display.
9. In the **Advanced** section, leave the defaults.
10. 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

1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home.
2. On the **Welcome** page, **Getting started** page, or the **Pipelines** page, choose **Create pipeline**.
3. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. In **Step 2: 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.

5. CodePipeline provides V1 and V2 type pipelines, which differ in characteristics and price. The V2 type is the only type you can choose in the console. For more information, see pipeline types. For information about pricing for CodePipeline, see Pricing.
6. In **Service role**, do one of the following:

   - Choose **New service role** to allow CodePipeline to create a new service role in IAM.
   - Choose **Existing service role** to use a service role already created in IAM. In **Role name**, choose your service role from the list.

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

8. In **Step 3: 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 source bucket for your application. 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**.

9. In **Step 4: Add build stage**, choose **Skip build stage**, and then accept the warning message by choosing **Skip** again. Choose **Next**.

10. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

    Choose **Next**.

11. In **Step 6: Add deploy stage**, in **Deploy provider**, choose **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**.

    > **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.

12. In **Step 7: Review**, review the information, and then choose **Create pipeline**.

13. 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 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 web page displays for the sample application you uploaded to your S3 bucket.

For more information about stages, actions, and how pipelines work, see CodePipeline concepts.

(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
- Add the deployment group as another stage in your pipeline

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

1. Open the CodeDeploy console at https://console.aws.amazon.com/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**.
8. Under **Deployment configuration**, choose `CodeDeployDefault.OneAtaTime`.
9. Under **Load Balancer**, clear **Enable load balancing**.
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.

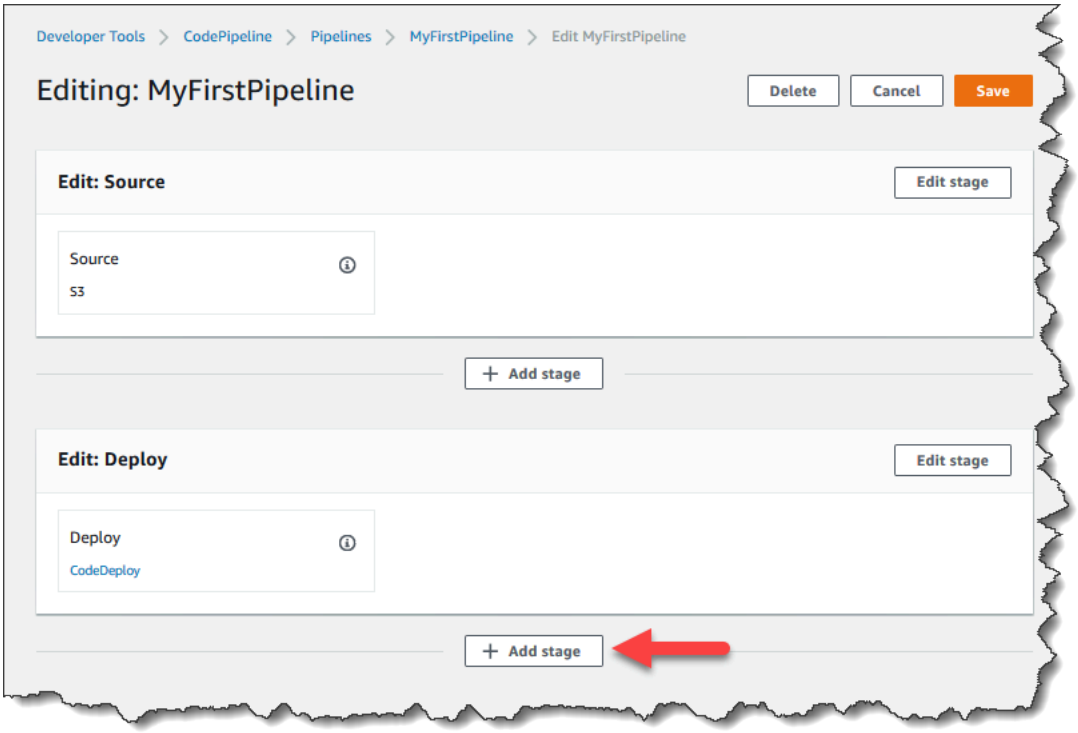
Topics

- Create a third stage (console)
- Create a third stage (CLI)

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.

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, 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 **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:

```
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:

```
{
    "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:

```
    ...
        ],
        "artifactStore": {
            "type": "S3"
            "location": "amzn-s3-demo-bucket",
        },
        "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:
>
> ```
> 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.

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

5. 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.

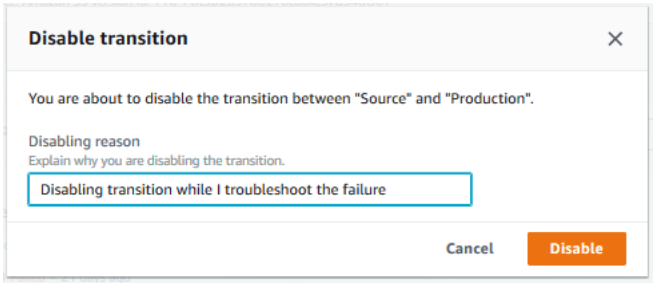
(Optional) Step 6: Disable and enable transitions between stages in CodePipeline

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.

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. 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 Amazon 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.
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.

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.

> **Important**
>
> As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

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

1. Open the CodeCommit console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codecommit/.
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.

> **Note**
>
> You are not required to set up a local repository. You can also use the console to upload files as described in Step 2: Add sample code to your 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. Next, download a sample and save it into a folder or directory on your local computer.

   a. Choose one of the following. Choose `SampleApp_Linux.zip` if you want to follow the steps in this tutorial for Linux 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.

   The sample application contains the following files for deploying with CodeDeploy:

   - `appspec.yml` – The application specification file (AppSpec file) is a YAML-formatted file used by CodeDeploy to manage a deployment. For more information about the AppSpec file, see CodeDeploy AppSpec File reference in the *AWS CodeDeploy User Guide*.
   - `index.html` – The index file contains the home page for the deployed sample application.
   - `LICENSE.txt` – The license file contains license information for the sample application.
   - Files for scripts – The sample application uses scripts to write text files to a location on your instance. One file is written for each of several CodeDeploy deployment lifecycle events as follows:

- (Linux sample only) `scripts` folder – The folder contains the following shell scripts to install dependencies and start and stop the sample application for the automated deployment: `install_dependencies`, `start_server`, and `stop_server`.
- (Windows sample only) `before-install.bat` – This is a batch script for the `BeforeInstall` deployment lifecycle event, which will run to remove old files written during previous deployments of this sample and create a location on your instance to which to write the new files.

b. Download the compressed (zipped) file.

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. To upload files to your repository, use one of the following methods.

   a. 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. To add a file under a folder, choose **Create file** and then enter the folder name with the file name, such as `scripts/install_dependencies`. Paste the file contents into the new file.

      Commit the change by entering your user name and email address.

      Choose **Commit changes**.

iv. Repeat this step for each file.

Your repository contents should look like this:

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

b. To use git commands to upload your files:

i. Change directories to your local repo:

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

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 "Add sample application files"
```

iv. Run the following command to push the files from your local repo to your CodeCommit repository:

```
git push
```

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

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

In this step, you create the Amazon EC2 instance where you deploy a sample application. As part of this process, create an instance role that allows install and management of the CodeDeploy agent on the instance. The CodeDeploy agent is a software package that enables an instance to be used in CodeDeploy deployments. You also attach policies that allow the instance to fetch files that the CodeDeploy agent uses to deploy your application and to allow the instance to be managed by SSM.

To create an instance role

1. Open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/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**. Under **Select your use case**, choose **EC2**. Choose **Next: Permissions**.
5. Search for and select the policy named **`AmazonEC2RoleforAWSCodeDeploy`**.
6. Search for and select the policy named **`AmazonSSMManagedInstanceCore`**. Choose **Next: Tags**.
7. 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://eusc-de-east-1.console.amazonaws-eusc.eu/ec2/.
2. From the side navigation, choose **Instances**, and select **Launch instances** from the top of the page.

3. In **Name**, enter **`MyCodePipelineDemo`**. This assigns the instance a tag **Key** of **`Name`** and a tag **Value** of **`MyCodePipelineDemo`**. Later, you create a CodeDeploy application that deploys the sample application to this instance. CodeDeploy selects instances to deploy based on the tags.
4. Under **Application and OS Images (Amazon Machine Image)**, locate the **Amazon Linux** AMI option with the AWS logo, and make sure it is selected. (This AMI is described as the Amazon Linux 2 AMI (HVM) and is labeled "Free tier eligible".)
5. Under **Instance type**, choose the free tier eligible `t2.micro` type as the hardware configuration for your instance.
6. Under **Key pair (login)**, choose a key pair or create one.

   You can also 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.

7. Under **Network settings**, do the following.

   In **Auto-assign Public IP**, make sure the status is **Enable**.

   For the created security group, choose **HTTP**, and then under **Source type**, choose **My IP**.
8. Expand **Advanced details**. In **IAM instance profile**, choose the IAM role you created in the previous procedure (for example, **`EC2InstanceRole`**).
9. Under **Summary**, under **Number of instances**, enter 1.
10. Choose **Launch instance**.
11. 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**.)

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://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. From the console dashboard, choose **Roles**.
3. Choose **Create role**.
4. Under **Select trusted entity**, choose **AWS service**. Under **Use case**, choose **CodeDeploy**. Choose **CodeDeploy** from the options listed. Choose **Next**. The `AWSCodeDeployRole` managed policy is already attached to the role.
5. Choose **Next**.
6. Enter a name for the role (for example, **`CodeDeployRole`**), and then choose **Create role**.

To create an application in CodeDeploy

1. Open the CodeDeploy console at https://console.aws.amazon.com/codedeploy.
2. If the **Applications** page does not appear, on the 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 ARN of the service role you created earlier (for example, **`arn:aws:iam::`*`account_ID`*`:role/CodeDeployRole`**).
4. Under **Deployment type**, choose **In-place**.

5. Under **Environment configuration**, choose **Amazon EC2 Instances**. In the **Key** field, enter **`Name`**. In the **Value** field, enter the name you used to tag the instance (for example, **`MyCodePipelineDemo`**).
6. Under **Agent configuration with AWS Systems Manager**, choose **Now and schedule updates**. This installs the agent on the instance. The Linux instance is already configured with the SSM agent and will now be updated with the CodeDeploy agent.
7. Under **Deployment configuration**, choose `CodeDeployDefault.OneAtaTime`.
8. Under **Load Balancer**, make sure **Enable load balancing** is not selected. You do not need to set up a load balancer or choose a target group for this example.
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

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

   Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
2. On the **Welcome** page, **Getting started** page, or the **Pipelines** page, choose **Create pipeline**.
3. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. In **Step 2: Choose pipeline settings**, in **Pipeline name**, enter **`MyFirstPipeline`**.
5. CodePipeline provides V1 and V2 type pipelines, which differ in characteristics and price. The V2 type is the only type you can choose in the console. For more information, see pipeline types. For information about pricing for CodePipeline, see Pricing.
6. In **Service role**, choose **New service role** to allow CodePipeline to create a service role in IAM.
7. Leave the settings under **Advanced settings** at their defaults, and then choose **Next**.
8. In **Step 3: Add source stage**, in **Source provider**, choose **CodeCommit**. In **Repository name**, choose the name of the CodeCommit repository you created in Step 1: Create a CodeCommit repository. In **Branch name**, choose `main`, 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**.

9. In **Step 4: 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.

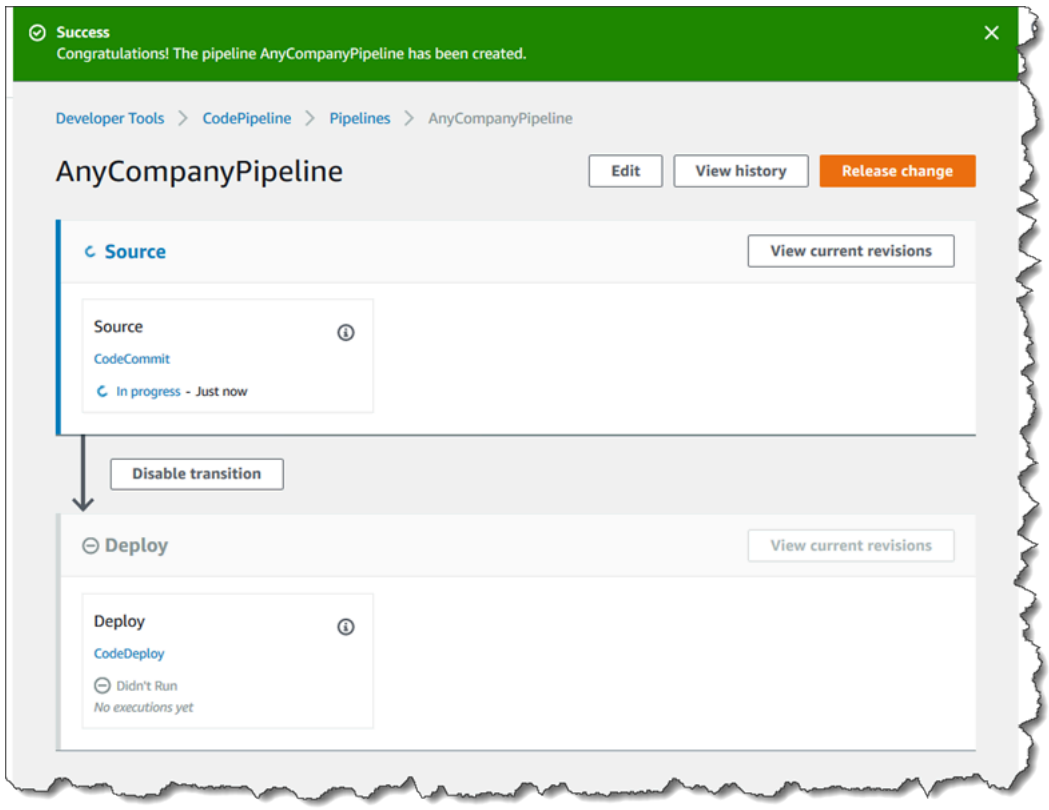
10. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

    Choose **Next**.

11. In **Step 6: Add deploy stage**, in **Deploy provider**, choose **CodeDeploy**. In **Application name**, choose **`MyDemoApplication`**. In **Deployment group**, choose **`MyDemoDeploymentGroup`**, and then choose **Next step**.

12. In **Step 7: Review**, review the information, and then choose **Create pipeline**.

13. 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 **CodeDeploy**. This opens the CodeDeploy console. If **Succeeded** is not displayed, see Troubleshooting CodePipeline.
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.

The web page displays for the sample application you downloaded and pushed to your CodeCommit repository.

For more information about stages, actions, and how pipelines work, see CodePipeline concepts.

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.

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

    h1 {
      font-size: 250%;
```

```
      font-weight: normal;
      margin-bottom: 0;
    }

    h2 {
      font-size: 175%;
      font-weight: normal;
      margin-bottom: 0;
    }
  </style>
</head>
<body>
  <div align="center"><h1>Updated Sample Deployment</h1></div>
  <div align="center"><h2>This application was updated using CodePipeline,
 CodeCommit, and CodeDeploy.</h2></div>
  <div align="center">
    <p>Learn more:</p>
    <p><a href="https://docs.aws.eu/codepipeline/latest/userguide/">CodePipeline
 User Guide</a></p>
    <p><a href="https://docs.aws.eu/codecommit/latest/userguide/">CodeCommit User
 Guide</a></p>
    <p><a href="https://docs.aws.eu/codedeploy/latest/userguide/">CodeDeploy User
 Guide</a></p>
  </div>
</body>
</html>
```

4. Commit and push your changes to your CodeCommit repository by running the following commands, one at a time:

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

```
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.
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.

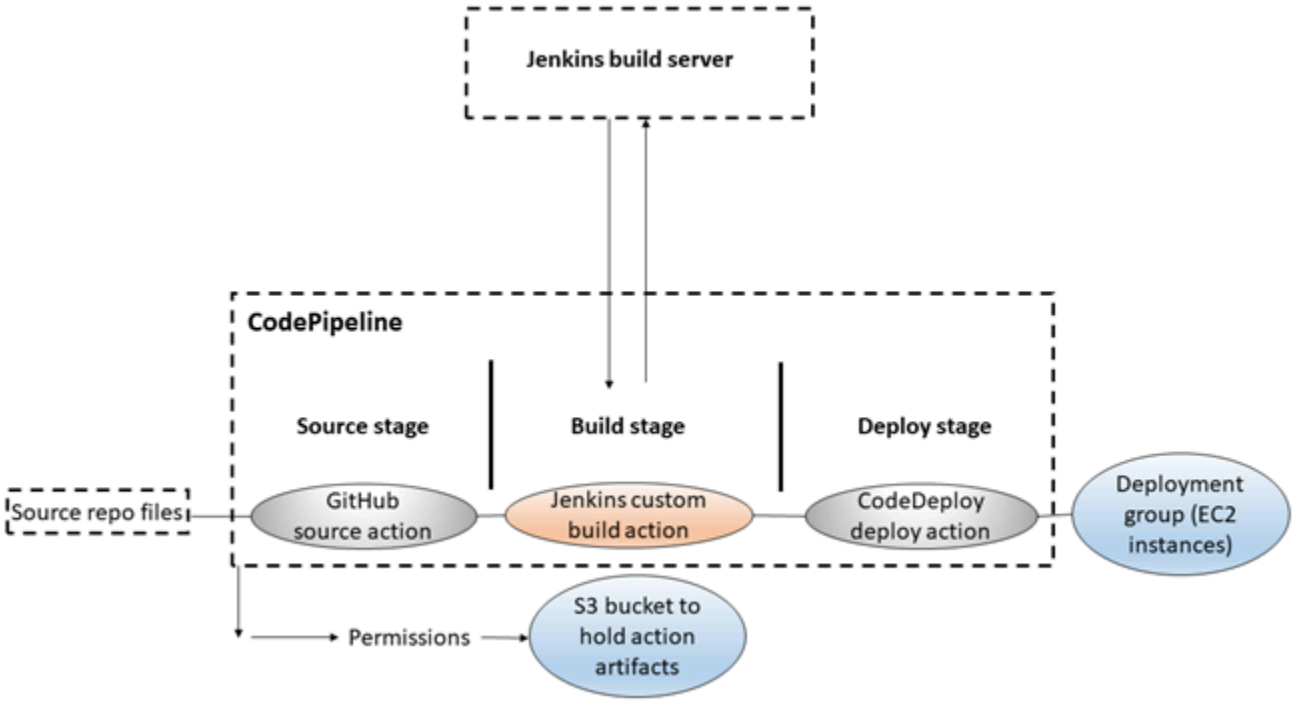
Step 8: Further reading

Learn more about how CodePipeline works:

- For more information about stages, actions, and how pipelines work, see CodePipeline concepts.
- For information about the actions you can perform using CodePipeline, see Integrations with CodePipeline action types.
- Try this more advanced tutorial, Tutorial: Create a four-stage pipeline. 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) or Tutorial: Create a simple pipeline (CodeCommit repository), 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. The following diagram shows the initial three-stage pipeline.



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.

Important

As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

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

Topics

- Step 1: Complete prerequisites
- Step 2: Create a pipeline in CodePipeline
- Step 3: Add another stage to your pipeline
- Step 4: Clean up resources

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 or role 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. 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
- Create an IAM role to use for Jenkins integration
- Install and configure Jenkins and the CodePipeline Plugin for Jenkins

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 codepipeline-jenkins-aws-codedeploy_linux.zip.
   - If you will be deploying your sample to Windows Server instances, choose CodePipeline-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://eusc-de-east-1.console.amazonaws-eusc.eu/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**.
4. Choose **Next: Permissions**. On the **Attach permissions policies** page, select the `AWSCodePipelineCustomActionAccess` managed policy, and then choose **Next: Tags**. Choose **Next: Review**.
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.

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 in Product and service integrations with CodePipeline.
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.

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.

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:

```
rake
```

> **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

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

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 creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
5. On the **Step 2: Choose pipeline settings** page, in **Pipeline name**, enter the name for your pipeline.
6. CodePipeline provides V1 and V2 type pipelines, which differ in characteristics and price. The V2 type is the only type you can choose in the console. For more information, see pipeline types. For information about pricing for CodePipeline, see Pricing.
7. In **Service role**, choose **New service role** to allow CodePipeline to create a service role in IAM.
8. Leave the settings under **Advanced settings** at their defaults, and choose **Next**.
9. On the **Step 3: Add source stage** page, in **Source provider**, choose **GitHub**.
10. Under **Connection**, choose an existing connection or create a new one. To create or manage a connection for your GitHub source action, see GitHub connections.
11. In **Step 4: 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**.
12. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

    Choose **Next**.

13. In **Step 6: Add deploy stage**, reuse the CodeDeploy application and deployment group you created in Tutorial: Create a simple pipeline (S3 bucket). 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.

14. In **Step 7: Review**, review the information, and then choose **Create pipeline**.
15. 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 as 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
- Create a Jenkins project for testing the deployment
- Create a fourth stage

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.

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 or role 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.

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.
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 Monitoring CodePipeline events.

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

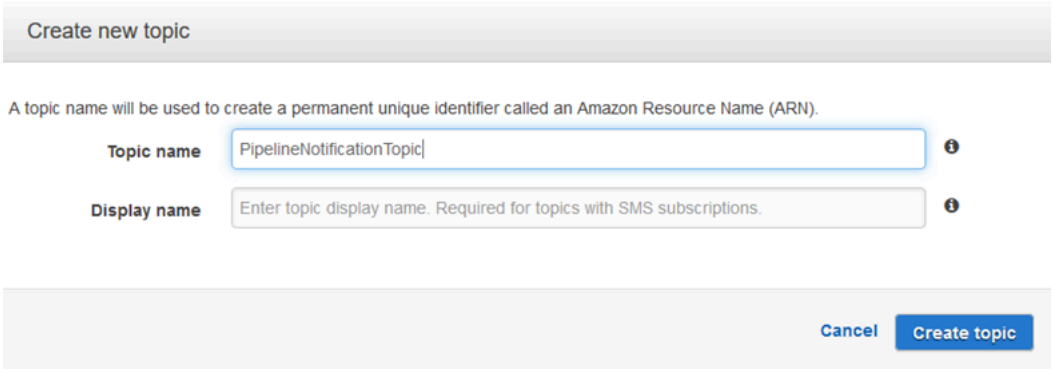
- Step 1: Set up an email notification using Amazon SNS
- Step 2: Create a rule and add the SNS topic as the target
- Step 3: Clean up resources

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://eusc-de-east-1.console.amazonaws-eusc.eu/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.
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.

1. Open the CloudWatch console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudwatch/.
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:

   ```
   {
     "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.

   ```
   { "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 with AWS Device Farm

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. 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 GitHub commit and then uses CodeBuild to build the app and Device Farm to test it.

> **Important**
>
> As part of creating a pipeline in the console, an S3 artifact bucket will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline,

make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

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.

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.

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-9afc-4c1cfexample`.

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

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.

```
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://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.

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

   c. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.

   d. On the **Step 2: Choose pipeline settings** page, in **Pipeline name**, enter the name for your pipeline.

   e. CodePipeline provides V1 and V2 type pipelines, which differ in characteristics and price. The V2 type is the only type you can choose in the console. For more information, see pipeline types. For information about pricing for CodePipeline, see Pricing.

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

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

h. On the **Step 3: Add source stage** page, in **Source provider**, choose **GitHub (via GitHub App)**.

i. Under **Connection**, choose an existing connection or create a new one. To create or manage a connection for your GitHub source action, see GitHub connections.

j. In **Repository**, choose the source repository.

k. In **Branch**, choose the branch that you want to use.

l. Leave the remaining defaults for the source action. Choose **Next**.

4. In **Step 4: Add build stage**, add a build stage:

a. In **Build provider**, choose **Other build providers**, and then 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:5.0**.

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. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

   Choose **Next**.

6. On the **Step 6: Add deploy stage** page, choose **Skip deploy stage**, and then accept the warning message by choosing **Skip** again. Choose **Next**.

7. On **Step 7: Review**, choose **Create pipeline**. You should see a diagram that shows the source and build stages.

8. 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. To get the available device pool ARNs for the project, including the ARN for Top Devices, use the AWS CLI to enter the following command:

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

i. In **AppType**, enter **Android**.

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

> **Note**
>
> 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 with AWS Device Farm

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**
>
> As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

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.

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

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-9afc-4c1cfexample`.

```
https://<region-URL>/devicefarm/home?region=us-west-2#/projects/
eec4905f-98f8-40aa-9afc-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 source bucket for your application 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://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.

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

   c. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.

   d. On the **Step 2: Choose pipeline settings** page, in **Pipeline name**, enter the name for your pipeline.

   e. CodePipeline provides V1 and V2 type pipelines, which differ in characteristics and price. The V2 type is the only type you can choose in the console. For more information, see pipeline types. For information about pricing for CodePipeline, see Pricing.

   f. 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.

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

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

   h. On the **Step 3: Add source stage** page, in **Source provider**, choose **Amazon S3**.

i. In **Amazon S3 location**, enter the bucket, such as `my-storage-bucket`, and object key, such as `s3-ios-test-1.zip` for your .zip file.

j. Choose **Next**.

4. In **Step 4: Add build stage**, 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.

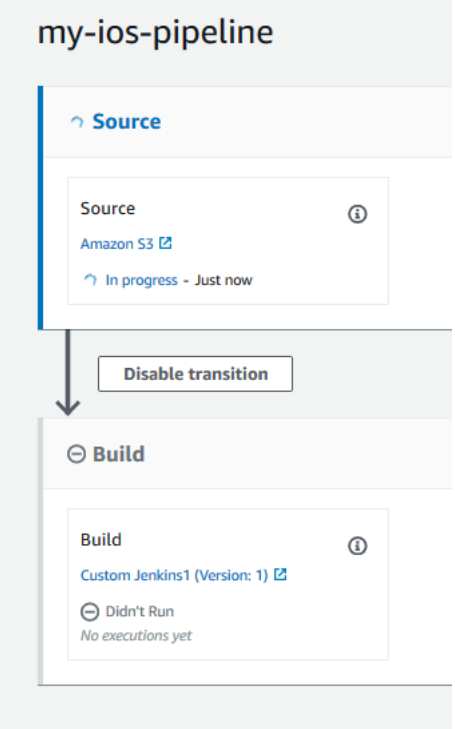
   e. Choose **Next**.

   f. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

      Choose **Next**.

   g. On the **Step 6: Add deploy stage** page, choose **Skip deploy stage**, and then accept the warning message by choosing **Skip** again.

   h. On **Step 7: 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. To get the available device pool ARNs for the project, including the ARN for Top Devices, use the AWS CLI to enter the following command:

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

k. In **AppType**, enter **iOS**.

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

- **iOS**
- **Android**
- **Web**

l. 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`.

m. 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.

If you're using one of the built-in Device Farm tests, enter the type of test configured in your Device Farm project, such as BUILTIN_FUZZ. In **FuzzEventCount**, enter a time in milliseconds, such as 6000. In **FuzzEventThrottle**, enter a time in milliseconds, such as 50.

If you aren't using one of the built-in Device Farm tests, 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_NODE**
- **APPIUM_RUBY**
- **APPIUM_PYTHON**
- **APPIUM_WEB_JAVA_JUNIT**
- **APPIUM_WEB_JAVA_TESTNG**
- **APPIUM_WEB_NODE**

- **APPIUM_WEB_RUBY**
- **APPIUM_WEB_PYTHON**
- **BUILTIN_FUZZ**
- **INSTRUMENTATION**
- **XCTEST**
- **XCTEST_UI**

> **Note**
>
> 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 Service Catalog

Service Catalog enables you to create and provision products based on AWS CloudFormation templates.

> **Important**
>
> As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

This tutorial shows you how to create and configure a pipeline to deploy your product template to 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 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 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 Service Catalog without a configuration file.

  > **Note**
  >
  > The template file can be created in YAML or JSON format.

- Create a product in Service Catalog and upload a template file to your source repository. Provide product version and deployment configuration in a separate configuration file. See Option 2: Deploy to Service Catalog using a configuration file.

Option 1: Deploy to 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 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`.

   ```
   {
     "AWSTemplateFormatVersion": "2010-09-09",
   ```

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

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

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

Step 2: Create a product in Service Catalog

1. As an IT administrator, sign in to the 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. Copy and retain this 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:

   a. Sign in to the AWS Management Console and open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.

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

   c. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.

d. In **Step 2: Choose pipeline settings**, in **Pipeline name**, enter a name for your pipeline.

e. CodePipeline provides V1 and V2 type pipelines, which differ in characteristics and price. The V2 type is the only type you can choose in the console. For more information, see pipeline types. For information about pricing for CodePipeline, see Pricing.

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

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

2. To add a source stage on the **Step 3: Add source stage** page, 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 **Step 4: Add build stage**, choose **Skip build stage**, and then accept the warning message by choosing **Skip** again.

4. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

   Choose **Next**.

5. In **Step 6: Add deploy stage**, complete the following:

a. In **Deploy provider**, choose **AWS Service Catalog**.

b. For deployment configuration, choose **Enter deployment configuration**.

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

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

e. In **Product type**, choose **CloudFormation template**.

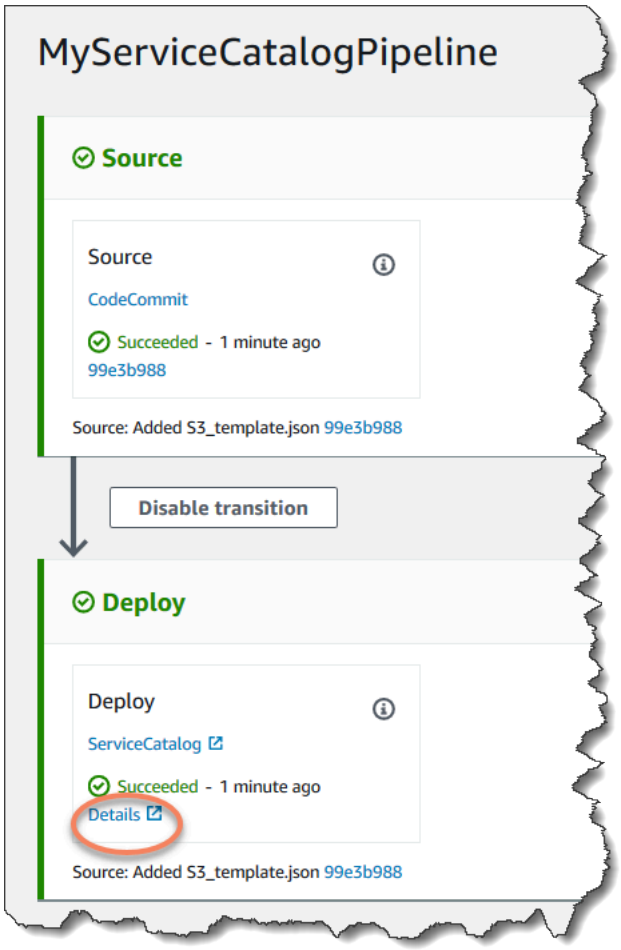
f. In **Product version name**, enter the name of the product version you specified in Service Catalog. If you want to have the template change deployed to a new product version, enter a product version name that has not been used for any previous product version in the same product.

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

h. Choose **Next**.

6. In **Step 7: Review**, review your pipeline settings, and then choose **Create**.

7. After your pipeline runs successfully, on the deployment stage, choose **Details**. This opens your product in Service Catalog.



8. 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 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 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 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 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`.

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

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

   ```
   {
       "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 Service Catalog

1. As an IT administrator, sign in to the 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 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 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://eusc-de-east-1.console.amazonaws-eusc.eu/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**.

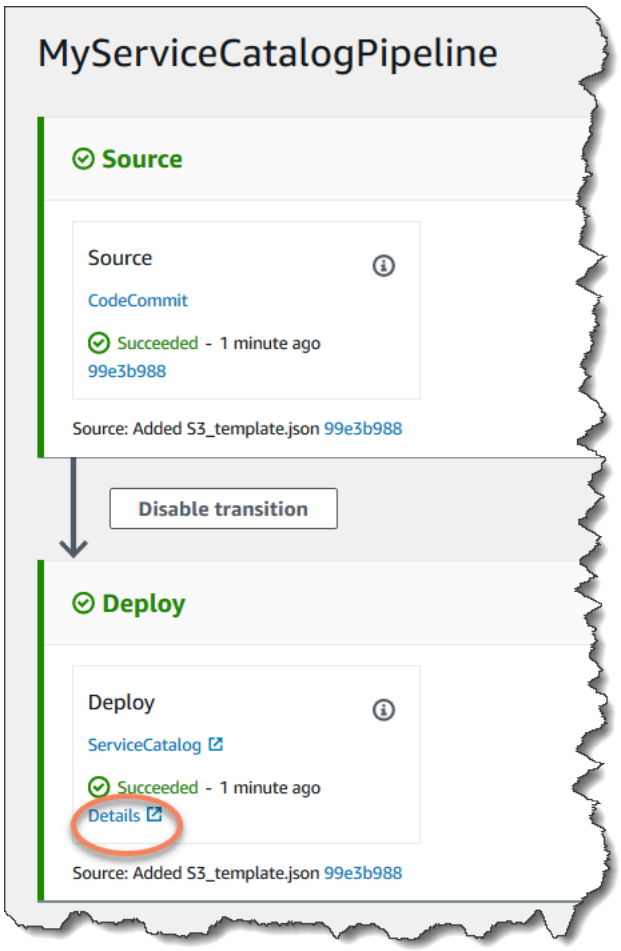
c. In **Product ID**, paste the product ID you copied from the Service Catalog console.

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 Service Catalog.



7. Under your product information, choose your version name to open the product template. View the template deployment.

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 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.

> **Important**
>
> As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

Topics

- Example 1: Create an AWS CodeCommit pipeline with AWS CloudFormation
- Example 2: Create an Amazon S3 pipeline with AWS CloudFormation

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:

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

Create your pipeline in AWS CloudFormation

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 to push the files to your repository.
2. Open the AWS CloudFormation console and choose **Create Stack**. Choose **With new resources (standard)**.
3. Under **Specify template**, choose **Upload a template**. Select **Choose file** and then choose 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 stack**.
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 policy to allow AWS CloudFormation to create the required Amazon CloudWatch Events resources for the CodeCommit pipeline:

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

8. Sign in to the AWS Management Console and open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.

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

   > **Note**
   >
   > To view the pipeline that was created, find the **Logical ID** column under the **Resources** tab for your stack in CloudFormation. Note the name in the **Physical ID** column for the pipeline. In CodePipeline, you can view the pipeline with the same Physical ID (pipeline name) in the Region where you created your stack.

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).
- 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.

  Save the .zip file on your local computer. You upload the .zip file after the stack is created.

Create your pipeline in AWS CloudFormation

1. Open the AWS CloudFormation console, and choose **Create Stack**. Choose **With new resources (standard)**.
2. In **Choose a template**, choose **Upload a template**. Select **Choose file**, and then choose 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 stack**.
6. After your stack creation is complete, view the event list to check for any errors.

   Troubleshooting

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

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

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

   > **Note**
   >
   > To view the pipeline that was created, find the **Logical ID** column under the **Resources** tab for your stack in CloudFormation. Note the name in the **Physical ID** column for the pipeline. In CodePipeline, you can view the pipeline with the same Physical ID (pipeline name) in the Region where you created 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 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.

8. Sign in to the AWS Management Console and open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.

   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 policy to allow AWS CloudFormation to create the required CloudTrail resources for the Amazon S3 pipeline:

```
{
     "Effect": "Allow",
     "Action": [
```

```
            "cloudtrail:CreateTrail",
            "cloudtrail:DeleteTrail",
            "cloudtrail:StartLogging",
            "cloudtrail:StopLogging",
            "cloudtrail:PutEventSelectors"
        ],
        "Resource": "resource_ARN"
}
```

7. Sign in to the AWS Management Console and open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/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.

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.

> **Important**
>
> As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

Topics

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

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).
- 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 for your Region. For example, use the table at https://docs.aws.amazon.com/documentdb/latest/developerguide/quick_start_cfn.html#quick_start_cfn-launch_stack to choose the Region and download

the template. Download the template for an Amazon DocumentDB Cluster. The file name is `documentdb_full_stack.yaml`.

2. Unzip the `documentdb_full_stack.yaml` 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
     Description: The purpose of this instance.
   ```

   b. For this example, add the following `StackName` output to your `Outputs:` section in the template.

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

3. Upload the template file to your AWS CodeCommit repository. You must upload the unzipped and edited template file to the root directory of your repository.

   To use the CodeCommit console to upload your files:

   a. Open the CodeCommit console, and choose your repository from the **Repositories** list.

   b. Choose **Add file**, and then choose **Upload file**.

   c. Select **Choose file**, and then browse for your file. Commit the change by entering your user name and email address. Choose **Commit changes**.

   Your file should look like this at the root level in your repository:

   ```
   documentdb_full_stack.yaml
   ```

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 a 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 reference.

To create a pipeline with the wizard

1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home.
2. On the **Welcome** page, **Getting started** page, or the **Pipelines** page, choose **Create pipeline**.
3. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. In **Step 2: Choose pipeline settings**, in **Pipeline name**, enter **`MyCFNDeployPipeline`**.
5. CodePipeline provides V1 and V2 type pipelines, which differ in characteristics and price. The V2 type is the only type you can choose in the console. For more information, see pipeline types. For information about pricing for CodePipeline, see Pricing.
6. In **Service role**, do one of the following:
   - Choose **New service role** to allow CodePipeline to create a service role in IAM.
   - Choose **Existing service role**. In **Role name**, choose your service role from the list.
7. In **Artifact store**:
   a. 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 and CodePipeline pipeline structure reference.

Choose **Next**.

8. In **Step 3: 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.

   c. In **Branch name**, choose the name of the branch that contains your latest code update.

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

   Choose **Next**.

9. In **Step 4: Add build stage**, choose **Skip build stage**, and then accept the warning message by choosing **Skip** again.

   Choose **Next**.

10. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

    Choose **Next**.

11. In **Step 6: 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.

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

If you don't enter overrides, the template creates a stack with default values.

f. Choose **Next**.

g. On **Step 7: Review**, choose **Create pipeline**. You should see a diagram that shows the pipeline stages. Allow your pipeline to run. Your two-stage pipeline is complete and ready for the additional stages to be added.

Step 3: Add a CloudFormation deployment action to create the change set

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

1. Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.

   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. Choose to edit the **Deploy** stage.

4. Add a deployment action that will create a change set for the stack that was created in the previous action. You add this action after the existing action in the stage.

   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.

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

   e. In **Change set name**, enter the name of the change set.

   ```
   my-changeset
   ```

   f. In **Parameter Overrides**, change the `Purpose` parameter from `testing` to `production`.

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

   g. Choose **Done** to save 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. Choose to edit the **Deploy** stage.
3. 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 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 CloudFormation to execute the change set after the manual approval action.

1. Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.

   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. Choose to edit the **Deploy** stage.
4. Add a deployment action that will execute the change set that was approved in the previous manual 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.

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

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

   ```
   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 CloudFormation to get the stack name from the variable in the outputs file and delete the stack.

1. Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.

   Under **Pipelines**, choose your pipeline and choose **View**. The diagram shows your pipeline source and deployment stages.
2. Choose to edit the pipeline.
3. Choose to edit the **Deploy** stage.
4. 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.

   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.

Important

As part of creating a pipeline in the console, an S3 artifact bucket will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

Note

This tutorial is for the Amazon ECS standard deployment action for CodePipeline. For a tutorial that uses the Amazon ECS to CodeDeploy blue/green deployment action in

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

Note

This tutorial is for the Amazon ECS standard deployment action for CodePipeline with a source action. For a tutorial that uses the Amazon ECS standard deployment action along with the ECRBuildAndPublish build action in CodePipeline to push your image, see Tutorial: Build and push a Docker image to Amazon ECR with CodePipeline (V2 type).

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.

- 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*.

Important

The Amazon ECS standard deployment action for CodePipeline creates its own revision of the task definition based on the revision used by the Amazon ECS service. If you create

new revisions for the task definition without updating the Amazon ECS service, the deployment action will ignore those revisions.

Below is a sample task definition used for this tutorial. The value you use for `name` and `family` will be used in the next step for your build specification file.

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

```
      "memory": null,
      "memoryReservation": 128,
      "volumesFrom": [],
      "stopTimeout": null,
      "image": "image_name",
      "startTimeout": null,
      "firelensConfiguration": null,
      "dependsOn": null,
      "disableNetworking": null,
      "interactive": null,
      "healthCheck": null,
      "essential": true,
      "links": null,
      "hostname": null,
      "extraHosts": null,
      "pseudoTerminal": null,
      "user": null,
      "readonlyRootFilesystem": null,
      "dockerLabels": null,
      "systemControls": null,
      "privileged": null,
      "name": "hello-world"
    }
  ],
  "placementConstraints": [],
  "memory": "2048",
  "taskRoleArn": null,
  "compatibilities": [
    "EC2",
    "FARGATE"
  ],
  "taskDefinitionArn": "ARN",
  "family": "hello-world",
  "requiresAttributes": [],
  "pidMode": null,
  "requiresCompatibilities": [
    "FARGATE"
  ],
  "networkMode": "awsvpc",
  "cpu": "1024",
  "revision": 1,
  "status": "ACTIVE",
  "inferenceAccelerators": null,
  "proxyConfiguration": null,
```

```
  "volumes": []
}
```

- 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*.

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

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.

Paste this sample text to create your `buildspec.yml` file, and replace the values for your image and task definition. This text uses the example account ID 111122223333.

```
version: 0.2

phases:
  pre_build:
```

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

The build specification was written for the sample task definition that was provided in Prerequisites, 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.

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.

```
git push
```

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

1. Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
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: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. In **Step 2: Choose pipeline settings**, in **Pipeline name**, type the name for your pipeline. For this tutorial, the pipeline name is **hello-world**.
5. In **Pipeline type**, keep the default selection at **V2**. Pipeline types differ in characteristics and price. For more information, see Pipeline types. Choose **Next**.
6. On the **Step 3: 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**.
7. On the **Step 4: 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:3.0`**.

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**. You might need to expand **Advanced**.

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.

8. On the **Step 5: 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**.

9. On the **Step 6: Review** page, review your pipeline configuration and choose **Create pipeline** to create the pipeline.

> **Note**
>
> Now that the pipeline has been created, it attempts to run through the different pipeline stages. However, the default CodeBuild role created by the wizard does not have permissions to execute all of the commands contained in the `buildspec.yml` file, so the build stage fails. The next section adds the permissions for the build stage.

Step 3: Add Amazon ECR Permissions to the CodeBuild Role

The CodePipeline wizard created an IAM role for the CodeBuild build project, called **codebuild-*`build-project-name`*-service-role**. For this tutorial, the name is **codebuild-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

1. Open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
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.
2. Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
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 roll back if there is an issue.

Important

As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

Note

This tutorial is for the Amazon ECS to CodeDeploy blue/green deployment action for CodePipeline. For a tutorial that uses the Amazon ECS standard deployment action in CodePipeline, see Tutorial: Amazon ECS Standard Deployment with CodePipeline.

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

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
- Step 1: Create image and push to an Amazon ECR repository
- Step 2: Create task definition and AppSpec source files and push to a CodeCommit repository
- Step 3: Create your Application Load Balancer and target groups
- Step 4: Create your Amazon ECS cluster and service
- Step 5: Create your CodeDeploy application and deployment group (ECS compute platform)
- Step 6: Create your pipeline
- Step 7: Make a change to your pipeline and verify deployment

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).
- Launch an Amazon EC2 Linux instance and install Docker to create an image as shown in this tutorial. If you already have an image you want to use, you can skip this prerequisite.

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:

   ```
   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 and the 111122223333 account ID.

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-west-2.amazonaws.com/nginx
```

4. Push the image to Amazon ECR using the `repositoryUri` from the earlier step.

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/nginx:latest
```

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.

> **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*.

```
{
    "executionRoleArn": "arn:aws:iam::account_ID:role/ecsTaskExecutionRole",
    "containerDefinitions": [
        {
            "name": "sample-website",
            "image": "nginx",
            "essential": true,
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ]
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "networkMode": "awsvpc",
    "cpu": "256",
    "memory": "512",
    "family": "ecs-demo"
}
```

2. Register your task definition with the `taskdef.json` file.

```
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.

```
{
```

```
    "executionRoleArn": "arn:aws:iam::account_ID:role/ecsTaskExecutionRole",
    "containerDefinitions": [
        {
            "name": "sample-website",
            "image": "<IMAGE1_NAME>",
            "essential": true,
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ]
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "networkMode": "awsvpc",
    "cpu": "256",
    "memory": "512",
    "family": "ecs-demo"
}
```

To create an AppSpec file

- The AppSpec file is used for CodeDeploy deployments. The file, which includes optional fields, uses this format:

```
version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: "task-definition-ARN"
        LoadBalancerInfo:
          ContainerName: "container-name"
          ContainerPort: container-port-number
# Optional properties
        PlatformVersion: "LATEST"
        NetworkConfiguration:
            AwsvpcConfiguration:
```

```
                Subnets: ["subnet-name-1", "subnet-name-2"]
                SecurityGroups: ["security-group"]
                AssignPublicIp: "ENABLED"
Hooks:
- BeforeInstall: "BeforeInstallHookFunctionName"
- AfterInstall: "AfterInstallHookFunctionName"
- AfterAllowTestTraffic: "AfterAllowTestTrafficHookFunctionName"
- BeforeAllowTraffic: "BeforeAllowTrafficHookFunctionName"
- AfterAllowTraffic: "AfterAllowTrafficHookFunctionName"
```

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.

```
version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: <TASK_DEFINITION>
        LoadBalancerInfo:
          ContainerName: "sample-website"
          ContainerPort: 80
```

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:

```
/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.

To verify your default VPC and public subnets

1. Sign in to the AWS Management Console and open the Amazon VPC console at https://eusc-de-east-1.console.amazonaws-eusc.eu/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://eusc-de-east-1.console.amazonaws-eusc.eu/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:

   a. Under **Load Balancer Protocol**, choose **HTTP**. Under **Load Balancer Port**, enter **80**.

   b. Choose **Add listener**.

   c. Under **Load Balancer Protocol** for the second listener, choose **HTTP**. Under **Load Balancer Port**, enter **8080**.

9. Under **Availability Zones**, in **VPC**, choose the default VPC. Next, choose the two default subnets you want to use.
10. Choose **Next: Configure Security Settings**.
11. Choose **Next: Configure Security Groups**.
12. Choose **Select an existing security group**, and make a note of the security group ID.
13. Choose **Next: Configure Routing**.
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**.
5. In **Protocol** choose **HTTP**. In **Port**, enter **`8080`**.
6. In **VPC**, choose the default VPC.
7. Choose **Create**.

> **Note**
>
> You must have two target groups created for your load balancer in order for your deployment to run. You only need to make a note of the ARN of your first target group. This ARN is used in the `create-service` JSON file in the next step.

To update your load balancer to include your second target group

1. Open the Amazon EC2 console. In the navigation pane, choose **Load Balancers**.
2. Choose your load balancer, and then choose the **Listeners** tab. Choose the listener with port 8080, and then choose **Edit**.
3. Choose the pencil icon next to **Forward to**. Choose your second target group, and then choose the check mark. Choose **Update** to save the updates.

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 classic console at https://eusc-de-east-1.console.amazonaws-eusc.eu/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.

   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.

   > **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.

```
{
    "taskDefinition": "family:revision-number",
    "cluster": "my-cluster",
    "loadBalancers": [
        {
            "targetGroupArn": "target-group-arn",
            "containerName": "sample-website",
            "containerPort": 80
        }
    ],
    "desiredCount": 1,
    "launchType": "FARGATE",
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "CODE_DEPLOY"
    },
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "subnets": [
                "subnet-1",
                "subnet-2"
```

```
            ],
            "securityGroups": [
                "security-group"
            ],
            "assignPublicIp": "ENABLED"
        }
    }
}
```

2. Run the **create-service** command, specifying the JSON file:

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

This example creates a service named `my-service`.

> **Note**
>
> This example command creates a service named my-service. If you already have a service with this name, the command returns an error.

```
aws ecs create-service --service-name my-service --cli-input-json file://create-service.json
```

The output returns the description fields for your service.

3. Run the **describe-services** command to verify that your service was created.

```
aws ecs describe-services --cluster cluster-name --services service-name
```

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:
    a. Open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
    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

1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home.
2. On the **Welcome** page, **Getting started** page, or the **Pipelines** page, choose **Create pipeline**.
3. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. In **Step 2: Choose pipeline settings**, in **Pipeline name**, enter **`MyImagePipeline`**.
5. CodePipeline provides V1 and V2 type pipelines, which differ in characteristics and price. The V2 type is the only type you can choose in the console. For more information, see pipeline types. For information about pricing for CodePipeline, see Pricing.
6. In **Service role**, choose **New service role** to allow CodePipeline to create a service role in IAM.
7. Leave the settings under **Advanced settings** at their defaults, and then choose **Next**.
8. In **Step 3: 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. In **Branch name**, choose the name of the branch that contains your latest code update.

Choose **Next**.

9. In **Step 4: Add build stage**, choose **Skip build stage**, and then accept the warning message by choosing **Skip** again. Choose **Next**.
10. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

    Choose **Next**.
11. In **Step 6: 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`**.

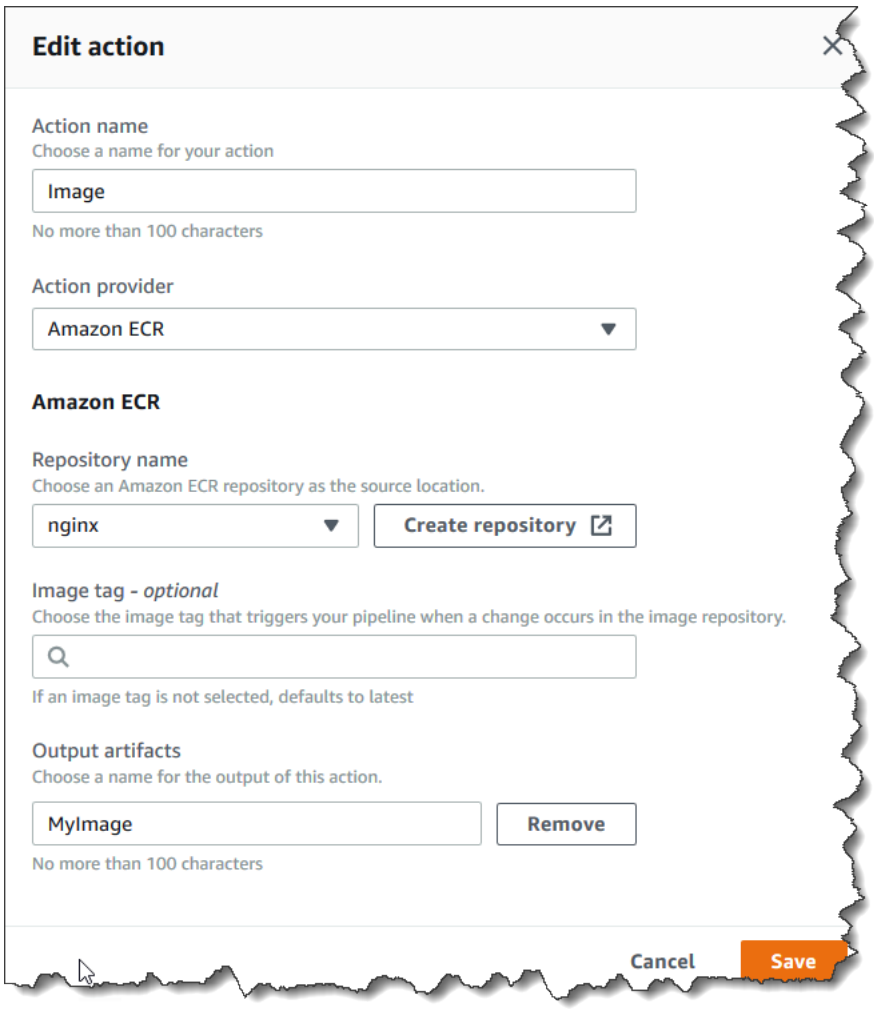
    > **Note**
    >
    > At this point, do not fill in any information under **Dynamically update task definition image**.

    d. Choose **Next**.
12. In **Step 7: 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**.



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.

> **Important**
>
> As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

> **Note**
>
> This feature is not available in the Asia Pacific (Hong Kong) or Europe (Milan) Region. To use other deploy actions available in that Region, see Deploy action integrations.

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.

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).
- 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:

   ```
   ask api list-skills
   ```

   Locate your skill by name and then copy the associated ID in the `skillId` field.

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

1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home.
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. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
5. In **Step 2: Choose pipeline settings**, in **Pipeline name**, enter **`MyAlexaPipeline`**.
6. CodePipeline provides V1 and V2 type pipelines, which differ in characteristics and price. The V2 type is the only type you can choose in the console. For more information, see pipeline types. For information about pricing for CodePipeline, see Pricing.

7. In **Service role**, choose **New service role** to allow CodePipeline to create a service role in IAM.
8. Leave the settings under **Advanced settings** at their defaults, and then choose **Next**.
9. In **Step 3: 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. In **Branch name**, choose the name of the branch that contains your latest code update.

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

   Choose **Next**.
10. In **Step 4: Add build stage**, choose **Skip build stage**, and then accept the warning message by choosing **Skip** again.

    Choose **Next**.
11. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

    Choose **Next**.
12. In **Step 6: Add deploy stage**:

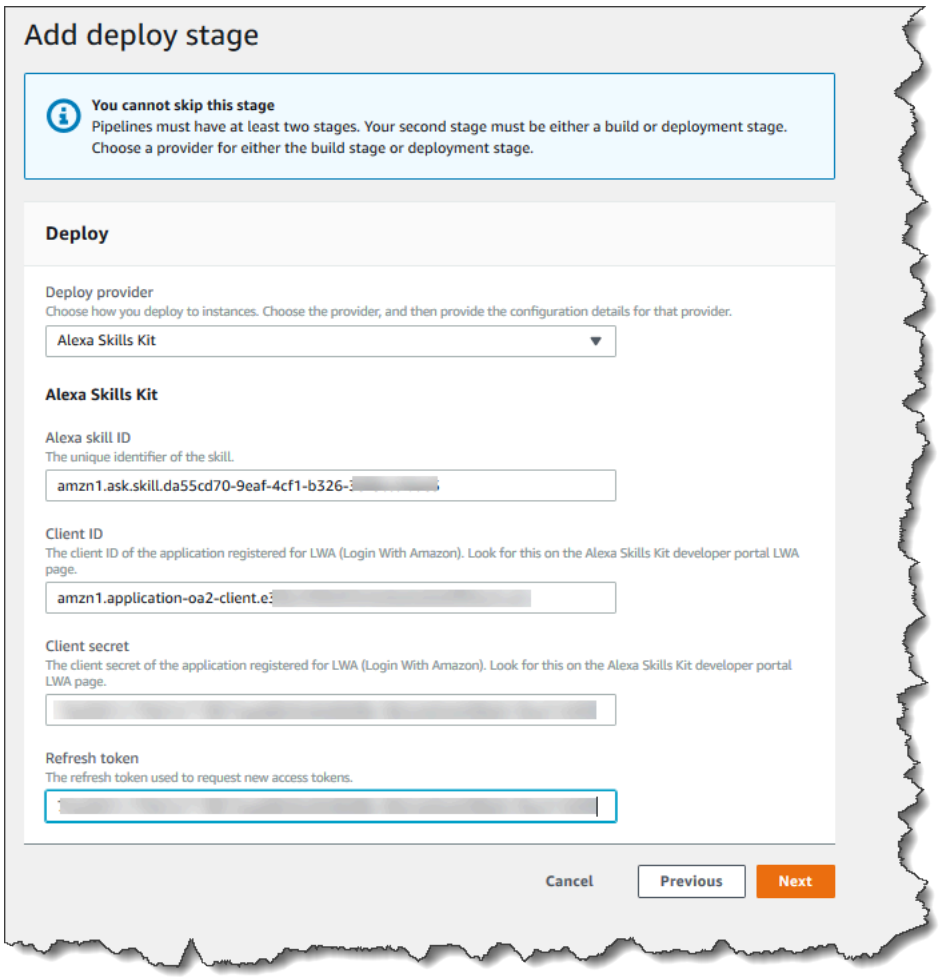
    a. In **Deploy provider**, choose **Alexa Skills Kit**.

    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**.

13. In **Step 7: 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 or add your website files in your source location, the deployment creates the website with your latest files.

Important

As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

Note

Even if you delete files from the source repository, the S3 deploy action does not delete S3 objects corresponding to deleted 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.
- Create a pipeline that compiles sample TypeScript code into JavaScript and deploys the CodeBuild output artifact to your S3 bucket for archive. This example creates a pipeline with an Amazon S3 source action, a CodeBuild build action, and an Amazon S3 deployment action. See Option 2: Deploy built archive files to Amazon S3 from an S3 source bucket.

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.

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).
- 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

1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home.
2. On the **Welcome** page, **Getting started** page, or **Pipelines** page, choose **Create pipeline**.
3. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. In **Step 2: Choose pipeline settings**, in **Pipeline name**, enter **`MyS3DeployPipeline`**.
5. In **Pipeline type**, choose **V2**. For more information, see Pipeline types. Choose **Next**.
6. In **Service role**, choose **New service role** to allow CodePipeline to create a service role in IAM.
7. Leave the settings under **Advanced settings** at their defaults, and then choose **Next**.
8. In **Step 3: 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. 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 `main` 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**.
9. In **Step 4: Add build stage**, choose **Skip build stage**, and then accept the warning message by choosing **Skip** again.

   Choose **Next**.
10. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

    Choose **Next**.

11. In **Step 6: Add deploy stage**:

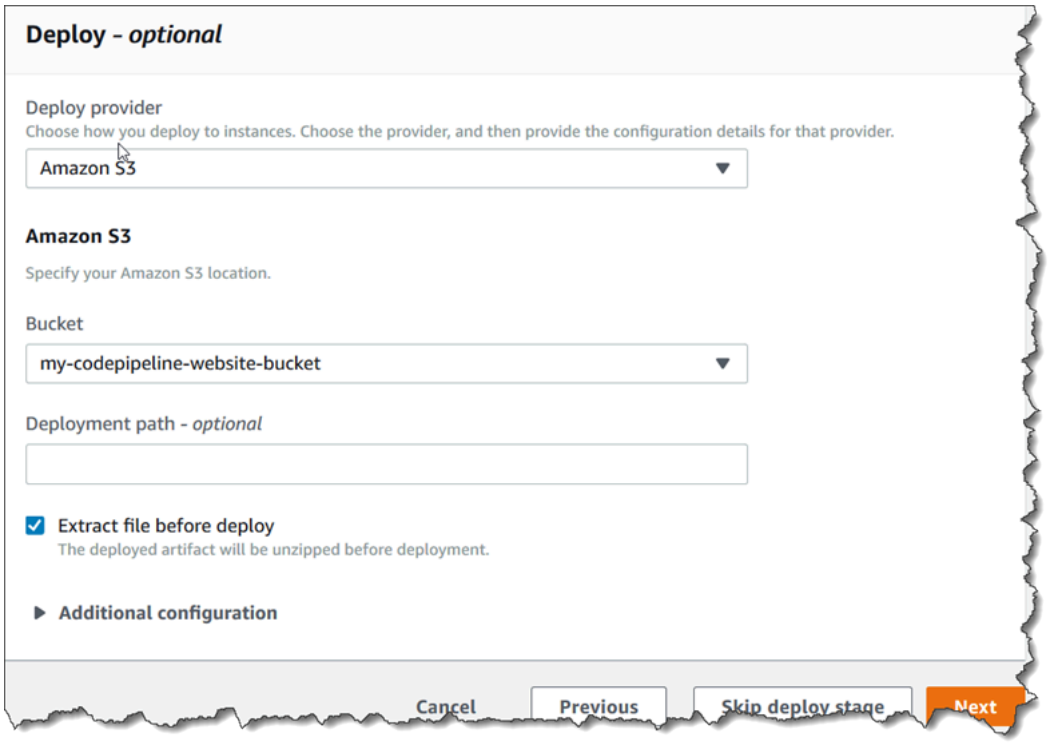
    a. In **Deploy provider**, choose **Amazon S3**.

    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**.

12. In **Step 7: Review**, review the information, and then choose **Create pipeline**.
13. 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
```

14. Access your endpoint to test the website. Your endpoint follows this format: `http://bucket-name.s3-website-region.amazonaws.com/`.

    Example endpoint: `http://my-bucket.s3-website-us-west-2.amazonaws.com/`.

    The sample web page appears.

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).
- 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

1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home.
2. On the **Welcome** page, **Getting started** page, or **Pipelines** page, choose **Create pipeline**.
3. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. In **Step 2: Choose pipeline settings**, in **Pipeline name**, enter **`MyS3DeployPipeline`**.
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 3: 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**.
8. In **Step 4: Add build stage**:

   a. In **Build provider**, choose **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**.
9. In **Step 5: Add deploy stage**:

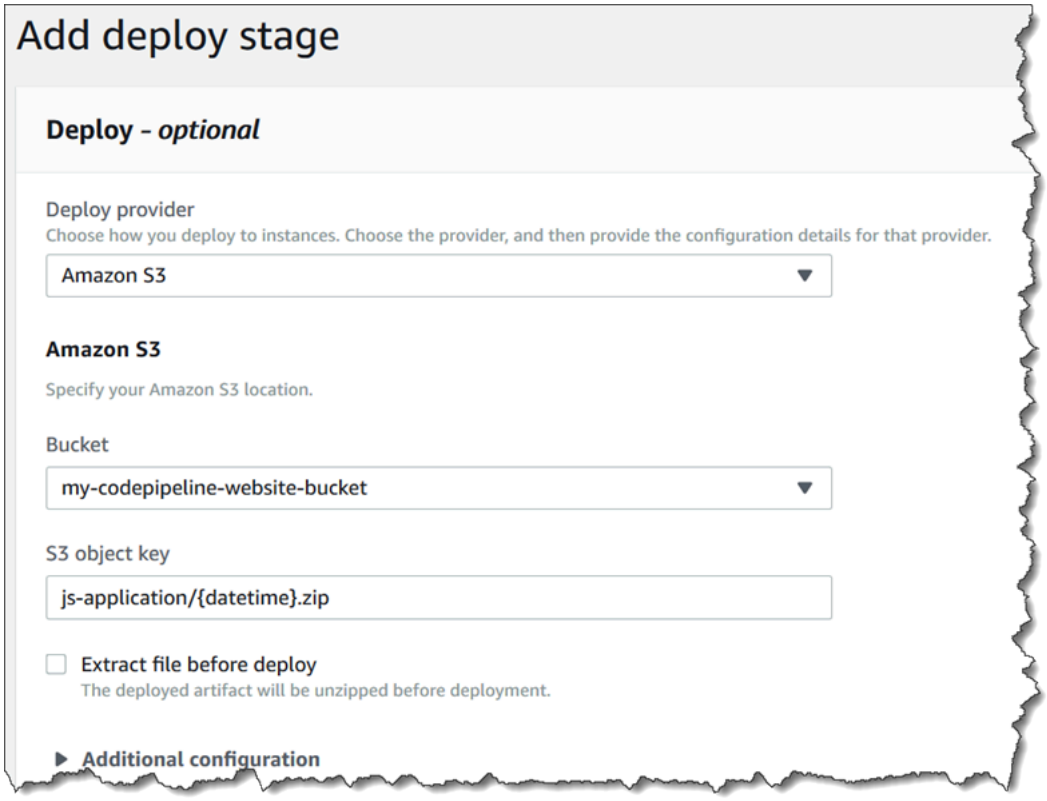
a. In **Deploy provider**, choose **Amazon S3**.

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.



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**.

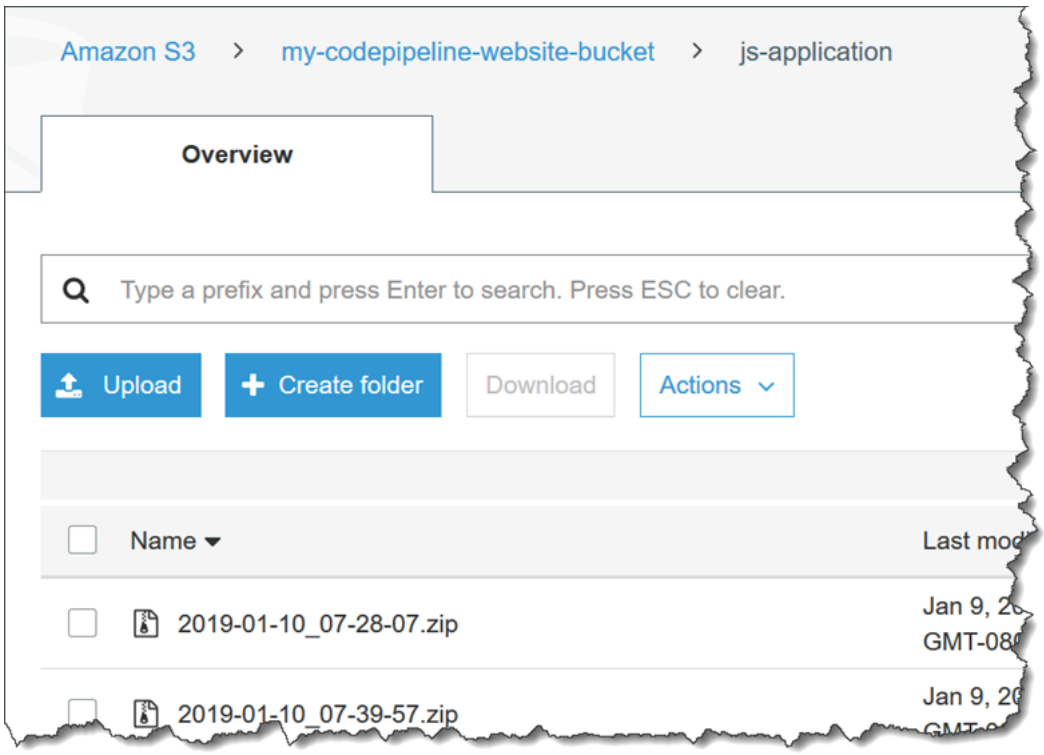
10. In **Step 6: Review**, review the information, and then choose **Create pipeline**.

11. 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:

```
var HelloGreeting = /** @class */ (function () {
    function HelloGreeting() {
        this.message = "Hello!";
    }
    return HelloGreeting;
}());
function greet(greeting) {
    console.log(greeting.message);
}
var greeting = new HelloGreeting();
greet(greeting);
```

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:



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.

> **Important**
>
> As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

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.

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:
      - 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://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
2. If necessary, switch to the AWS Region where you want to publish your serverless application.
3. On the **Welcome** page, **Getting started** page, or the **Pipelines** page, choose **Create pipeline**.
4. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
5. Choose **Create pipeline**. On the **Step 2: Choose pipeline settings** page, in **Pipeline name**, enter the name for your pipeline.
6. In **Pipeline type**, choose **V2**. For more information, see Pipeline types. Choose **Next**.
7. In **Service role**, choose **New service role** to allow CodePipeline to create a service role in IAM.
8. Leave the settings under **Advanced settings** at their defaults, and then choose **Next**.
9. On the **Step 3: Add source stage** page, in **Source provider**, choose **GitHub**.
10. Under **Connection**, choose an existing connection or create a new one. To create or manage a connection for your GitHub source action, see GitHub connections.
11. In **Repository**, choose your GitHub source repository.
12. In **Branch**, choose your GitHub branch.
13. Leave the remaining defaults for the source action. Choose **Next**.
14. On the **Step 4: 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**.

15. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

    Choose **Next**.

16. On the **Step 6: Add deploy stage** page, choose **Skip deploy stage**, and then accept the warning message by choosing **Skip** again. Choose **Next**.

17. On **Step 7: Review**, choose **Create pipeline**. You should see a diagram that shows the stages.

18. 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 *`bucketname`* with the name of your S3 bucket.

```
        {
            "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 CloudFormation Stack** to open the 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.

1. Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
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: Lambda function deployments with CodePipeline

This tutorial helps you to create a deploy action in CodePipeline that deploys your code to your function you have configured in Lambda. You will create a sample Lambda function where you will create an alias and version, add the zipped Lambda function to your source location, and run the Lambda action in your pipeline.

> **Note**
>
> As part of creating a pipeline in the console, an S3 artifact bucket will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

> **Note**
>
> The Lambda deploy action is only available for V2 type pipelines.

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.

- A source control repository, such as GitHub, or a source S3 bucket (this tutorial uses S3) where you will store a `.zip` file that you create for your Lambda function.
- You must use an existing CodePipeline service role that has been updated with the permissions for this action. To update your service role, see Service role policy permissions for the Lambda deploy action.

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

Step 1: Create your sample Lambda function

In this step, you create the Lambda function where you will deploy.

To create your Lambda function

1. Access the Lambda console and follow the steps in the following tutorial to create a sample Lambda function: link.
2. From the top navigation, choose **Create**, and select **Start from scratch** from the top of the page.
3. In **Name**, enter **`MyLambdaFunction`**.
4. Publish a new version. This will be the version that the alias will point to.

    a. Select your function.

b. Choose the **Actions** dropdown.

c. Choose **Publish new version**.

d. (Optional) Add to the description in **Description**.

e. Choose **Publish**.

5. Create an alias for your function, such as `aliasV1`.
6. Make sure the alias is pointing to the version that you just created (such as 1).

> **Note**
>
> If you choose $LATEST, you cannot use traffic shifting features because Lambda does not support $LATEST for an alias pointing to more than 1 version.

Step 2: Upload the function file to your repository

Download the function and save it as a zip file. Upload the zipped file to your S3 bucket using the following steps.

To add a `.zip` file to your source repository

1. Open your S3 bucket.
2. Choose **Upload**.
3. Upload the zip file containing your `sample_lambda_source.zip` file to your source bucket.

   Make a note of the path.

```
object key
```

Step 3: Creating your pipeline

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

To create your pipeline

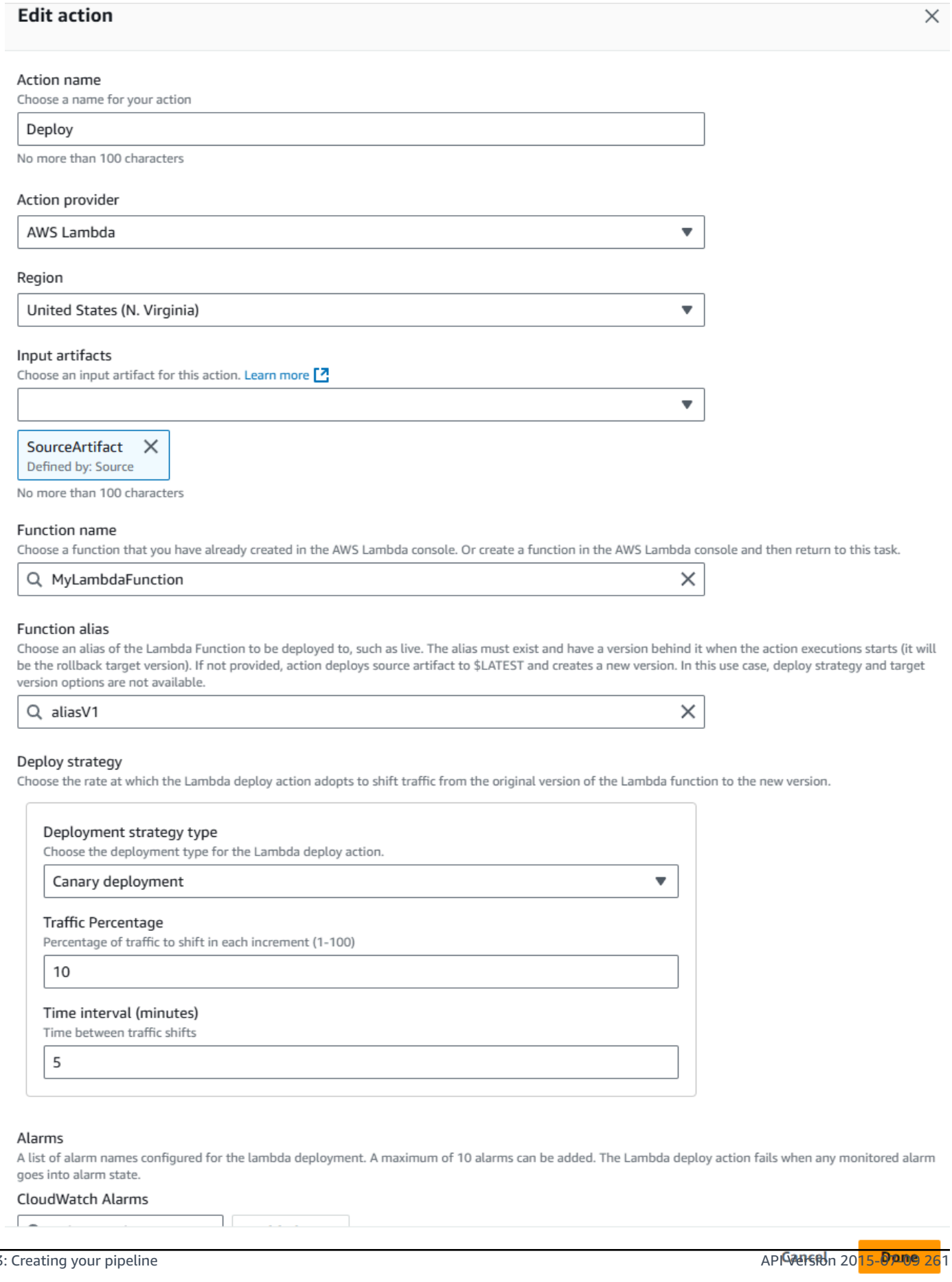
1. Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.

2. On the **Welcome** page, **Getting started** page, or the **Pipelines** page, choose **Create pipeline**.
3. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. In **Step 2: Choose pipeline settings**, in **Pipeline name**, enter **`MyPipeline`**.
5. CodePipeline provides V1 and V2 type pipelines, which differ in characteristics and price. The V2 type is the only type you can choose in the console. For more information, see pipeline types. For information about pricing for CodePipeline, see Pricing.
6. In **Service role**, choose **Use existing service role**, and then choose the CodePipeline service role that has been updated with the required permissions for this action. To configure your CodePipeline service role for this action, see Service role policy permissions for the Lambda deploy action.
7. Leave the settings under **Advanced settings** at their defaults, and then choose **Next**.
8. On the **Step 3: Add source stage** page, add a source stage:

   a. In **Source provider**, choose **Amazon S3**.

   b. In **Object key**, add the name of your .zip file, including the file extension, such as `sample_lambda_source.zip`.

   Choose **Next**.
9. On the **Step 4: Add build stage** page, choose **Skip**.
10. On the **Step 5: Add test stage** page, choose **Skip**.
11. On the **Step 6: Add deploy stage** page, choose **Lambda**.

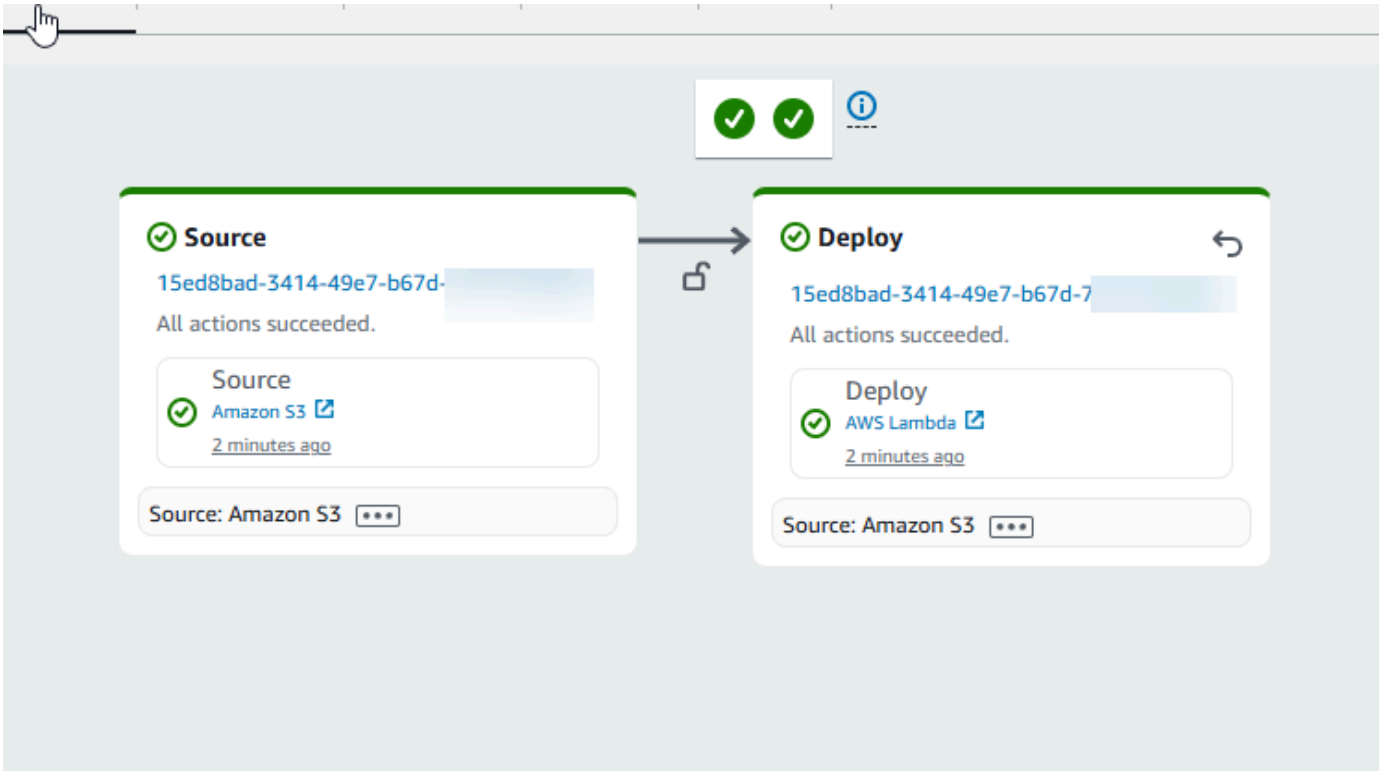

Edit action
Action name
Choose a name for your action
Deploy
No more than 100 characters
Action provider
AWS Lambda
Region
United States (N. Virginia)
Input artifacts
Choose an input artifact for this action. Learn more
SourceArtifact
Defined by: Source
No more than 100 characters
Function name
Choose a function that you have already created in the AWS Lambda console. Or create a function in the AWS Lambda console and then return to this task.
MyLambdaFunction
Function alias
Choose an alias of the Lambda Function to be deployed to, such as live. The alias must exist and have a version behind it when the action executions starts (it will be the rollback target version). If not provided, action deploys source artifact to $LATEST and creates a new version. In this use case, deploy strategy and target version options are not available.
aliasV1
Deploy strategy
Choose the rate at which the Lambda deploy action adopts to shift traffic from the original version of the Lambda function to the new version.
Deployment strategy type
Choose the deployment type for the Lambda deploy action.
Canary deployment
Traffic Percentage
Percentage of traffic to shift in each increment (1-100)
10
Time interval (minutes)
Time between traffic shifts
5
Alarms
A list of alarm names configured for the lambda deployment. A maximum of 10 alarms can be added. The Lambda deploy action fails when any monitored alarm goes into alarm state.
CloudWatch Alarms
Cancel
Done

a. Add your function name and alias.

b. Choose your deploy strategy.

c. Choose **Next**.

12. On the **Step 7: Review** page, review your pipeline configuration and choose **Create pipeline** to create the pipeline.



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.
2. Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
3. Choose your pipeline from the list.
4. Watch the pipeline progress through its stages. Your pipeline should complete and your action deploys to your Lambda function.

Learn more

The Lambda deploy action allows two methods of deployment. One method is traffic shifting alone without an input artifact from the source action. The other method is updating function code using an input artifact from the source action, then publishing a new version based on the updated code. For the second method, if the alias is provided, CodePipeline will do the traffic shifting as well. This Lambda deploy action tutorial demonstrates updating your function using a source artifact.

To learn more about the action, see the action reference page at AWS Lambda deploy action reference.

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 reference.

> **Important**
>
> As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

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

- Step 1: Create a Lambda function
- Step 2: Add a Lambda invoke action and manual approval action to your pipeline

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).
- 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://eusc-de-east-1.console.amazonaws-eusc.eu/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 your pipeline later in the tutorial, but you will add the code assuming the variable will be passed.

   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`.

```
const putJobSuccess = async (message) => {
```

```
        const params = {
            jobId: jobId,
            outputVariables: {
                testRunId: Math.floor(Math.random() * 1000).toString(),
                dateTime: Date(Date.now()).toString(),
                region: lambdaRegion
            }
        };
```

In your new function, on the **Code** tab, paste the following example code under `index.mjs`.

```
import { CodePipeline } from '@aws-sdk/client-codepipeline';

export const handler = async (event, context) => {
    const codepipeline = new CodePipeline({});

    // Retrieve the Job ID from the Lambda action
    const jobId = event["CodePipeline.job"].id;

    // Retrieve UserParameters
    const params =
 event["CodePipeline.job"].data.actionConfiguration.configuration.UserParameters;

    // The region from where the lambda function is being executed
    const lambdaRegion = process.env.AWS_REGION;

    // Notify CodePipeline of a successful job
    const putJobSuccess = async (message) => {
        const params = {
            jobId: jobId,
            outputVariables: {
                testRunId: Math.floor(Math.random() * 1000).toString(),
                dateTime: Date(Date.now()).toString(),
                region: lambdaRegion
            }
        };

        try {
            await codepipeline.putJobSuccessResult(params);
            return message;
        } catch (err) {
            throw err;
        }
```

```
    };

    // Notify CodePipeline of a failed job
    const putJobFailure = async (message) => {
        const params = {
            jobId: jobId,
            failureDetails: {
                message: JSON.stringify(message),
                type: 'JobFailed',
                externalExecutionId: context.invokeid
            }
        };

        try {
            await codepipeline.putJobFailureResult(params);
            throw message;
        } catch (err) {
            throw err;
        }
    };

    try {
        console.log("Testing commit - " + params);

        // Your tests here

        // Succeed the job
        return await putJobSuccess("Tests passed.");
    } catch (ex) {
        // If any of the assertions failed then fail the job
        return await putJobFailure(ex);
    }
};
```

7. Allow the function to auto save.
8. Copy the Amazon Resource Name (ARN) contained in the **Function ARN** field at the top of the screen.
9. As a last step, open the AWS Identity and Access Management (IAM) console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/. Modify the Lambda execution role to add the following policy: AWSCodePipelineCustomActionAccess. For the steps to create a Lambda execution role or modify the role policy, see Step 2: Create the Lambda function .

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

1. Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.

   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 **Add action group** 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 **FunctionName**, add the ARN of the Lambda function that you created.

   f. In **Variable namespace**, add the namespace name, such as **`TestVariables`**.

   g. In **Output artifacts**, add the output artifact name, such as **`LambdaArtifact`**.

   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`**.

   c. In **Action provider**, choose **Manual 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.

```
Make sure to review the code before approving this action. Test Run ID:
 #{TestVariables.testRunId}
```

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.

Important

As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

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
- Step 1: Create the sample state machine
- Step 2: Add a Step Functions invoke action to your pipeline

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) or Tutorial: Create a simple pipeline (CodeCommit repository).

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.

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:

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. 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**.
18. On your completed pipeline, choose **AWS Step Functions** in your invoke action. In the AWS Step Functions console, view your state machine execution ID. The ID shows your state machine name `HelloWorld` and the state machine execution ID with the prefix `my-prefix`.

```
arn:aws:states:us-west-2:account-ID:execution:HelloWorld:my-
prefix-0d9a0900-3609-4ebc-925e-83d9618fcca1
```

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.

> **Important**
>
> As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline,

make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

Topics

- Prerequisites
- Step 1: Create your AWS AppConfig resources
- Step 2: Upload files to your S3 source bucket
- Step 3: Create your pipeline
- Step 4: Make a change to any source file and verify deployment

Prerequisites

Before you begin, you must complete the following:

- This example uses an S3 source for your pipeline. Create or use an Amazon S3 bucket with versioning enabled. Follow the instructions in Step 1: Create an S3 source bucket for your application 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, create your configuration file or files. Then 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:

   ```
   Hello World!
   ```

2. Use the following steps to zip and upload your configuration files.

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:

   ```
   .
   ### appconfig-configurations
       ### MyConfigurations
           ### us-east-1
           #   ### configuration.json
           ### us-west-2
               ### configuration.json
   ```

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 an Amazon S3 action where the source artifacts are the files for your configuration.
- A deployment stage with an AppConfig deployment action.

To create a pipeline with the wizard

1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home.
2. On the **Welcome** page, **Getting started** page, or the **Pipelines** page, choose **Create pipeline**.
3. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. In **Step 2: Choose pipeline settings**, in **Pipeline name**, enter **`MyAppConfigPipeline`**.
5. CodePipeline provides V1 and V2 type pipelines, which differ in characteristics and price. The V2 type is the only type you can choose in the console. For more information, see pipeline types. For information about pricing for CodePipeline, see Pricing.
6. In **Service role**, choose **New service role** to allow CodePipeline to create a service role in IAM.
7. Leave the settings under **Advanced settings** at their defaults, and then choose **Next**.
8. In **Step 3: 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**.
9. In **Step 4: Add build stage**, choose **Skip build stage**, and then accept the warning message by choosing **Skip** again.

   Choose **Next**.
10. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

    Choose **Next**.
11. In **Step 6: 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 .zip file. For this example, enter the following file path: `appconfig-configurations/MyConfigurations/us-west-2/configuration.json`.

g. Choose **Next**.

12. In **Step 7: 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.

Tutorial: Use full clone with a GitHub pipeline source

You can choose the full clone option for your GitHub source action in CodePipeline. Use this option to run CodeBuild commands for Git metadata in your pipeline build action.

> **Note**
>
> The full clone option described here refers to specifying whether CodePipeline should clone repository metadata, which can only be used by CodeBuild commands. To use a GitHub user access token for use with CodeBuild projects, follow the steps here to install the AWS Connector for GitHub app and then leave the App installation field empty. AWS CodeConnections will use the user access token for the connection.

> **Important**
>
> As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

In this tutorial, you will create a pipeline that connects to your GitHub repository, uses the full clone option for source data, and runs a CodeBuild build that clones your repository and performs Git commands for the repository.

Topics

- Prerequisites
- Step 1: Create a README file
- Step 2: Create your pipeline and build project
- Step 3: Update the CodeBuild service role policy to use connections
- Step 4: View repository commands in build output

Prerequisites

Before you begin, you must do the following:

- Create a GitHub repository with your GitHub account.
- Have your GitHub credentials ready. When you use the AWS Management Console to set up a connection, you are asked to sign in with your GitHub credentials.

Step 1: Create a README file

After you create your GitHub repository, use these steps to add a README file.

1. Log in to your GitHub repository and choose your repository.
2. To create a new file, choose **Add file > Create new file**. Name the file `README.md` and add the following text.

```
This is a GitHub repository!
```

3. Choose **Commit changes**.

   Make sure the `README.md` file is at the root level of your repository.

Step 2: Create your pipeline and build project

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

- A source stage with a connection to your GitHub repository and action.
- A build stage with an AWS CodeBuild build action.

To create a pipeline with the wizard

1. Sign in to the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
2. On the **Welcome** page, **Getting started** page, or **Pipelines** page, choose **Create pipeline**.
3. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. In **Step 2: Choose pipeline settings**, in **Pipeline name**, enter **`MyGitHubPipeline`**.
5. In **Pipeline type**, choose **V1** for the purposes of this tutorial. You can also choose **V2**; however, note that pipeline types differ in characteristics and price. For more information, see Pipeline types.
6. In **Service role**, choose **New service role**.

   > **Note**
   >
   > If you choose instead to use your existing CodePipeline service role, make sure that you have added the `codestar-connections:UseConnection` IAM permission to your service role policy. For instructions for the CodePipeline service role, see Add permissions to the CodePipeline service role.

7. Under **Advanced settings**, leave the defaults. 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.

> **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.

Choose **Next**.

8. On the **Step 3: Add source stage** page, add a source stage:

   a. In **Source provider**, choose **GitHub (via GitHub App)**.

   b. Under **Connection**, choose an existing connection or create a new one. To create or manage a connection for your GitHub source action, see GitHub connections.

      You install one app for all of your connections to a particular provider. If you have already installed the AWS Connector for GitHub app, choose it and skip this step.

      > **Note**
      >
      > If you want to create a user access token, make sure that you've already installed the AWS Connector for GitHub app and then leave the App installation field empty. AWS CodeConnections will use the user access token for the connection. For more information, see Access your source provider in CodeBuild.

   c. In **Repository name**, choose the name of your GitHub repository.

   d. In **Branch name**, choose the repository branch you want to use.

   e. Make sure the **Start the pipeline on source code change** option is selected.

   f. Under **Output artifact format**, choose **Full clone** to enable the Git clone option for the source repository. Only actions provided by CodeBuild can use the Git clone option. You will use Step 3: Update the CodeBuild service role policy to use connections in this tutorial to update the permissions for your CodeBuild project service role to use this option.

   Choose **Next**.

9. In **Step 4: 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:5.0**.

f. For **Service role**, choose **New service role**.

> **Note**
>
> Note the name of your CodeBuild service role. You will need the role name for the final step in this tutorial.

g. Under **Buildspec**, for **Build specifications**, choose **Insert build commands**. Choose **Switch to editor**, and paste the following under **Build commands**.

> **Note**
>
> In the env section of the build spec, make sure the credential helper for git commands is enabled as shown in this example.

```
version: 0.2

env:
  git-credential-helper: yes
phases:
  install:
    #If you use the Ubuntu standard image 2.0 or later, you must specify
 runtime-versions.
    #If you specify runtime-versions and use an image other than Ubuntu
 standard image 2.0, the build fails.
    runtime-versions:
      nodejs: 12
      # name: version
    #commands:
      # - command
      # - command
  pre_build:
    commands:
      - ls -lt
      - cat README.md
```

```
  build:
    commands:
      - git log | head -100
      - git status
      - ls
      - git archive --format=zip HEAD > application.zip
  #post_build:
    #commands:
      # - command
      # - command
artifacts:
  files:
    - application.zip
    # - location
  #name: $(date +%Y-%m-%d)
  #discard-paths: yes
  #base-directory: location
#cache:
  #paths:
    # - paths
```

h. Choose **Continue to CodePipeline**. This returns to the CodePipeline console and creates a CodeBuild project that uses your build commands 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**.

10. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

    Choose **Next**.

11. On the **Step 6: Add deploy stage** page, choose **Skip deploy stage**, and then accept the warning message by choosing **Skip** again. Choose **Next**.

12. On **Step 7: Review**, choose **Create pipeline**.

Step 3: Update the CodeBuild service role policy to use connections

The initial pipeline run will fail because the CodeBuild service role must be updated with permissions to use connections. Add the `codestar-connections:UseConnection` IAM permission to your service role policy. For instructions to update the policy in the IAM console,

see Add CodeBuild GitClone permissions for connections to Bitbucket, GitHub, GitHub Enterprise Server, or GitLab.com.

Step 4: View repository commands in build output

1. When your service role is successfully updated, choose **Retry** on the failed CodeBuild stage.
2. After the pipeline runs successfully, on your successful build stage, choose **View details**.

   On the details page, choose the **Logs** tab. View the CodeBuild build output. The commands output the value of the entered variable.

   The commands output the `README.md` file contents, list the files in the directory, clone the repository, view the log, and archive the repository as a ZIP file.

Tutorial: Use full clone with a CodeCommit pipeline source

You can choose the full clone option for your CodeCommit source action in CodePipeline. Use this option to allow CodeBuild to access Git metadata in your pipeline build action.

In this tutorial, you create a pipeline that accesses your CodeCommit repository, uses the full clone option for source data, and runs a CodeBuild build that clones your repository and performs Git commands for the repository.

> **Note**
>
> CodeBuild actions are the only downstream actions that support the use of Git metadata available with the Git clone option. Also, while your pipeline can contain cross-account actions, the CodeCommit action and the CodeBuild action must be in the same account for the full clone option to succeed.

> **Important**
>
> As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline,

make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

Topics

- Prerequisites
- Step 1: Create a README file
- Step 2: Create your pipeline and build project
- Step 3: Update the CodeBuild service role policy to clone the repository
- Step 4: View repository commands in build output

Prerequisites

Before you begin, you must create a CodeCommit repository in the same AWS account and Region as your pipeline.

Step 1: Create a README file

Use these steps to add a README file to your source repository. The README file provides an example source file for the CodeBuild downstream action to read.

To add a README file

1. Log in to your repository and choose your repository.
2. To create a new file, choose **Add file > Create file**. Name the file `README.md` and add the following text.

   ```
   This is a CodeCommit repository!
   ```

3. Choose **Commit changes**.

   Make sure the `README.md` file is at the root level of your repository.

Step 2: Create your pipeline and build project

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

- A source stage with a CodeCommit source action.
- A build stage with an AWS CodeBuild build action.

To create a pipeline with the wizard

1. Sign in to the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
2. On the **Welcome** page, **Getting started** page, or **Pipelines** page, choose **Create pipeline**.
3. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. In **Step 2: Choose pipeline settings**, in **Pipeline name**, enter **`MyCodeCommitPipeline`**.
5. CodePipeline provides V1 and V2 type pipelines, which differ in characteristics and price. The V2 type is the only type you can choose in the console. For more information, see pipeline types. For information about pricing for CodePipeline, see Pricing.
6. In **Service role**, do one of the following:
   - Choose **Existing service role**.
   - Choose your existing CodePipeline service role. This role must have the `codecommit:GetRepository` IAM permission to your service role policy. See Add permissions to the CodePipeline service role.
7. Under **Advanced settings**, leave the defaults. Choose **Next**.
8. On the **Step 3: Add source stage** page, do the following:

   a. In **Source provider**, choose **CodeCommit**.

   b. In **Repository name**, choose the name of your repository.

   c. In **Branch name**, choose your branch name.

   d. Make sure the **Start the pipeline on source code change** option is selected.

   e. Under **Output artifact format**, choose **Full clone** to enable the Git clone option for the source repository. Only actions provided by CodeBuild can use the Git clone option.

   Choose **Next**.
9. In **Step 4: Add build stage**, do the following:

   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:5.0**.

f. For **Service role**, choose **New service role**.

> **Note**
>
> Note the name of your CodeBuild service role. You will need the role name for the final step in this tutorial.

g. Under **Buildspec**, for **Build specifications**, choose **Insert build commands**. Choose **Switch to editor**, and then under **Build commands** paste the following code.

```
version: 0.2

env:
  git-credential-helper: yes
phases:
  install:
    #If you use the Ubuntu standard image 2.0 or later, you must specify
 runtime-versions.
    #If you specify runtime-versions and use an image other than Ubuntu
 standard image 2.0, the build fails.
    runtime-versions:
      nodejs: 12
      # name: version
    #commands:
      # - command
      # - command
  pre_build:
    commands:
      - ls -lt
      - cat README.md
  build:
    commands:
      - git log | head -100
      - git status
      - ls
      - git describe --all
  #post_build:
    #commands:
```

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

h. Choose **Continue to CodePipeline**. This returns you to the CodePipeline console and creates a CodeBuild project that uses your build commands 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**.

10. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

    Choose **Next**.

11. On the **Step 6: Add deploy stage** page, choose **Skip deploy stage**, and then accept the warning message by choosing **Skip** again. Choose **Next**.

12. On **Step 7: Review**, choose **Create pipeline**.

Step 3: Update the CodeBuild service role policy to clone the repository

The initial pipeline run will fail because you need to update the CodeBuild service role with permissions to pull from your repository.

Add the `codecommit:GitPull` IAM permission to your service role policy. For instructions to update the policy in the IAM console, see Add CodeBuild GitClone permissions for CodeCommit source actions.

Step 4: View repository commands in build output

To view the build output

1. When your service role is successfully updated, choose **Retry** on the failed CodeBuild stage.

2. After the pipeline runs successfully, on your successful build stage, choose **View details**.

   On the details page, choose the **Logs** tab. View the CodeBuild build output. The commands output the value of the entered variable.

   The commands output the `README.md` file contents, list the files in the directory, clone the repository, view the log, and run `git describe --all`.

Tutorial: Create a pipeline with AWS CloudFormation StackSets deployment actions

In this tutorial, you use the AWS CodePipeline console to create a pipeline with deployment actions for creating a stack set and creating stack instances. When the pipeline runs, the template creates a stack set and also creates and updates the instances where the stack set is deployed.

> **Important**
>
> As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

There are two ways to manage permissions for a stack set: self-managed and AWS-managed IAM roles. This tutorial provides examples with self-managed permissions.

To most effectively use StackSets in CodePipeline, you should have a clear understanding of the concepts behind CloudFormation StackSets and how they work. See StackSets concepts in the *AWS CloudFormation User Guide*.

Topics

- Prerequisites
- Step 1: Upload the sample AWS CloudFormation template and parameter file
- Step 2: Create your pipeline
- Step 3: View initial deployment

- Step 4: Add a CloudFormationStackInstances action
- Step 5: View stack set resources for your deployment
- Step 6: Make an update to your stack set

Prerequisites

For stack set operations, you use two different accounts: an administration account and a target account. You create stack sets in the administrator account. You create individual stacks that belong to a stack set in the target account.

To create an administrator role with your administrator account

- Follow the instructions in Set up basic permissions for stack set operations. Your role must be named **`AWSCloudFormationStackSetAdministrationRole`**.

To create a service role in the target account

- Create a service role in the target account that trusts the administrator account. Follow the instructions in Set up basic permissions for stack set operations. Your role must be named **`AWSCloudFormationStackSetExecutionRole`**.

Step 1: Upload the sample AWS CloudFormation template and parameter file

Create a source bucket for your stack set template and parameters files. Download the sample AWS CloudFormation template file, set up a parameters file, and then zip the files before upload to your S3 source bucket.

> **Note**
>
> Make sure to ZIP the source files before you upload to your S3 source bucket, even if the only source file is the template.

To create an S3 source bucket

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://eusc-de-east-1.console.amazonaws-eusc.eu/s3/.
2. Choose **Create bucket**.
3. In **Bucket name**, enter a name for your bucket.

   In **Region**, choose the Region where you want to create your pipeline. 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**.

To create the AWS CloudFormation template file

1. Download the following sample template file for generating CloudTrail configuration for stack sets: https://s3.amazonaws.com/cloudformation-stackset-sample-templates-us-east-1/EnableAWSCloudtrail.yml.
2. Save the file as `template.yml`.

To create the parameters.txt file

1. Create a file with the parameters for your deployment. Parameters are values that you want to update in your stack at runtime. The following sample file updates the template parameters for your stack set to enable logging validation and global events.

```
[
  {
    "ParameterKey": "EnableLogFileValidation",
    "ParameterValue": "true"
  },
  {
    "ParameterKey": "IncludeGlobalEvents",
    "ParameterValue": "true"
  }
]
```

2. Save the file as `parameters.txt`.

To create the accounts.txt file

1. Create a file with the accounts where you want to create instances, as shown in the following sample file.

```
[
    "111111222222","333333444444"
]
```

2. Save the file as `accounts.txt`.

To create and upload source files

1. Combine the files into a single ZIP file. Your files should look like this in your ZIP file.

```
template.yml
parameters.txt
accounts.txt
```

2. Upload the ZIP file to your S3 bucket. This file is the source artifact created by the **Create Pipeline** wizard for your deployment action in CodePipeline.

Step 2: Create your pipeline

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

- A source stage with an S3 source action where the source artifact is your template file and any supporting source files.
- A deployment stage with an CloudFormation stack set deployment action that creates the stack set.
- A deployment stage with an CloudFormation stack instances deployment action that creates the stacks and instances within the target accounts.

To create a pipeline with a CloudFormationStackSet action

1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home.
2. On the **Welcome** page, **Getting started** page, or **Pipelines** page, choose **Create pipeline**.

3. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. In **Step 2: Choose pipeline settings**, in **Pipeline name**, enter **`MyStackSetsPipeline`**.
5. In **Pipeline type**, choose **V1** for the purposes of this tutorial. You can also choose **V2**; however, note that pipeline types differ in characteristics and price. For more information, see Pipeline types.
6. In **Service role**, choose **New service role** to allow CodePipeline to create a service role in IAM.
7. In **Artifact store**, leave the defaults.

   > **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 and CodePipeline pipeline structure reference.

   Choose **Next**.
8. On the **Step 3: Add source stage** page, in **Source provider**, choose **Amazon S3**.
9. In **Bucket**, enter the S3 source bucket you created for this tutorial, such as `BucketName`. In **S3 object key**, enter the file path and file name for your ZIP file, such as `MyFiles.zip`.
10. Choose **Next**.
11. In **Step 4: Add build stage**, choose **Skip build stage**, and then accept the warning message by choosing **Skip** again.

    Choose **Next**.
12. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

    Choose **Next**.
13. In **Step 6: Add deploy stage**:

    a. In **Deploy provider**, choose **AWS CloudFormation Stack Set**.

b. In **Stack set name**, enter a name for the stack set. This is the name of the stack set that the template creates.

> **Note**
>
> Make a note of your stack set name. You will use it when you add the second StackSets deployment action to your pipeline.

c. In **Template path**, enter the artifact name and file path where you uploaded your template file. For example, enter the following using the default source artifact name `SourceArtifact`.

```
SourceArtifact::template.yml
```

d. In **Deployment targets**, enter the artifact name and file path where you uploaded your accounts file. For example, enter the following using the default source artifact name `SourceArtifact`.

```
SourceArtifact::accounts.txt
```

e. In **Deployment target AWS Regions**, enter one Region for deployment of your initial stack instance, such as `us-east-1`.

f. Expand **Deployment options**. In **Parameters**, enter the artifact name and file path where you uploaded your parameters file. For example, enter the following using the default source artifact name `SourceArtifact`.

```
SourceArtifact::parameters.txt
```

To enter the parameters as a literal input rather than a file path, enter the following:

```
ParameterKey=EnableLogFileValidation,ParameterValue=true
ParameterKey=IncludeGlobalEvents,ParameterValue=true
```

g. In **Capabilities**, choose CAPABILITY_IAM and CAPABILITY_NAMED_IAM.

h. In **Permission model**, choose SELF_MANAGED.

i. In **Failure tolerance percentage**, enter `20`.

j. In **Max concurrent percentage**, enter 25.

k. Choose **Next**.

l. In **Step 7: Review**, choose **Create pipeline**. Your pipeline displays.

m. Allow your pipeline to run.

Step 3: View initial deployment

View the resources and status for your initial deployment. After verifying the deployment successfully created your stack set, you can add the second action to your **Deploy** stage.

To view the resources

1. Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
2. Under **Pipelines**, choose your pipeline and choose **View**. The diagram shows your pipeline source and deployment stages.
3. Choose the CloudFormation action on the **CloudFormationStackSet** action in your pipeline. The template, resources, and events for your stack set are shown in the CloudFormation console.
4. In the left navigation panel, choose **StackSets**. In the list, choose the new stack set.
5. Choose the **Stack instances** tab. Verify that one stack instance for each account you provided was created in the us-east-1 Region. Verify that the status for each stack instance is CURRENT.

Step 4: Add a CloudFormationStackInstances action

Create a next action in your pipeline that will allow CloudFormation StackSets to create the remaining stack instances.

To create a next action in your pipeline

1. Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.

   Under **Pipelines**, choose your pipeline and choose **View**. The diagram shows your pipeline source and deployment stages.
2. Choose to edit the pipeline. The pipeline displays in **Edit** mode.
3. On the **Deploy** stage, choose **Edit**.

4. Under the **AWS CloudFormation Stack Set** deploy action, choose **Add action group**.
5. On the **Edit action** page, add the action details:
   a. In **Action name**, enter a name for the action.
   b. In **Action provider**, choose **AWS CloudFormation Stack Instances**.
   c. Under **Input artifacts**, choose **SourceArtifact**.
   d. In **Stack set name**, enter the name for the stack set. This is the name of the stack set that you provided in the first action.
   e. In **Deployment targets**, enter the artifact name and file path where you uploaded your accounts file. For example, enter the following using the default source artifact name `SourceArtifact`.

      ```
      SourceArtifact::accounts.txt
      ```

   f. In **Deployment target AWS Regions**, enter the Regions for deployment of your remaining stack instances, such as `us-east-2` and `eu-central-1` as follows:

      ```
      us-east2, eu-central-1
      ```

   g. In **Failure tolerance percentage**, enter `20`.
   h. In **Max concurrent percentage**, enter 25.
   i. Choose **Save**.
   j. Manually release a change. Your updated pipeline displays with two actions in the Deploy stage.

Step 5: View stack set resources for your deployment

You can view the resources and status for your stack set deployment.

To view the resources

1. Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
2. Under **Pipelines**, choose your pipeline and then choose **View**. The diagram shows your pipeline source and deployment stages.

3. Choose the CloudFormation action on the **`AWS CloudFormation Stack Instances`** action in your pipeline. The template, resources, and events for your stack set are shown in the CloudFormation console.
4. In the left navigation panel, choose **StackSets**. In the list, choose your stack set.
5. Choose the **Stack instances** tab. Verify that all remaining stack instances for each account you provided were created or updated in the expected Regions. Verify that the status for each stack instance is CURRENT.

Step 6: Make an update to your stack set

Make an update to your stack set and deploy the update to instances. In this example, you also make a change to the deployment targets you want to designate for update. The instances that are not part of the update move to an outdated status.

1. Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
2. Under **Pipelines**, choose your pipeline and then choose **Edit**. On the **Deploy** stage, choose **Edit**.
3. Choose to edit the **AWS CloudFormation Stack Set** action in your pipeline. In **Description**, write over the existing description with a new description for the stack set.
4. Choose to edit the **AWS CloudFormation Stack Instances** action in your pipeline. In **Deployment target AWS Regions**, delete the `us-east-2` value that was entered when the action was created.
5. Save the changes. Choose **Release change** to run your pipeline.
6. Open your action in CloudFormation. Choose the **StackSet info** tab. In **StackSet description**, verify that the new description is shown.
7. Choose the **Stack instances** tab. Under **Status**, verify that the status for the stack instances in us-east-2 is OUTDATED.

Tutorial: Create a variable check rule for a pipeline as an entry condition

In this tutorial, you configure a pipeline that continuously delivers files using GitHub as the source action provider in your source stage. The completed pipeline detects changes when you make a

change to the source files in your source repository. The pipeline runs and then checks the output variables against the source repository name and branch name provided in the condition for entry to the build stage.

Important

As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

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.

This example uses the example pipeline with a GitHub (Version2) source action and a CodeBuild build action where the entry condition for the build stage will check for variables.

Prerequisites

Before you begin, you must do the following:

- Create a GitHub repository with your GitHub account.
- Have your GitHub credentials ready. When you use the AWS Management Console to set up a connection, you are asked to sign in with your GitHub credentials.
- A connection to your repository to set up GitHub (via GitHub App) as the source action for your pipeline. To create a connection to your GitHub repository, see GitHub connections.

Step 1: Create sample source file and add to your GitHub repository

In this section, you create and add your example source file to the repository that the pipeline uses for your source stage. For this example, you produce and add the following:

- A `README.md` file.

After you create your GitHub repository, use these steps to add your README file.

1. Log in to your GitHub repository and choose your repository.
2. To create a new file, choose **Add file**, and then choose **Create new file**. Name the file `README.md` and add the following text.

   ```
   This is a GitHub repository!
   ```

3. Choose **Commit changes**. For the purposes of this tutorial, add a commit message that contains the capitalized word "Update" as in the following example:

   ```
   Update to source files
   ```

   > **Note**
   >
   > The rule check for strings is case-sensitive.

   Make sure the `README.md` file is at the root level of your repository.

Step 2: Create your pipeline

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

- A source stage with a connection to your GitHub repository and action.
- A CodeBuild build stage where the stage has an On Entry condition configured for the variable check rule.

To create a pipeline with the wizard

1. Sign in to the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
2. On the **Welcome** page, **Getting started** page, or **Pipelines** page, choose **Create pipeline**.
3. On the **Step 1: Choose creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. In **Step 2: Choose pipeline settings**, in **Pipeline name**, enter **`MyVarCheckPipeline`**.
5. CodePipeline provides V1 and V2 type pipelines, which differ in characteristics and price. The V2 type is the only type you can choose in the console. For more information, see pipeline types. For information about pricing for CodePipeline, see Pricing.
6. In **Service role**, choose **New service role**.

   > **Note**
   >
   > If you choose instead to use your existing CodePipeline service role, make sure that you have added the `codeconnections:UseConnection` IAM permission to your service role policy. For instructions for the CodePipeline service role, see Add permissions to the CodePipeline service role.

7. Under **Advanced settings**, leave the defaults.

   Choose **Next**.
8. On the **Step 3: Add source stage** page, add a source stage:

   a. In **Source provider**, choose **GitHub (via GitHub App)**.

   b. Under **Connection**, choose an existing connection or create a new one. To create or manage a connection for your GitHub source action, see GitHub connections.

   c. In **Repository name**, choose the name of your GitHub repository.

   d. In **Branch name**, choose the repository branch you want to use.

   e. Make sure the **No trigger** option is selected.

   Choose **Next**.
9. In **Step 4: 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:5.0**.

f. For **Service role**, choose **New service role**.

> **Note**
>
> Note the name of your CodeBuild service role. You will need the role name for the final step in this tutorial.

g. Under **Buildspec**, for **Build specifications**, choose **Insert build commands**. Choose **Switch to editor**, and paste the following under **Build commands**.

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

```
      -
  #post_build:
    #commands:
      # - command
      # - command
artifacts:
  files:
     - '*'
    # - location
  name: $(date +%Y-%m-%d)
  #discard-paths: yes
  #base-directory: location
#cache:
  #paths:
    # - paths
```

h. Choose **Continue to CodePipeline**. This returns to the CodePipeline console and creates a CodeBuild project that uses your build commands 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**.

10. In **Step 5: Add test stage**, choose **Skip test stage**, and then accept the warning message by choosing **Skip** again.

    Choose **Next**.

11. On the **Step 6: Add deploy stage** page, choose **Skip deploy stage**, and then accept the warning message by choosing **Skip** again. Choose **Next**.

12. On **Step 7: Review**, choose **Create pipeline**.

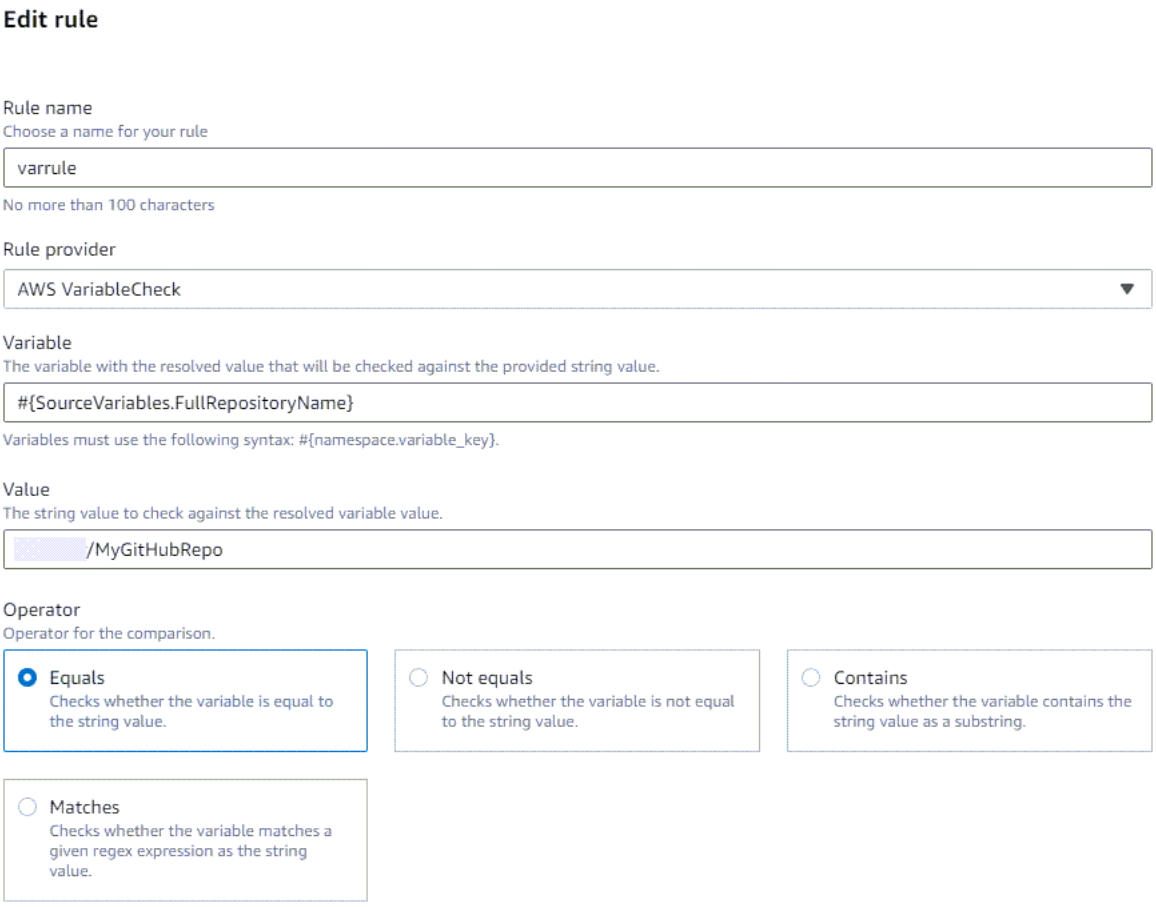
Step 2: Edit the build stage to add the condition and rule

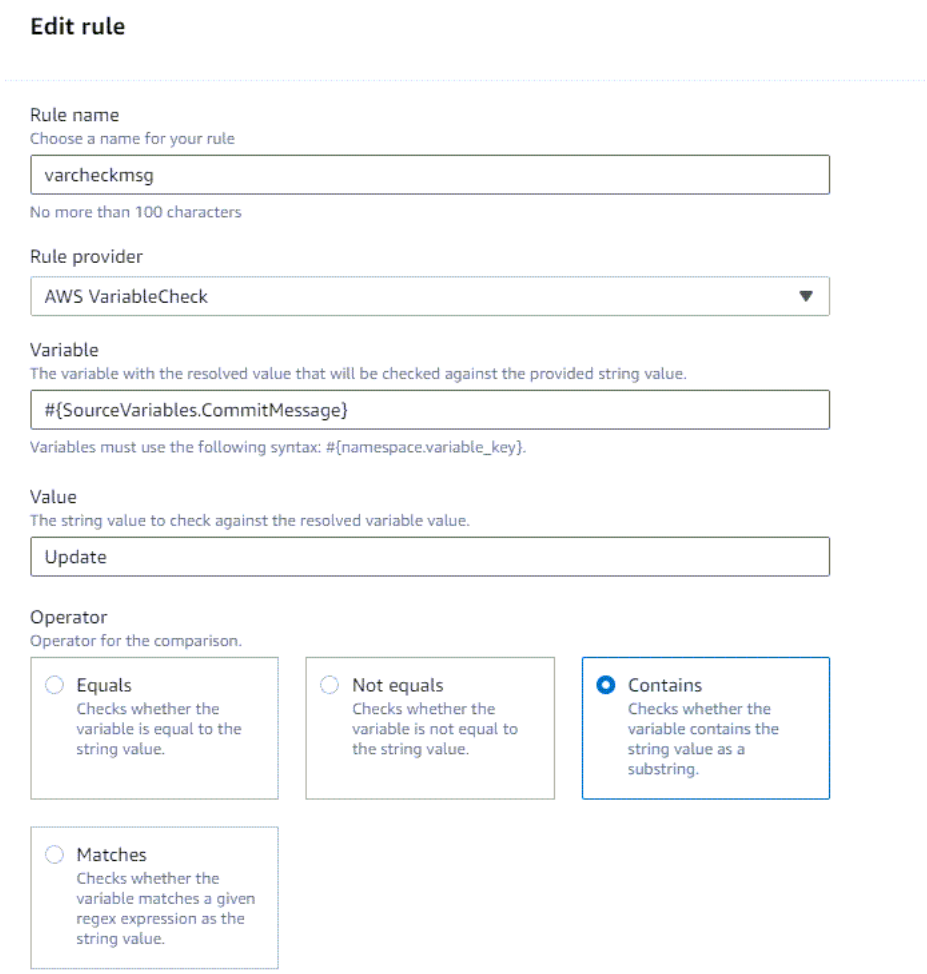
In this step, you edit the stage to add an On Entry condition for the variable check rule.

1. Choose your pipeline, and then choose **Edit**. Choose to add an entry rule on the build stage.

   In **Rule provider**, choose **VariableCheck**.

2. In **Variable**, enter the variable or variables that you want to check. In **Value**, enter the string value to check against the resolved variable. In the following example screens, a rule is created for an "equals" check, and another rule is created for a "contains" check.


Edit rule
Rule name
Choose a name for your rule
varrule
No more than 100 characters
Rule provider
AWS VariableCheck
Variable
The variable with the resolved value that will be checked against the provided string value.
#{SourceVariables.FullRepositoryName}
Variables must use the following syntax: #{namespace.variable_key}.
Value
The string value to check against the resolved variable value.
/MyGitHubRepo
Operator
Operator for the comparison.
Equals
Checks whether the variable is equal to the string value.
Not equals
Checks whether the variable is not equal to the string value.
Contains
Checks whether the variable contains the string value as a substring.
Matches
Checks whether the variable matches a given regex expression as the string value.



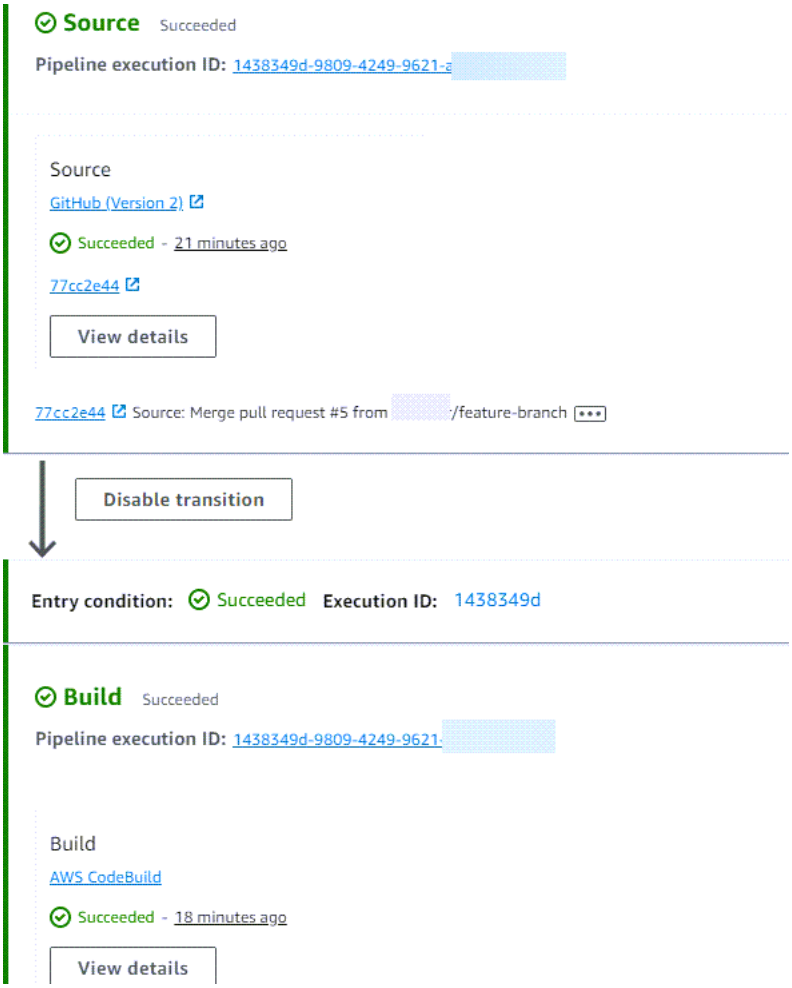
3. Choose **Save**.

   Choose **Done**.

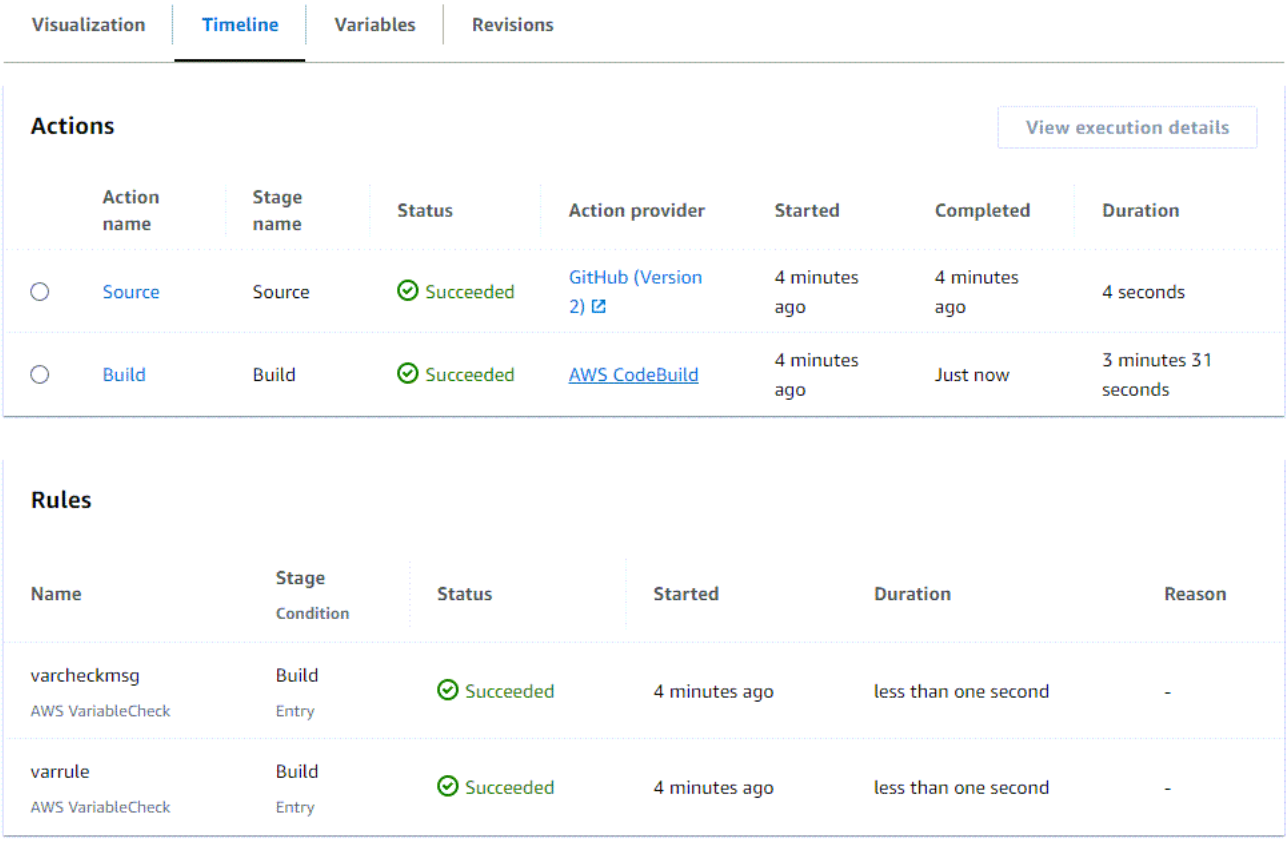
Step 3: Run the pipeline and view resolved variables

In this step, you view the resolved values and results of the variable check rule.

1. View the resolved run after the rule check is successful as shown in the following example.



2. View the variable information on the **Timeline** tab.


Visualization
Timeline
Variables
Revisions
Actions
View execution details
Action name
Stage name
Status
Action provider
Started
Completed
Duration
Source
Source
Succeeded
GitHub (Version 2)
4 minutes ago
4 minutes ago
4 seconds
Build
Build
Succeeded
AWS CodeBuild
4 minutes ago
Just now
3 minutes 31 seconds
Rules
Name
Stage
Condition
Status
Started
Duration
Reason
varcheckmsg
AWS VariableCheck
Build
Entry
Succeeded
4 minutes ago
less than one second
-
varrule
AWS VariableCheck
Build
Entry
Succeeded
4 minutes ago
less than one second
-

CodePipeline use cases

The following sections describe use cases for CodePipeline.

Topics

- Use cases for CodePipeline

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.

Topics

- Use CodePipeline with Amazon S3, AWS CodeCommit, and AWS CodeDeploy
- Use CodePipeline with third-party action providers (GitHub and Jenkins)
- Use CodePipeline to compile, build, and test code with CodeBuild
- Use CodePipeline with Amazon ECS for continuous delivery of container-based applications to the cloud
- Use CodePipeline with Elastic Beanstalk for continuous delivery of web applications to the cloud
- Use CodePipeline with AWS Lambda for continuous delivery of Lambda-based and serverless applications
- Use CodePipeline with CloudFormation templates for continuous delivery to the cloud

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) 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) 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 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 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 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 CloudFormation templates for continuous delivery to the cloud

You can use CloudFormation with CodePipeline for continuous delivery and automation. For more information, see Continuous Delivery with CodePipeline. CloudFormation is also used to create the templates for pipelines created in AWS CodeStar.

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 (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Milan)*

- Europe (Paris)
- Europe (Stockholm)
- Asia Pacific (Hong Kong)*
- Asia Pacific (Mumbai)
- Asia Pacific (Tokyo)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- South America (São Paulo)
- AWS GovCloud (US-West)

* You must enable this Region before you can use it.

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 `eusc-de-east-1` 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. Make sure that the route table for private subnets points to the virtual private gateway.
6. 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."

Define CI/CD pipelines with stages and actions

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.

Before you can create a pipeline, you must first complete the steps in Getting started with CodePipeline.

For more information about pipelines, see CodePipeline concepts, CodePipeline tutorials, and, if you want to use the AWS CLI to create a pipeline, Create a pipeline, stages, and actions. To view a list of pipelines, see View pipelines and details in CodePipeline.

Topics

- Create a pipeline, stages, and actions
- Edit a pipeline in CodePipeline
- View pipelines and details in CodePipeline
- Delete a pipeline in CodePipeline
- Create a pipeline in CodePipeline that uses resources from another AWS account
- Migrate polling pipelines to use event-based change detection
- Create the CodePipeline service role
- Tagging resources
- Tag a pipeline in CodePipeline
- Create a notification rule

Create a pipeline, stages, and actions

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.

Important

As part of creating a pipeline, an S3 artifact bucket provided by the customer will be used by CodePipeline for artifacts. (This is different from the bucket used for an S3 source action.) If the S3 artifact bucket is in a different account from the account for your pipeline, make sure that the S3 artifact bucket is owned by AWS accounts that are safe and will be dependable.

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.

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.

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 and branch or your S3 source bucket.

Note

When you use the console to create or edit a pipeline, the change detection 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, see CodeCommit source actions and EventBridge.

Topics

- Create a custom pipeline (console)
- Create a pipeline (CLI)
- Create a pipeline from static templates

Create a custom pipeline (console)

To create a custom 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

1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home.
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 creation option** page, under **Creation options**, choose the **Build custom pipeline** option. Choose **Next**.
4. On the **Step 2: 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.

5. In **Pipeline type**, choose one of the following options. Pipeline types differ in characteristics and price. For more information, see Pipeline types.

   - **V1** type pipelines have a JSON structure that contains standard pipeline, stage, and action-level parameters.
   - **V2** type pipelines have the same structure as a V1 type, along with additional parameter support, such as triggers on Git tags and pipeline-level variables.

6. In **Service role**, do one of the following:

   - Choose **New service role** to allow CodePipeline to create a new service role in IAM.
   - 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.

   For more information about the service role and its policy statement, see Manage the CodePipeline service role.

7. (Optional) Under **Variables**, choose **Add variable** to add variables at the pipeline level.

   For more information about variables at the pipeline level, see Variables reference. For a tutorial with a pipeline-level variable that is passed at the time of the pipeline execution, see Tutorial: Use pipeline-level variables.

> **Note**
>
> While it is optional to add variables at the pipeline level, for a pipeline specified with variables at the pipeline level where no values are provided, the pipeline execution will fail.

8. (Optional) Expand **Advanced settings**.
9. 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 and CodePipeline pipeline structure reference.

10. In **Encryption key**, do one of the following:

    a. To use the CodePipeline default AWS KMS key to encrypt the data in the pipeline artifact store (S3 bucket), choose **Default AWS Managed Key**.

    b. To use your customer managed key to encrypt the data in the pipeline artifact store (S3 bucket), choose **Customer Managed Key**. Choose the key ID, key ARN, or alias ARN.

11. Choose **Next**.

Step 2: Create a source stage

1. On the **Step 3: Add source stage** page, in **Source provider**, choose the type of repository where your source code is stored, specify its required options. The additional fields display depending on the source provider selected as follows.

   - For **Bitbucket Cloud, GitHub (via GitHub App), GitHub Enterprise Server, GitLab.com, or GitLab self-managed**:

     1. Under **Connection**, choose an existing connection or create a new one. To create or manage a connection for your GitHub source action, see GitHub connections.
     2. Choose the repository you want to use as the source location for your pipeline.

        Choose to add a trigger or filter on trigger types to start your pipeline. For more information about working with triggers, see Add trigger with code push or pull request event types. For more information about filtering with glob patterns, see Working with glob patterns in syntax.
     3. In **Output artifact format**, choose the format for your artifacts.
        - To store output artifacts from the GitHub action using the default method, choose **CodePipeline default**. The action accesses the files from the GitHub 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.

          If you choose this option, you will need to update the permissions for your CodeBuild project service role as shown in Troubleshooting CodePipeline. For a tutorial that shows you how to use the **Full clone** option, see Tutorial: Use full clone with a GitHub pipeline source.

   - 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.
  - In **Output artifact format**, choose the format for your artifacts.
    - To store output artifacts from the CodeCommit action using the default method, choose **CodePipeline default**. The action accesses the files from the CodeCommit 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.

      If you choose this option, you will need to add the `codecommit:GitPull` permission to your CodeBuild service role as shown in Add CodeBuild GitClone permissions for CodeCommit source actions. You will also need to add the `codecommit:GetRepository` permissions to your CodePipeline service role as shown in Add permissions to the CodePipeline service role. For a tutorial that shows you how to use the **Full clone** option, see Tutorial: Use full clone with a GitHub pipeline source.
  - 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.

> **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 Platforms. For more information about deploying revisions with CodeDeploy, see Uploading Your Application Revision and Prepare a Revision.

2. To configure the stage for automatic retry, choose **Enable automatic retry on stage failure**. For more information about automatic retry, see Configure a stage for automatic retry on failure.
3. Choose **Next**.

Step 4: Create a build stage

This step is optional if you plan to create a deployment stage.

1. On the **Step 4: Add build stage** page, do one of the following, and then choose **Next**:

   - Choose **Skip build stage** if you plan to create a test or deployment stage.
   - To choose the Commands action for your build stage, choose **Commands**.

> **Note**
>
> Running the Commands action will incur separate charges in AWS CodeBuild. If you plan to insert build commands as part of a CodeBuild action, go ahead and choose Other build providers, and then select CodeBuild.

In **Commands**, enter the shell commands for your action. For more information about the Commands action, see Commands action reference.

- To choose other build providers such as CodeBuild, choose **Other providers**. 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.
- 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*.

  Under **Build specifications**, the CodeBuild buildspec file is optional, and you can enter commands instead. In **Insert build commands**, enter the shell commands for your action. For more information about considerations for using build commands, see Commands action reference. Choose **Use a buildspec file** if you want to run commands in other phases or if you have a long list of commands.

  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 credentials. 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.

2. To configure the stage for automatic retry, choose **Enable automatic retry on stage failure**. For more information about automatic retry, see Configure a stage for automatic retry on failure.
3. Choose **Next**.

Step 5: Create a test stage

This step is optional if you plan to create a build or deployment stage.

1. On the **Step 5: Add test stage** page, do one of the following, and then choose **Next**:
   - Choose **Skip test stage** if you plan to create a build or deployment stage.
   - In **Test provider**, choose the test action provider and complete the appropriate fields.
2. Choose **Next**.

Step 6: Create a deployment stage

This step is optional if you have already created a build stage.

1. On the **Step 6: Add deploy stage** page, do one of the following, and then choose **Next**:

   - Choose **Skip deploy stage** if you created a build or test stage in the previous steps.

     > **Note**
     >
     > This option does not appear if you have already skipped the build or test 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.

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.

  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.

- **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 Service Catalog with your pipeline, see Tutorial: Create a pipeline that deploys to Service Catalog.

- **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.

- **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 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 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.

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.

2. To configure the stage for automatic retry, choose **Enable automatic retry on stage failure**. For more information about automatic retry, see Configure a stage for automatic retry on failure.
3. To configure the stage for automatic rollback, choose **Configure automatic rollback on stage failure**. For more information about automatic rollback, see Configure a stage for automatic rollback.
4. Choose **Next step**.

Step 7: Review the pipeline

- On the **Step 7: 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 pipelines and details in CodePipeline. For more information about making changes to your pipeline, see Edit a pipeline in CodePipeline.

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 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. 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. 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. (Optional) You can add one or more variables at the pipeline level. You can reference this value in configuration of CodePipeline actions. You can add the variable names and values when you create the pipeline, and you can also choose to assign values when you start the pipeline in the console.

> **Note**
>
> While it is optional to add variables at the pipeline level, for a pipeline specified with variables at the pipeline level where no values are provided, the pipeline execution will fail.

A variable at the pipeline level is resolved at run time of pipeline. All variables are immutable, meaning that they cannot be updated after a value is assigned. Variables at the pipeline level with resolved values will display in the history for each execution.

You provide variables at the pipeline level using the variables attribute in the pipeline structure. In the following example, the variable `Variable1` has a value of `Value1`.

```
      "variables": [
           {
               "name": "Timeout",
               "defaultValue": "1000",
               "description": "description"
           }
       ]
```

Add this structure to your pipeline JSON, or to the example JSON in the following step. For more information about variables, including namespace information, see Variables reference.

3. 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:

```
{
    "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": "amzn-s3-demo-source-bucket",
                            "S3ObjectKey": "ExampleCodePipelineSampleBundle.zip",
                            "PollForSourceChanges": "false"
                        },
                        "runOrder": 1
                    }
                ]
            },
            {
                "name": "Staging",
                "actions": [
                    {
                        "inputArtifacts": [
                            {
                                "name": "MyApp"
                            }
                        ],
                        "name": "Deploy-CodeDeploy-Application",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
```

```
                        "version": "1",
                        "provider": "CodeDeploy"
                    },
                    "outputArtifacts": [],
                    "configuration": {
                        "ApplicationName": "CodePipelineDemoApplication",
                        "DeploymentGroupName": "CodePipelineDemoFleet"
                    },
                    "runOrder": 1
                }
            ]
        }
    ],
    "artifactStore": {
        "type": "S3",
        "location": "codepipeline-us-east-2-250656481468"
    },
    "name": "MyFirstPipeline",
    "version": 1,
    "variables": [
        {
            "name": "Timeout",
            "defaultValue": "1000",
            "description": "description"
        }
      ]
    },
    "triggers": [
        {
            "providerType": "CodeStarSourceConnection",
            "gitConfiguration": {
                "sourceActionName": "Source",
                "push": [
                    {
                        "tags": {
                            "includes": [
                                "v1"
                            ],
                            "excludes": [
                                "v2"
                            ]
                        }
                    }
                ]
```

```
                }
            }
        ]
    "metadata": {
        "pipelineArn": "arn:aws:codepipeline:us-
east-2:80398EXAMPLE:MyFirstPipeline",
        "updated": 1501626591.112,
        "created": 1501626591.112
    },
    "tags": [{
      "key": "Project",
      "value": "ProjectA"
    }]
}
```

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.

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. 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.

4. 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.

   - 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.

5. 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 pipelines and details in CodePipeline.
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 an EventBridge rule for a CodeCommit source (CLI).
   - For a pipeline with an Amazon S3 source, you must manually create the CloudWatch Events rule and AWS CloudTrail trail, as described in Connecting to Amazon S3 source actions that use EventBridge and AWS CloudTrail.

Create a pipeline from static templates

You can create a pipeline in the console that uses a template to configure a pipeline with source code and properties that you specify. You must provide the source file location and information about the source providers you will use for your actions. You can specify a source action for Amazon ECR or any third-party repository supported by AWS CodeConnections, such as GitHub.

The template will create a stack in CloudFormation for your pipeline that includes the following resources:

- A pipeline is created with the V2 pipeline type. In **Pipeline type**, choose one of the following options. Pipeline types differ in characteristics and price. For more information, see Pipeline types.
- A service role is created for your pipeline and referenced in the template.
- An artifact store is created using 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.

To view the collection of open source starter templates that are used for the static template creation wizard, see the repository at https://github.com/aws/codepipeline-starter-templates.

When you use static templates to create a pipeline, the pipeline structure is configured in each template according to the needs of the use case. For example, the template for a deployment to CloudFormation is used as an example in this procedure. The template generates a pipeline named **DeployToCloudFormationService** with the following structure:

- A build stage that contains a source action with configuration that you specify in the wizard.
- A deployment stage with a deploy action and associated resource stack in CloudFormation.

When you use a static template to create a pipeline, 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: Access the console

1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home.
2. On the **Welcome** page, choose **Create pipeline**.

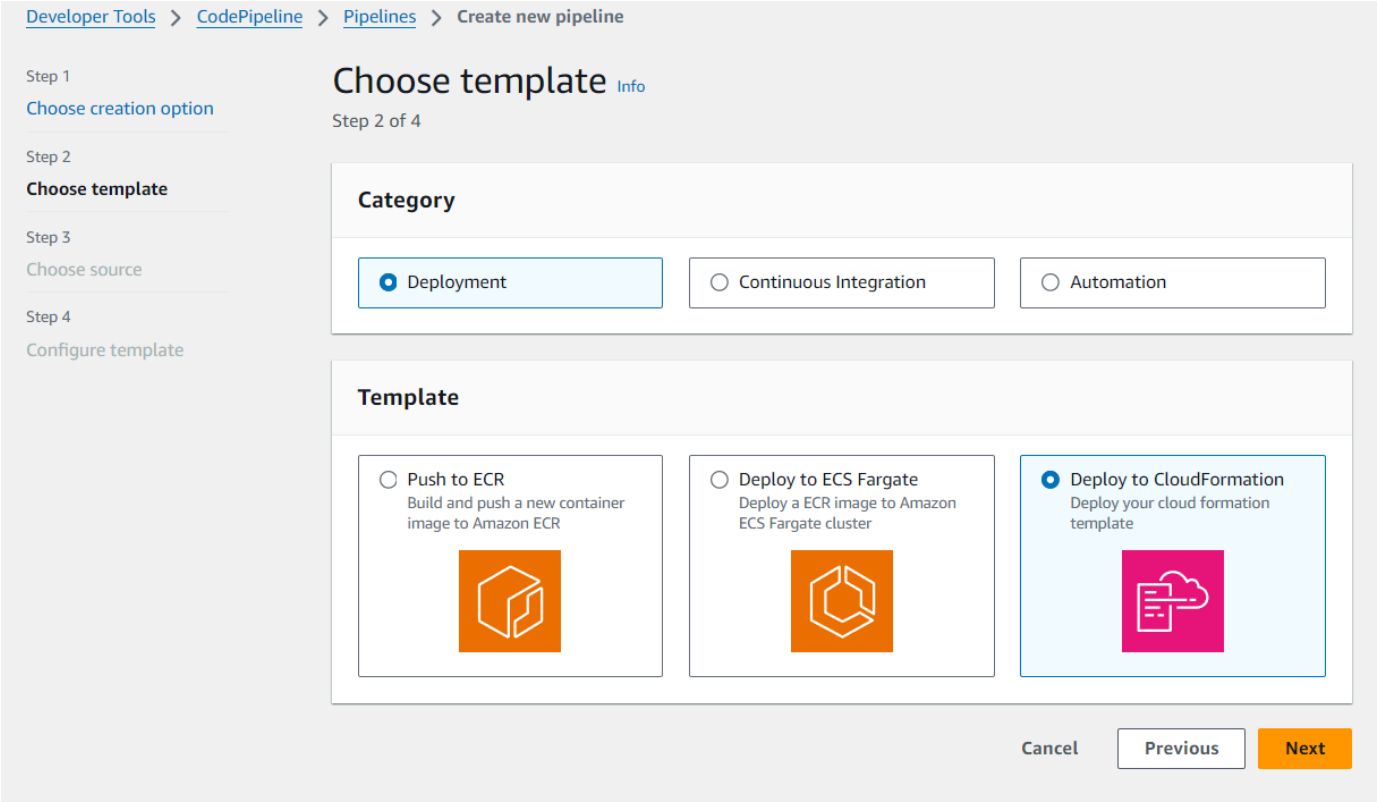
   If this is your first time using CodePipeline, choose **Get Started**.

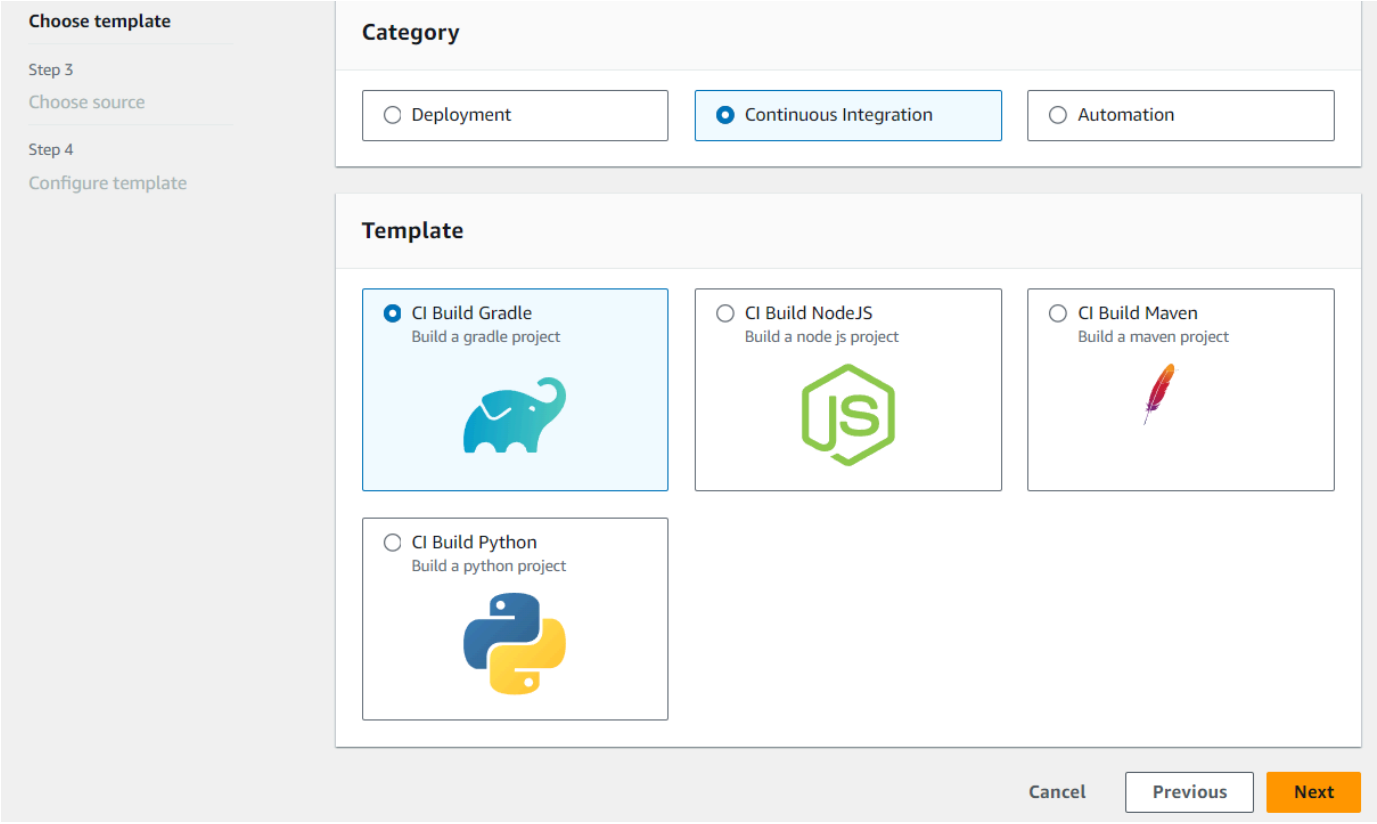
Step 2: Choose template

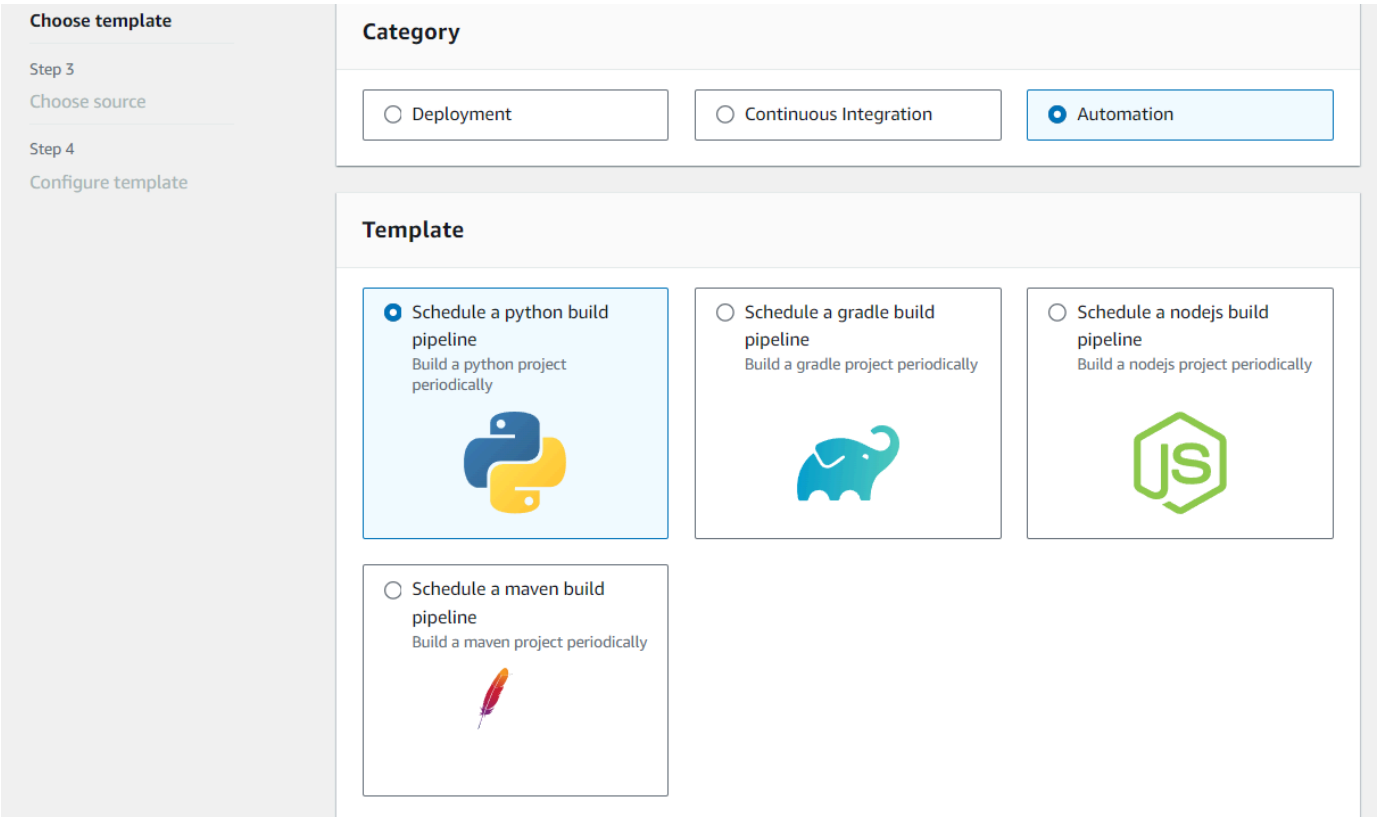
Choose a template to create a pipeline with a deployment stage, automation, or a CI pipeline.

1. On the **Step 1: Choose creation option** page, do one of the following, and then choose **Next**:

   - Choose **Deployment** if you plan to create a deployment stage. View the options for templates that deploy to ECR or CloudFormation. For this example, choose **Deployment**, and then choose to deploy to CloudFormation.
   - Choose **Continuous Integration** if you plan to create a CI pipeline. View the options for CI pipelines, such as building to Gradle.
   - Choose **Automation** if you plan to create an automated pipeline. View the options for automation, such as scheduling a python build.

2. 


Choose template
Step 3
Choose source
Step 4
Configure template
Category
Deployment
Continuous Integration
Automation
Template
CI Build Gradle
Build a gradle project
CI Build NodeJS
Build a node js project
CI Build Maven
Build a maven project
CI Build Python
Build a python project
Cancel
Previous
Next


Choose template
Step 3
Choose source
Step 4
Configure template
Category
Deployment
Continuous Integration
Automation
Template
Schedule a python build pipeline
Build a python project periodically
Schedule a gradle build pipeline
Build a gradle project periodically
Schedule a nodejs build pipeline
Build a nodejs project periodically
Schedule a maven build pipeline
Build a maven project periodically

Step 3: Choose source

- On the **Step 3: Choose source** page, in **Source provider**, choose the provider for the repository where your source code is stored, specify its required options, and then choose **Next step**.

  - For **Bitbucket Cloud, GitHub (via GitHub App), GitHub Enterprise Server, GitLab.com, or GitLab self-managed**:

    1. Under **Connection**, choose an existing connection or create a new one. To create or manage a connection for your GitHub source action, see GitHub connections.
    2. Choose the repository you want to use as the source location for your pipeline.

       Choose to add a trigger or filter on trigger types to start your pipeline. For more information about working with triggers, see Add trigger with code push or pull request event types. For more information about filtering with glob patterns, see Working with glob patterns in syntax.
    3. In **Output artifact format**, choose the format for your artifacts.
       - To store output artifacts from the GitHub action using the default method, choose **CodePipeline default**. The action accesses the files from the GitHub 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.

         If you choose this option, you will need to update the permissions for your CodeBuild project service role as shown in Troubleshooting CodePipeline. For a tutorial that shows you how to use the **Full clone** option, see Tutorial: Use full clone with a GitHub pipeline source.

  - 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.

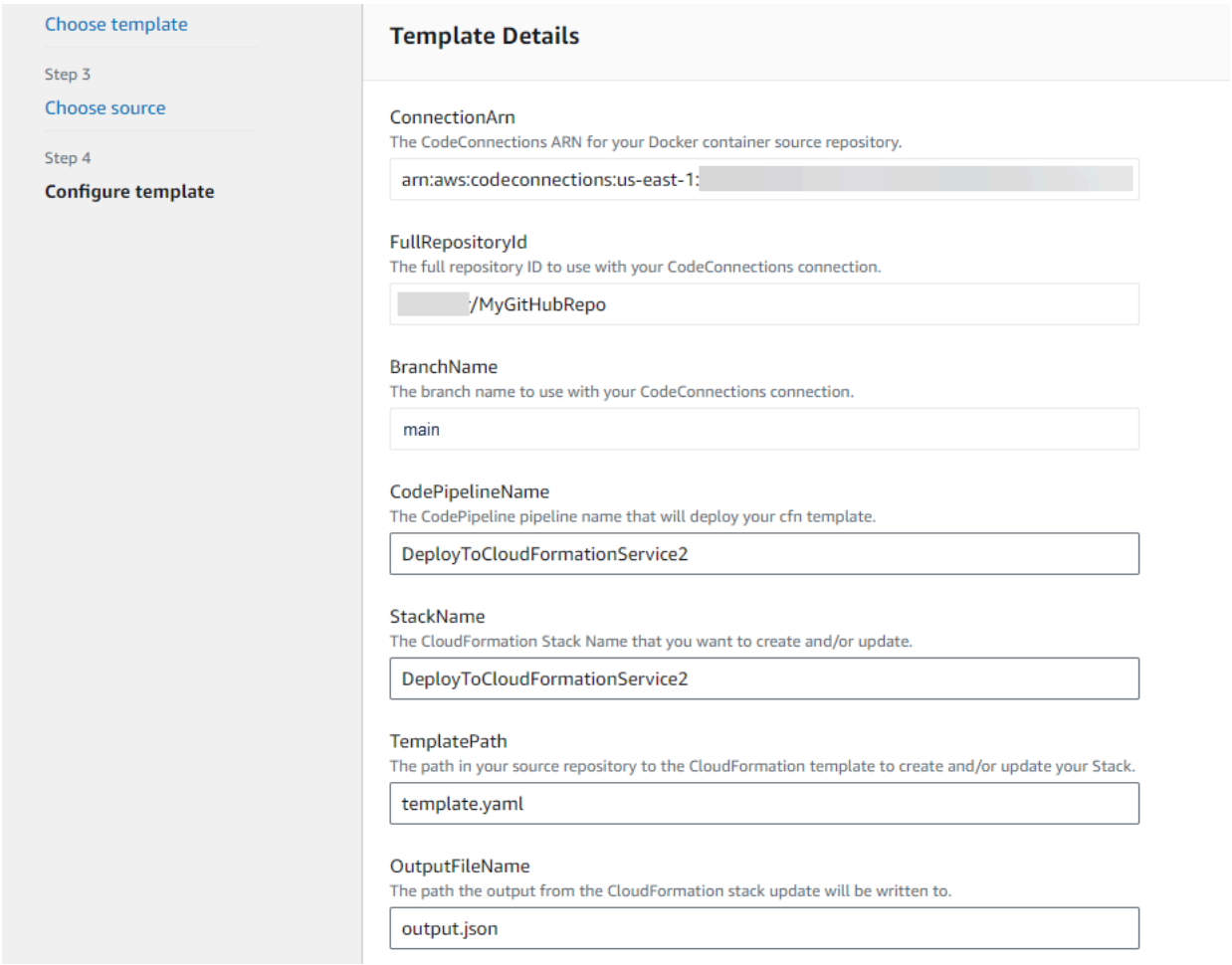
    When you include an Amazon ECR source stage in your pipeline, the source action generates an `imageDetail.json` file as an output artifact when you commit a change. For

information about the `imageDetail.json` file, see imageDetail.json file for Amazon ECS blue/green deployment actions.

> **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 Platforms. For more information about deploying revisions with CodeDeploy, see Uploading Your Application Revision and Prepare a Revision.

Step 4: Configure template

For this example, the deployment to CloudFormation was selected. In this step, add configuration for your template.



1. In **Step 4: Configure template**, in **Stack name**, enter a name for your pipeline.
2. Edit the placeholder IAM policy for the permissions that apply for your template.
3. Choose **Create pipeline from template**
4. A message displays stating that your pipeline resources are being created.

Step 5: View the pipeline

- Now that you've created your pipeline, you can view it in the CodePipeline console and view the stack in CloudFormation. The pipeline starts to run after you create it. For more information, see View pipelines and details in CodePipeline. For more information about making changes to your pipeline, see Edit a pipeline in CodePipeline.

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.

You can edit your pipeline type, variables, and triggers using the pipeline edit page. You can also add or change stages and actions in your pipeline.

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.

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.

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.

> **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 an EventBridge rule for a CodeCommit source (CLI). For more information about using the CLI to create or update an Amazon S3 pipeline, see Create an EventBridge rule for an Amazon S3 source (CLI).

Topics

- Edit a pipeline (console)
- Edit a pipeline (AWS CLI)

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.

When you update a pipeline, CodePipeline gracefully completes all the running actions and then fails the stages and pipeline executions where the running actions were completed. When a pipeline is updated, you will need to re-run your pipeline. For more information on running a pipeline, see Start a pipeline manually.

To edit 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 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. To edit the pipeline type, choose **Edit** on the **Edit: Pipeline properties** card. Choose one of the following options, and then choose **Done**.

   - **V1** type pipelines have a JSON structure that contains standard pipeline, stage, and action-level parameters.
   - **V2** type pipelines have the same structure as a V1 type, along with additional parameter support, such as triggers and pipeline-level variables.

   Pipeline types differ in characteristics and price. For more information, see Pipeline types.
5. To edit the pipeline variables, choose **Edit variables** on the **Edit: Variables** card. Add or change variables for the pipeline level, and then choose **Done**.

   For more information about variables at the pipeline level, see Variables reference. For a tutorial with a pipeline-level variable that is passed at the time of the pipeline execution, see Tutorial: Use pipeline-level variables.

> **Note**
>
> While it is optional to add variables at the pipeline level, for a pipeline specified with variables at the pipeline level where no values are provided, the pipeline execution will fail.

6. To edit the pipeline triggers, choose **Edit triggers** on the **Edit: Triggers** card. Add or change triggers, and then choose **Done**.

   For more information about adding triggers, see the steps for creating a connection to Bitbucket Cloud, GitHub (via GitHub App), GitHub Enterprise Server, GitLab.com, or GitLab self-managed, such as GitHub connections.

7. To edit stages and actions 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.
   - 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.
   - To configure a stage to roll back automatically on failure, choose **Edit stage**, and then choose the checkbox **Configure automatic rollback on stage failure**.

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 examples of adding action providers and using the default fields for each provider, see Create a custom pipeline (console).

   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.

8. 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.

9. 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 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.

When you update a pipeline, CodePipeline gracefully completes all the running actions and then fails the stages and pipeline executions where the running actions were completed. When a pipeline is updated, you will need to re-run your pipeline. For more information on running a pipeline, see Start a pipeline manually.

> **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:

```
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.

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

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

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.

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. 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.

   - 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 an EventBridge rule for a CodeCommit source (CLI).
   - For an Amazon S3 source, you must create the CloudWatch Events rule and AWS CloudTrail trail, as described in Connecting to Amazon S3 source actions that use EventBridge and AWS CloudTrail.

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.

```
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 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.

For more information about the structure of a pipeline and expected values, see CodePipeline pipeline structure reference and AWS CodePipeline API Reference.

View pipelines and details 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 pipelines (console)
- View action details in a pipeline (console)
- View the pipeline ARN and service role ARN (console)
- View pipeline details and history (CLI)
- View rule results for stage conditions in execution history

View pipelines (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.

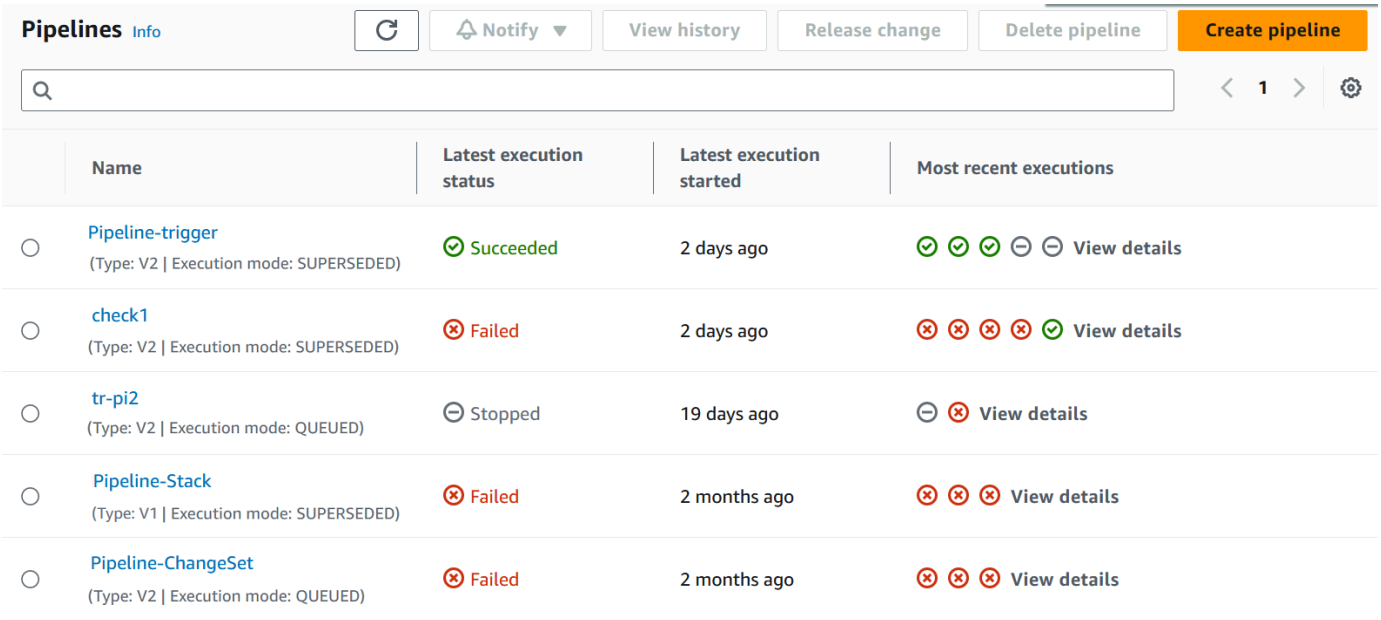
To view pipelines

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

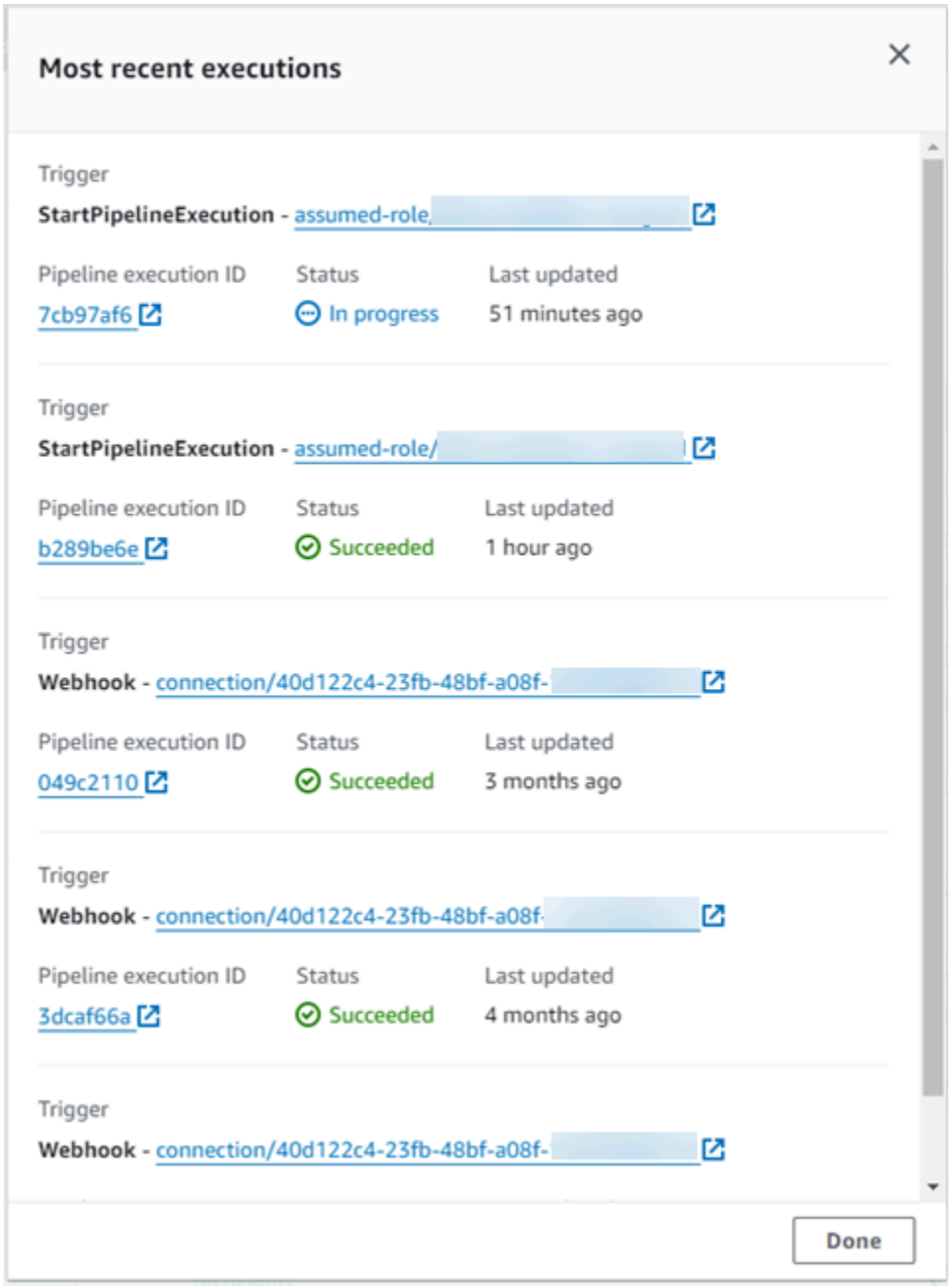
   The **Pipelines** page displays. A list of all your pipelines for that Region are shown.

   The name, type, status, version, creation date, and date when last modified of all pipelines associated with your AWS account are displayed, along with the most recently started execution time.

2. The status for the five most recent executions is shown.



   Choose **View details** next to a specific row to display a details dialog box listing the most recent executions.

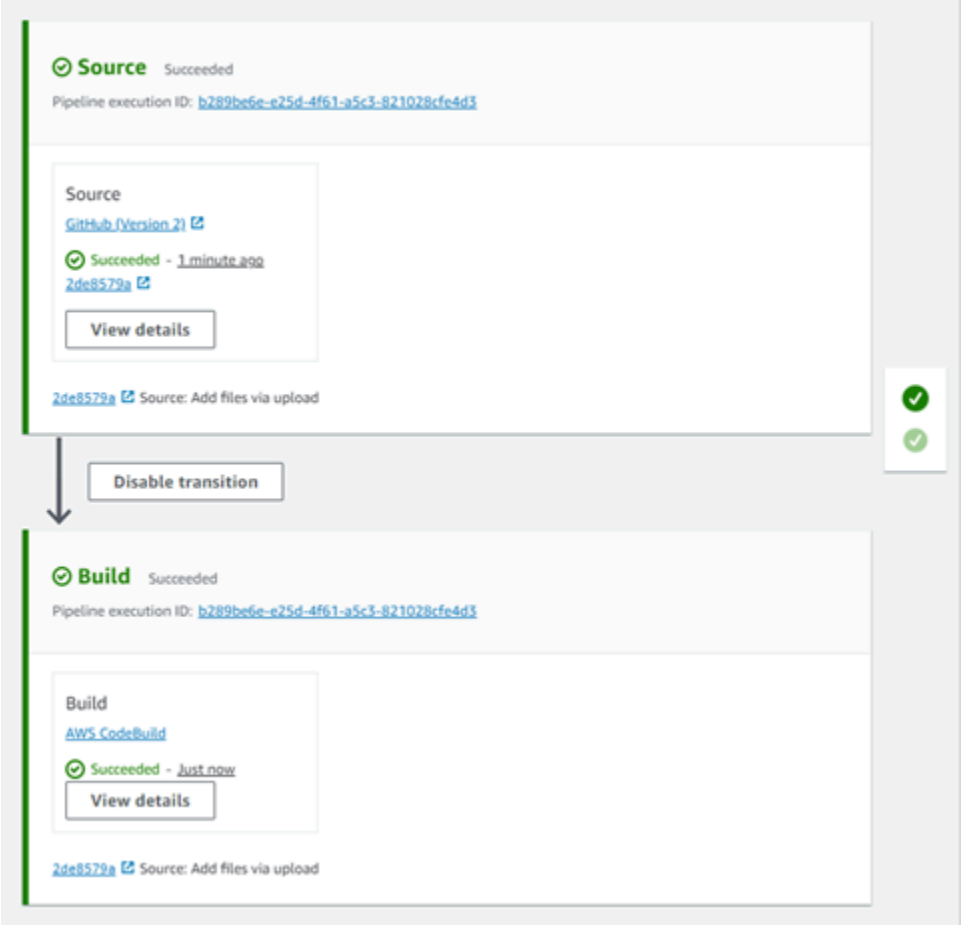


To view details about the most recent executions for the pipeline, choose **View history**. For past executions, you can view revision details associated with source artifacts, such as execution IDs, status, start and end times, duration, and commit IDs and messages.

> **Note**
>
> For a pipeline in PARALLEL execution mode, the main pipeline view does not show the pipeline structure or in-progress executions. For a pipeline in PARALLEL execution mode, you access the pipeline structure by choosing the ID for the execution you

want to view from the execution history page. Choose **History** in the left navigation, choose the execution ID for the parallel execution, and then view the pipeline on the **Visualization** tab.

3. To see details for a single pipeline, in **Name**, choose the 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.
- A **View details** button that opens a dialog box with details about the action execution, logs, and action configuration.

> **Note**
>
> The **Logs** tab is available for CodeBuild and CloudFormation actions that have run in the account of the pipeline.

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 for an action 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.
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. The status from the last time the action ran, including the results of that action (**Succeeded** or **Failed**) is displayed.

View action details in a pipeline (console)

You can view details for a pipeline, including details for actions in each stage.

> **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 action details in a pipeline

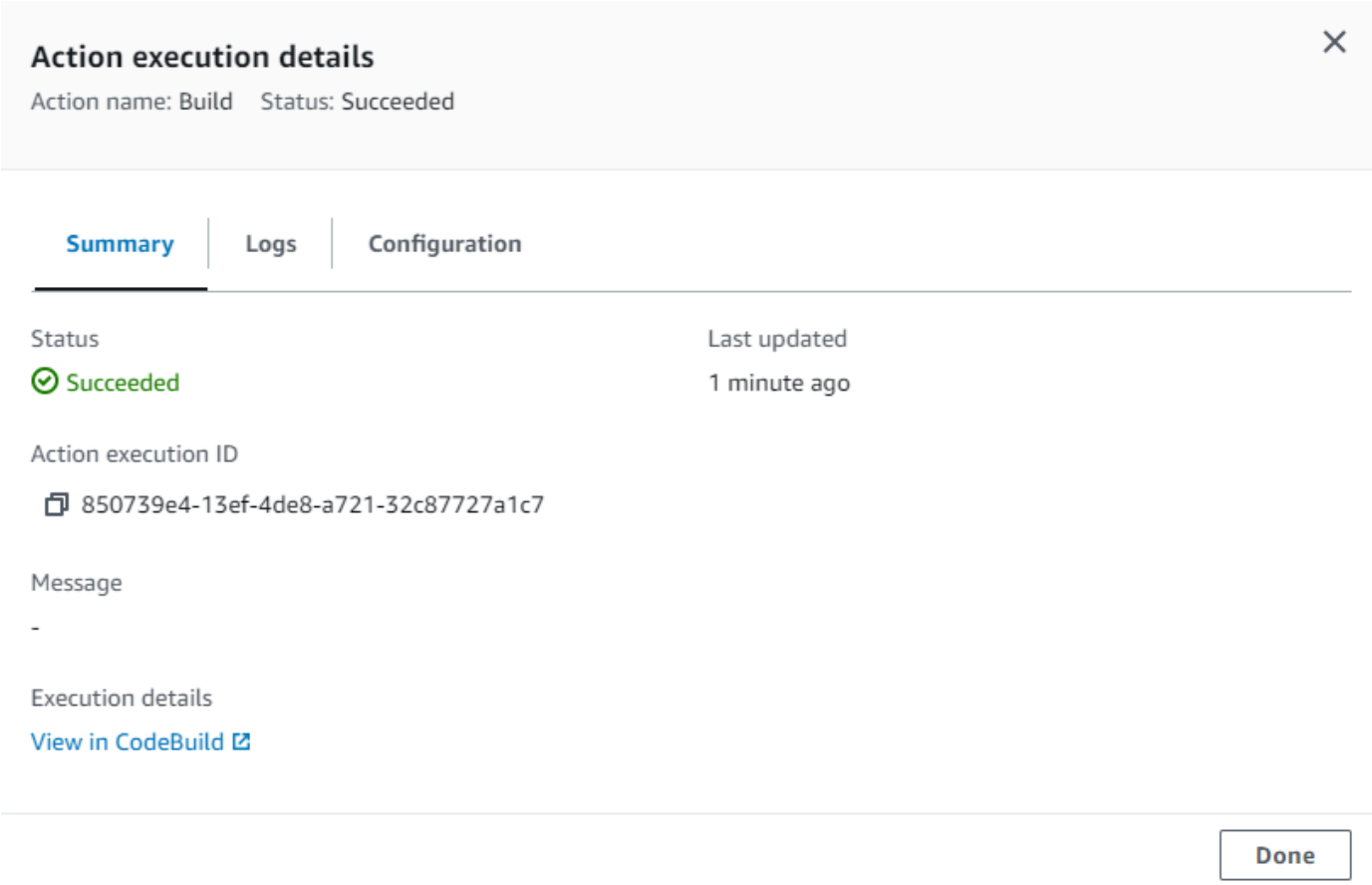
1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home.

   The **Pipelines** page displays.

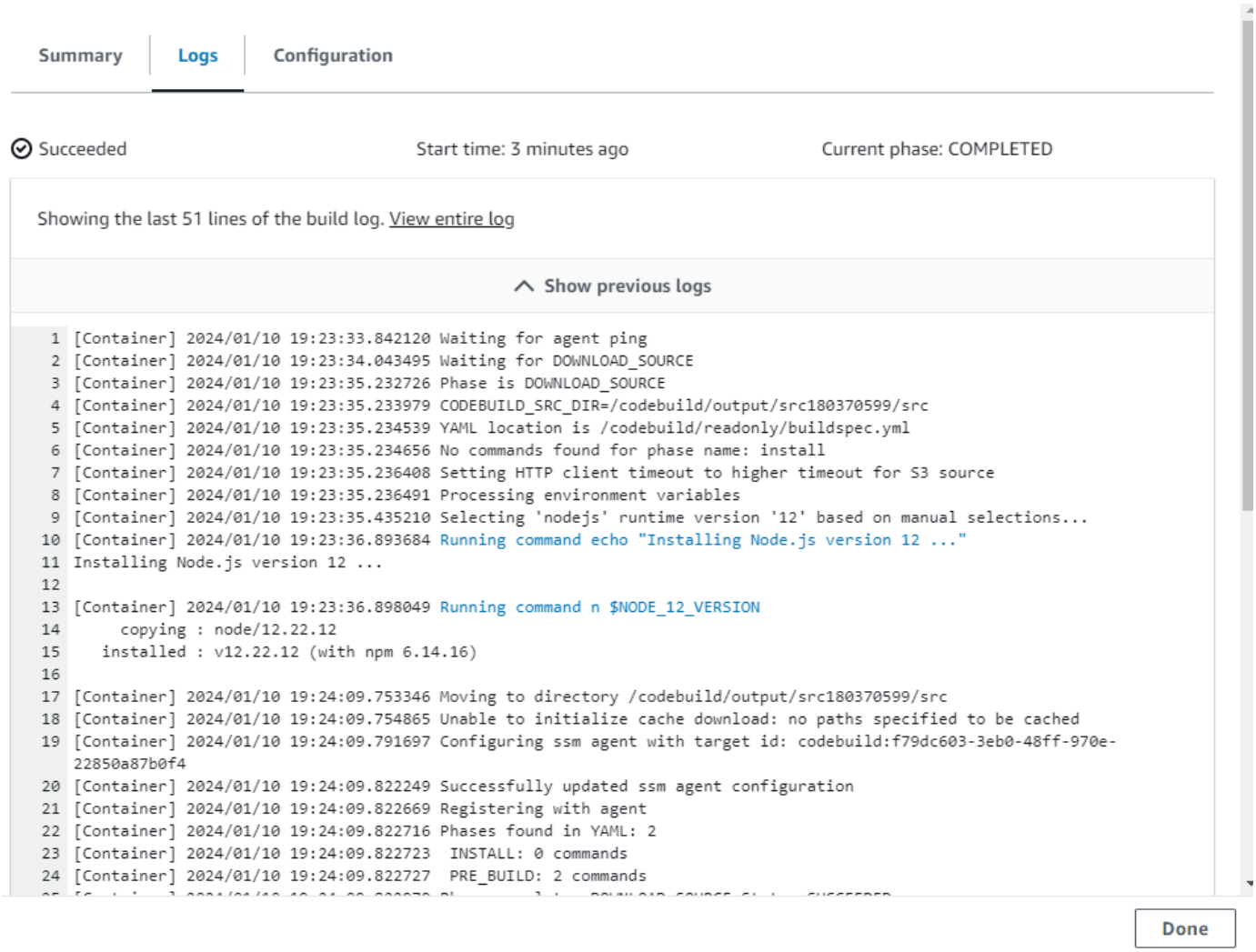
2. On any action, choose **View details** to open a dialog box with details about the action execution, logs, and action configuration.

   > **Note**
   >
   > The **Logs** tab is available for CodeBuild and CloudFormation actions.

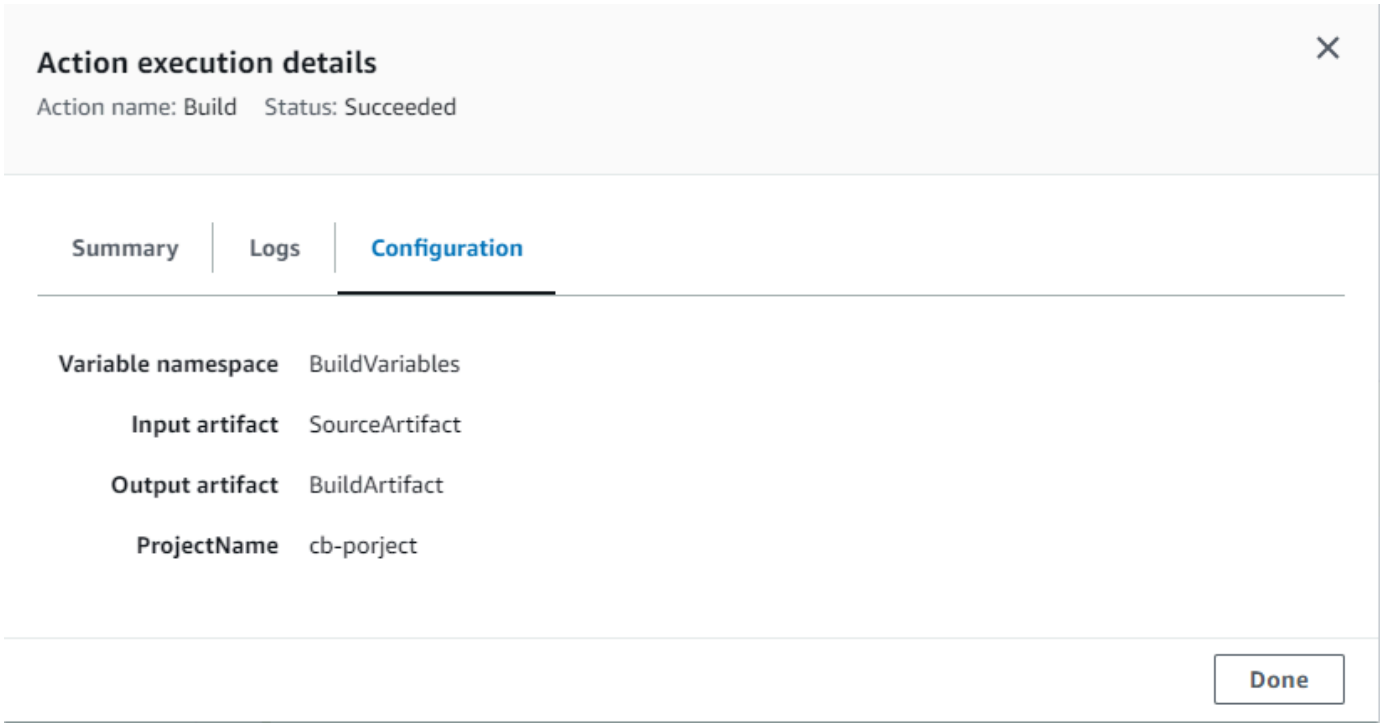
3. To see the action summary for an action in a stage of a pipeline, choose **View details** on the action, and then choose the **Summary** tab.



4. To see the action logs for an action with logs, choose **View details** on the action, and then choose the **Logs** tab.



5. To see the configuration details for an action, choose the **Configuration** tab.



View the pipeline ARN and service role ARN (console)

You can use the console to view pipeline settings, such as the pipeline ARN, the service role ARN, and the pipeline artifact store.

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 will be displayed.
2. Choose the name of your pipeline, and then choose **Settings** in the left-hand navigation pane. The page shows the following:

   - The pipeline name
   - The pipeline Amazon Resource Name (ARN)

     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`

- The CodePipeline service role ARN for your pipeline
- The pipeline version
- The name and location of the artifact store for the pipeline

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.

1. Open a terminal (Linux, macOS, or Unix) or command prompt (Windows) and use the AWS CLI to run the **list-pipelines** command:

   ```
   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:

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

   This command returns the structure of the pipeline.

View rule results for stage conditions in execution history

You can view the rule results for an execution where a stage condition ran a rule and engaged a result for the stage, such as rollback or failure.

Valid status values for conditions and rules are as follows: `InProgress | Failed | Errored | Succeeded | Cancelled | Abandoned | Overridden`

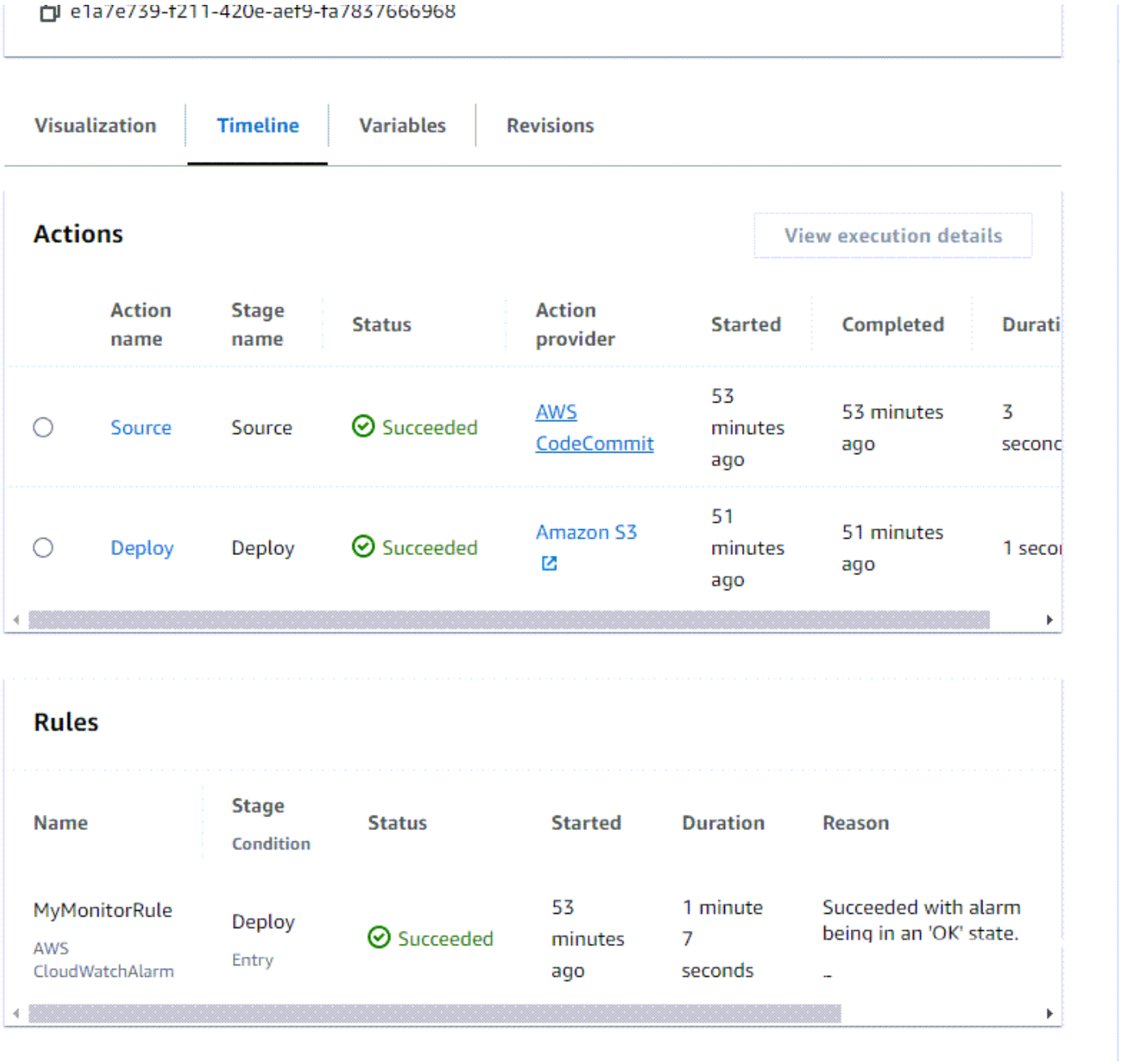
View rule results for stage conditions in execution history (console)

You can use the console to view the rule results for an execution where a stage condition ran a rule and engaged a result for the stage.

View rule results for stage conditions (console)

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 view.
3. Choose **History**, and then choose the execution. On the history page, choose the **Timeline** tab. Under **Rules**, view the rule results for the execution.



View rule results for stage conditions with `list-rule-executions` (CLI)

You can use the CLI to view the rule results for an execution where a stage condition ran a rule and engaged a result for the stage.

- Open a terminal (Linux, macOS, or Unix) or command prompt (Windows) and use the AWS CLI to run the **list-rule-executions** command for a pipeline named *`MyPipeline`*:

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

This command returns a list of all of the completed rule executions associated with the pipeline.

The following example shows the returned data for a pipeline with a stage condition where the rule is named *MyMonitorRule*.

```
{
    "ruleExecutionDetails": [
        {
            "pipelineExecutionId": "e1a7e739-f211-420e-aef9-fa7837666968",
            "ruleExecutionId": "3aafc0c7-0e1c-44f1-b357-d1b16a28e483",
            "pipelineVersion": 9,
            "stageName": "Deploy",
            "ruleName": "MyMonitorRule",
            "startTime": "2024-07-29T15:55:01.271000+00:00",
            "lastUpdateTime": "2024-07-29T15:56:08.682000+00:00",
            "status": "Succeeded",
            "input": {
                "ruleTypeId": {
                    "category": "Rule",
                    "owner": "AWS",
                    "provider": "CloudWatchAlarm",
                    "version": "1"
                },
                "configuration": {
                    "AlarmName": "CWAlarm",
                    "WaitTime": "1"
                },
                "resolvedConfiguration": {
                    "AlarmName": "CWAlarm",
                    "WaitTime": "1"
                },
                "region": "us-east-1",
                "inputArtifacts": []
            },
            "output": {
                "executionResult": {
                    "externalExecutionSummary": "Succeeded with alarm 'CWAlarm'
 being i
```

```
n an 'OK' state."
                }
            }
        }
```

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)
- Delete a pipeline (CLI)

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

Aliases are recognized only in the account that created the KMS key. For cross-account actions, you can only use the key ID or key ARN to identify the key. Cross-account actions involve using the role from the other account (AccountB), so specifying the key ID will use the key from the other account (AccountB).

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 (*CodePipeline_Service_Role*) 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 (*CodePipeline_Service_Role*).
- 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 the CodePipeline service role in *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
- Step 1: Set up account policies and roles
- Step 2: Edit the pipeline

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 the role or roles 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, CodePipeline_Service_Role). 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-2:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE***). 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)
- Configure policies and roles in the account that owns the AWS resource (AccountB)

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://eusc-de-east-1.console.amazonaws-eusc.eu/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*`-`*1234567EXAMPLE*, where *region* is the AWS Region in which you created the pipeline and *1234567EXAMPLE* 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, create a policy that 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.
6. Choose **Save**, and then close the policy editor.
7. Choose **Save** to save the permissions for the Amazon S3 bucket.

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://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. In the navigation pane, choose **Roles**.
3. In the list of roles, under **Role Name**, choose the name of the service role for CodePipeline.
4. On the **Permissions** tab, choose **Add inline policy**.
5. Choose the **JSON** tab, and enter 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-eusc:iam::111122223333:role/*"
        ]
    }
}
```

6. Choose **Review policy**.
7. In **Name**, enter a name for this policy. Choose **Create 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 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 the CodePipeline service role in *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://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. In the navigation pane, 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 **Permissions** tab, choose **Add inline policy**.
5. Choose the **JSON** tab, and enter 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-eusc:s3:::amzn-s3-demo-bucket/*"
```

```
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws-eusc:s3:::amzn-s3-demo-bucket"
      ]
    }
  ]
}
```

6. Choose **Review policy**.
7. In **Name**, enter a name for this policy. Choose **Create policy**.
8. Create a second policy for AWS KMS where ***arn:aws:kms:eusc-de-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-eusc:kms:eusc-de-
east-1:111122223333:key/2222222-3333333-4444-556677EXAMPLE"
            ]
        }
    ]
}
```

> **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.

9. Choose **Review policy**.
10. In **Name**, enter a name for this policy. Choose **Create policy**.

Now create an IAM role to use for cross-account access, and configure it so that the CodePipeline service role in *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

1. Sign in to the AWS Management Console with *AccountB* and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam.
2. In the navigation pane, choose **Roles**. Choose **Create role**.
3. Under **Select type of trusted entity**, choose **Another AWS account**. Under **Specify accounts that can use this role**, in **Account ID**, enter the AWS account ID for the account that will create the pipeline in CodePipeline (*AccountA*), and then choose **Next: Permissions**.

> **Important**
>
> This step creates the trust relationship policy between *AccountB* and *AccountA*. However, this grants root level access to the account, and CodePipeline recommends scoping it down to the CodePipeline service role in *AccountA*. Follow step 16 to restrict permissions.

4. Under**Attach permissions policies**, choose **AmazonS3ReadOnlyAccess**, and then choose **Next: Tags**.

> **Note**
>
> This is not the policy you will use. You must choose a policy to complete the wizard.

5. Choose **Next: Review**. 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. Choose **Create Role**.
6. From the list of roles, choose the role you just created (for example, *CrossAccount_Role*) to open the **Summary** page for that role.
7. On the **Permissions** tab, choose **Add inline policy**.
8. Choose the **JSON** tab, and enter 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": "*"
    }
  ]
}
```

9. Choose **Review policy**.
10. In **Name**, enter a name for this policy. Choose **Create policy**.
11. On the **Permissions** tab, choose **Add inline policy**.
12. Choose the **JSON** tab, and enter 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"
        ],
        "Resource": [
          "arn:aws-eusc:s3:::amzn-s3-demo-bucket/*"
        ]
      }
    ]
}
```

13. Choose **Review policy**.
14. In **Name**, enter a name for this policy. Choose **Create policy**.
15. On the **Permissions** tab, find **AmazonS3ReadOnlyAccess** in the list of policies under **Policy Name**, and choose the delete icon (**X**) next to the policy. When prompted, choose **Detach**.
16. Select the **Trust Relationship** tab, and then choose **Edit trust policy**. Choose the **Add a principal** option on the left column. For **Principal type**, choose **IAM Roles**, and then provide the ARN for the CodePipeline service role in *AccountA*. Remove `arn:aws:iam::Account_A:root` from the list for **AWS Principals**, and then choose **Update policy**.

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:eusc-de-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:eusc-de-
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,
                        "roleArn":
 "arn:aws:iam::012ID_ACCOUNT_B:role/CrossAccount_Role"
                    }
                ]
            }
```

> **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:

> **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.

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.
2. Sign in to the AWS Management Console with *AccountA* 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.
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.

Migrate polling pipelines to use event-based change detection

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: event-based change detection and polling. We recommend using event-based change detection for pipelines.

Use the procedures included here to migrate (update) your polling pipelines to the event-based change detection method for your pipeline.

The recommended event-based change detection method for pipelines is determined by the pipeline source, such as CodeCommit. In that case, for example, the polling pipeline would need to migrate to event-based change detection with EventBridge.

How to migrate polling pipelines

To migrate polling pipelines, determine your polling pipelines and then determine the recommended event-based change detection method:

- Use the steps in Viewing polling pipelines in your account to determine your polling pipelines.
- In the table, find your pipeline source type and then choose the procedure with the implementation you want to use to migrate your polling pipeline. Each section contains multiple methods for migration, such as using the CLI or CloudFormation.

How to migrate pipelines to the recommended change detection method

Pipeline source	Recommended event-based detection method	Migration procedures
AWS CodeCommit	EventBridge (recommended).	See Migrate polling pipelines with a CodeCommit source.
Amazon S3	EventBridge and bucket enabled for event notifications (recommended).	See Migrate polling pipelines with an S3 source enabled for events.
Amazon S3	EventBridge and an AWS CloudTrail trail.	See Migrate polling pipelines with an S3 source and CloudTrail trail.

How to migrate pipelines to the recommended change detection method		
Pipeline source	**Recommended event-based detection method**	**Migration procedures**
GitHub (via GitHub App)	Connections (recommended)	See Migrate polling pipelines for a GitHub (via OAuth app) source action to connections.
GitHub (via OAuth app)	Webhooks	See Migrate polling pipelines for a GitHub (via OAuth app) source action to webhooks.

Important

For applicable pipeline action configuration updates, such as pipelines with a GitHub (via OAuth app) action, 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 an EventBridge 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. For more information, see Quotas in AWS CodePipeline.

Viewing polling pipelines in your account

As a first step, use one of the following scripts to determine which pipelines in your account are configured for polling. These are the pipelines to migrate to event-based change detection.

Viewing polling pipelines in your account (script)

Follow these steps to use a script to determine pipelines in your account that are using polling.

1. Open a terminal window, and then do one of the following:

- Run the following command to create a new script named **PollingPipelinesExtractor.sh**.

```
vi PollingPipelinesExtractor.sh
```

- To use a python script, run the following command to create a new python script named **PollingPipelinesExtractor.py**.

```
vi PollingPipelinesExtractor.py
```

2. Copy and paste the following code into the **PollingPipelinesExtractor** script. Do one of the following:

- Copy and paste the following code into the **PollingPipelinesExtractor.sh** script.

```
#!/bin/bash

set +x

POLLING_PIPELINES=()
LAST_EXECUTED_DATES=()
NEXT_TOKEN=null
HAS_NEXT_TOKEN=true
if [[ $# -eq 0 ]] ; then
    echo 'Please provide region name'
    exit 0
fi
REGION=$1


while [ "$HAS_NEXT_TOKEN" != "false" ]; do
    if [ "$NEXT_TOKEN" != "null" ];
        then
            LIST_PIPELINES_RESPONSE=$(aws codepipeline list-pipelines --region
 $REGION --next-token $NEXT_TOKEN)
        else
            LIST_PIPELINES_RESPONSE=$(aws codepipeline list-pipelines --region
 $REGION)
    fi
    LIST_PIPELINES=$(jq -r '.pipelines[].name' <<< "$LIST_PIPELINES_RESPONSE")
    NEXT_TOKEN=$(jq -r '.nextToken' <<< "$LIST_PIPELINES_RESPONSE")
    if [ "$NEXT_TOKEN" == "null" ];
        then
```

```
            HAS_NEXT_TOKEN=false
    fi

    for pipline_name in $LIST_PIPELINES
    do
        PIPELINE=$(aws codepipeline get-pipeline --name $pipline_name --region
 $REGION)
        HAS_POLLABLE_ACTIONS=$(jq '.pipeline.stages[].actions[] |
 select(.actionTypeId.category == "Source") | select(.actionTypeId.owner
 == ("ThirdParty","AWS")) | select(.actionTypeId.provider ==
 ("GitHub","S3","CodeCommit")) | select(.configuration.PollForSourceChanges ==
 ("true",null))' <<< "$PIPELINE")
        if [ ! -z "$HAS_POLLABLE_ACTIONS" ];
        then
            POLLING_PIPELINES+=("$pipline_name")
            PIPELINE_EXECUTIONS=$(aws codepipeline list-pipeline-executions --
pipeline-name $pipline_name --region $REGION)
            LAST_EXECUTION=$(jq -r '.pipelineExecutionSummaries[0]' <<<
 "$PIPELINE_EXECUTIONS")
            if [ "$LAST_EXECUTION" != "null" ];
                then
                    LAST_EXECUTED_TIMESTAMP=$(jq -r '.startTime' <<<
 "$LAST_EXECUTION")
                    LAST_EXECUTED_DATE="$(date -r ${LAST_EXECUTED_TIMESTAMP%.*})"
                else
                    LAST_EXECUTED_DATE="Not executed in last year"
            fi
            LAST_EXECUTED_DATES+=("$LAST_EXECUTED_DATE")
        fi
    done

done

fileName=$REGION-$(date +%s)
printf "| %-30s | %-30s |\n" "Polling Pipeline Name" "Last Executed Time"
printf "| %-30s | %-30s |\n" "_____________________" "___________________"
for i in "${!POLLING_PIPELINES[@]}"; do
  printf "| %-30s | %-30s |\n" "${POLLING_PIPELINES[i]}"
 "${LAST_EXECUTED_DATES[i]}"
  printf "${POLLING_PIPELINES[i]}," >> $fileName.csv
done

printf "\nSaving Polling Pipeline Names to file $fileName.csv."
```

- Copy and paste the following code into the **PollingPipelinesExtractor.py** script.

```
import boto3
import sys
import time
import math

hasNextToken = True
nextToken = ""
pollablePipelines = []
lastExecutedTimes = []
if len(sys.argv) == 1:
    raise Exception("Please provide region name.")
session = boto3.Session(profile_name='default', region_name=sys.argv[1])
codepipeline = session.client('codepipeline')

def is_pollable_action(action):
    actionTypeId = action['actionTypeId']
    configuration = action['configuration']
    return actionTypeId['owner'] in {"AWS", "ThirdParty"}
 and actionTypeId['provider'] in {"GitHub", "CodeCommit",
 "S3"} and ('PollForSourceChanges' not in configuration or
 configuration['PollForSourceChanges'] == 'true')

def has_pollable_actions(pipeline):
    hasPollableAction = False
    pipelineDefinition = codepipeline.get_pipeline(name=pipeline['name'])
['pipeline']
    for action in pipelineDefinition['stages'][0]['actions']:
        hasPollableAction = is_pollable_action(action)
        if hasPollableAction:
            break
    return hasPollableAction

def get_last_executed_time(pipelineName):

 pipelineExecutions=codepipeline.list_pipeline_executions(pipelineName=pipelineName)
['pipelineExecutionSummaries']
    if pipelineExecutions:
        return pipelineExecutions[0]['startTime'].strftime("%A %m/%d/%Y, %H:%M:
%S")
    else:
        return "Not executed in last year"
```

```
while hasNextToken:
    if nextToken=="":
        list_pipelines_response = codepipeline.list_pipelines()
    else:
        list_pipelines_response =
 codepipeline.list_pipelines(nextToken=nextToken)
    if 'nextToken' in list_pipelines_response:
        nextToken = list_pipelines_response['nextToken']
    else:
        hasNextToken= False
    for pipeline in list_pipelines_response['pipelines']:
        if has_pollable_actions(pipeline):
            pollablePipelines.append(pipeline['name'])
            lastExecutedTimes.append(get_last_executed_time(pipeline['name']))

fileName="{region}-
{timeNow}.csv".format(region=sys.argv[1],timeNow=math.trunc(time.time()))
file = open(fileName, 'w')

print ("{:<30} {:<30} {:<30}".format('Polling Pipeline Name', '|','Last Executed
 Time'))
print ("{:<30} {:<30} {:<30}".format('_____________________',
 '|','___________________'))
for i in range(len(pollablePipelines)):
    print("{:<30} {:<30} {:<30}".format(pollablePipelines[i], '|',
 lastExecutedTimes[i]))
    file.write("{pipeline},".format(pipeline=pollablePipelines[i]))
file.close()
print("\nSaving Polling Pipeline Names to file
 {fileName}".format(fileName=fileName))
```

3. For each Region where you have pipelines, you must run the script for that Region. To run the script, do one of the following:

   - Run the following command to run the script named **PollingPipelinesExtractor.sh**. In this example, the Region is us-west-2.

     ```
     ./PollingPipelinesExtractor.sh us-west-2
     ```

   - For the python script, run the following command to run the python script named**PollingPipelinesExtractor.py**. In this example, the Region is us-west-2.

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

In the following sample output from the script, the Region us-west-2 returned a list of polling pipelines and shows the last execution time for each pipeline.

```
 % ./pollingPipelineExtractor.sh us-west-2

| Polling Pipeline Name | Last Executed Time           |
| _____________________ | ____________________         |
| myCodeBuildPipeline   | Wed Mar 8 09:35:49 PST 2023  |
| myCodeCommitPipeline  | Mon Apr 24 22:32:32 PDT 2023 |
| TestPipeline          | Not executed in last year    |

Saving list of polling pipeline names to us-west-2-1682496174.csv...%
```

Analyze the script output and, for each pipeline in the list, update the polling source to the recommended event-based change detection method.

> **Note**
>
> Your polling pipelines are determined by the pipeline's action configuration for the `PollForSourceChanges` parameter. If the pipeline source configuration has the `PollForSourceChanges` parameter omitted, then CodePipeline defaults to polling your repository for source changes. This behavior is the same as if `PollForSourceChanges` is included and set to true. For more information, see the configuration parameters for your pipeline's source action, such as the Amazon S3 source action configuration parameters in Amazon S3 source action reference.

Note that this script also generates a .csv file containing the list of polling pipelines in your account and saves the .csv file to the current working folder.

Migrate polling pipelines with a CodeCommit source

You can migrate your polling pipeline to use EventBridge to detect changes in your CodeCommit source repository or your Amazon S3 source bucket.

CodeCommit -- For a pipeline with a CodeCommit source, modify the pipeline so that change detection is automated through EventBridge. Choose from the following methods to implement the migration:

- **Console:** Migrate polling pipelines (CodeCommit or Amazon S3 source) (console)
- **CLI:** Migrate polling pipelines (CodeCommit source) (CLI)
- **CloudFormation:** Migrate polling pipelines (CodeCommit source) (CloudFormation template)

Migrate polling pipelines (CodeCommit or Amazon S3 source) (console)

You can use the CodePipeline console to update your pipeline to use EventBridge 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 EventBridge rule and IAM role yourself. For more information, see CodeCommit source actions and EventBridge.

Use these steps to edit a pipeline that is using periodic checks. If you want to create a pipeline, see Create a pipeline, stages, and actions.

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 CloudWatch Events to automatically start my pipeline when a change occurs (recommended)**.

   A message appears showing the EventBridge 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 EventBridge 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.

Migrate polling pipelines (CodeCommit source) (CLI)

Follow these steps to edit a pipeline that is using polling (periodic checks) to use an EventBridge rule to start the pipeline. If you want to create a pipeline, see Create a pipeline, stages, and actions.

To build an event-driven pipeline with CodeCommit, you edit the `PollForSourceChanges` parameter of your pipeline and then create the following resources:

- EventBridge 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 Valid settings for the `PollForSourceChanges` parameter.

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:

```
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.

```
"configuration": {
    "PollForSourceChanges": "false",
    "BranchName": "main",
    "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:

```
"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 an EventBridge rule with CodeCommit as the event source and CodePipeline as the target

1. Add permissions for EventBridge to use CodePipeline to invoke the rule. For more information, see Using resource-based policies for Amazon EventBridge.

   a. Use the following sample to create the trust policy that allows EventBridge to assume the service role. Name the trust policy `trustpolicyforEB.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.

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. Create the permissions policy JSON, as shown in this sample, for the pipeline named `MyFirstPipeline`. Name the permissions policy `permissionspolicyforEB.json`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws-eusc:codepipeline:us-
west-2:111122223333:MyFirstPipeline"
            ]
        }
    ]
}
```

d. Use the following command to attach the `CodePipeline-Permissions-Policy-for-EB` permissions policy to the `Role-for-MyRule` role.

Why am I making this change? Adding this policy to the role creates permissions for EventBridge.

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

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 CloudFormation to create the event.

The following sample command creates a rule called `MyCodeCommitRepoRule`.

```
aws events put-rule --name "MyCodeCommitRepoRule" --event-pattern "{\"source\":
[\"aws.codecommit\"],\"detail-type\":[\"CodeCommit Repository State Change\"],
\"resources\":[\"repository-ARN\"],\"detail\":{\"referenceType\":[\"branch\"],
\"referenceName\":[\"main\"]}}" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-
MyRule"
```

3. 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.

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

4. (Optional) To configure an input transformer with source overrides for a specific image ID, use the following JSON in your CLI command. The following example configures an override where:

   - The `actionName`, `Source` in this example, is the dynamic value, defined at pipeline creation, not derived from the source event.
   - The `revisionType`, `COMMIT_ID` in this example, is the dynamic value, defined at pipeline creation, not derived from the source event.
   - The `revisionValue`, <*revisionValue*> in this example, is derived from the source event variable.

```
{
    "Rule": "my-rule",
    "Targets": [
        {
            "Id": "MyTargetId",
```

```
            "Arn": "pipeline-ARN",
            "InputTransformer": {
                "sourceRevisions": {
                    "actionName": "Source",
                    "revisionType": "COMMIT_ID",
                    "revisionValue": "<revisionValue>"
                },
                "variables": [
                    {
                        "name": "Branch_Name",
                        "value": "value"
                    }
                ]
            }
        }
    ]
}
```

Migrate polling pipelines (CodeCommit source) (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 EventBridge rule
- An IAM role for your EventBridge rule

If you use 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

```
Resources:
```

```
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: codecommit-polling-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: CodeCommit
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                BranchName: !Ref BranchName
                RepositoryName: !Ref RepositoryName
                PollForSourceChanges: true
              RunOrder: 1
```

JSON

```
"Stages": [
    {
        "Name": "Source",
  "Actions": [{
      "Name": "SourceAction",
      "ActionTypeId": {
    "Category": "Source",
    "Owner": "AWS",
    "Version": 1,
    "Provider": "CodeCommit"
      },
      "OutputArtifacts": [{
          "Name": "SourceOutput"
      }],
      "Configuration": {
          "BranchName": {
```

```
        "Ref": "BranchName"
    },
    "RepositoryName": {
        "Ref": "RepositoryName"
    },
    "PollForSourceChanges": true
            },
            "RunOrder": 1
        }]
    },
```

To update your pipeline CloudFormation template and create EventBridge rule

1. In the template, under `Resources`, use the `AWS::IAM::Role` 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 CloudFormation to create permissions for EventBridge. This resource is added to your CloudFormation stack.

   YAML

   ```
   EventRole:
     Type: AWS::IAM::Role
     Properties:
       AssumeRolePolicyDocument:
         Version: 2012-10-17
         Statement:
           -
             Effect: Allow
             Principal:
               Service:
                 - events.amazonaws.com
             Action: sts:AssumeRole
       Path: /
       Policies:
   ```

```
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
```

JSON

```
"EventRole": {
  "Type": "AWS::IAM::Role",
  "Properties": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": [
              "events.amazonaws.com"
            ]
          },
          "Action": "sts:AssumeRole"
        }
      ]
    },
    "Path": "/",
    "Policies": [
      {
        "PolicyName": "eb-pipeline-execution",
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Action": "codepipeline:StartPipelineExecution",
              "Resource": {
                "Fn::Join": [
                  "",
```

```
                  [
                    "arn:aws:codepipeline:",
                    {
                      "Ref": "AWS::Region"
                    },
                    ":",
                    {
                      "Ref": "AWS::AccountId"
                    },
                    ":",
                    {
                      "Ref": "AppPipeline"
                    }
                  ]

...
```

2. In the template, under `Resources`, use the `AWS::Events::Rule` CloudFormation resource to add an EventBridge rule. This event pattern creates an event that monitors push changes to your repository. When EventBridge 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 CloudFormation to create the event. This resource is added to your CloudFormation stack.

   YAML

```
  EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventPattern:
        source:
          - aws.codecommit
        detail-type:
          - 'CodeCommit Repository State Change'
        resources:
          - !Join [ '', [ 'arn:aws:codecommit:', !Ref 'AWS::Region', ':', !Ref
'AWS::AccountId', ':', !Ref RepositoryName ] ]
        detail:
          event:
            - referenceCreated
            - referenceUpdated
          referenceType:
```

```
            - branch
          referenceName:
            - main
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline
```

JSON

```
"EventRule": {
  "Type": "AWS::Events::Rule",
  "Properties": {
    "EventPattern": {
      "source": [
        "aws.codecommit"
      ],
      "detail-type": [
        "CodeCommit Repository State Change"
      ],
      "resources": [
        {
          "Fn::Join": [
            "",
            [
              "arn:aws:codecommit:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
                "Ref": "RepositoryName"
              }
            ]
          ]
```

```
        }
      ],
      "detail": {
        "event": [
          "referenceCreated",
          "referenceUpdated"
        ],
        "referenceType": [
          "branch"
        ],
        "referenceName": [
          "main"
        ]
      }
    },
    "Targets": [
      {
        "Arn": {
          "Fn::Join": [
            "",
            [
              "arn:aws:codepipeline:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
                "Ref": "AppPipeline"
              }
            ]
          ]
        },
        "RoleArn": {
          "Fn::GetAtt": [
            "EventRole",
            "Arn"
          ]
        },
        "Id": "codepipeline-AppPipeline"
      }
```

```
    ]
  }
},
```

3. (Optional) To configure an input transformer with source overrides for a specific image ID, use the following YAML snippet. The following example configures an override where:

   - The `actionName`, `Source` in this example, is the dynamic value, defined at pipeline creation, not derived from the source event.
   - The `revisionType`, `COMMIT_ID` in this example, is the dynamic value, defined at pipeline creation, not derived from the source event.
   - The `revisionValue`, <*`revisionValue`*> in this example, is derived from the source event variable.
   - The output variables for `BranchName` and `Value` are specified.

```
Rule: my-rule
Targets:
- Id: MyTargetId
  Arn: pipeline-ARN
  InputTransformer:
    sourceRevisions:
      actionName: Source
      revisionType: COMMIT_ID
      revisionValue: <revisionValue>
    variables:
    - name: BranchName
      value: value
```

4. Save the updated template to your local computer, and then open the CloudFormation console.
5. Choose your stack, and then choose **Create Change Set for Current Stack**.
6. Upload the template, and then view the changes listed in CloudFormation. These are the changes to 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**
>
> 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 Valid settings for the `PollForSourceChanges` parameter.

- 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

```
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: CodeCommit
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                BranchName: !Ref BranchName
                RepositoryName: !Ref RepositoryName
                PollForSourceChanges: false
              RunOrder: 1
```

  JSON

```
{
  "Name": "Source",
  "Actions": [
```

```
    {
      "Name": "SourceAction",
      "ActionTypeId": {
        "Category": "Source",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "CodeCommit"
      },
      "OutputArtifacts": [
        {
          "Name": "SourceOutput"
        }
      ],
      "Configuration": {
        "BranchName": {
          "Ref": "BranchName"
        },
        "RepositoryName": {
          "Ref": "RepositoryName"
        },
        "PollForSourceChanges": false
      },
      "RunOrder": 1
    }
  ]
},
```

Example

When you create these resources with CloudFormation, your pipeline is triggered when files in your repository are created or updated. Here is the final template snippet:

YAML

```
Resources:
  EventRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
```

```
          -
            Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region',
':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
 EventRule:
   Type: AWS::Events::Rule
   Properties:
     EventPattern:
       source:
         - aws.codecommit
       detail-type:
         - 'CodeCommit Repository State Change'
       resources:
         - !Join [ '', [ 'arn:aws:codecommit:', !Ref 'AWS::Region', ':', !Ref
'AWS::AccountId', ':', !Ref RepositoryName ] ]
       detail:
         event:
           - referenceCreated
           - referenceUpdated
         referenceType:
           - branch
         referenceName:
           - main
     Targets:
       -
         Arn:
           !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
'AWS::AccountId', ':', !Ref AppPipeline ] ]
         RoleArn: !GetAtt EventRole.Arn
         Id: codepipeline-AppPipeline
```

```
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: codecommit-events-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: CodeCommit
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                BranchName: !Ref BranchName
                RepositoryName: !Ref RepositoryName
                PollForSourceChanges: false
              RunOrder: 1


...
```

JSON

```
    "Resources": {

...

        "EventRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
```

```
                            "Service": [
                                "events.amazonaws.com"
                            ]
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "Path": "/",
            "Policies": [
                {
                    "PolicyName": "eb-pipeline-execution",
                    "PolicyDocument": {
                        "Version": "2012-10-17",
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Action": "codepipeline:StartPipelineExecution",
                                "Resource": {
                                    "Fn::Join": [
                                        "",
                                        [
                                            "arn:aws:codepipeline:",
                                            {
                                                "Ref": "AWS::Region"
                                            },
                                            ":",
                                            {
                                                "Ref": "AWS::AccountId"
                                            },
                                            ":",
                                            {
                                                "Ref": "AppPipeline"
                                            }
                                        ]
                                    ]
                                }
                            }
                        ]
                    }
                }
            ]
        }
    },
```

```
        "EventRule": {
            "Type": "AWS::Events::Rule",
            "Properties": {
                "EventPattern": {
                    "source": [
                        "aws.codecommit"
                    ],
                    "detail-type": [
                        "CodeCommit Repository State Change"
                    ],
                    "resources": [
                        {
                            "Fn::Join": [
                                "",
                                [
                                    "arn:aws:codecommit:",
                                    {
                                        "Ref": "AWS::Region"
                                    },
                                    ":",
                                    {
                                        "Ref": "AWS::AccountId"
                                    },
                                    ":",
                                    {
                                        "Ref": "RepositoryName"
                                    }
                                ]
                            ]
                        }
                    ],
                    "detail": {
                        "event": [
                            "referenceCreated",
                            "referenceUpdated"
                        ],
                        "referenceType": [
                            "branch"
                        ],
                        "referenceName": [
                            "main"
                        ]
                    }
                },
```

```
                "Targets": [
                    {
                        "Arn": {
                            "Fn::Join": [
                                "",
                                [
                                    "arn:aws:codepipeline:",
                                    {
                                        "Ref": "AWS::Region"
                                    },
                                    ":",
                                    {
                                        "Ref": "AWS::AccountId"
                                    },
                                    ":",
                                    {
                                        "Ref": "AppPipeline"
                                    }
                                ]
                            ]
                        },
                        "RoleArn": {
                            "Fn::GetAtt": [
                                "EventRole",
                                "Arn"
                            ]
                        },
                        "Id": "codepipeline-AppPipeline"
                    }
                ]
            }
        },
        "AppPipeline": {
            "Type": "AWS::CodePipeline::Pipeline",
            "Properties": {
                "Name": "codecommit-events-pipeline",
                "RoleArn": {
                    "Fn::GetAtt": [
                        "CodePipelineServiceRole",
                        "Arn"
                    ]
                },
                "Stages": [
                    {
```

```
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "AWS",
                                    "Version": 1,
                                    "Provider": "CodeCommit"
                                },
                                "OutputArtifacts": [
                                    {
                                        "Name": "SourceOutput"
                                    }
                                ],
                                "Configuration": {
                                    "BranchName": {
                                        "Ref": "BranchName"
                                    },
                                    "RepositoryName": {
                                        "Ref": "RepositoryName"
                                    },
                                    "PollForSourceChanges": false
                                },
                                "RunOrder": 1
                            }
                        ]
                    },

...
```

Migrate polling pipelines with an S3 source enabled for events

For a pipeline with an Amazon S3 source, modify the pipeline so that change detection is automated through EventBridge and with a source bucket that is enabled for event notifications. This is the recommended method if you are using the CLI or CloudFormation to migrate your pipeline.

> **Note**
>
> This includes using a bucket that is enabled for event notifications, where you do not need to create a separate CloudTrail trail. If you are using the console, then an event rule and CloudTrail trail are set up for you. For those steps, see Migrate polling pipelines with an S3 source and CloudTrail trail.

- **CLI:** Migrate polling pipelines with an S3 source and CloudTrail trail (CLI)
- **CloudFormation:** Migrate polling pipelines with an S3 source and CloudTrail trail (CloudFormation template)

Migrate polling pipelines with an S3 source enabled for events (CLI)

Follow these steps to edit a pipeline that is using polling (periodic checks) to use an event in EventBridge instead. If you want to create a pipeline, see Create a pipeline, stages, and actions.

To build an event-driven pipeline with Amazon S3, you edit the `PollForSourceChanges` parameter of your pipeline and then create the following resources:

- EventBridge event rule
- IAM role to allow the EventBridge event to start your pipeline

To create an EventBridge rule with Amazon S3 as the event source and CodePipeline as the target and apply the permissions policy

1. Grant permissions for EventBridge to use CodePipeline to invoke the rule. For more information, see Using resource-based policies for Amazon EventBridge.

   a. Use the following sample to create the trust policy to allow EventBridge to assume the service role. Name it `trustpolicyforEB.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 EventBridge.

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. Create the permissions policy JSON, as shown here for the pipeline named `MyFirstPipeline`. Name the permissions policy `permissionspolicyforEB.json`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws-eusc:codepipeline:us-
west-2:111122223333:MyFirstPipeline"
            ]
        }
    ]
}
```

d. Use the following command to attach the new `CodePipeline-Permissions-Policy-for-EB` 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-EB --policy-document file://permissionspolicyforEB.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 `EnabledS3SourceRule`.

```
aws events put-rule --name "EnabledS3SourceRule" --event-pattern "{\"source\":
[\"aws.s3\"],\"detail-type\":[\"Object Created\"],\"detail\":{\"bucket\":{\"name\":
[\"amzn-s3-demo-source-bucket\"]}}}" --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 `EnabledS3SourceRule`, 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 EnabledS3SourceRule --targets Id=codepipeline-
AppPipeline,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 Valid settings for the `PollForSourceChanges` parameter.

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:

```
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 `amzn-s3-demo-source-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.

```
"configuration": {
    "S3Bucket": "amzn-s3-demo-source-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:

```
"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.

Migrate polling pipelines with an S3 source enabled for events (CloudFormation template)

This procedure is for a pipeline where the source bucket has events enabled.

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:

- EventBridge rule and IAM role to allow this event to start your pipeline.

If you use 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": "SourceObjectKey"
                                },
                                "PollForSourceChanges": true
                            },
                            "RunOrder": 1
                        }
                    ]
                },


...
```

To create an EventBridge 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` 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 CloudFormation to create permissions for EventBridge. This resource is added to your CloudFormation stack.

YAML

```
  EventRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]


...
```

JSON

```
  "EventRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": [
```

```
                "events.amazonaws.com"
              ]
            },
            "Action": "sts:AssumeRole"
          }
        ]
      },
      "Path": "/",
      "Policies": [
        {
          "PolicyName": "eb-pipeline-execution",
          "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
              {
                "Effect": "Allow",
                "Action": "codepipeline:StartPipelineExecution",
                "Resource": {
                  "Fn::Join": [
                    "",
                    [
                      "arn:aws:codepipeline:",
                      {
                        "Ref": "AWS::Region"
                      },
                      ":",
                      {
                        "Ref": "AWS::AccountId"
                      },
                      ":",
                      {
                        "Ref": "AppPipeline"
                      }
                    ]
                  ]

...
```

2. Use the `AWS::Events::Rule` CloudFormation resource to add an EventBridge rule. This event pattern creates an event that monitors creation or deletion of objects in your Amazon S3 source bucket. In addition, include a target of your pipeline. When an object is created, this rule invokes `StartPipelineExecution` on your target pipeline.

Why am I making this change? Adding the `AWS::Events::Rule` resource enables CloudFormation to create the event. This resource is added to your CloudFormation stack.

YAML

```
  EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventBusName: default
      EventPattern:
        source:
          - aws.s3
        detail-type:
          - Object Created
        detail:
          bucket:
            name:
              - !Ref SourceBucket
      Name: EnabledS3SourceRule
      State: ENABLED
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline


...
```

JSON

```
  "EventRule": {
    "Type": "AWS::Events::Rule",
    "Properties": {
  "EventBusName": "default",
  "EventPattern": {
      "source": [
   "aws.s3"
      ],
```

```
      "detail-type": [
    "Object Created"
      ],
      "detail": {
    "bucket": {
        "name": [
      "s3-pipeline-source-fra-bucket"
        ]
        }
            }
  },
  "Name": "EnabledS3SourceRule",
        "State": "ENABLED",
        "Targets": [
        {
          "Arn": {
            "Fn::Join": [
              "",
              [
                "arn:aws:codepipeline:",
                {
                  "Ref": "AWS::Region"
                },
                ":",
                {
                  "Ref": "AWS::AccountId"
                },
                ":",
                {
                  "Ref": "AppPipeline"
                }
              ]
            ]
          },
          "RoleArn": {
            "Fn::GetAtt": [
              "EventRole",
              "Arn"
            ]
          },
          "Id": "codepipeline-AppPipeline"
        }
      ]
    }
```

```
  }
},

...
```

3. Save your updated template to your local computer, and open the CloudFormation console.
4. Choose your stack, and then choose **Create Change Set for Current Stack**.
5. Upload your updated template, and then view the changes listed in CloudFormation. These are the changes that will 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 Valid settings for the `PollForSourceChanges` parameter.

- 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

```
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
```

```
            OutputArtifacts:
              - Name: SourceOutput
            Configuration:
              S3Bucket: !Ref SourceBucket
              S3ObjectKey: !Ref SourceObjectKey
              PollForSourceChanges: false
            RunOrder: 1
```

JSON

```
{
   "Name": "SourceAction",
   "ActionTypeId": {
     "Category": "Source",
     "Owner": "AWS",
     "Version": 1,
     "Provider": "S3"
   },
   "OutputArtifacts": [
     {
       "Name": "SourceOutput"
     }
   ],
   "Configuration": {
     "S3Bucket": {
       "Ref": "SourceBucket"
     },
     "S3ObjectKey": {
       "Ref": "SourceObjectKey"
     },
     "PollForSourceChanges": false
   },
   "RunOrder": 1
 }
```

Example

When you use 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 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

```
Parameters:
  SourceObjectKey:
    Description: 'S3 source artifact'
    Type: String
    Default: SampleApp_Linux.zip
  ApplicationName:
    Description: 'CodeDeploy application name'
    Type: String
    Default: DemoApplication
  BetaFleet:
    Description: 'Fleet configured in CodeDeploy'
    Type: String
    Default: DemoFleet

Resources:
  SourceBucket:
    Type: AWS::S3::Bucket
    Properties:
      NotificationConfiguration:
        EventBridgeConfiguration:
          EventBridgeEnabled: true
      VersioningConfiguration:
        Status: Enabled
  CodePipelineArtifactStoreBucket:
    Type: AWS::S3::Bucket
  CodePipelineArtifactStoreBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref CodePipelineArtifactStoreBucket
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          -
```

```
            Sid: DenyUnEncryptedObjectUploads
            Effect: Deny
            Principal: '*'
            Action: s3:PutObject
            Resource: !Join [ '', [ !GetAtt CodePipelineArtifactStoreBucket.Arn, '/
*' ] ]
            Condition:
              StringNotEquals:
                s3:x-amz-server-side-encryption: aws:kms
          -
            Sid: DenyInsecureConnections
            Effect: Deny
            Principal: '*'
            Action: s3:*
            Resource: !Sub ${CodePipelineArtifactStoreBucket.Arn}/*
            Condition:
              Bool:
                aws:SecureTransport: false
  CodePipelineServiceRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - codepipeline.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: AWS-CodePipeline-Service-3
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action:
                  - codecommit:CancelUploadArchive
                  - codecommit:GetBranch
                  - codecommit:GetCommit
                  - codecommit:GetUploadArchiveStatus
```

```
            - codecommit:UploadArchive
          Resource: 'resource_ARN'
        -
          Effect: Allow
          Action:
            - codedeploy:CreateDeployment
            - codedeploy:GetApplicationRevision
            - codedeploy:GetDeployment
            - codedeploy:GetDeploymentConfig
            - codedeploy:RegisterApplicationRevision
          Resource: 'resource_ARN'
        -
          Effect: Allow
          Action:
            - codebuild:BatchGetBuilds
            - codebuild:StartBuild
          Resource: 'resource_ARN'
        -
          Effect: Allow
          Action:
            - devicefarm:ListProjects
            - devicefarm:ListDevicePools
            - devicefarm:GetRun
            - devicefarm:GetUpload
            - devicefarm:CreateUpload
            - devicefarm:ScheduleRun
          Resource: 'resource_ARN'
        -
          Effect: Allow
          Action:
            - lambda:InvokeFunction
            - lambda:ListFunctions
          Resource: 'resource_ARN'
        -
          Effect: Allow
          Action:
            - iam:PassRole
          Resource: 'resource_ARN'
        -
          Effect: Allow
          Action:
            - elasticbeanstalk:*
            - ec2:*
            - elasticloadbalancing:*
```

```
                - autoscaling:*
                - cloudwatch:*
                - s3:*
                - sns:*
                - cloudformation:*
                - rds:*
                - sqs:*
                - ecs:*
              Resource: 'resource_ARN'
AppPipeline:
  Type: AWS::CodePipeline::Pipeline
  Properties:
    Name: s3-events-pipeline
    RoleArn:
      !GetAtt CodePipelineServiceRole.Arn
    Stages:
      -
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
              - Name: SourceOutput
            Configuration:
              S3Bucket: !Ref SourceBucket
              S3ObjectKey: !Ref SourceObjectKey
              PollForSourceChanges: false
            RunOrder: 1
      -
        Name: Beta
        Actions:
          -
            Name: BetaAction
            InputArtifacts:
              - Name: SourceOutput
            ActionTypeId:
              Category: Deploy
              Owner: AWS
              Version: 1
```

```
              Provider: CodeDeploy
            Configuration:
              ApplicationName: !Ref ApplicationName
              DeploymentGroupName: !Ref BetaFleet
            RunOrder: 1
    ArtifactStore:
      Type: S3
      Location: !Ref CodePipelineArtifactStoreBucket
 EventRole:
   Type: AWS::IAM::Role
   Properties:
     AssumeRolePolicyDocument:
       Version: 2012-10-17
       Statement:
         -
           Effect: Allow
           Principal:
             Service:
               - events.amazonaws.com
           Action: sts:AssumeRole
     Path: /
     Policies:
       -
         PolicyName: eb-pipeline-execution
         PolicyDocument:
           Version: 2012-10-17
           Statement:
             -
               Effect: Allow
               Action: codepipeline:StartPipelineExecution
               Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region',
':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
 EventRule:
   Type: AWS::Events::Rule
   Properties:
     EventBusName: default
     EventPattern:
       source:
         - aws.s3
       detail-type:
         - Object Created
       detail:
         bucket:
           name:
```

```
            - !Ref SourceBucket
      Name: EnabledS3SourceRule
      State: ENABLED
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline
```

JSON

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "appconfig:StartDeployment",
                "appconfig:StopDeployment",
                "appconfig:GetDeployment"
            ],
            "Resource": [
                "arn:aws-eusc:appconfig:*:111122223333:application/
[[Application]]",
                "arn:aws-eusc:appconfig:*:111122223333:application/
[[Application]]/*",
                "arn:aws-eusc:appconfig:*:111122223333:deploymentstrategy/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Migrate polling pipelines with an S3 source and CloudTrail trail

For a pipeline with an Amazon S3 source, modify the pipeline so that change detection is automated through EventBridge. Choose from the following methods to implement the migration:

- **Console:** Migrate polling pipelines (CodeCommit or Amazon S3 source) (console)
- **CLI:** Migrate polling pipelines with an S3 source and CloudTrail trail (CLI)
- **CloudFormation:** Migrate polling pipelines with an S3 source and CloudTrail trail (CloudFormation template)

Migrate polling pipelines with an S3 source and CloudTrail trail (CLI)

Follow these steps to edit a pipeline that is using polling (periodic checks) to use an event in EventBridge instead. If you want to create a pipeline, see Create a pipeline, stages, and actions.

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.
- EventBridge event
- IAM role to allow the EventBridge 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 `amzn-s3-demo-source-bucket`.

```
aws cloudtrail create-trail --name my-trail --s3-bucket-name amzn-s3-demo-source-
bucket
```

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 EventBridge.

   Example:

   The following command uses `--name` to start logging on a trail named `my-trail`.

   ```
   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 EventBridge 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 `amzn-s3-demo-source-bucket/myFolder`.

   ```
   aws cloudtrail put-event-selectors --trail-name my-trail --event-selectors
    '[{ "ReadWriteType": "WriteOnly", "IncludeManagementEvents":false,
    "DataResources": [{ "Type": "AWS::S3::Object", "Values": ["arn:aws:s3:::amzn-s3-
   demo-source-bucket/myFolder/file.zip"] }] }]'
   ```

To create an EventBridge rule with Amazon S3 as the event source and CodePipeline as the target and apply the permissions policy

1. Grant permissions for EventBridge to use CodePipeline to invoke the rule. For more information, see Using resource-based policies for Amazon EventBridge.

   a. Use the following sample to create the trust policy to allow EventBridge to assume the service role. Name it `trustpolicyforEB.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 EventBridge.

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. Create the permissions policy JSON, as shown here for the pipeline named `MyFirstPipeline`. Name the permissions policy `permissionspolicyforEB.json`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws-eusc:codepipeline:us-
west-2:111122223333:MyFirstPipeline"
```

```
            ]
        }
    ]
}
```

d. Use the following command to attach the new `CodePipeline-Permissions-Policy-for-EB` 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-EB --policy-document file://permissionspolicyforEB.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\":[\"amzn-s3-
demo-source-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
```

4. (Optional) To configure an input transformer with source overrides for a specific image ID, use the following JSON in your CLI command. The following example configures an override where:

   - The `actionName`, `Source` in this example, is the dynamic value, defined at pipeline creation, not derived from the source event.

- The `revisionType`, `S3_OBJECT_VERSION_ID` in this example, is the dynamic value, defined at pipeline creation, not derived from the source event.
- The `revisionValue`, <*revisionValue*> in this example, is derived from the source event variable.

```
{
    "Rule": "my-rule",
    "Targets": [
        {
            "Id": "MyTargetId",
            "Arn": "ARN",
            "InputTransformer": {
                "InputPathsMap": {
                    "revisionValue": "$.detail.object.version-id"
                },
                "InputTemplate": {
                    "sourceRevisions": {
                        "actionName": "Source",
                        "revisionType": "S3_OBJECT_VERSION_ID",
                        "revisionValue": "<revisionValue>"
                    }
                }
            }
        }
    ]
}
```

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 Valid settings for the `PollForSourceChanges` parameter.

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:

```
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 `amzn-s3-demo-source-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.

```
"configuration": {
    "S3Bucket": "amzn-s3-demo-source-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:

```
"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.

Migrate polling pipelines with an S3 source and CloudTrail trail (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:

- EventBridge 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 data events for trails and Logging management events for trails.
- EventBridge rule and IAM role to allow this event to start your pipeline.

If you use 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": "SourceObjectKey"
                                    },
                                    "PollForSourceChanges": true
                                },
                                "RunOrder": 1
                            }
                        ]
                    },


...
```

To create an EventBridge 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` 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 CloudFormation to create permissions for EventBridge. This resource is added to your CloudFormation stack.

YAML

```
  EventRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]


...
```

JSON

```
  "EventRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
```

```
      "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": [
                "events.amazonaws.com"
              ]
            },
            "Action": "sts:AssumeRole"
          }
        ]
      },
      "Path": "/",
      "Policies": [
        {
          "PolicyName": "eb-pipeline-execution",
          "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
              {
                "Effect": "Allow",
                "Action": "codepipeline:StartPipelineExecution",
                "Resource": {
                  "Fn::Join": [
                    "",
                    [
                      "arn:aws:codepipeline:",
                      {
                        "Ref": "AWS::Region"
                      },
                      ":",
                      {
                        "Ref": "AWS::AccountId"
                      },
                      ":",
                      {
                        "Ref": "AppPipeline"
                      }
                    ]
                  ]
```

```
...
```

2. Use the `AWS::Events::Rule` CloudFormation resource to add an EventBridge 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 CloudFormation to create the event. This resource is added to your CloudFormation stack.

   YAML

```
EventRule:
  Type: AWS::Events::Rule
  Properties:
    EventPattern:
      source:
        - aws.s3
      detail-type:
        - 'AWS API Call via CloudTrail'
      detail:
        eventSource:
          - s3.amazonaws.com
        eventName:
          - CopyObject
          - PutObject
          - CompleteMultipartUpload
        requestParameters:
          bucketName:
            - !Ref SourceBucket
          key:
            - !Ref SourceObjectKey
    Targets:
      -
        Arn:
          !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
'AWS::AccountId', ':', !Ref AppPipeline ] ]
        RoleArn: !GetAtt EventRole.Arn
        Id: codepipeline-AppPipeline
```

```
...
```

JSON

```
"EventRule": {
  "Type": "AWS::Events::Rule",
  "Properties": {
    "EventPattern": {
      "source": [
        "aws.s3"
      ],
      "detail-type": [
        "AWS API Call via CloudTrail"
      ],
      "detail": {
        "eventSource": [
          "s3.amazonaws.com"
        ],
        "eventName": [
          "CopyObject",
          "PutObject",
          "CompleteMultipartUpload"
        ],
        "requestParameters": {
          "bucketName": [
            {
              "Ref": "SourceBucket"
            }
          ],
          "key": [
            {
              "Ref": "SourceObjectKey"
            }
          ]
        }
      }
    },
    "Targets": [
      {
        "Arn": {
          "Fn::Join": [
            "",
```

```
              [
                "arn:aws:codepipeline:",
                {
                  "Ref": "AWS::Region"
                },
                ":",
                {
                  "Ref": "AWS::AccountId"
                },
                ":",
                {
                  "Ref": "AppPipeline"
                }
              ]
            ]
          },
          "RoleArn": {
            "Fn::GetAtt": [
              "EventRole",
              "Arn"
            ]
          },
          "Id": "codepipeline-AppPipeline"
        }
      ]
    }
  }
},

...
```

3. Add this snippet to your first template to allow cross-stack functionality:

   YAML

```
Outputs:
  SourceBucketARN:
    Description: "S3 bucket ARN that Cloudtrail will use"
    Value: !GetAtt SourceBucket.Arn
    Export:
      Name: SourceBucketARN
```

JSON

```
  "Outputs" : {
    "SourceBucketARN" : {
      "Description" : "S3 bucket ARN that Cloudtrail will use",
      "Value" : { "Fn::GetAtt": ["SourceBucket", "Arn"] },
      "Export" : {
        "Name" : "SourceBucketARN"
      }
    }

...
```

4. (Optional) To configure an input transformer with source overrides for a specific image ID, use the following YAML snippet. The following example configures an override where:

   - The `actionName`, `Source` in this example, is the dynamic value, defined at pipeline creation, not derived from the source event.
   - The `revisionType`, `S3_OBJECT_VERSION_ID` in this example, is the dynamic value, defined at pipeline creation, not derived from the source event.
   - The `revisionValue`, <*revisionValue*> in this example, is derived from the source event variable.

```
---
Rule: my-rule
Targets:
- Id: MyTargetId
  Arn: pipeline-ARN
  InputTransformer:
    InputPathsMap:
      revisionValue: "$.detail.object.version-id"
    InputTemplate:
      sourceRevisions:
        actionName: Source
        revisionType: S3_OBJECT_VERSION_ID
        revisionValue: '<revisionValue>'
```

5. Save your updated template to your local computer, and open the CloudFormation console.
6. Choose your stack, and then choose **Create Change Set for Current Stack**.

7. Upload your updated template, and then view the changes listed in CloudFormation. These are the changes that will be made to the stack. You should see your new resources in the list.
8. 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 Valid settings for the `PollForSourceChanges` parameter.

- 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

```
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
              - Name: SourceOutput
            Configuration:
              S3Bucket: !Ref SourceBucket
              S3ObjectKey: !Ref SourceObjectKey
              PollForSourceChanges: false
            RunOrder: 1
```

JSON

```
{
   "Name": "SourceAction",
   "ActionTypeId": {
     "Category": "Source",
     "Owner": "AWS",
     "Version": 1,
     "Provider": "S3"
   },
   "OutputArtifacts": [
     {
       "Name": "SourceOutput"
     }
   ],
   "Configuration": {
     "S3Bucket": {
       "Ref": "SourceBucket"
     },
     "S3ObjectKey": {
       "Ref": "SourceObjectKey"
     },
     "PollForSourceChanges": false
   },
   "RunOrder": 1
 }
```

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` 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
            Resource: !Join [ '', [ !GetAtt AWSCloudTrailBucket.Arn, '/
AWSLogs/', !Ref 'AWS::AccountId', '/*' ] ]
            Condition:
              StringEquals:
                s3:x-amz-acl: bucket-owner-full-control
  AWSCloudTrailBucket:
    Type: AWS::S3::Bucket
```

```
    DeletionPolicy: Retain
  AwsCloudTrail:
    DependsOn:
      - AWSCloudTrailBucketPolicy
    Type: AWS::CloudTrail::Trail
    Properties:
      S3BucketName: !Ref AWSCloudTrailBucket
      EventSelectors:
        -
          DataResources:
            -
              Type: AWS::S3::Object
              Values:
                - !Join [ '', [ !ImportValue SourceBucketARN, '/', !Ref
 SourceObjectKey ] ]
          ReadWriteType: WriteOnly
          IncludeManagementEvents: false
      IncludeGlobalServiceEvents: true
      IsLogging: true
      IsMultiRegionTrail: true


...
```

JSON

```
{
  "Parameters": {
    "SourceObjectKey": {
      "Description": "S3 source artifact",
      "Type": "String",
      "Default": "SampleApp_Linux.zip"
    }
  },
  "Resources": {
    "AWSCloudTrailBucket": {
      "Type": "AWS::S3::Bucket",
        "DeletionPolicy": "Retain"
    },
    "AWSCloudTrailBucketPolicy": {
      "Type": "AWS::S3::BucketPolicy",
      "Properties": {
        "Bucket": {
```

```
            "Ref": "AWSCloudTrailBucket"
          },
          "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
              {
                "Sid": "AWSCloudTrailAclCheck",
                "Effect": "Allow",
                "Principal": {
                  "Service": [
                    "cloudtrail.amazonaws.com"
                  ]
                },
                "Action": "s3:GetBucketAcl",
                "Resource": {
                  "Fn::GetAtt": [
                    "AWSCloudTrailBucket",
                    "Arn"
                  ]
                }
              },
              {
                "Sid": "AWSCloudTrailWrite",
                "Effect": "Allow",
                "Principal": {
                  "Service": [
                    "cloudtrail.amazonaws.com"
                  ]
                },
                "Action": "s3:PutObject",
                "Resource": {
                  "Fn::Join": [
                    "",
                    [
                      {
                        "Fn::GetAtt": [
                          "AWSCloudTrailBucket",
                          "Arn"
                        ]
                      },
                      "/AWSLogs/",
                      {
                        "Ref": "AWS::AccountId"
                      },
```

```
                "/*"
              ]
            ]
          },
          "Condition": {
            "StringEquals": {
              "s3:x-amz-acl": "bucket-owner-full-control"
            }
          }
        }
      ]
    }
  }
},
"AwsCloudTrail": {
  "DependsOn": [
    "AWSCloudTrailBucketPolicy"
  ],
  "Type": "AWS::CloudTrail::Trail",
  "Properties": {
    "S3BucketName": {
      "Ref": "AWSCloudTrailBucket"
    },
    "EventSelectors": [
      {
        "DataResources": [
          {
            "Type": "AWS::S3::Object",
            "Values": [
              {
                "Fn::Join": [
                  "",
                  [
                    {
                      "Fn::ImportValue": "SourceBucketARN"
                    },
                    "/",
                    {
                      "Ref": "SourceObjectKey"
                    }
                  ]
                ]
              }
            ]
```

```
              }
            ],
            "ReadWriteType": "WriteOnly",
            "IncludeManagementEvents": false
          }
        ],
        "IncludeGlobalServiceEvents": true,
        "IsLogging": true,
        "IsMultiRegionTrail": true
      }
    }
  }
}

...
```

Example

When you use 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 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
            Resource: !Join [ '', [ !GetAtt CodePipelineArtifactStoreBucket.Arn, '/
*' ] ]
            Condition:
              StringNotEquals:
                s3:x-amz-server-side-encryption: aws:kms
          -
            Sid: DenyInsecureConnections
            Effect: Deny
            Principal: '*'
            Action: s3:*
            Resource: !Join [ '', [ !GetAtt CodePipelineArtifactStoreBucket.Arn, '/
*' ] ]
            Condition:
              Bool:
                aws:SecureTransport: false
  CodePipelineServiceRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - codepipeline.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: AWS-CodePipeline-Service-3
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
```

```
      Effect: Allow
      Action:
        - codecommit:CancelUploadArchive
        - codecommit:GetBranch
        - codecommit:GetCommit
        - codecommit:GetUploadArchiveStatus
        - codecommit:UploadArchive
      Resource: 'resource_ARN'
    -
      Effect: Allow
      Action:
        - codedeploy:CreateDeployment
        - codedeploy:GetApplicationRevision
        - codedeploy:GetDeployment
        - codedeploy:GetDeploymentConfig
        - codedeploy:RegisterApplicationRevision
      Resource: 'resource_ARN'
    -
      Effect: Allow
      Action:
        - codebuild:BatchGetBuilds
        - codebuild:StartBuild
      Resource: 'resource_ARN'
    -
      Effect: Allow
      Action:
        - devicefarm:ListProjects
        - devicefarm:ListDevicePools
        - devicefarm:GetRun
        - devicefarm:GetUpload
        - devicefarm:CreateUpload
        - devicefarm:ScheduleRun
      Resource: 'resource_ARN'
    -
      Effect: Allow
      Action:
        - lambda:InvokeFunction
        - lambda:ListFunctions
      Resource: 'resource_ARN'
    -
      Effect: Allow
      Action:
        - iam:PassRole
      Resource: 'resource_ARN'
```

```
            -
              Effect: Allow
              Action:
                - elasticbeanstalk:*
                - ec2:*
                - elasticloadbalancing:*
                - autoscaling:*
                - cloudwatch:*
                - s3:*
                - sns:*
                - cloudformation:*
                - rds:*
                - sqs:*
                - ecs:*
              Resource: 'resource_ARN'
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: s3-events-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: S3
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                S3Bucket: !Ref SourceBucket
                S3ObjectKey: !Ref SourceObjectKey
                PollForSourceChanges: false
              RunOrder: 1
        -
          Name: Beta
          Actions:
            -
              Name: BetaAction
```

```
            InputArtifacts:
              - Name: SourceOutput
            ActionTypeId:
              Category: Deploy
              Owner: AWS
              Version: 1
              Provider: CodeDeploy
            Configuration:
              ApplicationName: !Ref ApplicationName
              DeploymentGroupName: !Ref BetaFleet
            RunOrder: 1
    ArtifactStore:
      Type: S3
      Location: !Ref CodePipelineArtifactStoreBucket
 EventRole:
   Type: AWS::IAM::Role
   Properties:
     AssumeRolePolicyDocument:
       Version: 2012-10-17
       Statement:
         -
           Effect: Allow
           Principal:
             Service:
               - events.amazonaws.com
           Action: sts:AssumeRole
     Path: /
     Policies:
       -
         PolicyName: eb-pipeline-execution
         PolicyDocument:
           Version: 2012-10-17
           Statement:
             -
               Effect: Allow
               Action: codepipeline:StartPipelineExecution
               Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region',
':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
 EventRule:
   Type: AWS::Events::Rule
   Properties:
     EventPattern:
       source:
         - aws.s3
```

```
        detail-type:
          - 'AWS API Call via CloudTrail'
        detail:
          eventSource:
            - s3.amazonaws.com
          eventName:
            - PutObject
            - CompleteMultipartUpload
          resources:
            ARN:
              - !Join [ '', [ !GetAtt SourceBucket.Arn, '/', !Ref
 SourceObjectKey ] ]
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline

Outputs:
  SourceBucketARN:
    Description: "S3 bucket ARN that Cloudtrail will use"
    Value: !GetAtt SourceBucket.Arn
    Export:
      Name: SourceBucketARN
```

JSON

```
    "Resources": {
        "SourceBucket": {
            "Type": "AWS::S3::Bucket",
            "Properties": {
                "VersioningConfiguration": {
                    "Status": "Enabled"
                }
            }
        },
        "CodePipelineArtifactStoreBucket": {
            "Type": "AWS::S3::Bucket"
        },
        "CodePipelineArtifactStoreBucketPolicy": {
            "Type": "AWS::S3::BucketPolicy",
```

```
            "Properties": {
                "Bucket": {
                    "Ref": "CodePipelineArtifactStoreBucket"
                },
                "PolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Sid": "DenyUnEncryptedObjectUploads",
                            "Effect": "Deny",
                            "Principal": "*",
                            "Action": "s3:PutObject",
                            "Resource": {
                                "Fn::Join": [
                                    "",
                                    [
                                        {
                                            "Fn::GetAtt": [
                                                "CodePipelineArtifactStoreBucket",
                                                "Arn"
                                            ]
                                        },
                                        "/*"
                                    ]
                                ]
                            },
                            "Condition": {
                                "StringNotEquals": {
                                    "s3:x-amz-server-side-encryption": "aws:kms"
                                }
                            }
                        },
                        {
                            "Sid": "DenyInsecureConnections",
                            "Effect": "Deny",
                            "Principal": "*",
                            "Action": "s3:*",
                            "Resource": {
                                "Fn::Join": [
                                    "",
                                    [
                                        {
                                            "Fn::GetAtt": [
                                                "CodePipelineArtifactStoreBucket",
```

```
                                        "Arn"
                                    ]
                                },
                                "/*"
                            ]
                        ]
                    },
                    "Condition": {
                        "Bool": {
                            "aws:SecureTransport": false
                        }
                    }
                }
            ]
        }
    }
},
"CodePipelineServiceRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": [
                            "codepipeline.amazonaws.com"
                        ]
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        },
        "Path": "/",
        "Policies": [
            {
                "PolicyName": "AWS-CodePipeline-Service-3",
                "PolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Action": [
```

```
                "codecommit:CancelUploadArchive",
                "codecommit:GetBranch",
                "codecommit:GetCommit",
                "codecommit:GetUploadArchiveStatus",
                "codecommit:UploadArchive"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "codedeploy:CreateDeployment",
                "codedeploy:GetApplicationRevision",
                "codedeploy:GetDeployment",
                "codedeploy:GetDeploymentConfig",
                "codedeploy:RegisterApplicationRevision"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "codebuild:BatchGetBuilds",
                "codebuild:StartBuild"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "devicefarm:ListProjects",
                "devicefarm:ListDevicePools",
                "devicefarm:GetRun",
                "devicefarm:GetUpload",
                "devicefarm:CreateUpload",
                "devicefarm:ScheduleRun"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction",
                "lambda:ListFunctions"
```

```
                                ],
                                "Resource": "resource_ARN"
                            },
                            {
                                "Effect": "Allow",
                                "Action": [
                                    "iam:PassRole"
                                ],
                                "Resource": "resource_ARN"
                            },
                            {
                                "Effect": "Allow",
                                "Action": [
                                    "elasticbeanstalk:*",
                                    "ec2:*",
                                    "elasticloadbalancing:*",
                                    "autoscaling:*",
                                    "cloudwatch:*",
                                    "s3:*",
                                    "sns:*",
                                    "cloudformation:*",
                                    "rds:*",
                                    "sqs:*",
                                    "ecs:*"
                                ],
                                "Resource": "resource_ARN"
                            }
                        ]
                    }
                }
            ]
        }
    },
    "AppPipeline": {
        "Type": "AWS::CodePipeline::Pipeline",
        "Properties": {
            "Name": "s3-events-pipeline",
            "RoleArn": {
                "Fn::GetAtt": [
                    "CodePipelineServiceRole",
                    "Arn"
                ]
            },
            "Stages": [
```

```
                    {
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "AWS",
                                    "Version": 1,
                                    "Provider": "S3"
                                },
                                "OutputArtifacts": [
                                    {
                                        "Name": "SourceOutput"
                                    }
                                ],
                                "Configuration": {
                                    "S3Bucket": {
                                        "Ref": "SourceBucket"
                                    },
                                    "S3ObjectKey": {
                                        "Ref": "SourceObjectKey"
                                    },
                                    "PollForSourceChanges": false
                                },
                                "RunOrder": 1
                            }
                        ]
                    },
                    {
                        "Name": "Beta",
                        "Actions": [
                            {
                                "Name": "BetaAction",
                                "InputArtifacts": [
                                    {
                                        "Name": "SourceOutput"
                                    }
                                ],
                                "ActionTypeId": {
                                    "Category": "Deploy",
                                    "Owner": "AWS",
                                    "Version": 1,
                                    "Provider": "CodeDeploy"
```

```
                            },
                            "Configuration": {
                                "ApplicationName": {
                                    "Ref": "ApplicationName"
                                },
                                "DeploymentGroupName": {
                                    "Ref": "BetaFleet"
                                }
                            },
                            "RunOrder": 1
                        }
                    ]
                }
            ],
            "ArtifactStore": {
                "Type": "S3",
                "Location": {
                    "Ref": "CodePipelineArtifactStoreBucket"
                }
            }
        }
    },
    "EventRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": [
                                "events.amazonaws.com"
                            ]
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "Path": "/",
            "Policies": [
                {
                    "PolicyName": "eb-pipeline-execution",
                    "PolicyDocument": {
```

```
                        "Version": "2012-10-17",
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Action": "codepipeline:StartPipelineExecution",
                                "Resource": {
                                    "Fn::Join": [
                                        "",
                                        [
                                            "arn:aws:codepipeline:",
                                            {
                                                "Ref": "AWS::Region"
                                            },
                                            ":",
                                            {
                                                "Ref": "AWS::AccountId"
                                            },
                                            ":",
                                            {
                                                "Ref": "AppPipeline"
                                            }
                                        ]
                                    ]
                                }
                            }
                        ]
                    }
                }
            ]
        }
    },
    "EventRule": {
        "Type": "AWS::Events::Rule",
        "Properties": {
            "EventPattern": {
                "source": [
                    "aws.s3"
                ],
                "detail-type": [
                    "AWS API Call via CloudTrail"
                ],
                "detail": {
                    "eventSource": [
                        "s3.amazonaws.com"
```

```
                    ],
                    "eventName": [
                        "PutObject",
                        "CompleteMultipartUpload"
                    ],
                    "resources": {
                        "ARN": [
                            {
                                "Fn::Join": [
                                    "",
                                    [
                                        {
                                            "Fn::GetAtt": [
                                                "SourceBucket",
                                                "Arn"
                                            ]
                                        },
                                        "/",
                                        {
                                            "Ref": "SourceObjectKey"
                                        }
                                    ]
                                ]
                            }
                        ]
                    }
                }
            },
            "Targets": [
                {
                    "Arn": {
                        "Fn::Join": [
                            "",
                            [
                                "arn:aws:codepipeline:",
                                {
                                    "Ref": "AWS::Region"
                                },
                                ":",
                                {
                                    "Ref": "AWS::AccountId"
                                },
                                ":",
                                {
```

```
                                    "Ref": "AppPipeline"
                                }
                            ]
                        ]
                    },
                    "RoleArn": {
                        "Fn::GetAtt": [
                            "EventRole",
                            "Arn"
                        ]
                    },
                    "Id": "codepipeline-AppPipeline"
                }
            ]
        }
    }
},
"Outputs" : {
    "SourceBucketARN" : {
        "Description" : "S3 bucket ARN that Cloudtrail will use",
        "Value" : { "Fn::GetAtt": ["SourceBucket", "Arn"] },
        "Export" : {
            "Name" : "SourceBucketARN"
        }
    }
}
}


...
```

Migrate polling pipelines for a GitHub (via OAuth app) source action to connections

You can migrate a GitHub (via OAuth app) source action to use connections for your external repository. This is the recommended change detection method for pipelines with a GitHub (via OAuth app) source action.

For a pipeline with a GitHub (via OAuth app) source action, we recommend modifying the pipeline to use a GitHub (via GitHub App) action so that change detection is automated through AWS CodeConnections. For more information about working with connections, see GitHub connections.

Create a connection to GitHub (console)

You can use the console to create a connection to GitHub.

Step 1: Replace your GitHub (via OAuth app) action

Use the pipeline edit page to replace your GitHub (via OAuth app) action with a GitHub (via GitHub App) action.

To replace your GitHub (via OAuth app) action

1. Sign in to the CodePipeline console.
2. Choose your pipeline, and choose **Edit**. Choose **Edit stage** on your source stage. A message displays that recommends you update your action.
3. In **Action provider**, choose **GitHub (via GitHub App)**.
4. Do one of the following:

   - Under **Connection**, if you have not already created a connection to your provider, choose **Connect to GitHub**. Proceed to Step 2: Create a connection to GitHub.
   - Under **Connection**, if you have already created a connection to your provider, choose the connection. Proceed to Step 3: Save the Source Action for Your Connection.

Step 2: Create a connection to GitHub

After you choose to create the connection, the **Connect to GitHub** page is shown.

To create a connection to GitHub

1. Under **GitHub connection settings**, your connection name is shown in **Connection name**.

   Under **GitHub Apps**, choose an app installation or choose **Install a new app** to create one.

   > **Note**
   >
   > You install one app for all of your connections to a particular provider. If you have already installed the GitHub app, choose it and skip this step.

2. If the authorization page for GitHub displays, log in with your credentials and then choose to continue.

3. On the app installation page, a message shows that the AWS CodeStar app is trying to connect to your GitHub account.

> **Note**
>
> You only install the app once for each GitHub account. If you previously installed the app, you can choose **Configure** to proceed to a modification page for your app installation, or you can use the back button to return to the console.

4. On the **Install AWS CodeStar** page, choose **Install**.
5. On the **Connect to GitHub** page, the connection ID for your new installation is displayed. Choose **Connect**.

Step 3: Save your GitHub source action

Complete your updates on the **Edit action** page to save your new source action.

To save your GitHub source action

1. In **Repository**, enter the name of your third-party repository. In **Branch**, enter the branch where you want your pipeline to detect source changes.

> **Note**
>
> In **Repository**, type `owner-name/repository-name` as shown in this example:
>
> ```
> my-account/my-repository
> ```

2. In **Output artifact format**, choose the format for your artifacts.
   - To store output artifacts from the GitHub action using the default method, choose **CodePipeline default**. The action accesses the files from the GitHub 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.

If you choose this option, you will need to update the permissions for your CodeBuild project service role as shown in Add CodeBuild GitClone permissions for connections to Bitbucket, GitHub, GitHub Enterprise Server, or GitLab.com. For a tutorial that shows you how to use the **Full clone** option, see Tutorial: Use full clone with a GitHub pipeline source.

3. In **Output artifacts**, you can retain the name of the output artifact for this action, such as `SourceArtifact`. Choose **Done** to close the **Edit action** page.
4. Choose **Done** to close the stage editing page. Choose **Save** to close the pipeline editing page.

Create a connection to GitHub (CLI)

You can use the AWS Command Line Interface (AWS CLI) to create a connection to GitHub.

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 CloudFormation, use the console to edit the connection to make its status `AVAILABLE`.

To create a connection to GitHub

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 `GitHub` and the specified connection name is `MyConnection`.

```
aws codeconnections create-connection --provider-type GitHub --connection-name
 MyConnection
```

If successful, this command returns the connection ARN information similar to the following.

```
{
    "ConnectionArn": "arn:aws:codeconnections:us-west-2:account_id:connection/
aEXAMPLE-8aad-4d5d-8878-dfcab0bc441f"
}
```

2. Use the console to complete the connection.

Migrate polling pipelines for a GitHub (via OAuth app) source action to webhooks

You can migrate your pipeline to use webhooks to detect changes in your GitHub source repository. This migration to webhooks is for the GitHub (via OAuth app) action only.

- **Console:** Migrate polling pipelines to webhooks (GitHub (via OAuth app) source actions) (console)
- **CLI:** Migrate polling pipelines to webhooks (GitHub (via OAuth app) source actions) (CLI)
- **CloudFormation:** Update pipelines for push events (GitHub (via OAuth app) source actions) (CloudFormation template)

> **Important**
>
> When creating CodePipeline webhooks, do not use your own credentials or reuse the same secret token across multiple webhooks. For optimal security, generate a unique secret token for each webhook you create. The secret token is an arbitrary string that you provide, which GitHub uses to compute and sign the webhook payloads sent to CodePipeline, for protecting the integrity and authenticity of the webhook payloads. Using your own credentials or reusing the same token across multiple webhooks can lead to security vulnerabilities.

Migrate polling pipelines to webhooks (GitHub (via OAuth app) source actions) (console)

For the GitHub (via OAuth app) source action, you can use the CodePipeline console to update your pipeline to use webhooks to detect changes in your GitHub source repository.

Follow these steps to edit a pipeline that is using polling (periodic checks) to use EventBridge instead. If you want to create a pipeline, see Create a pipeline, stages, and actions.

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: AWS CodePipeline will create a webhook for you. You can opt-out in the options below. 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.

Migrate polling pipelines to webhooks (GitHub (via OAuth app) source actions) (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, stages, and actions.

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 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 CloudFormation pipeline is that `PollForSourceChanges` defaults to true and does not display in the pipeline structure output. For more information about PollForSourceChanges defaults, see Valid settings for the `PollForSourceChanges` parameter.

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`:

```
{
    "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.

```
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`.

```
{
    "webhook": {
        "url": "https://webhooks.domain.com/
trigger111111111EXAMPLE1111111111111111",
        "definition": {
            "authenticationConfiguration": {
                "SecretToken": "secret"
            },
            "name": "my-webhook",
            "authentication": "GITHUB_HMAC",
            "targetPipeline": "pipeline_name",
            "targetAction": "Source",
            "filters": [
                {
                    "jsonPath": "$.ref",
                    "matchEquals": "refs/heads/{Branch}"
                }
            ]
        },
        "arn": "arn:aws:codepipeline:eu-central-1:ACCOUNT_ID:webhook:my-webhook"
    },
    "tags": [{
      "key": "Project",
      "value": "ProjectA"
    }]
}
```

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.

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`.

```
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 Valid settings for the `PollForSourceChanges` parameter.

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": "name",
       "Repo": "UserGitHubRepo",
       "PollForSourceChanges": "false",
       "Branch": "main",
       "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.

Update pipelines for push events (GitHub (via OAuth app) source actions) (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 a GitHub webhook resource to your template.

If you use 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
              ActionTypeId:
                Category: Source
                Owner: ThirdParty
                Version: 1
                Provider: GitHub
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                Owner: !Ref GitHubOwner
                Repo: !Ref RepositoryName
                Branch: !Ref BranchName
                OAuthToken:
 {{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
 PollForSourceChanges: true
```

```
            RunOrder: 1


...
```

JSON

```
        "AppPipeline": {
            "Type": "AWS::CodePipeline::Pipeline",
            "Properties": {
                "Name": "github-polling-pipeline",
                "RoleArn": {
                    "Fn::GetAtt": [
                        "CodePipelineServiceRole",
                        "Arn"
                    ]
                },
                "Stages": [
                    {
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "ThirdParty",
                                    "Version": 1,
                                    "Provider": "GitHub"
                                },
                                "OutputArtifacts": [
                                    {
                                        "Name": "SourceOutput"
                                    }
                                ],
                                "Configuration": {
                                    "Owner": {
                                        "Ref": "GitHubOwner"
                                    },
                                    "Repo": {
                                        "Ref": "RepositoryName"
                                    },
                                    "Branch": {
                                        "Ref": "BranchName"
```

```
                                    },
                                    "OAuthToken":
"{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}",
                                    "PollForSourceChanges": true
                                },
                                "RunOrder": 1
                            }
                        ]
                    },


...
```

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 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 CloudFormation template to store your credentials.

When you use the CLI or 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 CloudFormation pipeline is that `PollForSourceChanges` defaults to true and does not display in the pipeline structure output. For more information about PollForSourceChanges defaults, see Valid settings for the `PollForSourceChanges` parameter.

1. In the template, under `Resources`, add your parameters:

   YAML

   ```
   Parameters:
           GitHubOwner:
             Type: String

   ...
   ```

   JSON

   ```
   {
    "Parameters": {
     "BranchName": {
      "Description": "GitHub branch name",
      "Type": "String",
      "Default": "main"
     },
     "GitHubOwner": {
      "Type": "String"
     },

   ...
   ```

2. Use the `AWS::CodePipeline::Webhook` 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

```
 AppPipelineWebhook:
   Type: AWS::CodePipeline::Webhook
   Properties:
     Authentication: GITHUB_HMAC
     AuthenticationConfiguration:
       SecretToken:
{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
     Filters:
       -
         JsonPath: "$.ref"
         MatchEquals: refs/heads/{Branch}
     TargetPipeline: !Ref AppPipeline
     TargetAction: SourceAction
     Name: AppPipelineWebhook
     TargetPipelineVersion: !GetAtt AppPipeline.Version
     RegisterWithThirdParty: true


...
```

JSON

```
"AppPipelineWebhook": {
    "Type": "AWS::CodePipeline::Webhook",
    "Properties": {
        "Authentication": "GITHUB_HMAC",
        "AuthenticationConfiguration": {
            "SecretToken":
 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}"
        },
        "Filters": [{
            "JsonPath": "$.ref",
            "MatchEquals": "refs/heads/{Branch}"
        }],
        "TargetPipeline": {
            "Ref": "AppPipeline"
```

```
        },
        "TargetAction": "SourceAction",
        "Name": "AppPipelineWebhook",
        "TargetPipelineVersion": {
            "Fn::GetAtt": [
              "AppPipeline",
              "Version"
            ]
        },
        "RegisterWithThirdParty": true
    }
},

...
```

3. Save the updated template to your local computer, and then open the 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 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 Valid settings for the `PollForSourceChanges` parameter.

- 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

```
         Name: Source
         Actions:
           -
             Name: SourceAction
             ActionTypeId:
               Category: Source
               Owner: ThirdParty
               Version: 1
               Provider: GitHub
             OutputArtifacts:
               - Name: SourceOutput
             Configuration:
               Owner: !Ref GitHubOwner
               Repo: !Ref RepositoryName
               Branch: !Ref BranchName
               OAuthToken:
{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
               PollForSourceChanges: false
             RunOrder: 1
```

JSON

```
{
   "Name": "Source",
   "Actions": [{
 "Name": "SourceAction",
 "ActionTypeId": {
     "Category": "Source",
     "Owner": "ThirdParty",
     "Version": 1,
     "Provider": "GitHub"
},
"OutputArtifacts": [{
    "Name": "SourceOutput"
}],
"Configuration": {
    "Owner": {
  "Ref": "GitHubOwner"
    },
    "Repo": {
```

```
   "Ref": "RepositoryName"
     },
     "Branch": {
         "Ref": "BranchName"
     },
     "OAuthToken":
 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}",
     PollForSourceChanges: false
 },
 "RunOrder": 1
    }]
```

Example

When you create these resources with 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:
        SecretToken: {{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
      Filters:
        -
          JsonPath: "$.ref"
          MatchEquals: refs/heads/{Branch}
      TargetPipeline: !Ref AppPipeline
      TargetAction: SourceAction
      Name: AppPipelineWebhook
      TargetPipelineVersion: !GetAtt AppPipeline.Version
      RegisterWithThirdParty: true
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
```

```
      Name: github-events-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: ThirdParty
                Version: 1
                Provider: GitHub
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                Owner: !Ref GitHubOwner
                Repo: !Ref RepositoryName
                Branch: !Ref BranchName
                OAuthToken:
 {{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
                PollForSourceChanges: false
              RunOrder: 1

...
```

JSON

```
{
    "Parameters": {
        "BranchName": {
            "Description": "GitHub branch name",
            "Type": "String",
            "Default": "main"
        },
        "RepositoryName": {
            "Description": "GitHub repository name",
            "Type": "String",
            "Default": "test"
        },
        "GitHubOwner": {
            "Type": "String"
```

```
        },
        "ApplicationName": {
            "Description": "CodeDeploy application name",
            "Type": "String",
            "Default": "DemoApplication"
        },
        "BetaFleet": {
            "Description": "Fleet configured in CodeDeploy",
            "Type": "String",
            "Default": "DemoFleet"
        }
    },
    "Resources": {

...

        },
        "AppPipelineWebhook": {
            "Type": "AWS::CodePipeline::Webhook",
            "Properties": {
                "Authentication": "GITHUB_HMAC",
                "AuthenticationConfiguration": {
                    "SecretToken": {

 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}"
                    }
                },
                "Filters": [
                    {
                        "JsonPath": "$.ref",
                        "MatchEquals": "refs/heads/{Branch}"
                    }
                ],
                "TargetPipeline": {
                    "Ref": "AppPipeline"
                },
                "TargetAction": "SourceAction",
                "Name": "AppPipelineWebhook",
                "TargetPipelineVersion": {
                    "Fn::GetAtt": [
                        "AppPipeline",
                        "Version"
                    ]
                },
```

```
                "RegisterWithThirdParty": true
            }
        },
        "AppPipeline": {
            "Type": "AWS::CodePipeline::Pipeline",
            "Properties": {
                "Name": "github-events-pipeline",
                "RoleArn": {
                    "Fn::GetAtt": [
                        "CodePipelineServiceRole",
                        "Arn"
                    ]
                },
                "Stages": [
                    {
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "ThirdParty",
                                    "Version": 1,
                                    "Provider": "GitHub"
                                },
                                "OutputArtifacts": [
                                    {
                                        "Name": "SourceOutput"
                                    }
                                ],
                                "Configuration": {
                                    "Owner": {
                                        "Ref": "GitHubOwner"
                                    },
                                    "Repo": {
                                        "Ref": "RepositoryName"
                                    },
                                    "Branch": {
                                        "Ref": "BranchName"
                                    },
                                    "OAuthToken":
"{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}",
                                    "PollForSourceChanges": false
                                },
```

```
                                "RunOrder": 1

...
```

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 a CodePipeline service role. A service role is required to create a pipeline, and the pipeline is always associated to that service role.

Before you create a pipeline with the AWS CLI, you must create a CodePipeline service role for your pipeline. For an example CloudFormation template with the service role and policy specified, see the tutorials in Tutorial: Create a pipeline that uses variables from AWS CloudFormation deployment actions.

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.

Create the CodePipeline service role (console)

When you use the console to create a pipeline, you create the CodePipeline service role with the pipeline creation wizard.

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

   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.

2. In **Service role**, choose **New service role** to allow CodePipeline to create a new service role in IAM.
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 CloudFormation, you must create a CodePipeline service role for your pipeline and attach the service role policy and the trust policy. To use the CLI to create your service role, use the steps below to first create a trust policy JSON and the role policy JSON as separate files in the directory where you will run the CLI commands.

> **Note**
>
> We recommend that you allow only administrative users to create any service role. A person with permissions to create a role and attach any policy can escalate their own permissions. Instead, create a policy that allows them to create only the roles that they need or have an administrator create the service role on their behalf.

1. In a terminal window, enter the following command to create a file named `TrustPolicy.json`, where you will paste the role policy JSON. This example uses VIM.

```
vim TrustPolicy.json
```

2. Paste the following JSON into the file.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "codepipeline.amazonaws.com"
            },
```

```
            "Action": "sts:AssumeRole"
        }
    ]
}
```

To save and exit the file, enter the following VIM command:

```
:wq
```

3. In a terminal window, enter the following command to create a file named `RolePolicy.json`, where you will paste the role policy JSON. This example uses VIM.

```
vim RolePolicy.json
```

4. Paste the JSON policy into the file. Use the minimum service role policy as provided in CodePipeline service role policy. In addition, add the appropriate permissions to your service role based on the actions you plan to use. For a list of actions and a link to each action's required service role permissions, see Add permissions to the CodePipeline service role.

   Make sure to scope down permissions as much as possible by scoping down to the resource level in the `Resource` field.

   To save and exit the file, enter the following VIM command:

```
:wq
```

5. Enter the following command to create the role and attach the trust role policy. The policy name format is normally the same as the role name format. This example uses the role name `MyRole` and the policy `TrustPolicy` that was created as a separate file.

```
aws iam create-role --role-name MyRole --assume-role-policy-document file://
TrustPolicy.json
```

6. Enter the following command to create the role policy and attach it to the role. The policy name format is normally the same as the role name format. This example uses the role name `MyRole` and the policy `MyRole` that was created as a separate file.

```
aws iam put-role-policy --role-name MyRole --policy-name RolePolicy --policy-
document file://RolePolicy.json
```

7. To view the created role name and trust policy, enter the following command for the role named `MyRole`:

```
aws iam get-role --role-name MyRole
```

8. Use the service role ARN when you create your pipeline with the AWS CLI or CloudFormation.

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
- Tag a custom action in CodePipeline

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.

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.

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)
- Tag pipelines (CLI)

Tag pipelines (console)

You can use the console or the CLI to tag resources. Pipelines are the only CodePipeline resource that can be managed with either the console or the CLI.

Topics

- Add tags to a pipeline (console)
- View tags for a pipeline (console)
- Edit tags for a pipeline (console)
- Remove tags from a pipeline (console)

Add tags to a pipeline (console)

You can use the console to add tags to an existing pipeline.

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

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.

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

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

1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home.
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)
- View tags for a pipeline (CLI)
- Edit tags for a pipeline (CLI)
- Remove tags from a pipeline (CLI)

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, stages, and actions.

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:aws:codepipeline:us-west-2:account-id:MyPipeline` ARN value:

```
aws codepipeline list-tags-for-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:MyPipeline
```

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:

```
aws codepipeline tag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:MyPipeline --tags key=Project,value=ProjectA
```

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*:

```
aws codepipeline untag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:MyPipeline --tag-keys Project IscontainerBased
```

If successful, this command returns nothing. To verify the tags associated with the pipeline, run the **list-tags-for-resource** command.

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://eusc-de-east-1.console.amazonaws-eusc.eu/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 **Amazon Q Developer in chat applications (Slack)** or **SNS topic**. In **Choose target**, choose the name of the client (for a Slack client configured in Amazon Q Developer in chat applications) 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 Amazon Q Developer in chat applications client as part of creating a notification rule. If you choose Amazon Q Developer in chat applications (Slack), you will see a button directing you to configure a client in Amazon Q Developer in chat applications. Choosing that option opens the Amazon Q Developer in chat applications console. For more information, see Configure Integrations Between Notifications and Amazon Q Developer in chat applications.

- 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 Amazon Q Developer in chat applications to send notifications to Amazon Chime chatrooms or Slack channels. For more information, see Configure Integration Between Notifications and Amazon Q Developer in chat applications.

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:

```
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 acccount 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:

```
{
    "Name": "MyNotificationRule",
    "EventTypeIds": [
        "codepipeline-pipeline-pipeline-execution-started"
    ],
    "Resource": "arn:aws:codebuild:us-east-2:123456789012:MyDemoPipeline",
    "Targets": [
        {
            "TargetType": "SNS",
```

```
            "TargetAddress": "arn:aws:sns:us-east-2:123456789012:codestar-
notifications-MyNotificationTopic"
        }
    ],
    "Status": "ENABLED",
    "DetailType": "FULL"
}
```

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:

```
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:

```
{
    "Arn": "arn:aws:codestar-notifications:us-east-1:123456789012:notificationrule/
dc82df7a-EXAMPLE"
}
```

Source actions and change detection methods

When you add a source action to your pipeline, the actions work with additional resources described in the table.

> **Note**
>
> The CodeCommit and S3 source actions require either a configured change detection resource (an EventBridge rule) or use the option to poll the repository for source changes. For pipelines with a Bitbucket, GitHub, or GitHub Enterprise Server source action, you do not have to set up a webhook or default to polling. The connections action manages change detection for you.

Source	Uses additional resources?	Steps	
Amazon S3 with CloudTrail resources	This source action uses an event rule and additional CloudTrail resources. When you use the CLI or CloudFormation to create this action, you also create and manage these resources.	See Create a pipeline, stages, and actions and Connecting to Amazon S3 source actions that use EventBridge and AWS CloudTrail	
Amazon S3 without CloudTrail resources	This source action uses a bucket enabled for events with an event rule without the need for additional CloudTrail resources. When you use the CLI or CloudFormation to create this action, you also create	See Create a pipeline, stages, and actions and Connecting to Amazon S3 source actions with a source enabled for events	

Source	Uses additional resources?	Steps	
	and manage these resources.		
Bitbucket Cloud	This source action uses a connection resource.	See Bitbucket Cloud connections	
AWS CodeCommit	Amazon EventBridge (recommended). This is the default for pipelines with an CodeCommit source created or edited in the console.	See Create a pipeline, stages, and actions and CodeCommit source actions and EventBridge	
Amazon ECR	Amazon EventBridge. This is created by the wizard for pipelines with an Amazon ECR source created or edited in the console.	See Create a pipeline, stages, and actions and Amazon ECR source actions and EventBridge resources.	
GitHub or GitHub Enterprise Cloud	This source action uses a connection resource.	See GitHub connections	
GitHub Enterprise Server	This source action uses a connection resource and a host resource.	See GitHub Enterprise Server connections	
GitLab.com	This source action uses a connection resource.	See GitLab.com connections	
GitLab self-managed	This source action uses a connection resource and a host resource.	See Connections for GitLab self-managed	

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.

If you want to turn off change detection for a source action that uses connections, see CodeStarSourceConnection for Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed actions.

Connect to first-party source providers using source actions

You can use the AWS CodePipeline console or the AWS CLI to connect to source action providers, such as CodeCommit or S3.

> **Note**
>
> When you use the console to create or edit a pipeline, the change detection 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, see CodeCommit source actions and EventBridge.

Topics

- Amazon ECR source actions and EventBridge resources
- Connecting to Amazon S3 source actions with a source enabled for events
- Connecting to Amazon S3 source actions that use EventBridge and AWS CloudTrail
- CodeCommit source actions and EventBridge

Amazon ECR source actions and EventBridge resources

To add an Amazon ECR source action in CodePipeline, you can choose either to:

- Use the CodePipeline console **Create pipeline** wizard (Create a custom pipeline (console)) or **Edit action** page to choose the **Amazon ECR** provider option. The console creates an EventBridge rule that starts your pipeline when the source changes.
- Use the CLI to add the action configuration for the ECR action and create additional resources as follows:
  - Use the ECR example action configuration in Amazon ECR source action reference to create your action as shown in Create a pipeline (CLI).
  - The change detection method defaults to starting the pipeline by polling the source. You should disable periodic checks and create the change detection rule manually. Use one of the following methods: Create an EventBridge rule for an Amazon ECR source (console), Create an

EventBridge rule for an Amazon ECR source (CLI), or Create an EventBridge rule for an Amazon ECR source (CloudFormation template).

Topics

- Create an EventBridge rule for an Amazon ECR source (console)
- Create an EventBridge rule for an Amazon ECR source (CLI)
- Create an EventBridge rule for an Amazon ECR source (CloudFormation template)

Create an EventBridge rule for an Amazon ECR source (console)

To create an EventBridge rule for use in CodePipeline operations (Amazon ECR source)

1. Open the Amazon EventBridge console at https://eusc-de-east-1.console.amazonaws-eusc.eu/events/.
2. In the navigation pane, choose **Events**.
3. Choose **Create rule**, and then under **Event source**, from **Service Name**, choose **Elastic Container Registry (ECR)**.
4. In **Event Source**, choose **Event Pattern**.

   Choose **Edit**, and then paste the following example event pattern in the **Event Source** window for a `eb-test` repository with an image tag of `cli-testing`:

```
{
    "detail-type": [
        "ECR Image Action"
    ],
    "source": [
        "aws.ecr"
    ],
    "detail": {
        "action-type": [
            "PUSH"
        ],
        "image-tag": [
            "latest"
        ],
        "repository-name": [
            "eb-test"
```

```
        ],
        "result": [
            "SUCCESS"
        ]
    }
}
```

> **Note**
>
> To view the full event pattern supported for Amazon ECR events, see Amazon ECR Events and EventBridge or Amazon Elastic Container Registry Events.

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 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 EventBridge permissions to invoke the target associated with your EventBridge rule (in this case, the target is CodePipeline).

   - Choose **Create a new role for this specific resource** to create a service role that gives EventBridge permissions to start your pipeline executions.
   - Choose **Use existing role** to enter a service role that gives EventBridge permissions to start your pipeline executions.

9. (Optional) To specify source overrides with a specific image ID, use the input transformer to pass the data as JSON parameters. You can also use the input transformer to pass pipeline variables.

   - Expand **Additional settings**.

Under **Configure target input**, choose **Configure input transformer**.

In the dialog window, choose **Enter my own**. In the **Input path** box, type the following key-value pairs.

```
{"revisionValue": "$.detail.image-digest"}
```

- In the **Template** box, type the following key-value pairs.

```
{
    "sourceRevisions": [
        {
            "actionName": "Source",
            "revisionType": "IMAGE_DIGEST",
            "revisionValue": "<revisionValue>"
        }
    ],
     "variables": [
        {
            "name": "Variable_Name",
            "value": "Variable_Value"
        }
    ]
}
```

- Choose **Confirm**.

10. Review your rule setup to make sure it meets your requirements.
11. Choose **Configure details**.
12. On the **Configure rule details** page, enter a name and description for the rule, and then choose **State** to enable the rule.
13. If you're satisfied with the rule, choose **Create rule**.

Create an EventBridge 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 EventBridge and Event Patterns.

To create an EventBridge rule with Amazon ECR as the event source and CodePipeline as the target

1. Add permissions for EventBridge to use CodePipeline to invoke the rule. For more information, see Using resource-based policies for Amazon EventBridge.

   a. Use the following sample to create the trust policy that allows EventBridge to assume the service role. Name the trust policy `trustpolicyforEB.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.

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

   c. Create the permissions policy JSON, as shown in this sample, for the pipeline named `MyFirstPipeline`. Name the permissions policy `permissionspolicyforEB.json`.

      JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws-eusc:codepipeline:us-
west-2:111122223333:MyFirstPipeline"
            ]
        }
    ]
}
```

d. Use the following command to attach the `CodePipeline-Permissions-Policy-for-EB` permissions policy to the `Role-for-MyRule` role.

Why am I making this change? Adding this policy to the role creates permissions for EventBridge.

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

2. Call the **put-rule** command and include the `--name`, `--event-pattern`, and `--role-arn` parameters.

Why am I making this change? You 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 started 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\":[\"eb-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
   ```

4. (Optional) To configure an input transformer with source overrides for a specific image ID, use the following JSON in your CLI command. The following example configures an override where:

   - The `actionName`, `Source` in this example, is the dynamic value, defined at pipeline creation, not derived from the source event.
   - The `revisionType`, `IMAGE_DIGEST` in this example, is the dynamic value, defined at pipeline creation, not derived from the source event.
   - The `revisionValue`, <*`revisionValue`*> in this example, is derived from the source event variable.

   ```
   {
       "Rule": "my-rule",
       "Targets": [
           {
               "Id": "MyTargetId",
               "Arn": "ARN",
               "InputTransformer": {
                   "InputPathsMap": {
                       "revisionValue": "$.detail.image-digest"
                   },
   ```

```
                "InputTemplate": {
                    "sourceRevisions": [
                        {
                            "actionName": "Source",
                            "revisionType": "IMAGE_DIGEST",
                            "revisionValue": "<revisionValue>"
                        }
                    ]
                }
            }
        }
    ]
}
```

Create an EventBridge rule for an Amazon ECR source (CloudFormation template)

To use CloudFormation to create a rule, use the template snippet as shown here.

To update your pipeline CloudFormation template and create EventBridge rule

1. In the template, under `Resources`, use the `AWS::IAM::Role` 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? You must create a role that can be assumed by EventBridge to start an execution in our pipeline.

   YAML

   ```
   EventRole:
     Type: AWS::IAM::Role
     Properties:
       AssumeRolePolicyDocument:
         Version: 2012-10-17
         Statement:
   ```

```
          -
            Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Sub arn:aws:codepipeline:${AWS::Region}:
${AWS::AccountId}:${AppPipeline}
```

JSON

```
{
    "EventRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": [
                                "events.amazonaws.com"
                            ]
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "Path": "/",
            "Policies": [
                {
                    "PolicyName": "eb-pipeline-execution",
```

```
                    "PolicyDocument": {
                        "Version": "2012-10-17",
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Action": "codepipeline:StartPipelineExecution",
                                "Resource": {
                                    "Fn::Sub": "arn:aws:codepipeline:
${AWS::Region}:${AWS::AccountId}:${AppPipeline}"
                                }
                            }
                        ]
                    }
                }
            ]
        }
    }
}
...
```

2. In the template, under `Resources`, use the `AWS::Events::Rule` CloudFormation resource to add an EventBridge rule for the Amazon ECR source. This event pattern creates an event that monitors commits to your repository. When EventBridge detects a repository state change, the rule invokes `StartPipelineExecution` on your target pipeline.

   Why am I making this change? You 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 started by the event.

   This snippet uses an image named `eb-test` with a tag of `latest`.

   YAML

```
EventRule:
  Type: 'AWS::Events::Rule'
  Properties:
    EventPattern:
      detail:
        action-type: [PUSH]
        image-tag: [latest]
        repository-name: [eb-test]
        result: [SUCCESS]
      detail-type: [ECR Image Action]
```

```
      source: [aws.ecr]
    Targets:
      - Arn: !Sub arn:aws:codepipeline:${AWS::Region}:${AWS::AccountId}:
${AppPipeline}
        RoleArn: !GetAtt
          - EventRole
          - Arn
        Id: codepipeline-AppPipeline
```

JSON

```
{
    "EventRule": {
        "Type": "AWS::Events::Rule",
        "Properties": {
            "EventPattern": {
                "detail": {
                    "action-type": [
                        "PUSH"
                    ],
                    "image-tag": [
                        "latest"
                    ],
                    "repository-name": [
                        "eb-test"
                    ],
                    "result": [
                        "SUCCESS"
                    ]
                },
                "detail-type": [
                    "ECR Image Action"
                ],
                "source": [
                    "aws.ecr"
                ]
            },
            "Targets": [
                {
                    "Arn": {
                        "Fn::Sub": "arn:aws:codepipeline:${AWS::Region}:
${AWS::AccountId}:${AppPipeline}"
                    },
```

```
                    "RoleArn": {
                        "Fn::GetAtt": [
                            "EventRole",
                            "Arn"
                        ]
                    },
                    "Id": "codepipeline-AppPipeline"
                }
            ]
        }
    }
},
```

> **Note**
>
> To view the full event pattern supported for Amazon ECR events, see Amazon ECR Events and EventBridge or Amazon Elastic Container Registry Events.

3. (Optional) To configure an input transformer with source overrides for a specific image ID, use the following YAML snippet. The following example configures an override where:

   - The `actionName`, `Source` in this example, is the dynamic value, defined at pipeline creation, not derived from the source event.
   - The `revisionType`, `IMAGE_DIGEST` in this example, is the dynamic value, defined at pipeline creation, not derived from the source event.
   - The `revisionValue`, <*`revisionValue`*> in this example, is derived from the source event variable.

```
---
Rule: my-rule
Targets:
- Id: MyTargetId
  Arn: ARN
  InputTransformer:
    InputPathsMap:
      revisionValue: "$.detail.image-digest"
    InputTemplate:
      sourceRevisions:
        actionName: Source
```

```
        revisionType: IMAGE_DIGEST
        revisionValue: '<revisionValue>'
```

4. Save the updated template to your local computer, and then open the CloudFormation console.
5. Choose your stack, and then choose **Create Change Set for Current Stack**.
6. Upload the template, and then view the changes listed in CloudFormation. These are the changes to be made to the stack. You should see your new resources in the list.
7. Choose **Execute**.

Connecting to Amazon S3 source actions with a source enabled for events

The instructions in this section provide the steps for creating the S3 source action that does not require that you create or manage AWS CloudTrail resources.

> **Important**
>
> The procedure to create this action without AWS CloudTrail resources is not available in the console. To use the CLI, see the procedures here or see Migrate polling pipelines with an S3 source enabled for events.

For a pipeline with an Amazon S3 source, modify the pipeline so that change detection is automated through EventBridge and with a source bucket that is enabled for event notifications. This is the recommend method if you are using the CLI or CloudFormation to migrate your pipeline.

> **Note**
>
> This includes using a bucket that is enabled for event notifications, where you do not need to create a separate CloudTrail trail. If you are using the console, then an event rule and CloudTrail trail are set up for you. For those steps, see Migrate polling pipelines with an S3 source and CloudTrail trail.

- **CLI:** Migrate polling pipelines with an S3 source and CloudTrail trail (CLI)

- **CloudFormation:** Migrate polling pipelines with an S3 source and CloudTrail trail (CloudFormation template)

Create pipelines with an S3 source enabled for events (CLI)

Follow these steps to create a pipeline with an S3 source that uses an event in EventBridge for change detection. For the full steps to create a pipeline with the CLI, see Create a pipeline, stages, and actions.

To build an event-driven pipeline with Amazon S3, you edit the `PollForSourceChanges` parameter of your pipeline and then create the following resources:

- EventBridge event rule
- IAM role to allow the EventBridge event to start your pipeline

To create an EventBridge rule with Amazon S3 as the event source and CodePipeline as the target and apply the permissions policy

1. Grant permissions for EventBridge to use CodePipeline to invoke the rule. For more information, see Using resource-based policies for Amazon EventBridge.

   a. Use the following sample to create the trust policy to allow EventBridge to assume the service role. Name it `trustpolicyforEB.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 EventBridge.

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. Create the permissions policy JSON, as shown here for the pipeline named `MyFirstPipeline`. Name the permissions policy `permissionspolicyforEB.json`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws-eusc:codepipeline:us-
west-2:111122223333:MyFirstPipeline"
            ]
        }
    ]
}
```

d. Use the following command to attach the new `CodePipeline-Permissions-Policy-for-EB` 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-EB --policy-document file://permissionspolicyforEB.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 `EnabledS3SourceRule`.

```
aws events put-rule --name "EnabledS3SourceRule" --event-pattern "{\"source\":
[\"aws.s3\"],\"detail-type\":[\"Object Created\"],\"detail\":{\"bucket\":{\"name\":
[\"amzn-s3-demo-source-bucket\"]}}}" --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 `EnabledS3SourceRule`, 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 EnabledS3SourceRule --targets Id=codepipeline-
   AppPipeline,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 Valid settings for the `PollForSourceChanges` parameter.

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:

   ```
   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 `amzn-s3-demo-source-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.

```
"configuration": {
    "S3Bucket": "amzn-s3-demo-source-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:

   ```
   "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.

Create pipelines with an S3 source enabled for events (CloudFormation template)

This procedure is for a pipeline where the source bucket has events enabled.

Use these steps to create a pipeline with an Amazon S3 source for 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:

- EventBridge rule and IAM role to allow this event to start your pipeline.

If you use 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": "SourceObjectKey"
                                },
                                "PollForSourceChanges": true
                            },
                            "RunOrder": 1
                        }
                    ]
                },


...
```

To create an EventBridge 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` 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 CloudFormation to create permissions for EventBridge. This resource is added to your CloudFormation stack.

   YAML

   ```
   EventRole:
     Type: AWS::IAM::Role
     Properties:
       AssumeRolePolicyDocument:
         Version: 2012-10-17
         Statement:
           -
             Effect: Allow
             Principal:
               Service:
   ```

```
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]


...
```

JSON

```
 "EventRole": {
   "Type": "AWS::IAM::Role",
   "Properties": {
     "AssumeRolePolicyDocument": {
       "Version": "2012-10-17",
       "Statement": [
         {
           "Effect": "Allow",
           "Principal": {
             "Service": [
               "events.amazonaws.com"
             ]
           },
           "Action": "sts:AssumeRole"
         }
       ]
     },
     "Path": "/",
     "Policies": [
       {
         "PolicyName": "eb-pipeline-execution",
         "PolicyDocument": {
           "Version": "2012-10-17",
```

```
            "Statement": [
              {
                "Effect": "Allow",
                "Action": "codepipeline:StartPipelineExecution",
                "Resource": {
                  "Fn::Join": [
                    "",
                    [
                      "arn:aws:codepipeline:",
                      {
                        "Ref": "AWS::Region"
                      },
                      ":",
                      {
                        "Ref": "AWS::AccountId"
                      },
                      ":",
                      {
                        "Ref": "AppPipeline"
                      }
                    ]
                  ]

...
```

2. Use the `AWS::Events::Rule` CloudFormation resource to add an EventBridge rule. This event pattern creates an event that monitors creation or deletion of objects in your Amazon S3 source bucket. In addition, include a target of your pipeline. When an object is created, this rule invokes `StartPipelineExecution` on your target pipeline.

   Why am I making this change? Adding the `AWS::Events::Rule` resource enables CloudFormation to create the event. This resource is added to your CloudFormation stack.

   YAML

```
  EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventBusName: default
      EventPattern:
        source:
          - aws.s3
        detail-type:
```

```
          - Object Created
        detail:
          bucket:
            name:
              - !Ref SourceBucket
      Name: EnabledS3SourceRule
      State: ENABLED
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline


...
```

JSON

```
 "EventRule": {
   "Type": "AWS::Events::Rule",
   "Properties": {
 "EventBusName": "default",
 "EventPattern": {
     "source": [
  "aws.s3"
     ],
     "detail-type": [
   "Object Created"
     ],
     "detail": {
   "bucket": {
       "name": [
     "s3-pipeline-source-fra-bucket"
       ]
       }
           }
 },
 "Name": "EnabledS3SourceRule",
       "State": "ENABLED",
       "Targets": [
```

```
        {
          "Arn": {
            "Fn::Join": [
              "",
              [
                "arn:aws:codepipeline:",
                {
                  "Ref": "AWS::Region"
                },
                ":",
                {
                  "Ref": "AWS::AccountId"
                },
                ":",
                {
                  "Ref": "AppPipeline"
                }
              ]
            ]
          },
          "RoleArn": {
            "Fn::GetAtt": [
              "EventRole",
              "Arn"
            ]
          },
          "Id": "codepipeline-AppPipeline"
        }
      ]
    }
  }
},

...
```

3. Save your updated template to your local computer, and open the CloudFormation console.
4. Choose your stack, and then choose **Create Change Set for Current Stack**.
5. Upload your updated template, and then view the changes listed in CloudFormation. These are the changes that will 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 Valid settings for the `PollForSourceChanges` parameter.

- 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

```
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: S3
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                S3Bucket: !Ref SourceBucket
                S3ObjectKey: !Ref SourceObjectKey
                PollForSourceChanges: false
              RunOrder: 1
```

  JSON

```
{
   "Name": "SourceAction",
   "ActionTypeId": {
```

```
      "Category": "Source",
      "Owner": "AWS",
      "Version": 1,
      "Provider": "S3"
    },
    "OutputArtifacts": [
      {
        "Name": "SourceOutput"
      }
    ],
    "Configuration": {
      "S3Bucket": {
        "Ref": "SourceBucket"
      },
      "S3ObjectKey": {
        "Ref": "SourceObjectKey"
      },
      "PollForSourceChanges": false
    },
    "RunOrder": 1
  }
```

Example

When you use 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 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

```
Parameters:
  SourceObjectKey:
    Description: 'S3 source artifact'
    Type: String
```

```
    Default: SampleApp_Linux.zip
  ApplicationName:
    Description: 'CodeDeploy application name'
    Type: String
    Default: DemoApplication
  BetaFleet:
    Description: 'Fleet configured in CodeDeploy'
    Type: String
    Default: DemoFleet

Resources:
  SourceBucket:
    Type: AWS::S3::Bucket
    Properties:
      NotificationConfiguration:
        EventBridgeConfiguration:
          EventBridgeEnabled: true
      VersioningConfiguration:
        Status: Enabled
  CodePipelineArtifactStoreBucket:
    Type: AWS::S3::Bucket
  CodePipelineArtifactStoreBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref CodePipelineArtifactStoreBucket
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Sid: DenyUnEncryptedObjectUploads
            Effect: Deny
            Principal: '*'
            Action: s3:PutObject
            Resource: !Join [ '', [ !GetAtt CodePipelineArtifactStoreBucket.Arn, '/
*' ] ]
            Condition:
              StringNotEquals:
                s3:x-amz-server-side-encryption: aws:kms
          -
            Sid: DenyInsecureConnections
            Effect: Deny
            Principal: '*'
            Action: s3:*
            Resource: !Sub ${CodePipelineArtifactStoreBucket.Arn}/*
```

```
            Condition:
              Bool:
                aws:SecureTransport: false
  CodePipelineServiceRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - codepipeline.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: AWS-CodePipeline-Service-3
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action:
                  - codecommit:CancelUploadArchive
                  - codecommit:GetBranch
                  - codecommit:GetCommit
                  - codecommit:GetUploadArchiveStatus
                  - codecommit:UploadArchive
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
                  - codedeploy:CreateDeployment
                  - codedeploy:GetApplicationRevision
                  - codedeploy:GetDeployment
                  - codedeploy:GetDeploymentConfig
                  - codedeploy:RegisterApplicationRevision
                Resource: 'resource_ARN'
              -
                Effect: Allow
                Action:
                  - codebuild:BatchGetBuilds
```

```
                - codebuild:StartBuild
              Resource: 'resource_ARN'
            -
              Effect: Allow
              Action:
                - devicefarm:ListProjects
                - devicefarm:ListDevicePools
                - devicefarm:GetRun
                - devicefarm:GetUpload
                - devicefarm:CreateUpload
                - devicefarm:ScheduleRun
              Resource: 'resource_ARN'
            -
              Effect: Allow
              Action:
                - lambda:InvokeFunction
                - lambda:ListFunctions
              Resource: 'resource_ARN'
            -
              Effect: Allow
              Action:
                - iam:PassRole
              Resource: 'resource_ARN'
            -
              Effect: Allow
              Action:
                - elasticbeanstalk:*
                - ec2:*
                - elasticloadbalancing:*
                - autoscaling:*
                - cloudwatch:*
                - s3:*
                - sns:*
                - cloudformation:*
                - rds:*
                - sqs:*
                - ecs:*
              Resource: 'resource_ARN'
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: s3-events-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
```

```
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: S3
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                S3Bucket: !Ref SourceBucket
                S3ObjectKey: !Ref SourceObjectKey
                PollForSourceChanges: false
              RunOrder: 1
        -
          Name: Beta
          Actions:
            -
              Name: BetaAction
              InputArtifacts:
                - Name: SourceOutput
              ActionTypeId:
                Category: Deploy
                Owner: AWS
                Version: 1
                Provider: CodeDeploy
              Configuration:
                ApplicationName: !Ref ApplicationName
                DeploymentGroupName: !Ref BetaFleet
              RunOrder: 1
      ArtifactStore:
        Type: S3
        Location: !Ref CodePipelineArtifactStoreBucket
  EventRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
```

```
            Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region',
':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
 EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventBusName: default
      EventPattern:
        source:
          - aws.s3
        detail-type:
          - Object Created
        detail:
          bucket:
            name:
              - !Ref SourceBucket
      Name: EnabledS3SourceRule
      State: ENABLED
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline
```

JSON

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "appconfig:StartDeployment",
                "appconfig:StopDeployment",
                "appconfig:GetDeployment"
            ],
            "Resource": [
                "arn:aws-eusc:appconfig:*:111122223333:application/
[[Application]]",
                "arn:aws-eusc:appconfig:*:111122223333:application/
[[Application]]/*",
                "arn:aws-eusc:appconfig:*:111122223333:deploymentstrategy/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Connecting to Amazon S3 source actions that use EventBridge and AWS CloudTrail

The instructions in this section provide the steps for creating the S3 source action that uses AWS CloudTrail resources that you must create and manage. To use the S3 source action with EventBridge that does not require additional AWS CloudTrail resources, use the CLI instructions at Migrate polling pipelines with an S3 source enabled for events.

Important

This procedure provides the steps for creating the S3 source action that uses AWS CloudTrail resources that you must create and manage. The procedure to create this action

without AWS CloudTrail resources is not available in the console. To use the CLI, see Migrate polling pipelines with an S3 source enabled for events.

To add an Amazon S3 source action in CodePipeline, you choose either to:

- Use the CodePipeline console **Create pipeline** wizard (Create a custom pipeline (console)) or **Edit action** page to choose the **S3** provider option. The console creates an EventBridge rule and a CloudTrail trail that starts your pipeline when the source changes.
- Use the AWS CLI to add the action configuration for the S3 action and create additional resources as follows:
  - Use the S3 example action configuration in Amazon S3 source action reference to create your action as shown in Create a pipeline (CLI).
  - The change detection method defaults to starting the pipeline by polling the source. You should disable periodic checks and create the change detection rule and trail manually. Use one of the following methods: Create an EventBridge rule for an Amazon S3 source (console), Create an EventBridge rule for an Amazon S3 source (CLI), or Create an EventBridge rule for an Amazon S3 source (CloudFormation template).

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 EventBridge rule. The EventBridge rule detects the source change and then starts your pipeline.

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 EventBridge 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 an EventBridge rule for an Amazon S3 source (console)

Before you set up a rule in EventBridge, 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 EventBridge 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. 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.
5. Under **Trail log bucket and folder**, 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. Complete the required encryption key information and choose **Next**.
6. For **Event type**, choose **Management events**.
7. For **Management events**, choose **Write**. The trail records Amazon S3 object-level API activity (for example, `GetObject` and `PutObject`) on the specified bucket and prefix.
8. Choose **Write**.
9. If you're satisfied with the trail, choose **Create trail**.

To create an EventBridge rule that targets your pipeline with an Amazon S3 source

1. Open the Amazon EventBridge console at https://eusc-de-east-1.console.amazonaws-eusc.eu/events/.
2. In the navigation pane, choose **Rules**. Leave the default bus selected or choose an event bus. Choose **Create rule**.

3. In **Name**, enter a name for your rule.
4. Under **Rule type**, choose **Rule with an event pattern**. Choose **Next**.
5. Under **Event source**, choose **AWS events or EventBridge partner events**.
6. Under **Sample event type**, choose **AWS events**.
7. In **Sample events**, type S3 as the keyword to filter on. Choose **AWS API call via CloudTrail**.
8. Under **Creation method**, choose **Customer pattern (JSON editor)**.

   Paste the event pattern provided below. Make sure to add the bucket name and S3 object key (or key name) which uniquely identifies the object in the bucket as `requestParameters`. In this example, a rule is created for a bucket named `amzn-s3-demo-source-bucket` and an object key of `my-files.zip`. 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:

```
{
    "source": [
        "aws.s3"
    ],
    "detail-type": [
        "AWS API Call via CloudTrail"
    ],
    "detail": {
        "eventSource": [
            "s3.amazonaws.com"
        ],
        "eventName": [
            "CopyObject",
            "CompleteMultipartUpload",
            "PutObject"
        ],
        "requestParameters": {
            "bucketName": [
                "amzn-s3-demo-source-bucket"
            ],
            "key": [
                "my-files.zip"
            ]
        }
    }
```

```
}
```

9. Choose **Next**.
10. In **Target types**, choose **AWS service**.
11. In **Select a target**, choose **CodePipeline**. In **Pipeline ARN**, enter the pipeline ARN for the pipeline to be started 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

12. To create or specify an IAM service role that grants EventBridge permissions to invoke the target associated with your EventBridge rule (in this case, the target is CodePipeline):

- Choose **Create a new role for this specific resource** to create a service role that gives EventBridge permissions to start your pipeline executions.
- Choose **Use existing role** to enter a service role that gives EventBridge permissions to start your pipeline executions.

13. (Optional) To specify source overrides with a specific image ID, use the input transformer to pass the data as JSON parameters. You can also use the input transformer to pass pipeline variables.

- Expand **Additional settings**.

  Under **Configure target input**, choose **Configure input transformer**.

  In the dialog window, choose **Enter my own**. In the **Input path** box, type the following key-value pairs.

```
{"revisionValue": "$.detail.object.version-id"}
```

- In the **Template** box, type the following key-value pairs.

```

```

```
                                {
    "sourceRevisions": [
        {
            "actionName": "Source",
            "revisionType": "S3_OBJECT_VERSION_ID",
            "revisionValue": "<revisionValue>"
        }
    ],
     "variables": [
        {
            "name": "Variable_Name",
            "value": "Variable_Value"
        }
    ]
}
```

- Choose **Confirm**.

14. Choose **Next**.
15. On the **Tags** page, choose **Next**.
16. On the **Review and create** page, review the rule configuration. If you're satisfied with the rule, choose **Create rule**.

Create an EventBridge 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 `amzn-s3-demo-source-bucket`.

```
aws cloudtrail create-trail --name my-trail --s3-bucket-name amzn-s3-demo-source-bucket
```

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 EventBridge.

   Example:

   The following command uses `--name` to start logging on a trail named `my-trail`.

   ```
   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 EventBridge 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 `amzn-s3-demo-source-bucket/myFolder`.

   ```
   aws cloudtrail put-event-selectors --trail-name my-trail --event-selectors
    '[{ "ReadWriteType": "WriteOnly", "IncludeManagementEvents":false,
    "DataResources": [{ "Type": "AWS::S3::Object", "Values": ["arn:aws:s3:::amzn-s3-demo-source-bucket/myFolder/file.zip"] }] }]'
   ```

To create an EventBridge rule with Amazon S3 as the event source and CodePipeline as the target and apply the permissions policy

1. Grant permissions for EventBridge to use CodePipeline to invoke the rule. For more information, see Using resource-based policies for Amazon EventBridge.

a. Use the following sample to create the trust policy to allow EventBridge to assume the service role. Name it `trustpolicyforEB.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 EventBridge.

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. Create the permissions policy JSON, as shown here for the pipeline named `MyFirstPipeline`. Name the permissions policy `permissionspolicyforEB.json`.

JSON

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

```
            "Resource": [
                "arn:aws-eusc:codepipeline:us-
west-2:111122223333:MyFirstPipeline"
            ]
        }
    ]
}
```

d. Use the following command to attach the new `CodePipeline-Permissions-Policy-for-EB` 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-EB --policy-document file://permissionspolicyforEB.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\":[\"amzn-s3-
demo-source-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
```

4. (Optional) To configure an input transformer with source overrides for a specific image ID, use the following JSON in your CLI command. The following example configures an override where:

- The `actionName`, `Source` in this example, is the dynamic value, defined at pipeline creation, not derived from the source event.
- The `revisionType`, `S3_OBJECT_VERSION_ID` in this example, is the dynamic value, defined at pipeline creation, not derived from the source event.
- The `revisionValue`, `<revisionValue>` in this example, is derived from the source event variable.

```
{
    "Rule": "my-rule",
    "Targets": [
        {
            "Id": "MyTargetId",
            "Arn": "ARN",
            "InputTransformer": {
                "InputPathsMap": {
                    "revisionValue": "$.detail.object.version-id"
                },
                "InputTemplate": {
                    "sourceRevisions": {
                        "actionName": "Source",
                        "revisionType": "S3_OBJECT_VERSION_ID",
                        "revisionValue": "<revisionValue>"
                    }
                }
            }
        }
    ]
}
```

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 Valid settings for the `PollForSourceChanges` parameter.

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:

```
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 `amzn-s3-demo-source-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.

```
"configuration": {
    "S3Bucket": "amzn-s3-demo-source-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:

```
"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.

Create an EventBridge rule for an Amazon S3 source (CloudFormation template)

To use CloudFormation to create a rule, update your template as shown here.

To create an EventBridge 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` 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 CloudFormation to create permissions for EventBridge. This resource is added to your CloudFormation stack.

YAML

```
  EventRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]


...
```

JSON

```
  "EventRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": [
```

```
                "events.amazonaws.com"
              ]
            },
            "Action": "sts:AssumeRole"
          }
        ]
      },
      "Path": "/",
      "Policies": [
        {
          "PolicyName": "eb-pipeline-execution",
          "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
              {
                "Effect": "Allow",
                "Action": "codepipeline:StartPipelineExecution",
                "Resource": {
                  "Fn::Join": [
                    "",
                    [
                      "arn:aws:codepipeline:",
                      {
                        "Ref": "AWS::Region"
                      },
                      ":",
                      {
                        "Ref": "AWS::AccountId"
                      },
                      ":",
                      {
                        "Ref": "AppPipeline"
                      }
                    ]
                  ]

...
```

2. Use the `AWS::Events::Rule` CloudFormation resource to add an EventBridge 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 CloudFormation to create the event. This resource is added to your CloudFormation stack.

YAML

```
  EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventPattern:
        source:
          - aws.s3
        detail-type:
          - 'AWS API Call via CloudTrail'
        detail:
          eventSource:
            - s3.amazonaws.com
          eventName:
            - CopyObject
            - PutObject
            - CompleteMultipartUpload
          requestParameters:
            bucketName:
              - !Ref SourceBucket
            key:
              - !Ref SourceObjectKey
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline


...
```

JSON

```
  "EventRule": {
    "Type": "AWS::Events::Rule",
    "Properties": {
```

```
      "EventPattern": {
        "source": [
          "aws.s3"
        ],
        "detail-type": [
          "AWS API Call via CloudTrail"
        ],
        "detail": {
          "eventSource": [
            "s3.amazonaws.com"
          ],
          "eventName": [
            "CopyObject",
            "PutObject",
            "CompleteMultipartUpload"
          ],
          "requestParameters": {
            "bucketName": [
              {
                "Ref": "SourceBucket"
              }
            ],
            "key": [
              {
                "Ref": "SourceObjectKey"
              }
            ]
          }
        }
      },
      "Targets": [
        {
          "Arn": {
            "Fn::Join": [
              "",
              [
                "arn:aws:codepipeline:",
                {
                  "Ref": "AWS::Region"
                },
                ":",
                {
                  "Ref": "AWS::AccountId"
                },
```

```
                ":",
                {
                  "Ref": "AppPipeline"
                }
              ]
            ]
          },
          "RoleArn": {
            "Fn::GetAtt": [
              "EventRole",
              "Arn"
            ]
          },
          "Id": "codepipeline-AppPipeline"
        }
      ]
    }
  }
},

...
```

3. Add this snippet to your first template to allow cross-stack functionality:

   YAML

```
Outputs:
  SourceBucketARN:
    Description: "S3 bucket ARN that Cloudtrail will use"
    Value: !GetAtt SourceBucket.Arn
    Export:
      Name: SourceBucketARN
```

   JSON

```
  "Outputs" : {
    "SourceBucketARN" : {
      "Description" : "S3 bucket ARN that Cloudtrail will use",
      "Value" : { "Fn::GetAtt": ["SourceBucket", "Arn"] },
      "Export" : {
        "Name" : "SourceBucketARN"
      }
    }
```

```
...
```

4. (Optional) To configure an input transformer with source overrides for a specific image ID, use the following YAML snippet. The following example configures an override where:

   - The `actionName`, `Source` in this example, is the dynamic value, defined at pipeline creation, not derived from the source event.
   - The `revisionType`, `S3_OBJECT_VERSION_ID` in this example, is the dynamic value, defined at pipeline creation, not derived from the source event.
   - The `revisionValue`, <*`revisionValue`*> in this example, is derived from the source event variable.

```
---
Rule: my-rule
Targets:
- Id: MyTargetId
  Arn: pipeline-ARN
  InputTransformer:
    InputPathsMap:
      revisionValue: "$.detail.object.version-id"
    InputTemplate:
      sourceRevisions:
        actionName: Source
        revisionType: S3_OBJECT_VERSION_ID
        revisionValue: '<revisionValue>'
```

5. Save your updated template to your local computer, and open the CloudFormation console.
6. Choose your stack, and then choose **Create Change Set for Current Stack**.
7. Upload your updated template, and then view the changes listed in CloudFormation. These are the changes that will be made to the stack. You should see your new resources in the list.
8. 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 Valid settings for the `PollForSourceChanges` parameter.

- 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

```
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: S3
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                S3Bucket: !Ref SourceBucket
                S3ObjectKey: !Ref SourceObjectKey
                PollForSourceChanges: false
              RunOrder: 1
```

  JSON

```
{
   "Name": "SourceAction",
   "ActionTypeId": {
```

```
      "Category": "Source",
      "Owner": "AWS",
      "Version": 1,
      "Provider": "S3"
    },
    "OutputArtifacts": [
      {
        "Name": "SourceOutput"
      }
    ],
    "Configuration": {
      "S3Bucket": {
        "Ref": "SourceBucket"
      },
      "S3ObjectKey": {
        "Ref": "SourceObjectKey"
      },
      "PollForSourceChanges": false
    },
    "RunOrder": 1
  }
```

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` 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
            Resource: !Join [ '', [ !GetAtt AWSCloudTrailBucket.Arn, '/
AWSLogs/', !Ref 'AWS::AccountId', '/*' ] ]
            Condition:
              StringEquals:
                s3:x-amz-acl: bucket-owner-full-control
  AWSCloudTrailBucket:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
  AwsCloudTrail:
    DependsOn:
      - AWSCloudTrailBucketPolicy
    Type: AWS::CloudTrail::Trail
    Properties:
      S3BucketName: !Ref AWSCloudTrailBucket
```

```
      EventSelectors:
        -
          DataResources:
            -
              Type: AWS::S3::Object
              Values:
                - !Join [ '', [ !ImportValue SourceBucketARN, '/', !Ref
 SourceObjectKey ] ]
          ReadWriteType: WriteOnly
          IncludeManagementEvents: false
      IncludeGlobalServiceEvents: true
      IsLogging: true
      IsMultiRegionTrail: true


...
```

JSON

```
{
  "Parameters": {
    "SourceObjectKey": {
      "Description": "S3 source artifact",
      "Type": "String",
      "Default": "SampleApp_Linux.zip"
    }
  },
  "Resources": {
    "AWSCloudTrailBucket": {
      "Type": "AWS::S3::Bucket",
        "DeletionPolicy": "Retain"
    },
    "AWSCloudTrailBucketPolicy": {
      "Type": "AWS::S3::BucketPolicy",
      "Properties": {
        "Bucket": {
          "Ref": "AWSCloudTrailBucket"
        },
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "AWSCloudTrailAclCheck",
```

```
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "cloudtrail.amazonaws.com"
        ]
      },
      "Action": "s3:GetBucketAcl",
      "Resource": {
        "Fn::GetAtt": [
          "AWSCloudTrailBucket",
          "Arn"
        ]
      }
    },
    {
      "Sid": "AWSCloudTrailWrite",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "cloudtrail.amazonaws.com"
        ]
      },
      "Action": "s3:PutObject",
      "Resource": {
        "Fn::Join": [
          "",
          [
            {
              "Fn::GetAtt": [
                "AWSCloudTrailBucket",
                "Arn"
              ]
            },
            "/AWSLogs/",
            {
              "Ref": "AWS::AccountId"
            },
            "/*"
          ]
        ]
      },
      "Condition": {
        "StringEquals": {
          "s3:x-amz-acl": "bucket-owner-full-control"
```

```
              }
            }
          }
        ]
      }
    }
  },
  "AwsCloudTrail": {
    "DependsOn": [
      "AWSCloudTrailBucketPolicy"
    ],
    "Type": "AWS::CloudTrail::Trail",
    "Properties": {
      "S3BucketName": {
        "Ref": "AWSCloudTrailBucket"
      },
      "EventSelectors": [
        {
          "DataResources": [
            {
              "Type": "AWS::S3::Object",
              "Values": [
                {
                  "Fn::Join": [
                    "",
                    [
                      {
                        "Fn::ImportValue": "SourceBucketARN"
                      },
                      "/",
                      {
                        "Ref": "SourceObjectKey"
                      }
                    ]
                  ]
                }
              ]
            }
          ],
          "ReadWriteType": "WriteOnly",
          "IncludeManagementEvents": false
        }
      ],
      "IncludeGlobalServiceEvents": true,
```

```
        "IsLogging": true,
        "IsMultiRegionTrail": true
      }
    }
  }
}

...
```

CodeCommit source actions and EventBridge

To add a CodeCommit source action in CodePipeline, you can choose either to:

- Use the CodePipeline console **Create pipeline** wizard (Create a custom pipeline (console)) or **Edit action** page to choose the **CodeCommit** provider option. The console creates an EventBridge rule that starts your pipeline when the source changes.
- Use the AWS CLI to add the action configuration for the `CodeCommit` action and create additional resources as follows:
  - Use the `CodeCommit` example action configuration in CodeCommit source action reference to create your action as shown in Create a pipeline (CLI).
  - The change detection method defaults to starting the pipeline by polling the source. You should disable periodic checks and create the change detection rule manually. Use one of the following methods: Create an EventBridge rule for a CodeCommit source (console), Create an EventBridge rule for a CodeCommit source (CLI), or Create an EventBridge rule for a CodeCommit source (CloudFormation template).

Topics

- Create an EventBridge rule for a CodeCommit source (console)
- Create an EventBridge rule for a CodeCommit source (CLI)
- Create an EventBridge rule for a CodeCommit source (CloudFormation template)

Create an EventBridge rule for a CodeCommit source (console)

> **Important**
>
> If you use the console to create or edit your pipeline, your EventBridge rule is created for you.

To create an EventBridge rule for use in CodePipeline operations

1. Open the Amazon EventBridge console at https://eusc-de-east-1.console.amazonaws-eusc.eu/events/.
2. In the navigation pane, choose **Rules**. Leave the default bus selected or choose an event bus. Choose **Create rule**.
3. In **Name**, enter a name for your rule.
4. Under **Rule type**, choose **Rule with an event pattern**. Choose **Next**.
5. Under **Event source**, choose **AWS events or EventBridge partner events**.
6. Under **Sample event type**, choose **AWS events**.
7. In **Sample events**, type CodeCommit as the keyword to filter on. Choose **CodeCommit Repository State Change**.
8. Under **Creation method**, choose **Customer pattern (JSON editor)**.

   Paste the event pattern provided below. The following is a sample CodeCommit event pattern in the **Event** window for a `MyTestRepo` repository with a branch named `main`:

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

```
      "branch"
    ],
    "referenceName": [
      "main"
    ]
  }
}
```

9. In **Targets**, choose **CodePipeline**.
10. Enter the pipeline ARN for the pipeline to be started 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`

11. To create or specify an IAM service role that grants EventBridge permissions to invoke the target associated with your EventBridge rule (in this case, the target is CodePipeline):

    - Choose **Create a new role for this specific resource** to create a service role that gives EventBridge permissions to start your pipeline executions.
    - Choose **Use existing role** to enter a service role that gives EventBridge permissions to start your pipeline executions.

12. (Optional) To specify source overrides with a specific image ID, use the input transformer to pass the data as JSON parameters. You can also use the input transformer to pass pipeline variables.

    - Expand **Additional settings**.

      Under **Configure target input**, choose **Configure input transformer**.

      In the dialog window, choose **Enter my own**. In the **Input path** box, type the following key-value pairs.

      ```
      {"revisionValue": "$.detail.image-digest"}
      ```

    - In the **Template** box, type the following key-value pairs.

```
{
    "sourceRevisions": [
        {
            "actionName": "Source",
            "revisionType": "COMMIT_ID",
            "revisionValue": "<revisionValue>"
        }
    ],
    "variables": [
        {
            "name": "Branch_Name",
            "value": "value"
        }
    ]
}
```

- Choose **Confirm**.

13. Choose **Next**.
14. On the **Tags** page, choose **Next**.
15. On the **Review and create** page, review the rule configuration. If you're satisfied with the rule, choose **Create rule**.

Create an EventBridge 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.
- The event pattern for the source and detail fields used by the rule. For more information, see Amazon EventBridge and Event Patterns.

To create an EventBridge rule with CodeCommit as the event source and CodePipeline as the target

1. Add permissions for EventBridge to use CodePipeline to invoke the rule. For more information, see Using resource-based policies for Amazon EventBridge.

a. Use the following sample to create the trust policy that allows EventBridge to assume the service role. Name the trust policy `trustpolicyforEB.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.

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. Create the permissions policy JSON, as shown in this sample, for the pipeline named `MyFirstPipeline`. Name the permissions policy `permissionspolicyforEB.json`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws-eusc:codepipeline:us-
west-2:111122223333:MyFirstPipeline"
```

```
            ]
        }
    ]
}
```

d. Use the following command to attach the `CodePipeline-Permissions-Policy-for-EB` permissions policy to the `Role-for-MyRule` role.

Why am I making this change? Adding this policy to the role creates permissions for EventBridge.

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

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 CloudFormation to create the event.

The following sample command creates a rule called `MyCodeCommitRepoRule`.

```
aws events put-rule --name "MyCodeCommitRepoRule" --event-pattern "{\"source\":[\"aws.codecommit\"],\"detail-type\":[\"CodeCommit Repository State Change\"],\"resources\":[\"repository-ARN\"],\"detail\":{\"referenceType\":[\"branch\"],\"referenceName\":[\"main\"]}}" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-MyRule"
```

3. 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.

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

4. (Optional) To configure an input transformer with source overrides for a specific image ID, use the following JSON in your CLI command. The following example configures an override where:

   - The `actionName`, `Source` in this example, is the dynamic value, defined at pipeline creation, not derived from the source event.
   - The `revisionType`, `COMMIT_ID` in this example, is the dynamic value, defined at pipeline creation, not derived from the source event.
   - The `revisionValue`, <*revisionValue*> in this example, is derived from the source event variable.

```
{
    "Rule": "my-rule",
    "Targets": [
        {
            "Id": "MyTargetId",
            "Arn": "pipeline-ARN",
            "InputTransformer": {
                "sourceRevisions": {
                    "actionName": "Source",
                    "revisionType": "COMMIT_ID",
                    "revisionValue": "<revisionValue>"
                },
                "variables": [
                    {
                        "name": "Branch_Name",
                        "value": "value"
                    }
                ]
            }
        }
    ]
}
```

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 Valid settings for the `PollForSourceChanges` parameter.

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:

   ```
   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.

   ```
   "configuration": {
       "PollForSourceChanges": "false",
       "BranchName": "main",
       "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:

   ```
   "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.

Create an EventBridge rule for a CodeCommit source (CloudFormation template)

To use CloudFormation to create a rule, update your template as shown here.

To update your pipeline CloudFormation template and create EventBridge rule

1. In the template, under `Resources`, use the `AWS::IAM::Role` 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 CloudFormation to create permissions for EventBridge. This resource is added to your CloudFormation stack.

YAML

```
  EventRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref
 'AWS::Region', ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
```

JSON

```
"EventRole": {
  "Type": "AWS::IAM::Role",
  "Properties": {
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
```

```
          "Principal": {
            "Service": [
              "events.amazonaws.com"
            ]
          },
          "Action": "sts:AssumeRole"
        }
      ]
    },
    "Path": "/",
    "Policies": [
      {
        "PolicyName": "eb-pipeline-execution",
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Action": "codepipeline:StartPipelineExecution",
              "Resource": {
                "Fn::Join": [
                  "",
                  [
                    "arn:aws:codepipeline:",
                    {
                      "Ref": "AWS::Region"
                    },
                    ":",
                    {
                      "Ref": "AWS::AccountId"
                    },
                    ":",
                    {
                      "Ref": "AppPipeline"
                    }
                  ]

...
```

2. In the template, under `Resources`, use the `AWS::Events::Rule` CloudFormation resource to add an EventBridge rule. This event pattern creates an event that monitors push changes to your repository. When EventBridge 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 CloudFormation to create the event. This resource is added to your CloudFormation stack.

YAML

```
  EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventPattern:
        source:
          - aws.codecommit
        detail-type:
          - 'CodeCommit Repository State Change'
        resources:
          - !Join [ '', [ 'arn:aws:codecommit:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref RepositoryName ] ]
        detail:
          event:
            - referenceCreated
            - referenceUpdated
          referenceType:
            - branch
          referenceName:
            - main
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline
```

JSON

```
"EventRule": {
  "Type": "AWS::Events::Rule",
  "Properties": {
    "EventPattern": {
      "source": [
        "aws.codecommit"
      ],
```

```
      "detail-type": [
        "CodeCommit Repository State Change"
      ],
      "resources": [
        {
          "Fn::Join": [
            "",
            [
              "arn:aws:codecommit:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
                "Ref": "RepositoryName"
              }
            ]
          ]
        }
      ],
      "detail": {
        "event": [
          "referenceCreated",
          "referenceUpdated"
        ],
        "referenceType": [
          "branch"
        ],
        "referenceName": [
          "main"
        ]
      }
    },
    "Targets": [
      {
        "Arn": {
          "Fn::Join": [
            "",
            [
              "arn:aws:codepipeline:",
```

```
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
                "Ref": "AppPipeline"
              }
            ]
          ]
        },
        "RoleArn": {
          "Fn::GetAtt": [
            "EventRole",
            "Arn"
          ]
        },
        "Id": "codepipeline-AppPipeline"
      }
    ]
  }
},
```

3. (Optional) To configure an input transformer with source overrides for a specific image ID, use the following YAML snippet. The following example configures an override where:

   - The `actionName`, `Source` in this example, is the dynamic value, defined at pipeline creation, not derived from the source event.
   - The `revisionType`, `COMMIT_ID` in this example, is the dynamic value, defined at pipeline creation, not derived from the source event.
   - The `revisionValue`, <*`revisionValue`*> in this example, is derived from the source event variable.
   - The output variables for `BranchName` and `Value` are specified.

```
Rule: my-rule
Targets:
- Id: MyTargetId
```

```
Arn: pipeline-ARN
InputTransformer:
  sourceRevisions:
    actionName: Source
    revisionType: COMMIT_ID
    revisionValue: <revisionValue>
  variables:
  - name: BranchName
    value: value
```

4. Save the updated template to your local computer, and then open the CloudFormation console.
5. Choose your stack, and then choose **Create Change Set for Current Stack**.
6. Upload the template, and then view the changes listed in CloudFormation. These are the changes to 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**
>
> 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 Valid settings for the `PollForSourceChanges` parameter.

- 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

```
        Name: Source
        Actions:
          -
```

```
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: CodeCommit
            OutputArtifacts:
              - Name: SourceOutput
            Configuration:
              BranchName: !Ref BranchName
              RepositoryName: !Ref RepositoryName
              PollForSourceChanges: false
            RunOrder: 1
```

JSON

```
{
  "Name": "Source",
  "Actions": [
    {
      "Name": "SourceAction",
      "ActionTypeId": {
        "Category": "Source",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "CodeCommit"
      },
      "OutputArtifacts": [
        {
          "Name": "SourceOutput"
        }
      ],
      "Configuration": {
        "BranchName": {
          "Ref": "BranchName"
        },
        "RepositoryName": {
          "Ref": "RepositoryName"
        },
        "PollForSourceChanges": false
      },
      "RunOrder": 1
    }
```

```
    ]
},
```

Add third-party source providers to pipelines using AWS CodeConnections

You can use the AWS CodePipeline console or the AWS CLI to connect your pipeline to third-party repositories.

> **Note**
>
> When you use the console to create or edit a pipeline, the change detection 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, see CodeCommit source actions and EventBridge.

Topics

- Azure DevOps connections
- Bitbucket Cloud connections
- GitHub connections
- GitHub Enterprise Server connections
- GitLab.com connections
- Connections for GitLab self-managed
- Use a connection shared with another account

Azure DevOps connections

Connections allow you to authorize and establish configurations that associate your third-party provider with your AWS resources. To associate your third-party repository as a source for your pipeline, you use a connection.

> **Note**
>
> Instead of creating or using an existing connection in your account, you can use a shared connection between another AWS account. See Use a connection shared with another account.

To add an Azure DevOps source action in CodePipeline, you can choose either to:

- Use the CodePipeline console **Create pipeline** wizard or **Edit action** page to choose the **Azure DevOps** provider option. See Create a connection to Bitbucket Cloud (console) to add the action. The console helps you create a connections resource.
- Use the CLI to add the action configuration for the `CreateSourceConnection` action with the `Azure DevOps` provider as follows:
  - To create your connections resources, see Create a connection to Azure DevOps (CLI) to create a connections resource with the CLI.
  - Use the `CreateSourceConnection` example action configuration in CodeStarSourceConnection for Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed actions to add your action as shown in Create a pipeline (CLI).

> **Note**
>
> You can also create a connection using the Developer Tools console under **Settings**. See Create a Connection.

Before you begin:

- You must have created an account with the provider of the third-party repository, such as Azure DevOps
- You must have already created a third-party code repository, such as an Azure DevOps repository.

> **Note**
>
> Azure DevOps connections only provide access to repositories owned by the Azure DevOps account that was used to create the connection.
> To install the connection, you must have **Administer** permissions for the Azure account.

Topics

- Create a connection to Azure DevOps (console)

- Create a connection to Azure DevOps (CLI)

Create a connection to Azure DevOps (console)

Use these steps to use the CodePipeline console to add a connections action for your Azure DevOps repository.

Step 1: Create or edit your pipeline

To create or edit your pipeline

1. Sign in to the CodePipeline console.
2. Choose one of the following.
   - Choose to create a pipeline. Follow the steps in *Create a Pipeline* to complete the first screen and choose **Next**. On the **Source** page, under **Source Provider**, choose **Azure DevOps**.
   - Choose to edit an existing pipeline. Choose **Edit**, and then choose **Edit stage**. Choose to add or edit your source action. On the **Edit action** page, under **Action name**, enter the name for your action. In **Action provider**, choose **Azure DevOps**.
3. Do one of the following:
   - Under **Connection**, if you have not already created a connection to your provider, choose **Connect to Azure DevOps**. Proceed to Step 2: Create a Connection to Azure DevOps.
   - Under **Connection**, if you have already created a connection to your provider, choose the connection. Proceed to Step 3: Save the Source Action for Your Connection.

Step 2: Create a connection to Azure DevOps

To create a connection to Azure DevOps

1. To create a connection to an Azure DevOps repository, under **Select a provider**, choose **Azure DevOps**. In **Connection name**, enter the name for the connection that you want to create. Choose **Connect to Azure DevOps**, and proceed to Step 2.

Add source stage Info

Step 3 of 7

Source

Source provider

This is where you stored your input artifacts for your pipeline. Choose the provider and then provide the connection details.

Azure DevOps

Connection

Choose an existing connection that you have already configured, or create a new one and then return to this task.

or Connect to AzureDevOps

Owner

Choose an owner in your Azure DevOps account.

Repository name

Choose a repository in your Azure DevOps account.

You can type or paste the group path to any project that the provided credentials can access. Use the format 'group/subgroup/project'.

Default branch

Default branch will be used only when pipeline execution starts from a different source or manually started.

Output artifact format

Choose the output artifact format.

- CodePipeline default
  AWS CodePipeline uses the default zip format for artifacts in the pipeline. Does not include Git metadata about the repository.
- Full clone
  AWS CodePipeline passes metadata about the repository that allows subsequent actions to do a full Git clone. Only supported for AWS CodeBuild actions. Learn more

Enable automatic retry on stage failure

Webhook events

Webhook - *optional*

Start your pipeline on push and pull request events.

Webhook event filters - *optional*

Starts your pipeline on a specific event.

Remove filters

Cancel Previous Next

2. On the **Connect to Azure DevOps** settings page, your connection name displays.

   Under **Azure DevOps apps**, choose an app installation or choose **Install a new app** to create one.

   > **Note**
   >
   > You only install the app once for each Azure DevOps account. If you have already installed the connector app, choose it and move to the last step in this section.

3. If the login page for Microsoft displays, log in with your credentials and then choose to continue.
4. On the app installation page, a message shows that the connector app is trying to connect to your Azure DevOps account.

   Choose **Grant access**.
5. On the connection page, the connection ID for your new installation is displayed. Choose **Connect**. The created connection displays in the connections list.

Step 3: Save your Azure DevOps source action

Use these steps on the wizard or **Edit action** page to save your source action with your connection information.

To complete and save your source action with your connection

1. In **Repository name**, choose the name of your third-party repository.
2. Under **Pipeline triggers** you can add triggers if your action is a AWS CodeConnections action. To configure the pipeline trigger configuration and to optionally filter with triggers, see more details in Add trigger with code push or pull request event types.
3. In **Output artifact format**, you must choose the format for your artifacts.

   - To store output artifacts from the Azure DevOps action using the default method, choose **CodePipeline default**. The action accesses the files from the 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.

If you choose this option, you will need to update the permissions for your CodeBuild project service role as shown in Add CodeBuild GitClone permissions for connections to Bitbucket, GitHub, GitHub Enterprise Server, or GitLab.com.

4. Choose **Next** on the wizard or **Save** on the **Edit action** page.

Create a connection to Azure DevOps (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 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 `AzureDevOps` and the specified connection name is `MyConnection`.

```
aws codeconnections create-connection --provider-type AzureDevOps --connection-name
 MyConnection
```

If successful, this command returns the connection ARN information similar to the following.

```
{
    "ConnectionArn": "arn:aws:codeconnections: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.

3. The pipeline defaults to detect changes on code push to the connection source repository. To configure the pipeline trigger configuration for manual release or for Git tags, do one of the following:

   - To configure the pipeline trigger configuration to start with a manual release only, add the following line to the configuration:

     ```
     "DetectChanges": "false",
     ```

   - To configure the pipeline trigger configuration to filter with triggers, see more details in Add trigger with code push or pull request event types. For example, the following adds Git tags to the pipeline level of the pipeline JSON definition. In this example, `release-v0` and `release-v1` are the Git tags to include, and `release-v2` is the Git tag to exclude.

     ```
     "triggers": [
                 {
                     "providerType": "CodeStarSourceConnection",
                     "gitConfiguration": {
                         "sourceActionName": "Source",
                         "push": [
                             {
                                 "tags": {
                                     "includes": [
                                         "release-v0", "release-v1"
                                     ],
                                     "excludes": [
                                         "release-v2"
                                     ]
                                 }
                             }
                         ]
                     }
                 }
             ]
     ```

Bitbucket Cloud connections

Connections allow you to authorize and establish configurations that associate your third-party provider with your AWS resources. To associate your third-party repository as a source for your pipeline, you use a connection.

> **Note**
>
> Instead of creating or using an existing connection in your account, you can use a shared connection between another AWS account. See Use a connection shared with another account.

To add a Bitbucket Cloud source action in CodePipeline, you can choose either to:

- Use the CodePipeline console **Create pipeline** wizard or **Edit action** page to choose the **Bitbucket** provider option. See Create a connection to Bitbucket Cloud (console) to add the action. The console helps you create a connections resource.

  > **Note**
  >
  > You can create connections to a Bitbucket Cloud repository. Installed Bitbucket provider types, such as Bitbucket Server, are not supported.

- Use the CLI to add the action configuration for the `CreateSourceConnection` action with the `Bitbucket` provider as follows:
  - To create your connections resources, see Create a connection to Bitbucket Cloud (CLI) to create a connections resource with the CLI.
  - Use the `CreateSourceConnection` example action configuration in CodeStarSourceConnection for Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed actions to add your action as shown in Create a pipeline (CLI).

> **Note**
>
> You can also create a connection using the Developer Tools console under **Settings**. See Create a Connection.

Before you begin:

- You must have created an account with the provider of the third-party repository, such as Bitbucket Cloud.

- You must have already created a third-party code repository, such as a Bitbucket Cloud repository.

> **Note**
>
> Bitbucket Cloud connections only provide access to repositories owned by the Bitbucket Cloud account that was used to create the connection.
> If the application is being installed in a Bitbucket Cloud workspace, you need **Administer workspace** permissions. Otherwise, the option to install the app will not display.

Topics

- Create a connection to Bitbucket Cloud (console)
- Create a connection to Bitbucket Cloud (CLI)

Create a connection to Bitbucket Cloud (console)

Use these steps to use the CodePipeline console to add a connections action for your Bitbucket repository.

> **Note**
>
> You can create connections to a Bitbucket Cloud repository. Installed Bitbucket provider types, such as Bitbucket Server, are not supported.

Step 1: Create or edit your pipeline

To create or edit your pipeline

1. Sign in to the CodePipeline console.
2. Choose one of the following.
   - Choose to create a pipeline. Follow the steps in *Create a Pipeline* to complete the first screen and choose **Next**. On the **Source** page, under **Source Provider**, choose **Bitbucket**.

- Choose to edit an existing pipeline. Choose **Edit**, and then choose **Edit stage**. Choose to add or edit your source action. On the **Edit action** page, under **Action name**, enter the name for your action. In **Action provider**, choose **Bitbucket**.

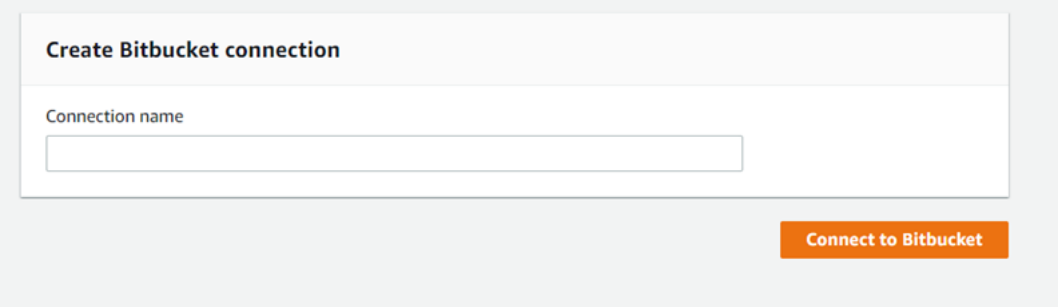
3. Do one of the following:

- Under **Connection**, if you have not already created a connection to your provider, choose **Connect to Bitbucket**. Proceed to Step 2: Create a Connection to Bitbucket.
- Under **Connection**, if you have already created a connection to your provider, choose the connection. Proceed to Step 3: Save the Source Action for Your Connection.

Step 2: Create a connection to Bitbucket Cloud

To create a connection to Bitbucket Cloud

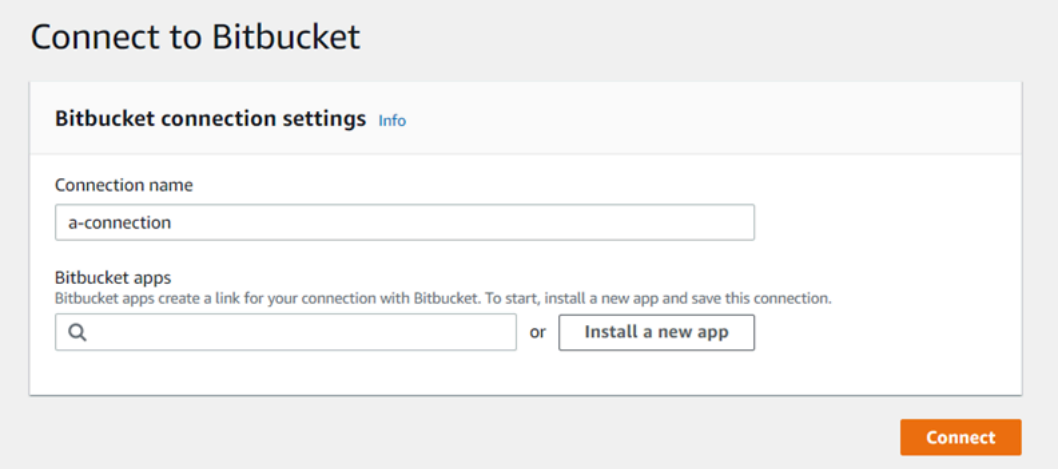
1. On the **Connect to Bitbucket** settings page, enter your connection name and choose **Connect to Bitbucket**.



   The **Bitbucket apps** field appears.

2. Under **Bitbucket apps**, choose an app installation or choose **Install a new app** to create one.

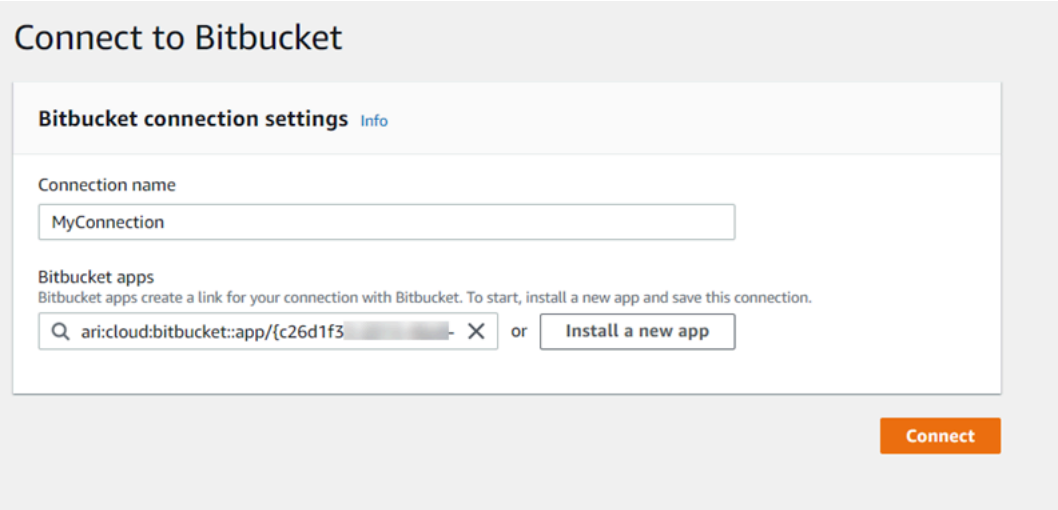
> **Note**
>
> You only install the app once for each Bitbucket Cloud workspace or account. If you have already installed the Bitbucket app, choose it and move to step 4.



3. If the login page for Bitbucket Cloud displays, log in with your credentials and then choose to continue.
4. On the app installation page, a message shows that the AWS CodeStar app is trying to connect to your Bitbucket account.

   If you are using a Bitbucket workspace, change the **Authorize for** option to the workspace. Only workspaces where you have administrator access will display.

   Choose **Grant access**.
5. In **Bitbucket apps**, the connection ID for your new installation is displayed. Choose **Connect**. The created connection displays in the connections list.



Step 3: Save your Bitbucket Cloud source action

Use these steps on the wizard or **Edit action** page to save your source action with your connection information.

To complete and save your source action with your connection

1. In **Repository name**, choose the name of your third-party repository.
2. Under **Pipeline triggers** you can add triggers if your action is a AWS CodeConnections action. To configure the pipeline trigger configuration and to optionally filter with triggers, see more details in Add trigger with code push or pull request event types.
3. In **Output artifact format**, you must choose the format for your artifacts.
   - To store output artifacts from the Bitbucket Cloud action using the default method, choose **CodePipeline default**. The action accesses the files from the Bitbucket Cloud 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.

     If you choose this option, you will need to update the permissions for your CodeBuild project service role as shown in Add CodeBuild GitClone permissions for connections to Bitbucket, GitHub, GitHub Enterprise Server, or GitLab.com.
4. Choose **Next** on the wizard or **Save** on the **Edit action** page.

Create a connection to Bitbucket Cloud (CLI)

You can use the AWS Command Line Interface (AWS CLI) to create a connection.

> **Note**
>
> You can create connections to a Bitbucket Cloud repository. Installed Bitbucket provider types, such as Bitbucket Server, are not supported.

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 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`.

   ```
   aws codestar-connections create-connection --provider-type Bitbucket --connection-name MyConnection
   ```

   If successful, this command returns the connection ARN information similar to the following.

   ```
   {
       "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.
3. The pipeline defaults to detect changes on code push to the connection source repository. To configure the pipeline trigger configuration for manual release or for Git tags, do one of the following:

   - To configure the pipeline trigger configuration to start with a manual release only, add the following line to the configuration:

     ```
     "DetectChanges": "false",
     ```

   - To configure the pipeline trigger configuration to filter with triggers, see more details in Add trigger with code push or pull request event types. For example, the following adds Git tags to the pipeline level of the pipeline JSON definition. In this example, `release-v0` and `release-v1` are the Git tags to include, and `release-v2` is the Git tag to exclude.

```
"triggers": [
            {
                "providerType": "CodeStarSourceConnection",
                "gitConfiguration": {
                    "sourceActionName": "Source",
                    "push": [
                        {
                            "tags": {
                                "includes": [
                                    "release-v0", "release-v1"
                                ],
                                "excludes": [
                                    "release-v2"
                                ]
                            }
                        }
                    ]
                }
            }
        ]
```

GitHub connections

You use connections to authorize and establish configurations that associate your third-party provider with your AWS resources.

> **Note**
>
> Instead of creating or using an existing connection in your account, you can use a shared connection between another AWS account. See Use a connection shared with another account.

To add a source action for your GitHub or GitHub Enterprise Cloud repository in CodePipeline, you can choose either to:

- Use the CodePipeline console **Create pipeline** wizard or **Edit action** page to choose the **GitHub (via GitHub App)** provider option. See Create a connection to GitHub (console) to add the action. The console helps you create a connections resource.

> **Note**
>
> For a tutorial that walks you through how to add a GitHub connection and use the **Full clone** option in your pipeline to clone metadata, see Tutorial: Use full clone with a GitHub pipeline source.

- Use the CLI to add the action configuration for the `CodeStarSourceConnection` action with the `GitHub` provider with the CLI steps shown in Create a pipeline (CLI).

> **Note**
>
> You can also create a connection using the Developer Tools console under **Settings**. See Create a Connection.

Before you begin:

- You must have created an account with GitHub.
- You must have already created a GitHub code repository.
- If your CodePipeline service role was created before December 18, 2019, you might need to update its permissions to use `codestar-connections:UseConnection` for AWS CodeStar connections. For instructions, see Add permissions to the CodePipeline service role.

> **Note**
>
> To create the connection, you must be the GitHub organization owner. For repositories that are not under an organization, you must be the repository owner.

Topics

- Create a connection to GitHub (console)
- Create a connection to GitHub (CLI)

Create a connection to GitHub (console)

Use these steps to use the CodePipeline console to add a connections action for your GitHub or GitHub Enterprise Cloud repository.

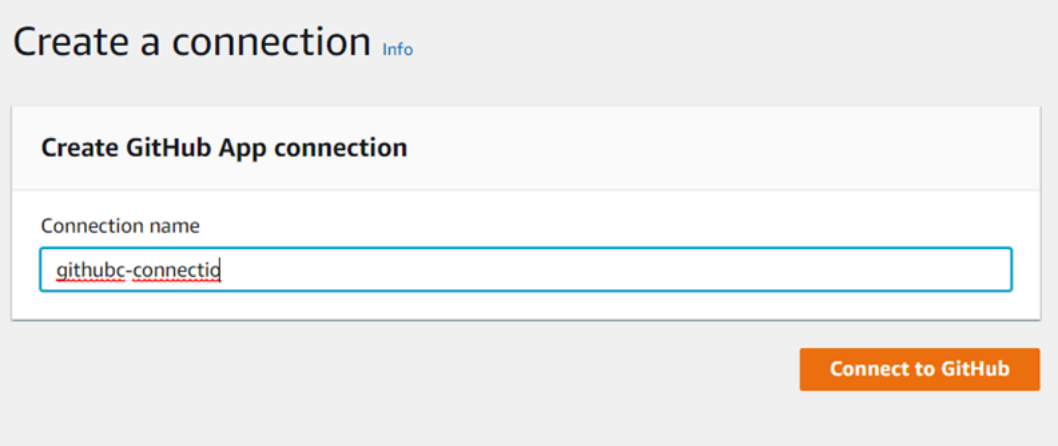
> **Note**
>
> In these steps, you can select specific repositories under **Repository Access**. Any repositories that are not selected will not be accessible or visible by CodePipeline.

Step 1: Create or edit your pipeline

1. Sign in to the CodePipeline console.
2. Choose one of the following.
   - Choose to create a pipeline. Follow the steps in *Create a Pipeline* to complete the first screen and choose **Next**. On the **Source** page, under **Source Provider**, choose **GitHub (via GitHub App)**.
   - Choose to edit an existing pipeline. Choose **Edit**, and then choose **Edit stage**. Choose to add or edit your source action. On the **Edit action** page, under **Action name**, enter the name for your action. In **Action provider**, choose **GitHub (via GitHub App)**.
3. Do one of the following:
   - Under **Connection**, if you have not already created a connection to your provider, choose **Connect to GitHub**. Proceed to Step 2: Create a Connection to GitHub.
   - Under **Connection**, if you have already created a connection to your provider, choose the connection. Proceed to Step 3: Save the source action for your connection.

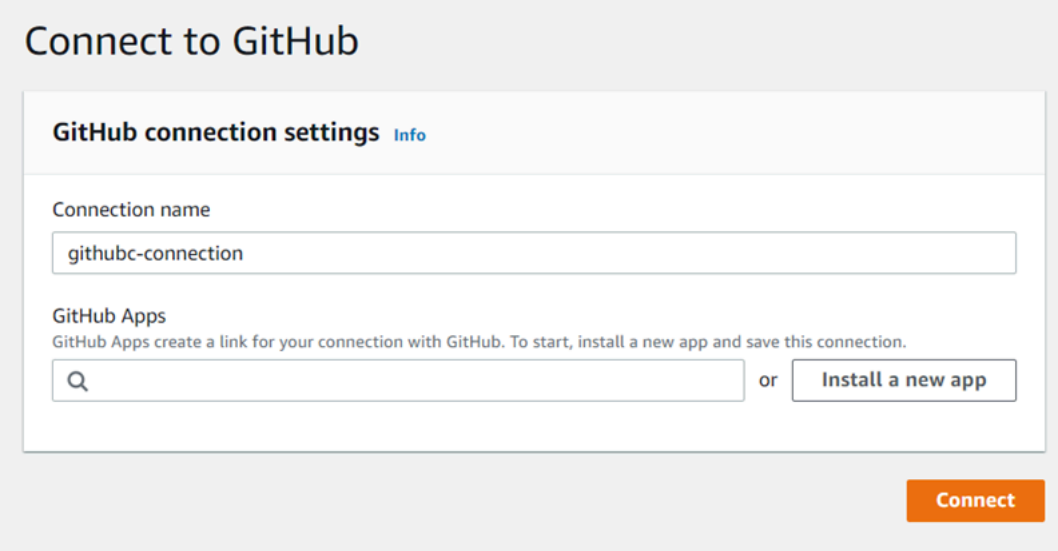
Step 2: Create a connection to GitHub

After you choose to create the connection, the **Connect to GitHub** page appears.



To create a connection to GitHub

1. Under **GitHub connection settings**, your connection name appears in **Connection name**. Choose **Connect to GitHub**. The access request page appears.
2. Choose **Authorize AWS Connector for GitHub**. The connection page displays and shows the **GitHub Apps** field.



3. Under **GitHub Apps**, choose an app installation or choose **Install a new app** to create one.

   You install one app for all of your connections to a particular provider. If you have already installed the AWS Connector for GitHub app, choose it and skip this step.

> **Note**
>
> If you want to create a user access token, make sure that you've already installed the AWS Connector for GitHub app and then leave the App installation field empty. AWS CodeConnections will use the user access token for the connection.

4. On the **Install AWS Connector for GitHub** page, choose the account where you want to install the app.

> **Note**
>
> You only install the app once for each GitHub account. If you previously installed the app, you can choose **Configure** to proceed to a modification page for your app installation, or you can use the back button to return to the console.

5. On the **Install AWS Connector for GitHub** page, leave the defaults, and choose **Install**.
6. On the **Connect to GitHub** page, the connection ID for your new installation appears in **GitHub Apps**. Choose **Connect**.

Step 3: Save your GitHub source action

Use these steps on the **Edit action** page to save your source action with your connection information.

To save your GitHub source action

1. In **Repository name**, choose the name of your third-party repository.
2. Under **Pipeline triggers** you can add triggers if your action is a AWS CodeConnections action. To configure the pipeline trigger configuration and to optionally filter with triggers, see more details in Add trigger with code push or pull request event types.
3. In **Output artifact format**, you must choose the format for your artifacts.
   - To store output artifacts from the GitHub action using the default method, choose **CodePipeline default**. The action accesses the files from the GitHub 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.

  If you choose this option, you will need to update the permissions for your CodeBuild project service role as shown in Add CodeBuild GitClone permissions for connections to Bitbucket, GitHub, GitHub Enterprise Server, or GitLab.com. For a tutorial that shows you how to use the **Full clone** option, see Tutorial: Use full clone with a GitHub pipeline source.

4. Choose **Next** on the wizard or **Save** on the **Edit action** page.

Create a connection to GitHub (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 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 `GitHub` and the specified connection name is `MyConnection`.

   ```
   aws codestar-connections create-connection --provider-type GitHub --connection-name
    MyConnection
   ```

   If successful, this command returns the connection ARN information similar to the following.

   ```
   {
       "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.
3. The pipeline defaults to detect changes on code push to the connection source repository. To configure the pipeline trigger configuration for manual release or for Git tags, do one of the following:

   - To configure the pipeline trigger configuration to start with a manual release only, add the following line to the configuration:

     ```
     "DetectChanges": "false",
     ```

   - To configure the pipeline trigger configuration to filter with triggers, see more details in Add trigger with code push or pull request event types. For example, the following adds to the pipeline level of the pipeline JSON definition. In this example, `release-v0` and `release-v1` are the Git tags to include, and `release-v2` is the Git tag to exclude.

     ```
     "triggers": [
                 {
                     "providerType": "CodeStarSourceConnection",
                     "gitConfiguration": {
                         "sourceActionName": "Source",
                         "push": [
                             {
                                 "tags": {
                                     "includes": [
                                         "release-v0", "release-v1"
                                     ],
                                     "excludes": [
                                         "release-v2"
                                     ]
                                 }
                             }
                         ]
                     }
                 }
             ]
     ```

GitHub Enterprise Server connections

Connections allow you to authorize and establish configurations that associate your third-party provider with your AWS resources. To associate your third-party repository as a source for your pipeline, you use a connection.

> **Note**
>
> Instead of creating or using an existing connection in your account, you can use a shared connection between another AWS account. See Use a connection shared with another account.

To add a GitHub Enterprise Server source action in CodePipeline, you can choose either to:

- Use the CodePipeline console **Create pipeline** wizard or **Edit action** page to choose the **GitHub Enterprise Server** provider option. See Create a connection to GitHub Enterprise Server (console) to add the action. The console helps you create a host resource and a connections resource.
- Use the CLI to add the action configuration for the `CreateSourceConnection` action with the `GitHubEnterpriseServer` provider and create your resources:
  - To create your connections resources, see Create a host and connection to GitHub Enterprise Server (CLI) to create a host resource and a connections resource with the CLI.
  - Use the `CreateSourceConnection` example action configuration in CodeStarSourceConnection for Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed actions to add your action as shown in Create a pipeline (CLI).

> **Note**
>
> You can also create a connection using the Developer Tools console under **Settings**. See Create a Connection.

Before you begin:

- You must have created an account with GitHub Enterprise Server and installed the GitHub Enterprise Server instance on your infrastructure.

> **Note**
>
> Each VPC can only be associated with one host (GitHub Enterprise Server instance) at a time.

- You must have already created a code repository with GitHub Enterprise Server.

Topics

- Create a connection to GitHub Enterprise Server (console)
- Create a host and connection to GitHub Enterprise Server (CLI)

Create a connection to GitHub Enterprise Server (console)

Use these steps to use the CodePipeline console to add a connections action for your GitHub Enterprise Server repository.

> **Note**
>
> GitHub Enterprise Server connections only provide access to repositories owned by the GitHub Enterprise Server account that was used to create the connection.

Before you begin:

For a host connection to GitHub Enterprise Server, you must have completed the steps to create a host resource for your connection. See Manage hosts for connections.

Step 1: Create or edit your pipeline

To create or edit your pipeline

1. Sign in to the CodePipeline console.
2. Choose one of the following.

   - Choose to create a pipeline. Follow the steps in *Create a Pipeline* to complete the first screen and choose **Next**. On the **Source** page, under **Source provider**, choose **GitHub Enterprise Server**.

- Choose to edit an existing pipeline. Choose **Edit**, and then choose **Edit stage**. Choose to add or edit your source action. On the **Edit action** page, under **Action name**, enter the name for your action. In **Action provider**, choose **GitHub Enterprise Server**.

3. Do one of the following:

- Under **Connection**, if you have not already created a connection to your provider, choose **Connect to GitHub Enterprise Server**. Proceed to Step 2: Create a Connection to GitHub Enterprise Server.
- Under **Connection**, if you have already created a connection to your provider, choose the connection. Proceed to Step 3: Save the Source Action for Your Connection.

Create a connection to GitHub Enterprise Server

After you choose to create the connection, the **Connect to GitHub Enterprise Server** page is shown.

> **Important**
>
> AWS CodeConnections does not support GitHub Enterprise Server version 2.22.0 due to a known issue in the release. To connect, upgrade to version 2.22.1 or the latest available version.

To connect to GitHub Enterprise Server

1. In **Connection name**, enter the name for your connection.
2. In **URL**, enter the endpoint for your server.

> **Note**
>
> If the provided URL has already been used to set up a GitHub Enterprise Server for a connection, you will be prompted to choose the host resource ARN that was created previously for that endpoint.

3. If you have launched your server into an Amazon VPC and you want to connect with your VPC, choose **Use a VPC** and complete the following.

a. In **VPC ID**, choose your VPC ID. Make sure to choose the VPC for the infrastructure where your GitHub Enterprise Server instance is installed or a VPC with access to your GitHub Enterprise Server instance through VPN or Direct Connect.

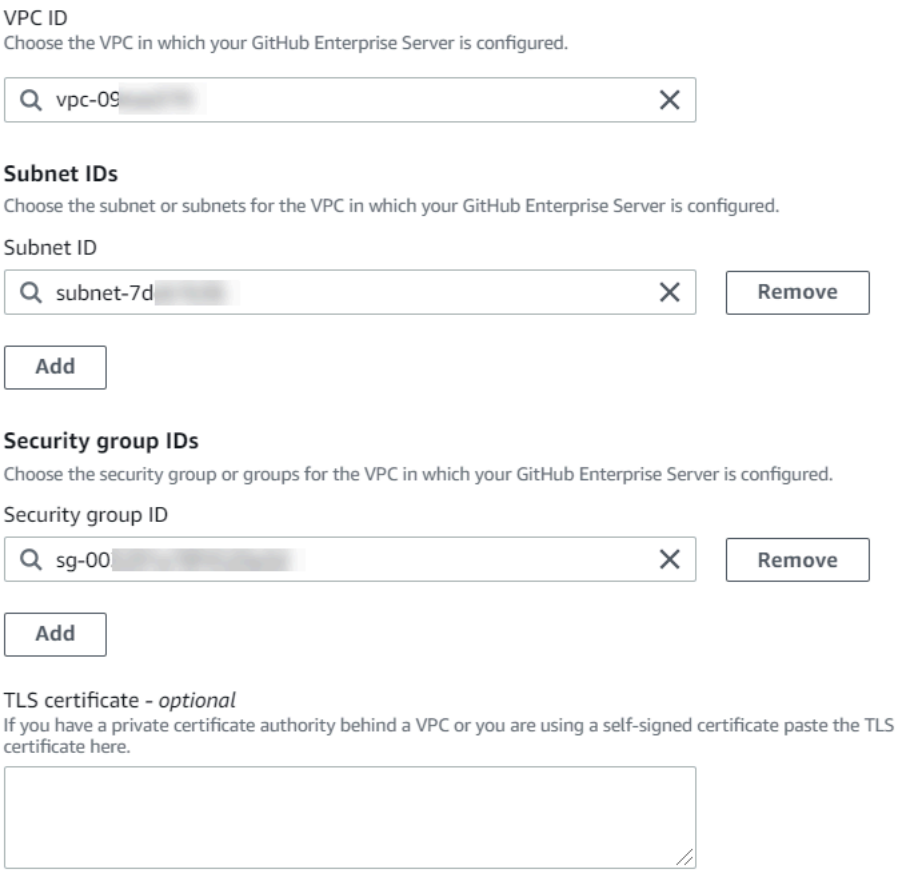
b. Under **Subnet ID**, choose **Add**. In the field, choose the subnet ID you want to use for your host. You can choose up to 10 subnets.

   Make sure to choose the subnet for the infrastructure where your GitHub Enterprise Server instance is installed or a subnet with access to your installed GitHub Enterprise Server instance through VPN or Direct Connect.

c. Under **Security group IDs**, choose **Add**. In the field, choose the security group you want to use for your host. You can choose up to 10 security groups.

   Make sure to choose the security group for the infrastructure where your GitHub Enterprise Server instance is installed or a security group with access to your installed GitHub Enterprise Server instance through VPN or Direct Connect.

d. If you have a private VPC configured, and you have configured your GitHub Enterprise Server instance to perform TLS validation using a non-public certificate authority, in **TLS certificate**, enter your certificate ID. The TLS Certificate value should be the public key of the certificate.



4. Choose **Connect to GitHub Enterprise Server**. The created connection is shown with a **Pending** status. A host resource is created for the connection with the server information you provided. For the host name, the URL is used.
5. Choose **Update pending connection.**
6. If prompted, on the GitHub Enterprise login page, sign in with your GitHub Enterprise credentials.
7. On the **Create GitHub App** page, choose a name for your app.
8. On the GitHub authorization page, choose **Authorize <app-name>**.
9. On the app installation page, a message shows that the connector app is ready to be installed. If you have multiple organizations, you might be prompted to choose the organization where you want to install the app.

   Choose the repository settings where you want to install the app. Choose **Install**.
10. The connection page shows the created connection in an **Available** status.

Step 3: Save your GitHub Enterprise Server source action

Use these steps on the wizard or **Edit action** page to save your source action with your connection information.

To complete and save your source action with your connection

1. In **Repository name**, choose the name of your third-party repository.
2. Under **Pipeline triggers** you can add triggers if your action is a AWS CodeConnections action. To configure the pipeline trigger configuration and to optionally filter with triggers, see more details in Add trigger with code push or pull request event types.
3. In **Output artifact format**, you must choose the format for your artifacts.
   - To store output artifacts from the GitHub Enterprise Server action using the default method, choose **CodePipeline default**. The action accesses the files from the GitHub Enterprise Server 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.
4. Choose **Next** on the wizard or **Save** on the **Edit action** page.

Create a host and connection to GitHub Enterprise Server (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 CloudFormation, use the console to edit the connection to make its status `AVAILABLE`.

You can use the AWS Command Line Interface (AWS CLI) to create a host for installed connections.

> **Note**
>
> You only create a host once per GitHub Enterprise Server account. All of your connections to a specific GitHub Enterprise Server account will use the same host.

You use a host to represent the endpoint for the infrastructure where your third-party provider is installed. After you complete the host creation with the CLI, the host is in **Pending** status. You then set up, or register, the host to move it to an **Available** status. After the host is available, you complete the steps to create a connection.

To do this, use the **create-host** command.

> **Important**
>
> A host created through the AWS CLI is in `Pending` status by default. After you create a host with the CLI, use the console or the CLI to set up the host to make its status `Available`.

To create a host

1. Open a terminal (Linux, macOS, or Unix) or command prompt (Windows). Use the AWS CLI to run the **create-host** command, specifying the `--name`, `--provider-type`, and `--provider-endpoint` for your connection. In this example, the third-party provider name is `GitHubEnterpriseServer` and the endpoint is `my-instance.dev`.

   ```
   aws codestar-connections create-host --name MyHost --provider-type
    GitHubEnterpriseServer --provider-endpoint "https://my-instance.dev"
   ```

   If successful, this command returns the host Amazon Resource Name (ARN) information similar to the following.

   ```
   {
       "HostArn": "arn:aws:codestar-connections:us-west-2:account_id:host/My-
   Host-28aef605"
   }
   ```

   After this step, the host is in `PENDING` status.

2. Use the console to complete the host setup and move the host to an `Available` status.

To create a connection to GitHub Enterprise Server

1. Open a terminal (Linux, macOS, or Unix) or command prompt (Windows). Use the AWS CLI to run the **create-connection** command, specifying the `--host-arn` and `--connection-name` for your connection.

```
aws codestar-connections create-connection --host-arn arn:aws:codestar-
connections:us-west-2:account_id:host/MyHost-234EXAMPLE --connection-name
 MyConnection
```

   If successful, this command returns the connection ARN information similar to the following.

```
{
    "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/
aEXAMPLE-8aad"
}
```

2. Use the console to set up the pending connection.
3. The pipeline defaults to detect changes on code push to the connection source repository. To configure the pipeline trigger configuration for manual release or for Git tags, do one of the following:

   - To configure the pipeline trigger configuration to start with a manual release only, add the following line to the configuration:

```
"DetectChanges": "false",
```

   - To configure the pipeline trigger configuration to filter with triggers, see more details in Add trigger with code push or pull request event types. For example, the following adds to the pipeline level of the pipeline JSON definition. In this example, `release-v0` and `release-v1` are the Git tags to include, and `release-v2` is the Git tag to exclude.

```
"triggers": [
            {
                "providerType": "CodeStarSourceConnection",
                "gitConfiguration": {
                    "sourceActionName": "Source",
```

```
                "push": [
                    {
                        "tags": {
                            "includes": [
                                "release-v0", "release-v1"
                            ],
                            "excludes": [
                                "release-v2"
                            ]
                        }
                    }
                ]
            }
        }
    ]
```

GitLab.com connections

Connections allow you to authorize and establish configurations that associate your third-party provider with your AWS resources. To associate your third-party repository as a source for your pipeline, you use a connection.

> **Note**
>
> Instead of creating or using an existing connection in your account, you can use a shared connection between another AWS account. See Use a connection shared with another account.

To add a GitLab.com source action in CodePipeline, you can choose either to:

- Use the CodePipeline console **Create pipeline** wizard or **Edit action** page to choose the **GitLab** provider option. See Create a connection to GitLab.com (console) to add the action. The console helps you create a connections resource.
- Use the CLI to add the action configuration for the `CreateSourceConnection` action with the `GitLab` provider as follows:
  - To create your connections resources, see Create a connection to GitLab.com (CLI) to create a connections resource with the CLI.

- Use the `CreateSourceConnection` example action configuration in CodeStarSourceConnection for Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed actions to add your action as shown in Create a pipeline (CLI).

> **Note**
>
> You can also create a connection using the Developer Tools console under **Settings**. See Create a Connection.

> **Note**
>
> By authorizing this connection installation in GitLab.com, you grant our service permissions to process your data by accessing your account, and you can revoke the permissions at any time by uninstalling the application.

Before you begin:

- You must have already created an account with GitLab.com.

  > **Note**
  >
  > Connections only provide access to repositories owned by the account that was used to create and authorize the connection.

  > **Note**
  >
  > You can create connections to a repository where you have the **Owner** role in GitLab, and then the connection can be used with the repository with resources such as CodePipeline. For repositories in groups, you do not need to be the group owner.

- To specify a source for your pipeline, you must have already created a repository on gitlab.com.

Topics

- Create a connection to GitLab.com (console)
- Create a connection to GitLab.com (CLI)

Create a connection to GitLab.com (console)

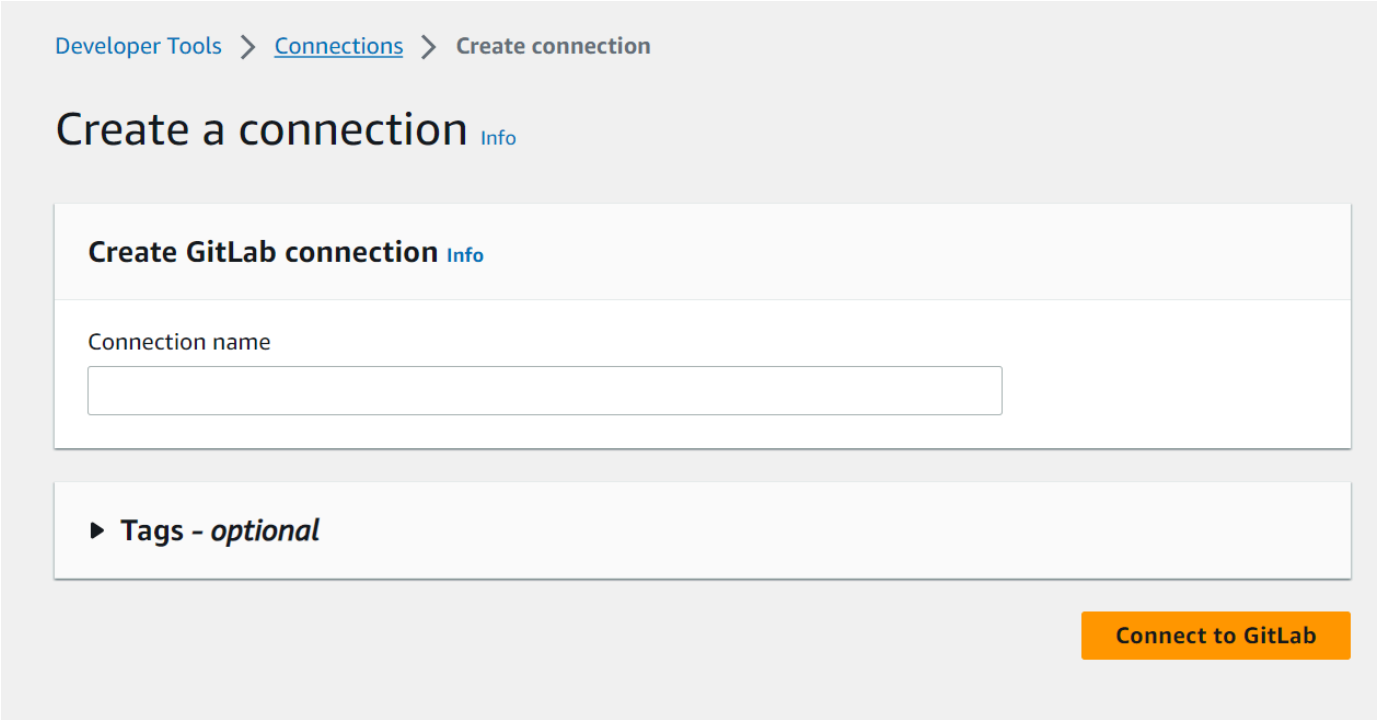
Use these steps to use the CodePipeline console to add a connections action for your project (repository) in GitLab.

To create or edit your pipeline

1. Sign in to the CodePipeline console.
2. Choose one of the following.
   - Choose to create a pipeline. Follow the steps in *Create a Pipeline* to complete the first screen and choose **Next**. On the **Source** page, under **Source Provider**, choose **GitLab**.
   - Choose to edit an existing pipeline. Choose **Edit**, and then choose **Edit stage**. Choose to add or edit your source action. On the **Edit action** page, under **Action name**, enter the name for your action. In **Action provider**, choose **GitLab**.
3. Do one of the following:
   - Under **Connection**, if you have not already created a connection to your provider, choose **Connect to GitLab**. Proceed to step 4 to create the connection.
   - Under **Connection**, if you have already created a connection to your provider, choose the connection. Proceed to step 9.

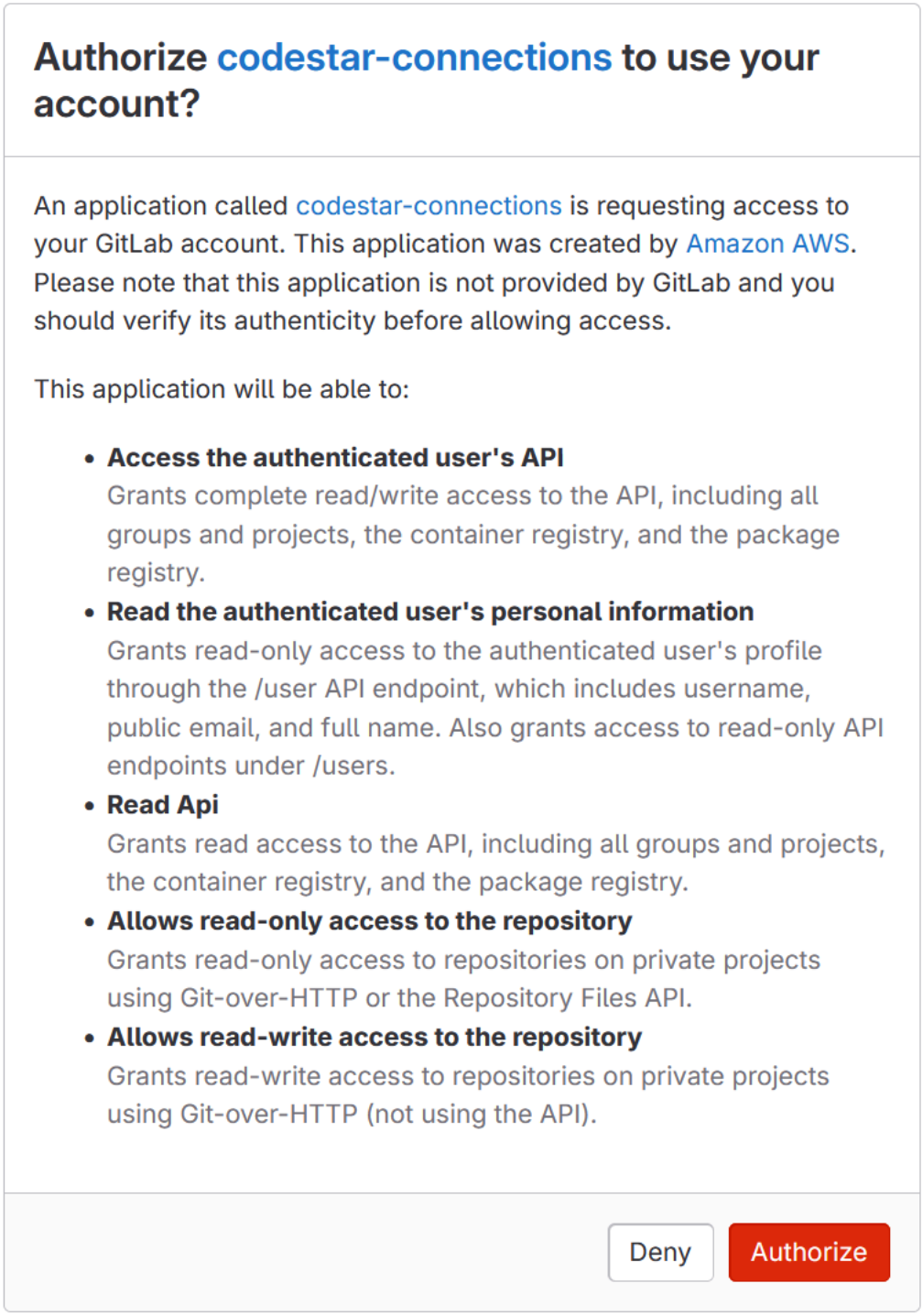
   > **Note**
   >
   > If you close the pop-up window before a GitLab.com connection is created, you need to refresh the page.

4. To create a connection to a GitLab.com repository, under **Select a provider**, choose **GitLab**. In **Connection name**, enter the name for the connection that you want to create. Choose **Connect to GitLab**.



5. When the sign-in page for GitLab.com displays, log in with your credentials, and then choose **Sign in**.
6. If this is your first time authorizing the connection, an authorization page displays with a message requesting authorization for the connection to access your GitLab.com account.

   Choose **Authorize**.



7. The browser returns to the connections console page. Under **Create GitLab connection**, the new connection is shown in **Connection name**.
8. Choose **Connect to GitLab**.

   You will be returned to the CodePipeline console.

> **Note**
>
> After a GitLab.com connection is successfully created, a success banner will be displayed on the main window.
> If you have not previously logged in to GitLab on the current machine, you will need to manually close the pop-up window.

9. In **Repository name**, choose the name of your project in GitLab by specifying the project path with the namespace. For example, for a group-level repository, enter the repository name in the following format: `group-name/repository-name`. For more information about the path and namespace, see the `path_with_namespace` field in https://docs.gitlab.com/ee/api/projects.html#get-single-project. For more information about the namespace in GitLab, see https://docs.gitlab.com/ee/user/namespace/.

> **Note**
>
> For groups in GitLab, you must manually specify the project path with the namespace. For example, for a repository named `myrepo` in a group `mygroup`, enter the following: `mygroup/myrepo`. You can find the project path with the namespace in the URL in GitLab.

10. Under **Pipeline triggers** you can add triggers if your action is a AWS CodeConnections action. To configure the pipeline trigger configuration and to optionally filter with triggers, see more details in Add trigger with code push or pull request event types.
11. In **Branch name**, choose the branch where you want your pipeline to detect source changes.

> **Note**
>
> If the branch name does not populate automatically, then you do not have **Owner** access to the repository. Either the project name is not valid, or the connection used doesn't have access to the project/repository.

12. In **Output artifact format**, you must choose the format for your artifacts.

    - To store output artifacts from the GitLab.com action using the default method, choose **CodePipeline default**. The action accesses the files from the GitLab.com 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.

  If you choose this option, you will need to update the permissions for your CodeBuild project service role as shown in Add CodeBuild GitClone permissions for connections to Bitbucket, GitHub, GitHub Enterprise Server, or GitLab.com. For a tutorial that shows you how to use the **Full clone** option, see Tutorial: Use full clone with a GitHub pipeline source.

13. Choose to save the source action and continue.

Create a connection to GitLab.com (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 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 `GitLab` and the specified connection name is `MyConnection`.

   ```
   aws codestar-connections create-connection --provider-type GitLab --connection-name
    MyConnection
   ```

   If successful, this command returns the connection ARN information similar to the following.

   ```
   {
       "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.
3. The pipeline defaults to detect changes on code push to the connection source repository. To configure the pipeline trigger configuration for manual release or for Git tags, do one of the following:

   - To configure the pipeline trigger configuration to start with a manual release only, add the following line to the configuration:

     ```
     "DetectChanges": "false",
     ```

   - To configure the pipeline trigger configuration to filter with triggers, see more details in Add trigger with code push or pull request event types. For example, the following adds to the pipeline level of the pipeline JSON definition. In this example, `release-v0` and `release-v1` are the Git tags to include, and `release-v2` is the Git tag to exclude.

     ```
     "triggers": [
                 {
                     "providerType": "CodeStarSourceConnection",
                     "gitConfiguration": {
                         "sourceActionName": "Source",
                         "push": [
                             {
                                 "tags": {
                                     "includes": [
                                         "release-v0", "release-v1"
                                     ],
                                     "excludes": [
                                         "release-v2"
                                     ]
                                 }
                             }
                         ]
                     }
                 }
             ]
     ```

Connections for GitLab self-managed

Connections allow you to authorize and establish configurations that associate your third-party provider with your AWS resources. To associate your third-party repository as a source for your pipeline, you use a connection.

> **Note**
>
> Instead of creating or using an existing connection in your account, you can use a shared connection between another AWS account. See Use a connection shared with another account.

To add a GitLab self-managed source action in CodePipeline, you can choose either to:

- Use the CodePipeline console **Create pipeline** wizard or **Edit action** page to choose the **GitLab self-managed** provider option. See Create a connection to GitLab self-managed (console) to add the action. The console helps you create a host resource and a connections resource.
- Use the CLI to add the action configuration for the `CreateSourceConnection` action with the `GitLabSelfManaged` provider and create your resources:
  - To create your connections resources, see Create a host and connection to GitLab self-managed (CLI) to create a host resource and a connections resource with the CLI.
  - Use the `CreateSourceConnection` example action configuration in CodeStarSourceConnection for Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed actions to add your action as shown in Create a pipeline (CLI).

> **Note**
>
> You can also create a connection using the Developer Tools console under **Settings**. See Create a Connection.

Before you begin:

- You must have already created an account with GitLab and have GitLab Enterprise Edition or GitLab Community Edition with a self-managed installation. For more information, see https://docs.gitlab.com/ee/subscriptions/self_managed/.

> **Note**
>
> Connections only provide access for the account that was used to create and authorize the connection.

> **Note**
>
> You can create connections to a repository where you have the **Owner** role in GitLab, and then the connection can be used with resources such as CodePipeline. For repositories in groups, you do not need to be the group owner.

- You must have already created a GitLab personal access token (PAT) with the `api` scope. If Admin Mode is enabled on your GitLab instance, the PAT must also include the `admin_mode` scope. For more information, see Personal access tokens and Access token scopes on the GitLab website. You must be an administrator to create and use the PAT.

> **Note**
>
> Your PAT is used to authorize the host and is not otherwise stored or used by connections. To set up a host, you can create a temporary PAT and then after you set up the host, you can delete the PAT.

- You can choose to set up your host ahead of time. You can set up a host with or without a VPC. For details about VPC configuration and additional information about creating a host, see Create a host.

Topics

- Create a connection to GitLab self-managed (console)
- Create a host and connection to GitLab self-managed (CLI)

Create a connection to GitLab self-managed (console)

Use these steps to use the CodePipeline console to add a connections action for your GitLab self-managed repository.

> **Note**
>
> GitLab self-managed connections only provide access to repositories owned by the GitLab self-managed account that was used to create the connection.

Before you begin:

For a host connection to GitLab self-managed, you must have completed the steps to create a host resource for your connection. See Manage hosts for connections.

Step 1: Create or edit your pipeline

To create or edit your pipeline

1. Sign in to the CodePipeline console.
2. Choose one of the following.
   - Choose to create a pipeline. Follow the steps in *Create a Pipeline* to complete the first screen and choose **Next**. On the **Source** page, under **Source provider**, choose **GitLab self-managed**.
   - Choose to edit an existing pipeline. Choose **Edit**, and then choose **Edit stage**. Choose to add or edit your source action. On the **Edit action** page, under **Action name**, enter the name for your action. In **Action provider**, choose **GitLab self-managed**.
3. Do one of the following:
   - Under **Connection**, if you have not already created a connection to your provider, choose **Connect to GitLab self-managed**. Proceed to Step 2: Create a Connection to GitLab self-managed.
   - Under **Connection**, if you have already created a connection to your provider, choose the connection, and then proceed to Step 3: Save your GitLab self-managed source action.

Step 2: Create a connection to GitLab self-managed

After you choose to create the connection, the **Connect to GitLab self-managed** page is shown.

To connect to GitLab self-managed

1. In **Connection name**, enter the name for your connection.

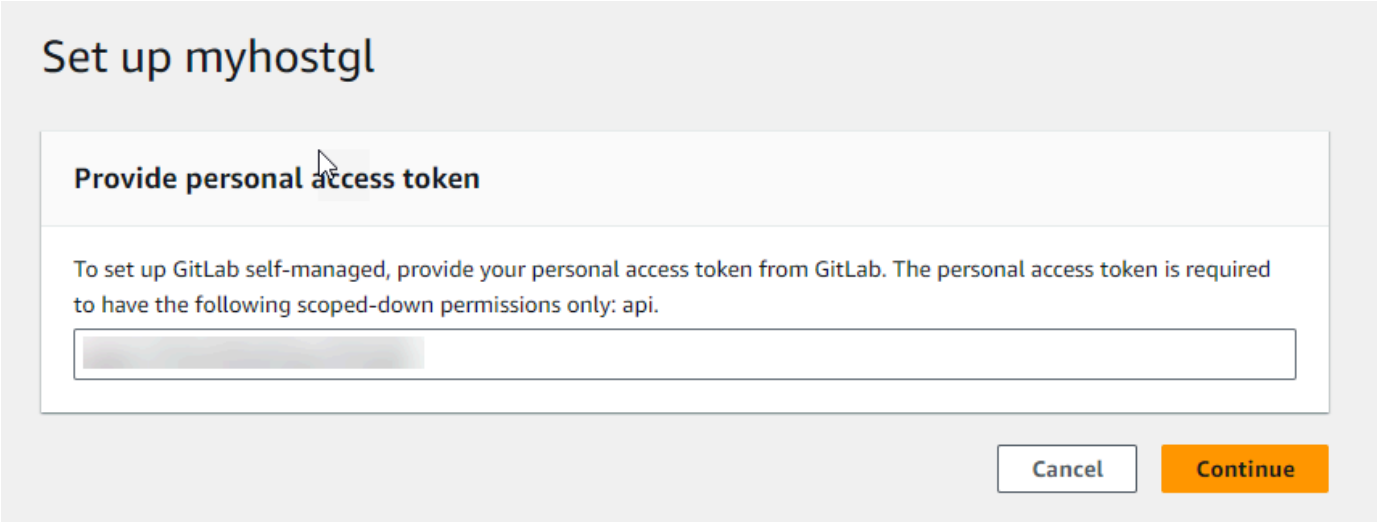
2. In **URL**, enter the endpoint for your server.

   > **Note**
   >
   > If the provided URL has already been used to set up a host for a connection, you will be prompted to choose the host resource ARN that was created previously for that endpoint.

3. If you have launched your server into an Amazon VPC and you want to connect with your VPC, choose **Use a VPC** and complete the information for the VPC.
4. Choose **Connect to GitLab self-managed**. The created connection is shown with a **Pending** status. A host resource is created for the connection with the server information you provided. For the host name, the URL is used.
5. Choose **Update pending connection**.
6. If a page opens with a redirect message confirming that you want to continue to the provider, choose **Continue**. Enter the authorization for the provider.
7. A **Set up *`host_name`*** page displays. In **Provide personal access token**, provide your GitLab PAT with the `api` scope. If Admin Mode is enabled on your GitLab instance, the PAT must also include the `admin_mode` scope. For more information, see Access token scopes on the GitLab website.

   > **Note**
   >
   > Only an administrator can create and use the PAT.

   Choose **Continue**.



8. The connection page shows the created connection in an **Available** status.

Step 3: Save your GitLab self-managed source action

Use these steps on the wizard or **Edit action** page to save your source action with your connection information.

To complete and save your source action with your connection

1. In **Repository name**, choose the name of your third-party repository.
2. Under **Pipeline triggers** you can add triggers if your action is a AWS CodeConnections action. To configure the pipeline trigger configuration and to optionally filter with triggers, see more details in Add trigger with code push or pull request event types.
3. In **Output artifact format**, you must choose the format for your artifacts.
   - To store output artifacts from the GitLab self-managed action using the default method, choose **CodePipeline default**. The action accesses the files from the 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.
4. Choose **Next** on the wizard or **Save** on the **Edit action** page.

Create a host and connection to GitLab self-managed (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 CloudFormation, use the console to edit the connection to make its status `AVAILABLE`.

You can use the AWS Command Line Interface (AWS CLI) to create a host for installed connections.

You use a host to represent the endpoint for the infrastructure where your third-party provider is installed. After you complete the host creation with the CLI, the host is in **Pending** status. You then set up, or register, the host to move it to an **Available** status. After the host is available, you complete the steps to create a connection.

To do this, use the **create-host** command.

> **Important**
>
> A host created through the AWS CLI is in `Pending` status by default. After you create a host with the CLI, use the console or the CLI to set up the host to make its status `Available`.

To create a host

1. Open a terminal (Linux, macOS, or Unix) or command prompt (Windows). Use the AWS CLI to run the **create-host** command, specifying the `--name`, `--provider-type`, and `--provider-endpoint` for your connection. In this example, the third-party provider name is `GitLabSelfManaged` and the endpoint is `my-instance.dev`.

   ```
   aws codestar-connections create-host --name MyHost --provider-type
    GitLabSelfManaged --provider-endpoint "https://my-instance.dev"
   ```

   If successful, this command returns the host Amazon Resource Name (ARN) information similar to the following.

```
{
    "HostArn": "arn:aws:codestar-connections:us-west-2:account_id:host/My-
Host-28aef605"
}
```

After this step, the host is in PENDING status.

2. Use the console to complete the host setup and move the host to an Available status.

To create a connection to GitLab self-managed

1. Open a terminal (Linux, macOS, or Unix) or command prompt (Windows). Use the AWS CLI to run the **create-connection** command, specifying the --host-arn and --connection-name for your connection.

```
aws codestar-connections create-connection --host-arn arn:aws:codestar-
connections:us-west-2:account_id:host/MyHost-234EXAMPLE --connection-name
 MyConnection
```

If successful, this command returns the connection ARN information similar to the following.

```
{
    "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/
aEXAMPLE-8aad"
}
```

2. Use the console to set up the pending connection.
3. The pipeline defaults to detect changes on code push to the connection source repository. To configure the pipeline trigger configuration for manual release or for Git tags, do one of the following:

   - To configure the pipeline trigger configuration to start with a manual release only, add the following line to the configuration:

     ```
     "DetectChanges": "false",
     ```

   - To configure the pipeline trigger configuration to filter with triggers, see more details in Add trigger with code push or pull request event types. For example, the following adds to the

pipeline level of the pipeline JSON definition. In this example, `release-v0` and `release-v1` are the Git tags to include, and `release-v2` is the Git tag to exclude.

```
"triggers": [
            {
                "providerType": "CodeStarSourceConnection",
                "gitConfiguration": {
                    "sourceActionName": "Source",
                    "push": [
                        {
                            "tags": {
                                "includes": [
                                    "release-v0", "release-v1"
                                ],
                                "excludes": [
                                    "release-v2"
                                ]
                            }
                        }
                    ]
                }
            }
        ]
```

Use a connection shared with another account

You can create and manage a shared connection using AWS RAM. This allows connections to be shared between AWS accounts for access to third-party repositories. This allows a single connection to be used in CodePipeline pipelines across accounts while reducing the need for users to manage and administer separate connections in each account.

To use shared connections in CodePipeline, do the following.

- Create a connection using the Developer Tools console under **Settings**. See Create a Connection.
- Set up the resource share using AWS RAM. See Share connections with AWS accounts.
- When you use the CodePipeline console **Create pipeline** wizard or **Edit action** page to choose the connection provider, such as the **Bitbucket** provider option, you can choose the connection that has been shared with the target account.

Automate starting pipelines using triggers and filtering

Triggers allow you to configure your pipeline to start on a particular event type or filtered event type, such as when a change on a particular branch or pull request is detected. Triggers are configurable for source actions with connections that use the `CodeStarSourceConnection` action in CodePipeline, such as GitHub, Bitbucket, and GitLab. For more information about source actions that use connections, see Add third-party source providers to pipelines using AWS CodeConnections.

Source actions, such as CodeCommit and S3, use automated change detection to start pipelines when a change is made. For more information, see CodeCommit source actions and EventBridge.

You specify triggers using the console or CLI.

You specify filter types as follows:

- **No filter**

  This trigger configuration starts your pipeline on any push to the default branch specified as part of action configuration.
- **Specify filter**

  You add a filter that starts your pipeline on a specific filter, such as on branch names for a code push, and fetches the exact commit. This also configures the pipeline not to start automatically on any change.
  - **Push**
    - Valid filter combinations are:
      - **Git tags**

        Include or exclude
      - **branches**

        Include or exclude
      - **branches + file paths**

        Include or exclude
  - **Pull request**
    - Valid filter combinations are:

- **branches**

  Include or exclude

- **branches + file paths**

  Include or exclude

- **Do not detect changes**

  This does not add a trigger and the pipeline does not start automatically on any change.

The following table provides valid filter options for each event type. The table also shows which trigger configurations default to true or false for automatic change detection in the action configuration.

Trigger configuration	Event type	Filter options	Detect changes
Add a trigger – no filter	none	none	true
Add a trigger – filter on code push	push event	Git tags, branches, file paths	false
Add a trigger – filter for pull requests	pull requests	branches, file paths	false
No trigger – do not detect	none	none	false

Note

This trigger type uses automated change detection (as the `Webhook` trigger type). The source action providers that use this trigger type are connections configured for code push (Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed).

For field definitions and further reference for triggers, see

For a list of field definitions in the JSON structure, see `triggers`.

For filtering, regular expression patterns in glob format are supported as detailed in Working with glob patterns in syntax.

> **Note**
>
> In certain cases, for pipelines with triggers that are filtered on file paths, the pipeline might not start when a branch with a file path filter is first created. For more information, see Pipelines with connections that use trigger filtering by file paths might not start at branch creation.

Considerations for trigger filters

The following considerations apply when using triggers.

- You cannot add more than one trigger per source action.
- You can add multiple filter types to a trigger. For an example, see 4: A trigger with two push filter types with conflicting includes and excludes.
- For a trigger with branch and file paths filters, when pushing the branch for the first time, the pipeline won't run since there is not access to the list of files changed for the newly created branch.
- Merging a pull request might trigger two pipeline executions in cases where push (branches filter) and pull request (branches filter) trigger configurations intersect.
- For a filter that triggers your pipeline on pull request events, for the Closed pull request event type, the third-party repository provider for your connection might have a separate status for a merge event. For example, in Bitbucket, the Git event for a merge is not a pull request closure event. However, in GitHub, merging a pull request is a closure event. For more information, see Pull request events for triggers by provider.
- When multiple source actions in a pipeline reference different branches of the same repository through a connection, only one branch reliably triggers the pipeline. The connection's webhook subscription is registered for the combination of pipeline and repository, not per branch. As a workaround, use a separate pipeline for each branch.

Pull request events for triggers by provider

The following table provides a summary of the Git events, such as for pull request closure, that result in pull request event types by provider.

	Repository provider for your connection			
PR event for trigger	**Bitbucket**	**GitHub**	**GHES**	**GitLab**
Open - This option triggers the pipeline when a pull request is created for the branch/file path.	Creating a pull request results in an **Opened** Git event.	Creating a pull request results in an **Opened** Git event.	Creating a pull request results in an **Opened** Git event.	Creating a pull request results in an **Opened** Git event.
Update - This option triggers the pipeline when a pull request revision is published for the branch/file path.	Publishing an update results in an **Updated** Git event.	Publishing an update results in an **Updated** Git event.	Publishing an update results in an **Updated** Git event.	Publishing an update results in an **Updated** Git event.
Closed - This option triggers the pipeline when a pull request is closed for the branch/file path.	Merging a pull request in Bitbucket results in a **Closed** Git event. **Important:** Manually closing a pull request in Bitbucket without merging does not result	Merging or manually closing a pull request results in a **Closed** Git event.	Merging or manually closing a pull request results in a **Closed** Git event.	Merging or manually closing a pull request results in a **Closed** Git event.

	Repository provider for your connection			
PR event for trigger	**Bitbucket**	**GitHub**	**GHES**	**GitLab**
	in a **Closed** Git event.			

Examples for trigger filters

For a Git configuration with filters for push and pull request event types, the specified filters might conflict with each other. The following are examples of valid filter combinations for push and pull request events. A trigger can contain multiple filter types, such as two push filter types in the trigger configuration, and the push and pull request filter types will use an OR operation between them, meaning any match will start the pipeline. Similarly, each filter type can include multiple filters such as filePaths and branches; these filters will use an AND operation, meaning only a full match will start the pipeline. Each filter type can contain includes and excludes, where excludes take precedence over includes. If a branch or file path matches an exclude pattern, it will not trigger the pipeline, even if it also matches an include pattern. When a commit changes multiple files, each file is evaluated independently against the filter; if any changed file passes (matches an include and does not match an exclude), the pipeline will start. Names inside of the include/exclude, such as branch names, use an OR operation. The following list summarizes the operations for each part of the Git configuration object.

For a list of field definitions in the JSON structure and a detailed reference for includes and excludes, see `triggers`.

Example 1: A filter type with filters for branches and file paths (AND operation)

For a single filter type such as pull request, you can combine filters, and these filters will use an AND operation, meaning only a full match will start the pipeline. The following example shows a Git configuration for a push event type with two different filters (`filePaths` and `branches`). In the following example, `filePaths` will be AND'ed with `branches`:

```
{
  "filePaths": {
    "includes": ["common/**/*.js"]
  },
```

```
  "branches": {
    "includes": ["feature/**"]
  }
}
```

With the Git configuration above, this example shows an event that will start the pipeline execution because the AND operation succeeds. In other words, the file path `common/app.js` is included for the filter, which starts the pipeline as an AND even if the branch `refs/heads/feature/triggers` specified did not have an impact.

```
{
  "ref": "refs/heads/feature/triggers",
  ...
  "commits": [
    {
      ...
      "modified": [
        "common/app.js"
      ]
      ...
    }
  ]
}
```

The following example shows an event for a trigger with the above configuration that will not start the pipeline execution because the branch is able to filter, but the file path is not.

```
{
   "ref": "refs/heads/feature/triggers",
  ...
  "commits": [
    {
      ...
      "modified": [
        "src/Main.java"
      ]
      ...
    }
  ]
}
```

Example 2: Excludes take precedence over includes

Within a single filter, excludes take precedence over includes. The following example shows a Git configuration with a single filter (`branches`) within the configuration object. This means that if a branch matches an exclude pattern (`feature-branch` in the example), the pipeline will not be triggered, even if it also matches an include pattern. If the include pattern matches and no exclude pattern matches, such as for the `main` branch, then the pipeline will be triggered.

For the following example JSON:

- Pushing a commit to the `main` branch will trigger the pipeline
- Pushing a commit to the `feature-branch` branch will not trigger the pipeline.

```
{
  "branches": {
      "Includes": [
          "main"
      ],
      "Excludes": [
          "feature-branch"
      ]
   }
```

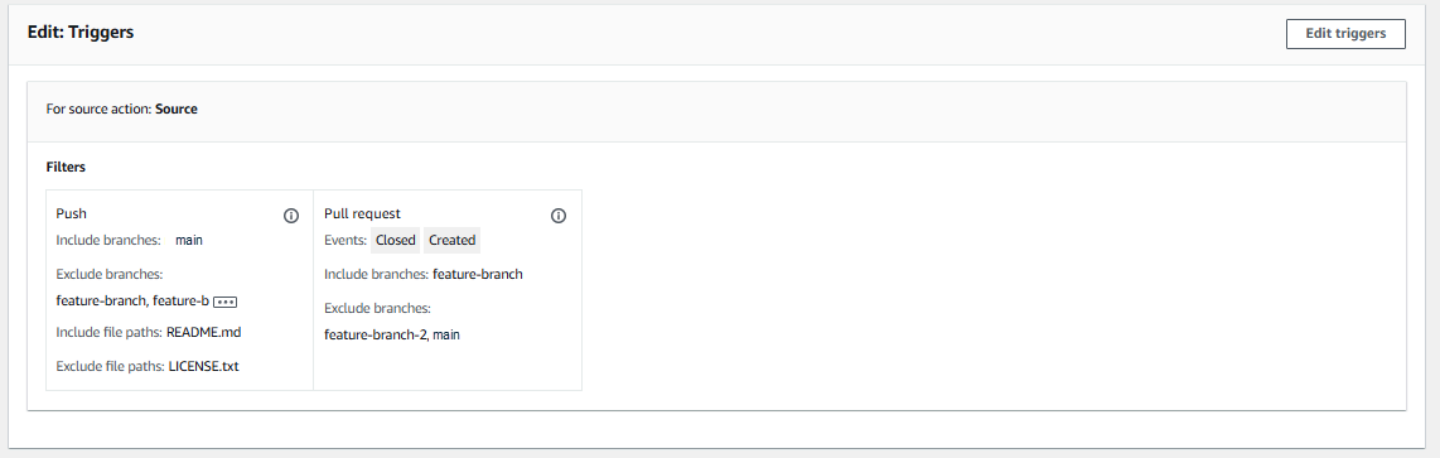
Example 3: A trigger with push and pull request filter types (OR operation), filters for file paths and branches (AND operation), and includes/excludes (excludes take precedence)

Trigger configuration objects, such as a trigger that contains a push event type and a pull request event type, use an OR operation between the two event types. The following example shows a trigger configuration with a push event type with the `main` branch included and one pull request event type with the same branch `main` excluded. Additionally, the push event type has one file path `LICENSE.txt` excluded and one file path `README.MD` included. For the second event type, a pull request that is either `Closed` or `Created` on the `feature-branch` branch (included) starts the pipeline, and the pipeline does not start when creating or closing a pull request on the `feature-branch-2` or `main` branches (excluded). Within each event type, excludes take precedence over includes. For example, for a pull request event on the `feature-branch` branch (included for the pull request), the push event type excludes the `feature-branch` branch, so the push will not trigger the pipeline.

For the following example,

- Pushing a commit to the `main` branch (included) for the `README.MD` file path (included) will trigger the pipeline.
- On the `feature-branch` branch (excluded), pushing a commit will not trigger the pipeline.
- On the included branch, editing the `README.MD` file path (included) triggers the pipeline.
- On the included branch, editing only the `LICENSE.TXT` file path (excluded) does not trigger the pipeline. However, if the same commit also changes `README.MD` (included), the pipeline will trigger because each file is evaluated independently.
- On the `feature-branch` branch, closing a pull request will trigger the pipeline because `feature-branch` is included for the pull request event type and the event type CLOSED matches.

The following image shows the configuration.



The following is the example JSON for the configuration.

```
"triggers": [
            {
                "providerType": "CodeStarSourceConnection",
                "gitConfiguration": {
                    "sourceActionName": "Source",
                    "push": [
                        {
                            "branches": {
                                "includes": [
                                    "main"
                                ],
                                "excludes": [
                                    "feature-branch",
```

```
                                "feature-branch-2"
                            ]
                        },
                        "filePaths": {
                            "includes": [
                                "README.md"
                            ],
                            "excludes": [
                                "LICENSE.txt"
                            ]
                        }
                    }
                ],
                "pullRequest": [
                    {
                        "events": [
                            "CLOSED",
                            "OPEN"
                        ],
                        "branches": {
                            "includes": [
                                "feature-branch"
                            ],
                            "excludes": [
                                "feature-branch-2",
                                "main"
                            ]
                        }
                    }
                ]
            }
        }
    ]
},
```

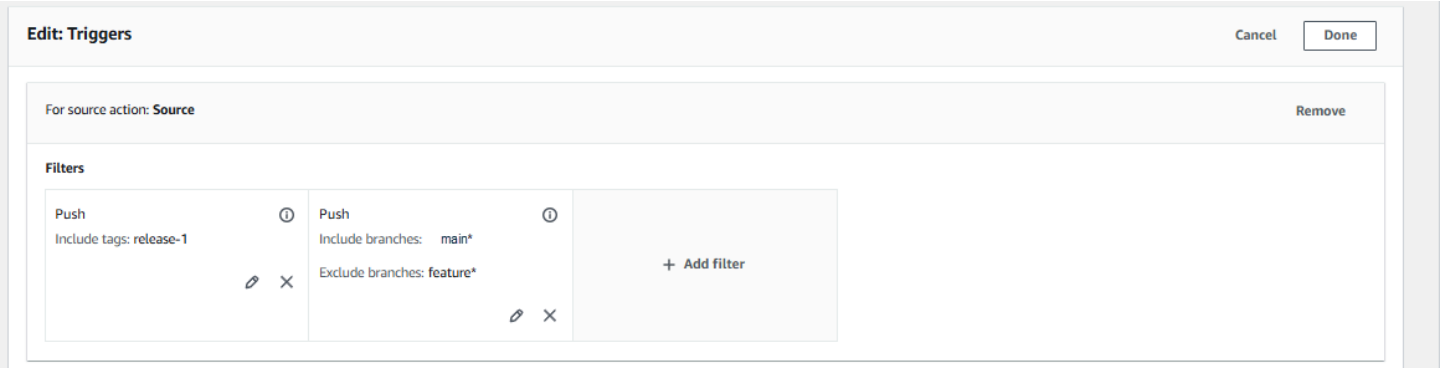
Example 4: A trigger with two push filter types with conflicting includes and excludes

The following image shows a push filter type specifying to filter on the tag `release-1` (included). A second push filter type is added specifying to filter on the branch `main` (included) and to not start for a push to the `feature*` branches (excluded).

For the following example:

- Pushing a release from the tag `release-1` (included for the first push filter) on the `feature-branch` branch (excluded as `feature*` for the second push filter) will trigger the pipeline because the two push filter types use an OR operation between them, and the first push filter (tag `release-1`) matches.
- Pushing a release from the `main` branch (included for the second Push filter) will start the pipeline.

The following example of the Edit page shows the two Push filter types and their configuration for includes and excludes.



The following is the example JSON for the configuration.

```
"triggers": [
            {
                "providerType": "CodeStarSourceConnection",
                "gitConfiguration": {
                    "sourceActionName": "Source",
                    "push": [
                        {
                            "tags": {
                                "includes": [
                                    "release-1"
                                ]
                            }
                        },
                        {
                            "branches": {
                                "includes": [
                                    "main*"
                                ],
                                "excludes": [
```

```
                                "feature*"
                            ]
                        }
                    }
                ]
            }
        }
    ]
},
```

Example 5: Trigger configured while default action configuration BranchName is used for a manual start

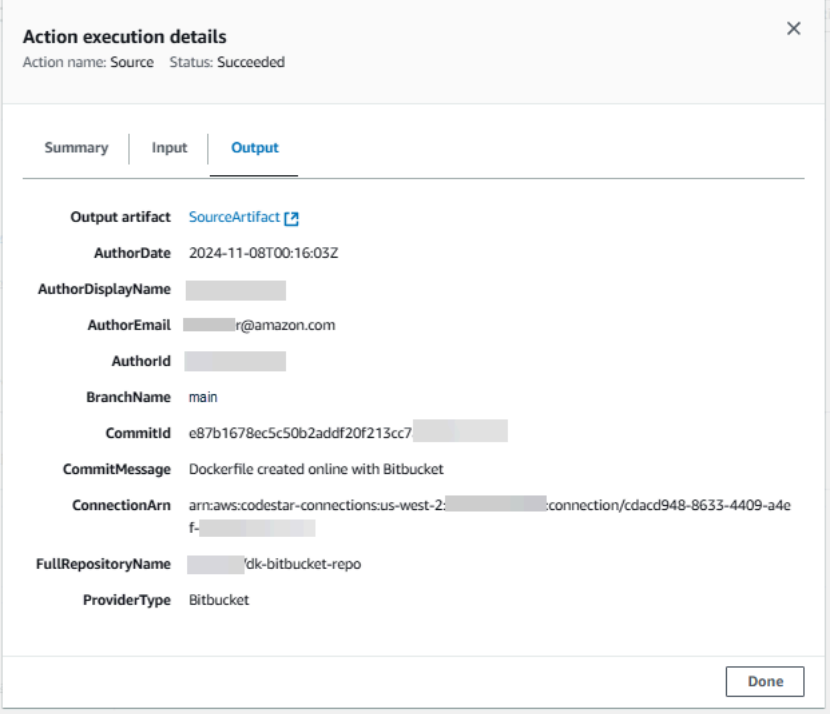
The action configuration default `BranchName` field defines a single branch that will be used when the pipeline is started manually, while triggers with filters can be used for any branch or branches that you specify.

The following is the example JSON for the action configuration showing the `BranchName` field.

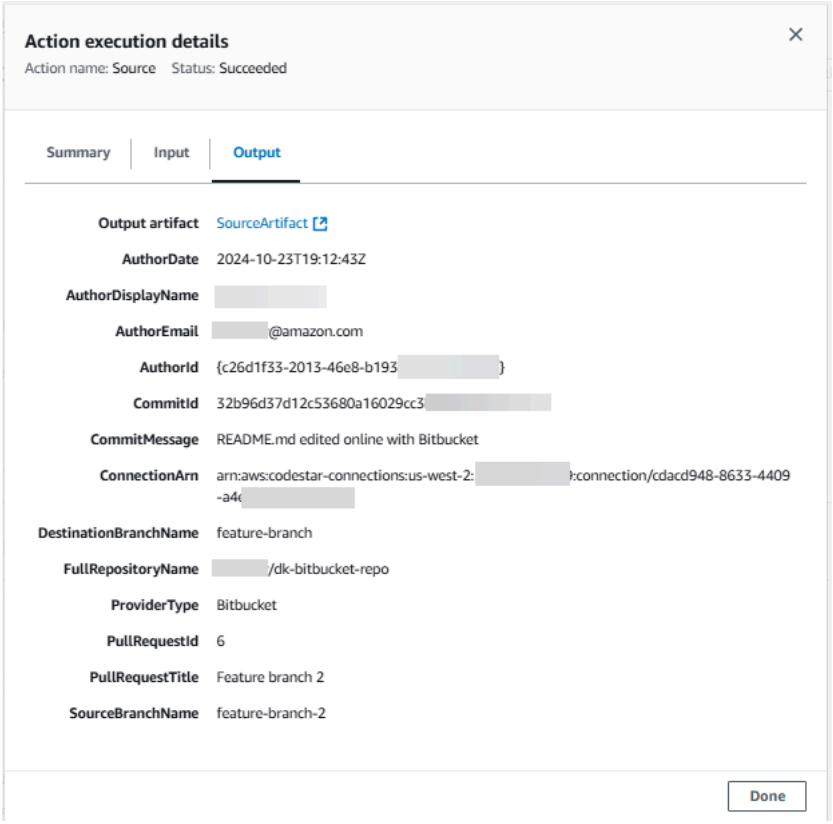
```
{
                "name": "Source",
                "actions": [
                    {
                        "name": "Source",
                        "actionTypeId": {
                            "category": "Source",
                            "owner": "AWS",
                            "provider": "CodeStarSourceConnection",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "BranchName": "main",
                            "ConnectionArn": "ARN",
                            "DetectChanges": "false",
                            "FullRepositoryId": "owner-name/my-bitbucket-repo",
                            "OutputArtifactFormat": "CODE_ZIP"
                        },
                        "outputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
```

```
            ],
            "inputArtifacts": [],
            "region": "us-west-2",
            "namespace": "SourceVariables"
        }
    ],
```

The following example action output shows the default branch main was used when the pipeline was manually started.



The following example action output shows the pull request and branch that was used for the trigger when filtered by pull request.



Add trigger on code push with no filter

Triggers allow you to configure your pipeline to start on a particular event type, such as a code push or pull request. Triggers are configurable for source actions with connections that use the CodeStarSourceConnection action in CodePipeline, such as GitHub, Bitbucket, and GitLab.

Adding a trigger (console)

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 edit. Otherwise, use these steps on the pipeline creation wizard.
3. On the pipeline details page, choose **Edit**.
4. On the **Edit** page, choose the source action you want to edit. Choose **Edit triggers**. Choose to add a trigger.
5. In **Trigger type**, choose **No filter**.

Add trigger with code push or pull request event types

You can configure filters for pipeline triggers to have pipeline executions started for different Git events, such as tag or branch push, changes in specific file paths, a pull request opened into a specific branch, and so on. You can use the AWS CodePipeline console or the **create-pipeline** and **update-pipeline** commands in the AWS CLI to configure trigger filters.

> **Note**
>
> The action configuration `BranchName` field defines a single branch, while triggers with filters can be used for any branch or branches that you specify. For a pipeline where triggers are used to filter branches by push or pull request, the pipeline won't use the default `BranchName` field branch in the action configuration. However, the branch in the `BranchName` field in the action configuration is the default when the pipeline is started manually. For an example, see 5: Trigger configured while default action configuration BranchName is used for a manual start.

You can specify filters for the following trigger types:

- **Push**

  A push trigger starts a pipeline when a change is pushed to your source repository. The execution will use the commit from the branch that you're pushing *to* (that is, the destination branch). You can filter push triggers on branches, file paths, or Git tags.

- **Pull request**

  A pull request trigger starts a pipeline when a pull request is opened, updated, or closed in your source repository. The execution will use the commit from the source branch that you're pulling *from* (that is, the source branch). You can filter pull request triggers on branches and file paths.

  The supported event types for pull requests are the following. All other pull request events are ignored.

  - Opened
  - Updated
  - Closed (merged)

> **Note**
>
> Certain pull request event behavior can differ by provider. For details, see Pull request events for triggers by provider.

The pipeline definition allows you to combine different filters within the same push trigger configuration. For details about the pipeline definition, see Add filters for push and pull request event types (CLI). For a list of field definitions, see triggers in the *Pipeline structure reference* in this guide.

Topics

- Add filters for push and pull request event types (console)
- Add filters for push and pull request event types (CLI)
- Add filters for push and pull request event types (CloudFormation templates)

Add filters for push and pull request event types (console)

You can use the console to add filters for push events and include or exclude branches or file paths.

Add filters (console)

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 edit. Otherwise, use these steps on the pipeline creation wizard.
3. On the pipeline details page, choose **Edit**.
4. On the **Edit** page, choose the source action you want to edit. Choose **Edit triggers**. Choose **Specify filter**.
5. In **Event type**, choose **Push** from the following options.

   - Choose **Push** to start the pipeline when a change is pushed to your source repository. Choosing this enables the fields to specify filters for branches and file paths or Git tags.

- Choose **Pull request** to start the pipeline when a pull request is opened, updated, or closed in your source repository. Choosing this enables the fields to specify filters for destination branches and file paths.

6. Under **Push**, in **Filter type**, choose one of the following options.

- Choose **Branch** to specify the branches in your source repository that the trigger monitors in order to know when to start a workflow run. In **Include**, enter patterns for branch names in glob format that you want to specify for the trigger configuration to start your pipeline on changes in the specified branches. In **Exclude**, enter the regex patterns for branch names in glob format that you want to specify for the trigger configuration to ignore and to not start your pipeline on changes in the specified branches. See Working with glob patterns in syntax for more information.

  > **Note**
  >
  > If the include and exclude both have the same pattern, then the default is to exclude the pattern.

  You can use glob patterns to define your branch names. For example, use `main*` to match all branches beginning with `main`. See Working with glob patterns in syntax for more information.

  For a push trigger, specify the branches you're pushing *to*, that is, the *destination* branches. For a pull request trigger, specify destination branches you're opening a pull request to.

- (Optional) Under **File paths**, specify file paths for your trigger. Enter the names in **Include** and **Exclude** as appropriate.

  You can use glob patterns to define your file path names. For example, use `prod*` to match all file paths beginning with `prod`. See Working with glob patterns in syntax for more information.

- Choose **Tags** to configure the pipeline trigger configuration to start with Git tags. In **Include**, enter patterns for tag names in glob format that you want to specify for the trigger configuration to start your pipeline on release of the specified tag or tags. In **Exclude**, enter the regex patterns for tag names in glob format that you want to specify for the trigger configuration to ignore and to not start your pipeline on release of the specified tag or tags.

If the include and exclude both have the same tag pattern, then the default is to exclude the tag pattern.

7. Under **Push**, in **Filter type**, choose one of the following options.

- Choose **Branch** to specify the branches in your source repository that the trigger monitors in order to know when to start a workflow run. In **Include**, enter patterns for branch names in glob format that you want to specify for the trigger configuration to start your pipeline on changes in the specified branches. In **Exclude**, enter the regex patterns for branch names in glob format that you want to specify for the trigger configuration to ignore and to not start your pipeline on changes in the specified branches. See Working with glob patterns in syntax for more information.

  > **Note**
  >
  > If the include and exclude both have the same pattern, then the default is to exclude the pattern.

  You can use glob patterns to define your branch names. For example, use `main*` to match all branches beginning with `main`. See Working with glob patterns in syntax for more information.

  For a push trigger, specify the branches you're pushing *to*, that is, the *destination* branches. For a pull request trigger, specify destination branches you're opening a pull request to.

- (Optional) Under **File paths**, specify file paths for your trigger. Enter the names in **Include** and **Exclude** as appropriate.

  You can use glob patterns to define your file path names. For example, use `prod*` to match all file paths beginning with `prod`. See Working with glob patterns in syntax for more information.

- Choose **Pull request** to configure the pipeline trigger configuration to start with pull request events that you specify.

Add filters for push and pull request event types (CLI)

You can update the pipeline JSON to add filters for triggers.

To use the AWS CLI to create or update your pipeline, use the `create-pipeline` or `update-pipeline` command.

The following example JSON structure provides a reference for the field definitions under `create-pipeline`.

For a list of field definitions, see triggers in the *Pipeline structure reference* in this guide.

```
{
    "pipeline": {
        "name": "MyServicePipeline",
        "triggers": [
            {
                "provider": "Connection",
                "gitConfiguration": {
                    "sourceActionName": "ApplicationSource",
                    "push": [
                        {
                            "filePaths": {
                                "includes": [
                                    "projectA/**",
                                    "common/**/*.js"
                                ],
                                "excludes": [
                                    "**/README.md",
                                    "**/LICENSE",
                                    "**/CONTRIBUTING.md"
                                ]
                            },
                            "branches": {
                                "includes": [
                                    "feature/**",
                                    "release/**"
                                ],
                                "excludes": [
                                    "mainline"
                                ]
                            },
                            "tags": {
                                "includes": [
                                    "release-v0", "release-v1"
                                ],
                                "excludes": [
```

```
                                "release-v2"
                            ]
                        }
                    }
                ],
                "pullRequest": [
                    {
                        "events": [
                            "CLOSED"
                        ],
                        "branches": {
                            "includes": [
                                "feature/**",
                                "release/**"
                            ],
                            "excludes": [
                                "mainline"
                            ]
                        },
                        "filePaths": {
                            "includes": [
                                "projectA/**",
                                "common/**/*.js"
                            ],
                            "excludes": [
                                "**/README.md",
                                "**/LICENSE",
                                "**/CONTRIBUTING.md"
                            ]
                        }
                    }
                ]
            }
        }
    ],
    "stages": [
        {
            "name": "Source",
            "actions": [
                {
                    "name": "ApplicationSource",
                    "configuration": {
                        "BranchName": "mainline",
```

```
                            "ConnectionArn": "arn:aws:codestar-connections:eu-
central-1:111122223333:connection/fe9ff2e8-ee25-40c9-829e-65f8EXAMPLE",
                            "FullRepositoryId": "monorepo-example",
                            "OutputArtifactFormat": "CODE_ZIP"
                        }
                    }
                ]
            }
        ]
    }
}
```

Add filters for push and pull request event types (CloudFormation templates)

You can update the pipeline resource in CloudFormation to add trigger filtering.

The following example template snippet provides a YAML reference for triggers field definitions. For a list of field definitions, see triggers in the *Pipeline structure reference* in this guide.

For a full template example for a connection source and trigger filter configuration, see Pipeline with two stages and trigger configuration in the *CloudFormation User Guide*.

```
pipeline:
  name: MyServicePipeline
  executionMode: PARALLEL
  triggers:
    - provider: CodeConnection
      gitConfiguration:
        sourceActionName: ApplicationSource
        push:
          - filePaths:
              includes:
                - projectA/**
                - common/**/*.js
              excludes:
                - '**/README.md'
                - '**/LICENSE'
                - '**/CONTRIBUTING.md'
            branches:
              includes:
                - feature/**
```

```
                - release/**
              excludes:
                - mainline
          - tags:
              includes:
                - release-v0
                - release-v1
              excludes:
                - release-v2
        pullRequest:
          - events:
              - CLOSED
            branches:
              includes:
                - feature/**
                - release/**
              excludes:
                - mainline
            filePaths:
              includes:
                - projectA/**
                - common/**/*.js
              excludes:
                - '**/README.md'
                - '**/LICENSE'
                - '**/CONTRIBUTING.md'
  stages:
    - name: Source
      actions:
        - name: ApplicationSource
          configuration:
            BranchName: mainline
            ConnectionArn: arn:aws:codestar-connections:eu-
central-1:111122223333:connection/fe9ff2e8-ee25-40c9-829e-65f85EXAMPLE
            FullRepositoryId: monorepo-example
            OutputArtifactFormat: CODE_ZIP
```

Add trigger to turn off change detection

Triggers allow you to configure your pipeline to start on a particular event type, such as a code push or pull request. Triggers are configurable for source actions with connections that use the `CodeStarSourceConnection` action in CodePipeline, such as GitHub, Bitbucket, and GitLab.

Adding a trigger to turn off change detection (console)

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 edit. Otherwise, use these steps on the pipeline creation wizard.
3. On the pipeline details page, choose **Edit**.
4. On the **Edit** page, choose the source action you want to edit. Choose **Edit triggers**. Choose to add a trigger.
5. In **Trigger type**, choose **Do not detect changes**.

Start and stop pipelines manually

You can use the AWS CodePipeline console or the AWS CLI to manually start and stop pipelines. There are automated methods for starting pipelines with event-based change detection as detailed in, or trigger configuration as detailed in .

Topics

- Start a pipeline in CodePipeline
- Stop a pipeline execution in CodePipeline

Start a pipeline in CodePipeline

Each pipeline execution can be started based on a different trigger. Each pipeline execution can have a different type of trigger, depending on how the pipeline is started. The trigger type for each execution is shown in the execution history for a pipeline. Trigger types can depend on the source action provider as follows:

> **Note**
>
> You cannot specify more than one trigger per source action.

- **Pipeline creation**: When a pipeline is created, a pipeline execution starts automatically. This is the `CreatePipeline` trigger type in the **Execution history**.
- **Changes on revised objects**: This category represents the `PutActionRevision` trigger type in the **Execution history**.
- **Change detection on branch and commit for a code push**: This category represents the `CloudWatchEvent` trigger type in the **Execution history**. When a change is detected to a source commit and branch in the source repository, your pipeline starts. This trigger type uses automated change detection. The source action providers that use this trigger type are S3 and CodeCommit. This type is also used for a schedule that starts your pipeline. See Start a pipeline on a schedule.
- **Polling for source changes**: This category represents the `PollForSourceChanges` trigger type in the **Execution history**. When a change is detected to a source commit and branch in the source repository through polling, your pipeline starts. This trigger type is not recommended and

should be migrated to use automated change detection. The source action providers that use this trigger type are S3 and CodeCommit.

- **Webhook events for third-party sources**: This category represents the `Webhook` trigger type in the **Execution history**. When a change is detected by a webhook event, your pipeline starts. This trigger type uses automated change detection. The source action providers that use this trigger type are connections configured for code push (Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed).
- **WebhookV2 events for third-party sources**: This category represents the `WebhookV2` trigger type in the **Execution history**. This type is for executions that are triggered based on triggers defined in the pipeline definition. When a release with a specified Git tag is detected, your pipeline starts. You can use Git tags to mark a commit with a name or other identifier that helps other repository users understand its importance. You can also use Git tags to identify a particular commit in the history of a repository. This trigger type disables automated change detection. The source action providers that use this trigger type are connections configured for Git tags (Bitbucket Cloud, GitHub, GitHub Enterprise Server, and GitLab.com).
- **Manually starting a pipeline**: This category represents the `StartPipelineExecution` trigger type in the **Execution history**. You can use the console or the AWS CLI to start a pipeline manually. For information, see Start a pipeline manually.
- **RollbackStage**: This category represents the `RollbackStage` trigger type in the **Execution history**. You can use the console or the AWS CLI to roll back a stage manually or automatically. For information, see Configuring stage rollback.

When you add a source action to your pipeline that uses automated change detection trigger types, the actions work with additional resources. Creating each source action is detailed in separate sections due to these additional resources for change detection. For details about each source provider and the change detection methods required for automated change detection, see Source actions and change detection methods.

Topics

- Start a pipeline manually
- Start a pipeline on a schedule
- Start a pipeline with a source revision override

Start a pipeline manually

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)
- Start a pipeline manually (CLI)

Start a pipeline manually (console)

To manually start a pipeline and run the most recent revision through a pipeline

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 want to start.
3. On the pipeline details page, choose **Release change**. If the pipeline is configured to pass parameters (pipeline variables), then choosing **Release change** opens the **Release change** window. In **Pipeline variables**, in the field or fields for the variables at the pipeline level, enter the value or values you want to pass for this pipeline execution. For more information, see Variables reference.

   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*:

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

To start a pipeline where variables are configured at the pipeline level, use the **start-pipeline-execution** command with the optional **--variables** argument to start the pipeline and add the variables that will be used in the execution. For example, to add a variable `var1` with a value of 1, use the following command:

```
aws codepipeline start-pipeline-execution --name MyFirstPipeline --variables
 name=var1,value=1
```

2. To verify success, view the returned object. This command returns an execution ID, similar to the following:

```
{
    "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 pipelines (console) and View pipeline details and history (CLI).

Start a pipeline on a schedule

You can set up a rule in EventBridge to start a pipeline on a schedule.

Create an EventBridge rule that schedules your pipeline to start (console)

To create an EventBridge rule with a schedule as the event source

1. Open the Amazon EventBridge console at https://eusc-de-east-1.console.amazonaws-eusc.eu/events/.
2. In the navigation pane, choose **Rules**.
3. Choose **Create rule**, and then under **Rule detail**, 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 for this schedule.

   > **Note**
   >
   > You can find the pipeline ARN under **Settings** in the console. See View the pipeline ARN and service role ARN (console).

7. Choose one of the following to create or specify an IAM service role that gives EventBridge permissions to invoke the target associated with your EventBridge rule (in this case, the target is CodePipeline).

   - Choose **Create a new role for this specific resource** to create a service role that grants EventBridge permissions to start your pipeline executions.
   - Choose **Use existing role** to enter a service role that grants EventBridge permissions to start your pipeline executions.
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 an EventBridge 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 an EventBridge 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 EventBridge on a schedule.

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

2. 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.

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

3. Grant permissions for EventBridge to use CodePipeline to invoke the rule. For more information, see Using resource-based policies for Amazon EventBridge.

   a. Use the following sample to create the trust policy to allow EventBridge to assume the service role. Name it `trustpolicyforEB.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.

```
aws iam create-role --role-name Role-for-MyRule --assume-role-policy-document
 file://trustpolicyforEB.json
```

c. Create the permissions policy JSON as shown in this sample for the pipeline named `MyFirstPipeline`. Name the permissions policy `permissionspolicyforEB.json`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:StartPipelineExecution"
            ],
            "Resource": [
                "arn:aws-eusc:codepipeline:us-
west-2:111122223333:MyFirstPipeline"
            ]
        }
    ]
}
```

d. Use the following command to attach the new `CodePipeline-Permissions-Policy-for-EB` 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-EB --policy-document file://permissionspolicyforCWE.json
```

Start a pipeline with a source revision override

You can use overrides to start a pipeline with a specific source revision ID that you provide for the pipeline execution. For example, if you want to start a pipeline that will process a specific commit ID from your CodeCommit source, you can add the commit ID as an override when you start your pipeline.

Note

You can also create a source override using input transform entry to use the `revisionValue` in EventBridge for your pipeline event, where the `revisionValue` is derived from the source event variable for your object key, commit, or image ID. For more information, see the optional step for input transform entry included in the procedures under Amazon ECR source actions and EventBridge resources, Connecting to Amazon S3 source actions with a source enabled for events, or CodeCommit source actions and EventBridge.

There are four types of source revision for `revisionType`:

- `COMMIT_ID`
- `IMAGE_DIGEST`
- `S3_OBJECT_VERSION_ID`
- `S3_OBJECT_KEY`

Note

For the `COMMIT_ID` and `IMAGE_DIGEST` types of source revisions, the source revision ID applies to all content in the repository, across all branches.

Note

For the `S3_OBJECT_VERSION_ID` and `S3_OBJECT_KEY` types of source revisions, either of the types can be used independently, or they can be used together to override the source with a specific ObjectKey and VersionID. For `S3_OBJECT_KEY`, the configuration parameter

`AllowOverrideForS3ObjectKey` needs to be set to `true`. For more information on S3 source configuration parameters, see Configuration parameters.

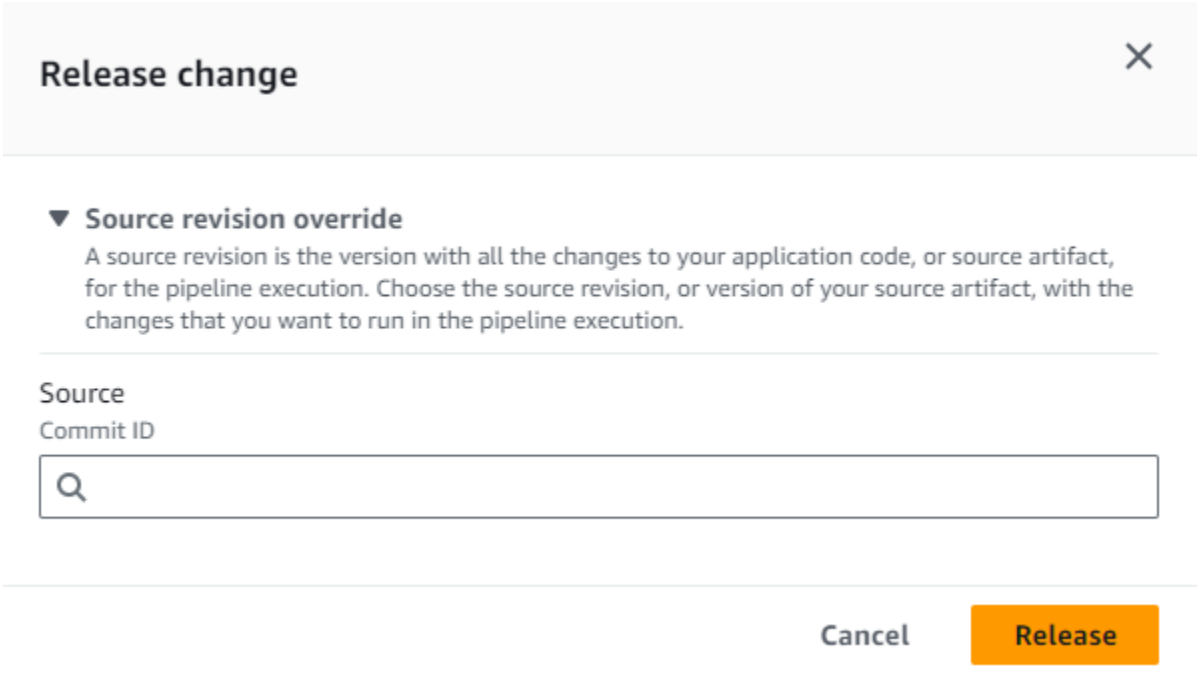
Topics

- Start a pipeline with a source revision override (console)
- Start a pipeline with a source revision override (CLI)

Start a pipeline with a source revision override (console)

To manually start a pipeline and run the most recent revision through a pipeline

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 want to start.
3. On the pipeline details page, choose **Release change**. Choosing **Release change** opens the **Release change** window. For **Source revision override**, choose the arrow to expand the field. In **Source**, enter the source revision ID. For example, if your pipeline has a CodeCommit source, choose the commit ID from the field that you want to use.



Start a pipeline with a source revision override (CLI)

To manually start a pipeline and run the specified source revision ID for 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. You also use the **--source-revisions** argument to provide the source revision ID. The source revision is made up of the actionName, revisionType, and revisionValue. Valid revisionType values are `COMMIT_ID | IMAGE_DIGEST | S3_OBJECT_VERSION_ID | S3_OBJECT_KEY`.

   In the following example, to start running the specified change through a pipeline named **codecommit-pipeline**, the following command specifies a source action name of Source, a revision type of `COMMIT_ID`, and a commit ID of `78a25c18755ccac3f2a9eec099dEXAMPLE`.

   ```
   aws codepipeline start-pipeline-execution --name codecommit-pipeline --source-
   revisions
    actionName=Source,revisionType=COMMIT_ID,revisionValue=78a25c18755ccac3f2a9eec099dEXAMPLE
    --region us-west-1
   ```

2. To verify success, view the returned object. This command returns an execution ID, similar to the following:

   ```
   {
       "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 pipelines (console) and View pipeline details and history (CLI).

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)
- Stop an Inbound Execution (Console)
- Stop a pipeline execution (CLI)
- Stop an Inbound Execution (CLI)

Stop a pipeline execution (console)

You can use the console to stop a pipeline execution. Choose an execution, and then choose the method for stopping the pipeline execution.

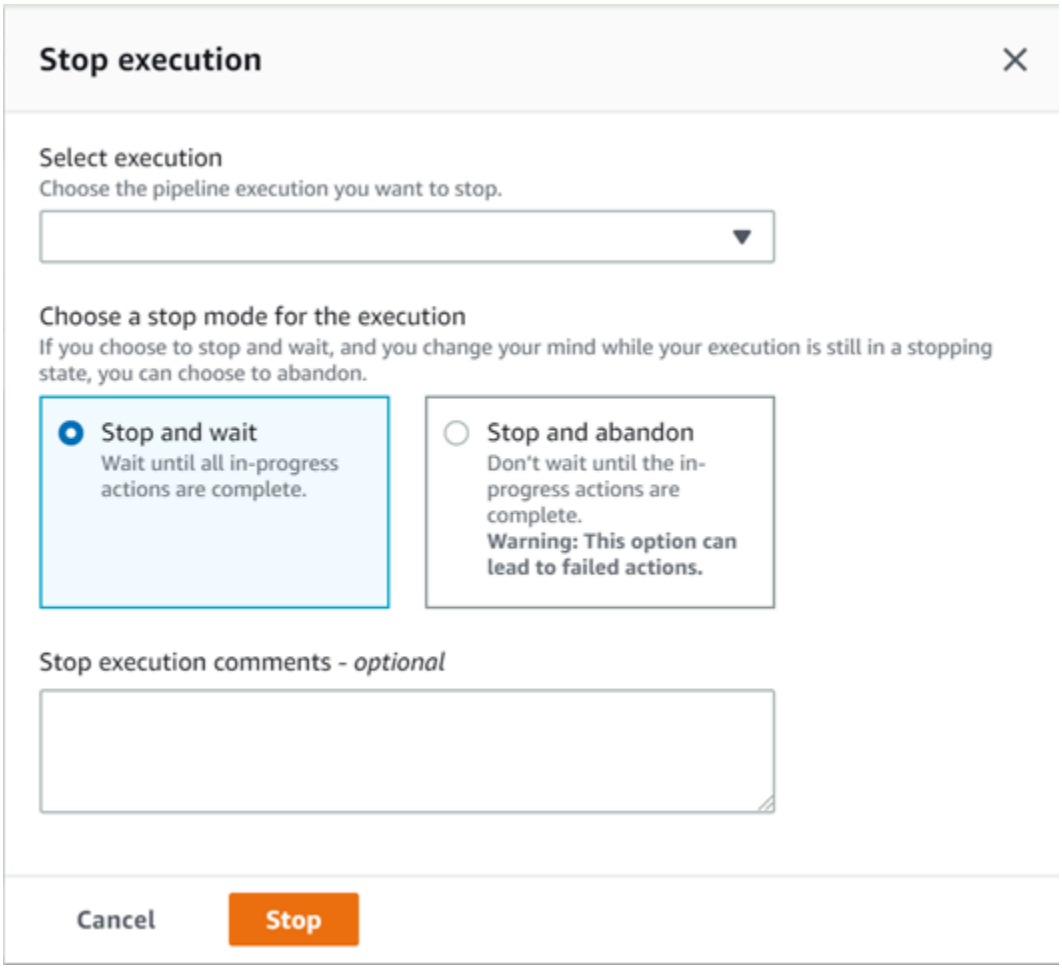
> **Note**
>
> You can also stop a pipeline execution that is an inbound execution. To learn more about stopping an inbound execution, see Stop an Inbound Execution (Console).

1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home.
2. Do one of the following:

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

   - 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.

   > **Note**
   >
   > The execution is displayed only if it is still in progress. Executions that are already complete are not displayed.



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 an Inbound Execution (Console)

You can use the console to stop an inbound execution. An inbound execution is a pipeline execution that is waiting to enter a stage where the transition has been disabled. When the transition is enabled, an inbound execution that is `InProgress` continues to enter the stage. An inbound execution that is `Stopped` does not enter the stage.

> **Note**
>
> After an inbound execution has been stopped, it cannot be retried.

If you do not see an inbound execution, then there are no pending executions at a disabled stage transition.

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 will be displayed.
2. Choose the name of the pipeline for which you want to stop the inbound execution. Do one of the following:

   - In the Pipeline view, choose the inbound execution ID and then choose to stop the execution.
   - Choose the pipeline and choose **View history**. In the execution history, choose the inbound execution ID and then choose to stop the execution.
3. In the **Stop execution** modal, follow the steps in the section above to select the execution ID and specify the stop method.

   Use the **get-pipeline-state** command to view the status of the inbound execution.

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:

```
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:

     ```
     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:

     ```
     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:

```
aws codepipeline stop-pipeline-execution --pipeline-name MyFirstPipeline --
pipeline-execution-id d-EXAMPLE --abandon --reason "Stopping execution for a bug
 fix"
```

Stop an Inbound Execution (CLI)

You can use the CLI to stop an inbound execution. An inbound execution is a pipeline execution that is waiting to enter a stage where the transition has been disabled. When the transition is enabled, an inbound execution that is `InProgress` continues to enter the stage. An inbound execution that is `Stopped` does not enter the stage.

> **Note**
>
> After an inbound execution has been stopped, it cannot be retried.

If you do not see an inbound execution, then there are no pending executions at a disabled stage transition.

To use the AWS CLI to manually stop an inbound execution, use the **stop-pipeline-execution** command with the following parameters:

- Inbound Execution ID (required)
- Comments (optional)
- Pipeline name (required)
- Abandon flag (optional, the default is false)

Command format:

```
aws codepipeline stop-pipeline-execution --pipeline-name Pipeline_Name --
pipeline-execution-id Inbound_Execution_ID [--abandon | --no-abandon] [--
reason STOP_EXECUTION_REASON]
```

Follow the steps in the procedure above to enter the command and specify the stop method.

Use the **get-pipeline-state** command to view the status of the inbound execution.

View history and set the mode for pipeline executions

To analyze pipeline progress, you can view pipeline and action execution history. To configure the release method for processing pipeline executions, set the execution mode.

Topics

- View executions in CodePipeline
- Set or change the pipeline execution mode

View executions in CodePipeline

You can use the AWS CodePipeline console or the AWS CLI to view execution status, view execution history, and retry failed stages or actions.

Topics

- View pipeline execution history (console)
- View execution status (console)
- View an inbound execution (Console)
- View pipeline execution source revisions (console)
- View action executions (console)
- View action artifacts and artifact store information (console)
- View pipeline details and history (CLI)

View pipeline execution 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.

> **Note**
>
> When switching between specific execution modes, the pipeline view and history might change. For more information, see Set or change the pipeline execution mode.

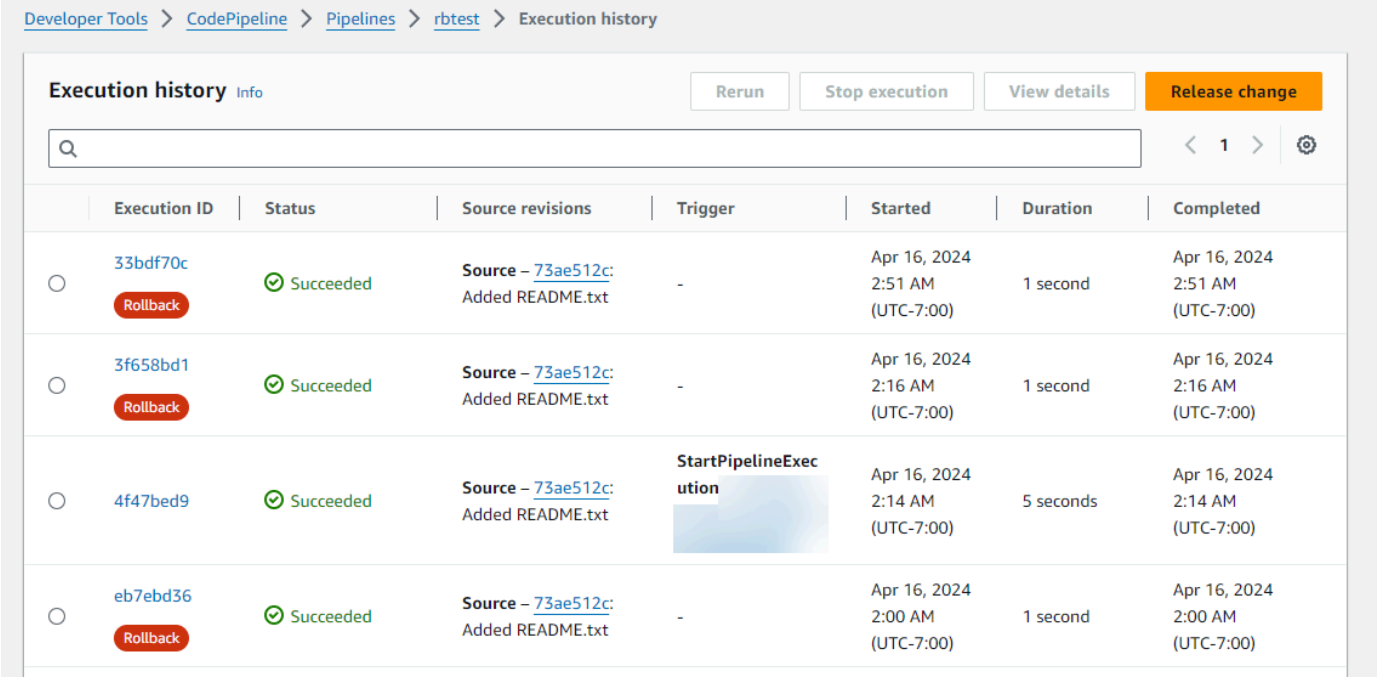
Execution history is retained for up to 12 months.

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**.

> **Note**
>
> For a pipeline in PARALLEL execution mode, the main pipeline view does not show the pipeline structure or in-progress executions. For a pipeline in PARALLEL execution mode, you access the pipeline structure by choosing the ID for the execution you want to view from the execution history page. Choose **History** in the left navigation, choose the execution ID for the parallel execution, and then view the pipeline on the **Visualization** tab.



4. View the status, source revisions, change details, and triggers related to each execution for your pipeline. Pipeline executions that have been rolled back will show the execution type **Rollback** on the details screen in the console. For the failed execution that triggered the automatic rollback, the failed execution ID is shown.
5. Choose an execution. The detail view shows execution details, the **Timeline** tab, the **Visualization** tab, and the **Variables** tab. Variable values for variables at the pipeline level are resolved at the time of pipeline execution and can be viewed in the execution history for each execution.

> **Note**
>
> Output variables from pipeline actions can be viewed on the Output variables tab under the history for each action execution.

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:

> **Note**
>
> The following pipeline states also apply to a pipeline execution that is an inbound execution. To view an inbound execution and its status, see View an inbound execution (Console).

Pipeline-level states

Pipeline state	Description
InProgress	The pipeline execution is currently running.
Stopping	The pipeline execution is stopping due to a request to either stop and wait or stop and abandon the pipeline execution.
Stopped	The stopping process is complete, and the pipeline execution is stopped.
Succeeded	The pipeline execution was completed successfully.
Superseded	While this pipeline execution was waiting for the next stage to be completed, a newer pipeline execution advanced and continued through the pipeline instead.
Failed	The pipeline execution was not completed successfully.

Stage-level states

Stage state	Description
InProgress	The stage is currently running.
Stopping	The stage execution is stopping due to a request to either stop and wait or stop and abandon the pipeline execution.
Stopped	The stopping process is complete, and the stage execution is stopped.
Succeeded	The stage was completed successfully.

Stage state	Description
Failed	The stage was not completed successfully.

Action-level states

Action state	Description
InProgress	The action is currently running.
Abandoned	The action is abandoned due to a request to stop and abandon the pipeline execution.
Succeeded	The action was completed successfully.
Failed	For approval actions, the FAILED state means the action was either rejected by the reviewer or failed due to an incorrect action configura tion.

View an inbound execution (Console)

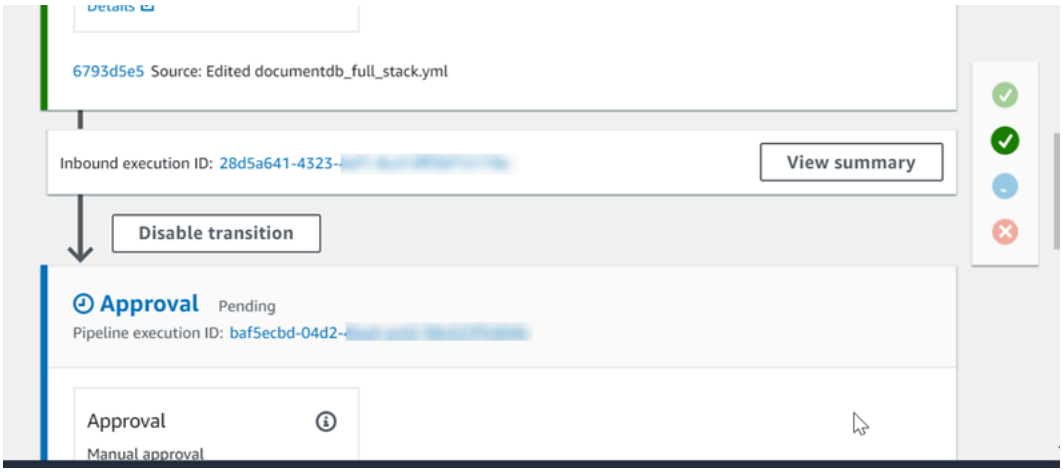
You can use the console to view the status and details for an inbound execution. When the transition is enabled or the stage becomes available, an inbound execution that is `InProgress` continues and enters the stage. An inbound execution with a `Stopped` status does not enter the stage. An inbound execution status changes to `Failed` if the pipeline is edited. When you edit a pipeline, all in-progress executions do not continue, and the execution status changes to `Failed`.

If you do not see an inbound execution, then there are no pending executions at a disabled stage transition.

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 will be displayed.
2. Choose the name of the pipeline for which you want to view the inbound execution, Do one of the following:

- Choose **View**. In the pipeline diagram, in the **Inbound execution ID** field in front of your disabled transition, you can view the inbound execution ID.



  Choose **View summary** to see execution details, such as the execution ID, source trigger, and the name of the next stage.

- Choose the pipeline and choose **View history**.

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:

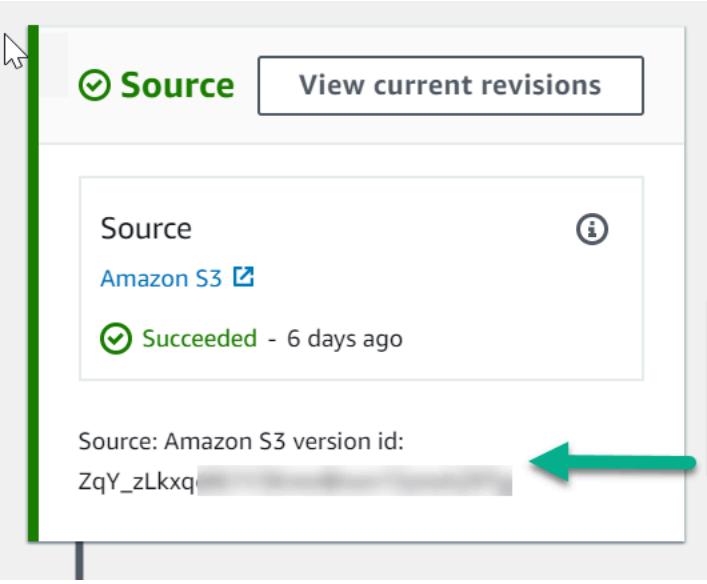
- **Summary**: Summary information about the most recent revision of the artifact. For GitHub and CodeCommit repositories, the commit message. For Amazon S3 buckets or actions, the user-provided content of a codepipeline-artifact-revision-summary key specified in the object metadata.
- **revisionUrl**: 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.

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

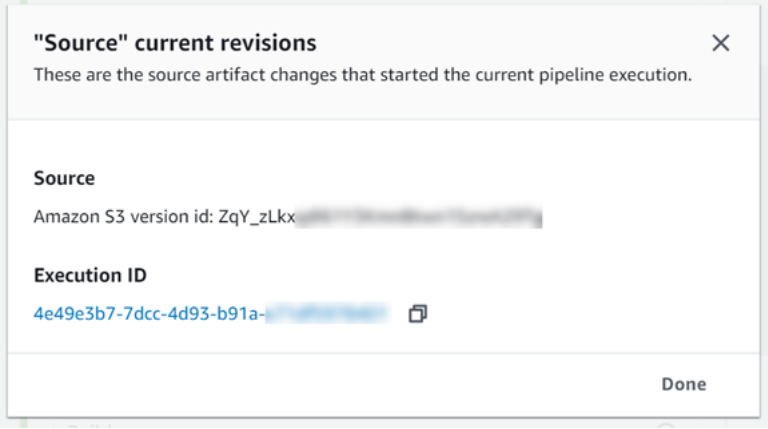
   The names of all pipelines associated with your AWS account will be displayed.

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.
   - Locate an action for which you want to view source revision details, and then find the revision information at the bottom of its stage:



   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.

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. 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.

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. 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, on the **Execution** tab, view the status and timing of the action execution.
6. On the **Configuration** tab, view the resource configuration for the action (for example, the CodeBuild build project name).
7. On the **Artifacts** tab, view the artifact details in **Artifact type** and **Artifact provider**. Choose the link under **Artifact name** to view the artifacts in the artifact store.
8. On the **Output variables** tab, view the resolved variables from actions in the pipeline for the action execution.

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 execution history with list-pipeline-executions (CLI)
- View pipeline state with get-pipeline-state (CLI)
- View inbound execution status with get-pipeline-state (CLI)
- View status and source revisions with get-pipeline-execution (CLI)
- View action executions with list-action-executions (CLI)

View execution history with `list-pipeline-executions` (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:

```
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.

Pipeline executions that have been rolled back will show the execution type `Rollback`. For the failed execution that triggered the automatic rollback, the failed execution ID is shown.

The following example shows the returned data for a pipeline named *MyFirstPipeline* that has had three executions:

```
{
    "pipelineExecutionSummaries": [
        {
            "pipelineExecutionId": "eb7ebd36-353a-4551-90dc-18ca5EXAMPLE",
            "status": "Succeeded",
            "startTime": "2024-04-16T09:00:28.185000+00:00",
            "lastUpdateTime": "2024-04-16T09:00:29.665000+00:00",
            "sourceRevisions": [
                {
                    "actionName": "Source",
                    "revisionId": "revision_ID",
                    "revisionSummary": "Added README.txt",
                    "revisionUrl": "console-URL"
                }
            ],
            "trigger": {
                "triggerType": "StartPipelineExecution",
                "triggerDetail": "trigger_ARN"
            },
            "executionMode": "SUPERSEDED"
        },
        {
            "pipelineExecutionId": "fcd61d8b-4532-4384-9da1-2aca1EXAMPLE",
            "status": "Succeeded",
            "startTime": "2024-04-16T08:58:56.601000+00:00",
            "lastUpdateTime": "2024-04-16T08:59:04.274000+00:00",
            "sourceRevisions": [
```

```
                {
                    "actionName": "Source",
                    "revisionId": "revision_ID",
                    "revisionSummary": "Added README.txt",
                    "revisionUrl": "console_URL"
                }
            ],
            "trigger": {
                "triggerType": "StartPipelineExecution",
                "triggerDetail": "trigger_ARN"
            },
            "executionMode": "SUPERSEDED"
        }
```

To view more details about a pipeline execution, run the **get-pipeline-execution**, 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 pipeline state with `get-pipeline-state` (CLI)

You can use the CLI to 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",
                      "entityUrl": "https://console.aws.amazon.com/s3/home?#",
                      "latestExecution": {
                          "status": "Succeeded",
                          "lastStatusChange": 1427298837.768
                      }
                  }
              ],
              "stageName": "Source"
          },
          {
  ```

```
            "actionStates": [
                {
                    "actionName": "Deploy-CodeDeploy-Application",
                    "entityUrl": "https://console.aws.amazon.com/codedeploy/home?
#",
                    "latestExecution": {
                        "status": "Succeeded",
                        "lastStatusChange": 1427298939.456,
                        "externalExecutionUrl": "https://console.aws.amazon.com/?
#",
                        "externalExecutionId": ""c53dbd42-This-Is-An-Example"",
                        "summary": "Deployment Succeeded"
                    }
                }
            ],
            "inboundTransitionState": {
                "enabled": true
            },
            "stageName": "Staging"
        },
        {
            "actionStates": [
                {
                    "actionName": "Deploy-Second-Deployment",
                    "entityUrl": "https://console.aws.amazon.com/codedeploy/home?
#",
                    "latestExecution": {
                        "status": "Failed",
                        "errorDetails": {
                            "message": "Deployment Group is already deploying
 deployment ...",
                            "code": "JobFailed"
                        },
                        "lastStatusChange": 1427246155.648
                    }
                }
            ],
            "inboundTransitionState": {
                "disabledReason": "Disabled while I investigate the failure",
                "enabled": false,
                "lastChangedAt": 1427246517.847,
                "lastChangedBy": "arn:aws:iam::80398EXAMPLE:user/CodePipelineUser"
            },
            "stageName": "Production"
```

```
        }
    ]
}
```

View inbound execution status with `get-pipeline-state` (CLI)

You can use the CLI to view inbound execution status. When the transition is enabled or the stage becomes available, an inbound execution that is `InProgress` continues and enters the stage. An inbound execution with a `Stopped` status does not enter the stage. An inbound execution status changes to `Failed` if the pipeline is edited. When you edit a pipeline, all in-progress executions do not continue, and the execution status changes to `Failed`.

- 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 output also shows pipeline execution ID in each stage, and whether there is an inbound execution ID for a stage with a disabled transition.

  The following example shows the returned data for a two-stage pipeline named *MyFirstPipeline*, where the first stage shows an enabled transition and a successful pipeline execution, and the second stage, named `Beta`, shows a disabled transition and an inbound execution ID. The inbound execution can have an `InProgress`, `Stopped`, or `FAILED` state.

  ```
  {
      "pipelineName": "MyFirstPipeline",
      "pipelineVersion": 2,
      "stageStates": [
          {
              "stageName": "Source",
              "inboundTransitionState": {
                  "enabled": true
              },
              "actionStates": [
                  {
  ```

```
                    "actionName": "SourceAction",
                    "currentRevision": {
                        "revisionId": "PARcnxX_u0SMRBnKh83pHL09.zPRLLMu"
                    },
                    "latestExecution": {
                        "actionExecutionId": "14c8b311-0e34-4bda-EXAMPLE",
                        "status": "Succeeded",
                        "summary": "Amazon S3 version id: PARcnxX_u0EXAMPLE",
                        "lastStatusChange": 1586273484.137,
                        "externalExecutionId": "PARcnxX_u0EXAMPLE"
                    },
                    "entityUrl": "https://console.aws.amazon.com/s3/home?#"
                }
            ],
            "latestExecution": {
                "pipelineExecutionId": "27a47e06-6644-42aa-EXAMPLE",
                "status": "Succeeded"
            }
        },
        {
            "stageName": "Beta",
            "inboundExecution": {
                "pipelineExecutionId": "27a47e06-6644-42aa-958a-EXAMPLE",
                "status": "InProgress"
            },
            "inboundTransitionState": {
                "enabled": false,
                "lastChangedBy": "USER_ARN",
                "lastChangedAt": 1586273583.949,
                "disabledReason": "disabled"
            },
                    "currentRevision": {
            "actionStates": [
                {
                    "actionName": "BetaAction",
                    "latestExecution": {
                        "actionExecutionId": "a748f4bf-0b52-4024-98cf-EXAMPLE",
                        "status": "Succeeded",
                        "summary": "Deployment Succeeded",
                        "lastStatusChange": 1586272707.343,
                        "externalExecutionId": "d-KFGF3EXAMPLE",
                        "externalExecutionUrl": "https://us-
west-2.console.aws.amazon.com/codedeploy/home?#/deployments/d-KFGF3WTS2"
                    },
```

```
                    "entityUrl": "https://us-west-2.console.aws.amazon.com/
codedeploy/home?#/applications/my-application"
                }
            ],
            "latestExecution": {
                "pipelineExecutionId": "f6bf1671-d706-4b28-EXAMPLE",
                "status": "Succeeded"
            }
        }
    ],
    "created": 1585622700.512,
    "updated": 1586273472.662
}
```

View status and source revisions with `get-pipeline-execution` (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 status and 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:

   ```
   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:

   ```
   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. ```
   {
       "pipelineExecution": {
           "artifactRevisions": [
               {
                   "created": 1427298837.7689769,
                   "name": "MyApp",
                   "revisionChangeIdentifier": "1427298921.3976923",
                   "revisionId": "7636d59f3c461cEXAMPLE8417dbc6371",
   ```
   ```

```
                "revisionSummary": "Updating the application for feature 12-4820",
                "revisionUrl": "https://api.github.com/repos/anycompany/MyApp/git/
commits/7636d59f3c461cEXAMPLE8417dbc6371"
            }
        ],
        "pipelineExecutionId": "3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE",
        "pipelineName": "MyFirstPipeline",
        "pipelineVersion": 2,
        "status": "Succeeded",
        "executionMode": "SUPERSEDED",
        "executionType": "ROLLBACK",
        "rollbackMetadata": {
            "rollbackTargetPipelineExecutionId": "4f47bed9-6998-476c-a49d-
e60beEXAMPLE"
         }
    }
}
```

View action executions with `list-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:

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

    The following shows a portion of sample output for this command:

    ```
    {
    ```

```
    "actionExecutionDetails": [
        {
            "actionExecutionId": "ID",
            "lastUpdateTime": 1552958312.034,
            "startTime": 1552958246.542,
            "pipelineExecutionId": "Execution_ID",
            "actionName": "Build",
            "status": "Failed",
            "output": {
                "executionResult": {
                    "externalExecutionUrl": "Project_ID",
                    "externalExecutionSummary": "Build terminated with state:
 FAILED",
                    "externalExecutionId": "ID"
                },
                "outputArtifacts": []
            },
            "stageName": "Beta",
            "pipelineVersion": 8,
            "input": {
                "configuration": {
                    "ProjectName": "java-project"
                },
                "region": "us-east-1",
                "inputArtifacts": [
                    {
                        "s3location": {
                            "bucket": "codepipeline-us-east-1-ID",
                            "key": "MyFirstPipeline/MyApp/Object.zip"
                        },
                        "name": "MyApp"
                    }
                ],
                "actionTypeId": {
                    "version": "1",
                    "category": "Build",
                    "owner": "AWS",
                    "provider": "CodeBuild"
                }
            }
        },

. . .
```

- 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"
                    }
                ]
            },


. . .
```

Set or change the pipeline execution mode

You can set the execution mode for your pipeline to specify how multiple executions are handled.

For more information about pipeline execution modes, see How pipeline executions work.

Important

For pipelines in PARALLEL mode, when editing the pipeline execution mode to QUEUED or SUPERSEDED, the pipeline state will not display the updated state as PARALLEL. For more information, see Pipelines changed from PARALLEL mode will display a previous execution mode.

Important

For pipelines in PARALLEL mode, when editing the pipeline execution mode to QUEUED or SUPERSEDED, the pipeline definition for the pipeline in each mode will not be updated. For more information, see Pipelines in PARALLEL mode have an outdated pipeline definition if edited when changing to QUEUED or SUPERSEDED mode.

Important

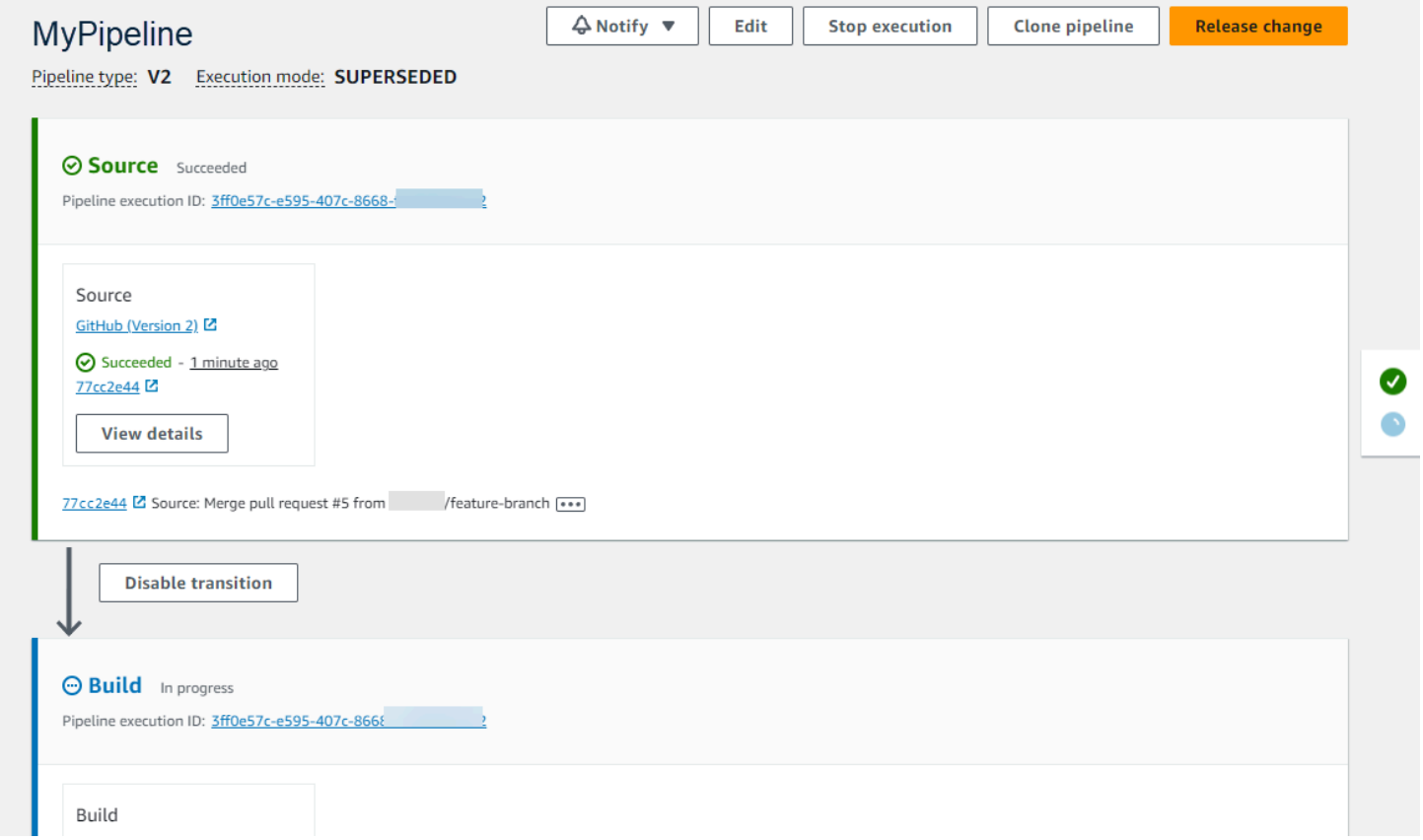
For pipelines in PARALLEL mode, stage rollback is not available. Similarly, failure conditions with a rollback result type cannot be added to a PARALLEL mode pipeline.

Considerations for viewing execution modes

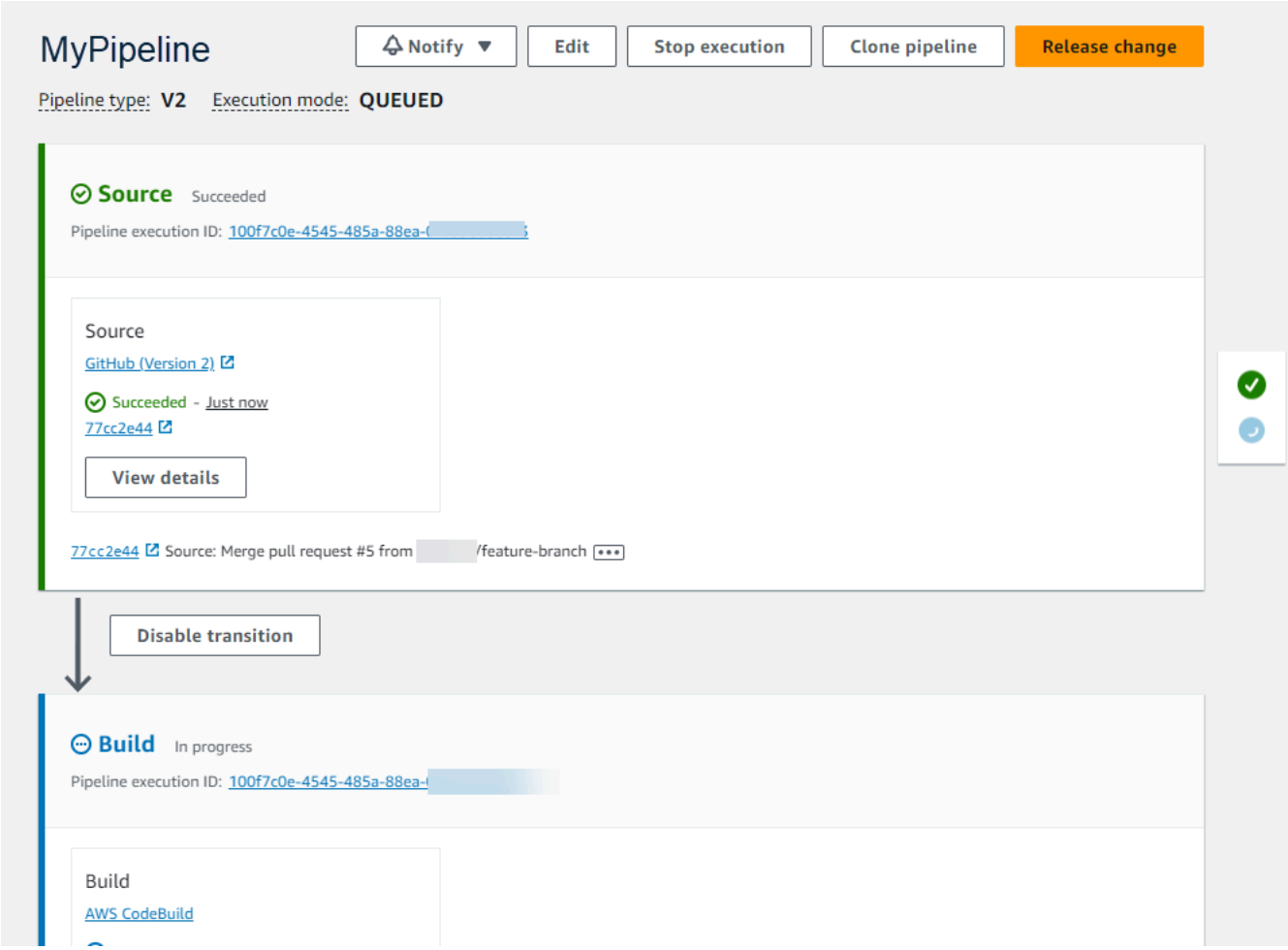
There are considerations for viewing pipelines in specific execution modes.

For SUPERSEDED and QUEUED modes, use the pipeline view to see in-progress executions, and click the execution ID to view details and history. For PARALLEL mode, click the execution ID to view the in-progress execution on the Visualization tab.

The following shows the view for SUPERSEDED mode in CodePipeline.

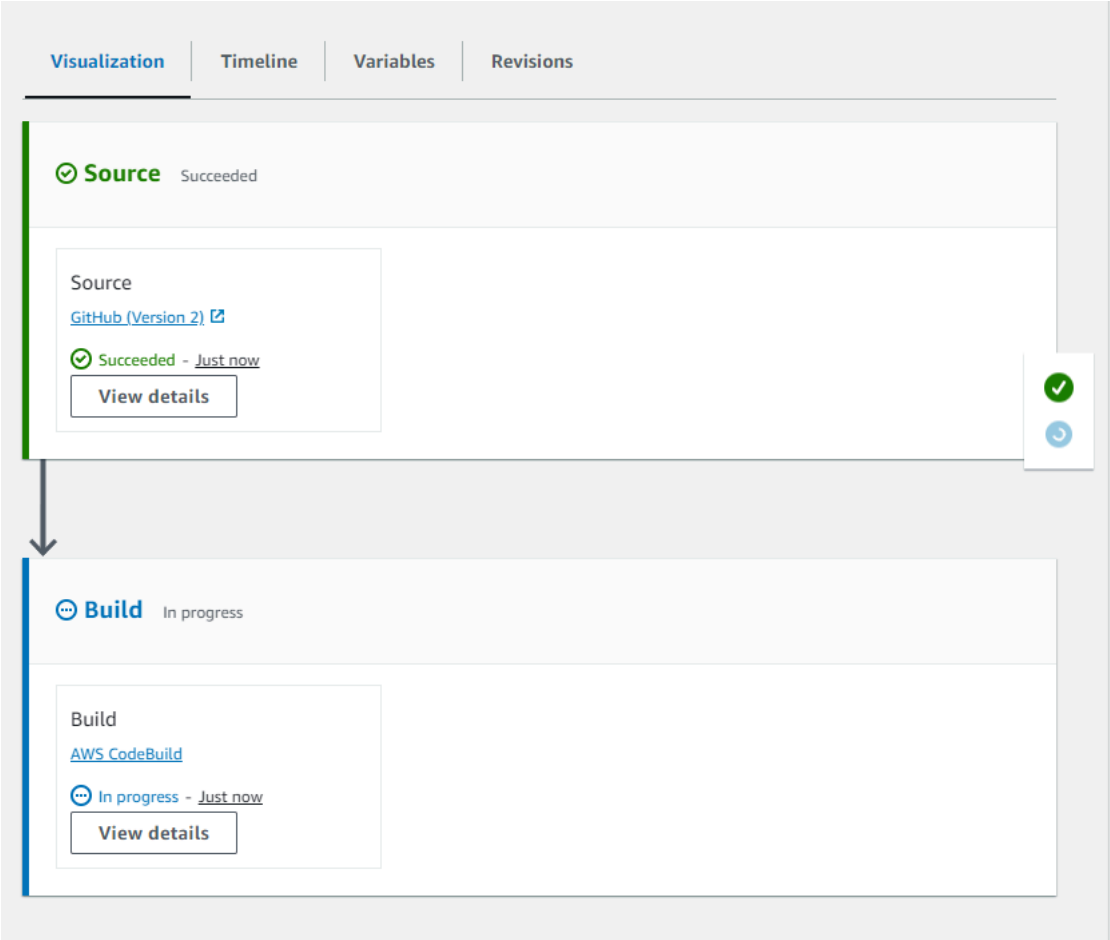


The following shows the view for QUEUED mode in CodePipeline.



The following shows the view for PARALLEL mode in CodePipeline.

> **Important**
>
> For pipelines in PARALLEL mode, stage rollback is not available. Similarly, failure conditions with a rollback result type cannot be added to a PARALLEL mode pipeline.



Considerations for switching between execution modes

The following are considerations for pipelines when changing the mode for the pipeline. When switching from one execution mode to another in **Edit** mode and then saving the change, certain views or states might adjust.

For example, when switching from PARALLEL mode to a QUEUED or SUPERSEDED mode, the execution started in PARALLEL mode will continue to run. These can be viewed on the execution history page. The pipeline view will show the execution that ran on QUEUED or SUPERSEDED mode earlier or an empty state otherwise.

As another example, when switching from QUEUED or SUPERSEDED to PARALLEL mode, you will no longer see the pipeline view/state page. To view an execution in PARALLEL mode, use the visualization tab on the execution details page. Executions started in SUPERSEDED or QUEUED mode will be canceled.

The following table provides more detail.

Mode change	Pending and active execution details	Pipeline state details
SUPERSEDED to SUPERSEDED / SUPERSEDED to QUEUED	• Active executions are canceled after in-progress actions complete. • Pending executions are canceled.	The pipeline state, such as **canceled**, is preserved between the version of the first mode and the second mode.
QUEUED to QUEUED / QUEUED to SUPERSEDED	• Active executions are canceled after in-progress actions complete. • Pending executions are canceled.	The pipeline state, such as canceled, is preserved between the version of the first mode and the second mode.
PARALLEL to PARALLEL	All executions are allowed to run independently of pipeline definition updates.	Empty. Parallel mode does not have a pipeline state.
SUPERSEDED to PARALLEL / QUEUED to PARALLEL	• Active executions are canceled after in-progress actions complete. • Pending executions are canceled.	Empty. Parallel mode does not have a pipeline state.

Set or change the pipeline execution mode (console)

You can use the console to set the pipeline execution mode.

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 edit.
3. On the pipeline details page, choose **Edit**.
4. On the **Edit** page, choose **Edit: Pipeline properties**.

5. Choose the mode for your pipeline.

   - **Superseded**
   - **Queued (Pipeline type V2 required)**
   - **Parallel (Pipeline type V2 required)**

6. On the **Edit** page, choose **Done**.

Set the pipeline execution mode (CLI)

To use the AWS CLI to set the pipeline execution mode, use the `create-pipeline` or `update-pipeline` command.

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:

   ```
   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 pipeline execution mode you want to set, such as QUEUED.

   ```
   "executionMode": "QUEUED"
   ```

   The following example shows how you would set the execution mode to QUEUED in an example pipeline with two stages.

   ```
   {
       "pipeline": {
           "name": "MyPipeline",
           "roleArn": "arn:aws:iam::111122223333:role/service-role/
   AWSCodePipelineServiceRole-us-east-1-dkpippe",
           "artifactStore": {
               "type": "S3",
               "location": "bucket"
           },
           "stages": [
   ```

```
            {
                "name": "Source",
                "actions": [
                    {
                        "name": "Source",
                        "actionTypeId": {
                            "category": "Source",
                            "owner": "AWS",
                            "provider": "CodeCommit",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "BranchName": "main",
                            "OutputArtifactFormat": "CODE_ZIP",
                            "PollForSourceChanges": "true",
                            "RepositoryName": "MyDemoRepo"
                        },
                        "outputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "inputArtifacts": [],
                        "region": "us-east-1",
                        "namespace": "SourceVariables"
                    }
                ]
            },
            {
                "name": "Build",
                "actions": [
                    {
                        "name": "Build",
                        "actionTypeId": {
                            "category": "Build",
                            "owner": "AWS",
                            "provider": "CodeBuild",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "ProjectName": "MyBuildProject"
                        },
```

```
                        "outputArtifacts": [
                            {
                                "name": "BuildArtifact"
                            }
                        ],
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "region": "us-east-1",
                        "namespace": "BuildVariables"
                    }
                ]
            }
        ],
        "version": 1,
        "executionMode": "QUEUED"
    }
}
```

3. 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.

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 start the pipeline manually to run that revision through the updated pipeline.

Roll back or retry stages

You can use the AWS CodePipeline console or the AWS CLI to manually roll back or retry a stage or actions in a stage. To configure conditions for stages where you want to use Rollback or Retry as the configured result, see Configure conditions for a stage.

Topics

- Configuring stage retry for a failed stage or failed actions
- Configuring stage rollback

Configuring stage retry for a failed stage or failed actions

You can retry a stage that has failed without having to run a pipeline again from the beginning. You do this by either retrying the failed actions in a stage or by retrying all actions in the stage starting from the first action in the stage. When you retry the failed actions in a stage, all actions that are still in progress continue working, and failed actions are triggered again. When you retry a failed stage from the first action in the stage, the stage cannot have any actions in progress. Before a stage can be retried, it must either have all actions failed or some actions failed and some succeeded.

> **Important**
>
> Retrying a failed stage retries all actions in the stage from the first action in the stage, and retrying failed actions retries all failed actions in the stage. This overrides output artifacts of previously successful actions in the same execution.
> Although artifacts may be overridden, the execution history of previously successful actions is still retained.

If you are using the console to view a pipeline, either a **Retry stage** button or a **Retry failed actions** button appears on the stage that 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 might not be able to retry a stage:

- All actions in the stage succeeded, and so the stage is not in a failed status.
- The overall pipeline structure changed after the stage failed.
- Another retry attempt in the stage is already in progress.

Topics

- Considerations for stage retry
- Retry a failed stage manually
- Configure a stage for automatic retry on failure

Considerations for stage retry

Considerations for stage retry are as follows:

- You can only configure automatic retry on stage failure for one retry attempt.
- You can configure automatic retry on stage failure for all actions, including `Source` actions.

Retry a failed stage manually

You can manually retry a failed stage using the console or CLI.

You can also configure a stage for retry automatically on failure as detailed in Configure a stage for automatic retry on failure.

Retry a failed stage manually (console)

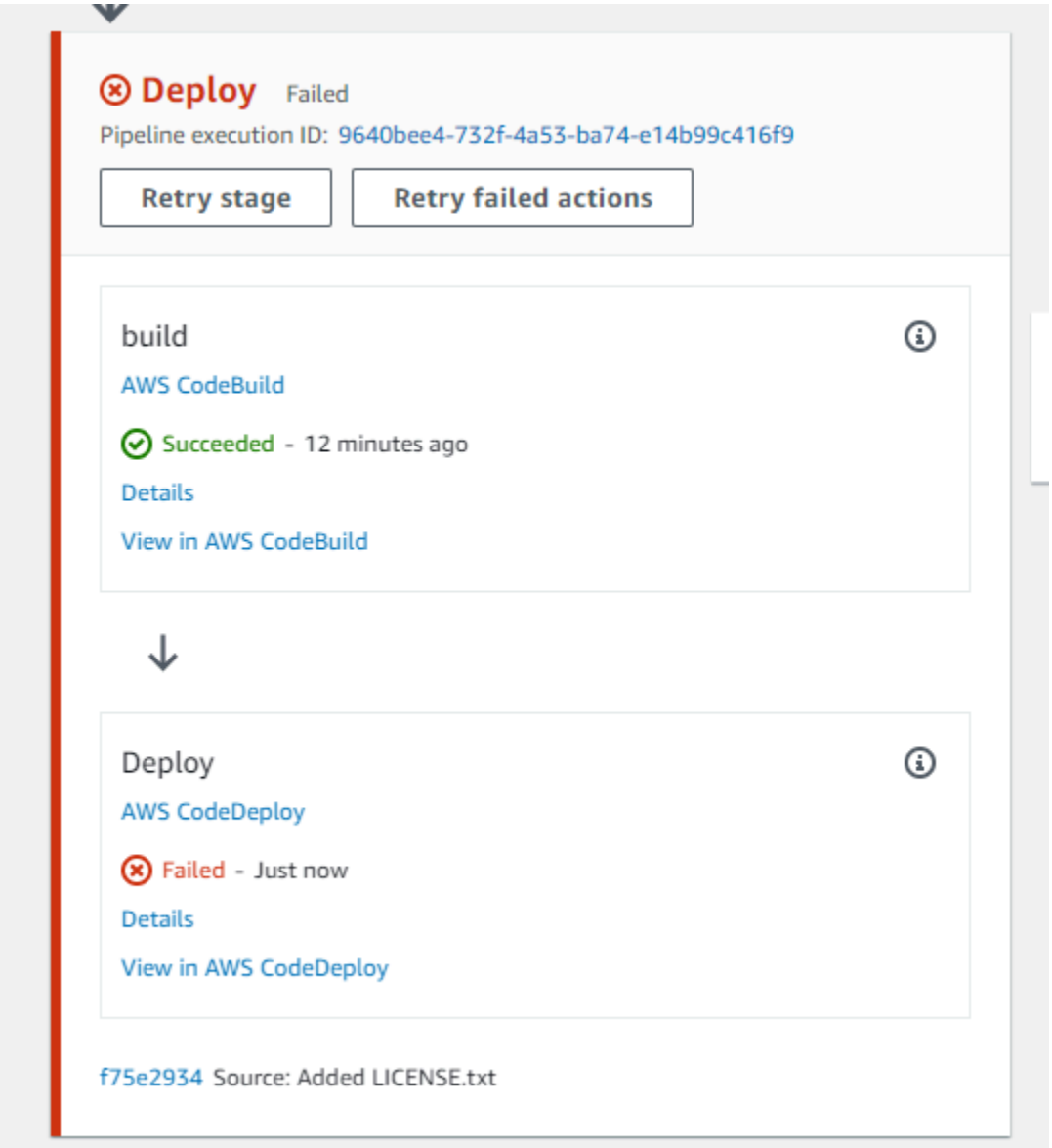
To retry a failed stage or failed actions in a stage - console

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.

3. Locate the stage with the failed action, and then choose one of the following:

   - To retry all actions in the stage, choose **Retry stage**.
   - To retry only failed actions in the stage, choose **Retry failed actions**.



If all retried actions in the stage are completed successfully, the pipeline continues to run.

Retry a failed stage manually (CLI)

To retry a failed stage or failed actions in a stage - CLI

To use the AWS CLI to retry all actions or all 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 ALL_ACTIONS/FAILED_ACTIONS
```

> **Note**
>
> The values you can use for `retry-mode` are `FAILED_ACTIONS` and `ALL_ACTIONS`.

1. 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"
   }
   ```

2. 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.

   a. 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:

```
{
    "updated": 1427245911.525,
    "created": 1427245911.525,
    "pipelineVersion": 1,
    "pipelineName": "MyFirstPipeline",
    "stageStates": [
        {
            "actionStates": [...],
            "stageName": "Source",
            "latestExecution": {
                "pipelineExecutionId": "9811f7cb-7cf7-SUCCESS",
                "status": "Succeeded"
            }
        },
        {
            "actionStates": [...],
            "stageName": "Staging",
            "latestExecution": {
                "pipelineExecutionId": "3137f7cb-7cf7-EXAMPLE",
                "status": "Failed"
            }
        }
    ]
}
```

b. 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.

For the preceding MyFirstPipeline example, your file would look something like this:

```
{
    "pipelineName": "MyFirstPipeline",
    "stageName": "Staging",
    "pipelineExecutionId": "3137f7cb-7cf7-EXAMPLE",
    "retryMode": "FAILED_ACTIONS"
}
```

c. Save the file with a name like **`retry-failed-actions.json`**.

d. Call the file you created when you run the **retry-stage-execution** command. For example:

> **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
```

e. 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 pipelines and details in CodePipeline.

Configure a stage for automatic retry on failure

You can configure a stage for automatic retry on failure. The stage will make one retry attempt and display a retry status on the failed stage on the **View** pipeline page.

You can configure the retry mode by specifying that the stage should automatically retry either all actions in the failed stage or only failed actions in the stage.

Configure a stage for automatic retry on failure (console)

You can use the console to configure a stage for automatic retry.

Configure a stage for automatic retry (console)

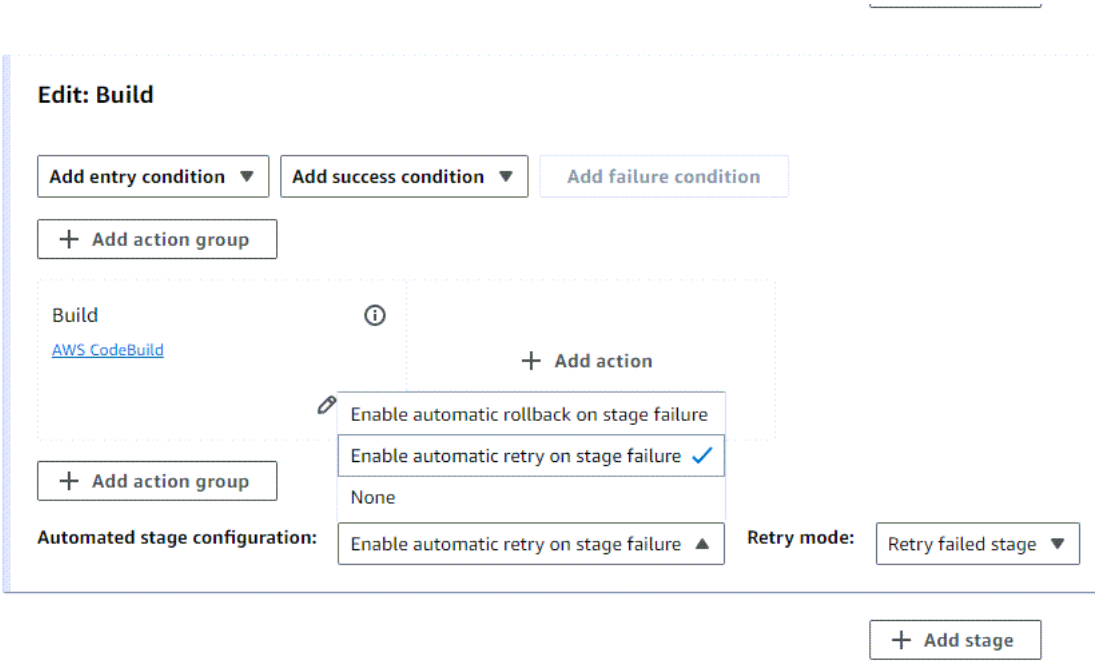
1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home.

The names and status of all pipelines associated with your AWS account are displayed.

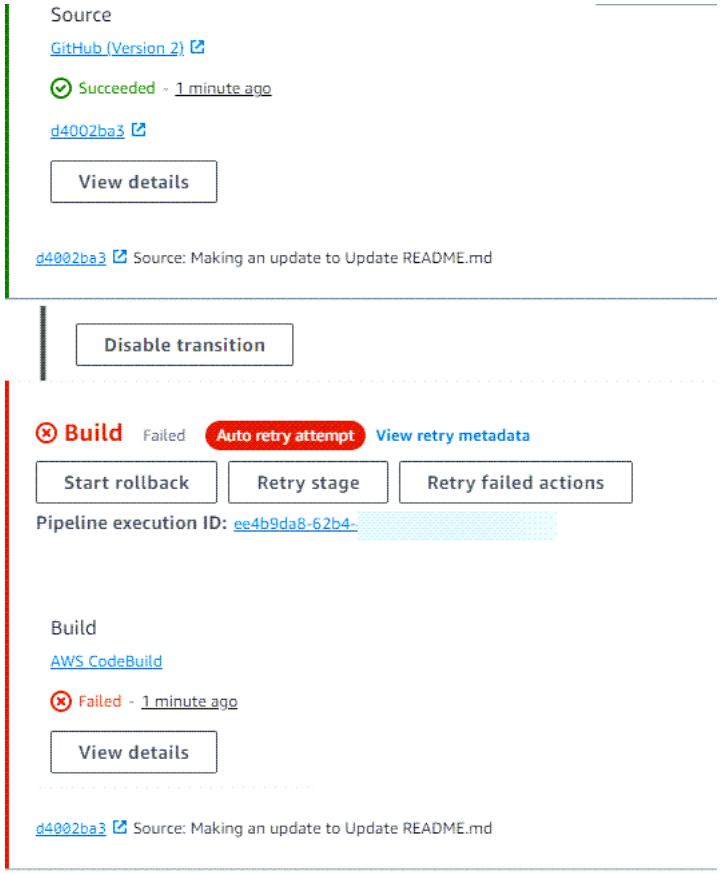
2. In **Name**, choose the name of the pipeline you want to edit.
3. On the pipeline details page, choose **Edit**.
4. On the **Edit** page, for the action you want to edit, choose **Edit stage**.
5. Choose **Automated stage configuration:**, and then choose **Enable automatic retry on stage failure**. Save the changes to your pipeline.
6. In **Automated stage configuration:**, choose one of the following retry modes:

   - To specify that the mode will retry all actions in the stage, choose **Retry failed stage**.
   - To specify that the mode will only retry failed actions in the stage, choose **Retry failed actions**.

   Save the changes to your pipeline.



7. After your pipeline runs, if the stage failure occurs, the automatic retry attempt will be made. The following examples show a build stage that has been retried automatically.



8. To view details about the retry attempt, choose . The window displays.

Configure a stage for automatic retry (CLI)

To use the AWS CLI to configure a stage to automatically retry on failure, use the commands to create or update a pipeline as detailed in Create a pipeline, stages, and actions and Edit a pipeline in CodePipeline.

- Open a terminal (Linux, macOS, or Unix) or command prompt (Windows) and use the AWS CLI to run the `update-pipeline` command, specifying the failure condition in the pipeline structure. The following example configures automatic retry for a stage named `S3Deploy`:

```
{
                "name": "S3Deploy",
                "actions": [
                    {
                        "name": "s3deployaction",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "provider": "S3",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "BucketName": "static-website-bucket",
                            "Extract": "false",
                            "ObjectKey": "SampleApp.zip"
                        },
                        "outputArtifacts": [],
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "region": "us-east-1"
                    }
                ],
                 "onFailure": {
                    "result": "RETRY",
                    "retryConfiguration": {
                        "retryMode": "ALL_ACTIONS",
                    },
            }
```

Configure a stage for automatic retry (CloudFormation)

To use CloudFormation to configure a stage for automatic retry on failure, use the `OnFailure` stage lifecycle parameter. Use the `RetryConfiguration` parameter to configure the retry mode.

```
OnFailure:
     Result: RETRY
     RetryConfiguration:
         RetryMode: ALL_ACTIONS
```

- Update the template as shown in the following snippet. The following example configures automatic retry for a stage named `Release`:

```
AppPipeline:
  Type: AWS::CodePipeline::Pipeline
  Properties:
    RoleArn:
      Ref: CodePipelineServiceRole
    Stages:
      -
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
              -
                Name: SourceOutput
            Configuration:
              S3Bucket:
                Ref: SourceS3Bucket
              S3ObjectKey:
                Ref: SourceS3ObjectKey
            RunOrder: 1
      -
        Name: Release
        Actions:
          -
            Name: ReleaseAction
            InputArtifacts:
              -
                Name: SourceOutput
            ActionTypeId:
```

```
            Category: Deploy
            Owner: AWS
            Version: 1
            Provider: CodeDeploy
          Configuration:
            ApplicationName:
              Ref: ApplicationName
            DeploymentGroupName:
              Ref: DeploymentGroupName
          RunOrder: 1
     OnFailure:
         Result: RETRY
         RetryConfiguration:
            RetryMode: ALL_ACTIONS
  ArtifactStore:
    Type: S3
    Location:
      Ref: ArtifactStoreS3Location
    EncryptionKey:
      Id: arn:aws:kms:useast-1:ACCOUNT-ID:key/KEY-ID
      Type: KMS
  DisableInboundStageTransitions:
    -
      StageName: Release
      Reason: "Disabling the transition until integration tests are completed"
  Tags:
    - Key: Project
      Value: ProjectA
    - Key: IsContainerBased
      Value: 'true'
```

For more information about configuring stage retry on failure, see OnFailure under `StageDeclaration` in the *CloudFormation User Guide*.

Configuring stage rollback

You can roll back a stage to an execution that was successful in that stage. You can preconfigure a stage for rollback on failure, or you can manually roll back a stage. The rolled back operation will result in a new execution. The target pipeline execution chosen for rollback is used to retrieve source revisions and variables.

The type of execution, either standard or rollback, displays in the pipeline history, pipeline state, and pipeline execution details.

Topics

- Considerations for rollbacks
- Roll back a stage manually
- Configure a stage for automatic rollback
- View rollback status in execution listing
- View rollback status details

Considerations for rollbacks

Considerations for stage rollback are as follows:

- You cannot roll back a source stage.
- The pipeline can only roll back to a previous execution if the previous execution was started in the current pipeline structure version.
- You cannot roll back to a target execution ID that is a rollback execution type.
- CodePipeline will use the variables and artifacts from the execution to which it is rolling back.

Roll back a stage manually

You can manually roll back a stage using the console or CLI. The pipeline can only roll back to a previous execution if the previous execution was started in the current pipeline structure version.

You can also configure a stage to roll back automatically on failure as detailed in Configure a stage for automatic rollback.

Roll back a stage manually (console)

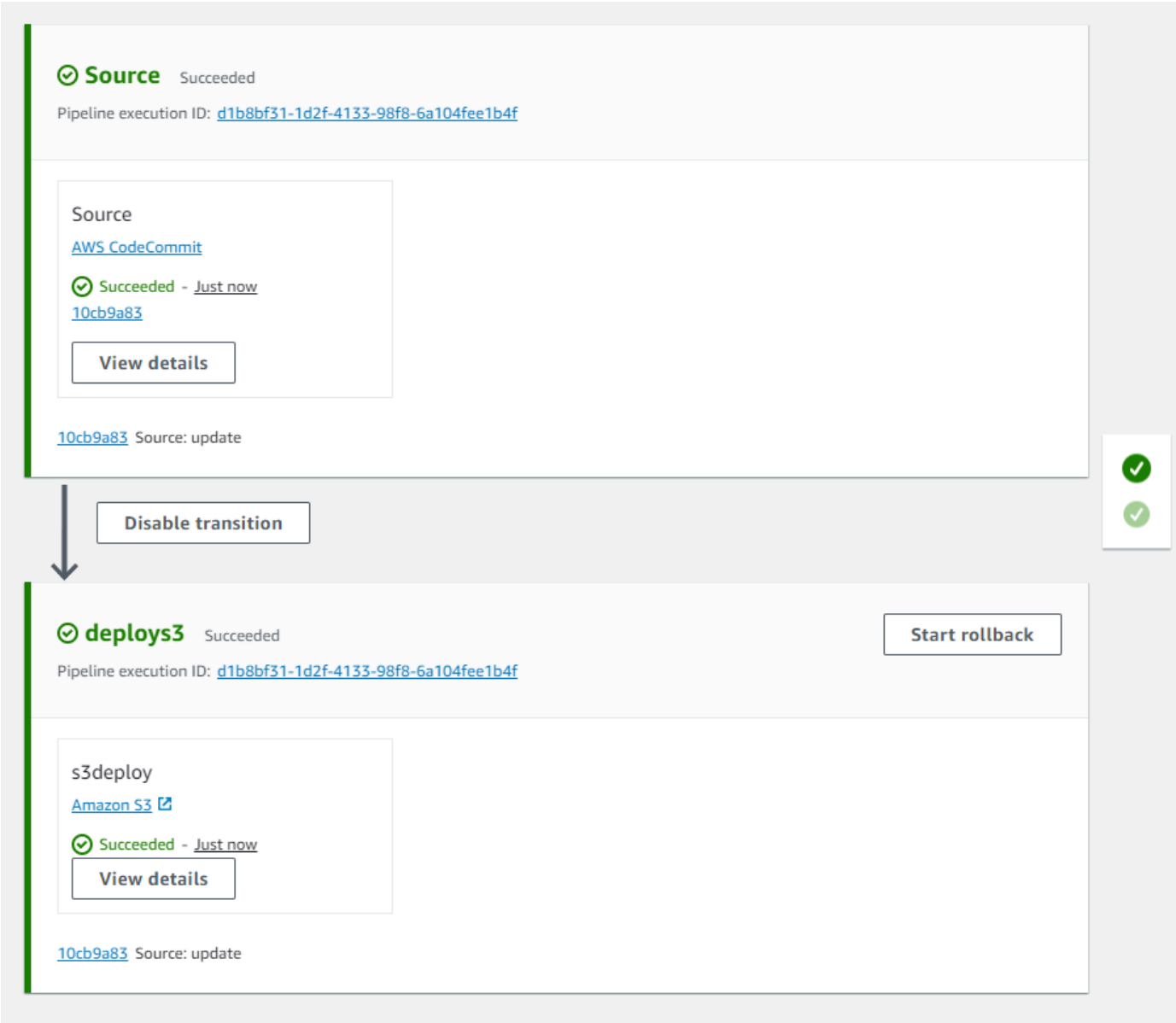
You can use the console to manually roll back a stage to a target pipeline execution. When a stage is rolled back, a **Rollback** label displays on the pipeline visualization in the console.

Roll back a stage manually (console)

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

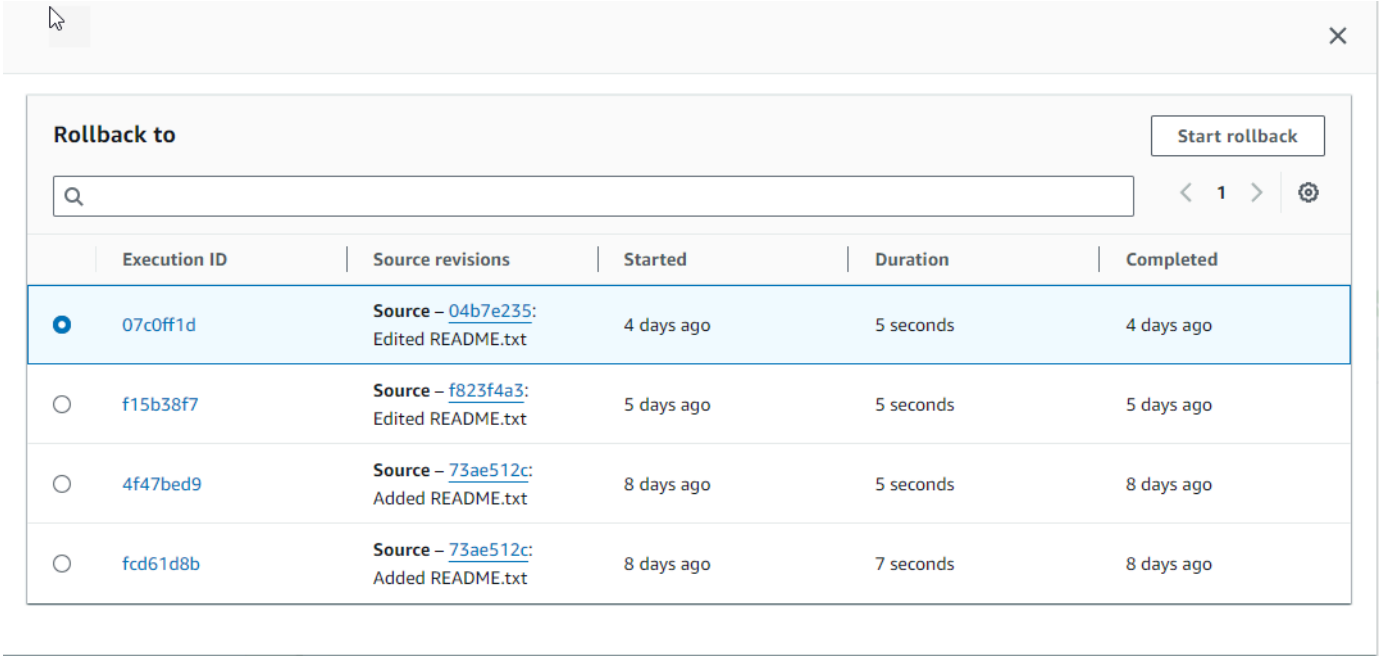
The names and status of all pipelines associated with your AWS account are displayed.

2. In **Name**, choose the name of the pipeline with the stage to roll back.

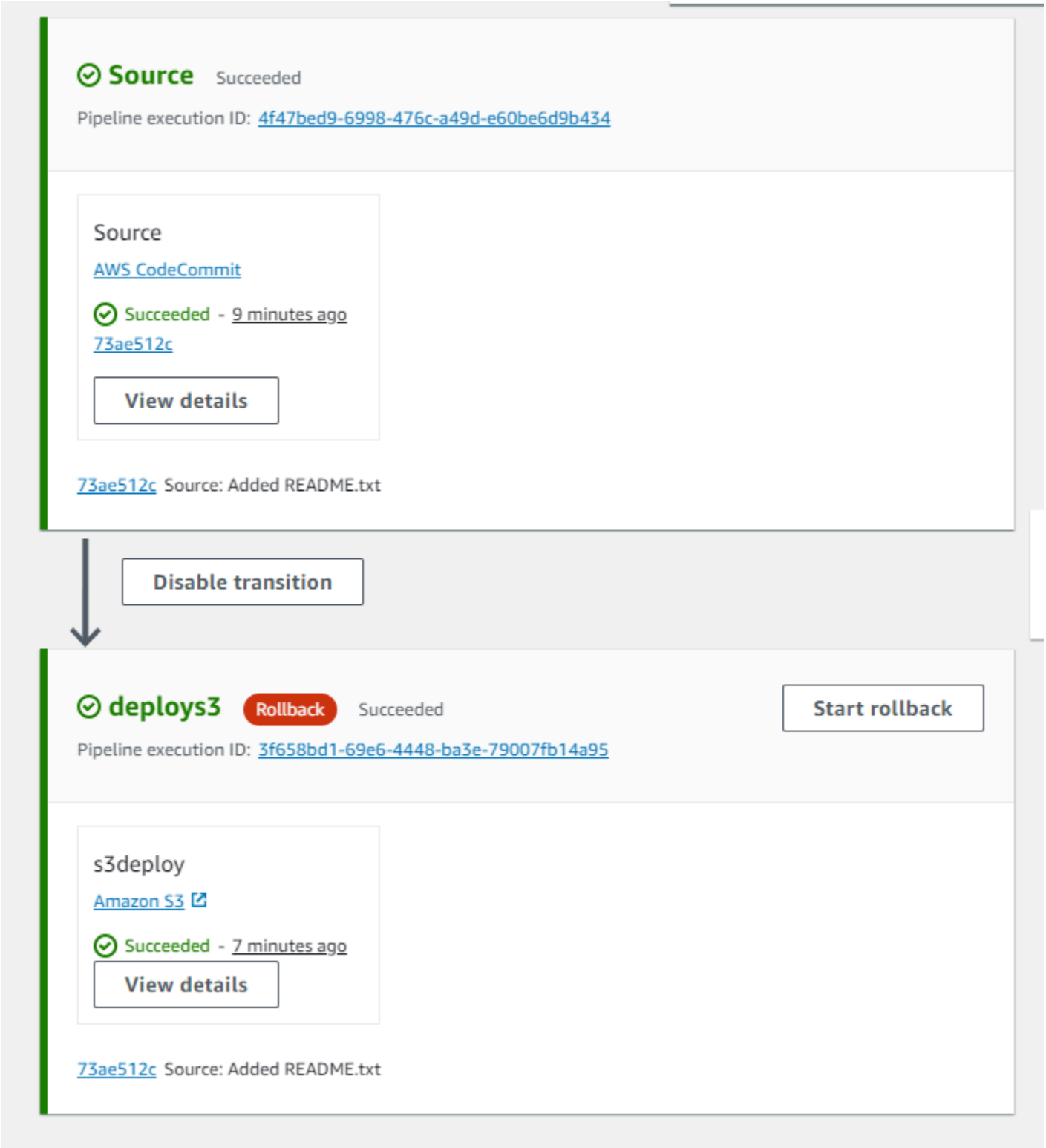


3. On the stage, choose **Start rollback**. The **Roll back to** page displays.
4. Choose the target execution to which you want to roll back the stage.

> **Note**
>
> The list of target pipeline executions available will be all executions in the current pipeline version beginning on February 1, 2024.



The following diagram shows an example of the rolled back stage with the new execution ID.



Roll back a stage manually (CLI)

To use the AWS CLI to manually roll back a stage, use the `rollback-stage` command.

You can also roll back a stage manually as detailed in Roll back a stage manually.

> **Note**
>
> The list of target pipeline executions available will be all executions in the current pipeline version beginning on February 1, 2024.

To roll back a stage manually (CLI)

1. The CLI command for manual rollback will require the execution ID of a previously successful pipeline execution in the stage. To get the target pipeline execution ID that you will specify, use the list-pipeline-executions command with a filter that will return successful executions in the stage. Open a terminal (Linux, macOS, or Unix) or command prompt (Windows) and use the AWS CLI to run the `list-pipeline-executions` command, specifying the name of the pipeline and the filter for successful executions in the stage. In this example, the output will list pipeline executions for the pipeline named MyFirstPipeline and for successful executions in the stage named `deploys3`.

   ```
   aws codepipeline list-pipeline-executions --pipeline-name MyFirstPipeline --filter
    succeededInStage={stageName=deploys3}
   ```

   In the output, copy the execution ID of the previously successful execution that you want to specify for rollback. You will use this in the next step as the target execution ID.

2. Open a terminal (Linux, macOS, or Unix) or command prompt (Windows) and use the AWS CLI to run the `rollback-stage` command, specifying the name of the pipeline, the name of the stage, and the target execution that you want to roll back to. For example, to roll back a stage named Deploy for a pipeline named *`MyFirstPipeline`*:

   ```
   aws codepipeline rollback-stage --pipeline-name MyFirstPipeline --stage-name Deploy
    --target-pipeline-execution-id bc022580-4193-491b-8923-9728dEXAMPLE
   ```

   The output returns the execution ID for the new rolled-back execution. This is a separate ID that uses the source revisions and parameters of the specified target execution.

Configure a stage for automatic rollback

You can configure stages in a pipeline to roll back automatically on failure. When the stage fails, the stage is rolled back to the most recent successful execution. The pipeline can only roll back to a previous execution if the previous execution was started in the current pipeline structure version. Since automatic rollback configuration is part of the pipeline definition, your pipeline stage will auto-rollback only after there is a successful pipeline execution in the pipeline stage.

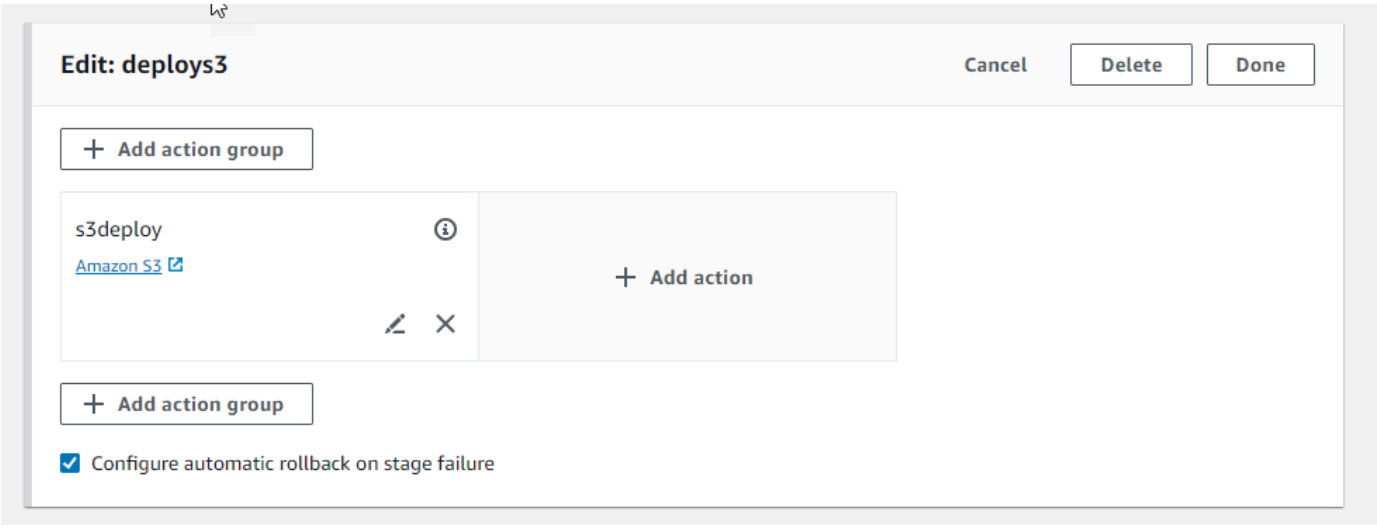
Configure a stage for automatic rollback (console)

You can roll back a stage to a specified previous successful execution. For more information, see RollbackStage in the *CodePipeline API Guide*.

Configure a stage for automatic rollback (console)

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 edit.
3. On the pipeline details page, choose **Edit**.
4. On the **Edit** page, for the action you want to edit, choose **Edit stage**.
5. Choose **Automated stage configuration:**, and then choose **Configure automatic rollback on stage failure**. Save the changes to your pipeline.



Configure a stage for automatic rollback (CLI)

To use the AWS CLI to configure a failed stage to automatically roll back to the most recent successful execution, use the commands to create or update a pipeline as detailed in Create a pipeline, stages, and actions and Edit a pipeline in CodePipeline.

- Open a terminal (Linux, macOS, or Unix) or command prompt (Windows) and use the AWS CLI to run the `update-pipeline` command, specifying the failure condition in the pipeline structure. The following example configures automatic rollback for a stage named `S3Deploy`:

```
{
                "name": "S3Deploy",
                "actions": [
                    {
                        "name": "s3deployaction",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "provider": "S3",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "BucketName": "static-website-bucket",
                            "Extract": "false",
                            "ObjectKey": "SampleApp.zip"
                        },
                        "outputArtifacts": [],
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "region": "us-east-1"
                    }
                ],
                "onFailure": {
                    "result": "ROLLBACK"
                }
            }
```

For more information about configuring failure conditions for stage rollback, see FailureConditions in the *CodePipeline API Reference*.

Configure a stage for automatic rollback (CloudFormation)

To use CloudFormation to configure a stage to roll back automatically on failure, use the `OnFailure` parameter. On failure, the stage will automatically roll back to the most recent successful execution.

```
OnFailure:
     Result: ROLLBACK
```

- Update the template as shown in the following snippet. The following example configures automatic rollback for a stage named `Release`:

```
AppPipeline:
  Type: AWS::CodePipeline::Pipeline
  Properties:
    RoleArn:
      Ref: CodePipelineServiceRole
    Stages:
      -
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
              -
                Name: SourceOutput
            Configuration:
              S3Bucket:
                Ref: SourceS3Bucket
              S3ObjectKey:
                Ref: SourceS3ObjectKey
            RunOrder: 1
```

```
      -
        Name: Release
        Actions:
          -
            Name: ReleaseAction
            InputArtifacts:
              -
                Name: SourceOutput
            ActionTypeId:
              Category: Deploy
              Owner: AWS
              Version: 1
              Provider: CodeDeploy
            Configuration:
              ApplicationName:
                Ref: ApplicationName
              DeploymentGroupName:
                Ref: DeploymentGroupName
            RunOrder: 1
       OnFailure:
            Result: ROLLBACK
    ArtifactStore:
      Type: S3
      Location:
        Ref: ArtifactStoreS3Location
      EncryptionKey:
        Id: arn:aws:kms:useast-1:ACCOUNT-ID:key/KEY-ID
        Type: KMS
    DisableInboundStageTransitions:
      -
        StageName: Release
        Reason: "Disabling the transition until integration tests are completed"
    Tags:
      - Key: Project
        Value: ProjectA
      - Key: IsContainerBased
        Value: 'true'
```

For more information about configuring failure conditions for stage rollback, see OnFailure under `StageDeclaration` in the *CloudFormation User Guide*.

View rollback status in execution listing

You can view the status and target execution ID for a rollback execution.

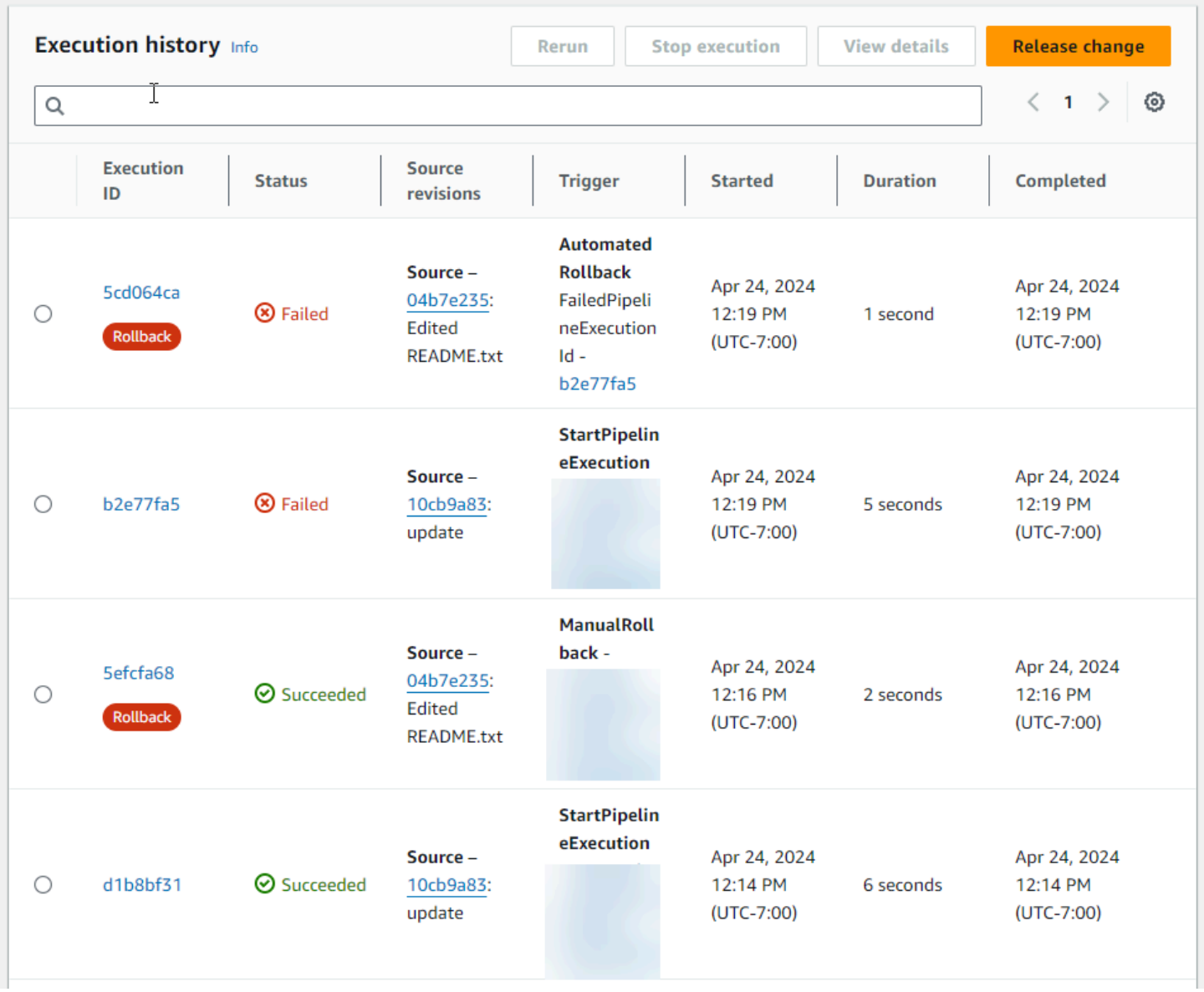
View rollback status in list of executions (console)

You can use the console to view the status and target execution ID for a rollback execution in the execution listing.

View rollback execution status in list of executions (console)

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 view.
3. Choose **History**. The list of executions shows the label **Rollback**.



Choose the execution ID for which you want to view details.

View rollback status with `list-pipeline-executions` (CLI)

You can use the CLI to view the status and target execution ID for a rollback execution.

- Open a terminal (Linux, macOS, or Unix) or command prompt (Windows) and use the AWS CLI to run the **list-pipeline-executions** command:

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

  This command returns a list of all of the completed executions associated with the pipeline.

The following example shows the returned data for a pipeline named *MyFirstPipeline* where a rollback execution started the pipeline.

```
{
    "pipelineExecutionSummaries": [
        {
            "pipelineExecutionId": "eb7ebd36-353a-4551-90dc-18ca5EXAMPLE",
            "status": "Succeeded",
            "startTime": "2024-04-16T09:00:28.185000+00:00",
            "lastUpdateTime": "2024-04-16T09:00:29.665000+00:00",
            "sourceRevisions": [
                {
                    "actionName": "Source",
                    "revisionId": "revision_ID",
                    "revisionSummary": "Added README.txt",
                    "revisionUrl": "console-URL"
                }
            ],
            "trigger": {
                "triggerType": "ManualRollback",
                "triggerDetail": "{arn:aws:sts::<account_ID>:assumed-role/<role>"}"
            },
            "executionMode": "SUPERSEDED",
            "executionType": "ROLLBACK",
            "rollbackMetadata": {
                "rollbackTargetPipelineExecutionId":
 "f15b38f7-20bf-4c9e-94ed-2535eEXAMPLE"
            }
        },
        {
            "pipelineExecutionId": "fcd61d8b-4532-4384-9da1-2aca1EXAMPLE",
            "status": "Succeeded",
            "startTime": "2024-04-16T08:58:56.601000+00:00",
            "lastUpdateTime": "2024-04-16T08:59:04.274000+00:00",
            "sourceRevisions": [
                {
                    "actionName": "Source",
                    "revisionId": "revision_ID",
                    "revisionSummary": "Added README.txt",
                    "revisionUrl": "console_URL"
                }
            ],
```

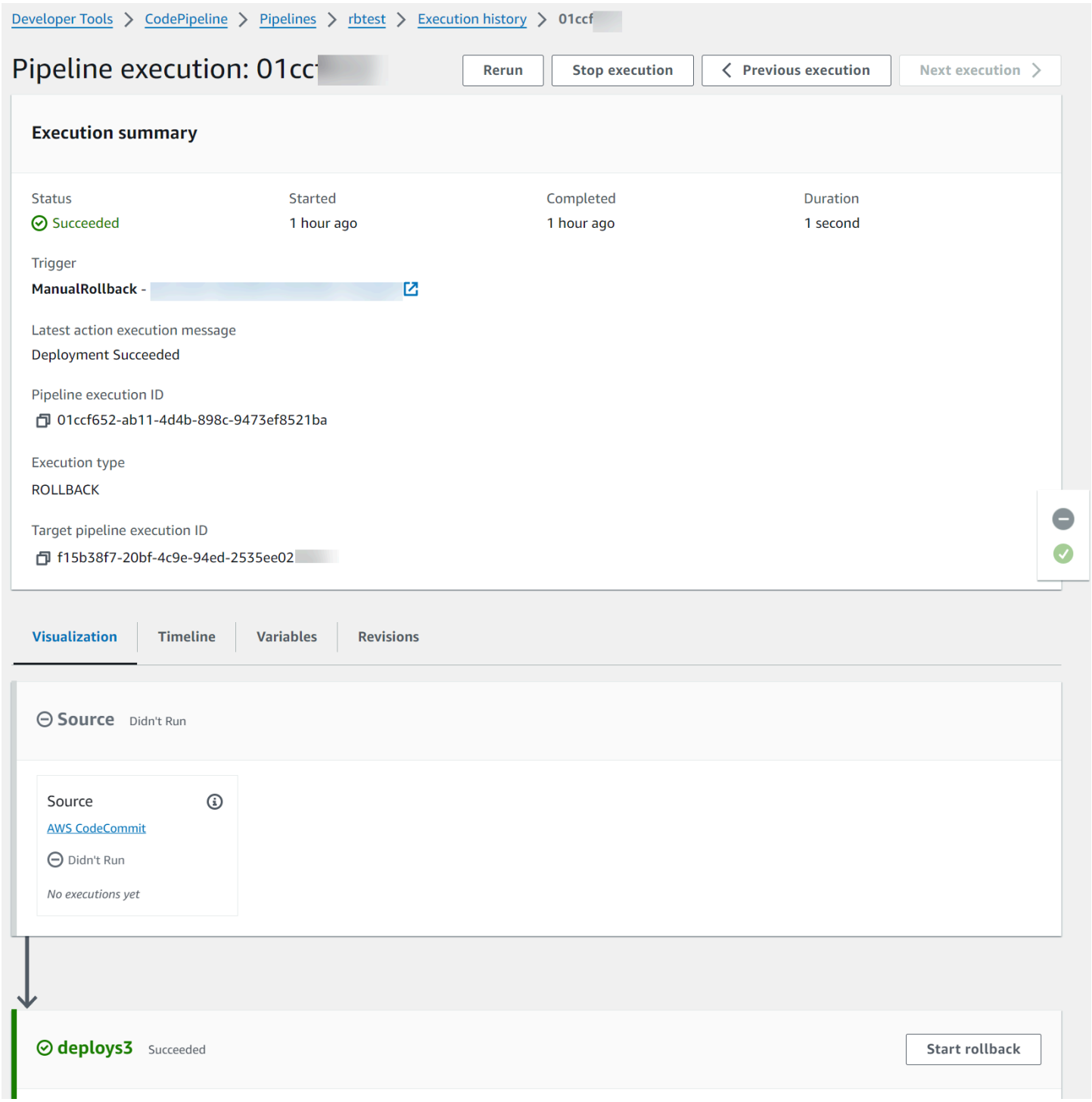
```
            "trigger": {
                "triggerType": "StartPipelineExecution",
                "triggerDetail": "arn:aws:sts::<account_ID>:assumed-role/<role>"
            },
            "executionMode": "SUPERSEDED"
        },
        {
            "pipelineExecutionId": "5cd064ca-bff7-425f-8653-f41d9EXAMPLE",
            "status": "Failed",
            "startTime": "2024-04-24T19:19:50.781000+00:00",
            "lastUpdateTime": "2024-04-24T19:19:52.119000+00:00",
            "sourceRevisions": [
                {
                    "actionName": "Source",
                    "revisionId": "<revision_ID>",
                    "revisionSummary": "Edited README.txt",
                    "revisionUrl": "<revision_URL>"
                }
            ],
            "trigger": {
                "triggerType": "AutomatedRollback",
                "triggerDetail": "{\"FailedPipelineExecutionId\":
\"b2e77fa5-9285-4dea-ae66-4389EXAMPLE\"}"
            },
            "executionMode": "SUPERSEDED",
            "executionType": "ROLLBACK",
            "rollbackMetadata": {
                "rollbackTargetPipelineExecutionId": "5efcfa68-d838-4ca7-
a63b-4a743EXAMPLE"
            }
         },
```

View rollback status details

You can view the status and target execution ID for a rollback execution.

View rollback status on detail page (console)

You can use the console to view the status and target pipeline execution ID for a rollback execution.



View rollback details with `get-pipeline-execution` (CLI)

Pipeline executions that have been rolled back will show in the output for getting the pipeline execution.

- To view details about a pipeline, run the **get-pipeline-execution** command, specifying the unique name of the pipeline. For example, to view details about a pipeline named *MyFirstPipeline*, enter the following:

```
aws codepipeline get-pipeline-execution --pipeline-name MyFirstPipeline --pipeline-
execution-id 3f658bd1-69e6-4448-ba3e-79007EXAMPLE
```

This command returns the structure of the pipeline.

The following example shows the returned data for a portion of a pipeline named *MyFirstPipeline*, where the rollback execution ID and metadata are shown.

```
{
    "pipelineExecution": {
        "pipelineName": "MyFirstPipeline",
        "pipelineVersion": 6,
        "pipelineExecutionId": "2004a94e-8b46-4c34-a695-c8d20EXAMPLE",
        "status": "Succeeded",
        "artifactRevisions": [
            {
                "name": "SourceArtifact",
                "revisionId": "<ID>",
                "revisionSummary": "Added README.txt",
                "revisionUrl": "<console_URL>"
            }
        ],
        "trigger": {
            "triggerType": "ManualRollback",
            "triggerDetail": "arn:aws:sts::<account_ID>:assumed-role/<role>"
        },
        "executionMode": "SUPERSEDED",
        "executionType": "ROLLBACK",
        "rollbackMetadata": {
            "rollbackTargetPipelineExecutionId": "4f47bed9-6998-476c-a49d-
e60beEXAMPLE"
        }
    }
}
```

View rollback state with `get-pipeline-state` (CLI)

Pipeline executions that have been rolled back will show in the output for getting the pipeline state.

- To view details about a pipeline, run the **get-pipeline-state** command, specifying the unique name of the pipeline. For example, to view state details about a pipeline named *MyFirstPipeline*, enter the following:

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

The following example shows the returned data with the rollback execution type.

```
{
    "pipelineName": "MyFirstPipeline",
    "pipelineVersion": 7,
    "stageStates": [
        {
            "stageName": "Source",
            "inboundExecutions": [],
            "inboundTransitionState": {
                "enabled": true
            },
            "actionStates": [
                {
                    "actionName": "Source",
                    "currentRevision": {
                        "revisionId": "<Revision_ID>"
                    },
                    "latestExecution": {
                        "actionExecutionId": "13bbd05d-
b439-4e35-9c7e-887cb789b126",
                        "status": "Succeeded",
                        "summary": "update",
                        "lastStatusChange": "2024-04-24T20:13:45.799000+00:00",
                        "externalExecutionId": "10cbEXAMPLEID"
                    },
                    "entityUrl": "console-url",
                    "revisionUrl": "console-url"
                }
            ],
            "latestExecution": {
```

```
                "pipelineExecutionId": "cf95a8ca-0819-4279-ae31-03978EXAMPLE",
                "status": "Succeeded"
            }
        },
        {
            "stageName": "deploys3",
            "inboundExecutions": [],
            "inboundTransitionState": {
                "enabled": true
            },
            "actionStates": [
                {
                    "actionName": "s3deploy",
                    "latestExecution": {
                        "actionExecutionId":
 "3bc4e3eb-75eb-45b9-8574-8599aEXAMPLE",
                        "status": "Succeeded",
                        "summary": "Deployment Succeeded",
                        "lastStatusChange": "2024-04-24T20:14:07.577000+00:00",
                        "externalExecutionId": "mybucket/SampleApp.zip"
                    },
                    "entityUrl": "console-URL"
                }
            ],
            "latestExecution": {
                "pipelineExecutionId": "fdf6b2ae-1472-4b00-9a83-1624eEXAMPLE",
                "status": "Succeeded",
                "type": "ROLLBACK"
            }
        }
    ],
    "created": "2024-04-15T21:29:01.635000+00:00",
    "updated": "2024-04-24T20:12:24.480000+00:00"
}
```

Configure conditions for a stage

You can specify a condition for a stage, such as checking for a specific variable in the pipeline run, and then engage a result for the condition, such as skipping the stage or failing the stage. A pipeline can be set up to check for stage conditions during the run, where you specify checks for a stage and then specify how the stage should continue when certain conditions are met. Conditions contain one or more rules that are available in a list of rules in CodePipeline. If all rules in a condition succeed, then the condition is met. You can configure conditions so that when the criteria are not met, the specified result engages.

Each condition has a rule set which is an ordered set of rules that are evaluated together. Therefore, if one rule fails in the condition, then the condition fails. You can override rule conditions at pipeline runtime.

Conditions are used for specific types of expressions and each has specific options for results available as follows:

- **Entry** - The conditions for making checks that, if met, allow entry to a stage. Rules are engaged with the following result options: **Fail** or **Skip**
- **On Failure** - The conditions for making checks for the stage when it fails. Rules are engaged with the following result option: **Rollback**
- **On Success** - The conditions for making checks for the stage when it succeeds. Rules are engaged with the following result options: **Rollback** or **Fail**

Conditions are supported by a set of *rules* for each type of condition.

For each type of condition, there are specific actions that are set up by the condition. The action is the result of the succeeded or failed condition check. For example, the condition for entry (entry condition) encounters an alarm (rule), then the check is successful and the result (action) is that the stage entry is blocked.

You can also use the AWS CodePipeline console or the AWS CLI to manually roll back or retry a stage or actions in a stage. See Configure conditions for a stage.

Topics

- Use cases for stage conditions

- Considerations for results configured for stage conditions
- Considerations for rules configured for stage conditions
- Creating Entry conditions
- Creating On Failure conditions
- Creating On Success conditions
- Deleting stage conditions
- Overriding stage conditions

Use cases for stage conditions

Stage conditions have multiple use cases for setting up release and change safety in pipelines. The following are sample use cases for stage conditions.

- Use an Entry condition to define a condition that will check the CloudWatch alarm state, which will then block a change if the production environment is not in a healthy state.
- Use an Entry condition with a wait time of 60 to define a condition to be evaluated when all the actions in a stage have successfully completed, and then roll back the changes if a CloudWatch alarm goes into ALARM state within 60 minutes.
- Use an On Success condition to define a condition so that when the stage completes successfully, the rule will check whether the current time is in the deployment window and then deploy if the rule succeeds.

Considerations for results configured for stage conditions

Considerations for stage conditions are as follows:

- You cannot use automatic stage retry with onFailure conditions.
- When configuring a condition with a **Rollback** result, the stage can only roll back to a previous execution if available in the current pipeline structure version.
- When configuring a condition with a **Rollback** result, you cannot roll back to a target execution ID that is a rollback execution type.
- For Entry conditions that use the **Skip** result to skip the stage if the condition fails, only the `LambdaInvoke` and `VariableCheck` rules are supported.

- You cannot perform a manual stage retry on a stage in **Skipped** status.
- You cannot perform a manual rollback to a stage in **Skipped** status.
- You cannot override a condition if the condition is configured with a **Skip** result.
- With the exception of **Skip** results, you can override a stage condition when starting a pipeline execution. For a stage condition where an override is engaged, the execution will perform as detailed in the following table.

Type	Configured result on condition failure	Stage status	Override behavior
Entry	Fail	In-progress	The stage proceeds.
Entry	Skip	Skipped	Not applicable.
OnFailure	Rollback	Failed	The stage is failed.
OnSuccess	Rollback	Succeeded	The stage proceeds.
OnSuccess	Fail	Failed	The stage proceeds.

Considerations for rules configured for stage conditions

Considerations for the available rules for stage conditions are as follows:

- For the `LambdaInvoke` rule, you must first configure the Lambda function to be used in the rule. Have the Lambda function ARN ready to provide when you configure the rule.
- For the `CloudWatchAlarm` rule, you must first configure the CloudWatch Events event to be used in the rule. Have the event ARN ready to provide when you configure the rule.

Creating Entry conditions

You can configure Entry conditions for a stage using the console or CLI. You will configure the corresponding rules and results for each condition. For a rollback result, the pipeline can only roll back to a previous execution if the previous execution was started in the current pipeline structure version.

The steps provide an example Entry condition that uses a monitor rule.

For more information, see Condition, RuleTypeId, and RuleExecution in the *CodePipeline API Guide*.

Creating Entry conditions - CloudWatchAlarm rule example (Console)

You can configure Entry conditions for a stage, along with the rules and results you want the stage to perform when the conditions are met.

Configure an Entry condition (console)

1. Complete any prerequisites, such as creating the resource and ARN for a rule where a resource is provided, such as the **AWS CloudWatchAlarm**.
2. 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.
3. In **Name**, choose the name of the pipeline you want to edit.
4. On the pipeline details page, choose **Edit**.
5. On the **Edit** page, for the action you want to edit, choose **Edit stage**.
6. Choose **Add entry condition**. The **Before stage entry condition** card displays with the **Fail** option available for this condition.
7. Choose **Add rule**, and then complete the following.

   a. In **Rule name**, enter a name for your rule. For this example, enter `MyAlarmRule`.

   b. In **Rule provider**, choose the preconfigured rule provider to add to your condition. For this example, choose **AWS CloudWatchAlarm**, and then complete the following steps.

   c. In **Region**, choose the Region for your condition or leave the default.

   d. In **Alarm name**, choose the CloudWatch resource to use for the rule. You must have already created the resource in your account.

   e. (Optional) In **Wait time**, enter the amount of time CodePipeline will wait if the alarm is in ALARM state when it is first evaluated. If the alarm is OK state when the rule is first checked, the rule will immediately succeed.

   f. (Optional) Enter any specific alarm states to monitor, and enter the role ARN if appropriate.

   g. When you are done editing the stage, choose **Done**. On the pipeline edit page, choose **Save**.
8. After the run, view the result.

Creating Entry conditions with Skip result and `VariableCheck` rule (console)

You can configure Entry conditions for a stage so that if the entry condition is not met, the stage is skipped. If the condition fails, then the result engages and the stage is skipped. When a stage is skipped, the stage status is **Skipped**, and the action status is **Didn't Run**. For considerations for stage conditions with Skip results, see Considerations for results configured for stage conditions.

In the following example, the variable check rule finds that the value is not a match, and the build stage is skipped.

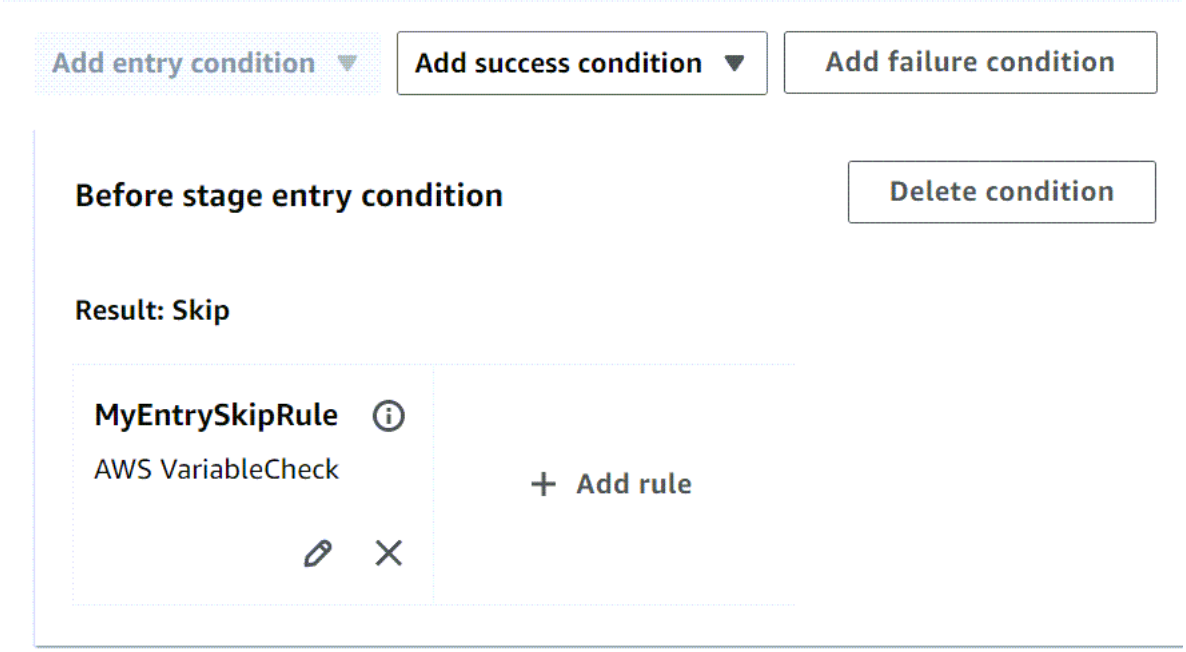
Configure an Entry condition with a Skip result (console)

1. Complete any prerequisites, such as creating the resource and ARN for a rule where a resource is provided, such as the **AWS CloudWatchAlarm**.
2. 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.
3. In **Name**, choose the name of the pipeline you want to edit.
4. On the pipeline details page, choose **Edit**.
5. On the **Edit** page, for the action you want to edit, choose **Edit stage**.
6. Choose **Add entry condition**, and then choose **Skip** as the result.
7. Choose **Add rule**, and then complete the following.

   a. In **Rule name**, enter a name for your rule. For this example, enter `MyAlarmRule`.

   b. In **Rule provider**, choose the preconfigured rule provider to add to your condition. For this example, choose **VariableCheck**, and then complete the following steps.

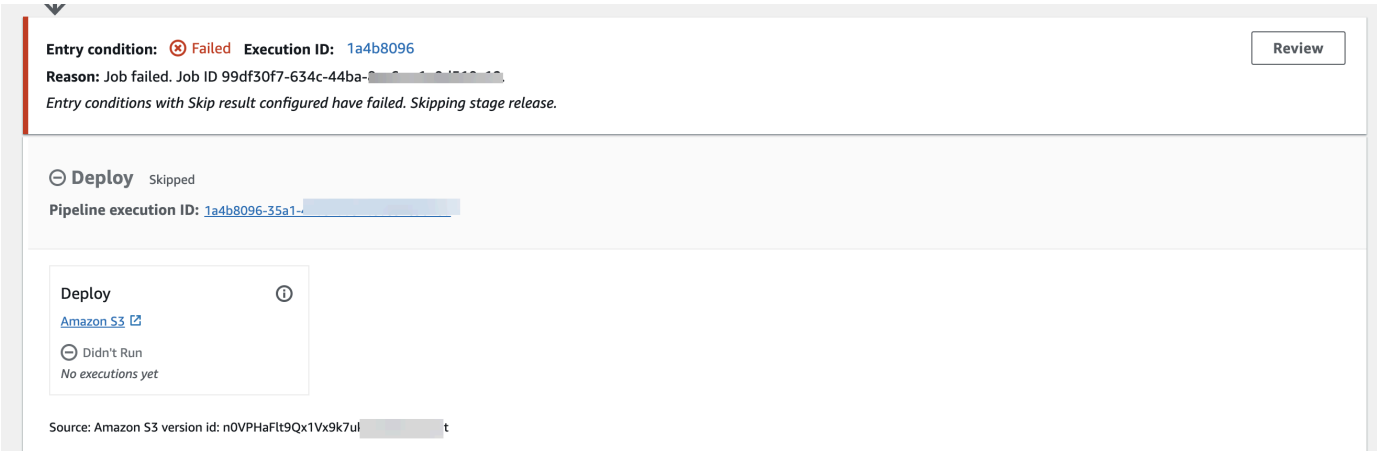


c. In **Region**, choose the Region for your condition or leave the default.

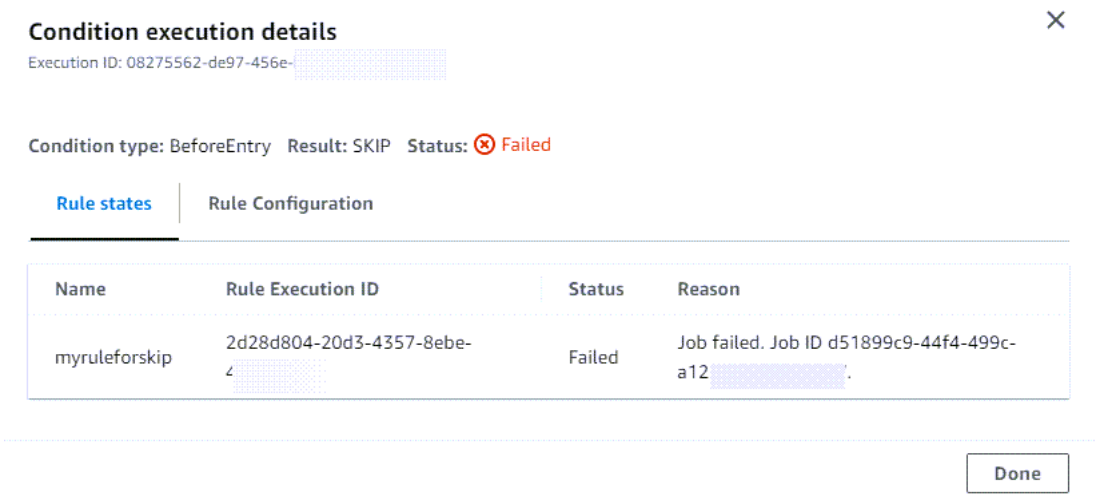
d. In **Variable**, choose the variable to compare with, such as `#{SourceVariables.FullRepositoryName}` for a pipeline that has a GitHub (via GitHub App) source action. Enter the repository name and choose the operator, such as **Equals**.

e. When you are done editing the stage, choose **Done**. On the pipeline edit page, choose **Save**.

8. After the run, view the result.



9. To review details, choose **Review**. The detail in the following example shows that the configured result for the condition is **Skip**, which cannot be overridden. The rule status is **Failed** due to the condition not being met.



Creating Entry conditions (CLI)

To use the AWS CLI to configure an Entry condition, use the commands to create or update a pipeline as detailed in Create a pipeline, stages, and actions and Edit a pipeline in CodePipeline.

Configure the condition and rule or rules (CLI)

- Open a terminal (Linux, macOS, or Unix) or command prompt (Windows) and use the AWS CLI to run the `update-pipeline` command, specifying the failure condition in the pipeline structure. The following example configures an Entry condition for a stage named `Deploy`:

```
{
    "name": "Deploy",
    "actions": [
        {
            "name": "Deploy",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "provider": "S3",
                "version": "1"
            },
            "runOrder": 1,
            "configuration": {
                "BucketName": "MyBucket",
                "Extract": "false",
                "ObjectKey": "object.xml"
            },
```

```
            "outputArtifacts": [],
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-east-1",
            "namespace": "DeployVariables"
        }
    ],
    "beforeEntry": {
        "conditions": [
            {
                "result": "FAIL",
                "rules": [
                    {
                        "name": "MyAlarmRule",
                        "ruleTypeId": {
                            "category": "Rule",
                            "owner": "AWS",
                            "provider": "CloudWatchAlarm",
                            "version": "1"
                        },
                        "configuration": {
                            "AlarmName": "CWAlarm",
                            "WaitTime": "1"
                        },
                        "inputArtifacts": [],
                        "region": "us-east-1"
                    }
                ]
            }
        ]
    }
}
```

For more information about configuring success conditions for stage rollback, see SuccessConditions in the *CodePipeline API Reference*.

Creating Entry conditions (CFN)

To use CloudFormation to configure an Entry condition, use the `beforeEntry` parameter. On entry, the stage will run the rule and perform the result.

```
beforeEntry:
     Result: FAIL
```

- Update the template as shown in the following snippet. The following example configures an Entry condition with a rule named `MyMonitorRule`:

```
Name: Deploy
Actions:
- Name: Deploy
  ActionTypeId:
    Category: Deploy
    Owner: AWS
    Provider: S3
    Version: '1'
  RunOrder: 1
  Configuration:
    BucketName: MyBucket
    Extract: 'false'
    ObjectKey: object.xml
  OutputArtifacts: []
  InputArtifacts:
  - Name: SourceArtifact
  Region: us-east-1
  Namespace: DeployVariables
BeforeEntry:
  Conditions:
  - Result: FAIL
    Rules:
    - Name: MyMonitorRule
      RuleTypeId:
        Category: Rule
        Owner: AWS
        Provider: CloudWatchAlarm
        Version: '1'
      Configuration:
        AlarmName: CWAlarm
        WaitTime: '1'
```

```
      InputArtifacts: []
      Region: us-east-1
```

For more information about configuring beforeEntry conditions, see AWS::CodePipeline::Pipeline BeforeEntryConditions under `StageDeclaration` in the *CloudFormation User Guide*.

Creating On Failure conditions

You can configure On Failure conditions for a stage using the console or CLI. You will configure the corresponding rules and results for each condition. For a rollback result, the pipeline can only roll back to a previous execution if the previous execution was started in the current pipeline structure version.

Creating On Failure conditions (Console)

You can configure On Failure conditions for a stage, along with the rules and results you want the stage to perform when the conditions are met.

Configure an On Failure condition (console)

1. Complete any prerequisites, such as creating the resource and ARN for a rule where a resource is provided, such as the **LambdaInvoke** rule.
2. 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.
3. In **Name**, choose the name of the pipeline you want to edit.
4. On the pipeline details page, choose **Edit**.
5. On the **Edit** page, for the action you want to edit, choose **Edit stage**.
6. Choose **Add failure condition**. The **Failure condition** card displays with the **Rollback** option available for this condition.
7. Choose **Add rule**, and then complete the following.

   a. In **Rule name**, enter a name for your rule. For this example, enter `MyLambdaRule`.

   b. In **Rule provider**, choose the preconfigured rule provider to add to your condition. For this example, choose **AWS LambdaInvoke**, and then complete the following steps.

c. In **Region**, choose the Region for your condition or leave the default.

d. In **Input artifacts**, choose the source artifact.

e. In **Function name**, choose the Lambda resource to use for the rule. You must have already created the resource in your account.

f. (Optional) In **User parameters**, enter any pairs that represent parameters for additional configuration.

g. (Optional) In **Role Arn**, enter the role ARN if configured.

h. (Optional) In **Timeout in Minutes**, enter the time in minutes that the rule should wait before timeout.

i. When you are done editing the stage, choose **Done**. On the pipeline edit page, choose **Save**.

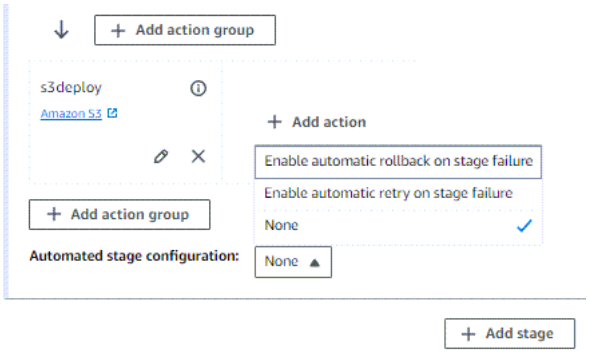
Creating onFailure conditions with a Retry result example (Console)

You can configure onFailure conditions for a stage so that if the entry condition is not met, the stage is retried. As part of this result, you configure the retry mode, specifying whether to retry the failed actions or to retry the failed stage.

Configure an onFailure condition with a Retry result (console)

1. Complete any prerequisites, such as creating the resource and ARN for a rule where a resource is provided, such as the **AWS CloudWatchAlarm**.
2. 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.
3. In **Name**, choose the name of the pipeline you want to edit.
4. On the pipeline details page, choose **Edit**.
5. On the **Edit** page, for the action you want to edit, choose **Edit stage**.
6. At the bottom of the stage, under **Automated stage configuration:**, choose **Enable automatic retry on stage failure**. In **Retry mode**, choose either **Retry failed stage** or **Retry failed actions**.



7. Choose to add an onFailure condition and then choose **Add rule** and enter a rule for the condition.

   a. In **Rule name**, enter a name for your rule. For this example, enter `MyAlarmRule`.

   b. In **Rule provider**, choose the preconfigured rule provider to add to your condition. For this example, choose **CloudWatchAlarm**, and then complete the following steps.

   c. In **Region**, choose the Region for your condition or leave the default.

   d. In **Alarm Name**, choose the configured resource for the alert.

   e. When you are done editing the stage, choose **Done**. On the pipeline edit page, choose **Save**.

8. After the run, view the result.

Creating On Failure conditions (CLI)

To use the AWS CLI to configure an On Failure condition, use the commands to create or update a pipeline as detailed in Create a pipeline, stages, and actions and Edit a pipeline in CodePipeline.

Configure the condition and rule or rules (CLI)

- Open a terminal (Linux, macOS, or Unix) or command prompt (Windows) and use the AWS CLI to run the `update-pipeline` command, specifying the failure condition in the pipeline structure. The following example configures an On Failure condition for a stage named `Deploy`:

```
{
    "name": "Deploy",
    "actions": [
        {
            "name": "Deploy",
```

```
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "provider": "S3",
                "version": "1"
            },
            "runOrder": 1,
            "configuration": {
                "BucketName": "MyBucket",
                "Extract": "false",
                "ObjectKey": "object.xml"
            },
            "outputArtifacts": [],
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-east-1",
            "namespace": "DeployVariables"
        }
    ],
    "onFailure": {
        "conditions": [
            {
                "result": "ROLLBACK",
                "rules": [
                    {
                        "name": "MyLambdaRule",
                        "ruleTypeId": {
                            "category": "Rule",
                            "owner": "AWS",
                            "provider": "LambdaInvoke",
                            "version": "1"
                        },
                        "configuration": {
                            "FunctionName": "my-function"
                        },
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "region": "us-east-1"
```

```
                    }
                ]
            }
        ]
    }
}
```

For more information about configuring failure conditions, see FailureConditions in the *CodePipeline API Reference*.

Creating On Failure conditions (CFN)

To use CloudFormation to configure an On Failure condition, use the `OnFailure` parameter. On failure, the stage will run the rule and perform the result.

```
OnFailure:
     Result: ROLLBACK
```

- Update the template as shown in the following snippet. The following example configures an OnFailure condition with a rule named `MyMonitorRule`:

```
name: Deploy
actions:
- name: Deploy
  actionTypeId:
    category: Deploy
    owner: AWS
    provider: S3
    version: '1'
  runOrder: 1
  configuration:
    BucketName: MyBucket
    Extract: 'false'
    ObjectKey: object.xml
  outputArtifacts: []
  inputArtifacts:
  - name: SourceArtifact
  region: us-east-1
  namespace: DeployVariables
OnFailure:
  conditions:
```

```
  - result: ROLLBACK
    rules:
    - name: MyMonitorRule
      ruleTypeId:
        category: Rule
        owner: AWS
        provider: CloudWatchAlarm
        version: '1'
      configuration:
        AlarmName: AlarmOnHelloWorldInvocation
        AlarmStates: ALARM
        WaitTime: '1'
      inputArtifacts: []
      region: us-east-1
```

For more information about configuring failure conditions, see OnFailure under `StageDeclaration` in the *CloudFormation User Guide*.

Creating On Success conditions

You can configure On Success conditions for a stage using the console or CLI. You will configure the corresponding rules and results for each condition. For a rollback result, the pipeline can only roll back to a previous execution if the previous execution was started in the current pipeline structure version.

The steps provide an example On Success condition that uses a deployment window rule.

For more information, see Condition, RuleTypeId, and RuleExecution in the *CodePipeline API Guide*.

Creating On Success conditions (Console)

You can configure On Success conditions for a stage, along with the rules and results you want the stage to perform when the conditions are met.

Configure an On Success condition (console)

1. Complete any prerequisites, such as creating the resource and ARN for a rule where a resource is provided, such as the AWS LambdaRule.
2. 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.

3. In **Name**, choose the name of the pipeline you want to edit.
4. On the pipeline details page, choose **Edit**.
5. On the **Edit** page, for the action you want to edit, choose **Edit stage**.
6. Choose **Add success condition**. The **On stage success condition** card displays. Choose the **Rollback** or **Fail** option shown as the available results for this condition type.
7. Choose **Add rule**, and then complete the following.

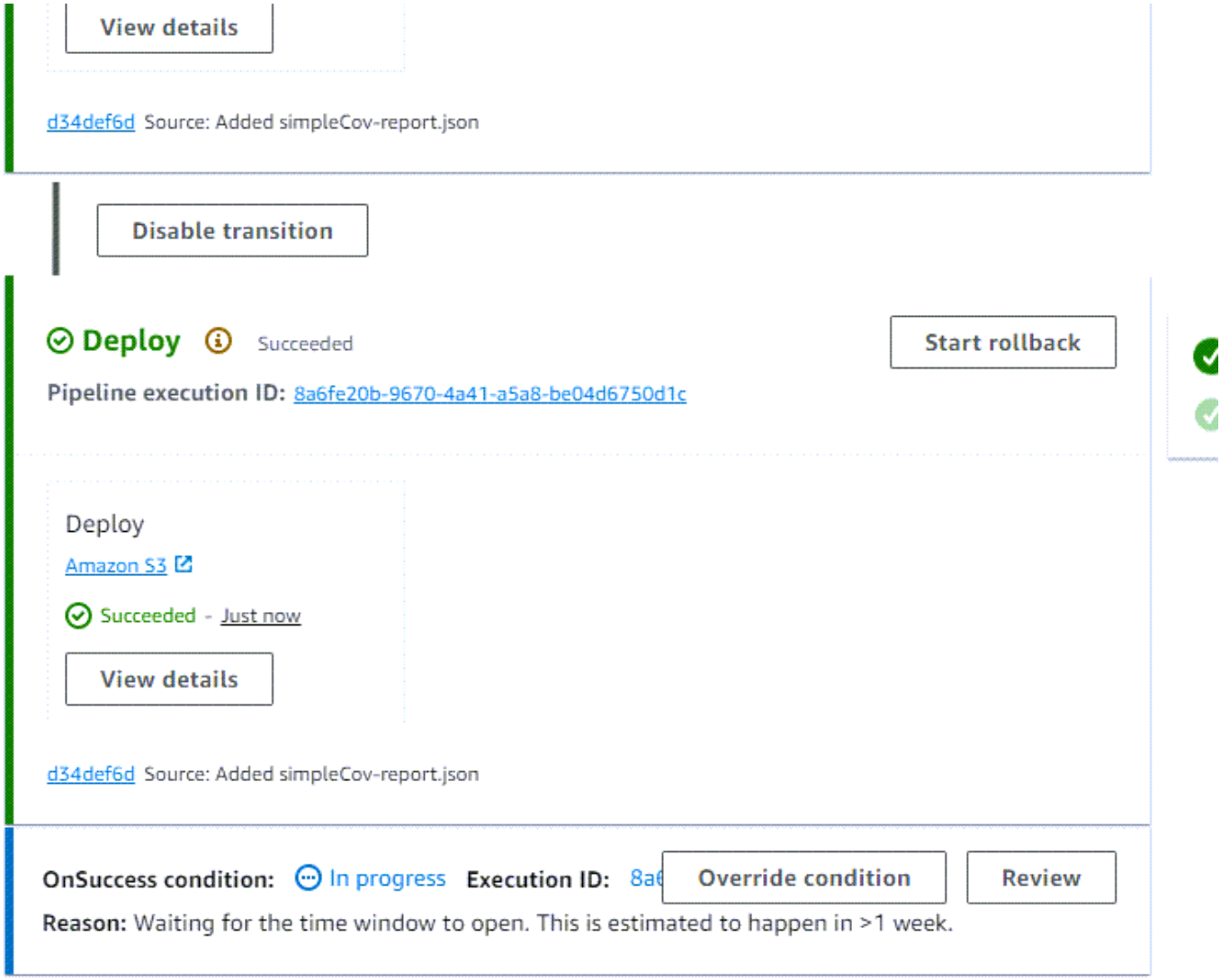
   a. In **Rule name**, enter a name for your condition. For this example, enter `MyDeploymentRule`.

   b. In **Rule provider**, choose the preconfigured rule to add to your condition. For this example, choose **AWS DeploymentWindow**, and then complete the following steps.

   c. In **Region**, choose the Region for your condition or leave the default.

   d. In **Cron**, enter the cron expression for the deployment window. The cron expression defines the days and times when the deployment should be allowed. For reference information about cron expressions, see Using cron and rate expressions to schedule rules.

   e. (Optional) In **TimeZone**, enter the time zone for the deployment window.
8. After the run, view the result.



Creating On Success conditions (CLI)

To use the AWS CLI to configure an On Success condition, use the commands to create or update a pipeline as detailed in Create a pipeline, stages, and actions and Edit a pipeline in CodePipeline.

Configure the condition and rule or rules (CLI)

- Open a terminal (Linux, macOS, or Unix) or command prompt (Windows) and use the AWS CLI to run the `update-pipeline` command, specifying the failure condition in the pipeline structure. The following example configures an On Success condition for a stage named `Deploy`, where the rule is named `MyDeploymentRule`:

```
{
    "name": "Deploy",
    "actions": [
        {
            "name": "Deploy",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "provider": "S3",
                "version": "1"
            },
            "runOrder": 1,
            "configuration": {
                "BucketName": "MyBucket",
                "Extract": "false",
                "ObjectKey": "object.xml"
            },
            "outputArtifacts": [],
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-east-1",
            "namespace": "DeployVariables"
        }
    ],
    "onSuccess": {
       "conditions": [
            {
                "result": "FAIL",
                "rules": [
                    {
                        "name": "MyAlarmRule",
                        "ruleTypeId": {
                            "category": "Rule",
                            "owner": "AWS",
                            "provider": "CloudWatchAlarm",
                            "version": "1"
                        },
                        "configuration": {
                            "AlarmName": "CWAlarm",
                            "WaitTime": "1"
```

```
                    },
                    "inputArtifacts": [],
                    "region": "us-east-1"
                }
            ]
        }
    ]
  }
}
```

For more information about configuring success conditions, see SuccessConditions in the *CodePipeline API Reference*.

Create an On Success condition (CFN)

To use CloudFormation to configure an On Success condition, use the `OnSuccess` parameter. On success, the stage will run the rule and perform the result.

```
OnSuccess:
     Result: ROLLBACK
```

- Update the template as shown in the following snippet. The following example configures an OnSuccess condition with a rule named `MyDeploymentWindowRule`:

```
name: Deploy
actions:
- name: Deploy
  actionTypeId:
    category: Deploy
    owner: AWS
    provider: S3
    version: '1'
  runOrder: 1
  configuration:
    BucketName: MyBucket
    Extract: 'false'
    ObjectKey: object.xml
  outputArtifacts: []
  inputArtifacts:
  - name: SourceArtifact
  region: us-east-1
```

```
  namespace: DeployVariables
onSuccess:
   conditions:
  - result: FAIL
    rules:
    - name: MyMonitorRule
      ruleTypeId:
        category: Rule
        owner: AWS
        provider: CloudWatchAlarm
        version: '1'
      configuration:
        AlarmName: CWAlarm
        WaitTime: '1'
      inputArtifacts: []
      region: us-east-1
```

For more information about configuring failure conditions for stage rollback, see OnFailure under `StageDeclaration` in the *CloudFormation User Guide*.

Deleting stage conditions

You can delete stage conditions that have been configured for your pipeline.

To delete a stage condition

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

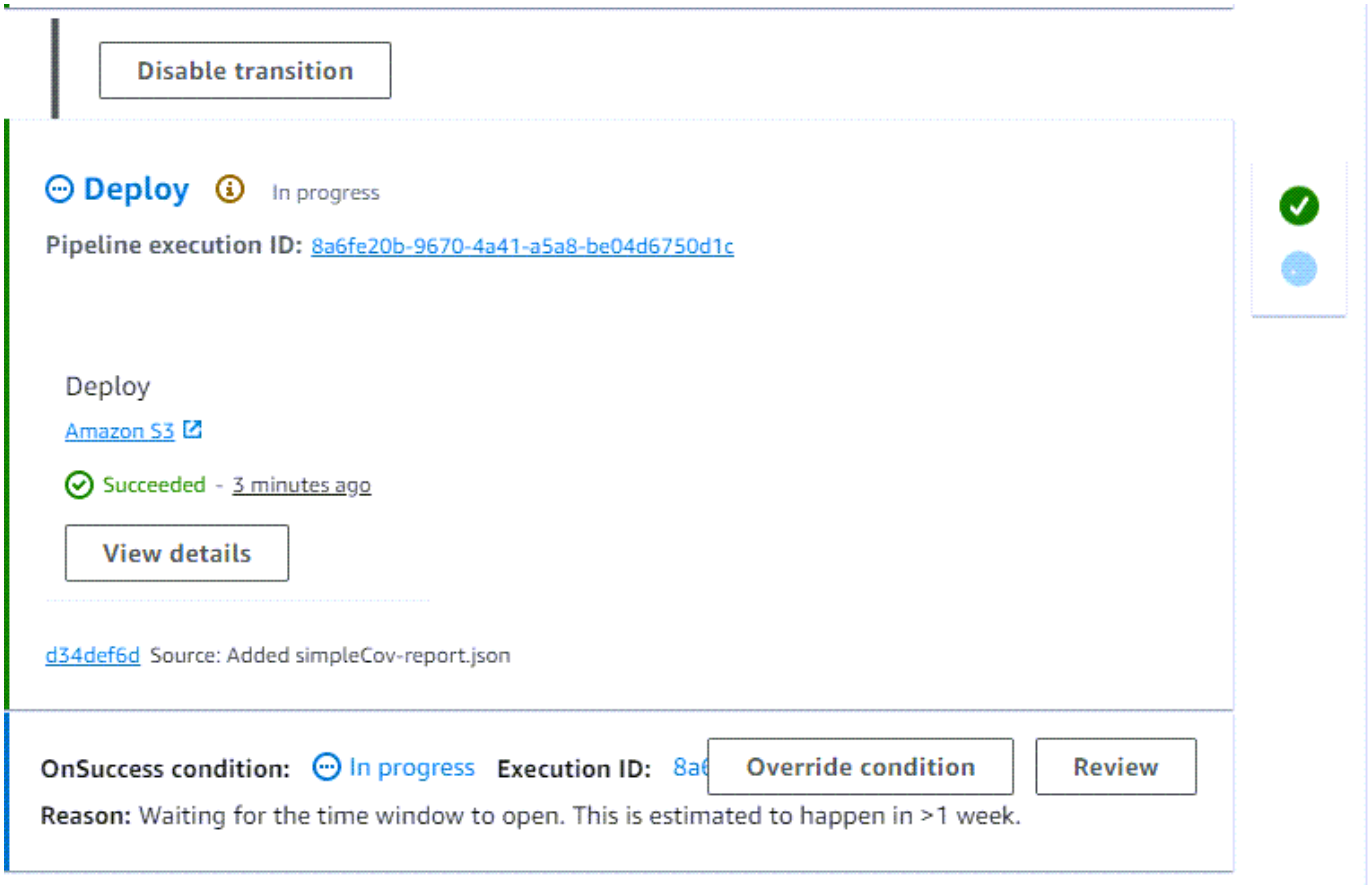
   The names and status of all pipelines associated with your AWS account are displayed.
2. In **Name**, choose the name of the pipeline you want to edit.
3. On the pipeline details page, choose **Edit**.
4. On the **Edit** page, for the condition you want to edit, choose **Edit stage**.
5. Next to the condition that you want to delete, choose **Delete condition**.

Overriding stage conditions

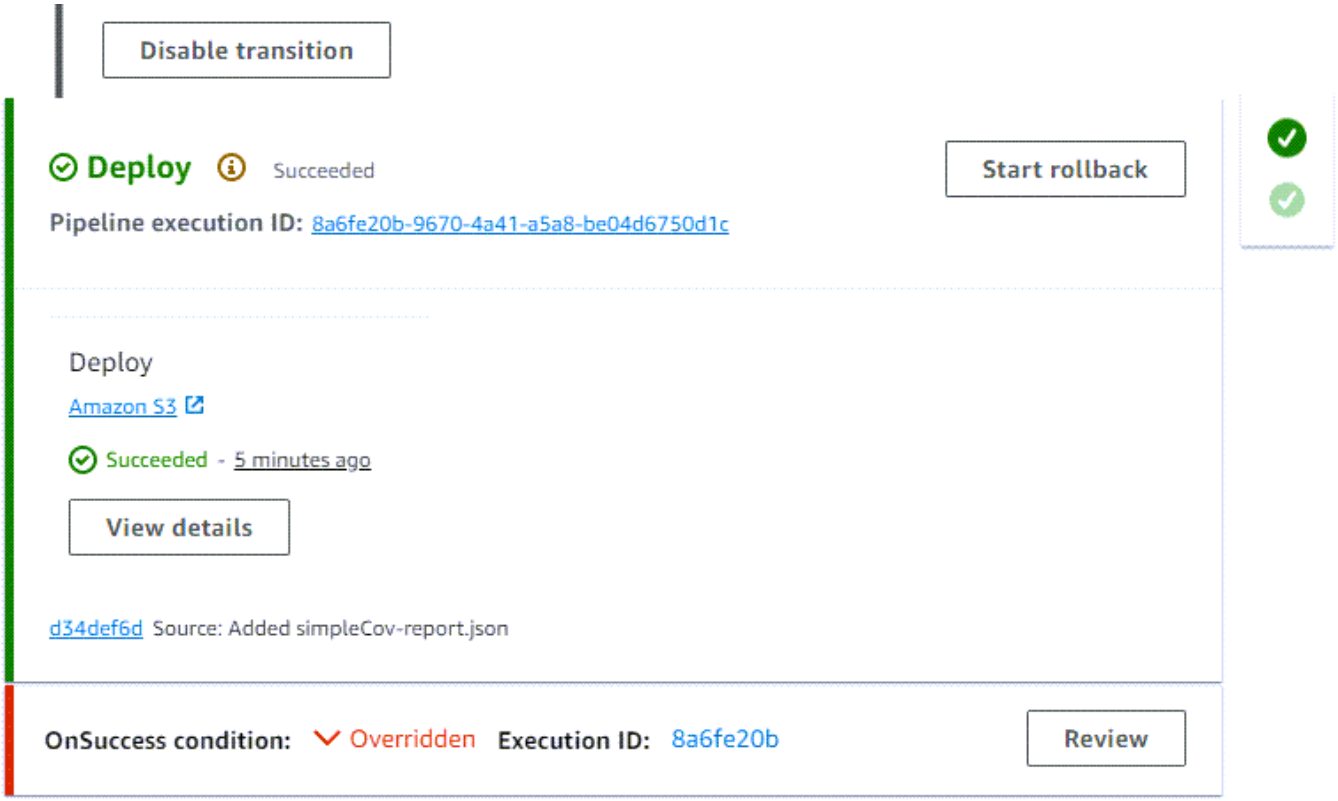
You can override stage conditions that have been configured for your pipeline. In the console, when the stage and rule are running, you can choose to override the stage condition. This results in the stage running.

To override a stage condition

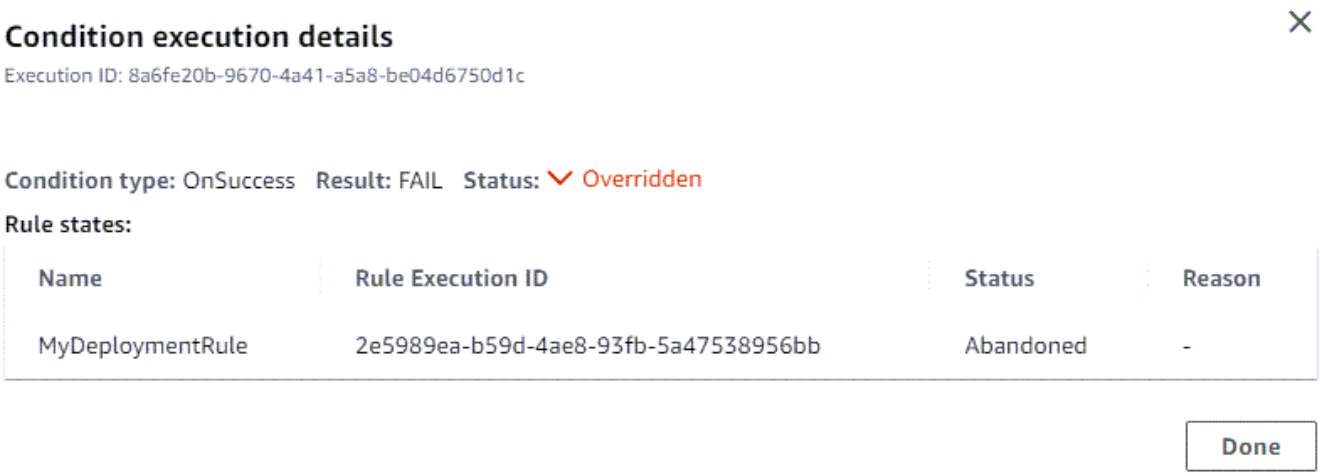
1. In this example, the pipeline stage is running with a condition. The **Override** button is enabled.



2. Next to the condition that you want to override, choose **Override**.



3. To review details, choose **Review**. The detail in the following example shows that the configured result for the condition is Fail, which has been overridden. The rule status is Abandoned due to the override.



Use action types, custom actions, and approval actions

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 service and partner products and services you can integrate into your pipeline based on action type, see Integrations with CodePipeline action types.

Topics

- Working with action types
- Create and add a custom action in CodePipeline
- Tag a custom action in CodePipeline
- Invoke an AWS Lambda function in a pipeline in CodePipeline
- Add a manual approval action to a stage
- Add a cross-region action in CodePipeline
- Working with variables

Working with action types

Action types are preconfigured actions that you as a provider create for customers by using one of the supported integration models in AWS CodePipeline.

You can request, view, and update action types. If the action type is created for your account as the owner, you can use the AWS CLI to view or update your action type properties and structure. If you

are the provider or owner of the action type, your customers can choose the action and add it to their pipelines after it is available in CodePipeline.

> **Note**
>
> You create actions with `custom` in the `owner` field to run with a job worker. You do not create them with an integration model. For information about custom actions, see Create and add a custom action in CodePipeline.

Action type components

The following components make up an action type.

- **Action type ID** – The *ID* consists of the category, owner, provider, and version. The following example shows an action type ID with an owner of `ThirdParty`, a category of `Test`, a provider named `TestProvider`, and a version of 1.

```
{
    "Category": "Test",
    "Owner": "ThirdParty",
    "Provider": "TestProvider",
    "Version": "1"
},
```

- **Executor configuration** – The integration model, or action engine, specified when the action is created. When you specify the executor for an action type, you choose one of two types:
  - *Lambda:* The action type owner writes the integration as a Lambda function, which is invoked by CodePipeline whenever there is a job available for the action.
  - *JobWorker:* The action type owner writes the integration as a job worker that polls for available jobs on customer pipelines. The job worker then runs the job and submits the job result back to CodePipeline by using CodePipeline APIs.

    > **Note**
    >
    > The job worker integration model is not the preferred integration model.

- **Input and output artifacts:** Limits for the artifacts that the action type owner designates for customers of the action.
- **Permissions:** The permissions strategy that designates customers who can access the third-party action type. The permissions strategies available depend on the chosen integration model for the action type.
- **URLs:** Deep links to resources that the customer can interact with, such as the action type owner's configuration page.

Topics

- Request an action type
- Add an available action type to a pipeline (console)
- View an action type
- Update an action type

Request an action type

When a new CodePipeline action type is requested by a third-party provider, the action type is created for the action type owner in CodePipeline, and the owner can manage and view the action type.

An action type can be either a private or public action. When your action type is created, it is private. To request an action type be changed to a public action, contact the CodePipeline service team.

Before you create your action definition file, executor resources, and action type request for the CodePipeline team, you must choose an integration model.

Step 1: Choose your integration model

Choose your integration model and then create the configuration for that model. After you choose the integration model, you must configure your integration resources.

- For the Lambda integration model, you create a Lambda function and add permissions. Add permissions to your integrator Lambda function to provide the CodePipeline service with permissions to invoke it using the CodePipeline service principal:

`codepipeline.amazonaws.com`. The permissions can be added using CloudFormation or the command line.

- Example for adding permissions using CloudFormation:

```
  CodePipelineLambdaBasedActionPermission:
    Type: 'AWS::Lambda::Permission'
    Properties:
      Action: 'lambda:invokeFunction'
      FunctionName: {"Fn::Sub": "arn:${AWS::Partition}:lambda:${AWS::Region}:
${AWS::AccountId}:function:function-name"}
      Principal: codepipeline.amazonaws.com
```

- Documentation for command line

- For the job worker integration model, you create an integration with a list of allowed accounts where the job worker polls for jobs with the CodePipeline APIs.

Step 2: Create an action type definition file

You define an action type in an action type definition file using JSON. In the file, you include the action category, the integration model used to manage the action type, and configuration properties.

> **Note**
>
> After you create a public action, you can't change the action type property under `properties` from `optional` to `required`. You also can't change the `owner`.

For more information about the action type definition file parameters, see ActionTypeDeclaration and UpdateActionType in the CodePipeline API Reference.

There are eight sections in the action type definition file:

- `description`: The description for the action type to be updated.
- `executor`: Information about the executor for an action type that was created with a supported integration model, either `Lambda` or `job worker`. You can only provide either `jobWorkerExecutorConfiguration` or `lambdaExecutorConfiguration`, based on your executor type.

- `configuration`: Resources for the configuration of the action type, based on the chosen integration model. For the Lambda integration model, use the Lambda function ARN. For the job worker integration model, use the account or list of accounts from where the job worker runs.
- `jobTimeout`: The timeout in seconds for the job. An action execution can consist of multiple jobs. This is the timeout for a single job, and not for the entire action execution.

  > **Note**
  >
  > For the Lambda integration model, the maximum timeout is 15 minutes.

- `policyStatementsTemplate`: The policy statement that specifies the permissions in the CodePipeline customer's account that are needed to successfully run an action execution.
- `type`: The integration model used to create and update the action type, either `Lambda` or `JobWorker`.

- `id`: The category, owner, provider, and version ID for the action type:
  - `category`: The kind of action can be taken in the stage: Source, Build, Deploy, Test, Invoke, or Approval.
  - `provider`: The provider of the action type being called, such as the provider company or product name. The provider name is supplied when the action type is created.
  - `owner`: The creator of the action type being called: `AWS` or `ThirdParty`.
  - `version`: A string used to version the action type. For the first version, set the version number to 1.
- `inputArtifactDetails`: The number of artifacts to expect from the previous stage in the pipeline.
- `outputArtifactDetails`: The number of artifacts to expect from the result from the action type stage.
- `permissions`: Details identifying the accounts with permissions to use the action type.
- `properties`: The parameters required for your project tasks to complete.
  - `description`: The description of the property that is displayed to users.
  - `optional`: Whether the configuration property is optional.
  - noEcho: Whether the field value entered by the customer is omitted from the log. If `true`, then the value is redacted when returned with a GetPipeline API request.
  - `key`: Whether the configuration property is a key.

- `queryable`: Whether the property is used with polling. An action type can have up to one queryable property. If it has one, that property must be both required and not secret.
- `name`: The property name that is displayed to users.

- `urls`: A list of the URLs CodePipeline displays to your users.
  - `entityUrlTemplate`: URL to the external resources for the action type, such as a configuration page.
  - `executionUrlTemplate`: URL to the details for the latest run of the action.
  - `revisionUrlTemplate`: URL displayed in the CodePipeline console to the page where customers can update or change the configuration of the external action.
  - `thirdPartyConfigurationUrl`: URL of a page where users can sign up for an external service and perform initial configuration of the action provided by that service.

The following code shows an example action type definition file.

```
{
   "actionType": {
      "description": "string",
      "executor": {
         "configuration": {
            "jobWorkerExecutorConfiguration": {
               "pollingAccounts": [ "string" ],
               "pollingServicePrincipals": [ "string" ]
            },
            "lambdaExecutorConfiguration": {
               "lambdaFunctionArn": "string"
            }
         },
         "jobTimeout": number,
         "policyStatementsTemplate": "string",
         "type": "string"
      },
      "id": {
         "category": "string",
         "owner": "string",
         "provider": "string",
         "version": "string"
      },
      "inputArtifactDetails": {
         "maximumCount": number,
```

```
         "minimumCount": number
      },
      "outputArtifactDetails": {
         "maximumCount": number,
         "minimumCount": number
      },
      "permissions": {
         "allowedAccounts": [ "string" ]
      },
      "properties": [
         {
            "description": "string",
            "key": boolean,
            "name": "string",
            "noEcho": boolean,
            "optional": boolean,
            "queryable": boolean
         }
      ],
      "urls": {
         "configurationUrl": "string",
         "entityUrlTemplate": "string",
         "executionUrlTemplate": "string",
         "revisionUrlTemplate": "string"
      }
   }
}
```

Step 3: Register Your Integration with CodePipeline

To register your action type with CodePipeline, you contact the CodePipeline service team with your request.

The CodePipeline service team registers the new action type integration by making changes in the service codebase. CodePipeline registers two new actions: a *public action* and a *private action*. You use the private action for testing, and then when ready, you activate the public action to serve customer traffic.

To register a request for a Lambda integration

- Send a request to the CodePipeline service team using the following form.

```
This issue will track the onboarding of [Name] in CodePipeline.


[Contact engineer] will be the primary point of contact for this integration.

Name of the action type as you want it to appear to customers: Example.com Testing

Initial onboard checklist:

1. Attach an action type definition file in JSON format. This includes the schema
 for the action type

2. A list of test accounts for the allowlist which can access the new action type
 [{account, account_name}]

3. The Lambda function ARN

4. List of AWS Regions where your action will be available

5. Will this be available as a public action?
```

To register a request for a job worker integration

- Send a request to the CodePipeline service team using the following form.

```
This issue will track the onboarding of [Name] in CodePipeline.

[Contact engineer] will be the primary point of contact for this integration.


Name of the action type as you want it to appear to customers: Example.com Testing

Initial onboard checklist:

1. Attach an action type definition file in JSON format. This includes the schema
 for the action type.

2. A list of test accounts for the allowlist which can access the new action type
 [{account, account_name}]

3. URL information:
```

```
Website URL: https://www.example.com/%TestThirdPartyName%/%TestVersionNumber%

Example URL pattern where customers will be able to review their configuration
 information for the action: https://www.example.com/%TestThirdPartyName%/
%customer-ID%/%CustomerActionConfiguration%

Example runtime URL pattern: https://www.example.com/%TestThirdPartyName%/
%customer-ID%/%TestRunId%

4. List of AWS Regions where your action will be available

5. Will this be available as a public action?
```

Step 4: Activate Your New Integration

Contact the CodePipeline service team when you are ready to use the new integration publicly.

Add an available action type to a pipeline (console)

You add your action type to a pipeline so that you can test it. You can do this by creating a new pipeline or editing an existing one.

> **Note**
>
> If your action type is a source, build, or deploy category action, you can add it by creating a pipeline. If your action type is in the test category, you must add it by editing an existing pipeline.

To add your action type to an existing pipeline from the CodePipeline console

1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home.
2. In the list of pipelines, choose the pipeline where you want to add the action type.
3. On the summary view page of the pipeline, choose **Edit**.
4. Choose to edit the stage. In the stage where you want to add your action type, choose **Add action group**. The **Edit action** page displays.
5. On the **Edit action** page, in **Action name**, enter a name for the action. This is the name that displays for the stage in your pipeline.

6. In **Action provider**, choose your action type from the list.

   Note that the value in the list is based on the `provider` specified in the action type definition file.

7. In **Input artifacts**, enter the artifact name in this format:

   `Artifactname``::`*`FileName`*

   Note that the minimum and maximum quantities allowed are defined based on the `inputArtifactDetails` specified in the action type definition file.

8. Choose **Connect to <Action_Name>**.

   A browser window opens and connects to the website you have created for your action type.

9. Log in to your website as a customer and complete the steps a customer takes to use your action type. Your steps will vary depending on your action category, website, and configuration, but usually include a completion action that returns the customer to the **Edit action** page.

10. In the CodePipeline **Edit action** page, the additional configuration fields for the action display. The fields that display are the configuration properties that you specified in the action definition file. Enter the information in the fields that are customized for your action type.

    For example, if the action definition file specified a property named `Host`, then a field with the label **Host** is shown on the **Edit action** page for your action.

11. In **Output artifacts**, enter the artifact name in this format:

    `Artifactname``::`*`FileName`*

    Note that the minimum and maximum quantities allowed are defined based on the `outputArtifactDetails` specified in the action type definition file.

12. Choose **Done** to return to the pipeline details page.

    > **Note**
    >
    > Your customers can optionally use the CLI to add the action type to their pipeline.

13. To test your action, commit a change to the source specified in the source stage of the pipeline or follow the steps in Manually Start a Pipeline.

To create a pipeline with your action type, follow the steps in Create a pipeline, stages, and actions and choose your action type from as many stages as you will test.

View an action type

You can use the CLI to view your action type. Use the **get-action-type** command to view action types that have been created using an integration model.

To view an action type

1. Create an input JSON file and name the file `file.json`. Add your action type ID in JSON format as shown in the following example.

```
{
    "category": "Test",
    "owner": "ThirdParty",
    "provider": "TestProvider",
    "version": "1"
}
```

2. In a terminal window or at the command line, run the **get-action-type** command.

```
aws codepipeline get-action-type --cli-input-json file://file.json
```

This command returns the action definition output for an action type. This example shows an action type that was created with the Lambda integration model.

```
{
    "actionType": {
        "executor": {
            "configuration": {
                "lambdaExecutorConfiguration": {
                    "lambdaFunctionArn": "arn:aws:lambda:us-west-2:<account-
id>:function:my-function"
                }
            },
            "type": "Lambda"
        },
        "id": {
            "category": "Test",
            "owner": "ThirdParty",
            "provider": "TestProvider",
```

```
            "version": "1"
        },
        "inputArtifactDetails": {
            "minimumCount": 0,
            "maximumCount": 1
        },
        "outputArtifactDetails": {
            "minimumCount": 0,
            "maximumCount": 1
        },
        "permissions": {
            "allowedAccounts": [
                "<account-id>"
            ]
        },
        "properties": []
    }
}
```

Update an action type

You can use the CLI to edit action types that are created with an integration model.

For a public action type, you can't update the owner, you can't change optional properties to required, and you can only add new optional properties.

1. Use the `get-action-type` command to get the structure for your action type. Copy the structure.
2. Create an input JSON file and name it `action.json`. Paste the action type structure you copied in the previous step into it. Update any parameters you want to change. You can also add optional parameters.

   For more information about the parameters for the input file, see the action definition file description in Step 2: Create an action type definition file.

   The following example shows how to update an example action type created with the Lambda integration model. This example makes the following changes:

   - Changes the `provider` name to `TestProvider1`.
   - Add a job timeout limit of 900 seconds.

- Adds an action configuration property named `Host` that is displayed to the customer using the action.

```
{
    "actionType": {
        "executor": {
            "configuration": {
                "lambdaExecutorConfiguration": {
                    "lambdaFunctionArn": "arn:aws:lambda:us-west-2:<account-
id>:function:my-function"
                }
            },
            "type": "Lambda",
            "jobTimeout": 900
        },
        "id": {
            "category": "Test",
            "owner": "ThirdParty",
            "provider": "TestProvider1",
            "version": "1"
        },
        "inputArtifactDetails": {
            "minimumCount": 0,
            "maximumCount": 1
        },
        "outputArtifactDetails": {
            "minimumCount": 0,
            "maximumCount": 1
        },
        "permissions": {
            "allowedAccounts": [
                "account-id"
            ]
        },
        "properties": {
         "description": "Owned build action parameter description",
         "optional": true,
         "noEcho": false,
         "key": true,
         "queryable": false,
         "name": "Host"
         }
    }
```

```
}
```

3. At the terminal or command line, run the **update-action-type** command

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

This command returns the action type output to match your updated parameters.

Create and add a custom action in CodePipeline

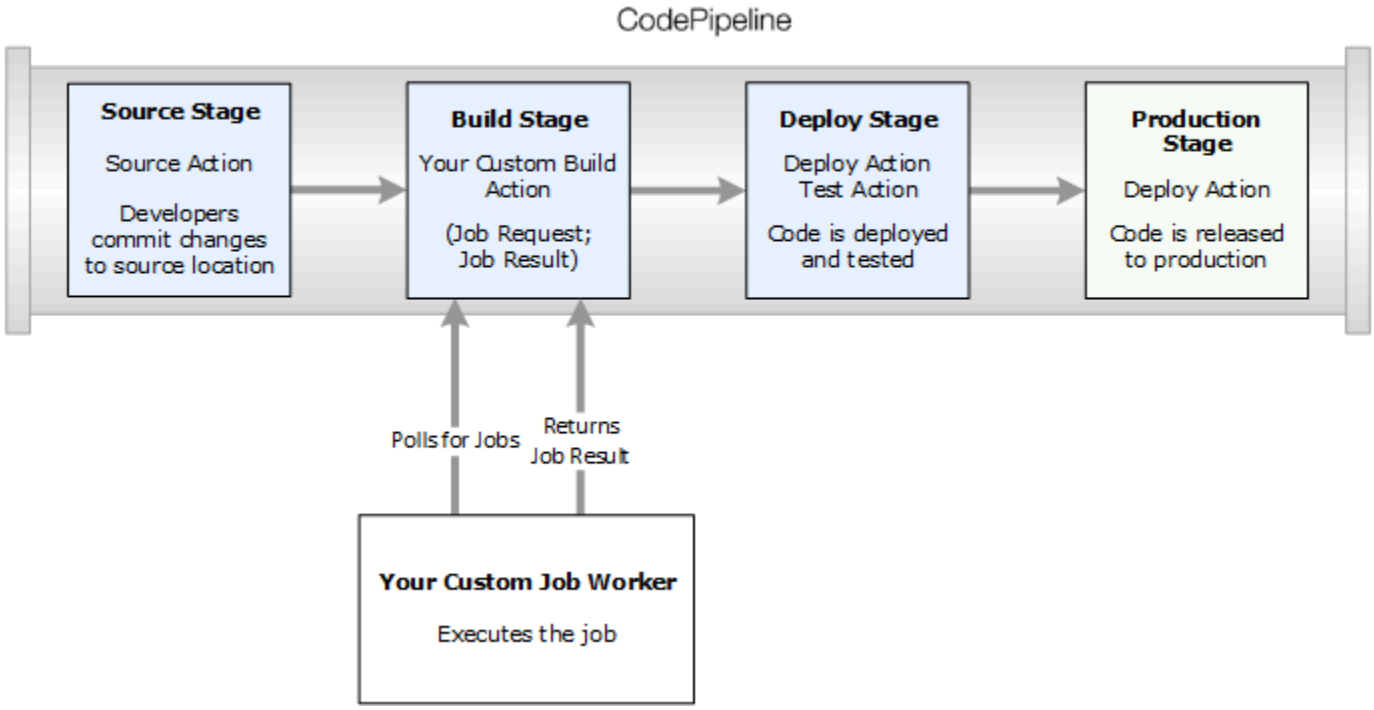
AWS CodePipeline includes a number of actions that help you configure build, test, and deploy resources for your automated release process. If your release process includes activities that are not included in the default actions, such as an internally developed build process or a test suite, you can 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:



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 provide the revision and its artifacts to the next action. If a failure is returned, the pipeline will not provide 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.

Topics

- Create a custom action
- Create a job worker for your custom action
- Add a custom action to a pipeline

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:

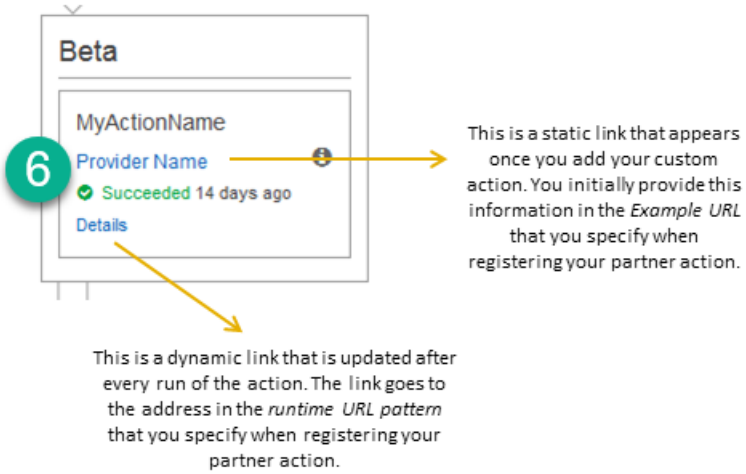
```
{
    "category": "Build",
    "provider": "My-Build-Provider-Name",
    "version": "1",
    "settings": {
        "entityUrlTemplate": "https://my-build-instance/job/{Config:ProjectName}/",
        "executionUrlTemplate": "https://my-build-instance/job/
{Config:ProjectName}/lastSuccessfulBuild/{ExternalExecutionId}/"
    },
    "configurationProperties": [{
        "name": "ProjectName",
        "required": true,
        "key": true,
        "secret": false,
        "queryable": false,
        "description": "The name of the build project must be provided when this
 action is added to the pipeline.",
        "type": "String"
    }],
    "inputArtifactDetails": {
        "maximumCount": integer,
        "minimumCount": integer
    },
    "outputArtifactDetails": {
        "maximumCount": integer,
        "minimumCount": integer
    },
    "tags": [{
      "key": "Project",
      "value": "ProjectA"
    }]
}
```

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.

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:



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.

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.

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:

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

   ```
   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:

```
{
    "actionType": {
        "inputArtifactDetails": {
            "maximumCount": 1,
            "minimumCount": 1
       },
       "actionConfigurationProperties": [
            {
                "secret": false,
                "required": true,
                "name": "ProjectName",
                "key": true,
                "description": "The name of the build project must be provided when
 this action is added to the pipeline."
            }
        ],
        "outputArtifactDetails": {
            "maximumCount": 0,
            "minimumCount": 0
        },
        "id": {
            "category": "Build",
            "owner": "Custom",
            "version": "1",
            "provider": "My-Build-Provider-Name"
        },
        "settings": {
            "entityUrlTemplate": "https://my-build-instance/job/
{Config:ProjectName}/",
            "executionUrlTemplate": "https://my-build-instance/job/mybuildjob/
lastSuccessfulBuild/{ExternalExecutionId}/"
        }
    }
}
```

Note

As part of the output of the **create-custom-action-type** command, the `id` section includes `"owner": "Custom"`. CodePipeline automatically assigns `Custom` as the

owner of custom action types. This value can't be assigned or changed when you use the **create-custom-action-type** command or the **update-pipeline** command.

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
- Develop a job worker for your custom action
- Custom job worker architecture and examples

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*.

Create one of the following to use under your AWS account when running your custom action and job worker.

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
IAM	(Recommended) Use console credentials as temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Login for AWS local development in the *AWS Command Line Interface User Guide*.

Which user needs programmatic access?	To	By
		• For AWS SDKs, see Login for AWS local development in the *AWS SDKs and Tools Reference Guide*.
Workforce identity (Users managed in IAM Identity Center)	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the *AWS Command Line Interface User Guide*. • For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentials with AWS resources in the *IAM User Guide*.

Which user needs programmatic access?	To	By
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Authenticating using IAM user credentials in the *AWS Command Line Interface User Guide*. • For AWS SDKs and tools, see Authenticate using long-term credentials in the *AWS SDKs and Tools Reference Guide*. • For AWS APIs, see Managing access keys for IAM users in the *IAM User Guide*.

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

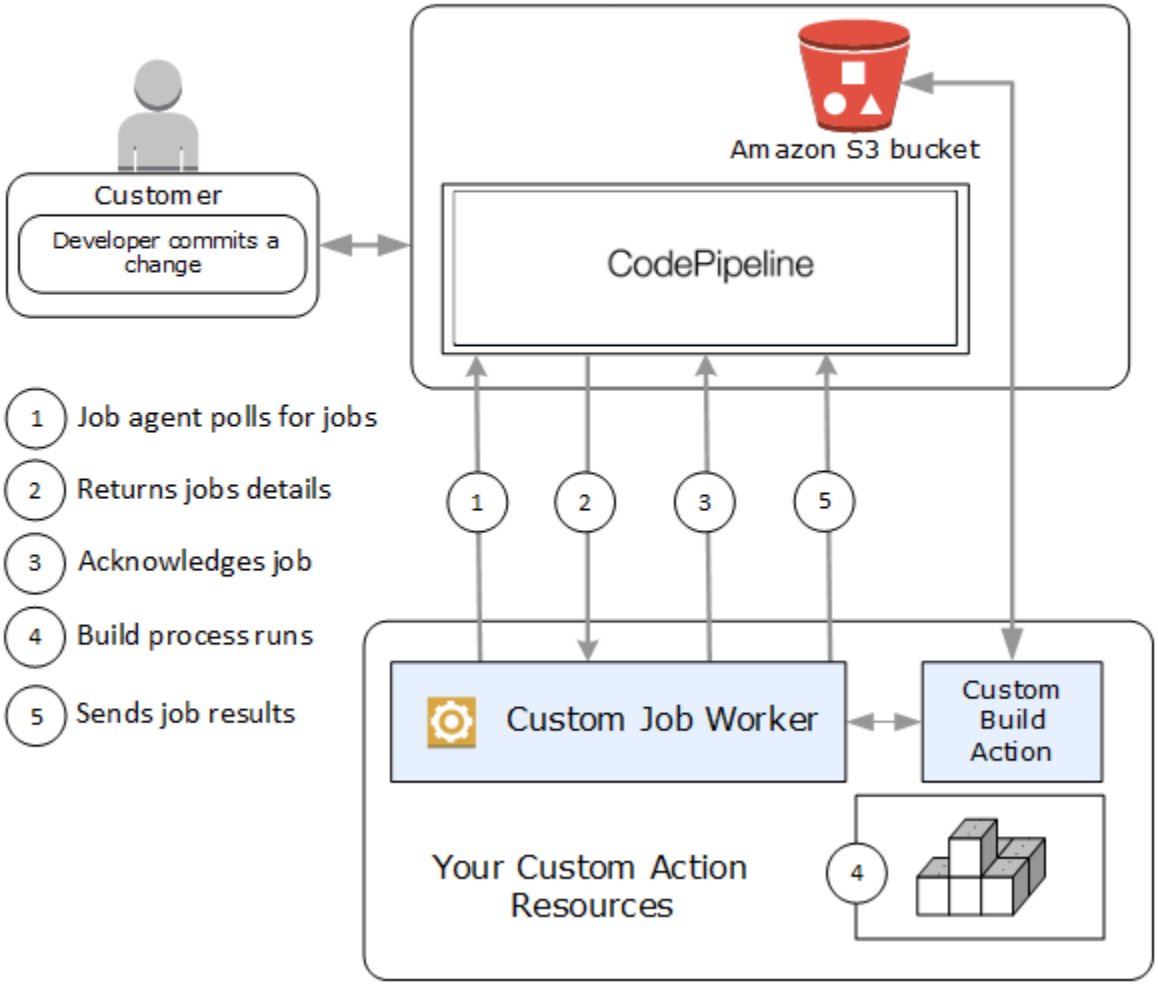
```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codepipeline:PollForJobs",
        "codepipeline:AcknowledgeJob",
        "codepipeline:GetJobDetails",
        "codepipeline:PutJobSuccessResult",
```

```
        "codepipeline:PutJobFailureResult"
      ],
      "Resource": [
        "arn:aws-eusc:codepipeline:us-east-2::actionType:custom/
Build/MyBuildProject/1/"
      ]
    }
  ]
}
```

> **Note**
>
> Consider using the `AWSCodePipelineCustomActionAccess` managed policy.

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.



1. Your job worker polls CodePipeline for jobs using `PollForJobs`.
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). 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 AWS KMS.

  To upload artifacts to the Amazon S3 bucket, you must additionally configure the Amazon S3 `PutObject` request to use encryption. Currently only AWS Key Management Service (AWS KMS) is supported for encryption. AWS KMS uses AWS KMS keys. In order to know whether to use an AWS managed 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 property is set, you should use that customer managed key ID when configuring AWS KMS. If the key property is null, you use the AWS managed key. CodePipeline uses the AWS managed key unless otherwise configured.

  For an example that shows how to create the AWS KMS parameters in Java or .NET, 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.

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 an existing pipeline (CLI)

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:

   ```
   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:

```
,
    {
      "name": "MyBuildStage",
      "actions":  [
              {
                "inputArtifacts": [
                {
                   "name": "MyApp"
                 }
                   ],
                    "name": "MyBuildCustomAction",
                    "actionTypeId": {
                        "category": "Build",
                        "owner": "Custom",
                        "version": "1",
                        "provider": "My-Build-Provider-Name"
                    },
                    "outputArtifacts": [
                        {
                          "name": "MyBuiltApp"
                        }
                    ],
                    "configuration": {
                        "ProjectName": "MyBuildProject"
                    },
                    "runOrder": 1
                }
            ]
        },
        {
            "name": "Staging",
            "actions": [
                    {
                        "inputArtifacts": [
                            {
                                "name": "MyBuiltApp"
                            }
```

```
                        ],
                        "name": "Deploy-CodeDeploy-Application",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeDeploy"
                        },
                        "outputArtifacts": [],
                        "configuration": {
                            "ApplicationName": "CodePipelineDemoApplication",
                            "DeploymentGroupName": "CodePipelineDemoFleet"
                        },
                        "runOrder": 1
                    }
                ]
            }
    ]
}
```

3. 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.

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.

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
- View tags for a custom action
- Edit tags for a custom action
- Remove tags from a custom action

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.

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*:

```
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
```

If successful, this command returns nothing.

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`:

```
aws codepipeline list-tags-for-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:actiontype:Owner/Category/Provider/Version
```

If successful, this command returns information similar to the following:

```
{
    "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:

```
aws codepipeline tag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:actiontype:Owner/Category/Provider/Version --tags
 key=TestActionType,value=IntegrationTest
```

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*:

```
aws codepipeline untag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:actiontype:Owner/Category/Provider/Version --tag-keys TestActionType
```

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.

> **Important**
>
> Do not log the JSON event that CodePipeline sends to Lambda because this can result in user credentials being logged in CloudWatch Logs. The CodePipeline role uses a JSON event to pass temporary credentials to Lambda in the `artifactCredentials` field. For an example event, see Example JSON event.

Here are some ways Lambda functions can be used in pipelines:

- To create resources on demand in one stage of a pipeline using 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.

> **Note**
>
> Creating and running Lambda functions might result in charges to your AWS account. For more information, see Pricing.

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.

> **Note**
>
> When using cross-region Lambda invoke action in CodePipeline, the status of the Lambda execution using the PutJobSuccessResult and PutJobFailureResult should be sent to the AWS Region where the Lambda function is present and not to the Region where CodePipeline exists.

This topic includes sample functions to demonstrate the flexibility of working with Lambda functions in CodePipeline:

- Basic Lambda function
  - 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

- 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.

Topics

- Step 1: Create a pipeline
- Step 2: Create the Lambda function
- Step 3: Add the Lambda function to a pipeline in the CodePipeline console
- Step 4: Test the pipeline with the Lambda function
- Step 5: Next steps
- Example JSON event
- Additional sample functions

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. 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) 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 **Details** tab for the instance, copy the IP address in **Public IPv4 address** (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://eusc-de-east-1.console.amazonaws-eusc.eu/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": "arn:aws-eusc:logs:*:*:*"
    },
    {
      "Action": [
        "codepipeline:PutJobSuccessResult",
        "codepipeline:PutJobFailureResult"
        ],
        "Effect": "Allow",
        "Resource": "*"
     }
  ]
}
```

3. Choose **Review policy**.
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://eusc-de-east-1.console.amazonaws-eusc.eu/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 20.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.

```
import { CodePipelineClient, PutJobSuccessResultCommand,
 PutJobFailureResultCommand } from "@aws-sdk/client-codepipeline";
import http from 'http';
import assert from 'assert';

export const handler = (event, context) => {

    const codepipeline = new CodePipelineClient();

    // Retrieve the Job ID from the Lambda action
    const jobId = event["CodePipeline.job"].id;

    // Retrieve the value of UserParameters from the Lambda action configuration in
 CodePipeline, in this case a URL which will be
```

```
    // health checked by this function.
    const url =
event["CodePipeline.job"].data.actionConfiguration.configuration.UserParameters;

    // Notify CodePipeline of a successful job
    const putJobSuccess = async function(message) {
        const command = new PutJobSuccessResultCommand({
            jobId: jobId
        });
         try {
             await codepipeline.send(command);
             context.succeed(message);
         } catch (err) {
             context.fail(err);
         }
    };

    // Notify CodePipeline of a failed job
    const putJobFailure = async function(message) {
        const command = new PutJobFailureResultCommand({
            jobId: jobId,
            failureDetails: {
                message: JSON.stringify(message),
                type: 'JobFailed',
                externalExecutionId: context.awsRequestId
            }
        });
        await codepipeline.send(command);
        context.fail(message);
    };

    // Validate the URL passed in UserParameters
    if(!url || url.indexOf('http://') === -1) {
        putJobFailure('The UserParameters field must contain a valid URL address to
test, including http:// or https://');
        return;
    }

    // Helper function to make a HTTP GET request to the page.
    // The helper will test the response and succeed or fail the job accordingly
    const getPage = function(url, callback) {
        var pageObject = {
            body: '',
            statusCode: 0,
```

```
            contains: function(search) {
                return this.body.indexOf(search) > -1;
            }
        };
        http.get(url, function(response) {
            pageObject.body = '';
            pageObject.statusCode = response.statusCode;

            response.on('data', function (chunk) {
                pageObject.body += chunk;
            });

            response.on('end', function () {
                callback(pageObject);
            });

            response.resume();
        }).on('error', function(error) {
            // Fail the job if our request failed
            putJobFailure(error);
        });
    };

    getPage(url, function(returnedPage) {
        try {
            // Check if the HTTP response has a 200 status
            assert(returnedPage.statusCode === 200);
            // Check if the page contains the text "Congratulations"
            // You can change this to check for different text, or add other tests
 as required
            assert(returnedPage.contains('Congratulations'));

            // Succeed the job
            putJobSuccess("Tests passed.");
        } catch (ex) {
            // If any of the assertions failed then fail the job
            putJobFailure(ex);
        }
    });
};
```

6. 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 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

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

   > **Note**
   >
   > This topic uses an IP address, but in a real-world scenario, you could provide your registered website name instead (for example, **`http://www.example.com`**). For more

information about event data and handlers in AWS Lambda, see Programming Model in the *AWS Lambda Developer Guide*.

6. On the **Edit action** page, choose **Save**.

Step 4: Test the pipeline with the Lambda function

To test the function, release the most recent change through the pipeline.

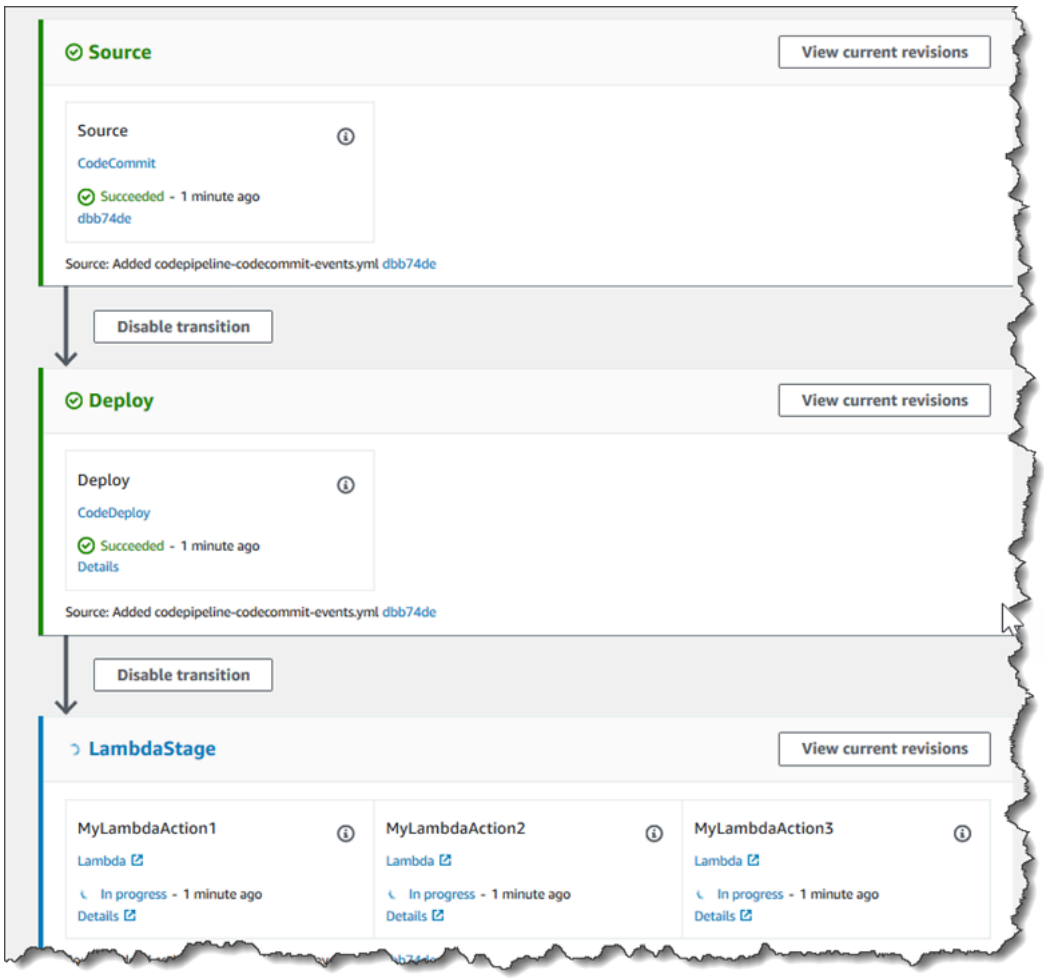
To use the console to run the most recent version of an artifact through a pipeline

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, Delete a pipeline in CodePipeline, 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"
                }
            },
            "inputArtifacts": [
                {
                    "location": {
                        "s3Location": {
                            "bucketName": "the name of the bucket configured as the
pipeline artifact store in Amazon S3, for example codepipeline-us-east-2-1234567890",
                            "objectKey": "the name of the application, for example
CodePipelineDemoApplication.zip"
                        },
                        "type": "S3"
                    },
                    "revision": null,
                    "name": "ArtifactName"
                }
            ],
            "outputArtifacts": [],
            "artifactCredentials": {
                "secretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
                "sessionToken": "MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
3rrszlaEXAMPLE=",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE"
            },
```

```
            "continuationToken": "A continuation token if continuing job",
            "encryptionKey": {
              "id": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
              "type": "KMS"
            }
        }
    }
}
```

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

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 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 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 CloudFormation, Amazon S3, and CodePipeline, as shown in this sample policy:

JSON

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

```
        },
        {
            "Action": [
                "s3:*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

To create the CloudFormation template, open any plain-text editor and copy and paste the following code:

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",
  "Description" : "CloudFormation template which creates an S3 bucket",
  "Resources" : {
    "MySampleBucket" : {
      "Type" : "AWS::S3::Bucket",
      "Properties" : {
      }
    }
  },
  "Outputs" : {
    "BucketName" : {
      "Value" : { "Ref" : "MySampleBucket" },
      "Description" : "The name of the S3 bucket"
    }
  }
}
```

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.

> **Note**
>
> The sample Lambda function expects these file names and compressed structure. However, you can substitute your own 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 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 **Choose or create an execution role**, select **Use 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:

```
from __future__ import print_function
from boto3.session import Session

import json
import urllib
import boto3
import zipfile
import tempfile
import botocore
import traceback

print('Loading function')

cf = boto3.client('cloudformation')
code_pipeline = boto3.client('codepipeline')

def find_artifact(artifacts, name):
    """Finds the artifact 'name' among the 'artifacts'

    Args:
```

```
        artifacts: The list of artifacts available to the function
        name: The artifact we wish to use
    Returns:
        The artifact dictionary found
    Raises:
        Exception: If no matching artifact is found

    """
    for artifact in artifacts:
        if artifact['name'] == name:
            return artifact

    raise Exception('Input artifact named "{0}" not found in event'.format(name))

def get_template(s3, artifact, file_in_zip):
    """Gets the template artifact

    Downloads the artifact from the S3 artifact store to a temporary file
    then extracts the zip and returns the file containing the CloudFormation
    template.

    Args:
        artifact: The artifact to download
        file_in_zip: The path to the file within the zip containing the template

    Returns:
        The CloudFormation template as a string

    Raises:
        Exception: Any exception thrown while downloading the artifact or unzipping
 it

    """
    tmp_file = tempfile.NamedTemporaryFile()
    bucket = artifact['location']['s3Location']['bucketName']
    key = artifact['location']['s3Location']['objectKey']

    with tempfile.NamedTemporaryFile() as tmp_file:
        s3.download_file(bucket, key, tmp_file.name)
        with zipfile.ZipFile(tmp_file.name, 'r') as zip:
            return zip.read(file_in_zip)

def update_stack(stack, template):
    """Start a CloudFormation stack update
```

```
    Args:
        stack: The stack to update
        template: The template to apply

    Returns:
        True if an update was started, false if there were no changes
        to the template since the last update.

    Raises:
        Exception: Any exception besides "No updates are to be performed."

    """
    try:
        cf.update_stack(StackName=stack, TemplateBody=template)
        return True

    except botocore.exceptions.ClientError as e:
        if e.response['Error']['Message'] == 'No updates are to be performed.':
            return False
        else:
            raise Exception('Error updating CloudFormation stack
 "{0}"'.format(stack), e)

def stack_exists(stack):
    """Check if a stack exists or not

    Args:
        stack: The stack to check

    Returns:
        True or False depending on whether the stack exists

    Raises:
        Any exceptions raised .describe_stacks() besides that
        the stack doesn't exist.

    """
    try:
        cf.describe_stacks(StackName=stack)
        return True
    except botocore.exceptions.ClientError as e:
        if "does not exist" in e.response['Error']['Message']:
            return False
```

```
        else:
            raise e

def create_stack(stack, template):
    """Starts a new CloudFormation stack creation

    Args:
        stack: The stack to be created
        template: The template for the stack to be created with

    Throws:
        Exception: Any exception thrown by .create_stack()
    """
    cf.create_stack(StackName=stack, TemplateBody=template)

def get_stack_status(stack):
    """Get the status of an existing CloudFormation stack

    Args:
        stack: The name of the stack to check

    Returns:
        The CloudFormation status string of the stack such as CREATE_COMPLETE

    Raises:
        Exception: Any exception thrown by .describe_stacks()

    """
    stack_description = cf.describe_stacks(StackName=stack)
    return stack_description['Stacks'][0]['StackStatus']

def put_job_success(job, message):
    """Notify CodePipeline of a successful job

    Args:
        job: The CodePipeline job ID
        message: A message to be logged relating to the job status

    Raises:
        Exception: Any exception thrown by .put_job_success_result()

    """
    print('Putting job success')
    print(message)
```

```
    code_pipeline.put_job_success_result(jobId=job)

def put_job_failure(job, message):
    """Notify CodePipeline of a failed job

    Args:
        job: The CodePipeline job ID
        message: A message to be logged relating to the job status

    Raises:
        Exception: Any exception thrown by .put_job_failure_result()

    """
    print('Putting job failure')
    print(message)
    code_pipeline.put_job_failure_result(jobId=job, failureDetails={'message':
 message, 'type': 'JobFailed'})

def continue_job_later(job, message):
    """Notify CodePipeline of a continuing job

    This will cause CodePipeline to invoke the function again with the
    supplied continuation token.

    Args:
        job: The JobID
        message: A message to be logged relating to the job status
        continuation_token: The continuation token

    Raises:
        Exception: Any exception thrown by .put_job_success_result()

    """

    # Use the continuation token to keep track of any job execution state
    # This data will be available when a new job is scheduled to continue the
 current execution
    continuation_token = json.dumps({'previous_job_id': job})

    print('Putting job continuation')
    print(message)
    code_pipeline.put_job_success_result(jobId=job,
 continuationToken=continuation_token)
```

```
def start_update_or_create(job_id, stack, template):
    """Starts the stack update or create process

    If the stack exists then update, otherwise create.

    Args:
        job_id: The ID of the CodePipeline job
        stack: The stack to create or update
        template: The template to create/update the stack with

    """
    if stack_exists(stack):
        status = get_stack_status(stack)
        if status not in ['CREATE_COMPLETE', 'ROLLBACK_COMPLETE',
 'UPDATE_COMPLETE']:
            # If the CloudFormation stack is not in a state where
            # it can be updated again then fail the job right away.
            put_job_failure(job_id, 'Stack cannot be updated when status is: ' +
 status)
            return

        were_updates = update_stack(stack, template)

        if were_updates:
            # If there were updates then continue the job so it can monitor
            # the progress of the update.
            continue_job_later(job_id, 'Stack update started')

        else:
            # If there were no updates then succeed the job immediately
            put_job_success(job_id, 'There were no stack updates')
    else:
        # If the stack doesn't already exist then create it instead
        # of updating it.
        create_stack(stack, template)
        # Continue the job so the pipeline will wait for the CloudFormation
        # stack to be created.
        continue_job_later(job_id, 'Stack create started')

def check_stack_update_status(job_id, stack):
    """Monitor an already-running CloudFormation update/create

    Succeeds, fails or continues the job depending on the stack status.
```

```
    Args:
        job_id: The CodePipeline job ID
        stack: The stack to monitor

    """
    status = get_stack_status(stack)
    if status in ['UPDATE_COMPLETE', 'CREATE_COMPLETE']:
        # If the update/create finished successfully then
        # succeed the job and don't continue.
        put_job_success(job_id, 'Stack update complete')

    elif status in ['UPDATE_IN_PROGRESS', 'UPDATE_ROLLBACK_IN_PROGRESS',
    'UPDATE_ROLLBACK_COMPLETE_CLEANUP_IN_PROGRESS', 'CREATE_IN_PROGRESS',
    'ROLLBACK_IN_PROGRESS', 'UPDATE_COMPLETE_CLEANUP_IN_PROGRESS']:
        # If the job isn't finished yet then continue it
        continue_job_later(job_id, 'Stack update still in progress')

    else:
        # If the Stack is a state which isn't "in progress" or "complete"
        # then the stack update/create has failed so end the job with
        # a failed result.
        put_job_failure(job_id, 'Update failed: ' + status)

def get_user_params(job_data):
    """Decodes the JSON user parameters and validates the required properties.

    Args:
        job_data: The job data structure containing the UserParameters string which
 should be a valid JSON structure

    Returns:
        The JSON parameters decoded as a dictionary.

    Raises:
        Exception: The JSON can't be decoded or a property is missing.

    """
    try:
        # Get the user parameters which contain the stack, artifact and file
 settings
        user_parameters = job_data['actionConfiguration']['configuration']
['UserParameters']
        decoded_parameters = json.loads(user_parameters)
```

```
    except Exception as e:
        # We're expecting the user parameters to be encoded as JSON
        # so we can pass multiple values. If the JSON can't be decoded
        # then fail the job with a helpful message.
        raise Exception('UserParameters could not be decoded as JSON')

    if 'stack' not in decoded_parameters:
        # Validate that the stack is provided, otherwise fail the job
        # with a helpful message.
        raise Exception('Your UserParameters JSON must include the stack name')

    if 'artifact' not in decoded_parameters:
        # Validate that the artifact name is provided, otherwise fail the job
        # with a helpful message.
        raise Exception('Your UserParameters JSON must include the artifact name')

    if 'file' not in decoded_parameters:
        # Validate that the template file is provided, otherwise fail the job
        # with a helpful message.
        raise Exception('Your UserParameters JSON must include the template file
 name')

    return decoded_parameters

def setup_s3_client(job_data):
    """Creates an S3 client

    Uses the credentials passed in the event by CodePipeline. These
    credentials can be used to access the artifact bucket.

    Args:
        job_data: The job data structure

    Returns:
        An S3 client with the appropriate credentials

    """
    key_id = job_data['artifactCredentials']['accessKeyId']
    key_secret = job_data['artifactCredentials']['secretAccessKey']
    session_token = job_data['artifactCredentials']['sessionToken']

    session = Session(aws_access_key_id=key_id,
        aws_secret_access_key=key_secret,
        aws_session_token=session_token)
```

```
    return session.client('s3',
 config=botocore.client.Config(signature_version='s3v4'))

def lambda_handler(event, context):
    """The Lambda function handler

    If a continuing job then checks the CloudFormation stack status
    and updates the job accordingly.

    If a new job then kick of an update or creation of the target
    CloudFormation stack.

    Args:
        event: The event passed by Lambda
        context: The context passed by Lambda

    """
    try:
        # Extract the Job ID
        job_id = event['CodePipeline.job']['id']

        # Extract the Job Data
        job_data = event['CodePipeline.job']['data']

        # Extract the params
        params = get_user_params(job_data)

        # Get the list of artifacts passed to the function
        artifacts = job_data['inputArtifacts']

        stack = params['stack']
        artifact = params['artifact']
        template_file = params['file']

        if 'continuationToken' in job_data:
            # If we're continuing then the create/update has already been triggered
            # we just need to check if it has finished.
            check_stack_update_status(job_id, stack)
        else:
            # Get the artifact details
            artifact_data = find_artifact(artifacts, artifact)
            # Get S3 client to access artifact with
            s3 = setup_s3_client(job_data)
            # Get the JSON template file out of the artifact
```

```
            template = get_template(s3, artifact_data, template_file)
            # Kick off a stack update or create
            start_update_or_create(job_id, stack, template)

    except Exception as e:
        # If any other exceptions which we didn't expect are raised
        # then fail the job and log the exception message.
        print('Function failed due to exception.')
        print(e)
        traceback.print_exc()
        put_job_failure(job_id, 'Function exception: ' + str(e))

    print('Function complete.')
    return "Complete."
```

6. Leave **Handler** at the default value, and leave **Role** at the name you selected or created earlier, **`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. Choose **Edit** for the pipeline stage you want to change, and choose **Add action group**. On the **Edit action** page, in **Action name**, enter a name for your action. In **Action provider**, choose **Lambda**.

   Under **Input artifacts**, choose `MyTemplate`. In **UserParameters**, you must provide a JSON string with three parameters:

   - Stack name
   - CloudFormation template name and path to the file
   - Input artifact

   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: {"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.

Add a manual approval action to a stage

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.

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 roles 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 to 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 policies for AWS CodePipeline.

Grant approval permission to all pipelines and approval actions

For users who need to perform approval actions in CodePipeline, use the `AWSCodePipelineApproverAccess` managed policy.

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

  Create a permission set. Follow the instructions in Create a permission set in the *AWS IAM Identity Center User Guide*.
- Users managed in IAM through an identity provider:

  Create a role for identity federation. Follow the instructions in Create a role for a third-party identity provider (federation) in the *IAM User Guide*.
- IAM users:
  - Create a role that your user can assume. Follow the instructions in Create a role for an IAM user in the *IAM User Guide*.
  - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in Adding permissions to a user (console) in the *IAM User Guide*.

Specify approval permission for specific pipelines and approval actions

For users who need to perform approval actions in CodePipeline, use the following custom policy. In the policy below, specify the individual resources a 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):

> **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`.

To use the JSON policy editor to create a policy

1. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. In the navigation pane on the left, choose **Policies**.

   If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.
3. At the top of the page, choose **Create policy**.
4. In the **Policy editor** section, choose the **JSON** option.
5. Enter the following JSON policy document:

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Effect": "Allow",
           "Action": [
               "codepipeline:ListPipelines"
           ],
           "Resource": [
               "*"
           ]
       },
       {
           "Effect": "Allow",
           "Action": [
               "codepipeline:GetPipeline",
               "codepipeline:GetPipelineState",
               "codepipeline:GetPipelineExecution"
           ],
           "Resource": "arn:aws:codepipeline:us-
east-2:80398EXAMPLE:MyFirstPipeline"
       },
       {
           "Effect": "Allow",
           "Action": [
               "codepipeline:PutApprovalResult"
           ],
           "Resource": "arn:aws:codepipeline:us-
east-2:80398EXAMPLE:MyFirstPipeline/MyApprovalStage/MyApprovalAction"
```

```
        }
    ]
}
```

6. Choose **Next**.

> **Note**
>
> You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see Policy restructuring in the *IAM User Guide*.

7. On the **Review and create** page, enter a **Policy name** and a **Description** (optional) for the policy that you are creating. Review **Permissions defined in this policy** to see the permissions that are granted by your policy.
8. Choose **Create policy** to save your new policy.

Grant Amazon SNS permissions to a CodePipeline 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.

In the policy below, specify the policy for publishing with SNS. For the following policy, you can name it `SNSPublish`. Use the following policy by attaching it to your service role.

> **Important**
>
> Make sure you are signed in to the AWS Management Console with the same account information you used in Getting started with CodePipeline.

JSON

```
{
```

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

To use the JSON policy editor to create a policy

1. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. In the navigation pane on the left, choose **Policies**.

   If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.
3. At the top of the page, choose **Create policy**.
4. In the **Policy editor** section, choose the **JSON** option.
5. Enter or paste a JSON policy document. For details about the IAM policy language, see IAM JSON policy reference.
6. Resolve any security warnings, errors, or general warnings generated during policy validation, and then choose **Next**.

   > **Note**
   >
   > You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see Policy restructuring in the *IAM User Guide*.
7. (Optional) When you create or edit a policy in the AWS Management Console, you can generate a JSON or YAML policy template that you can use in CloudFormation templates.

   To do this, in the **Policy editor** choose **Actions**, and then choose **Generate CloudFormation template**. To learn more about CloudFormation, see AWS Identity and Access Management resource type reference in the *AWS CloudFormation User Guide*.

8. When you are finished adding permissions to the policy, choose **Next**.
9. On the **Review and create** page, enter a **Policy name** and a **Description** (optional) for the policy that you are creating. Review **Permissions defined in this policy** to see the permissions that are granted by your policy.
10. (Optional) Add metadata to the policy by attaching tags as key-value pairs. For more information about using tags in IAM, see Tags for AWS Identity and Access Management resources in the *IAM User Guide*.
11. Choose **Create policy** to save your new 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.

> **Note**
>
> Approval actions can't be added to Source stages. Source stages can contain only source actions.

If you want to use Amazon SNS to send notifications when an approval action is ready for review, you must first complete the following prerequisites:

- Grant permission to your CodePipeline service role to access Amazon SNS resources. For information, see Grant Amazon SNS permissions to a CodePipeline service role.
- Grant permission to one or more IAM identities in your organization to update the status of an approval action. For information, see Grant approval permissions to an IAM user in CodePipeline.

In this example, you create a new approval stage and add a manual approval action to the stage. You can also add a manual approval action to an existing stage that contains other actions.

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.

1. Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
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.
   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, stages, and actions and Edit a pipeline in CodePipeline.

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.

```
{
    "name": "MyApprovalStage",
    "actions": [
        {
            "name": "MyApprovalAction",
            "actionTypeId": {
                "category": "Approval",
                "owner": "AWS",
                "version": "1",
                "provider": "Manual"
            },
            "inputArtifacts": [],
            "outputArtifacts": [],
            "configuration": {
                "NotificationArn": "arn:aws:sns:us-
east-2:80398EXAMPLE:MyApprovalTopic",
                "ExternalEntityLink": "http://example.com",
                "CustomData": "The latest changes include feedback from Bob."},
            "runOrder": 1
        }
    ]
}
```

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.

```
,
{
```

```
    "name": "Production",
    "actions": [
        {
            "inputArtifacts": [],
            "name": "MyApprovalAction",
            "actionTypeId": {
                "category": "Approval",
                "owner": "AWS",
                "version": "1",
                "provider": "Manual"
            },
            "outputArtifacts": [],
            "configuration": {
                "NotificationArn": "arn:aws:sns:us-
east-2:80398EXAMPLE:MyApprovalTopic",
                "ExternalEntityLink": "http://example.com",
                "CustomData": "The latest changes include feedback from Bob."
            },
            "runOrder": 1
        },
        {
            "inputArtifacts": [
                {
                    "name": "MyApp"
                }
            ],
            "name": "MyDeploymentAction",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "version": "1",
                "provider": "CodeDeploy"
            },
            "outputArtifacts": [],
            "configuration": {
                "ApplicationName": "MyDemoApplication",
                "DeploymentGroupName": "MyProductionFleet"
            },
            "runOrder": 2
        }
    ]
}
```

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."

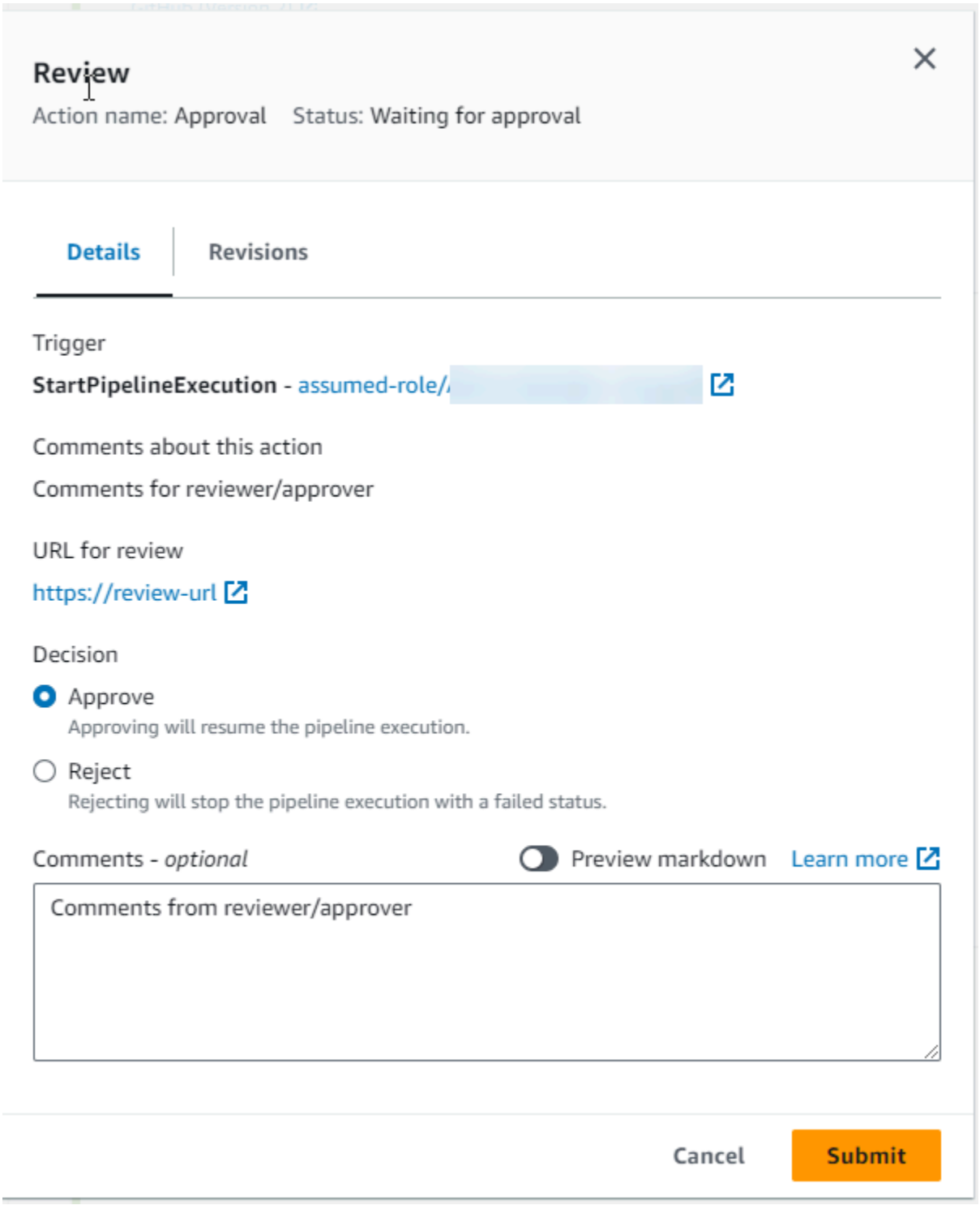
If the person who added the approval action to the pipeline configured notifications, you might receive an email with the pipeline information and status for approval.

Approve or reject an approval action (console)

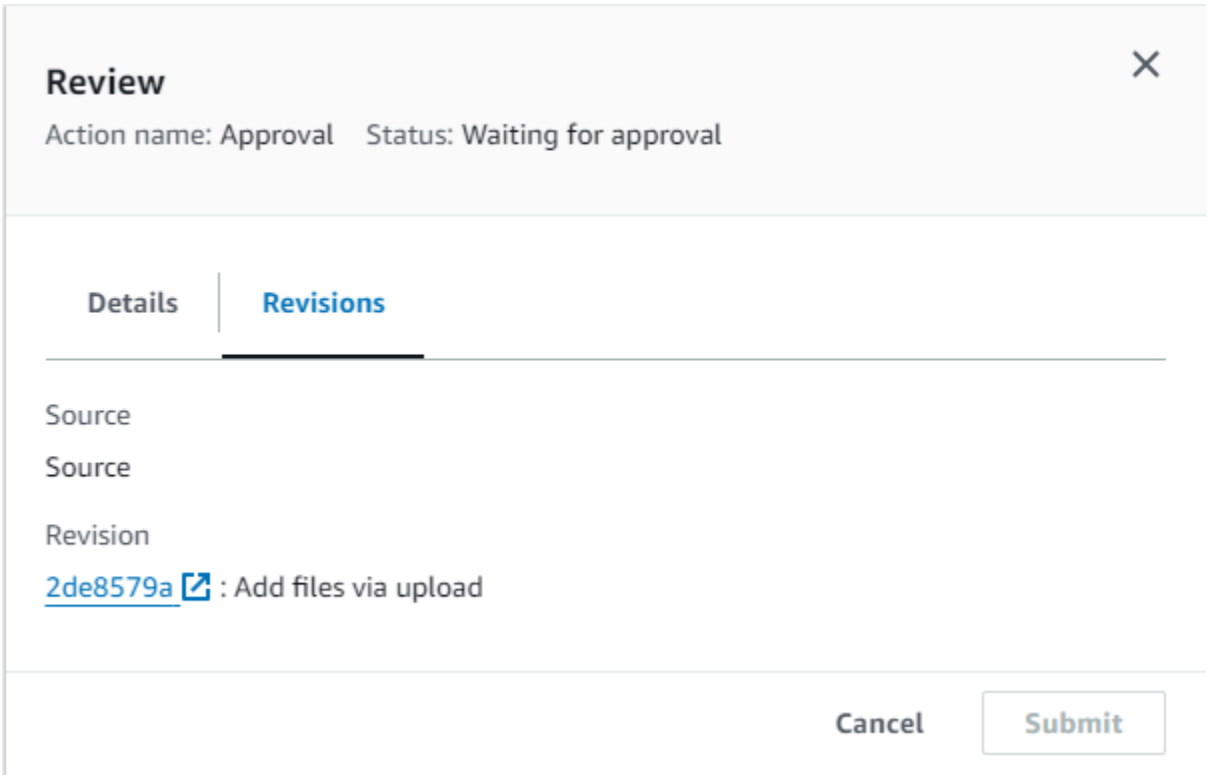
If you receive a notification that includes a direct link to an approval action, choose the **Approve or reject** link, sign in to the console, and then continue with step 7 here. Otherwise, follow all of these steps.

1. Open the CodePipeline console at https://eusc-de-east-1.console.amazonaws-eusc.eu/codepipeline/.
2. On the **All Pipelines** page, choose the name of the pipeline.
3. Locate the stage with the approval action. Choose **Review**.

   The **Review** dialog box displays. The **Details** tab shows the review content and comments.



The **Revisions** tab shows the source revisions for the execution.



4. On the **Details** tab, 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 **URL for review** link in the action to open the target webpage, and then review the content.
6. In the **Review** window, enter review comments, such as why you are approving or rejecting the action, and then choose **Approve** or **Reject**.
7. Choose **Submit**.

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:

```
aws 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:

```
{
    "created": 1467929497.204,
    "pipelineName": "MyFirstPipeline",
    "pipelineVersion": 1,
    "stageStates": [
        {
            "actionStates": [
                {
                    "actionName": "MyApprovalAction",
                    "currentRevision": {
                        "created": 1467929497.204,
                        "revisionChangeId": "CEM7d6Tp7zfelUSLCPPwo234xEXAMPLE",
                        "revisionId": "HYGp7zmwbCPPwo23xCMdTeqIlEXAMPLE"
                    },
                    "latestExecution": {
                        "lastUpdatedBy": "identity",
                        "summary": "The new design needs to be reviewed before
 release.",
                        "token": "1a2b3c4d-573f-4ea7-a67E-XAMPLETOKEN"
                    }
                }
//More content might appear here
}
```

3. 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:

```
{
  "pipelineName": "MyFirstPipeline",
```

```
  "stageName": "MyApprovalStage",
  "actionName": "MyApprovalAction",
  "token": "1a2b3c4d-573f-4ea7-a67E-XAMPLETOKEN",
  "result": {
    "status": "Approved",
    "summary": "The new design looks good. Ready to release to customers."
  }
}
```

4. 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:

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

```
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 and Invoking Lambda Functions Using Amazon SNS Notifications 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**
>
> Cross-region actions are supported and can only be created in those AWS Regions where CodePipeline is supported. For a list of the supported AWS Regions for CodePipeline, see Quotas in AWS CodePipeline.

You can use the console, AWS CLI, or CloudFormation to add cross-region actions in pipelines.

> **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.

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.

> **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, 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

> **Note**
>
> When using cross-region Lambda invoke action in CodePipeline, the status of the Lambda execution using the PutJobSuccessResult and PutJobFailureResult should be sent to the AWS Region where the Lambda function is present and not to the Region where CodePipeline exists.

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 custom pipeline (console).

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.

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

1. Sign in to the console at http://console.aws.amazon.com/codesuite/codepipeline/home.
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

1. Sign in to the console at http://console.aws.amazon.com/codesuite/codepipeline/home.
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

1. Sign in to the console at http://console.aws.amazon.com/codesuite/codepipeline/home.
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`.

```
{
                "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*.

```
"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`.

```
        "artifactStores": {
            "us-west-2": {
                "type": "S3",
                "location": "my-storage-bucket"
            },
            "us-east-1": {
                "type": "S3",
                "location": "my-storage-bucket-us-east-1"
            }
        },
```

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:

```
"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.

```
{
    "pipeline": {
        "version": 4,
        "roleArn": "ARN",
        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "inputArtifacts": [],
                        "name": "Source",
                        "actionTypeId": {
                            "category": "Source",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeCommit"
                        },
                        "outputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "configuration": {
                            "PollForSourceChanges": "false",
                            "BranchName": "main",
                            "RepositoryName": "MyTestRepo"
                        },
                        "runOrder": 1
                    }
                ]
            },
```

```
            {
                "name": "Deploy",
                "actions": [
                    {
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "name": "Deploy",
                        "region": "us-east-1",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeDeploy"
                        },
                        "outputArtifacts": [],
                        "configuration": {
                            "ApplicationName": "name",
                            "DeploymentGroupName": "name"
                        },
                        "runOrder": 1
                    }
                ]
            }
        ],
        "name": "AnyCompanyPipeline",
        "artifactStores": {
            "us-west-2": {
                "type": "S3",
                "location": "my-storage-bucket"
            },
            "us-east-1": {
                "type": "S3",
                "location": "my-storage-bucket-us-east-1"
            }
        }
    }
}
```

> **Note**
>
> 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 action is displayed in the console.

Add a cross-region action to a pipeline (CloudFormation)

You can use CloudFormation to add a cross-region action to an existing pipeline.

To add a cross-region action with 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
                ItemType:
                  Type: InputArtifact
              Name:
                Type: String
                Required: true
              OutputArtifacts:
                Type: Array
                ItemType:
                  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:
  SecondRegionMap:
    RegionA:
      SecondRegion: "RegionB"
    RegionB:
      SecondRegion: "RegionA"

...

          Properties:
            ArtifactStores:
              -
                Region: RegionB
                ArtifactStore:
                  Type: "S3"
                  Location: test-cross-region-artifact-store-bucket-RegionB
              -
                Region: RegionA
                ArtifactStore:
                  Type: "S3"
                  Location: test-cross-region-artifact-store-bucket-RegionA
```

The following YAML example shows the *RegionA* bucket as `us-west-2` and adds the new *RegionB* bucket, `eu-central-1`:

```
Mappings:
  SecondRegionMap:
    us-west-2:
      SecondRegion: "eu-central-1"
    eu-central-1:
      SecondRegion: "us-west-2"

...

          Properties:
            ArtifactStores:
              -
                Region: eu-central-1
                ArtifactStore:
                  Type: "S3"
                  Location: test-cross-region-artifact-store-bucket-eu-central-1
              -
                Region: us-west-2
                ArtifactStore:
                  Type: "S3"
                  Location: test-cross-region-artifact-store-bucket-us-west-2
```

3. Save the updated template to your local computer, and then open the 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 CloudFormation. These are the changes to be made to the stack. You should see your new resources in the list.
6. Choose **Execute**.

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.

- You configure the downstream action to consume the variables generated by the action.

  You can view the details for each action execution to see the values for each output variable that was generated by the action in execution-time.

To see step-by-step examples of using variables:

- For a tutorial with a Lambda action that uses variables from an upstream action (CodeCommit) and generates output variables, see Tutorial: Using variables with Lambda invoke actions.
- For a tutorial with a CloudFormation action that references stack output variables from an upstream CloudFormation action, see Tutorial: Create a pipeline that uses variables from AWS CloudFormation deployment actions.
- For an example manual approval action with message text that references output variables that resolve to the CodeCommit commit ID and commit message, see Example: Use variables in manual approvals.
- For an example CodeBuild action with an environment variable that resolves to the GitHub branch name, see Example: Use a BranchName variable with CodeBuild environment variables.
- CodeBuild actions produce as variables all environment variables that were exported as part of the build. For more information, see CodeBuild action output variables. For a list of the environment variables you can use in CodeBuild, see Environment variables in build environments in the *AWS CodeBuild User Guide*.

Topics

- Configure actions for variables
- View output variables
- Example: Use variables in manual approvals
- Example: Use a BranchName variable with CodeBuild environment variables

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.

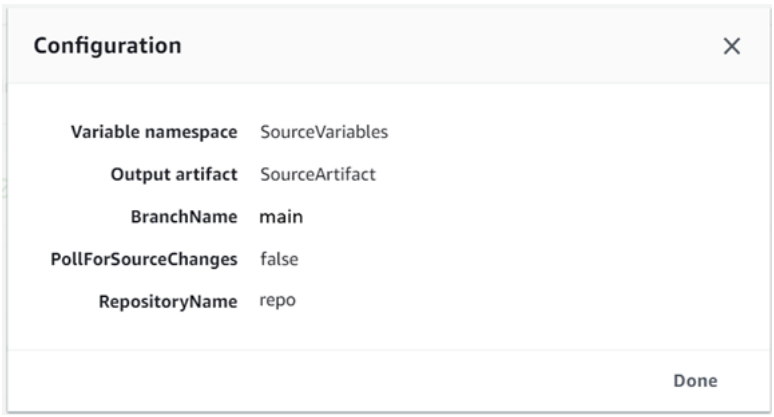
To create a pipeline with variables

1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home.
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**. For example, enter the execution ID with the variable syntax `#{codepipeline.PipelineExecutionId}` and commit ID with the variable syntax `#{SourceVariables.CommitId}`.

   > **Note**
   >
   > You can enter variable syntax in any action configuration field in the wizard.

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 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

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

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

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, stages, and actions 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`.

```
. . .
{
          "inputArtifacts": [],
          "name": "Source",
          "region": "us-west-2",
          "namespace": "SourceVariables",
          "actionTypeId": {
            "category": "Source",
            "owner": "AWS",
            "version": "1",
            "provider": "CodeCommit"
          },
          "outputArtifacts": [

. . .
```

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}`.

```
{
    "name": "Build",
    "actions": [
        {
            "outputArtifacts": [
                {
                    "name": "BuildArtifacts"
                }
            ],
            "name": "Build",
            "configuration": {
                "EnvironmentVariables": "[{\"name\":\"Execution_ID\",\"value
\":\"#{codepipeline.PipelineExecutionId}\",\"type\":\"PLAINTEXT\"},{\"name\":
\"Commit_ID\",\"value\":\"#{SourceVariables.CommitId}\",\"type\":\"PLAINTEXT\"}]",
                "ProjectName": "env-var-test"
            },
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-west-2",
            "actionTypeId": {
                "provider": "CodeBuild",
                "category": "Build",
                "version": "1",
                "owner": "AWS"
            },
            "runOrder": 1
        }
    ]
},
```

2. 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
```

View output variables

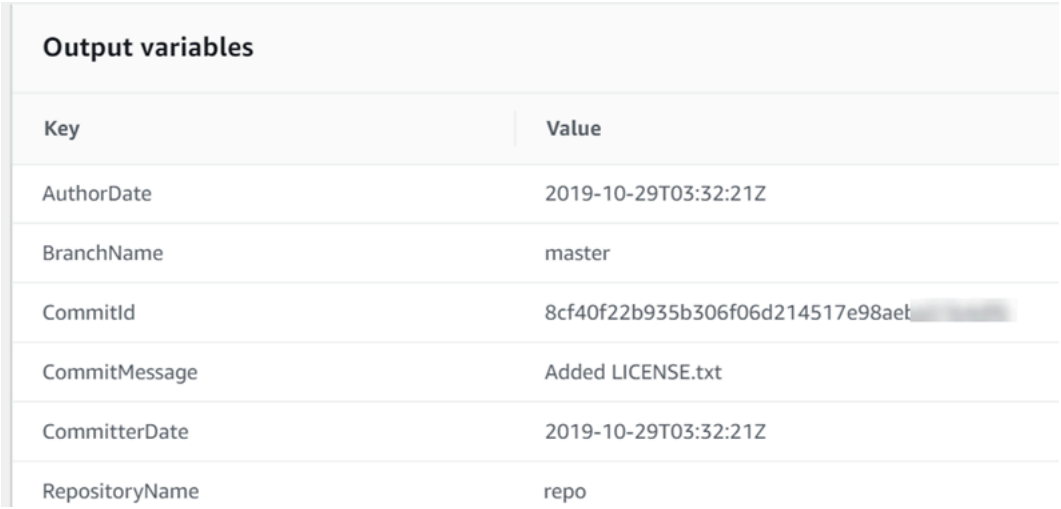
You can view the action execution details to view the variables for that action, specific to each execution.

View variables (console)

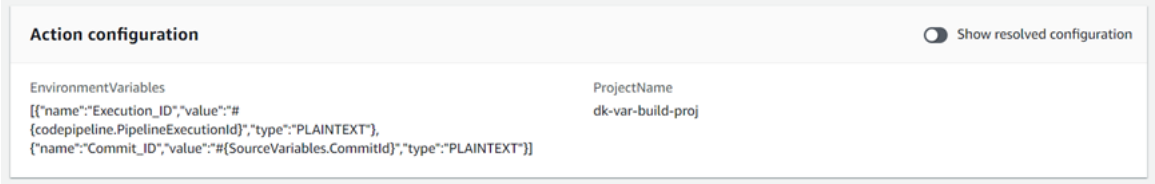
You can use the console to view variables for an action.

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

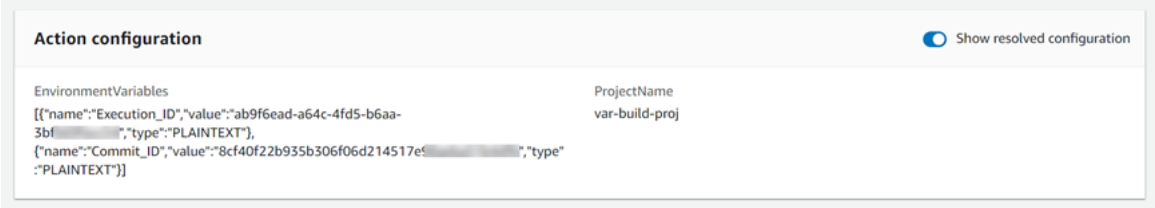
   The names of all pipelines associated with your AWS account are displayed.
2. In **Name**, choose the name of the pipeline.
3. Choose **View history**.
4. After the pipeline runs successfully, you can view the variables produced by the source action. Choose **View history**. Choose **Source** in the 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**.



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": "main",
                    "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",
            "configuration": {
                "EnvironmentVariables": "[{\"name\":\"Execution_ID\",\"value
\":\"#{codepipeline.PipelineExecutionId}\",\"type\":\"PLAINTEXT\"},{\"name\":
\"Commit_ID\",\"value\":\"#{SourceVariables.CommitId}\",\"type\":\"PLAINTEXT\"}]",
                "ProjectName": "env-var-test"
            },
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-west-2",
            "actionTypeId": {
                "provider": "CodeBuild",
                "category": "Build",
                "version": "1",
                "owner": "AWS"
            },
            "runOrder": 1
        }
    ]
},
```

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.

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. 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.
3. In **Action provider**, choose **Manual approval**.
4. In **URL for review**, add the variable syntax for `CommitId` to your CodeCommit URL. 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}`.

   In **Comments**, in `CommitMessage`, enter the commit message:

   ```
   Please approve this change. Commit message: #{SourceVariables.CommitMessage}
   ```
5. After the pipeline runs successfully, you can view the variable values in the approval message.

Example: Use a BranchName variable with CodeBuild environment variables

When you add a CodeBuild action to your pipeline, you can use CodeBuild environment variables to reference a `BranchName` output variable from an upstream source action. With an output variable from an action in CodePipeline, you can create your own CodeBuild environment variables for use in your build commands.

This example shows you how to add output variable syntax from a GitHub source action to a CodeBuild environment variable. The output variable syntax in this example represents the GitHub source action output variable for `BranchName`. After the action runs successfully, the variable resolves to show the GitHub branch name.

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. Choose the pipeline you want to add the approval to.

2. To edit your pipeline, choose **Edit**. On the stage that contains your CodeBuild action, choose **Edit stage**.
3. Choose the icon to edit your CodeBuild action.
4. On the **Edit action** page, under **Environment variables**, enter the following:

   - In **Name**, enter a name for your environment variable.
   - In **Value**, enter the variable syntax for your pipeline output variable, which includes the namespace assigned to your source action. For example, the output variable syntax for a GitHub action with the default namespace `SourceVariables` is `#{SourceVariables.BranchName}`.
   - In **Type**, choose **Plaintext**.
5. After the pipeline runs successfully, you can see how the resolved output variable is the value in the environment variable. Choose one of the following:

   - **CodePipeline console:** Choose your pipeline, and then choose **History**. Choose the most recent pipeline execution.
     - Under **Timeline**, choose the selector for **Source**. This is the source action that generates GitHub output variables. Choose **View execution details**. Under **Output variables**, view the list of output variables generated by this action.
     - Under **Timeline**, choose the selector for **Build**. This is the build action that specifies the CodeBuild environment variables for your build project. Choose **View execution details**. Under **Action configuration**, view your CodeBuild environment variables. Choose **Show resolved configuration**. Your environment variable value is the resolved `BranchName` output variable from the GitHub source action. In this example, the resolved value is `main`.

       For more information, see View variables (console).
   - **CodeBuild console:** Choose your build project and choose the link for your build run. Under **Environment variables**, your resolved output variable is the value for the CodeBuild environment variable. In this example, the environment variable **Name** is `BranchName` and the **Value** is the resolved `BranchName` output variable from the GitHub source action. In this example, the resolved value is `main`.

Build logs
Phase details
Reports
Environment variables
Build details
Resource utilization
Name
Value
Type
BranchName
main
PLAINTEXT

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 Add a manual approval action to a stage.

Topics

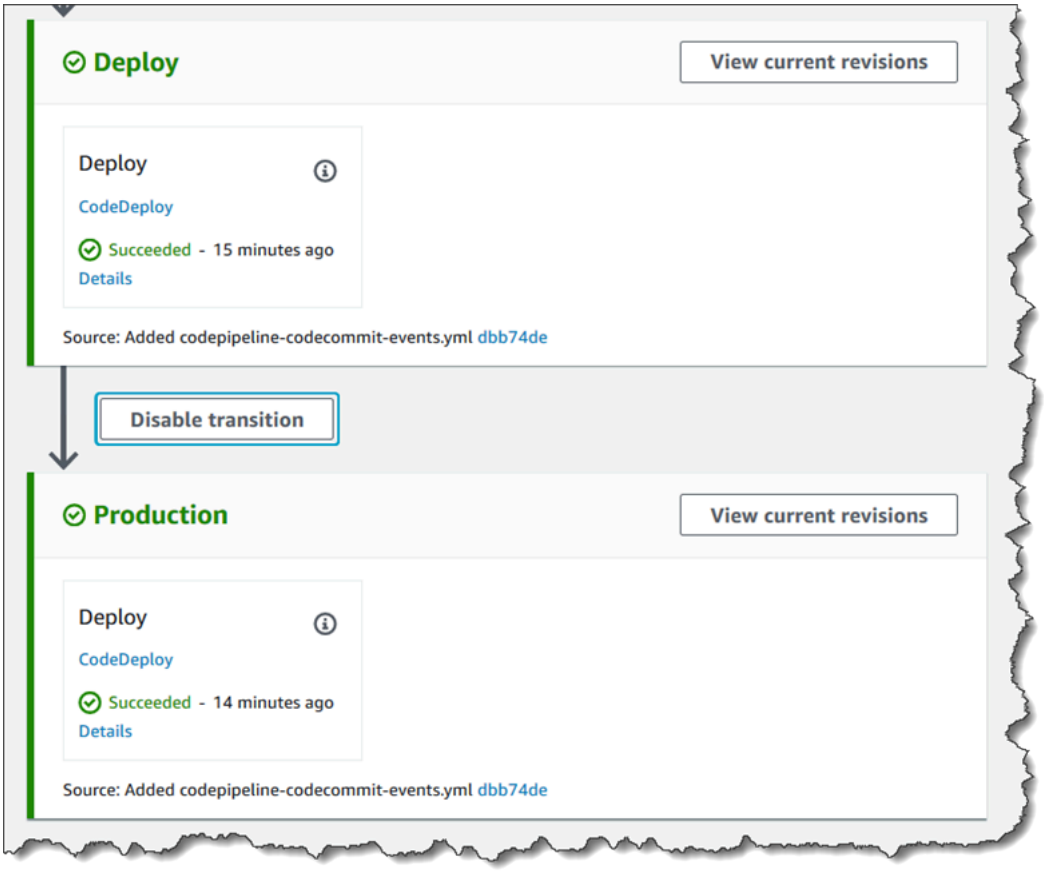
- Disable or enable transitions (console)
- Disable or enable transitions (CLI)

Disable or enable transitions (console)

To disable or enable transitions in a pipeline

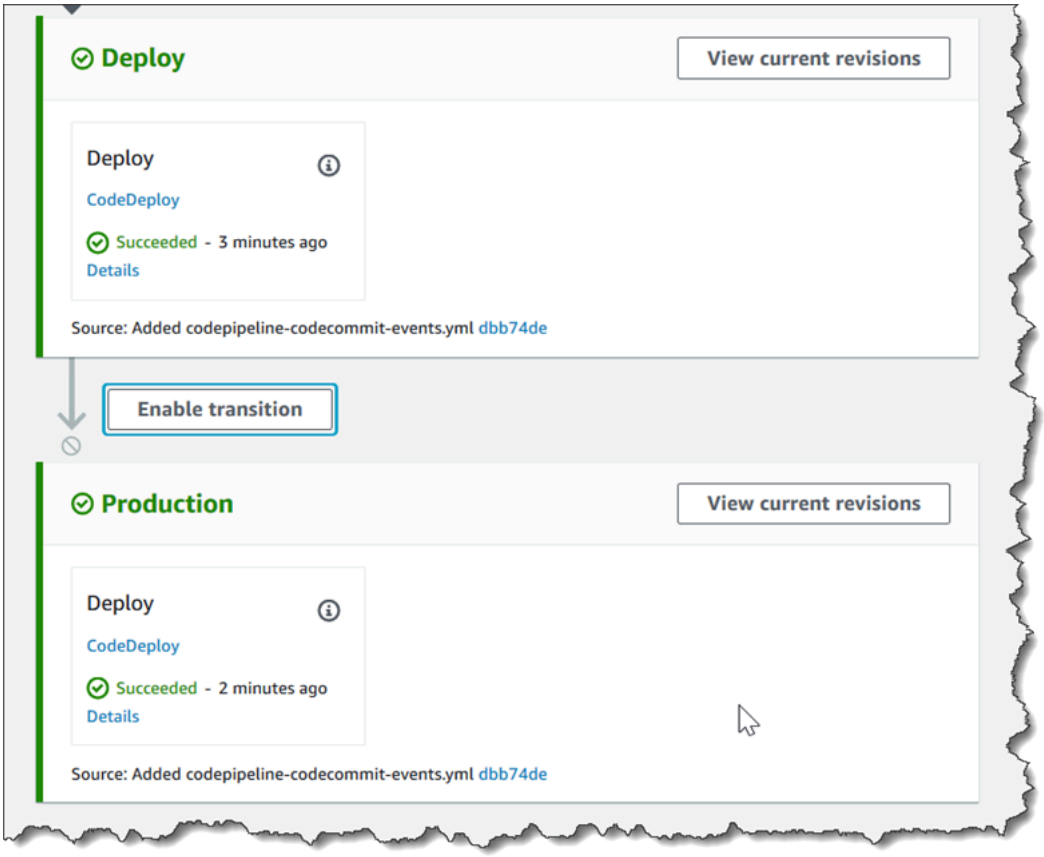
1. Sign in to the AWS Management Console and open the CodePipeline console at http://console.aws.amazon.com/codesuite/codepipeline/home.

   The names 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:

   ```
   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 pipelines (console) and View pipeline details and history (CLI).

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:

   ```
   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 enabled, either view the pipeline in the CodePipeline console or run the **get-pipeline-state** command. For more information, see View pipelines (console) and View pipeline details and history (CLI).

Monitoring pipelines

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:

- **EventBridge event bus events** — You can monitor CodePipeline events in EventBridge, which detects changes in your pipeline, stage, or action execution status. EventBridge routes that data to targets such as AWS Lambda and Amazon Simple Notification Service. EventBridge events are the same as those that appear in Amazon CloudWatch Events.
- **Notifications for pipeline events in the Developer Tools console** — You can monitor CodePipeline events with notifications that you set up in the console and then create an Amazon Simple Notification Service topic and subscription for. For more information, see What are notifications in the *Developer Tools Console User Guide*.
- **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

- Monitoring CodePipeline events
- Events placeholder bucket reference

- Logging CodePipeline API calls with AWS CloudTrail
- CodePipeline CloudWatch metrics

Monitoring CodePipeline events

You can monitor CodePipeline events in EventBridge, which delivers a stream of real-time data from your own applications, software-as-a-service (SaaS) applications, and AWS services. EventBridge routes that data to targets such as AWS Lambda and Amazon Simple Notification Service. These events are the same as those that appear in Amazon CloudWatch Events, which delivers a near real-time stream of system events that describe changes in AWS resources. For more information, see What Is Amazon EventBridge? in the *Amazon EventBridge User Guide*.

> **Note**
>
> Amazon EventBridge is the preferred way to manage your events. Amazon CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch Events or EventBridge will appear in each console.

Events are composed of rules. A rule is configured by choosing the following:

- **Event Pattern.** Each rule is expressed as an event pattern with the source and type of events to monitor, and event targets. To monitor events, you create a rule with the service you are monitoring as the event source, such as CodePipeline. For example, you can create a rule with an event pattern that uses CodePipeline as an event source to trigger the rule when there are changes in the state of a pipeline, stage, or action.
- **Targets.** The new rule receives a selected service as the event target. You might want to set up a target service to send notifications, capture state information, take corrective action, initiate events, or take other actions. When you add your target, you must also grant permissions to EventBridge to allow it to invoke the selected target service.

Each type of execution state change event emits notifications with specific message content, where:

- The initial `version` entry shows the version number for the 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.
- The `pipeline-execution-attempt` entry shows the number of attempts, or retries, for the specific execution ID.

CodePipeline reports an event to EventBridge whenever the state of a resource in your AWS account changes. Events are emitted on a guaranteed, at-least-once basis for the following resources:

- Pipeline executions
- Stage executions
- Action executions

Events are emitted by EventBridge with the event pattern and schema detailed above. For processed events, such as events you receive through notifications you have configured in the Developer Tools console, the event message includes event pattern fields with some variation. For example, the `detail-type` field is converted to `detailType`. For more information, refer to the **PutEvents** API call in the Amazon EventBridge API Reference.

The following examples show events for CodePipeline. Where possible, each example shows the schema for an emitted event along with the schema for a processed event.

Topics

- Detail types
- Pipeline-level events
- Stage-level events
- Action-level events
- Create a Rule That Sends a Notification on a Pipeline Event

Detail types

When you set up events to monitor, you can choose the detail type for the event.

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"`.

> **Note**
>
> Events emitted by EventBridge contain the `detail-type` parameter, which is converted to `detailType` when events are processed.

Detail type	State	Description
CodePipeline Pipeline Execution State Change	CANCELED	The pipeline execution was canceled because the pipeline structure was updated.
	FAILED	The pipeline execution was not completed successfully.
	RESUMED	A failed pipeline execution has been retried in response to the `RetryStageExecution` API call.
	STARTED	The pipeline execution is currently running.
	STOPPED	The stopping process is complete, and the pipeline execution is stopped.
	STOPPING	The pipeline execution is stopping due to a request to either stop and wait or stop and abandon the pipeline execution.
	SUCCEEDED	The pipeline execution was completed successfully.

Detail type	State	Description
	SUPERSEDED	While this pipeline execution was waiting for the next stage to be completed, a newer pipeline execution advanced and continued through the pipeline instead.
CodePipeline Stage Execution State Change	CANCELED	The stage was canceled because the pipeline structure was updated.
	FAILED	The stage was not completed successfully.
	RESUMED	A failed stage has been retried in response to the `RetryStageExecution` API call.
	STARTED	The stage is currently running.
	STOPPED	The stopping process is complete, and the stage execution is stopped.
	STOPPING	The stage execution is stopping due to a request to either stop and wait or stop and abandon the pipeline execution.
	SUCCEEDED	The stage was completed successfully.
CodePipeline Action Execution State Change	ABANDONED	The action is abandoned due to a request to stop and abandon the pipeline execution.
	CANCELED	The action was canceled because the pipeline structure was updated.
	FAILED	For approval actions, the FAILED state means the action was either rejected by the reviewer or failed due to an incorrect action configuration.
	STARTED	The action is currently running.
	SUCCEEDED	The action was completed successfully.

Pipeline-level events

Pipeline-level events are emitted when there is a state change for a pipeline execution.

Topics

- Pipeline STARTED event
- Pipeline STOPPING event
- Pipeline SUCCEEDED event
- Pipeline SUCCEEDED (example with Git tags)
- Pipeline FAILED event
- Pipeline FAILED (example with Git tags)

Pipeline STARTED event

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. The `id` field represents the event ID, and the `account` field represents the account ID where the pipeline is created.

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:03:07Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "trigger-type": "StartPipelineExecution",
            "trigger-detail": "arn:aws:sts::123456789012:assumed-role/Admin/my-user"
```

```
        },
        "state": "STARTED",
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Pipeline Execution State Change",
    "region": "us-east-1",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:44:50Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
east-1:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "trigger-type": "StartPipelineExecution",
            "trigger-detail": "arn:aws:sts::123456789012:assumed-role/Admin/my-user"
        },
        "state": "STARTED",
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "additionalAttributes": {}
}
```

Pipeline STOPPING event

When a pipeline execution is stopping, it emits an event that sends notifications with the following content. This example is for the pipeline named `myPipeline` in the `us-west-2` Region.

```
{
    "version": "0",
```

```
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:02:20Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "STOPPING",
        "version": 3.0,
        "pipeline-execution-attempt": 1.0
        "stop-execution-comments": "Stopping the pipeline for an update"
    }
}
```

Pipeline SUCCEEDED event

When a pipeline execution succeeds, 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.

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:03:44Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "SUCCEEDED",
```

```
        "version": 3.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Pipeline Execution State Change",
    "region": "us-east-1",
    "source": "aws.codepipeline",
    "time": "2021-06-30T22:13:51Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "SUCCEEDED",
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {}
}
```

Pipeline SUCCEEDED (example with Git tags)

When a pipeline execution has a stage that has been retried and succeeded, it emits an event that sends notifications with the following content. This example is for the pipeline named `myPipeline` in the `eu-central-1` Region where the `execution-trigger` is configured for Git tags.

> **Note**
>
> The `execution-trigger` field will have either `tag-name` or `branch-name`, depending on what kind of event triggered the pipeline.

```
{
    "version": "0",
    "id": "b128b002-09fd-4574-4eba-27152726c777",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2023-10-26T13:50:53Z",
    "region": "eu-central-1",
    "resources": [
        "arn:aws:codepipeline:eu-central-1:123456789012:BuildFromTag"
    ],
    "detail": {
        "pipeline": "BuildFromTag",
        "execution-id": "e17b5773-cc0d-4db2-9ad7-594c73888de8",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "author-display-name": "Mary Major",
            "full-repository-name": "mmajor/sample-project",
            "provider-type": "GitLab",
            "author-email": "email_address",
            "commit-message": "Update file README.md",
            "author-date": "2023-08-16T21:08:08Z",
            "tag-name": "gitlab-v4.2.1",
            "commit-id": "commit_ID",
            "connection-arn": "arn:aws:codestar-connections:eu-
central-1:123456789012:connection/0f5b706a-1a1d-46c5-86b6-f177321bcfb2",
            "author-id": "Mary Major"
        },
        "state": "SUCCEEDED",
        "version": 32.0,
        "pipeline-execution-attempt": 1.0
    }
}
```


Pipeline FAILED event

When a pipeline execution fails, it emits an event that sends notifications with the following content. This example is for the pipeline named "myPipeline" in the us-west-2 Region.

Emitted event


```
{
    "version": "0",
```

```
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:55:43Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "FAILED",
        "version": 4.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Pipeline Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:46:16Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "FAILED",
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {
        "failedActionCount": 1,
        "failedActions": [
```

```
            {
                "action": "Deploy",
                "additionalInformation": "Deployment <ID> failed"
            }
        ],
        "failedStage": "Deploy"
    }
```

Pipeline FAILED (example with Git tags)

Unless it fails at the source stage, for a pipeline configure with triggers, it emits an event that sends notifications with the following content. This example is for the pipeline named `myPipeline` in the `eu-central-1` Region where the `execution-trigger` is configured for Git tags.

> **Note**
>
> The `execution-trigger` field will have either `tag-name` or `branch-name`, depending on what kind of event triggered the pipeline.

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:55:43Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "author-display-name": "Mary Major",
            "full-repository-name": "mmajor/sample-project",
            "provider-type": "GitLab",
```

```
            "author-email": "email_address",
            "commit-message": "Update file README.md",
            "author-date": "2023-08-16T21:08:08Z",
            "tag-name": "gitlab-v4.2.1",
            "commit-id": "commit_ID",
            "connection-arn": "arn:aws:codestar-connections:eu-
central-1:123456789012:connection/0f5b706a-1a1d-46c5-86b6-f177321bcfb2",
            "author-id": "Mary Major"
        },
        "state": "FAILED",
        "version": 4.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Pipeline Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:46:16Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "author-display-name": "Mary Major",
            "full-repository-name": "mmajor/sample-project",
            "provider-type": "GitLab",
            "author-email": "email_address",
            "commit-message": "Update file README.md",
            "author-date": "2023-08-16T21:08:08Z",
            "tag-name": "gitlab-v4.2.1",
            "commit-id": "commit_ID",
            "connection-arn": "arn:aws:codestar-connections:eu-
central-1:123456789012:connection/0f5b706a-1a1d-46c5-86b6-f177321bcfb2",
            "author-id": "Mary Major"
        },
        "state": "FAILED",
```

```
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {
        "failedActionCount": 1,
        "failedActions": [
            {
                "action": "Deploy",
                "additionalInformation": "Deployment <ID> failed"
            }
        ],
        "failedStage": "Deploy"
    }
```

Stage-level events

Stage-level events are emitted when there is a state change for a stage execution.

Topics

- Stage STARTED event
- Stage STOPPING event
- Stage STOPPED event
- Stage RESUMED after stage retry event

Stage STARTED event

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`.

Emitted event

```
{
    "version": "0",
    "id": 01234567-EXAMPLE,
    "detail-type": "CodePipeline Stage Execution State Change",
```

```
    "source": "aws.codepipeline",
    "account": 123456789012,
    "time": "2020-01-24T22:03:07Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "version": 1.0,
        "execution-id": 12345678-1234-5678-abcd-12345678abcd,
        "start-time": "2023-10-26T13:49:39.208Z",
        "stage": "Prod",
        "state": "STARTED",
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Stage Execution State Change",
    "region": "us-east-1",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:45:40Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "stage": "Source",
        "state": "STARTED",
        "version": 1.0,
        "pipeline-execution-attempt": 0.0
    },
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "additionalAttributes": {
        "sourceActions": [
            {
```

```
                "sourceActionName": "Source",
                "sourceActionProvider": "CodeCommit",
                "sourceActionVariables": {
                    "BranchName": "main",
                    "CommitId": "<ID>",
                    "RepositoryName": "my-repo"
                }
            }
        ]
    }
}
```

Stage STOPPING event

When a stage execution is stopping, it emits an event that sends notifications with the following content. This example is for the pipeline named `myPipeline` in the `us-west-2` Region, for the stage `Deploy`.

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Stage Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:02:20Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "stage": "Deploy",
        "state": "STOPPING",
        "version": 3.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Stage STOPPED event

When a stage execution is stopped, it emits an event that sends notifications with the following content. This example is for the pipeline named `myPipeline` in the `us-west-2` Region, for the stage `Deploy`.

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Stage Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:21:39Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "stage": "Deploy",
        "state": "STOPPED",
        "version": 3.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Stage RESUMED after stage retry event

When a stage execution is resumed and has a stage that has been retried, it emits an event that sends notifications with the following content.

When a stage has been retried, the `stage-last-retry-attempt-time` field displays, as shown in the example. The field displays on all stage events if a retry was performed.

> **Note**
>
> The `stage-last-retry-attempt-time` field will be present in all the subsequent stage events after a stage has been retried.

```
{
    "version": "0",
    "id": "38656bcd-a798-5f92-c738-02a71be484e1",
    "detail-type": "CodePipeline Stage Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2023-10-26T14:14:56Z",
    "region": "eu-central-1",
    "resources": [
        "arn:aws:codepipeline:eu-central-1:123456789012:BuildFromTag"
    ],
    "detail": {
        "pipeline": "BuildFromTag",
        "execution-id": "05dafb6a-5a56-4951-a858-968795364846",
        "stage-last-retry-attempt-time": "2023-10-26T14:14:56.305Z",
        "stage": "Build",
        "state": "RESUMED",
        "version": 32.0,
        "pipeline-execution-attempt": 2.0
    }
}
```


Action-level events

Action-level events are emitted when there is a state change for an action execution.

Topics

- Action STARTED event
- Action SUCCEEDED event
- Action FAILED event
- Action ABANDONED event

Action STARTED event

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 deployment action `myAction`.

Emitted event

```
{
    "version": "0",
    "id": 01234567-EXAMPLE,
    "detail-type": "CodePipeline Action Execution State Change",
    "source": "aws.codepipeline",
    "account": 123456789012,
    "time": "2020-01-24T22:03:07Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": 12345678-1234-5678-abcd-12345678abcd,
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Prod",
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "myAction",
        "state": "STARTED",
        "type": {
            "owner": "AWS",
            "category": "Deploy",
            "provider": "CodeDeploy",
            "version": "1"
        },
        "version": 2.0
        "pipeline-execution-attempt": 1.0
        "input-artifacts": [
            {
                "name": "SourceArtifact",
                "s3location": {
                    "bucket": "codepipeline-us-east-1-BUCKETEXAMPLE",
                    "key": "myPipeline/SourceArti/KEYEXAMPLE"
                }
            }
        ]
    }
}
```

Processed event


```
{
    "account": "123456789012",
    "detailType": "CodePipeline Action Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:45:44Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Deploy",
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Deploy",
        "input-artifacts": [
            {
                "name": "SourceArtifact",
                "s3location": {
                    "bucket": "codepipeline-us-east-1-EXAMPLE",
                    "key": "myPipeline/SourceArti/EXAMPLE"
                }
            }
        ],
        "state": "STARTED",
        "region": "us-east-1",
        "type": {
            "owner": "AWS",
            "provider": "CodeDeploy",
            "category": "Deploy",
            "version": "1"
        },
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "additionalAttributes": {}
}
```

Action SUCCEEDED event

When an action execution succeeds, it emits an event that sends notifications with the following content. This example is for the pipeline named "myPipeline" in the us-west-2 Region, for the source action "Source". For this event type, there are two different region fields. The event region field specifies the Region for the pipeline event. The region field under the detail section specifies the Region for the action.

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Action Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:03:11Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Source",
        "execution-result": {
            "external-execution-url": "https://us-west-2.console.aws.amazon.com/
codecommit/home#/repository/my-repo/commit/8cf40f2EXAMPLE",
            "external-execution-summary": "Added LICENSE.txt",
            "external-execution-id": "8cf40fEXAMPLE"
        },
        "output-artifacts": [
            {
                "name": "SourceArtifact",
                "s3location": {
                    "bucket": "codepipeline-us-west-2-BUCKETEXAMPLE",
                    "key": "myPipeline/SourceArti/KEYEXAMPLE"
                }
            }
        ],
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Source",
```

```
        "state": "SUCCEEDED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeCommit",
            "category": "Source",
            "version": "1"
        },
        "version": 3.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Action Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:45:44Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:ACCOUNT:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "arn:aws:codepipeline:us-west-2:123456789012:myPipeline",
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Source",
        "execution-result": {
            "external-execution-url": "https://us-west-2.console.aws.amazon.com/
codecommit/home#/repository/my-repo/commit/8cf40f2EXAMPLE",
            "external-execution-summary": "Edited index.html",
            "external-execution-id": "36ab3ab7EXAMPLE"
        },
        "output-artifacts": [
            {
                "name": "SourceArtifact",
                "s3location": {
                    "bucket": "codepipeline-us-west-2-EXAMPLE",
                    "key": "myPipeline/SourceArti/EXAMPLE"
                }
            }
        ],
```

```
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Source",
        "state": "SUCCEEDED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeCommit",
            "category": "Source",
            "version": "1"
        },
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {}
}
```

Action FAILED event

When an action execution fails, it emits an event that sends notifications with the following content. This example is for the pipeline named "myPipeline" in the us-west-2 Region, for the action "Deploy".

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Action Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:55:43Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:51:09.981Z",
```

```
        "stage": "Deploy",
        "execution-result": {
            "external-execution-url": "https://us-west-2.console.aws.amazon.com/
codedeploy/home?#/deployments/<ID>",
            "external-execution-summary": "Deployment <ID> failed",
            "external-execution-id": "<ID>",
            "error-code": "JobFailed"
        },
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Deploy",
        "state": "FAILED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeDeploy",
            "category": "Deploy",
            "version": "1"
        },
        "version": 4.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Action Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:46:16Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "stage": "Deploy",
        "execution-result": {
            "external-execution-url": "https://console.aws.amazon.com/codedeploy/
home?region=us-west-2#/deployments/<ID>",
            "external-execution-summary": "Deployment <ID> failed",
            "external-execution-id": "<ID>",
            "error-code": "JobFailed"
```

```
        },
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Deploy",
        "state": "FAILED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeDeploy",
            "category": "Deploy",
            "version": "1"
        },
        "version": 13.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {
        "additionalInformation": "Deployment <ID> failed"
    }
}
```

Action ABANDONED event

When an action execution is abandoned, it emits an event that sends notifications with the following content. This example is for the pipeline named `"myPipeline"` in the `us-west-2` Region, for the action `"Deploy"`.

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Action Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:21:39Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
```

```
        "stage": "Deploy",
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Deploy",
        "state": "ABANDONED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeDeploy",
            "category": "Deploy",
            "version": "1"
        },
        "version": 3.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Create a Rule That Sends a Notification on a Pipeline Event

A rule watches for certain events and then routes them to AWS targets that you choose. You can create a rule that performs an AWS action automatically when another AWS action happens, or a rule that performs an AWS action regularly on a set schedule.

Topics

- Send a Notification When Pipeline State Changes (Console)
- Send a Notification When Pipeline State Changes (CLI)

Send a Notification When Pipeline State Changes (Console)

These steps show how to use the EventBridge console to create a rule to send notifications of changes in CodePipeline.

To create an EventBridge rule that targets your pipeline with an Amazon S3 source

1. Open the Amazon EventBridge console at https://eusc-de-east-1.console.amazonaws-eusc.eu/events/.
2. In the navigation pane, choose **Rules**. Leave the default bus selected or choose an event bus. Choose **Create rule**.
3. In **Name**, enter a name for your rule.
4. Under **Rule type**, choose **Rule with an event pattern**. Choose **Next**.

5. Under **Event pattern**, choose **AWS services**.
6. 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**.
7. 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.
8. For event patterns that are more detailed than the selectors allow, you can also use the **Edit pattern** option in the **Event pattern** window to designate an event pattern in JSON format.

   > **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 **Event pattern** window.

   - **Example**

     Use this sample event pattern to capture failed deploy and build actions across all the pipelines.

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

```
  "detail": {
    "state": [
      "FAILED"
    ],
    "type": {
      "category": ["Deploy", "Build"]
    }
  }
}
```

- **Example**

  Use this sample event pattern to capture all rejected or failed approval actions across all the pipelines.

```
{
 "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.

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

```
    "CodePipeline Stage Execution State Change"
  ],
  "detail": {
    "pipeline": ["myPipeline", "my2ndPipeline"]
  }
}
```

9. Choose **Next**.
10. In **Target types**, choose **AWS service**.
11. In **Select a target**, choose **CodePipeline**. In **Pipeline ARN**, enter the pipeline ARN for the pipeline to be started 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

12. To create or specify an IAM service role that grants EventBridge permissions to invoke the target associated with your EventBridge rule (in this case, the target is CodePipeline):
    - Choose **Create a new role for this specific resource** to create a service role that gives EventBridge permissions to start your pipeline executions.
    - Choose **Use existing role** to enter a service role that gives EventBridge permissions to start your pipeline executions.
13. Choose **Next**.
14. On the **Tags** page, choose **Next**.
15. On the **Review and create** page, review the rule configuration. If you're satisfied with the rule, choose **Create rule**.

Send a Notification When Pipeline State Changes (CLI)

These steps show how to use the CLI to create an 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 EventBridge and Event Patterns.

To create an EventBridge 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 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 Amazon SNS topic.

   The following sample command specifies that for the rule called `MyPipelineStateChanges`, 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 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 EventBridge to use the designated target service to invoke the notification. For more information, see Using resource-based policies for Amazon EventBridge.

Events placeholder bucket reference

This section is a reference only. For information about creating a pipeline with event detection resources, see Source actions and change detection methods.

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 custom bucket, but you cannot access the data from the placeholder bucket.

- If you used the CLI or 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.

> **Note**
>
> Data written to CloudTrail placeholder buckets automatically expires after one day and is not retained.

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

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.

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) 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":"AKIAI44QH8DHBEXAMPLE",
   "arn":"arn:aws:iam::80398EXAMPLE:user/JaneDoe-CodePipeline",
   "accountId":"80398EXAMPLE",
   "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
   "userName":"JaneDoe-CodePipeline",
   "sessionContext": {
   "attributes":{
```

```
    "mfaAuthenticated":"false",
    "creationDate":"2015-06-17T14:44:03Z"
    }
   },
"invokedBy":"signin.amazonaws.com"},
"eventTime":"2015-06-17T19:12:20Z",
"eventSource":"codepipeline.amazonaws.com",
"eventName":"UpdatePipeline",
"awsRegion":"us-east-2",
"sourceIPAddress":"192.0.2.64",
"userAgent":"signin.amazonaws.com",
"requestParameters":{
  "pipeline":{
  "version":1,
  "roleArn":"arn:aws:iam::80398EXAMPLE:role/CodePipeline_Service_Role",
  "name":"MyFirstPipeline",
  "stages":[
    {
    "actions":[
     {
         "name":"MySourceStage",
       "actionType":{
       "owner":"AWS",
       "version":"1",
       "category":"Source",
       "provider":"S3"
    },
   "inputArtifacts":[],
   "outputArtifacts":[
    {"name":"MyApp"}
    ],
   "runOrder":1,
   "configuration":{
    "S3Bucket":"amzn-s3-demo-source-bucket",
    "S3ObjectKey":"sampleapp_linux.zip"
     }
     }
    ],
      "name":"Source"
    },
    (...)
              },
"responseElements":{
  "pipeline":{
```

```
      "version":2,
      (...)
            },
      "requestID":"2c4af5c9-7ce8-EXAMPLE",
      "eventID":""c53dbd42-This-Is-An-Example"",
      "eventType":"AwsApiCall",
      "recipientAccountId":"80398EXAMPLE"
      }
       ]
}
```

CodePipeline CloudWatch metrics

You can use metrics in CloudWatch to measure CodePipeline pipelines. You can use the following metrics to measure pipeline duration and failed pipeline executions.

You can monitor your pipelines at two levels:

Pipeline level

These metrics are at the pipeline level. Choose **By Pipeline** in CloudWatch. Available pipelines will be listed in CloudWatch.

AWS account level

These metrics are for all pipelines in an account. To see metrics at the AWS account level, choose **Account Metrics** in CloudWatch.

For more information about viewing metrics, see View available metrics in the *Amazon CloudWatch User Guide*.

Topics

- PipelineDuration
- FailedPipelineExecutions

PipelineDuration

The `PipelineDuration` metric measures the duration of the pipeline execution. This metric is available for the pipeline level only.

Units: Seconds

FailedPipelineExecutions

The `FailedPipelineExecutions` metric measures the number of pipeline executions that failed. This metric is available at the pipeline level and the account level.

This metric includes a separate count for pipeline executions with a retry of a failed action. In other words, when a failed action is retried in a pipeline, this will count as a separate pipeline execution metric. For example, for a pipeline with an S3 source action where the S3 source action fails the first time, that makes one execution, and then the source action is retried and fails again. For this example, the metric will be the following:

```
FailedPipelineExecutionAttempts: 2
```

Units: Count

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"
- Deployment error: A pipeline configured with an AWS Elastic Beanstalk deploy action hangs instead of failing if the "DescribeEvents" permission is missing
- 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."
- Pipeline error: A Jenkins build or test action runs for a long time and then fails due to lack of credentials or permissions
- Pipeline error: A pipeline created in one AWS Region using a bucket created in another AWS Region returns an "InternalError" with the code "JobFailed"
- 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
- Pipeline artifact folder names appear to be truncated
- Add CodeBuild GitClone permissions for connections to Bitbucket, GitHub, GitHub Enterprise Server, or GitLab.com
- Add CodeBuild GitClone permissions for CodeCommit source actions
- 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>"
- GitHub (via OAuth app) source action: Repository list shows different repositories
- GitHub (via GitHub App) source action: Unable to complete the connection for a repository
- Amazon S3 error: CodePipeline service role <ARN> is getting S3 access denied for the S3 bucket <BucketName>
- Pipelines with an Amazon S3, Amazon ECR, or CodeCommit source no longer start automatically
- Connections error when connecting to GitHub: "A problem occurred, make sure cookies are enabled in your browser" or "An organization owner must install the GitHub app"

- Pipelines with execution mode changed to QUEUED or PARALLEL mode fail when run limit reached
- Pipelines in PARALLEL mode have an outdated pipeline definition if edited when changing to QUEUED or SUPERSEDED mode
- Pipelines changed from PARALLEL mode will display a previous execution mode
- Pipelines with connections that use trigger filtering by file paths might not start at branch creation
- Pipelines with connections that use trigger filtering by file paths might not start when file limit is reached
- CodeCommit or S3 source revisions in PARALLEL mode might not match EventBridge event
- EC2 Deploy action fails with an error message No such file
- EKS Deploy action fails with a cluster unreachable error message
- Pipeline does not trigger for all branches when multiple source actions reference the same repository
- Need help with a different issue?

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.

After you apply the edited policy, follow the steps in Start a pipeline manually 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 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 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.

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 policies for AWS CodePipeline.

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: 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:

```
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.

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.

Add CodeBuild GitClone permissions for connections to Bitbucket, GitHub, GitHub Enterprise Server, or GitLab.com

When you use an AWS CodeConnections in a source action and a CodeBuild action, there are two ways the input artifact can be passed to the build:

- Default: The source action produces a zip file that contains the code that CodeBuild downloads.
- Full clone: The source code can be directly downloaded to the build environment.

  The Full clone mode allows you to interact with the source code as a working Git repository. To use this mode, you must grant your CodeBuild environment permissions to use the connection.

To add permissions to your CodeBuild service role policy, you create a customer-managed policy that you attach to your CodeBuild service role. The following steps create a policy where the `UseConnection` permission is specified in the `action` field, the connection ARN is specified in the `Resource` field, and the source repository ID is limited via `Condition`.

To use the console to add the UseConnection permissions

1. To find the connection ARN and the source repository ID for your pipeline, open your pipeline, click the (i) icon on your source action and switch to the **Input** tab.

   An example connection ARN is:

   ```
   arn:aws:codeconnections:eu-central-1:123456789123:connection/
   sample-1908-4932-9ecc-2ddacee15095
   ```

   An example of the source repository ID:

   ```
   owner/test-app
   ```

   You add the connection ARN and the repository ID to your CodeBuild service role policy.
2. To find your CodeBuild service role, choose the build project used in your pipeline and navigate to the **Build details** tab.
3. Choose the **Service role** link. This opens the IAM console where you can add a new policy that grants access to your connection.
4. In the IAM console, choose **Add permissions**, and then choose **Create inline policy**.

   Use the following sample policy template. Add your connection ARN in the `Resource` field and your repository ID in `codeconnections:FullRepositoryId` of the `Condition` field, as shown in this example:

   JSON

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

```
            "Resource": "arn:aws-eusc:codeconnections:eu-
central-1:123456789123:connection/my-connection-id",
            "Condition": {
                "StringEquals": {
                    "codeconnections:FullRepositoryId": "my-repository-id"
                }
            }
        }
    ]
}
```

Use the `Condition` field to scope your policy permissions down further based on your build spec requirements (see `CodeConnection` conditions documentation).

On the **JSON** tab, paste your policy.

5. Choose **Next**. Enter a name for the policy (for example, **`connection-permissions`**), and then choose **Create policy**.

   You will see **`connection-permissions`** policy attached to your role **Permissions policies**.

Add CodeBuild GitClone permissions for CodeCommit source actions

When your pipeline has a CodeCommit source action, there are two ways you can pass the input artifact to the build:

- **Default** – The source action produces a zip file that contains the code that CodeBuild downloads.
- **Full clone** – The source code can be directly downloaded to the build environment.

  The **Full clone** option allows you to interact with the source code as a working Git repository. To use this mode, you must add permissions for your CodeBuild environment to pull from your repository.

To add permissions to your CodeBuild service role policy, you create a customer-managed policy that you attach to your CodeBuild service role. The following steps create a policy that specifies the `codecommit:GitPull` permission in the `action` field.

To use the console to add the GitPull permissions

1. To find your CodeBuild service role, open the build project used in your pipeline and navigate to the **Build details** tab.
2. Choose the **Service role** link. This opens the IAM console where you can add a new policy that grants access to your repository.
3. In the IAM console, choose **Attach policies**, and then choose **Create policy**.
4. On the **JSON** tab, paste the following sample policy.

```
{
    "Action": [
        "codecommit:GitPull"
    ],
    "Resource": "*",
    "Effect": "Allow"
},
```

5. Choose **Review policy**. Enter a name for the policy (for example, **`codecommit-gitpull`**), and then choose **Create policy**.
6. Return to the page where you were attaching permissions, refresh the policy list, and select the policy you just created. Choose **Attach policies**.

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 task definition file is a required artifact for the CodePipeline deploy action to Amazon ECS through CodeDeploy (the `CodeDeployToECS` action). The maximum artifact ZIP size in the `CodeDeployToECS` deploy action is 3 MB. The following error message is returned when the file is not found or the artifact size exceeds 3 MB:

Exception while trying to read the task definition artifact file from: <source artifact name>

Possible fixes: Make sure the task definition file is included as an artifact. If the file already exists, make sure the compressed size is less than 3 MB.

GitHub (via OAuth app) source action: Repository list shows different repositories

Problem:

After a successful authorization for a GitHub (via OAuth app) action in the CodePipeline console, you can choose from a list of your GitHub repositories. If the list does not include the repositories you expected to see, then you can troubleshoot the account used for authorization.

Possible fixes: The list of repositories provided in the CodePipeline console are based on the GitHub organization the authorized account belongs to. Verify that the account you are using to authorize with GitHub is the account associated with the GitHub organization where your repository is created.

GitHub (via GitHub App) source action: Unable to complete the connection for a repository

Problem:

Because a connection to a GitHub repository uses the AWS Connector for GitHub, you need organization owner permissions or admin permissions to the repository to create the connection.

Possible fixes: For information about permission levels for a GitHub repository, see https://docs.github.com/en/free-pro-team@latest/github/setting-up-and-managing-organizations-and-teams/permission-levels-for-an-organization.

Amazon S3 error: CodePipeline service role <ARN> is getting S3 access denied for the S3 bucket <BucketName>

Problem:

While in progress, the CodeCommit action in CodePipeline checks that the pipeline artifact bucket exists. If the action does not have permission to check, an `AccessDenied` error occurs in Amazon S3 and the following error message displays in CodePipeline:

CodePipeline service role "arn:aws:iam::*AccountID*:role/service-role/*RoleID*" is getting S3 access denied for the S3 bucket "*BucketName*"

The CloudTrail logs for the action also log the `AccessDenied` error.

Possible fixes: Do the following:

- For the policy attached to your CodePipeline service role, add `s3:ListBucket` to the list of actions in your policy. For instructions on how to view your service role policy, see View the pipeline ARN and service role ARN (console). Edit the policy statement for your service role as detailed in Add permissions to the CodePipeline service role.
- For the resource-based policy attached to the Amazon S3 artifact bucket for your pipeline, also called the *artifact bucket policy*, add a statement to allow the `s3:ListBucket` permission to be used by your CodePipeline service role.

  To add your policy to the artifact bucket

  1. Follow the steps in View the pipeline ARN and service role ARN (console) to choose your artifact bucket on the pipeline **Settings** page and then view it in the Amazon S3 console.
  2. Choose **Permissions**.
  3. Under **Bucket policy**, choose **Edit**.
  4. In the **Policy** text field, enter a new bucket policy, or edit the existing policy as shown in the following example. The bucket policy is a JSON file, so you must enter valid JSON.

     The following example shows a bucket policy statement for an artifact bucket where the example role ID for the service role is *AROAEXAMPLEID*.

     ```
     {
         "Effect": "Allow",
         "Principal": "*",
         "Action": "s3:ListBucket",
         "Resource": "arn:aws:s3:::BucketName",
         "Condition": {
             "StringLike": {
                 "aws:userid": "AROAEXAMPLEID:*"
             }
         }
     }
     ```

     The following example shows the same bucket policy statement after the permission is added.

JSON

```
{
    "Version":"2012-10-17",
    "Id": "SSEAndSSLPolicy",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:ListBucket",
            "Resource": "arn:aws-eusc:s3:::amzn-s3-demo-bucket",
            "Condition": {
                "StringLike": {
                    "aws:userid": "AROAEXAMPLEID:*"
                }
            }
        },
        {
            "Sid": "DenyUnEncryptedObjectUploads",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws-eusc:s3:::amzn-s3-demo-bucket/*",
            "Condition": {
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "aws:kms"
                }
            }
        },
        {
            "Sid": "DenyInsecureConnections",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws-eusc:s3:::amzn-s3-demo-bucket/*",
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": false
                }
            }
        }
    ]
```

```
}
```

For more information, see the steps in https://aws.amazon.com/blogs/security/writing-iam-policies-how-to-grant-access-to-an-amazon-s3-bucket/.

5. Choose **Save**.

After you apply the edited policy, follow the steps in Start a pipeline manually to manually rerun your pipeline.

Pipelines with an Amazon S3, Amazon ECR, or CodeCommit source no longer start automatically

Problem:

After making a change to configuration settings for an action that uses event rules (EventBridge or CloudWatch Events) for change detection, the console might not detect a change where source trigger identifiers are similar and have identical initial characters. Because the new event rule is not created by the console, the pipeline no longer starts automatically.

An example of a minor change at the end of the parameter name for CodeCommit would be changing your CodeCommit branch name `MyTestBranch-1` to `MyTestBranch-2`. Because the change is at the end of the branch name, the event rule for the source action might not update or create a rule for the new source settings.

This applies to source actions that use CWE events for change detection as follows:

Source action	Parameters / trigger identifiers (console)
Amazon ECR	**Repository name** **Image tag**
Amazon S3	**Bucket** **S3 object key**
CodeCommit	**Repository name**

Source action	Parameters / trigger identifiers (console)
	Branch name

Possible fixes:

Do one of the following:

- Change the CodeCommit/S3/ECR configuration settings so that changes are made to the starting portion of the parameter value.

  Example: Change your branch name `release-branch` to `2nd-release-branch`. Avoid a change at the end of the name, such as `release-branch-2`.
- Change the CodeCommit/S3/ECR configuration settings for each pipeline.

  Example: Change your branch name `myRepo/myBranch` to `myDeployRepo/myDeployBranch`. Avoid a change at the end of the name, such as `myRepo/myBranch2`.
- Instead of the console, use the CLI or CloudFormation to create and update your change-detection event rules. For instructions on creating event rules for an S3 source action, see Connecting to Amazon S3 source actions that use EventBridge and AWS CloudTrail. For instructions on creating event rules for an Amazon ECR action, see Amazon ECR source actions and EventBridge resources. For instructions on creating event rules for a CodeCommit action, see CodeCommit source actions and EventBridge.

After you edit your action configuration in the console, accept the updated change-detection resources created by the console.

Connections error when connecting to GitHub: "A problem occurred, make sure cookies are enabled in your browser" or "An organization owner must install the GitHub app"

Problem:

To create the connection for a GitHub source action in CodePipeline, you must be the GitHub organization owner. For repositories that are not under an organization, you must be the repository

owner. When a connection is created by someone other than the organization owner, a request is created for the organization owner, and one of the following errors display:

A problem occurred, make sure cookies are enabled in your browser

OR

An organization owner must install the GitHub app

Possible fixes: For repositories in a GitHub organization, the organization owner must create the connection to the GitHub repository. For repositories that are not under an organization, you must be the repository owner.

Pipelines with execution mode changed to QUEUED or PARALLEL mode fail when run limit reached

Problem: The maximum number of concurrent executions for a pipeline in QUEUED mode is 50 executions. When this limit is reached, the pipeline fails without a status message.

Possible fixes: When editing the pipeline definition for execution mode, make the edit separately from other edit actions.

For more information about QUEUED or PARALLEL execution mode, see CodePipeline concepts.

Pipelines in PARALLEL mode have an outdated pipeline definition if edited when changing to QUEUED or SUPERSEDED mode

Problem: For pipelines in parallel mode, when editing the pipeline execution mode to QUEUED or SUPERSEDED, the pipeline definition for PARALLEL mode will not be updated. The updated pipeline definition when updating PARALLEL mode is not used in the SUPERSEDED or QUEUED mode.

Possible fixes: For pipelines in parallel mode, when editing the pipeline execution mode to QUEUED or SUPERSEDED, avoid updating the pipeline definition at the same time.

For more information about QUEUED or PARALLEL execution mode, see CodePipeline concepts.

Pipelines changed from PARALLEL mode will display a previous execution mode

Problem: For pipelines in PARALLEL mode, when editing the pipeline execution mode to QUEUED or SUPERSEDED, the pipeline state will not display the updated state as PARALLEL. If the pipeline changed from PARALLEL to QUEUED or SUPERSEDED, the state for the pipeline in SUPERSEDED or QUEUED mode will be the last known state in either of those modes. If the pipeline was never run in that mode before, then the state will be empty.

Possible fixes: For pipelines in parallel mode, when editing the pipeline execution mode to QUEUED or SUPERSEDED, note that the execution mode display will not show the PARALLEL state.

For more information about QUEUED or PARALLEL execution mode, see CodePipeline concepts.

Pipelines with connections that use trigger filtering by file paths might not start at branch creation

Description: For pipelines with source actions that use connections, such as a BitBucket source action, you can set up a trigger with a Git configuration that allows you to filter by file paths to start your pipeline. In certain cases, for pipelines with triggers that are filtered on file paths, the pipeline might not start when a branch with a file path filter is first created, since this does not allow the AWS CodeConnections connection to resolve the files that changed. When the Git configuration for the trigger is set up to filter on file paths the pipeline will not start when the branch with the filter has just been created in the source repository. For more information about filtering on file paths, see Add trigger with code push or pull request event types.

Result: For example, pipelines in CodePipeline that have a file path filter on a branch "B" will not be triggered when branch "B" is created. If there are no file path filters, the pipeline will still start.

Pipelines with connections that use trigger filtering by file paths might not start when file limit is reached

Description: For pipelines with source actions that use connections, such as a BitBucket source action, you can set up a trigger with a Git configuration that allows you to filter by file paths to start your pipeline. CodePipeline retrieves up to the first 100 files; therefore, when the Git

configuration for the trigger is set up to filter on file paths, the pipeline might not start if there are over 100 files. For more information about filtering on file paths, see Add trigger with code push or pull request event types.

Result: For example, if a diff contains 150 files, CodePipeline looks at the first 100 files (in no particular order) to check against the file path filter specified. If the file that matches the file path filter is not among the 100 files retrieved by CodePipeline, the pipeline will not be invoked.

CodeCommit or S3 source revisions in PARALLEL mode might not match EventBridge event

Description: For pipeline executions in PARALLEL mode, an execution might start with the most recent change, such as the CodeCommit repository commit, that might not be the same as the change for the EventBridge event. In some cases, where a split second might be between commits or image tags that start the pipeline, when CodePipeline receives the event and starts that execution, another commit or image tag has been pushed, CodePipeline (for example, the CodeCommit action) will clone the HEAD commit at that moment.

Result: For pipelines in PARALLEL mode with a CodeCommit or S3 source, regardless of the change that triggered the pipeline execution, the source action will always clone the HEAD at the time it is started. For example, for a pipeline in PARALLEL mode, a commit is pushed, which starts the pipeline for execution 1, and the second pipeline execution uses the second commit.

EC2 Deploy action fails with an error message `No such file`

Description: After the EC2 deploy action unzips the artifacts in the target directory on the instances, the action runs the script. If the script is in the target directory but the action is unable to run the script, the action fails on that instance, and the remaining instances fail deployment.

An error similar to the following error messages displays in the logs for a deployment where the target directory is `/home/ec2-user/deploy/` and the source repository path is `myRepo/postScript.sh`.

- ```
  Instance i-0145a2d3f3EXAMPLE is FAILED on event AFTER_DEPLOY, message:
  ----------ERROR-------
  chmod: cannot access '/home/ec2-user/deploy/myRepo/postScript.sh': No such file or
   directory
  ```
  ```

```
/var/lib/<path>/_script.sh: line 2: /home/ec2-user/deploy/myRepo/postScript.sh: No
 such file or directory
failed to run commands: exit status 127
```

- ```
  Executing commands on instances i-0145a2d3f3EXAMPLE, SSM command id <ID>, commands:
   chmod u+x /home/ec2-user/deploy/script.sh
  ----------ERROR-------: No such file or directory
  ```

Result: The deployment action fails in the pipeline.

Possible fixes: To troubleshoot, use the following steps.

1. View the logs to verify which instance caused the script to fail.
2. Change directory (cd) to the target directory on your instance. Test run the script on the instance.
3. In your source repository, edit the script file to remove any comments or commands that might be causing the issue.

EKS Deploy action fails with a `cluster unreachable` error message

Description: After the EKS deploy action runs, the action fails with a `cluster unreachable` error message. The message shows an access issue on the cluster due to missing permissions. Based on the type of files (Helm chart or Kubernetes manifest files), the error message displays as follows.

- For an EKS deploy action that is using a Helm chart, an error similar to the following error message displays.

  ```
  error message:

  helm upgrade --install my-release test-chart --wait
  Error: Kubernetes cluster unreachable: the server has asked for the client to provide
   credentials
  ```

- For an EKS deploy action that is using Kubernetes manifest files, an error similar to the following error message displays.
  ```

```
kubectl apply -f deployment.yaml
Error: error validating "deployment.yaml": error validating data: failed to download
 openapi: the server has asked for the client to provide credentials
```

Result: The deployment action fails in the pipeline.

Possible fixes: If using an existing role, the CodePipeline service role must be updated with the required permissions to use the EKS deploy action. Additionally, to allow the CodePipeline service role access to your cluster, you must add an access entry to your cluster and specify the service role for the access entry.

1. Verify that the CodePipeline service role has the required permissions for the EKS deploy action. For the permissions reference, see Service role policy permissions.
2. Add an access entry to your cluster and specify the CodePipeline service role for the access. For an example, see Step 4: Create an access entry for the CodePipeline service role.

Pipeline does not trigger for all branches when multiple source actions reference the same repository

Problem: A pipeline has multiple source actions that use connections (such as a Bitbucket, GitHub, or GitLab connection), with each source action pointing to a different branch of the same repository. Only one of the branches triggers the pipeline when you push changes. The connection's webhook subscription is registered for the combination of pipeline and repository, not per branch. As a result, multiple source actions targeting different branches of the same repository within a single pipeline are not supported.

Possible fixes: Use a separate pipeline for each branch that you want to trigger independently.

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.

> **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 data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The 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 .
- **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
- Identity and access management for AWS CodePipeline
- Logging and monitoring in CodePipeline
- Compliance validation for AWS CodePipeline
- Resilience in AWS CodePipeline
- Infrastructure security in AWS CodePipeline
- Security best practices

Data protection in AWS CodePipeline

The AWS shared responsibility model applies to data protection in . As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are

responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see Data Privacy FAQ. For information about data protection in Europe, see the General Data Protection Regulation (GDPR) Center.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary 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. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

The following security best practices also address data protection in CodePipeline:

- Configure server-side encryption for artifacts stored in Amazon S3 for CodePipeline
- Use AWS Secrets Manager to track database passwords or third-party API keys

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, and
- 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 AWS KMS keys. Code artifacts are stored in a customer-owned S3 bucket and encrypted with either the AWS managed key or a customer managed key. For more information, see Configure server-side encryption for artifacts stored in Amazon S3 for CodePipeline.

Encryption in transit

All service-to-service communication is encrypted in transit using SSL/TLS.

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 key when you create a pipeline using the Create Pipeline wizard. The AWS managed key is encrypted along with object data and managed by AWS.
- You can create and manage your own customer managed key.

> **Important**
>
> CodePipeline only supports symmetric KMS keys. Do not use an asymmetric KMS key 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 customer managed 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-eusc:s3:::codepipeline-us-west-2-89050EXAMPLE/
*",
            "Condition": {
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "aws:kms"
                }
            }
        },
```

```
        {
            "Sid": "DenyInsecureConnections",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws-eusc: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 KMS keys 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 AWS managed key
- Configure server-side encryption for S3 buckets using CloudFormation or the AWS CLI

View your AWS managed key

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 AWS KMS using the AWS managed key for Amazon S3 (the `aws/s3` key). This AWS managed 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 AWS managed key

1. Sign in to the AWS Management Console and open the AWS KMS console.

2. If a welcome page appears, choose **Get started now**.
3. In the service navigation pane, choose **AWS managed keys**.
4. 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.
5. In the list, 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 CloudFormation or the AWS CLI

When you use 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 key. You can also use your own keys instead of the AWS managed key. Some reasons to choose 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.

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 KMS keys, you can create a customer managed key, and then change your applications or aliases to use the new customer managed key. Or, you can enable automatic key rotation for an existing customer managed key.

To rotate your customer managed key, see Rotating keys.

> **Important**
>
> CodePipeline only supports symmetric KMS keys. Do not use an asymmetric KMS key to encrypt the data in your S3 bucket.

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 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*.
- To view instructions for adding Secrets Manager dynamic references to your CloudFormation templates, see https://aws.amazon.com/blogs/security/how-to-create-and-retrieve-secrets-managed-in-aws-secrets-manager-using-aws-cloudformation-template/.

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
- Authenticating with identities

- Managing access using policies
- How AWS CodePipeline works with IAM
- AWS CodePipeline identity-based policy examples
- AWS CodePipeline resource-based policy examples
- Troubleshooting AWS CodePipeline identity and access
- permissions reference
- Manage the CodePipeline service role

Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshooting AWS CodePipeline identity and access)
- **Service administrator** - determine user access and submit permission requests (see How AWS CodePipeline works with IAM)
- **IAM administrator** - write policies to manage access (see AWS CodePipeline identity-based policy examples)

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you

don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see Choose 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. Examples include IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. You must specify a principal in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. 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 that work with IAM, see AWS services that work with IAM in the *IAM User Guide*.

Topics

- CodePipeline identity-based policies
- CodePipeline resource-based policies
- Authorization based on CodePipeline tags
- CodePipeline IAM roles

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

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions 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:

```
"Action": [
```

```
      "codepipeline:action1",
      "codepipeline:action2"
```

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

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

resources and operations

In , 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. 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, you can find the pipeline ARN under **Settings** in the console. For more information, see View the pipeline ARN and service role ARN (console).

Resource Type	ARN Format
Pipeline	arn:aws:codepipeline:*region*:*account*:*pipeline-name*

Resource Type	ARN Format
Stage	arn:aws:codepipeline:*region*:*account*:*pipeline-name* /*stage-name*
Action	arn:aws:codepipeline:*region*:*account*:*pipeline-name* /*stage-name* /*action-name*
Custom action	arn:aws:codepipeline:*region*:*account*:actionty pe:*owner*/*category*/*provider*/*version*
All resources	arn:aws:codepipeline:*
All resources owned by the specified account in the specified Region	arn:aws:codepipeline:*region*:*account*:*

Note

Most services in AWS treat a colon (:) or a forward slash (/) as the same character in ARNs. However, 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 , 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 permissions reference 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:

```
"Resource": "arn:aws:codepipeline:us-east-2:111222333444:*"
```

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 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"]
```

provides a set of operations to work with the resources. For a list of available operations, see permissions reference.

Condition keys

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys 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.

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,

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.

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.

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*.

To learn how to create a pipeline that uses resources from another account, and for the related example policies, see Create a pipeline in CodePipeline that uses resources from another AWS account.

Topics

- Policy best practices
- Viewing resources in the console
- Allow users to view their own permissions
- Identity-based policies (IAM) examples
- Using tags to control access to CodePipeline resources
- Permissions required to use the console
- Permissions required to view compute logs in the console
- AWS managed policies for AWS CodePipeline
- Customer managed policy examples

Policy best practices

Identity-based policies 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 with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Viewing resources in the console

The 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:

- AWS CodeBuild: `ListProjects`
- CodeCommit: `ListRepositories`
- CodeDeploy: `ListApplications`
- CodePipeline: `ListPipelines`

Results are not returned for a service's resources if you do not have permissions for that service. Even if you have permissions for viewing resources, some resources are not returned if there is an explicit `Deny` to view those resources.

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-eusc: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 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 grants permissions to disable and enable transitions between all stages in the pipeline named `MyFirstPipeline` in the `us-west-2 region`:

JSON

```
{
  "Version":"2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "codepipeline:EnableStageTransition",
        "codepipeline:DisableStageTransition"
      ],
      "Resource" : [
        "arn:aws-eusc: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 console. This policy is based on the `AWSCodePipeline_ReadOnlyAccess` 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 . 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

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "codepipeline:GetPipeline",
        "codepipeline:GetPipelineState",
        "codepipeline:GetPipelineExecution",
        "codepipeline:ListPipelineExecutions",
        "codepipeline:ListActionExecutions",
        "codepipeline:ListActionTypes",
        "codepipeline:ListPipelines",
        "codepipeline:ListTagsForResource",
        "iam:ListRoles",
        "s3:ListAllMyBuckets",
        "codecommit:ListRepositories",
        "codedeploy:ListApplications",
        "lambda:ListFunctions",
        "codestar-notifications:ListNotificationRules",
        "codestar-notifications:ListEventTypes",
        "codestar-notifications:ListTargets"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws-eusc:codepipeline:us-
west-2:111222333444:MyFirstPipeline"
    },
    {
      "Action": [
        "codepipeline:GetPipeline",
        "codepipeline:GetPipelineState",
        "codepipeline:GetPipelineExecution",
        "codepipeline:ListPipelineExecutions",
        "codepipeline:ListActionExecutions",
        "codepipeline:ListActionTypes",
        "codepipeline:ListPipelines",
        "codepipeline:ListTagsForResource",
        "iam:ListRoles",
        "s3:GetBucketPolicy",
        "s3:GetObject",
        "s3:ListBucket",
        "codecommit:ListBranches",
```

```
        "codedeploy:GetApplication",
        "codedeploy:GetDeploymentGroup",
        "codedeploy:ListDeploymentGroups",
        "elasticbeanstalk:DescribeApplications",
        "elasticbeanstalk:DescribeEnvironments",
        "lambda:GetFunctionConfiguration",
        "opsworks:DescribeApps",
        "opsworks:DescribeLayers",
        "opsworks:DescribeStacks"
      ],
      "Effect": "Allow",
      "Resource": "*"
    },
    {
      "Sid": "CodeStarNotificationsReadOnlyAccess",
      "Effect": "Allow",
      "Action": [
        "codestar-notifications:DescribeNotificationRule"
      ],
      "Resource": "*",
      "Condition": {
        "ArnLike": {
          "codestar-notifications:NotificationsForResource": "arn:aws-
eusc:iam::*:role/Service*"
        }
      }
    }
  ]
}
```

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-eusc: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-
eusc: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 your administrative user. For more information about how to create IAM roles and to explore example IAM policy statements for , see Customer managed policy examples.

Using tags to control access to CodePipeline resources

Conditions in IAM 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. 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.

String condition operators let you construct `Condition` elements that restrict access based on comparing a key to a string value. You can add `IfExists` to the end of any condition operator name except the Null condition. You do this to say "If the policy key is present in the context of the request, process the key as specified in the policy. If the key is not present, evaluate the condition element as true." For example, you can use `StringEqualsIfExists` to restrict by condition keys that might not be present on other types of resources.

For the complete syntax and semantics of tag condition keys, see Controlling Access Using Tags. For additional information about condition keys, see the following resources. The CodePipeline policy examples in this section align with the following information about condition keys and expand on it with examples of nuances for CodePipeline such as nesting of resources.

- String condition operators
- AWS services that work with IAM
- SCP syntax
- IAM JSON policy elements: Condition
- aws:RequestTag/tag-key
- Condition keys for CodePipeline

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 `AWSCodePipeline_FullAccess` 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 where specific tags are listed in the request. 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 the following example, the intent of the policy is to deny unauthorized users permission to create a pipeline with the tag values specified. However, creating a pipeline requires accessing resources in addition to the pipeline itself (for example, pipeline actions and stages). Because the `'Resource'` specified in the policy is `'*'`, the policy is evaluated against every resource that has an ARN and is created when the pipeline is being created. These additional resources do not have the tag condition key, so the `StringEquals` check fails, and the user is not granted the ability to create any pipeline. To address this, use the `StringEqualsIfExists` condition operator instead. This way, the test only happens if the condition key exists.

You could read the following as: "If the resource being checked has a tag `"RequestTag/Project"` condition key, then allow the action only if the key value begins with `projectA`. If the resource being checked does not have that condition key, then don't worry about it."

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. A customer's administrator must attach this IAM policy to unauthorized administrative users, in addition to the managed user policy.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codepipeline:CreatePipeline",
        "codepipeline:TagResource"
      ],
      "Resource": "*",
```

```
      "Condition": {
        "StringEqualsIfExists": {
          "aws:RequestTag/Project": ["ProjectA", "ProjectB"]
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
        "codepipeline:UntagResource"
      ],
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["Project"]
        }
      }
    }
  ]
}
```

Example 2: Limit tagging actions based on resource tags

The `AWSCodePipeline_FullAccess` 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.

JSON

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

```
        "codepipeline:TagResource"
      ],
      "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`. In other words, the only tag key which can be specified is `Project`, and its value must be `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 users or roles who don't have the `AWSCodePipeline_FullAccess` 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:CreatePipeline",
        "codepipeline:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/Project": "ProjectA"
```

```
        },
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["Project"]
        }
      }
    }
  ]
}
```

Example 4: Limit untagging actions based on resource tags

The `AWSCodePipeline_FullAccess` 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`.

Also, 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 completely remove the `Project` tag. A customer's administrator must attach this IAM policy to unauthorized users or roles, in addition to the managed user policy.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codepipeline:UntagResource"
      ],
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["Project"]
        }
      }
    }
  ]
```

```
}
```

Permissions required to use the console

To use in the 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
- AWS CodeBuild
- 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 console, also attach the `AWSCodePipeline_ReadOnlyAccess` managed policy to the user, as described in AWS managed policies for AWS CodePipeline.

You don't need to allow minimum console permissions for users who are making calls to the AWS CLI or the API.

Permissions required to view compute logs in the console

To view the logs in the Commands action on the CodePipeline console, the console role must have permissions. To view logs in the console, add the `logs:GetLogEvents` permissions to the console role.

In the console role policy statement, scope down the permissions to the pipeline level as shown in the following example.

```
{
    "Effect": "Allow",
    "Action": [
        "Action": "logs:GetLogEvents"
    ],
    "Resource": "arn:aws:logs:*:YOUR_AWS_ACCOUNT_ID:log-group:/aws/
codepipeline/YOUR_PIPELINE_NAME:*"
}
```

AWS managed policies for AWS CodePipeline

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

Important

The AWS managed policies `AWSCodePipelineFullAccess` and `AWSCodePipelineReadOnlyAccess` have been replaced. Use the `AWSCodePipeline_FullAccess` and `AWSCodePipeline_ReadOnlyAccess` policies.

AWS managed policy: `AWSCodePipeline_FullAccess`

This is a policy that grants full access to CodePipeline. To view the JSON policy document in the IAM console, see AWSCodePipeline_FullAccess.

Permissions details

This policy includes the following permissions.

- `codepipeline` – Grants permissions to CodePipeline.
- `chatbot` – Grants permissions to allow principals to manage resources in Amazon Q Developer in chat applications.
- `cloudformation` – Grants permissions to allow principals to manage resource stacks in CloudFormation.
- `cloudtrail` – Grants permissions to allow principals to manage logging resources in CloudTrail.
- `codebuild` – Grants permissions to allow principals to access build resources in CodeBuild.
- `codecommit` – Grants permissions to allow principals to access source resources in CodeCommit.
- `codedeploy` – Grants permissions to allow principals to access deployment resources in CodeDeploy.
- `codestar-notifications` – Grants permissions to allow principals to access resources in AWS CodeStar Notifications.
- `ec2` – Grants permissions to allow deployments in CodeCatalyst to manage elastic load balancing in Amazon EC2.
- `ecr` – Grants permissions to allow access to resources in Amazon ECR.
- `elasticbeanstalk` – Grants permissions to allow principals to access resources in Elastic Beanstalk.
- `iam` – Grants permissions to allow principals to manage roles and policies in IAM.
- `lambda` – Grants permissions to allow principals to manage resources in Lambda.

- `events` – Grants permissions to allow principals to manage resources in CloudWatch Events.
- `opsworks` – Grants permissions to allow principals to manage resources in AWS OpsWorks.
- `s3` – Grants permissions to allow principals to manage resources in Amazon S3.
- `sns` – Grants permissions to allow principals to manage notification resources in Amazon SNS.
- `states` – Grants permissions to allow principals to view state machines in AWS Step Functions. A state machine consists of a collection of states that manage tasks and transition between states.

For the policy, see AWSCodePipeline_FullAccess.

AWS managed policy: `AWSCodePipeline_ReadOnlyAccess`

This is a policy that grants read-only access to CodePipeline. To view the JSON policy document in the IAM console, see AWSCodePipeline_ReadOnlyAccess.

Permissions details

This policy includes the following permissions.

- `codepipeline` – Grants permissions to actions in CodePipeline.
- `codestar-notifications` – Grants permissions to allow principals to access resources in AWS CodeStar Notifications.
- `s3` – Grants permissions to allow principals to manage resources in Amazon S3.
- `sns` – Grants permissions to allow principals to manage notification resources in Amazon SNS.

For the policy, see AWSCodePipeline_ReadOnlyAccess.

AWS managed policy: `AWSCodePipelineApproverAccess`

This is a policy that grants permission to approve or reject a manual approval action. To view the JSON policy document in the IAM console, see AWSCodePipelineApproverAccess.

Permissions details

This policy includes the following permissions.

- `codepipeline` – Grants permissions to actions in CodePipeline.

For the policy, see AWSCodePipelineApproverAccess.

AWS managed policy: `AWSCodePipelineCustomActionAccess`

This is a policy that grants permission to create custom actions in CodePipeline or integrate Jenkins resources for build or test actions. To view the JSON policy document in the IAM console, see AWSCodePipelineCustomActionAccess.

Permissions details

This policy includes the following permissions.

- `codepipeline` – Grants permissions to actions in CodePipeline.

For the policy, see AWSCodePipelineCustomActionAccess.

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

This managed policy grants permissions for CodePipeline along with the related services CodeCommit, CodeBuild, CodeDeploy, and AWS CodeStar Notifications. The policy also grants permissions that you need for working with other services that integrate with your pipelines,

such as Amazon S3, Elastic Beanstalk, CloudTrail, Amazon EC2, and CloudFormation. 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 Amazon Q Developer in chat applications clients configured for Slack.

The `AWSCodePipeline_FullAccess` managed policy includes the following statements to allow full access to notifications.

```
    {
        "Sid": "CodeStarNotificationsReadWriteAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:CreateNotificationRule",
            "codestar-notifications:DescribeNotificationRule",
            "codestar-notifications:UpdateNotificationRule",
            "codestar-notifications:DeleteNotificationRule",
            "codestar-notifications:Subscribe",
            "codestar-notifications:Unsubscribe"
        ],
        "Resource": "*",
        "Condition" : {
            "StringLike" : {"codestar-notifications:NotificationsForResource" :
 "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline"}
        }
    },
    {
        "Sid": "CodeStarNotificationsListAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:ListNotificationRules",
            "codestar-notifications:ListTargets",
            "codestar-notifications:ListTagsforResource",
            "codestar-notifications:ListEventTypes"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CodeStarNotificationsSNSTopicCreateAccess",
        "Effect": "Allow",
        "Action": [
            "sns:CreateTopic",
            "sns:SetTopicAttributes"
        ],
```

```
        "Resource": "arn:aws:sns:*:*:codestar-notifications*"
    },
    {
        "Sid": "SNSTopicListAccess",
        "Effect": "Allow",
        "Action": [
            "sns:ListTopics"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CodeStarNotificationsChatbotAccess",
        "Effect": "Allow",
        "Action": [
            "chatbot:DescribeSlackChannelConfigurations",
            "chatbot:ListMicrosoftTeamsChannelConfigurations"
          ],
       "Resource": "*"
    }
```

Permissions related to notifications in read-only managed policies

The `AWSCodePipeline_ReadOnlyAccess` 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.

```
   {
        "Sid": "CodeStarNotificationsPowerUserAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:DescribeNotificationRule"
        ],
        "Resource": "*",
        "Condition" : {
            "StringLike" : {"codestar-notifications:NotificationsForResource" :
 "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline"}
        }
    },
    {
        "Sid": "CodeStarNotificationsListAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:ListNotificationRules",
```

```
            "codestar-notifications:ListEventTypes",
            "codestar-notifications:ListTargets"
        ],
        "Resource": "*"
    }
```

For more information about IAM and notifications, see Identity and Access Management for AWS CodeStar Notifications.

AWS CodePipeline updates to AWS managed policies

View details about updates to AWS managed policies for CodePipeline since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the CodePipeline Document history page.

Change	Description	Date
AWSCodePipeline_FullAccess – Updates to existing policy	CodePipeline added a permission to this policy to support `ListStacks` in CloudFormation.	March 15, 2024
AWSCodePipeline_FullAccess – Updates to existing policy	This policy was updated to add permissions for Amazon Q Developer in chat applications. For more information, see CodePipeline managed policies and notifications.	June 21, 2023
AWSCodePipeline_FullAccess and AWSCodePipeline_ReadOnlyAccess managed policies – Updates to existing policy	CodePipeline added a permission to these policies to support an additional notification type using Amazon Q Developer in chat applications, `chatbot:ListMicrosoftTeamsChannelConfigurations` .	May 16, 2023

Change	Description	Date
AWSCodePipelineFullAccess – Deprecated	This policy has been replaced by `AWSCodePipeline_FullAccess` . After November 17, 2022, this policy can not be attached to any new users, groups, or roles. For more information, see AWS managed policies for AWS CodePipeline.	November 17, 2022
AWSCodePipelineReadOnlyAccess – Deprecated	This policy has been replaced by `AWSCodePipeline_ReadOnlyAccess` . After November 17, 2022, this policy can not be attached to any new users, groups, or roles. For more information, see AWS managed policies for AWS CodePipeline.	November 17, 2022
CodePipeline started tracking changes	CodePipeline started tracking changes for its AWS managed policies.	March 12, 2021

Customer managed policy examples

In this section, you can find example user policies that grant permissions for various actions. These policies work when you are using the 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 console.

> **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
- Example 2: Grant permissions to enable and disable transitions between stages
- Example 3: Grant permissions to get a list of all available action types
- Example 4: Grant permissions to approve or reject manual approval actions
- Example 5: Grant permissions to poll for jobs for a custom action
- Example 6: Attach or edit a policy for Jenkins integration with AWS CodePipeline
- Example 7: Configure cross-account access to a pipeline

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"
            ],
            "Resource": "arn:aws-eusc:codepipeline:us-
west-2:111222333444:MyFirstPipeline"
        }
    ]
}
```

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"
            ],
            "Resource": "arn:aws-eusc:codepipeline:us-
west-2:111222333444:MyFirstPipeline/*"
        }
    ]
}
```

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`:

```
"Resource": "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline/Staging"
```

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:

JSON

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

```
            "Action": [
                "codepipeline:ListActionTypes"
            ],
            "Resource": "arn:aws-eusc:codepipeline:us-
west-2:111222333444:actiontype:*"
        }
    ]
}
```

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`:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:PutApprovalResult"
            ],
            "Resource": "arn:aws-eusc:codepipeline:us-
west-2:111222333444:MyFirstPipeline/Staging/*"
        }
    ]
}
```

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.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:PollForJobs"
            ],
            "Resource": [
                "arn:aws-eusc:codepipeline:us-
west-2:111222333444:actionType:Custom/Test/TestProvider/1"
            ]
        }
    ]
}
```

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 identity for that integration and attach an IAM policy that has the minimum permissions required for integration between Jenkins and . This policy is the same as the `AWSCodePipelineCustomActionAccess` managed policy. The following example shows a policy for Jenkins integration:

JSON

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

    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:AcknowledgeJob",
                "codepipeline:GetJobDetails",
                "codepipeline:PollForJobs",
                "codepipeline:PutJobFailureResult",
                "codepipeline:PutJobSuccessResult"
            ],
```

```
            "Resource": "*"
        }
    ]
}
```

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 console. This policy is based on the `AWSCodePipeline_ReadOnlyAccess` 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 . 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

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

    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:GetPipeline",
                "codepipeline:GetPipelineState",
                "codepipeline:ListActionTypes",
                "codepipeline:ListPipelines",
                "iam:ListRoles",
                "s3:GetBucketPolicy",
                "s3:GetObject",
                "s3:ListAllMyBuckets",
                "s3:ListBucket",
                "codedeploy:GetApplication",
```

```
                "codedeploy:GetDeploymentGroup",
                "codedeploy:ListApplications",
                "codedeploy:ListDeploymentGroups",
                "elasticbeanstalk:DescribeApplications",
                "elasticbeanstalk:DescribeEnvironments",
                "lambda:GetFunctionConfiguration",
                "lambda:ListFunctions"
            ],
            "Resource": "arn:aws-eusc:codepipeline:us-
east-2:111122223333:MyFirstPipeline"
        }
    ]
}
```

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 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-
eusc:iam::111122223333:role/CrossAccountPipelineViewers"
        }
    ]
}
```

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:

JSON

```
{
    "Version":"2012-10-17",
    "Id": "SSEAndSSLPolicy",
    "Statement": [
        {
            "Sid": "DenyUnEncryptedObjectUploads",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws-eusc:s3:::amzn-s3-demo-bucket/*",
            "Condition": {
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "aws:kms"
                }
            }
        },
        {
            "Sid": "DenyInsecureConnections",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws-eusc:s3:::amzn-s3-demo-bucket/*",
            "Condition": {
```

```
                "Bool": {
                    "aws:SecureTransport": false
                }
            }
        }
    ]
}
```

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
- I am not authorized to perform iam:PassRole
- I'm an administrator and want to allow others to access CodePipeline
- I want to allow people outside of my AWS account to access my CodePipeline resources

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 `mateojackson` IAM user tries to use the console to view details about a pipeline, but does not have `codepipeline:GetPipeline` permissions.

```
User: arn:aws-eusc:iam::123456789012:user/mateojackson 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'm an administrator and want to allow others to access CodePipeline

To allow others to access CodePipeline, you must grant permission to the people or applications that need access. If you are using AWS IAM Identity Center to manage people and applications, you assign permission sets to users or groups to define their level of access. Permission sets automatically create and assign IAM policies to IAM roles that are associated with the person or application. For more information, see Permission sets in the *AWS IAM Identity Center User Guide*.

If you are not using IAM Identity Center, you must create IAM entities (users or roles) for the people or applications that need access. You must then attach a policy to the entity that grants them the correct permissions in CodePipeline. After the permissions are granted, provide the credentials to the user or application developer. They will use those credentials to access AWS. To learn more about creating IAM users, groups, policies, and permissions, see IAM Identities and Policies and permissions in IAM 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.
- 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 Cross account resource access in IAM in the *IAM User Guide*.

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 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.

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 confirm 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

`arn:aws:codepipeline:`*`region`*`:`*`account`*`:actiontype:`*`owner`*`/`*`category`*`/`*`provider`*`/`*`versio`

CreatePipeline

Action: `codepipeline:CreatePipeline`

Required to create a pipeline.

Resources:

Pipeline

`arn:aws:codepipeline:`*`region`*`:`*`account`*`:`*`pipeline-name`

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*`/`*versio*

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`*`/`*`versio`*

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

`arn:aws:codepipeline:`*`region`*`:`*`account`*`:actiontype:`*`owner`*`/`*`category`*`/`*`provider`*`/`*`versio`*

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

`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

`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

`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`

UntagResource

Action: `codepipeline:UntagResource`

Required to untag the 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`

UpdatePipeline

Action: `codepipeline:UpdatePipeline`

Required to update a specified pipeline with edits or changes to its structure.

Resources:

Pipeline

`arn:aws:codepipeline:region:account:pipeline-name`

Manage the CodePipeline service role

The 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 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` in `eu-west-2`, the console names the role and policy `AWSCodePipelineServiceRole-eu-west-2-MyFirstPipeline`.

CodePipeline service role policy

The CodePipeline service role policy statement contains the minimum permissions for managing pipelines. You can edit the service role statement to remove or add access to resources you do not use. See the appropriate action reference for the minimum required permissions CodePipeline uses for each action.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "AllowS3BucketAccess",
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketVersioning",
        "s3:GetBucketAcl",
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws-eusc:s3:::[[pipeArtifactBucketNames]]"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "{{accountId}}"
        }
      }
    },
    {
      "Sid": "AllowS3ObjectAccess",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl",
        "s3:GetObject",
        "s3:GetObjectVersion",
```

```
        "s3:PutObjectTagging",
        "s3:GetObjectTagging",
        "s3:GetObjectVersionTagging"
      ],
      "Resource": [
        "arn:aws-eusc:s3:::[[pipeArtifactBucketNames]]/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "{{accountId}}"
        }
      }
    }
  ]
}
```

Note

In the policy, the following permissions are required when the S3 objects in your source bucket have tags in them:

```
s3:PutObjectTagging
s3:GetObjectTagging
s3:GetObjectVersionTagging
```

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 for an AWS service.

The following table shows when support was added for other AWS services.

AWS service	CodePipeline support date
CodePipeline invoke action support added. See Service role policy permissions for the CodePipeline invoke action.	March 14, 2025
EC2 action support added. See Service role policy permissions for the EC2 deploy action.	February 21, 2025
EKS action support added. See Service role policy permissions.	February 20, 2025
Amazon Elastic Container Registry `ECRBuildAndPublish` action support added. See Service role permissions: `ECRBuildAndPublish` action.	November 22, 2024
Amazon Inspector `InspectorScan` action support added. See Service role permissions: `InspectorScan` action.	November 22, 2024
Commands action support added. See Service role permissions: Commands action.	October 03, 2024

AWS service	CodePipeline support date
CloudFormation action support added. See Service role permissions: CloudFormationStackSet action and Service role permissions: CloudFormationStackInstances action.	December 30, 2020
CodeCommit full clone output artifact format action support added. See Service role permissions: CodeCommit action.	November 11, 2020
AWS CodeBuild batch builds action support added. See Service role permissions: CodeCommit action.	July 30, 2020
AWS AppConfig action support added. See Service role permissions: AppConfig action.	June 22, 2020
AWS Step Functions action support added. See Service role permissions: StepFunctions action.	May 27, 2020
AWS CodeStar Connections action support added. See Service role permissions: AWS CodeConnections action.	December 18, 2019
S3 deploy action support added. See Service role permissions: S3 deploy action.	January 16, 2019
The CodeDeployToECS action action support added. See Service role permissions: CodeDeployToECS action.	November 27, 2018
Amazon ECR action support added. See Service role permissions: Amazon ECR action.	November 27, 2018

AWS service	CodePipeline support date
Service Catalog action support added. See Service role permissions: Service Catalog action.	October 16, 2018
AWS Device Farm action support added. See Service role permissions: AWS Device Farm action.	July 19, 2018
Amazon ECS action support added. See Service role permissions: Amazon ECS standard action.	December 12, 2017 / Update for opt in for tagging authorization on July 21, 2017
CodeCommit action support added. See Service role permissions: CodeCommit action.	April 18, 2016
AWS OpsWorks action support added. See Service role permissions: AWS OpsWorks action.	June 2, 2016
CloudFormation action support added. See Service role permissions: CloudFormation action.	November 3, 2016
AWS CodeBuild action support added. See Service role permissions: CodeBuild action.	December 1, 2016
Elastic Beanstalk action support added. See Service role permissions: `ElasticBeanstalk` deploy action.	Initial service launch
CodeDeploy action support added. See Service role permissions: AWS CodeDeploy action.	Initial service launch
S3 source action support added. See Service role permissions: S3 source action.	Initial service launch

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://eusc-de-east-1.console.amazonaws-eusc.eu/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**.
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 more information about creating a policy that grants least-privilege access, see https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege.

5. Choose **Review 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.
- 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 Monitoring CodePipeline events.

Compliance validation for AWS CodePipeline

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. For more information about your compliance responsibility when using AWS services, see AWS Security Documentation.

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 AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access CodePipeline through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

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 or action configuration that needs to include secrets, such as tokens or passwords, do not enter secrets directly in the action configuration, or default values of variables defined at pipeline level or CloudFormation configuration, because the information will display in logs. Use Secrets Manager to set up and store secrets, and then use the referenced secret in the pipeline and action configuration, as described in Use AWS Secrets Manager to track database passwords or third-party API keys.
- 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 keys, as described in Configure server-side encryption for artifacts stored in Amazon S3 for CodePipeline.
- 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.

CodePipeline pipeline structure reference

You can use CodePipeline to structure a CI/CD pipeline of automated steps that accomplish tasks that build, test, and deploy your application source code. This reference section provides details about the JSON structure and parameters in your pipeline. For a high-level list of concepts that describe how pipelines are used, see CodePipeline concepts.

- When you create a pipeline, you choose an available source action and provider, such as an S3 bucket, CodeCommit repository, Bitbucket repository, or GitHub repository that contains your source code and starts your pipeline when you commit a source code change. This reference section provides reference information about the available sources for your pipeline. For more information about how to work with source actions, see Start a pipeline in CodePipeline.
- You can choose the test, build, and deploy actions and providers that you want to automatically include when your pipeline runs. This reference section provides reference information about the available actions and how they fit in your pipeline JSON.
- Your finished pipeline will consist of a source stage along with additional stages where you configure actions to deploy and test your application. For a conceptual example of a DevOps pipeline that deploys your application, see DevOps pipeline example.

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, which apply to all pipelines.

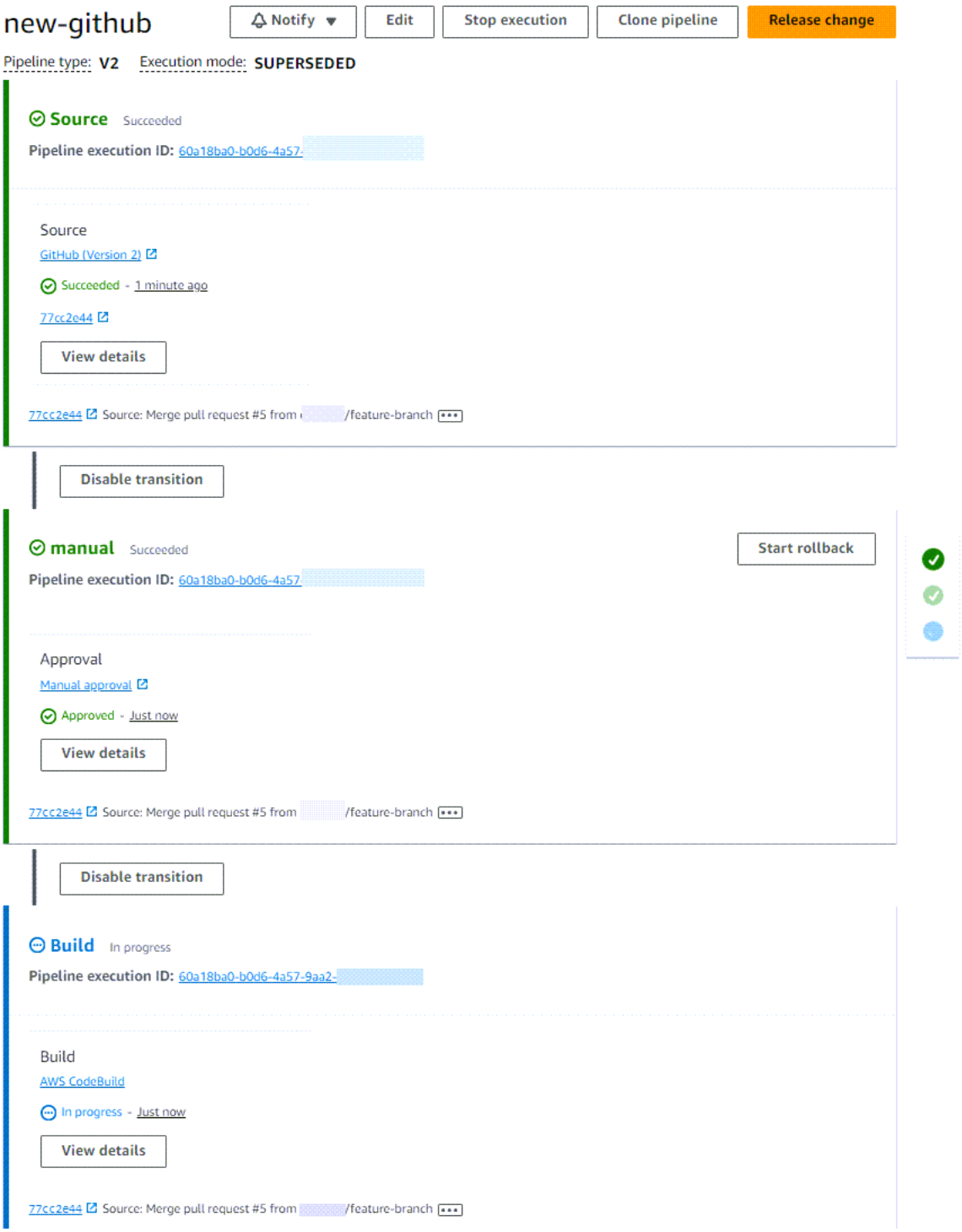
The following sections provide high level parameters and their position in the pipeline structure. Pipeline structure requirements are detailed in each section for the following pipeline component types:

- Field reference for the Pipeline declaration
- Field reference for the Stage declaration
- Field reference for the Action declaration
- List of Valid action providers in CodePipeline by action type
- Reference for Valid settings for the `PollForSourceChanges` parameter

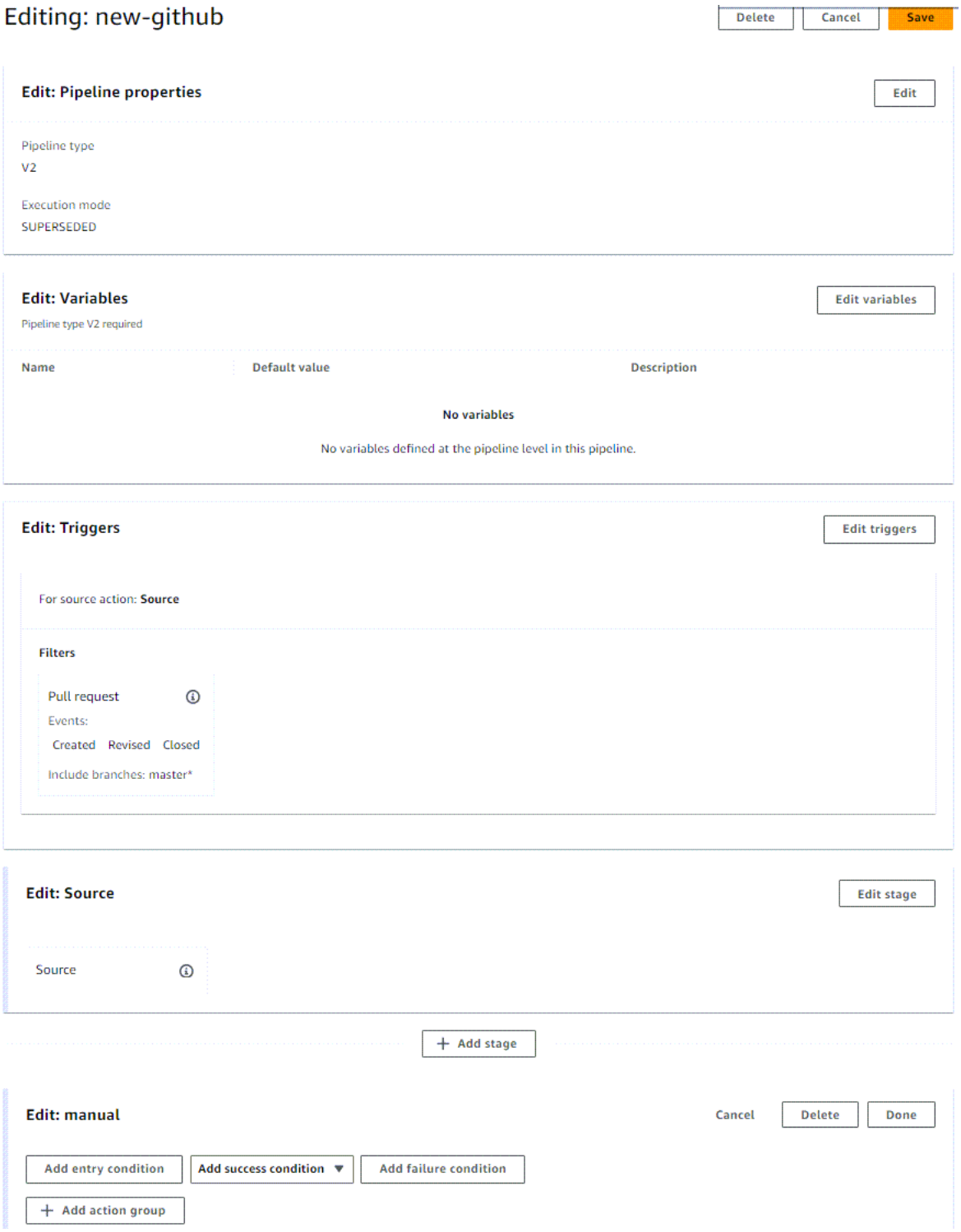
- Reference for Valid input and output artifacts for each action type
- List of links for Valid configuration parameters for each provider type

For more information, see the PipelineDeclaration object in the *CodePipeline API Guide*.

The following example pipeline console view shows the pipeline named new-github, stages named `Source`, `manual`, and `Build`, and actions from GitHub (via GitHub App), manual approval, and CodeBuild action providers.



The pipeline editing mode, when viewed in the console diagram, allows you to edit source overrides, triggers, and actions as shown in the following example.



Topics

- Pipeline declaration
- Stage declaration
- Action declaration

- Valid action providers in CodePipeline
- Valid settings for the PollForSourceChanges parameter
- Valid input and output artifacts for each action type
- Valid configuration parameters for each provider type

Pipeline declaration

The pipeline and metadata level of a pipeline has a basic structure that includes the following parameters and syntax. The pipeline parameter represents the structure of actions and stages to be performed in the pipeline.

For more information, see the PipelineDeclaration object in the *CodePipeline API Guide*.

The following example shows the pipeline and metadata level of the pipeline structure in both JSON and YAML for a V2 type pipeline.

YAML

```
pipeline:
  name: MyPipeline
  roleArn: >-
    arn:aws:iam::ACCOUNT_ID:role/service-role/AWSCodePipelineServiceRole-us-west-2-
MyPipeline
  artifactStore:
    type: S3
    location: amzn-s3-demo-bucket
  stages:
    ...
  version: 6
  executionMode: SUPERSEDED
  pipelineType: V2
  variables:
  - name: MyVariable
    defaultValue: '1'
  triggers:
  - providerType: CodeStarSourceConnection
    gitConfiguration:
      sourceActionName: Source
      push:
      - branches:
```

```
          includes:
          - main
          excludes:
          - feature-branch
      pullRequest:
      - events:
        - CLOSED
        branches:
          includes:
          - main*
metadata:
  pipelineArn: 'arn:aws:codepipeline:us-west-2:ACCOUNT_ID:MyPipeline'
  created: '2019-12-12T06:49:02.733000+00:00'
  updated: '2020-09-10T06:34:07.447000+00:00'
  pollingDisabledAt: '2020-09-10T06:34:07.447000+00:00'
```

JSON

```
{
    "pipeline": {
        "name": "MyPipeline",
        "roleArn": "arn:aws:iam::ACCOUNT_ID:role/service-role/
AWSCodePipelineServiceRole-us-west-2-MyPipeline",
        "artifactStore": {
            "type": "S3",
            "location": "amzn-s3-demo-bucket"
        },
        "stages": {
            ...
    },
        "version": 6,
        "executionMode": "SUPERSEDED",
               "pipelineType": "V2",
        "variables": [
            {
                "name": "MyVariable",
                "defaultValue": "1"
            }
        ],
        "triggers": [
            {
                "providerType": "CodeStarSourceConnection",
                "gitConfiguration": {
```

```
                    "sourceActionName": "Source",
                    "push": [
                        {
                            "branches": {
                                "includes": [
                                    "main"
                                ],
                                "excludes": [
                                    "feature-branch"
                                ]
                            }
                        }
                    ],
                    "pullRequest": [
                        {
                            "events": [
                                "CLOSED"
                            ],
                            "branches": {
                                "includes": [
                                    "main*"
                                ]
                            }
                        }
                    ]
                }
            }
        ]
    },
    "metadata": {
        "pipelineArn": "arn:aws:codepipeline:us-west-2:ACCOUNT_ID:MyPipeline",
        "created": "2019-12-12T06:49:02.733000+00:00",
        "updated": "2020-09-10T06:34:07.447000+00:00",
        "pollingDisabledAt": "2020-09-10T06:34:07.447000+00:00"
    }
}
```

name

The name of the pipeline. 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.

`roleArn`

The IAM ARN for the CodePipeline service role, such as arn:aws:iam::80398EXAMPLE:role/CodePipeline_Service_Role.

To use the console to view the pipeline service role ARN instead of the JSON structure, choose your pipeline in the console, and then choose **Settings**. Under the **General** tab, the **Service role ARN** field displays.

`artifactStore` OR `artifactStores`

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.

> **Note**
>
> In the pipeline structure, you must include either `artifactStore` or `artifactStores` in your pipeline, but you cannot use both. If you create a cross-region action in your pipeline, you must use `artifactStores`.

The following example shows the basic structure for a pipeline with cross-Region actions that uses the `artifactStores` parameter:

```
    "pipeline": {
```

```
        "name": "YourPipelineName",
        "roleArn": "CodePipeline_Service_Role",
        "artifactStores": {
            "us-east-1": {
                "type": "S3",
                "location": "S3 artifact bucket name, such as amzn-s3-demo-bucket"
            },
            "us-west-2": {
                "type": "S3",
                "location": "S3 artifact bucket name, such as amzn-s3-demo-bucket"
            }
        },
        "stages": [
            {

...
```

type

The location type for the artifact bucket, specified as Amazon S3.

location

The name of the Amazon S3 bucket automatically generated for you the first time you create a pipeline using the console, such as codepipeline-us-east-2-1234567890, or any Amazon S3 bucket you provision for this purpose

stages

This parameter contains the name of each stage in the pipeline. For more information about the parameters and syntax at the stage level of the pipeline structure, see the StageDeclaration object in the *CodePipeline API Guide*.

The pipeline structure for stages 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.

> **Important**
>
> Pipelines that are inactive for longer than 30 days will have polling disabled for the pipeline. For more information, see pollingDisabledAt in the pipeline structure reference. For the steps to migrate your pipeline from polling to event-based change detection, see Change Detection Methods.

`version`

The version number of a pipeline is automatically generated and updated every time you update the pipeline.

`executionMode`

You can set the pipeline execution mode so that you can specify the pipeline behavior for consecutive runs, such as queueing, superseding, or running in parallel mode. For more information, see Set or change the pipeline execution mode.

> **Important**
>
> For pipelines in PARALLEL mode, stage rollback is not available. Similarly, failure conditions with a rollback result type cannot be added to a PARALLEL mode pipeline.

`pipelineType`

The pipeline type specifies the available structure and features in the pipeline, such as for a V2 type pipeline. For more information, see Pipeline types.

variables

Variables at the pipeline level are defined when the pipeline is created and resolved at pipeline run time. For more information, see Variables reference. For a tutorial with a pipeline-level variable that is passed at the time of the pipeline execution, see Tutorial: Use pipeline-level variables.

triggers

Triggers allow you to configure your pipeline to start on a particular event type or filtered event type, such as when a change on a particular branch or pull request is detected. Triggers are configurable for source actions with connections that use the `CodeStarSourceConnection` action in CodePipeline, such as GitHub, Bitbucket, and GitLab. For more information about source actions that use connections, see Add third-party source providers to pipelines using AWS CodeConnections.

For more information and more detailed examples, see Automate starting pipelines using triggers and filtering.

For filtering, regular expression patterns in glob format are supported as detailed in Working with glob patterns in syntax.

Note

The CodeCommit and S3 source actions require either a configured change detection resource (an EventBridge rule) or use the option to poll the repository for source changes. For pipelines with a Bitbucket, GitHub, or GitHub Enterprise Server source action, you do not have to set up a webhook or default to polling. The connections action manages change detection for you.

Important

Pipelines that are inactive for longer than 30 days will have polling disabled for the pipeline. For more information, see pollingDisabledAt in the pipeline structure reference. For the steps to migrate your pipeline from polling to event-based change detection, see Change Detection Methods.

`gitConfiguration` fields

The Git configuration for the trigger, including the event types and any parameters for filtering by branches, file paths, tags, or pull request events.

The fields in the JSON structure are defined as follows:

- `sourceActionName`: The name of the pipeline source action with the Git configuration.
- `push`: Push events with filtering. These events use an OR operation between different push filters and an AND operation inside filters.
  - `branches`: The branches to filter on. Branches use an AND operation between includes and excludes.
    - `includes`: Patterns to filter on for branches that will be included. Includes use an OR operation.
    - `excludes`: Patterns to filter on for branches that will be excluded. Excludes use an OR operation.
  - `filePaths`: The file path names to filter on.
    - `includes`: Patterns to filter on for file paths that will be included. Includes use an OR operation.
    - `excludes`: Patterns to filter on for file paths that will be excluded. Excludes use an OR operation.
  - `tags`: The tag names to filter on.
    - `includes`: Patterns to filter on for tags that will be included. Includes use an OR operation.
    - `excludes`: Patterns to filter on for tags that will be excluded. Excludes use an OR operation.
- `pullRequest`: Pull request events with filtering on pull request events and pull request filters.
  - `events`: Filters on open, updated, or closed pull request events as specified.
  - `branches`: The branches to filter on. Branches use an AND operation between includes and excludes.
    - `includes`: Patterns to filter on for branches that will be included. Includes use an OR operation.
    - `excludes`: Patterns to filter on for branches that will be excluded. Excludes use an OR operation.
  - `filePaths`: The file path names to filter on.

- `includes`: Patterns to filter on for file paths that will be included. Includes use an OR operation.
- `excludes`: Patterns to filter on for file paths that will be excluded. Excludes use an OR operation.

The following is an example of the trigger configuration for push and pull request event types.

```
"triggers": [
            {
                "provider": "Connection",
                "gitConfiguration": {
                    "sourceActionName": "ApplicationSource",
                    "push": [
                        {
                            "filePaths": {
                                "includes": [
                                    "projectA/**",
                                    "common/**/*.js"
                                ],
                                "excludes": [
                                    "**/README.md",
                                    "**/LICENSE",
                                    "**/CONTRIBUTING.md"
                                ]
                            },
                            "branches": {
                                "includes": [
                                    "feature/**",
                                    "release/**"
                                ],
                                "excludes": [
                                    "mainline"
                                ]
                            },
                            "tags": {
                                "includes": [
                                    "release-v0", "release-v1"
                                ],
                                "excludes": [
                                    "release-v2"
                                ]
                            }
```

```
                    }
                ],
                "pullRequest": [
                    {
                        "events": [
                            "CLOSED"
                        ],
                        "branches": {
                            "includes": [
                                "feature/**",
                                "release/**"
                            ],
                            "excludes": [
                                "mainline"
                            ]
                        },
                        "filePaths": {
                            "includes": [
                                "projectA/**",
                                "common/**/*.js"
                            ],
                            "excludes": [
                                "**/README.md",
                                "**/LICENSE",
                                "**/CONTRIBUTING.md"
                            ]
                        }
                    }
                ]
            }
        }
    ],
```

Event type push fields for include and exclude

Include and exclude behavior for levels of Git configuration fields for **push** event types are shown in the following list:

```
push (OR operation is used between push and pullRequest or multiples)
    filePaths (AND operation is used between filePaths, branches, and tags)
        includes (AND operation is used between includes and excludes)
            **/FILE.md, **/FILE2 (OR operation is used between file path names)
        excludes (AND operation is used between includes and excludes)
```

```
            **/FILE.md, **/FILE2 (OR operation is used between file path names)
    branches (AND operation is used between filePaths, branches, and tags)
        includes (AND operation is used between includes and excludes)
            BRANCH/**", "BRANCH2/** (OR operation is used between branch names)
        excludes (AND operation is used between includes and excludes)
            BRANCH/**", "BRANCH2/** (OR operation is used between branch names)
    tags (AND operation is used between filePaths, branches, and tags)
         includes (AND operation is used between includes and excludes)
            TAG/**", "TAG2/** (OR operation is used between tag names)
         excludes (AND operation is used between includes and excludes)
            TAG/**", "TAG2/** (OR operation is used between tag names)
```

Event type `pull request` fields for include and exclude

Include and exclude behavior for levels of Git configuration fields for **pull request** event types are shown in the following list:

```
pullRequest (OR operation is used between push and pullRequest or multiples)
    events (AND operation is used between events, filePaths, and branches). Includes/
excludes are N/A for pull request events.
    filePaths (AND operation is used between events, filePaths, and branches)
        includes (AND operation is used between includes and excludes)
            **/FILE.md, **/FILE2 (OR operation is used between file path names)
        excludes (AND operation is used between includes and excludes)
            **/FILE.md, **/FILE2 (OR operation is used between file path names)
    branches (AND operation is used between events, filePaths, and branches)
        includes (AND operation is used between includes and excludes)
            BRANCH/**", "BRANCH2/** (OR operation is used between branch names)
        excludes (AND operation is used between includes and excludes)
            BRANCH/**", "BRANCH2/** (OR operation is used between branch names)
```

`metadata`

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.

`pipelineArn`

The Amazon Resource Name (ARN) of the pipeline.

To use the console to view the pipeline ARN instead of the JSON structure, choose your pipeline in the console, and then choose **Settings**. Under the **General** tab, the **Pipeline ARN** field displays.

`created`

The date and time when the pipeline was created.

`updated`

The date and time when the pipeline was last updated.

`pollingDisabledAt`

The date and time when, for a pipeline that is configured for polling for change detection, when polling was disabled.

Pipelines that are inactive for longer than 30 days will have polling disabled for the pipeline.

- Inactive pipelines will have polling disabled after 30 days of no executions.
- Pipelines using EventBridge, CodeStar Connections, or webhooks will not be affected.
- Active pipelines will not be affected.

For more information, see the `pollingDisabledAt` parameter under PipelineMetadata object in the *CodePipeline API Guide*. For the steps to migrate your pipeline from polling to event-based change detection, see Change Detection Methods.

Stage declaration

The stage level of a pipeline has a basic structure that includes the following parameters and syntax. For more information, see the StageDeclaration object in the *CodePipeline API Guide*.

The following example shows the stage level of the pipeline structure in both JSON and YAML. The example shows two stages named `Source` and `Build`. The example contains two conditions, one for `onSuccess` and one for `beforeEntry`.

YAML

```
pipeline:
  name: MyPipeline
  roleArn: >-
    arn:aws:iam::ACCOUNT_ID:role/service-role/AWSCodePipelineServiceRole-us-west-2-
MyPipeline
```

```
  artifactStore:
    type: S3
    location: amzn-s3-demo-bucket
  stages:
    - name: Source
      actions:
        - name: Source
          ...
    - name: Build
      actions:
        - name: Build
          ...
      onSuccess:
        conditions:
        - result: ROLLBACK
          rules:
          - name: DeploymentWindowRule
         ...
      beforeEntry:
        conditions:
        - result: FAIL
          rules:
          - name: MyLambdaRule
         ...
  version: 6
metadata:
  pipelineArn: 'arn:aws:codepipeline:us-west-2:ACCOUNT_ID:MyPipeline'
  created: '2019-12-12T06:49:02.733000+00:00'
  updated: '2020-09-10T06:34:07.447000+00:00'
```

JSON

```
{
    "pipeline": {
        "name": "MyPipeline",
        "roleArn": "arn:aws:iam::ACCOUNT_ID:role/service-role/
AWSCodePipelineServiceRole-us-west-2-MyPipeline",
        "artifactStore": {
            "type": "S3",
            "location": "amzn-s3-demo-bucket"
        },
        "stages": [
            {
```

```
                "name": "Source",
                "actions": [
                    {
                        "name": "Source",
                        ...
                    }
                ]
            },
            {
                "name": "Build",
                "actions": [
                    {
                        "name": "Build",
                        ...
                    }
                ],
                "onSuccess": {
                    "conditions": [
                        {
                            "result": "ROLLBACK",
                            "rules": [
                                {
                                    "name": "DeploymentWindowRule",
                                    ...
                                }
                            ]
                        }
                    ]
                },
                "beforeEntry": {
                    "conditions": [
                        {
                            "result": "FAIL",
                            "rules": [
                                {
                                    "name": "MyLambdaRule",
                                     ...
                                }
                            ]
                        }
                    ]
                }
            }
        ],
```

```
            }
        ],
        "version": 6
    },
    "metadata": {
        "pipelineArn": "arn:aws:codepipeline:us-west-2:ACCOUNT_ID:MyPipeline",
        "created": "2019-12-12T06:49:02.733000+00:00",
        "updated": "2020-09-10T06:34:07.447000+00:00"
    }
}
```

name

The name of the stage.

actions

The action level of a pipeline has a basic structure that includes the following parameters and syntax. To view parameters and examples, see Action declaration.

conditions

Conditions contain one or more rules that are available in a list of rules in CodePipeline. If all rules in a condition succeed, then the condition is met. You can configure conditions so that when the criteria are not met, the specified result engages.

You can configure the following types of conditions:

- `beforeEntry`
- `onFailure`
- `onSuccess`

For more information and examples, see Configure conditions for a stage.

`rules`

Each condition has a rule set which is an ordered set of rules that are evaluated together. Therefore, if one rule fails in the condition, then the condition fails. You can override rule conditions at pipeline runtime.

The available rules are provided in the Rule reference. For more information, see the Rule structure reference at Rule structure reference.

Action declaration

The action level of a pipeline has a basic structure that includes the following parameters and syntax. For more information, see the ActionDeclaration object in the *CodePipeline API Guide*.

The following example shows the action level of the pipeline structure in both JSON and YAML.

YAML

```
. . .

  stages:
    - name: Source
      actions:
        - name: Source
          actionTypeId:
            category: Source
            owner: AWS
            provider: S3
            version: '1'
          runOrder: 1
          configuration:
            PollForSourceChanges: 'false'
            S3Bucket: amzn-s3-demo-bucket
            S3ObjectKey: codedeploy_linux.zip
          outputArtifacts:
            - name: SourceArtifact
          inputArtifacts: []
          region: us-west-2
          namespace: SourceVariables
    - name: Build
      actions:
```

```
        - name: Build
          actionTypeId:
            category: Build
            owner: AWS
            provider: CodeBuild
            version: '1'
          runOrder: 1
          configuration:
            EnvironmentVariables: >-
              [{"name":"ETag","value":"#{SourceVariables.ETag}","type":"PLAINTEXT"}]
            ProjectName: my-project
          outputArtifacts:
            - name: BuildArtifact
          inputArtifacts:
            - name: SourceArtifact
          region: us-west-2
          namespace: BuildVariables
          runOrder: 1
          configuration:
            CustomData: >-
              Here are the exported variables from the build action: S3 ETAG:
              #{BuildVariables.ETag}
          outputArtifacts: []
          inputArtifacts: []
          region: us-west-2
```

JSON

```
. . .

        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "name": "Source",
                        "actionTypeId": {
                            "category": "Source",
                            "owner": "AWS",
                            "provider": "S3",
                            "version": "1"
                        },
```

```
                        "runOrder": 1,
                        "configuration": {
                            "PollForSourceChanges": "false",
                            "S3Bucket": "amzn-s3-demo-bucket",
                            "S3ObjectKey": "aws-codepipeline-s3-aws-
codedeploy_linux.zip"
                        },
                        "outputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "inputArtifacts": [],
                        "region": "us-west-2",
                        "namespace": "SourceVariables"
                    }
                ]
            },
            {
                "name": "Build",
                "actions": [
                    {
                        "name": "Build",
                        "actionTypeId": {
                            "category": "Build",
                            "owner": "AWS",
                            "provider": "CodeBuild",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "EnvironmentVariables": "[{\"name\":\"ETag\",\"value\":
\"#{SourceVariables.ETag}\",\"type\":\"PLAINTEXT\"}]",
                            "ProjectName": "my-build-project"
                        },
                        "outputArtifacts": [
                            {
                                "name": "BuildArtifact"
                            }
                        ],
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
```

```
                    ],
                    "region": "us-west-2",
                    "namespace": "BuildVariables"
                }
            ]

. . .
```

For a list of example `configuration` details appropriate to the provider type, see Valid configuration parameters for each provider type.

The action structure has the following requirements:

- All action names within a stage must be unique.
- A source action is required for each pipeline.
- Source actions that do not use a connection can be configured for change detection or to turn off change detection. See Change Detection Methods.
- 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.
- 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

Variable	Example of console input	Output
datetime	js-application/{datetime}.zip	UTC timestamp in this format: *<YYYY>-<MM>-DD>_<HH>-<MM>-<SS>* Example: js-application/2019-01-10_07-39-57.zip

Variable	Example of console input	Output
uuid	js-application/{uuid}.zip	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): *<8-digits>-<4-digits>-4-digits>-<4-digits>-<12-digits>* Example: js-application/54a60075-b96a-4bf3-9013-db3a9EXAMPLE.zip

`name`

The name of the action.

`region`

For actions where the provider is an AWS service, the AWS Region of the resource.

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

`roleArn`

The ARN of the IAM service role that performs the declared action. This is assumed through the roleArn that is specified at the pipeline level.

namespace

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 reference.

> **Note**
>
> For Amazon ECR, Amazon S3, or CodeCommit sources, you can also create a source override using input transform entry to use the `revisionValue` in EventBridge for your pipeline event, where the `revisionValue` is derived from the source event variable for your object key, commit, or image ID. For more information, see the optional step for input transform entry included in the procedures under Amazon ECR source actions and EventBridge resources, Connecting to Amazon S3 source actions with a source enabled for events, or CodeCommit source actions and EventBridge.

actionTypeId

The action type ID is identified as a combination of the following four fields.

category

The type of action, or step, in the pipeline, such as a source action. Each action type has a specific set of valid action providers. For a list of valid providers by action type, see the Action structure reference.

These are the valid `actionTypeId` categories (action types) for CodePipeline:

- `Source`
- `Build`
- `Approval`
- `Deploy`
- `Test`
- `Invoke`
- `Compute`

owner

For all currently supported action types, the only valid owner string is AWS, ThirdParty, or Custom. For the valid owner string for a specific action, see the Action structure reference.

For more information, see the CodePipeline API Reference.

version

The version of the action.

provider

The action provider, such as CodeBuild.

- Valid provider types for an action category depend on the category. For example, for a source action category, a valid provider type is S3, CodeStarSourceConnection, CodeCommit, or Amazon ECR. This example shows the structure for a source action with an S3 provider:

```
"actionTypeId": {
  "category": "Source",
  "owner": "AWS",
  "version": "1",
  "provider": "S3"},
```

InputArtifacts

This field contains the input artifact structure, if supported for the action category. 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:

```
"outputArtifacts": [
    {
    "MyApp"
    }
],
```

and there are no other output artifacts, then the input artifact of a following action must be:

```
"inputArtifacts": [
```

```
    {
    "MyApp"
    }
],
```

For example, a source action cannot have input artifacts because it is the first action in the pipeline. However, a source action will always have output artifacts that are processed by the following action. The output artifacts for a source action are the application files from the source repository, zipped and provided through the artifact bucket, that are processed by the following action, such as a CodeBuild action that acts on the application files with build commands.

As an example of actions that cannot have output artifacts, deploy actions do not have output artifacts because these actions are generally the last action in a pipeline.

`name`

The artifact name for the action's input artifacts.

`outputArtifacts`

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"`.

This field contains the output artifact structure, if supported for the action category. The output 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:

```
"outputArtifacts": [
    {
    "MyApp"
    }
],
```

and there are no other output artifacts, then the input artifact of a following action must be:

```
"inputArtifacts": [
    {
```

```
    "MyApp"
    }
],
```

For example, a source action cannot have input artifacts because it is the first action in the pipeline. However, a source action will always have output artifacts that are processed by the following action. The output artifacts for a source action are the application files from the source repository, zipped and provided through the artifact bucket, that are processed by the following action, such as a CodeBuild action that acts on the application files with build commands.

As an example of actions that cannot have output artifacts, deploy actions do not have output artifacts because these actions are generally the last action in a pipeline.

`name`

The artifact name for the action's output artifacts.

`configuration` (by action provider)

The action configuration contains details and parameters appropriate to the provider type. In the section below, the example action configuration parameters are specific to the S3 source action.

The action configuration and input/output artifact limits can vary by action provider. For a list of action configuration examples by action provider, see Action structure reference and the table in Valid configuration parameters for each provider type. The table provides a link to the action reference for each provider type, which lists the configuration parameters for each action in detail. For a table with the input and output artifact limits for each action provider, see Valid input and output artifacts for each action type.

The following considerations apply to working with actions:

- Source actions do not have input artifacts, and deploy actions do not have output artifacts.
- For source action providers that do not use a connection, such as S3, you must use the `PollForSourceChanges` parameter to specify whether you want your pipeline to start automatically when a change is detected. See Valid settings for the `PollForSourceChanges` parameter.
- To configure automated change detection to start your pipeline, or to disable change detection, see Source actions and change detection methods.

- To configure triggers with filtering, use the source action for connections, and then see Automate starting pipelines using triggers and filtering.
- For the output variables for each action, see Variables reference.

> **Note**
>
> For Amazon ECR, Amazon S3, or CodeCommit sources, you can also create a source override using input transform entry to use the `revisionValue` in EventBridge for your pipeline event, where the `revisionValue` is derived from the source event variable for your object key, commit, or image ID. For more information, see the optional step for input transform entry included in the procedures under Amazon ECR source actions and EventBridge resources, Connecting to Amazon S3 source actions with a source enabled for events, or CodeCommit source actions and EventBridge.

> **Important**
>
> Pipelines that are inactive for longer than 30 days will have polling disabled for the pipeline. For more information, see pollingDisabledAt in the pipeline structure reference. For the steps to migrate your pipeline from polling to event-based change detection, see Change Detection Methods.

> **Note**
>
> The CodeCommit and S3 source actions require either a configured change detection resource (an EventBridge rule) or use the option to poll the repository for source changes. For pipelines with a Bitbucket, GitHub, or GitHub Enterprise Server source action, you do not have to set up a webhook or default to polling. The connections action manages change detection for you.

`runOrder`

A positive integer that indicates the run order of the action within the stage. Parallel actions in the stage are shown as having the same integer. For example, two actions with a run order of two will run in parallel after the first action in the stage runs.

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 does 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.

Valid action 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.

Each action category has a valid list of action providers. To reference the valid action providers for each action category, see the Action structure reference.

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 the Action structure reference or Valid input and output artifacts for each action type.

This table lists valid providers by action type.

> **Note**
>
> For Bitbucket, GitHub, or GitHub Enterprise Server actions, refer to the CodeStarSourceConnection for Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed actions action reference topic.

Valid action providers by action type

Action category	Valid action providers	Pipeline type supported	Action reference
Source	Amazon S3	V1, V2	Amazon S3 source action reference
	Amazon ECR	V1, V2	Amazon ECR source action reference
	CodeCommit	V1, V2	CodeCommit source action reference
	CodeStarSourceConnection (for Bitbucket, GitHub, GitHub Enterprise Server actions)	V1, V2	CodeStarSourceConnection for Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed actions
Build	Amazon ECR `ECRBuildAndPublish` action	V2 only	`ECRBuildAndPublish` build action reference
	CodeBuild	V1, V2	AWS CodeBuild build and test action reference

Action category	Valid action providers	Pipeline type supported	Action reference
	Commands action (see Compute)	V2 only	
	Custom CloudBees	V1, V2	Valid input and output artifacts for each action type
	Custom Jenkins	V1, V2	Valid input and output artifacts for each action type
	Custom TeamCity	V1, V2	Valid input and output artifacts for each action type
Test	CodeBuild	V1, V2	AWS CodeBuild build and test action reference
	AWS Device Farm	V1, V2	Valid input and output artifacts for each action type
	Custom BlazeMeter	V1, V2	Valid input and output artifacts for each action type
	ThirdParty GhostInspector		Valid input and output artifacts for each action type
	Custom Jenkins		Valid input and output artifacts for each action type

Action category	Valid action providers	Pipeline type supported	Action reference
	ThirdParty Micro Focus StormRunner Load		Valid input and output artifacts for each action type
	ThirdParty Nouvola		Valid input and output artifacts for each action type
	ThirdParty Runscope		Valid input and output artifacts for each action type
Deploy	Amazon S3		Amazon S3 deploy action reference
	CloudFormation		CloudFormation deploy action reference
	CodeDeploy		Valid input and output artifacts for each action type
	EC2 Deploy action	V2 only	Amazon EC2 action reference
	Amazon ECS		Valid input and output artifacts for each action type
	Amazon ECS (Blue/Green) (this is the `CodeDeployToECS` action)		Valid input and output artifacts for each action type
	Amazon EKS action	V2 only	Amazon Elastic Kubernetes Service EKS deploy action reference

Action category	Valid action providers	Pipeline type supported	Action reference
	Elastic Beanstalk		Valid input and output artifacts for each action type
	AWS AppConfig		AWS AppConfig deploy action reference
	OpsWorks		Valid input and output artifacts for each action type
	Service Catalog		Valid input and output artifacts for each action type
	Amazon Alexa		Valid input and output artifacts for each action type
	Custom XebiaLabs		Valid input and output artifacts for each action type
Approval	Manual		Valid input and output artifacts for each action type
Invoke	CodePipeline Invoke action		AWS CodePipeline invoke action reference
	AWS Lambda		AWS Lambda invoke action reference
	AWS Step Functions		AWS Step Functions invoke action reference

Action category	Valid action providers	Pipeline type supported	Action reference
	InspectorScan		Amazon Inspector `InspectorScan` invoke action reference
Compute	Commands action		Commands action reference

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.

Valid 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 CloudFormation template.
- Create change detection resources (CloudWatch Events rule, 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.

- **`PollForSourceChanges` parameter defaults**

Source	Creation method	Example "configuration" JSON structure output
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": "main", "PollForSourceChanges": "false", "RepositoryName": "my-repo"`
	Pipeline is created with the CLI or CloudFormation, and the `PollForSourceChanges` parameter is not displayed in JSON output, but it sets to `true`.[2]	`BranchName": "main", "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 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": " main" "OAuthToken": " ****"`

Source	Creation method	Example "configuration" JSON structure output
	Pipeline is created with the CLI or CloudFormation, and the `PollForSourceChanges` parameter is not displayed in JSON output, but it sets to `true`.[2]	`"Owner": "MyGitHubAccountName ", "Repo": "MyGitHubRepositoryName ", "Branch": " main", "OAuthToken": " ****"`
	[2] If `PollForSourceChanges` has been added at any point to the JSON structure or the 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.	

Valid input and output artifacts for each action type

Depending on the action type and provider, you can have the following number of input and output artifacts.

Action type constraints for artifacts

Owner	Type of action	Provider	Valid number of input artifacts	Valid number of output artifacts
AWS	Source	S3	0	1
AWS	Source	CodeCommit	0	1
AWS	Source	ECR	0	1
ThirdParty	Source	CodeStarSourceConnection	0	1

Owner	Type of action	Provider	Valid number of input artifacts	Valid number of output artifacts
AWS	Build	CodeBuild	1 to 5	0 to 5
AWS	Test	CodeBuild	1 to 5	0 to 5
AWS	Test	DeviceFarm	1	0
AWS	Approval	ThirdParty	0	0
AWS	Deploy	S3	1	0
AWS	Deploy	CloudFormation	0 to 10	0 to 1
AWS	Deploy	CodeDeploy	1	0
AWS	Deploy	ElasticBeanstalk	1	0
AWS	Deploy	OpsWorks	1	0
AWS	Deploy	ECS	1	0
AWS	Deploy	ServiceCatalog	1	0
AWS	Invoke	Lambda	0 to 5	0 to 5
ThirdParty	Deploy	AlexaSkillsKit	1 to 2	0
Custom	Build	Jenkins	0 to 5	0 to 5
Custom	Test	Jenkins	0 to 5	0 to 5
Custom	Any supported category	As specified in the custom action	0 to 5	0 to 5

Valid configuration parameters for each provider type

This section lists valid `configuration` parameters for each action 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

Name of provider	Provider name in action type	Configuration properties	Required/ Optional
Amazon S3 (Deploy action provider)	For more information, including examples related to Amazon S3 Deploy action parameters, see Amazon S3 deploy action reference.		
Amazon S3 (Source action provider)	For more information, including examples related to Amazon S3 source action parameters, see Amazon S3 source action reference.		
Amazon ECR	For more information, including examples related to Amazon ECR parameters, see Amazon ECR source action reference.		
CodeCommit	For more information, including examples related to CodeCommit parameters, see CodeCommit source action reference.		
CodeStarSourceConnection action for Bitbucket, GitHub (via GitHub app), GHES, and GitLab	For more information, including examples of the action configuration, see Configuration parameters.		

Name of provider	Provider name in action type	Configuration properties	Required/ Optional
GitHub (via OAuth app)	For more information, including examples related to GitHub parameters, see GitHub (via OAuth app) source action reference. This is the Version 1 GitHub action.		
AWS CloudFormation	For more information, including examples related to AWS CloudFormation parameters, see CloudFormation deploy action reference.		
CodeBuild	For more description and examples related to CodeBuild parameters, see AWS CodeBuild build and test action reference.		
CodeDeploy	For more description and examples related to CodeDeploy parameters, see AWS CodeDeploy deploy action reference.		
AWS Device Farm	For more description and examples related to AWS Device Farm parameters, see AWS Device Farm test action reference.		
AWS Elastic Beanstalk	`ElasticBeanstalk`	`ApplicationName`	Required
		`EnvironmentName`	Required
AWS Lambda	For more information, including examples related to AWS Lambda parameters, see AWS Lambda invoke action reference.		
AWS OpsWorks Stacks	`OpsWorks`	`Stack`	Required
		`Layer`	Optional
		App	Required
Amazon ECS	For more description and examples related to Amazon ECS parameters, see Amazon Elastic Container Service deploy action reference.		

Name of provider	Provider name in action type	Configuration properties	Required/ Optional
Amazon ECS and CodeDeploy(Blue/Green)	For more description and examples related to Amazon ECS and CodeDeploy blue/green parameters, see Amazon Elastic Container Service and CodeDeploy blue-green deploy action reference.		
Service Catalog	`ServiceCatalog`	`TemplateFilePath`	Required
		`ProductVersionName`	Required
		`ProductType`	Required
		`ProductVersionDescription`	Optional
		`ProductId`	Required
Alexa Skills Kit	`AlexaSkillsKit`	`ClientId`	Required
		`ClientSecret`	Required
		`RefreshToken`	Required
		`SkillId`	Required
Jenkins	The name of the action you provided in the CodePipeline Plugin for Jenkins (for example, *`MyJenkins ProviderName`*)	`ProjectName`	Required
Manual Approval	`Manual`	`CustomData`	Optional
		`ExternalEntityLink`	Optional
		`NotificationArn`	Optional

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 manual approval:

```
"configuration": {
  "CustomData": "Comments on the manual approval",
  "ExternalEntityLink": "http://my-url.com",
  "NotificationArn": "arn:aws:sns:us-west-2:12345EXAMPLE:Notification"
}
```

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.

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

- Amazon EC2 action reference
- Amazon ECR source action reference
- ECRBuildAndPublish build action reference
- Amazon Elastic Container Service and CodeDeploy blue-green deploy action reference
- Amazon Elastic Container Service deploy action reference
- Amazon Elastic Kubernetes Service EKS deploy action reference
- AWS Lambda deploy action reference
- Amazon S3 deploy action reference
- Amazon S3 source action reference
- AWS AppConfig deploy action reference
- CloudFormation deploy action reference
- CloudFormation StackSets deploy action reference
- AWS CodeBuild build and test action reference
- AWS CodePipeline invoke action reference
- CodeCommit source action reference

- AWS CodeDeploy deploy action reference
- CodeStarSourceConnection for Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed actions
- Commands action reference
- AWS Device Farm test action reference
- Elastic Beanstalk deploy action reference
- Amazon Inspector InspectorScan invoke action reference
- AWS Lambda invoke action reference
- AWS OpsWorks deploy action reference
- AWS Service Catalog deploy action reference
- AWS Step Functions invoke action reference

Amazon EC2 action reference

You use an Amazon EC2 EC2 action to deploy application code to your deployment fleet. Your deployment fleet can consist of Amazon EC2 Linux instances or Linux SSM-managed nodes. Your instances must have the SSM agent installed.

> **Note**
>
> This action supports Linux instance types only. The maximum fleet size supported is 500 instances.

The action will choose a number of instances based on a specified maximum. The failed instances from previous instances will be chosen first. The action will skip the deployment on certain instances if the instance has already received deployment of the same input artifact, such as a case where the action failed previously.

> **Note**
>
> This action is only supported for V2 type pipelines.

Topics

- Action type
- Configuration parameters
- Input artifacts
- Output artifacts
- Service role policy permissions for the EC2 deploy action
- Deploy spec file reference
- Action declaration
- Action declaration with Deploy spec example
- See also

Action type

- Category: `Deploy`
- Owner: `AWS`
- Provider: `EC2`
- Version: 1

Configuration parameters

InstanceTagKey

Required: Yes

The tag key of the instances that you created in Amazon EC2, such as `Name`.

InstanceTagValue

Required: No

The tag value of the instances that you created in Amazon EC2, such as `my-instances`.

When this value is not specified, all instances with **InstanceTagKey** will be matched.

InstanceType

Required: Yes

The type of instances or SSM nodes created in Amazon EC2. The valid values are `EC2` and `SSM_MANAGED_NODE`.

You must have already created, tagged, and installed the SSM agent on all instances.

> **Note**
>
> When you create the instance, you create or use an existing EC2 instance role. To avoid `Access Denied` errors, you must add S3 bucket permissions to the instance role to give the instance permissions to the CodePipeline artifact bucket. Create a default role or update your existing role with the `s3:GetObject` permission scoped down to the artifact bucket for your pipeline's Region.

TargetDirectory

Required: Yes (If script is specified)

The directory to be used on your Amazon EC2 instance to run scripts.

DeploySpec

Required: Yes (If deploy spec is specified)

The file to be used to configure deployment install and lifecycle events. For deploy spec field descriptions and information, see Deploy spec file reference. To view an action configuration with the deploy spec file specified, see the example in Action declaration with Deploy spec example.

MaxBatch

Required: No

The maximum number of instances allowed to deploy in parallel.

MaxError

Required: No

The maximum number of instance errors allowed during deployment.

TargetGroupNameList

Required: No

The list of target group names for deployment. You must have already created the target groups.

Target groups provide a set of instances to process specific requests. If the target group is specified, instances will be removed from the target group before deployment and added back to the target group after deployment.

PreScript

Required: No

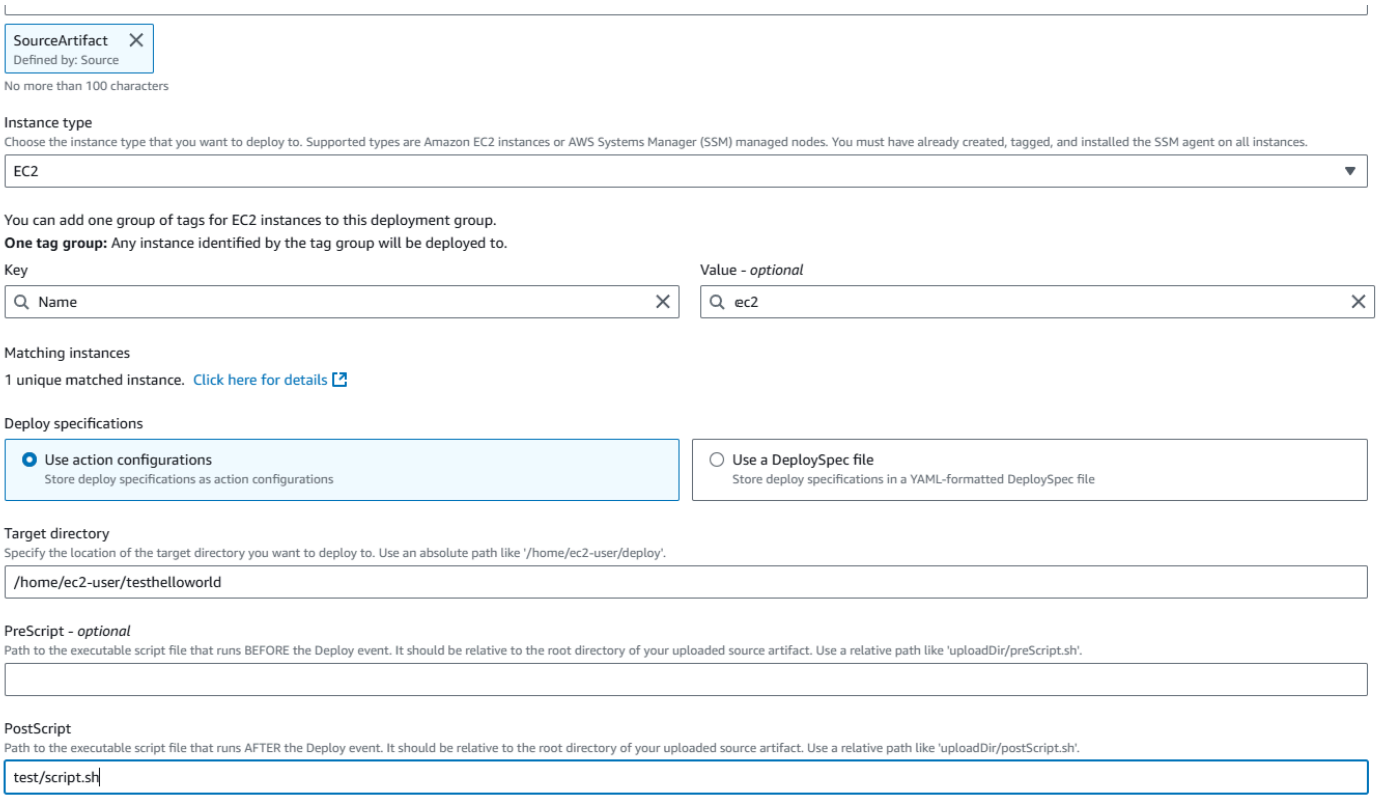
The script to be run before the action Deploy phase.

PostScript

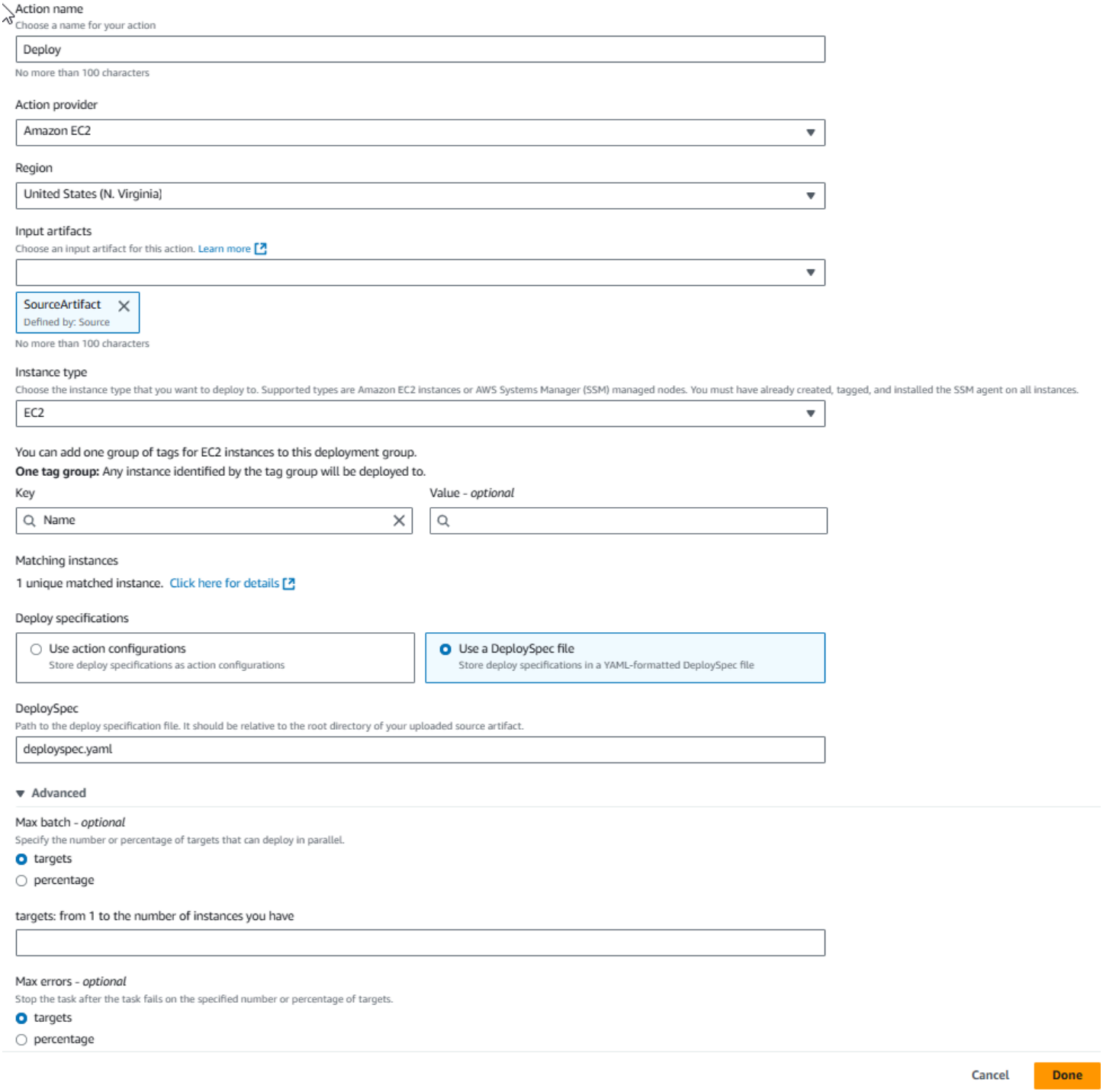
Required: Yes

The script to be run after the action Deploy phase.

The following image shows an example of the **Edit** page for the action where **Use action configurations** is chosen.



The following image shows an example of the **Edit** page for the action where **Use a DeploySpec file** is chosen.



Input artifacts

- **Number of artifacts:** 1

- **Description:** The files provided, if any, to support the script actions during the deployment.

Output artifacts

- **Number of artifacts:** `0`
- **Description:** Output artifacts do not apply for this action type.

Service role policy permissions for the EC2 deploy action

When CodePipeline runs the action, CodePipeline service role requires the following permissions, appropriately scoped down for access with least privilege.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "StatementWithAllResource",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances",
                "elasticloadbalancing:DescribeTargetGroupAttributes",
                "elasticloadbalancing:DescribeTargetGroups",
                "elasticloadbalancing:DescribeTargetHealth",
                "ssm:CancelCommand",
                "ssm:DescribeInstanceInformation",
                "ssm:ListCommandInvocations"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "StatementForLogs",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
```

```
            ],
            "Resource": [
                "arn:aws-eusc:logs:eusc-de-east-1:111122223333:log-group:/aws/
codepipeline/{{pipelineName}}:*"
            ]
        },
        {
            "Sid": "StatementForElasticloadbalancing",
            "Effect": "Allow",
            "Action": [
                "elasticloadbalancing:DeregisterTargets",
                "elasticloadbalancing:RegisterTargets"
            ],
            "Resource": [
                "arn:aws-eusc:elasticloadbalancing:eusc-de-
east-1:111122223333:targetgroup/[[targetGroupName]]/*"
            ]
        },
        {
            "Sid": "StatementForSsmOnTaggedInstances",
            "Effect": "Allow",
            "Action": [
                "ssm:SendCommand"
            ],
            "Resource": [
                "arn:aws-eusc:ec2:eusc-de-east-1:111122223333:instance/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/{{tagKey}}": "{{tagValue}}"
                }
            }
        },
        {
            "Sid": "StatementForSsmApprovedDocuments",
            "Effect": "Allow",
            "Action": [
                "ssm:SendCommand"
            ],
            "Resource": [
                "arn:aws-eusc:ssm:eusc-de-east-1::document/AWS-
RunPowerShellScript",
                "arn:aws-eusc:ssm:eusc-de-east-1::document/AWS-RunShellScript"
            ]
```

```
        }
    ]
}
```

Log groups for your pipeline in CloudWatch logs

When CodePipeline runs the action, CodePipeline creates a log group using the name of the pipeline as follows. This enables you to scope down permissions to log resources using the pipeline name.

```
/aws/codepipeline/MyPipelineName
```

The following permissions for logging are included in the above updates for the service role.

- logs:CreateLogGroup
- logs:CreateLogStream
- logs:PutLogEvents

To view logs in the console using the action details dialog page, the permission to view logs must be added to the console role. For more information, see the console permissions policy example in Permissions required to view compute logs in the console.

Service role policy permissions for CloudWatch logs

When CodePipeline runs the action, CodePipeline creates a log group using the name of the pipeline as follows. This enables you to scope down permissions to log resources using the pipeline name.

```
/aws/codepipeline/MyPipelineName
```

To view logs in the console using the action details dialog page, the permission to view logs must be added to the console role. For more information, see the console permissions policy example in Permissions required to view compute logs in the console.

Deploy spec file reference

When CodePipeline runs the action, you can specify a spec file to configure deployment to your instances. The deploy spec file specifies what to install and which lifecycle event hooks to run

in response to deployment lifecycle events. The deploy spec file is always YAML-formatted. The deploy spec file is used to:

- Map the source files in your application revision to their destinations on the instance.
- Specify custom permissions for deployed files.
- Specify scripts to be run on each instance at various stages of the deployment process.

The deploy spec file supports specific deployment configuration parameters supported by CodeDeploy with the AppSpec file. You can use your existing AppSpec file directly, and any unsupported parameters will be ignored. For more information about the AppSpec file in CodeDeploy, see the Application Specification file reference in the *CodeDeploy User Guide*.

The file deployment parameters are specified as follows.

- `files` - The deploy spec file designates the `source:` and `destination:` for the deployment files.
- `scripts` - The scripted events for the deployment. Two events are supported: `BeforeDeploy` and `AfterDeploy`.
- `hooks` - The lifecycle hooks for the event. The following hooks are supported: `ApplicationStop`, `BeforeInstall`, `AfterInstall`, `ApplicationStart`, and `ValidateService`.

> **Note**
>
> The hooks parameter is available for AppSpec compatibility with CodeDeploy and is only available in version 0.0 (AppSpec format). For this format, CodePipeline will perform a best effort mapping of the events.

Correct YAML spacing must be used in the spec file; otherwise, an error is raised if the locations and number of spaces in a deploy spec file are not correct. For more information about spacing, see the YAML specification.

An example deploy spec file is below.

```
version: 0.1
files:
  - source: /index.html
```

```
    destination: /var/www/html/
scripts:
  BeforeDeploy:
    - location: scripts/install_dependencies
      timeout: 300
      runas: myuser
  AfterDeploy:
    - location: scripts/start_server
      timeout: 300
      runas: myuser
```

To view an action configuration with the deploy spec file specified, see the example in Action declaration with Deploy spec example.

Action declaration

YAML

```
name: DeployEC2
actions:
- name: EC2
  actionTypeId:
    category: Deploy
    owner: AWS
    provider: EC2
    version: '1'
  runOrder: 1
  configuration:
    InstanceTagKey: Name
    InstanceTagValue: my-instances
    InstanceType: EC2
    PostScript: "test/script.sh",
    TargetDirectory: "/home/ec2-user/deploy"
  outputArtifacts: []
  inputArtifacts:
  - name: SourceArtifact
  region: us-east-1
```

JSON

```
{
    "name": "DeployEC2",
```

```
    "actions": [
        {
            "name": "EC2Deploy",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "provider": "EC2",
                "version": "1"
            },
            "runOrder": 1,
            "configuration": {
                "InstanceTagKey": "Name",
                "InstanceTagValue": "my-instances",
                "InstanceType": "EC2",
                "PostScript": "test/script.sh",
                "TargetDirectory": "/home/ec2-user/deploy"
            },
            "outputArtifacts": [],
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-east-1"
        }
    ]
},
```

Action declaration with Deploy spec example

YAML

```
name: DeployEC2
actions:
- name: EC2
  actionTypeId:
    category: Deploy
    owner: AWS
    provider: EC2
    version: '1'
  runOrder: 1
  configuration:
```

```
    DeploySpec: "deployspec.yaml"
    InstanceTagKey: Name
    InstanceTagValue: my-instances
    InstanceType: EC2
  outputArtifacts: []
  inputArtifacts:
  - name: SourceArtifact
  region: us-east-1
```

JSON

```
{
    "name": "DeployEC2",
    "actions": [
        {
            "name": "EC2Deploy",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "provider": "EC2",
                "version": "1"
            },
            "runOrder": 1,
            "configuration": {
                "DeploySpec": "deployspec.yaml",
                "InstanceTagKey": "Name",
                "InstanceTagValue": "my-instances",
                "InstanceType": "EC2"
            },
            "outputArtifacts": [],
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-east-1"
        }
    ]
},
```

See also

The following related resources can help you as you work with this action.

- Tutorial: Deploy to Amazon EC2 instances with CodePipeline – This tutorial walks you through the creation of EC2 instances where you will deploy a script file, along with creation of the pipeline using the EC2 action.
- EC2 Deploy action fails with an error message `No such file` – This topic describes troubleshooting for file not found errors with the EC2 action.

Amazon ECR source action reference

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.

When you use the console to create or edit your pipeline, CodePipeline creates an EventBridge rule that starts your pipeline when a change occurs in the repository.

> **Note**
>
> For Amazon ECR, Amazon S3, or CodeCommit sources, you can also create a source override using input transform entry to use the `revisionValue` in EventBridge for your pipeline event, where the `revisionValue` is derived from the source event variable for your object key, commit, or image ID. For more information, see the optional step for input transform entry included in the procedures under Amazon ECR source actions and EventBridge resources, Connecting to Amazon S3 source actions with a source enabled for events, or CodeCommit source actions and EventBridge.

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

- Configuration parameters
- Input artifacts
- Output artifacts
- Output variables
- Service role permissions: Amazon ECR action
- Action declaration (Amazon ECR example)
- See also

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.

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 reference.

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.

> **Note**
>
> The `ImageTag` output variable is not output when the source revision is overridden.

ImageDigest

The `sha256` digest of the image manifest.

ImageURI

The URI for the image.

Service role permissions: Amazon ECR action

For Amazon ECR support, add the following to your policy statement:

```
{
    "Effect": "Allow",
    "Action": [
        "ecr:DescribeImages"
    ],
    "Resource": "resource_ARN"
},
```

For more information about this action, see Amazon ECR source action reference.

Action declaration (Amazon ECR example)

YAML

```
Name: Source
Actions:
  - InputArtifacts: []
    ActionTypeId:
      Version: '1'
      Owner: AWS
      Category: Source
      Provider: ECR
    OutputArtifacts:
      - Name: SourceArtifact
    RunOrder: 1
    Configuration:
      ImageTag: latest
      RepositoryName: my-image-repo

    Name: ImageSource
```

JSON

```
{
```

```
    "Name": "Source",
    "Actions": [
        {
            "InputArtifacts": [],
            "ActionTypeId": {
                "Version": "1",
                "Owner": "AWS",
                "Category": "Source",
                "Provider": "ECR"
            },
            "OutputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "RunOrder": 1,
            "Configuration": {
                "ImageTag": "latest",
                "RepositoryName": "my-image-repo"
            },
            "Name": "ImageSource"
        }
    ]
},
```

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 – This tutorial provides a sample AppSpec 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.

`ECRBuildAndPublish` build action reference

This build action allows you to automate building and pushing a new image when a change occurs in your source. This action builds based on a specified Dockerfile location and pushes the image. This build action is not the same as the Amazon ECR source action in CodePipeline, which triggers

pipeline when a change occurs in your Amazon ECR source repository. For information about that action, see Amazon ECR source action reference.

This is not a source action that will trigger the pipeline. This action builds an image and pushes it to your Amazon ECR image repository.

You must have already created an Amazon ECR repository and have added a Dockerfile to your source code repository, such as GitHub, before you add the action to your pipeline.

Important

This action uses CodePipeline managed CodeBuild compute to run commands in a build environment. Running the commands action will incur separate charges in AWS CodeBuild.

Note

This action is only available for V2 type pipelines.

Topics

- Action type
- Configuration parameters
- Input artifacts
- Output artifacts
- Output variables
- Service role permissions: ECRBuildAndPublish action
- Action declaration
- See also

Action type

- Category: `Build`
- Owner: `AWS`
- Provider: `ECRBuildAndPublish`
- Version: `1`

Configuration parameters

ECRRepositoryName

Required: Yes

The name of the Amazon ECR repository where the image is pushed.

DockerFilePath

Required: No

The location of the Dockerfile used to build the image. Optionally, you can provide an alternate Dockerfile location if not at the root level.

> **Note**
>
> If a value for `DockerFilePath` is not specified, the value defaults to the source repository root level.

ImageTags

Required: No

The tags used for the image. You can enter multiple tags as a comma-delimited list of strings.

> **Note**
>
> If a value for `ImageTags` is not specified, the value defaults to `latest`.

RegistryType

Required: No

Specifies whether the repository is public or private. Valid values are `private | public`.

> **Note**
>
> If a value for `RegistryType` is not specified, the value defaults to `private`.

Input artifacts

- **Number of artifacts:** 1
- **Description:** The artifact produced by the source action that contains the Dockerfile needed to build the image.

Output artifacts

- **Number of artifacts:** `0`

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 reference.

ECRImageDigestId

The `sha256` digest of the image manifest.

ECRRepositoryName

The name of the Amazon ECR repository where the image was pushed.

Service role permissions: `ECRBuildAndPublish` action

For the `ECRBuildAndPublish` action support, add the following to your policy statement:

```
{
    "Statement": [
         {
            "Sid": "ECRRepositoryAllResourcePolicy",
            "Effect": "Allow",
            "Action": [
                "ecr:DescribeRepositories",
                "ecr:GetAuthorizationToken",
                "ecr-public:DescribeRepositories",
```

```
                "ecr-public:GetAuthorizationToken"
            ],
        "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken",
                "ecr:InitiateLayerUpload",
                "ecr:UploadLayerPart",
                "ecr:CompleteLayerUpload",
                "ecr:PutImage",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchCheckLayerAvailability"
            ],
            "Resource": "PrivateECR_Resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr-public:GetAuthorizationToken",
                "ecr-public:DescribeRepositories",
                "ecr-public:InitiateLayerUpload",
                "ecr-public:UploadLayerPart",
                "ecr-public:CompleteLayerUpload",
                "ecr-public:PutImage",
                "ecr-public:BatchCheckLayerAvailability",
                "sts:GetServiceBearerToken"
            ],
            "Resource": "PublicECR_Resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sts:GetServiceBearerToken"
            ],
            "Resource": "*"
        }
    ]
}
```

In addition, if not already added for the `Commands` action, add the following permissions to your service role in order to view CloudWatch logs.

```
{
    "Effect": "Allow",
    "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
    ],
    "Resource": "resource_ARN"
},
```

Note

Scope down the permissions to the pipeline resource level by using resource-based permissions in the service role policy statement.

For more information about this action, see ECRBuildAndPublish build action reference.

Action declaration

YAML

```
name: ECRBuild
actionTypeId:
  category: Build
  owner: AWS
  provider: ECRBuildAndPublish
  version: '1'
runOrder: 1
configuration:
  ECRRepositoryName: actions/my-imagerepo
outputArtifacts: []
inputArtifacts:
- name: SourceArtifact
region: us-east-1
namespace: BuildVariables
```

JSON

```
{
    "name": "ECRBuild",
```

```
    "actionTypeId": {
        "category": "Build",
        "owner": "AWS",
        "provider": "ECRBuildAndPublish",
        "version": "1"
    },
    "runOrder": 1,
    "configuration": {
        "ECRRepositoryName": "actions/my-imagerepo"
    },
    "outputArtifacts": [],
    "inputArtifacts": [
        {
            "name": "SourceArtifact"
        }
    ],
    "region": "us-east-1",
    "namespace": "BuildVariables"
},
```

See also

The following related resources can help you as you work with this action.

- Tutorial: Build and push a Docker image to Amazon ECR with CodePipeline (V2 type) – This tutorial provides a sample Dockerfile and instructions to create a pipeline that pushes your image to ECR on a change to your source repository and then deploys to Amazon ECS.

Amazon Elastic Container Service and CodeDeploy blue-green deploy action reference

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

The completed pipeline detects changes to your images or task definition file and uses CodeDeploy to route and deploy traffic to an Amazon ECS cluster and load balancer. CodeDeploy creates a new listener on your load balancer which can target your new task through a special port. You can

also configure the pipeline to use a source location, such as a CodeCommit repository, where your Amazon ECS task definition is stored.

Before you create your pipeline, you must have already created the Amazon ECS resources, the CodeDeploy resources, and the load balancer and target group. You must have already tagged and stored the image in your image repository, and uploaded the task definition and AppSpec file to your file repository.

> **Note**
>
> This topic describes the Amazon ECS to CodeDeploy blue/green deployment action for CodePipeline. For reference information about Amazon ECS standard deployment actions in CodePipeline, see Amazon Elastic Container Service deploy action reference.

Topics

- Action type
- Configuration parameters
- Input artifacts
- Output artifacts
- Service role permissions: CodeDeployToECS action
- Action declaration
- See also

Action type

- Category: `Deploy`
- Owner: `AWS`
- Provider: `CodeDeployToECS`
- Version: `1`

Configuration parameters

ApplicationName

Required: Yes

The name of the application in CodeDeploy. Before you create your pipeline, you must have already created the application in CodeDeploy.

DeploymentGroupName

Required: Yes

The deployment group specified for Amazon ECS task sets that you created for your CodeDeploy application. Before you create your pipeline, you must have already created the deployment group in CodeDeploy.

TaskDefinitionTemplateArtifact

Required: Yes

The name of the input artifact that provides the task definition file to the deployment action. This is generally the name of the output artifact from the source action. When you use the console, the default name for the source action output artifact is `SourceArtifact`.

AppSpecTemplateArtifact

Required: Yes

The name of the input artifact that provides the AppSpec file to the deployment action. This value is updated when your pipeline runs. This is generally the name of the output artifact from the source action. When you use the console, the default name for the source action output artifact is `SourceArtifact`. For `TaskDefinition` in AppSpec file, you can keep the `<TASK_DEFINITION>` placeholder text as shown here.

AppSpecTemplatePath

Required: No

The file name of the AppSpec file stored in the pipeline source file location, such as your pipeline's CodeCommit repository. The default file name is `appspec.yaml`. If your AppSpec file has the same name and is stored at the root level in your file repository, you do not need to provide the file name. If the path is not the default, enter the path and file name.

TaskDefinitionTemplatePath

Required: No

The file name of the task definition stored in the pipeline file source location, such as your pipeline's CodeCommit repository. The default file name is `taskdef.json`. If your task definition file has the same name and is stored at the root level in your file repository, you do not need to provide the file name. If the path is not the default, enter the path and file name.

Image<Number>ArtifactName

Required: No

The name of the input artifact that provides the image to the deployment action. This is generally the image repository's output artifact, such as output from the Amazon ECR source action.

Available values for `<Number>` are 1 through 4.

Image<Number>ContainerName

Required: No

The name of the image available from the image repository, such as the Amazon ECR source repository.

Available values for `<Number>` are 1 through 4.

Input artifacts

- **Number of Artifacts:** `1 to 5`
- **Description:** The `CodeDeployToECS` action first looks for the task definition file and the AppSpec file in the source file repository, next looks for the image in the image repository, then dynamically generates a new revision of task definition, and finally runs the AppSpec commands to deploy the task set and container to the cluster.

  The `CodeDeployToECS` action looks for an `imageDetail.json` file that maps the image URI to the image. When you commit a change to your Amazon ECR image repository, the pipeline ECR source action creates an `imageDetail.json` file for that commit. You can also manually add an `imageDetail.json` file for a pipeline where the action is not automated. For information about

the `imageDetail.json` file, see [imageDetail.json file for Amazon ECS blue/green deployment actions](#).

The `CodeDeployToECS` action dynamically generates a new revision of the task definition. In this phase, this action replaces placeholders in task definition file into image URI retrieved from imageDetail.json files. For example, if you set *IMAGE1_NAME* as Image1ContainerName parameter, you should specify the placeholder *<IMAGE1_NAME>* as the value of image field in your task definition file. In this case, the CodeDeployToECS action replaces the placeholder *<IMAGE1_NAME>* into actual image URI retrieved from imageDetail.json in the artifact which you specify as Image1ArtifactName.

For task definition updates, the CodeDeploy `AppSpec.yaml` file contains the `TaskDefinition` property.

```
TaskDefinition: <TASK_DEFINITION>
```

This property will be updated by the `CodeDeployToECS` action after the new task definition is created.

For the value of the `TaskDefinition` field, the placeholder text must be <TASK_DEFINITION>. The `CodeDeployToECS` action replaces this placeholder with the actual ARN of the dynamically generated task definition.

Output artifacts

- **Number of Artifacts:** `0`
- **Description:** Output artifacts do not apply for this action type.

Service role permissions: `CodeDeployToECS` action

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

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCodeDeployDeploymentActions",
            "Action": [
                "codedeploy:CreateDeployment",
                "codedeploy:GetDeployment"
            ],
            "Resource": [
                "arn:aws-eusc:codedeploy:*:111122223333:deploymentgroup:
[[ApplicationName]]/*"
            ],
            "Effect": "Allow"
        },
        {
            "Sid": "AllowCodeDeployApplicationActions",
            "Action": [
                "codedeploy:GetApplication",
                "codedeploy:GetApplicationRevision",
                "codedeploy:RegisterApplicationRevision"
            ],
            "Resource": [
                "arn:aws-eusc:codedeploy:*:111122223333:application:
[[ApplicationName]]",
                "arn:aws-eusc:codedeploy:*:111122223333:application:
[[ApplicationName]]/*"
            ],
            "Effect": "Allow"
        },
        {
            "Sid": "AllowCodeDeployDeploymentConfigAccess",
            "Action": [
                "codedeploy:GetDeploymentConfig"
            ],
            "Resource": [
                "arn:aws-eusc:codedeploy:*:111122223333:deploymentconfig:*"
            ],
            "Effect": "Allow"
        },
        {
            "Sid": "AllowECSRegisterTaskDefinition",
            "Action": [
                "ecs:RegisterTaskDefinition"
            ],
```

```
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
        },
        {
            "Sid": "AllowPassRoleToECS",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws-eusc:iam::111122223333:role/[[PassRoles]]"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": [
                        "ecs.amazonaws.com",
                        "ecs-tasks.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

You can opt in to using tagging authorization in Amazon ECS. By opting in, you must grant the following permissions: `ecs:TagResource`. For more information about how to opt in and to determine whether the permission is required and tag authorization is enforced, see Tagging authorization timeline in the Amazon Elastic Container Service Developer Guide.

You must also add the `iam:PassRole` permissions to use IAM roles for tasks. For more information, see Amazon ECS task execution IAM role and IAM Roles for Tasks.

You can also add `ecs-tasks.amazonaws.com` to the list of services under the `iam:PassedToService` condition, as shown in the above example.

Action declaration

YAML

```
Name: Deploy
Actions:
```

```
  - Name: Deploy
    ActionTypeId:
      Category: Deploy
      Owner: AWS
      Provider: CodeDeployToECS
      Version: '1'
    RunOrder: 1
    Configuration:
      AppSpecTemplateArtifact: SourceArtifact
      ApplicationName: ecs-cd-application
      DeploymentGroupName: ecs-deployment-group
      Image1ArtifactName: MyImage
      Image1ContainerName: IMAGE1_NAME
      TaskDefinitionTemplatePath: taskdef.json
      AppSpecTemplatePath: appspec.yaml
      TaskDefinitionTemplateArtifact: SourceArtifact
    OutputArtifacts: []
    InputArtifacts:
      - Name: SourceArtifact
      - Name: MyImage
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
        {
            "Name": "Deploy",
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "CodeDeployToECS",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "AppSpecTemplateArtifact": "SourceArtifact",
                "ApplicationName": "ecs-cd-application",
                "DeploymentGroupName": "ecs-deployment-group",
                "Image1ArtifactName": "MyImage",
                "Image1ContainerName": "IMAGE1_NAME",
```

```
                "TaskDefinitionTemplatePath": "taskdef.json",
                "AppSpecTemplatePath": "appspec.yaml",
                "TaskDefinitionTemplateArtifact": "SourceArtifact"
            },
            "OutputArtifacts": [],
            "InputArtifacts": [
                {
                    "Name": "SourceArtifact"
                },
                {
                    "Name": "MyImage"
                }
            ],
            "Region": "us-west-2",
            "Namespace": "DeployVariables"
        }
    ]
}
```

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 – This tutorial walks you through creation of the CodeDeploy and Amazon ECS resources you need for a blue/green deployment. The tutorial shows you how to push a Docker image to Amazon ECR and create an Amazon ECS task definition that lists your Docker image name, container name, Amazon ECS service name, and load balancer configuration. The tutorial then walks you through creating the AppSpec file and pipeline for your deployment.

  > **Note**
  >
  > This topic and tutorial describe the CodeDeploy/ECS blue/green action for CodePipeline. For information about ECS standard actions in CodePipeline, see Tutorial: Continuous Deployment with CodePipeline.

- *AWS CodeDeploy User Guide* – For information about how to use the load balancer, production listener, target groups, and your Amazon ECS application in a blue/green deployment, see Tutorial: Deploy an Amazon ECS Service. This reference information in the *AWS CodeDeploy User Guide* provides an overview for blue/green deployments with Amazon ECS and AWS CodeDeploy.

- *Amazon Elastic Container Service Developer Guide* – For information about working with Docker images and containers, ECS services and clusters, and ECS task sets, see What Is Amazon ECS?

Amazon Elastic Container Service deploy action reference

You can use an Amazon ECS action to deploy an Amazon ECS service and task set. An Amazon ECS service is a container application that is deployed to an Amazon ECS cluster. An Amazon ECS cluster is a collection of instances that host your container application in the cloud. The deployment requires a task definition that you create in Amazon ECS and an image definitions file that CodePipeline uses to deploy the image.

Important

The Amazon ECS standard deployment action for CodePipeline creates its own revision of the task definition based on the revision used by the Amazon ECS service. If you create new revisions for the task definition without updating the Amazon ECS service, the deployment action will ignore those revisions.

Before you create your pipeline, you must have already created the Amazon ECS resources, tagged and stored the image in your image repository, and uploaded the buildspec file to your file repository.

Note

This reference topic describes the Amazon ECS standard deployment action for CodePipeline. For reference information about Amazon ECS to CodeDeploy blue/green deployment actions in CodePipeline, see Amazon Elastic Container Service and CodeDeploy blue-green deploy action reference.

Topics

- Action type
- Configuration parameters
- Input artifacts
- Output artifacts

- Service role permissions: Amazon ECS standard action
- Action declaration
- See also

Action type

- Category: `Deploy`
- Owner: `AWS`
- Provider: `ECS`
- Version: 1

Configuration parameters

ClusterName

Required: Yes

The Amazon ECS cluster in Amazon ECS.

ServiceName

Required: Yes

The Amazon ECS service that you created in Amazon ECS.

FileName

Required: No

The name of your image definitions file, the JSON file that describes your service's container name and the image and tag. You use this file for ECS standard deployments. For more information, see Input artifacts and imagedefinitions.json file for Amazon ECS standard deployment actions.

DeploymentTimeout

Required: No

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"
```

Input artifacts

- **Number of artifacts:** 1
- **Description:** The action looks for an `imagedefinitions.json` file in the source file repository for the pipeline. An image definitions document is a JSON file that describes your Amazon ECS container name and the image and tag. CodePipeline uses the file to retrieve the image from your image repository such as Amazon ECR. You can manually add an `imagedefinitions.json` file for a pipeline where the action is not automated. For information about the `imagedefinitions.json` file, see imagedefinitions.json file for Amazon ECS standard deployment actions.

  The action requires an existing image that has already been pushed to your image repository. Because the image mapping is provided by the `imagedefinitions.json` file, the action does not require that the Amazon ECR source be included as a source action in the pipeline.

Output artifacts

- **Number of artifacts:** `0`
- **Description:** Output artifacts do not apply for this action type.

Service role permissions: Amazon ECS standard action

For Amazon ECS, the following are the minimum permissions needed to create pipelines with an Amazon ECS deploy action.

JSON

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

```
                "ecs:DescribeTaskDefinition",
                "ecs:RegisterTaskDefinition"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Sid": "ECSServicePermissions",
            "Effect": "Allow",
            "Action": [
                "ecs:DescribeServices",
                "ecs:UpdateService"
            ],
            "Resource": [
                "arn:aws-eusc:ecs:*:111122223333:service/[[clusters]]/*"
            ]
        },
        {
            "Sid": "ECSTagResource",
            "Effect": "Allow",
            "Action": [
                "ecs:TagResource"
            ],
            "Resource": [
                "arn:aws-eusc:ecs:*:111122223333:task-definition/
[[taskDefinitions]]:*"
            ],
            "Condition": {
                "StringEquals": {
                    "ecs:CreateAction": [
                        "RegisterTaskDefinition"
                    ]
                }
            }
        },
        {
            "Sid": "IamPassRolePermissions",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [
                "arn:aws-eusc:iam::111122223333:role/[[passRoles]]"
            ],
            "Condition": {
```

```
                "StringEquals": {
                    "iam:PassedToService": [
                        "ecs.amazonaws.com",
                        "ecs-tasks.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

You can opt in to using tagging authorization in Amazon ECS. By opting in, you must grant the following permissions: `ecs:TagResource`. For more information about how to opt in and to determine whether the permission is required and tag authorization is enforced, see Tagging authorization timeline in the Amazon Elastic Container Service Developer Guide.

You must add the `iam:PassRole` permissions to use IAM roles for tasks. For more information, see Amazon ECS task execution IAM role and IAM Roles for Tasks. Use the following policy text.

Action declaration

YAML

```
Name: DeployECS
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: ECS
  Version: '1'
RunOrder: 2
Configuration:
  ClusterName: my-ecs-cluster
  ServiceName: sample-app-service
  FileName: imagedefinitions.json
  DeploymentTimeout: '15'
OutputArtifacts: []
InputArtifacts:
  - Name: my-image
```

JSON

```
{
    "Name": "DeployECS",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "ECS",
        "Version": "1"
    },
    "RunOrder": 2,
    "Configuration": {
        "ClusterName": "my-ecs-cluster",
        "ServiceName": "sample-app-service",
        "FileName": "imagedefinitions.json",
        "DeploymentTimeout": "15"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "my-image"
        }
    ]
},
```

See also

The following related resources can help you as you work with this action.

- See Tutorial: Build and push a Docker image to Amazon ECR with CodePipeline (V2 type) for a tutorial that shows you how to use the ECRBuildandPublish action to push an image and then use the ECS standard action to deploy to Amazon ECS.
- Tutorial: Continuous Deployment with CodePipeline – This tutorial shows you how to create a Dockerfile that you store in a source file repository such as CodeCommit. Next, the tutorial shows you how to incorporate a CodeBuild buildspec file that builds and pushes your Docker image to Amazon ECR and creates your imagedefinitions.json file. Finally, you create an Amazon ECS service and task definition, and then you create your pipeline with an Amazon ECS deployment action.

> **Note**
>
> This topic and tutorial describe the Amazon ECS standard deployment action for CodePipeline. For information about Amazon ECS to CodeDeploy blue/green deployment actions in CodePipeline, see Tutorial: Create a pipeline with an Amazon ECR source and ECS-to-CodeDeploy deployment.

- *Amazon Elastic Container Service Developer Guide* – For information about working with Docker images and containers, Amazon ECS services and clusters, and Amazon ECS task sets, see What Is Amazon ECS?

Amazon Elastic Kubernetes Service EKS deploy action reference

You can use the `EKSDeploy` action to deploy an Amazon EKS service. The deployment requires a Kubernetes manifest file that CodePipeline uses to deploy the image.

Before you create your pipeline, you must have already created the Amazon EKS resources and have stored the image in your image repository. Optionally, you can provide VPC information for your cluster.

> **Important**
>
> This action uses CodePipeline managed CodeBuild compute to run commands in a build environment. Running the commands action will incur separate charges in AWS CodeBuild.

> **Note**
>
> The EKS deploy action is only available for V2 type pipelines.

The EKS action supports both public and private EKS clusters. Private clusters are the type recommended by EKS; however, both types are supported.

The EKS action is supported for cross-account actions. To add a cross-account EKS action, add `actionRoleArn` from your target account in the action declaration.

Topics

- Action type
- Configuration parameters
- Input artifacts
- Output artifacts
- Environment variables
- Output variables
- Service role policy permissions
- Action declaration
- See also

Action type

- Category: `Deploy`
- Owner: `AWS`
- Provider: `EKS`
- Version: 1

Configuration parameters

ClusterName

Required: Yes

The Amazon EKS cluster in Amazon EKS.

Options under Helm

The following are available options when **Helm** is the selected deployment tool.

HelmReleaseName

Required: Yes (Required only for **Helm** type)

The release name for your deployment.

HelmChartLocation

Required: Yes (Required only for **Helm** type)

The chart location for your deployment.

HelmValuesFiles

Required: No (Optional only for **Helm** type)

To override for Helm values files, enter the comma-separated Helm values files in the Helm chart location.

Options under Kubectl

The following are available options when **Kubectl** is the selected deployment tool.

ManifestFiles

Required: Yes (Required only for **Kubectl** type)

The name of your manifest file, the text file that describes your service's container name and the image and tag. You use this file to parameterize your image URI and other information. You can use an environment variable for this purpose.

You store this file in the source repository for your pipeline.

Namespace

Required: No

The Kubernetes namespace to be used in `kubectl` or `helm` commands.

Subnets

Required: No

The subnets for the VPC for your cluster. These are part of the same VPC that is attached to your cluster. You can also provide subnets that aren't already attached to your cluster and specify them here.

SecurityGroupIds

Required: No

The security groups for the VPC for your cluster. These are part of the same VPC that is attached to your cluster. You can also provide security groups that aren't already attached to your cluster and specify them here.

Input artifacts

- **Number of artifacts:** 1
- **Description:** The action looks for the Kubernetes manifest file or Helm chart in the source file repository for the pipeline. If you want to use Helm charts in .tgz format stored in an S3 bucket, you can do so by configuring the S3 Bucket/Key as your source action. For example, the object key provided would be `my-chart-0.1.0.tgz`.

Output artifacts

- **Number of artifacts:** `0`
- **Description:** Output artifacts do not apply for this action type.

Environment variables

Used to replace variables such as image repositories or image tags in manifest files or Helm chart values files.

Key

The key in a key-value environment variable pair, such as `$IMAGE_TAG`.

Value

The value for the key-value pair, such as `v1.0`. The value can be parameterized with output variables from pipeline actions or pipeline variables. For example, pipeline can have an ECRBuildAndPublish action that creates an ECR image with `${codepipeline.PipelineExecutionId}`, and the EKS action can use this image using `${codepipeline.PipelineExecutionId}` as the value of the environment variable.

Output variables

EKSClusterName

The Amazon EKS cluster in Amazon EKS.

Service role policy permissions

To run this action, the following permissions must be available in your pipeline's service role policy.

- **EC2 actions:** When CodePipeline runs the action EC2 instance permissions are required. Note that this is not the same as the EC2 instance role required when you create your EKS cluster.

  If you are using an existing service role, to use this action, you will need to add the following permissions for the service role.

  - ec2:CreateNetworkInterface
  - ec2:DescribeDhcpOptions
  - ec2:DescribeNetworkInterfaces
  - ec2:DeleteNetworkInterface
  - ec2:DescribeSubnets
  - ec2:DescribeSecurityGroups
  - ec2:DescribeVpcs
- **EKS actions:** When CodePipeline runs the action, EKS cluster permissions are required. Note that this is not the same as the IAM EKS cluster role required when you create your EKS cluster.

  If you are using an existing service role, to use this action, you will need to add the following permission for the service role.

  - eks:DescribeCluster
- **Log stream actions:** When CodePipeline runs the action, CodePipeline creates a log group using the name of the pipeline as follows. This enables you to scope down permissions to log resources using the pipeline name.

  ```
  /aws/codepipeline/MyPipelineName
  ```

  If you are using an existing service role, to use this action, you will need to add the following permissions for the service role.

  - logs:CreateLogGroup
  - logs:CreateLogStream
  - logs:PutLogEvents

In the service role policy statement, scope down the permissions to the resource level as shown in the following example.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster"
            ],
            "Resource": "arn:aws-
eusc:eks:*:111122223333:cluster/YOUR_CLUSTER_NAME"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:CreateNetworkInterfacePermission",
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeVpcs",
                "ec2:DescribeRouteTables"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:CreateLogGroup",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws-eusc:logs:*:111122223333:log-group:/aws/
codepipeline/YOUR_PIPELINE_NAME",
                "arn:aws-eusc:logs:*:111122223333:log-group:/aws/
codepipeline/YOUR_PIPELINE_NAME:*"
```

```
            ]
        }
    ]
}
```

To view logs in the console using the action details dialog page, the permission to view logs must be added to the console role. For more information, see the console permissions policy example in Permissions required to view compute logs in the console.

Adding the service role as an access entry for your cluster

After the permissions are available in your pipeline's service role policy, you configure your cluster permissions by adding the CodePipeline service role as an access entry for your cluster.

You can also use an action role that has the updated permissions. For more information, see the tutorial example in Step 4: Create an access entry for the CodePipeline service role.

Action declaration

YAML

```
Name: DeployEKS
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: EKS
  Version: '1'
RunOrder: 2
Configuration:
  ClusterName: my-eks-cluster
  ManifestFiles: ManifestFile.json
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
```

JSON

```
{
    "Name": "DeployECS",
    "ActionTypeId": {
```

```
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "EKS",
        "Version": "1"
    },
    "RunOrder": 2,
    "Configuration": {
        "ClusterName": "my-eks-cluster",
        "ManifestFiles": "ManifestFile.json"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ]
},
```

See also

The following related resources can help you as you work with this action.

- See Tutorial: Deploy to Amazon EKS with CodePipeline for a tutorial that demonstrates how to create an EKS cluster and Kubernetes manifest file to add the action to your pipeline.

AWS Lambda deploy action reference

You use an AWS Lambda deploy action to manage deploying your application code for your serverless deployment. You can deploy a function and use deployment strategies for traffic deployment as follows:

- Canary and linear deployments for traffic shifting
- All at once deployments

> **Note**
>
> This action is only supported for V2 type pipelines.

Topics

- Action type
- Configuration parameters
- Input artifacts
- Output artifacts
- Output variables
- Service role policy permissions for the Lambda deploy action
- Action declaration
- See also

Action type

- Category: `Deploy`
- Owner: `AWS`
- Provider: `Lambda`
- Version: `1`

Configuration parameters

FunctionName

Required: Yes

The name of the function that you created in Lambda, such as `MyLambdaFunction`.

You must have already created a version.

FunctionAlias

Required: No

The alias of the function that you created in Lambda and is the function to be deployed to, such as `live`. The alias must exist and have one version behind it when the action execution starts. (It will be the rollback target version.)

If not provided, the action deploys the source artifact to `$LATEST` and creates a new version. In this use case, the deploy strategy and target version options are not available.

PublishedTargetVersion

Required: No

The desired Lambda Function version to be deployed to FunctionAlias. It can be pipeline or action level variables, such as `#{variables.lambdaTargetVersion}`. The version must be published when action execution starts.

Required if no input artifact is provided.

DeployStrategy

Required: No (Default is `AllAtOnce`)

Determines the rate that the Lambda deploy action adopts to shift traffic from the original version of the Lambda function to the new version for **FunctionAlias**. Available deploy strategies are canary or linear. Accepted formats:

- `AllAtOnce` -

  Shifts all traffic to the updated Lambda functions at once.

  If not specified, the default is `AllAtOnce`.
- `Canary10Percent5Minutes` - Shifts 10 percent of traffic in the first increment. The remaining 90 percent is deployed five minutes later.

  The values for both percentage and minutes can be changed.
- `Linear10PercentEvery1Minute` - Shifts 10 percent of traffic every minute until all traffic is shifted.

  The values for both percentage and minutes can be changed.

The following considerations apply for this field:

- Maximum total wait time is 2 days.
- Only available when **FunctionAlias** is provided.

Alarms

Required: No

A comma-separated list of alarm names configured for the Lambda deployment. A maximum of 10 alarms can be added. The action fails when monitored alarms go to the ALARM state.

The following image shows an example of the Edit page for the action.

Edit action

Action name
Choose a name for your action
Deploy
No more than 100 characters

Action provider
AWS Lambda

Region
United States (N. Virginia)

Input artifacts
Choose an input artifact for this action. Learn more

SourceArtifact
Defined by: Source
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 AWS Lambda console and then return to this task.
MyLambdaFunction

Function alias
Choose an alias of the Lambda Function to be deployed to, such as live. The alias must exist and have a version behind it when the action executions starts (it will be the rollback target version). If not provided, action deploys source artifact to $LATEST and creates a new version. In this use case, deploy strategy and target version options are not available.
aliasV1

Deploy strategy
Choose the rate at which the Lambda deploy action adopts to shift traffic from the original version of the Lambda function to the new version.

Deployment strategy type
Choose the deployment type for the Lambda deploy action.
Canary deployment

Traffic Percentage
Percentage of traffic to shift in each increment (1-100)
10

Time interval (minutes)
Time between traffic shifts
5

Alarms
A list of alarm names configured for the lambda deployment. A maximum of 10 alarms can be added. The Lambda deploy action fails when any monitored alarm goes into alarm state.
CloudWatch Alarms

Cancel Done

Input artifacts

- **Number of artifacts:** 1
- **Description:** The files provided, if any, to support the script actions during the deployment.

Output artifacts

- **Number of artifacts:** 0
- **Description:** Output artifacts do not apply for this action type.

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 reference.

FunctionVersion

The new Lambda function version that was deployed.

Service role policy permissions for the Lambda deploy action

When CodePipeline runs the action, CodePipeline service role requires the following permissions, appropriately scoped down for access with least privilege.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "StatementForLambda",
            "Effect": "Allow",
            "Action": [
                "lambda:GetAlias",
```

```
                "lambda:GetFunctionConfiguration",
                "lambda:GetProvisionedConcurrencyConfig",
                "lambda:PublishVersion",
                "lambda:UpdateAlias",
                "lambda:UpdateFunctionCode"
            ],
            "Resource": [
                "arn:aws-eusc:lambda:eusc-de-east-1:111122223333:function:
{{FunctionName}}",
                "arn:aws-eusc:lambda:eusc-de-east-1:111122223333:function:
{{FunctionName}}:*"
            ]
        },
        {
            "Sid": "StatementForCloudWatch",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:DescribeAlarms"
            ],
            "Resource": [
                "arn:aws-eusc:cloudwatch:eusc-de-east-1:111122223333:alarm:
{{AlarmNames}}"
            ]
        },
        {
            "Sid": "StatementForLogs1",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup"
            ],
            "Resource": [
                "arn:aws-eusc:logs:eusc-de-east-1:111122223333:log-group:/eusc-
de-east-1/codepipeline/{{pipelineName}}",
                "arn:aws-eusc:logs:eusc-de-east-1:111122223333:log-group:/eusc-
de-east-1/codepipeline/{{pipelineName}}:*"
            ]
        },
        {
            "Sid": "StatementForLogs2",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
```

```
            "Resource": [
                "arn:aws-eusc:logs:eusc-de-east-1:111122223333:log-group:/eusc-
de-east-1/codepipeline/{{pipelineName}}:log-stream:*"
            ]
        }
    ]
}
```

Action declaration

YAML

```
name: Deploy
actionTypeId:
  category: Deploy
  owner: AWS
  provider: Lambda
  version: '1'
runOrder: 1
configuration:
  DeployStrategy: Canary10Percent5Minutes
  FunctionAlias: aliasV1
  FunctionName: MyLambdaFunction
outputArtifacts: []
inputArtifacts:
- name: SourceArtifact
region: us-east-1
namespace: DeployVariables
```

JSON

```
{
    "name": "Deploy",
    "actionTypeId": {
        "category": "Deploy",
        "owner": "AWS",
        "provider": "Lambda",
        "version": "1"
    },
    "runOrder": 1,
    "configuration": {
        "DeployStrategy": "Canary10Percent5Minutes",
```

```
        "FunctionAlias": "aliasV1",
        "FunctionName": "MyLambdaFunction"
    },
    "outputArtifacts": [],
    "inputArtifacts": [
        {
            "name": "SourceArtifact"
        }
    ],
    "region": "us-east-1",
    "namespace": "DeployVariables"
},
```

See also

The following related resources can help you as you work with this action.

- Tutorial: Lambda function deployments with CodePipeline – This tutorial walks you through the creation of a sample Lambda function where you will create an alias and version, add the zipped Lambda function to your source location, and run the Lambda action in your pipeline.

Amazon S3 deploy action reference

You use an Amazon S3 deploy action to deploy files to an Amazon S3 bucket for static website hosting or archive. You can specify whether to extract deployment files before upload to your bucket.

> **Note**
>
> This reference topic describes the Amazon S3 deployment action for CodePipeline where the deployment platform is an Amazon S3 bucket configured for hosting. For reference information about the Amazon S3 source action in CodePipeline, see Amazon S3 source action reference.

Topics

- Action type
- Configuration parameters

- Input artifacts
- Output artifacts
- Service role permissions: S3 deploy action
- Example action configuration
- See also

Action type

- Category: `Deploy`
- Owner: `AWS`
- Provider: `S3`
- Version: 1

Configuration parameters

BucketName

Required: Yes

The name of the Amazon S3 bucket where files are to be deployed.

Extract

Required: Yes

If true, specifies that files are to be extracted before upload. Otherwise, application files remain zipped for upload, such as in the case of a hosted static website. If false, then the `ObjectKey` is required.

ObjectKey

Conditional. Required if `Extract` = false

The name of the Amazon S3 object key that uniquely identifies the object in the S3 bucket.

KMSEncryptionKeyARN

Required: No

The ARN of the AWS KMS encryption key for the host bucket. The `KMSEncryptionKeyARN` parameter encrypts uploaded artifacts with the provided AWS KMS key. For a 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 KMS key. For cross-account actions, you can only use the key ID or key ARN to identify the key. Cross-account actions involve using the role from the other account (AccountB), so specifying the key ID will use the key from the other account (AccountB).

> **Important**
>
> CodePipeline only supports symmetric KMS keys. Do not use an asymmetric KMS key to encrypt the data in your S3 bucket.

CannedACL

Required: No

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.

CacheControl

Required: No

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"
```

Input artifacts

- **Number of Artifacts:** 1

- **Description:** The files for deployment or archive are obtained from the source repository, zipped, and uploaded by CodePipeline.

Output artifacts

- **Number of artifacts:** `0`
- **Description:** Output artifacts do not apply for this action type.

Service role permissions: S3 deploy action

For S3 deploy action support, add the following to your policy statement:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl",
        "s3:PutObjectVersionAcl",
        "s3:GetBucketVersioning",
        "s3:GetBucketAcl",
        "s3:GetBucketLocation"
      ],
      "Resource": [
        "arn:aws-eusc:s3:::[[s3DeployBuckets]]",
        "arn:aws-eusc:s3:::[[s3DeployBuckets]]/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
        }
      }
    }
  ]
}
```

For the S3 deploy action support, if your S3 objects have tags, you must also add the following permissions to your policy statement:

```
"s3:GetObjectTagging",
"s3:GetObjectVersionTagging",
"s3:PutObjectTagging"
```

Example action configuration

The following show examples for the action configuration.

Example configuration when `Extract` is set to `false`

The following example shows the default action configuration when the action is created with the `Extract` field set to `false`.

YAML

```
Name: Deploy
Actions:
  - Name: Deploy
    ActionTypeId:
      Category: Deploy
      Owner: AWS
      Provider: S3
      Version: '1'
    RunOrder: 1
    Configuration:
      BucketName: website-bucket
      Extract: 'false'
      ObjectKey: MyWebsite
    OutputArtifacts: []
    InputArtifacts:
      - Name: SourceArtifact
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
```

```
        {
            "Name": "Deploy",
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "S3",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "BucketName": "website-bucket",
                "Extract": "false",
                "ObjectKey": "MyWebsite"
                },
            "OutputArtifacts": [],
            "InputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "Region": "us-west-2",
            "Namespace": "DeployVariables"
        }
    ]
},
```

Example configuration when `Extract` is set to `true`

The following example shows the default action configuration when the action is created with the `Extract` field set to `true`.

YAML

```
Name: Deploy
Actions:
  - Name: Deploy
    ActionTypeId:
      Category: Deploy
      Owner: AWS
      Provider: S3
      Version: '1'
    RunOrder: 1
```

```
    Configuration:
      BucketName: website-bucket
      Extract: 'true'
    OutputArtifacts: []
    InputArtifacts:
      - Name: SourceArtifact
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
        {
            "Name": "Deploy",
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "S3",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "BucketName": "website-bucket",
                "Extract": "true"
                },
            "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.

- Tutorial: Create a pipeline that uses Amazon S3 as a deployment provider – This tutorial walks you through two examples for creating a pipeline with an S3 deploy action. You download sample files, upload the files to your CodeCommit repository, create your S3 bucket, and configure your bucket for hosting. Next, you use the CodePipeline console to create your pipeline and specify an Amazon S3 deployment configuration.
- Amazon S3 source action reference – This action reference provides reference information and examples for Amazon S3 source actions in CodePipeline.

Amazon S3 source action reference

A source action triggers the pipeline when a new object is uploaded to the configured bucket and object key.

This reference topic describes the Amazon S3 source action for CodePipeline where the source location is an Amazon S3 bucket configured for versioning. For reference information about the Amazon S3 deploy action in CodePipeline, see Amazon S3 deploy action reference.

Before you connect the pipeline, create an Amazon S3 source bucket, enable versioning, and upload your source files as a single ZIP file.

You can create a new bucket in Amazon S3 or select an existing Amazon S3 bucket to use as the source location for your application files. For more information about enabling versioning, see Using versioning in the *Amazon S3 User Guide*.

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

If you use the console to create or edit your pipeline, CodePipeline creates an EventBridge rule that starts your pipeline when a change occurs in the Amazon S3 source bucket.

You can also create a source override using input transform entry to use the `revisionValue` in EventBridge for your pipeline event, where the `revisionValue` is derived from the source event variable for your object key, commit, or image ID. For more information, see the optional step for input transform entry included in the procedures under Connecting to Amazon S3 source actions with a source enabled for events.

Topics

- Action type
- Configuration parameters
- Input artifacts
- Output artifacts
- Output variables
- Service role permissions: S3 source action
- Action declaration
- See also

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.

AllowOverrideForS3ObjectKey

Required: No

`AllowOverrideForS3ObjectKey` controls whether source overrides from `StartPipelineExecution` can override the already configured `S3ObjectKey` in the source

action. For more information on source overrides with the S3 Object Key, see Start a pipeline with a source revision override.

> **Important**
>
> If you omit `AllowOverrideForS3ObjectKey`, CodePipeline defaults the ability to override the S3 ObjectKey in the source action by setting this parameter to `false`.

Valid values for this parameter:

- `true`: If set, the pre-configured S3 Object Key can be overridden by source revision overrides during a pipeline execution.

  > **Note**
  >
  > If you intend to allow all CodePipeline users the ability to override the pre-configured S3 Object Key while starting a new pipeline execution, you must set `AllowOverrideForS3ObjectKey` to `true`.

- `false`:

  If set, CodePipeline will not allow the S3 Object Key to be overridden using source revision overrides. This is also the default value for this parameter.

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 Migrate polling pipelines with an S3 source and CloudTrail trail (CLI) or Migrate polling pipelines with an S3 source and CloudTrail trail (CloudFormation template).

> **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 reference.

BucketName

The name of the Amazon S3 bucket related to the source change that triggered the pipeline.

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.

ObjectKey

The name of the Amazon S3 object key related to the source change that triggered the pipeline.

VersionId

The version ID for the version of the object related to the source change that triggered the pipeline.

Service role permissions: S3 source action

For S3 source action support, add the following to your policy statement:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:GetBucketVersioning",
        "s3:GetBucketAcl",
        "s3:GetBucketLocation",
        "s3:GetObjectTagging",
        "s3:GetObjectVersionTagging"
      ],
      "Resource": [
        "arn:aws-eusc:s3:::[[S3Bucket]]",
        "arn:aws-eusc:s3:::[[S3Bucket]]/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceAccount": "111122223333"
        }
```

```
      }
    }
  ]
}
```

Action declaration

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: amzn-s3-demo-source-bucket
      S3ObjectKey: my-application.zip
      PollForSourceChanges: 'false'
    InputArtifacts: []
```

JSON

```
{
    "Name": "Source",
    "Actions": [
        {
            "RunOrder": 1,
            "OutputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "ActionTypeId": {
                "Provider": "S3",
                "Owner": "AWS",
```

```
                "Version": "1",
                "Category": "Source"
            },
            "Region": "us-west-2",
            "Name": "Source",
            "Configuration": {
                "S3Bucket": "amzn-s3-demo-source-bucket",
                "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) – This tutorial provides a sample AppSpec 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 AppConfig deploy action reference

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.

Service role permissions: AppConfig action

When CodePipeline runs the action, the CodePipeline service role policy requires the following permissions, appropriately scoped down to the resource level in order to maintain access with least privilege.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "appconfig:StartDeployment",
                "appconfig:StopDeployment",
                "appconfig:GetDeployment"
            ],
            "Resource": [
                "arn:aws-eusc:appconfig:*:111122223333:application/
[[Application]]",
                "arn:aws-eusc:appconfig:*:111122223333:application/
[[Application]]/*",
                "arn:aws-eusc:appconfig:*:111122223333:deploymentstrategy/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Example action configuration

YAML

```
name: Deploy
actions:
  - name: Deploy
    actionTypeId:
      category: Deploy
      owner: AWS
      provider: AppConfig
```

```
      version: '1'
    runOrder: 1
    configuration:
      Application: 2s2qv57
      ConfigurationProfile: PvjrpU
      DeploymentStrategy: frqt7ir
      Environment: 9tm27yd
      InputArtifactConfigurationPath: /
    outputArtifacts: []
    inputArtifacts:
      - name: SourceArtifact
    region: us-west-2
    namespace: DeployVariables
```

JSON

```
{
    "name": "Deploy",
    "actions": [
        {
            "name": "Deploy",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "provider": "AppConfig",
                "version": "1"
            },
            "runOrder": 1,
            "configuration": {
                "Application": "2s2qv57",
                "ConfigurationProfile": "PvjrpU",
                "DeploymentStrategy": "frqt7ir",
                "Environment": "9tm27yd",
                "InputArtifactConfigurationPath": "/"
            },
            "outputArtifacts": [],
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-west-2",
            "namespace": "DeployVariables"
```

```
        }
    ]
}
```

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 – 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.

CloudFormation deploy action reference

Executes an operation on an 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 CloudFormation template. A change set creates a comparison that can be viewed without altering the original stack. For information about the types of CloudFormation actions that can be performed on stacks and change sets, see the `ActionMode` parameter.

To construct an error message for an CloudFormation action where a stack operation has failed, CodePipeline calls the CloudFormation `DescribeStackEvents` API. If an action IAM role has permission to access that API, the details about the first failed resource will be included in the CodePipeline error message. Otherwise, if the role policy does not have the appropriate permission, CodePipeline will ignore accessing the API and show a generic error message instead. To do this, the `cloudformation:DescribeStackEvents` permission must be added to the service role or other IAM roles for the pipeline.

If you do not want the resource details surfaced in the pipeline error messages, you can revoke this permission for the action IAM role by removing the `cloudformation:DescribeStackEvents` permission.

Topics

- Action type

- Configuration parameters
- Input artifacts
- Output artifacts
- Output variables
- Service role permissions: CloudFormation action
- Action declaration
- See also

Action type

- Category: `Deploy`
- Owner: `AWS`
- Provider: `CloudFormation`
- Version: `1`

Configuration parameters

ActionMode

Required: Yes

`ActionMode` is the name of the action 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, 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, CloudFormation deletes it, and then creates a new one.
- `CREATE_UPDATE` creates the stack if it doesn't exist. If the stack exists, 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, CloudFormation deletes the stack, and then creates a new stack. If the stack isn't in a failed state, CloudFormation updates it.

  The stack is in a failed state when any of the following status types are displayed in 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 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 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 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.

> **Note**
>
> This role must be in the same account as the role for the action that is running, as configured in the action declaration `RoleArn`.

This property is required for the following action modes:

- CREATE_UPDATE
- REPLACE_ON_FAILURE
- DELETE_ONLY
- CHANGE_SET_REPLACE

> **Note**
>
> CloudFormation is given an S3-signed URL to the template; therefore, this `RoleArn` does not need permission to access the artifact bucket. However, the action `RoleArn` *does* need permission to access the artifact bucket, in order to generate the signed URL.

TemplatePath

Required: Conditional

`TemplatePath` represents the CloudFormation template file. You include the file in an input artifact to this action. The file name follows this format:

`Artifactname``::`*`TemplateFileName`*

`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 CloudFormation template file containing the template body has a minimum length of 1 byte and a maximum length of 1 MB. For CloudFormation deployment actions in CodePipeline, the maximum input artifact size is always 256 MB. For more information, see Quotas in AWS CodePipeline and 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 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 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 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 CloudFormation stack.

  For more information about the outputs section you can create for your 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 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:

- `CHANGE_SET_EXECUTE`
- `CHANGE_SET_REPLACE`
- `CREATE_UPDATE`

- REPLACE_ON_FAILURE

For more information, see Variables reference. 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.

Service role permissions: CloudFormation action

When CodePipeline runs the action, the CodePipeline service role policy requires the following permissions, appropriately scoped down to the pipeline resource ARN in order to maintain access with least privilege. For example, add the following to your policy statement:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCFNStackAccess",
            "Effect": "Allow",
            "Action": [
                "cloudformation:CreateStack",
                "cloudformation:UpdateStack",
                "cloudformation:DeleteStack",
                "cloudformation:DescribeStacks",
                "cloudformation:DescribeStackResources",
                "cloudformation:DescribeStackEvents",
                "cloudformation:GetTemplate",
                "cloudformation:DescribeChangeSet",
                "cloudformation:CreateChangeSet",
                "cloudformation:DeleteChangeSet",
                "cloudformation:ExecuteChangeSet"
            ],
            "Resource": [
                "arn:aws-eusc:cloudformation:*:111122223333:stack/
[[cfnDeployStackNames]]/*"
            ]
        },
        {
            "Sid": "ValidateTemplate",
```

```
            "Effect": "Allow",
            "Action": [
                "cloudformation:ValidateTemplate"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AllowIAMPassRole",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws-eusc:iam::111122223333:role/[[cfnExecutionRoles]]"
            ],
            "Condition": {
                "StringEqualsIfExists": {
                    "iam:PassedToService": [
                        "cloudformation.amazonaws.com"
                    ]
                }
            }
        }
    ]
}
```

Note that the `cloudformation:DescribeStackEvents` permission is optional. It allows the CloudFormation action to show a more detailed error message. This permission can be revoked from the IAM role if you don't want resource details surfaced in the pipeline error messages.

Action declaration

YAML

```
Name: ExecuteChangeSet
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormation
  Version: '1'
RunOrder: 2
Configuration:
```

```
  ActionMode: CHANGE_SET_EXECUTE
  Capabilities: CAPABILITY_NAMED_IAM,CAPABILITY_AUTO_EXPAND
  ChangeSetName: pipeline-changeset
  ParameterOverrides: '{"ProjectId": "my-project","CodeDeployRole":
 "CodeDeploy_Role_ARN"}'
  RoleArn: CloudFormation_Role_ARN
  StackName: my-project--lambda
  TemplateConfiguration: 'my-project--BuildArtifact::template-configuration.json'
  TemplatePath: 'my-project--BuildArtifact::template-export.yml'
OutputArtifacts: []
InputArtifacts:
  - Name: my-project-BuildArtifact
```

JSON

```
{
    "Name": "ExecuteChangeSet",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormation",
        "Version": "1"
    },
    "RunOrder": 2,
    "Configuration": {
        "ActionMode": "CHANGE_SET_EXECUTE",
        "Capabilities": "CAPABILITY_NAMED_IAM,CAPABILITY_AUTO_EXPAND",
        "ChangeSetName": "pipeline-changeset",
        "ParameterOverrides": "{\"ProjectId\": \"my-project\",\"CodeDeployRole\":
 \"CodeDeploy_Role_ARN\"}",
        "RoleArn": "CloudFormation_Role_ARN",
        "StackName": "my-project--lambda",
        "TemplateConfiguration": "my-project--BuildArtifact::template-
configuration.json",
        "TemplatePath": "my-project--BuildArtifact::template-export.yml"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
             "Name": "my-project-BuildArtifact"
        }
    ]
},
```

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 CloudFormation templates.

CloudFormation StackSets deploy action reference

CodePipeline offers the ability to perform CloudFormation StackSets operations as part of your CI/CD process. You use a stack set to create stacks in AWS accounts across AWS Regions by using a single CloudFormation template. All the resources included in each stack are defined by the stack set's CloudFormation template. When you create the stack set, you specify the template to use, as well as any parameters and capabilities that the template requires.

For more information about concepts for CloudFormation StackSets, see StackSets concepts in the *AWS CloudFormation User Guide*.

You integrate your pipeline with CloudFormation StackSets through two distinct action types that you use together:

- The `CloudFormationStackSet` action creates or updates a stack set or stack instances from the template stored in the pipeline source location. Each time a stack set is created or updated, it initiates a deployment of those changes to specified instances. In the console, you can choose the **CloudFormation Stack Set** action provider when you create or edit your pipeline.
- The `CloudFormationStackInstances` action deploys changes from the `CloudFormationStackSet` action to specified instances, creates new stack instances, and defines parameter overrides to specified instances. In the console, you can choose the **CloudFormation Stack Instances** action provider when you edit an existing pipeline.

You can use these actions to deploy to target AWS accounts or target AWS Organizations organizational unit IDs.

Note

To deploy to target AWS Organizations accounts or organizational unit IDs and use the service-managed permissions model, you must enable trusted access between AWS CloudFormation StackSets and AWS Organizations. For more information, see Enabling trusted access with AWS CloudFormation Stacksets.

Topics

- How CloudFormation StackSets actions work
- How to structure StackSets actions in a pipeline
- The CloudFormationStackSet action
- The CloudFormationStackInstances action
- Service role permissions: CloudFormationStackSet action
- Service role permissions: CloudFormationStackInstances action
- Permissions models for stack set operations
- Template parameter data types
- See also

How CloudFormation StackSets actions work

A `CloudFormationStackSet` action creates or updates resources depending on whether the action is running for the first time.

The `CloudFormationStackSet` action *creates* or *updates* the stack set and deploys those changes to specified instances.

Note

If you use this action to make an update that includes adding stack instances, the new instances are deployed first and the update is completed last. The new instances first receive the old version, and then the update is applied to all instances.

- *Create*: When no instances are specified and the stack set does not exist, the **CloudFormationStackSet** action creates the stack set without creating any instances.

- *Update*: When the **CloudFormationStackSet** action is run for a stack set that is already created, the action updates the stack set. If no instances are specified and the stack set already exists, all instances are updated. If this action is used to update specific instances, all remaining instances move to an OUTDATED status.

  You can use the **CloudFormationStackSet** action to update the stack set in the following ways.

  - Update the template on some or all instances.
  - Update parameters on some or all instances.
  - Update the execution role for the stack set (this must match the execution role specified in the Administrator role).
  - Change the permissions model (only if no instances have been created).
  - Enable/Disable `AutoDeployment` if the stack set permissions model is `Service Managed`.
  - Act as a delegated administrator in a member account if the stack set permissions model is `Service Managed`.
  - Update the Administrator role.
  - Update the description on the stack set.
  - Add deployment targets to the stack set update to create new stack instances.

The `CloudFormationStackInstances` action creates new stack instances or updates outdated stack instances. An instance becomes outdated when a stack set is updated, but not all instances within it are updated.

- *Create*: If the stack already exists, the `CloudFormationStackInstances` action only updates instances and does not create stack instances.
- *Update*: After the `CloudFormationStackSet` action is performed, if the template or parameters have been updated in only some instances, the rest will be marked `OUTDATED`. In later pipeline stages, `CloudFormationStackInstances` updates the rest of the instances in the stack set in waves so that all instances are marked `CURRENT`. This action can also be used to add additional instances or override parameters on new or existing instances.

As part of an update, the `CloudFormationStackSet` and `CloudFormationStackInstances` actions can specify new deployment targets, which creates new stack instances.

As part of an update, the `CloudFormationStackSet` and `CloudFormationStackInstances` actions do not delete stack sets, instances, or resources. When the action updates a stack but

does not specify all instances to be updated, the instances that were not specified for update are removed from the update and set to a status of `OUTDATED`.

During a deployment, stack instances can also show a status of `OUTDATED` if the deployment to instances failed.

How to structure StackSets actions in a pipeline

As a best practice, you should construct your pipeline so that the stack set is created and initially deploys to a subset or a single instance. After you test your deployment and view the generated stack set, then add the `CloudFormationStackInstances` action so that the remaining instances are created and updated.

Use the console or the CLI to create the recommended pipeline structure as follows:

1. Create a pipeline with a source action (required) and the `CloudFormationStackSet` action as the deploy action. Run your pipeline.
2. When your pipeline first runs, the `CloudFormationStackSet` action *creates* your stack set and at least one initial instance. Verify the stack set creation and review the deployment to your initial instance. For example, for initial stack set creation for account Account-A where `us-east-1` is the specified Region, the stack instance is created with the stack set:

Stack instance	Region	Status
StackInstanceID-1	us-east-1	CURRENT

3. Edit your pipeline to add `CloudFormationStackInstances` as the second deployment action to create/update stack instances for the targets you designate. For example, for stack instance creation for account `Account-A` where the `us-east-2` and `eu-central-1` Regions are specified, the remaining stack instances are created and the initial instance remains updated as follows:

Stack instance	Region	Status
StackInstanceID-1	us-east-1	CURRENT
StackInstanceID-2	us-east-2	CURRENT
StackInstanceID-3	eu-central-1	CURRENT

4. Run your pipeline as needed to update your stack set and update or create stack instances.

When you initiate a stack update where you have removed deployment targets from the action configuration, then the stack instances that were not designated for update are removed from the deployment and move into an OUTDATED status. For example, for stack instance update for account `Account-A` where the `us-east-2` Region is removed from the action configuration, the remaining stack instances are created and the removed instance is set to OUTDATED as follows:

Stack instance	Region	Status
StackInstanceID-1	us-east-1	CURRENT
StackInstanceID-2	us-east-2	OUTDATED
StackInstanceID-3	eu-central-1	CURRENT

For more information about best practices for deploying stack sets, see Best practices for StackSets in the *AWS CloudFormation User Guide*.

The `CloudFormationStackSet` action

This action creates or updates a stack set from the template stored in the pipeline source location.

After you define a stack set, you can create, update, or delete stacks in the target accounts and Regions specified in the configuration parameters. When creating, updating and deleting stacks, you can specify other preferences, such as the order of Regions for operations to be performed, the failure tolerance percentage beyond which stack operations stop, and the number of accounts in which operations are performed on stacks concurrently.

A stack set is a regional resource. If you create a stack set in one AWS Region, you cannot access it from other Regions.

When this action is used as an update action to the stack set, updates to the stack are not allowed without a deployment to at least one stack instance.

Topics

- Action type

- Configuration parameters
- Input artifacts
- Output artifacts
- Output variables
- Example CloudFormationStackSet action configuration

Action type

- Category: `Deploy`
- Owner: `AWS`
- Provider: `CloudFormationStackSet`
- Version: 1

Configuration parameters

StackSetName

Required: Yes

The name to associate with the stack set. This name must be unique in the Region where it is created.

The name may only contain alphanumeric and hyphen characters. It must begin with an alphabetic character and be 128 characters or fewer.

Description

Required: No

A description of the stack set. You can use this to describe the stack set's purpose or other relevant information.

TemplatePath

Required: Yes

The location of the template that defines the resources in the stack set. This must point to a template with a maximum size of 460,800 bytes.

Enter the path to the source artifact name and template file in the format `"InputArtifactName::TemplateFileName"`, as shown in the following example.

```
SourceArtifact::template.txt
```

Parameters

Required: No

A list of template parameters for your stack set that update during a deployment.

You can provide parameters as a literal list or a file path:

- You can enter parameters in the following shorthand syntax format: `ParameterKey=string,ParameterValue=string,UsePreviousValue=boolean,ResolvedV`
  `ParameterKey=string,ParameterValue=string,UsePreviousValue=boolean,ResolvedV`
  For more information about these data types, see Template parameter data types.

  The following example shows a parameter named `BucketName` with the value `amzn-s3-demo-source-bucket`.

  ```
  ParameterKey=BucketName,ParameterValue=amzn-s3-demo-source-bucket
  ```

  The following example shows an entry with multiple parameters:

  ```
  ParameterKey=BucketName,ParameterValue=amzn-s3-demo-source-bucket
  ParameterKey=Asset1,ParameterValue=true
  ParameterKey=Asset2,ParameterValue=true
  ```

- You can enter the location of the file containing a list of template parameter overrides entered in the format `"InputArtifactName::ParametersFileName"`, as shown in the following example.

  ```
  SourceArtifact::parameters.txt
  ```

  The following example shows the file contents for `parameters.txt`.

  ```
  [
  ```

```
    {
        "ParameterKey": "KeyName",
        "ParameterValue": "true"
    },
    {
        "ParameterKey": "KeyName",
        "ParameterValue": "true"
    }
]
```

Capabilities

Required: No

Indicates that the template can create and update resources, depending on the types of resources in the template.

You must use this property if you have IAM resources in your stack template or you create a stack directly from a template containing macros. For the CloudFormation action to successfully operate in this way, you must use one of the following capabilities:

- `CAPABILITY_IAM`
- `CAPABILITY_NAMED_IAM`

You can specify more than one capability by using a comma and no spaces between capabilities. The example in Example **CloudFormationStackSet** action configuration shows an entry with multiple capabilities.

PermissionModel

Required: No

Determines how IAM roles are created and managed. If the field is not specified, the default is used. For information, see Permissions models for stack set operations.

Valid values are:

- `SELF_MANAGED` (default): You must create administrator and execution roles to deploy to target accounts.
- `SERVICE_MANAGED`: CloudFormation StackSets automatically creates the IAM roles required to deploy to accounts managed by AWS Organizations. This requires an account to be a member of an Organization.

> **Note**
>
> This parameter can only be changed when no stack instances exist in the stack set.

AdministrationRoleArn

> **Note**
>
> Because CloudFormation StackSets performs operations across multiple accounts, you must define the necessary permissions in those accounts before you can create the stack set.

Required: No

> **Note**
>
> This parameter is optional for the SELF_MANAGED permissions model and is not used for the SERVICE_MANAGED permissions model.

The ARN of the IAM role in the administrator account used to perform stack set operations.

The name may contain alphanumeric characters, any of the following characters: _+=,.@-, and no spaces. The name is not case sensitive. This role name must be a minimum length of 20 characters and maximum length of 2048 characters. Role names must be unique within the account. The role name specified here must be an existing role name. If you do not specify the role name, it is set to AWSCloudFormationStackSetAdministrationRole. If you specify ServiceManaged, you must not define a role name.

ExecutionRoleName

> **Note**
>
> Because CloudFormation StackSets performs operations across multiple accounts, you must define the necessary permissions in those accounts before you can create the stack set.

Required: No

> **Note**
>
> This parameter is optional for the SELF_MANAGED permissions model and is not used for the SERVICE_MANAGED permissions model.

The name of the IAM role in the target accounts used to perform stack set operations. The name may contain alphanumeric characters, any of the following characters: _+=,.@-, and no spaces. The name is not case sensitive. This role name must be a minimum length of 1 character and maximum length of 64 characters. Role names must be unique within the account. The role name specified here must be an existing role name. Do not specify this role if you are using customized execution roles. If you do not specify the role name, it is set to `AWSCloudFormationStackSetExecutionRole`. If you set Service_Managed to true, you must not define a role name.

OrganizationsAutoDeployment

Required: No

> **Note**
>
> This parameter is optional for the SERVICE_MANAGED permissions model and is not used for the SELF_MANAGED permissions model.

Describes whether CloudFormation StackSets automatically deploys to AWS Organizations accounts that are added to a target organization or organizational unit (OU). If `OrganizationsAutoDeployment` is specified, do not specify `DeploymentTargets` and `Regions`.

> **Note**
>
> If no input is provided for `OrganizationsAutoDeployment`, then the default value is `Disabled`.

Valid values are:

- `Enabled`. Required: No.

  StackSets automatically deploys additional stack instances to AWS Organizations accounts that are added to a target organization or organizational unit (OU) in the specified Regions. If an account is removed from a target organization or OU, CloudFormation StackSets deletes stack instances from the account in the specified Regions.

- `Disabled`. Required: No.

  StackSets does not automatically deploy additional stack instances to AWS Organizations accounts that are added to a target organization or organizational unit (OU) in the specified Regions.

- `EnabledWithStackRetention`. Required: No.

  Stack resources are retained when an account is removed from a target organization or OU.

DeploymentTargets

Required: No

> **Note**
>
> For the SERVICE_MANAGED permissions model, you can provide either the organization root ID or organizational Unit IDs for deployment targets. For the SELF_MANAGED permissions model, you can only provide accounts.

> **Note**
>
> When this parameter is selected, you must also select **Regions**.

A list of AWS accounts or organizational unit IDs where stack set instances should be created/updated.

- **Accounts**:

  You can provide accounts as a literal list or a file path:

  - *Literal*: Enter parameters in the shorthand syntax format `account_ID,account_ID`, as shown in the following example.

```
111111222222,333333444444
```

- *File path:* The location of the file containing a list of AWS accounts where stack set instances should be created/updated, entered in the format `InputArtifactName::AccountsFileName`. If you use file path to specify either **accounts** or **OrganizationalUnitIds**, the file format must be in JSON, as shown in the following example.

  ```
  SourceArtifact::accounts.txt
  ```

  The following example shows the file contents for `accounts.txt`.

  ```
  [
      "111111222222"
  ]
  ```

  The following example shows the file contents for `accounts.txt` when listing more than one account:

  ```
  [
      "111111222222","333333444444"
  ]
  ```

- **OrganizationalUnitIds**:

  > **Note**
  >
  > This parameter is optional for the SERVICE_MANAGED permissions model and is not used for the SELF_MANAGED permissions model. Do not use this if you select **OrganizationsAutoDeployment**.

  The AWS organizational units in which to update associated stack instances.

  You can provide organizational unit IDs as a literal list or a file path:

  - *Literal*: Enter an array of strings separated by commas, as shown in the following example.

```
ou-examplerootid111-exampleouid111,ou-examplerootid222-exampleouid222
```

- *File path:* The location of the file containing a list of OrganizationalUnitIds in which to create or update stack set instances. If you use file path to specify either **accounts** or **OrganizationalUnitIds**, the file format must be in JSON, as shown in the following example.

  Enter a path to the file in the format `InputArtifactName::OrganizationalUnitIdsFileName`.

  ```
  SourceArtifact::OU-IDs.txt
  ```

  The following example shows the file contents for `OU-IDs.txt`:

  ```
  [
      "ou-examplerootid111-exampleouid111","ou-examplerootid222-exampleouid222"
  ]
  ```

Regions

Required: No

> **Note**
>
> When this parameter is selected, you must also select **DeploymentTargets**.

A list of AWS Regions where stack set instances are created or updated. Regions are updated in the order in which they are entered.

Enter a list of valid AWS Regions in the format `Region1,Region2`, as shown in the following example.

```
us-west-2,us-east-1
```

FailureTolerancePercentage

Required: No

The percentage of accounts per Region for which this stack operation can fail before CloudFormation stops the operation in that Region. If the operation is stopped in a Region, CloudFormation doesn't attempt the operation in subsequent Regions. When calculating the number of accounts based on the specified percentage, CloudFormation rounds *down* to the next whole number.

MaxConcurrentPercentage

Required: No

The maximum percentage of accounts in which to perform this operation at one time. When calculating the number of accounts based on the specified percentage, CloudFormation rounds *down* to the next whole number. If rounding down would result in zero, CloudFormation sets the number as one instead. Although you use this setting to specify the *maximum*, for large deployments the actual number of accounts acted upon concurrently may be lower due to service throttling.

RegionConcurrencyType

Required: No

You can specify if the stack set should deploy across AWS Regions sequentially or in parallel by configuring the Region concurrency deployment parameter. When the Region concurrency is specified to deploy stacks across multiple AWS Regions in parallel, this can result in faster overall deployment times.

- *Parallel*: Stack set deployments will be conducted at the same time, as long as a Region's deployment failures don't exceed a specified failure tolerance.
- *Sequential*: Stack set deployments will be conducted one at a time, as long as a Region's deployment failures don't exceed a specified failure tolerance. Sequential deployment is the default selection.

ConcurrencyMode

Required: No

The concurrency mode allows you to choose how the concurrency level behaves during stack set operations, whether with strict or soft failure tolerance. **Strict Failure Tolerance** lowers the deployment speed as stack set operation failures occur because concurrency decreases for each failure. **Soft Failure Tolerance** prioritizes deployment speed while still leveraging CloudFormation safety capabilities.

- STRICT_FAILURE_TOLERANCE: This option dynamically lowers the concurrency level to ensure the number of failed accounts never exceeds a particular failure tolerance. This is the default behavior.
- SOFT_FAILURE_TOLERANCE: This option decouples failure tolerance from the actual concurrency. This allows stack set operations to run at a set concurrency level, regardless of the number of failures.

CallAs

Required: No

> **Note**
>
> This parameter is optional for the SERVICE_MANAGED permissions model and is not used for the SELF_MANAGED permissions model.

Specifies whether you are acting in the organization's management account or as a delegated administrator in a member account.

> **Note**
>
> If this parameter is set to DELEGATED_ADMIN, make sure that the pipeline IAM role has `organizations:ListDelegatedAdministrators` permission. Otherwise, the action will fail while running with an error similar to the following: `Account used is not a delegated administrator`.

- SELF: Stack set deployment will use service-managed permissions while signed in to the management account.
- DELEGATED_ADMIN: Stack set deployment will use service-managed permissions while signed in to a delegated administrator account.

Input artifacts

You must include at least one input artifact that contains the template for the stack set in a `CloudFormationStackSet` action. You can include more input artifacts for lists of deployment targets, accounts, and parameters.

- **Number of artifacts:** `1 to 3`
- **Description:** You can include artifacts to provide:
  - The stack template file. (See the `TemplatePath` parameter.)
  - The parameters file. (See the `Parameters` parameter.)
  - The accounts file. (See the `DeploymentTargets` parameter.)

Output artifacts

- **Number of artifacts:** `0`
- **Description:** Output artifacts do not apply for this action type.

Output variables

If you configure this action, it 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.

- **StackSetId**: The ID of the stack set.
- **OperationId**: The ID of the stack set operation.

For more information, see Variables reference.

Example CloudFormationStackSet action configuration

The following examples show the action configuration for the **CloudFormationStackSet** action.

Example for the self-managed permissions model

The following example shows a **CloudFormationStackSet** action where the deployment target entered is an AWS account ID.

YAML

```
Name: CreateStackSet
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormationStackSet
```

```
  Version: '1'
RunOrder: 1
Configuration:
  DeploymentTargets: '111111222222'
  FailureTolerancePercentage: '20'
  MaxConcurrentPercentage: '25'
  PermissionModel: SELF_MANAGED
  Regions: us-east-1
  StackSetName: my-stackset
  TemplatePath: 'SourceArtifact::template.json'
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: us-west-2
Namespace: DeployVariables
```

JSON

```
{
    "Name": "CreateStackSet",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormationStackSet",
        "Version": "1"
    },
    "RunOrder": 1,
    "Configuration": {
        "DeploymentTargets": "111111222222",
        "FailureTolerancePercentage": "20",
        "MaxConcurrentPercentage": "25",
        "PermissionModel": "SELF_MANAGED",
        "Regions": "us-east-1",
        "StackSetName": "my-stackset",
        "TemplatePath": "SourceArtifact::template.json"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "us-west-2",
```

```
    "Namespace": "DeployVariables"
}
```

Example for the service-managed permissions model

The following example shows a **CloudFormationStackSet** action for the service-managed permissions model where the option for auto deployment to AWS Organizations is enabled with stack retention.

YAML

```
Name: Deploy
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormationStackSet
  Version: '1'
RunOrder: 1
Configuration:
  Capabilities: 'CAPABILITY_IAM,CAPABILITY_NAMED_IAM'
  OrganizationsAutoDeployment: EnabledWithStackRetention
  PermissionModel: SERVICE_MANAGED
  StackSetName: stacks-orgs
  TemplatePath: 'SourceArtifact::template.json'
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: eu-central-1
Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormationStackSet",
        "Version": "1"
    },
    "RunOrder": 1,
    "Configuration": {
```

```
        "Capabilities": "CAPABILITY_IAM,CAPABILITY_NAMED_IAM",
        "OrganizationsAutoDeployment": "EnabledWithStackRetention",
        "PermissionModel": "SERVICE_MANAGED",
        "StackSetName": "stacks-orgs",
        "TemplatePath": "SourceArtifact::template.json"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "eu-central-1",
    "Namespace": "DeployVariables"
}
```

The CloudFormationStackInstances action

This action creates new instances and deploys stack sets to specified instances. A stack instance is a reference to a stack in a target account within a Region. A stack instance can exist without a stack; for example, if the stack creation is not successful, the stack instance shows the reason for stack creation failure. A stack instance is associated with only one stack set.

After the initial creation of a stack set, you can add new stack instances by using `CloudFormationStackInstances`. Template parameter values can be overridden at the stack instance level during create or update stack set instance operations.

Each stack set has one template and set of template parameters. When you update the template or template parameters, you update them for the entire set. Then all instance statuses are set to `OUTDATED` until the changes are deployed to that instance.

To override parameter values on specific instances, for example, if the template contains a parameter for `stage` with a value of `prod`, you can override the value of that parameter to be `beta` or `gamma`.

Topics

- Action type
- Configuration parameters
- Input artifacts

- Output artifacts
- Output variables
- Example action configuration

Action type

- Category: `Deploy`
- Owner: `AWS`
- Provider: `CloudFormationStackInstances`
- Version: 1

Configuration parameters

StackSetName

Required: Yes

The name to associate with the stack set. This name must be unique in the Region where it is created.

The name may only contain alphanumeric and hyphen characters. It must begin with an alphabetic character and be 128 characters or fewer.

DeploymentTargets

Required: No

> **Note**
>
> For the SERVICE_MANAGED permissions model, you can provide either the organization root ID or organizational Unit IDs for deployment targets. For the SELF_MANAGED permissions model, you can only provide accounts.

> **Note**
>
> When this parameter is selected, you must also select **Regions**.

A list of AWS accounts or organizational unit IDs where stack set instances should be created/updated.

- **Accounts**:

  You can provide accounts as a literal list or a file path:

  - *Literal*: Enter parameters in the shorthand syntax format `account_ID,account_ID`, as shown in the following example.

    ```
    111111222222,333333444444
    ```

  - *File path:* The location of the file containing a list of AWS accounts where stack set instances should be created/updated, entered in the format `InputArtifactName::AccountsFileName`. If you use file path to specify either **accounts** or **OrganizationalUnitIds**, the file format must be in JSON, as shown in the following example.

    ```
    SourceArtifact::accounts.txt
    ```

    The following example shows the file contents for `accounts.txt`:

    ```
    [
        "111111222222"
    ]
    ```

    The following example shows the file contents for `accounts.txt` when listing more than one account:

    ```
    [
        "111111222222","333333444444"
    ]
    ```

- **OrganizationalUnitIds**:

  > **Note**
  >
  > This parameter is optional for the SERVICE_MANAGED permissions model and is not used for the SELF_MANAGED permissions model. Do not use this if you select **OrganizationsAutoDeployment**.

The AWS organizational units in which to update associated stack instances.

You can provide organizational unit IDs as a literal list or a file path.

- *Literal*: Enter an array of strings separated by commas, as shown in the following example.

```
ou-examplerootid111-exampleouid111,ou-examplerootid222-exampleouid222
```

- *File path:* The location of the file containing a list of OrganizationalUnitIds in which to create or update stack set instances. If you use file path to specify either **accounts** or **OrganizationalUnitIds**, the file format must be in JSON, as shown in the following example.

  Enter a path to the file in the format `InputArtifactName::OrganizationalUnitIdsFileName`.

```
SourceArtifact::OU-IDs.txt
```

  The following example shows the file contents for `OU-IDs.txt`:

```
[
    "ou-examplerootid111-exampleouid111","ou-examplerootid222-exampleouid222"
]
```

Regions

Required: Yes

> **Note**
>
> When this parameter is selected, you must also select **DeploymentTargets**.

A list of AWS Regions where stack set instances are created or updated. Regions are updated in the order in which they are entered.

Enter a list of valid AWS Regions in the format: `Region1,Region2`, as shown in the following example.

```
us-west-2,us-east-1
```

ParameterOverrides

Required: No

A list of stack set parameters that you want to override in the selected stack instances. Overridden parameter values are applied to all stack instances in the specified accounts and Regions.

You can provide parameters as a literal list or a file path:

- You can enter parameters in the following shorthand syntax format: `ParameterKey=string,ParameterValue=string,UsePreviousValue=boolean,ResolvedV ParameterKey=string,ParameterValue=string,UsePreviousValue=boolean,ResolvedV` For more information about these data types, see Template parameter data types.

  The following example shows a parameter named `BucketName` with the value `amzn-s3-demo-source-bucket`.

  ```
  ParameterKey=BucketName,ParameterValue=amzn-s3-demo-source-bucket
  ```

  The following example shows an entry with multiple parameters.

  ```
  ParameterKey=BucketName,ParameterValue=amzn-s3-demo-source-bucket
  ParameterKey=Asset1,ParameterValue=true
  ParameterKey=Asset2,ParameterValue=true
  ```

- You can enter the location of the file containing a list of template parameter overrides entered in the format `InputArtifactName::ParameterOverridesFileName`, as shown in the following example.

  ```
  SourceArtifact::parameter-overrides.txt
  ```

  The following example shows the file contents for `parameter-overrides.txt`.

  ```
  [
      {
  ```

```
        "ParameterKey": "KeyName",
        "ParameterValue": "true"
    },
    {
        "ParameterKey": "KeyName",
        "ParameterValue": "true"
    }
]
```

FailureTolerancePercentage

Required: No

The percentage of accounts per Region for which this stack operation can fail before CloudFormation stops the operation in that Region. If the operation is stopped in a Region, CloudFormation doesn't attempt the operation in subsequent Regions. When calculating the number of accounts based on the specified percentage, CloudFormation rounds *down* to the next whole number.

MaxConcurrentPercentage

Required: No

The maximum percentage of accounts on which to perform this operation at one time. When calculating the number of accounts based on the specified percentage, CloudFormation rounds *down* to the next whole number. If rounding down would result in zero, CloudFormation sets the number as one instead. Although you specify the *maximum*, for large deployments the actual number of accounts acted upon concurrently may be lower due to service throttling.

RegionConcurrencyType

Required: No

You can specify if the stack set should deploy across AWS Regions sequentially or in parallel by configuring the Region concurrency deployment parameter. When the Region concurrency is specified to deploy stacks across multiple AWS Regions in parallel, this can result in faster overall deployment times.

- *Parallel*: Stack set deployments will be conducted at the same time, as long as a Region's deployment failures don't exceed a specified failure tolerance.
- *Sequential*: Stack set deployments will be conducted one at a time, as long as a Region's deployment failures don't exceed a specified failure tolerance. Sequential deployment is the default selection.

ConcurrencyMode

Required: No

The concurrency mode allows you to choose how the concurrency level behaves during stack set operations, whether with strict or soft failure tolerance. **Strict Failure Tolerance** lowers the deployment speed as stack set operation failures occur because concurrency decreases for each failure. **Soft Failure Tolerance** prioritizes deployment speed while still leveraging CloudFormation safety capabilities.

- `STRICT_FAILURE_TOLERANCE`: This option dynamically lowers the concurrency level to ensure the number of failed accounts never exceeds a particular failure tolerance. This is the default behavior.
- `SOFT_FAILURE_TOLERANCE`: This option decouples failure tolerance from the actual concurrency. This allows stack set operations to run at a set concurrency level, regardless of the number of failures.

CallAs

Required: No

> **Note**
>
> This parameter is optional for the `SERVICE_MANAGED` permissions model and is not used for the `SELF_MANAGED` permissions model.

Specifies whether you are acting in the organization's management account or as a delegated administrator in a member account.

> **Note**
>
> If this parameter is set to `DELEGATED_ADMIN`, make sure that the pipeline IAM role has `organizations:ListDelegatedAdministrators` permission. Otherwise, the action will fail while running with an error similar to the following: `Account used is not a delegated administrator`.

- `SELF`: Stack set deployment will use service-managed permissions while signed in to the management account.

- `DELEGATED_ADMIN`: Stack set deployment will use service-managed permissions while signed in to a delegated administrator account.

Input artifacts

`CloudFormationStackInstances` can contain artifacts that list deployment targets and parameters.

- **Number of artifacts:** `0 to 2`
- **Description:** As input, the stack set action optionally accepts artifacts for these purposes:
  - To provide the parameters file to use. (See the `ParameterOverrides` parameter.)
  - To provide the target accounts file to use. (See the `DeploymentTargets` parameter.)

Output artifacts

- **Number of artifacts:** `0`
- **Description:** Output artifacts do not apply for this action type.

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.

- **StackSetId**: The ID of the stack set.
- **OperationId**: The ID of the stack set operation.

For more information, see Variables reference.

Example action configuration

The following examples show the action configuration for the **CloudFormationStackInstances** action.

Example for the self-managed permissions model

The following example shows a **CloudFormationStackInstances** action where the deployment target entered is an AWS account ID 111111222222.

YAML

```
Name: my-instances
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormationStackInstances
  Version: '1'
RunOrder: 2
Configuration:
  DeploymentTargets: '111111222222'
  Regions: 'us-east-1,us-east-2,us-west-1,us-west-2'
  StackSetName: my-stackset
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: us-west-2
```

JSON

```
{
    "Name": "my-instances",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormationStackInstances",
        "Version": "1"
    },
    "RunOrder": 2,
    "Configuration": {
        "DeploymentTargets": "111111222222",
        "Regions": "us-east-1,us-east-2,us-west-1,us-west-2",
        "StackSetName": "my-stackset"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
```

```
        }
    ],
    "Region": "us-west-2"
}
```

Example for the service-managed permissions model

The following example shows a **CloudFormationStackInstances** action for the service-managed permissions model where the deployment target is an AWS Organizations organizational unit ID `ou-1111-1example`.

YAML

```
Name: Instances
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormationStackInstances
  Version: '1'
RunOrder: 2
Configuration:
  DeploymentTargets: ou-1111-1example
  Regions: us-east-1
  StackSetName: my-stackset
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: eu-central-1
```

JSON

```
{
    "Name": "Instances",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormationStackInstances",
        "Version": "1"
    },
    "RunOrder": 2,
    "Configuration": {
        "DeploymentTargets": "ou-1111-1example",
```

```
        "Regions": "us-east-1",
        "StackSetName": "my-stackset"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "eu-central-1"
}
```

Service role permissions: `CloudFormationStackSet` action

For CloudFormation StackSets actions, the following minimum permissions are required.

For the `CloudFormationStackSet` action, add the following to your policy statement:

```
{
    "Effect": "Allow",
    "Action": [
        "cloudformation:CreateStackSet",
        "cloudformation:UpdateStackSet",
        "cloudformation:CreateStackInstances",
        "cloudformation:DescribeStackSetOperation",
        "cloudformation:DescribeStackSet",
        "cloudformation:ListStackInstances"
    ],
    "Resource": "resource_ARN"
},
```

Service role permissions: `CloudFormationStackInstances` action

For the `CloudFormationStackInstances` action, add the following to your policy statement:

```
{
    "Effect": "Allow",
    "Action": [
        "cloudformation:CreateStackInstances",
        "cloudformation:DescribeStackSetOperation"
    ],
    "Resource": "resource_ARN"
```

```
},
```

Permissions models for stack set operations

Because CloudFormation StackSets performs operations across multiple accounts, you must define the necessary permissions in those accounts before you can create the stack set. You can define permissions through self-managed permissions or service-managed permissions.

With self-managed permissions, you create the two IAM roles required by StackSets - an administrator role such as the AWSCloudFormationStackSetAdministrationRole in the account where you define the stack set and an execution role such as the AWSCloudFormationStackSetExecutionRole in each of the accounts where you deploy stack set instances. Using this permissions model, StackSets can deploy to any AWS account in which the user has permissions to create an IAM role. For more information, see Grant self-managed permissions in the *AWS CloudFormation User Guide*.

> **Note**
>
> Because CloudFormation StackSets performs operations across multiple accounts, you must define the necessary permissions in those accounts before you can create the stack set.

With service-managed permissions, you can deploy stack instances to accounts managed by AWS Organizations. Using this permissions model, you don't have to create the necessary IAM roles because StackSets creates the IAM roles on your behalf. With this model, you can also enable automatic deployments to accounts that are added to the organization in the future. See Enable trusted access with AWS Organizations in the *AWS CloudFormation User Guide*.

Template parameter data types

The template parameters used in stack set operations include the following data types. For more information, see DescribeStackSet.

ParameterKey

- Description: The key associated with the parameter. If you don't specify a key and value for a particular parameter, AWS CloudFormation uses the default value that is specified in the template.
- Example:

```
"ParameterKey=BucketName,ParameterValue=amzn-s3-demo-source-bucket"
```

ParameterValue

- Description: The input value associated with the parameter.
- Example:

```
"ParameterKey=BucketName,ParameterValue=amzn-s3-demo-source-bucket"
```

UsePreviousValue

- During a stack update, use the existing parameter value that the stack is using for a given parameter key. If you specify `true`, do not specify a parameter value.
- Example:

```
"ParameterKey=Asset1,UsePreviousValue=true"
```

Each stack set has one template and set of template parameters. When you update the template or template parameters, you update them for the entire set. Then all instance statuses are set to OUTDATED until the changes are deployed to that instance.

To override parameter values on specific instances, for example, if the template contains a parameter for `stage` with a value of `prod`, you can override the value of that parameter to be `beta` or `gamma`.

See also

The following related resources can help you as you work with this action.

- Parameter types – This reference chapter in the *AWS CloudFormation User Guide* provides more descriptions and examples for CloudFormation template parameters.
- Best practices – For more information about best practices for deploying stack sets, see https://docs.aws.eu/AWSCloudFormation/latest/UserGuide/stacksets-bestpractices.html in the *AWS CloudFormation User Guide*.
- AWS CloudFormation API Reference – You can reference the following CloudFormation actions in the *AWS CloudFormation API Reference* for more information about the parameters used in stack set operations:

- The CreateStackSet action creates a stack set.
- The UpdateStackSet action updates the stack set and associated stack instances in the specified accounts and Regions. Even if the stack set operation created by updating the stack set fails (completely or partially, below or above a specified failure tolerance), the stack set is updated with these changes. Subsequent CreateStackInstances calls on the specified stack set use the updated stack set.
- The CreateStackInstances action creates a stack instance for all specified regions within all specified accounts on a self-managed permission model, or within all specified deployment targets on a service-managed permission model. You can override parameters for the instances created by this action. If the instances already exist, CreateStackInstances calls UpdateStackInstances with the same input parameters. When you use this action to create instances, it does not change the status of other stack instances.
- The UpdateStackInstances action brings stack instances up to date with the stack set for all specified regions within all specified accounts on a self-managed permission model, or within all specified deployment targets on a service-managed permission model. You can override parameters for the instances updated by this action. When you use this action to update a subset of instances, it does not change the status of other stack instances.
- The DescribeStackSetOperation action returns the description of the specified stack set operation.
- The DescribeStackSet action returns the description of the specified stack set.

AWS CodeBuild build and test action reference

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 buildspec 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
- Configuration parameters
- Input artifacts
- Output artifacts
- Output variables
- Service role permissions: CodeBuild action
- Action declaration (CodeBuild example)
- See also

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 buildspec file and runs the buildspec 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.

BuildspecOverride

Required: No

An inline buildspec definition or buildspec file declaration that overrides the latest one defined in the build project, for this build only. The buildspec defined on the project is not changed.

If this value is set, it can be one of the following:

- An inline buildspec definition. For more information, see the syntax reference at Buildspec syntax.
- The path to an alternate buildspec file relative to the value of the built-in `CODEBUILD_SRC_DIR` environment variable or the path to an S3 bucket. The bucket must be in the same AWS Region as the build project. Specify the buildspec file using its ARN (for example, `arn:aws:s3:::my-codebuild-sample2/buildspec.yml`). If this value is not provided or is set to an empty string, the source code must contain a buildspec file in its root directory. For more information about adding a path, see Buildspec File Name and Storage Location.

> **Note**
>
> Since this property allows you to change the build commands that will run in the container, you should note that an IAM principal with the ability to call this API and set this parameter can override the default settings. Moreover, we encourage that you use a trustworthy buildspec location like a file in your source repository or a Amazon S3 bucket.

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).

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 credentials. 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 in the AWS CodeBuild API Reference. For an example CodeBuild action with an environment variable that resolves to the GitHub branch name, see Example: Use a BranchName variable with CodeBuild environment variables.

ServiceRoleArnOverride

Required: No

An IAM role ARN that CodeBuild uses as the service role for builds triggered by this action, overriding the default service role configured on the CodeBuild project. When absent, the CodeBuild project uses its default service role.

The specified role must have a trust policy that allows the `codebuild.amazonaws.com` service to assume it. Both the identity used to create or update the pipeline and the pipeline service role must have `iam:PassRole` permission for the specified role. For an example policy, see Service role permissions: CodeBuild action.

As a security best practice, when a CodeBuild project is shared across multiple pipelines, we recommend specifying a scoped role for each pipeline to follow the principle of least privilege. For pipelines using CodeConnections source actions, see Add CodeBuild GitClone permissions for connections to Bitbucket, GitHub, GitHub Enterprise Server, or GitLab.com for an example of scoping the role's connection permissions to a specific repository.

> **Note**
>
> The role must be in the same AWS account as the CodeBuild project.

Input artifacts

- **Number of artifacts:** `1 to 5`
- **Description:** CodeBuild looks for the buildspec file and runs the buildspec commands from the directory of the primary source artifact. When either a single input source is specified or when more than one input source is specified for the CodeBuild action, the single artifact, or the

primary artifact in the case of multiple input sources, 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 buildspec file available to subsequent actions in the pipeline. When only one output artifact is defined, this artifact can be defined directly under the `artifacts` section of the buildspec file. When more than one output artifact is specified, all artifacts referenced must be defined as secondary artifacts in the buildspec file. The names of the output artifacts in CodePipeline must match the artifact identifiers in the buildspec file.

> **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.

Output variables

This action will produce as variables all environment variables that were exported as part of the build. For more details on how to export environment variables, see EnvironmentVariable in the *AWS CodeBuild API Guide*.

For more information about using CodeBuild environment variables in CodePipeline, see the examples in CodeBuild action output variables. For a list of the environment variables you can use in CodeBuild, see Environment variables in build environments in the *AWS CodeBuild User Guide*.

Service role permissions: CodeBuild action

For CodeBuild support, add the following to your policy statement:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": [
                "codebuild:BatchGetBuilds",
                "codebuild:StartBuild",
                "codebuild:BatchGetBuildBatches",
                "codebuild:StartBuildBatch"
            ],
            "Resource": [
                "arn:aws-eusc:codebuild:*:111122223333:project/[[ProjectName]]"
            ],
            "Effect": "Allow"
        }
    ]
}
```

If you use the `ServiceRoleArnOverride` parameter, add the following `iam:PassRole` permission to allow CodePipeline to pass the role to CodeBuild:

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

```
    "Action": "iam:PassRole",
    "Resource": "arn:aws:iam::111122223333:role/your-codebuild-override-role",
    "Condition": {
        "StringEquals": {
            "iam:PassedToService": "codebuild.amazonaws.com"
        }
    }
}
```

Action declaration (CodeBuild example)

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"}]'
      ServiceRoleArnOverride: 'arn:aws:iam::111122223333:role/my-scoped-codebuild-
role'
    OutputArtifacts:
      - Name: MyPipeline-BuildArtifact
    InputArtifacts:
      - Name: MyApplicationSource1
      - Name: MyApplicationSource2
```

JSON

```
{
```

```
    "Name": "Build",
    "Actions": [
        {
            "Name": "PackageExport",
            "ActionTypeId": {
                "Category": "Build",
                "Owner": "AWS",
                "Provider": "CodeBuild",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "BatchEnabled": "true",
                "CombineArtifacts": "true",
                "ProjectName": "my-build-project",
                "PrimarySource": "MyApplicationSource1",
                "EnvironmentVariables": "[{\"name\":\"TEST_VARIABLE\",\"value\":
\"TEST_VALUE\",\"type\":\"PLAINTEXT\"},{\"name\":\"ParamStoreTest\",\"value\":
\"PARAMETER_NAME\",\"type\":\"PARAMETER_STORE\"}]",
                "ServiceRoleArnOverride": "arn:aws:iam::111122223333:role/my-scoped-
codebuild-role"
            },
            "OutputArtifacts": [
                {
                    "Name": "MyPipeline-BuildArtifact"
                }
            ],
            "InputArtifacts": [
                {
                    "Name": "MyApplicationSource1"
                },
                {
                    "Name": "MyApplicationSource2"
                }
            ]
        }
    ]
}
```

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 with AWS Device Farm – This tutorial provides a sample buildspec 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 buildspec files. For a list of the environment variables you can use in CodeBuild, see Environment variables in build environments in the *AWS CodeBuild User Guide*.

AWS CodePipeline invoke action reference

You use a CodePipeline invoke action to simplify triggering downstream pipeline executions and passing pipeline variables and source revisions between pipelines.

> **Note**
>
> This action is only supported for V2 type pipelines.

Topics

- Action type
- Configuration parameters
- Input artifacts
- Output artifacts
- Service role policy permissions for the CodePipeline invoke action
- Action declaration
- See also

Action type

- Category: `Invoke`

- Owner: AWS
- Provider: CodePipeline
- Version: 1

Configuration parameters

PipelineName

Required: Yes

The name of the pipeline that will, upon running, start the current target pipeline. You must have already created the invoking pipeline. The action will start the s3-pipeline-test (target) pipeline when the (invoking) pipeline named my-s3-pipeline starts an execution.

SourceRevisions

Required: No

The source revisions that you want the target pipeline to use when it is started by the invoking pipeline. For example, an S3 source action provides output variables such as the S3 Version ID and Object Key. You can specify a revision value to be used when the pipeline is invoked.

For the CLI, you specify source revisions as a serialized JSON string. For more information about using source revision overrides, see SourceRevisionOverride in the *CodePipeline API Guide*.

The mapping uses a string format as shown in the following example:

```
[{"actionName":"Source","revisionType":"S3_OBJECT_VERSION_ID","revision
Value":"zq8mjNEXAMPLE"}]
```

Variables

Required: No

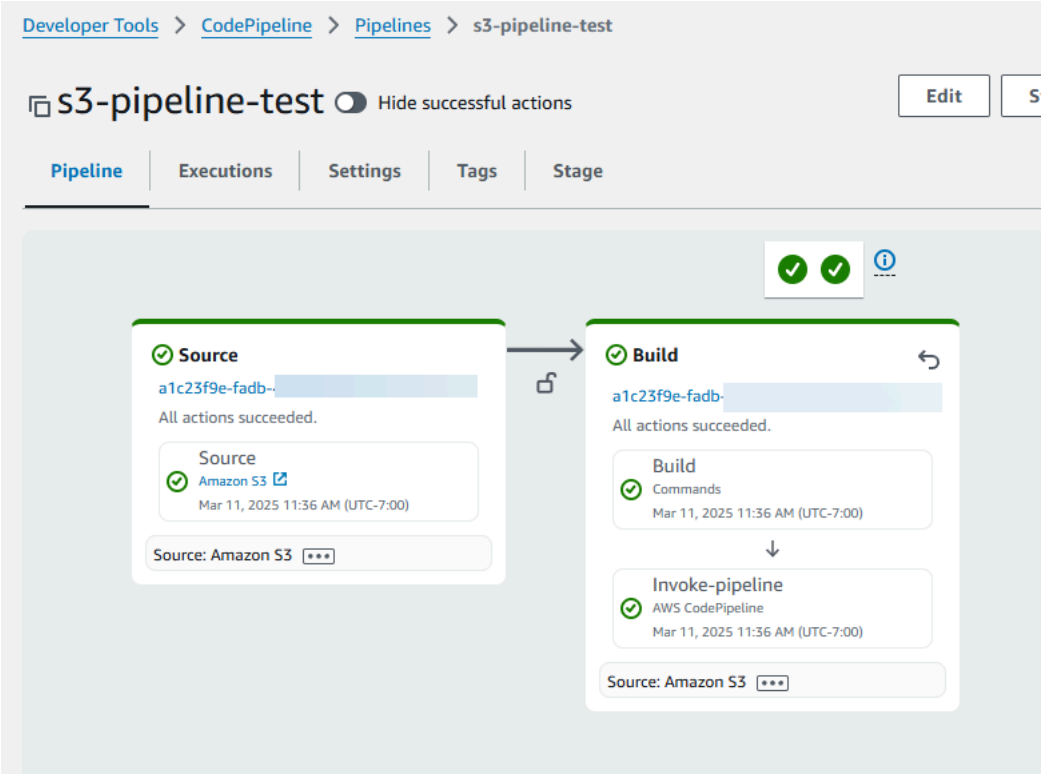
The names and values of variables that you want the action to support.

For the CLI, you specify variables as a serialized JSON string. For more information about using pipeline variables, see PipelineVariable in the *CodePipeline API Guide*.

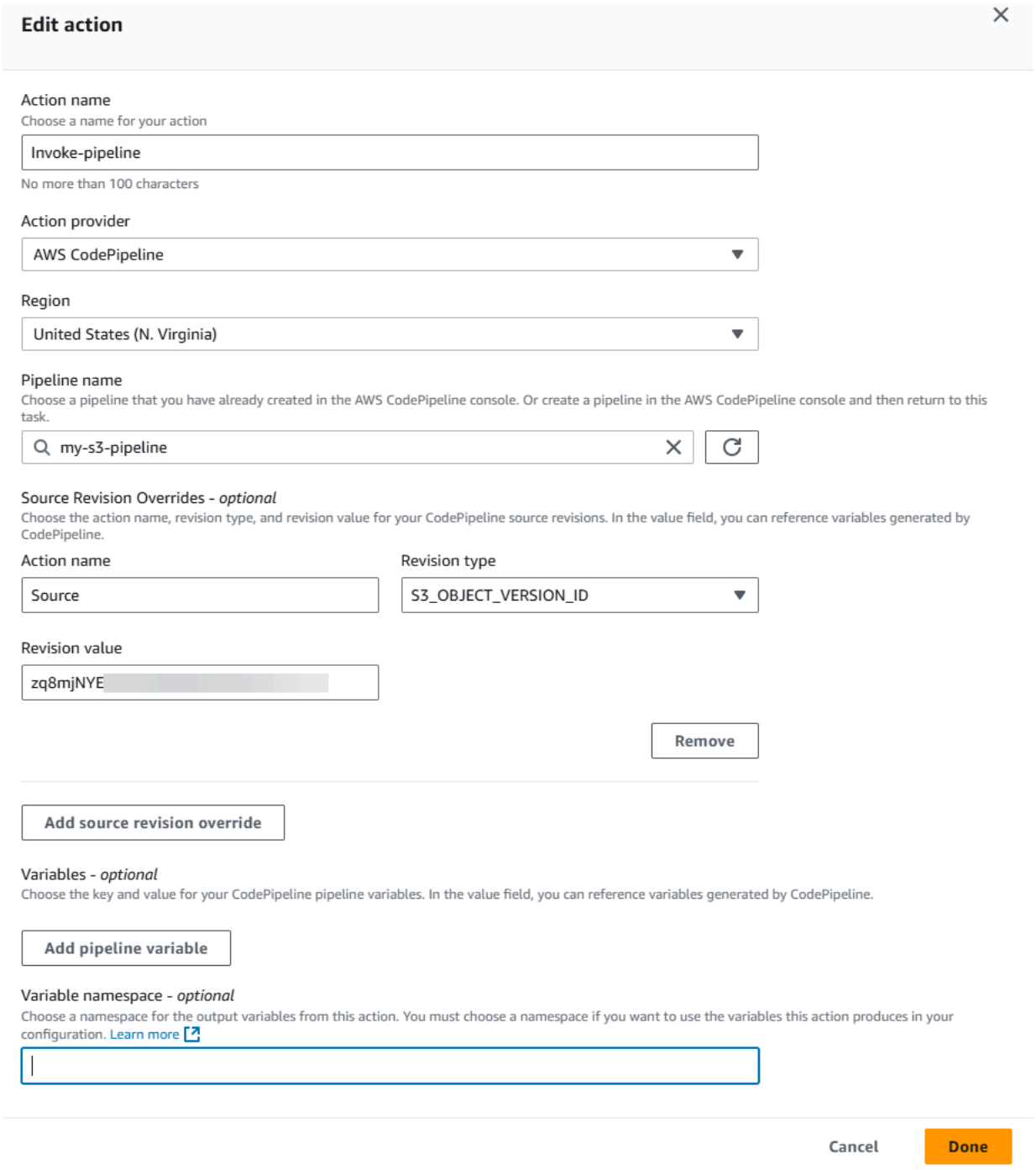
The mapping uses a string format as shown in the following example:

```
[{"name":"VAR1","value":"VALUE1"}]
```

The following image shows an example of the action added to a pipeline in the console.



The following image shows an example of the **Edit** page for the action. In the following example, the pipeline named `s3-pipeline-test` has a pipeline invoke action configured as shown for the console. The action will start the `s3-pipeline-test` pipeline when the pipeline named `my-s3-pipeline` completes an execution. The example shows that source revision override for the S3_OBJECT_VERSION_ID source override with specified revision value of `zq8mjNYEexample`.



Input artifacts

- **Number of artifacts:** `0`
- **Description:** Input artifacts do not apply for this action type.

Output artifacts

- **Number of artifacts:** `0`
- **Description:** Output artifacts do not apply for this action type.

Service role policy permissions for the CodePipeline invoke action

When CodePipeline runs the action, the CodePipeline service role policy requires the `codepipeline:StartPipelineExecution` permission, appropriately scoped down to the pipeline resource ARN in order to maintain access with least privilege.

```
{
           "Sid": "StatementForPipelineInvokeAction",
           "Effect": "Allow",
           "Action": "codepipeline:StartPipelineExecution",
           "Resource": [
               "arn:aws:codepipeline:{{region}}:{{AccountId}}:{{pipelineName}}"
           ]
       }
```

Action declaration

YAML

```
name: Invoke-pipeline
actionTypeId:
  category: Invoke
  owner: AWS
  provider: CodePipeline
  version: '1'
runOrder: 2
configuration:
  PipelineName: my-s3-pipeline
  SourceRevisions:
 '[{"actionName":"Source","revisionType":"S3_OBJECT_VERSION_ID","revision
Value":"zq8mjNEXAMPLE"}]'
  Variables: '[{"name":"VAR1","value":"VALUE1"}]'
```

JSON

```
{
    "name": "Invoke-pipeline",
    "actionTypeId": {
        "category": "Invoke",
        "owner": "AWS",
        "provider": "CodePipeline",
        "version": "1"
    },
    "runOrder": 2,
    "configuration": {
        "PipelineName": "my-s3-pipeline",
        "SourceRevisions": "[{\"actionName\":\"Source\",\"revisionType\":
\"S3_OBJECT_VERSION_ID\",\"revisionValue\":\"zq8mjNEXAMPLE\"}]",
        "Variables": "[{\"name\":\"VAR1\",\"value\":\"VALUE1\"}]"
    }
},
```

See also

The following related resources can help you as you work with this action.

- Start a pipeline with a source revision override – This section describes starting a pipeline with source revisions manually or through the EventBridge event input transformer.

CodeCommit source action reference

Starts 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 an EventBridge rule that starts your pipeline when a change occurs in the repository.

> **Note**
>
> For Amazon ECR, Amazon S3, or CodeCommit sources, you can also create a source override using input transform entry to use the `revisionValue` in EventBridge for your pipeline event, where the `revisionValue` is derived from the source event variable for your object key, commit, or image ID. For more information, see the optional step for input transform

entry included in the procedures under Amazon ECR source actions and EventBridge resources, Connecting to Amazon S3 source actions with a source enabled for events, or CodeCommit source actions and EventBridge.

You must have already created a CodeCommit repository before you connect the pipeline through a CodeCommit action.

After a code change is detected, you have the following options for passing the code to subsequent actions:

- **Default** – Configures the CodeCommit source action to output a ZIP file with a shallow copy of your commit.
- **Full clone** – Configures the source action to output a Git URL reference to the repository 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.

Topics

- Action type
- Configuration parameters
- Input artifacts
- Output artifacts
- Output variables
- Service role permissions: CodeCommit action
- Example action configuration
- See also

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 Migrate polling pipelines (CodeCommit source) (CLI) or Migrate polling pipelines (CodeCommit source) (CloudFormation template).

> **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.

OutputArtifactFormat

Required: No

The output artifact format. Values 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.
> If you choose this option, you need to add the `codecommit:GitPull` permission to your CodeBuild service role as shown in Add CodeBuild GitClone permissions for CodeCommit source actions. You also need to add the `codecommit:GetRepository` permission to your CodePipeline service role as shown in Add permissions to the CodePipeline service role. For a tutorial that shows you how to use the **Full clone** option, see Tutorial: Use full clone with a CodeCommit pipeline source.

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 reference.

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.

CommitterDate

The date when the commit was committed, in timestamp format.

Service role permissions: CodeCommit action

When CodePipeline runs the action, the CodePipeline service role policy requires the following permissions, appropriately scoped down to the pipeline resource ARN in order to maintain access with least privilege. For example, add the following to your policy statement:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codecommit:CancelUploadArchive",
                "codecommit:GetBranch",
                "codecommit:GetCommit",
                "codecommit:GetRepository",
                "codecommit:GetUploadArchiveStatus",
                "codecommit:UploadArchive"
            ],
            "Resource": [
                "arn:aws-eusc:codecommit:*:111122223333:
[[codecommitRepostories]]"
            ]
        }
    ]
}
```

Example action configuration

Example for default output artifact format

YAML

```
name: Source
actionTypeId:
  category: Source
  owner: AWS
  provider: CodeCommit
  version: '1'
runOrder: 1
configuration:
  BranchName: main
  PollForSourceChanges: 'false'
  RepositoryName: MyWebsite
```

```
outputArtifacts:
  - name: Artifact_MyWebsiteStack
inputArtifacts: []
region: us-west-2
namespace: SourceVariables
```

JSON

```
{
    "name": "Source",
    "actionTypeId": {
        "category": "Source",
        "owner": "AWS",
        "provider": "CodeCommit",
        "version": "1"
    },
    "runOrder": 1,
    "configuration": {
        "BranchName": "main",
        "PollForSourceChanges": "false",
        "RepositoryName": "MyWebsite"
    },
    "outputArtifacts": [
        {
            "name": "Artifact_MyWebsiteStack"
        }
    ],
    "inputArtifacts": [],
    "region": "us-west-2",
    "namespace": "SourceVariables"
}
```

Example for full clone output artifact format

YAML

```
name: Source
actionTypeId:
  category: Source
  owner: AWS
  provider: CodeCommit
  version: '1'
```

```
runOrder: 1
configuration:
  BranchName: main
  OutputArtifactFormat: CODEBUILD_CLONE_REF
  PollForSourceChanges: 'false'
  RepositoryName: MyWebsite
outputArtifacts:
  - name: SourceArtifact
inputArtifacts: []
region: us-west-2
namespace: SourceVariables
```

JSON

```
{
    "name": "Source",
    "actionTypeId": {
        "category": "Source",
        "owner": "AWS",
        "provider": "CodeCommit",
        "version": "1"
    },
    "runOrder": 1,
    "configuration": {
        "BranchName": "main",
        "OutputArtifactFormat": "CODEBUILD_CLONE_REF",
        "PollForSourceChanges": "false",
        "RepositoryName": "MyWebsite"
    },
    "outputArtifacts": [
        {
            "name": "SourceArtifact"
        }
    ],
    "inputArtifacts": [],
    "region": "us-west-2",
    "namespace": "SourceVariables"
}
```

See also

The following related resources can help you as you work with this action.

- Tutorial: Create a simple pipeline (CodeCommit repository) – This tutorial provides a sample AppSpec 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.

AWS CodeDeploy deploy action reference

You use an AWS CodeDeploy action to deploy application code to your deployment fleet. Your deployment fleet can consist of Amazon EC2 instances, on-premises instances, or both.

> **Note**
>
> This reference topic describes the CodeDeploy deployment action for CodePipeline where the deployment platform is Amazon EC2. For reference information about Amazon Elastic Container Service to CodeDeploy blue/green deployment actions in CodePipeline, see Amazon Elastic Container Service and CodeDeploy blue-green deploy action reference.

Topics

- Action type
- Configuration parameters
- Input artifacts
- Output artifacts
- Service role permissions: AWS CodeDeploy action
- Action declaration
- See also

Action type

- Category: `Deploy`
- Owner: `AWS`
- Provider: `CodeDeploy`
- Version: 1

Configuration parameters

ApplicationName

Required: Yes

The name of the application that you created in CodeDeploy.

DeploymentGroupName

Required: Yes

The deployment group that you created in CodeDeploy.

Input artifacts

- **Number of artifacts:** 1
- **Description:** The AppSpec file that CodeDeploy uses to determine:
  - What to install onto your instances from your application revision in Amazon S3 or GitHub.
  - Which lifecycle event hooks to run in response to deployment lifecycle events.

  For more information about the AppSpec file, see the CodeDeploy AppSpec File Reference.

Output artifacts

- **Number of artifacts:** `0`
- **Description:** Output artifacts do not apply for this action type.

Service role permissions: AWS CodeDeploy action

For AWS CodeDeploy support, add the following to your policy statement:

JSON

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

```
            "Effect": "Allow",
            "Action": [
                "codedeploy:CreateDeployment",
                "codedeploy:GetApplication",
                "codedeploy:GetDeployment",
                "codedeploy:RegisterApplicationRevision",
                "codedeploy:ListDeployments",
                "codedeploy:ListDeploymentGroups",
                "codedeploy:GetDeploymentGroup"
            ],
            "Resource": [
                "arn:aws-eusc:codedeploy:*:111122223333:application:
[[codedeployApplications]]",
                "arn:aws-eusc:codedeploy:*:111122223333:deploymentgroup:
[[codedeployApplications]]/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "codedeploy:GetDeploymentConfig"
            ],
            "Resource": [
                "arn:aws-eusc:codedeploy:*:111122223333:deploymentconfig:
[[deploymentConfigs]]"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "codedeploy:ListDeploymentConfigs"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Action declaration

YAML

```
Name: Deploy
Actions:
  - Name: Deploy
    ActionTypeId:
      Category: Deploy
      Owner: AWS
      Provider: CodeDeploy
      Version: '1'
    RunOrder: 1
    Configuration:
      ApplicationName: my-application
      DeploymentGroupName: my-deployment-group
    OutputArtifacts: []
    InputArtifacts:
      - Name: SourceArtifact
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
        {
            "Name": "Deploy",
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "CodeDeploy",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "ApplicationName": "my-application",
                "DeploymentGroupName": "my-deployment-group"
            },
            "OutputArtifacts": [],
            "InputArtifacts": [
                {
```

```
                    "Name": "SourceArtifact"
                }
            ],
            "Region": "us-west-2",
            "Namespace": "DeployVariables"
        }
    ]
},
```

See also

The following related resources can help you as you work with this action.

- Tutorial: Create a simple pipeline (S3 bucket) – This tutorial walks you through the creation of a source bucket, EC2 instances, and CodeDeploy resources to deploy a sample application. You then build your pipeline with a CodeDeploy deployment action that deploys code maintained in your S3 bucket to your Amazon EC2 instance.
- Tutorial: Create a simple pipeline (CodeCommit repository) – This tutorial walks you through the creation of your CodeCommit source repository, EC2 instances, and CodeDeploy resources to deploy a sample application. You then build your pipeline with a CodeDeploy deployment action that deploys code from your CodeCommit repository to your Amazon EC2 instance.
- CodeDeploy AppSpec File Reference – This reference chapter in the *AWS CodeDeploy User Guide* provides reference information and examples for CodeDeploy AppSpec files.

CodeStarSourceConnection for Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed actions

Source actions for connections are supported by AWS CodeConnections. AWS CodeConnections allows you to create and manage connections between AWS resources and third-party repositories such as GitHub. Starts 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 run or when a webhook event is sent from the source provider.

You can configure actions in your pipeline to use a Git configuration that allows you to start your pipeline with triggers. To configure the pipeline trigger configuration to filter with triggers, see more details in Add trigger with code push or pull request event types.

Connections can associate your AWS resources with the following third-party repositories:

- Bitbucket Cloud (through the **Bitbucket** provider option in the CodePipeline console or the `Bitbucket` provider in the CLI)

  > **Note**
  >
  > You can create connections to a Bitbucket Cloud repository. Installed Bitbucket provider types, such as Bitbucket Server, are not supported.

-

  > **Note**
  >
  > If you are using a Bitbucket workspace, you must have administrator access to create the connection.

- GitHub and GitHub Enterprise Cloud (through the **GitHub (via GitHub App)** provider option in the CodePipeline console or the `GitHub` provider in the CLI)

  > **Note**
  >
  > If your repository is in a GitHub organization, you must be the organization owner to create the connection. If you are using a repository that is not in an organization, you must be the repository owner.

- GitHub Enterprise Server (through the **GitHub Enterprise Server** provider option in the CodePipeline console or the `GitHub Enterprise Server` provider in the CLI)
- GitLab.com (through the **GitLab** provider option in the CodePipeline console or the `GitLab` provider in the CLI)

  > **Note**
  >
  > You can create connections to a repository where you have the **Owner** role in GitLab, and then the connection can be used with the repository with resources such as CodePipeline. For repositories in groups, you do not need to be the group owner.

- Self-managed installation for GitLab (Enterprise Edition or Community Edition) (through the **GitLab self-managed** provider option in the CodePipeline console or the `GitLabSelfManaged` provider in the CLI)

> **Note**
>
> Each connection supports all of the repositories you have with that provider. You only need to create a new connection for each provider type.

Connections allow your pipeline to detect source changes through the third-party provider's installation app. For example, webhooks are used to subscribe to GitHub event types and can be installed on an organization, a repository, or a GitHub App. Your connection installs a repository webhook on your GitHub App that subscribes to GitHub push type events.

After a code change is detected, you have the following options for passing the code to subsequent actions:

- Default: Like other existing CodePipeline source actions, `CodeStarSourceConnection` can output a ZIP file with a shallow copy of your commit.
- Full clone: `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 the AWS Connector installation app to your third-party account when you create a connection. You must have already created your third-party provider account and repository before you can connect through the `CodeStarSourceConnection` action.

> **Note**
>
> To create or attach a policy to your 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.

Note

To use connections in the Europe (Milan) AWS Region, you must:

1. Install a Region-specific app
2. Enable the Region

This Region-specific app supports connections in the Europe (Milan) Region. It is published on the third-party provider site, and it is separate from the existing app supporting connections for other Regions. By installing this app, you authorize third-party providers to share your data with the service for this Region only, and you can revoke the permissions at any time by uninstalling the app.
The service will not process or store your data unless you enable the Region. By enabling this Region, you grant our service permissions to process and store your data.
Even if the Region is not enabled, third-party providers can still share your data with our service if the Region-specific app remains installed, so make sure to uninstall the app once you disable the Region. For more information, see Enabling a Region.

Topics

- Action type
- Configuration parameters
- Input artifacts
- Output artifacts
- Output variables
- Service role permissions: AWS CodeConnections action
- Action declaration
- Installing the installation app and creating a connection
- See also

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`

> **Important**
>
> You must maintain the correct case for the **FullRepositoryId** value. For example, if your user name is `some-user` and repo name is `My-Repo`, the recommended value of **FullRepositoryId** is `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.
> If you choose this option, you will need to update the permissions for your CodeBuild project service role as shown in Add CodeBuild GitClone permissions for connections to Bitbucket, GitHub, GitHub Enterprise Server, or GitLab.com. For a tutorial that shows you how to use the **Full clone** option, see Tutorial: Use full clone with a GitHub pipeline source.

DetectChanges

Required: No

Controls automatically starting your pipeline when a new commit is made on the configured repository and branch. If unspecified, the default value is `true`, and the field does not display by default. Valid values for this parameter:

- `true`: CodePipeline automatically starts your pipeline on new commits.
- `false`: CodePipeline does not start your pipeline on new commits.

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.
> If you choose this option, you will need to update the permissions for your CodeBuild project service role as shown in Troubleshooting CodePipeline. For a tutorial that shows you how to use the **Full clone** option, see Tutorial: Use full clone with a GitHub pipeline source.

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 reference.

AuthorDate

The date when the commit was authored, in timestamp format.

BranchName

The name of the branch for the repository where the source change was made.

CommitId

The commit ID that triggered the pipeline execution.

CommitMessage

The description message, if any, associated with the commit that triggered the pipeline execution.

ConnectionArn

The connection ARN that is configured and authenticated for the source provider.

FullRepositoryName

The name of the repository where the commit that triggered the pipeline was made.

Service role permissions: AWS CodeConnections action

For AWS CodeConnections, the following permission is required to create pipelines with a source that uses a connection, such as Bitbucket Cloud.

```
{
    "Effect": "Allow",
    "Action": [
        "codeconnections:UseConnection",
        "codestar-connections:UseConnection"
    ],
    "Resource": "resource_ARN"
},
```

Note

Both actions are required. Operations invoked through call `codestar-connections:UseConnection`, while operations invoked outside of call `codeconnections:UseConnection`. Granting both actions ensures the role works for both code paths, regardless of whether the connection's resource ARN uses the `codeconnections` or `codestar-connections` service prefix.

For more information about the IAM permissions for connections, see Connections permissions reference.

Action declaration

In the following example, the output artifact is set to the default ZIP format of `CODE_ZIP` for the connection with ARN `arn:aws:codestar-connections:region:account-id:connection/connection-id`.

YAML

```
Name: Source
```

```
Actions:
  - InputArtifacts: []
    ActionTypeId:
      Version: '1'
      Owner: AWS
      Category: Source
      Provider: CodeStarSourceConnection
    OutputArtifacts:
      - Name: SourceArtifact
    RunOrder: 1
    Configuration:
      ConnectionArn: "arn:aws:codestar-connections:region:account-
id:connection/connection-id"
      FullRepositoryId: "some-user/my-repo"
      BranchName: "main"
      OutputArtifactFormat: "CODE_ZIP"
    Name: ApplicationSource
```

JSON

```
{
    "Name": "Source",
    "Actions": [
        {
            "InputArtifacts": [],
            "ActionTypeId": {
                "Version": "1",
                "Owner": "AWS",
                "Category": "Source",
                "Provider": "CodeStarSourceConnection"
            },
            "OutputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "RunOrder": 1,
            "Configuration": {
                "ConnectionArn": "arn:aws:codestar-connections:region:account-
id:connection/connection-id",
                "FullRepositoryId": "some-user/my-repo",
                "BranchName": "main",
                "OutputArtifactFormat": "CODE_ZIP"
```

```
            },
            "Name": "ApplicationSource"
        }
    ]
},
```

Installing the installation app and creating a connection

The first time you use the console to add a new connection to a third-party 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 CloudFormation template, you must provide the connection ARN of a connection that has already gone through the installation handshake. Otherwise, the pipeline is not triggered.

> **Note**
>
> For a `CodeStarSourceConnection` 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.

- AWS::CodeStarConnections::Connection – The CloudFormation template reference for the AWS CodeStar Connections resource provides parameters and examples for connections in CloudFormation templates.
- AWS CodeStar Connections API Reference – The *AWS CodeStar Connections API Reference* provides reference information for the available connections actions.
- To view the steps for creating a pipeline with source actions supported by connections, see the following:
  - For Bitbucket Cloud, use the **Bitbucket** option in the console or the `CodestarSourceConnection` action in the CLI. See Bitbucket Cloud connections.

- For GitHub and GitHub Enterprise Cloud, use the **GitHub** provider option in the console or the `CodestarSourceConnection` action in the CLI. See GitHub connections.
- For GitHub Enterprise Server, use the **GitHub Enterprise Server** provider option in the console or the `CodestarSourceConnection` action in the CLI. See GitHub Enterprise Server connections.
- For GitLab.com, use the **GitLab** provider option in the console or the `CodestarSourceConnection` action with the `GitLab` provider in the CLI. See GitLab.com connections.

- To view a Getting Started tutorial that creates a pipeline with a Bitbucket source and a CodeBuild action, see Getting started with connections.
- For a tutorial that shows you how to connect to a GitHub repository and use the **Full clone** option with a downstream CodeBuild action, see Tutorial: Use full clone with a GitHub pipeline source.

Commands action reference

The Commands action allows you to run shell commands in a virtual compute instance. When you run the action, commands specified in the action configuration are run in a separate container. All artifacts that are specified as input artifacts to a CodeBuild action are available inside of the container running the commands. This action allows you to specify commands without first creating a CodeBuild project. For more information, see ActionDeclaration and OutputArtifact in the *AWS CodePipeline API Reference*.

> **Important**
>
> This action uses CodePipeline managed CodeBuild compute to run commands in a build environment. Running the commands action will incur separate charges in AWS CodeBuild.

> **Note**
>
> The Commands action is only available for V2 type pipelines.

Topics

- Considerations for the Commands action

- Service role policy permissions
- Action type
- Configuration parameters
- Input artifacts
- Output artifacts
- Environment variables
- Service role permissions: Commands action
- Action declaration (example)
- See also

Considerations for the Commands action

The following considerations apply for the Commands action.

- The commands action uses CodeBuild resources similar to the CodeBuild action, while allowing shell environment commands in a virtual compute instance without the need to associate or create a build project.

  > **Note**
  >
  > Running the commands action will incur separate charges in AWS CodeBuild.

- Because the Commands action in CodePipeline uses CodeBuild resources, the builds run by the action will be attributed to the build limits for your account in CodeBuild. Builds run by the Commands action will count toward the concurrent build limits as configured for that account.
- The timeout for builds with the Commands action is 55 minutes, as based on CodeBuild builds.
- The compute instance uses an isolated build environment in CodeBuild.

  > **Note**
  >
  > Because the isolated build environment is used at the account level, an instance might be reused for another pipeline execution.

- All formats are supported except multi-line formats. You must use single-line format when entering commands.

- The commands action is supported for cross-account actions. To add a cross-account commands action, add `actionRoleArn` from your target account in the action declaration.
- For this action, CodePipeline will assume the pipeline service role and use that role to allow access to resources at runtime. It is recommended to configure the service role so that the permissions are scoped down to the action level.
- The permissions added to the CodePipeline service role are detailed in Add permissions to the CodePipeline service role.
- The permission needed to view logs in the console is detailed in Permissions required to view compute logs in the console.
- Unlike other actions in CodePipeline, you do not set fields in the action configuration; you set the action configuration fields outside of the action configuration.

Service role policy permissions

When CodePipeline runs the action, CodePipeline creates a log group using the name of the pipeline as follows. This enables you to scope down permissions to log resources using the pipeline name.

```
/aws/codepipeline/MyPipelineName
```

If you are using an existing service role, to use the Commands action, you will need to add the following permissions for the service role.

- logs:CreateLogGroup
- logs:CreateLogStream
- logs:PutLogEvents

In the service role policy statement, scope down the permissions to the pipeline level as shown in the following example.

```
{
    "Effect": "Allow",
    "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
```

```
        "logs:PutLogEvents"
    ],
    "Resource": [
        "arn:aws:logs:*:YOUR_AWS_ACCOUNT_ID:log-group:/aws/
codepipeline/YOUR_PIPELINE_NAME",
        "arn:aws:logs:*:YOUR_AWS_ACCOUNT_ID:log-group:/aws/
codepipeline/YOUR_PIPELINE_NAME:*"
   ]
}
```

To view logs in the console using the action details dialog page, the permission to view logs must be added to the console role. For more information, see the console permissions policy example in Permissions required to view compute logs in the console.

Action type

- Category: `Compute`
- Owner: `AWS`
- Provider: `Commands`
- Version: 1

Configuration parameters

Commands

Required: Yes

You can provide shell commands for the `Commands` action to run. In the console, commands are entered on separate lines. In the CLI, commands are entered as separate strings.

> **Note**
>
> Multi-line formats are not supported and will result in an error message. Single-line format must be used for entering commands in the **Commands** field.

> **Important**
>
> The EnvironmentType and ComputeType values match those in CodeBuild. We support a subset of the available types. For more information, see Build Environment Compute Types.

EnvironmentType

Required: No

The OS image for the build environment that supports the Commands action. The following are valid values for build environments:

- LINUX_CONTAINER
- WINDOWS_SERVER_2022_CONTAINER

The selection for **EnvironmentType** will then allow the compute type for that OS in the **ComputeType** field. For more information about the CodeBuild compute types available for this action, see the Build environment compute modes and types reference in the CodeBuild User Guide.

> **Note**
>
> If not specified, the compute defaults to the following for the build environment:
>
> - **Compute type:** BUILD_GENERAL1_SMALL
> - **Environment type:** LINUX_CONTAINER

ComputeType

Required: No

Based on the selection for EnvironmentType, the compute type can be provided. The following are available values for compute; however, note that the options available can vary by OS.

- BUILD_GENERAL1_SMALL
- BUILD_GENERAL1_MEDIUM
- BUILD_GENERAL1_LARGE

> **Important**
>
> Some compute types are not compatible with certain environment types.
> For example, WINDOWS_SERVER_2022_CONTAINER is not compatible with BUILD_GENERAL1_SMALL. Using incompatible combinations causes the action to fail and generates a runtime error.

outputVariables

Required: No

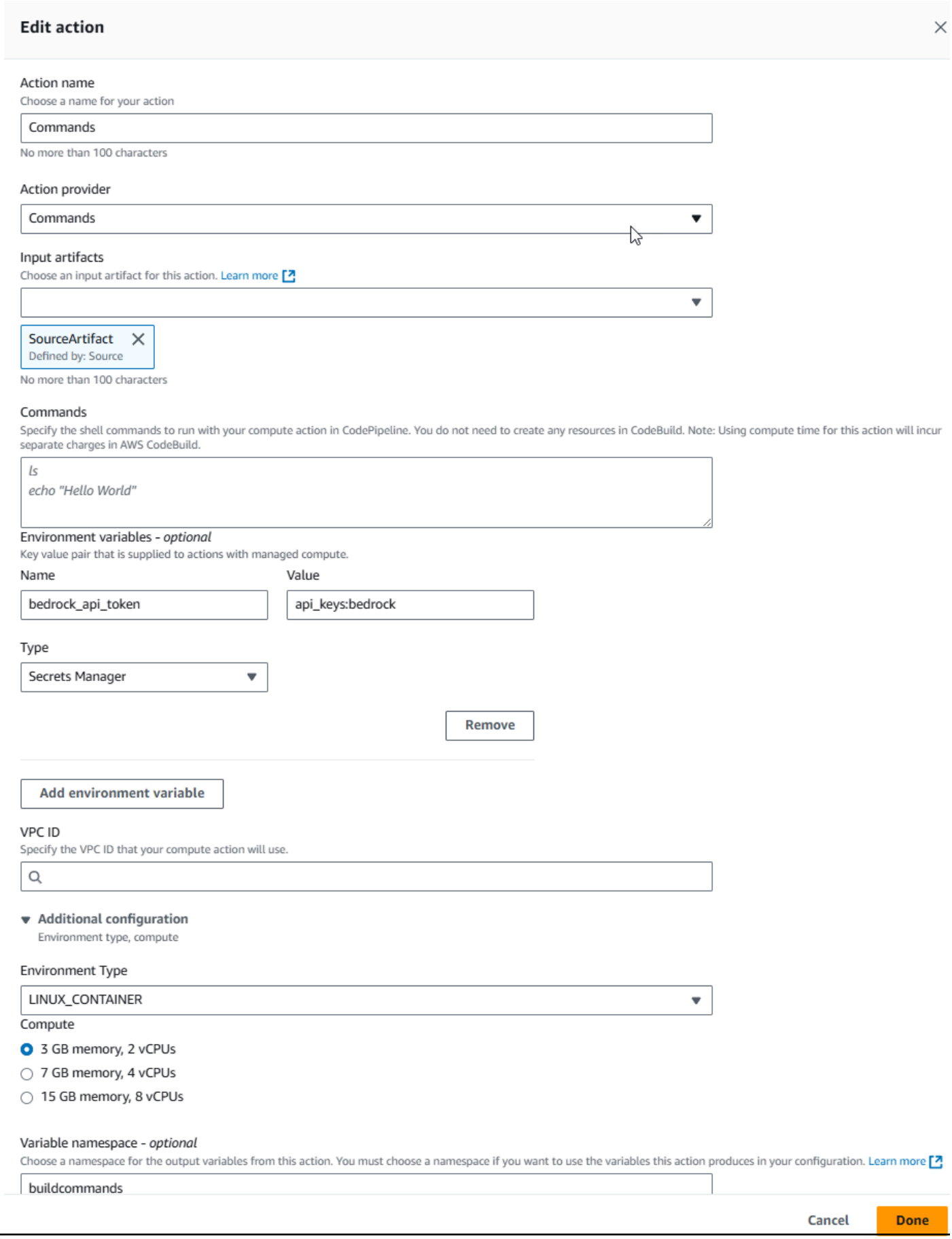
Specify the names of the variables in your environment that you want to export. For a reference of CodeBuild environment variables, see Environment variables in build environments in the *CodeBuild User Guide*.

Files

Required: No

You can provide files that you want to export as output artifacts for the action.

The supported format for files is the same as for CodeBuild file patterns. For example, enter **/ for all files. For more information, see Build specification reference for CodeBuild in the *CodeBuild User Guide*.


Edit action
Action name
Choose a name for your action
Commands
No more than 100 characters
Action provider
Commands
Input artifacts
Choose an input artifact for this action. Learn more
SourceArtifact
Defined by: Source
No more than 100 characters
Commands
Specify the shell commands to run with your compute action in CodePipeline. You do not need to create any resources in CodeBuild. Note: Using compute time for this action will incur separate charges in AWS CodeBuild.
ls
echo "Hello World"
Environment variables - optional
Key value pair that is supplied to actions with managed compute.
Name
Value
bedrock_api_token
api_keys:bedrock
Type
Secrets Manager
Remove
Add environment variable
VPC ID
Specify the VPC ID that your compute action will use.
Additional configuration
Environment type, compute
Environment Type
LINUX_CONTAINER
Compute
3 GB memory, 2 vCPUs
7 GB memory, 4 vCPUs
15 GB memory, 8 vCPUs
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
buildcommands
Cancel
Done

VpcId

Required: No

The VPC ID for your resources.

Subnets

Required: No

The subnets for the VPC. This field is needed when your commands need to connect to resources in a VPC.

SecurityGroupIds

Required: No

The security groups for the VPC. This field is needed when your commands need to connect to resources in a VPC.

The following is a JSON example of the action with configuration fields shown for environment and compute type, along with an example environment variable.

```
{
            "name": "Commands1",
            "actionTypeId": {
              "category": "Compute",
              "owner": "AWS",
              "provider": "Commands",
              "version": "1"
            },
            "inputArtifacts": [
              {
                "name": "SourceArtifact"
              }
            ],
            "commands": [
              "ls",
              "echo hello",
              "echo $BEDROCK_TOKEN",
            ],
            "configuration": {
              "EnvironmentType": "LINUX_CONTAINER",
```

```
            "ComputeType": "BUILD_GENERAL1_MEDIUM"
          },
          "environmentVariables": [
            {
              "name": "BEDROCK_TOKEN",
              "value": "apiTokens:bedrockToken",
              "type": "SECRETS_MANAGER"
            }
          ],
          "runOrder": 1
        }
```

Input artifacts

- **Number of artifacts:** `1 to 10`

Output artifacts

- **Number of artifacts:** `0 to 1`

Environment variables

Key

The key in a key-value environment variable pair, such as `BEDROCK_TOKEN`.

Value

The value for the key-value pair, such as `apiTokens:bedrockToken`. The value can be parameterized with output variables from pipeline actions or pipeline variables.

When using the `SECRETS_MANAGER` type, this value must be the name of a secret you have already stored in AWS Secrets Manager.

Type

Specifies the type of use for the environment variable value. The value can be either `PLAINTEXT` or `SECRETS_MANAGER`. If the value is `SECRETS_MANAGER`, provide the Secrets reference in the `EnvironmentVariable` value. When not specified, this defaults to `PLAINTEXT`.

> **Note**
>
> We strongly discourage the use of *plaintext* environment variables to store sensitive values, especially AWS credentials. When you use the CodeBuild console or AWS CLI, *plaintext* environment variables are displayed in plain text. For sensitive values, we recommend that you use the `SECRETS_MANAGER` type instead.

> **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 an example action declaration showing an environment variable, see Configuration parameters.

> **Note**
>
> - The `SECRETS_MANAGER` type is only supported for the Commands action.
> - Secrets referenced in the Commands action will be redacted in the build logs similar to CodeBuild. But pipeline users who have **Edit** access to the pipeline can still potentially access these secret values by modifying the commands.
> - To use the SecretsManager, you must add the following permissions to your pipeline service role:
>
> ```
> {
>             "Effect": "Allow",
>             "Action": [
>                 "secretsmanager:GetSecretValue"
>             ],
>             "Resource": [
>                 "SECRET_ARN"
>             ]
>         }
> ```

Service role permissions: Commands action

For Commands support, add the following to your policy statement:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws-eusc:iam::*:role/Service*",
                "arn:aws-eusc:iam::*:role/Service*"
            ]
        }
    ]
}
```

Action declaration (example)

YAML

```
name: Commands_action
actionTypeId:
  category: Compute
  owner: AWS
  provider: Commands
  version: '1'
runOrder: 1
configuration: {}
commands:
- ls
- echo hello
- 'echo pipeline Execution Id is #{codepipeline.PipelineExecutionId}'
outputArtifacts:
```

```
- name: BuildArtifact
  files:
  - **/
inputArtifacts:
- name: SourceArtifact
outputVariables:
- AWS_DEFAULT_REGION
region: us-east-1
namespace: compute
```

JSON

```
{
    "name": "Commands_action",
    "actionTypeId": {
        "category": "Compute",
        "owner": "AWS",
        "provider": "Commands",
        "version": "1"
    },
    "runOrder": 1,
    "configuration": {},
    "commands": [
        "ls",
        "echo hello",
        "echo pipeline Execution Id is #{codepipeline.PipelineExecutionId}"
    ],
    "outputArtifacts": [
        {
            "name": "BuildArtifact",
            "files": [
                "**/"
            ]
        }
    ],
    "inputArtifacts": [
        {
            "name": "SourceArtifact"
        }
    ],
    "outputVariables": [
        "AWS_DEFAULT_REGION"
    ],
```

```
    "region": "us-east-1",
    "namespace": "compute"
}
```

See also

The following related resources can help you as you work with this action.

- Tutorial: Create a pipeline that runs commands with compute (V2 type) – This tutorial provides a sample pipeline with the Commands action.

AWS Device Farm test action reference

In your pipeline, you can configure a test action that uses AWS Device Farm to run and test your application on devices. Device Farm uses test pools of devices and testing frameworks to test applications on specific devices. For information about the types of testing frameworks supported by the Device Farm action, see Working with Test Types in AWS Device Farm.

Topics

- Action type
- Configuration parameters
- Input artifacts
- Output artifacts
- Service role permissions: AWS Device Farm action
- Action declaration
- See also

Action type

- Category: `Test`
- Owner: `AWS`
- Provider: `DeviceFarm`
- Version: `1`

Configuration parameters

AppType

Required: Yes

The OS and type of application you are testing. The following is a list of valid values:

- `iOS`
- `Android`
- `Web`

ProjectId

Required: Yes

The Device Farm project ID.

To find your project ID, in the Device Farm console, choose your project. In the browser, copy the URL of your new project. The URL contains the project ID. The project ID is the value in the URL after `projects/`. In the following example, the project ID is `eec4905f-98f8-40aa-9afc-4c1cfexample`.

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

App

Required: Yes

The name and location of the application file in your input artifact. For example: `s3-ios-test-1.ipa`

TestSpec

Conditional: Yes

The location of the test spec definition file in your input artifact. This is required for custom mode test.

DevicePoolArn

Required: Yes

The Device Farm device pool ARN.

To get the available device pool ARNs for the project, including the ARN for Top Devices, use the AWS CLI to enter the following command:

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

TestType

Required: Yes

Specifies the supported testing framework for your test. The following is a list of valid values for `TestType`:

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

> **Note**
>
> The following test types are not supported by the action in CodePipeline: `WEB_PERFORMANCE_PROFILE`, `REMOTE_ACCESS_RECORD`, and `REMOTE_ACCESS_REPLAY`.

For information about Device Farm test types, see Working with Test Types in AWS Device Farm.

RadioBluetoothEnabled

Required: No

A Boolean value that indicates whether to enable Bluetooth at the beginning of the test.

RecordAppPerformanceData

Required: No

A Boolean value that indicates whether to record device performance data such as CPU, FPS, and memory performance during the test.

RecordVideo

Required: No

A Boolean value that indicates whether to record video during the test.

RadioWifiEnabled

Required: No

A Boolean value that indicates whether to enable Wi-Fi at the beginning of the test.

RadioNfcEnabled

Required: No

A Boolean value that indicates whether to enable NFC at the beginning of the test.

RadioGpsEnabled

Required: No

A Boolean value that indicates whether to enable GPS at the beginning of the test.

Test

Required: No

The name and path of the test definition file in your source location. The path is relative to the root of the input artifact for your test.

FuzzEventCount

Required: No

The number of user interface events for the fuzz test to perform, between 1 and 10,000.

FuzzEventThrottle

Required: No

The number of milliseconds for the fuzz test to wait before performing the next user interface event, between 1 and 1,000.

FuzzRandomizerSeed

Required: No

A seed for the fuzz test to use for randomizing user interface events. Using the same number for subsequent fuzz tests results in identical event sequences.

CustomHostMachineArtifacts

Required: No

The location on the host machine where custom artifacts will be stored.

CustomDeviceArtifacts

Required: No

The location on the device where custom artifacts will be stored.

UnmeteredDevicesOnly

Required: No

A Boolean value that indicates whether to only use your unmetered devices when running tests in this step.

JobTimeoutMinutes

Required: No

The number of minutes a test run will execute per device before it times out.

Latitude

Required: No

The latitude of the device expressed in geographic coordinate system degrees.

Longitude

Required: No

The longitude of the device expressed in geographic coordinate system degrees.

Input artifacts

- **Number of artifacts:** 1
- **Description:** The set of artifacts to be made available to the test action. Device Farm looks for the built application and test definitions to use.

Output artifacts

- **Number of Artifacts:** 0
- **Description:** Output artifacts do not apply for this action type.

Service role permissions: AWS Device Farm action

When CodePipeline runs the action, the CodePipeline service role policy requires the following permissions, appropriately scoped down to the pipeline resource ARN in order to maintain access with least privilege. For example, add the following to your policy statement:

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

```
},
```

Action declaration

YAML

```
Name: Test
Actions:
  - Name: TestDeviceFarm
    ActionTypeId: null
    category: Test
    owner: AWS
    provider: DeviceFarm
    version: '1'
RunOrder: 1
Configuration:
  App: s3-ios-test-1.ipa
  AppType: iOS
  DevicePoolArn: >-
    arn:aws:devicefarm:us-west-2::devicepool:0EXAMPLE-d7d7-48a5-ba5c-b33d66efa1f5
  ProjectId: eec4905f-98f8-40aa-9afc-4c1cfEXAMPLE
  TestType: APPIUM_PYTHON
  TestSpec: example-spec.yml
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: us-west-2
```

JSON

```
{
    "Name": "Test",
    "Actions": [
        {
            "Name": "TestDeviceFarm",
            "ActionTypeId": null,
            "category": "Test",
            "owner": "AWS",
            "provider": "DeviceFarm",
            "version": "1"
        }
    ],
    "RunOrder": 1,
```

```
    "Configuration": {
        "App": "s3-ios-test-1.ipa",
        "AppType": "iOS",
        "DevicePoolArn": "arn:aws:devicefarm:us-west-2::devicepool:0EXAMPLE-
d7d7-48a5-ba5c-b33d66efa1f5",
        "ProjectId": "eec4905f-98f8-40aa-9afc-4c1cfEXAMPLE",
        "TestType": "APPIUM_PYTHON",
        "TestSpec": "example-spec.yml"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "us-west-2"
},
```

See also

The following related resources can help you as you work with this action.

- Working with Test Types in Device Farm – This reference chapter in the *Device Farm Developer Guide* provides more description about the Android, iOS, and Web Application testing frameworks supported by Device Farm.
- Actions in Device Farm – The API calls and parameters in the *Device Farm API Reference* can help you work with Device Farm projects.
- Tutorial: Create a pipeline that builds and tests your Android app with AWS Device Farm – This tutorial provides a sample buildspec file and sample application to create a pipeline with a GitHub source that builds and tests an Android app with CodeBuild and Device Farm.
- Tutorial: Create a pipeline that tests your iOS app with AWS Device Farm – This tutorial provides a sample application to create a pipeline with an Amazon S3 source that tests a built iOS app with Device Farm.

Elastic Beanstalk deploy action reference

Elastic Beanstalk is a platform within AWS that is used for deploying and scaling web applications. You use an Elastic Beanstalk action to deploy application code to your deployment environment.

Topics

- Action type
- Configuration parameters
- Input artifacts
- Output artifacts
- Service role permissions: ElasticBeanstalk deploy action
- Action declaration
- See also

Action type

- Category: `Deploy`
- Owner: `AWS`
- Provider: `ElasticBeanstalk`
- Version: 1

Configuration parameters

ApplicationName

Required: Yes

The name of the application that you created in Elastic Beanstalk.

EnvironmentName

Required: Yes

The name of the environment that you created in Elastic Beanstalk. An environment is a collection of AWS resources running an application version. Each environment runs only one application version at a time; however, you can run the same application version or different application versions in many environments simultaneously.

Input artifacts

- **Number of artifacts:** 1

- **Description:** The input artifact for the action.

Output artifacts

- **Number of artifacts:** `0`
- **Description:** Output artifacts do not apply for this action type.

Service role permissions: `ElasticBeanstalk` deploy action

For Elastic Beanstalk, the following are the minimum permissions needed to create pipelines with an `ElasticBeanstalk` deploy action.

```
{
    "Effect": "Allow",
    "Action": [
        "elasticbeanstalk:*",
        "ec2:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "cloudformation:*",
        "rds:*",
        "sqs:*",
        "ecs:*"
    ],
    "Resource": "resource_ARN"
},
```

> **Note**
>
> You should replace wildcards in the resource policy with the resources for the account you want to limit access to. For more information about creating a policy that grants least-privilege access, see https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege.

Action declaration

YAML

```
Name: Deploy
Actions:
  - Name: Deploy
    ActionTypeId:
      Category: Deploy
      Owner: AWS
      Provider: ElasticBeanstalk
      Version: '1'
    RunOrder: 1
    Configuration:
      ApplicationName: my-application
      EnvironmentName: my-environment
    OutputArtifacts: []
    InputArtifacts:
      - Name: SourceArtifact
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
        {
            "Name": "Deploy",
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "ElasticBeanstalk",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "ApplicationName": "my-application",
                "EnvironmentName": "my-environment"
            },
            "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.

- Deploying a Flask application to Elastic Beanstalk – This tutorial walks you through the creation of your application and environment resources in Elastic Beanstalk using a sample Flask application. You can then build your pipeline with an Elastic Beanstalk deployment action that deploys your application from your source repository to your Elastic Beanstalk environment.

Amazon Inspector `InspectorScan` invoke action reference

Amazon Inspector is a vulnerability management service that automatically discovers workloads and continually scans them for software vulnerabilities and unintended network exposure. The `InspectorScan` action in CodePipeline automates detecting and fixing security vulnerabilities in your open source code. The action is a managed compute action with security scanning capabilities. You can use InspectorScan with application source code in your third-party repository, such as GitHub or Bitbucket Cloud, or with images for container applications. Your action will scan and report on vulnerability levels and alerts that you configure.

> **Important**
>
> This action uses CodePipeline managed CodeBuild compute to run commands in a build environment. Running the action will incur separate charges in AWS CodeBuild.

Topics

- Action type ID
- Configuration parameters

- Input artifacts
- Output artifacts
- Output variables
- Service role permissions: InspectorScan action
- Action declaration
- See also

Action type ID

- Category: `Invoke`
- Owner: `AWS`
- Provider: `InspectorScan`
- Version: 1

Example:

```
{
    "Category": "Invoke",
    "Owner": "AWS",
    "Provider": "InspectorScan",
    "Version": "1"
},
```

Configuration parameters

InspectorRunMode

(Required) The string that indicates the mode of the scan. Valid values are `SourceCodeScan | ECRImageScan`.

ECRRepositoryName

The name of the Amazon ECR repository where the image was pushed.

ImageTag

The tag used for the image.

The parameters for this action scan for levels of vulnerability that you specify. The following levels for vulnerability thresholds are available:

CriticalThreshold

The number of critical severity vulnerabilities found in your source beyond which CodePipeline should fail the action.

HighThreshold

The number of high severity vulnerabilities found in your source beyond which CodePipeline should fail the action.

MediumThreshold

The number of medium severity vulnerabilities found in your source beyond which CodePipeline should fail the action.

LowThreshold

The number of low severity vulnerabilities found in your source beyond which CodePipeline should fail the action.

Action name
Choose a name for your action
Scan
No more than 100 characters

Action provider
AWS InspectorScan

Region
US East (N. Virginia)

Input artifacts
Choose an input artifact for this action. Learn more

SourceArtifact
Defined by: Source
No more than 100 characters

Inspector Run mode
InspectorRunMode: SourceCodeScan or ECRImageScan.

Source Code Scan
SourceCodeScan: Scan the source code. Choose an input artifact for this run mode.

ECR Image Scan
ECRImageScan: Scan the ECR image.

Critical Threshold - *optional*
Threshold value for critical severity vulnerabilities to allow the action to succeed. Default: Integer.MAX_VALUE

High Threshold - *optional*
Threshold value for high severity vulnerabilities to allow the action to succeed. Default: Integer.MAX_VALUE

Medium Threshold - *optional*
Threshold value for medium severity vulnerabilities to allow the action to succeed. Default: Integer.MAX_VALUE

Low Threshold - *optional*
Threshold value for low severity vulnerabilities to allow the action to succeed. Default: Integer.MAX_VALUE

Input artifacts

- **Number of artifacts:** 1
- **Description:** The source code to scan for vulnerabilities. If the scan is for an ECR repository, this input artifact is not needed.

Output artifacts

- **Number of artifacts:** 1
- **Description:** Vulnerability details of your source in the form of a Software Bill of Materials (SBOM) file.

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 reference.

HighestScannedSeverity

The highest severity output from the scan. Valid values are `medium | high | critical`.

Service role permissions: `InspectorScan` action

For the `InspectorScan` action support, add the following to your policy statement:

```
{
        "Effect": "Allow",
        "Action": "inspector-scan:ScanSbom",
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "ecr:GetDownloadUrlForLayer",
```

```
            "ecr:BatchGetImage",
            "ecr:BatchCheckLayerAvailability"
        ],
        "Resource": "resource_ARN"
    },
```

In addition, if not already added for the Commands action, add the following permissions to your service role in order to view CloudWatch logs.

```
{
    "Effect": "Allow",
    "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
    ],
    "Resource": "resource_ARN"
},
```

> **Note**
>
> Scope down the permissions to the pipeline resource level by using resource-based permissions in the service role policy statement.

Action declaration

YAML

```
name: Scan
actionTypeId:
  category: Invoke
  owner: AWS
  provider: InspectorScan
  version: '1'
runOrder: 1
configuration:
  InspectorRunMode: SourceCodeScan
outputArtifacts:
- name: output
inputArtifacts:
```

```
- name: SourceArtifact
region: us-east-1
```

JSON

```
{
                        "name": "Scan",
                        "actionTypeId": {
                            "category": "Invoke",
                            "owner": "AWS",
                            "provider": "InspectorScan",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "InspectorRunMode": "SourceCodeScan"
                        },
                        "outputArtifacts": [
                            {
                                "name": "output"
                            }
                        ],
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "region": "us-east-1"
                    },
```

See also

The following related resources can help you as you work with this action.

- For more information about Amazon Inspector, see the Amazon Inspector User Guide.

AWS Lambda invoke action reference

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. 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.

> **Important**
>
> Do not log the JSON event that CodePipeline sends to Lambda because this can result in user credentials being logged in CloudWatch Logs. The CodePipeline role uses a JSON event to pass temporary credentials to Lambda in the `artifactCredentials` field. For an example event, see Example JSON event.

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 reference.

Example action configuration

YAML

```
Name: Lambda
Actions:
  - Name: Lambda
    ActionTypeId:
      Category: Invoke
      Owner: AWS
      Provider: Lambda
      Version: '1'
    RunOrder: 1
    Configuration:
      FunctionName: myLambdaFunction
      UserParameters: 'http://192.0.2.4'
    OutputArtifacts: []
    InputArtifacts: []
    Region: us-west-2
```

JSON

```
{
    "Name": "Lambda",
    "Actions": [
        {
            "Name": "Lambda",
            "ActionTypeId": {
                "Category": "Invoke",
                "Owner": "AWS",
                "Provider": "Lambda",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "FunctionName": "myLambdaFunction",
                "UserParameters": "http://192.0.2.4"
            },
            "OutputArtifacts": [],
            "InputArtifacts": [],
            "Region": "us-west-2"
        }
    ]
},
```

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.

```
{
    "CodePipeline.job": {
        "id": "11111111-abcd-1111-abcd-111111abcdef",
        "accountId": "111111111111",
        "data": {
            "actionConfiguration": {
                "configuration": {
                    "FunctionName": "MyLambdaFunction",
                    "UserParameters": "input_parameter"
```

```
                }
            },
            "inputArtifacts": [
                {
                    "location": {
                        "s3Location": {
                            "bucketName": "bucket_name",
                            "objectKey": "filename"
                        },
                        "type": "S3"
                    },
                    "revision": null,
                    "name": "ArtifactName"
                }
            ],
            "outputArtifacts": [],
            "artifactCredentials": {
                "secretAccessKey": "secret_key",
                "sessionToken": "session_token",
                "accessKeyId": "access_key_ID"
            },
            "continuationToken": "token_ID",
            "encryptionKey": {
              "id": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
              "type": "KMS"
            }
        }
    }
}
```

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.
  - `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.
    - `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.
    - `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 access key ID 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 CloudFormation User Guide – For more information about Lambda actions and CloudFormation artifacts for pipelines, see Using Parameter Override Functions with CodePipeline Pipelines, Automating Deployment of Lambda-based Applications, and AWS CloudFormation Artifacts.
- Invoke an AWS Lambda function in a pipeline in CodePipeline – This procedure provides a sample Lambda function and shows you how to use the console to create a pipeline with a Lambda invoke action.

AWS OpsWorks deploy action reference

You use an AWS OpsWorks action to deploy with OpsWorks using your pipeline.

Action type

- Category: `Deploy`
- Owner: `AWS`
- Provider: `OpsWorks`
- Version: `1`

Configuration parameters

App

Required: Yes

The OpsWorks app. The app represents the code you want to deploy and run.

Stack

Required: Yes

The OpsWorks stack. A stack is a container for your application infrastructure.

Layer

Required: No

The OpsWorks layer. A layer specifies the configuration and resources for a set of instances.

Input artifacts

- **Number of artifacts:** `1`
- **Description:** This is the input artifact for your action.

Output artifacts

- **Number of artifacts:** `0 to 1`
- **Description:** Output artifacts do not apply for this action type.

Service role permissions: AWS OpsWorks action

For AWS OpsWorks support, add the following to your policy statement:

```
{
    "Effect": "Allow",
    "Action": [
        "opsworks:CreateDeployment",
        "opsworks:DescribeApps",
        "opsworks:DescribeCommands",
        "opsworks:DescribeDeployments",
        "opsworks:DescribeInstances",
        "opsworks:DescribeStacks",
        "opsworks:UpdateApp",
        "opsworks:UpdateStack"
    ],
    "Resource": "resource_ARN"
},
```

Example action configuration

YAML

```
Name: ActionName
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Version: 1
  Provider: OpsWorks
InputArtifacts:
  - Name: myInputArtifact
Configuration:
  Stack: my-stack
  App: my-app
```

JSON

```
{
    "Name": "ActionName",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "OpsWorks"
    },
    "InputArtifacts": [
        {
            "Name": "myInputArtifact"
        }
    ],
    "Configuration": {
        "Stack": "my-stack",
        "App": "my-app"
    }
}
```

See also

The following related resources can help you as you work with this action.

- AWS OpsWorks User Guide – For information about deploying with AWS OpsWorks, see the *AWS OpsWorks User Guide*.

AWS Service Catalog deploy action reference

You use an AWS Service Catalog action to deploy templates using your pipeline. These are resource templates that you have created in Service Catalog.

Action type

- Category: `Deploy`
- Owner: `AWS`
- Provider: `ServiceCatalog`
- Version: `1`

Configuration parameters

TemplateFilePath

Required: Yes

The file path for your resource template in your source location.

ProductVersionName

Required: Yes

The product version in Service Catalog.

ProductType

Required: Yes

The product type in Service Catalog.

ProductId

Required: Yes

The product ID in Service Catalog.

ProductVersionDescription

Required: No

The product version description in Service Catalog.

Input artifacts

- **Number of artifacts:** 1
- **Description:** This is the input artifact for your action.

Output artifacts

- **Number of artifacts:** 0
- **Description:** Output artifacts do not apply for this action type.

Service role permissions: Service Catalog action

For Service Catalog support, add the following to your policy statement:

```
{
    "Effect": "Allow",
    "Action": [
        "servicecatalog:ListProvisioningArtifacts",
        "servicecatalog:CreateProvisioningArtifact",
        "servicecatalog:DescribeProvisioningArtifact",
        "servicecatalog:DeleteProvisioningArtifact",
        "servicecatalog:UpdateProduct"
    ],
    "Resource": "resource_ARN"
},
{
    "Effect": "Allow",
    "Action": [
        "cloudformation:ValidateTemplate"
    ],
    "Resource": "resource_ARN"
}
```

Example action configurations by type of configuration file

The following example shows a valid configuration for a deploy action that uses 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 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"
}
```

Example action configuration

YAML

```
Name: ActionName
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Version: 1
  Provider: ServiceCatalog
OutputArtifacts:
- Name: myOutputArtifact
Configuration:
  TemplateFilePath: S3_template.json
  ProductVersionName: devops S3 v2
  ProductType: CLOUD_FORMATION_TEMPLATE
  ProductVersionDescription: Product version description
  ProductId: prod-example123456
```

JSON

```
{
    "Name": "ActionName",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "ServiceCatalog"
    },
    "OutputArtifacts": [
        {
            "Name": "myOutputArtifact"
        }
    ],
    "Configuration": {
        "TemplateFilePath": "S3_template.json",
        "ProductVersionName": "devops S3 v2",
        "ProductType": "CLOUD_FORMATION_TEMPLATE",
        "ProductVersionDescription": "Product version description",
        "ProductId": "prod-example123456"
    }
}
```

See also

The following related resources can help you as you work with this action.

- Service Catalog User Guide – For information about resources and templates in Service Catalog, see the *Service Catalog User Guide*.
- Tutorial: Create a pipeline that deploys to Service Catalog – This tutorial shows you how to create and configure a pipeline to deploy your product template to Service Catalog and deliver changes you have made in your source repository.

AWS Step Functions invoke action reference

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.

> **Note**
>
> The Step Functions action runs on Lambda, and it therefore has artifact size quotas that are the same as the artifact size quotas for Lambda functions. For more information, see Lambda quotas in the Lambda Developer Guide.

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-6bafd7fde791
```

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:

  ```
  {"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:**
  - **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 reference.

StateMachineArn

The ARN of the state machine.

ExecutionArn

The ARN of the execution of the state machine. Standard state machines only.

Service role permissions: `StepFunctions` action

For the `StepFunctions` action, the following are the minimum permissions needed to create pipelines with a Step Functions invoke action.

```
{
    "Effect": "Allow",
    "Action": [
        "states:DescribeStateMachine",
        "states:DescribeExecution",
        "states:StartExecution"
    ],
    "Resource": "resource_ARN"
},
```

Example action configuration

Example for default input

YAML

```
Name: ActionName
ActionTypeId:
  Category: Invoke
  Owner: AWS
  Version: 1
  Provider: StepFunctions
OutputArtifacts:
  - Name: myOutputArtifact
Configuration:
  StateMachineArn: arn:aws:states:us-east-1:111122223333:stateMachine:HelloWorld-
StateMachine
  ExecutionNamePrefix: my-prefix
```

JSON

```
{
    "Name": "ActionName",
    "ActionTypeId": {
        "Category": "Invoke",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "StepFunctions"
    },
    "OutputArtifacts": [
        {
            "Name": "myOutputArtifact"
        }
    ],
    "Configuration": {
        "StateMachineArn": "arn:aws:states:us-
east-1:111122223333:stateMachine:HelloWorld-StateMachine",
        "ExecutionNamePrefix": "my-prefix"
    }
}
```

Example for literal input

YAML

```
Name: ActionName
ActionTypeId:
  Category: Invoke
  Owner: AWS
  Version: 1
  Provider: StepFunctions
OutputArtifacts:
  - Name: myOutputArtifact
Configuration:
  StateMachineArn: arn:aws:states:us-east-1:111122223333:stateMachine:HelloWorld-
StateMachine
  ExecutionNamePrefix: my-prefix
  Input: '{"action": "test"}'
```

JSON

```
{
    "Name": "ActionName",
    "ActionTypeId": {
        "Category": "Invoke",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "StepFunctions"
    },
    "OutputArtifacts": [
        {
            "Name": "myOutputArtifact"
        }
    ],
    "Configuration": {
        "StateMachineArn": "arn:aws:states:us-
east-1:111122223333:stateMachine:HelloWorld-StateMachine",
        "ExecutionNamePrefix": "my-prefix",
        "Input": "{\"action\": \"test\"}"
    }
}
```

Example for input file

YAML

```
Name: ActionName
InputArtifacts:
  - Name: myInputArtifact
ActionTypeId:
  Category: Invoke
  Owner: AWS
  Version: 1
  Provider: StepFunctions
OutputArtifacts:
  - Name: myOutputArtifact
Configuration:
  StateMachineArn: 'arn:aws:states:us-east-1:111122223333:stateMachine:HelloWorld-
StateMachine'
  ExecutionNamePrefix: my-prefix
  InputType: FilePath
  Input: assets/input.json
```

JSON

```
{
    "Name": "ActionName",
    "InputArtifacts": [
        {
            "Name": "myInputArtifact"
        }
    ],
    "ActionTypeId": {
        "Category": "Invoke",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "StepFunctions"
    },
    "OutputArtifacts": [
        {
            "Name": "myOutputArtifact"
        }
    ],
    "Configuration": {
```

```
        "StateMachineArn": "arn:aws:states:us-
east-1:111122223333:stateMachine:HelloWorld-StateMachine",
        "ExecutionNamePrefix": "my-prefix",
        "InputType": "FilePath",
        "Input": "assets/input.json"
    }
}
```

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 – 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.

Rule structure reference

This section is a reference for rule configuration only. For a conceptual overview of the pipeline structure, see CodePipeline pipeline structure reference.

Each rule provider in CodePipeline uses a set of required and optional configuration fields in the pipeline structure. This section provides the following reference information by rule provider:

- Valid values for the `RuleType` fields included in the pipeline structure rule block, such as `Owner` and `Provider`.
- Descriptions and other reference information for the `Configuration` parameters (required and optional) included in the pipeline structure rule section.
- Valid example JSON and YAML rule configuration fields.

Reference information is available for the following rule providers:

Topics

- CloudWatchAlarm
- CodeBuild rule
- Commands
- DeploymentWindow
- LambdaInvoke
- VariableCheck

CloudWatchAlarm

When you create a condition, you can add the `CloudWatchAlarm` rule. This section provides a reference for the rule parameters. For more information about rules and conditions, see How do stage conditions work?.

You must have already created an alarm in Amazon CloudWatch as a separate resource.

Topics

- Rule type
- Configuration parameters

- Example rule configuration
- See also

Rule type

- Category: `Rule`
- Owner: `AWS`
- Provider: `CloudWatchAlarm`
- Version: `1`

Configuration parameters

AlarmName

Required: Yes

The name of the CloudWatch alarm. This is a separate resource created in CloudWatch.

AlarmStates

Required: No

The desired CloudWatch alarm states for the rule to monitor. Valid values are ALARM, OK, and INSUFFICIENT_DATA.

WaitTime

Required: No

The wait time in minutes to allow for a state change before running the rule result. For example, configure 20 minutes to wait for an alarm state to change to OK before applying the rule result.

Example rule configuration

YAML

```
rules:
```

```
    - name: MyMonitorRule
      ruleTypeId:
        category: Rule
        owner: AWS
        provider: CloudWatchAlarm
        version: '1'
      configuration:
        AlarmName: CWAlarm
        WaitTime: '1'
      inputArtifacts: []
      region: us-east-1
```

JSON

```
                    "rules": [
                        {
                            "name": "MyMonitorRule",
                            "ruleTypeId": {
                                "category": "Rule",
                                "owner": "AWS",
                                "provider": "CloudWatchAlarm",
                                "version": "1"
                            },
                            "configuration": {
                                "AlarmName": "CWAlarm",
                                "WaitTime": "1"
                            },
                            "inputArtifacts": [],
                            "region": "us-east-1"
                        }
                    ]
                }
```

See also

The following related resources can help you as you work with this rule.

- Creating On Failure conditions – This section provides steps for creating an On Failure condition with an alarm rule.

CodeBuild rule

When you create a condition, you can add the CodeBuild rule. This section provides a reference for the rule parameters. For more information about rules and conditions, see How do stage conditions work?.

You can use the CodeBuild rule to create a condition where the succesful run of your build project meets the rule criteria, such as the build run being successful for a beforeEntry condition.

> **Note**
>
> For beforeEntry conditions that are configured with the **Skip** result, only the following rules are available: `LambdaInvoke` and `VariableCheck`.

Topics

- Service role policy permissions
- Rule type
- Configuration parameters
- Example rule configuration
- See also

Service role policy permissions

For permissions for this rule, add the following to your CodePipeline service role policy statement. Scope down permissions to the resource level.

```
{
    "Effect": "Allow",
    "Action": [
        "codebuild:BatchGetBuilds",
        "codebuild:StartBuild"
    ],
    "Resource": "resource_ARN"
},
```

Rule type

- Category: `Rule`
- 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 buildspec file and runs the buildspec 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).

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 credentials. 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 in the AWS CodeBuild API Reference. For an example CodeBuild action with an environment variable that resolves to the GitHub branch name, see Example: Use a BranchName variable with CodeBuild environment variables.

Example rule configuration

YAML

```
name: codebuild-rule
ruleTypeId:
  category: Rule
  owner: AWS
  provider: CodeBuild
  version: '1'
configuration:
  ProjectName: my-buildproject
  EnvironmentVariables: '[{"name":"VAR1","value":"variable","type":"PLAINTEXT"}]'
inputArtifacts:
- name: SourceArtifact
region: us-east-1
```

JSON

```
{
    "name": "codebuild-rule",
    "ruleTypeId": {
        "category": "Rule",
        "owner": "AWS",
        "provider": "CodeBuild",
        "version": "1"
    },
    "configuration": {
        "ProjectName": "my-buildproject"
    },
    "inputArtifacts": [
        {
            "name": "SourceArtifact",
            "EnvironmentVariables": "[{\"name\":\"VAR1\",\"value\":\"variable\",
\"type\":\"PLAINTEXT\"}]"
        }
    ],
    "region": "us-east-1"
```

```
}
```

See also

The following related resources can help you as you work with this rule.

- For more information about rules and conditions, see Condition, RuleTypeId, and RuleExecution in the *CodePipeline API Guide*.

Commands

When you create a condition, you can add the `Commands` rule. This section provides a reference for the rule parameters. For more information about rules and conditions, see How do stage conditions work?.

You can use the `Commands` rule to create a condition where the succesful commands meet the rule criteria, such as the output and file path for the commands being successful for a beforeEntry condition.

> **Note**
>
> For beforeEntry conditions that are configured with the **Skip** result, only the following rules are available: `LambdaInvoke` and `VariableCheck`.

Topics

- Considerations for the Commands rule
- Service role policy permissions
- Rule type
- Configuration parameters
- Example rule configuration
- See also

Considerations for the Commands rule

The following considerations apply for the Commands rule.

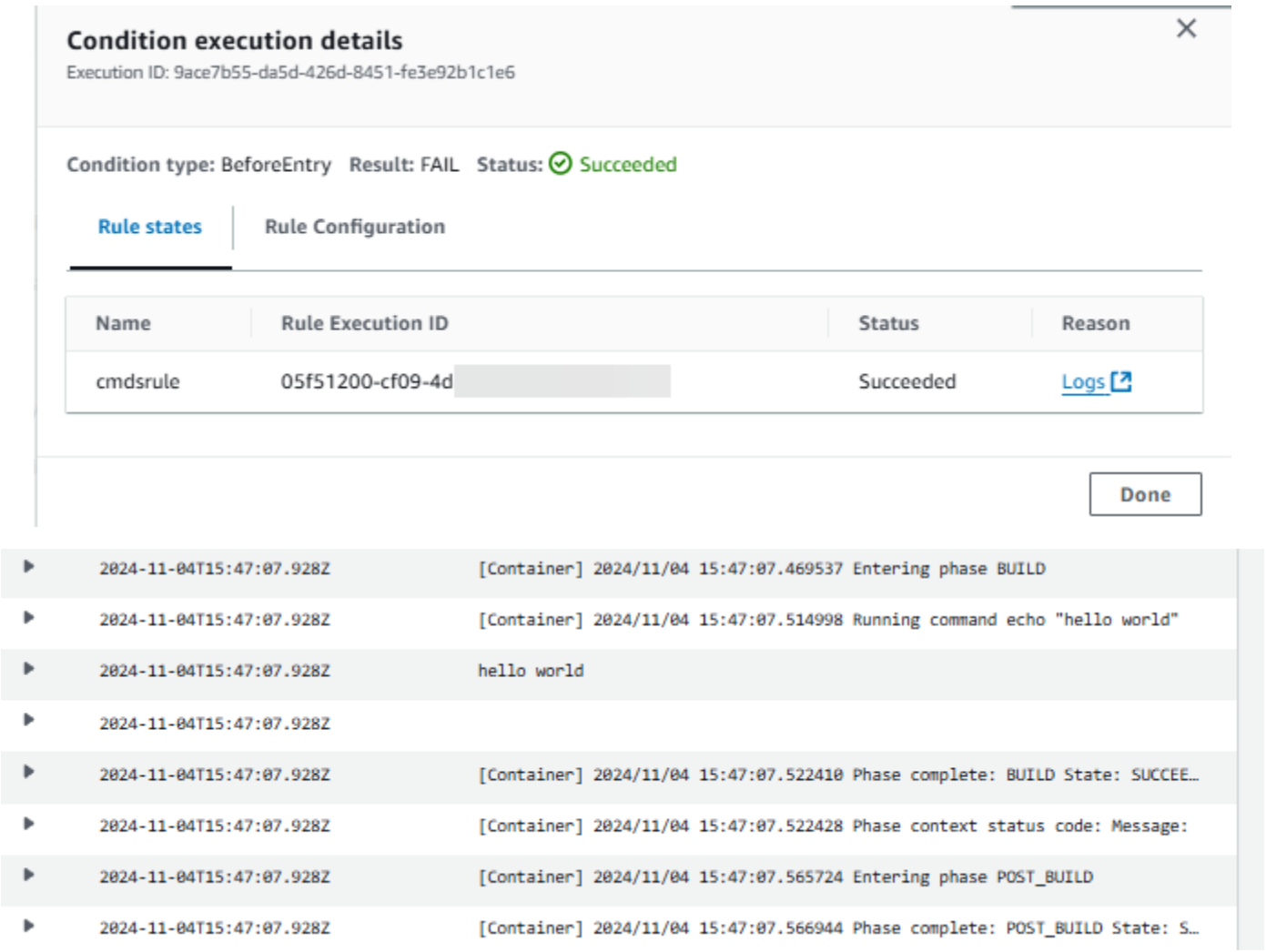
- The commands rule uses CodeBuild resources similar to the CodeBuild action, while allowing shell environment commands in a virtual compute instance without the need to associate or create a build project.

  > **Note**
  >
  > Running the commands rule will incur separate charges in AWS CodeBuild.

- Because the Commands rule in CodePipeline uses CodeBuild resources, the builds run by the action will be attributed to the build limits for your account in CodeBuild. Builds run by the Commands rule will count toward the concurrent build limits as configured for that account.
- The timeout for builds with the Commands rule is 55 minutes, as based on CodeBuild builds.
- The compute instance uses an isolated build environment in CodeBuild.

  > **Note**
  >
  > Because the isolated build environment is used at the account level, an instance might be reused for another pipeline execution.

- All formats are supported except multi-line formats. You must use single-line format when entering commands.
- For this rule, CodePipeline will assume the pipeline service role and use that role to allow access to resources at runtime. It is recommended to configure the service role so that the permissions are scoped down to the action level.
- The permissions added to the CodePipeline service role are detailed in Add permissions to the CodePipeline service role.
- The permission needed to view logs in the console is detailed in Permissions required to view compute logs in the console . In the following example screens, use the **Logs** link to view logs for a successful Commands rule in CloudWatch logs.



- Unlike other actions in CodePipeline, you do not set fields in the action configuration; you set the action configuration fields outside of the action configuration.

Service role policy permissions

When CodePipeline runs the rule, CodePipeline creates a log group using the name of the pipeline as follows. This enables you to scope down permissions to log resources using the pipeline name.

```
/aws/codepipeline/MyPipelineName
```

If you are using an existing service role, to use the Commands action, you will need to add the following permissions for the service role.

- logs:CreateLogGroup

- logs:CreateLogStream
- logs:PutLogEvents

In the service role policy statement, scope down the permissions to the pipeline level as shown in the following example.

```
{
    "Effect": "Allow",
    "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
    ],
    "Resource": "arn:aws:logs:*:YOUR_AWS_ACCOUNT_ID:log-group:/aws/
codepipeline/YOUR_PIPELINE_NAME:*"
}
```

To view logs in the console using the action details dialog page, the permission to view logs must be added to the console role. For more information, see the console permissions policy example in Permissions required to view compute logs in the console.

Rule type

- Category: `Rule`
- Owner: `AWS`
- Provider: `Commands`
- Version: 1

Configuration parameters

Commands

Required: Yes

You can provide shell commands for the `Commands` rule to run. In the console, commands are entered on separate lines. In the CLI, commands are entered as separate strings.

> **Note**
>
> Multi-line formats are not supported and will result in an error message. Single-line format must be used for entering commands in the **Commands** field.

The following details provide the default compute that is used for the Commands rule. For more information, see Build environment compute modes and types reference in the CodeBuild User Guide.

- **CodeBuild image:** aws/codebuild/amazonlinux2-x86_64-standard:5.0
- **Compute type:** Linux Small
- **Environment computeType value:** BUILD_GENERAL1_SMALL
- **Environment type value:** LINUX_CONTAINER

Example rule configuration

YAML

```
result: FAIL
rules:
- name: CommandsRule
  ruleTypeId:
    category: Rule
    owner: AWS
    provider: Commands
    version: '1'
  configuration: {}
  commands:
  - ls
  - printenv
  inputArtifacts:
  - name: SourceArtifact
  region: us-east-1
```

JSON

```
{
    "result": "FAIL",
    "rules": [
```

```
        {
            "name": "CommandsRule",
            "ruleTypeId": {
                "category": "Rule",
                "owner": "AWS",
                "provider": "Commands",
                "version": "1"
            },
            "configuration": {},
            "commands": [
                "ls",
                "printenv"
            ],
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-east-1"
        }
    ]
}
```

See also

The following related resources can help you as you work with this rule.

- For more information about rules and conditions, see Condition, RuleTypeId, and RuleExecution in the *CodePipeline API Guide*.

DeploymentWindow

When you create a condition, you can add the `DeploymentWindow` rule. This section provides a reference for the rule parameters. For more information about rules and conditions, see How do stage conditions work?.

Topics

- Rule type
- Configuration parameters

- Example rule configuration
- See also

Rule type

- Category: `Rule`
- Owner: `AWS`
- Provider: `DeploymentWindow`
- Version: `1`

Configuration parameters

Cron

Required: Yes

The expression that defines the days and times when the deployment will be allowed. Cron expressions are comprised of 6 required fields and one optional field separated by white space. The cron expression fields allow you to specify a schedule pattern with a cron expression as follows.

Field name	Allowed values	Allowed special characters
Seconds	N/A	*
Minutes	0-59	, - * /
Hours	0-23	, - * /
Day-of-month	1-31	, - * ? / L W
Month	1-12 or JAN-DEC	, - * /
Day-of-Week	1-7 or SUN-SAT	, - * ? / L #
Year (Optional)	empty, 1970-2199	, - * /

- The '*' character is used to specify all values. For example, "*" in the minute field means "every minute".
- The '?' character is allowed for the day-of-month and day-of-week fields. It is used to specify 'no specific value'. This is useful when you need to specify something in one of the two fields, but not the other.
- The '-' character is used to specify ranges. For example, "10-12" in the hour field means "the hours 10, 11 and 12".
- The ',' character is used to specify additional values. For example "MON,WED,FRI" in the day-of-week field means "the days Monday, Wednesday, and Friday".
- The '/' character is used to specify increments. For example "0/15" in the seconds field means "the seconds 0, 15, 30, and 45". And "5/15" in the seconds field means "the seconds 5, 20, 35, and 50". Specifying '*' before the '/' is equivalent to specifying 0 is the value to start with.
- The 'L' character is allowed for the day-of-month and day-of-week fields. This character is short-hand for "last", but it has different meaning in each of the two fields. For example, the value "L" in the day-of-month field means "the last day of the month" - day 31 for January, day 28 for February on non-leap years. If used in the day-of-week field by itself, it simply means "7" or "SAT". But if used in the day-of-week field after another value, it means "the last <specified_day> day of the month" - for example "6L" means "the last friday of the month". You can also specify an offset from the last day of the month, such as "L-3" which would mean the third-to-last day of the calendar month.
- The 'W' character is allowed for the day-of-month field. This character is used to specify the weekday (Monday-Friday) nearest the given day. As an example, if you were to specify "15W" as the value for the day-of-month field, the meaning is: "the nearest weekday to the 15th of the month". So if the 15th is a Saturday, the trigger will fire on Friday the 14th. If the 15th is a Sunday, the trigger will fire on Monday the 16th. If the 15th is a Tuesday, then it will fire on Tuesday the 15th.
- The 'L' and 'W' characters can also be combined for the day-of-month expression to yield 'LW', which translates to "last weekday of the month".
- The '#' character is allowed for the day-of-week field. This character is used to specify "the nth" <specified_day> day of the month. For example, the value of "6#3" in the day-of-week field means the third Friday of the month (day 6 = Friday and "#3" = the 3rd one in the month).
- The legal characters and the names of months and days of the week are not case sensitive.

TimeZone

Required: No

The time zone for the deployment window. The regular expression matches patterns in the following formats:

- **Region/City format**. The value matches a time zone in the format Region/City or Region/City_City. For example, `America/New_York` or `Europe/Berlin`.
- **UTC format**. The value matches the string UTC optionally followed by an offset in the format +HH:MM or -HH:MM. For example, UTC, `UTC+05:30`, or `UTC-03:00`. This is the default format if the parameter is not set otherwise.
- **Abbreviation format**. The value matches a 3 to 5 character abbreviation for a time zone. For example, `EST` or `IST`.

  For a table of valid TimeZoneID values, see https://docs.oracle.com/middleware/1221/wcs/tag-ref/MISC/TimeZones.html. Note that certain abbreviations are duplicated abbreviations, such as CST for Central Standard Time, China Standard Time, and Cuba Standard Time.

Example rule configuration

YAML

```
- name: MyDeploymentRule
  ruleTypeId:
    category: Rule
    owner: AWS
    provider: DeploymentWindow
    version: '1'
  configuration:
    Cron: 0 0 9-17 ? * MON-FRI *
    TimeZone: PST
  inputArtifacts: []
  region: us-east-1
```

JSON

```
[
                    {
                        "name": "MyDeploymentRule",
```

```
                    "ruleTypeId": {
                        "category": "Rule",
                        "owner": "AWS",
                        "provider": "DeploymentWindow",
                        "version": "1"
                    },
                    "configuration": {
                        "Cron": "0 0 9-17 ? * MON-FRI *",
                        "TimeZone": "PST"
                    },
                    "inputArtifacts": [],
                    "region": "us-east-1"
                }
            ]
```

See also

The following related resources can help you as you work with this rule.

- Creating On Success conditions – This section provides steps for creating an On Success condition with a deployment window rule.
- For more information about rules and conditions, see Condition, RuleTypeId, and RuleExecution in the *CodePipeline API Guide*.

LambdaInvoke

When you create a condition, you can add the `LambdaInvoke` rule. This section provides a reference for the rule parameters. For more information about rules and conditions, see How do stage conditions work?.

You must have already created a function in Lambda as a separate resource.

Topics

- Rule type
- Configuration parameters
- Example rule configuration
- See also

Rule type

- Category: `Rule`
- Owner: `AWS`
- Provider: `LambdaInvoke`
- Version: 1

Configuration parameters

FunctionName

Required: Yes

The name of the Lambda function.

UserParameters

Required: No

These are parameters that are provided as input for the function in key-value pair format.

Example rule configuration

YAML

```
- name: MyLambdaRule
  ruleTypeId:
    category: Rule
    owner: AWS
    provider: LambdaInvoke
    version: '1'
  configuration:
    FunctionName: my-function
  inputArtifacts:
  - name: SourceArtifact
  region: us-east-1
```

JSON

```
[
```

```
    {
        "name": "MyLambdaRule",
        "ruleTypeId": {
            "category": "Rule",
            "owner": "AWS",
            "provider": "LambdaInvoke",
            "version": "1"
        },
        "configuration": {
            "FunctionName": "my-function"
        },
        "inputArtifacts": [
            {
                "name": "SourceArtifact"
            }
        ],
        "region": "us-east-1"
    }
]
```

See also

The following related resources can help you as you work with this rule.

- Creating On Failure conditions – This section provides steps for creating an On Failure condition with an alarm rule.

VariableCheck

When you create a condition, you can add the `VariableCheck` rule. This section provides a reference for the rule parameters. For more information about rules and conditions, see How do stage conditions work?.

You can use the `VariableCheck` rule to create a condition where the output variable is checked against a provided expression. The rule passes the check when the variable value meets the rule criteria, such as the value being equal or greater than a specified output variable.

Topics

- Rule type

- Configuration parameters
- Example rule configuration
- See also

Rule type

- Category: `Rule`
- Owner: `AWS`
- Provider: `VariableCheck`
- Version: `1`

Configuration parameters

Operator

Required: Yes

The operator that indicates which operation to perform for the variable check.

In the following example, the output variable for the repository name will be checked for whether it is equal to `MyDemoRepo`.

```
"configuration": {
    "Variable": "#{SourceVariables.RepositoryName}",
    "Value": "MyDemoRepo",
    "Operator": "EQ"
},
```

The following operators are available to create an expression as follows.

- **Equals** - Choose this operator to check whether the variable is equal to the string value.

  CLI parameter: `EQ`

- **Contains** - Choose this operator to check whether the variable contains the string value as a substring.

  CLI parameter: `CONTAINS`

- **Matches** - Choose this operator to check whether the variable matches a given regex expression as the string value.

  All regular expressions in CodePipeline conform to the Java regex syntax. For a comprehensive description of the Java regex syntax and its constructs, see java.util.regex.Pattern.

  CLI parameter: `MATCHES`
- **Not equals** - Choose this operator to check whether the variable is not equal to the string value.

  CLI parameter: `NE`

Variable

Required: Yes

The pipeline variables to check.

Value

Required: Yes

The value for the expression to check against.

In the following example, the output variable for the repository name will be checked for whether it is equal to `MyDemoRepo`.

```
        "configuration": {
            "Variable": "#{SourceVariables.RepositoryName}",
            "Value": "MyDemoRepo",
            "Operator": "EQ"
        },
```

In the following JSON example, two separate rules are defined, one for an `EQ` (equals) statement that checks the repository and branch name formatted as #{SourceVariables.RepositoryName} and one for `CONTAINS` that checks the commit message output variable formatted as #{SourceVariables.CommitMessage} against the provided value "update."

```
"beforeEntry": {
                "conditions": [
                    {
```

```
                        "result": "FAIL",
                        "rules": [
                            {
                                "name": "MyVarCheckRule",
                                "ruleTypeId": {
                                    "category": "Rule",
                                    "owner": "AWS",
                                    "provider": "VariableCheck",
                                    "version": "1"
                                },
                                "configuration": {
                                    "Operator": "EQ",
                                    "Value": "MyDemoRepo",
                                    "Variable": "#{SourceVariables.RepositoryName}"
                                },
                                "inputArtifacts": [],
                                "region": "us-east-1"
                            },
                            {
                                "name": "MyVarCheckRuleContains",
                                "ruleTypeId": {
                                    "category": "Rule",
                                    "owner": "AWS",
                                    "provider": "VariableCheck",
                                    "version": "1"
                                },
                                "configuration": {
                                    "Operator": "CONTAINS",
                                    "Value": "update",
                                    "Variable": "#{SourceVariables.CommitMessage}"
                                },
                                "inputArtifacts": [],
                                "region": "us-east-1"
                            }
                        ]
                    }
                ]
            }
        }
    ],
```

Example rule configuration

YAML

```
- name: MyVariableCheck
  ruleTypeId:
    category: Rule
    owner: AWS
    provider: VariableCheck
    version: '1'
  configuration:
    Variable: "#{SourceVariables.RepositoryName}"
    Value: MyDemoRepo
    Operator: EQ
  inputArtifacts: []
  region: us-west-2
```

JSON

```
"rules": [
    {
        "name": "MyVariableCheck",
        "ruleTypeId": {
            "category": "Rule",
            "owner": "AWS",
            "provider": "VariableCheck",
            "version": "1"
        },
        "configuration": {
            "Variable": "#{SourceVariables.RepositoryName}",
            "Value": "MyDemoRepo",
            "Operator": "EQ"
        },
        "inputArtifacts": [],
        "region": "us-west-2"
    }
]
```

See also

The following related resources can help you as you work with this rule.

- Tutorial: Create a variable check rule for a pipeline as an entry condition – This section provides a tutorial with steps for creating an On Entry condition with a variable check rule.
- Variables reference – This section provides reference information and examples for pipeline variables.
- For more information about rules and conditions, see Condition, RuleTypeId, and RuleExecution in the *CodePipeline API Guide*.

Integration model reference

There are several pre-built integrations for third-party services to help build existing customer tools into the pipeline release process. Partners, or third-party service providers, use an integration model to implement action types for use in CodePipeline.

Use this reference when you are planning or working with action types that are managed with a supported integration model in CodePipeline.

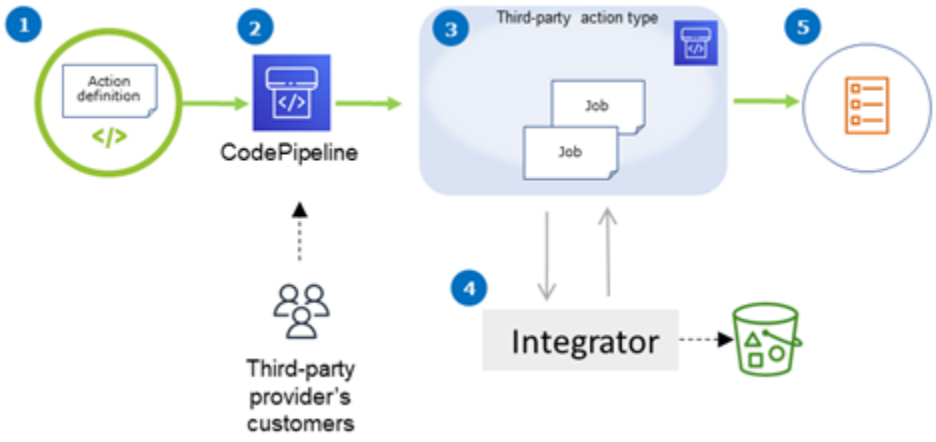
To certify your third-party action type as a partner integration with CodePipeline, reference the AWS Partner Network (APN). This information is a supplement to the AWS CLI Reference.

Topics

- How third-party action types work with the integrator
- Concepts
- Supported integration models
- Lambda integration model
- Job worker integration model

How third-party action types work with the integrator

You can add third-party action types to customer pipelines to complete tasks on customer resources. The integrator manages job requests and runs the action with CodePipeline. The following diagram shows a third-party action type created for customers to use in their pipeline. After the customer configures the action, the action runs and creates job requests that are handled by the integrator's action engine.



The diagram shows the following steps:

1. The action definition is registered and made available in CodePipeline. The third-party action is available for customers of the third-party provider.
2. The provider's customer chooses and configures the action in CodePipeline.
3. The action runs and jobs are queued in CodePipeline. When the job is ready in CodePipeline, it sends a job request.
4. The integrator (the job worker for third-party polling APIs or the Lambda function) picks up the job request, returns a confirmation, and works on the artifacts for the actions.
5. The integrator returns success/failure output (the job worker uses success/failure APIs or the Lambda function sends success/failure output) with a job result and a continuation token.

For information about the steps you can use to request, view, and update an action type, see Working with action types.

Concepts

This section uses the following terms for third-party action types:

Action type

A repeatable process that can be re-used in pipelines that performs the same continuous delivery workloads. Action types are identified by an `Owner`, `Category`, `Provider`, and `Version`. For example:

```
{

    "Category": "Deploy",
    "Owner": "AWS",
    "Provider": "CodeDeploy",
    "Version": "1"
},
```

All actions of the same type share the same implementation.

Action

A single instance of an action type, one of the discrete processes that happens inside a stage of a pipeline. This typically includes the user values specific to the pipeline that this action runs in.

Action definition

The schema for an action type that defines the properties required to configure the action and input/output artifacts.

Action execution

A collection of jobs that have been run to determine whether the action on the customer's pipeline was successful or not.

Action execution engine

A property of the action execution configuration that defines the integration type used by an action type. Valid values are `JobWorker` and `Lambda`.

Integration

Describes a piece of software run by an integrator to implement an action type. CodePipeline supports two integration types corresponding to the two supported action engines `JobWorker` and `Lambda`.

Integrator

The person who owns the implementation of an action type.

Job

A piece of work with pipeline and customer context to execute an integration. An action execution is composed of one or more jobs.

Job worker

The service that processes the customer input and runs a job.

Supported integration models

CodePipeline has two integration models:

- **Lambda integration model:** This integration model is the preferred way to work with action types in CodePipeline. The Lambda integration model uses a Lambda function to process job requests when your action runs.

- **Job worker integration model:** The job worker integration model is the previously used model for third-party integrations. The job worker integration model uses a job worker configured to contact the CodePipeline APIs to process job requests when your action runs.

For comparison, the following table describes the features of the two models:

	Lambda integration model	Job worker integration model
Description	The integrator writes the integration as a Lambda function, which is invoked by CodePipeline whenever there is a job available for the action. The Lambda function does not poll for available jobs but instead waits until the next job request is received.	The integrator writes the integration as a job worker that polls constantly for available jobs on the customer's pipelines. The job worker then executes the job and submits the job result back to CodePipeline by using CodePipeline APIs.
Infrastructure	AWS Lambda	Deploy job worker code to integrator's infrastructure, like Amazon EC2 instances.
Development effort	The integration only contains the business logic.	The integration needs to interact with CodePipeline APIs in addition to containing the business logic.
Ops effort	Lesser ops effort since infrastructure is only AWS resources.	Higher ops effort because the job worker needs its standalone hardware.
Max Job Run Time	If the integration needs to actively run for more than 15 minutes, this model cannot be used. This action is for integrators who need to start a process (for example, initiate a build on customer's code artifact) and	Very long running jobs (hours/days) can be sustained using this model.

	Lambda integration model	Job worker integration model
	return a result when it finishes. We do not recommend that the integrator continue waiting on the build to finish. Instead, return a *continuation*. CodePipeline creates a new job in another 30 seconds if a continuation is received from the integrator's code to check on the job until it finishes.	

Lambda integration model

The supported Lambda integration model includes creating the Lambda function and defining output for the third-party action type.

Update your Lambda function to handle the input from CodePipeline

You can create a new Lambda function. You can add business logic to your Lambda function that is run whenever there's a job available on your pipeline for your action type. For instance, given the context of the customer and pipeline, you might want to start a build in your service for the customer.

Use the following parameters to update your Lambda function to handle the input from CodePipeline.

Format:

- `jobId`:
  - The unique system-generated ID of the job.
  - Type: String
  - Pattern: [0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}
- `accountId`:
  - The ID of the customer's AWS account to use when performing the job.
  - Type: String

- Pattern: [0-9]{12}

- `data`:
  - Other information about a job that an integration uses to complete the job.
  - Contains a map of the following:
    - `actionConfiguration`:
      - The configuration data for the action. The action configuration fields are a mapping of key-value pairs for your customer to enter values. The keys are determined by the key parameters in the action type definition file when you set up your action. In this example, the values are determined by the user of the action specifying information in the `Username` and `Password` fields.
      - Type: String to string map, optionally present

        Example:

        ```
        "configuration": {
            "Username": "MyUser",
            "Password": "MyPassword"
        },
        ```
    - `encryptionKey`:
      - Represents information about the key used to encrypt data in the artifact store, such as an AWS KMS key.
      - Contents: Type of the data type `encryptionKey`, optionally present
    - `inputArtifacts`:
      - List of information about an artifact to be worked on, such as a test or build artifact.
      - Contents: List of the data type `Artifact`, optionally present
    - `outputArtifacts`:
      - List of information about the output of an action.
      - Contents: List of the data type `Artifact`, optionally present
    - `actionCredentials`:
      - Represents an AWS session credentials object. These credentials are temporary credentials that are issued by AWS STS. They can be used to access input and output artifacts in the S3 bucket used to store artifacts for the pipeline in CodePipeline.

These credentials also have the same permissions as the specified policy statements template in the action type definition file.

- Contents: Type of the data type `AWSSessionCredentials`, optionally present

- `actionExecutionId`:
  - The external ID of the run of the action.
  - Type: String
- `continuationToken`:
  - A system-generated token, such as a deployment ID, required by a job to continue the job asynchronously.
  - Type: String, optionally present

Data Types:

- `encryptionKey`:
  - `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: String
  - `type`:
    - The type of encryption key, such as an AWS KMS key.
    - Type: String
    - Valid values: `KMS`
- `Artifact`:
  - `name`:
    - The artifact's name.
    - Type: String, optionally present
  - `revision`:
    - The artifact's revision ID. Depending on the type of object, this could be a commit ID (GitHub) or a revision ID (Amazon S3).
    - Type: String, optionally present
  - `location`:

    - The location of an artifact.
    - Contents: Type of the data type `ArtifactLocation`, optionally present
- `ArtifactLocation`:
  - `type`:
    - The type of artifact in the location.
    - Type: String, optionally present
    - Valid values: `S3`
  - `s3Location`:
    - The location of the S3 bucket that contains a revision.
    - Contents: Type of the data type `S3Location`, optionally present
- `S3Location`:
  - `bucketName`:
    - The name of the S3 bucket.
    - Type: String
  - `objectKey`:
    - The key of the object in the S3 bucket, which uniquely identifies the object in the bucket.
    - Type: String
- `AWSSessionCredentials`:
  - `accessKeyId`:
    - The access key for the session.
    - Type: String
  - `secretAccessKey`:
    - The secret access key for the session.
    - Type: String
  - `sessionToken`:
    - The token for the session.
    - Type: String

Example:

```
{
```

```
    "jobId": "01234567-abcd-abcd-abcd-012345678910",
    "accountId": "012345678910",
    "data": {
        "actionConfiguration": {
            "key1": "value1",
            "key2": "value2"
        },
        "encryptionKey": {
            "id": "123-abc",
            "type": "KMS"
        },
        "inputArtifacts": [
            {
                "name": "input-art-name",
                "location": {
                    "type": "S3",
                    "s3Location": {
                        "bucketName": "inputBucket",
                        "objectKey": "inputKey"
                    }
                }
            }
        ],
        "outputArtifacts": [
            {
                "name": "output-art-name",
                "location": {
                    "type": "S3",
                    "s3Location": {
                        "bucketName": "outputBucket",
                        "objectKey": "outputKey"
                    }
                }
            }
        ],
        "actionExecutionId": "actionExecutionId",
        "actionCredentials": {
            "accessKeyId": "access-id",
            "secretAccessKey": "secret-id",
            "sessionToken": "session-id"
        },
        "continuationToken": "continueId-xxyyzz"
    }
```

```
}
```

Return the results from your Lambda function to CodePipeline

The integrator's job worker resource must return a valid payload in success, failure, or continuation cases.

Format:

- `result`: The result of the job.
  - Required
  - Valid values (case insensitive):
    - `Success`: Indicates a job is successful and terminal.
    - `Continue`: Indicates a job is successful and must continue, for example if the job worker is reinvoked for the same action execution.
    - `Fail`: Indicates a job has failed and is terminal.
- `failureType`: A failure type to be associated with a failed job.

  The `failureType` category for partner actions describes the type of failure that was encountered while running the job. Integrators set the type along with the failure message when returning a job failure result back to CodePipeline.
  - Optional. Required if result is `Fail`.
  - Must be null if `result` is `Success` or `Continue`
  - Valid values:
    - ConfigurationError
    - JobFailed
    - PermissionsError
    - RevisionOutOfSync
    - RevisionUnavailable
    - SystemUnavailable
- `continuation`: Continuation state to be passed to the next job within the current action execution.
  - Optional. Required if result is `Continue`.
  - Must be null if `result` is `Success` or `Fail`.

- Properties:
  - `State`: A hash of the state to be passed.
- `status`: Status of the action execution.
  - Optional.
  - Properties:
    - `ExternalExecutionId`: An optional external execution ID or commit ID to associate with the job.
    - `Summary`: An optional summary of what occurred. In failure scenarios, this becomes the failure message that the user sees.
- `outputVariables`: A set of key/value pairs to be passed to the next action execution.
  - Optional.
  - Must be null if `result` is `Continue` or `Fail`.

Example:

```
{
    "result": "success",
    "failureType": null,
    "continuation": null,
    "status": {
        "externalExecutionId": "my-commit-id-123",
        "summary": "everything is dandy"
    },
    "outputVariables": {
        "FirstOne": "Nice",
        "SecondOne": "Nicest",
        ...
    }
}
```

Use continuation tokens to wait for results from an asynchronous process

The `continuation` token is part of the payload and result of your Lambda function. It is a way to pass job state to CodePipeline and indicate that the job needs to be continued. For example, after an integrator starts a build for the customer on their resource, it does not wait for the build

to complete, but indicates to CodePipeline that it does not have a terminal result by returning the `result` as `continue` and returning the build's unique ID to CodePipeline as `continuation` token.

> **Note**
>
> Lambda functions can only run up to 15 minutes. If the job needs to run longer, you can use continuation tokens.

The CodePipeline team invokes the integrator after 30 seconds with the same `continuation` token in its payload so that it can check on it for completion. If the build completes, the integrator returns terminal success/fail result, else continues.

Provide CodePipeline the permissions to invoke the integrator Lambda function at runtime

You add permissions to your integrator Lambda function to provide the CodePipeline service with permissions to invoke it using the CodePipeline service principal: `codepipeline.amazonaws.com`. You can add permissions by using CloudFormation or the command line. For an example, see Working with action types.

Job worker integration model

After you have designed your high-level workflow, you can create your job worker. Although the specifics of the third-party action determine what is needed for the job worker, most job workers for third-party actions include the following functionality:

- Polling for jobs from CodePipeline using `PollForThirdPartyJobs`.
- Acknowledging jobs and returning results to CodePipeline using `AcknowledgeThirdPartyJob`, `PutThirdPartyJobSuccessResult`, and `PutThirdPartyJobFailureResult`.
- 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 AWS KMS.

  To upload artifacts to the Amazon S3 bucket, you must also configure the Amazon S3 `PutObject` request to use encryption through AWS Key Management Service (AWS KMS).

AWS KMS uses AWS KMS keys. In order to know whether to use the AWS managed key or a customer managed key to upload artifacts, your job worker must look at the job data and check the encryption key property. If the property is set, you should use that customer managed key ID when configuring AWS KMS. If the key property is null, you use the AWS managed key. CodePipeline uses the AWS managed key unless otherwise configured.

For an example that shows how to create the AWS KMS parameters in Java or .NET, 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.

Choose and configure a permissions management strategy for your job worker

To develop a job worker for your third-party action in CodePipeline, you need a strategy for the integration of user and permission management.

The simplest strategy is to add the infrastructure you need for your job worker by creating Amazon EC2 instances with an AWS Identity and Access Management (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 job worker and CodePipeline.

Learn more about Amazon EC2 and determine whether it is the right choice for your integration. For information, see Amazon EC2 - Virtual Server Hosting. For information about setting up an Amazon EC2 instance, 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 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. For information, see Manage Federation.

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

  Create a permission set. Follow the instructions in Create a permission set in the *AWS IAM Identity Center User Guide*.

- Users managed in IAM through an identity provider:

  Create a role for identity federation. Follow the instructions in Create a role for a third-party identity provider (federation) in the *IAM User Guide*.
- IAM users:
  - Create a role that your user can assume. Follow the instructions in Create a role for an IAM user in the *IAM User Guide*.
  - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in Adding permissions to a user (console) in the *IAM User Guide*.

The following is an example policy you might create for use with your third-party 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:PollForThirdPartyJobs",
        "codepipeline:AcknowledgeThirdPartyJob",
        "codepipeline:GetThirdPartyJobDetails",
        "codepipeline:PutThirdPartyJobSuccessResult",
        "codepipeline:PutThirdPartyJobFailureResult"
      ],
      "Resource": [
        "arn:aws-eusc:codepipeline:us-east-2::actionType:ThirdParty/
Build/Provider/1/"
      ]
    }
  ]
}
```

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, stages, and actions.

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. For a tutorial that uses the Amazon ECS standard deployment action in CodePipeline, see Tutorial: Amazon ECS Standard Deployment with CodePipeline. For another example tutorial that uses the Amazon ECS standard deployment action in CodePipeline along with the ECRBuildAndPublish action, see Tutorial: Build and push a Docker image to Amazon ECR with CodePipeline (V2 type).
- Amazon ECS blue/green deployments require an `imageDetail.json` file as an input to the deploy action. For a tutorial with a sample blue/green deployment, see Tutorial: Create a pipeline with an Amazon ECR source and ECS-to-CodeDeploy deployment.
  - 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
- imageDetail.json file for Amazon ECS blue/green deployment actions

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 image repository, such as Amazon ECR.

> **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.

Key	Value
name	*container_name*
imageUri	*imageUri*

> **Note**
>
> The name field is used for the container image name, meaning the name for the Docker image.

Here is the JSON structure, where the container name is `sample-app`, the image URI is `ecs-repo`, and the tag is `latest`:

```
[
  {
    "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:

```
[
  {
    "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 is created so that it skips 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 buildspec 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 image repository, such as Amazon ECR.

> **Note**
>
> The name of the file must be `imageDetail.json`.

For a description of the action and its parameters, see Amazon Elastic Container Service and CodeDeploy blue-green deploy action reference.

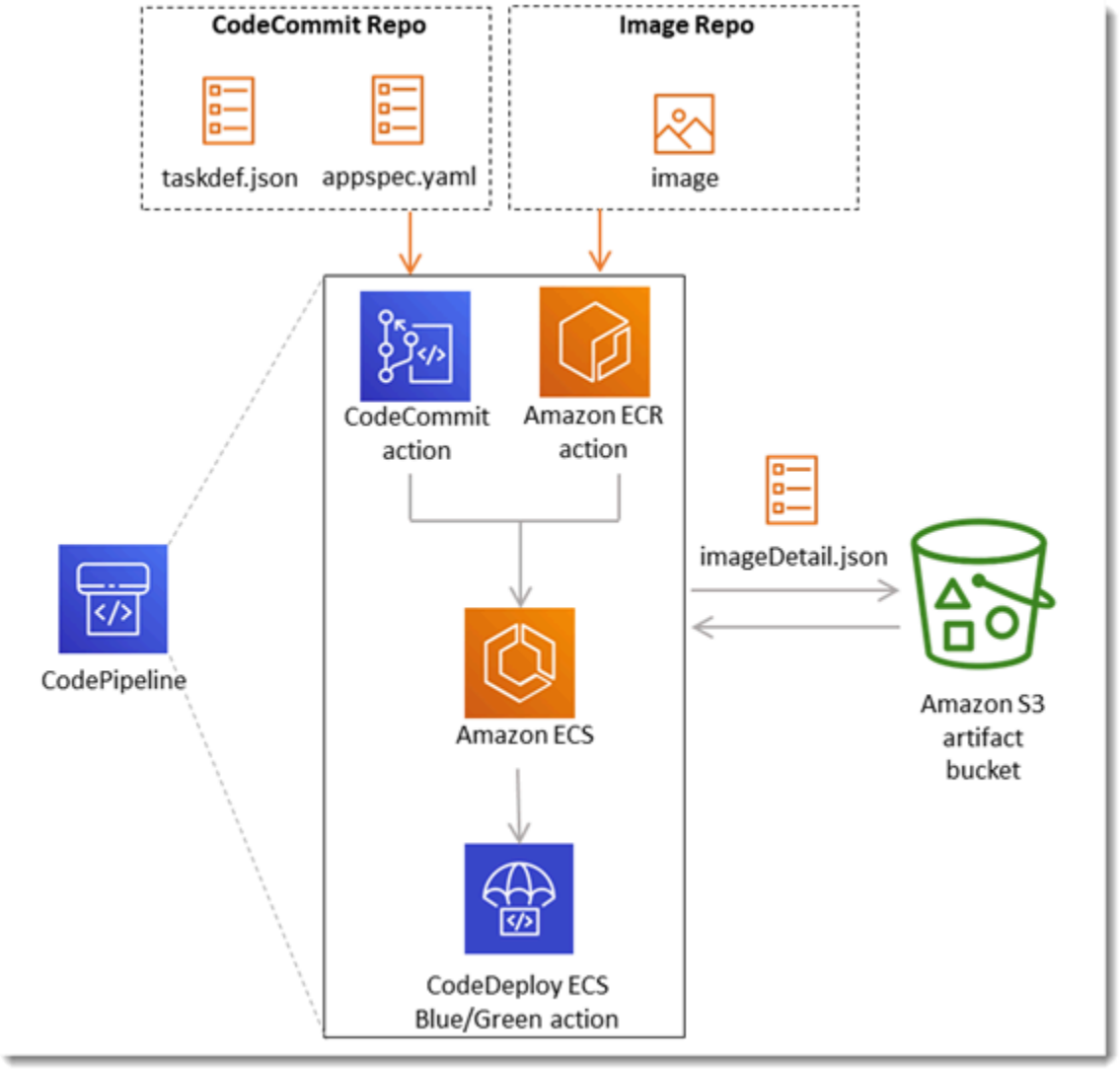
You must create the `imageDetail.json` file as a source or build artifact so that it is an input artifact for the deploy action. You can use one of these methods to provide the `imageDetail.json` file in the pipeline:

- 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.



The `imageDetail.json` file provides the image URI. It must be constructed with the following key-value pair.

Key	Value
ImageURI	*`image_URI`*

imageDetail.json

Here is the JSON structure, where the image URI is `ACCOUNTID.dkr.ecr.us-west-2.amazonaws.com/dk-image-repo@sha256:example3`:

```
{
```

```
"ImageURI": "ACCOUNTID.dkr.ecr.us-west-2.amazonaws.com/dk-image-
repo@sha256:example3"
}
```

imageDetail.json (generated by ECR)

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`:

```
{
    "ImageSizeInBytes": "44728918",
    "ImageDigest":
 "sha256:EXAMPLE1122334455667788990bfea42ea2d3b8a1ee8329ba7e68694950afd3",
    "Version": "1.0",
    "ImagePushedAt": "Mon Jan 21 20:04:00 UTC 2019",
    "RegistryId": "EXAMPLE12233",
    "RepositoryName": "dk-image-repo",
    "ImageURI": "ACCOUNTID.dkr.ecr.us-west-2.amazonaws.com/dk-image-
repo@sha256:example3",
    "ImageTags": [
        "latest"
    ]
}
```

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 buildspec 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:

      ```
      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.

      ```
      artifacts:
        files:
          - imageDetail.json
      ```

Variables reference

This section is a reference only. For information about creating variables, see Working with variables.

Variables allow you to configure your pipeline actions with values that are determined at the time of the pipeline execution or the action execution.

Some action providers produce a defined set of variables. You choose from default variable keys for that action provider, such as commit ID.

> **Important**
>
> When passing secret parameters, do not enter the value directly. The value is rendered as plaintext and is therefore readable. For security reasons, do not use plaintext with secrets. We strongly recommend that you use AWS Secrets Manager to store secrets.

To see step-by-step examples of using variables:

- For a tutorial with a pipeline-level variable that is passed at the time of the pipeline execution, see Tutorial: Use pipeline-level variables.
- For a tutorial with a Lambda action that uses variables from an upstream action (CodeCommit) and generates output variables, see Tutorial: Using variables with Lambda invoke actions.
- For a tutorial with a CloudFormation action that references stack output variables from an upstream CloudFormation action, see Tutorial: Create a pipeline that uses variables from AWS CloudFormation deployment actions.
- For an example manual approval action with message text that references output variables that resolve to the CodeCommit commit ID and commit message, see Example: Use variables in manual approvals.
- For an example CodeBuild action with an environment variable that resolves to the GitHub branch name, see Example: Use a BranchName variable with CodeBuild environment variables.
- CodeBuild actions produce as variables all environment variables that were exported as part of the build. For more information, see CodeBuild action output variables.

Variable Limits

For limit information, see Quotas in AWS CodePipeline.

> **Note**
>
> When you enter 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
- Use cases for variables
- Configuring variables
- Variable resolution
- Rules for variables
- Variables available for pipeline actions

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 three 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.
- Variables at the pipeline level are defined when the pipeline is created and resolved at pipeline run time.

  You specify pipeline-level variables when the pipeline is created, and you can provide values at the time of the 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. For a list of available key names by action provider, see Variables

available for pipeline actions. To see which variables each action type produces, see the CodePipeline Action structure reference.

To reference these variables in your action configuration, you must use the variable reference syntax with the correct namespace.

For an example variable workflow, see Configuring variables.

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 three 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}
  ```

- **The variables namespace at the pipeline level**

  This is the namespace assigned to variables at the pipeline level. The namespace for all variables at the pipeline level is `variables`. Example variable reference:

  ```
  #{variables.variable_name}
  ```

- **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}
```

Use cases for variables

The following are a few of the most common use cases for variables at the pipeline level, helping you determine how you might use variables for your specific needs.

- Variables at the pipeline level are for CodePipeline customers who want to use the same pipeline each time with minor variations in the inputs to the action configuration. Any developer who starts a pipeline adds the variable value in the UI when the pipeline starts. With this configuration, you pass parameters for that execution only.
- With pipeline-level variables, you can pass dynamic inputs to actions in the pipeline. You can migrate your parameterized pipelines to CodePipeline without having to maintain different versions of the same pipeline, or create complex pipelines.
- You can use pipeline-level variables to pass input parameters that allow you to re-use a pipeline with each execution, such as when you want to specify which version you want to deploy to a production environment, so you don't have to duplicate pipelines.
- You can use a single pipeline to deploy resources to multiple build and deployment environments. For example, for a pipeline with a CodeCommit repository, deploying from a specified branch and target deployment environment can be done with CodeBuild and CodeDeploy parameters passed at the pipeline level.

Note

For Amazon ECR, Amazon S3, or CodeCommit sources, you can also create a source override using input transform entry to use the `revisionValue` in EventBridge for your pipeline event, where the `revisionValue` is derived from the source event variable for your object key, commit, or image ID. For more information, see the optional step for input transform entry included in the procedures under Amazon ECR source actions and EventBridge resources, Connecting to Amazon S3 source actions with a source enabled for events, or CodeCommit source actions and EventBridge.

Configuring variables

You can configure variables at the pipeline level or the action level in the pipeline structure.

Configuring variables at the pipeline level

You can add one or more variables at the pipeline level. You can reference this value in the configuration of CodePipeline actions. You can add the variable names, default values, and descriptions when you create the pipeline. Variables are resolved at the time of execution.

> **Note**
>
> If a default value is not defined for a variable at pipeline level, the variable is considered as required. You have to specify overrides for all required variables when you are starting a pipeline, otherwise the pipeline execution will fail with a validation error.

You provide variables at the pipeline level using the variables attribute in the pipeline structure. In the following example, the variable `Variable1` has a value of `Value1`.

```
"variables": [
     {
         "name": "Variable1",
         "defaultValue": "Value1",
         "description": "description"
     }
 ]
```

For an example in the pipeline JSON structure, see Create a pipeline, stages, and actions.

For a tutorial with a pipeline-level variable that is passed at the time of the pipeline execution, see Tutorial: Use pipeline-level variables.

Note that using pipeline-level variables in any kind of Source action is not supported.

> **Note**
>
> If the `variables` namespace is already used in some of actions within the pipeline, you must update the action definition and choose another namespace for the conflicting action.

Configuring variables at the action level

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`.

```
{
    "name": "Source",
    "actions": [
        {
            "outputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "name": "Source",
            "namespace": "SourceVariables",
            "configuration": {
                "RepositoryName": "MyRepo",
                "BranchName": "mainline",
                "PollForSourceChanges": "false"
            },
            "inputArtifacts": [],
            "region": "us-west-2",
            "actionTypeId": {
                "provider": "CodeCommit",
                "category": "Source",
                "version": "1",
                "owner": "AWS"
            },
            "runOrder": 1
        }
```

```
    ]
},
```

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

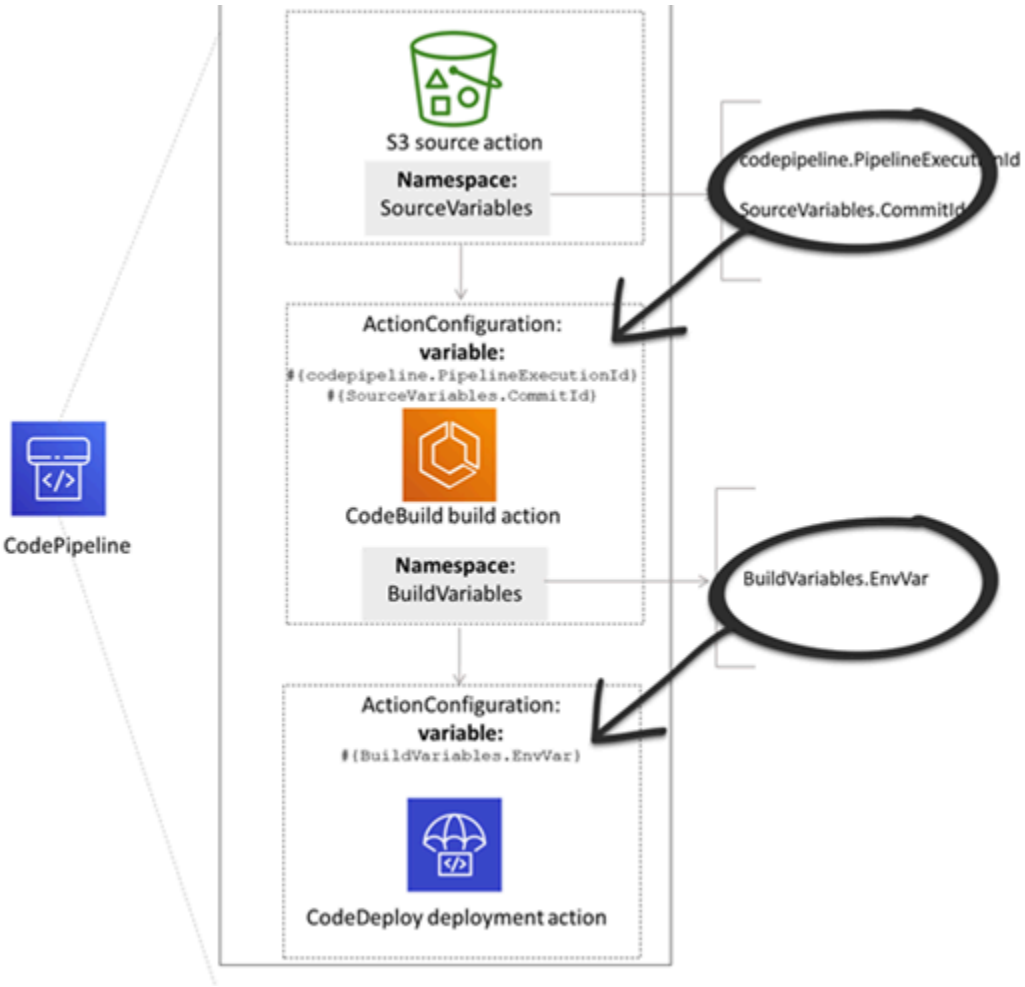
In this example, the build action's configuration field shows environment variables that are updated upon the action execution. The example specifies the namespace and variable for execution ID with `#{codepipeline.PipelineExecutionId}` and the namespace and variable for commit ID with **`#{SourceVariables.CommitId}`**.

```
{
    "name": "Build",
    "actions": [
        {
            "outputArtifacts": [
                {
                    "name": "BuildArtifact"
                }
            ],
            "name": "Build",
            "configuration": {
                "EnvironmentVariables": "[{\"name\":\"Release_ID\",\"value\":
\"#{codepipeline.PipelineExecutionId}\",\"type\":\"PLAINTEXT\"},{\"name\":\"Commit_ID
\",\"value\":\"#{SourceVariables.CommitId}\",\"type\":\"PLAINTEXT\"}]",
                "ProjectName": "env-var-test"
            },
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-west-2",
            "actionTypeId": {
                "provider": "CodeBuild",
                "category": "Build",
```

```
                "version": "1",
                "owner": "AWS"
            },
            "runOrder": 1
        }
    ]
},
```

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.
- If the namespace isn't specified, the variables produced by that action cannot be referenced in any action configuration.
- To reference variables produced by an action, the referencing action must occur after the action that produces the variables. This means it is either in a later stage than the action producing the variables, or in the same stage but at a higher run order.

Variables available for pipeline actions

The action provider determines which variables can be generated by the action.

For step-by-step procedures for managing variables, see Working with variables.

Actions with defined variable keys

Unlike a namespace which you can choose, the following actions use variable keys that cannot be edited. For example, for the Amazon S3 action provider, only the `ETag` and `VersionId` variable keys are available.

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.

Topics

- CodePipeline execution ID variable
- Amazon ECR action output variables
- CloudFormation StackSets action output variables
- CodeCommit action output variables
- CodeStarSourceConnection action output variables

- GitHub action output variables (GitHub (via OAuth app) action)
- S3 action output variables

CodePipeline execution ID variable

CodePipeline execution ID variable

Provider	Variable key	Example value	Example variable syntax
codepipeline	`PipelineExecutionId`	8abc75f0-fbf8-4f4c-bfEXAMPLE	`#{codepipeline.PipelineExecutionId}`

Amazon ECR action output variables

Amazon ECR variables

Variable key	Example value	Example variable syntax
`ImageDigest`	sha256:EXAMPLE1122334455	`#{SourceVariables.ImageDigest}`
`ImageTag`	latest	`#{SourceVariables.ImageTag}`
`ImageURI`	11111EXAMPLE.dkr.ecr.us-west-2.amazonaws.com/ecs-repo:latest	`#{SourceVariables.ImageURI}`
`RegistryId`	EXAMPLE12233	`#{SourceVariables.RegistryId}`
`RepositoryName`	my-image-repo	`#{SourceVariables.RepositoryName}`

CloudFormation StackSets action output variables

CloudFormation StackSets variables

Variable key	Example value	Example variable syntax
`OperationId`	11111111-2bbb-111-2bbb-11111example	`#{DeployVariables.OperationId}`
`StackSetId`	my-stackset:1111aaaa-1111-2222-2bbb-11111example	`#{DeployVariables.StackSetId}`

CodeCommit action output variables

CodeCommit variables

Variable key	Example value	Example variable syntax
`AuthorDate`	2019-10-29T03:32:21Z	`#{SourceVariables.AuthorDate}`
`BranchName`	development	`#{SourceVariables.BranchName}`
`CommitId`	exampleb01f91b31	`#{SourceVariables.CommitId}`
`CommitMessage`	Fixed a bug (100 KB maximum size)	`#{SourceVariables.CommitMessage}`
`CommitterDate`	2019-10-29T03:32:21Z	`#{SourceVariables.CommitterDate}`
`RepositoryName`	myCodeCommitRepo	`#{SourceVariables.RepositoryName}`

CodeStarSourceConnection action output variables

The output variables available depend on the trigger type (push, pull request, or tag). Variables in the following table marked as *All* are produced for every trigger type. Variables marked with a specific trigger type are only produced when the pipeline execution is started by that trigger.

`CodeStarSourceConnection` variables (Bitbucket Cloud, GitHub, GitHub Enterprise Repository, and GitLab.com)

Variable key	Example value	Example variable syntax	Trigger type
`AuthorDate`	2019-10-29T03:32:21Z	`#{SourceVariables.AuthorDate}`	All
`AuthorDisplayName`	Jane Doe	`#{SourceVariables.AuthorDisplayName}`	All
`AuthorEmail`	jane.doe@example.com	`#{SourceVariables.AuthorEmail}`	All
`AuthorId`	janedoe	`#{SourceVariables.AuthorId}`	All
`BranchName`	development	`#{SourceVariables.BranchName}`	Push, Tag
`CommitId`	exampleb01f91b31	`#{SourceVariables.CommitId}`	All

Variable key	Example value	Example variable syntax	Trigger type
CommitMessage	Fixed a bug (100 KB maximum size)	#{SourceVariables.CommitMessage}	All
ConnectionArn	arn:aws:codestar-connections:region:*account-id* :connection/*connection-id*	#{SourceVariables.ConnectionArn}	All
DestinationBranchName	main	#{SourceVariables.DestinationBranchName}	Pull request
FullRepositoryName	username/GitHubRepo	#{SourceVariables.FullRepositoryName}	All
ProviderType	GitHub	#{SourceVariables.ProviderType}	All
PullRequestId	1	#{SourceVariables.PullRequestId}	Pull request
PullRequestTitle	Add new feature	#{SourceVariables.PullRequestTitle}	Pull request

Variable key	Example value	Example variable syntax	Trigger type
SourceBranchName	feature-branch	#{SourceVariables.SourceBranchName}	Pull request
TagName	v1.0.0	#{SourceVariables.TagName}	Tag

GitHub action output variables (GitHub (via OAuth app) action)

GitHub variables (GitHub (via OAuth app) action)

Variable key	Example value	Example variable syntax
AuthorDate	2019-10-29T03:32:21Z	#{SourceVariables.AuthorDate}
BranchName	main	#{SourceVariables.BranchName}
CommitId	exampleb01f91b31	#{SourceVariables.CommitId}
CommitMessage	Fixed a bug (100 KB maximum size)	#{SourceVariables.CommitMessage}
CommitterDate	2019-10-29T03:32:21Z	#{SourceVariables.CommitterDate}
CommitUrl		#{SourceVariables.CommitUrl}
RepositoryName	myGitHubRepo	#{SourceVariables.RepositoryName}

S3 action output variables

S3 variables

Variable key	Example value	Example variable syntax
`ETag`	example28be1c3	`#{SourceVariables.ETag}`
`VersionId`	exampleta_IUQCv	`#{SourceVariables.VersionId}`

Actions with user-configured variable keys

For CodeBuild, CloudFormation, and Lambda actions, the variable keys are configured by the user.

Topics

- CloudFormation action output variables
- CodeBuild action output variables
- Lambda action output variables

CloudFormation action output variables

CloudFormation variables

Variable key	Example variable syntax
For 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 • CHANGE_SET_EXECUTE	`#{DeployVariables.StackName}`

Variable key	Example variable syntax
• CHANGE_SET_REPLACE • REPLACE_ON_FAILURE For more information about these action modes, see CloudFormation deploy action reference. For a tutorial that shows you how to create a pipeline with an CloudFormation deployment action in a pipeline that uses CloudFormation output variables, see Tutorial: Create a pipeline that uses variables from AWS CloudFormation deployment actions.	

CodeBuild action output variables

CodeBuild variables

Variable key	Example variable syntax
For CodeBuild actions, variables are produced from values generated by exported environment variables. Set up a CodeBuild environment variable by editing your CodeBuild action in CodePipeline or by adding the environment variable to the buildspec. Add instructions to your CodeBuild buildspec to add the environment variable under the exported variables section. See env/exported-variables in the *AWS CodeBuild User Guide*.	`#{BuildVariables.EnvVar}`

Lambda action output variables

Lambda variables

Variable key	Example variable syntax
The Lambda action will produce as variables all key-value pairs that are included in the `outputVariables` section of the PutJobSuccessResult API request.	`#{TestVariables.testRunId}`

Variable key	Example variable syntax
For a tutorial with a Lambda action that uses variables from an upstream action (CodeCommit) and generates output variables, see Tutorial: Using variables with Lambda invoke actions.	

Working with glob patterns in syntax

When you specify the files or paths that are used in pipeline artifacts or source locations, you can specify the artifact depending on the action type. For example, for the S3 action, you specify the S3 object key.

For triggers, you can specify filters. You can use glob patterns to specify filters. The following are examples.

When the syntax is "glob" then the String representation of the path is matched using a limited pattern language with a syntax that resembles regular expressions. For example:

- `*.java` Specifies a path that represents a file name ending in .java
- `*.*` Specifies file names containing a dot
- `*.{java,class}` Specifies file names ending with .java or .class
- `foo.?` Specifies file names starting with foo. and a single character extension

The following rules are used to interpret glob patterns:

- To specify zero or more characters of a name component in directory boundaries, use `*`.
- To specify zero or more characters of a name component crossing directory boundaries, use `**`.
- To specify one character of a name component, use `?`.
- To escape characters that would otherwise be interpreted as special characters, use the backslash character (\).
- To specify a single character out of a set of characters, use `[ ]`.
- To specify a single file that is in the root of your build location or source repository location, use `my-file.jar`.
- To specify a single file in a subdirectory, use `directory/my-file.jar` or `directory/subdirectory/my-file.jar`.
- To specify all files, use `"**"`. The `**` glob pattern indicates to match any number of subdirectories.
- To specify all files and directories in a directory named `directory`, use `"directory/**"`. The `**` glob pattern indicates to match any number of subdirectories.

- To specify all files in a directory named `directory`, but not any of its subdirectories, use `"directory/*"`.
- Within a bracket expression the `*`, `?` and `\` characters match themselves. The (-) character matches itself if it is the first character within the brackets, or the first character after the `!` if negating.
- The `{ }` characters are a group of subpatterns, where the group matches if any subpattern in the group matches. The `","` character is used to separate the subpatterns. Groups cannot be nested.

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. For a migration guide with instructions for updating your polling pipelines to use the recommended event-based change detection method, see Migrate polling pipelines to use event-based change detection.

Update a GitHub (via OAuth app) source action to a GitHub (via GitHub App) source action

In AWS CodePipeline, there are two supported versions of the GitHub source action:

- **Recommended:** The GitHub (via GitHub App) action uses GitHub app-based auth backed by a CodeStarSourceConnection for Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed actions resource. It installs an AWS CodeStar Connections application into your GitHub organization so that you can manage access in GitHub.
- **Not recommended:** The GitHub (via OAuth app) action uses OAuth tokens to authenticate with GitHub and uses a separate webhook to detect changes. This is no longer the recommended method.

There are some important advantages to using the GitHub (via GitHub App) action instead of the GitHub (via OAuth app) action:

- With connections, CodePipeline no longer requires OAuth apps or personal access tokens to access your repository. When you create a connection, you install a GitHub App that manages authentication to your GitHub repository and allows permissions at the organization level. You must authorize OAuth tokens as a user to access the repository. For more information about OAuth-based GitHub access in contrast to App-based GitHub access, see https://docs.github.com/en/developers/apps/differences-between-github-apps-and-oauth-apps.
- When you manage GitHub (via GitHub App) actions in the CLI or CloudFormation, you no longer have to store your personal access token as a secret in Secrets Manager. You no longer have to dynamically reference the stored secret in your CodePipeline action configuration. You instead add the connection ARN to your action configuration. For an example action configuration, see CodeStarSourceConnection for Bitbucket Cloud, GitHub, GitHub Enterprise Server, GitLab.com, and GitLab self-managed actions.
- When you create a connection resource to use with your GitHub (via GitHub App) action in CodePipeline, you can use the same connection resource to associate other supported services, such as CodeGuru Reviewer, with your repository.
- In GitHub (via GitHub App), you can clone repositories to access git metadata in subsequent CodeBuild actions, while in GitHub (via OAuth app) you can only download the source.

- An administrator installs the app for your organization's repositories. You no longer have to track OAuth tokens that depend on the individual who created the token.

All apps installed to an organization have access to the same set of repositories. To change who can access each repository, modify the IAM policy for each connection. For an example, see Example: A scoped-down policy for using connections with a specified repository.

You can use the steps in this topic to delete your GitHub (via OAuth app) source action and add a GitHub (via GitHub App) source action from the CodePipeline console.

Topics

- Step 1: Replace your (via OAuth app) GitHub action
- Step 2: Create a connection to GitHub
- Step 3: Save your GitHub source action

Step 1: Replace your (via OAuth app) GitHub action

Use the pipeline edit page to replace your (via OAuth app) GitHub action with a GitHub (via GitHub App) action.

To replace your (via OAuth app) GitHub action

1. Sign in to the CodePipeline console.
2. Choose your pipeline, and choose **Edit**. Choose **Edit stage** on your source stage. A message displays that recommends you update your action.
3. In **Action provider**, choose **GitHub (via GitHub App)**.
4. Do one of the following:

   - Under **Connection**, if you have not already created a connection to your provider, choose **Connect to GitHub**. Proceed to Step 2: Create a connection to GitHub.
   - Under **Connection**, if you have already created a connection to your provider, choose the connection. Proceed to Step 3: Save the Source Action for Your Connection.

Step 2: Create a connection to GitHub

After you choose to create the connection, the **Connect to GitHub** page is shown.

To create a connection to GitHub

1. Under **GitHub connection settings**, your connection name is shown in **Connection name**.

   Under **GitHub Apps**, choose an app installation or choose **Install a new app** to create one.

   > **Note**
   >
   > You install one app for all of your connections to a particular provider. If you have already installed the GitHub app, choose it and skip this step.

2. If the authorization page for GitHub displays, log in with your credentials and then choose to continue.
3. On the app installation page, a message shows that the AWS CodeStar app is trying to connect to your GitHub account.

   > **Note**
   >
   > You only install the app once for each GitHub account. If you previously installed the app, you can choose **Configure** to proceed to a modification page for your app installation, or you can use the back button to return to the console.

4. On the **Install AWS CodeStar** page, choose **Install**.
5. On the **Connect to GitHub** page, the connection ID for your new installation is displayed. Choose **Connect**.

Step 3: Save your GitHub source action

Complete your updates on the **Edit action** page to save your new source action.

To save your GitHub source action

1. In **Repository**, enter the name of your third-party repository. In **Branch**, enter the branch where you want your pipeline to detect source changes.

   > **Note**
   >
   > In **Repository**, type `owner-name/repository-name` as shown in this example:

```
my-account/my-repository
```

2. In **Output artifact format**, choose the format for your artifacts.

   - To store output artifacts from the GitHub action using the default method, choose **CodePipeline default**. The action accesses the files from the GitHub 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.

     If you choose this option, you will need to update the permissions for your CodeBuild project service role as shown in Add CodeBuild GitClone permissions for connections to Bitbucket, GitHub, GitHub Enterprise Server, or GitLab.com. For a tutorial that shows you how to use the **Full clone** option, see Tutorial: Use full clone with a GitHub pipeline source.

3. In **Output artifacts**, you can retain the name of the output artifact for this action, such as `SourceArtifact`. Choose **Done** to close the **Edit action** page.
4. Choose **Done** to close the stage editing page. Choose **Save** to close the pipeline editing page.

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. It can take up to two weeks to process requests for a quota increase.

Resource	Default
Length of time before an action times out (This is configurable timeouts. See the following table for non-configurable timeouts)	CloudFormation deployment action: 3 days CodeDeploy and CodeDeploy ECS (blue/green) deployment actions: 5 days AWS Lambda invoke action: 24 hours **Note** While the action is running, CodePipeline periodically contacts Lambda for a status. The Lambda function replies with a status where the action execution is either successful, failed, or in progress. If the Lambda function has sent no reply after 20 minutes, the action times out. If, during the 20 minutes, the Lambda function has replied that the action is still in progress, CodePipeline restarts the 20-minute timer and tries again. If not successful after 24 hours, CodePipeline sets the Lambda invoke action state to failed. Lambda has a separate timeout for Lambda functions that is not

Resource	Default
	related to the CodePipeline action timeout. Amazon S3 deployment action: 90 minutes **Note** If the upload to S3 times out during deployment of a large ZIP file, the action fails with a timeout error. Try breaking up the ZIP file into smaller files. Manual approval action account level default timeout: 7 days **Note** The default timeout for the manual approval action can be overridden for a specific action in the pipeline, and it is configurable up to 86400 minutes (60 days) with a minimum value of 5 minutes. For more information, see ActionDeclaration in the *CodePipeline API Reference*. When configured, this timeout is applied for the action. Otherwise, the account level default is used. All other actions: 1 hour

Resource	Default
	Note The Amazon ECS deployment action timeout is configurable up to one hour (the default timeout).
Maximum number of total pipelines per Region in an AWS account	1000 **Note** Pipelines configured for either polling or event-based change detection are counted toward this quota.
Maximum number of pipelines set to polling for source changes, per AWS Region	300 **Note** This quota is fixed and cannot be changed. If you reach the limit for polling pipelines, you can still configure additional pipelines that use event-based change detection . For more information, see Source actions and change detection methods.[1]
Maximum number of webhooks per Region in an AWS account	300
Number of custom actions per Region in an AWS account	50

Resource	Default
[1]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 Migrate polling pipelines (CodeCommit or Amazon S3 source) (console). • To update an Amazon S3 source action, see Migrate polling pipelines (CodeCommit or Amazon S3 source) (console). • To update a GitHub source action, see Migrate polling pipelines to webhooks (GitHub (via OAuth app) source actions) (console).	

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 a list of the CodePipeline service endpoints for each Region, see AWS CodePipeline endpoints and quotas in the *AWS General Reference*.

For information about structural requirements, see CodePipeline pipeline structure reference.

AWS Regions where you can create a pipeline	US East (N. Virginia) US East (Ohio) US West (N. California) US West (Oregon) Canada (Central) Europe (Frankfurt) Europe (Zurich)* Europe (Stockholm) Europe (Milan)* Europe (Spain)*

	Europe (Ireland)
	Europe (London)
	Europe (Paris)
	Israel (Tel Aviv)*
	Africa (Cape Town)*
	Asia Pacific (Hong Kong)*
	Asia Pacific (Mumbai)
	Asia Pacific (Hyderabad)*
	Asia Pacific (Tokyo)
	Asia Pacific (Seoul)
	Asia Pacific (Osaka)
	Asia Pacific (Singapore)
	Asia Pacific (Sydney)
	Asia Pacific (Jakarta)*
	Asia Pacific (Melbourne)*
	Asia Pacific (Malaysia)*
	Asia Pacific (New Zealand)*
	South America (São Paulo)
	Middle East (Bahrain)*
	Middle East (UAE)*
	AWS GovCloud (US-West)
	AWS GovCloud (US-East)

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 character s. 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 artifact names	Artifact 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 - (minus sign), and _ (underscore). Any other characters, such as spaces, are not allowed.
Characters allowed in partner action names	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: 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 a pipeline name	Pipeline 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 a stage name	Stage 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.
Length of time before an action times out	CodeBuild build action: 36 hours Test action: 8 hours Custom actions: 24 hours Step Functions invoke action: 7 days Build timeout for the Commands action: 55 minutes

Maximum length of the action configuration key (for example, the CodeBuild configuration keys are `ProjectName` , `PrimarySource` , and `EnvironmentVariables`)	50 characters
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: `"RepositoryName": "my-repo-name-less-than-1000-characters"`)	1000 characters
Maximum number of actions per pipeline	1000
Maximum number of concurrent pipeline executions per pipeline (QUEUED|PARALLEL mode)	50
Maximum number of concurrent action executions per PARALLEL mode pipeline execution	5
Maximum number of files for an Amazon S3 object	100,000
Maximum number of parallel actions in a stage	100
Maximum number of sequential actions in a stage	100

Maximum size of artifacts in a source stage	Artifacts stored in Amazon S3 buckets: 7 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 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.
Maximum size of the image definitions JSON file used in pipelines deploying Amazon ECS containers and images	100 KB
Maximum size of input artifacts for CloudFormation actions	256 MB
Maximum size of input artifacts for the `CodeDeployToECS` action	3 MB
Maximum size of input artifacts for the `Step Functions` action	The Step Functions action runs on Lambda, and it therefore has artifact size quotas that are the same as the artifact size quotas for Lambda functions. For more information, see Lambda quotas in the Lambda Developer Guide.

Maximum size of the JSON object that can be stored in the `ParameterOverrides` property	For a CodePipeline deploy action with CloudFormation as the provider, the `ParameterOverrides` property is used to store a JSON object that specifies values for the CloudFormation template configuration file. There is a maximum size limit of 1 kilobyte for the JSON object that can be stored in the `ParameterOverrides` property.
Number of actions in a stage	Minimum of 1, maximum of 50
Number of artifacts allowed for each action	For the number of input and output artifacts allowed for each action, see the Valid input and output artifacts for each action type
Number of months that pipeline execution history information is retained	12
Number of stages in a pipeline	Minimum of 2, maximum of 50
Pipeline tags	Tags are case sensitive. Maximum of 50 per resource.
Pipeline tag key names	Any combination of Unicode letters, numbers, spaces, and allowed characters in UTF-8 between 1 and 128 characters in length. Allowed characters are + - = . _ : / @ Tag key names must be unique, and each key can have only one value. A tag cannot: • begin with AWS: • consist only of spaces • end with a space • contain emojis or any of the following characters: ? ^ * [\ ~ ! # $ % & * ( ) > < | " '

Pipeline tag values	Any combination of Unicode letters, numbers, spaces, and allowed characters in UTF-8 between 1 and 256 characters in length. Allowed characters are + - = . _ : / @ A key can have only one value, but many keys can have the same value. A tag cannot: • begin with AWS: • consist only of spaces • end with a space • contain emojis or any of the following characters: ? ^ * [\ ~ ! # $ % & * ( ) > < | " '

Triggers	There is a maximum of 50 triggers in a pipeline definition across the `push` and `pull request` configuration. There is a maximum of three filters per push trigger and pull request trigger. **Note** Duplicates for filters in the same event type array are not allowed. You can add up to 8 include and 8 exclude patterns, branches, and file paths for each event type (push, pull request). Allowed characters in patternvalues include all character types. For include and exclude patterns, there is a maximum length of 255 characters. For tag names, there is a maximum length of 255 characters. Maximum size of `triggers` array should not exceed 200 KB

Trigger filters	File paths: • Number of patterns: You can add up to 8 include and 8 exclude patterns. • Size of pattern: Each include or exclude pattern's size can be up to 255 characters. Branches: • Number of patterns: You can add up to 8 include and 8 exclude patterns. • Size of pattern: Each include or exclude pattern's size can be up to 255 characters. Pull requests: Branches: • Number of patterns: You can add up to 8 include and 8 exclude patterns. • Size of pattern: Each include or exclude pattern's size can be up to 255 characters.
Uniqueness of names	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. Stage names must be unique within a pipeline. Action names must be unique within a stage.

Quotas for output variables and namespaces	There is a maximum size limit of 122880 bytes for all output variables combined for a particular action. There is a maximum size limit of 100 KB for the total resolved action configuration for a particular action. Output variable names are case sensitive. Namespaces are case sensitive. Allowed characters include: • Lowercase letters a through z, inclusive. • Uppercase letters A through Z, inclusive. • Numbers 0 through 9, inclusive. • Special characters ^ (caret), @ (at sign), - (minus sign), _ (underscore), [(left bracket),] (right bracket), * (asterisk), $ (dollar sign). Any other characters, such as spaces, are not allowed.

Quotas for variables at the pipeline level	There is a maximum of 50 pipeline-level variables per pipeline. Variable names for variables at the pipeline level must be: • 128 characters maximum length • Lowercase letters a through z, inclusive. • Uppercase letters A through Z, inclusive. • Numbers 0 through 9, inclusive. • Special characters `@\-_]+` Any other characters, such as spaces, are not allowed. For variable values, there is a maximum length of 1000 characters For variable values, all characters are allowed. For variable descriptions, there is a maximum length of 200 characters.

* You must enable this Region before you can use it.

Appendix A: GitHub (via OAuth app) source actions

This appendix provides information about the GitHub (via OAuth app) action in CodePipeline.

> **Note**
>
> While we don't recommend using the GitHub (via OAuth app) action, existing pipelines with the GitHub (via OAuth app) action will continue to work without any impact. For a pipeline with a GitHub (via OAuth app) action, CodePipeline uses OAuth-based tokens to connect to your GitHub repository. By contrast, the GitHub action (via GitHub App) uses a connection resource to associate AWS resources to your GitHub repository. The connection resource uses app-based tokens to connect. For more information about updating your pipeline to the recommended GitHub action that uses a connection, see Update a GitHub (via OAuth app) source action to a GitHub (via GitHub App) source action. For more information about OAuth-based GitHub access in contrast to app-based GitHub access, see https://docs.github.com/en/developers/apps/differences-between-github-apps-and-oauth-apps.

To integrate with GitHub, CodePipeline uses a GitHub OAuth application for your pipeline. CodePipeline uses webhooks to manage change detection for your pipeline with the GitHub (via OAuth app) source action.

> **Note**
>
> When you configure a GitHub (via GitHub App) source action in CloudFormation, you do not include any GitHub token information or add a webhook resource. You configure a connections resource as shown in AWS::CodeStarConnections::Connection in the *CloudFormation User Guide.*

This reference contains the following sections for the GitHub (via OAuth app) action:

- For information about how to add a GitHub (via OAuth app) source action and webhook to a pipeline, see Adding a GitHub (via OAuth app) source action.
- For information about the configuration parameters and example YAML/JSON snippets for a GitHub (via OAuth app) source action, see GitHub (via OAuth app) source action reference.

Important

When creating CodePipeline webhooks, do not use your own credentials or reuse the same secret token across multiple webhooks. For optimal security, generate a unique secret token for each webhook you create. The secret token is an arbitrary string that you provide, which GitHub uses to compute and sign the webhook payloads sent to CodePipeline, for protecting the integrity and authenticity of the webhook payloads. Using your own credentials or reusing the same token across multiple webhooks can lead to security vulnerabilities.

Note

If a secret token was provided, it will be redacted in the response.

Topics

- Adding a GitHub (via OAuth app) source action
- GitHub (via OAuth app) source action reference

Adding a GitHub (via OAuth app) source action

You add GitHub (via OAuth app) source actions to CodePipeline by:

- Using the CodePipeline console **Create pipeline** wizard (Create a custom pipeline (console)) or **Edit action** page to choose the **GitHub** provider option. The console creates a webhook that starts your pipeline when the source changes.
- Using the CLI to add the action configuration for the `GitHub` action and creating additional resources as follows:
  - Using the `GitHub` example action configuration in GitHub (via OAuth app) source action reference to create the action as shown in Create a pipeline (CLI).
  - Disabling periodic checks and creating the change detection manually, because the change detection method defaults to starting the pipeline by polling the source. You migrate your polling pipeline to webhooks for GitHub (via OAuth app) actions.

GitHub (via OAuth app) source action reference

> **Note**
>
> While we don't recommend using the GitHub (via OAuth app) action, existing pipelines with the GitHub (via OAuth app) action will continue to work without any impact. For a pipeline with a GitHub (via OAuth app) source action, CodePipeline uses OAuth-based tokens to connect to your GitHub repository. By contrast, the new GitHub action (via GitHub App) uses a connection resource to associate AWS resources to your GitHub repository. The connection resource uses app-based tokens to connect. For more information about updating your pipeline to the recommended GitHub action that uses a connection, see Update a GitHub (via OAuth app) source action to a GitHub (via GitHub App) source action.

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
- Configuration parameters
- Input artifacts
- Output artifacts
- Output variables
- Action declaration (GitHub example)

- Connecting to GitHub (OAuth)
- See also

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 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 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 Migrate polling pipelines to webhooks (GitHub (via OAuth app) source actions) (CLI) or Update pipelines for push events (GitHub (via OAuth app) source actions) (CloudFormation template).

> **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.

  > **Note**
  >
  > If you omit `PollForSourceChanges`, CodePipeline defaults to polling your repository for source changes. This behavior is the same as if `PollForSourceChanges` is set to `true`.

- `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 reference.

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.

CommitterDate

The date when the commit was committed, in timestamp format.

Action declaration (GitHub example)

YAML

```
Name: Source
Actions:
  - InputArtifacts: []
    ActionTypeId:
      Version: '1'
      Owner: ThirdParty
      Category: Source
      Provider: GitHub
    OutputArtifacts:
      - Name: SourceArtifact
    RunOrder: 1
    Configuration:
      Owner: MyGitHubAccountName
      Repo: MyGitHubRepositoryName
      PollForSourceChanges: 'false'
      Branch: main
      OAuthToken: '{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}'
    Name: ApplicationSource
```

JSON

```
{
    "Name": "Source",
    "Actions": [
        {
            "InputArtifacts": [],
            "ActionTypeId": {
```

```
                "Version": "1",
                "Owner": "ThirdParty",
                "Category": "Source",
                "Provider": "GitHub"
            },
            "OutputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "RunOrder": 1,
            "Configuration": {
                "Owner": "MyGitHubAccountName",
                "Repo": "MyGitHubRepositoryName",
                "PollForSourceChanges": "false",
                "Branch": "main",
                "OAuthToken":
 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}"
            },
            "Name": "ApplicationSource"
        }
    ]
},
```

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 CloudFormation template, you must provide the value for a personal access token that you have already created in GitHub.

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 CloudFormation.
- Resource reference for the AWS CloudFormation User Guide AWS::CodeStar::GitHubRepository – This includes field definitions, examples, and snippets for the resource in CloudFormation.
- Tutorial: Create a pipeline that builds and tests your Android app with AWS Device Farm – This tutorial provides a sample buildspec 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.

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:** May 23, 2025

Change	Description	Date
Corrected documentation for pipeline trigger filter behavior	Corrected the documentation for trigger filter examples to accurately describe the relationship between includes and excludes. Excludes take precedence over includes for branch and file path filters. For more information, see Examples for trigger filters.	March 16, 2026
Support added for Azure DevOps	Support added for configuri ng connections for AWS resources to interact with Azure DevOps. For more information, see Connections for Azure DevOps.	August 5, 2025
Updated documentation for the EC2 deploy action deploy spec file	Added new information about using the deploy spec file for the EC2 deploy action with updated examples and configuration options. See the action reference page at Amazon EC2 action reference.	May 16, 2025

Added documentation for the Lambda deploy action	Added support for the new Lambda deploy action with updated examples and configuration options. See the action reference page at AWS Lambda deploy action reference.	May 16, 2025
Add fields to Commands action for compute type, environment type, and environment variable type	Added information about new fields that allow you to specify a build environme nt and compute type for the Commands action. A new type field for environment variables has also been added for the Commands action. See the action reference page at CodePipeline Comands action.	May 8, 2025
New topic to document the default service role policy	Added information about the minimum service role policy and updated the table of links to additional service role permissions for each action in CodePipeline. See the new rule reference page at CodePipeline service role policy.	March 26, 2025
New `CodePipeline` invoke action	Added information about the new `CodePipeline` invoke action. See the action reference page at CodePipel ine invoke action reference.	March 14, 2025

New `CodeBuild` rule	Added information about the new `CodeBuild` rule that is available to run build projects as a rule for stage conditions. See the new rule reference page at CodeBuild rule reference.	March 14, 2025
Sharing connections in AWS CodeConnections	For pipelines with connections using AWS CodeConnections, you can use connections configured for sharing between AWS accounts. You can configure shared connections in AWS Resource Access Manager. For more information, see Use a connection shared with another AWS account.	March 6, 2025

EventBridge input transform entry for source revisions	For pipelines with CodeCommit, Amazon ECR, and Amazon S3 sources, you can use EventBridge input transform entry for source revisions, where the `revisionValue` is derived from the source event variable for your object key (S3), commit (CodeComm it), or image ID (ECR). See Amazon ECR source actions and EventBridge resources , Connecting to Amazon S3 source actions with a source enabled for events, and CodeCommit source actions and EventBridge.	March 3, 2025
Build spec override for the AWS CodeBuild action	You can choose to override the build spec for the CodeBuild action while instead entering commands directly. See the action reference page at AWS CodeBuild build and test action reference. Also see Create a pipeline, stages, and actions.	March 3, 2025

New EC2 deploy action	Added information about the new EC2 deploy action. See the action reference page at Amazon EC2 action reference . For a tutorial, see Tutorial: Deploy to EC2 instances with CodePipeline.	February 21, 2025
New EKS deploy action	Added information about the new EKS deploy action. See the action reference page at EKS deploy action. For a tutorial, see Tutorial: Deploy to Amazon EKS with CodePipeline.	February 20, 2025
New CloudWatch metrics and dimensions for CodePipeline	Added metrics for pipelines in CloudWatch metrics. See CodePipeline CloudWatch metrics.	February 13, 2025
New Commands rule	Added information about the new Commands rule that is available to run shell commands as a rule for stage conditions. See the new rule reference page at Commands rule reference.	December 17, 2024

Expanded examples and reference information for triggers	Added descriptions for pull request event filters by provider in Pull request events for triggers by provider. Added expanded examples for triggers with more detailed informati on about includes and excludes in push events and pull request events in Pull request events for triggers by provider. Added additional JSON reference information about includes and excludes in Triggers.	December 17, 2024
New actions in the action catalog	You can now use the `ECRBuildAndPublish` and `InspectorScan` actions. For more information, see the ECRBuildAndPublish and InspectorScan action reference pages.	November 22, 2024
New automatic configuration for stage retry on failure	You can configure a stage to automatically retry a failed stage or failed actions in the stage. For more informati on, see Configure a stage for automatic retry on failure.	October 15, 2024

New `Skip` result for Entry conditions	Added information about the `Skip` result that is available for entry conditions. You can use the `VariableCheck` and `LambdaInvoke` rules with this configuration. See the steps at Creating Entry conditions with Skip result. For a list of considerations for conditions with Skip results, see Considerations for results configured for stage conditions.	October 15, 2024
New console steps to create pipelines from static templates	In the CodePipeline console, you can use a new pipeline creation wizard to choose from multiple static templates to generate pipeline resources in CloudFormation. For more information, see Create a pipeline from static templates.	October 9, 2024

New `Commands` action	Added information about the new `Commands` action that is available to run shell commands as an action in your pipeline. See the new action reference page at Commands action and the service role permissions at Add permissions to the CodePipeline service role. For a tutorial, see Tutorial: Create a pipeline that runs commands with compute.	October 3, 2024
New `VariableCheck` rule for conditions	Added information about the `VariableCheck` rule for stage conditions. See the new rule reference page at VariableCheck. For a tutorial, see Tutorial: Create a variable check rule for a pipeline as an entry condition.	September 27, 2024
Updates to connections for GitHub	Added information about using the GitHub user access token with connections to GitHub (GitHub V2 actions). User access tokens are used with CodeBuild projects. See GitHub connections and Tutorial: Use full clone with a GitHub pipeline source.	September 16, 2024

Updates to restructure the pipeline JSON reference and the guide table of contents	The guide has been restructured, including some section title changes, to enhance the useability of the reference and task sections.	August 16, 2024
Update to the secret token field in the response for `PutWebhook` and `ListWebhooks` actions	The secret token field for the `PutWebhook` and `ListWebhooks` actions has been updated. If a secret token was provided, it will be redacted in the response. See the notes added to Appendix A: GitHub version 1 source actions. For related updates in the *CodePipeline API Guide*, see PutWebhook and ListWebhooks.	August 6, 2024
Added new content for stage conditions and rules	You can now configure stage conditions and rules for V2 type pipelines. See Concepts, How do stage conditions work?, and Configure conditions for a stage. A reference chapter for rules has been added that provides reference information. See the CodePipeline rule reference.	July 30, 2024

Added new reference information for pipeline types and related features	A new analysis script is available for evaluating the cost for moving to V2 type pipelines. See What type of pipeline is right for me?. A reference table has been added that provides links across all CodePipeline service documention by feature. See the CodePipeline feature reference.	July 11, 2024
Updates to the S3 source action to add a new option for source overrides	A new option for source overrides named `S3_OBJECT_KEY` is available for the S3 source action. A new `AllowOverrideForS3ObjectKey` parameter has been added for the S3 source action. See the Amazon S3 source action reference page and Start a pipeline with a source revision override.	June 7, 2024
Updates to the S3 source action to add new output variables	New output variables named `BucketName` and `ObjectKey` are available for the S3 source action. See the Amazon S3 source action reference page.	June 5, 2024
Updates to the `CloudFormationStackSet` and `CloudFormationStackInstances` actions	The `CallAs` parameter was added for the `CloudFormationStackSet` and `CloudFormationStackInstances` action. See the action reference page.	May 2, 2024

Support for stage-level rollbacks	You can manually or automatically roll back a stage to a previous successful pipeline execution for the stage. See Configuring stage rollback and Concepts.	April 26, 2024
Updates to Region availability for StackSets and Step Functions actions	The StackSets and Step Functions actions are now available in all Regions where CodePipeline is available. See CloudFormation StackSets action reference and AWS Step Functions action reference.	March 27, 2024
Updates to managed policy	The AWS managed policy `AWSCodePipeline_FullAccess` was updated. See AWS managed policies for AWS CodePipeline.	March 15, 2024
Support for configurable timeout for manual approval actions	Quota information added for new configurable timeout field for manual approval actions. For more information, see Quotas.	February 15, 2024

Support for trigger filtering by branches and file paths	Support added for trigger configuration that allows filtering on pull request status, branches, and file paths for V2 type pipelines . For more information, see Filtering triggers on code push or pull requests , Triggers, and Filter on feature branches to start your pipeline, and Quotas.	February 8, 2024
Support for new pipeline execution modes	Support added for PARALLEL and QUEUED pipeline execution modes. For more information, see Set the pipeline execution mode , How executions are processed in QUEUED mode, How executions are processed in PARALLEL mode, and Quotas.	February 8, 2024
Updates to console pages for viewing action details, reviewing manual approval actions, and the list pipelines page	Console updates documente d for new **View details** button and dialog box, new manual approval dialog, and new columns for recent execution s on the list pipelines page. For more information, see View pipelines (console), View action details in a pipeline, and Manage approval actions in pipelines.	January 10, 2024

Support for GitLab self-managed	Support added for configuring connections for AWS resources to interact with GitLab self-managed. For more information, see Connections for GitLab self-managed.	December 28, 2023
Updates to the `CloudFormationStackSet` and `CloudFormationStackInstances` actions	The `ConcurrencyMode` parameter was added for the `CloudFormationStackSet` and `CloudFormationStackInstances` action. See the action reference page.	December 19, 2023
Updates to AWS Device Farm action parameters in CodePipeline	The parameters for the AWS Device Farm action in CodePipeline have been updated. For more information, see AWS Device Farm action reference.	December 18, 2023
Support added for detailed error messages for the CloudFormation action in CodePipeline	CloudFormation action error messages can now surface details about resources that failed. For more information, see CloudFormation action reference.	December 15, 2023
Updates for starting a pipeline with source revision overrides in CodePipeline	You can now start a pipeline with a specified source revision. For more information, see Start a pipeline with a source revision override.	November 17, 2023

New supported Regions	CodePipeline is now available in Asia Pacific (Hyderabad), Asia Pacific (Jakarta), Asia Pacific (Melbourne), Asia Pacific (Osaka), Middle East (UAE), Europe (Spain), and Israel (Tel Aviv) Regions. The Events placeholder bucket reference topic and AWS service endpoints topic have been updated.	November 13, 2023
Updates for event fields in Amazon EventBridge	You can now view updated event fields in Amazon EventBridge. For more information, see Monitoring CodePipeline events.	November 9, 2023
Updates for new pipeline type V2 pipelines, triggers on Git tags, and pipeline variables in CodePipeline	You can now choose a pipeline type in CodePipeline. For a V2 type pipeline, you can now use a trigger configuration to start your pipeline on Git tags. With V2 type pipelines, you can also use variables at the pipeline level to pass input parameters for a pipeline execution. For more information, see Variables, Tutorial: Use pipeline-level variables, and Tutorial: Use Git tags to start your pipeline. For more information about pipeline types, see Pipeline types.	October 24, 2023

CodePipeline allows retrying all actions in a failed stage	For a failed stage in CodePipeline, you can retry the stage without re-running the pipeline. You do this either by retrying the failed actions in a stage or by retrying all actions in the stage starting from the first action in the stage. For more information, see.	October 17, 2023
Support for GitLab groups	Support added for configuring connections for AWS resources to interact with GitLab groups. For more information, see GitLab connections.	September 15, 2023
CodePipeline supports connections to GitLab.com	You can use connections to configure AWS resources to interact with GitLab.com. You can also choose the full clone option for using Git commands and metadata for downstream actions. For more information, see GitLab connections and the CodeStarSourceConnection action structure reference topic.	August 10, 2023

Update to the CloudFormationStackInstances action	The RegionConcurrencyType parameter was added for the CloudFormationStackInstances action. See the action reference page for the CloudFormationStackInstances action.	August 8, 2023
Update to the CloudFormationStackSet action	The RegionConcurrencyType parameter was added for the CloudFormationStackSet action. See the action reference page for the CloudFormationStackSet action.	July 24, 2023
Updates to managed policy	The AWS managed policy AWSCodePipeline_FullAccess was updated. See AWS managed policies for AWS CodePipeline.	June 21, 2023
Updates to migration procedures for polling pipelines	The procedures to migrate (update) polling pipelines to use event-based change detection have been updated with the steps for pipelines that use an Amazon S3 bucket enabled for notifications to EventBridge. For more information, see Migrate polling pipelines to use event-based change detection.	June 12, 2023

Updates to managed policies	The AWS managed policies `AWSCodePipeline_FullAccess` and `AWSCodePipeline_ReadOnlyAccess` have been updated with an additional permission. For more information, see AWS CodePipeline updates to AWS managed policies.	May 16, 2023
Updates to managed policies	The AWS managed policies `AWSCodePipelineFullAccess` and `AWSCodePipelineReadOnlyAccess` are deprecated. Use the `AWSCodePipeline_FullAccess` and `AWSCodePipeline_ReadOnlyAccess` policies. See AWS CodePipeline updates to AWS managed policies.	November 17, 2022

Updates to procedures that use CloudTrail	All console procedures, sample CLI commands, and sample CloudFormation snippets and templates for a pipeline with an S3 source have been updated with the option to choose Write and select false for Management events in CloudTrail. See the updated samples in Starting a pipeline, Tutorial: Create a pipeline with CloudFormation, Edit pipelines to use push events, and Update polling pipelines.	April 27, 2022
New supported integration with Snyk	You can use the Snyk invoke action in CodePipeline to automate security scanning for your open source code. For more information, refer to the Snyk action reference and Integrations.	June 10, 2021
New supported Region Europe (Milan)	CodePipeline is now available in Europe (Milan). The Limits topic and AWS service endpoints topic have been updated.	January 27, 2021

Change detection can be turned off for source actions with connections	You can use the CLI or SDK to update a `CodeStarSourceConnection` source action to turn off automatic change detection for the source repository. The CodeStarSourceConnection action structure reference topic has been updated with a description for the `DetectChanges` parameter.	January 8, 2021
CodePipeline now supports CloudFormation StackSets deployment actions	A new tutorial, Tutorial: Create a pipeline that uses CloudFormation StackSets as a deployment provider, provides steps to use CloudFormation StackSets to create and update your stack sets and stack instances with your pipeline. The CloudFormation StackSets action structure reference topic has also been added.	December 30, 2020
New supported Region Asia Pacific (Hong Kong)	CodePipeline is now available in Asia Pacific (Hong Kong). The Limits topic and AWS service endpoints topic have been updated.	December 22, 2020
View updated EventBridge event patterns in CodePipeline	Updated event patterns and statuses for pipeline, stage, and action level events have been added to Monitoring CodePipeline events.	December 21, 2020

View inbound pipeline executions in CodePipeline	You can use the console or the CLI to view inbound executions. For more information, see View an inbound execution (console) and View inbound execution status (CLI).	November 16, 2020
The CodeCommit source action in CodePipeline supports the full clone option	When you use a CodeCommit source action, you can choose the full clone option for using Git commands and metadata for downstream CodeBuild actions. For more information, see the CodeCommit action reference and Tutorial: Use full clone with a CodeCommit pipeline source.	November 11, 2020

CodePipeline supports connections to GitHub and GitHub Enterprise Server	You can use connections to configure AWS resources to interact with GitHub, GitHub Enterprise Cloud, and GitHub Enterprise Server. You can also choose the full clone option for using Git commands and metadata for downstream actions. For more information, see GitHub connections, GitHub Enterprise Server connections, and Tutorial: Use full clone with a GitHub pipeline source. If you have an existing pipeline with a GitHub source action, see Update a GitHub (via OAuth app) source action to a GitHub (via GitHub App) source action.	September 30, 2020
The CodeBuild action supports enabling batch builds in AWS CodePipeline	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).	July 30, 2020

AWS CodePipeline now supports AWS AppConfig deployment actions	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.	June 25, 2020
AWS CodePipeline now supports Amazon VPC in AWS GovCloud (US-West)	You can now connect directly to AWS CodePipeline through a private Amazon VPC endpoint in AWS GovCloud (US-West). For more informati on, see Use CodePipeline with Amazon Virtual Private Cloud.	June 2, 2020
AWS CodePipeline now supports AWS Step Functions invoke actions	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.	May 28, 2020
View, list, and update connections	You can list, delete, and update connections in the console. See List connections in CodePipeline.	May 21, 2020

Connections support tagging connections resources in the CLI	The connections resources now support tagging in the AWS CLI. Connections now integrate with AWS CodeGuru. See IAM Permissions Reference for Connections.	May 6, 2020
CodePipeline is now available in AWS GovCloud (US-West)	You can now use CodePipeline in AWS GovCloud (US-West). For more information, see Quotas.	April 8, 2020
The quotas topic shows which CodePipeline service quotas are configurable	The CodePipeline quotas topic has been reformatted. The documentation shows which service quotas are configurable and which quotas are non-configurable. See Quotas in AWS CodePipeline.	March 12, 2020
The Amazon ECS deployment action timeout is configurable	The Amazon ECS deployment action timeout is configurable up to one hour (the default timeout). See Quotas in AWS CodePipeline.	February 5, 2020

New topics describe how you can stop a pipeline execution	You can stop a pipeline execution in CodePipeline. You can either specify that the execution stops after in-progress actions are allowed to complete, or you can specify to stop the execution immediately and abandon in-progress actions. See How pipeline executions are stopped and Stop a pipeline execution in CodePipeline.	January 21, 2020
CodePipeline supports connections	You can use connections to configure AWS resources to interact with external code repositories. Each connection is a resource that can be used by services such as CodePipeline to connect to a third-party repository, such as Bitbucket Cloud. For more information, see Working with connections in CodePipeline.	December 18, 2019
Updated security, authentication, and access control topics	The security, authentication, and access control information for CodePipeline has been organized into a new Security chapter. For more information, see Security.	December 17, 2019

New topics describe how you can use variables in your pipelines	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.	November 14, 2019
New topics describe how pipeline executions work, why stages are locked during an execution, and when pipeline executions are superseded	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.	November 11, 2019
CodePipeline supports notification rules	You can now use notification rules to notify users of important changes in pipelines. For more information, see Create a notification rule.	November 5, 2019

CodeBuild environment variables available in CodePipeline	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.	October 14, 2019
New Region	CodePipeline is now available in Europe (Stockholm). The Limits topic and AWS service endpoints topic have been updated.	September 5, 2019
Specify canned ACLs and cache control for Amazon S3 deployment actions	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.	June 27, 2019

You can now add tags to resources in AWS CodePipeline	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.	May 15, 2019
You can now view action execution history in AWS CodePipeline	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.	March 20, 2019

AWS CodePipeline now supports publishing applications to the AWS Serverless Application Repository	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.	March 8, 2019
AWS CodePipeline now supports cross-region actions in the console	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.	February 14, 2019

AWS CodePipeline now supports Amazon S3 deployments	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.	January 16, 2019
AWS CodePipeline now supports Alexa Skills Kit deployments	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.	December 19, 2018

AWS CodePipeline now supports Amazon VPC endpoints powered by AWS PrivateLink	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 informati on, see Use CodePipeline with Amazon Virtual Private Cloud.	December 6, 2018
AWS CodePipeline now supports Amazon ECR source actions and ECS-to-Co deDeploy deployment actions	You can now use CodePipel ine and CodeDeploy with Amazon ECR and Amazon ECS for continuous deployment of container-based applicati ons. 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.	November 27, 2018

AWS CodePipeline now supports cross-region actions in a pipeline	A new topic, Add a Cross-region Action, contains steps for using the AWS CLI or 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.	November 12, 2018
AWS CodePipeline now integrates with Service Catalog	You can now add Service Catalog as a deployment action to your pipeline. This allows you to set up a pipeline to publish product updates to Service Catalog when you make a change in your source repository. The Integrations topic has been updated to reflect this support for Service Catalog. Two Service Catalog tutorials have been added to the AWS CodePipeline tutorials section.	October 16, 2018

AWS CodePipeline now integrates with AWS Device Farm	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.	July 19, 2018
AWS CodePipeline User Guide update notifications now available through RSS	The HTML version of the CodePipeline User Guide now supports an RSS feed of updates that are documente d in the Documentation Update History page. The RSS feed includes updates made after June 30, 2018 and later. Previously announced updates are still available in the Documentation Update History page. Use the RSS button in the top menu panel to subscribe to the feed.	June 30, 2018

Earlier updates

The following table describes important changes in each release of the CodePipeline User Guide on June 30, 2018 and earlier.

Change	Description	Date changed
Use webhooks to detect source changes in GitHub pipelines	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.	May 1, 2018
Updated topics	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 Source actions and change detection methods. The Tutorial: Create a simple pipeline (S3 bucket) 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, stages, and actions and Edit a pipeline in CodePipeline have also been updated. For more information, see Start a pipeline in CodePipeline.	March 22, 2018
Updated topic	CodePipeline is now available in Europe (Paris). The Quotas in AWS CodePipeline topic has been updated.	February 21, 2018
Updated topics	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 image, push it to your container registry,	December 12, 2017

Change	Description	Date changed
	and then deploy the updated image to an Amazon ECS service. The topics Product and service integrations with CodePipeline, Create a pipeline, stages, and actions, and CodePipeline pipeline structure reference have been updated to reflect this support for Amazon ECS.	
Updated topics	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 informati on about migrating your existing pipeline, see Source actions and change detection methods. The Tutorial: Create a simple pipeline (CodeCommit repository) 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, stages, and actions and Edit a pipeline in CodePipeline have also been updated. For more information, see Start a pipeline in CodePipel ine.	October 11, 2017
New and updated topics	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 has been added. For more information, see Monitoring CodePipeline events.	September 8, 2017

Change	Description	Date changed
New and updated topics	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. Informati on has been added for the PollForSourceChanges source action configuration option. For more information, see Start a pipeline in CodePipeline.	September 5, 2017
New Regions	CodePipeline is now available in Asia Pacific (Seoul) and Asia Pacific (Mumbai). The Quotas in AWS CodePipel ine topic and Regions and Endpoints topic have been updated.	July 27, 2017
New Regions	CodePipeline is now available in US West (N. California), Canada (Central), and Europe (London). The Quotas in AWS CodePipeline topic and Regions and Endpoints topic have been updated.	June 29, 2017
Updated topics	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 pipelines and details in CodePipeline, permissions reference, and Quotas in AWS CodePipeline have been updated to reflect this support.	June 22, 2017
Updated topic	Nouvola has been added to the list of available actions in Test action integrations.	May 18, 2017

Change	Description	Date changed
Updated topics	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.	April 7, 2017
Updated topics	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, Edit a pipeline in CodePipeline, and CodePipeline pipeline structure reference have been updated to reflect this support for AWS CodeBuild.	March 8, 2017
New and updated topics	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 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 integrati ng OpsWorks stacks into your pipelines.	February 8, 2017

Change	Description	Date changed
New Region	CodePipeline is now available in Asia Pacific (Tokyo). The Quotas in AWS CodePipeline topic and Regions and Endpoints topic have been updated.	December 14, 2016
New Region	CodePipeline is now available in South America (São Paulo). The Quotas in AWS CodePipeline topic and Regions and Endpoints topic have been updated.	December 7, 2016
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, Create a pipeline, stages, and actions, Authentication and Access Control, and CodePipeline pipeline structure reference have been updated to reflect this support for AWS CodeBuild. You can now use CodePipeline with CloudFormation and the AWS Serverless Application Model to continuously deliver your serverless applications. The topic Product and service integrations with CodePipeline has been updated to reflect this support. Product and service integrations with CodePipeline has been reorganized to group AWS and partner offerings by action type.	December 1, 2016
New Region	CodePipeline is now available in Europe (Frankfurt). The Quotas in AWS CodePipeline topic and Regions and Endpoints topic have been updated.	November 16, 2016

Change	Description	Date changed
Updated topics	AWS CloudFormation can now be selected as a deployment provider in pipelines, enabling you to take action on CloudFormation stacks and change sets as part of a pipeline execution. The topics Product and service integrations with CodePipeline, Create a pipeline, stages, and actions, Authentication and Access Control, and CodePipeline pipeline structure reference have been updated to reflect this support for AWS CloudFormation.	November 3, 2016
New Region	CodePipeline is now available in the Asia Pacific (Sydney) Region. The Quotas in AWS CodePipeline topic and Regions and Endpoints topic have been updated.	October 26, 2016
New Region	CodePipeline is now available in Asia Pacific (Singapore). The Quotas in AWS CodePipeline topic and Regions and Endpoints topic have been updated.	October 20, 2016
New Region	CodePipeline is now available in the US East (Ohio) Region. The Quotas in AWS CodePipeline topic and Regions and Endpoints topic have been updated.	October 17, 2016
Updated topic	Create a pipeline, stages, and actions has been updated to reflect support for displaying version identifiers of custom actions in the **Source provider** and **Build provider** lists.	September 22, 2016
Updated topic	The Add a manual approval action to a stage 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.	September 14, 2016

Change	Description	Date changed
New and updated topics	A new topic that 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 troublesh ooting failures in your pipeline. A new section, Monitoring pipelines, provides a central location for all topics related to monitoring the status and progress of your pipelines.	September 08, 2016
New and updated topics	A new section, Add a manual approval action to a stage, 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.	July 06, 2016
New Region	CodePipeline is now available in the Europe (Ireland) Region. The Quotas in AWS CodePipeline topic and Regions and endpoints topic have been updated.	June 23, 2016
New topic	A new topic, , has been added to describe how to retry a failed action or a group of parallel failed actions in stage.	June 22, 2016

Change	Description	Date changed
Updated topics	A number of topics, including Create a pipeline, stages, and actions, Authentication and Access Control, CodePipeline pipeline structure reference, and Product and service integrations with CodePipeline, 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 OpsWorks is currently available in the US East (N. Virginia) Region (eusc-de-east-1) only.	June 2, 2016
New and updated topics	A new topic, Tutorial: Create a simple pipeline (CodeCommit repository), 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, Tutorial: Create a four-stage pipeline, and Troubleshooting CodePipeline.	April 18, 2016
New topic	A new topic, Invoke an AWS Lambda function in a pipeline in CodePipeline, has been added. This topic contains sample AWS Lambda functions and steps for adding Lambda functions to pipelines.	January 27, 2016
Updated topic	A new section has been added to Authentication and Access Control, Resource-based Policies.	January 22, 2016
New topic	A new topic, Product and service integrations with CodePipeline, 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.	December 17, 2015

Change	Description	Date changed
Updated topic	Details of integration with Solano CI have been added to Product and service integrations with CodePipeline.	November 17, 2015
Updated topic	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 Tutorial: Create a four-stage pipeline.	November 9, 2015
New Region	CodePipeline is now available in the US West (Oregon) Region. The Quotas in AWS CodePipeline topic has been updated. Links have been added to Regions and Endpoints.	October 22, 2015
New topic	Two new topics, Configure server-side encryption for artifacts stored in Amazon S3 for CodePipeline and Create a pipeline in CodePipeline that uses resources from another AWS account, have been added. A new section has been added to Authentication and Access Control.	August 25, 2015
Updated topic	The Create and add a custom action in CodePipeline topic has been updated to reflect changes in the structure, including `inputArtifactDetails` and `outputArtifactDetails` .	August 17, 2015
Updated topic	The Troubleshooting CodePipeline topic has been updated with revised steps for troubleshooting problems with the service role and Elastic Beanstalk.	August 11, 2015
Updated topic	The Authentication and Access Control topic has been updated with the latest changes to the service role for CodePipeline.	August 6, 2015

Change	Description	Date changed
New topic	A Troubleshooting CodePipeline topic has been added. Updated steps have been added for IAM roles and Jenkins in Tutorial: Create a four-stage pipeline.	July 24, 2015
Topic update	Updated steps have been added for downloading the sample files in Tutorial: Create a simple pipeline (S3 bucket) and Tutorial: Create a four-stage pipeline.	July 22, 2015
Topic update	A temporary workaround for download issues with the sample files was added in Tutorial: Create a simple pipeline (S3 bucket).	July 17, 2015
Topic update	A link was added in Quotas in AWS CodePipeline to point to information about which limits can be changed.	July 15, 2015
Topic update	The managed policies section in Authentication and Access Control was updated.	July 10, 2015
Initial Public Release	This is the initial public release of the CodePipeline User Guide.	July 9, 2015

CodePipeline feature reference

This section is a high-level reference for features in CodePipeline documentation. For a conceptual overview of the pipeline structure, see CodePipeline pipeline structure reference.

This table is updated periodically with feature reference information.

Date of table update	Feature	CodePipeline User Guide	CodePipeline API Guide	CloudFormation Reference	AWS CDK Reference - L1 constructs
October 15, 2024	Automatic retry on stage failure	Configure a stage for automatic retry on failure	RetryConfiguration		
October 15, 2024	Skip result for beforeEntry conditions	Creating Entry conditions with Skip result and `VariableCheck` rule (console)	FailureConditions		
October 03, 2024	Commands action with new Compute category	Commands action reference	ActionDeclaration		
August 28, 2024	beforeEntry type for stage conditions	Configure conditions for a stage	BeforeEntryConditions	AWS::CodePipeline::Pipeline	beforeEntry

Date of table update	Feature	CodePipeline User Guide	CodePipeline API Guide	CloudFormation Reference	AWS CDK Reference - L1 constructs
				BeforeEntryConditions	
August 28, 2024	onSuccess type for stage conditions	Configure conditions for a stage	SuccessConditions	AWS::CodePipeline::Pipeline::SuccessConditions	onSuccess
April 26, 2024	Stage rollbacks and onFailure type for stage conditions	Configuring stage rollback	RollbackStage	onFailure	onFailure
February 08, 2024	PARALLEL and QUEUED execution modes	Set or change the pipeline execution mode	PipelineExecution	Execution Mode	Execution mode
November 17, 2023	Source overrides	Start a pipeline with a source revision override	SourceRevisionOverride	N/A	N/A
October 24, 2023	Pipeline types	What type of pipeline is right for me?	PipelineDeclaration	PipelineType	enum PipelineType

Date of table update	Feature	CodePipeline User Guide	CodePipeline API Guide	CloudFormation Reference	AWS CDK Reference - L1 constructs
October 24, 2023	Pipeline-level variables	Tutorial: Use pipeline-level variables	PipelineVariable	AWS::CodePipeline::Pipeline::PipelineVariableDeclaration	Pipeline-level variables
October 24, 2023	Triggers and filtering file paths / branches / pull requests	Automate starting pipelines using triggers and filtering Tutorial: Filter on branch names for pull requests to start your pipeline (V2 type)	ActionDeclaration	AWS::CodePipeline::Pipeline::PipelineTriggerDeclaration	Trigger
October 17, 2023	Retry a stage	Configuring stage retry for a failed stage or failed actions	ActionDeclaration	N/A	N/A