Using and Managing vRealize Automation Code Stream

14 December 2022

vRealize Automation 8.6

You can find the most up-to-date technical documentation on the VMware website at:

https://docs.vmware.com/

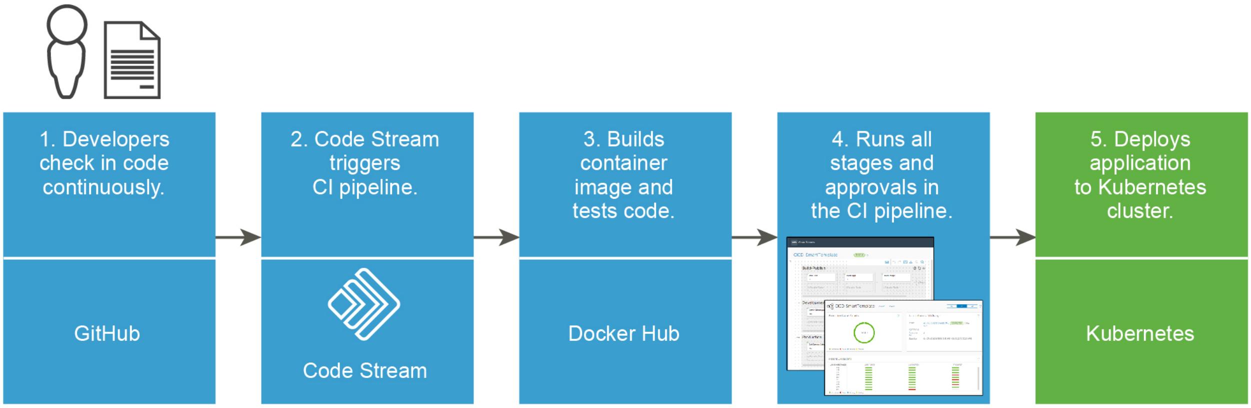
VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

What is Code Stream and how does it work

1

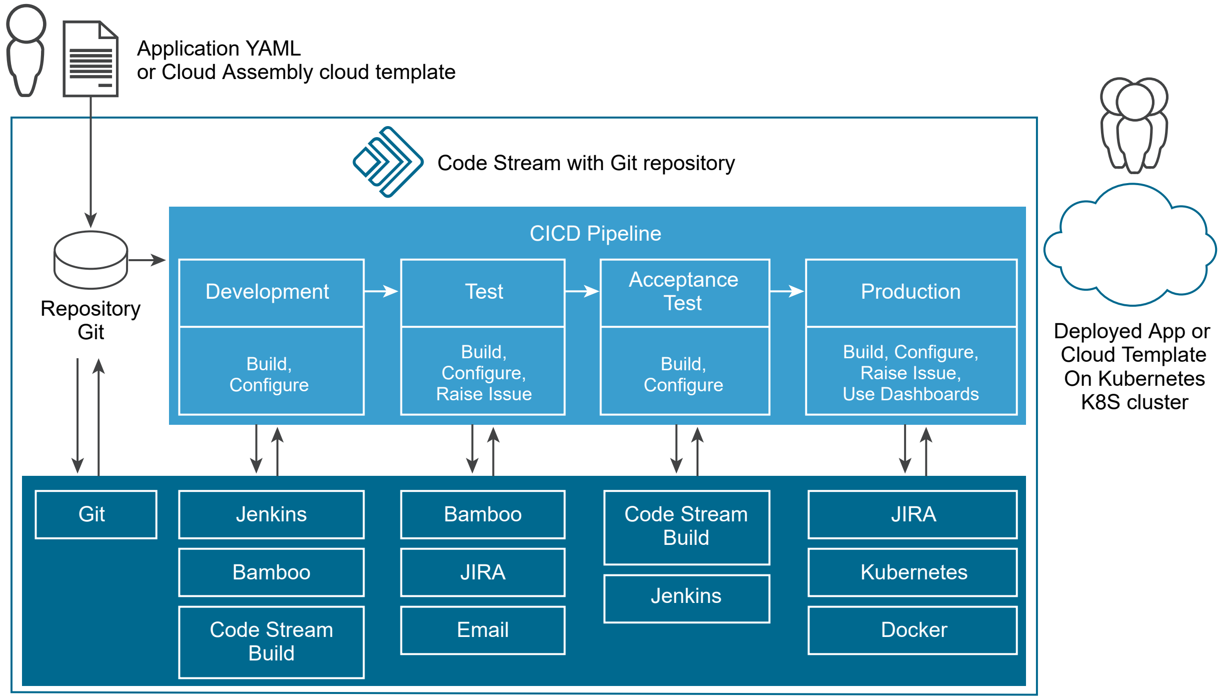
vRealize Automation Code Stream™ is a continuous integration and continuous delivery (CICD) tool. By creating pipelines that model the software release process in your DevOps lifecycle, you build the code infrastructure that delivers your software rapidly and continuously.


1. Developers check in code continuously.
GitHub
2. Code Stream triggers CI pipeline.
Code Stream
3. Builds container image and tests code.
Docker Hub
4. Runs all stages and approvals in the CI pipeline.
5. Deploys application to Kubernetes cluster.
Kubernetes

When you use Code Stream to deliver your software, you integrate two of the most important parts of your DevOps lifecycle: your release process and your developer tools. After the initial setup, which integrates Code Stream with your existing development tools, the pipelines automate your entire DevOps lifecycle.

Starting with vRealize Automation 8.2, Blueprints are called VMware Cloud Templates.

You create a pipeline that builds, tests, and releases your software. Code Stream uses that pipeline to progress your software from the source code repository, through testing, and on to production.



You can learn more about planning your continuous integration and continuous delivery pipelines at Chapter 4 Planning to natively build, integrate, and deliver your code in Code Stream .

How Code Stream Administrators use Code Stream

As an administrator, you create endpoints and ensure that working instances are available for developers. You can create, trigger, and manage pipelines, and more. You have the `Administrator` role, as described in How do I manage user access and approvals in Code Stream.

Table 1-1. How Code Stream Administrators support developers

To support developers...	Here's what you can do...
Provide and manage environments.	Create environments for developers to test and deploy their code. ■ Track status and send email notifications. ■ Keep your developers productive by ensuring that their environments continuously work. To find out more, see More resources for Code Stream Administrators and Developers. Also see Chapter 5 Tutorials for using Code Stream.
Provide endpoints.	Ensure that developers have working instances of endpoints that can connect to their pipelines.
Provide integrations with other services.	Ensure that integrations to other services are working. To find out more, see VMware Cloud Services documentation.
Create pipelines.	Create pipelines that model release processes. To find out more, see Chapter 3 Creating and using pipelines in Code Stream.

Table 1-1. How Code Stream Administrators support developers (continued)

To support developers...	Here's what you can do...
Trigger pipelines.	Ensure that pipelines run when events occur. ■ To trigger a standalone, continuous delivery (CD) pipeline whenever a build artifact is created or updated, use the Docker trigger. ■ To trigger a pipeline when a developer commits changes to their code, use the Git trigger. ■ To trigger a pipeline when developers review code, merge, and more, use the Gerrit trigger. ■ To run a standalone continuous delivery (CD) pipeline whenever a build artifact is created or updated, use the Docker trigger. To find out more, see Chapter 7 Triggering pipelines in Code Stream.
Manage pipelines and approvals.	Stay up-to-date on pipelines. ■ View pipeline status, and see who ran the pipelines. ■ View approvals on pipeline executions, and manage approvals for active and inactive pipeline executions. To find out more, see What are user operations and approvals in Code Stream. Also, see How do I use custom dashboards to track key performance indicators for my pipeline in Code Stream.
Monitor developer environments.	Create custom dashboards that monitor pipeline status, trends, metrics, and key indicators. Use the custom dashboards to monitor pipelines that pass or fail in developer environments. You can also identify and report on under used resources, and free up resources. You can also see: ■ How long a pipeline ran before it succeeded. ■ How long a pipeline waited for approval, and notify the user who must approve it. ■ Stages and tasks that fail most often. ■ Stages and tasks that take the most time to run. ■ Releases that development teams have in progress. ■ Applications that succeeded in being deployed and released. To find out more, see Chapter 8 Monitoring pipelines in Code Stream.
Troubleshoot problems.	Troubleshoot and resolve pipeline failures in developer environments. ■ Identify and resolve problems in continuous integration and continuous delivery environments (CICD). ■ Use the pipeline dashboards and create custom dashboards to see more. See Chapter 8 Monitoring pipelines in Code Stream. Also, see Chapter 2 Setting up Code Stream to model my release process.

Code Stream is part of VMware Cloud Services.

- Use Cloud Assembly to deploy cloud templates.
- Use Service Broker to get cloud templates from the catalog.

To learn about other things you can do, see VMware vRealize Automation Documentation.

How Developers Use Code Stream

As a developer, you use Code Stream to build and run pipelines, and monitor pipeline activity on the dashboards. You have the `User` role, as described in How do I manage user access and approvals in Code Stream.

After you run a pipeline, you'll want to know:

- If your code succeeded through all stages of the pipeline. To find out, observe the results in the pipeline executions.
- What to do if the pipeline failed, and what caused the failure. To find out, observe the top errors in the pipeline dashboards.

Table 1-2. Developers who use Code Stream

To integrate and release your code	Here's what you do
Build pipelines.	Test and deploy your code. Update your code when a pipeline fails.
Connect your pipeline to endpoints.	Connect the tasks in your pipeline to endpoints, such as a GitHub repository.
Run pipelines.	Add a user operation approval task so that another user can approve your pipeline at specific points.
View dashboards.	View the results on the pipeline dashboard. You can see trends, history, failures, and more.

For more information about getting started, see Getting Started with VMware Code Stream.

Find more documentation in the In-product Support panel

If you don’t find the information you need here, you can get more help in the product. ?

- Click and read the signposts and tooltips in the user interface to get the context-specific information that you need where and when you need it.
- Open the In-product support panel and read the topics that appear for the active user interface page. You can also search in the panel to get answers to questions.

More on Webhooks

You can create multiple webhooks for different branches by using the same Git endpoint and providing different values for the branch name in the webhook configuration page. To create another webhook for another branch in the same Git repository, you don't need to clone the Git endpoint multiple times for multiple branches. Instead, you provide the branch name in the webhook, which allows you to reuse the Git endpoint. If the branch in the Git webhook is the same as the branch in the endpoint, you don't need to provide branch name in the Git webhook page.

Setting up Code Stream to model my release process

2

To model your release process, you create a pipeline that represents the stages, tasks, and approvals that you normally use for releasing your software. Code Stream then automates the process that builds, tests, approves, and deploys your code.

Now that you have everything for modeling your software release process, here's how you do it in Code Stream.

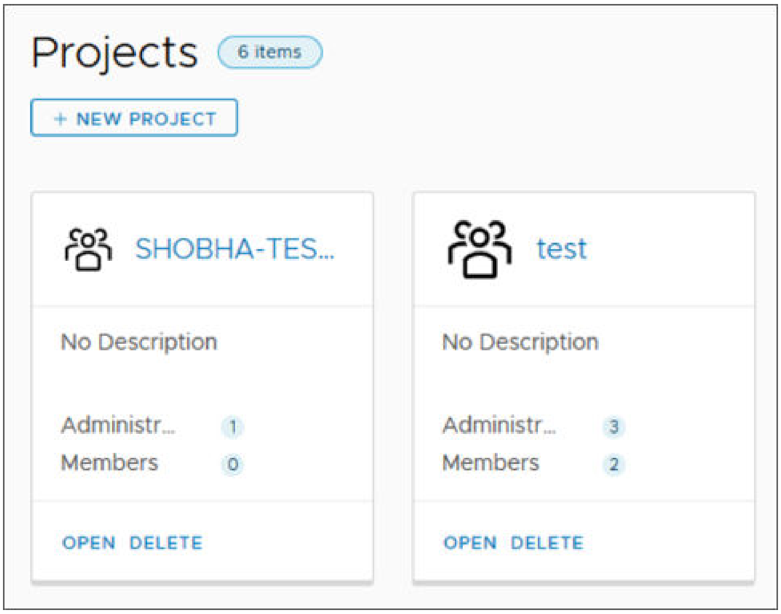
Prerequisites

- Verify whether any endpoints are already available. In Code Stream, click **Endpoints**.
- Learn about native ways that you can build and deploy your code. See Chapter 4 Planning to natively build, integrate, and deliver your code in Code Stream .
- Determine whether some of the resources that you will use in your pipeline must be marked as restricted. See How do I manage user access and approvals in VMware Code Stream.
- If you have the user role or the viewer role instead of the administrator role, determine who is the administrator for your Code Stream instance.

Procedure

1 Examine the projects available in Code Stream and select one that is right for you.

   - If no projects appear, ask a Code Stream administrator who can create a project and make you a member of the project. See How do I add a project in Code Stream.
   - If you are not a member of any projects listed, ask a Code Stream administrator who can add you as a member of a project.



2 Add any new endpoints that you need for your pipeline.

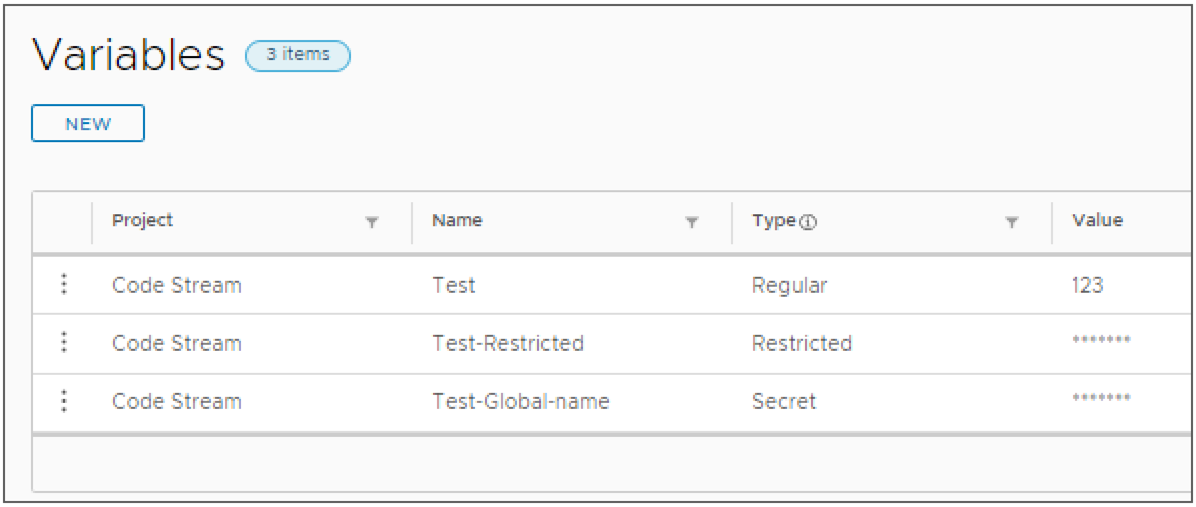
For example, you might need Git, Jenkins, Code Stream Build, Kubernetes, and Jira.

3 Create variables so that you can reuse values in your pipeline tasks.

To constrain the resources used in your pipelines, such as a host machine, use restricted variables. You can restrict the pipeline from continuing to run until another user explicitly approves it.

Administrators can create secret variables and restricted variables. Users can create secret variables.

You can reuse a variable as many times as you want across multiple pipelines. For example, a variable that defines a host machine can be `HostIPAddress`. To use the variable in a pipeline task, you enter `${var.HostIPAddress}`.



	Project	Name	Type	Value
⋮	Code Stream	Test	Regular	123
⋮	Code Stream	Test-Restricted	Restricted	*******
⋮	Code Stream	Test-Global-name	Secret	*******

4 If you are an administrator, mark any endpoints and variables that are vital to your business as restricted resources.

When a user who is not an administrator attempts to run a pipeline that includes a restricted resource, the pipeline stops at the task that uses the restricted resource. Then, an administrator must resume the pipeline.

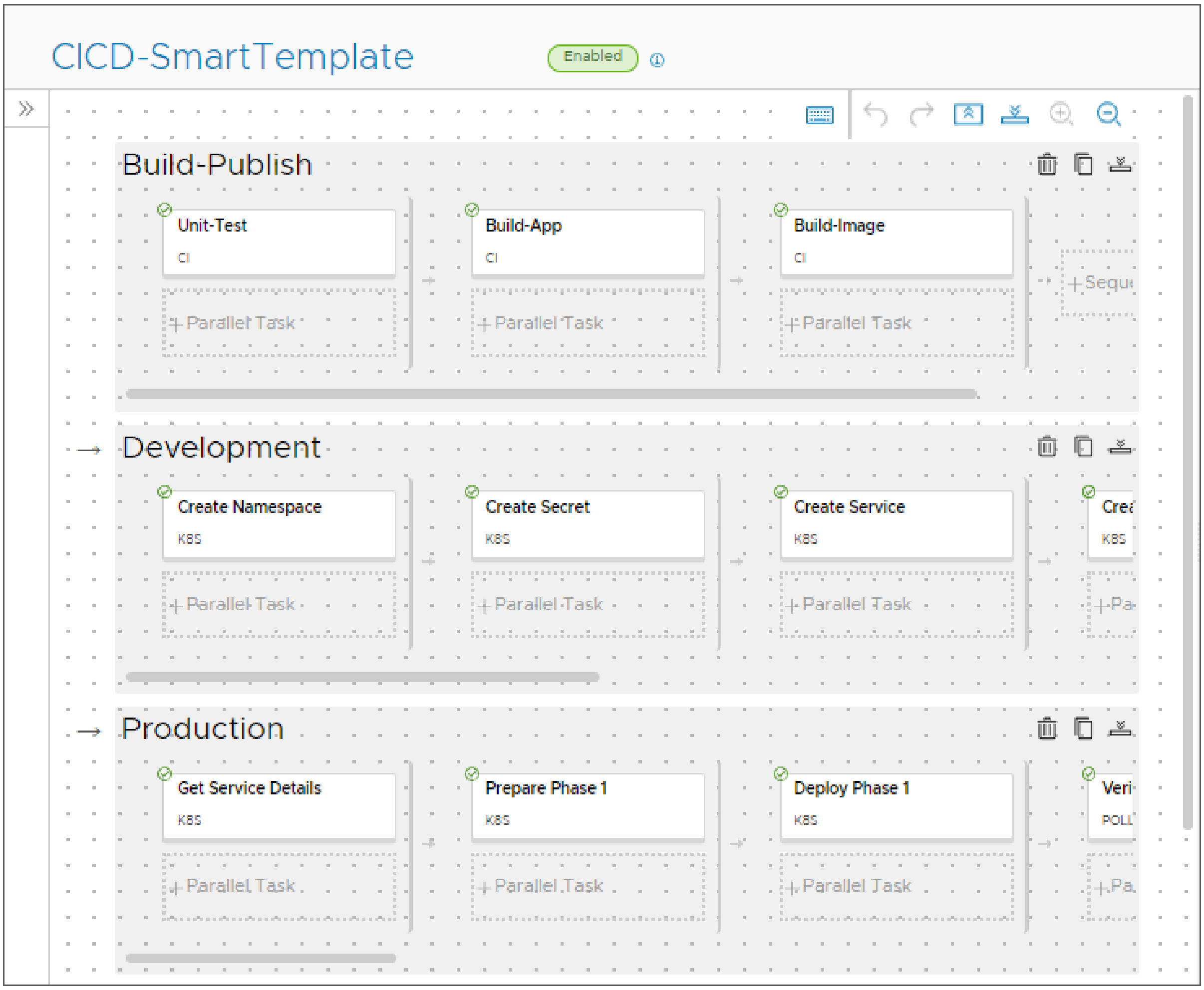
5 Plan the build strategy for your native CICD, CI, or CD pipeline.

Before you create a pipeline that continuously integrates (CI) and continuously deploys (CD) your code, plan your build strategy. The build plan helps you determine what Code Stream needs so that it can natively build, integrate, test, and deploy your code.

How to create a Code Stream native build	Results in this build strategy
Use one of the smart pipeline templates.	■ Builds all the stages and tasks for you. ■ Clones the source repository. ■ Builds and tests your code. ■ Containerizes your code for deployment. ■ Populates the pipeline task steps based on your selections.
Add stages and tasks manually.	You add stages, add tasks, and enter the information that populates them.

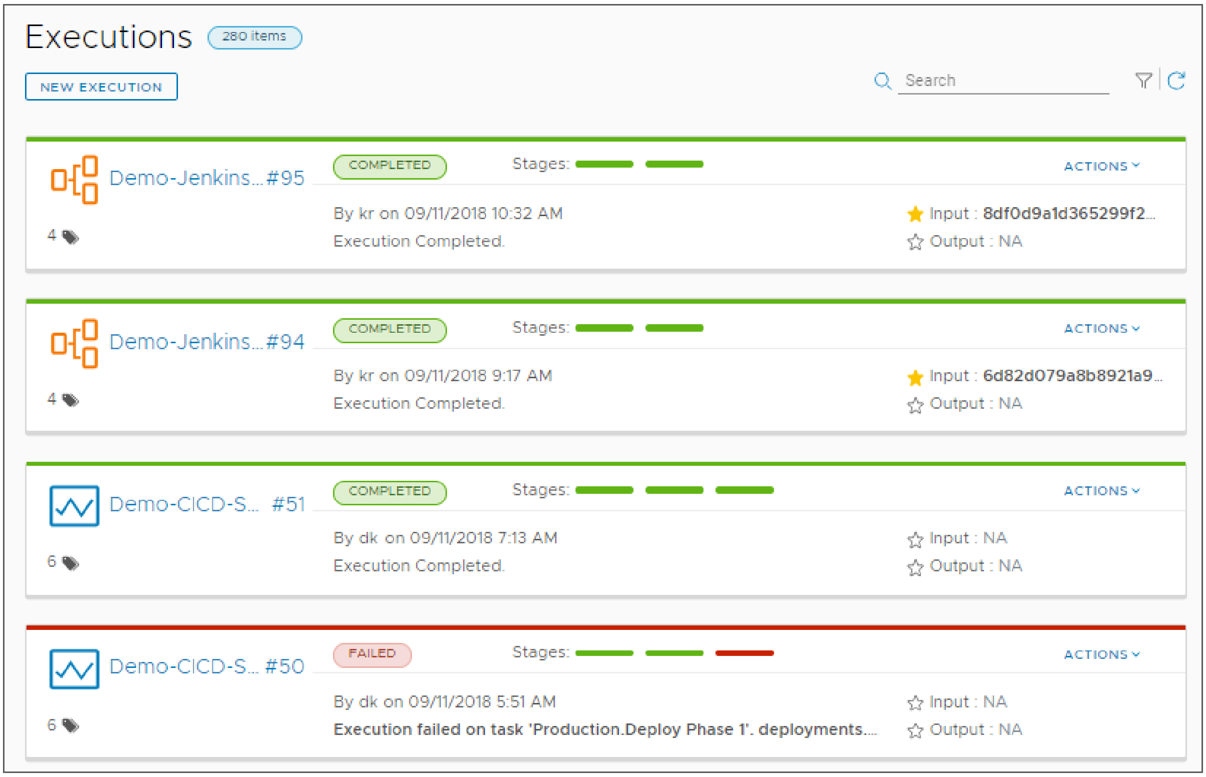
6 Create your pipeline by using a smart pipeline template, or by manually add stages and tasks to the pipeline.

Then, you mark any resources as restricted. Add approvals where needed. Apply any regular, restricted, or secret variables. Add any bindings between tasks.



7 Validate, enable, and run your pipeline.

8 View the pipeline executions.



Executions 280 items

NEW EXECUTION

Search

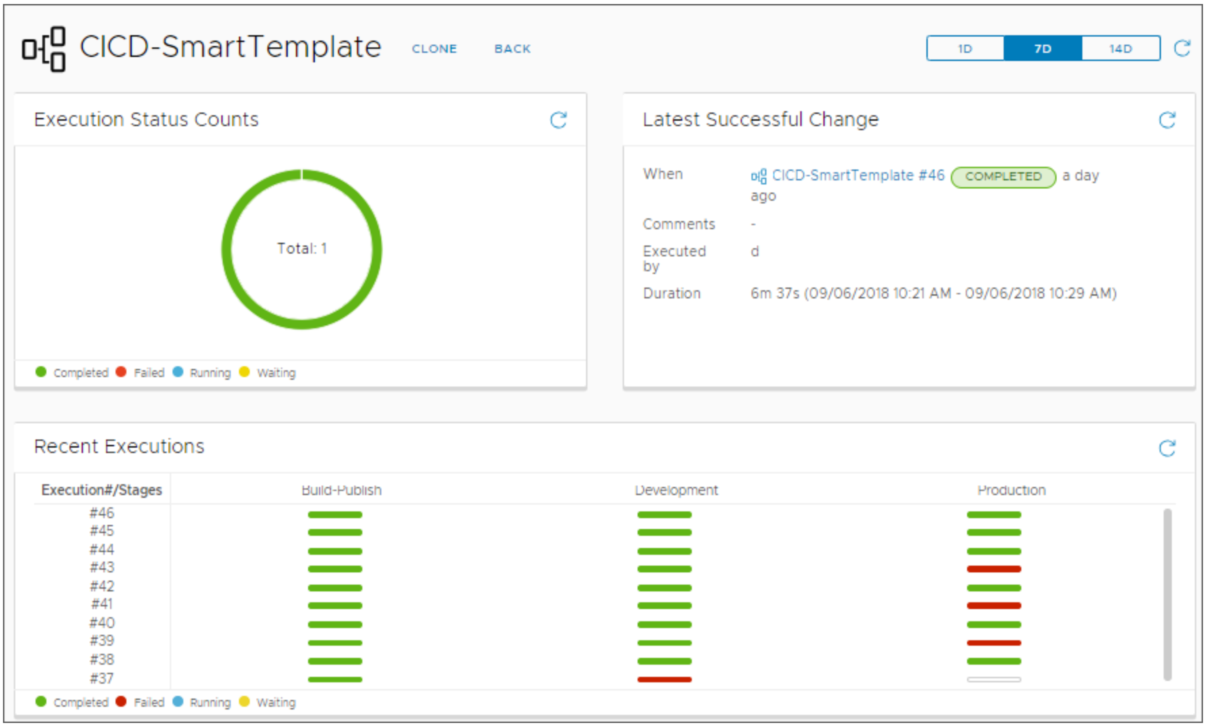
Demo-Jenkins...#95 COMPLETED Stages: 🟩 🟩 ACTIONS
4
By kr on 09/11/2018 10:32 AM
Execution Completed.
Input : 8df0d9a1d365299f2...
Output : NA

Demo-Jenkins...#94 COMPLETED Stages: 🟩 🟩 ACTIONS
4
By kr on 09/11/2018 9:17 AM
Execution Completed.
Input : 6d82d079a8b8921a9...
Output : NA

Demo-CICD-S... #51 COMPLETED Stages: 🟩 🟩 🟩 ACTIONS
6
By dk on 09/11/2018 7:13 AM
Execution Completed.
Input : NA
Output : NA

Demo-CICD-S... #50 FAILED Stages: 🟩 🟩 🟥 ACTIONS
6
By dk on 09/11/2018 5:51 AM
Execution failed on task 'Production.Deploy Phase 1'. deployments...
Input : NA
Output : NA

9 To track status and key performance indicators (KPIs), use the pipeline dashboards, and create any custom dashboards.



CICD-SmartTemplate CLONE BACK 1D 7D 14D

Execution Status Counts

Total: 1

● Completed ● Failed ● Running ● Waiting

Latest Successful Change

When CICD-SmartTemplate #46 COMPLETED a day ago
Comments -
Executed by d
Duration 6m 37s (09/06/2018 10:21 AM - 09/06/2018 10:29 AM)

Recent Executions

Execution#/Stages	Build-Publish	Development	Production
#46	🟩	🟩	🟩
#45	🟩	🟩	🟩
#44	🟩	🟩	🟩
#43	🟩	🟩	🟥
#42	🟩	🟩	🟩
#41	🟩	🟩	🟥
#40	🟩	🟩	🟩
#39	🟩	🟩	🟥
#38	🟩	🟩	🟩
#37	🟩	🟥	

● Completed ● Failed ● Running ● Waiting

Results

You created a pipeline that you can use in the selected project.

You can also export your pipeline YAML, then import it and reuse it in other projects.

What to do next

Learn about use cases that you might want to apply in your environment. See Chapter 5 Tutorials for using Code Stream.

How do I add a project in Code Stream

You create a project and add administrators and members to it. Project members can use features such as creating a pipeline and adding an endpoint. To create, delete, or update a project for a development team, you must be a Code Stream administrator.

A project must exist before you can create a pipeline. When you create a pipeline, you select a project that groups all your pipeline information together. Definitions for endpoints and variables also depend on an existing project.

Prerequisites

- Verify that you have the Code Stream administrator role. See What are Roles in Code Stream.

  If you do not have the Code Stream administrator role, but you have Cloud Assembly administrator role, you can create, update, or delete projects in the Cloud Assembly UI. See "How do I add a project for my Cloud Assembly development team" in *Using and Managing vRealize Automation Cloud Assembly*.

- If you are adding Active Directory groups to projects, verify that you configured Active Directory groups for your organization. See "How do I enable Active Directory groups in vRealize Automation for projects" in *Administering vRealize Automation*. If the groups are not synchronized, they are not available when you try to add them to a project.

Procedure

1. Select **Projects**, and click **New Project**.
2. Enter the project name.
3. Click **Create**.
4. Select the card for the newly created project, and click **Open**.
5. Click the **Users** tab and add users and assign roles.

   - The project administrator can add members.
   - The project member who has a service role can use services.
   - The project viewer can see projects but cannot create, update, or delete them.

   For more information about project roles, see How do I manage user access and approvals in Code Stream.

6 Click **Save**.

What to do next

Add endpoints and pipelines that use the project. See and Chapter 6 Connecting Code Stream to endpoints and Chapter 3 Creating and using pipelines in Code Stream.

After you create a pipeline, the name of the project that groups all your pipeline information together appears on pipeline cards and pipeline execution cards.

How do I manage user access and approvals in Code Stream

Code Stream provides several ways to ensure that users have the appropriate authorization and consent to work with pipelines that release your software applications.

Each member on a team has an assigned role, which gives specific permissions on pipelines, endpoints, and dashboards, and the ability to mark resources as restricted.

User operations and approvals enable you to control when a pipeline runs and must stop for an approval. Your role determines whether you can resume a pipeline, and run pipelines that include restricted endpoints or variables.

Use secret variables to hide and encrypt sensitive information. Use restricted variable for strings, passwords, and URLs that must be hidden and encrypted, and to restrict use in executions. For example, use a secret variable for a password or URL. You can use secret and restricted variables in any type of task in your pipeline.

What are Roles in Code Stream

Depending on your role in Code Stream, you can perform certain actions and access certain areas. For example, your role might enable you to create, update, and run pipelines. Or, you might only have permission to view pipelines.

`All actions except restricted` means this role has permission to perform create, read, update, and delete actions on entities except for restricted variables and endpoints.

Table 2-1. Service and Project level access permissions in Code Stream

	Code Stream Roles				
Access levels	**Code Stream Administrator**	**Code Stream Developer**	**Code Stream Executor**	**Code Stream Viewer**	**Code Stream User**
Code Stream service level access	All Actions	All actions except restricted	Execution actions	Read only	None
Project level access: Project Admin	All Actions	All Actions	All Actions	All Actions	All Actions

Table 2-1. Service and Project level access permissions in Code Stream (continued)

	Code Stream Roles				
Access levels	**Code Stream Administrator**	**Code Stream Developer**	**Code Stream Executor**	**Code Stream Viewer**	**Code Stream User**
Project level access: Project Member	All Actions	All actions except restricted	All actions except restricted	All actions except restricted	All actions except restricted
Project level access: Project Viewer	All Actions	All actions except restricted	Execution actions	Read only	Read only

Users who have the Project Admin role can perform all actions on projects where they are a Project administrator.

A Project administrator can create, read, update, and delete pipelines, variables, endpoints, dashboards, triggers, and start a pipeline that includes restricted endpoints or variables if these resources are in the project where the user is a Project administrator.

Users who have the Service Viewer role can see all the information that is available to the administrator. They cannot take any action unless an administrator makes them a project administrator or a project member. If the user is affiliated with a project, they have the permissions related to the role. The project viewer would not extend their permissions the way that the administrator or member role does. This role is read-only across all projects.

If you have read permissions in a project, you can still see restricted resources.

- To see restricted endpoints, which display a lock icon on the endpoint card, click **Configure > Endpoints**.
- To see restricted and secret variables, which display RESTRICTED or SECRET in the **Type** column, click **Configure > Variables**.

Table 2-2. Code Stream service role capabilities

UI Context	Capabilities	Code Stream Administrator role	Code Stream Developer role	Code Stream Executor role	Code Stream Viewer role	Code Stream User role
Pipelines						
	View pipelines	Yes	Yes	Yes	Yes	
	Create pipelines	Yes	Yes			
	Run pipelines	Yes	Yes	Yes		
	Run pipelines that include restricted endpoints or variables	Yes				

Table 2-2. Code Stream service role capabilities (continued)

UI Context	Capabilities	Code Stream Administrator role	Code Stream Developer role	Code Stream Executor role	Code Stream Viewer role	Code Stream User role
	Update pipelines	Yes	Yes			
	Delete pipelines	Yes	Yes			
Pipeline Executions						
	View pipeline executions	Yes	Yes	Yes	Yes	
	Resume, pause, and cancel pipeline executions	Yes	Yes	Yes		
	Resume pipelines that stop for approval on restricted resources	Yes				
Custom Integrations						
	Create custom integrations	Yes	Yes			
	Read custom integrations	Yes	Yes	Yes	Yes	
	Update custom integrations	Yes	Yes			
Endpoints						
	View executions	Yes	Yes	Yes	Yes	
	Create executions	Yes	Yes			
	Update executions	Yes	Yes			
	Delete executions	Yes	Yes			
Mark resources as restricted						
	Mark an endpoint or variable as restricted	Yes				

Table 2-2. Code Stream service role capabilities (continued)

UI Context	Capabilities	Code Stream Administrator role	Code Stream Developer role	Code Stream Executor role	Code Stream Viewer role	Code Stream User role
Dashboards						
	View dashboards	Yes	Yes	Yes	Yes	
	Create dashboards	Yes	Yes			
	Update dashboards	Yes	Yes			
	Delete dashboards	Yes	Yes			

Custom roles and permissions in Code Stream

You can create custom roles in Cloud Assembly that extend privileges to users who work with pipelines. When you create a custom role for Code Stream pipelines, you select one or more **Pipeline** permissions.

Select the minimal number of **Pipeline** permissions required for users who will be assigned this custom role.

When a user is assigned to a project and given a role in that project, and that user is assigned a custom role that includes one or more **Pipeline** permissions, they can perform all the actions that the permissions allow. For example, they can create restricted variables, manage restricted pipelines, create and manage custom integrations, and more.

Table 2-3. Pipeline permissions that you can assign to custom roles

Pipeline Permission	Code Stream Administrator	Code Stream Developer	Code Stream Executor	Code Stream Viewer	Code Stream User	Project Administrator	Project Member	Project Viewer
Manage Pipelines	Yes	Yes				Yes	Yes	
Manage Restricted Pipelines	Yes					Yes		
Manage Custom Integrations	Yes	Yes						
Execute Pipelines	Yes	Yes	Yes			Yes	Yes	
Execute Restricted Pipelines	Yes					Yes		

Table 2-3. Pipeline permissions that you can assign to custom roles (continued)

Pipeline Permission	Code Stream Administrator	Code Stream Developer	Code Stream Executor	Code Stream Viewer	Code Stream User	Project Administrator	Project Member	Project Viewer
Manage Executions	Yes					Yes		
Read. This permission is not visible.	Yes	Yes	Yes	Yes		Yes	Yes	Yes

Table 2-4. How you can use Pipeline permissions with custom roles

Permission	What you can do
Manage Pipelines	■ Create, update, delete, clone pipelines. ■ Release and unrelease pipelines to VMware Service Broker. ■ Create, update, and delete endpoints. ■ Create, update, and delete regular and secret variables. ■ Create, clone, update, and delete a Gerrit listener. ■ Connect and disconnect a Gerrit listener. ■ Create, clone, update, delete a Gerrit trigger. ■ Create, update, and delete a Git webhook. ■ Create, update, and delete a Docker webhook. ■ Use smart pipeline templates to create pipelines. ■ Import pipelines from YAML, and export them to YAML. ■ Create, update, and delete custom dashboards. ■ Read all custom integrations. ■ Read all restricted endpoints and variables, but cannot view their values.
Manage Restricted Pipelines	■ Create, update, and delete endpoints. ■ Mark endpoints as restricted, update restricted endpoints, and delete them. ■ Create, update, and delete regular and secret variables. ■ Create, update, and delete restricted variables. ■ All permissions that you can do with Manage Pipelines.
Manage Custom Integrations	■ Create and update custom integrations. ■ Version and release custom integrations. ■ Delete and deprecate custom integration versions. ■ Delete custom integrations.
Execute Pipelines	■ Run pipelines. ■ Pause, resume, and cancel pipeline executions. ■ Rerun pipeline executions. ■ Resume, rerun, and manually trigger a Gerrit trigger event. ■ Approve a user operation, and can do batch approvals of user operations.

Table 2-4. How you can use Pipeline permissions with custom roles (continued)

Permission	What you can do
Execute Restricted Pipelines	■ Run pipelines. ■ Pause, resume, cancel, and delete pipeline executions. ■ Rerun pipeline executions. ■ Sync a running pipeline execution. ■ Force delete a running pipeline execution. ■ Resume, rerun, delete, and manually trigger a Gerrit trigger event. ■ Resolve restricted items and continue the pipeline execution. ■ Switch user context and continue the pipeline execution after a User Operation task approval. ■ All permissions that you can do with Execute Pipelines.
Manage Executions	■ Run pipelines. ■ Pause, resume, cancel, and delete pipeline executions. ■ Rerun pipeline executions. ■ Resume, rerun, delete, and manually trigger a Gerrit trigger event. ■ All permissions that you can do with Execute Pipelines.

Custom roles can include combinations of permissions. These permissions are organized into groups of capabilities that enable users to manage or run pipelines, with and without restricted resources. These permissions represent all the capabilities that each role can perform in Code Stream.

For example, if you create a custom role and include the permission called **Manage Restricted Pipelines**, users who have the Code Stream Developer role can:

- Create, update, and delete endpoints.
- Mark endpoints as restricted, update restricted endpoints, and delete them.
- Create, update, and delete regular and secret variables.
- Create, update, and delete restricted variables.

Table 2-5. Example combinations of Pipeline permissions in custom roles

Number of Permissions Assigned to Custom Role	Examples of Combined Permissions	How to use this combination
Single permission	**Execute Pipelines**	
Two permissions	**Manage Pipelines** and **Execute Pipelines**	
Three permissions	**Manage Pipelines** and **Execute Pipelines** and **Execute Restricted Pipelines**	
	Manage Pipelines and **Manage Custom Integrations** and **Execute Restricted Pipelines**	This combination might apply to a Code Stream Developer role but be limited to the projects where the user is a member.

Table 2-5. Example combinations of Pipeline permissions in custom roles (continued)

Number of Permissions Assigned to Custom Role	Examples of Combined Permissions	How to use this combination
	Manage Pipelines and **Manage Custom Integrations** and **Manage Executions**	This combination might apply to a Code Stream Administrator but limited to the projects where user is a member.
	Manage Pipelines, **Manage Restricted Pipelines**, and **Manage Custom Integrations**	With this combination, a user has full permissions and can create and delete anything in Code Stream.

If you have the Administrator role

As an administrator, you can create custom integrations, endpoints, variables, triggers, pipelines, and dashboards.

Projects enable pipelines to access infrastructure resources. Administrators create projects so that users can group pipelines, endpoints, and dashboards together. Users then select the project in their pipelines. Each project includes an administrator and users with assigned roles.

With the Administrator role, you can mark endpoints and variables as restricted resources, and you can run pipelines that use restricted resources. If a non-administrative user runs the pipeline that includes a restricted endpoint or variable, the pipeline will stop at the task where the restricted variable is used, and an administrator must resume the pipeline.

As an administrator, you can also request that pipelines be published in vRealize Automation Service Broker.

If you have the Developer role

You can work with pipelines like an administrator can, except that you cannot work with restricted endpoints or variables.

If you run a pipeline that uses restricted endpoints or variables, the pipeline only runs up to the task that uses the restricted resource. Then, it stops, and a Code Stream administrator or project administrator must resume the pipeline.

If you have the User role

You can access Code Stream, but do not have any privileges as the other roles provide.

If you have the Viewer role

You can see the same resources that an administrator sees, such as pipelines, endpoints, pipeline executions, dashboards, custom integrations, and triggers, but you cannot create, update, or delete them. To perform actions, the Viewer role must also be given the project administrator or project member role.

Users who have the Viewer role can see projects. They can also see restricted endpoints and restricted variables, but cannot see the detailed information about them.

If you have the Executor role

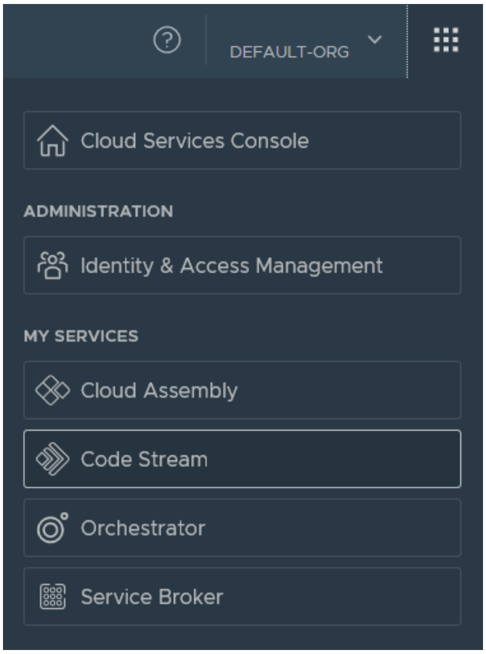
You can run pipelines and take action on user operation tasks. You can also resume, pause, and cancel pipeline executions. But, you cannot modify pipelines.

How do I assign and update roles

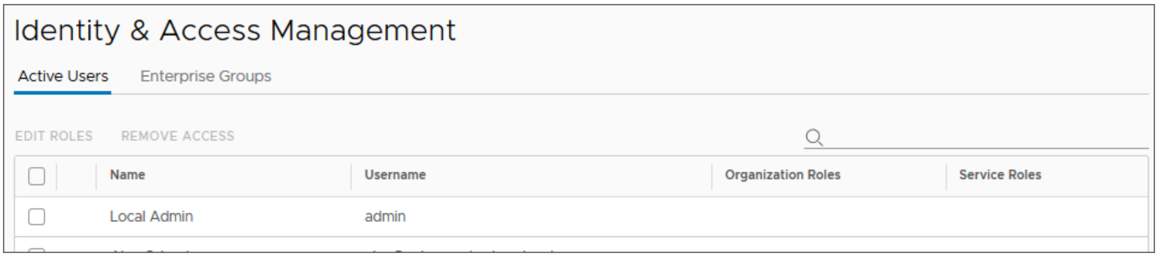
To assign and update roles for other users, you must be an administrator.

1 To see the active users and their roles, in vRealize Automation, click the nine dots at the upper right.

2 Click **Identity & Access Management**.



3 To display user names and roles, click **Active Users**.



□	Name	Username	Organization Roles	Service Roles
□	Local Admin	admin		

4 To add roles for a user, or change their roles, click the check box next to the user name, and click **Edit Roles**.

5 When you add or change user roles, you can also add access to services.

6 To save your changes, click **Save**.

What are user operations and approvals in Code Stream

The User Operations area displays pipeline runs that need approval. The required approver can either approve or reject the pipeline run.

When you create a pipeline, you might need to add an approval to a pipeline if:

- A team member needs to review your code.
- Another user needs to confirm a build artifact.
- You must ensure that all testing is complete.
- A task uses a resource that an administrator marked as restricted, and the task needs approval.
- The pipeline will release software to production.

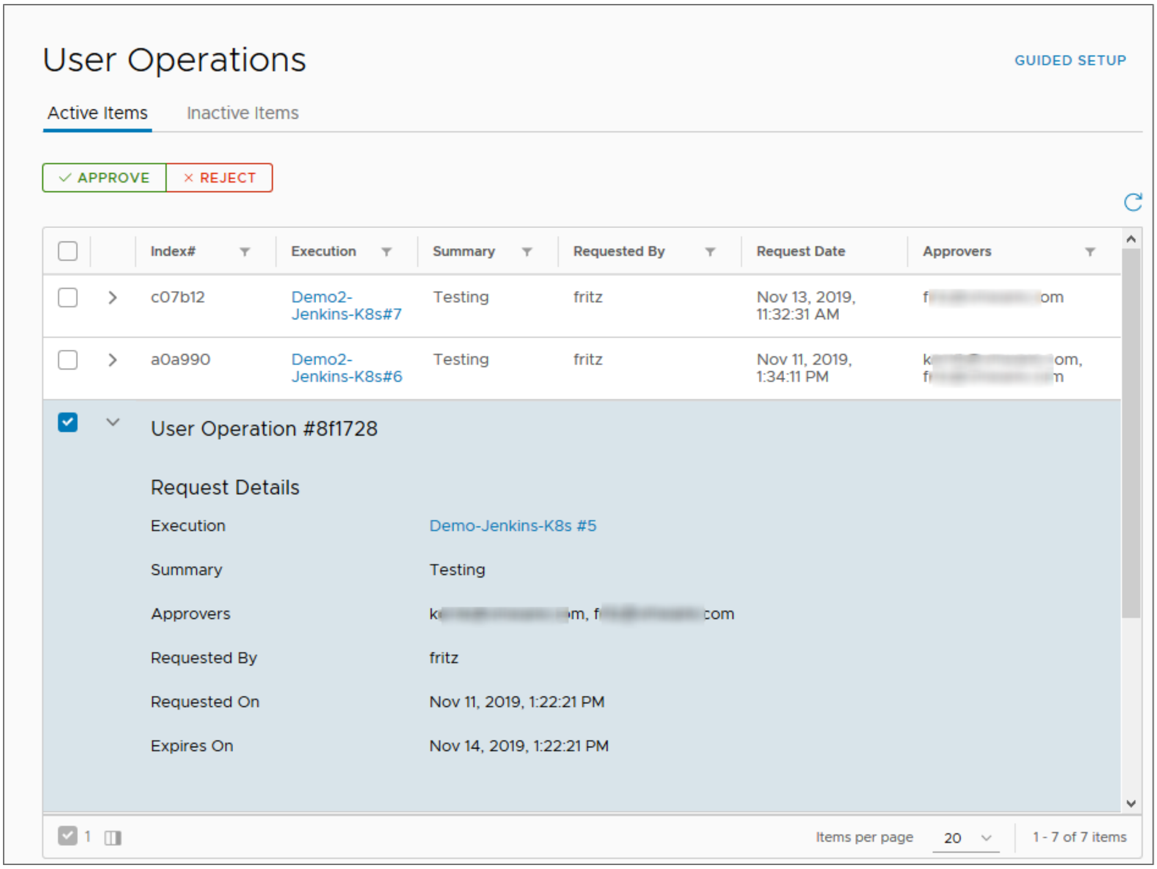
To determine whether to approve a pipeline task, the required approver must have permission and expertise.

When you add a User Operation task, you can set the expiration timeout in days, hours, or minutes. For example, you might need the required user to approve the pipeline in 30 minutes. If they don't approve it in 30 minutes, the pipeline fails as expected.

If you enable sending Email notifications, the User Operation task only sends notifications to approvers who have full email addresses, and not to approver names that are not in an email format.

After the required user approves the task:

- The pending pipeline execution can continue.
- When the pipeline continues, any previous pending requests for approval of that same user operation task are canceled.



User Operations

☐		Index#	Execution	Summary	Requested By	Request Date	Approvers
☐	>	c07b12	Demo2-Jenkins-K8s#7	Testing	fritz	Nov 13, 2019, 11:32:31 AM	f[redacted]om
☐	>	a0a990	Demo2-Jenkins-K8s#6	Testing	fritz	Nov 11, 2019, 1:34:11 PM	k[redacted]om, f[redacted]m
☑	⌄	User Operation #8f1728					

Request Details

Execution	Demo-Jenkins-K8s #5
Summary	Testing
Approvers	k[redacted]m, f[redacted]com
Requested By	fritz
Requested On	Nov 11, 2019, 1:22:21 PM
Expires On	Nov 14, 2019, 1:22:21 PM

In the User Operations area, items to approve or reject appear as active or inactive items. Each item maps to a user operation task in a pipeline.

- **Active Items** wait for the approver who must review the task, and approve or reject it. If you are a user who is on the approver list, you can expand the user operation row, and click **Accept** or **Reject**.
- **Inactive Items** were approved or rejected. If a user rejected the user operation, or if the approval on the task timed out, it can no longer be approved.

The Index# is a unique six-character alphanumeric string that you can use as a filter to search for a particular approval.

Pipeline approvals also appear in the **Executions** area.

- Pipelines that are waiting for approval indicate their status as waiting.
- Other states include queued, completed, and failed.
- If your pipeline is in a wait state, the required approver must approve your pipeline task.

Creating and using pipelines in Code Stream

3

You can use vRealize Automation Code Stream to model your build, test, and deploy process. With vRealize Automation Code Stream, you set up the infrastructure that supports your release cycle and create pipelines that model your software release activities. vRealize Automation Code Stream delivers your software from development code, through testing, and deploys it to your production instances.

Each pipeline includes stages and tasks. Stages represent your development phases, and tasks perform the required actions that deliver your software application through the stages.

What are Pipelines in vRealize Automation Code Stream

A pipeline is a continuous integration and continuous delivery model of your software release process. It releases your software from source code, through testing, to production. It includes a sequence of stages that include tasks that represent the activities in your software release cycle. Your software application flows from one stage to the next through the pipeline.

You add endpoints so that the tasks in your pipeline can connect to data sources, repositories, or notification systems.

Creating Pipelines

You can create a pipeline by starting with a blank canvas, using a smart pipeline template, or by importing YAML code.

- Use the blank canvas. For an example, see Planning a CICD native build in Code Stream before manually adding tasks.
- Use a smart pipeline template. For an example, see Chapter 4 Planning to natively build, integrate, and deliver your code in Code Stream .
- Import YAML code. Click **Pipelines > Import**. In the **Import** dialog box, select the YAML file or enter the YAML code, and click **Import**.

When you use the blank canvas to create a pipeline, you add stages, tasks, and approvals. The pipeline automates the process that builds, tests, deploys, and releases your application. The tasks in each stage run actions that build, test, and release your code through each stage.

Table 3-1. Example pipeline stages and uses

Example stage	Examples of what you can do
Development	In a development stage, you can provision a machine, retrieve an artifact, add a build task that creates a Docker host for continuous integration of your code, and more. For example: ■ To plan and create a continuous integration (CI) build, which delivers your code by using the native build capability in vRealize Automation Code Stream, see Planning a continuous integration native build in Code Stream before using the smart pipeline template.
Test	In a test stage, you can add a Jenkins task to test your software application, and include post-processing test tools such as JUnit and JaCoCo, and more. For example: ■ Integrate vRealize Automation Code Stream with Jenkins, and run a Jenkins job in your pipeline, which builds and tests your source code. See How do I integrate Code Stream with Jenkins. ■ Create custom scripts that extend the capability of vRealize Automation Code Stream to integrate with your own build, test, and deploy tools. See How do I integrate my own build, test, and deploy tools with Code Stream. ■ Track trends on post-processing for a continuous integration (CI) pipeline. See How do I use custom dashboards to track key performance indicators for my pipeline in Code Stream.
Production	In a production stage, you can integrate a cloud template in Cloud Assembly that provisions your infrastructure, deploys your software to a Kubernetes cluster, and more. For example: ■ To see example stages for development and production, which can deploy your software application in your own Blue-Green deployment model, see How do I deploy my application in Code Stream to my Blue-Green deployment. ■ To integrate a cloud template into your pipeline, see How do I automate the release of an application that I deploy from a YAML cloud template in Code Stream. You can also add a deployment task that runs a script to deploy the application. ■ To automate the deployment of your software applications to a Kubernetes cluster, How do I automate the release of an application in Code Stream to a Kubernetes cluster. ■ To integrate code into your pipeline and deploy your build image, see How do I continuously integrate code from my GitHub or GitLab repository into my pipeline in Code Stream.

You can export your pipeline as a YAML file. Click **Pipelines**, click a pipeline card, then click **Actions > Export**.

Approving pipelines

You can obtain an approval from another team member at specific points in your pipeline.

- To require approval on a pipeline by including a user operation task in a pipeline, see How do I run a pipeline and see results. This task sends an email notification to the user who must review it. The reviewer must either approve or reject the approval before the pipeline can continue to run. If the User Operation task has an expiration timeout set in days, hours, or minutes, the required user must approve the pipeline before the task expires. Otherwise, the pipeline fails as expected.

- In any stage of a pipeline, if a task or stage fails, you can have vRealize Automation Code Stream create a Jira ticket. See How do I create a Jira ticket in Code Stream when a pipeline task fails.

Triggering pipelines

Pipelines can trigger when developers check their code into the repository, or review code, or when it identifies a new or updated build artifact.

- To integrate vRealize Automation Code Stream with the Git lifecycle, and trigger a pipeline when developers update their code, use the Git trigger. See How do I use the Git trigger in Code Stream to run a pipeline.
- To integrate vRealize Automation Code Stream with the Gerrit code review lifecycle, and trigger a pipeline on code reviews, use the Gerrit trigger. See How do I use the Gerrit trigger in Code Stream to run a pipeline.
- To trigger a pipeline when a Docker build artifact is created or updated, use the Docker trigger. See How do I use the Docker trigger in Code Stream to run a continuous delivery pipeline.

For more information about the triggers that vRealize Automation Code Stream supports, see Chapter 7 Triggering pipelines in Code Stream.

This chapter includes the following topics:

How do I run a pipeline and see results

You can run a pipeline from the pipeline card, in pipeline edit mode, and from the pipeline execution. You can also use the available triggers to have Code Stream run a pipeline when certain events occur.

When all the stages and tasks in your pipeline are valid, the pipeline is ready to be released, run, or triggered.

To run or trigger your pipeline using Code Stream, you can enable and run the pipeline either from the pipeline card, or while you are in the pipeline. Then, you can view the pipeline execution to confirm that the pipeline built, tested, and deployed your code.

When a pipeline execution is in progress, you can delete the execution if you are an administrator or a non-admin user.

- Administrator: To delete a pipeline execution when it is running, click **Executions**. On the execution to delete, click **Actions > Delete**.
- Non-admin user: To delete a running pipeline execution, click **Executions**, and click **Alt Shift d**.

When a pipeline execution is in progress and appears to be stuck, an administrator can refresh the execution from the Executions page or the Execution details page.

- Executions page: Click **Executions**. On the execution to refresh, click **Actions > Sync**.
- Execution details page: Click **Executions**, click the link to the execution details, and click **Actions > Sync**.

To run a pipeline when specific events occur, use the triggers.

- Git trigger can run a pipeline when developers update code.
- Gerrit trigger can run a pipeline when code reviews occur.
- Docker trigger can run a pipeline when an artifact is created in a Docker registry.
- The `curl` command or `wget` command can have Jenkins run a pipeline after a Jenkins build finishes.

For more information about using the triggers, see Chapter 7 Triggering pipelines in Code Stream.

The following procedure shows you how to run a pipeline from the pipeline card, view executions, see execution details, and use the actions. It also shows you how to release a pipeline so that you can add it to vRealize Automation Service Broker.

Prerequisites

- Verify that one or more pipelines are created. See the examples in Chapter 5 Tutorials for using Code Stream.

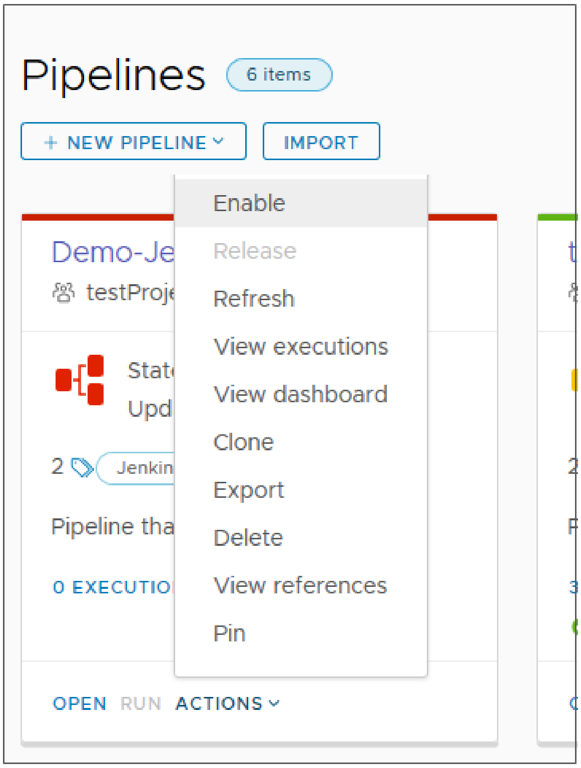
Procedure

1 Enable your pipeline.

To run or release a pipeline, you must enable it first.

a Click **Pipelines**.

b On your pipeline card, click **Actions** > **Enable**.



You can also enable your pipeline while you are in the pipeline. If your pipeline is already enabled, **Run** is active, and the **Actions** menu displays **Disable**.

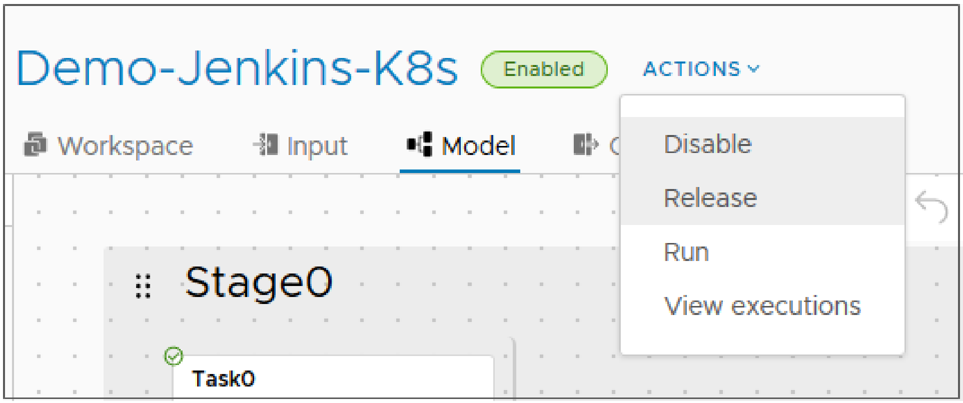
2 (Optional) Release your pipeline.

If you want to make your pipeline available as a catalog item in vRealize Automation Service Broker, you must release it in Code Stream.

a Click **Pipelines**.

b On your pipeline card, click **Actions > Release**.

You can also release your pipeline while you are in the pipeline.



After you release the pipeline, you open Service Broker to add the pipeline as a catalog item and run it. See how to add Code Stream pipelines to the Service Broker catalog in *Using and Managing VMware Service Broker*.

Note If the pipeline requires more that 120 minutes to run, provide an approximate execution time as a request timeout value. To set or review the request timeout for a project, open Service Broker as administrator and select **Infrastructure > Projects**. Click your project name and then click **Provisioning**.

If the request timeout value is not set, an execution that requires more than 120 minutes to run appears as failed with a callback timeout request error. However, the pipeline execution is not affected.

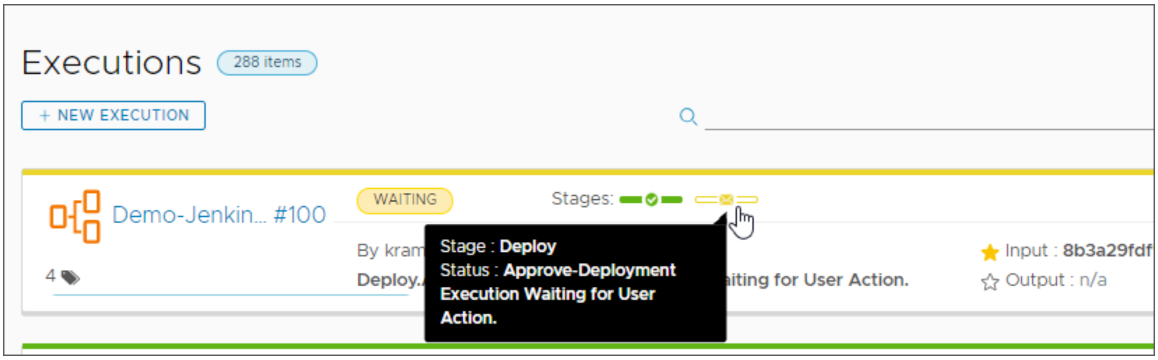
3 On the pipeline card, click **Run**.

4 To view the pipeline as it runs, click **Executions**.

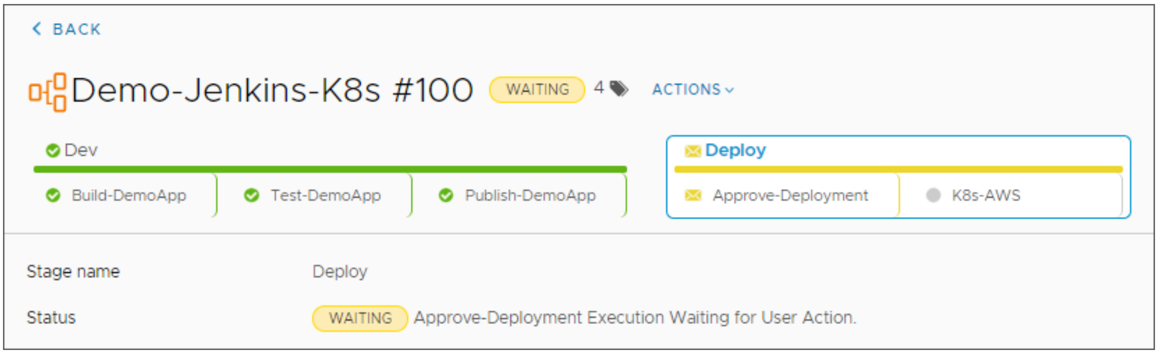
The pipeline runs each stage in sequence, and the pipeline execution displays a status icon for each stage. If the pipeline includes a user operation task, a user must approve the task for the pipeline to continue to run. When a user operation task is used, the pipeline stops running and waits for the required user to approve the task.

For example, you might use the user operation task to approve the deployment of code to a production environment.

If the User Operation task has an expiration timeout set in days, hours, or minutes, the required user must approve the pipeline before the task expires. Otherwise, the pipeline fails as expected.



5 To see the pipeline stage that is waiting for user approval, click the status icon for the stage.



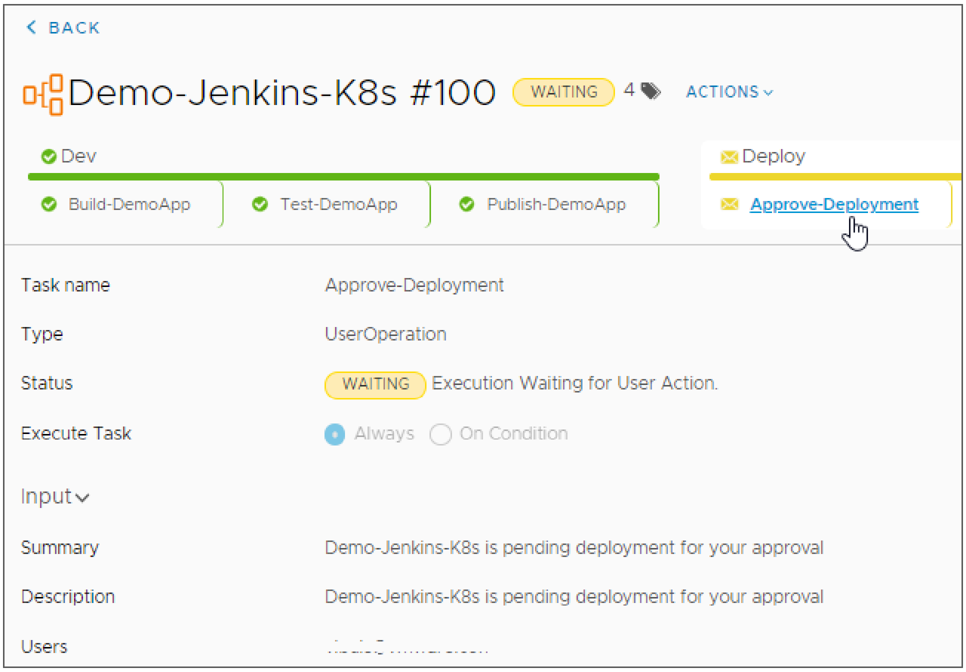
Stage name	Deploy
Status	WAITING Approve-Deployment Execution Waiting for User Action.

6 To see the details for the task, click the task.

After the required user approves the task, a user who has the appropriate role must resume the pipeline. For required roles, see How do I manage user access and approvals in Code Stream.

If an execution fails, you must triage and fix the cause of the failure. Then, go to the execution, and click **Actions > Re-run**.

You can resume primary pipeline executions and nested executions.



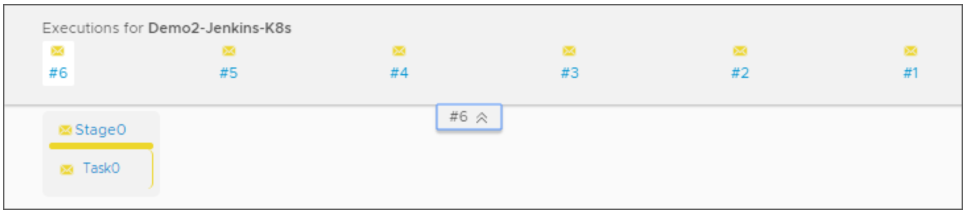
Task name	Approve-Deployment
Type	UserOperation
Status	WAITING Execution Waiting for User Action.
Execute Task	● Always ○ On Condition

Input

Summary	Demo-Jenkins-K8s is pending deployment for your approval
Description	Demo-Jenkins-K8s is pending deployment for your approval
Users	[redacted]

7 From the pipeline execution, you can click **Actions** to view the pipeline, and select an action such as **Pause**, **Cancel**, and more. When a pipeline execution is in progress, if you are an administer you can delete or sync the pipeline execution. If you are a non-admin user, you can delete a running pipeline.

8 To navigate easily between executions and see the details for a task, click **Executions**, and click a pipeline run. Then, click the tab at the top and select the pipeline run.



Results

Congratulations! You ran a pipeline, examined the pipeline execution, and viewed a user operation task that required approval for the pipeline to continue to run. You also used the **Actions** menu in the pipeline execution to return to the pipeline model so that you can make any required changes.

What to do next

To learn more about using Code Stream to automate your software release cycle, see Chapter 5 Tutorials for using Code Stream.

What types of tasks are available in Code Stream

When you configure your pipeline, you add specific types of tasks that the pipeline runs for the actions you need. Each task type integrates with another application and enables your pipeline as it builds, tests, and delivers your applications.

To run your pipeline, whether you must pull artifacts from a repository for deployment, run a remote script, or require approval on a user operation from a team member, Code Stream has the type of task for you!

Code Stream supports canceling a pipeline run on various types of tasks. When you click **Cancel** on a pipeline execution, the task, stage, or entire pipeline enters the canceling state and cancels the pipeline run.

Code Stream allows you to cancel the pipeline run on a task, stage, or the entire pipeline when using these tasks:

- Jenkins
- SSH
- PowerShell
- User Operation
- Pipeline
- Cloud template
- vRO
- POLL

Code Stream does not propagate the cancel behavior to third-party systems for these tasks: CI, Custom Integration, or Kubernetes. Code Stream marks the task as canceled and immediately stops fetching the status without waiting for the task to finish. The task might complete or fail on the third-party system but immediately stops running in Code Stream when you click **Cancel**.

Before you use a task in your pipeline, verify that the corresponding endpoint is available.

Table 3-2. Obtain an approval or set a decision point

Type of task	What it does	Examples and details
User Operation	A User Operation task enables a required approval that controls when a pipeline runs and must stop for an approval.	See How do I run a pipeline and see results. and How do I manage user access and approvals in Code Stream.
Condition	Adds a decision point, which determines whether the pipeline continues to run, or stops, based on condition expressions. When the condition is true, the pipeline runs successive tasks. When false, the pipeline stops.	See How do I use variable bindings in a condition task to run or stop a pipeline in Code Stream.

Table 3-3. Automate continuous integration and deployment

Type of task	What it does	Examples and details
Cloud template	Deploys an automation cloud template from GitHub and provisions an application, and automates the continuous integration and continuous delivery (CICD) of that cloud template for your deployment.	See How do I automate the release of an application that I deploy from a YAML cloud template in Code Stream. The cloud template parameters appear after you first select **Create** or **Update**, then select **Cloud Template** and **Version**. You can add these elements, which accommodate variable bindings, to the input text areas in the cloud template task: ■ Integer ■ Enumeration string ■ Boolean ■ Array variable When you use variable binding in the input, be aware of these exceptions. For enumerations, you must select an enumeration value from a fixed set. For Boolean values, you must enter the value in the input text area. The cloud template parameter appears in the cloud template task when a cloud template in Cloud Assembly includes input variables. For example, if a cloud template has an input type of `Integer`, you can enter the integer directly or as a variable by using variable binding.
CI	The CI task enables continuous integration of your code into your pipeline by pulling a Docker build image from a registry endpoint, and deploying it to a Kubernetes cluster. The CI task displays 100 lines of the log as output, and displays 500 lines when you download the logs. The CI tasks requires ephemeral ports 32768 to 61000.	See Planning a CICD native build in Code Stream before using the smart pipeline template.
Custom	The Custom task integrates Code Stream with your own build, test, and deploy tools.	See How do I integrate my own build, test, and deploy tools with Code Stream.

Table 3-3. Automate continuous integration and deployment (continued)

Type of task	What it does	Examples and details
Kubernetes	Automate the deployment of your software applications to Kubernetes clusters on AWS.	See How do I automate the release of an application in Code Stream to a Kubernetes cluster.
Pipeline	Nests a pipeline in a primary pipeline. When a pipeline is nested, it behaves as a task in the primary pipeline. On the Task tab of the primary pipeline, you can easily navigate to the nested pipeline by clicking the link to it. The nested pipeline opens in a new browser tab.	To find nested pipelines in **Executions**, enter `nested` in the search area.

Table 3-4. Integrate development, test, and deployment applications

Task type...	What it does...	Examples and details...
Bamboo	Interacts with a Bamboo continuous integration (CI) server, which continuously builds, tests, and integrates software in preparation for deployment, and triggers code builds when developers commit changes. It exposes the artifact locations that the Bamboo build produces so that the task can output the parameters for other tasks to use for build and deployment.	Connect to a Bamboo server endpoint and start a Bamboo build plan from your pipeline.
Jenkins	Triggers Jenkins jobs that build and test your source code, runs test cases, and can use custom scripts.	See How do I integrate Code Stream with Jenkins.
TFS	Allows you to connect your pipeline to Team Foundation Server to manage and invoke build projects, including configured jobs that build and test your code.	Code Stream supports Team Foundation Server 2013 and 2015.
vRO	Extends the capability of Code Stream by running predefined or custom workflows in vRealize Orchestrator.	See How do I integrate Code Stream with vRealize Orchestrator.

Table 3-5. Integrate other applications through an API

Task type...	What it does...	Examples and details...
REST	Integrates Code Stream with other applications that use a REST API so that you can continuously develop and deliver software applications that interact with each other.	See How do I use a REST API to integrate Code Stream with other applications.
Poll	Invokes a REST API and polls it until the pipeline task meets the exit criteria and completes. A Code Stream administrator can set the poll count to a maximum of 10000. The poll interval must be greater than or equal to 60 seconds. When you mark the **Continue on failure** check box, if the count or interval exceeds these values, the poll task continues to run. `POLL Iteration Count`: Appears in the pipeline execution and displays the number of times the POLL task requested a response from the URL. For example, if the POLL input is 65 and the actual times the POLL request ran is 4, the iteration count in the pipeline execution output would display 4 (out of 65).	See How do I use a REST API to integrate Code Stream with other applications.

Table 3-6. Run remote and user-defined scripts

Type of task	What it does	Examples and details
PowerShell	With the PowerShell task, Code Streamcan run script commands on a remote host. For example, a script can automate test tasks, and run administrative types of commands. The script can be remote or user-defined. It can connect over HTTP or HTTPS, and can use TLS. The Windows host must have the `winrm` service configured, and `winrm` must have `MaxShellsPerUser` and `MaxMemoryPerShellMB` configured. To run a PowerShell task, you must have an active session to the remote Windows host. **PowerShell Command Line Length** If you enter a base64 PowerShell command, be aware that you must calculate the overall command length. The Code Stream pipeline encodes and wraps a base64 PowerShell command in another command, which increases the overall length of the command. The maximum length allowed for a PowerShell `winrm` command is 8192 bytes. The command length limit is lower for the PowerShell task when it is encoded and wrapped. As a result, you must calculate the command length before you enter the PowerShell command. The command length limit for the Code Stream PowerShell task depends on the base64 encoded length of the original command. The command length is calculated as follows. `3 * (length of original command / 4)) - (numberOfPaddingCharacters) + 77 (Length of Write-output command)` The command length for Code Stream must be less than the maximum limit of 8192.	When you configure `MaxShellsPerUser` and `MaxMemoryPerShellMB`: - The acceptable value for `MaxShellsPerUser` is `500` for 50 concurrent pipelines, with 5 PowerShell tasks for each pipeline. To set the value, run: `winrm set winrm/config/winrs '@{MaxShellsPerUser="500"}'` - The acceptable memory value for `MaxMemoryPerShellMB` is `2048`. To set the value, run: `winrm set winrm/config/winrs '@{MaxMemoryPerShellMB="2048"}'` The script writes the output to a response file that another pipeline can consume.
SSH	The SSH task allows the Bash shell script task to run script commands on a remote host. For example, a script can automate test tasks, and run administrative types of commands. The script can be remote or user-defined. It can connect over HTTP or HTTPS, and requires a private key or password. The SSH service must be configured on the Linux host, and the SSHD configuration of `MaxSessions` must be set to `50`.	The script can be remote or user-defined. For example, a script might resemble: `message="Hello World" echo $message` The script writes the output to a response file that another pipeline can consume.

Table 3-6. Run remote and user-defined scripts (continued)

Type of task	What it does	Examples and details
	If you run many SSH tasks concurrently, increase the `MaxSessions` and `MaxOpenSessions` on the SSH host. Do not use your vRealize Automation instance as the SSH host if you need to modify the MaxSessions and MaxOpenSessions configuration settings.	

How do I use variable bindings in Code Stream pipelines

Binding a pipeline task means that you create a dependency for the task when the pipeline runs. You can create a binding for a pipeline task in several ways. You can bind a task to another task, bind it to a variable and expression, or bind it to a condition.

How to apply dollar bindings to cloud template variables in a cloud template task

You can apply dollar bindings to cloud template variables in a Code Stream pipeline cloud template task. The way you modify the variables in Code Stream depends on the coding of the variable properties in the cloud template.

If you must use dollar bindings in a cloud template task, but the current version of the cloud template that you're using in the cloud template task doesn't allow it, modify the cloud template in Cloud Assembly and deploy a new version. Then, use the new cloud template version in your cloud template task, and add the dollar bindings where needed.

To apply dollar bindings on the types of properties that the Cloud Assembly cloud template provides, you must have the correct permissions.

- You must have the same role as the person who created the cloud template deployment in Cloud Assembly.
- The person who models the pipeline and the person who runs the pipeline might be two different users and might have different roles.
- If a developer has the Code Stream Executor role and models the pipeline, the developer must also have the same Cloud Assembly role of the person who deployed the cloud template. For example, the required role might be Cloud Assembly administrator.
- Only the person who models the pipeline can create the pipeline and create the deployment because they have permission.

To use an API token in the cloud template task:

- The person who models the pipeline can give an API token to another user who has the Code Stream Executor role. Then, when the Executor runs the pipeline, it uses the API token and the credentials that the API token creates.

- When a user enters the API token in the cloud template task, it creates the credentials that the pipeline requires.
- To encrypt the API token value, click **Create Variable**.
- If you don't create a variable for the API token, and use it in the cloud template task, the API token value appears in plain text.

To apply dollar bindings to cloud template variables in a cloud template task, follow these steps.

You start with a cloud template that has input variable properties defined, such as `integerVar`, `stringVar`, `flavorVar`, `BooleanVar`, `objectVar`, and `arrayVar`. You can find the image properties defined in the `resources` section. The properties in the cloud template code might resemble:

```
formatVersion: 1
inputs:
  integerVar:
    type: integer
    encrypted: false
    default: 1
  stringVar:
    type: string
    encrypted: false
    default: bkix
  flavorVar:
    type: string
    encrypted: false
    default: medium
  BooleanVar:
    type: boolean
    encrypted: false
    default: true
  objectVar:
    type: object
    encrypted: false
    default:
      bkix2: bkix2
  arrayVar:
    type: array
    encrypted: false
    default:
      - '1'
      - '2'
resources:
  Cloud_Machine_1:
    type: Cloud.Machine
    properties:
      image: ubuntu
      flavor: micro
      count: '${input.integerVar}'
```

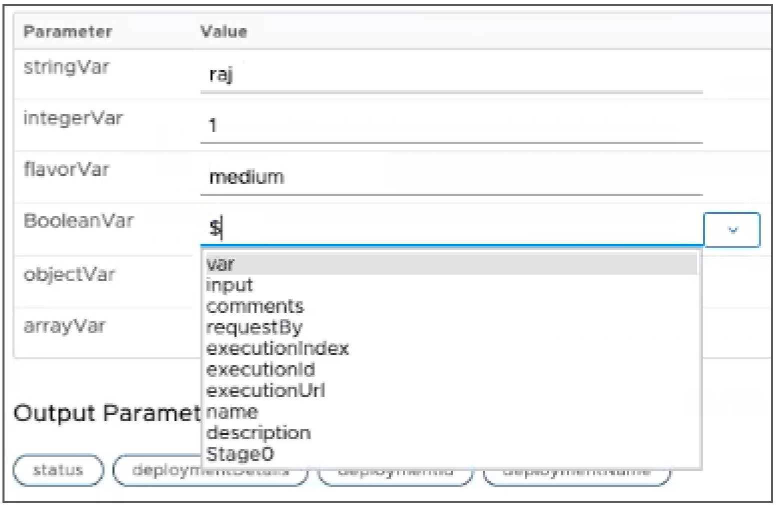
You can use dollar sign variables ($) for `image` and `flavor`. For example:

```
resources:
  Cloud_Machine_1:
    type: Cloud.Machine
    properties:
      input: '${input.image}'
      flavor: '${input.flavor}'
```

To use a cloud template in a Code Stream pipeline, and add dollar bindings to it, follow these steps.

1. In Code Stream, click **Pipelines > Blank Canvas**.
2. Add a **Cloud template** task to the pipeline.
3. In the Cloud template task, for **Cloud template source** select **Cloud Assembly Cloud Templates**, enter the cloud template name, and select the cloud template version.
4. Notice that you can enter an API token, which provides credentials for the pipeline. To create a variable that encrypts the API token in the cloud template task, click **Create Variable**.
5. In the **Parameter and Value** table that appears, notice the parameter values. The default value for `flavor` is `small` and the default value for `image` is `ubuntu`.
6. Let's say that you must change the cloud template in Cloud Assembly. For example, you:
   a. Set the `flavor` so that it uses a property of type `array`. Cloud Assembly allows comma-separated values for `Flavor` when the type is **`array`**.
   b. Click **Deploy**.
   c. On the Deployment Type page, enter a deployment name, and select the version of the cloud template.
   d. On the Deployment Inputs page, you can define one or more values for `Flavor`.
   e. Notice that the Deployment inputs include all the variables defined in your cloud template code, and appear as defined in the cloud template code. For example: `Integer Var`, `String Var`, `Flavor Var`, `Boolean Var`, `Object Var`, and `Array Var`. `String Var` and `Flavor Var` are string values, and `Boolean Var` is a check box.
   f. Click **Deploy**.
7. In Code Stream, select the new version of the cloud template, and enter values in the **Parameter and Value** table. Cloud templates support the following types of parameters, which enable Code Stream bindings by using dollar sign variables. Slight differences exist between the user interface of the Code Stream cloud template task and the user interface of the Cloud Assembly cloud template. Depending on the coding of a cloud template in Cloud Assembly, entering values in the cloud template task in Code Stream might not be allowed.
   a. For **flavorVar**, if the cloud template defined the type as string or array, enter a string or a comma-separated value array. An example array resembles **`test, test`**.

b For **BooleanVar**, in the drop-down menu select **true** or **false**. Or, to use a variable binding, enter **$** and select a variable binding from the list.



Parameter	Value
stringVar	raj
integerVar	1
flavorVar	medium
BooleanVar	$
objectVar	
arrayVar	

c For **objectVar**, enter the value with curly brackets and quotation marks in this format: `{"bkix":"bkix":}`.

d The **objectVar** will be passed to the cloud template, and can be used in various ways depending on the cloud template. It allows a string format for a JSON object, and you can add key-value pairs as comma-separated values in the key-value table. You can enter plain text for a JSON object, or a key-value pair as a normal stringified format for JSON.

e For **arrayVar**, enter the comma-separated input value as an array in this format: `["1","2"]`.

8 In the pipeline, you can bind an input parameter to an array.

a Click the **Input** tab.

b Enter a name for the input. For example, `arrayInput`.

c In the **Parameter and Value** table, click in **arrayVar** and enter `${input.arrayInput}`.

d After you save the pipeline and enable it, when the pipeline runs, you must provide an array input value. For example, enter `["1","2"]` and click **Run**.

Now you have learned how to use dollar sign ($) variable bindings in a cloud template in a Code Stream pipeline cloud template task.

How to pass a parameter to a pipeline when it runs

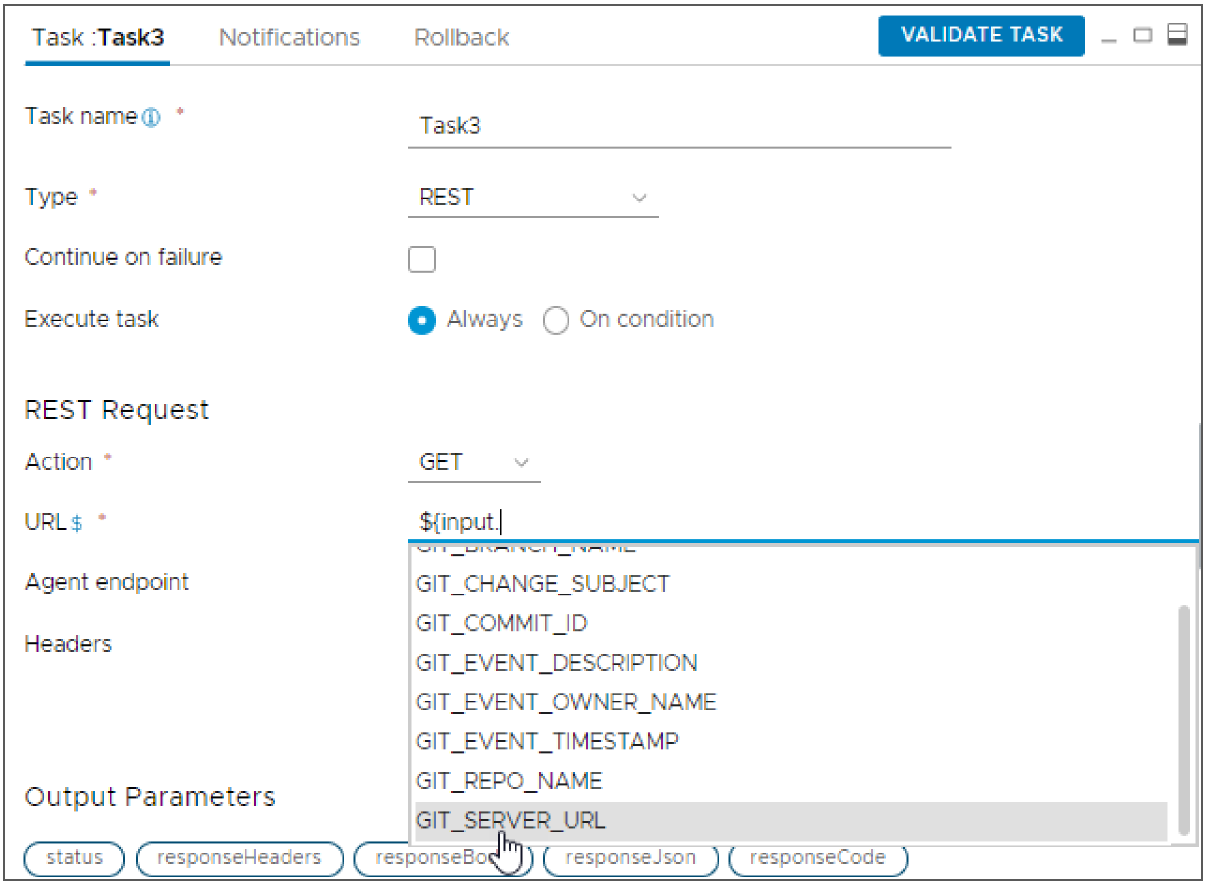
You can add input parameters to your pipeline to have Code Stream pass them to the pipeline. Then, when the pipeline runs, a user must enter the value for the input parameter. When you add output parameters to your pipeline, the pipeline tasks can use the output value from a task. Code Stream supports using parameters in many ways that support your own pipeline needs.

For example, to prompt a user for the URL to their Git server when a pipeline with a REST task runs, you can bind the REST task to a Git server URL.

To create the variable binding, you add a URL binding variable to the REST task. When the pipeline runs and reaches the REST task, a user must enter their URL to the Git server. Here's how you would create the binding:

1. In your pipeline, click the **Input** tab.
2. To set the parameter, for **Auto inject parameters** click **Git**.

   The list of Git parameters appears, and includes **GIT_SERVER_URL**. If you must use a default value for the Git server URL, edit this parameter.
3. Click **Model**, and click your REST task.
4. On the **Task** tab, in the **URL** area, enter **$**, then select **input** and **GIT_SERVER_URL**.



Task name * Task3

Type * REST

Continue on failure □

Execute task ● Always ○ On condition

REST Request

Action * GET

URL $ * ${input.

Agent endpoint

Headers

Output Parameters

   The entry resembles: **${input.GIT_SERVER_URL}**
5. To verify the integrity of the variable binding for the task, click **Validate Task**.

   Code Stream indicates that the task validated successfully.
6. When the pipeline runs the REST task, a user must enter the URL of the Git server. Otherwise, the task does not finish running.

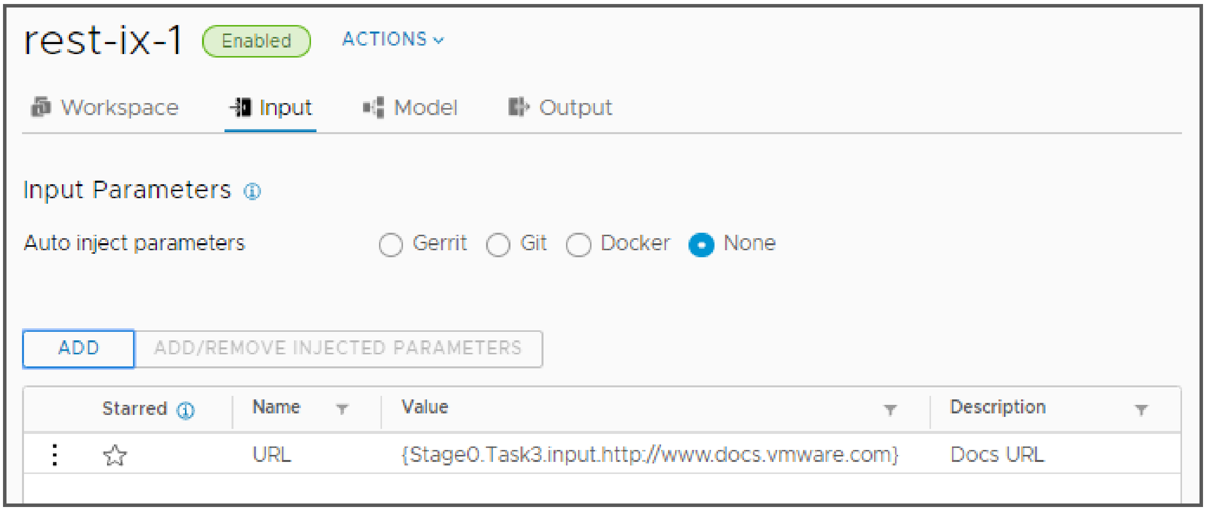
How to bind two pipeline tasks by creating input and output parameters

When you bind tasks together, you add a binding variable to the input configuration of the receiving task. Then, when the pipeline runs, a user replaces the binding variable with the required input.

To bind pipeline tasks together, you use the dollar sign variable ($) in the input parameters and output parameters. This example shows you how.

Let's say you need your pipeline to call a URL in a REST task, and output a response. To call the URL and output the response, you include both input and output parameters in your REST task. You also need a user who can approve the task, and include a User Operations task for another user who can approve it when the pipeline runs. This example shows you how to use expressions in the input and output parameters, and have the pipeline wait for approval on the task.

1 In your pipeline, click the **Input** tab.



Auto inject parameters ○ Gerrit ○ Git ○ Docker ● None

	Starred	Name	Value	Description
⋮	☆	URL	{Stage0.Task3.input.http://www.docs.vmware.com}	Docs URL

2 Leave the **Auto inject parameters** as **None**.

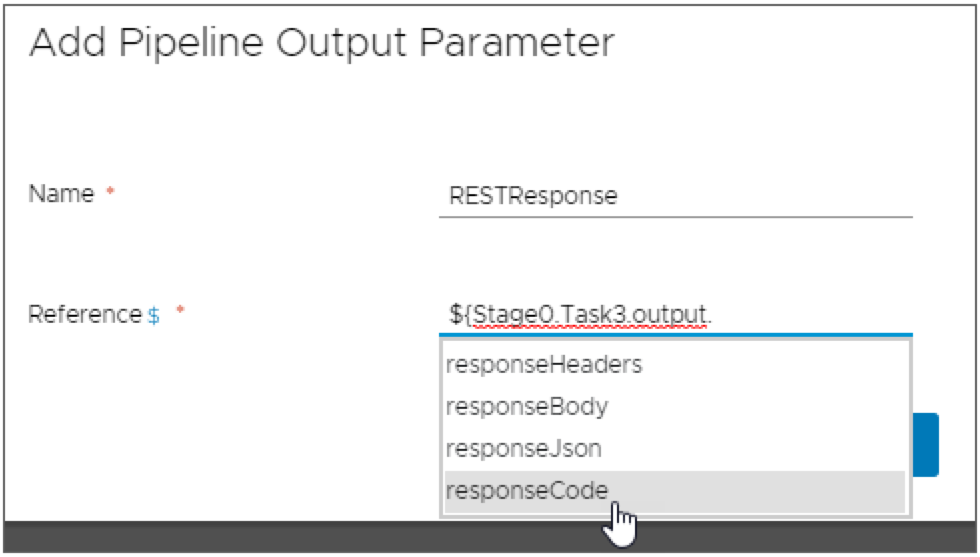
3 Click **Add**, and enter the parameter name, value, and description, and click **OK**. For example:

a Enter a URL name.

b Enter the value: `{Stage0.Task3.input.http://www.docs.vmware.com}`

c Enter a description.

4 Click the **Output** tab, click **Add**, and enter the output parameter name and mapping.



Add Pipeline Output Parameter

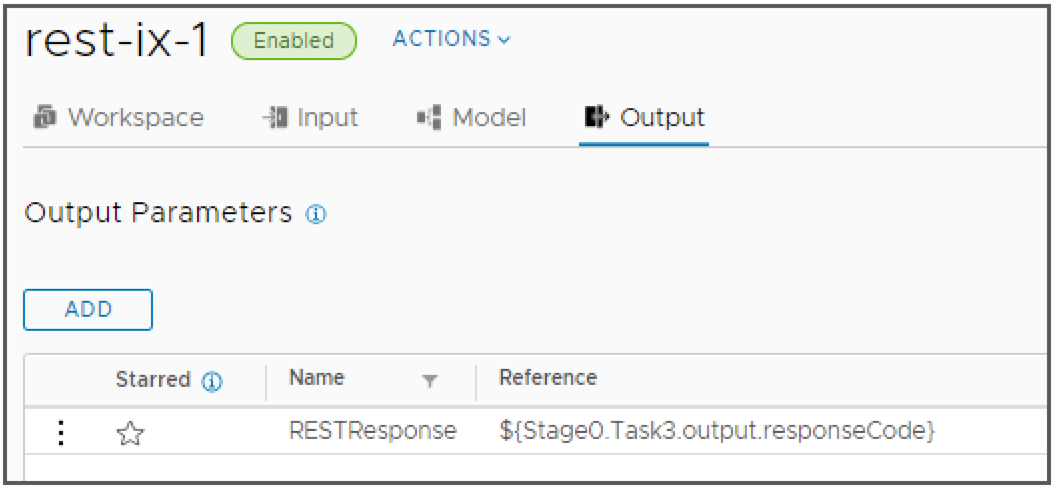
Name * RESTResponse

Reference $ * ${Stage0.Task3.output.

a Enter a unique output parameter name.

b Click in the **Reference** area, and enter `$`.

c Enter the task output mapping by selecting the options as they pop up. Select the **Stage0**, select **Task3**, select **output**, and select **responseCode**. Then, click **OK**.



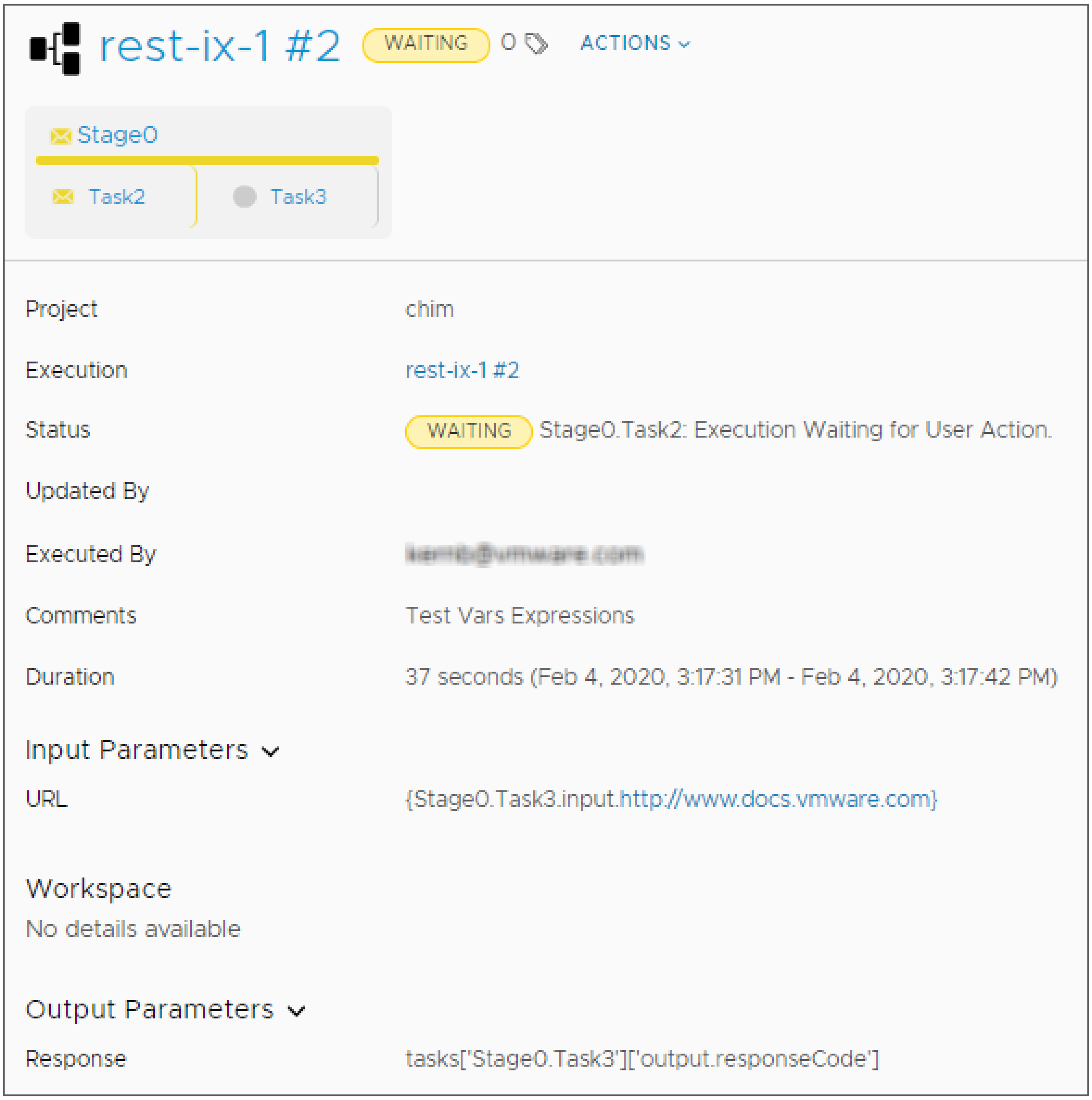
Starred	Name	Reference
☆	RESTResponse	${Stage0.Task3.output.responseCode}

5 Save your pipeline.

6 From the **Actions** menu, click **Run**.

7 Click **Actions > View executions**.

8 Click the pipeline execution, and examine the input parameters and output parameters that you defined.



Project	chim
Execution	rest-ix-1 #2
Status	WAITING Stage0.Task2: Execution Waiting for User Action.
Updated By	
Executed By	[redacted]
Comments	Test Vars Expressions
Duration	37 seconds (Feb 4, 2020, 3:17:31 PM - Feb 4, 2020, 3:17:42 PM)

Input Parameters

URL	{Stage0.Task3.input.http://www.docs.vmware.com}

Workspace

No details available

Output Parameters

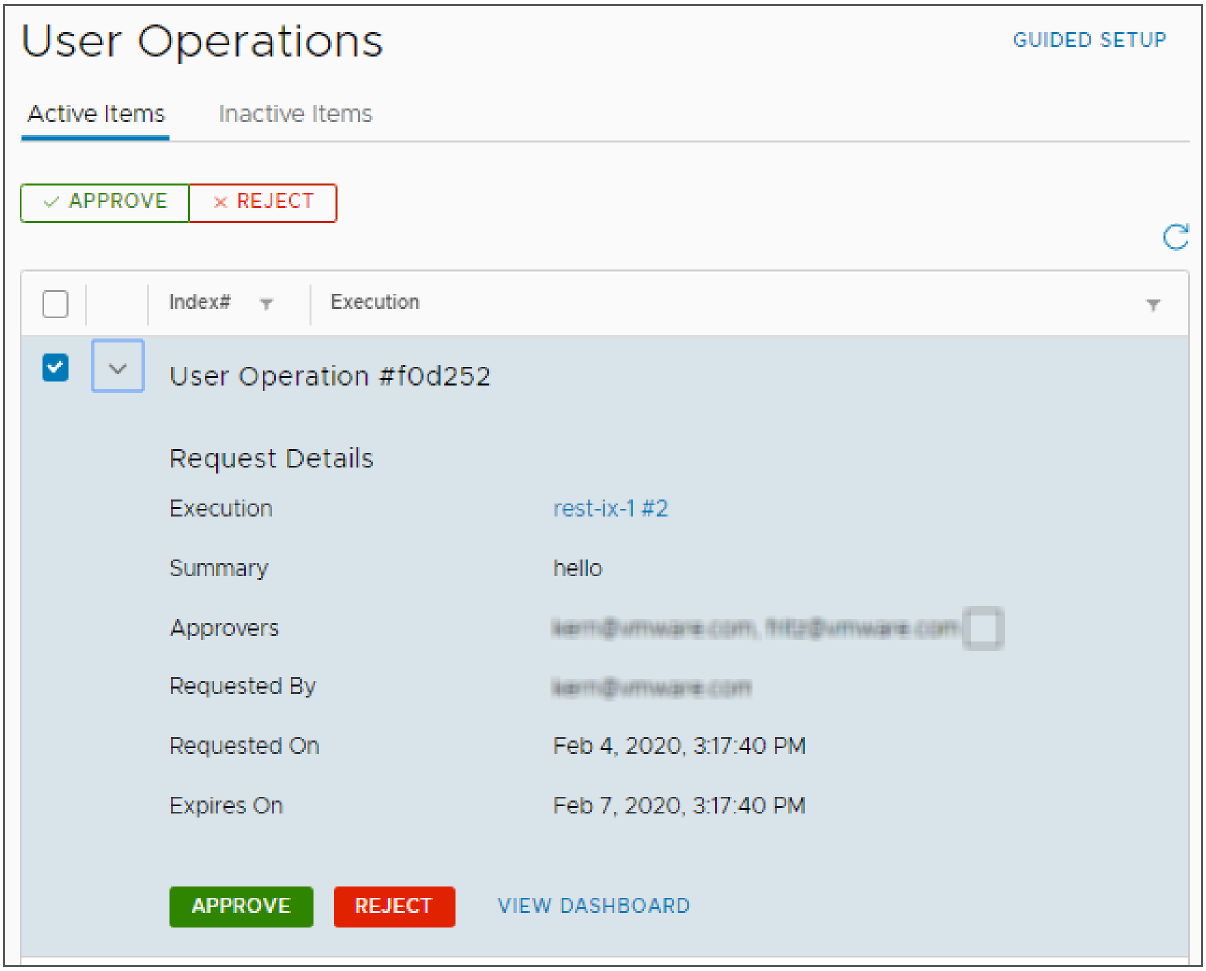
Response	tasks['Stage0.Task3']['output.responseCode']

9 To approve the pipeline, click **User Operations**, and view the list of approvals on the **Active Items** tab. Or, stay in the Executions, click the task, and click **Approve**.

10 To enable the **Approve** and **Reject** buttons, click the check box next to the execution.

11 To see the details, expand the drop-down arrow.

12 To approve the task, click **APPROVE**, enter a reason, and click **OK**.



☑ User Operation #f0d252

Request Details

Execution rest-ix-1 #2

Summary hello

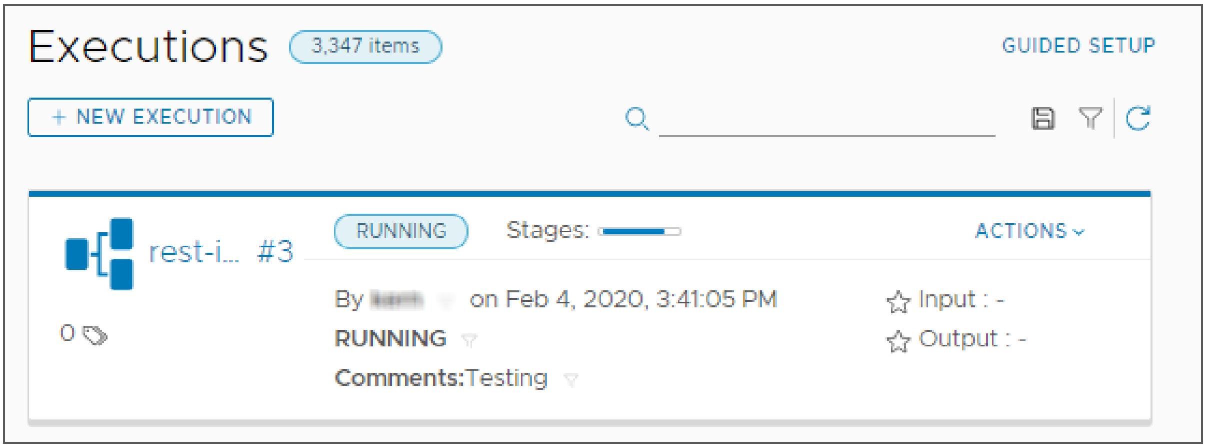
Approvers [redacted]

Requested By [redacted]

Requested On Feb 4, 2020, 3:17:40 PM

Expires On Feb 7, 2020, 3:17:40 PM

13 Click **Executions** and watch the pipeline continue.



By [redacted] on Feb 4, 2020, 3:41:05 PM

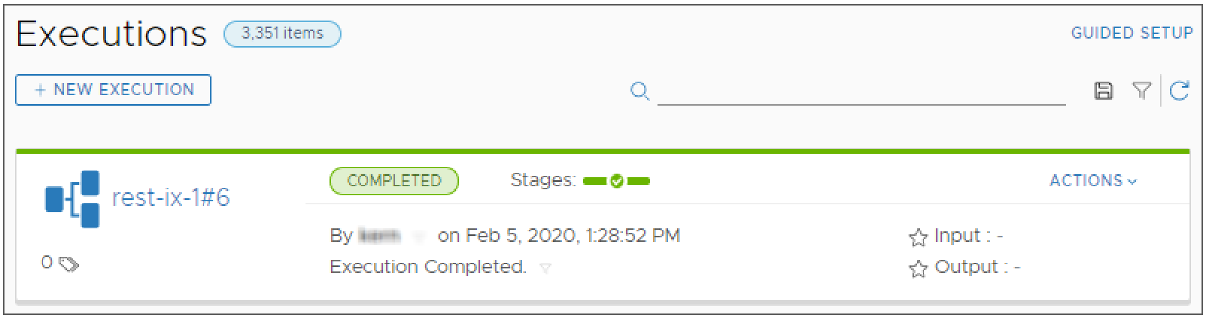
RUNNING

Comments: Testing

Input : -

Output : -

14 If the pipeline fails, correct any errors, then save the pipeline and run it again.



rest-ix-1#6

COMPLETED

Stages:

By [redacted] on Feb 5, 2020, 1:28:52 PM

Execution Completed.

Input : -

Output : -

How do I learn more about variables and expressions

To see details about using variables and expressions when you bind pipeline tasks, see What variables and expressions can I use when binding pipeline tasks in Code Stream.

To learn how to use the pipeline task output with a condition variable binding, see How do I use variable bindings in a condition task to run or stop a pipeline in Code Stream.

How do I use variable bindings in a condition task to run or stop a pipeline in Code Stream

You can have the output of a task in your pipeline determine whether the pipeline runs or stops based on a condition that you supply. To pass or fail the pipeline based on the task output, you use the Condition task.

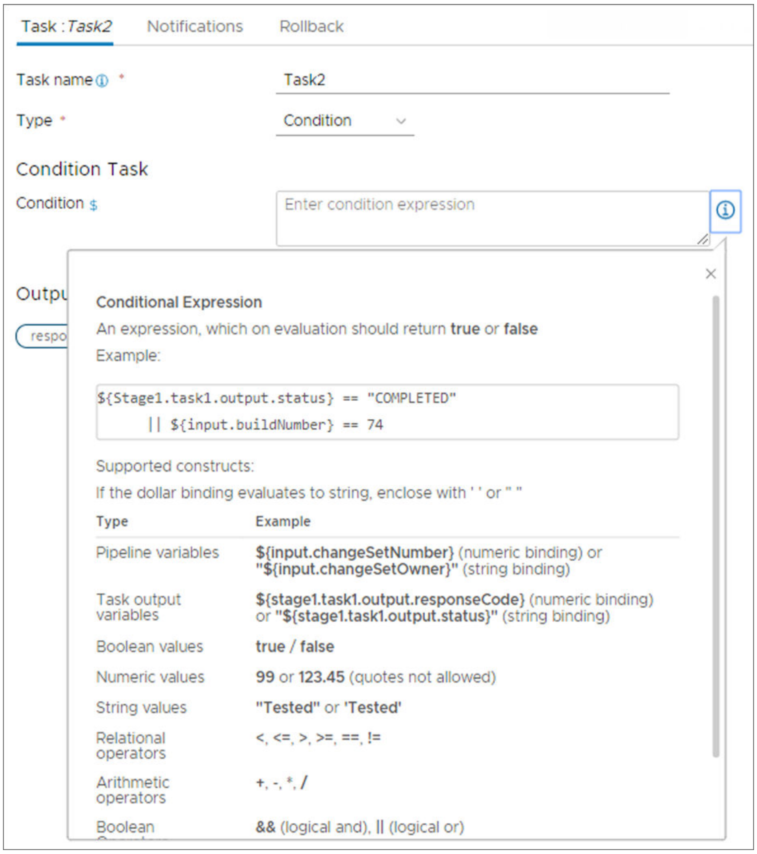
You use the **Condition** task as a decision point in your pipeline. By using the Condition task with a condition expression that you provide, you can evaluate any properties in your pipeline, stages, and tasks.

The result of the Condition task determines whether the next task in the pipeline runs.

- A true condition allows the pipeline run continue.
- A false condition stops the pipeline.

For examples of how to use the output value of one task as the input to the next task by binding the tasks together with a Condition task, see How do I use variable bindings in Code Stream pipelines.

Table 3-7. How the Condition task and its condition expression relate to the pipeline

Condition task	What it affects	What it does
Condition task	Pipeline	The **Condition** task determines whether the pipeline runs or stops at that point, based on whether the task output is true or false.
Condition expression	Condition task output	When the pipeline runs, the condition expression that you include in the **Condition** task produces a true or false output status. For example, a condition expression can require the Condition task output status as `Completed`, or use a build number of `74`. The condition expression appears on the Task tab in the Condition task. 

Task name * Task2

Type * Condition

Condition Task

Condition $ Enter condition expression

Conditional Expression

An expression, which on evaluation should return **true** or **false**

Example:

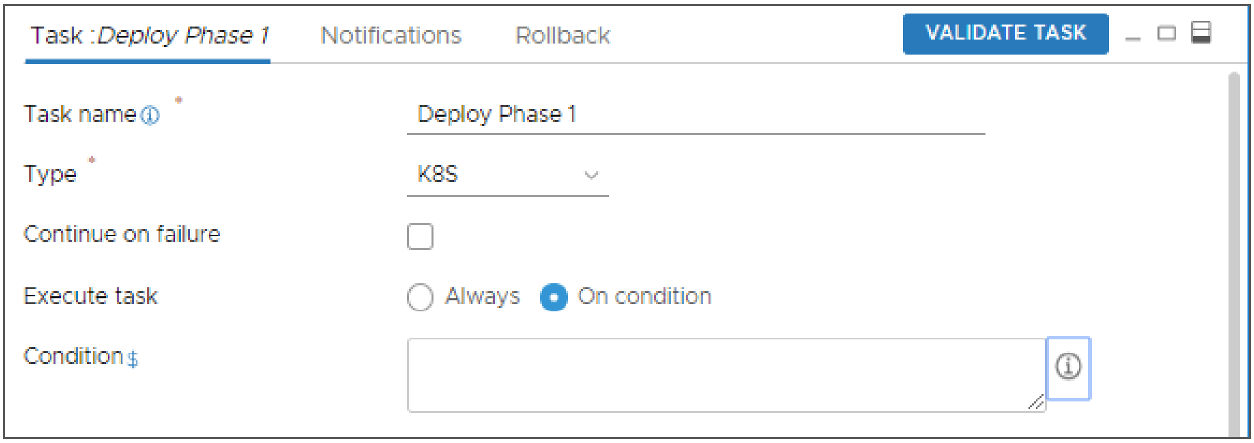
```
${Stage1.task1.output.status} == "COMPLETED"
      || ${input.buildNumber} == 74
```

Supported constructs:

If the dollar binding evaluates to string, enclose with ' ' or " "

Type	Example
Pipeline variables	**${input.changeSetNumber}** (numeric binding) or **"${input.changeSetOwner}"** (string binding)
Task output variables	**${stage1.task1.output.responseCode}** (numeric binding) or **"${stage1.task1.output.status}"** (string binding)
Boolean values	**true / false**
Numeric values	**99** or **123.45** (quotes not allowed)
String values	**"Tested"** or **'Tested'**
Relational operators	<, <=, >, >=, ==, !=
Arithmetic operators	+, -, *, /
Boolean	&& (logical and), || (logical or)

The **Condition** task differs in function and behavior from the **On Condition** setting in other types of tasks.



Task name ⓘ * Deploy Phase 1

Type * K8S

Continue on failure □

Execute task ○ Always ● On condition

Condition $

In other types of tasks, the **On Condition** determines whether the current task runs, rather than successive tasks, based on the evaluation of its precondition expression of true or false. The condition expression for the **On Condition** setting produces a true or false output status for the current task when the pipeline runs. The **On Condition** setting appears on the Task tab with its own condition expression.

This example uses the Condition task.

Prerequisites

- Verify that a pipeline exists, and that it includes stages and tasks.

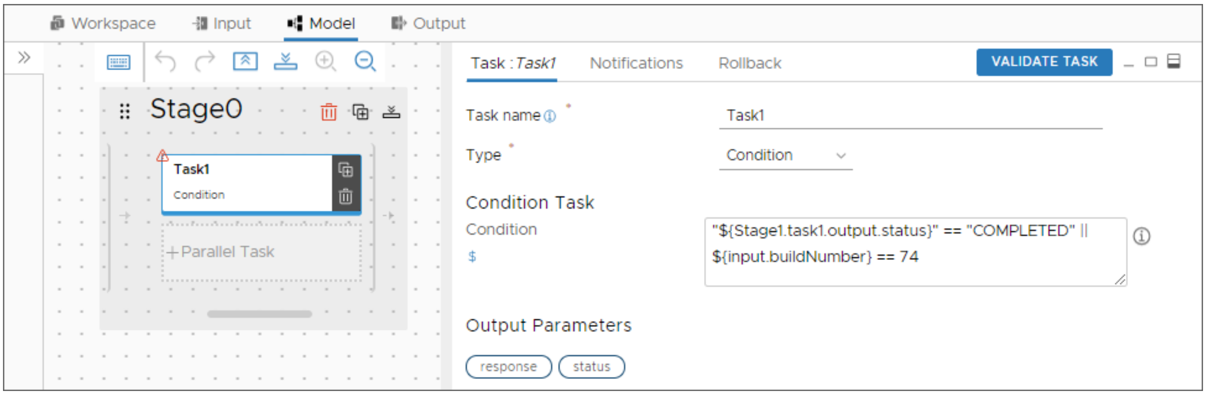
Procedure

1 In your pipeline, determine the decision point where the Condition task must appear.

2 Add the Condition task before the task that depends on its status of pass or fail.

3 Add a condition expression to the Condition task.

For example: `"${Stage1.task1.output.status}" == "COMPLETED" || ${input.buildNumber} == 74`



Task name ⓘ * Task1

Type * Condition

Condition Task

Condition $ "${Stage1.task1.output.status}" == "COMPLETED" || ${input.buildNumber} == 74

Output Parameters

4 Validate the task.

5 Save the pipeline, then enable and run it.

Results

Watch the pipeline executions and notice whether the pipeline continues running, or stops at the Condition task.

What to do next

If you roll back a pipeline deployment, you can also use the Condition task. For example, in a rollback pipeline, the Condition task helps Code Stream mark a pipeline failure based on the condition expression, and can trigger a single rollback flow for various failure types.

To roll back a deployment, see How do I roll back my deployment in Code Stream.

What variables and expressions can I use when binding pipeline tasks in Code Stream

With variables and expressions, you can use input parameters and output parameters with your pipeline tasks. The parameters you enter bind your pipeline task to one or more variables, expressions, or conditions, and determine the pipeline behavior when it runs.

Pipelines can run simple or complex software delivery solutions

When you bind pipeline tasks together, you can include default and complex expressions. As a result, your pipeline can run simple or complex software delivery solutions.

To create the parameters in your pipeline, click the **Input** or **Output** tab, and add a variable by entering the dollar sign **$** and an expression. For example, this parameter is used as a task input that calls a URL: `${Stage0.Task3.input.URL}`.

The format for variable bindings uses syntax components called scopes and keys. The `SCOPE` defines the context as input or output, and the `KEY` defines the details. In the parameter example `${Stage0.Task3.input.URL}`, the `input` is the `SCOPE` and the URL is the `KEY`.

Output properties of any task can resolve to any number of nested levels of variable binding.

To learn more about using variable bindings in pipelines, see How do I use variable bindings in Code Stream pipelines.

Using dollar expressions with scopes and keys to bind pipeline tasks

You can bind pipeline tasks together by using expressions in dollar sign variables. You enter expressions as `${SCOPE.KEY.<PATH>}`.

To determine the behavior of a pipeline task, in each expression, `SCOPE` is the context that Code Stream uses. The scope looks for a `KEY`, which defines the detail for the action that the task takes. When the value for `KEY` is a nested object, you can provide an optional `PATH`.

These examples describe `SCOPE` and `KEY`, and show you how you can use them in your pipeline.

Table 3-8. Using SCOPE and KEY

<table>
<tr><th>SCOPE</th><th>Purpose of expression and example</th><th>KEY</th><th>How to use SCOPE and KEY in your pipeline</th></tr>
<tr><td><b>input</b></td><td>Input properties of a pipeline:
<code>${input.input1}</code></td><td>Name of the input property</td><td>To refer to the input property of a pipeline in a task, use this format:
<pre>
tasks:
  mytask:
    type: REST
    input:
      url: $
{input.url}
      action: get
</pre>
<pre>
input:
  url: https://
www.vmware.com
</pre></td></tr>
<tr><td><b>output</b></td><td>Output properties of a pipeline:
<code>${output.output1}</code></td><td>Name of the output property</td><td>To refer to an output property for sending a notification, use this format:
<pre>
notifications:
 email:
 - endpoint:
MyEmailEndpoint
   subject:
"Deployment
Successful"
   event: COMPLETED
   to:
   -
user@example.org
   body: |
     Pipeline
deployed
the service
successfully.
Refer $
{output.serviceURL}
</pre></td></tr>
</table>

Table 3-8. Using SCOPE and KEY (continued)

<table>
<tr><th>SCOPE</th><th>Purpose of expression and example</th><th>KEY</th><th>How to use SCOPE and KEY in your pipeline</th></tr>
<tr><td><b>task input</b></td><td>Input to a task:
<code>${MY_STAGE.MY_TASK.input.SOMETHING}</code></td><td>Indicates the input of a task in a notification</td><td>When a Jenkins job starts, it can refer to the name of the job triggered from the task input. In this case, send a notification by using this format:
<pre>
notifications:
  email:
  - endpoint:
MyEmailEndpoint
    stage: MY_STAGE
    task: MY_TASK
    subject:
"Build Started"
    event: STARTED
    to:
    -
user@example.org
    body: |
      Jenkins job $
{MY_STAGE.MY_TASK.i
nput.job} started
for commit id $
{input.COMMITID}.
</pre></td></tr>
<tr><td><b>task output</b></td><td>Output of a task:
<code>${MY_STAGE.MY_TASK.output.SOMETHING}</code></td><td>Indicates the output of a task in a subsequent task</td><td>To refer to the output of pipeline task 1 in task 2, use this format:
<pre>
taskOrder:
  - task1
  - task2
tasks:
 task1:
   type: REST
   input:
     action: get
     url: https://
www.example.org/api
/status
 task2:
   type: REST
   input:
     action: post
     url: https://
status.internal.exa
mple.org/api/
activity
     payload: $
{MY_STAGE.task1.out
put.responseBody}
</pre></td></tr>
</table>

Table 3-8. Using SCOPE and KEY (continued)

SCOPE	Purpose of expression and example	KEY	How to use SCOPE and KEY in your pipeline

var	Variable:
	`${var.myVariable}`
		Refer to variable in an endpoint	To refer to a secret variable in an endpoint for a password, use this format:
			`---`
			`project: MyProject`
			`kind: ENDPOINT`
			`name: MyJenkinsServer`
			`type: jenkins`
			`properties:`
			` url: https://jenkins.example.com`
			` username: jenkinsUser`
			` password: ${var.jenkinsPassword}`

var	Variable:
	`${var.myVariable}`
		Refer to variable in a pipeline	To refer to variable in a pipeline URL, use this format:
			`tasks:`
			` task1:`
			`  type: REST`
			`  input:`
			`   action: get`
			`   url: ${var.MY_SERVER_URL}`

task status	Status of a task:
	`${MY_STAGE.MY_TASK.status}`
	`${MY_STAGE.MY_TASK.statusMessage}`		

stage status	Status of a stage:
	`${MY_STAGE.status}`
	`${MY_STAGE.statusMessage}`		

Default Expressions

You can use variables with expressions in your pipeline. This summary includes the default expressions that you can use.

Expression	Description
`${comments}`	Comments provided when at pipeline execution request.
`${duration}`	Duration of the pipeline execution.
`${endTime}`	End time of the pipeline execution in UTC, if concluded.
`${executedOn}`	Same as the start time, the starting time of the pipeline execution in UTC.
`${executionId}`	ID of the pipeline execution.
`${executionUrl}`	URL that navigates to the pipeline execution in the user interface.
`${name}`	Name of the pipeline.
`${requestBy}`	Name of the user who requested the execution.
`${stageName}`	Name of the current stage, when used in the scope of a stage.
`${startTime}`	Starting time of the pipeline execution in UTC.
`${status}`	Status of the execution.
`${statusMessage}`	Status message of the pipeline execution.
`${taskName}`	Name of the current task, when used at a task input or notification.

Using SCOPE and KEY in pipeline tasks

You can use expressions with any of the supported pipeline tasks. These examples show you how to define the `SCOPE` and `KEY`, and confirm the syntax. The code examples use `MY_STAGE` and `MY_TASK` as the pipeline stage and task names.

To find out more about available tasks, see What types of tasks are available in Code Stream.

Table 3-9. Gating tasks

Task	Scope	Key	How to use SCOPE and KEY in the task
User Operation			

	Input	`summary`: Summary of the request for the User Operation
		`description`: Description of the request for the User Operation
		`approvers`: List of approver email addresses, where each entry can be a variable with a comma, or use a semi-colon for separate emails
		`approverGroups`: List of approver group addresses for the platform and identity
		`sendemail`: Optionally sends an email notification upon request or response when set to true
		`expirationInDays`: Number of days that represents the expiry time of the request
			`${MY_STAGE.MY_TASK.input.summary}`
			`${MY_STAGE.MY_TASK.input.description}`
			`${MY_STAGE.MY_TASK.input.approvers}`
			`${MY_STAGE.MY_TASK.input.approverGroups}`
			`${MY_STAGE.MY_TASK.input.sendemail}`
			`${MY_STAGE.MY_TASK.input.expirationInDays}`

	Output	`index`: Six-digit hexadecimal string that represents the request
		`respondedBy`: Account name of the person who approved/rejected the User Operation
		`respondedByEmail`: Email address of the person who responded
		`comments`: Comments provided during response
			`${MY_STAGE.MY_TASK.output.index}`
			`${MY_STAGE.MY_TASK.output.respondedBy}`
			`${MY_STAGE.MY_TASK.output.respondedByEmail}`
			`${MY_STAGE.MY_TASK.output.comments}`

Condition			
	Input	`condition`: Condition to evaluate. When the condition evaluates to true, it marks the task as complete, whereas other responses fail the task	`${MY_STAGE.MY_TASK.input.condition}`
	Output	`result`: Result upon evaluation	`${MY_STAGE.MY_TASK.output.response}`

Table 3-10. Pipeline tasks

Task	Scope	Key	How to use SCOPE and KEY in the task
Pipeline			
	Input	`name`: Name of the pipeline to run `inputProperties`: Input properties to pass to the nested pipeline execution	`${MY_STAGE.MY_TASK.input.name}` `${MY_STAGE.MY_TASK.input.inputProperties}` # Refer to all properties `${MY_STAGE.MY_TASK.input.inputProperties.input1}` # Refer to value of input1

	Output	`executionStatus`: Status of the pipeline execution
		`executionIndex`: Index of the pipeline execution
		`outputProperties`: Output properties of a pipeline execution
			`${MY_STAGE.MY_TASK.output.executionStatus}`
			`${MY_STAGE.MY_TASK.output.executionIndex}`
			`${MY_STAGE.MY_TASK.output.outputProperties}` # Refer to all properties
			`${MY_STAGE.MY_TASK.output.outputProperties.output1}` # Refer to value of output1

Table 3-11. Automate continuous integration tasks

Task	Scope	Key	How to use SCOPE and KEY in the task
CI			

	Input	`steps`: A set of strings, which represent commands to run
		`export`: Environment variables to preserve after running the steps
		`artifacts`: Paths of artifacts to preserve in the shared path
		`process`: Set of configuration elements for JUnit, JaCoCo, Checkstyle, FindBugs processing
			`${MY_STAGE.MY_TASK.input.steps}`
			`${MY_STAGE.MY_TASK.input.export}`
			`${MY_STAGE.MY_TASK.input.artifacts}`
			`${MY_STAGE.MY_TASK.input.process}`
			`${MY_STAGE.MY_TASK.input.process[0].path}` # Refer to path of the first configuration

	Output	`exports`: Key-value pair, which represents the exported environment variables from the input `export`
		`artifacts`: Path of successfully preserved artifacts
		`processResponse`: Set of processed results for the input `process`
			`${MY_STAGE.MY_TASK.output.exports}` # Refer to all exports
			`${MY_STAGE.MY_TASK.output.exports.myvar}` # Refer to value of **`myvar`**
			`${MY_STAGE.MY_TASK.output.artifacts}`
			`${MY_STAGE.MY_TASK.output.processResponse}`
			`${MY_STAGE.MY_TASK.output.processResponse[0].result}` # Result of the first process configuration

Custom			

Table 3-11. Automate continuous integration tasks (continued)

Task	Scope	Key	How to use SCOPE and KEY in the task

	Input	`name`: Name of the custom integration
		`version`: A version of the custom integration, released or deprecated
		`properties`: Properties to send to the custom integration
			`${MY_STAGE.MY_TASK.input.name}`
			`${MY_STAGE.MY_TASK.input.version}`
			`${MY_STAGE.MY_TASK.input.properties}` #Refer to all properties
			`${MY_STAGE.MY_TASK.input.properties.property1}` #Refer to value of property1

	Output	`properties`: Output properties from the custom integration response	`${MY_STAGE.MY_TASK.output.properties}` #Refer to all properties `${MY_STAGE.MY_TASK.output.properties.property1}` #Refer to value of property1

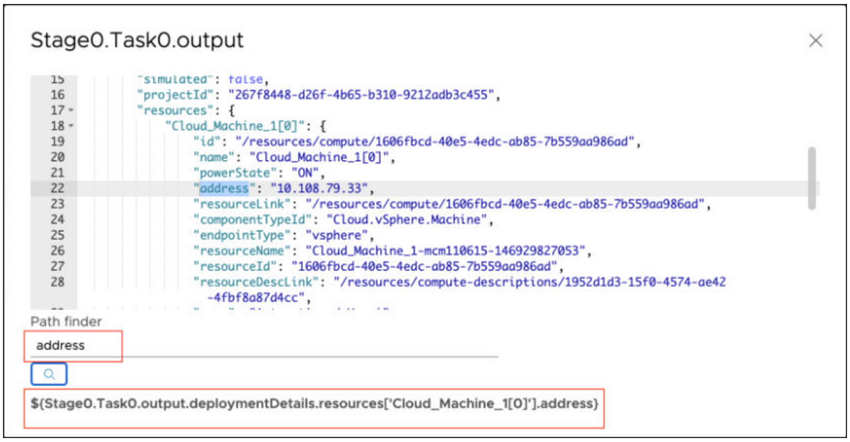
Table 3-12. Automate continuous deployment tasks: Cloud template

Task	Scope	Key	How to use SCOPE and KEY in the task
Cloud template			
	Input	`action`: One of **createDeployment**, **updateDeployment**, **deleteDeployment**, **rollbackDeployment** `blueprintInputParams`: Used for the **create deployment** and **update deployment** actions `allowDestroy`: Machines can be destroyed in the update deployment process. **CREATE_DEPLOYMENT** ■ `blueprintName`: Name of the cloud template ■ `blueprintVersion`: Version of the cloud template OR ■ `fileUrl`: URL of the remote cloud template YAML, after selecting a GIT server. **UPDATE_DEPLOYMENT** Any of these combinations: ■ `blueprintName`: Name of the cloud template ■ `blueprintVersion`: Version of the cloud template OR ■ `fileUrl`: URL of the remote cloud template YAML, after selecting a GIT server. ------ ■ `deploymentId`: ID of the deployment OR ■ `deploymentName`: Name of the deployment ------ **DELETE_DEPLOYMENT** ■ `deploymentId`: ID of the deployment	

Table 3-12. Automate continuous deployment tasks: Cloud template (continued)

Task	Scope	Key	How to use SCOPE and KEY in the task
		OR - `deploymentName`: Name of the deployment **ROLLBACK_DEPLOYMENT** Any of these combinations: - `deploymentId`: ID of the deployment OR - `deploymentName`: Name of the deployment ------ - `blueprintName`: Name of the cloud template - `rollbackVersion`: Version to roll back to	
	Output		Parameters that can bind to other tasks or to the output of a pipeline: - Deployment Name can be accessed as **`${Stage0.Task0.output.deploymentName}`** - Deployment Id can be accessed as **`${Stage0.Task0.output.deploymentId}`** - Deployment Details is a complex object, and internal details can be accessed by using the JSON results. To access any property, use the dot operator to follow the JSON hierarchy. For example, to access the address of resource Cloud_Machine_1[0], the **$** binding is: **`${Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].address}`** Similarly, for the flavor, the **$** binding is: **`${Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].flavor}`** In the Code Stream user interface, you can obtain the **$** bindings for any property. 1 In the task output property area, click **VIEW OUTPUT JSON**. 2 To find the **$** binding, enter any property. 3 Click the search icon, which displays the corresponding **$** binding.

Example JSON output:



Stage0.Task0.output

```
            "simulated": false,
            "projectId": "267f8448-d26f-4b65-b310-9212adb3c455",
            "resources": {
                "Cloud_Machine_1[0]": {
                    "id": "/resources/compute/1606fbcd-40e5-4edc-ab85-7b559aa986ad",
                    "name": "Cloud_Machine_1[0]",
                    "powerState": "ON",
                    "address": "10.108.79.33",
                    "resourceLink": "/resources/compute/1606fbcd-40e5-4edc-ab85-7b559aa986ad",
                    "componentTypeId": "Cloud.vSphere.Machine",
                    "endpointType": "vsphere",
                    "resourceName": "Cloud_Machine_1-mcm110615-146929827053",
                    "resourceId": "1606fbcd-40e5-4edc-ab85-7b559aa986ad",
                    "resourceDescLink": "/resources/compute-descriptions/1952d1d3-15f0-4574-ae42
                      -4fbf8a87d4cc",
```

Path finder

address

${Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].address}

Sample deployment details object:

```
{
    "id": "6a031f92-d0fa-42c8-bc9e-3b260ee2f65b",
    "name": "deployment_6a031f92-d0fa-42c8-bc9e-3b260ee2f65b",
    "description": "Pipeline Service triggered operation",
    "orgId": "434f6917-4e34-4537-b6c0-3bf3638a71bc",
    "blueprintId": "8d1dd801-3a32-4f3b-adde-27f8163dfe6f",
    "blueprintVersion": "1",
    "createdAt": "2020-08-27T13:50:24.546215Z",
    "createdBy": "user@vmware.com",
    "lastUpdatedAt": "2020-08-27T13:52:50.674957Z",
    "lastUpdatedBy": "user@vmware.com",
    "inputs": {},
    "simulated": false,
    "projectId": "267f8448-d26f-4b65-b310-9212adb3c455",
    "resources": {
        "Cloud_Machine_1[0]": {
            "id": "/resources/compute/1606fbcd-40e5-4edc-ab85-7b559aa986ad",
            "name": "Cloud_Machine_1[0]",
            "powerState": "ON",
            "address": "10.108.79.33",
            "resourceLink": "/resources/compute/1606fbcd-40e5-4edc-ab85-7b559aa986ad",
            "componentTypeId": "Cloud.vSphere.Machine",
            "endpointType": "vsphere",
            "resourceName": "Cloud_Machine_1-mcm110615-146929827053",
            "resourceId": "1606fbcd-40e5-4edc-ab85-7b559aa986ad",
            "resourceDescLink": "/resources/compute-descriptions/1952d1d3-15f0-4574-
ae42-4fbf8a87d4cc",
            "zone": "Automation / Vms",
            "countIndex": "0",
            "image": "ubuntu",
            "count": "1",
            "flavor": "small",
            "region": "MYBU",
            "_clusterAllocationSize": "1",
            "osType": "LINUX",
            "componentType": "Cloud.vSphere.Machine",
            "account": "bha"
        }
    },
    "status": "CREATE_SUCCESSFUL",
    "deploymentURI": "https://api.yourenv.com/automation-ui/#/deployment-ui;ash=/deployment/
6a031f92-d0fa-42c8-bc9e-3b260ee2f65b"
}
```

Table 3-13. Automate continuous deployment tasks: Kubernetes

Task	Scope	Key	How to use SCOPE and KEY in the task
Kubernetes			

	Input	`action`: One of **GET**, **CREATE**, **APPLY**, **DELETE**, **ROLLBACK**
		■ `timeout`: Overall timeout for any action
		■ `filterByLabel`: Additional label to filter on for action **`GET`** using K8S labelSelector
		GET, CREATE, DELETE, APPLY
		■ `yaml`: Inline YAML to process and send to Kubernetes
		■ `parameters`: KEY, VALUE pair - Replace **`$$KEY`** with **`VALUE`** in the in-line YAML input area
		■ `filePath`: Relative path from the SCM Git endpoint, if provided, from which to fetch the YAML
		■ `scmConstants`: KEY, VALUE pair - Replace **`$${KEY}`** with **`VALUE`** in the YAML fetched over SCM.
		■ `continueOnConflict`: When set to true, if a resource is already present, the task continues.
		ROLLBACK
		■ `resourceType`: Resource type to roll back
		■ `resourceName`: Resource name to roll back
		■ `namespace`: Namespace where the rollback must be performed
		■ `revision`: Revision to roll back to
			`${MY_STAGE.MY_TASK.input.action}`
			#Determines the action to perform.
			`${MY_STAGE.MY_TASK.input.timeout}`
			`${MY_STAGE.MY_TASK.input.filterByLabel}`
			`${MY_STAGE.MY_TASK.input.yaml}`
			`${MY_STAGE.MY_TASK.input.parameters}`
			`${MY_STAGE.MY_TASK.input.filePath}`
			`${MY_STAGE.MY_TASK.input.scmConstants}`
			`${MY_STAGE.MY_TASK.input.continueOnConflict}`
			`${MY_STAGE.MY_TASK.input.resourceType}`
			`${MY_STAGE.MY_TASK.input.resourceName}`
			`${MY_STAGE.MY_TASK.input.namespace}`
			`${MY_STAGE.MY_TASK.input.revision}`

	Output	`response`: Captures the entire response
		`response.<RESOURCE>`: Resource corresponds to configMaps, deployments, endpoints, ingresses, jobs, namespaces, pods, replicaSets, replicationControllers, secrets, services, statefulSets, nodes, loadBalancers.
		`response.<RESOURCE>.<KEY>`: The key corresponds to one of apiVersion, kind, metadata, spec
			`${MY_STAGE.MY_TASK.output.response}`
			`${MY_STAGE.MY_TASK.output.response.}`

Table 3-14. Integrate development, test, and deployment applications

Task	Scope	Key	How to use SCOPE and KEY in the task
Bamboo			

	Input	`plan`: Name of the plan
		`planKey`: Plan key
		`variables`: Variables to be passed to the plan
		`parameters`: Parameters to be passed to the plan
			`${MY_STAGE.MY_TASK.input.plan}`
			`${MY_STAGE.MY_TASK.input.planKey}`
			`${MY_STAGE.MY_TASK.input.variables}`
			`${MY_STAGE.MY_TASK.input.parameters}` # Refer to all parameters
			`${MY_STAGE.MY_TASK.input.parameters.param1}` # Refer to value of param1

	Output	`resultUrl`: URL of the resulting build
		`buildResultKey`: Key of the resulting build
		`buildNumber`: Build Number
		`buildTestSummary`: Summary of the tests that ran
		`successfulTestCount`: test result passed
		`failedTestCount`: test result failed
		`skippedTestCount`: test result skipped
		`artifacts`: Artifacts from the build
			`${MY_STAGE.MY_TASK.output.resultUrl}`
			`${MY_STAGE.MY_TASK.output.buildResultKey}`
			`${MY_STAGE.MY_TASK.output.buildNumber}`
			`${MY_STAGE.MY_TASK.output.buildTestSummary}` # Refer to all results
			`${MY_STAGE.MY_TASK.output.successfulTestCount}` # Refer to the specific test count
			`${MY_STAGE.MY_TASK.output.buildNumber}`

Jenkins			
	Input	`job`: Name of the Jenkins job `parameters`: Parameters to be passed to the job	`${MY_STAGE.MY_TASK.input.job}` `${MY_STAGE.MY_TASK.input.parameters}` # Refer to all parameters `${MY_STAGE.MY_TASK.input.parameters.param1}` # Refer to value of a parameter

	Output	`job`: Name of the Jenkins job
		`jobId`: ID of the resulting job, such as 1234
		`jobStatus`: Status in Jenkins
		`jobResults`: Collection of test/ code coverage results
		`jobUrl`: URL of the resulting job run
			`${MY_STAGE.MY_TASK.output.job}`
			`${MY_STAGE.MY_TASK.output.jobId}`
			`${MY_STAGE.MY_TASK.output.jobStatus}`
			`${MY_STAGE.MY_TASK.output.jobResults}` # Refer to all results
			`${MY_STAGE.MY_TASK.output.jobResults.junitResponse}` # Refer to JUnit results
			`${MY_STAGE.MY_TASK.output.jobResults.jacocoRespose}` # Refer to JaCoCo results
			`${MY_STAGE.MY_TASK.output.jobUrl}`

TFS			

Table 3-14. Integrate development, test, and deployment applications (continued)

Task	Scope	Key	How to use SCOPE and KEY in the task
	Input	projectCollection: Project collection from TFS teamProject: Selected project from the available collection buildDefinitionId: Build Definition ID to run	${MY_STAGE.MY_TASK.input.projectCollection} ${MY_STAGE.MY_TASK.input.teamProject} ${MY_STAGE.MY_TASK.input.buildDefinitionId}
	Output	buildId: Resulting build ID buildUrl: URL to visit the build summary logUrl: URL to visit for logs dropLocation: Drop location of artifacts if any	${MY_STAGE.MY_TASK.output.buildId} ${MY_STAGE.MY_TASK.output.buildUrl} ${MY_STAGE.MY_TASK.output.logUrl} ${MY_STAGE.MY_TASK.output.dropLocation}
vRO			
	Input	workflowId: ID of the workflow to be run parameters: Parameters to be passed to the workflow	${MY_STAGE.MY_TASK.input.workflowId} ${MY_STAGE.MY_TASK.input.parameters}
	Output	workflowExecutionId: ID of the workflow execution properties: Output properties from the workflow execution	${MY_STAGE.MY_TASK.output.workflowExecutionId} ${MY_STAGE.MY_TASK.output.properties}

Table 3-15. Integrate other applications through an API

Task	Scope	Key	How to use SCOPE and KEY in the task
REST			
	Input	url: URL to call action: HTTP method to use headers: HTTP headers to pass payload: Request payload fingerprint: Fingerprint to match for a URL that is https allowAllCerts: When set to true, can be any certificate that has a URL of https	${MY_STAGE.MY_TASK.input.url} ${MY_STAGE.MY_TASK.input.action} ${MY_STAGE.MY_TASK.input.headers} ${MY_STAGE.MY_TASK.input.payload} ${MY_STAGE.MY_TASK.input.fingerprint} ${MY_STAGE.MY_TASK.input.allowAllCerts}

Table 3-15. Integrate other applications through an API (continued)

Task	Scope	Key	How to use SCOPE and KEY in the task

	Output	`responseCode`: HTTP response code
		`responseHeaders`: HTTP response headers
		`responseBody`: String format of response received
		`responseJson`: Traversable response if the content-type is **application/ json**
			`${MY_STAGE.MY_TASK.output.responseCode}`
			`${MY_STAGE.MY_TASK.output.responseHeaders}`
			`${MY_STAGE.MY_TASK.output.responseHeaders.header1}` # Refer to response header 'header1'
			`${MY_STAGE.MY_TASK.output.responseBody}`
			`${MY_STAGE.MY_TASK.output.responseJson}` # Refer to response as JSON
			`${MY_STAGE.MY_TASK.output.responseJson.a.b.c}` # Refer to nested object following the a.b.c JSON path in response

Poll			

	Input	`url`: URL to call
		`headers`: HTTP headers to pass
		`exitCriteria`: Criteria to meet to for the task to succeed or fail. A key-value pair of 'success' → Expression, 'failure' → Expression
		`pollCount`: Number of iterations to perform. A Code Stream administrator can set the poll count to a maximum of 10000.
		`pollIntervalSeconds`: Number of seconds to wait between each iteration. The poll interval must be greater than or equal to 60 seconds.
		`ignoreFailure`: When set to true, ignores intermediate response failures
		`fingerprint`: Fingerprint to match for a URL that is https
		`allowAllCerts`: When set to true, can be any certificate that has a URL of https
			`${MY_STAGE.MY_TASK.input.url}`
			`${MY_STAGE.MY_TASK.input.headers}`
			`${MY_STAGE.MY_TASK.input.exitCriteria}`
			`${MY_STAGE.MY_TASK.input.pollCount}`
			`${MY_STAGE.MY_TASK.input.pollIntervalSeconds}`
			`${MY_STAGE.MY_TASK.input.ignoreFailure}`
			`${MY_STAGE.MY_TASK.input.fingerprint}`
			`${MY_STAGE.MY_TASK.input.allowAllCerts}`

	Output	`responseCode`: HTTP response code
		`responseBody`: String format of response received
		`responseJson`: Traversable response if the content-type is **application/ json**
			`${MY_STAGE.MY_TASK.output.responseCode}`
			`${MY_STAGE.MY_TASK.output.responseBody}`
			`${MY_STAGE.MY_TASK.output.responseJson} # Refer to response as JSON`

Table 3-16. Run remote and user-defined scripts

Task	Scope	Key	How to use SCOPE and KEY in the task
PowerShell To run a PowerShell task, you must: ■ Have an active session to a remote Windows host. ■ If you intend to enter a base64 PowerShell command, calculate the overall command length first. For details, see What types of tasks are available in Code Stream.			

	Input	`host`: IP address or hostname of the machine
		`username`: User name to use to connect
		`password`: Password to use to connect
		`useTLS`: Attempt https connection
		`trustCert`: When set to true, trusts self-signed certificates
		`script`: Script to run
		`workingDirectory`: Directory path to switch to before running the script
		`environmentVariables`: A key-value pair of environment variable to set
		`arguments`: Arguments to pass to the script
			`${MY_STAGE.MY_TASK.input.host}`
			`${MY_STAGE.MY_TASK.input.username}`
			`${MY_STAGE.MY_TASK.input.password}`
			`${MY_STAGE.MY_TASK.input.useTLS}`
			`${MY_STAGE.MY_TASK.input.trustCert}`
			`${MY_STAGE.MY_TASK.input.script}`
			`${MY_STAGE.MY_TASK.input.workingDirectory}`
			`${MY_STAGE.MY_TASK.input.environmentVariables}`
			`${MY_STAGE.MY_TASK.input.arguments}`

	Output	`response`: Content of the file `$SCRIPT_RESPONSE_FILE`
		`responseFilePath`: Value of `$SCRIPT_RESPONSE_FILE`
		`exitCode`: Process exit code
		`logFilePath`: Path to file containing stdout
		`errorFilePath`: Path to file containing stderr
			`${MY_STAGE.MY_TASK.output.response}`
			`${MY_STAGE.MY_TASK.output.responseFilePath}`
			`${MY_STAGE.MY_TASK.output.exitCode}`
			`${MY_STAGE.MY_TASK.output.logFilePath}`
			`${MY_STAGE.MY_TASK.output.errorFilePath}`

SSH			

Table 3-16. Run remote and user-defined scripts (continued)

Task	Scope	Key	How to use SCOPE and KEY in the task

	Input	`host`: IP address or hostname of the machine
		`username`: User name to use to connect
		`password`: Password to use to connect (optionally can use privateKey)
		`privateKey`: PrivateKey to use to connect
		`passphrase`: Optional passphrase to unlock privateKey
		`script`: Script to run
		`workingDirectory`: Directory path to switch to before running the script
		`environmentVariables`: Key-value pair of the environment variable to set
			`${MY_STAGE.MY_TASK.input.host}`
			`${MY_STAGE.MY_TASK.input.username}`
			`${MY_STAGE.MY_TASK.input.password}`
			`${MY_STAGE.MY_TASK.input.privateKey}`
			`${MY_STAGE.MY_TASK.input.passphrase}`
			`${MY_STAGE.MY_TASK.input.script}`
			`${MY_STAGE.MY_TASK.input.workingDirectory}`
			`${MY_STAGE.MY_TASK.input.environmentVariables}`

	Output	`response`: Content of the file `$SCRIPT_RESPONSE_FILE`
		`responseFilePath`: Value of `$SCRIPT_RESPONSE_FILE`
		`exitCode`: Process exit code
		`logFilePath`: Path to file containing stdout
		`errorFilePath`: Path to file containing stderr
			`${MY_STAGE.MY_TASK.output.response}`
			`${MY_STAGE.MY_TASK.output.responseFilePath}`
			`${MY_STAGE.MY_TASK.output.exitCode}`
			`${MY_STAGE.MY_TASK.output.logFilePath}`
			`${MY_STAGE.MY_TASK.output.errorFilePath}`

How to use a variable binding between tasks

This example shows you how to use variable bindings in your pipeline tasks.

Table 3-17. Sample syntax formats

Example	Syntax
To use a task output value for pipeline notifications and pipeline output properties	`${<Stage Key>.<Task Key>.output.<Task output key>}`
To refer to the previous task output value as an input for the current task	`${<Previous/Current Stage key>.<Previous task key not in current Task group>.output.<task output key>}`

To learn more

To learn more about binding variables in tasks, see:

- How do I use variable bindings in Code Stream pipelines
- How do I use variable bindings in a condition task to run or stop a pipeline in Code Stream
- What types of tasks are available in Code Stream

How do I send notifications about my pipeline in Code Stream

Notifications are ways to communicate with your teams and let them know the status of your pipelines in Code Stream.

To send notifications when a pipeline runs, you can configure Code Stream notifications based on the status of the entire pipeline, stage, or task.

- An email notification sends an email on:
  - Pipeline completion, waiting, failure, cancelation, or start.
  - Stage completion, failure, or start.
  - Task completion, waiting, failure, or start.
- A ticket notification creates a ticket and assigns it to a team member on:
  - Pipeline failure or completion.
  - Stage failure.
  - Task failure.
- A webhook notification sends a request to another application on:
  - Pipeline failure, completion, waiting, cancelation, or start.
  - Stage failure, completion, or start.
  - Task failure, completion, waiting, or start.

For example, you can configure an email notification on a user operation task to obtain approval at a specific point in your pipeline. When the pipeline runs, this task sends email to the person who must approve the task. If the User Operation task has an expiration timeout set in days, hours, or minutes, the required user must approve the pipeline before the task expires. Otherwise, the pipeline fails as expected.

To create a Jira ticket when a pipeline task fails, you can configure a notification. Or, to send a request to a Slack channel about the status of a pipeline based on the pipeline event, you can configure a webhook notification.

You can use variables in all types of notifications. For example, you can use **`${var}`** in the URL of a Webhook notification.

Prerequisites

- Verify that one or more pipelines are created. See the use cases in Chapter 5 Tutorials for using Code Stream.
- To send email notifications, confirm that you can access a working email server. For help, see your administrator.
- To create tickets, such as a Jira ticket, confirm that the endpoint exists. See What are Endpoints in Code Stream .
- To send a notification based on an integration, you create a webhook notification. Then, you confirm that the webhook is added and working. You can use notifications with applications such as Slack, GitHub, or GitLab.

Procedure

1. Open a pipeline.
2. To create a notification for the overall pipeline status, or the status of a stage or task:

To create a notification for:	What you do:
Pipeline status	Click a blank area on the pipeline canvas.
Status of a stage	Click a blank area in a stage of the pipeline.
Status of a task	Click a task in a stage of the pipeline.

3. Click the **Notifications** tab.
4. Click **Add**, select the type of notification, and configure the notification details.
5. To create a Slack notification when a pipeline succeeds, create a webhook notification.

   a. Select **Webhook**.

   b. To configure the Slack notification, enter the information.

   c. Click **Save.**

   d. When the pipeline runs, the Slack channel receives the notification of the pipeline status. For example, users might see the following on the Slack channel:

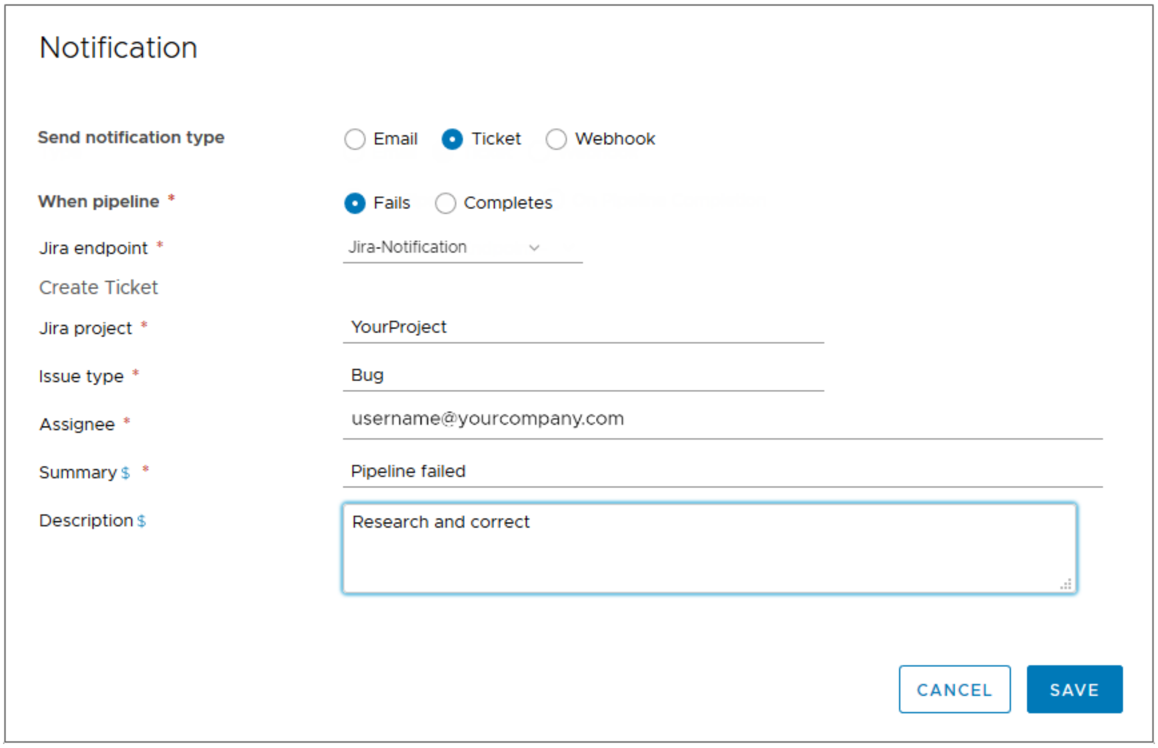
```
Codestream APP [12:01 AM]
Tested by User1 - Staging Pipeline 'User1-Pipeline', Pipeline ID
'e9b5884d809ce2755728177f70f8a' succeeded
```

6 To create a Jira ticket, configure the ticket information.

   a Select **Ticket**.

   b To configure the Jira notification, enter the information.

   c Click **Save.**



Notification

Send notification type: ○ Email ● Ticket ○ Webhook

When pipeline *: ● Fails ○ Completes

Jira endpoint *: Jira-Notification

Create Ticket

Jira project *: YourProject

Issue type *: Bug

Assignee *: username@yourcompany.com

Summary $ *: Pipeline failed

Description $: Research and correct

Results

Congratulations! You learned that you can create various types of notifications in several areas of your pipeline in Code Stream.

What to do next

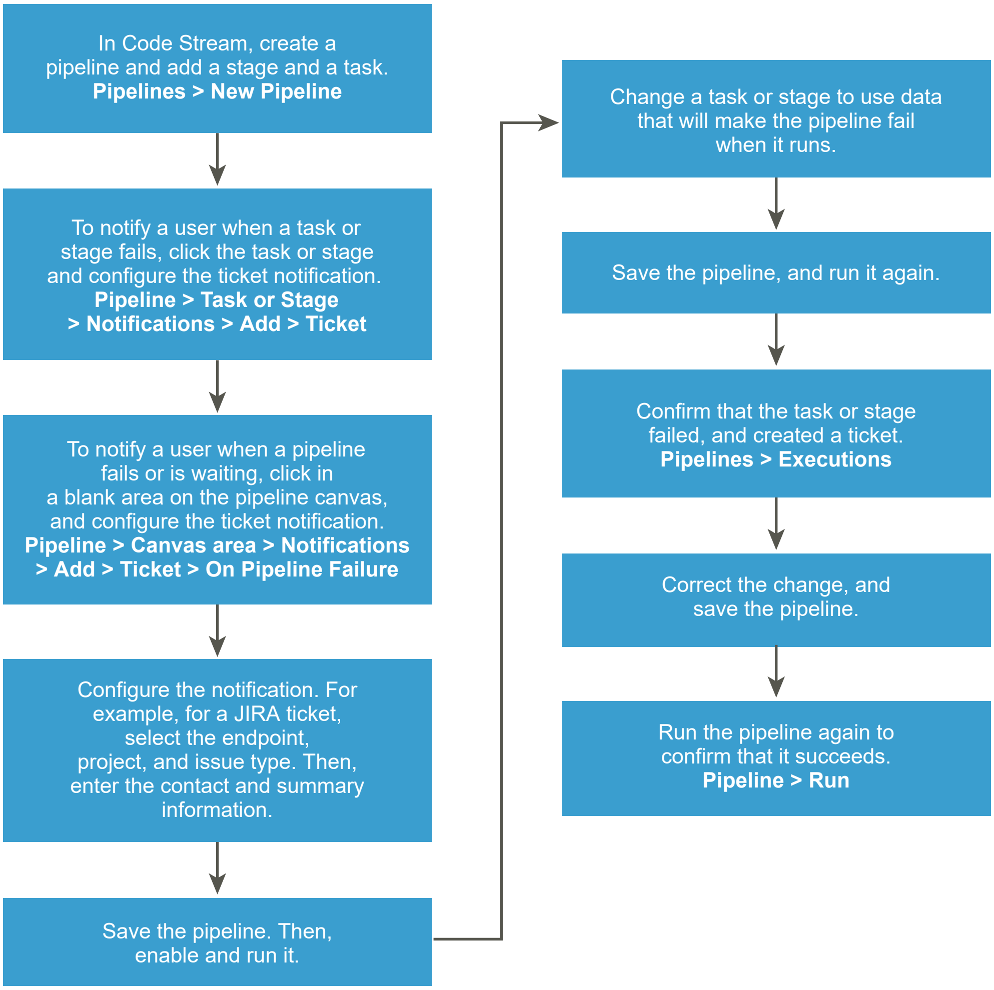
For a detailed example of how to create a notification, see How do I create a Jira ticket in Code Stream when a pipeline task fails.

How do I create a Jira ticket in Code Stream when a pipeline task fails

If a stage or task in your pipeline fails, you can have Code Stream create a Jira ticket. You can assign the ticket to the person who must resolve the problem. You can also create a ticket when the pipeline is waiting, or when it succeeds.

You can add and configure notifications on a task, stage, or pipeline. Code Stream creates the ticket based on the status of the task, stage, or pipeline where you add the notification. For example, if an endpoint is not available, you can have Code Stream create a Jira ticket for the task that fails because it cannot connect to the endpoint.

You can also create notifications when your pipeline succeeds. For example, you can inform your QA team about pipelines that succeed so that they can confirm the build and run a different test pipeline. Or, you can inform your performance team so that they can measure the performance of the pipeline and prepare for an update to staging or production.


In Code Stream, create a pipeline and add a stage and a task. **Pipelines > New Pipeline**

To notify a user when a task or stage fails, click the task or stage and configure the ticket notification. **Pipeline > Task or Stage > Notifications > Add > Ticket**

To notify a user when a pipeline fails or is waiting, click in a blank area on the pipeline canvas, and configure the ticket notification. **Pipeline > Canvas area > Notifications > Add > Ticket > On Pipeline Failure**

Configure the notification. For example, for a JIRA ticket, select the endpoint, project, and issue type. Then, enter the contact and summary information.

Save the pipeline. Then, enable and run it.

Change a task or stage to use data that will make the pipeline fail when it runs.

Save the pipeline, and run it again.

Confirm that the task or stage failed, and created a ticket. **Pipelines > Executions**

Correct the change, and save the pipeline.

Run the pipeline again to confirm that it succeeds. **Pipeline > Run**

This example creates a Jira ticket when a pipeline task fails.

Prerequisites

- Verify that you have a valid Jira account and can log in to your Jira instance.
- Verify that a Jira endpoint exists, and is working.

Procedure

1 In your pipeline, click a task.

2 In the task configuration area, click **Notifications**.

3 Click **Add**, and configure the ticket information.

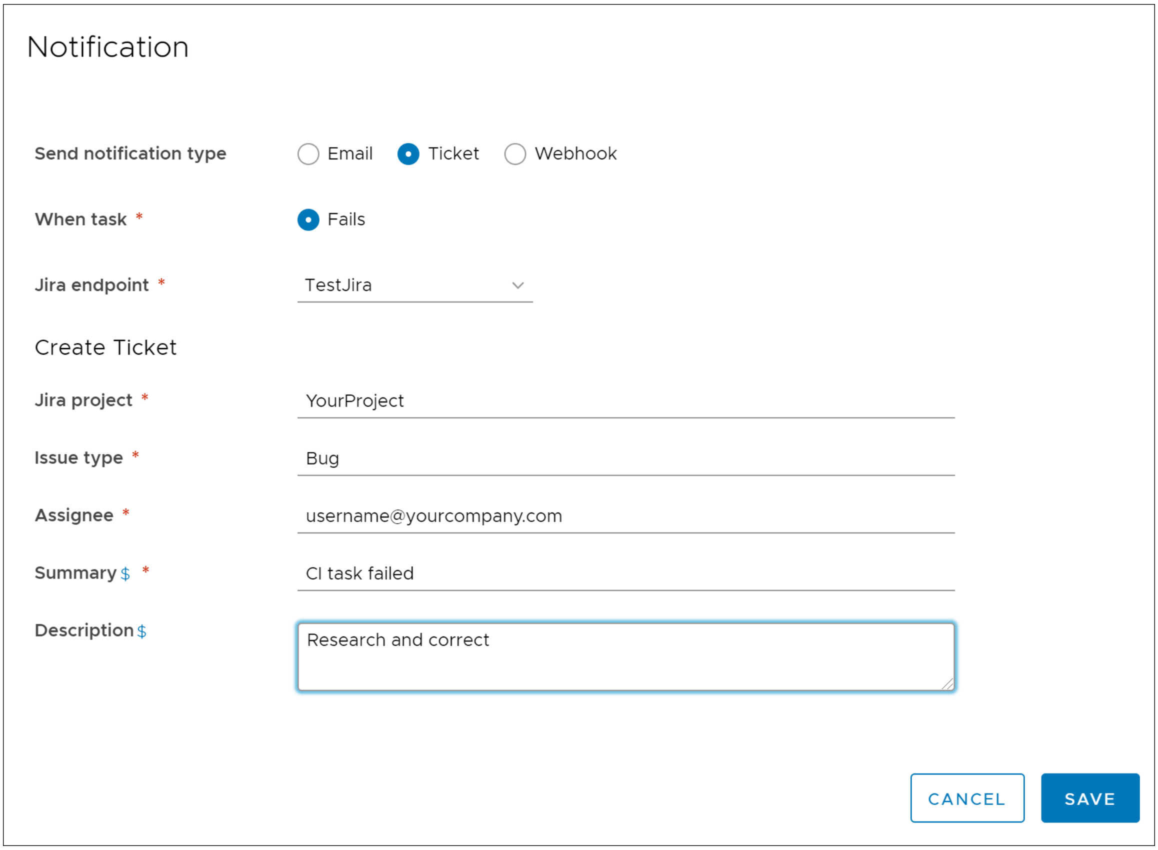
   a Click **Ticket**.

   b Select the Jira endpoint.

   c Enter the Jira project and type of issue.

   d Enter the email address for the person who receives the ticket.

   e Enter a summary and description of the ticket, then click **Save**.



Notification

Send notification type ○ Email ● Ticket ○ Webhook

When task * ● Fails

Jira endpoint * TestJira

Create Ticket

Jira project * YourProject

Issue type * Bug

Assignee * username@yourcompany.com

Summary $ * CI task failed

Description $ Research and correct

4 Save the pipeline, then enable and run it.

5 Test the ticket.

   a Change the task information to include data that makes the task fail.

   b Save the pipeline, and run it again.

   c Click **Executions**, and confirm that the pipeline failed.

   d In the execution, confirm that Code Stream created the ticket and sent it.

   e Change the task information back to correct it, then run the pipeline again and ensure that it succeeds.

Results

Congratulations! You had Code Stream create a Jira ticket when the pipeline task failed, and assigned it to the person who was required to solve it.

What to do next

Continue to add notifications to alert your team about your pipelines.

How do I roll back my deployment in Code Stream

You configure rollback as a pipeline with tasks that return your deployment to a previous stable state following a failure in a deployment pipeline. To roll back if a failure occurs, you attach the rollback pipeline to tasks or stages.

Depending upon your role, your reasons for rollback might vary.

- As a release engineer, I want Code Stream to verify success during a release so that I can know whether to continue with the release or roll back. Possible failures include task failure, a rejection in UserOps, exceeding the metrics threshold.
- As an environment owner, I want to redeploy a previous release so that I can quickly get an environment back to a known-good state.
- As an environment owner, I want to support roll back of a Blue-Green deployment so that I can minimize downtime from failed releases.

When you use a smart pipeline template to create a CD pipeline with the rollback option clicked, rollback is automatically added to tasks in the pipeline. In this use case, you will use the smart pipeline template to define rollback for an application deployment to a Kubernetes cluster using the rolling upgrade deployment model. The smart pipeline template creates a deployment pipeline and one or more rollback pipelines.

- In the deployment pipeline, rollback is required if Update Deployment or Verify Deployment tasks fail.
- In the rollback pipeline, deployment is updated with an old image.

You can also manually create a rollback pipeline using a blank template. Before creating a rollback pipeline, you will want to plan your rollback flow. For more background information about rollback, see Planning for rollback in Code Stream.

Prerequisites

- Verify that you are a member of a project in Code Stream. If you are not, ask a Code Stream administrator to add you as a member of a project. See How do I add a project in Code Stream.
- Set up the Kubernetes clusters where your pipeline will deploy your application. Set up one development cluster and one production cluster.
- Verify that you have a Docker registry setup.

- Identify a project that will group all your work, including your pipeline, endpoints, and dashboards.
- Familiarize yourself with the CD smart template as described in the CD portion of Planning a CICD native build in Code Stream before using the smart pipeline template, for example:
  - Create the Kubernetes development and production endpoints that deploy your application image to the Kubernetes clusters.
  - Prepare the Kubernetes YAML file that creates the Namespace, Service, and Deployment. If you need to download an image from a privately-owned repository, the YAML file must include a section with the Docker config Secret.

Procedure

1 Click **Pipelines** > **New Pipeline** > **Smart Template** > **Continuous Delivery**.

2 Enter the information in the smart pipeline template.

   a Select a project.

   b Enter a pipeline name such as **`RollingUpgrade-Example`**.

   c Select the environments for your application. To add rollback to your deployment, you must select **Prod**.

   d Click **Select**, choose a Kubernetes YAML file, and click **Process**.

   The smart pipeline template displays the available services and deployment environments.

   e Select the service that the pipeline will use for the deployment.

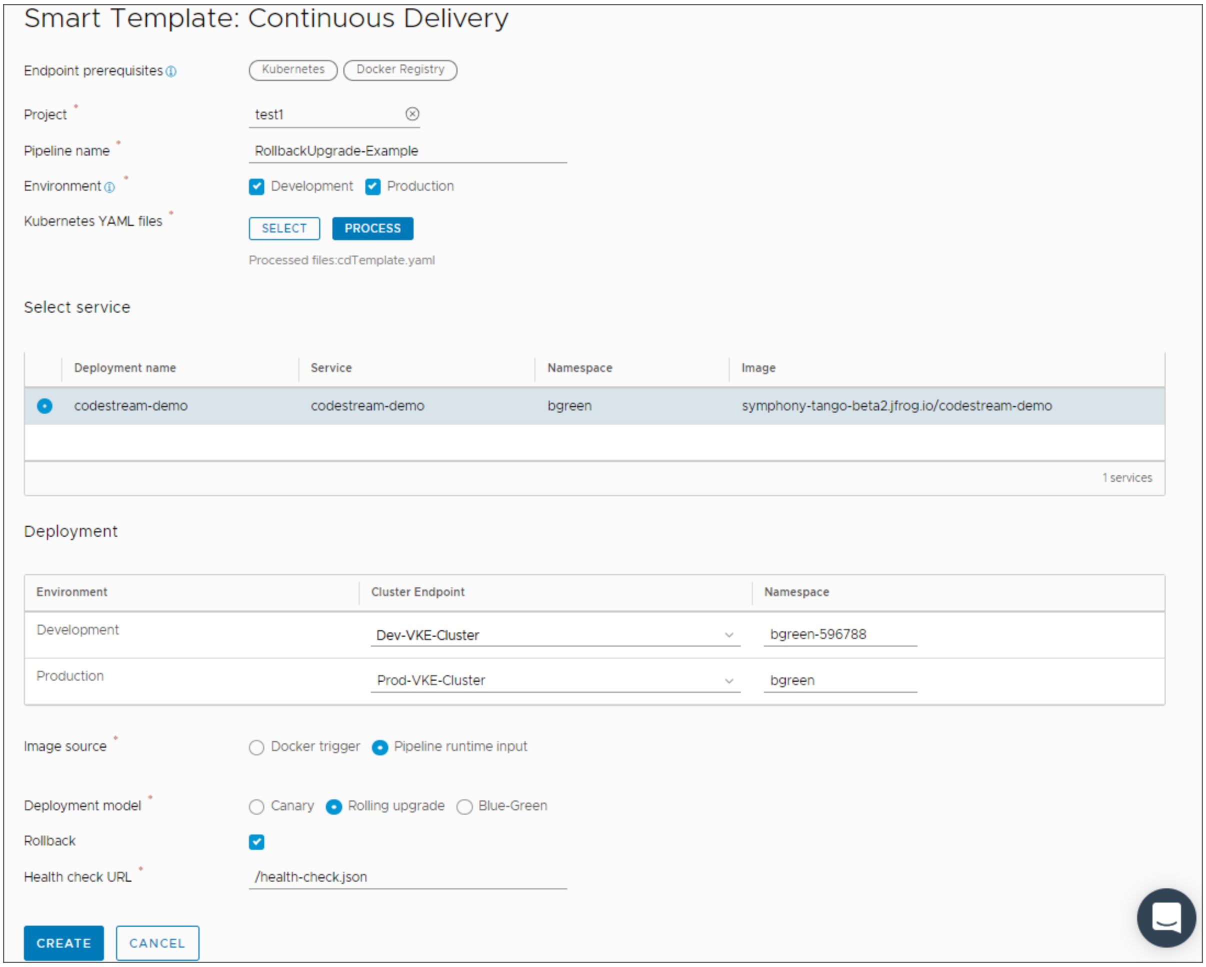
   f Select the cluster endpoints for the Dev environment and the Prod environment.

   g For the Image source, select **Pipeline runtime input**.

   h For the Deployment model, select **Rolling Upgrade**.

   i Click **Rollback**.

   j Provide the **Health check URL**.



Smart Template: Continuous Delivery

Endpoint prerequisites ⓘ Kubernetes Docker Registry

Project * test1

Pipeline name * RollbackUpgrade-Example

Environment ⓘ * ☑ Development ☑ Production

Kubernetes YAML files *

Processed files:cdTemplate.yaml

Select service

	Deployment name	Service	Namespace	Image
●	codestream-demo	codestream-demo	bgreen	symphony-tango-beta2.jfrog.io/codestream-demo

1 services

Deployment

Environment	Cluster Endpoint	Namespace
Development	Dev-VKE-Cluster	bgreen-596788
Production	Prod-VKE-Cluster	bgreen

Image source * ○ Docker trigger ● Pipeline runtime input

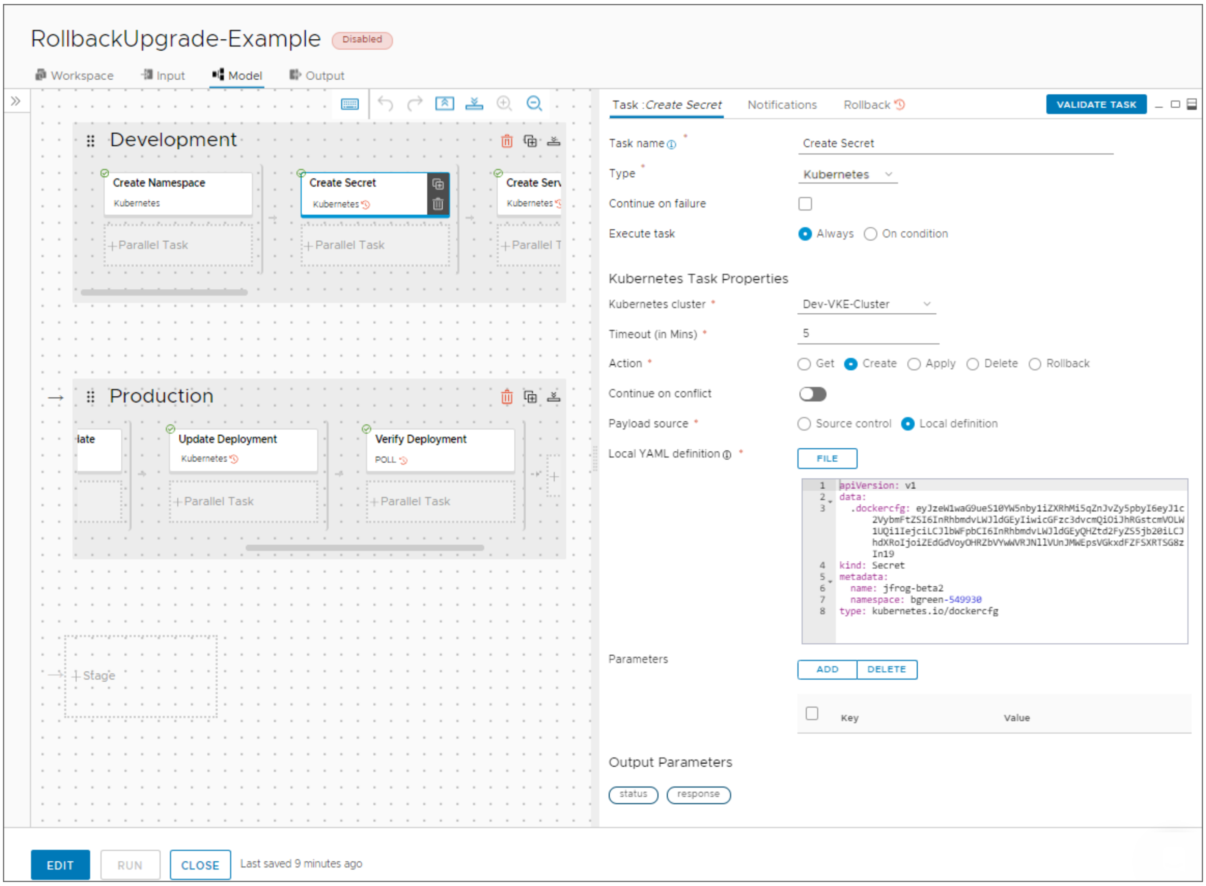
Deployment model * ○ Canary ● Rolling upgrade ○ Blue-Green

Rollback ☑

Health check URL * /health-check.json

3 To create the pipeline named RollbackUpgrade-Example, click **Create**.

The pipeline named RollbackUpgrade-Example appears, and the rollback icon appears on tasks that can roll back in the Development stage and the Production stage.



Task name * Create Secret

Type * Kubernetes

Continue on failure □

Execute task ● Always ○ On condition

Kubernetes Task Properties

Kubernetes cluster * Dev-VKE-Cluster

Timeout (in Mins) * 5

Action * ○ Get ● Create ○ Apply ○ Delete ○ Rollback

Continue on conflict

Payload source * ○ Source control ● Local definition

Local YAML definition * FILE

```
apiVersion: v1
data:
  .dockercfg: eyJzeW1waG9ueS10YW5nby1iZXRhMi5qZnJvZy5pbyI6eyJ1c
      2VybmFtZSI6InRhbmdvLWJldGEyIiwicGFzc3dvcmQiOiJhRGstcmVOLW
      1UQi1IejciLCJlbWFpbCI6InRhbmdvLWJldGEyQHZtd2FyZS5jb20iLCJ
      hdXRoIjoiZEdGdVoyOHRZbVYwWVRJNllVUnJMWEpsVGkxdFZFSXRTSG8z
      In19
kind: Secret
metadata:
  name: jfrog-beta2
  namespace: bgreen-549930
type: kubernetes.io/dockercfg
```

Parameters ADD DELETE

□	Key	Value

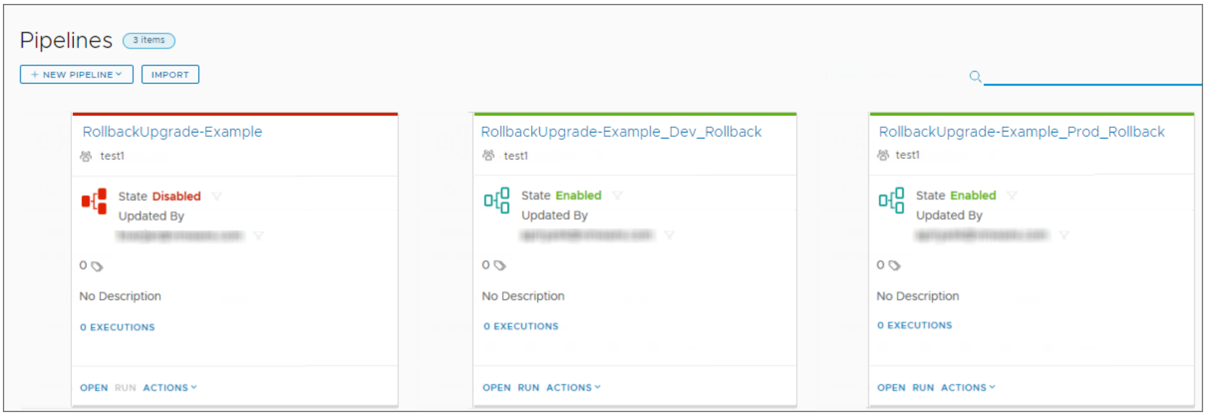
Output Parameters

status response

4 Close the pipeline.

On the Pipelines page, the pipeline that you created appears, and a new pipeline for each stage in your pipeline appears.

- RollingUpgrade-Example. Code Stream deactivates the pipeline that you created by default, which ensures that you review it before you run it.
- RollingUpgrade-Example_Dev_Rollback. Failure of tasks in the development stage, such as **Create service**, **Create secret**, **Create deployment**, and **Verify deployment** invoke this rollback development pipeline. To ensure the rollback of development tasks, Code Stream enables the rollback development pipeline by default.
- RollingUpgrade-Example_Prod_Rollback. Failure of tasks in the production stage, such as **Deploy phase 1**, **Verify phase 1**, **Deploy Rollout phase**, **Finish Rollout phase**, and **Verify rollout phase** invoke this rollback production pipeline. To ensure the rollback of production tasks, Code Stream enables the rollback production pipeline by default.



RollbackUpgrade-Example
test1
State Disabled
Updated By
[redacted]
0
No Description
0 EXECUTIONS
OPEN RUN ACTIONS ˅

RollbackUpgrade-Example_Dev_Rollback
test1
State Enabled
Updated By
[redacted]
0
No Description
0 EXECUTIONS
OPEN RUN ACTIONS ˅

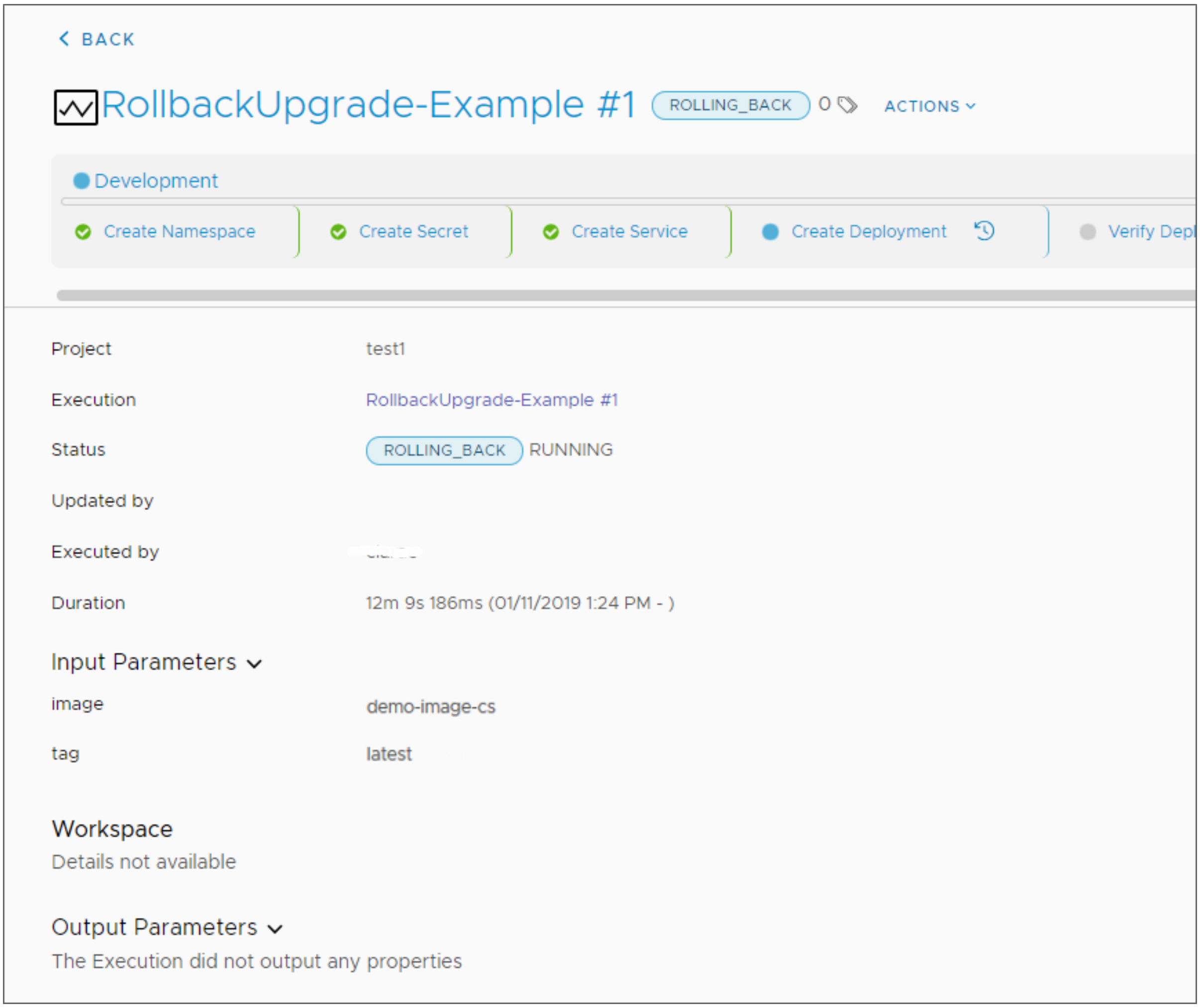
RollbackUpgrade-Example_Prod_Rollback
test1
State Enabled
Updated By
[redacted]
0
No Description
0 EXECUTIONS
OPEN RUN ACTIONS ˅

5 Enable and run the pipeline you created.

When you start the run, Code Stream prompts you for input parameters. You provide the image and tag for the endpoint in the Docker repository that you are using.

6 On the Executions page, select **Actions > View Execution** and watch the pipeline execution.

The pipeline starts **RUNNING** and moves through the Development stage tasks. If the pipeline fails to run a task during the Development stage, the pipeline named RollingUpgrade-Example_Dev_Rollback triggers and rolls back the deployment, and the pipeline status changes to **ROLLING_BACK**.



Project	test1
Execution	RollbackUpgrade-Example #1
Status	ROLLING_BACK RUNNING
Updated by	
Executed by	[redacted]
Duration	12m 9s 186ms (01/11/2019 1:24 PM -)

Input Parameters

image	demo-image-cs
tag	latest

Workspace

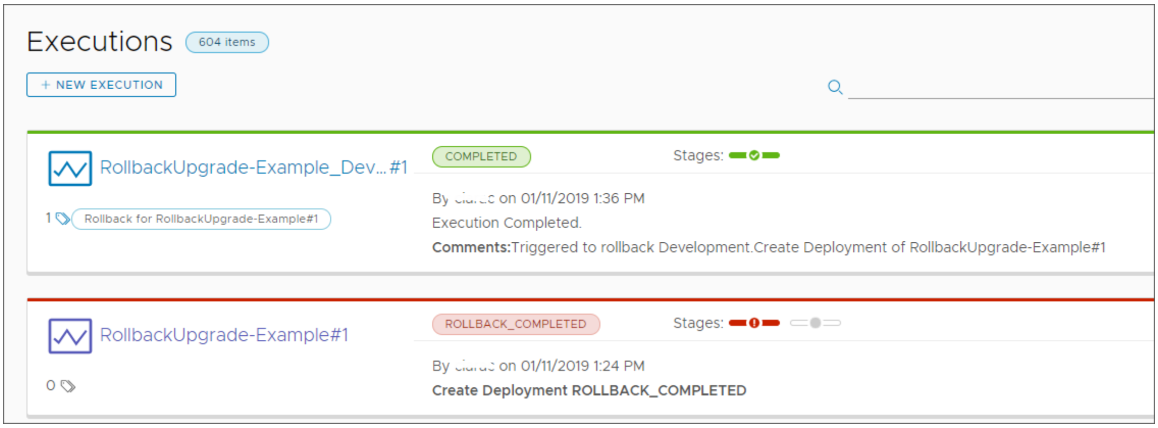
Details not available

Output Parameters

The Execution did not output any properties

After rollback, the Executions page lists two RollingUpgrade-Example pipeline executions.

- The pipeline you created rolled back and displays **ROLLBACK_COMPLETED**.
- The rollback development pipeline that triggered and performed the rollback displays **COMPLETED**.


Executions
604 items
+ NEW EXECUTION
RollbackUpgrade-Example_Dev... #1
1
Rollback for RollbackUpgrade-Example#1
COMPLETED
Stages:
RollbackUpgrade-Example#1
0
ROLLBACK_COMPLETED
Stages:

By [redacted] on 01/11/2019 1:36 PM
Execution Completed.
Comments: Triggered to rollback Development.Create Deployment of RollbackUpgrade-Example#1

By [redacted] on 01/11/2019 1:24 PM
Create Deployment ROLLBACK_COMPLETED

Results

Congratulations! You successfully defined a pipeline with rollback and watched Code Stream roll back the pipeline at the point of failure.

Planning to natively build, integrate, and deliver your code in Code Stream

4

Before you have Code Stream build, integrate, and deliver your code by using the native capability that creates a CICD, CI, or CD pipeline for you, plan your native build. Then, you can create your pipeline by using one of the smart pipeline templates, or by manually adding stages and tasks.

To plan for your continuous integration and continuous delivery build, we included several examples that show you how. These plans describe the prerequisites and overviews that can help you prepare and use the native build capability effectively when you build your pipelines.

This chapter includes the following topics:

Configuring the Pipeline Workspace

To run continuous integration tasks and custom tasks, you must configure a workspace for your Code Stream pipeline.

In the pipeline workspace, select the **Type** as Docker or Kubernetes, and provide the respective endpoint. The Docker and Kubernetes platforms manage the entire life cycle of the container that Code Stream deploys for running the continuous integration (CI) task or custom task.

- The Docker workspace requires the Docker host endpoint, builder image URL, image registry, working directory, cache, environment variables, CPU limit, and memory limit. You can also create a clone of the Git repository.

- The Kubernetes workspace requires the Kubernetes API endpoint, builder image URL, image registry, namespace, NodePort, Persistent Volume Claim (PVC), working directory, environment variables, CPU limit, and memory limit. You can also create a clone of the Git repository.

The pipeline workspace configuration has many common parameters, and other parameters that are specific to the type of workspace, as the following table describes.

Table 4-1. Workspace areas, details, and availability

Selection	Description	Details and availability
Type	Type of workspace.	Available with Docker or Kubernetes.
Host Endpoint	Host endpoint where the continuous integration (CI) and custom tasks run.	Available with the Docker workspace when you select the Docker host endpoint. Available with the Kubernetes workspace when you select the Kubernetes API endpoint.
Builder image URL	Name and location of the builder image. A container is created by using this image on the Docker host and the Kubernetes cluster. The continuous integration (CI) tasks and custom tasks run inside this container.	Example: **`fedora:latest`** The builder image must have `curl` or `wget`.
Image registry	If the builder image is available in a registry, and if the registry requires credentials, you must first create an Image Registry endpoint, then select it here so that the image can be pulled from the registry.	Available with the Docker and Kubernetes workspaces.
Working directory	The working directory is the location inside the container where the steps of the continuous integration (CI) task run, and is the location where the code is cloned when a Git webhook triggers a pipeline run.	Available with Docker or Kubernetes.
Namespace	If you do not enter a Namespace, Code Stream creates a unique name in the Kubernetes cluster that you provided.	Specific to the Kubernetes workspace.
Proxy	To communicate with the workspace pod in the Kubernetes cluster, Code Stream deploys a single proxy instance in the namespace `codestream-proxy` for each Kubernetes cluster. You can choose either the **NodePort** or **LoadBalancer** type, based on the cluster configuration. Which option you choose depends on the nature of the deployed Kubernetes cluster. ■ Typically, if the Kubernetes API server URL that gets specified in the endpoint is exposed through one of the master nodes, choose **NodePort**. ■ If the Kubernetes API server URL is exposed by a Load Balancer, as in the case of Amazon EKS (Elastic Kubernetes Service), choose **LoadBalancer**.	

Table 4-1. Workspace areas, details, and availability (continued)

Selection	Description	Details and availability
NodePort	Code Stream uses NodePort to communicate with the container running inside the Kubernetes cluster. If you do not select a port, Code Stream uses an ephemeral port that Kubernetes assigns. You must ensure that the firewall rules are configured to allow ingress to the ephemeral port range (30000-32767). If you enter a port, you must ensure that another service in the cluster is not already using it, and that the port is allowed in the firewall rules.	Specific to the Kubernetes workspace.
Persistent Volume Claim	Provides a way for the Kubernetes workspace to persist files across pipeline runs. When you provide a persistent volume claim name, it can store the logs, artifacts, and cache. For more information about creating a persistent volume claim, see the Kubernetes documentation at `https://kubernetes.io/docs/concepts/storage/persistent-volumes/`.	Specific to the Kubernetes workspace.
Environment variables	Key-value pairs that are passed here will be available to all continuous integration (CI) tasks and custom tasks in a pipeline when it runs.	Available with Docker or Kubernetes. References to variables can be passed here. Environment variables provided in the workspace are passed to all continuous integration (CI) tasks and custom tasks in the pipeline. If environment variables are not passed here, those variables must be explicitly passed to each continuous integration (CI) task and custom task in the pipeline.
CPU limits	Limits for CPU resources for the continuous integration (CI) container or custom task container.	The default is **`1`**.
Memory limits	Limits for memory for the continuous integration (CI) container or custom task container.	The unit is `MB`.

Table 4-1. Workspace areas, details, and availability (continued)

Selection	Description	Details and availability
Git clone	When you select **Git clone**, and a Git webhook invokes the pipeline, the code is cloned into the workspace (container).	If **Git clone** is not enabled, you must configure an additional, explicit continuous integration (CI) task in the pipeline to clone the code first, then perform other steps such as build and test.

Cache	The Code Stream workspace allows you to cache a set of directories or files to speed up subsequent pipeline runs. Examples of these directories include `.m2` and `npm_modules`. If you do not require caching of data between pipeline runs, you do not need to provide a persistent volume claim.
	Artifacts such as files or directories in the container are cached for re-use across pipeline runs. For example, `node_modules` or `.m2` folders can be cached. **Cache** accepts a list of paths.
	For example:
	`workspace:`
	`  type: K8S`
	`  endpoint: K8S-Micro`
	`  image: fedora:latest`
	`  registry: Docker Registry`
	`  path: ''`
	`  cache:`
	`    - /path/to/m2`
	`    - /path/to/node_modules`
		Specific to type of workspace.
		In the Docker workspace, **Cache** is achieved by using a shared path in the Docker host for persisting the cached data, artifacts, and logs.
		In the Kubernetes workspace, you can use **Cache** only when you provide a persistent volume claim. If you do not provide a persistent volume claim, **Cache** is not enabled.

When using a Kubernetes API endpoint in the pipeline workspace, Code Stream creates the necessary Kubernetes resources such as ConfigMap, Secret, and Pod to run the continuous integration (CI) task or custom task. Code Stream communicates with the container by using the NodePort.

To share data across pipeline runs, you must provide a persistent volume claim, and Code Stream will mount the persistent volume claim to the container to store the data, and use it for subsequent pipeline runs.

Planning a CICD native build in Code Stream before using the smart pipeline template

To create a continuous integration and continuous delivery (CICD) pipeline in Code Stream, you can use the CICD smart pipeline template. To plan your CICD native build, you gather the information for the smart pipeline template before you create the pipeline in this example plan.

To create a CICD pipeline, you must plan for both the continuous integration (CI) and continuous delivery (CD) stages of your pipeline.

After you enter the information in the smart pipeline template and save it, the template creates a pipeline that includes stages and tasks. It also indicates the deployment destination of your image based on the types of environment you select, such as Dev and Prod. The pipeline will publish your container image, and perform the actions required that run it. After your pipeline runs, you can monitor trends across the pipeline executions.

When a pipeline includes an image from Docker Hub, you must ensure that the image has `cURL` or `wget` embedded before you run the pipeline. When the pipeline runs, Code Stream downloads a binary file that uses `cURL` or `wget` to run commands.

For information about configuring the workspace, see Configuring the Pipeline Workspace.

Planning the Continuous Integration (CI) stage

To plan the CI stage of your pipeline, you set up the external and internal requirements, and determine the information needed for the CI portion of the smart pipeline template. Here is a summary.

This example uses a Docker workspace.

Endpoints and repositories that you'll need:

- A Git source code repository where developers check in their code. Code Stream pulls the latest code into the pipeline when developers commit changes.
- A Git endpoint for the repository where the developer source code resides.
- A Docker endpoint for the Docker build host that will run the build commands inside a container.
- A Kubernetes endpoint so that Code Stream can deploy your image to a Kubernetes cluster.
- A Builder image that creates the container on which the continuous integration tests run.
- An Image Registry endpoint so that the Docker build host can pull the builder image from it.

You'll need access to a project. The project groups all your work, including your pipeline, endpoints, and dashboards. Verify that you are a member of a project in Code Stream. If you are not, ask a Code Stream administrator to add you as a member of a project. See How do I add a project in Code Stream.

You'll need a Git webhook that enables Code Stream to use the Git trigger to trigger your pipeline when developers commit code changes. See How do I use the Git trigger in Code Stream to run a pipeline.

Your build toolsets:

- Your build type, such as Maven.
- All the post-process build tools that you use, such as JUnit, JaCoCo, Checkstyle, and FindBugs.

Your publishing tool:

- A tool such as Docker that will deploy your build container.

- An image tag, which is either the commit ID or the build number.

Your build workspace:

- A Docker build host, which is the Docker endpoint.
- An Image Registry. The CI part of the pipeline pulls the image from the selected registry endpoint. The container runs the CI tasks, and deploys your image. If the registry needs credentials, you must create an Image Registry endpoint, then select it here so that the host can pull the image from the registry.
- URL for the builder image that creates the container on which the continuous integration tasks run.

Planning the Continuous Delivery (CD) stage

To plan the CD stage of your pipeline, you set up the external and internal requirements, and determine the information to enter in the CD portion of the smart pipeline template.

Endpoints that you'll need:

- A Kubernetes endpoint so that Code Stream can deploy your image to a Kubernetes cluster.

Environment types and files:

- All the environment types where Code Stream will deploy your application, such as Dev and Prod. The smart pipeline template creates the stages and tasks in your pipeline based on the environment types you select.

  Table 4-2. Pipeline stages that the CICD smart pipeline template creates

Pipeline content	What it does
Build-Publish stage	Builds and tests your code, creates the builder image, and publishes the image to your Docker host.
Development stage	Uses a development Amazon Web Services (AWS) cluster to create and deploy your image. In this stage, you can create a namespace on the cluster, and create a secret key.
Production stage	Uses a production version of the VMware Tanzu Kubernetes Grid Integrated Edition (formerly known as VMware Enterprise PKS) to deploy your image to a production Kubernetes cluster.

- A Kubernetes YAML file that you select in the CD section of the CICD smart pipeline template.

  The Kubernetes YAML file includes three required sections for Namespace, Service, and Deployment and one optional section for Secret. If you plan to create a pipeline by downloading an image from a privately-owned repository, you must include a section with the Docker config Secret. If the pipeline you create only uses publicly available images, no secret is required. The following sample YAML file includes four sections.

  ```
  apiVersion: v1
  kind: Namespace
  metadata:
  ```

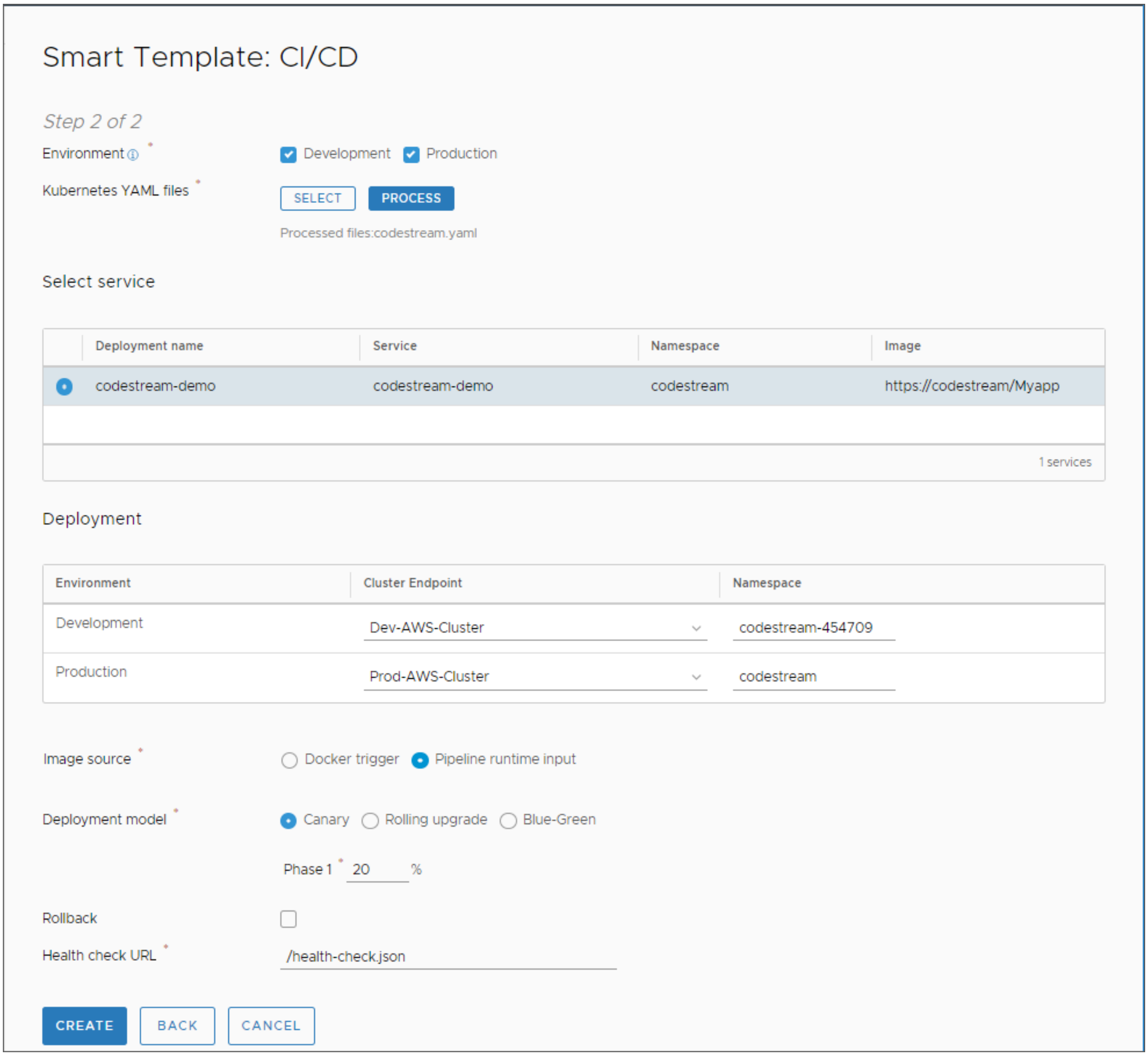
```
  name: codestream
  namespace: codestream
---
apiVersion: v1
data:
  .dockerconfigjson:
eyJhdXRocyI6eyJodHRwczovL2luZ1234567890lci5pby92MS8iOnsidXNlcm5hbWUiOiJhdXRvbWF0aW9uYmV0YSI
sInBhc3N3b3JkIjoiVk13YXJlQDEyMyIsImVtYWlsIjoiYXV0b21hdGlvbmJldGF1c2VyQGdtYWlsLmNvbSIsImF1dG
giOiJZWFYwYjIxaGRHbHZibUpsZEdFNlZrMTNZWEpsUURFeU13PT0ifX19
kind: Secret
metadata:
  name: dockerhub-secret
  namespace: codestream
type: kubernetes.io/dockerconfigjson
---
apiVersion: v1
kind: Service
metadata:
  name: codestream-demo
  namespace: codestream
  labels:
    app: codestream-demo
spec:
  ports:
    - port: 80
  selector:
    app: codestream-demo
    tier: frontend
  type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: codestream-demo
  namespace: codestream
  labels:
    app: codestream-demo
spec:
  replicas: 10
  selector:
    matchLabels:
      app: codestream-demo
      tier: frontend
  template:
    metadata:
      labels:
        app: codestream-demo
        tier: frontend
    spec:
      containers:
      - name: codestream-demo
        image: automationbeta/codestream-demo:01
        ports:
```

```
        - containerPort: 80
          name: codestream-demo
      imagePullSecrets:
      - name: dockerhub-secret
```

Note The Kubernetes YAML file is also used in the CD smart pipeline template, such as in the following use case examples:

- How do I deploy my application in Code Stream to my Blue-Green deployment
- How do I roll back my deployment in Code Stream
- How do I use the Docker trigger in Code Stream to run a continuous delivery pipeline

To apply the file in the Smart Template, click **Select** and select the Kubernetes YAML file. Then click **Process**. The smart pipeline template displays the available services and deployment environments. You select a service, the cluster endpoint, and the deployment strategy. For example, to use the Canary deployment model, select **Canary** and enter a percentage for the deployment phase.



Smart Template: CI/CD

Step 2 of 2

Environment * ☑ Development ☑ Production

Kubernetes YAML files *

Processed files:codestream.yaml

Select service

	Deployment name	Service	Namespace	Image
●	codestream-demo	codestream-demo	codestream	https://codestream/Myapp

1 services

Deployment

Environment	Cluster Endpoint	Namespace
Development	Dev-AWS-Cluster	codestream-454709
Production	Prod-AWS-Cluster	codestream

Image source * ○ Docker trigger ● Pipeline runtime input

Deployment model * ● Canary ○ Rolling upgrade ○ Blue-Green

Phase 1 * 20 %

Rollback □

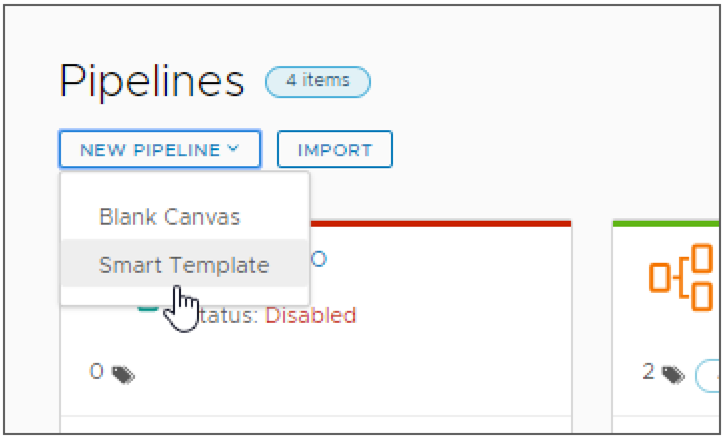
Health check URL * /health-check.json

To see an example of using the smart pipeline template to create a pipeline for a Blue-Green deployment, see How do I deploy my application in Code Stream to my Blue-Green deployment.

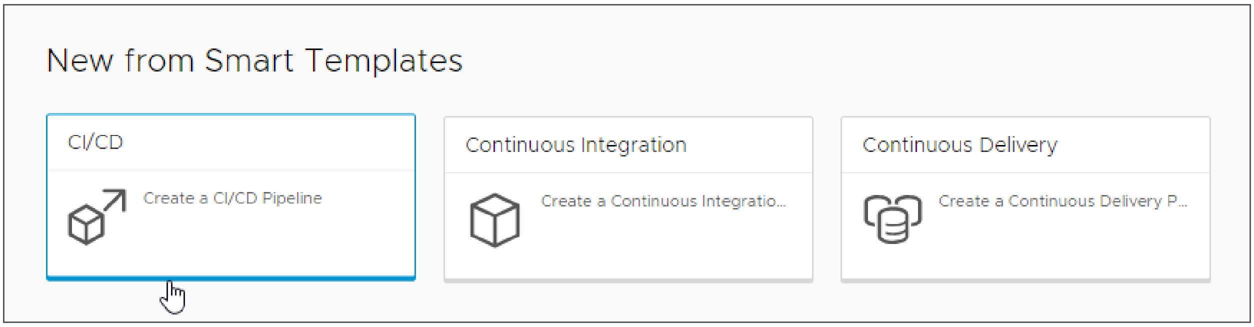
How you'll create the CICD pipeline by using the smart pipeline template

After you gather all the information and set up what you need, here's how you'll create a pipeline from the CICD smart pipeline template.

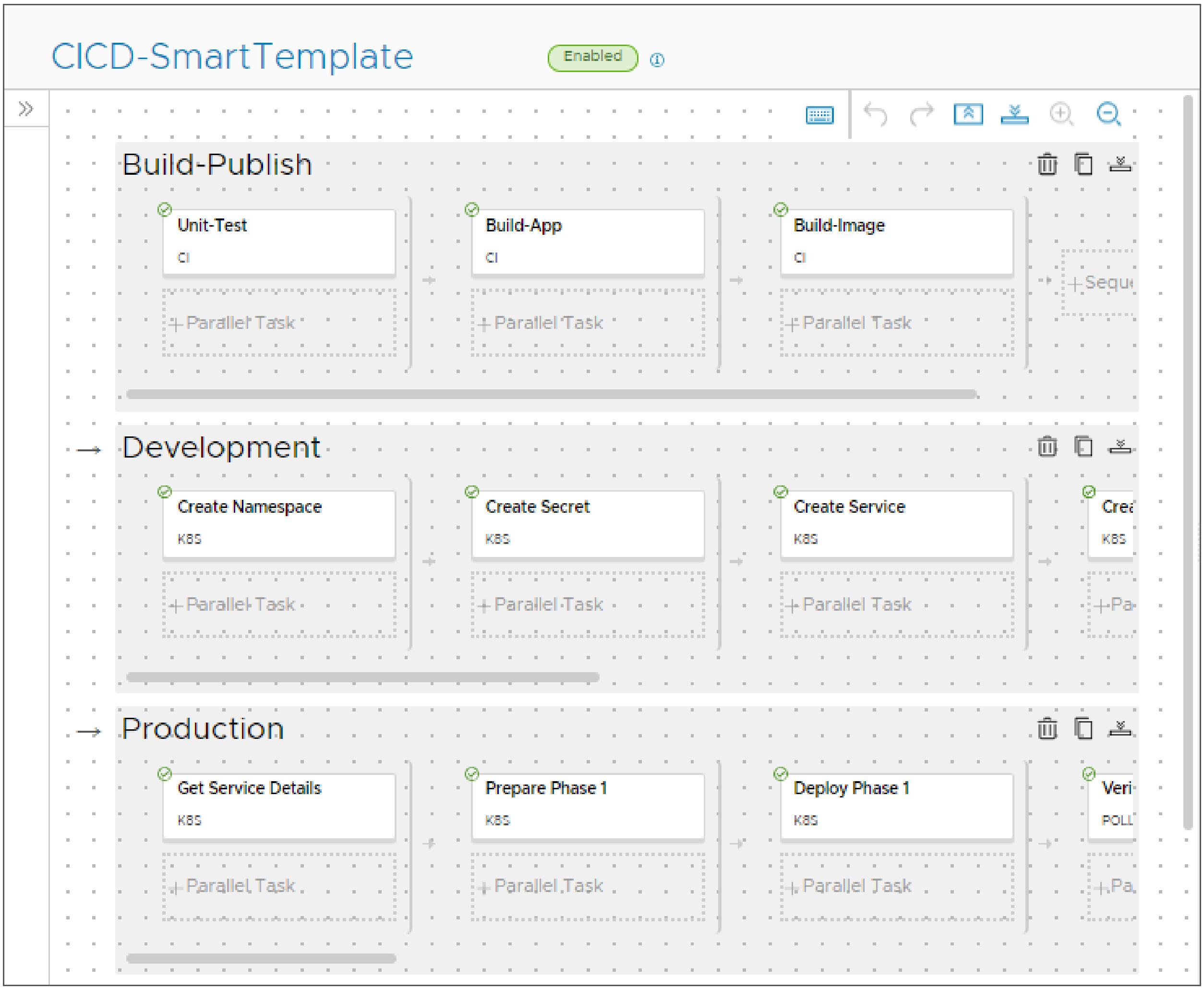
In Pipelines, you'll select **New Pipeline > Smart Templates**.



You'll select the CICD smart pipeline template.



You will fill out the template, and save the pipeline with the stages that it creates. If you need to make any final changes, you can edit the pipeline and save it.



Then, you will enable the pipeline and run it. After it runs, here are some things that you can look for:

- Verify that your pipeline succeeded. Click **Executions**, and search for your pipeline. If it failed, correct any errors and run it again.
- Verify that the Git webhook is operating correctly. The Git **Activity** tab displays the events. Click **Triggers > Git > Activity**.
- Look at the pipeline dashboard and examine the trends. Click **Dashboards**, and search for your pipeline dashboard. You can also create a custom dashboard to report on additional KPIs.

For a detailed example, see How do I continuously integrate code from my GitHub or GitLab repository into my pipeline in Code Stream.

Planning a continuous integration native build in Code Stream before using the smart pipeline template

To create a continuous integration (CI) pipeline in VMware Code Stream, you can use the continuous integration smart pipeline template. To plan your continuous integration native build, you gather the information for the smart pipeline template before you create the pipeline in this example plan.

When you fill out the smart pipeline template, it creates a continuous integration pipeline in your repository, and performs the actions so that the pipeline can run. After your pipeline runs, you can monitor trends across the pipeline executions.

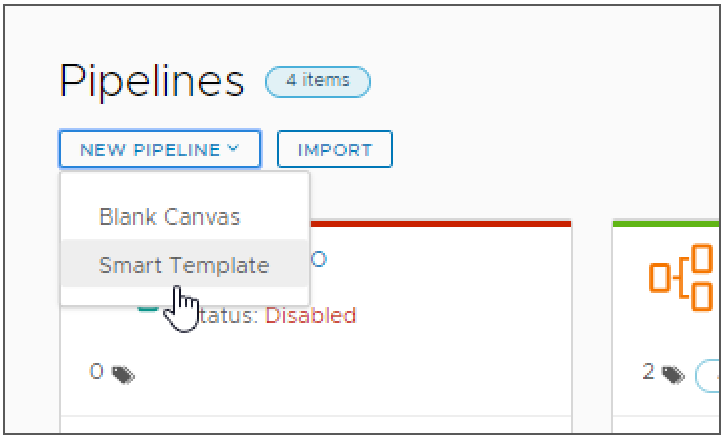
To plan your build before you use the continuous integration smart pipeline template:

- Identify a project that will group all your work, including your pipeline, endpoints, and dashboards.
- Gather the information for your build as described in the continuous delivery portion of Planning a CICD native build in Code Stream before using the smart pipeline template.

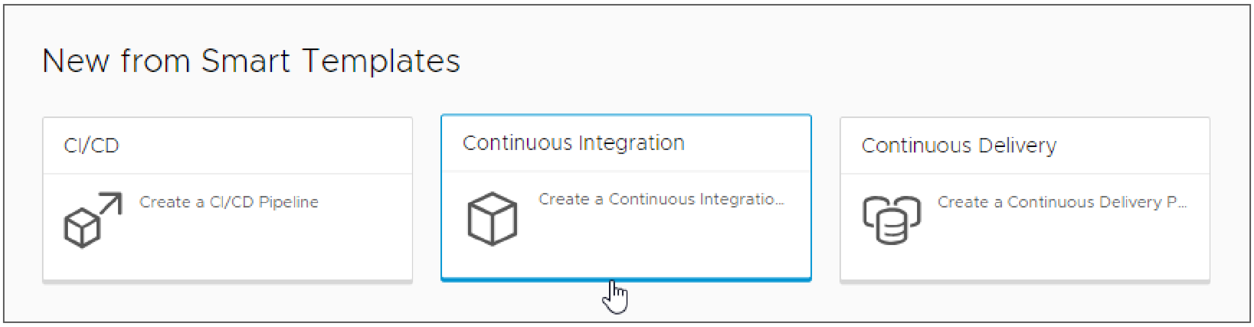
  For example, add a Kubernetes endpoint where Code Stream will deploy the container.

Then, you create a pipeline by using the continuous integration smart pipeline template.

In Pipelines, you select **Smart Templates**.



You select the continuous integration smart pipeline template.



To save the pipeline with the stages that it creates, you fill out the template, and enter a name for the pipeline. To save the pipeline with the stages that it creates, click **Create**.

The Code Stream pipeline workspace supports Docker and Kubernetes for continuous integration tasks and custom tasks.

For information about configuring the workspace, see Configuring the Pipeline Workspace.

To make any final changes, you can edit the pipeline. Then, you can enable the pipeline and run it. After the pipeline runs:

- Verify that your pipeline succeeded. Click **Executions**, and search for your pipeline. If it failed, correct any errors and run it again.
- Verify that the Git webhook is operating correctly. The Git **Activity** tab displays the events. Click **Triggers > Git > Activity**.
- Look at the pipeline dashboard and examine the trends. Click **Dashboards**, and search for your pipeline dashboard. To report on more key performance indicators, you can create a custom dashboard.

For a detailed example, see How do I continuously integrate code from my GitHub or GitLab repository into my pipeline in Code Stream.

Planning a continuous delivery native build in Code Stream before using the smart pipeline template

To create a continuous delivery (CD) pipeline in Code Stream, you can use the continuous delivery smart pipeline template. To plan your continuous delivery native build, you gather the information for the smart pipeline template before you create the pipeline in this example plan.

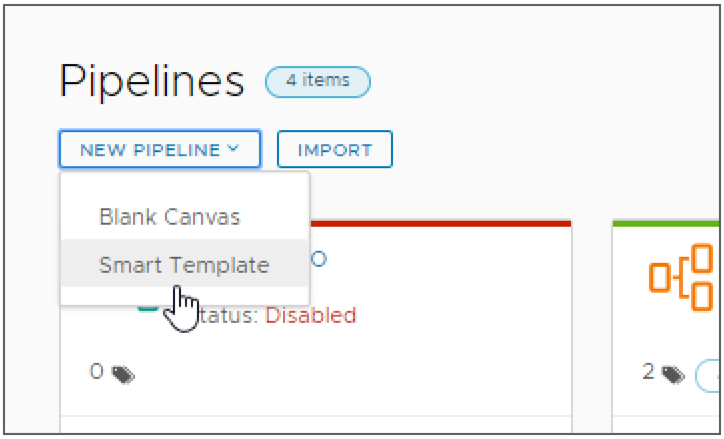
When you fill out the smart pipeline template, it creates a continuous delivery pipeline in your repository, and performs the actions so that the pipeline can run. After your pipeline runs, you can monitor trends across the pipeline executions.

To plan your build before you use the continuous delivery smart pipeline template:

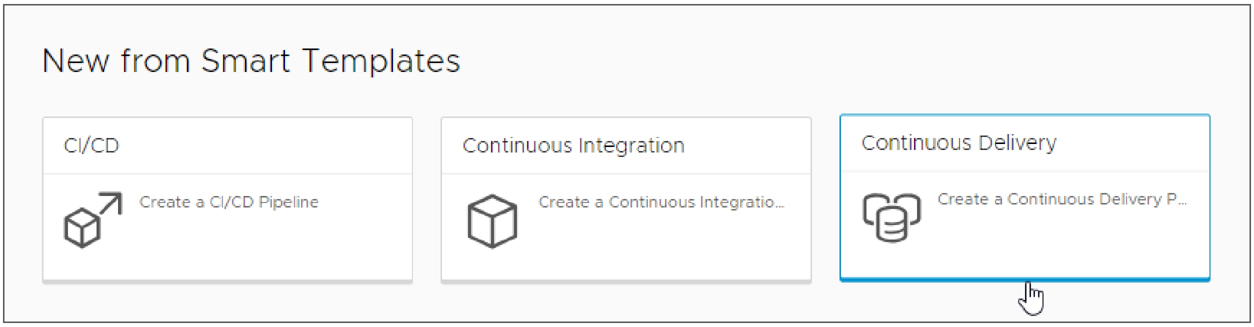
- Identify a project that will group all your work, including your pipeline, endpoints, and dashboards.
- Gather the information for your build as described in the continuous delivery portion of Planning a CICD native build in Code Stream before using the smart pipeline template. For example:
  - Add a Kubernetes endpoint where Code Stream will deploy the container.
  - Prepare the Kubernetes YAML file that creates the Namespace, Service, and Deployment. To download an image from a privately-owned repository, the YAML file must include a section with the Docker config Secret.

Then, you create a pipeline by using the continuous delivery smart pipeline template.

In Pipelines, you select **Smart Templates**.



You select the continuous delivery smart pipeline template.



You fill out the template, and enter a name for the pipeline. To save the pipeline with the stages that it creates, click **Create**.

The Code Stream pipeline workspace supports Docker and Kubernetes for continuous integration tasks and custom tasks.

For information about configuring the workspace, see Configuring the Pipeline Workspace.

To make any final changes, you can edit the pipeline. Then, you can enable the pipeline and run it. After the pipeline runs:

- Verify that your pipeline succeeded. Click **Executions**, and search for your pipeline. If it failed, correct any errors and run it again.
- Verify that the Git webhook is operating correctly. The Git **Activity** tab displays the events. Click **Triggers > Git > Activity**.
- Look at the pipeline dashboard and examine the trends. Click **Dashboards**, and search for your pipeline dashboard. To report on more key performance indicators, you can create a custom dashboard.

For a detailed example, see How do I continuously integrate code from my GitHub or GitLab repository into my pipeline in Code Stream.

Planning a CICD native build in Code Stream before manually adding tasks

To create a continuous integration and continuous delivery (CICD) pipeline in Code Stream, you can manually add stages and tasks. To plan your CICD native build, you'll gather the information you need, then create a pipeline and manually add stages and tasks to it.

You must plan for both the continuous integration (CI) and continuous delivery (CD) stages of your pipeline. After you create your pipeline and run it, you can monitor trends across the pipeline executions.

When a pipeline includes an image from Docker Hub, you must ensure that the image has `cURL` or `wget` embedded before you run the pipeline. When the pipeline runs, Code Stream downloads a binary file that uses `cURL` or `wget` to run commands.

The Code Stream pipeline workspace supports Docker and Kubernetes for continuous integration tasks and custom tasks.

For information about configuring the workspace, see Configuring the Pipeline Workspace.

Planning the external and internal requirements

To plan the CI and CD stages of your pipeline, the following requirements indicate what you must do before you create your pipeline.

This example uses a Docker workspace.

To create a pipeline from this example plan, you will use a Docker host, a Git repository, Maven, and several post-process build tools.

Endpoints and repositories that you'll need:

- A Git source code repository where developers check in their code. Code Stream pulls the latest code into the pipeline when developers commit changes.
- A Docker endpoint for the Docker build host that will run the build commands inside a container.
- A Builder image that creates the container on which the continuous integration tests run.
- An Image Registry endpoint so that the Docker build host can pull the builder image from it.

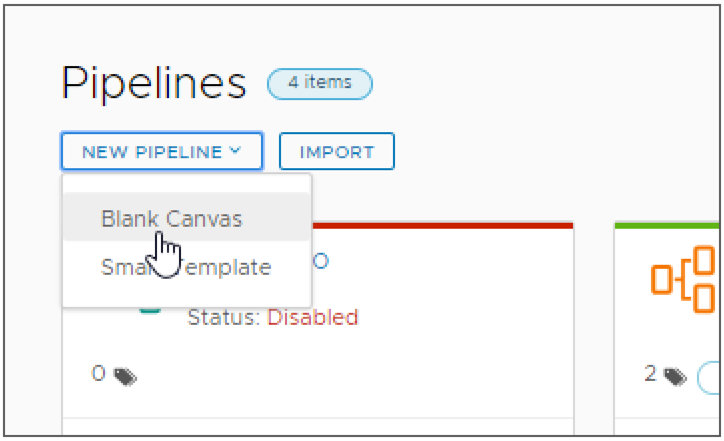
You'll need access to a project. The project groups all your work, including your pipeline, endpoints, and dashboards. Verify that you are a member of a project in Code Stream. If you are not, ask a Code Stream administrator to add you as a member of a project. See How do I add a project in Code Stream.

You'll need a Git webhook that enables Code Stream to use the Git trigger to trigger your pipeline when developers commit code changes. See How do I use the Git trigger in Code Stream to run a pipeline.

How you'll create the CICD pipeline and configure the workspace

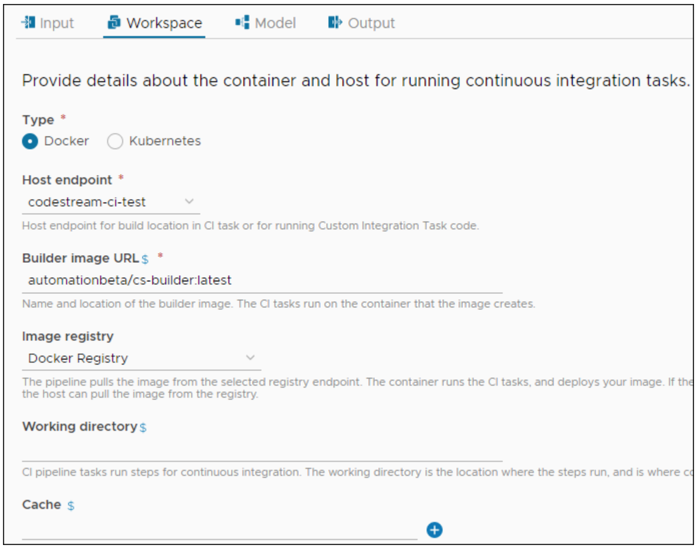
You'll need to create the pipeline, then configure the workspace, pipeline input parameters, and tasks.

To create the pipeline, you'll click **Pipelines > New Pipeline > Blank Canvas**.



On the Workspace tab, enter the continuous integration information:

- Include your Docker build host.
- Enter the URL for your builder image.
- Select the image registry endpoint so that the pipeline can pull the image from it. The container runs the CI tasks and deploys your image. If the registry needs credentials, you must first create the Image Registry endpoint, then select it here so that the host can pull the image from the registry.
- Add the artifacts that must be cached. For a build to succeed, artifacts such as directories are downloaded as dependencies. The cache is the location where these artifacts reside. For example, dependent artifacts can include the `.m2` directory for Maven, and the `node_modules` directory for Node.js. These directories are cached across pipeline executions to save time during builds.



Provide details about the container and host for running continuous integration tasks.

Type *
● Docker ○ Kubernetes

Host endpoint *
codestream-ci-test
Host endpoint for build location in CI task or for running Custom Integration Task code.

Builder image URL $ *
automationbeta/cs-builder:latest
Name and location of the builder image. The CI tasks run on the container that the image creates.

Image registry
Docker Registry
The pipeline pulls the image from the selected registry endpoint. The container runs the CI tasks, and deploys your image. If the
the host can pull the image from the registry.

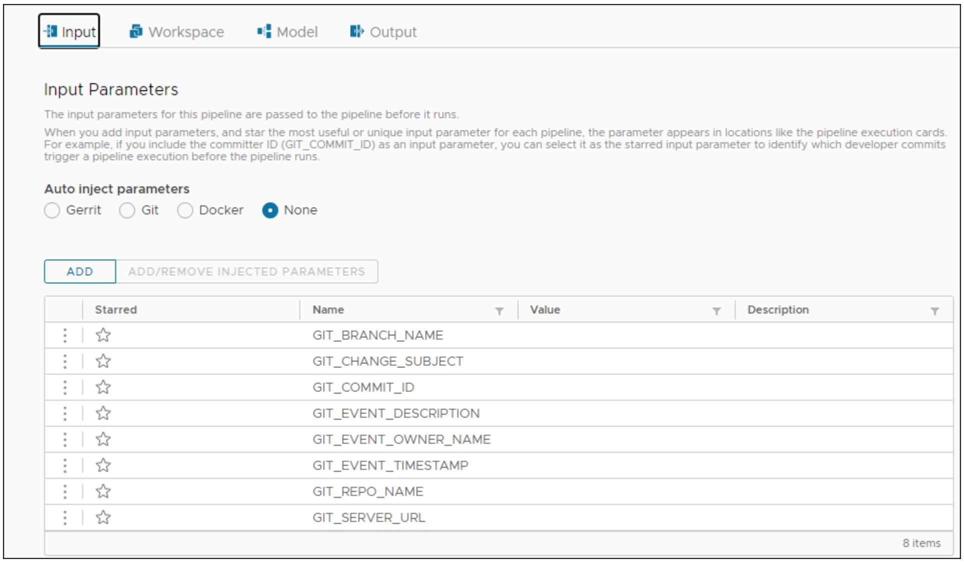
Working directory $

CI pipeline tasks run steps for continuous integration. The working directory is the location where the steps run, and is where c

Cache $

On the Input tab, configure the pipeline input parameters.

- If your pipeline will use input parameters from a Git, Gerrit, or Docker trigger event, select the trigger type for Auto inject parameters. Events can include Change Subject for Gerrit or Git, or Event Owner Name for Docker. If your pipeline will not use any input parameters passed from the event, leave Auto inject parameters set to **None**.
- To apply a value and description to a pipeline input parameter, click the three vertical dots, and click **Edit**. The value you enter is used as input to tasks, stages, or notifications.
- To add a pipeline input parameter, click **Add**. For example, you might add `approvers` to display a default value for every execution, but which you can override with a different approver at runtime.
- To add or remove an injected parameter, click **Add/Remove Injected Parameter**. For example, remove an unused parameter to reduce clutter on the results page and only display the input parameters that are used.



Input Parameters

The input parameters for this pipeline are passed to the pipeline before it runs.

When you add input parameters, and star the most useful or unique input parameter for each pipeline, the parameter appears in locations like the pipeline execution cards. For example, if you include the committer ID (GIT_COMMIT_ID) as an input parameter, you can select it as the starred input parameter to identify which developer commits trigger a pipeline execution before the pipeline runs.

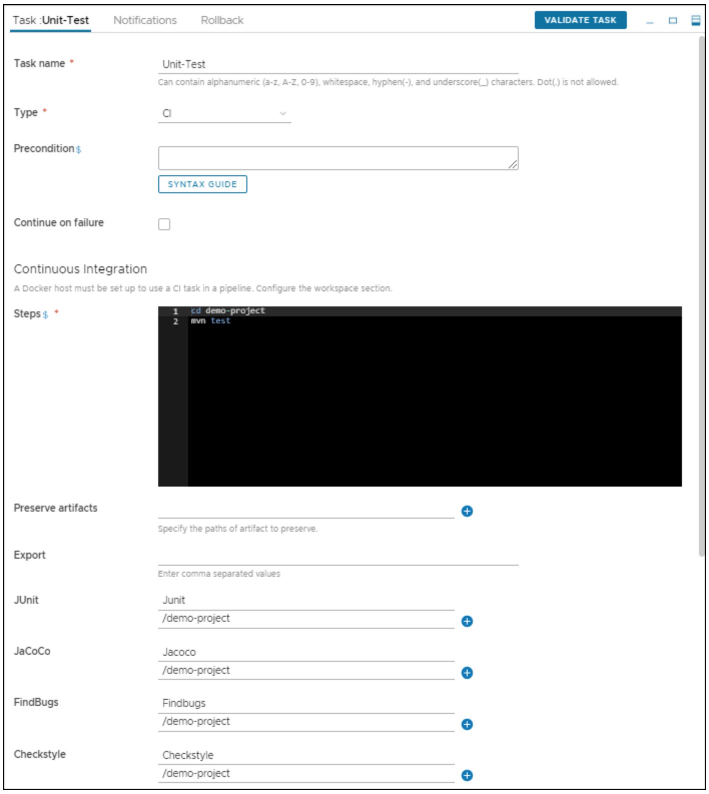
Auto inject parameters
○ Gerrit ○ Git ○ Docker ● None

	Starred	Name	Value	Description
⋮	☆	GIT_BRANCH_NAME		
⋮	☆	GIT_CHANGE_SUBJECT		
⋮	☆	GIT_COMMIT_ID		
⋮	☆	GIT_EVENT_DESCRIPTION		
⋮	☆	GIT_EVENT_OWNER_NAME		
⋮	☆	GIT_EVENT_TIMESTAMP		
⋮	☆	GIT_REPO_NAME		
⋮	☆	GIT_SERVER_URL		

8 items

Configure the pipeline to test your code:

- Add and configure a CI task.
- Include steps to run `mvn test` on your code.
- To identify any problems after the task runs, run post-process build tools, such as JUnit and JaCoCo, FindBugs, and Checkstyle.



Task name * Unit-Test

Can contain alphanumeric (a-z, A-Z, 0-9), whitespace, hyphen(-), and underscore(_) characters. Dot(.) is not allowed.

Type * CI

Precondition $

Continue on failure □

Continuous Integration

A Docker host must be set up to use a CI task in a pipeline. Configure the workspace section.

Steps $ *

```
cd demo-project
mvn test
```

Preserve artifacts

Specify the paths of artifact to preserve.

Export

Enter comma separated values

JUnit: Junit, /demo-project

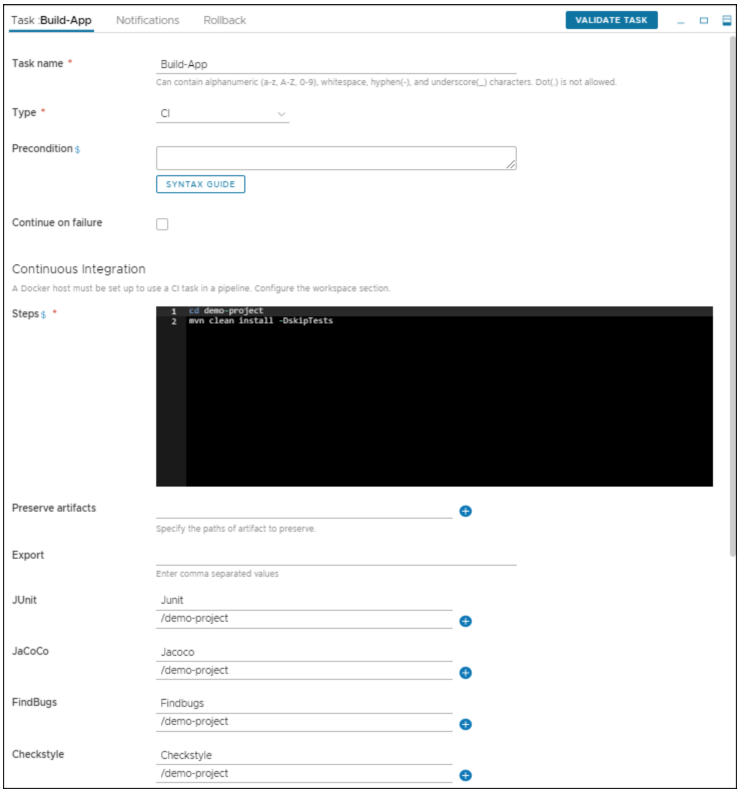
JaCoCo: Jacoco, /demo-project

FindBugs: Findbugs, /demo-project

Checkstyle: Checkstyle, /demo-project

Configure the pipeline to build your code:

- Add and configure a CI task.
- Include steps that run `mvn clean install` on your code.
- Include the location and the JAR filename so that it preserves your artifact.



Task name * Build-App

Can contain alphanumeric (a-z, A-Z, 0-9), whitespace, hyphen(-), and underscore(_) characters. Dot(.) is not allowed.

Type * CI

Precondition $

Continue on failure □

Continuous Integration

A Docker host must be set up to use a CI task in a pipeline. Configure the workspace section.

Steps $ *

```
cd demo-project
mvn clean install -DskipTests
```

Preserve artifacts

Specify the paths of artifact to preserve.

Export

Enter comma separated values

JUnit Junit /demo-project

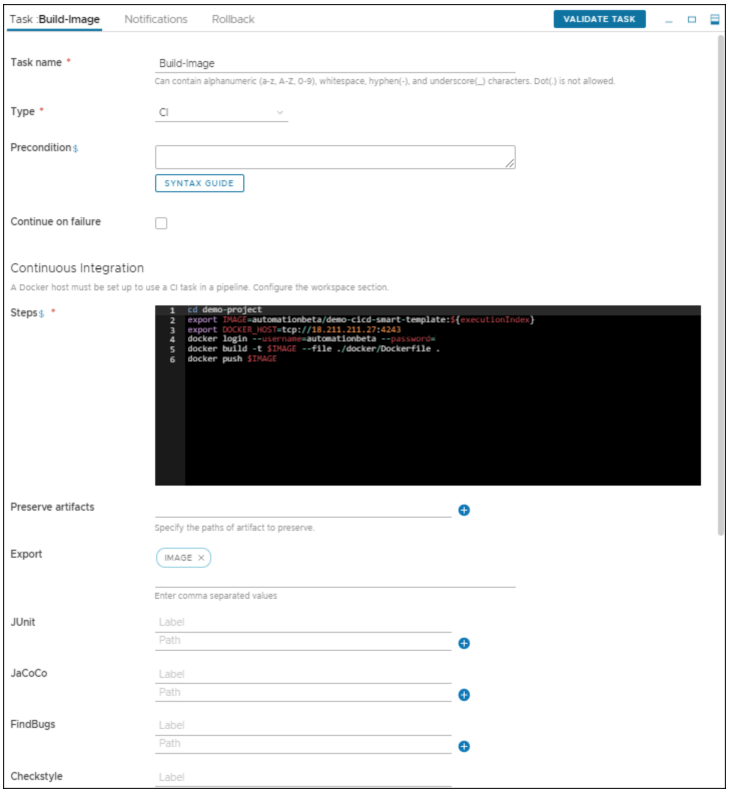
JaCoCo Jacoco /demo-project

FindBugs Findbugs /demo-project

Checkstyle Checkstyle /demo-project

Configure the pipeline to publish your image to your Docker host:

- Add and configure a CI task.
- Add steps that will commit, export, build, and push your image.
- Add the export key of `IMAGE` for the next task to consume.



Task name * Build-Image
Can contain alphanumeric (a-z, A-Z, 0-9), whitespace, hyphen(-), and underscore(_) characters. Dot(.) is not allowed.

Type * CI

Precondition $

Continue on failure ☐

Continuous Integration

A Docker host must be set up to use a CI task in a pipeline. Configure the workspace section.

Steps $ *

```
cd demo-project
export IMAGE=automationbeta/demo-cicd-smart-template:${executionIndex}
export DOCKER_HOST=tcp://18.211.211.27:4243
docker login --username=automationbeta --password=
docker build -t $IMAGE --file ./docker/Dockerfile .
docker push $IMAGE
```

Preserve artifacts
Specify the paths of artifact to preserve.

Export IMAGE ×
Enter comma separated values

JUnit Label Path

JaCoCo Label Path

FindBugs Label Path

Checkstyle Label

After you configure the workspace, input parameters, test tasks, and build tasks, save your pipeline.

How to enable and run your pipeline

After you configure your pipeline with stages and tasks, you can save and enable the pipeline.

Then, wait for the pipeline to run and finish, then verify that it succeeded. If it failed, correct any errors and run it again.

After the pipeline succeeds, here are some things you might want to confirm:

- Examine the pipeline execution and view the results of the task steps.
- In the workspace of the pipeline execution, locate the details about your container and the cloned Git repository.
- In the workspace, look at the results of your post-process tools and check for errors, code coverage, bugs, and style issues.
- Confirm that your artifact is preserved. Also confirm that the image was exported with the IMAGE name and value.
- Go to your Docker repository and verify that the pipeline published your container.

For a detailed example that shows how Code Stream continuously integrates your code, see How do I continuously integrate code from my GitHub or GitLab repository into my pipeline in Code Stream.

Planning for rollback in Code Stream

If a pipeline execution fails, you can use rollback to return your environment to a previously stable state. To use rollback, plan a rollback flow and understand how to implement it.

A rollback flow prescribes the steps required to reverse a failure in deployment. The flow takes the form of a rollback pipeline that includes one or more sequential tasks which vary depending on the type of deployment that executed and failed. For example, the deployment and rollback of a traditional application is different from the deployment and rollback of a container application.

To return to a good deployment state, a rollback pipeline typically includes tasks to:

- Clean up states or environments.
- Run a user-specified script to revert changes.
- Deploy a previous revision of a deployment.

To add rollback to an existing deployment pipeline, you attach the rollback pipeline to the tasks or stages in the deployment pipeline that you want to roll back before you run your deployment pipeline.

How do I configure rollback

To configure rollback in your deployment, you need to:

- Create a deployment pipeline.
- Identify potential failure points in the deployment pipeline that will trigger rollback so that you can attach your rollback pipeline. For example, you might attach your rollback pipeline to a condition or poll task type in the deployment pipeline that checks whether a previous task completed successfully. For information on condition tasks, see How do I use variable bindings in a condition task to run or stop a pipeline in Code Stream.
- Determine the scope of failure that will trigger the rollback pipeline such as a task or stage failure. You can also attach rollback to a stage.
- Decide what rollback task or tasks to execute in the event of a failure. You'll create your rollback pipeline with those tasks.

You can manually create a rollback pipeline, or Code Stream can create one for you.

- Using a blank canvas, you can manually create a rollback pipeline that follows a flow in parallel to an existing deployment pipeline. Then you attach the rollback pipeline to one or more tasks in the deployment pipeline that trigger rollback on failure.
- Using a smart pipeline template, you can configure a deployment pipeline with the rollback action. Then, Code Stream automatically creates one or more default rollback pipelines with predefined tasks that roll back the deployment on failure.

For a detailed example on how to configure a CD pipeline with rollback by using a smart pipeline template, see How do I roll back my deployment in Code Stream.

What happens if my deployment pipeline has multiple tasks or stages with rollback

If you have multiple tasks or tasks and stages with rollback added, be aware that the rollback sequence varies.

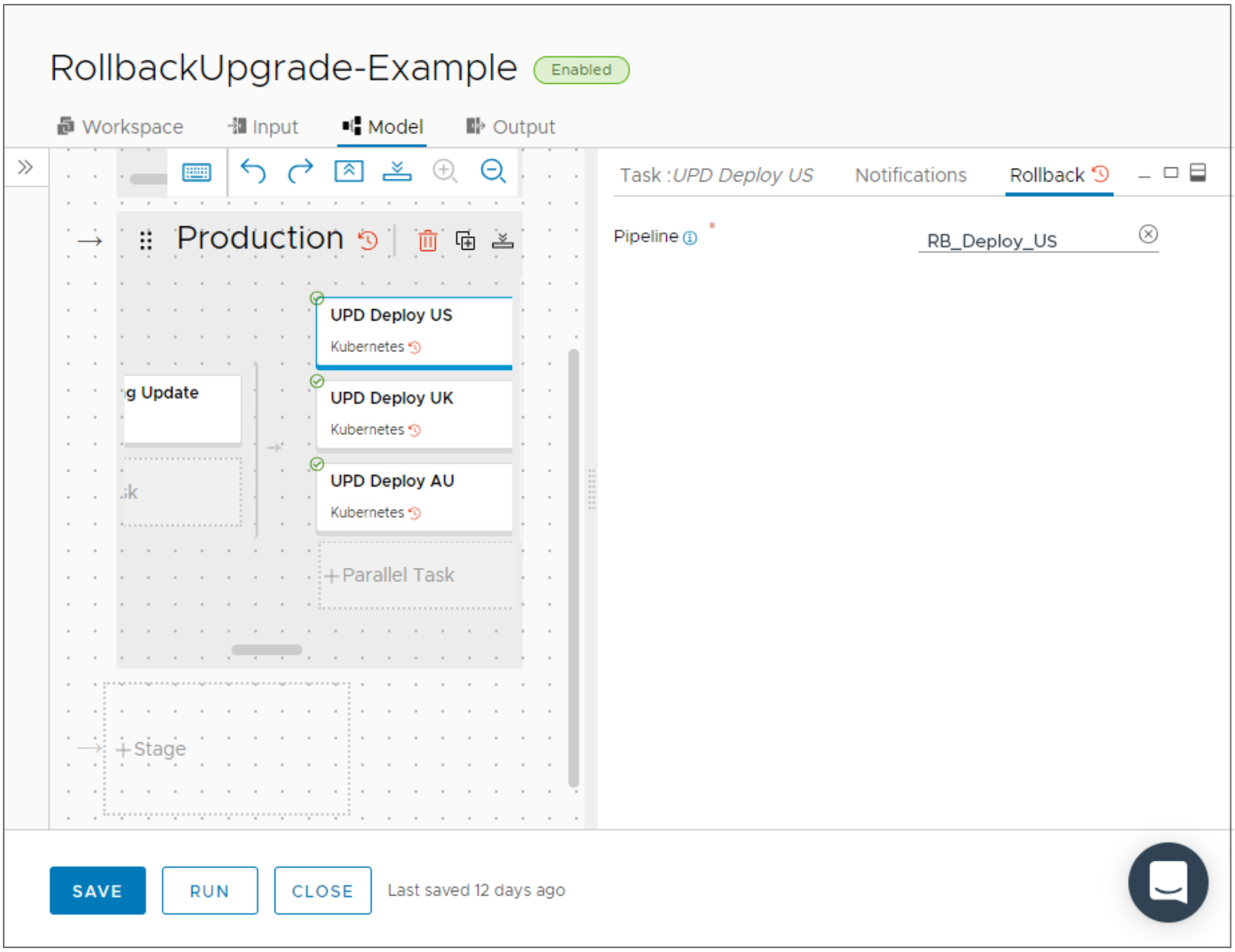
Table 4-3. Determining rollback sequence

If you add rollback to...	When does roll back occur...
Parallel tasks	If one of the parallel tasks fails, roll back for that task occurs after all the parallel tasks have completed or failed. Rollback does not occur immediately after the task fails.
Both the task within a stage, and the stage	If a task fails, the task rollback runs. If the task is in a group of parallel tasks, the task rollback runs after all the parallel tasks have completed or failed. After the task rollback completes or fails to complete, the stage rollback runs.

Consider a pipeline that has:

- A production stage with rollback.
- A group of parallel tasks, each task with its own rollback.

The task named **UPD Deploy US** has the rollback pipeline **RB_Deploy_US**. If **UPD Deploy US** fails, the rollback follows the flow defined in the **RB_Deploy_US** pipeline.



Pipeline * RB_Deploy_US

If **UPD Deploy US** fails, the **RB_Deploy_US** pipeline runs after **UPD Deploy UK** and **UPD Deploy AU** have also completed or failed. Rollback does not occur immediately after **UPD Deploy US** fails. And because the production stage also has rollback, after the **RB_Deploy_US** pipeline runs, the stage rollback pipeline runs.

Tutorials for using Code Stream

5

Code Stream models and supports your DevOps release lifecycle, and continuously tests and releases your applications to development environments and production environments.

You already set up everything you need so that you can use Code Stream. See Chapter 2 Setting up Code Stream to model my release process.

Now, you can create pipelines that automate the build and test of developer code before you release it to production. You can have Code Stream deploy container-based or traditional applications.

Table 5-1. Using Code Stream in your DevOps lifecycle

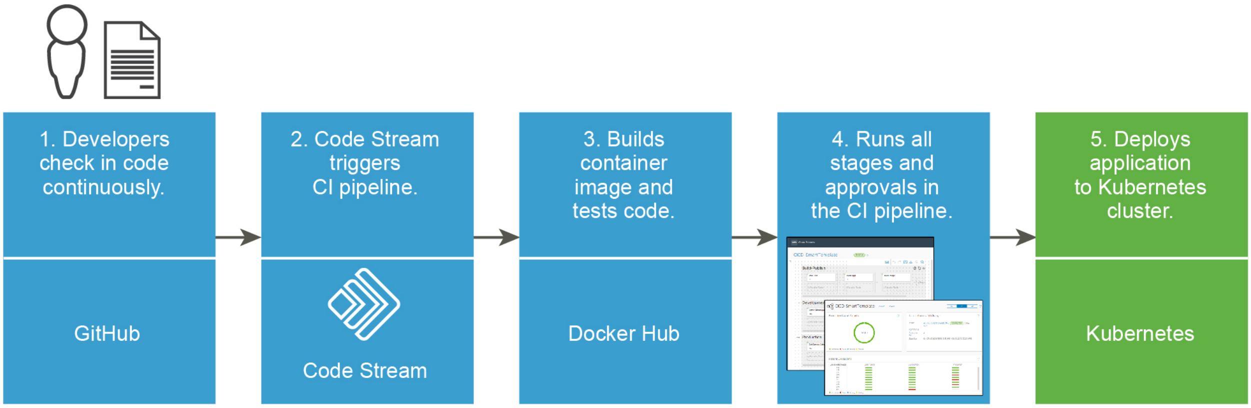
Features	Examples of what you can do
Use the native build capability in Code Stream.	Create Continuous Integration and Delivery (CICD), Continuous Integration (CI), and Continuous Delivery (CD) pipelines that continuously integrate, containerize, and deliver your code. ■ Use a smart pipeline template that creates a pipeline for you. ■ Manually add stages and tasks to a pipeline.
Release your applications and automate releases.	Integrate and release your applications in various ways. ■ Continuously integrate your code from a GitHub or a GitLab repository into your pipeline. ■ Integrate a Docker Host to run Continuous Integration tasks as documented in this blog article Creating a Docker host for vRealize Automation Code Stream. ■ Automate the deployment of your application by using a YAML cloud template. ■ Automate the deployment of your application to a Kubernetes cluster. ■ Release your application to a Blue-Green deployment. ■ Integrate Code Stream with your own build, test, and deploy tools. ■ Use a REST API that integrates Code Stream with other applications.
Track trends, metrics, and key performance indicators (KPIs).	Create custom dashboards and gain insight about the performance of your pipelines.
Resolve problems.	When a pipeline run fails, have Code Stream create a Jira ticket.

This chapter includes the following topics:

- How do I automate the release of an application that I deploy from a YAML cloud template in Code Stream
- How do I automate the release of an application in Code Stream to a Kubernetes cluster
- How do I deploy my application in Code Stream to my Blue-Green deployment
- How do I integrate my own build, test, and deploy tools with Code Stream
- How do I use the resource properties of a cloud template task in my next task
- How do I use a REST API to integrate Code Stream with other applications
- How do I leverage pipeline as code in Code Stream

How do I continuously integrate code from my GitHub or GitLab repository into my pipeline in Code Stream

As a developer, you want to continuously integrate your code from a GitHub repository or a GitLab Enterprise repository. Whenever your developers update their code and commit changes to the repository, Code Stream can listen for those changes, and trigger the pipeline.


1. Developers check in code continuously.
GitHub
2. Code Stream triggers CI pipeline.
Code Stream
3. Builds container image and tests code.
Docker Hub
4. Runs all stages and approvals in the CI pipeline.
5. Deploys application to Kubernetes cluster.
Kubernetes

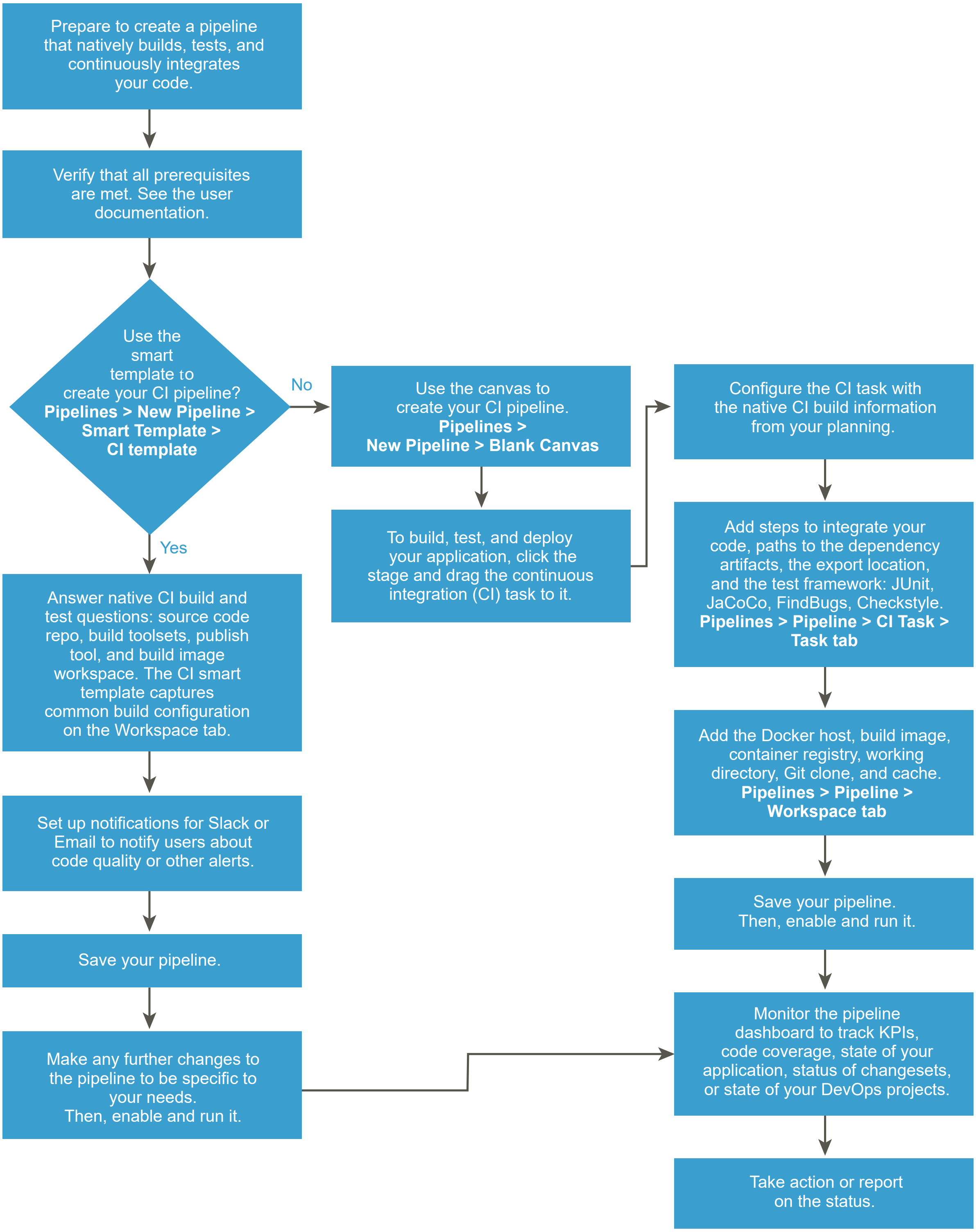
To have Code Stream trigger your pipeline on code changes, you use the Git trigger. Code Stream then triggers your pipeline every time you commit changes to your code.

The Code Stream pipeline workspace supports Docker and Kubernetes for continuous integration tasks and custom tasks.

For more information about configuring the workspace, see Configuring the Pipeline Workspace.

The following flowchart shows the workflow that you can take if you use a smart pipeline template to create your pipeline, or build the pipeline manually.

Figure 5-1. Workflow that uses a smart pipeline template or creates a pipeline manually


Prepare to create a pipeline that natively builds, tests, and continuously integrates your code.
Verify that all prerequisites are met. See the user documentation.
Use the smart template to create your CI pipeline? Pipelines > New Pipeline > Smart Template > CI template
No
Yes
Answer native CI build and test questions: source code repo, build toolsets, publish tool, and build image workspace. The CI smart template captures common build configuration on the Workspace tab.
Set up notifications for Slack or Email to notify users about code quality or other alerts.
Save your pipeline.
Make any further changes to the pipeline to be specific to your needs. Then, enable and run it.
Use the canvas to create your CI pipeline. Pipelines > New Pipeline > Blank Canvas
To build, test, and deploy your application, click the stage and drag the continuous integration (CI) task to it.
Configure the CI task with the native CI build information from your planning.
Add steps to integrate your code, paths to the dependency artifacts, the export location, and the test framework: JUnit, JaCoCo, FindBugs, Checkstyle. Pipelines > Pipeline > CI Task > Task tab
Add the Docker host, build image, container registry, working directory, Git clone, and cache. Pipelines > Pipeline > Workspace tab
Save your pipeline. Then, enable and run it.
Monitor the pipeline dashboard to track KPIs, code coverage, state of your application, status of changesets, or state of your DevOps projects.
Take action or report on the status.

The following example uses a Docker workspace.

To build your code, you use a Docker host. You use JUnit and JaCoCo as your test framework tools, which run unit tests and code coverage, and you include them in your pipeline.

Then you can use the continuous integration smart pipeline template that creates a continuous integration pipeline that builds, tests, and deploys your code to your project team Kubernetes cluster on AWS. To store the code dependency artifacts for your continuous integration task, which can save time in code builds, you can use a cache.

In the pipeline task that builds and tests your code, you can include several continuous integration steps. These steps can reside in the same working directory where Code Stream clones the source code when the pipeline triggers.

To deploy your code to the Kubernetes cluster, you can use a Kubernetes task in your pipeline. You must then enable and run your pipeline. Then, make a change to your code in the repository, and watch the pipeline trigger. To monitor and report on your pipeline trends after your pipeline runs, use the dashboards.

In the following example, to create a continuous integration pipeline that continuously integrates your code into your pipeline, you use the continuous integration smart pipeline template. This example uses a Docker workspace.

Optionally, you can manually create the pipeline, and add stages and tasks to it. For more information about planning a continuous integration build and manually creating the pipeline, see Planning a CICD native build in Code Stream before manually adding tasks.

Prerequisites

- Plan for your continuous integration build. See Planning a continuous integration native build in Code Stream before using the smart pipeline template.
- Verify that a GitLab source code repository exists. For help, see your Code Stream administrator.
- Add a Git endpoint. For an example, see How do I use the Git trigger in Code Stream to run a pipeline.
- To have Code Stream listen for changes in your GitHub repository or your GitLab repository, and trigger a pipeline when changes occur, add a webhook. For an example, see How do I use the Git trigger in Code Stream to run a pipeline.
- Add a Docker host endpoint, which creates a container for the continuous integration task that multiple continuous integration tasks can use. For more information about endpoints, see What are Endpoints in Code Stream .
- Obtain the image URL, the build host, and the URL for the build image. For help, see your Code Stream administrator.
- Verify that you use JUnit and JaCoCo for your test framework tools.
- Set up an external instance for your continuous integration build: Jenkins, TFS, or Bamboo. The Kubernetes plug-in deploys your code. For help, see your Code Stream administrator.

Procedure

1 Follow the prerequisites.

2 To create the pipeline by using the smart pipeline template, open the continuous integration smart pipeline template and fill out the form.

   a Click **Pipelines > New Pipeline > Smart Template > Continuous Integration**.

   b Answer the questions in the template about your source code repository, build toolsets, publishing tool, and the build image workspace.

   c Add Slack notifications or Email notifications for your team.

   d To have the smart pipeline template create the pipeline, click **Create**.

   e To make any further changes to the pipeline, click **Edit**, make your changes, and click **Save**.

   f Enable the pipeline and run it.

3 To create the pipeline manually, add stages and tasks to the canvas, and include your native continuous integration build information in the continuous integration task.

   a Click **Pipelines > New Pipeline > Blank Canvas**.

   b Click the stage, then drag the several continuous integration tasks from the navigation pane to the stage.

   c To configure the continuous integration task, click it, and click the **Task** tab.

   d Add the steps that continuously integrate your code.

   e Include the paths to the dependency artifacts.

   f Add the export location.

   g Add the test framework tools that you'll use.

   h Add the Docker host and build image.

   i Add the container registry, working directory, and cache.

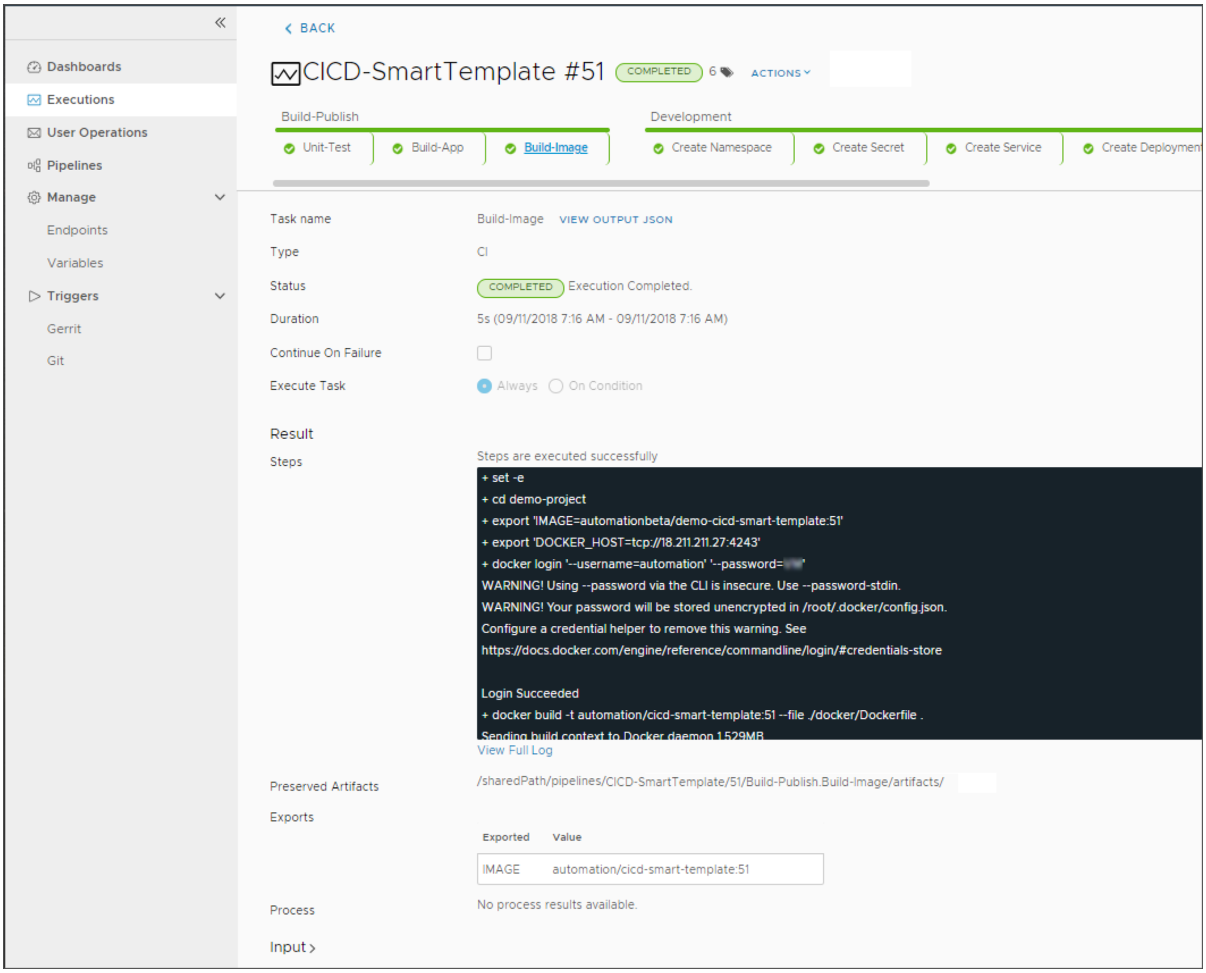
   j Save the pipeline, then enable it.

4 Make a change to your code in your GitHub repository or GitLab repository.

   The Git trigger activates your pipeline, which starts to run.

5 To verify that the code change triggered the pipeline, click **Triggers > Git > Activity**.

6 To view the execution for your pipeline, click **Executions**, and verify that the steps created and exported your build image.



Task name: Build-Image

Type: CI

Status: COMPLETED Execution Completed.

Duration: 5s (09/11/2018 7:16 AM - 09/11/2018 7:16 AM)

Continue On Failure: □

Execute Task: ● Always ○ On Condition

Result

Steps: Steps are executed successfully

```
+ set -e
+ cd demo-project
+ export 'IMAGE=automationbeta/demo-cicd-smart-template:51'
+ export 'DOCKER_HOST=tcp://18.211.211.27:4243'
+ docker login '--username=automation' '--password=[redacted]'
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
+ docker build -t automation/cicd-smart-template:51 --file ./docker/Dockerfile .
Sending build context to Docker daemon 1.529MB
```

Preserved Artifacts: /sharedPath/pipelines/CICD-SmartTemplate/51/Build-Publish.Build-Image/artifacts/

Exports:

Exported	Value
IMAGE	automation/cicd-smart-template:51

Process: No process results available.

7 To monitor the pipeline dashboard so that you can track KPIs and trends, click **Dashboards > Pipeline Dashboards**.

Results

Congratulations! You created a pipeline that continuously integrates your code from a GitHub repository or GitLab repository into your pipeline, and deploys your build image.

What to do next

To learn more, see More resources for Code Stream Administrators and Developers.

How do I automate the release of an application that I deploy from a YAML cloud template in Code Stream

As a developer, you need a pipeline that fetches an automation cloud template from an on-premises GitHub instance every time you commit a change. You need the pipeline to deploy a WordPress application to either Amazon Web Services (AWS) EC2 or a data center. Code

Stream calls the cloud template from the pipeline and automates the continuous integration and continuous delivery (CICD) of that cloud template to deploy your application.

To create and trigger your pipeline, you'll need a VMware Cloud Template.

For **Cloud template source** in your Code Stream cloud template task, you can select either:

- **Cloud Assembly template** as the source control. In this case, you do not need a GitLab or GitHub repository.
- **Source Control** if you use GitLab or GitHub for source control. In this case, you must have a Git webhook and trigger the pipeline through the webhook.

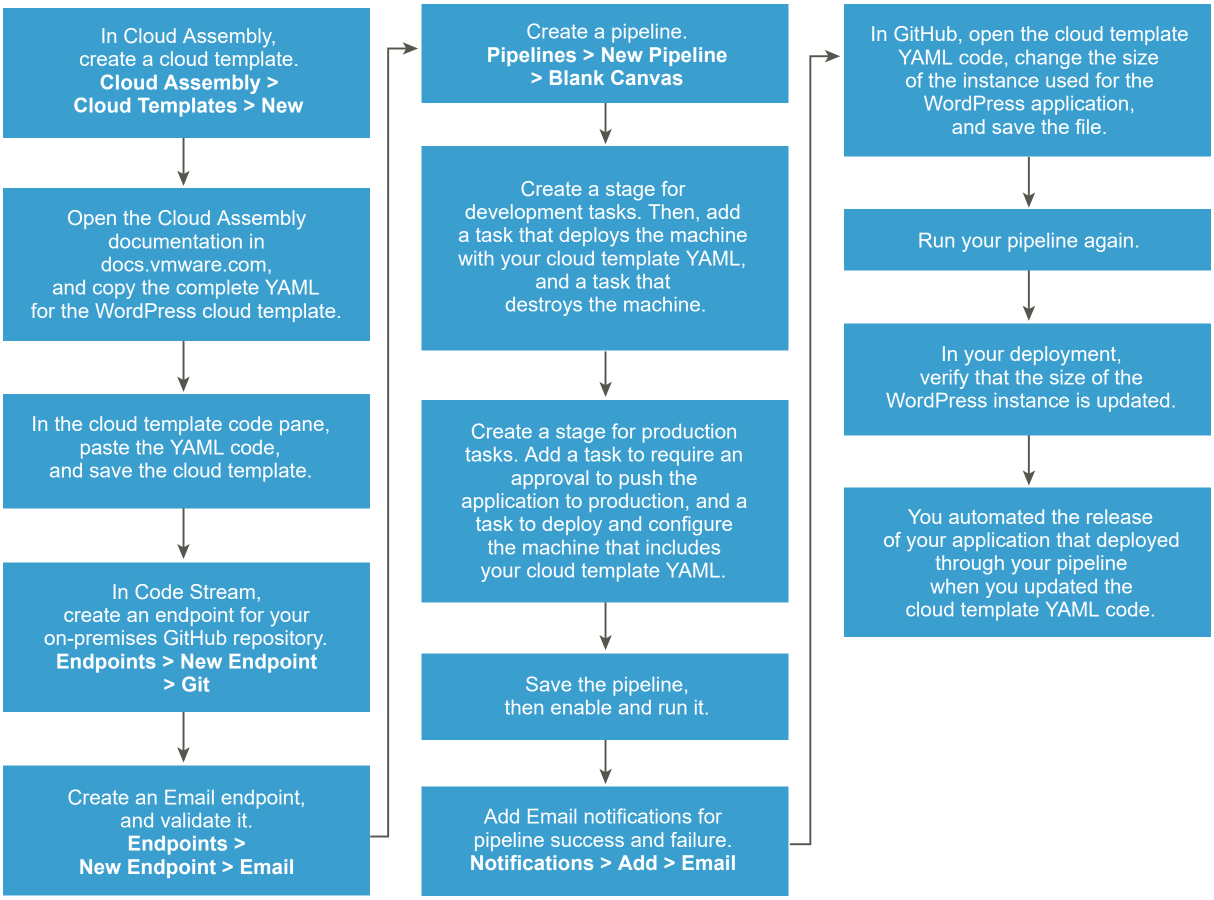
If you have a YAML cloud template in your GitHub repository, and want to use that cloud template in your pipeline, here's what you'll need to do.

1. In Cloud Assembly, push the cloud template to your GitHub repository.
2. In Code Stream, create a Git endpoint. Then, create a Git webhook that uses your Git endpoint and your pipeline.
3. To trigger your pipeline, update any file in your GitHub repository and commit your change.

If you don't have a YAML cloud template in your GitHub repository, and want to use a cloud template from source control, use this procedure to learn how. It shows you how to create a cloud template for a WordPress application, and trigger it from an on-premises GitHub repository. Whenever you make a change to the YAML cloud template, the pipeline triggers and automates the release of your application.

- In Cloud Assembly, you'll add a cloud account, add a cloud zone, and create the cloud template.
- In Code Stream, you'll add an endpoint for the on-premises GitHub repository that hosts your cloud template. Then, you'll add the cloud template to your pipeline.

This use case example shows you how to use a cloud template from an on-premises GitHub repository.


In Cloud Assembly, create a cloud template. **Cloud Assembly > Cloud Templates > New**
Open the Cloud Assembly documentation in docs.vmware.com, and copy the complete YAML for the WordPress cloud template.
In the cloud template code pane, paste the YAML code, and save the cloud template.
In Code Stream, create an endpoint for your on-premises GitHub repository. **Endpoints > New Endpoint > Git**
Create an Email endpoint, and validate it. **Endpoints > New Endpoint > Email**
Create a pipeline. **Pipelines > New Pipeline > Blank Canvas**
Create a stage for development tasks. Then, add a task that deploys the machine with your cloud template YAML, and a task that destroys the machine.
Create a stage for production tasks. Add a task to require an approval to push the application to production, and a task to deploy and configure the machine that includes your cloud template YAML.
Save the pipeline, then enable and run it.
Add Email notifications for pipeline success and failure. **Notifications > Add > Email**
In GitHub, open the cloud template YAML code, change the size of the instance used for the WordPress application, and save the file.
Run your pipeline again.
In your deployment, verify that the size of the WordPress instance is updated.
You automated the release of your application that deployed through your pipeline when you updated the cloud template YAML code.

Prerequisites

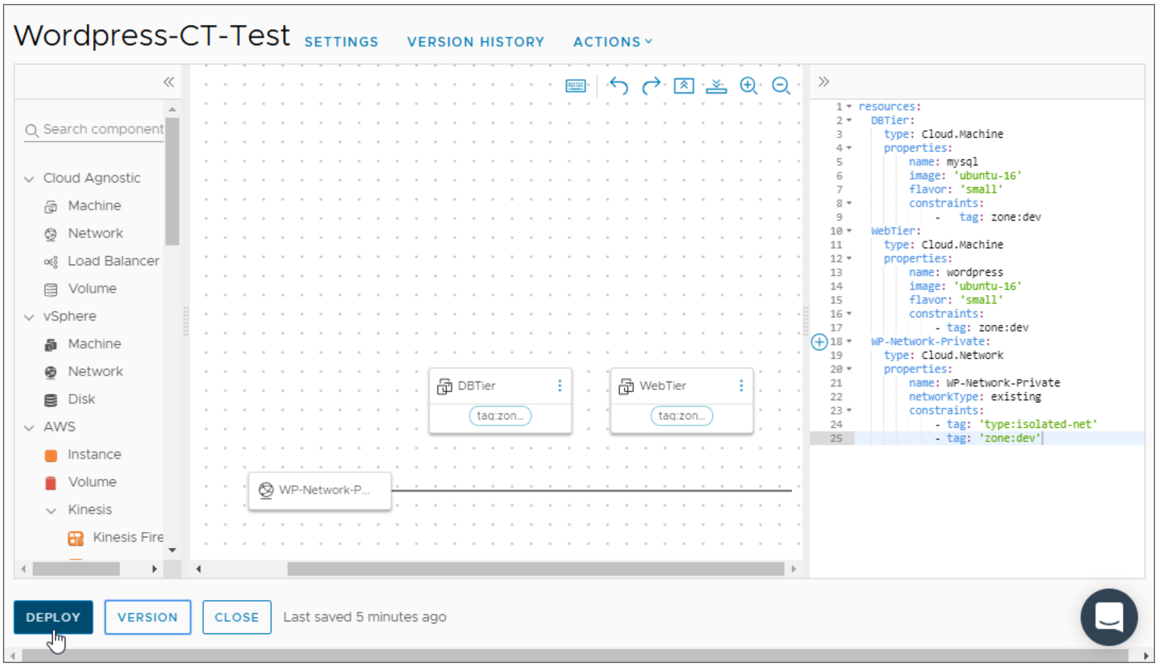
- Add a cloud account and a cloud zone in your vRealize Automation Cloud Assembly infrastructure. See the vRealize Automation Cloud Assembly documentation.
- To create your cloud template in the following procedure, copy the WordPress YAML code to your clipboard. See the cloud template YAML code in the WordPress use case in the vRealize Automation Cloud Assembly documentation.
- Add the YAML code for the WordPress application to your GitHub instance.
- Add a webhook for the Git trigger so that your pipeline can pull your YAML code whenever you commit changes to it. In Code Stream, click **Triggers > Git > Webhooks for Git**.
- To work with a cloud template task, you must have any of the Cloud Assembly roles.

Procedure

1 In Cloud Assembly, follow these steps.

   a Click **VMware Cloud Templates**, then create a cloud template and a deployment for the WordPress application.

   b Paste the WordPress YAML code that you copied to your clipboard into your cloud template, and deploy it.

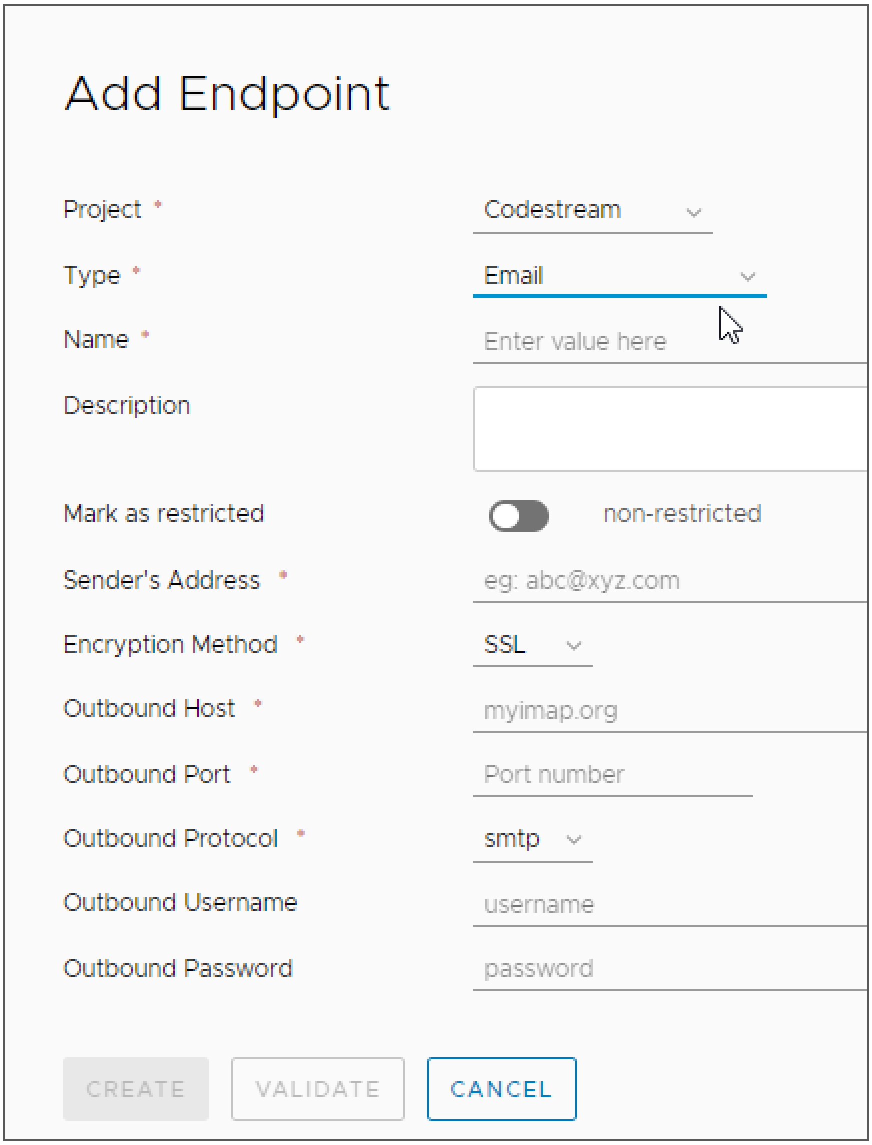


```yaml
resources:
  DBTier:
    type: Cloud.Machine
    properties:
        name: mysql
        image: 'ubuntu-16'
        flavor: 'small'
        constraints:
            -   tag: zone:dev
  WebTier:
    type: Cloud.Machine
    properties:
        name: wordpress
        image: 'ubuntu-16'
        flavor: 'small'
        constraints:
            - tag: zone:dev
  WP-Network-Private:
    type: Cloud.Network
    properties:
        name: WP-Network-Private
        networkType: existing
        constraints:
            - tag: 'type:isolated-net'
            - tag: 'zone:dev'
```

2 In Code Stream, create endpoints.

   a Create a Git endpoint for your on-premises GitHub repository where your YAML file resides.

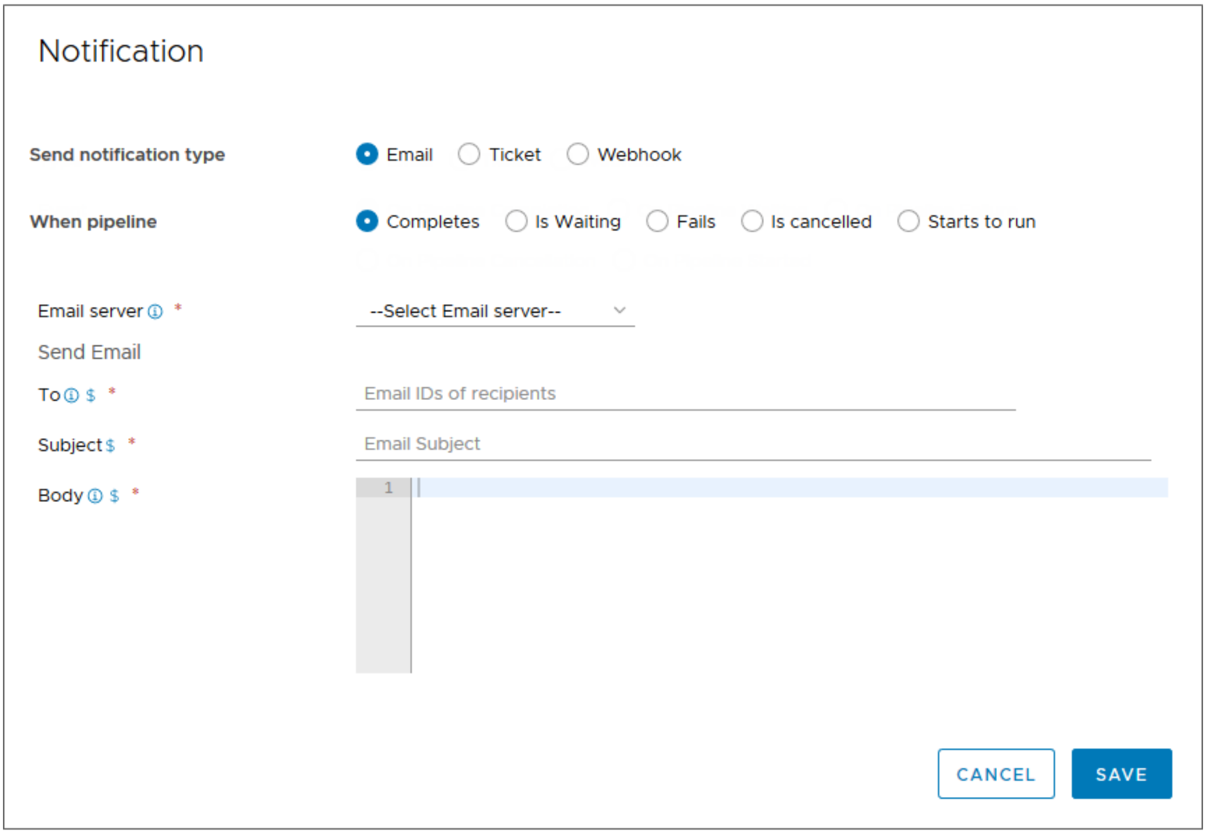
   b Add an Email endpoint that can notify users about the pipeline status when it runs.



Add Endpoint

Project *	Codestream
Type *	Email
Name *	Enter value here
Description	
Mark as restricted	non-restricted
Sender's Address *	eg: abc@xyz.com
Encryption Method *	SSL
Outbound Host *	myimap.org
Outbound Port *	Port number
Outbound Protocol *	smtp
Outbound Username	username
Outbound Password	password

3 Create a pipeline, and add notifications for pipeline success and failure.



Notification

Send notification type ● Email ○ Ticket ○ Webhook

When pipeline ● Completes ○ Is Waiting ○ Fails ○ Is cancelled ○ Starts to run

Email server * --Select Email server--

Send Email

To $ * Email IDs of recipients

Subject $ * Email Subject

Body $ *

4 Add a stage for development, and add a cloud template task.

a Add a cloud template task that deploys the machine, and configure the task to use the cloud template YAML for the WordPress application.

```
resources:
  DBTier:
    type: Cloud.Machine
    properties:
        name: mysql
        image: 'ubuntu-16'
        flavor: 'small'
        constraints:
            -   tag: zone:dev
  WebTier:
    type: Cloud.Machine
    properties:
        name: wordpress
        image: 'ubuntu-16'
        flavor: 'small'
        constraints:
            - tag: zone:dev
 WP-Network-Private:
    type: Cloud.Network
    properties:
        name: WP-Network-Private
        networkType: existing
        constraints:
            - tag: 'type:isolated-net'
            - tag: 'zone:dev'
```

b Add a cloud template task that destroys the machine to free up resources.

5 Add a stage for production, and include approval and deployment tasks.

   a Add a User Operation task to require approval to push the WordPress application to production.

   b Add a cloud template task to deploy the machine and configure it with the cloud template YAML for the WordPress application.

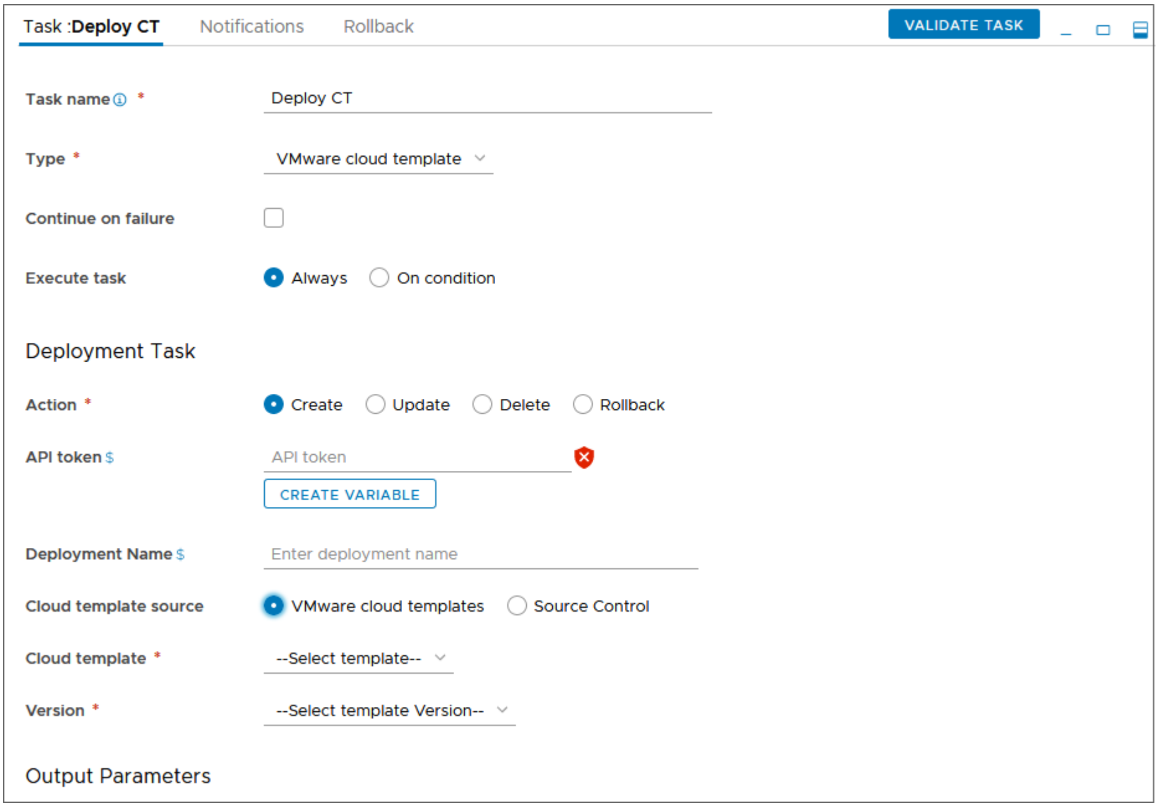
   When you select **Create**, the deployment name must be unique. If you leave the name blank, Code Stream assigns it a unique random name.

   Here's what you must know if you select **Rollback** in your own use case: If you select the **Rollback** action and enter a **Rollback Version**, the version must be in the form of `n-X`. For example, `n-1`, `n-2`, `n-3`, and so on. If you create and update the deployment in any location other than Code Stream, rollback is allowed.

   When you log in to Code Stream, it gets a user token, which is valid for 30 minutes. For long-running pipeline durations, when the task prior to the cloud template task takes 30 minutes or more to run, the user token expires. As a result, the cloud template task fails.

   To ensure that your pipeline can run longer than 30 minutes, you can enter an optional API token. When Code Stream invokes the cloud template, the API token persists and the cloud template task continues to use the API token.

   When you use the API token as a variable, it is encrypted. Otherwise, it is used as plain text.



Task name * Deploy CT

Type * VMware cloud template

Continue on failure □

Execute task ● Always ○ On condition

Deployment Task

Action * ● Create ○ Update ○ Delete ○ Rollback

API token $ API token

Deployment Name $ Enter deployment name

Cloud template source ● VMware cloud templates ○ Source Control

Cloud template * --Select template--

Version * --Select template Version--

Output Parameters

6 Run the pipeline.

To verify that each task completed successfully, click the task in the execution, and examine the status in the deployment details to see detailed resource information.

7 In GitHub, modify the flavor of the WordPress server instance from `small` to `medium`.

When you commit changes, the pipeline triggers. It pulls your updated code from the GitHub repository and builds your application.

```
WebTier:
    type: Cloud.Machine
    properties:
        name: wordpress
        image: 'ubuntu-16'
        flavor: 'medium'
        constraints:
            - tag: zone:dev
```

8 Run the pipeline again, verify that it succeeded, and that it changed the flavor of the WordPress instance from `small` to `medium`.

Results

Congratulations! You automated the release of your application that you deployed from a YAML cloud template.

What to do next

To learn more about how you can use Code Stream, see Chapter 5 Tutorials for using Code Stream.

For additional references, see More resources for Code Stream Administrators and Developers.

How do I automate the release of an application in Code Stream to a Kubernetes cluster

As a Code Stream administrator or developer, you can use Code Stream and VMware Tanzu Kubernetes Grid Integrated Edition (formerly known as VMware Enterprise PKS) to automate the deployment of your software applications to a Kubernetes cluster. This use case mentions other methods that you can use to automate the release of your application.

In this use case, you will create a pipeline that includes two stages, and will use Jenkins to build and deploy your application.

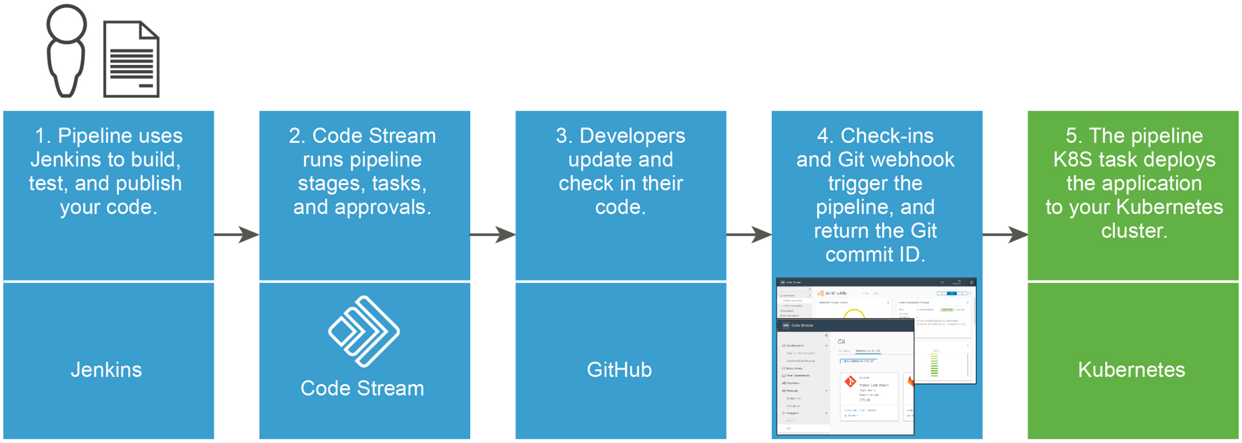
- The first stage is for development. It uses Jenkins to pull your code from a branch in your GitHub repository, then build, test, and publish it.
- The second stage is for deployment. It runs a user operation task that requires approval from key users before the pipeline can deploy your application to your Kubernetes cluster.

When using a Kubernetes API endpoint in the pipeline workspace, Code Stream creates the necessary Kubernetes resources such as ConfigMap, Secret, and Pod to run the continuous integration (CI) task or custom task. Code Stream communicates with the container by using the NodePort.

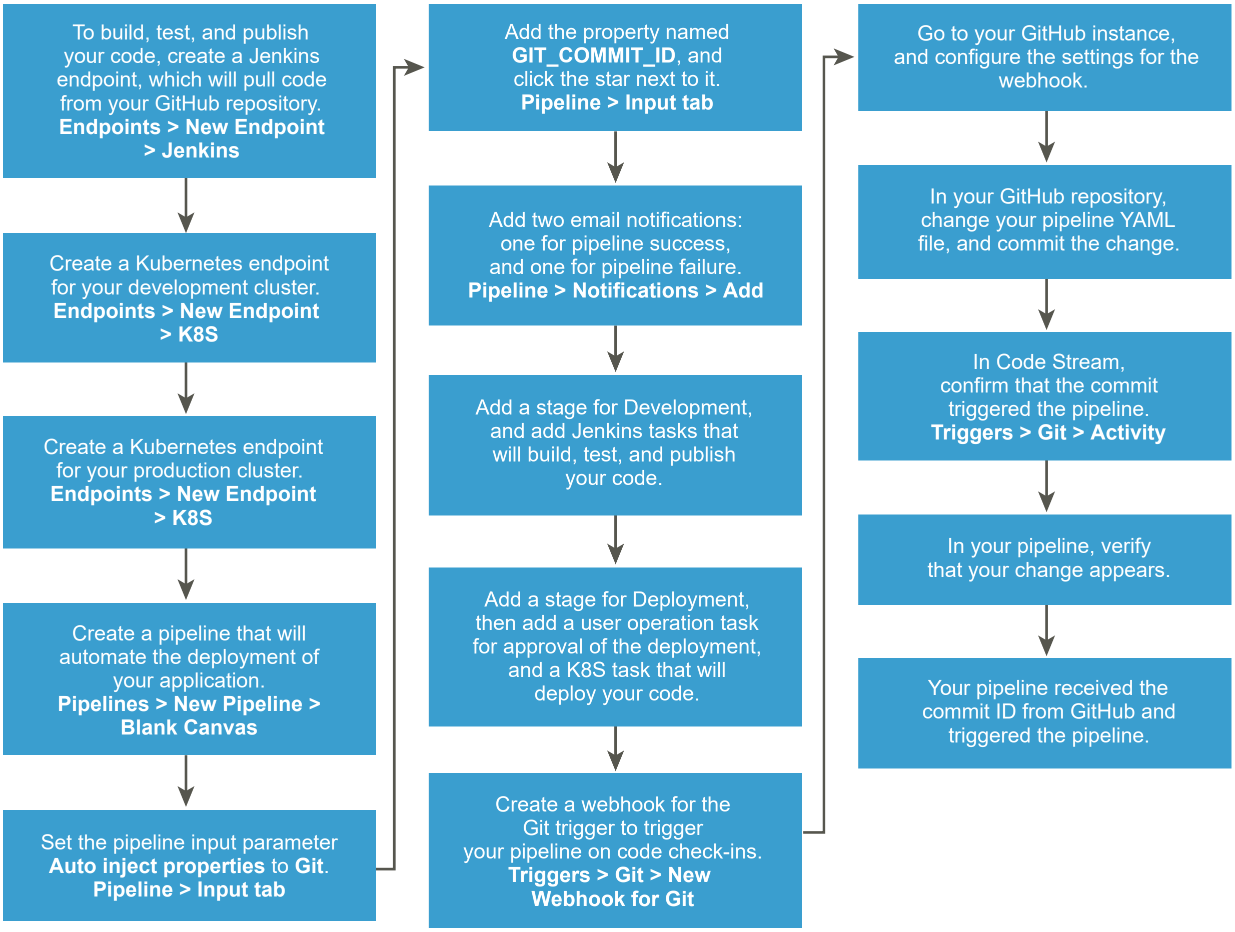
To share data across pipeline runs, you must provide a persistent volume claim, and Code Stream will mount the persistent volume claim to the container to store the data, and use it for subsequent pipeline runs.

The Code Stream pipeline workspace supports Docker and Kubernetes for continuous integration tasks and custom tasks.

For more information about configuring the workspace, see Configuring the Pipeline Workspace.


1. Pipeline uses Jenkins to build, test, and publish your code.
2. Code Stream runs pipeline stages, tasks, and approvals.
3. Developers update and check in their code.
4. Check-ins and Git webhook trigger the pipeline, and return the Git commit ID.
5. The pipeline K8S task deploys the application to your Kubernetes cluster.
Jenkins
Code Stream
GitHub
Kubernetes

The development tools, deployment instances, and pipeline YAML file must be available so that your pipeline can build, test, publish, and deploy your application. The pipeline will deploy your application to development and production instances of Kubernetes clusters on AWS.



To build, test, and publish your code, create a Jenkins endpoint, which will pull code from your GitHub repository. **Endpoints > New Endpoint > Jenkins**

Create a Kubernetes endpoint for your development cluster. **Endpoints > New Endpoint > K8S**

Create a Kubernetes endpoint for your production cluster. **Endpoints > New Endpoint > K8S**

Create a pipeline that will automate the deployment of your application. **Pipelines > New Pipeline > Blank Canvas**

Set the pipeline input parameter **Auto inject properties** to **Git**. **Pipeline > Input tab**

Add the property named **GIT_COMMIT_ID**, and click the star next to it. **Pipeline > Input tab**

Add two email notifications: one for pipeline success, and one for pipeline failure. **Pipeline > Notifications > Add**

Add a stage for Development, and add Jenkins tasks that will build, test, and publish your code.

Add a stage for Deployment, then add a user operation task for approval of the deployment, and a K8S task that will deploy your code.

Create a webhook for the Git trigger to trigger your pipeline on code check-ins. **Triggers > Git > New Webhook for Git**

Go to your GitHub instance, and configure the settings for the webhook.

In your GitHub repository, change your pipeline YAML file, and commit the change.

In Code Stream, confirm that the commit triggered the pipeline. **Triggers > Git > Activity**

In your pipeline, verify that your change appears.

Your pipeline received the commit ID from GitHub and triggered the pipeline.

Other methods that automate the release of your application:

- Instead of building your application by using Jenkins, you can use the Code Stream native build capability and a Docker build host.
- Instead of deploying your application to a Kubernetes cluster, you could deploy it to an Amazon Web Services (AWS) cluster.

For more information about using the Code Stream native build capability and a Docker host, see:

- Planning a CICD native build in Code Stream before using the smart pipeline template
- Planning a CICD native build in Code Stream before manually adding tasks

Prerequisites

- Verify that the application code to deploy resides in a working GitHub repository.
- Verify that you have a working instance of Jenkins.
- Verify that you have a working email server.
- In Code Stream, create an email endpoint that connects to your email server.
- Set up two Kubernetes clusters on Amazon Web Services (AWS), for development and production, where your pipeline will deploy your application.

- Verify that the GitHub repository contains the YAML code for your pipeline, and alternatively a YAML file that defines the metadata and specifications for your environment.

Procedure

1 In Code Stream, click **Endpoints > New Endpoint**, and create a Jenkins endpoint that you will use in your pipeline to pull code from your GitHub repository.

2 To create Kubernetes endpoints, click **New Endpoint**.

   a Create an endpoint for your development Kubernetes cluster.

   b Create an endpoint for your production Kubernetes cluster.

   The URL for your Kubernetes cluster might or might not include a port number.

   For example:

   `https://10.111.222.333:6443`

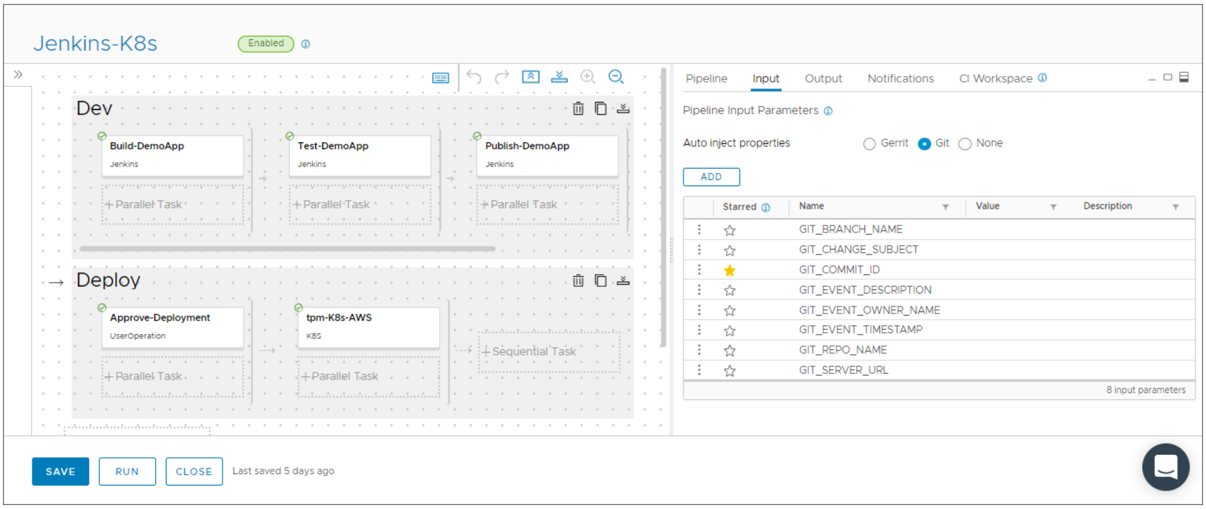
   `https://api.kubernetesserver.fa2c1d78-9f00-4e30-8268-4ab81862080d.k8s-user.com`

3 Create a pipeline that deploys a container of your application, such as Wordpress, to your development Kubernetes cluster, and set the input properties for the pipeline.

   a To allow your pipeline to recognize a code commit in GitHub that will trigger the pipeline, in the pipeline click the **Input** tab and select **Auto inject properties**.

   b Add the property named **GIT_COMMIT_ID**, and click the star next to it.

   When the pipeline runs, the pipeline execution will display the commit ID that the Git trigger returns.



Pipeline Input Parameters

Auto inject properties ○ Gerrit ● Git ○ None

ADD

Starred	Name	Value	Description
☆	GIT_BRANCH_NAME		
☆	GIT_CHANGE_SUBJECT		
★	GIT_COMMIT_ID		
☆	GIT_EVENT_DESCRIPTION		
☆	GIT_EVENT_OWNER_NAME		
☆	GIT_EVENT_TIMESTAMP		
☆	GIT_REPO_NAME		
☆	GIT_SERVER_URL		

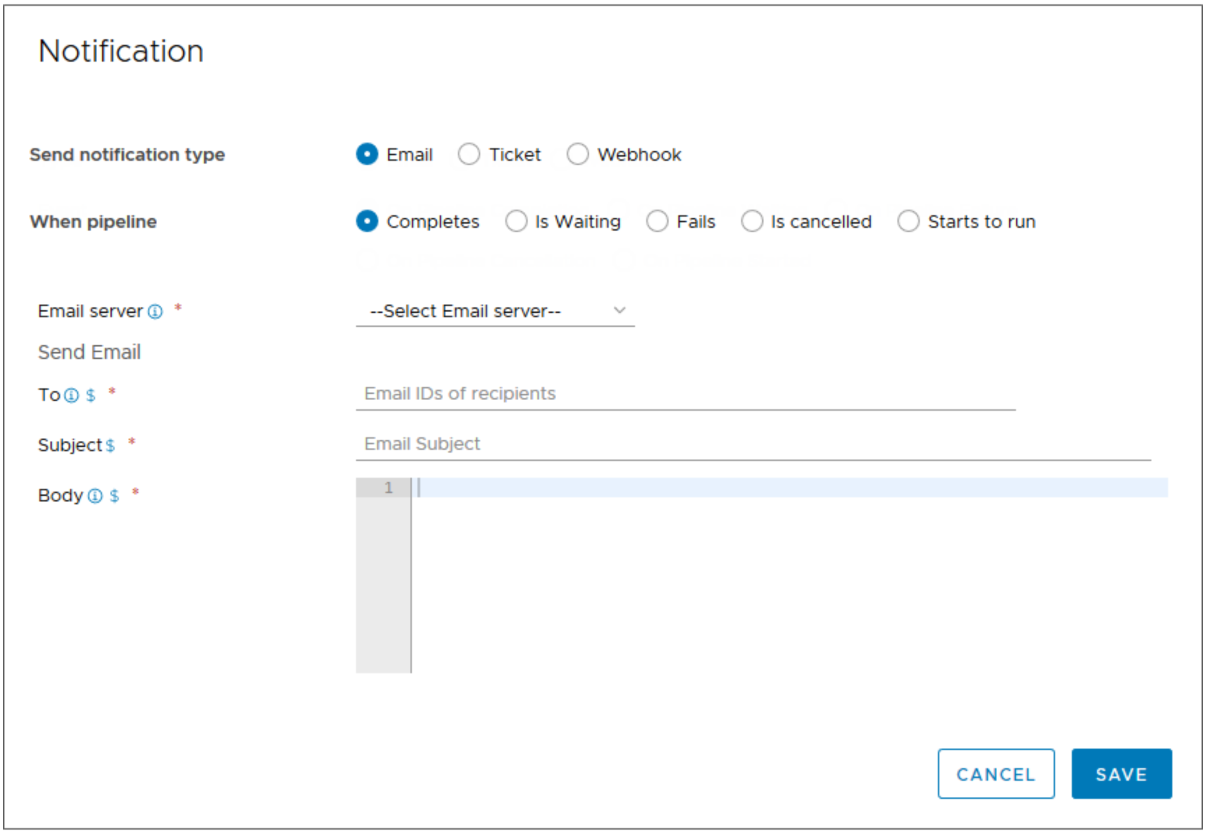
8 input parameters

4 Add notifications to send an Email when the pipeline succeeds or fails.

  a In the pipeline, click the **Notifications** tab, and click **Add**.

  b To add an email notification when the pipeline finishes running, select **Email**, and select **Completes**. Then, select the email server, enter email addresses, and click **Save**.

  c To add another email notification for a pipeline failure, select **Fails**, and click **Save**.



Notification

Send notification type ● Email ○ Ticket ○ Webhook

When pipeline ● Completes ○ Is Waiting ○ Fails ○ Is cancelled ○ Starts to run

Email server * --Select Email server--

Send Email

To $ * Email IDs of recipients

Subject $ * Email Subject

Body $ *

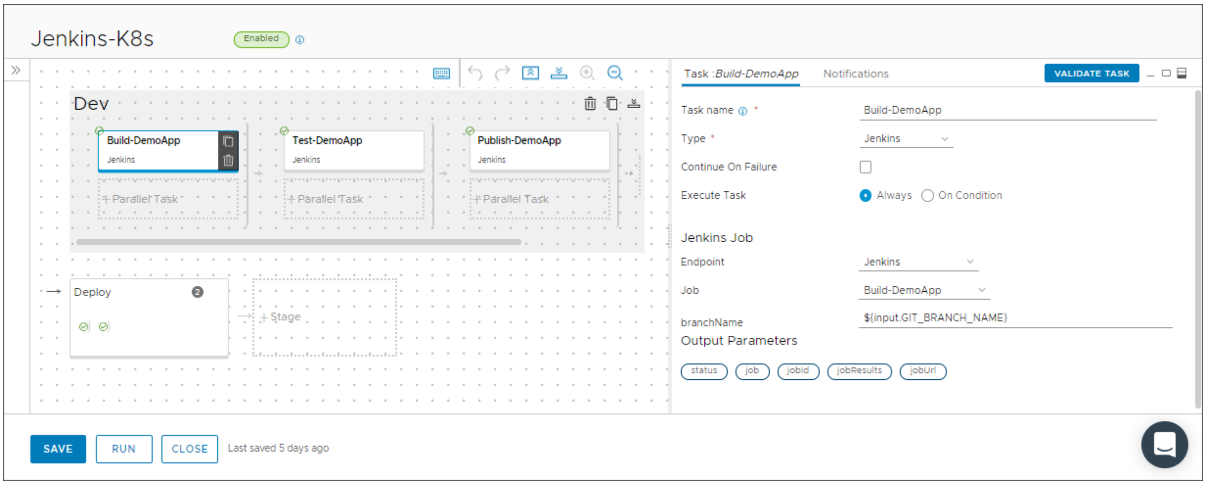
```
1 |
```

5 Add a development stage to your pipeline, and add tasks that build, test, and publish your application. Then, validate each task.

  a To build your application, add a Jenkins task that uses the Jenkins endpoint, and runs a build job from the Jenkins server. Then, for the pipeline to pull your code, enter the Git branch in this form: **`${input.GIT_BRANCH_NAME}`**

  b To test your application, add a Jenkins task that uses the same Jenkins endpoint, and runs a test job from the Jenkins server. Then, enter the same Git branch.

  c To publish your application, add a Jenkins task that uses the same Jenkins endpoint, and runs a publish job from the Jenkins server. Then, enter the same Git branch.



Task name * Build-DemoApp

Type * Jenkins

Continue On Failure □

Execute Task ● Always ○ On Condition

Jenkins Job

Endpoint Jenkins

Job Build-DemoApp

branchName ${input.GIT_BRANCH_NAME}

Output Parameters

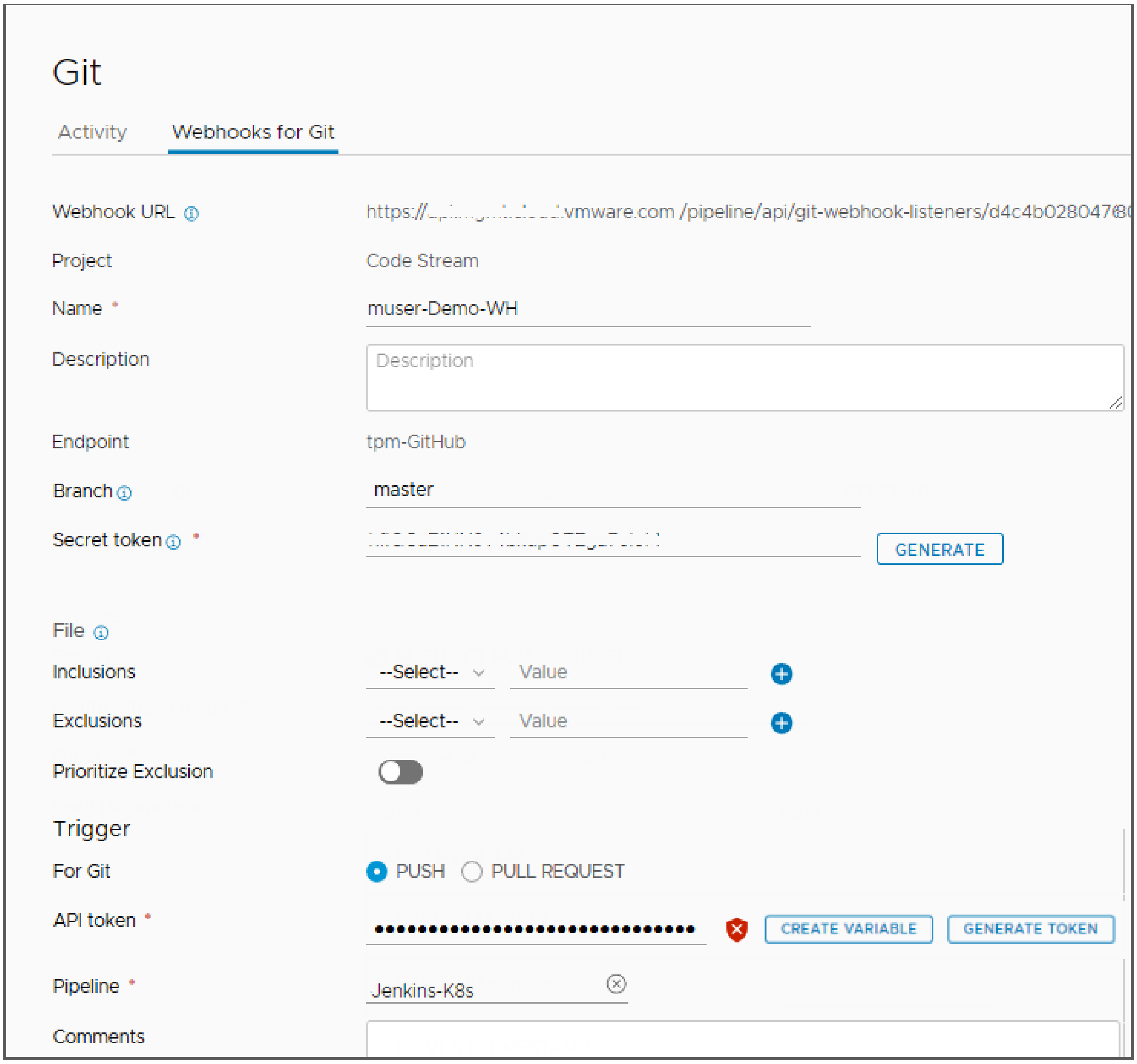
status job jobId jobResults jobUrl

6 Add a deployment stage to your pipeline, then add a task that requires an approval for deployment of your application, and another task that deploys the application to your Kubernetes cluster. Then, validate each task.

a To require an approval on the deployment of your application, add a User Operation task, add Email addresses for the users who must approve it, and enter a message. Then, enable **Send email**.

b To deploy your application, add a Kubernetes task. Then, in the Kubernetes task properties, select your development Kubernetes cluster, select the **Create** action, and select the **Local Definition** payload source. Then select your local YAML file.

7 Add a Git webhook that enables Code Stream to use the Git trigger, which triggers your pipeline when developers commit their code.



Git

Webhook URL ⓘ https://[redacted].vmware.com /pipeline/api/git-webhook-listeners/d4c4b0280478

Project Code Stream

Name * muser-Demo-WH

Description Description

Endpoint tpm-GitHub

Branch ⓘ master

Secret token ⓘ * [redacted]

File ⓘ

Inclusions --Select-- Value

Exclusions --Select-- Value

Prioritize Exclusion

Trigger

For Git ● PUSH ○ PULL REQUEST

API token * ••••••••••••••••••••••••••••••••

Pipeline * Jenkins-K8s

Comments

8 To test your pipeline, go to your GitHub repository, update your application YAML file, and commit the change.

a In Code Stream, verify that the commit appears.

a Click **Triggers > Git > Activity**.

b Look for the trigger of your pipeline.

c Click **Dashboards > Pipeline Dashboards**.

d On your pipeline dashboard, find the GIT_COMMIT_ID in the latest successful change area.

9 Check your pipeline code and verify that the change appears.

Results

Congratulations! You automated the deployment of your software application to your Kubernetes cluster.

Example: Example pipeline YAML that deploys an application to a Kubernetes cluster

For the type of pipeline used in this example, the YAML resembles the following code:

```
apiVersion: v1
kind: Namespace
metadata:
  name: ${input.GIT_BRANCH_NAME}
  namespace: ${input.GIT_BRANCH_NAME}
---
apiVersion: v1
data:
  .dockercfg:
eyJzeW1waG9ueS10YW5nby1iZXRhMi5qZnJvZy5pbyI6eyJ1c2VybmFtZSI6InRhbmdvLWJldGEyIiwicGFzc3dvcmQiOi
JhRGstcmVOLW1UQi1IejciLCJlbWFpbCI6InRhbmdvLWJldGEyQHZtd2FyZS5jb20iLCJhdXRoIjoiZEdGdVoyOHRZbVYw
WVRJNllVUnJMWEpsVGkxdFZFSXRTSG8zIn19
kind: Secret
metadata:
  name: jfrog
  namespace: ${input.GIT_BRANCH_NAME}
type: kubernetes.io/dockercfg
---
apiVersion: v1
kind: Service
metadata:
  name: codestream
  namespace: ${input.GIT_BRANCH_NAME}
  labels:
    app: codestream
spec:
  ports:
    - port: 80
  selector:
    app: codestream
    tier: frontend
  type: LoadBalancer
---
apiVersion: extensions/v1
kind: Deployment
metadata:
  name: codestream
  namespace: ${input.GIT_BRANCH_NAME}
  labels:
    app: codestream
spec:
  selector:
    matchLabels:
      app: codestream
      tier: frontend
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: codestream
        tier: frontend
    spec:
      containers:
      - name: codestream
        image: cas.jfrog.io/codestream:${input.GIT_BRANCH_NAME}-${Dev.PublishApp.output.jobId}
        ports:
        - containerPort: 80
          name: codestream
      imagePullSecrets:
      - name: jfrog
```

What to do next

To deploy your software application to your production Kubernetes cluster, perform the steps again and select your production cluster.

To learn more about integrating Code Stream with Jenkins, see How do I integrate Code Stream with Jenkins.

How do I deploy my application in Code Stream to my Blue-Green deployment

Blue-Green is a deployment model that uses two Docker hosts that you deploy and configure identically in a Kubernetes cluster. With the Blue and Green deployment model, you reduce the downtime that can occur in your environment when your pipelines in Code Stream deploy your applications.

The Blue and Green instances in your deployment model each serve a different purpose. Only one instance at a time accepts the live traffic that deploys your application, and each instance accepts that traffic at specific times. The Blue instance receives the first version of your application, and the Green instance receives the second.

The load balancer in your Blue-Green environment determines which route the live traffic takes as it deploys your application. By using the Blue-Green model, your environment remains operational, users don't notice any downtime, and your pipeline continuously integrates and deploys your application to your production environment.

The pipeline that you create in Code Stream represents your Blue-Green deployment model in two stages. One stage is for development, and the other stage is for production.

The Code Stream pipeline workspace supports Docker and Kubernetes for continuous integration tasks and custom tasks.

For information about configuring the workspace, see Configuring the Pipeline Workspace.

Table 5-2. Development stage tasks for Blue-Green deployment

Task type	Task
Kubernetes	Create a namespace for your Blue-Green deployment.
Kubernetes	Create a secret key for Docker Hub.
Kubernetes	Create the service used to deploy the application.
Kubernetes	Create the Blue deployment.
Poll	Verify the Blue deployment.
Kubernetes	Remove the namespace.

Table 5-3. Production stage tasks for Blue-Green deployment

Task type	Task
Kubernetes	Green gets the service details from Blue.
Kubernetes	Get the details for the Green replica set.
Kubernetes	Create the Green deployment, and use the secret key to pull the container image.
Kubernetes	Update the service.
Poll	Verify that the deployment succeeded on the production URL.
Kubernetes	Finish the Blue deployment.
Kubernetes	Remove the Blue deployment.

To deploy your application in your own Blue-Green deployment model, you create a pipeline in Code Stream that includes two stages. The first stage includes the Blue tasks that deploy your application to the Blue instance, and the second stage includes Green tasks that deploy your application to the Green instance.

You can create your pipeline by using the CICD smart pipeline template. The template creates your pipeline stages and tasks for you, and includes the deployment selections.

If you create your pipeline manually, you must plan your pipeline stages. For an example, see Planning a CICD native build in Code Stream before manually adding tasks.

In this example, you use the CICD smart pipeline template to create your Blue-Green pipeline.

Prerequisites

- Verify that you can access a working Kubernetes cluster on AWS.
- Verify that you set up a Blue-Green deployment environment, and configured your Blue and Green instances to be identical.

- Create a Kubernetes endpoint in Code Stream that deploys your application image to the Kubernetes cluster on AWS.
- Familiarize yourself with using the CICD smart pipeline template. See Planning a CICD native build in Code Stream before using the smart pipeline template.

Procedure

1 Click **Pipelines** > **New Pipeline** > **Smart Templates** > **CI/CD template**.

2 Enter the information for the CI portion of the CICD smart pipeline template, and click **Next**.

For help, see Planning a CICD native build in Code Stream before using the smart pipeline template.

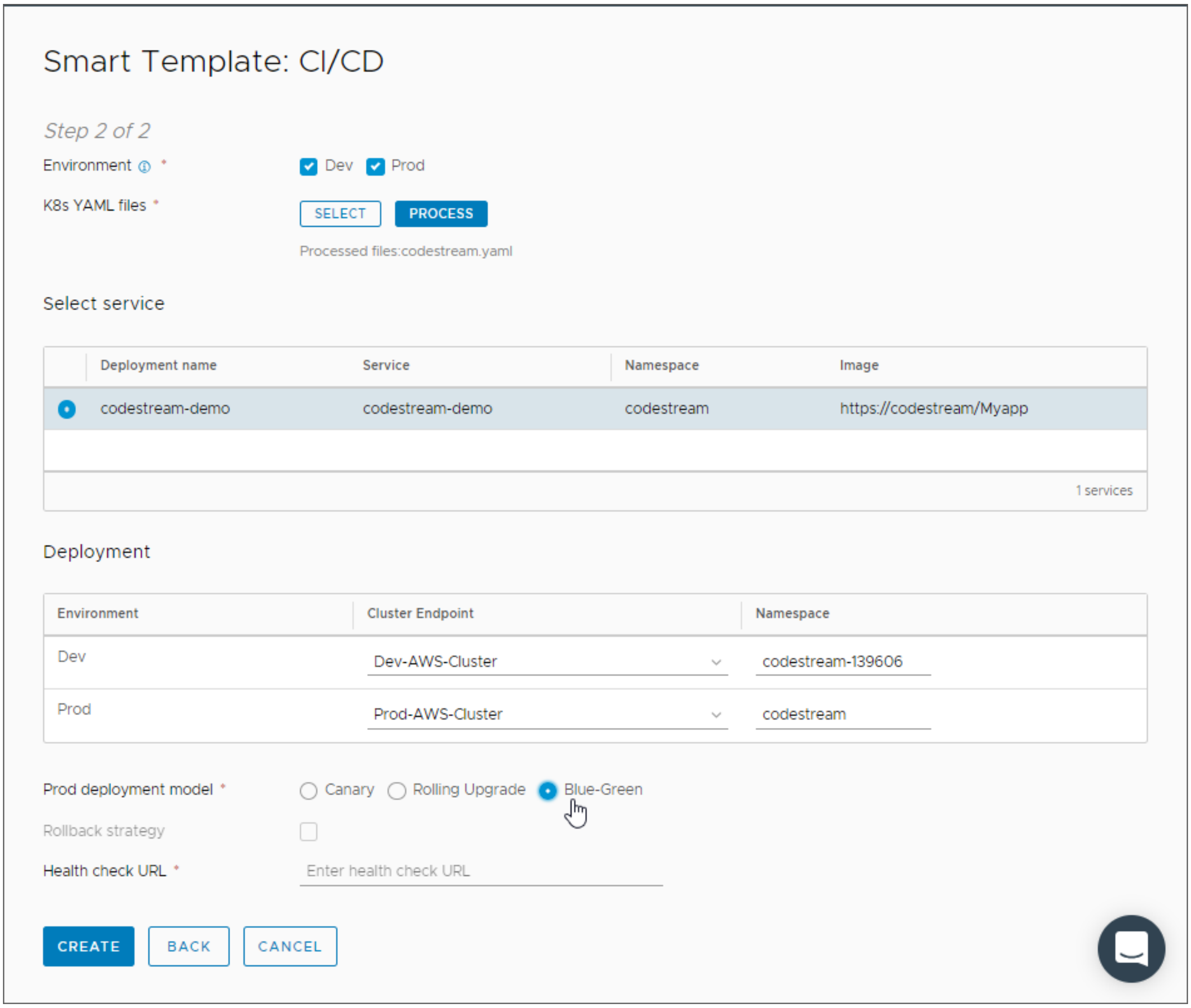
3 Complete the CD portion of the smart pipeline template

a Select the environments for your application deployment. For example, **Dev** and **Prod**.

b Select the service that the pipeline will use for the deployment.

c In the Deployment area, select the cluster endpoint for the Dev environment and the Prod environment.

d For the Production deployment model, select **Blue-Green**, and click **Create**.



Smart Template: CI/CD

Step 2 of 2

Environment ⓘ * ☑ Dev ☑ Prod

K8s YAML files * SELECT PROCESS

Processed files:codestream.yaml

Select service

	Deployment name	Service	Namespace	Image
●	codestream-demo	codestream-demo	codestream	https://codestream/Myapp

1 services

Deployment

Environment	Cluster Endpoint	Namespace
Dev	Dev-AWS-Cluster	codestream-139606
Prod	Prod-AWS-Cluster	codestream

Prod deployment model * ○ Canary ○ Rolling Upgrade ● Blue-Green

Rollback strategy □

Health check URL * Enter health check URL

Results

Congratulations! You used the smart pipeline template to create a pipeline that deploys your application to your Blue-Green instances in your Kubernetes production cluster on AWS.

Example: Example YAML code for some Blue-Green Deployment Tasks

The YAML code that appears in Kubernetes pipeline tasks for your Blue-Green deployment might resemble the following examples that create the Namespace, Service, and Deployment. If you need to download an image from a privately-owned repository, the YAML file must include a section with the Docker config Secret. See the CD portion of Planning a CICD native build in Code Stream before using the smart pipeline template.

After the smart pipeline template creates your pipeline, you can modify the tasks as needed for your own deployment.

YAML code to create an example namespace:

```
apiVersion: v1
kind: Namespace
```

```
metadata:
  name: codestream-82855
  namespace: codestream-82855
```

YAML code to create an example service:

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: codestream-demo
  name: codestream-demo
  namespace: bluegreen-799584
spec:
  minReadySeconds: 0
  ports:
  - port: 80
  selector:
    app: codestream-demo
    tier: frontend
  type: LoadBalancer
```

YAML code to create an example deployment:

```
apiVersion: extensions/v1
kind: Deployment
metadata:
  labels:
    app: codestream-demo
  name: codestream-demo
  namespace: bluegreen-799584
spec:
  minReadySeconds: 0
  replicas: 1
  selector:
    matchLabels:
      app: codestream-demo
      tier: frontend
  template:
    metadata:
      labels:
        app: codestream-demo
        tier: frontend
    spec:
      containers:
      - image: ${input.image}:${input.tag}
        name: codestream-demo
        ports:
        - containerPort: 80
          name: codestream-demo
      imagePullSecrets:
      - name: jfrog-2
      minReadySeconds: 0
```

What to do next

To learn more about how you can use Code Stream, see Chapter 5 Tutorials for using Code Stream.

To roll back a deployment, see How do I roll back my deployment in Code Stream.

For additional references, see More resources for Code Stream Administrators and Developers.

How do I integrate my own build, test, and deploy tools with Code Stream

As a DevOps administrator or developer, you can create custom scripts that extend the capability of Code Stream.

With your script, you can integrate Code Stream with your own Continuous Integration (CI) and Continuous Delivery (CD) tools and APIs that build, test, and deploy your applications. Custom scripts are especially useful if you do not expose your application APIs publicly.

Your custom script can do almost anything you need for your build, test, and deploy tools integrate with Code Stream. For example, your script can work with your pipeline workspace to support continuous integration tasks that build and test your application, and continuous delivery tasks that deploy your application. It can send a message to Slack when a pipeline finishes, and much more.

The Code Stream pipeline workspace supports Docker and Kubernetes for continuous integration tasks and custom tasks.

For more information about configuring the workspace, see Configuring the Pipeline Workspace.

You write your custom script in one of the supported languages. In the script, you include your business logic, and define inputs and outputs. Output types can include number, string, text, and password. You can create multiple versions of a custom script with different business logic, input, and output.

You have your pipeline run a version of your script in a custom task. The scripts that you create reside in your Code Stream instance.

When a pipeline uses a custom integration, if you attempt to delete the custom integration, an error message appears and indicates that you cannot delete it.

Deleting a custom integration removes all versions of your custom script. If you have an existing pipeline with a custom task that uses any version of the script, that pipeline will fail. To ensure that existing pipelines do not fail, you can deprecate and withdraw the version of your script that you no longer want used. If no pipeline is using that version, you can delete it.

Table 5-4. What you do after you write your custom script

What you do...	More information about this action...
Add a custom task to your pipeline.	The custom task: ■ Runs on the same container as other CI tasks in your pipeline. ■ Includes input and output variables that your script populates before the pipeline runs the custom task. ■ Supports multiple data types and various types of meta data that you define as inputs and outputs in your script.
Select your script in the custom task.	You declare the input and output properties in the script.
Save your pipeline, then enable and run it.	When the pipeline runs, the custom task calls the version of the script specified and runs the business logic in it, which integrates your build, test, and deploy tool with Code Stream.
After your pipeline runs, look at the executions.	Verify that the pipeline delivered the results you expected.

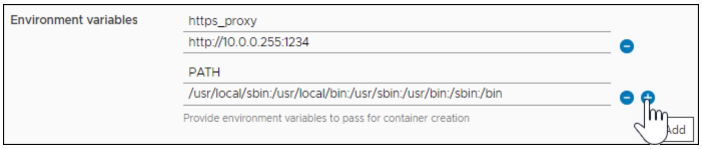
When you use a custom task that calls a Custom Integration version, you can include custom environment variables as name-value pairs on the pipeline **Workspace** tab. When the builder image creates the workspace container that runs the CI task and deploys your image, Code Stream passes the environment variables to that container.

For example, when your Code Stream instance requires a Web proxy, and you use a Docker host to create a container for a custom integration, Code Stream runs the pipeline and passes the Web proxy setting variables to that container.

Table 5-5. Example environment variable name-value pairs

Name	Value
HTTPS_PROXY	http://10.0.0.255:1234
https_proxy	http://10.0.0.255:1234
NO_PROXY	10.0.0.32, *.dept.vsphere.local
no_proxy	10.0.0.32, *.dept.vsphere.local
HTTP_PROXY	http://10.0.0.254:1234
http_proxy	http://10.0.0.254:1234
PATH	/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

Name-value pairs appear in the user interface like this:



Environment variables

https_proxy
http://10.0.0.255:1234

PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

Provide environment variables to pass for container creation

This example creates a custom integration that connects Code Stream to your Slack instance, and posts a message to a Slack channel.

Prerequisites

- To write your custom script, verify that you have one of these languages: Python 2, Python 3, Node.js, or any of the shell languages: Bash, sh, or zsh.
- Generate a container image by using the installed Node.js or the Python runtime.

Procedure

1 Create the custom integration.

   a Click **Custom Integrations > New**, and enter a relevant name.

   b Select the preferred runtime environment.

   c Click **Create**.

   Your script opens, and displays the code, which includes the required runtime environment. For example, `runtime: "nodejs"`. The script must include the runtime, which the builder image uses, so that the custom task that you add to your pipeline succeeds when the pipeline runs. Otherwise, the custom task fails.

   The main areas of your custom integration YAML include the runtime, code, input properties, and output properties. This procedure explains various types and syntax.

Custom integration YAML keys	Description
runtime	Task runtime environment where Code Stream runs the code, which can be one of these case-insensitive strings: ■ nodejs ■ python2 ■ python3 ■ shell If nothing is provided, shell is the assumed default.
code	Custom business logic to run as part of the custom task.
inputProperties	Array of input properties to capture as part of the custom task configuration. These properties are normally used in the code.
outputProperties	Array of output properties you can export from the custom task to propagate to the pipeline.

2 Declare the input properties in your script by using the available data types and meta data.

   The input properties are passed in as context to your script in the `code:` section of the YAML.

Custom task YAML input keys	Description	Required
`type`	Types of input to render: - **`text`** - **`textarea`** - **`number`** - **`checkbox`** - **`password`** - **`select`**	Yes
`name`	Name or string of the input to the custom task, which gets injected into the custom integration YAML code. Must be unique for each input property defined for a custom integration.	Yes
`title`	Text string label of the input property for the custom task on the pipeline model canvas. If left empty, **`name`** is used by default.	No
`required`	Determines whether a user must enter the input property when they configure the custom task. Set to true or false. When true, if a user does not provide a value when they configure the custom task on the pipeline canvas, the state of the task remains as unconfigured.	No
`placeHolder`	Default text for the input property entry area when no value is present. Maps to the html placeholder attribute. Only supported for certain input property types.	No
`defaultValue`	Default value that populates the input property entry area when the custom task renders on the pipeline model page.	No
`bindable`	Determines whether the input property accepts dollar sign variables when modeling the custom task on the pipeline canvas. Adds the **$** indicator next to the title. Only supported for certain input property types.	No
`labelMessage`	String that acts as a help tooltip for users. Adds a tooltip icon **i** next to the input title.	No
`enum`	Takes in an array of values that displays the select input property options. Only supported only for certain input property types. When a user selects an option, and saves it for the custom task, the value of **inputProperty** corresponds to this value and appears in the custom task modeling. For example, the value 2015. - 2015 - 2016 - 2017 - 2018 - 2019 - 2020	No

Custom task YAML input keys	Description	Required
`options`	Takes in an array of objects by using **optionKey** and **optionValue**. ■ **optionKey**. Value propagated to the code section of the task. ■ **optionValue**. String that displays the option in the user interface. Only supported only for certain input property types. Options: **optionKey**: key1. When selected and saved for the custom task, the value of this inputProperty corresponds to **key1** in the code section. **optionValue**: 'Label for 1'. Display value for **key1** in the user interface, and does not appear anywhere else for the custom task. **optionKey**: key2 **optionValue**: 'Label for 2' **optionKey**: key3 **optionValue**: 'Label for 3'	No
`minimum`	Takes in a number that acts as the minimum value that is valid for this input property. Only supported for number type input property.	No
`maximum`	Takes in a number that acts as the maximum value that is valid for this input property. Only supported for number type input property.	No

Table 5-6. Supported data types and meta data for custom scripts

Supported data types	Supported meta data for input
■ String ■ Text ■ List: as a list of any type ■ Map: as map[string]any ■ Secure: rendered as password text box, encrypted when you save the custom task ■ Number ■ Boolean: appears as text boxes ■ URL: same as string, with additional validation ■ Selection, radio button	■ type: One of String | Text ... ■ default: Default value ■ options: List or a map of options, to be used with selection or radio button ■ min: Minimum value or size ■ max: Maximum value or size ■ title: Detailed name of the text box ■ placeHolder: UI placeholder ■ description: Becomes a tool tip

For example:

```
inputProperties:
        - name: message
          type: text
          title: Message
          placeHolder: Message for Slack Channel
          defaultValue: Hello Slack
          bindable: true
          labelInfo: true
          labelMessage: This message is posted to the Slack channel link provided in the
code
```

3 Declare the output properties in your script.

The script captures output properties from the business logic `code:` section of your script, where you declare the context for the output.

When the pipeline runs, you can enter the response code for the task output. For example, **`200`**.

Keys that Code Stream supports for each **outputProperty**.

key	Description
type	Currently includes a single value of **`label`**.
name	Key that the code block of the custom integration YAML emits.
title	Label in the user interface that displays **outputProperty**.

For example:

```
outputProperties:
  - name: statusCode
    type: label
    title: Status Code
```

4 To interact with the input and output of your custom script, get an input property or set an output property by using **`context`**.

For an input property: `(context.getInput("key"))`

For an output property: `(context.setOutput("key", "value"))`

For Node.js:

```
var context = require("./context.js")
var message = context.getInput("message");
//Your Business logic
context.setOutput("statusCode", 200);
```

For Python:

```
from context import getInput, setOutput
message = getInput('message')
//Your Business logic
setOutput('statusCode', '200')
```

For Shell:

```
# Input, Output properties are environment variables
echo ${message} # Prints the input message
//Your Business logic
export statusCode=200 # Sets output property statusCode
```

5 In the `code:` section, declare all the business logic for your custom integration.

For example, with the Node.js runtime environment:

```
code: |
    var https = require('https');
    var context = require("./context.js")

    //Get the entered message from task config page and assign it to message var
    var message = context.getInput("message");
    var slackPayload = JSON.stringify(
        {
            text: message
        });

    const options = {
        hostname: 'hooks.slack.com',
        port: 443,
        path: '/YOUR_SLACK_WEBHOOK_PATH',
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'Content-Length': Buffer.byteLength(slackPayload)
        }
    };

    // Makes a https request and sets the output with statusCode which
    // will be displayed in task result page after execution
    const req = https.request(options, (res) => {
        context.setOutput("statusCode", res.statusCode);
    });

    req.on('error', (e) => {
        console.error(e);
    });
    req.write(slackPayload);
    req.end();
```

6 Before you version and release your custom integration script, download the context file for Python or Node.js and test the business logic that you included in your script.

a Place the pointer in the script, then click the context file button at the top of the canvas. For example, if your script is in Python click **CONTEXT.PY**.

b Modify the file and save it.

c On your development system, run and test your custom script with the help of the context file.

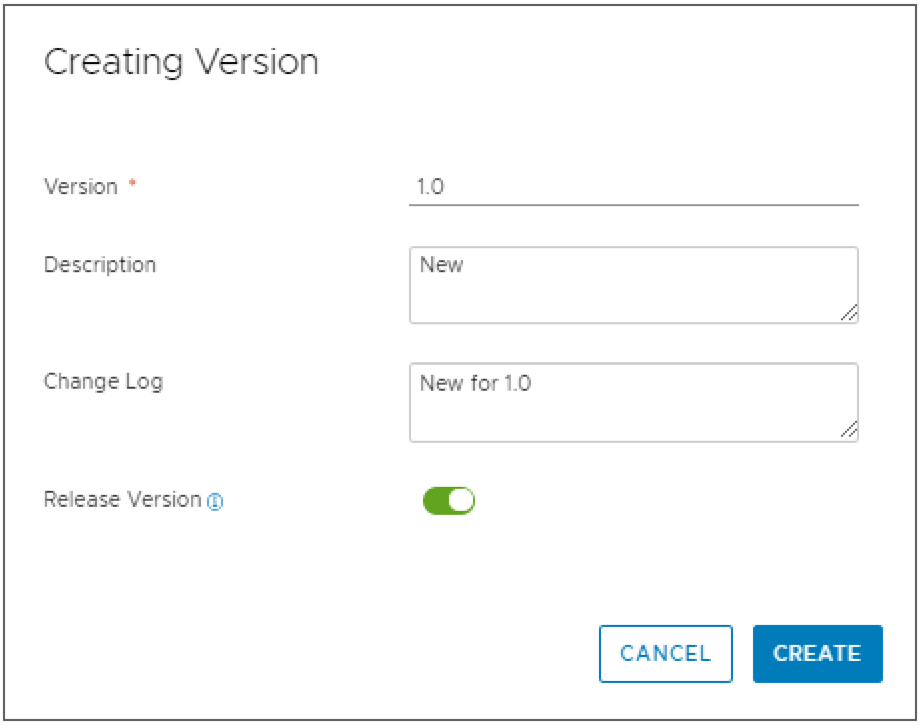
7 Apply a version to your custom integration script.

a Click **Version**.

b Enter the version information.

c Click **Release Version** so that you can select the script in your custom task.

d To create the version, click **Create**.



Creating Version

Version * 1.0

Description New

Change Log New for 1.0

Release Version

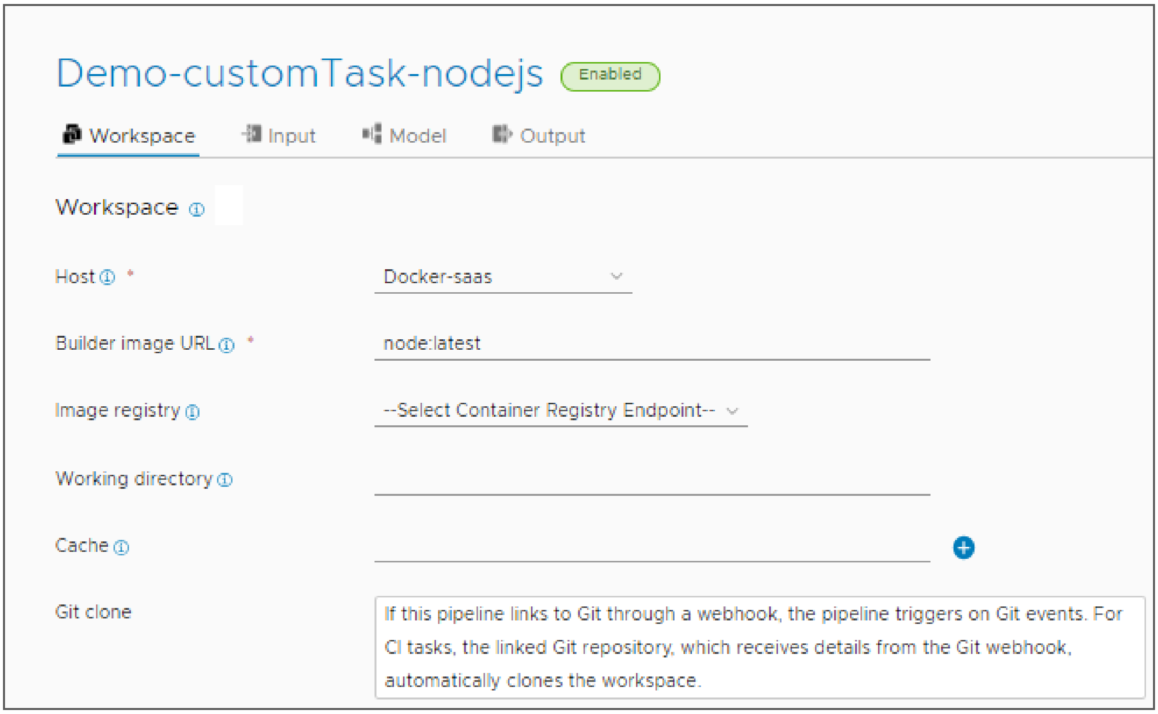
8 To save the script, click **Save**.

9 In your pipeline, configure the workspace.

This example uses a Docker workspace.

a Click the **Workspace** tab.

b Select the Docker host and the builder image URL.



Demo-customTask-nodejs Enabled

Workspace

Host *: Docker-saas

Builder image URL *: node:latest

Image registry: --Select Container Registry Endpoint--

Working directory:

Cache:

Git clone: If this pipeline links to Git through a webhook, the pipeline triggers on Git events. For CI tasks, the linked Git repository, which receives details from the Git webhook, automatically clones the workspace.

10 Add a custom task to your pipeline, and configure it.

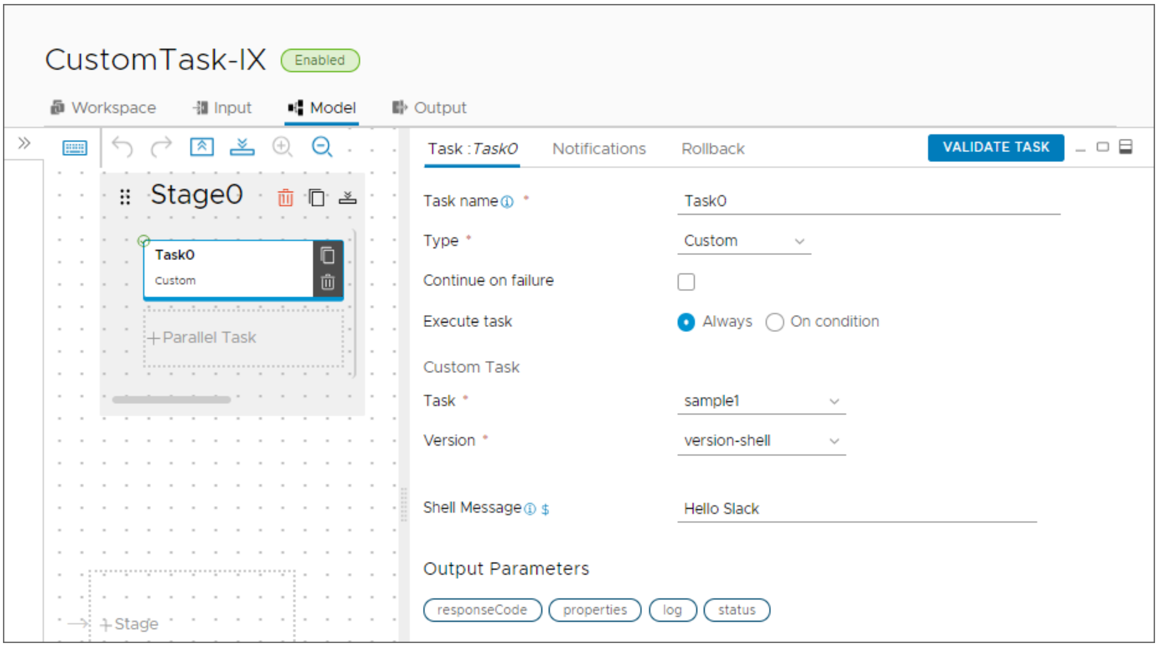
a Click the **Model** tab.

b Add a task, select the type as **Custom**, and enter a relevant name.

c Select your custom integration script and version.

d To display a custom message in Slack, enter the message text.

Any text you enter overrides the `defaultValue` in your custom integration script. For example:



Task name * Task0

Type * Custom

Continue on failure □

Execute task ● Always ○ On condition

Custom Task

Task * sample1

Version * version-shell

Shell Message $ Hello Slack

Output Parameters

responseCode properties log status

11 Save and enable your pipeline.

a Click **Save**.

b On the Pipeline tab, click **Enable pipeline** so that the circle moves to the right.

12 Run your pipeline.

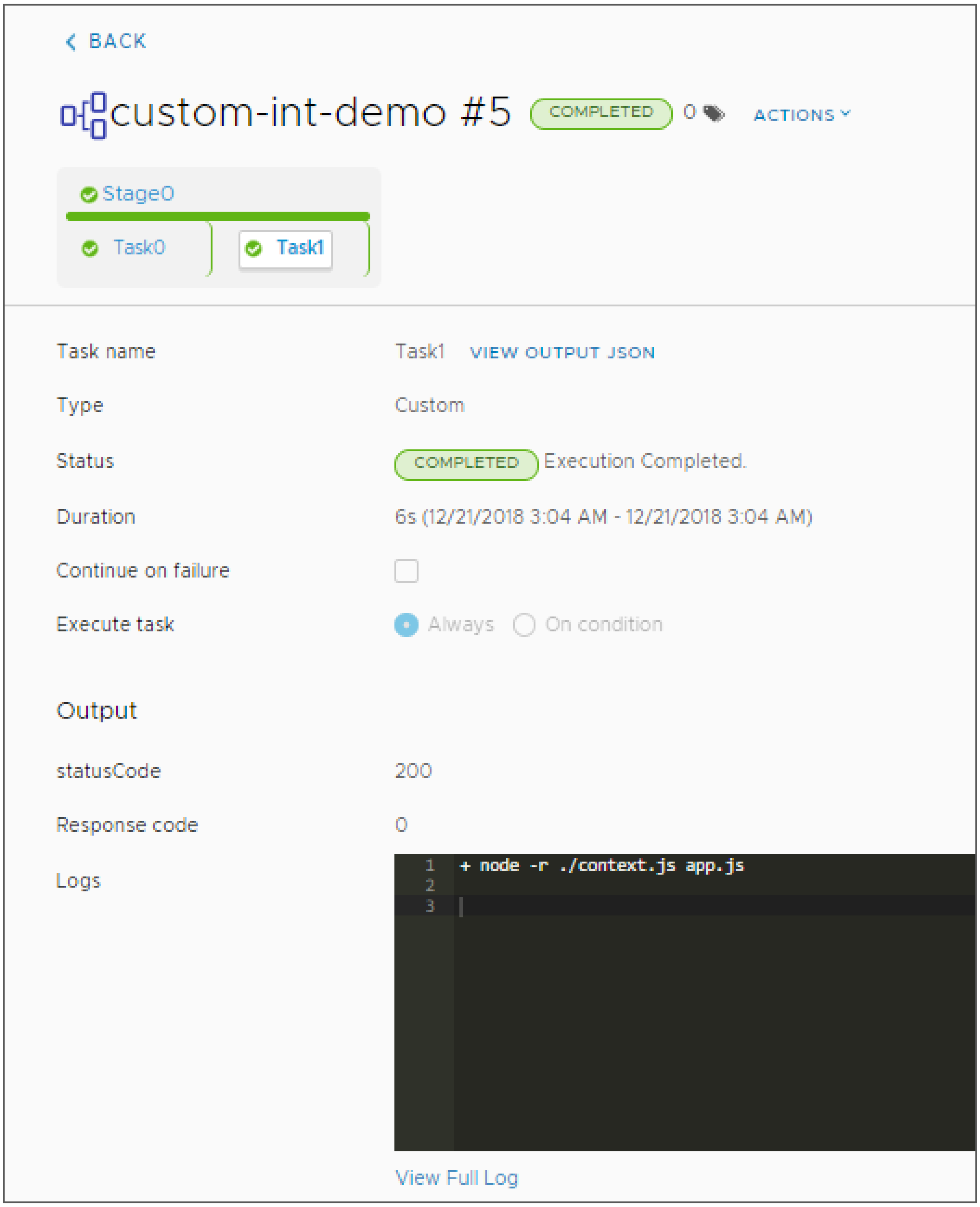
a Click **Run**.

b Look at the pipeline execution.

c Confirm that the output includes the expected status code, response code, status, and declared output.

  You defined **statusCode** as an output property. For example, a **statusCode** of `200` might indicate a successful Slack post, and a **responseCode** of `0` might indicate that the script succeeded without error.

d To confirm the output in the execution logs, click **Executions**, click the link to your pipeline, click the task, and look at the logged data. For example:



Task name: Task1

Type: Custom

Status: COMPLETED Execution Completed.

Duration: 6s (12/21/2018 3:04 AM - 12/21/2018 3:04 AM)

Continue on failure: □

Execute task: ● Always ○ On condition

Output

statusCode: 200

Response code: 0

Logs:

```
+ node -r ./context.js app.js


```

13 If an error occurs, troubleshoot the problem and run the pipeline again.

For example, if a file or module in the base image is missing, you must create another base image that includes the missing file. Then, provide the Docker file, and push the image through the pipeline.

Results

Congratulations! You created a custom integration script that connects Code Stream to your Slack instance, and posts a message to a Slack channel.

What to do next

Continue to create custom integrations to support using custom tasks in your pipelines, so that you can extend the capability of Code Stream in the automation of your software release lifecycle.

How do I use the resource properties of a cloud template task in my next task

When you use a cloud template task in Code Stream, a common question is how to use the output of that task in a subsequent task in your pipeline. To use the output of a cloud template task, such as a cloud machine, you must know how to find the resource properties in the deployment details of the cloud template task, and the IP address of the cloud machine.

For example, the deployment details of a VMware Cloud Template include the cloud machine resource and its IP address. In your pipeline, you can use the cloud machine and IP address as a variable to bind a cloud template task to a REST task.

The method that you use to find the IP address for the cloud machine is not typical, because the deployment of the VMware Cloud Template must finish before the deployment details are available. Then, you can use the resources from the VMware Cloud Template deployment to bind your pipeline tasks.

- The resource properties that appear in a cloud template task in your pipeline are defined in the VMware Cloud Template in Cloud Assembly.
- You might not know when a deployment of that cloud template finished.
- A cloud template task in Code Stream can only display the output properties of the VMware Cloud Template after the deployment finished.

This example can be especially useful if you are deploying an application and invoking various APIs. For example, if you use a cloud template task that calls a VMware Cloud Template, which deploys a Wordpress application with a REST API, you can locate the IP address of the deployed machine in the deployment details, and use the API to test it.

The cloud template task supports you to use variable binding by displaying the type ahead auto fill details. It is up to you how you bind the variable.

This example shows you how to:

- Find the deployment details and resource properties for your cloud template task in a pipeline that ran and succeeded.
- Find the cloud machine IP address in the resources section of the deployment details.
- Add a REST task subsequent to the cloud template task in your pipeline.
- Bind the cloud template task to the REST task by using the cloud machine IP address in the URL of the REST task.
- Run your pipeline and watch the binding work from the cloud template task to the REST task.

Prerequisites

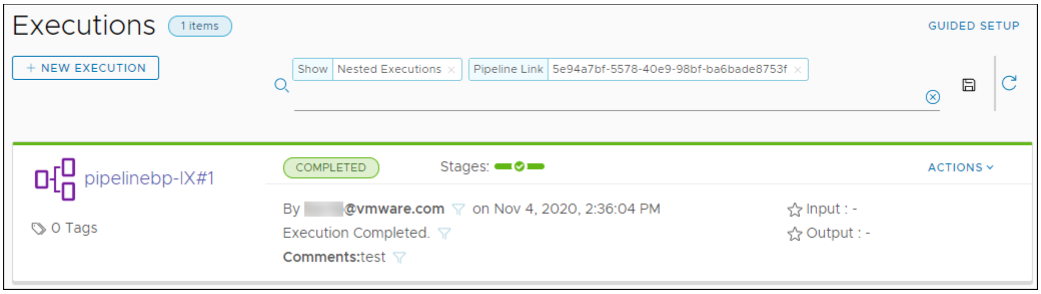
- Verify that you have a working VMware Cloud Template that is versioned.
- Verify that the deployment of the VMware Cloud Template succeeded in Cloud Assembly.
- Verify that you have a pipeline that includes a cloud template task that uses that VMware Cloud Template.
- Verify that your pipeline ran and succeeded.

Procedure

1 In your pipeline, locate the IP address of the cloud machine in the resources section of your cloud template task deployment details.

   a Click **Actions > View executions**.

   b In a pipeline run that succeeded, click the link to the pipeline execution.



Show Nested Executions

Pipeline Link 5e94a7bf-5578-40e9-98bf-ba6bade8753f

pipelinebp-IX#1

0 Tags

COMPLETED

By [redacted]@vmware.com on Nov 4, 2020, 2:36:04 PM

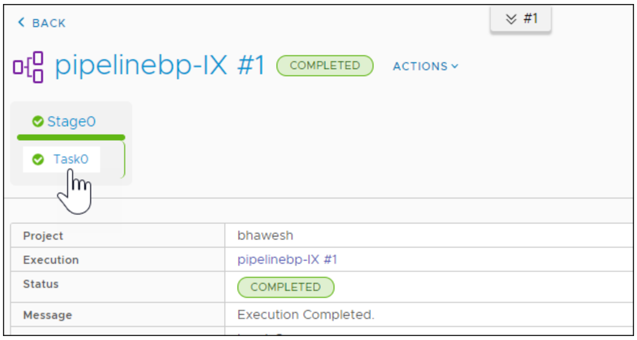
Execution Completed.

Comments:test

Input : -

Output : -

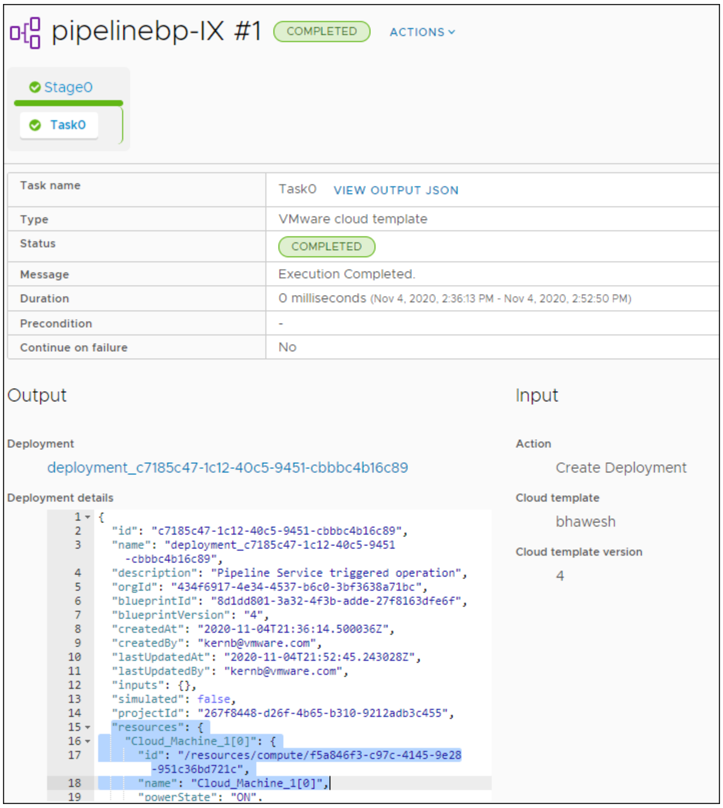
   c Under the pipeline name, click the link to the **Task**.



pipelinebp-IX #1 COMPLETED

Project	bhawesh
Execution	pipelinebp-IX #1
Status	COMPLETED
Message	Execution Completed.

d In the Output area, locate the Deployment details.



pipelinebp-IX #1 COMPLETED ACTIONS

Task name	Task0 VIEW OUTPUT JSON
Type	VMware cloud template
Status	COMPLETED
Message	Execution Completed.
Duration	0 milliseconds (Nov 4, 2020, 2:36:13 PM - Nov 4, 2020, 2:52:50 PM)
Precondition	-
Continue on failure	No

Output

Deployment

deployment_c7185c47-1c12-40c5-9451-cbbbc4b16c89

Deployment details

```
{
  "id": "c7185c47-1c12-40c5-9451-cbbbc4b16c89",
  "name": "deployment_c7185c47-1c12-40c5-9451
    -cbbbc4b16c89",
  "description": "Pipeline Service triggered operation",
  "orgId": "434f6917-4e34-4537-b6c0-3bf3638a71bc",
  "blueprintId": "8d1dd801-3a32-4f3b-adde-27f8163dfe6f",
  "blueprintVersion": "4",
  "createdAt": "2020-11-04T21:36:14.500036Z",
  "createdBy": "kernb@vmware.com",
  "lastUpdatedAt": "2020-11-04T21:52:45.243028Z",
  "lastUpdatedBy": "kernb@vmware.com",
  "inputs": {},
  "simulated": false,
  "projectId": "267f8448-d26f-4b65-b310-9212adb3c455",
  "resources": {
    "Cloud_Machine_1[0]": {
      "id": "/resources/compute/f5a846f3-c97c-4145-9e28
        -951c36bd721c",
      "name": "Cloud_Machine_1[0]",
      "powerState": "ON",
```

Input

Action

Create Deployment

Cloud template

bhawesh

Cloud template version

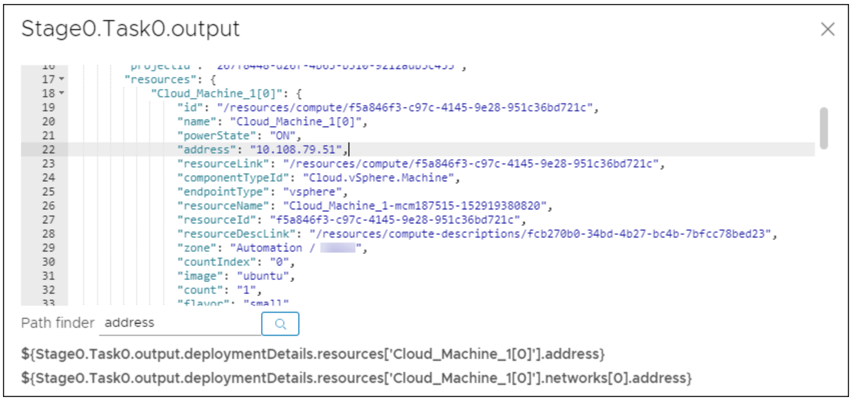
4

e In the resources section of the deployment details, locate the cloud machine name.

You will include the syntax for the cloud machine name in the URL of your REST task.

f To find the binding expression for the output property of the cloud template task, click **VIEW OUTPUT JSON**, search for the address property, and locate the cloud machine IP address.

The binding expression appears below the property and search icon in the JSON output.



Stage0.Task0.output

```
    "resources": {
        "Cloud_Machine_1[0]": {
            "id": "/resources/compute/f5a846f3-c97c-4145-9e28-951c36bd721c",
            "name": "Cloud_Machine_1[0]",
            "powerState": "ON",
            "address": "10.108.79.51",
            "resourceLink": "/resources/compute/f5a846f3-c97c-4145-9e28-951c36bd721c",
            "componentTypeId": "Cloud.vSphere.Machine",
            "endpointType": "vsphere",
            "resourceName": "Cloud_Machine_1-mcm187515-152919380820",
            "resourceId": "f5a846f3-c97c-4145-9e28-951c36bd721c",
            "resourceDescLink": "/resources/compute-descriptions/fcb270b0-34bd-4b27-bc4b-7bfcc78bed23",
            "zone": "Automation / [redacted]",
            "countIndex": "0",
            "image": "ubuntu",
            "count": "1",
            "flavor": "small"
```

Path finder: address

${Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].address}

${Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].networks[0].address}

The address resource property displays the cloud machine IP address. For example:

```
"resources": {
        "Cloud_Machine_1[0]": {
            "name": "Cloud_Machine_1[0]",
            "powerState": "ON",
            "address": "10.108.79.51",
            "resourceName": "Cloud_Machine_1-mcm187515-152919380820"
```

2 Return to your pipeline model, and enter the URL in your REST task.

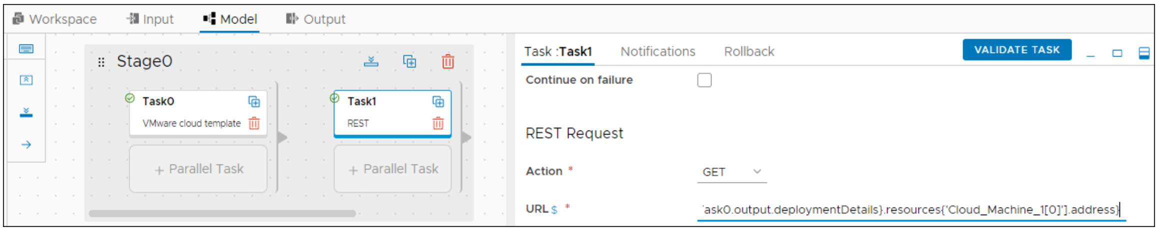
a Click **Actions > View Pipeline**.

b Click the REST task.

c In the REST Request URL area, enter **$**, select the **Stage**, **Task**, **output**, **deploymentDetails**, and enter **`resources`**.

The ability to type ahead with auto fill is available up to the point that you must enter **`resources`**.

d Enter the rest of the cloud machine resource from the deployment details as: **`{'Cloud_Machine_1[0]'].address}`**



Continue on failure ☐

REST Request

Action * GET

URL $ * 'ask0.output.deploymentDetails}.resources{'Cloud_Machine_1[0]'].address}

For the cloud machine entry, you must use the square bracket notation as shown.

The complete URL format is: `${Stage0.Task0.output.deploymentDetails.resources{'Cloud_Machine_1[0]'].address}`

3 Run your pipeline and watch the REST task use the cloud machine and IP address from the output of your cloud template task as the URL to test.

Results

Congratulations! You found the cloud machine name and IP address in the deployment details and JSON output of a cloud template task, and used them to bind your cloud template task output to your REST task URL input in your pipeline.

What to do next

Continue to explore using binding variables from resources in the cloud template task with other tasks in your pipeline.

How do I use a REST API to integrate Code Stream with other applications

Code Stream provides a REST plug-in, which allows you to integrate Code Stream with other applications that use a REST API so that you can continuously develop and deliver software applications that must interact with each other. The REST plug-in invokes an API, which sends and receives information between Code Stream and another application.

With the REST plug-in, you can:

- Integrate external REST API-based systems into a Code Stream pipeline.
- Integrate a Code Stream pipeline as part of the flow of external systems.

The REST plug-in works with any REST API, and supports GET, POST, PUT, PATCH, and DELETE methods to send or receive information between Code Stream and other applications.

Table 5-7. Preparing a pipeline to communicate over the REST API

What you do	What happens as a result
Add a REST task to your pipeline.	The REST task communicates information between applications, and can provide status information for a successive task in the pipeline stage.
In the REST task, select the REST action and include the URL.	The pipeline task calls the URL when the pipeline runs. For POST, PUT, and PATCH actions, you must include a payload. In the payload, you can bind your pipeline and task properties when the pipeline runs.
Consider this example.	Example use of the REST plug-in: You can add a REST task to create a tag on a Git commit for a build, and have the task post a request to get the check-in ID from the repository. The task can send a payload to your repository and create a tag for the build, and the repository can return the response with the tag.

Similar to using the REST plug-in to invoke an API, you can include a Poll task in your pipeline to invoke a REST API and poll it until it completes and the pipeline task meets the exit criteria.

You can also use REST APIs to import and export a pipeline, and use the example scripts to run a pipeline.

This procedure gets a simple URL.

Procedure

1 To create a pipeline, click **Pipelines > New Pipeline > Blank Canvas**.

2 In your pipeline stage, click **+ Sequential Task**.

3 In the task pane, add the REST task:

a Enter a name for the task.

b In the Type drop-down menu, select **REST**.

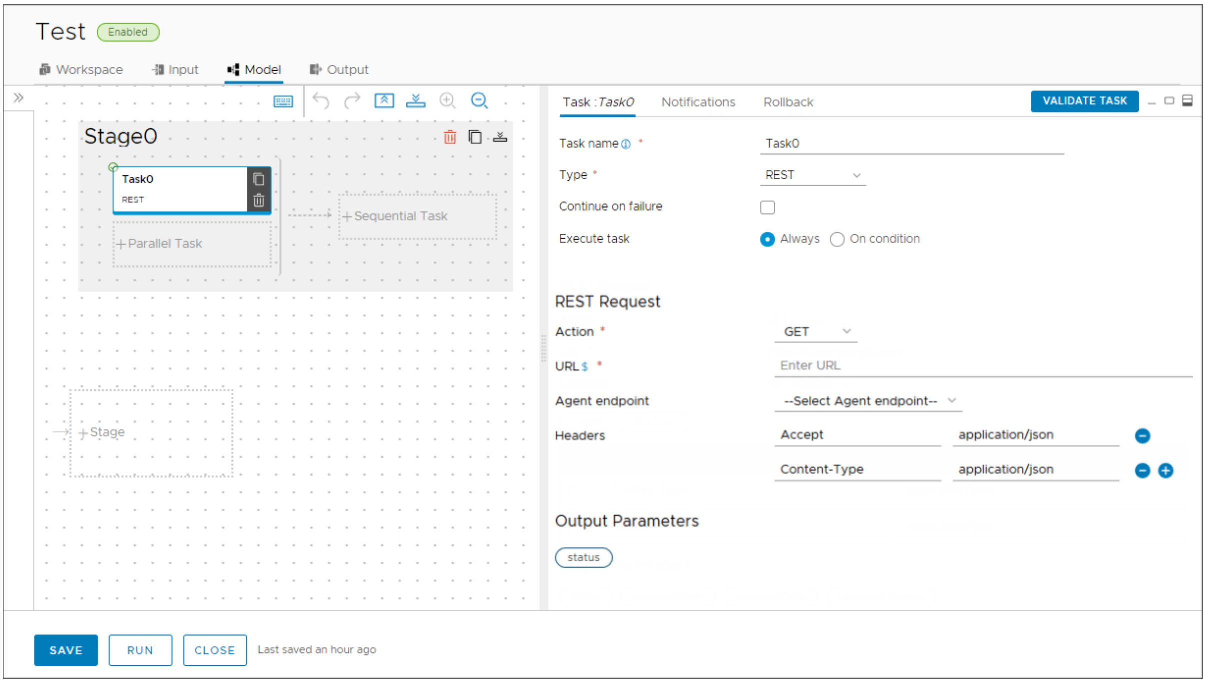
c In the REST Request area, select **GET**.

To have the REST task request data from another application, you select the GET method. To send data to another application, you select the POST method.

d Enter the URL that identifies the REST API endpoint. For example, `https://www.google.com`.

For a REST task to import data from another application, you can include the payload variable. For example, for an import action, you can enter `${Stage0.export.responseBody}`. If the response data size exceeds 5 MB, the REST task might fail.

e To provide authorization for the task, click **Add Headers** and enter a header key and value.



Task name * Task0

Type * REST

Continue on failure □

Execute task ● Always ○ On condition

REST Request

Action * GET

URL * Enter URL

Agent endpoint --Select Agent endpoint--

Headers

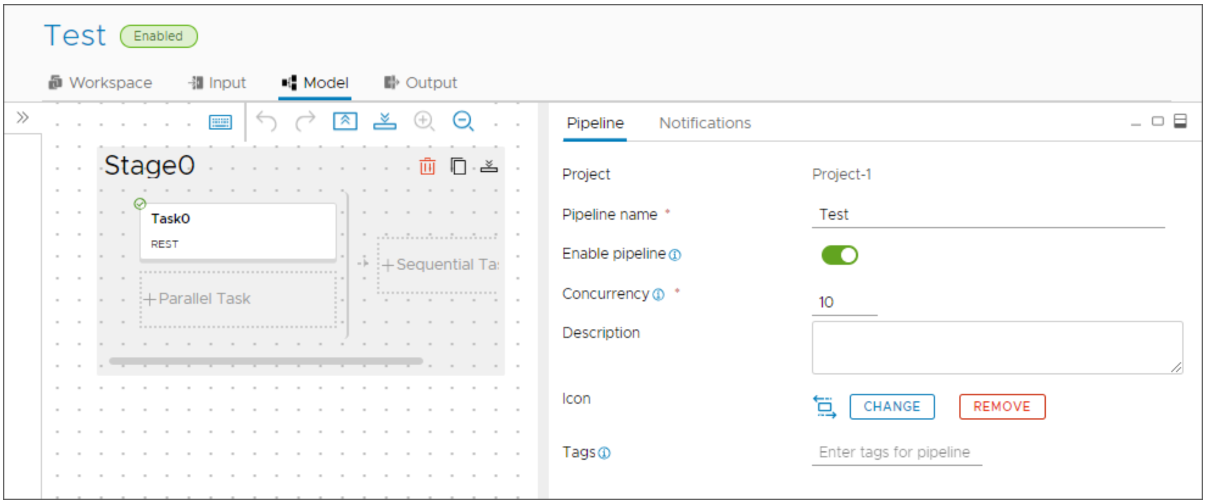
Accept	application/json
Content-Type	application/json

Output Parameters

status

4 To save your pipeline, click **Save**.

5 On the pipeline tab, click **Enable pipeline**.



Project Project-1

Pipeline name * Test

Enable pipeline

Concurrency * 10

Description

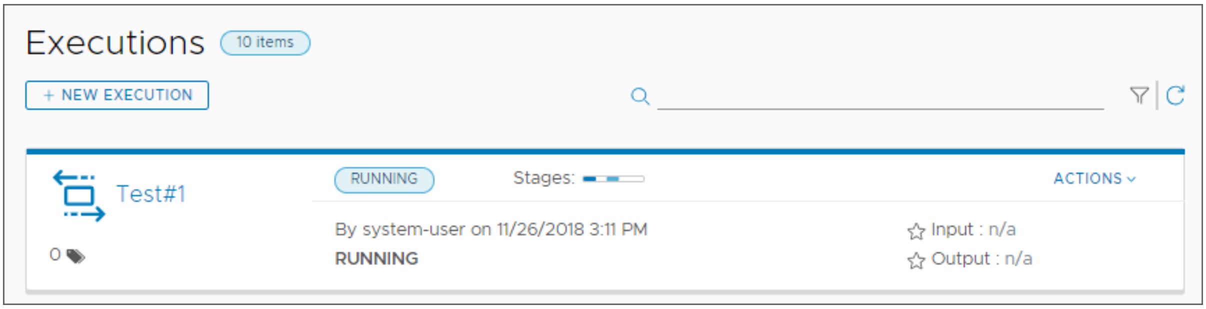
Icon

Tags Enter tags for pipeline

6 Click **Save**, then click **Close**.

7 Click **Run**.

8 To watch the pipeline run, click **Executions**.



Executions 10 items

+ NEW EXECUTION

Test#1 RUNNING Stages: ACTIONS

0

By system-user on 11/26/2018 3:11 PM

RUNNING

☆ Input : n/a

☆ Output : n/a

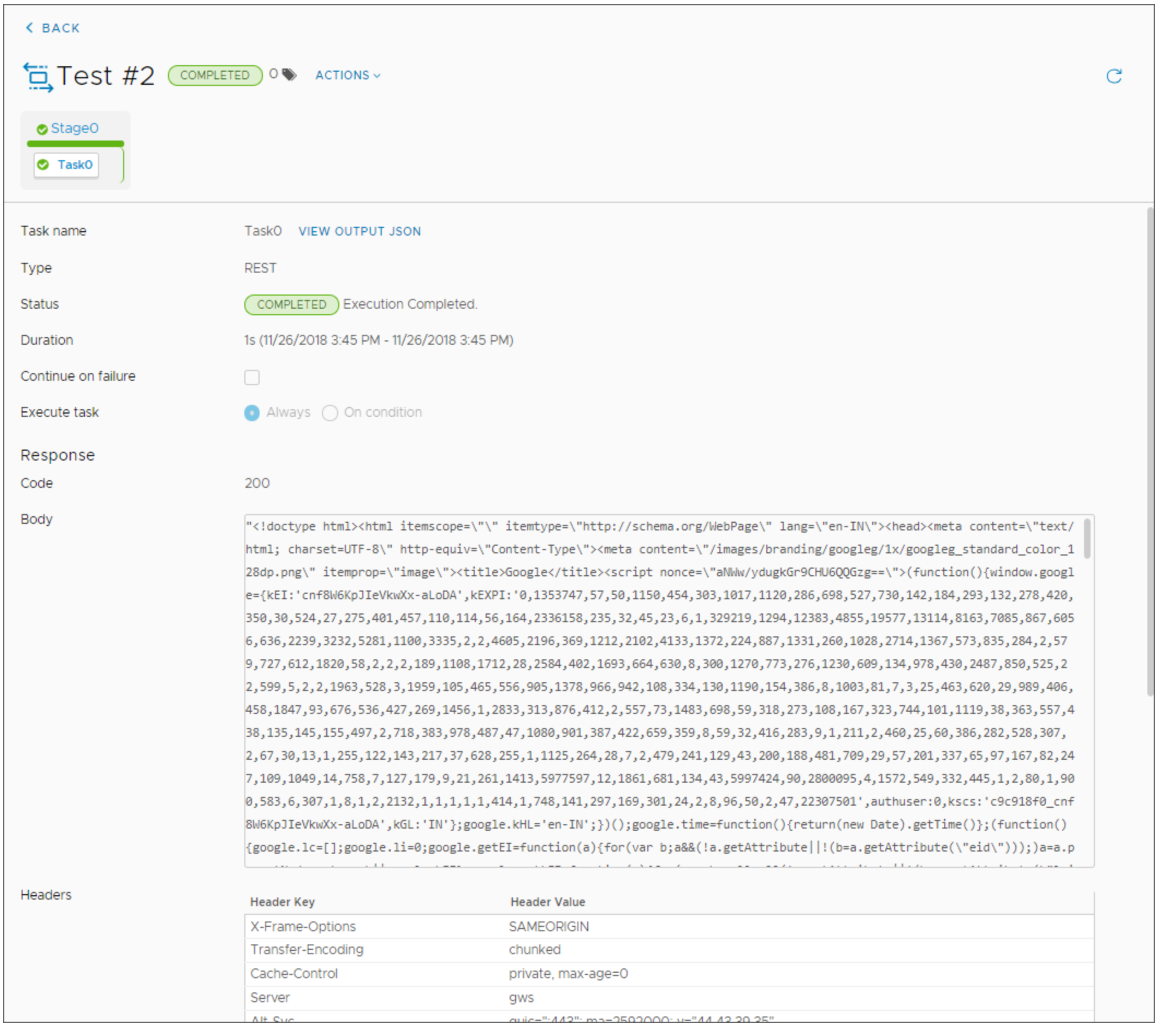
9 To verify that the REST task returns the information you expect, examine the pipeline execution and the task results.

   a After the pipeline completes, to confirm that the other application returned the data you requested, click the link to the pipeline execution.

   b Click the REST task in the pipeline.

   c In the pipeline execution, click the task, observe the task details, and verify that the REST task returned the expected results.

      The task details display the response code, body, header keys, and values.



Task name Task0 VIEW OUTPUT JSON

Type REST

Status COMPLETED Execution Completed.

Duration 1s (11/26/2018 3:45 PM - 11/26/2018 3:45 PM)

Continue on failure □

Execute task ● Always ○ On condition

Response

Code 200

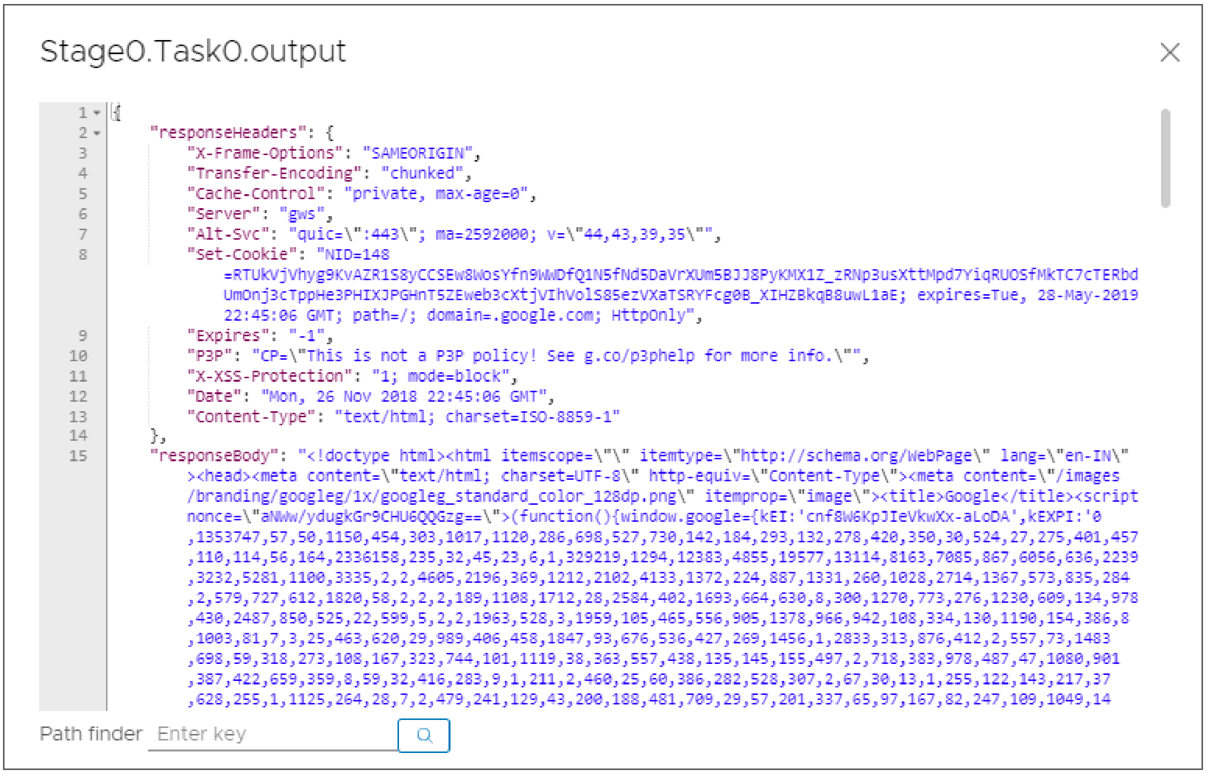
Body

```
"<!doctype html><html itemscope=\"\" itemtype=\"http://schema.org/WebPage\" lang=\"en-IN\"><head><meta content=\"text/
html; charset=UTF-8\" http-equiv=\"Content-Type\"><meta content=\"/images/branding/googleg/1x/googleg_standard_color_1
28dp.png\" itemprop=\"image\"><title>Google</title><script nonce=\"aNWw/ydugkGr9CHU6QQGzg==\">(function(){window.googl
e={kEI:'cnf8W6KpJIeVkwXx-aLoDA',kEXPI:'0,1353747,57,50,1150,454,303,1017,1120,286,698,527,730,142,184,293,132,278,420,
350,30,524,27,275,401,457,110,114,56,164,2336158,235,32,45,23,6,1,329219,1294,12383,4855,19577,13114,8163,7085,867,605
6,636,2239,3232,5281,1100,3335,2,2,4605,2196,369,1212,2102,4133,1372,224,887,1331,260,1028,2714,1367,573,835,284,2,57
9,727,612,1820,58,2,2,2,189,1108,1712,28,2584,402,1693,664,630,8,300,1270,773,276,1230,609,134,978,430,2487,850,525,2
2,599,5,2,2,1963,528,3,1959,105,465,556,905,1378,966,942,108,334,130,1190,154,386,8,1003,81,7,3,25,463,620,29,989,406,
458,1847,93,676,536,427,269,1456,1,2833,313,876,412,2,557,73,1483,698,59,318,273,108,167,323,744,101,1119,38,363,557,4
38,135,145,155,497,2,718,383,978,487,47,1080,901,387,422,659,359,8,59,32,416,283,9,1,211,2,460,25,60,386,282,528,307,
2,67,30,13,1,255,122,143,217,37,628,255,1,1125,264,28,7,2,479,241,129,43,200,188,481,709,29,57,201,337,65,97,167,82,24
7,109,1049,14,758,7,127,179,9,21,261,1413,5977597,12,1861,681,134,43,5997424,90,2800095,4,1572,549,332,445,1,2,80,1,90
0,583,6,307,1,8,1,2,2132,1,1,1,1,1,414,1,748,141,297,169,301,24,2,8,96,50,2,47,22307501',authuser:0,kscs:'c9c918f0_cnf
8W6KpJIeVkwXx-aLoDA',kGL:'IN'};google.kHL='en-IN';})();google.time=function(){return(new Date).getTime()};(function()
{google.lc=[];google.li=0;google.getEI=function(a){for(var b;a&&(!a.getAttribute||!(b=a.getAttribute(\"eid\")));)a=a.p
```

Headers

Header Key	Header Value
X-Frame-Options	SAMEORIGIN
Transfer-Encoding	chunked
Cache-Control	private, max-age=0
Server	gws
Alt-Svc	quic=":443"; ma=2592000; v="44,43,39,35"

10 To see the JSON output, click **VIEW OUTPUT JSON**.



Stage0.Task0.output

```
{
    "responseHeaders": {
        "X-Frame-Options": "SAMEORIGIN",
        "Transfer-Encoding": "chunked",
        "Cache-Control": "private, max-age=0",
        "Server": "gws",
        "Alt-Svc": "quic=\":443\"; ma=2592000; v=\"44,43,39,35\"",
        "Set-Cookie": "NID=148
            =RTUkVjVhyg9KvAZR1S8yCCSEw8WosYfn9WWDfQ1N5fNd5DaVrXUm5BJJ8PyKMX1Z_zRNp3usXttMpd7YiqRUOSfMkTC7cTERbd
            UmOnj3cTppHe3PHIXJPGHnT5ZEweb3cXtjVIhVolS85ezVXaTSRYFcg0B_XIHZBkqB8uwL1aE; expires=Tue, 28-May-2019
            22:45:06 GMT; path=/; domain=.google.com; HttpOnly",
        "Expires": "-1",
        "P3P": "CP=\"This is not a P3P policy! See g.co/p3phelp for more info.\"",
        "X-XSS-Protection": "1; mode=block",
        "Date": "Mon, 26 Nov 2018 22:45:06 GMT",
        "Content-Type": "text/html; charset=ISO-8859-1"
    },
    "responseBody": "<!doctype html><html itemscope=\"\" itemtype=\"http://schema.org/WebPage\" lang=\"en-IN\"
        ><head><meta content=\"text/html; charset=UTF-8\" http-equiv=\"Content-Type\"><meta content=\"/images
        /branding/googleg/1x/googleg_standard_color_128dp.png\" itemprop=\"image\"><title>Google</title><script
        nonce=\"aNWw/ydugkGr9CHU6QQGzg==\">(function(){window.google={kEI:'cnf8W6KpJIeVkwXx-aLoDA',kEXPI:'0
        ,1353747,57,50,1150,454,303,1017,1120,286,698,527,730,142,184,293,132,278,420,350,30,524,27,275,401,457
        ,110,114,56,164,2336158,235,32,45,23,6,1,329219,1294,12383,4855,19577,13114,8163,7085,867,6056,636,2239
        ,3232,5281,1100,3335,2,2,4605,2196,369,1212,2102,4133,1372,224,887,1331,260,1028,2714,1367,573,835,284
        ,2,579,727,612,1820,58,2,2,2,189,1108,1712,28,2584,402,1693,664,630,8,300,1270,773,276,1230,609,134,978
        ,430,2487,850,525,22,599,5,2,2,1963,528,3,1959,105,465,556,905,1378,966,942,108,334,130,1190,154,386,8
        ,1003,81,7,3,25,463,620,29,989,406,458,1847,93,676,536,427,269,1456,1,2833,313,876,412,2,557,73,1483
        ,698,59,318,273,108,167,323,744,101,1119,38,363,557,438,135,145,155,497,2,718,383,978,487,47,1080,901
        ,387,422,659,359,8,59,32,416,283,9,1,211,2,460,25,60,386,282,528,307,2,67,30,13,1,255,122,143,217,37
        ,628,255,1,1125,264,28,7,2,479,241,129,43,200,188,481,709,29,57,201,337,65,97,167,82,247,109,1049,14
```

Results

Congratulations! You configured a REST task that invoked a REST API and sent information between Code Stream and another application by using the REST plug-in.

What to do next

Continue to use REST tasks in your pipelines to run commands and integrate Code Stream with other applications so that you can develop and deliver your software applications. Consider using poll tasks that poll the API until it completes, and the pipeline task meets the exit criteria.

How do I leverage pipeline as code in Code Stream

As a DevOps administrator or developer, you might want to create a pipeline in Code Stream by using YAML code, instead of using the user interface. When you create pipelines as code, you can use any editor and insert comments in the pipeline code.

In your pipeline code, you can refer to external configurations such as environment variables and security credentials. When you update variables that you use in your pipeline code, you can update them without having to update the pipeline code.

You can use the pipeline YAML code as a template to clone and create other pipelines, and share the templates with others.

You can store your pipeline code templates in a source control repository, which versions them and tracks updates. By using a source control system, you can easily back up your pipeline code, and restore it if needed.

Prerequisites

- Verify that you have a code editor.
- If you plan to store your pipeline code in a source control repository, verify that you can access a working instance.

Procedure

1. In your code editor, create a file.
2. Copy and paste the sample pipeline code, and update it to reflect your specific pipeline needs.
3. To include an endpoint to your pipeline code, copy and paste the example endpoint code, and update it to reflect your endpoint.

   When using a Kubernetes API endpoint in the pipeline workspace, Code Stream creates the necessary Kubernetes resources such as ConfigMap, Secret, and Pod to run the continuous integration (CI) task or custom task. Code Stream communicates with the container by using the NodePort.

   The Code Stream pipeline workspace supports Docker and Kubernetes for continuous integration tasks and custom tasks.

   For more information about configuring the workspace, see Configuring the Pipeline Workspace.
4. Save the code.
5. To store and version your pipeline code, check the code into your source control repository.
6. When you create a continuous integration and delivery pipeline, you must import the Kubernetes YAML file.

   To import the Kubernetes YAML file, select it in the Continuous Delivery area of the smart pipeline template, and click **Process**. Or, use the API.

Results

By using the code examples, you created the YAML code that represents your pipeline and endpoints.

Example: Example YAML code for a pipeline and endpoints

This example YAML code includes sections that represent the workspace for the Code Stream native build, stages, tasks, notifications, and more in a pipeline.

For examples of code for supported plug-ins, see Chapter 6 Connecting Code Stream to endpoints

```
---
kind: PIPELINE
name: myPipelineName
tags:
  - tag1
  - tag2


# Ready for execution
enabled: false

#Max number of concurrent executions
concurrency: 10

#Input Properties
input:
  input1: '30'
  input2: 'Hello'

#Output Properties
output:
  BuildNo: '${Dev.task1.buildNo}'
  Image: '${Dev.task1.image}'

#Workspace Definition
ciWorkspace:
  image: docker:maven-latest
  path: /var/tmp
  endpoint: my-k8s
  cache:
    - ~/.m2

# Starred Properties
starred:
  input: input1
  output: output1

# Stages in order of execution
stageOrder:
  - Dev
  - QA
  - Prod

# Task Definition Section
stages:
  Dev:
    taskOrder:
      - Task1, Task6
      - Task2 Long, Task Long Long
      - Task5
    tasks:
      Task1:
        type: jenkins
        ignoreFailure: false
        preCondition: ''
        endpoints:
          jenkinsServer: myJenkins
        input:
          job: Add Two Numbers
          parameters:
            number1: 10
            number2: 20
      Task2:
        type: blah
        # repeats like Task1 above
  QA:
    taskOrder:
      - TaskA
      - TaskB
    tasks:
      TaskA:
        type: ssh
        ignoreFailure: false
        preCondition: ''
        input:
          host: x.y.z.w
          username: abcd
          password: ${var.mypassword}
          script: >
            echo "Hello, remote server"
      TaskB:
        type: blah
        # repeats like TaskA above

# Notificatons Section
notifications:
  email:
    - stage: Dev #optional ; if not found - use pipeline scope
      task: Task1 #optional; if not found use stage scope
      event: SUCCESS
      endpoint: default
      to:
         - user@yourcompany.com
         - abc@yourcompany.com
      subject: 'Pipeline ${name} has completed successfully'
      body: 'Pipeline ${name} has completed successfully'

  jira:
    - stage: QA #optional ; if not found - use pipeline scope
      task: TaskA #optional; if not found use stage scope
      event: FAILURE
      endpoint: myJiraServer
      issuetype: Bug
      project: Test
      assignee: abc
      summary: 'Pipeline ${name} has failed'
      description: |-
```

```
        Pipeline ${name} has failed
        Reason - ${resultsText}
  webhook:
    - stage: QA #optional ; if not found - use pipeline scope
      task: TaskB #optional; if not found use stage scope
      event: FAILURE
      agent: my-remote-agent
      url: 'http://www.abc.com'
      headers: #requestHeaders: '{"build_no":"123","header2":"456"}'
          Content-Type: application/json
          Accept: application/json
      payload: |-
        Pipeline ${name} has failed
        Reason - ${resultsJson}
---
```

This YAML code represents an example Jenkins endpoint.

```
---
name: My-Jenkins
tags:
- My-Jenkins
- Jenkins
kind: ENDPOINT
properties:
  offline: true
  pollInterval: 15.0
  retryWaitSeconds: 60.0
  retryCount: 5.0
  url: http://urlname.yourcompany.com:8080
description: Jenkins test server
type: your.jenkins:JenkinsServer
isLocked: false
---
```

This YAML code represents an example Kubernetes endpoint.

```
---
name: my-k8s
tags: [
  ]
kind: ENDPOINT
properties:
  kubernetesURL: https://urlname.examplelocation.amazonaws.com
  userName: admin
  password: encryptedpassword
description: ''
type: kubernetes:KubernetesServer
isLocked: false
---
```

What to do next

Run your pipeline, and make any adjustments as needed. See How do I run a pipeline and see results.

Connecting Code Stream to endpoints

6

Code Stream integrates with development tools through plug-ins. Supported plug-ins include Jenkins, Bamboo, vRealize Operations, Bugzilla, Team Foundation Server, Git, and more.

You can also develop your own plug-ins that integrate Code Stream with other development applications.

To integrate Code Stream with Jira, you do not need an external plug-in, because Code Stream includes the Jira ticket creation capability as a notification type. To create Jira tickets on pipeline status, you must add a Jira endpoint.

This chapter includes the following topics:

What are Endpoints in Code Stream

An endpoint is an instance of a DevOps application that connects to Code Stream and provides data for your pipelines to run, such as a data source, repository, or notification system.

Your role in Code Stream determines how you use endpoints.

- Administrators and developers can create, update, delete, and view endpoints.
- Administrators can mark an endpoint as restricted, and run pipelines that use restricted endpoints.
- Users who have the viewer role can see endpoints, but cannot create, update, or delete them.

For more information, see How do I manage user access and approvals in Code Stream.

To connect Code Stream to an endpoint, you add a task in your pipeline and configure it so that it communicates with the endpoint. To verify that Code Stream can connect to the endpoint, click **Validate**. Then, when you run the pipeline, your pipeline task connects to the endpoint to run the task.

For information about the task types that use these endpoints, see What types of tasks are available in Code Stream.

Table 6-1. Endpoints that Code Stream supports

Endpoint	What it provides	Versions supported	Requirements
Bamboo	Creates build plans.	6.9.*	
Docker	Native builds can use Docker hosts for deployment.		When a pipeline includes an image from Docker Hub, you must ensure that the image has `cURL` or `wget` embedded before you run the pipeline. When the pipeline runs, Code Stream downloads a binary file that uses `cURL` or `wget` to run commands.
Docker Registry	Registers container images so that a Docker build host can pull images.	2.7.1	
Gerrit	Connects to a Gerrit server for reviews and trigger	2.14.*	
Git	Triggers pipelines when developers update code and check it in to the repository.	Git Hub Enterprise 2.1.8 Git Lab Enterprise 11.9.12-ee	
Jenkins	Builds code artifacts.	1.6.* and 2.*	
Jira	Creates a Jira ticket when a pipeline task fails.	8.3.*	
Kubernetes	Automates the steps that deploy, scale, and manage containerized applications.	All versions supported for Cloud Assembly 8.4 and later 1.18 for Cloud Assembly 8.3 and prior	When using a Kubernetes API endpoint in the pipeline workspace, Code Stream creates the necessary Kubernetes resources such as ConfigMap, Secret, and Pod to run the continuous integration (CI) task or custom task. Code Stream communicates with the container by using the NodePort. For more information about configuring the workspace, see Configuring the Pipeline Workspace.
PowerShell	Create tasks that run PowerShell scripts on Windows or Linux machines.	4 and 5	
SSH	Create tasks that run SSH scripts on Windows or Linux machines.	7.0	

Table 6-1. Endpoints that Code Stream supports (continued)

Endpoint	What it provides	Versions supported	Requirements
TFS, Team Foundation Server	Manages source code, automated builds, testing, and related activities.	2015 and 2017	
vRealize Orchestrator	Arranges and automates the workflows in your build process.	7.* and 8.*	

Example YAML code for a GitHub endpoint

This example YAML code defines a GitHub endpoint that you can refer to in a Git task.

```
---
name: github-k8s
tags: [
  ]
kind: ENDPOINT
properties:
  serverType: GitHub
  repoURL: https://github.com/autouser/testrepok8s
  branch: master
  userName: autouser
  password: encryptedpassword
  privateToken: ''
description: ''
type: scm:git
isLocked: false
---
```

How do I integrate Code Stream with Jenkins

Code Stream provides a Jenkins plug-in, which triggers Jenkins jobs that build and test your source code. The Jenkins plug-in runs test cases, and can use custom scripts.

To run a Jenkins job in your pipeline, you use a Jenkins server, and add the Jenkins endpoint in Code Stream. Then, you create a pipeline and add a Jenkins task to it.

When you use the Jenkins task and a Jenkins endpoint in Code Stream, you can create a pipeline that supports multi-branch jobs in Jenkins. The multi-branch job includes individual jobs in each branch of a Git repository. When you create pipelines in Code Stream that support multi-branch jobs:

- The Jenkins task can run Jenkins jobs that reside in multiple folders on the Jenkins server.
- You can override the folder path in the Jenkins task configuration so that it uses a different folder path, which overrides the default path defined in the Jenkins endpoint in Code Stream.

- Multi-branch pipelines in Code Stream detect Jenkins job files of type `.groovy` in a Git repository or a GitHub repository, and start creating jobs for each branch that it scans in the repository.
- You can override the default path defined in the Jenkins endpoint with a path provided in the Jenkins task configuration, and run a job and pipeline that is associated with any branch inside a main Jenkins job.

Prerequisites

- Set up a Jenkins server that runs version 1.561 or later.
- Verify that you are a member of a project in Code Stream. If you are not a member, ask a Code Stream administrator to add you as a member of a project. See How do I add a project in Code Stream.
- Verify that a job exists on the Jenkins server so that your pipeline task can run it.

Procedure

1 Add and validate a Jenkins endpoint.

   a Click **Endpoints > New Endpoint**.

   b Select a project, and for the type of endpoint select **Jenkins**. Then, enter a name and a description.

   c If this endpoint is a business-critical component in your infrastructure, enable **Mark as restricted**.

   d Enter the URL for the Jenkins server.

e Enter the user name and password to log in to the Jenkins server. Then, enter the remaining information.

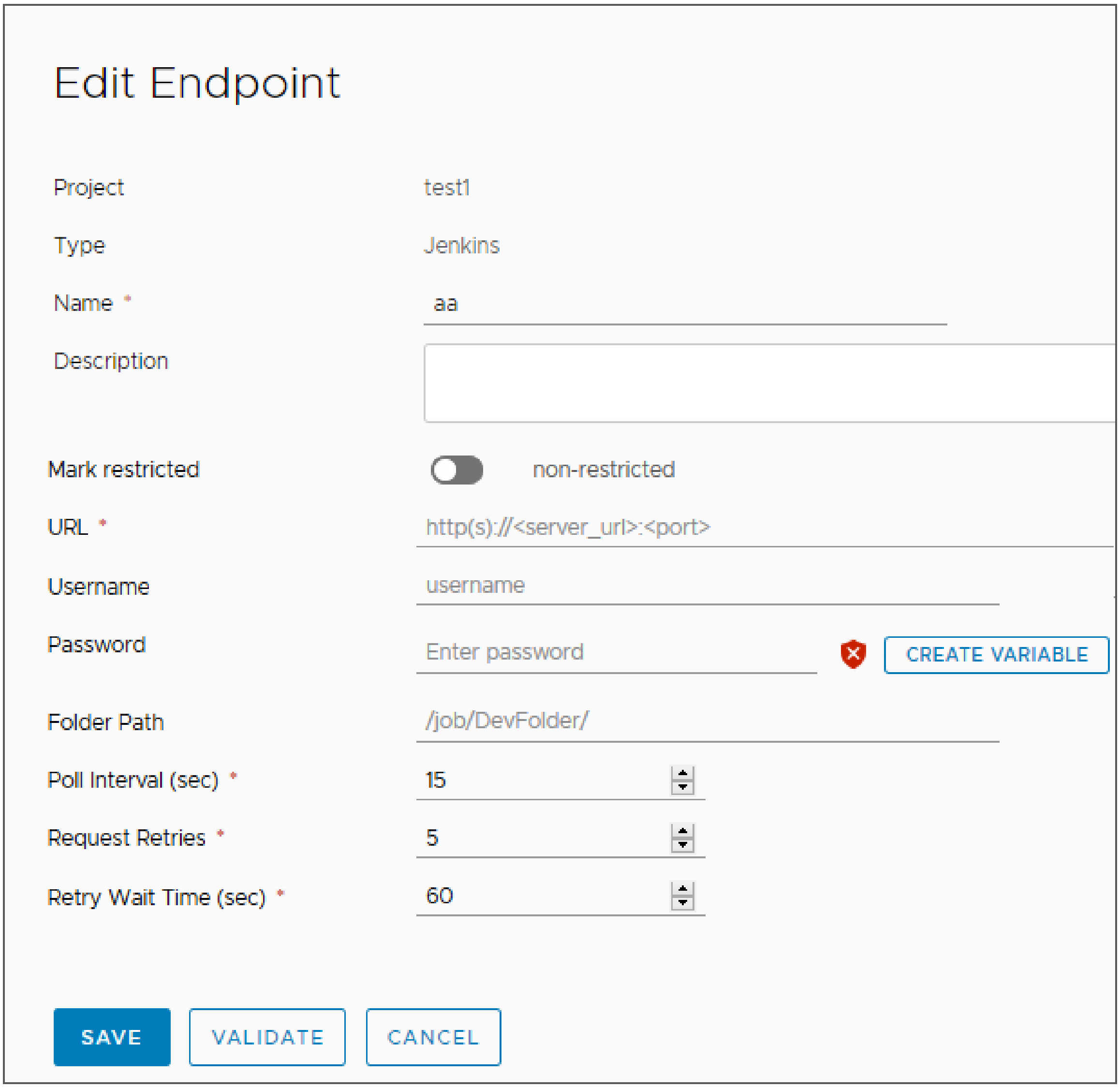
Table 6-2. Remaining information for the Jenkins endpoint

Endpoint entry	Description
Folder Path	Path for the folder that groups your jobs. Jenkins can run all jobs in the folder. You can create sub folders. For example: ■ `folder_1` can include `job_1` ■ `folder_1` can include `folder_2`, which can include `job_2` When you create an endpoint for `folder_1`, the folder path is `job/folder_1`, and the endpoint only lists `job_1`. To obtain the list of jobs in the child folder named `folder_2`, you must create another endpoint that uses the folder path as `/job/folder_1/job/folder_2/`.
Folder Path for multi-branch Jenkins jobs	To support multi-branch Jenkins jobs, in the Jenkins task, you enter the full path that includes the Jenkins server URL and the complete job path. When you include a folder path in the Jenkins task, that path overrides the path that appears in the Jenkins endpoint. With the custom folder path in the Jenkins task, Code Stream only displays jobs that are present in that folder. ■ For example: `https://server.yourcompany.com/job/project` ■ If the pipeline must also trigger the main Jenkins job, use: `https://server.yourcompany.com/job/project/job/main`
URL	Host URL of the Jenkins server. Enter the URL in the form of `protocol://host:port`. For example: `http://192.10.121.13:8080`
Polling Interval	Interval duration for Code Stream to poll the Jenkins server for updates.

Table 6-2. Remaining information for the Jenkins endpoint (continued)

Endpoint entry	Description
Request Retry Count	Number of times to retry the scheduled build request for the Jenkins server.
Retry Wait Time	Number of seconds to wait before retrying the build request for the Jenkins server.

f Click **Validate**, and verify that the endpoint connects to Code Stream. If it does not connect, correct any errors, then click **Save**.



Edit Endpoint

Project	test1
Type	Jenkins
Name *	aa
Description	
Mark restricted	non-restricted
URL *	http(s)://<server_url>:<port>
Username	username
Password	Enter password
Folder Path	/job/DevFolder/
Poll Interval (sec) *	15
Request Retries *	5
Retry Wait Time (sec) *	60

2 To build your code, create a pipeline, and add a task that uses your Jenkins endpoint.

a Click **Pipelines** > **New Pipeline** > **Blank Canvas**.

b Click the default stage.

c In the Task area, enter a name for the task.

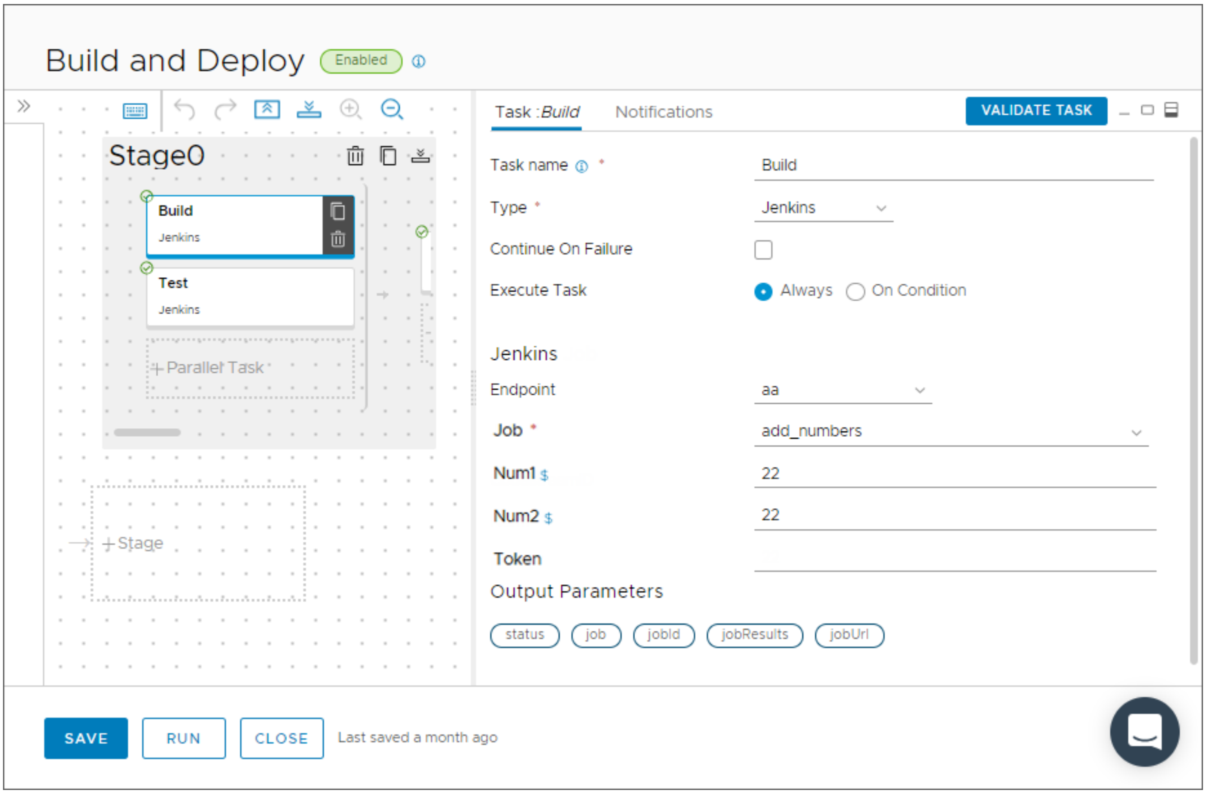
d Select the task type as **Jenkins**.

e Select the Jenkins endpoint that you created.

f From the drop-down menu, select a job from the Jenkins server that your pipeline will run.

g Enter the parameters for the job.

h Enter the authentication token for the Jenkins job.



Build and Deploy Enabled

Stage0

Build
Jenkins

Test
Jenkins

+ Parallel Task

+ Stage

Task : *Build* Notifications VALIDATE TASK

Task name * Build

Type * Jenkins

Continue On Failure □

Execute Task ● Always ○ On Condition

Jenkins

Endpoint aa

Job * add_numbers

Num1 $ 22

Num2 $ 22

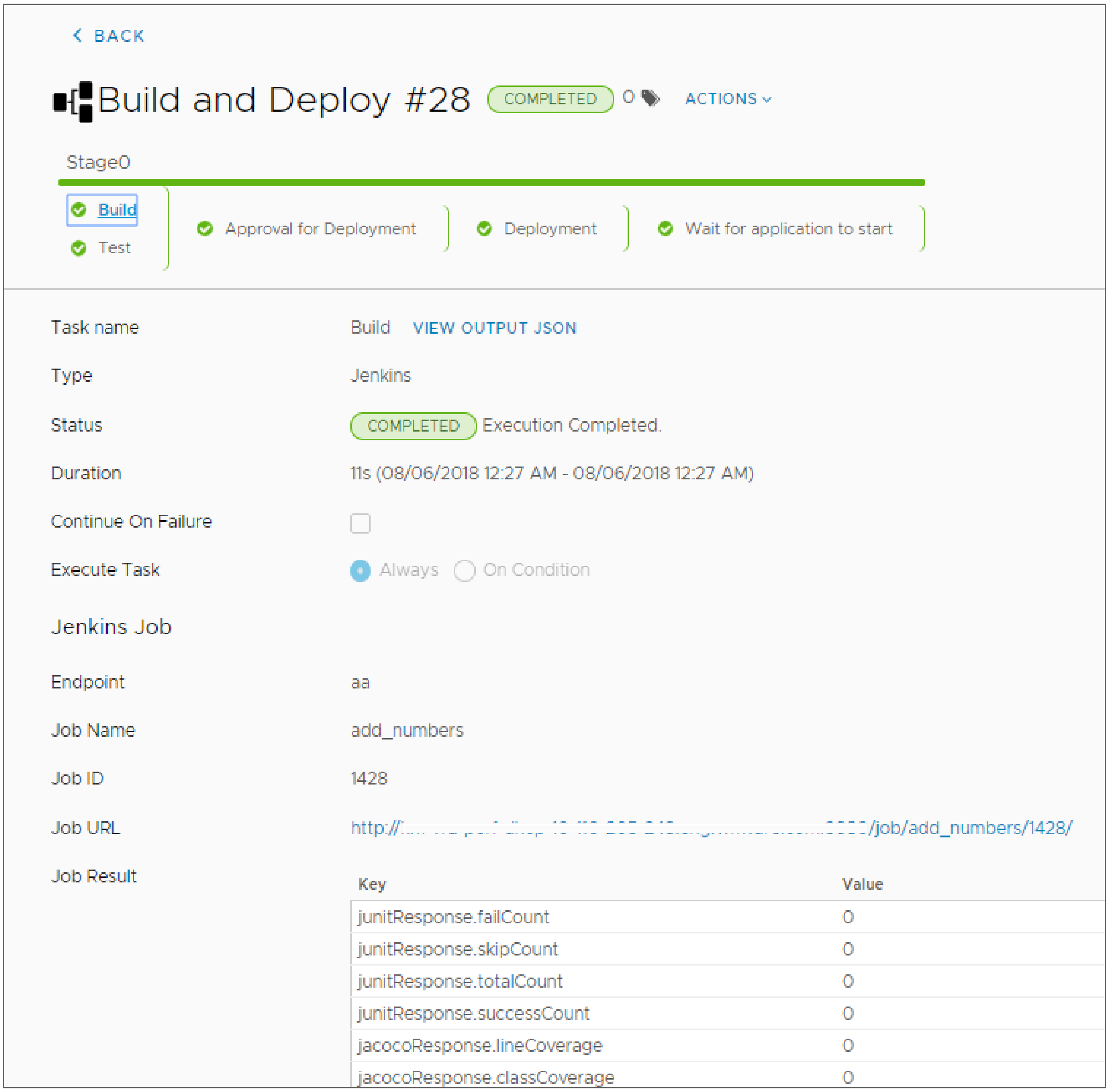
Token

Output Parameters

status job jobId jobResults jobUrl

SAVE RUN CLOSE Last saved a month ago

3 Enable and run your pipeline, and view the pipeline execution.



< BACK

Build and Deploy #28 COMPLETED 0 ACTIONS

Stage0

✓ Build
✓ Test
✓ Approval for Deployment
✓ Deployment
✓ Wait for application to start

Task name	Build VIEW OUTPUT JSON
Type	Jenkins
Status	COMPLETED Execution Completed.
Duration	11s (08/06/2018 12:27 AM - 08/06/2018 12:27 AM)
Continue On Failure	□
Execute Task	● Always ○ On Condition

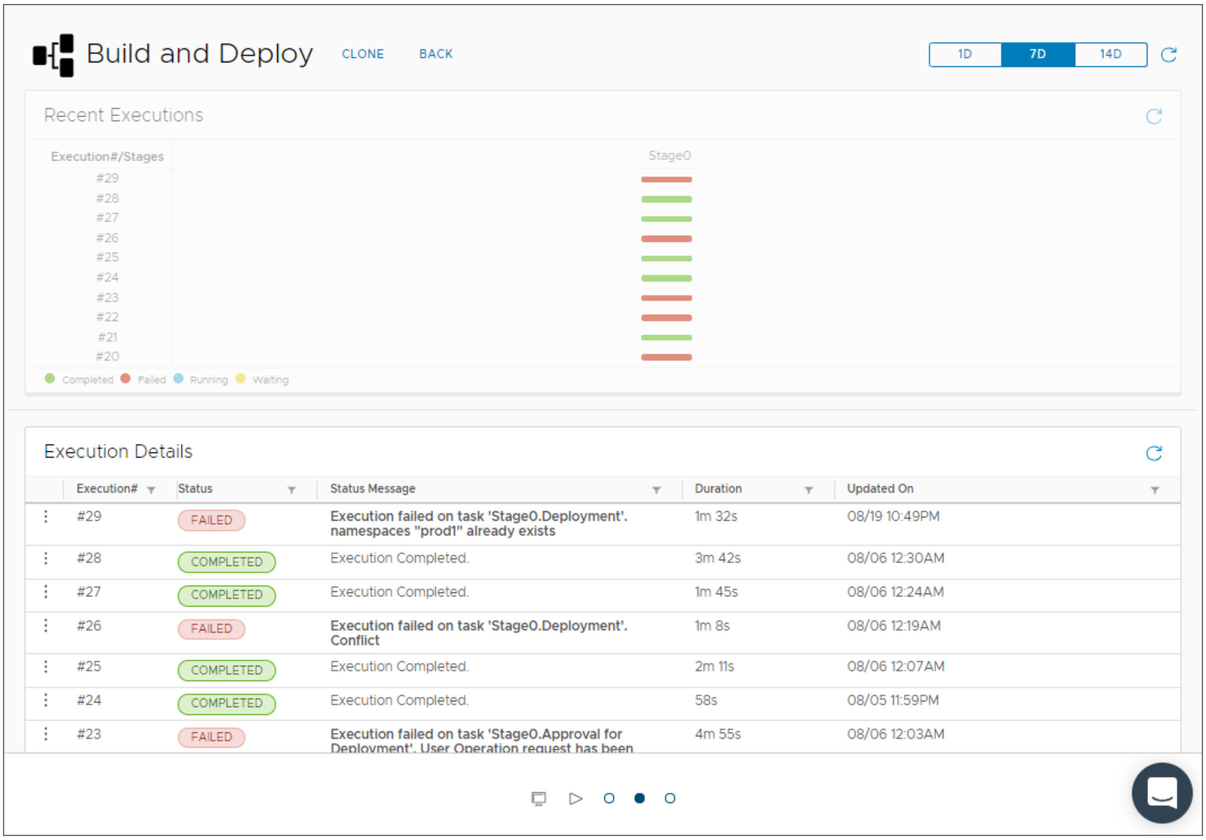
Jenkins Job

Endpoint	aa
Job Name	add_numbers
Job ID	1428
Job URL	http://[illegible]/job/add_numbers/1428/
Job Result	

Key	Value
junitResponse.failCount	0
junitResponse.skipCount	0
junitResponse.totalCount	0
junitResponse.successCount	0
jacocoResponse.lineCoverage	0
jacocoResponse.classCoverage	0

4 Look at the execution details and status on the pipeline dashboard.

You can identify any failures, and why it failed. You can also see trends about the pipeline execution durations, completions, and failures.



Execution Details

	Execution#	Status	Status Message	Duration	Updated On
⋮	#29	FAILED	Execution failed on task 'Stage0.Deployment'. namespaces "prod1" already exists	1m 32s	08/19 10:49PM
⋮	#28	COMPLETED	Execution Completed.	3m 42s	08/06 12:30AM
⋮	#27	COMPLETED	Execution Completed.	1m 45s	08/06 12:24AM
⋮	#26	FAILED	Execution failed on task 'Stage0.Deployment'. Conflict	1m 8s	08/06 12:19AM
⋮	#25	COMPLETED	Execution Completed.	2m 11s	08/06 12:07AM
⋮	#24	COMPLETED	Execution Completed.	58s	08/05 11:59PM
⋮	#23	FAILED	Execution failed on task 'Stage0.Approval for Deployment'. User Operation request has been	4m 55s	08/06 12:03AM

Results

Congratulations! You integrated Code Stream with Jenkins by adding an endpoint, creating a pipeline, and configuring a Jenkins task that builds your code.

Example: Example YAML for a Jenkins build task

For the type of Jenkins build task used in this example, the YAML resembles the following code, with notifications turned on:

```
test:
  type: Jenkins
  endpoints:
    jenkinsServer: jenkins
  input:
    job: Add two numbers
    parameters:
      Num1: '23'
      Num2: '23'
```

What to do next

Review the other sections to learn more. See Chapter 6 Connecting Code Stream to endpoints.

How do I integrate Code Stream with Git

Code Stream provides a way to trigger a pipeline if a code change occurs in your GitHub, GitLab, or Bitbucket repository. The Git trigger uses a Git endpoint on the branch of the repository that you want to monitor. Code Stream connects to the Git endpoint through a webhook.

To define a Git endpoint in Code Stream, you select a project and enter the branch of the Git repository where the endpoint is located. The project groups the pipeline with the endpoint and other related objects. When you choose the project in your webhook definition, you select the endpoint and pipeline to trigger.

Note If you define a webhook with your endpoint and you later edit the endpoint, you cannot change the endpoint details in the webhook. To change the endpoint details, you must delete and redefine the webhook with the endpoint. See How do I use the Git trigger in Code Stream to run a pipeline.

You can create multiple webhooks for different branches by using the same Git endpoint and providing different values for the branch name in the webhook configuration page. To create another webhook for another branch in the same Git repository, you don't need to clone the Git endpoint multiple times for multiple branches. Instead, you provide the branch name in the webhook, which allows you to reuse the Git endpoint. If the branch in the Git webhook is the same as the branch in the endpoint, you don't need to provide branch name in the Git webhook page.

Prerequisites

- Verify that you can access the GitHub, GitLab, or Bitbucket repository to which you plan to connect.
- Verify that you are a member of a project in Code Stream. If you are not, ask a Code Stream administrator to add you as a member of a project. See How do I add a project in Code Stream.

Procedure

1 Define a Git endpoint.

   a Click **Endpoints > New Endpoint**.

   b Select a project, and for the endpoint type select **Git**. Then, enter a name and description.

c If this endpoint is a business-critical component in your infrastructure, enable **Mark as restricted**.

When you use a restricted endpoint in a pipeline, an administrator can run the pipeline and must approve the pipeline execution. If an endpoint or variable is marked as restricted, and a non-administrative user triggers the pipeline, the pipeline pauses at that task, and waits for an administrator to resume it.

A Project administrator can start a pipeline that includes restricted endpoints or variables if these resources are in the project where the user is a Project administrator.

When a user who is not an administrator attempts to run a pipeline that includes a restricted resource, the pipeline stops at the task that uses the restricted resource. Then, an administrator must resume the pipeline.

For more information about restricted resources, and custom roles that include the permission called **Manage Restricted Pipelines**, see:

- How do I manage user access and approvals in Code Stream
- Chapter 2 Setting up Code Stream to model my release process

d Select one of the supported Git server types.

e Enter the URL for the repository with the API gateway for the server in the path. For example:

For GitHub, enter: **`https://api.github.com/vmware-example/repo-example`**

For BitBucket, enter: **`https://api.bitbucket.org/{user}/{repo name}`** or **`http(s)://{bitbucket-enterprise-server}/rest/api/1.0/users/{username}/repos/{repo name}`**

f Enter the branch in the repository where the endpoint is located.

g Select the Authentication type and enter the user name for GitHub, GitLab, or BitBucket. Then enter the private token that goes with the user name.

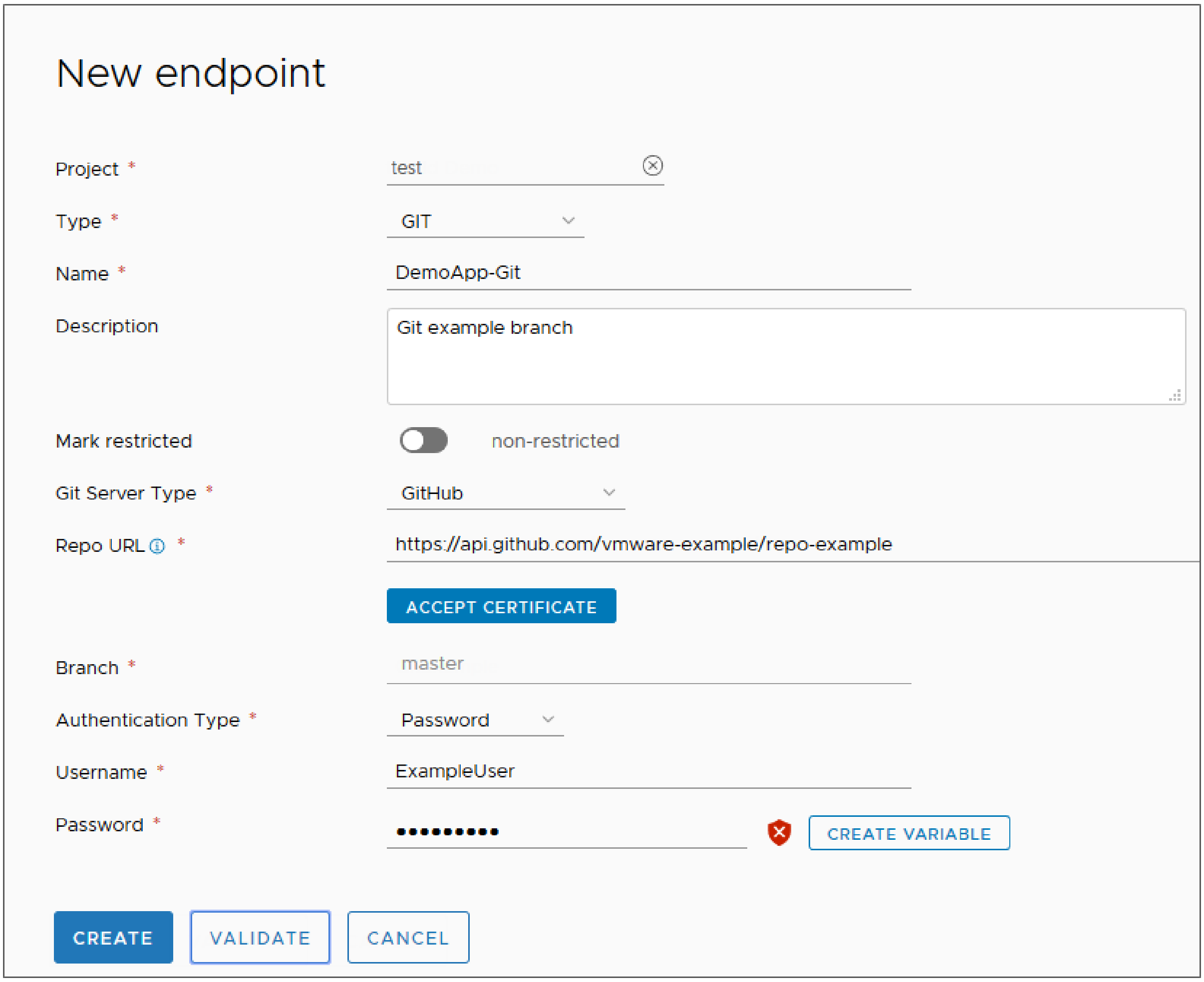
- Password. To create a webhook later, you must enter the private token for the password. Webhooks for Git do not support endpoints created using basic authentication.

  Use secret variables to hide and encrypt sensitive information. Use restricted variable for strings, passwords, and URLs that must be hidden and encrypted, and to restrict use in executions. For example, use a secret variable for a password or URL. You can use secret and restricted variables in any type of task in your pipeline.

- Private token. This token is Git-specific and provides access to a specific action. See https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html. You can also create a variable for the private token.

2 Click **Validate**, and verify that the endpoint connects to Code Stream.

If it does not connect, correct any errors, then click **Create**.



New endpoint

Project *	test
Type *	GIT
Name *	DemoApp-Git
Description	Git example branch
Mark restricted	non-restricted
Git Server Type *	GitHub
Repo URL *	https://api.github.com/vmware-example/repo-example
Branch *	master
Authentication Type *	Password
Username *	ExampleUser
Password *	••••••••••

What to do next

To learn more, review the other sections. See How do I use the Git trigger in Code Stream to run a pipeline.

How do I integrate Code Stream with Gerrit

Code Streamlets you trigger a pipeline when a code review occurs in your Gerrit project. The trigger for Gerrit definition includes the Gerrit project and the pipelines that must run for different event types.

The trigger for Gerrit uses a Gerrit listener on the Gerrit server that you will monitor. To define a Gerrit endpoint in Code Stream, you select a project and enter the URL for the Gerrit server. Then you specify the endpoint when you create a Gerrit listener on that server.

If you are using a Gerrit server as a Code Stream endpoint in a vRealize Automation instance that has FIPS enabled, you must verify that your Gerrit configuration file includes the correct message authentication keys. If the Gerrit server configuration file does not include the correct message authentication keys, the server cannot start up correctly, and displays this message: `PrivateKey/PassPhrase is incorrect`

Prerequisites

- Verify that you can access the Gerrit server to which you plan to connect.
- Verify that you are a member of a project in Code Stream. If you are not a member, ask a Code Stream administrator to add you as a member of a project. See How do I add a project in Code Stream.

Procedure

1 Define a Gerrit endpoint.

   a Click **Configure > Endpoints** and click **New Endpoint**.

   b Select a project, and for the type of endpoint, select **Gerrit**. Then, enter a name and a description.

   c If this endpoint is a business-critical component in your infrastructure, enable **Mark as restricted**.

   d Enter the URL for the Gerrit server.

   You can provide a port number with the URL or leave the value blank to use the default port.

   e Enter a username and password for the Gerrit server.

   If the password must be encrypted, click **Create Variable** and select the type:

   - Secret. The password resolves when a user who has any role runs the pipeline.
   - Restricted. The password resolves when a user who has the Admin role runs the pipeline.

   For the value, enter the password that must be secure, such as the password of a Jenkins server.

   f For the private key, enter the SSH key used to access the Gerrit server securely.

   This key is the RSA private key that resides in the `.ssh` directory.

   g (Optional) If a passphrase is associated with the private key, enter the passphrase.

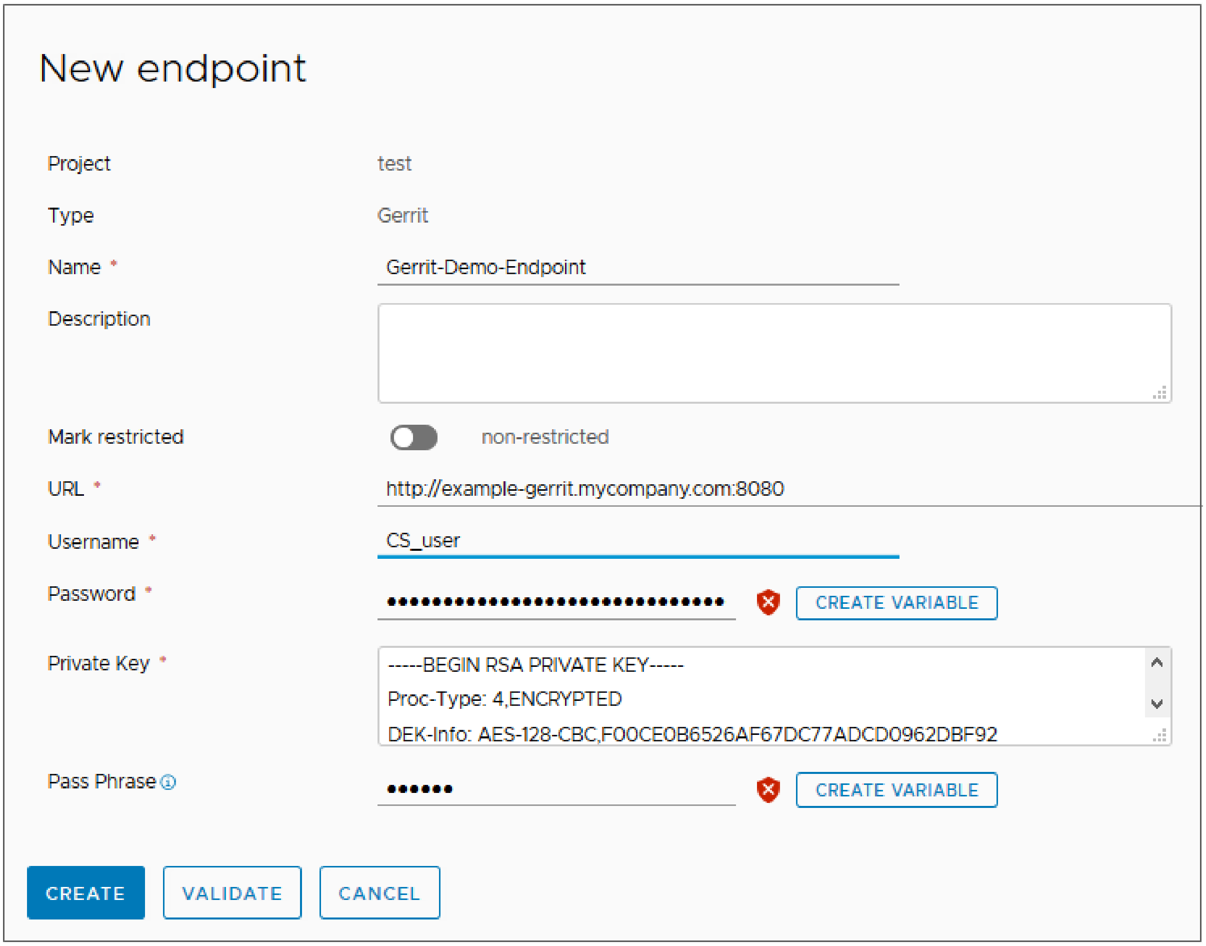
   To encrypt the passphrase, click **Create Variable** and select the type:

   - Secret. The password resolves when a user who has any role runs the pipeline.
   - Restricted. The password resolves when a user who has the Admin role runs the pipeline.

   For the value, enter the passphrase that must be secure, such as the passphrase for an SSH server.

2 Click **Validate**, and verify that the Gerrit endpoint in Code Stream connects to the Gerrit server.

   If it does not connect, correct any errors, then click **Validate** again.



New endpoint

Project	test
Type	Gerrit
Name *	Gerrit-Demo-Endpoint
Description	
Mark restricted	non-restricted
URL *	http://example-gerrit.mycompany.com:8080
Username *	CS_user
Password *	••••••••••••••••••••••••••••••••
Private Key *	-----BEGIN RSA PRIVATE KEY----- Proc-Type: 4,ENCRYPTED DEK-Info: AES-128-CBC,F00CE0B6526AF67DC77ADCD0962DBF92
Pass Phrase	••••••

3 Click **Create**.

4 Verify that the vRealize Automation environment has FIPS enabled, or have your Jenkins job create the environment with FIPS enabled by using the Jenkins URL.

   a To run the command from the command line, connect to your vRealize Automation 8.x appliance over SSH, and log in as the root user. For example, connect to your fully qualified domain name URL, such as `https://cava-1-234-567.yourcompanyFQDN.com` on port 22, 5480, or 443.

   b To check for FIPS on vRealize Automation, run the command `vracli security fips`.

   c Verify that the command returns `FIPS mode: strict`.

5 If your Gerrit server is an endpoint in a vRealize Automation instance that has FIPS enabled, ensure that your Gerrit configuration file includes the correct message authentication (MAC) keys.

   a Open Gerrit and create an SSH key pair.

   b Locate the Gerrit server configuration file at `'$site_path'/etc/gerrit.config`.

   c Verify that the Gerrit server configuration file includes one or more message authentication code (MAC) keys, except for `hmac-MD5`.

   Note In FIPS mode, `hmac-MD5` is not a supported MAC algorithm. To ensure that the Gerrit server starts up correctly, the Gerrit server configuration file must exclude this algorithm. If the Gerrit server does not start up correctly, it displays this message: `PrivateKey/PassPhrase is incorrect`

   Supported message authentication code (MAC) key names that begin with a plus sign (+) are enabled. The MAC key names that begin with a hyphen (-) are removed from the list of default MACs. By default, these supported MACs are available in Code Stream for the Gerrit server:

   - `hmac-md5-96`
   - `hmac-sha1`
   - `hmac-sha1-96`
   - `hmac-sha2-256`
   - `hmac-sha2-512`

What to do next

To learn more, review the other sections. See How do I use the Gerrit trigger in Code Stream to run a pipeline.

How do I integrate Code Stream with vRealize Orchestrator

Code Stream can integrate with vRealize Orchestrator (`vRO`) to extend its capability by running `vRO` workflows. vRealize Orchestrator includes many predefined workflows that can integrate with third-party tools. These workflows help to automate and manage your DevOps processes, automate bulk operations, and more.

For example, you can use a workflow in a `vRO` task in your pipeline to enable a user, remove a user, move VMs, integrate with test frameworks to test your code as the pipeline runs, and much more. You can browse examples of code for vRealize Orchestrator workflows in code.vmware.com.

With a vRealize Orchestrator workflow, your pipeline can run an action as it builds, tests, and deploys your application. You can include predefined workflows in your pipeline, or you can create and use custom workflows. Each workflow includes inputs, tasks, and outputs.

To run a `vRO` workflow in your pipeline, the workflow must appear in the list of available workflows in the `vRO` task that you include in your pipeline.

Before the workflow can appear in the `vRO` task in your pipeline, an administrator must perform the following steps in vRealize Orchestrator:

1. Apply the `CODESTREAM` tag to the `vRO` workflow.
2. Mark the `vRO` workflow as global.

Prerequisites

- Verify that as an administrator you can access an on-premises instance of vRealize Orchestrator. For help, see your own administrator and the vRealize Orchestrator documentation.
- Verify that you are a member of a project in Code Stream. If you are not, ask a Code Stream administrator to add you as a member of a project. See How do I add a project in Code Stream.
- In Code Stream, create a pipeline and add a stage.

Procedure

1 As an administrator, prepare a vRealize Orchestrator workflow for your pipeline to run.

   a. In vRealize Orchestrator, find the workflow that you need to use in your pipeline, such as a workflow to enable a user.

      If you need a workflow that does not exist, you can create it.

   b. In the search bar, enter **`Tag workflow`** to find the workflow named `Tag workflow`.

   c. On the card named `Tag workflow`, click **Run**, which displays the configuration area.

   d. In the `Tagged workflow` text area, enter the name of the workflow to use in your Code Stream pipeline, then select it from the list.

   e. In the `Tag` and `Value` text areas, enter `CODESTREAM` in capital letters.

   f. Click the check box named **Global tag**.

   g. Click **Run**, which attaches the tag named `CODESTREAM` to the workflow that you need to select in your Code Stream pipeline.

   h. In the navigation pane, click **Workflows** and confirm that the tag named `CODESTREAM` appears on the workflow card that your pipeline will run.

      After you log in to Code Stream, and add a `vRO` task to your pipeline, the tagged workflow appears in the workflow list.

2 In Code Stream, create an endpoint for your vRealize Orchestrator instance.

   a. Click **Endpoints > New Endpoint**.

   b. Select a project.

c Enter a relevant name.

d Enter the URL of the vRealize Orchestrator endpoint.

Use this format: **`https://host-n-01-234.eng.vmware.com:8281`**

Do not use this format: https://host-n-01-234.eng.vmware.com:8281/vco/api

The URL for a vRealize Orchestrator instance that is embedded in the vRealize Automation appliance, is the FQDN for the appliance without a port or path. For example: **`https://vra-appliance.yourdomain.local`**

For external vRealize Orchestrator appliances starting with vRealize Automation 8.x, the FQDN for the appliance is **`https://vro-appliance.yourdomain.local`**

If a problem occurs when you add the endpoint, you might need to import a YAML configuration with a SHA-256 certificate fingerprint with the colons removed. For example, **`B0:01:A2:72...`** becomes **`B001A272...`**. The sample YAML code resembles:

~~~~
```
---
project: Demo
kind: ENDPOINT
name: external-vro
description: ''
type: vro
properties:
  url: https://yourVROhost.yourdomain.local
  username: yourusername
  password: yourpassword
  fingerprint: <your_fingerprint>
```
~~~~

For external vRealize Orchestrator appliances included with vRealize Automation 7.x, the FQDN for the appliance is **`https://vro-appliance.yourdomain.local:8281/vco`**

e Click **Accept Certificate** in case the URL that you entered needs a certificate.

f Enter the user name and password for the vRealize Orchestrator server.

If you're using a non-local user for authentication, you must omit the domain part of the user name. For example, to authenticate with **`svc_vro@yourdomain.local`** you must enter **`svc_vro`** in the **Username** text area.

3 Prepare your pipeline to run the `vRO` task.

a Add a `vRO` task to your pipeline stage.

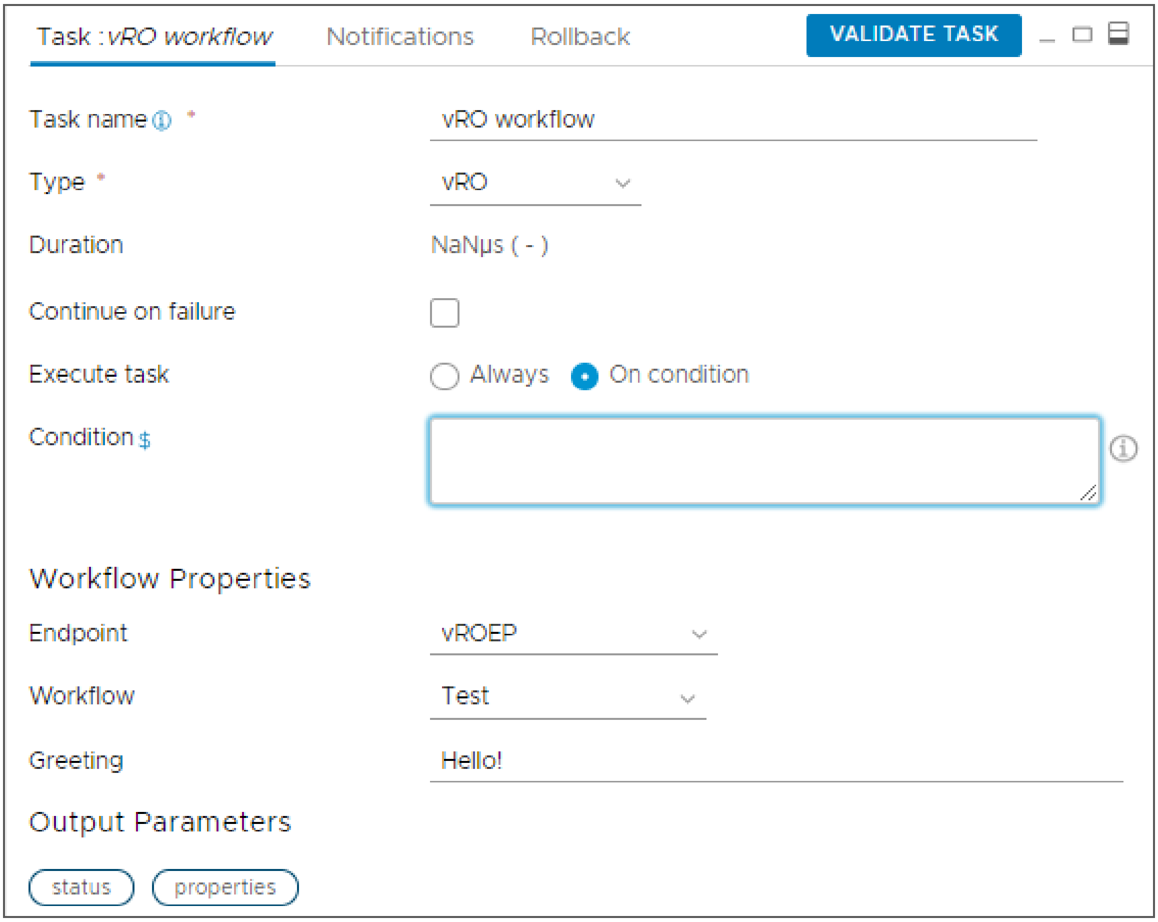
b Enter a relevant name.

c In the Workflow Properties area, select the vRealize Orchestrator endpoint.

d Select the workflow that you tagged as `CODESTREAM` in vRealize Orchestrator.

If you select a custom workflow that you created, you might need to enter the input parameter values.

e For **Execute task**, click **On condition**.



Task name *: vRO workflow

Type *: vRO

Duration: NaNμs (-)

Continue on failure: □

Execute task: ○ Always ● On condition

Condition $:

Workflow Properties

Endpoint: vROEP

Workflow: Test

Greeting: Hello!

Output Parameters

status properties

f Enter the conditions that apply when the pipeline runs.

<table>
<tr><th>When to run pipeline...</th><th>Select conditions...</th></tr>
<tr><td><b>On Condition</b></td><td>Runs the pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped.
The vRO task allows you to include a boolean expression, which uses the following operands and operators.
■ Pipeline variables such as ${pipeline.variableName}. Only use curly brackets when entering variables.
■ Task output variables such as ${Stage1.task1.machines[0].value.hostIp[0]}.
■ Default pipeline binding variables such as ${releasePipelineName}.
■ Case insensitive Boolean values such as, true, false, 'true', 'false'.
■ Integer or decimal values without quotation marks.
■ String values used with single or double quotation marks such as "test", 'test'.
■ String and Numeric types of values such as == Equals and != Not Equals.
■ Relational operators such as >, >=, <, and <=.
■ Boolean logic such as && and ||.
■ Arithmetic operators such as +, -, *, and /.
■ Nested expressions using round brackets.
■ Strings that include the literal value ABCD are evaluated as false, and the task is skipped.
■ Unary operators are not supported.
An example condition might be ${Stage1.task1.output} == “Passed” || ${pipeline.variableName} == 39</td></tr>
<tr><td><b>Always</b></td><td>If you select <b>Always</b>, the pipeline runs the task without conditions.</td></tr>
</table>

g Enter a message for the greeting.

h Click **Validate Task**, and correct any errors that occur.

4 Save, enable, and run your pipeline.

5 After the pipeline runs, examine the results.

a Click **Executions**.

b Click the pipeline.

c Click the task.

d Examine the results, input value, and properties.

You can identify the workflow execution ID, who responded to the task and when, and any comments they included.

Results

Congratulations! You tagged a vRealize Orchestrator workflow for use in Code Stream, and added a `vRO` task in your Code Stream pipeline so that it runs a workflow that automates an action in your DevOps environment.

Example: vRO task output format

The output format for a `vRO` task resembles this example.

```
[{
                "name": "result",
                "type": "STRING",
                "description": "Result of workflow run.",
                "value": ""
},
{
                "name": "message",
                "type": "STRING",
                "description": "Message",
                "value": ""
}]
```

What to do next

Continue to include `vRO` workflow tasks in your pipelines so that you can automate tasks in your development, test, and production environments.

Triggering pipelines in Code Stream

7

You can have Code Stream trigger a pipeline when certain events occur.

For example:

- The Docker trigger can run a pipeline when a new artifact gets created or updated.
- The trigger for Git can trigger a pipeline when developers update code.
- The trigger for Gerrit can trigger a pipeline when developers review code.
- The `curl` command or `wget` command can have Jenkins trigger the pipeline after a build completes.

This chapter includes the following topics:

How do I use the Docker trigger in Code Stream to run a continuous delivery pipeline

As a Code Stream administrator or developer, you can use the Docker trigger in Code Stream. The Docker trigger runs a standalone continuous delivery (CD) pipeline whenever a build artifact is created or updated. The Docker trigger runs your CD pipeline, which pushes the new or updated artifact as a container image to a Docker Hub repository. The CD pipeline can run as part of your automated builds.

For example, to continuously deploy your updated container image through your CD pipeline, use the Docker trigger. When your container image gets checked into the Docker registry, the webhook in Docker Hub notifies Code Stream that the image changed. This notification triggers the CD pipeline to run with the updated container image, and upload the image to the Docker Hub repository.

To use the Docker trigger, you perform several steps in Code Stream.

Table 7-1. How to use the Docker trigger

What you do...	More information about this action...
Create a Docker registry endpoint.	For Code Stream to trigger your pipeline, you must have a Docker Registry endpoint. If the endpoint does not exist, you can select an option that creates it when you add the webhook for the Docker trigger. The Docker registry endpoint includes the URL to the Docker Hub repository.
Add input parameters to the pipeline that auto inject Docker parameters when the pipeline runs.	You can inject Docker parameters into the pipeline. Parameters can include the Docker event owner name, image, repository name, repository namespace, and tag. In your CD pipeline, you include input parameters that the Docker webhook passes to the pipeline before the pipeline triggers.
Create a Docker webhook.	When you create the Docker webhook in Code Stream, it also creates a corresponding webhook in Docker Hub. The Docker webhook in Code Stream connects to Docker Hub through the URL that you include in the webhook. The webhooks communicate with each other, and trigger the pipeline when an artifact is created or updated in Docker Hub. If you update or delete the Docker webhook in Code Stream, the webhook in Docker Hub is also updated or deleted.
Add and configure a Kubernetes task in your pipeline.	When an artifact is created or updated in the Docker Hub repository, the pipeline triggers. Then, it deploys the artifact through the pipeline to the Docker host in your Kubernetes cluster.
Include a local YAML definition in the task.	The YAML definition that you apply to the deployment task includes the Docker container image. If you need to download an image from a privately-owned repository, the YAML file must include a section with the Docker config Secret. See the CD portion of Planning a CICD native build in Code Stream before using the smart pipeline template

When an artifact is created or updated in the Docker Hub repository, the webhook in Docker Hub notifies the webhook in Code Stream, which triggers the pipeline. The following actions occur:

1 Docker Hub sends a POST request to the URL in the webhook.

2 Code Stream runs the Docker trigger.

3 The Docker trigger starts your CD pipeline.

4 The CD pipeline pushes the artifact to the Docker Hub repository.

5 Code Stream triggers its Docker webhook, which runs a CD pipeline that deploys the artifact to your Docker host.

In this example, you create a Docker endpoint and a Docker webhook in Code Stream that deploys your application to your development Kubernetes cluster. The steps include the example code for the payload that Docker posts to the URL in the webhook, the API code that it uses, and the authentication code with the secure token.

Prerequisites

- Verify that a continuous delivery (CD) pipeline exists in your Code Stream instance. Also verify that it includes one or more Kubernetes tasks that deploy your application. See Chapter 4 Planning to natively build, integrate, and deliver your code in Code Stream .
- Verify that you can access an existing Kubernetes cluster where your CD pipeline can deploy your application for development.
- Verify that you are a member of a project in Code Stream. If you are not, ask a Code Stream administrator to add you as a member of a project. See How do I add a project in Code Stream.

Procedure

1 Create a Docker registry endpoint.

   a Click **Endpoints**.

   b Click **New Endpoint**.

   c Start typing name of existing project.

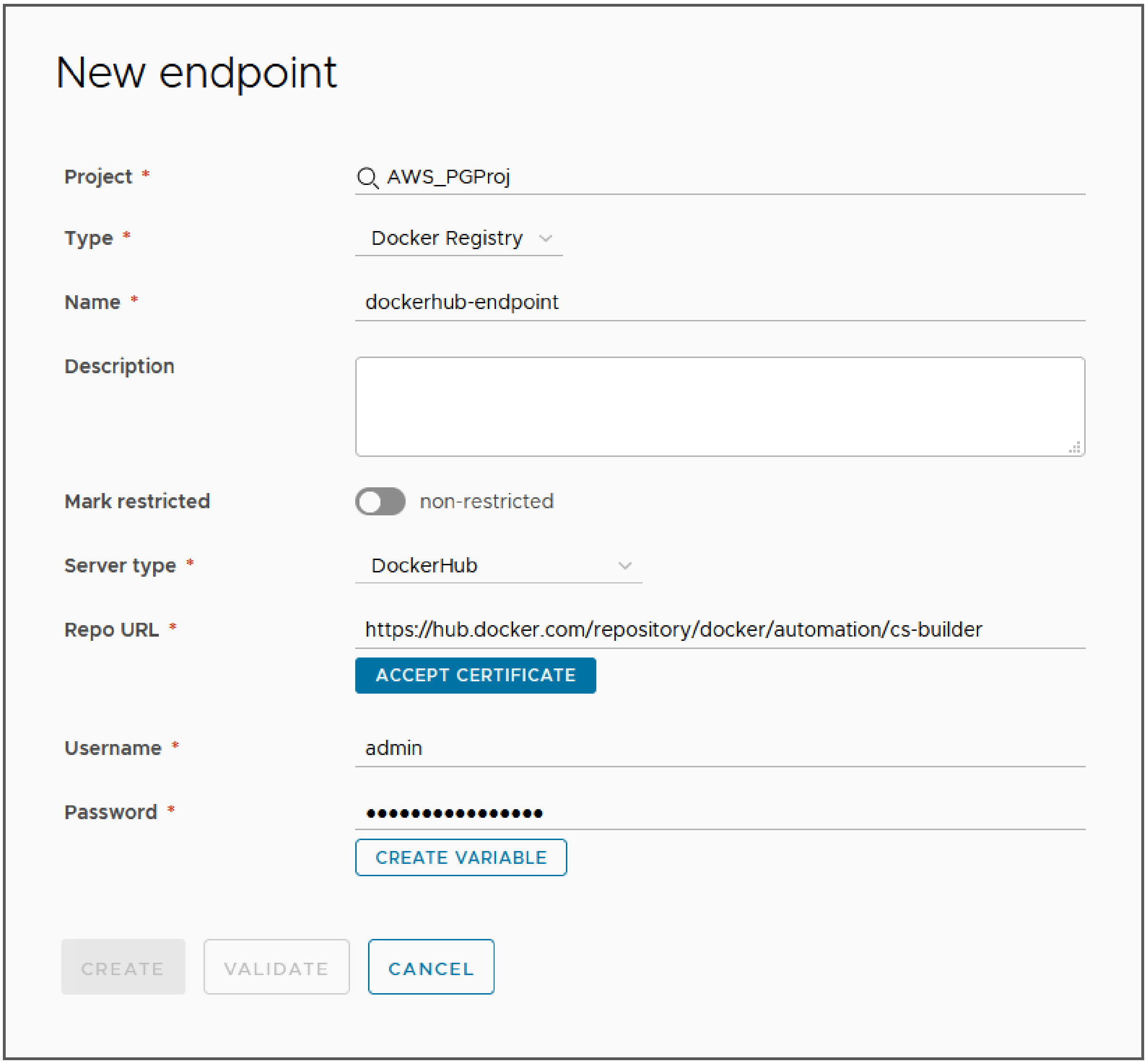
   d Select the type as **Docker Registry**.

   e Enter a relevant name.

   f Select the server type as **DockerHub**.

   g Enter the URL to the Docker Hub repository.

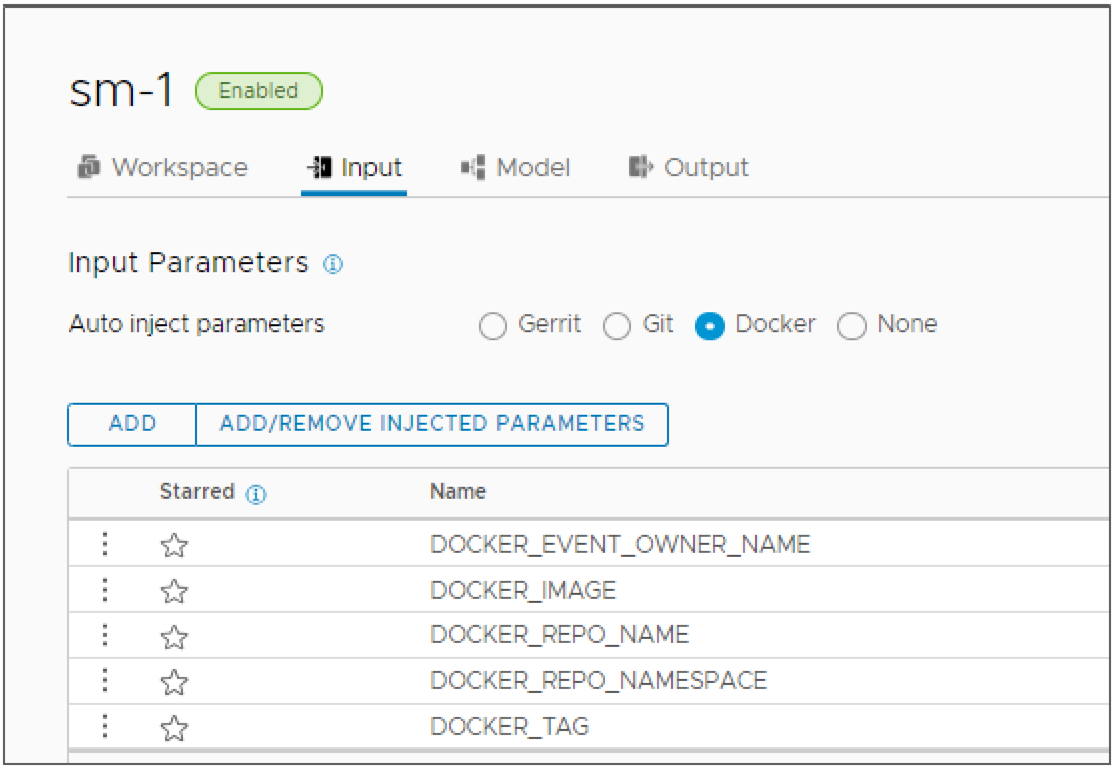
   h Enter the name and password that can access the repository.



New endpoint

Project *	AWS_PGProj
Type *	Docker Registry
Name *	dockerhub-endpoint
Description	
Mark restricted	non-restricted
Server type *	DockerHub
Repo URL *	https://hub.docker.com/repository/docker/automation/cs-builder
Username *	admin
Password *	●●●●●●●●●●●●●●●●

2 In your CD pipeline, set the input properties to auto inject Docker parameters when the pipeline runs.



Input Parameters

Auto inject parameters ○ Gerrit ○ Git ● Docker ○ None

	Starred	Name
⋮	☆	DOCKER_EVENT_OWNER_NAME
⋮	☆	DOCKER_IMAGE
⋮	☆	DOCKER_REPO_NAME
⋮	☆	DOCKER_REPO_NAMESPACE
⋮	☆	DOCKER_TAG

3 Create a Docker webhook.

a Click **Triggers > Docker**.

b Click **New Webhook for Docker**.

c Select a project.

d Enter a relevant name.

e Select your Docker registry endpoint.

If the endpoint does not yet exist, click **Create Endpoint** and create it.

f Select the pipeline with Docker injected parameters for the webhook to trigger. See Step 2.

If the pipeline was configured with custom added input parameters, the Input Parameters list displays parameters and values. You can enter values for input parameters that will be passed to the pipeline with the trigger event. Or you can leave the values blank, or use the default values if defined.

For more information about parameters on the input tab, see How you'll create the CICD pipeline and configure the workspace.

g Enter the API Token.

The CSP API token authenticates you for external API connections with Code Stream. To obtain the API token:

1 Click **Generate Token**.

2 Enter the email address associated with your user name and password and click **Generate**.

The token that you generate is valid for six months. It is also known as a refresh token.

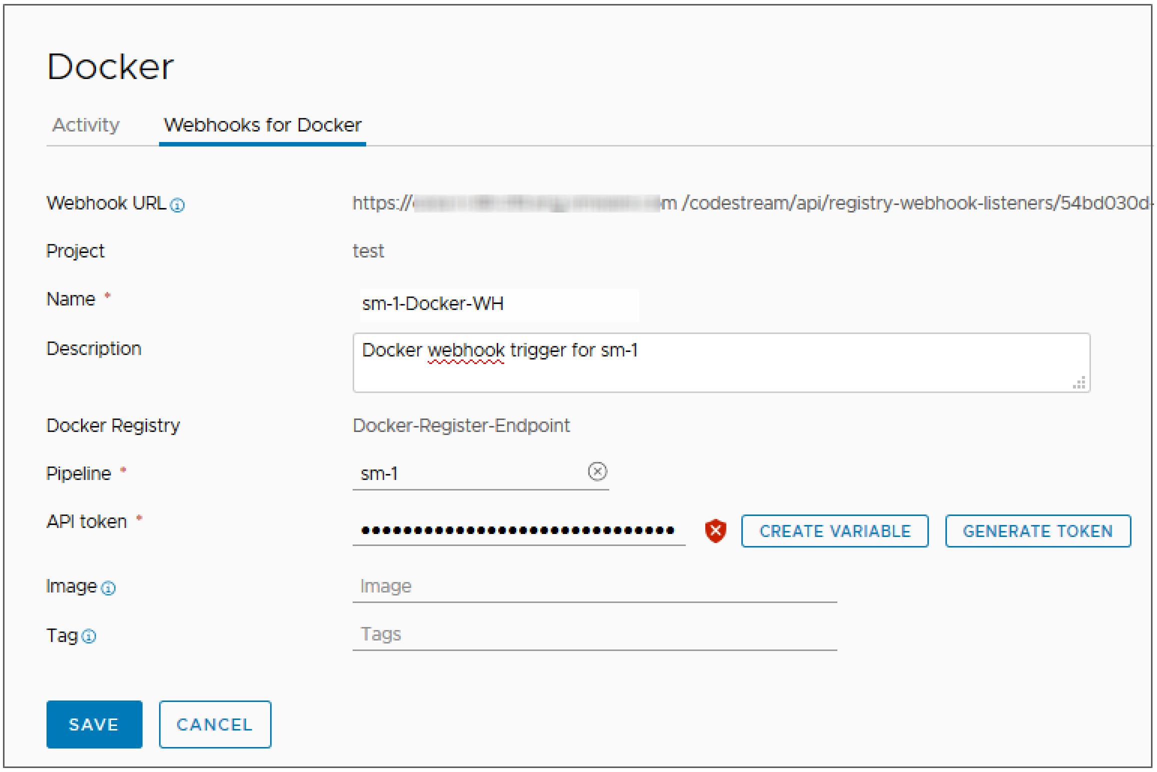
- To keep the token as a variable for future use, click **Create Variable**, enter a name for the variable and click **Save**.
- To keep the token as a text value for future use, click **Copy** and paste the token into a text file to save locally.

You can choose to both create a variable and store the token in a text file for future use.

3 Click **Close**.

h Enter the build image.

i Enter a tag.



Docker

Webhook URL ⓘ	https://[redacted]m /codestream/api/registry-webhook-listeners/54bd030d-
Project	test
Name *	sm-1-Docker-WH
Description	Docker webhook trigger for sm-1
Docker Registry	Docker-Register-Endpoint
Pipeline *	sm-1 ⓧ
API token *	••••••••••••••••••••••••••••••
Image ⓘ	Image
Tag ⓘ	Tags

j Click **Save**.

The webhook card appears with the Docker webhook enabled. If you want to make a dummy push to the Docker Hub repository without triggering the Docker webhook and running a pipeline, click **Disable**.

4 In your CD pipeline, configure your Kubernetes deployment task.

a In the Kubernetes task properties, select your development Kubernetes cluster.

b Select the **Create** action.

c Select the **Local Definition** for the payload source.

d Then select your local YAML file.

For example, Docker Hub might post this local YAML definition as the payload to the URL in the webhook:

```
{
"callback_url": "https://registry.hub.docker.com/u/svendowideit/testhook/hook/
2141b5bi5i5b02bec211i4eeih0242eg11000a/",
"push_data": {
"images": [
"27d47432a69bca5f2700e4dff7de0388ed65f9d3fb1ec645e2bc24c223dc1cc3",
"51a9c7c1f8bb2fa19bcd09789a34e63f35abb80044bc10196e304f6634cc582c",
"..."
],
"pushed_at": 1.417566161e+09,
"pusher": "trustedbuilder",
"tag": "latest"
},
"repository": {
"comment_count": 0,
"date_created": 1.417494799e+09,
"description": "",
"dockerfile": "#\n# BUILD\u0009\u0009docker build -t svendowideit/apt-
cacher .\n# RUN\u0009\u0009docker run -d -p 3142:3142 -name apt-cacher-
run apt-cacher\n#\n# and then you can run containers with:\n#
\u0009\u0009docker run -t -i -rm -e http_proxy http://192.168.1.2:3142/
debian bash\n#\nFROM\u0009\u0009ubuntu\n\n\nVOLUME\u0009\u0009[\/var/cache/apt-cacher-
ng\]\nRUN\u0009\u0009apt-get update ; apt-get install -yq apt-cacher-ng\n\nEXPOSE
\u0009\u00093142\nCMD\u0009\u0009chmod 777 /var/cache/apt-cacher-ng ; /etc/init.d/apt-
cacher-ng start ; tail -f /var/log/apt-cacher-ng/*\n",
"full_description": "Docker Hub based automated build from a GitHub repo",
"is_official": false,
"is_private": true,
"is_trusted": true,
"name": "testhook",
"namespace": "svendowideit",
"owner": "svendowideit",
"repo_name": "svendowideit/testhook",
"repo_url": "https://registry.hub.docker.com/u/svendowideit/testhook/",
"star_count": 0,
"status": "Active"
}
}
```

The API that creates the webhook in Docker Hub uses this form: `https://cloud.docker.com/v2/repositories/%3CUSERNAME%3E/%3CREPOSITORY%3E/webhook_pipeline/`

The JSON code body resembles:

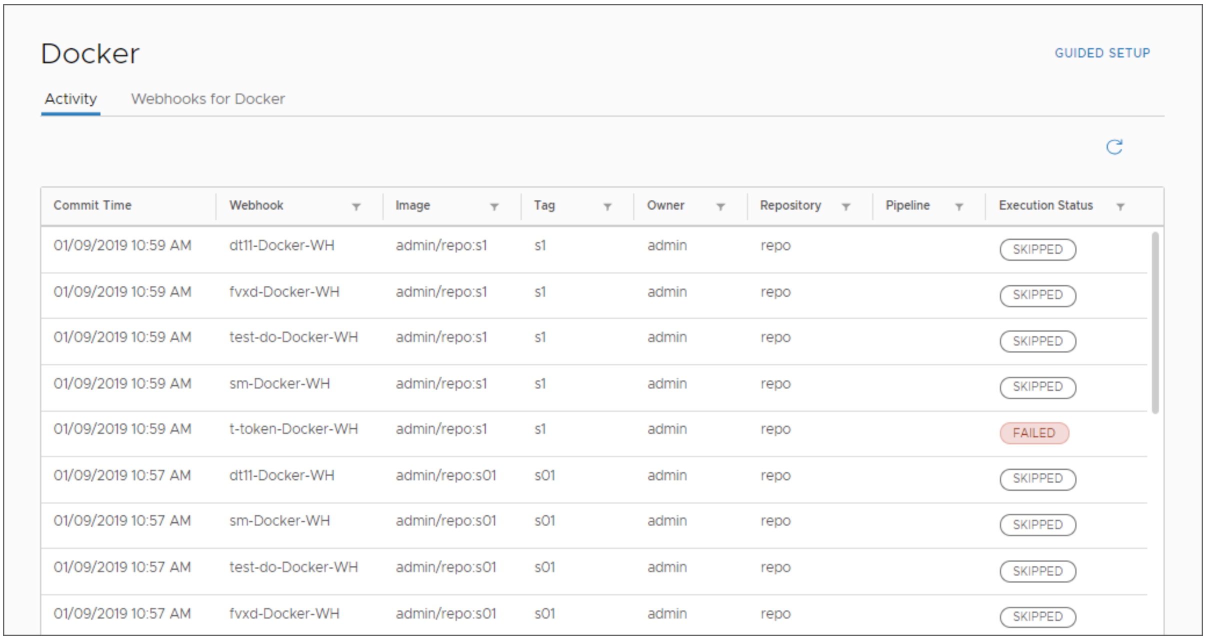
```
{
"name": "demo_webhook",
"webhooks": [
{
"name": "demo_webhook",
"hook_url": "http://www.google.com"
}
]
}
```

To receive events from the Docker Hub server, the authentication scheme for the Docker webhook that you create in Code Stream uses an allowlist authentication mechanism with a random string token for the webhook. It filters events based on the secure token, which you can append to `hook_url`.

Code Stream can verify any request from the Docker Hub server by using the configured secure token. For example: `hook_url = IP:Port/pipelines/api/docker-hub-webhooks?secureToken = ""`

5 Create a Docker artifact in your Docker Hub repository. Or, update an existing artifact.

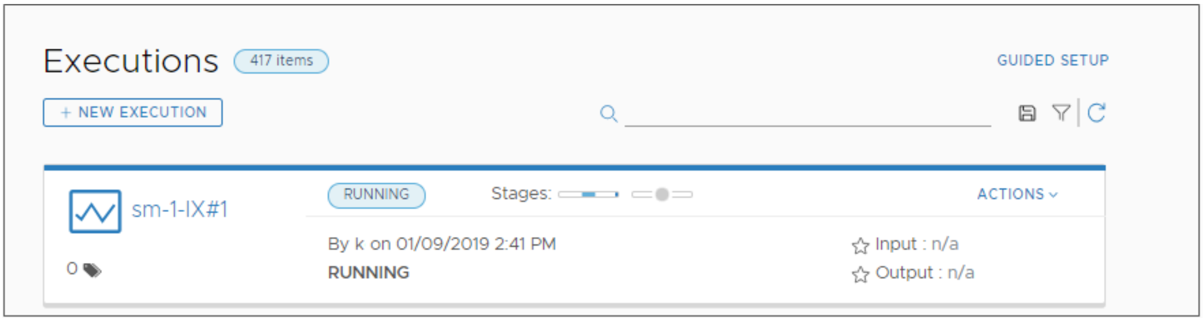
6 To confirm that the trigger occurred, and see the activity on the Docker webhook, click **Triggers** > **Docker** > **Activity**.



Docker

Commit Time	Webhook	Image	Tag	Owner	Repository	Pipeline	Execution Status
01/09/2019 10:59 AM	dt11-Docker-WH	admin/repo:s1	s1	admin	repo		SKIPPED
01/09/2019 10:59 AM	fvxd-Docker-WH	admin/repo:s1	s1	admin	repo		SKIPPED
01/09/2019 10:59 AM	test-do-Docker-WH	admin/repo:s1	s1	admin	repo		SKIPPED
01/09/2019 10:59 AM	sm-Docker-WH	admin/repo:s1	s1	admin	repo		SKIPPED
01/09/2019 10:59 AM	t-token-Docker-WH	admin/repo:s1	s1	admin	repo		FAILED
01/09/2019 10:57 AM	dt11-Docker-WH	admin/repo:s01	s01	admin	repo		SKIPPED
01/09/2019 10:57 AM	sm-Docker-WH	admin/repo:s01	s01	admin	repo		SKIPPED
01/09/2019 10:57 AM	test-do-Docker-WH	admin/repo:s01	s01	admin	repo		SKIPPED
01/09/2019 10:57 AM	fvxd-Docker-WH	admin/repo:s01	s01	admin	repo		SKIPPED

7 Click **Executions**, and observe your pipeline as it runs.



Executions 417 items

sm-1-IX#1 RUNNING Stages:

By k on 01/09/2019 2:41 PM

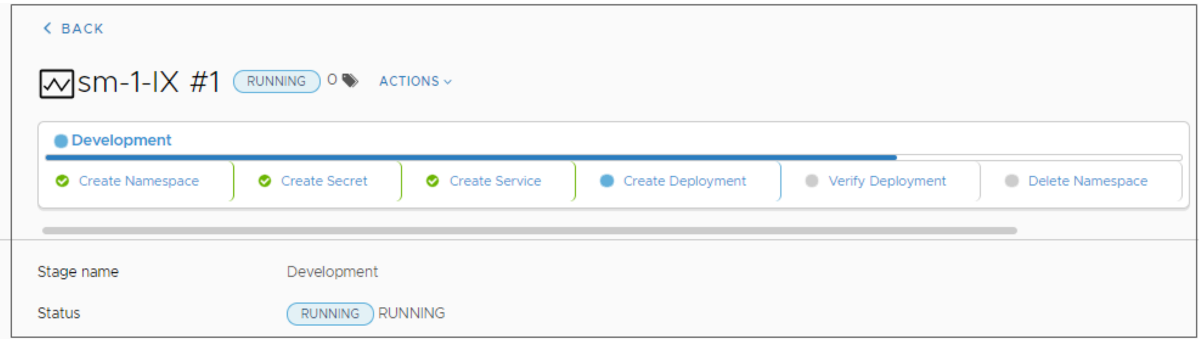
RUNNING

0

Input : n/a

Output : n/a

8 Click the running stage and view the tasks as the pipeline runs.



Stage name Development

Status RUNNING RUNNING

Results

Congratulations! You set up the Docker trigger to run your CD pipeline continuously. Your pipeline can now upload new and updated Docker artifacts to the Docker Hub repository.

What to do next

Verify that your new or updated artifact is deployed to the Docker host in your development Kubernetes cluster.

How do I use the Git trigger in Code Stream to run a pipeline

As a Code Stream administrator or developer, you can integrate integrate Code Stream with the Git lifecycle by using the Git trigger. When you make a code change in GitHub, GitLab, or Bitbucket Enterprise, the event communicates with Code Stream through a webhook and triggers a pipeline. The webhook works with GitLab, GitHub, and Bitbucket on-premises enterprise versions when both Cloud Assembly and the enterprise version are reachable on the same network.

When you add the webhook for Git in Code Stream, it also creates a webhook in the GitHub, GitLab, or the Bitbucket repository. If you update or delete the webhook later, that action also updates or deletes the webhook in GitHub, GitLab, or Bitbucket.

Your webhook definition must include a Git endpoint on the branch of the repository that you will monitor. To create the webhook, Code Stream uses the Git endpoint. If the endpoint does not exist, you can create it when you add the webhook. This example assumes that you have a predefined Git endpoint in GitHub.

Note Your Git endpoint must use a private token for authentication. If your Git endpoint uses a password for basic authentication, you will not be able to create a webhook.

You can create multiple webhooks for different branches by using the same Git endpoint and providing different values for the branch name in the webhook configuration page. To create another webhook for another branch in the same Git repository, instead of cloning the Git endpoint multiple times for multiple branches, you can provide the branch name in the webhook. This approach allows you to reuse the Git endpoint. If the branch in the Git webhook is the same as the branch in the endpoint, you don't need to provide branch name in the Git webhook page.

This example shows you how to use the Git trigger with a GitHub repository, but the prerequisites include preparations required if another Git server type is used.

Prerequisites

- Verify that you are a member of a project in Code Stream. If you are not, ask a Code Stream administrator to add you as a member of a project. See How do I add a project in Code Stream.
- Verify that you have a Git endpoint on the GitHub branch you want to monitor. See How do I integrate Code Stream with Git.
- Verify that you have rights to create a webhook in the Git repository.
- If configuring a webhook in GitLab, change the default network settings in GitLab enterprise to enable outbound requests and allow the creation of local webhooks.

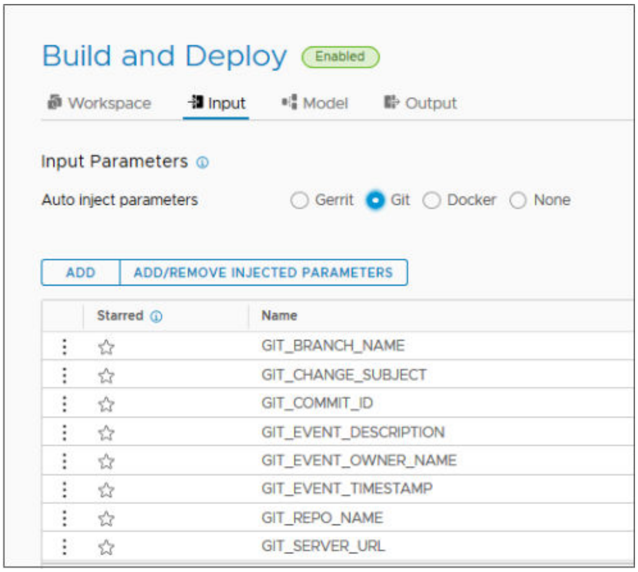
  Note This change is only required for GitLab enterprise. These settings do not apply to GitHub or Bitbucket.

  a Log in to your GitLab enterprise instance as administrator.

  b Go to network settings using a URL such as, `http://{gitlab-server}/admin/application_settings/network`.

  c Expand **Outbound requests** and click:

    - Allow requests to the local network from web hooks and services.
    - Allow requests to the local network from system hook.
- For the pipelines you want to trigger, verify that you have set the input properties to inject Git parameters when the pipeline runs.



Build and Deploy Enabled

Workspace · Input · Model · Output

Input Parameters

Auto inject parameters ○ Gerrit ● Git ○ Docker ○ None

ADD | ADD/REMOVE INJECTED PARAMETERS

	Starred	Name
⋮	☆	GIT_BRANCH_NAME
⋮	☆	GIT_CHANGE_SUBJECT
⋮	☆	GIT_COMMIT_ID
⋮	☆	GIT_EVENT_DESCRIPTION
⋮	☆	GIT_EVENT_OWNER_NAME
⋮	☆	GIT_EVENT_TIMESTAMP
⋮	☆	GIT_REPO_NAME
⋮	☆	GIT_SERVER_URL

For information about input parameters, see How you'll create the CICD pipeline and configure the workspace.

Procedure

1 In Code Stream, click **Triggers > Git**.

2 Click the **Webhooks for Git** tab, then click **New Webhook for Git**.

   a Select a project.

   b Enter a meaningful name and description for the webhook.

c Select a Git endpoint configured for the branch you want to monitor.

When you create your webhook, the webhook definition includes the current endpoint details.

- If you later change the Git type, Git server type, or Git repository URL in the endpoint, the webhook will no longer be able to trigger a pipeline because it will try to access the Git repository using the original endpoint details. You must delete the webhook and create it again with the endpoint.
- If you later change the authentication type, username, or private token in the endpoint, the webhook will continue to work.
- If you are using a BitBucket repository, the URL for the repository must be in one of these formats: `https://api.bitbucket.org/{user}/{repo name}` or `http(s)://{bitbucket-enterprise-server}/rest/api/1.0/users/{username}/repos/{repo name}`.

Note If you previously created a webhook using a Git endpoint that uses a password for basic authentication, you must delete and redefine the webhook with a Git endpoint that uses a private token for authentication.

See How do I integrate Code Stream with Git.

d (Optional) Enter the branch that you want the webhook to monitor.

If you leave the branch unspecified, the webhook monitors the branch that you configured for the Git endpoint.

e (Optional) Generate a secret token for the webhook.

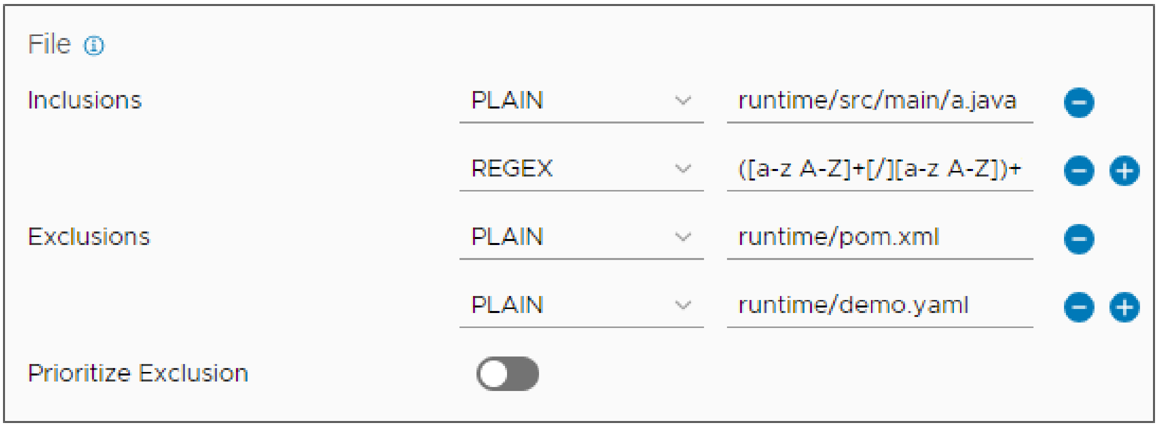
If you use a secret token, Code Stream generates a random string token for the webhook. Then, when the webhook receives Git event data, it sends the data with the secret token. Code Stream uses the information to determine if the calls are coming from the expected source such as the configured GitHub instance, repository, and branch. The secret token provides an extra layer of security that is used to verify that the Git event data is coming from the correct source.

f (Optional) Provide file inclusions or exclusions as conditions for the trigger.

- File inclusions. If any of the files in a commit match the files specified in the inclusion paths or regex, the pipelines will trigger. With a regex specified, Code Stream only triggers the pipelines when filenames in the changeset match the expression provided. The regex filter is useful when configuring a trigger for multiple pipelines on a single repository.
- File exclusions. When all the files in a commit match the specified files in the exclusion paths or regex, the pipelines do not trigger.
- Prioritize exclusions. When toggled on, Prioritize Exclusion ensures that pipelines do not trigger even if any of the files in a commit match the files specified in the exclusion paths or regex. The default setting is off.

If conditions meet both the file inclusions and file exclusions, pipelines do not trigger.

In the following example, both file inclusions and file exclusions are conditions for the trigger.



File

Inclusions	PLAIN	runtime/src/main/a.java
	REGEX	([a-z A-Z]+[/][a-z A-Z])+
Exclusions	PLAIN	runtime/pom.xml
	PLAIN	runtime/demo.yaml
Prioritize Exclusion		

- For file inclusions, a commit with any change to `runtime/src/main/a.java` or any Java file will trigger pipelines configured in the event configuration.
- For file exclusions, a commit with changes only in both files will not trigger the pipelines configured in the event configurations.

g For the Git event, select a **Push** or **Pull** request.

h Enter the API Token.

   The CSP API token authenticates you for external API connections with Code Stream. To obtain the API token:

   1 Click **Generate Token**.

   2 Enter the email address associated with your user name and password and click **Generate**.

      The token that you generate is valid for six months. It is also known as a refresh token.

      - To keep the token as a variable for future use, click **Create Variable**, enter a name for the variable and click **Save**.

      - To keep the token as a text value for future use, click **Copy** and paste the token into a text file to save locally.

      You can choose to both create a variable and store the token in a text file for future use.

   3 Click **Close**.

i Select the pipeline for the webhook to trigger.

   If the pipeline includes custom added input parameters, the Input Parameters list displays parameters and values. You can enter values for input parameters that pass to the pipeline with the trigger event. Or, you can leave the values blank, or use the default values if defined.

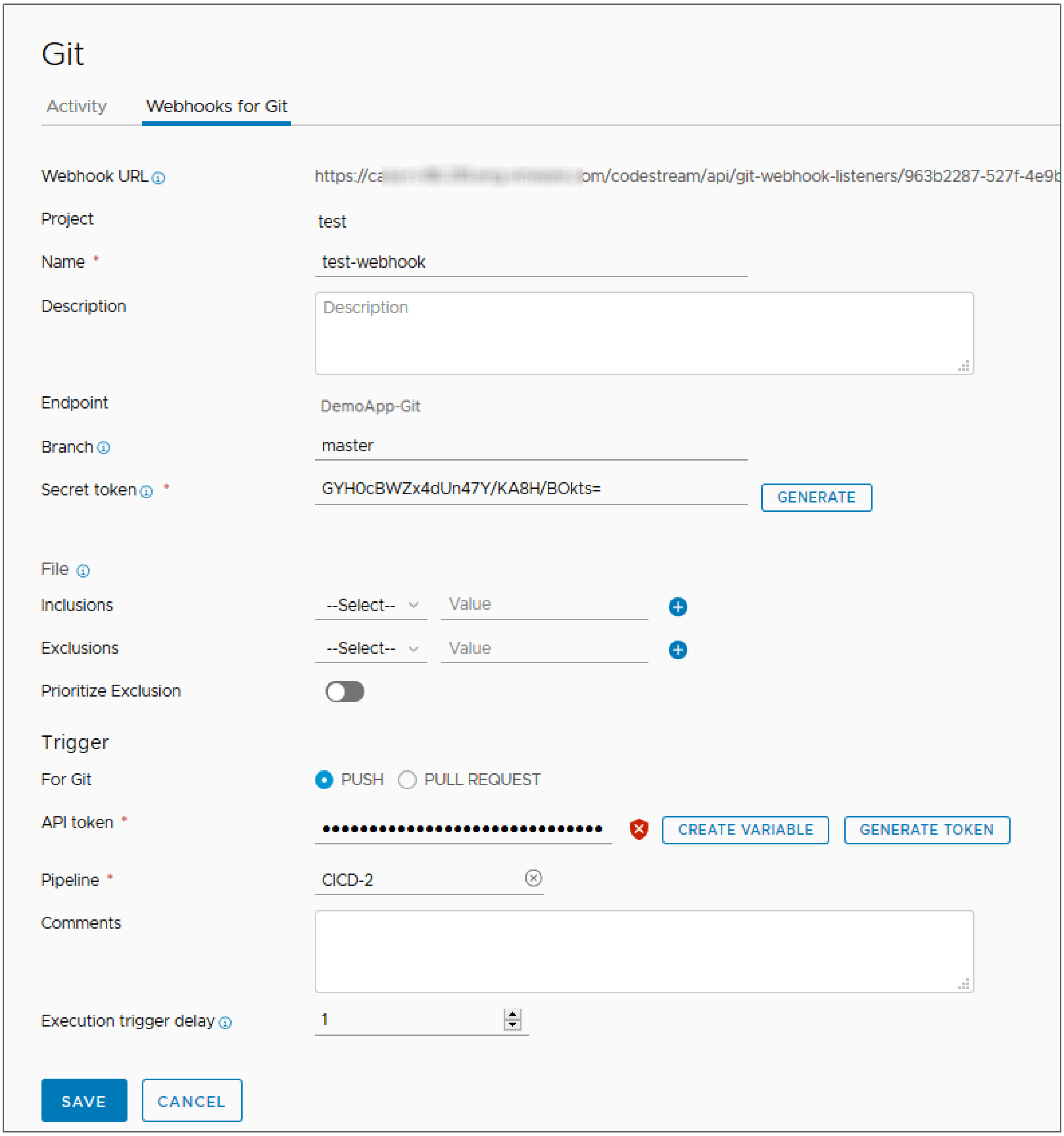
   For information about Auto inject input parameters for Git triggers, see the Prerequisites.

j Click **Create**.

   The webhook appears as a new card.

3 Click the webhook card.

   When the webhook data form reappears, you see a webhook URL added to the top of the form. The Git webhook connects to the GitHub repository through the webhook URL.



Git

Webhook URL ⓘ	https://ca[redacted]om/codestream/api/git-webhook-listeners/963b2287-527f-4e9b
Project	test
Name *	test-webhook
Description	Description
Endpoint	DemoApp-Git
Branch ⓘ	master
Secret token ⓘ *	GYH0cBWZx4dUn47Y/KA8H/BOkts=

File ⓘ

Inclusions	--Select--	Value
Exclusions	--Select--	Value
Prioritize Exclusion		

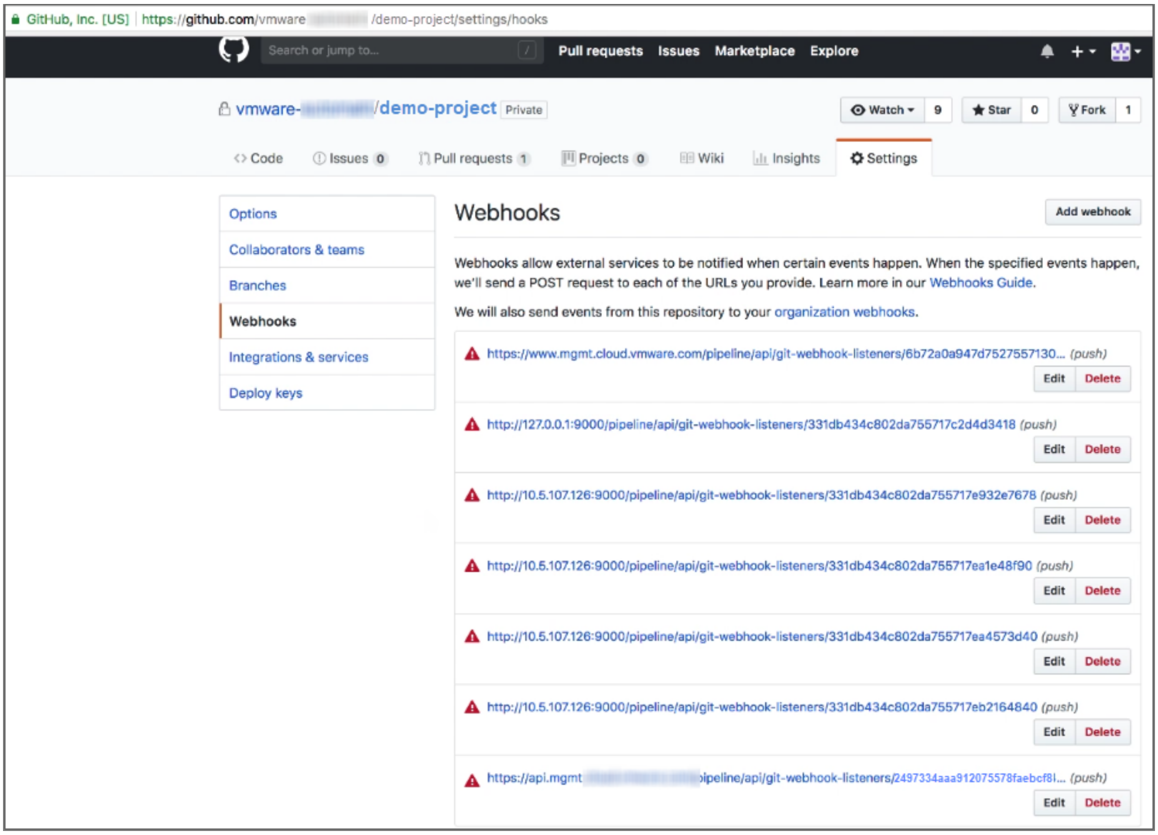
Trigger

For Git	● PUSH ○ PULL REQUEST
API token *	••••••••••••••••••••••••••••••••
Pipeline *	CICD-2
Comments	
Execution trigger delay ⓘ	1

4 In a new browser window, open the GitHub repository that connects through the webhook.

a To see the webhook that you added in Code Stream, click the **Settings** tab and select **Webhooks**.

At the bottom of the webhooks list, you see the same webhook URL.



Webhooks

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our Webhooks Guide.

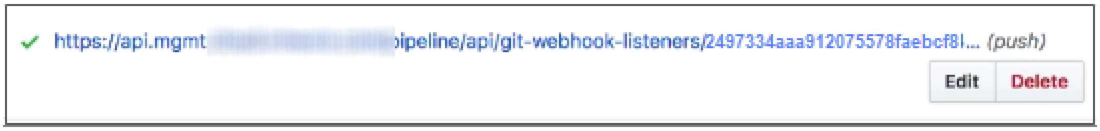
We will also send events from this repository to your organization webhooks.

```
https://www.mgmt.cloud.vmware.com/pipeline/api/git-webhook-listeners/6b72a0a947d7527557130... (push)
http://127.0.0.1:9000/pipeline/api/git-webhook-listeners/331db434c802da755717c2d4d3418 (push)
http://10.5.107.126:9000/pipeline/api/git-webhook-listeners/331db434c802da755717e932e7678 (push)
http://10.5.107.126:9000/pipeline/api/git-webhook-listeners/331db434c802da755717ea1e48f90 (push)
http://10.5.107.126:9000/pipeline/api/git-webhook-listeners/331db434c802da755717ea4573d40 (push)
http://10.5.107.126:9000/pipeline/api/git-webhook-listeners/331db434c802da755717eb2164840 (push)
https://api.mgmt[redacted]pipeline/api/git-webhook-listeners/2497334aaa912075578faebcf8l... (push)
```

b To make a code change, click the **Code** tab and select a file on the branch. After you edit the file, commit the change.

c To verify that the webhook URL is working, click the **Settings** tab and select **Webhooks** again.

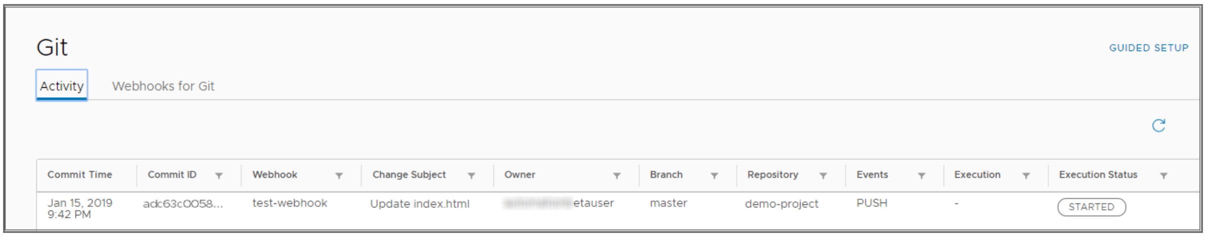
At the bottom of the webhooks list, a green checkmark appears next to the webhook URL.



```
✓ https://api.mgmt[redacted]pipeline/api/git-webhook-listeners/2497334aaa912075578faebcf8l... (push)
```

5 Return to Code Stream to view the activity on the Git webhook. Click **Triggers > Git > Activity**.

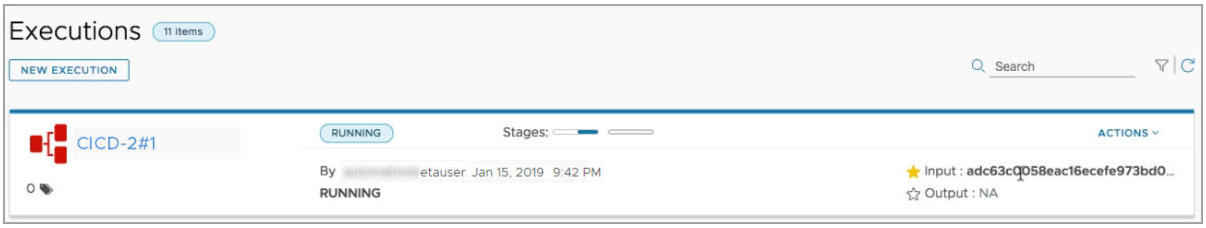
Under Execution Status, verify that the pipeline run has started.



Commit Time	Commit ID	Webhook	Change Subject	Owner	Branch	Repository	Events	Execution	Execution Status
Jan 15, 2019 9:42 PM	adc63c0058...	test-webhook	Update index.html	[redacted]etauser	master	demo-project	PUSH	-	STARTED

6 Click **Executions** and track your pipeline as it runs.

To observe the pipeline run, you can press the refresh button.



CICD-2#1

0

RUNNING

Stages:

By [redacted]etauser Jan 15, 2019 9:42 PM

RUNNING

Input : adc63c0058eac16ecefe973bd0...

Output : NA

Results

Congratulations! You successfully used the trigger for Git!

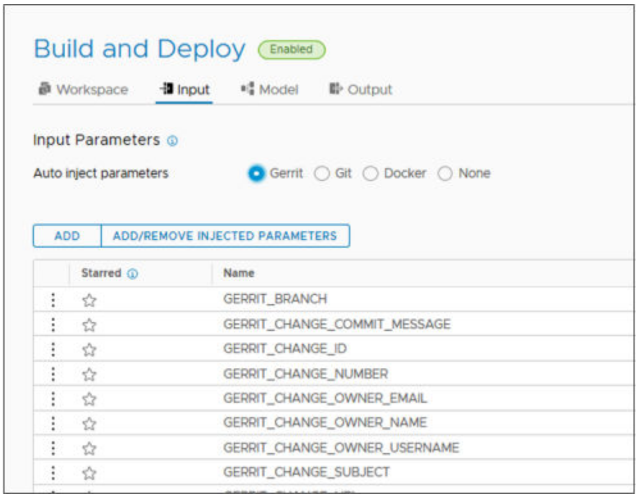
How do I use the Gerrit trigger in Code Stream to run a pipeline

As a Code Stream administrator or developer, you can integrate Code Stream with the Gerrit code review lifecycle by using the Gerrit trigger. The event triggers a pipeline to run when you create a patch set, publish drafts, merge code changes on the Gerrit project, or directly push changes on the Git branch.

When you add the Gerrit trigger, you select a Gerrit listener, a Gerrit project on the Gerrit server, and you configure Gerrit events. In this example, you first configure a Gerrit listener, then you use that listener in a Gerrit trigger with two events on three different pipelines.

Prerequisites

- Verify that you are a member of a project in Code Stream. If you are not, ask a Code Stream administrator to add you as a member of a project. See How do I add a project in Code Stream.
- Verify that you have a Gerrit endpoint configured in Code Stream. See How do I integrate Code Stream with Gerrit.
- For pipelines to trigger, verify that you set the input properties of the pipeline as **Gerrit**, which allows the pipeline to receive the Gerrit parameters as inputs when the pipeline runs.



Input Parameters

Auto inject parameters ● Gerrit ○ Git ○ Docker ○ None

Starred	Name
☆	GERRIT_BRANCH
☆	GERRIT_CHANGE_COMMIT_MESSAGE
☆	GERRIT_CHANGE_ID
☆	GERRIT_CHANGE_NUMBER
☆	GERRIT_CHANGE_OWNER_EMAIL
☆	GERRIT_CHANGE_OWNER_NAME
☆	GERRIT_CHANGE_OWNER_USERNAME
☆	GERRIT_CHANGE_SUBJECT

For information about input parameters, see How you'll create the CICD pipeline and configure the workspace.

Procedure

1 In Code Stream, click **Triggers** > **Gerrit**.

2 (Optional) Click the **Listeners** tab, then click **New Listener**.

Note If the Gerrit listener that you plan to use for the Gerrit trigger is already defined, skip this step.

a Select a project.

b Enter a name for the Gerrit listener.

c Select a Gerrit endpoint.

d Enter the API Token.

The CSP API token authenticates you for external API connections with Code Stream. To obtain the API token:

1 Click **Generate Token**.

2 Enter the email address associated with your user name and password and click **Generate**.

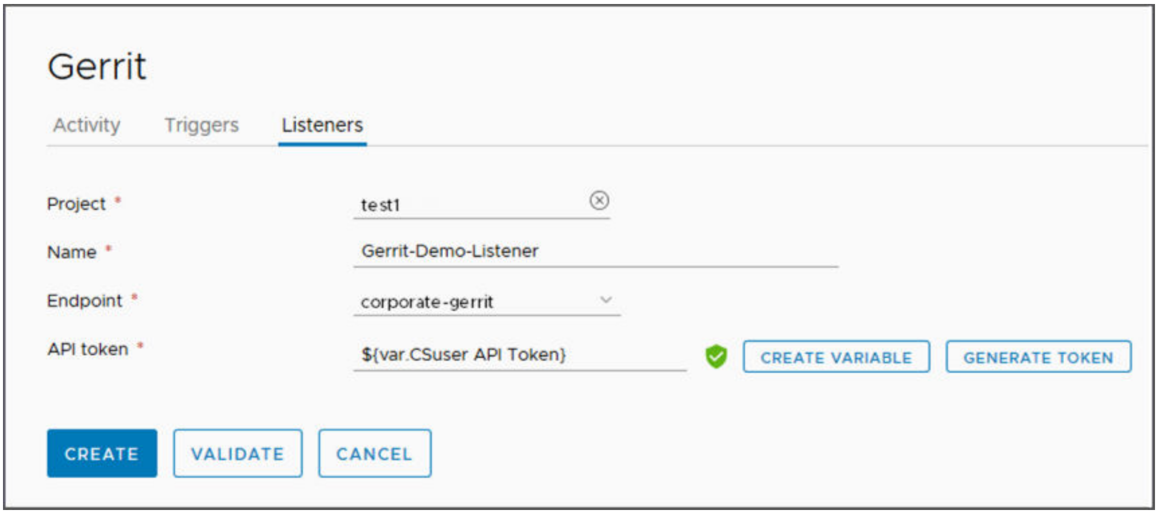
The token that you generate is valid for six months. It is also known as a refresh token.

- To keep the token as a variable for future use, click **Create Variable**, enter a name for the variable and click **Save**.
- To keep the token as a text value for future use, click **Copy** and paste the token into a text file to save locally.

You can choose to both create a variable and store the token in a text file for future use.

3 Click **Close**.

If you created a variable, the API token displays the variable name that you entered by using dollar binding. If you copied the token, the API token displays the masked token.



Gerrit

Project * test1

Name * Gerrit-Demo-Listener

Endpoint * corporate-gerrit

API token * ${var.CSuser API Token}

e To validate the token and endpoint details, click **Validate**.

Your token expires after 90 days.

   f. Click **Create**.

   g. On the listener card, click **Connect**.

      The listener starts monitoring all activity on the Gerrit server and listens for any enabled triggers on that server. To stop listening for a trigger on that server, you deactivate the trigger.

      Note To update a Gerrit endpoint that is connected to a listener, you must disconnect the listener before updating the endpoint.

      - Click **Configure > Triggers > Gerrit** .
      - Click the **Listeners** tab.
      - Click **Disconnect** on the listener that is connected to the endpoint that you want to update.

3. Click the **Triggers** tab, then click **New Trigger**.
4. Select a project on the Gerrit server.
5. Enter a name.

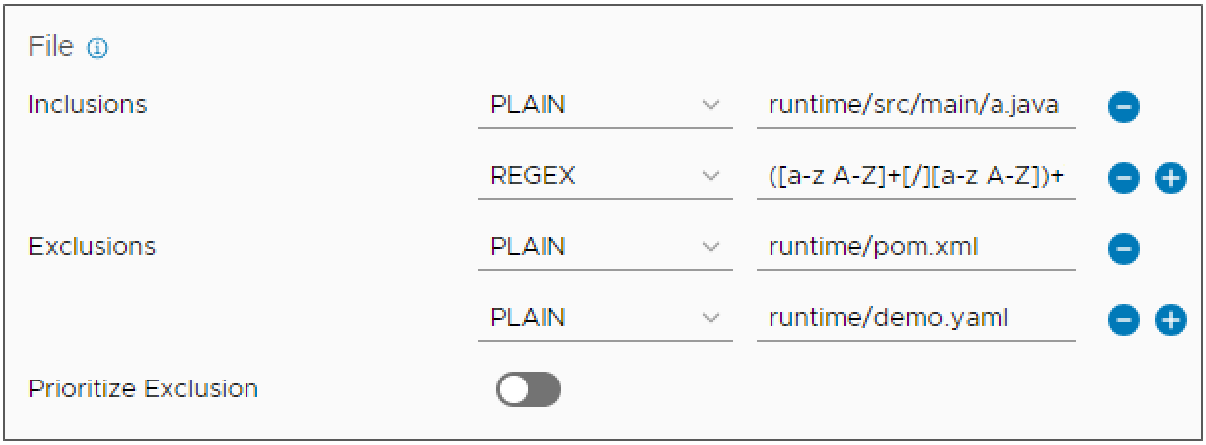
   The Gerrit trigger name must be unique.
6. Select a configured Gerrit listener.

   By using the Gerrit listener, Code Stream provides a list of Gerrit projects that are available on the server.
7. Select a project on the Gerrit server.
8. Enter the branch in the repository that the Gerrit listener will monitor.
9. (Optional) Provide file inclusions or exclusions as conditions for the trigger.

   - You provide file inclusions that trigger the pipelines. When any of the files in a commit match the files specified in the inclusion paths or regex, pipelines trigger. With a regex specified, Code Stream only triggers pipelines with filenames in the changeset that match the expression provided. The regex filter is useful when configuring a trigger for multiple pipelines on a single repository.
   - You provide file exclusions that keep pipelines from triggering. When all the files in a commit match the files specified in the exclusion paths or regex, the pipelines do not trigger.
   - **Prioritize Exclusion**, when toggled on, ensures that pipelines do not trigger. The pipelines won't trigger even if any of the files in a commit match the files specified in the exclusion paths or regex. The default setting for **Prioritize Exclusion** is turned off.

   If the conditions meet both the file inclusion and the file exclusion, pipelines do not trigger.

   In the following example, both the file inclusions and the file exclusions are conditions for the trigger.



File

Inclusions	PLAIN	runtime/src/main/a.java
	REGEX	([a-z A-Z]+[/][a-z A-Z])+
Exclusions	PLAIN	runtime/pom.xml
	PLAIN	runtime/demo.yaml
Prioritize Exclusion		

- For file inclusions, a commit that has any change to `runtime/src/main/a.java` or any Java file will trigger the pipelines configured in the event configuration.
- For file exclusions, a commit that has changes only in both files will not trigger the pipelines configured in the event configuration.

10 Click **New Configuration**.

a For a Gerrit event, select **Patchset Created**, **Draft Published**, or **Change Merged**. Or, for a direct push to Git that bypasses Gerrit, select **Direct Git push**.

b Select the pipeline that will trigger.

If the pipeline includes custom added input parameters, the Input Parameters list displays parameters and values. You can enter values for input parameters to be passed to the pipeline with the trigger event. Or, you can leave the values blank, or use the default values.

Note If default values are defined:

- Any values you enter for the input parameters will override the default values defined in the pipeline model.
- The default values in the trigger configuration will not change if the parameter values in the pipeline model change.

For information about Auto inject input parameters for Gerrit triggers, see the Prerequisites.

c For **Patchset Created**, **Draft Published**, and **Change Merged**, some actions appear with labels by default. You can change the label or add comments. Then, when the pipeline runs, the label or comment appears on the **Activity** tab as the **Action taken** for the pipeline.

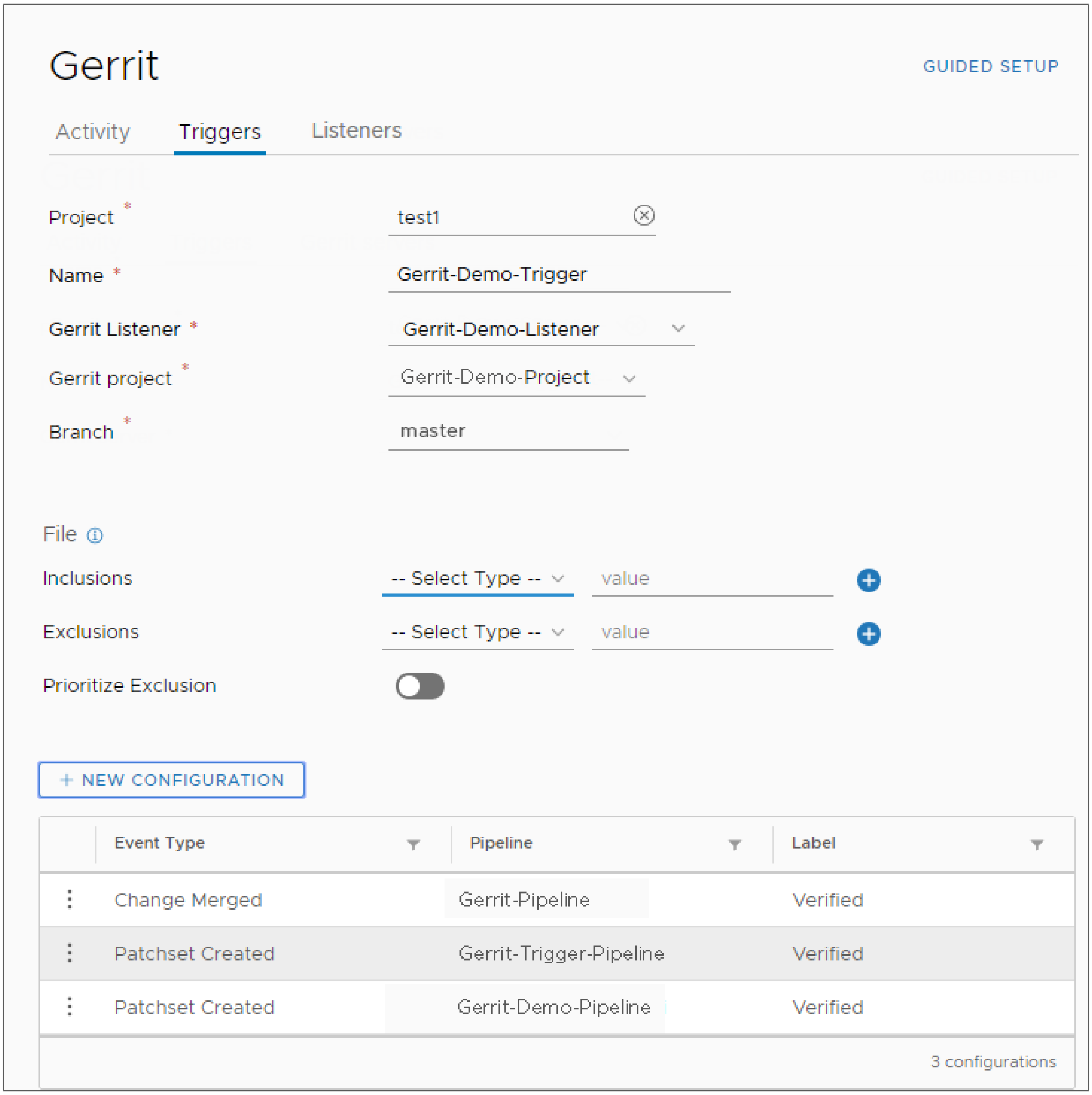
The Gerrit Event configuration allows you to enter comments by using a variable for the Success comment or Failure comment. For example: `${var.success}` and `${var.failure}`.

d Click **Save**.

To add multiple trigger events on multiple pipelines, click **New Configuration** again.

In the following example, you can see events for three pipelines:

- If a **Change Merged** event occurs in the Gerrit project, the pipeline named **Gerrit-Pipeline** triggers.
- If a **Patchset Created** event occurs in the Gerrit project, the pipelines named **Gerrit-Trigger-Pipeline** and **Gerrit-Demo-Pipeline** trigger.



Project * test1

Name * Gerrit-Demo-Trigger

Gerrit Listener * Gerrit-Demo-Listener

Gerrit project * Gerrit-Demo-Project

Branch * master

File

Inclusions -- Select Type -- value

Exclusions -- Select Type -- value

Prioritize Exclusion

Event Type	Pipeline	Label
Change Merged	Gerrit-Pipeline	Verified
Patchset Created	Gerrit-Trigger-Pipeline	Verified
Patchset Created	Gerrit-Demo-Pipeline	Verified

3 configurations

11 Click **Create**.

The Gerrit trigger appears as a new card on the **Triggers** tab, and is set as **Disabled** by default.

12 On the trigger card, click **Enable**.

After you enable the trigger, it can use the Gerrit listener, which starts monitoring events that occur on the branch of the Gerrit project.

To create a trigger that has the same file inclusion conditions or file exclusion conditions, but with a different repository than the one you included when you created the trigger, on the trigger card click **Actions > Clone**. Then, on the cloned trigger, click **Open**, and change the parameters.

Results

Congratulations! You successfully configured a Gerrit trigger with two events on three different pipelines.

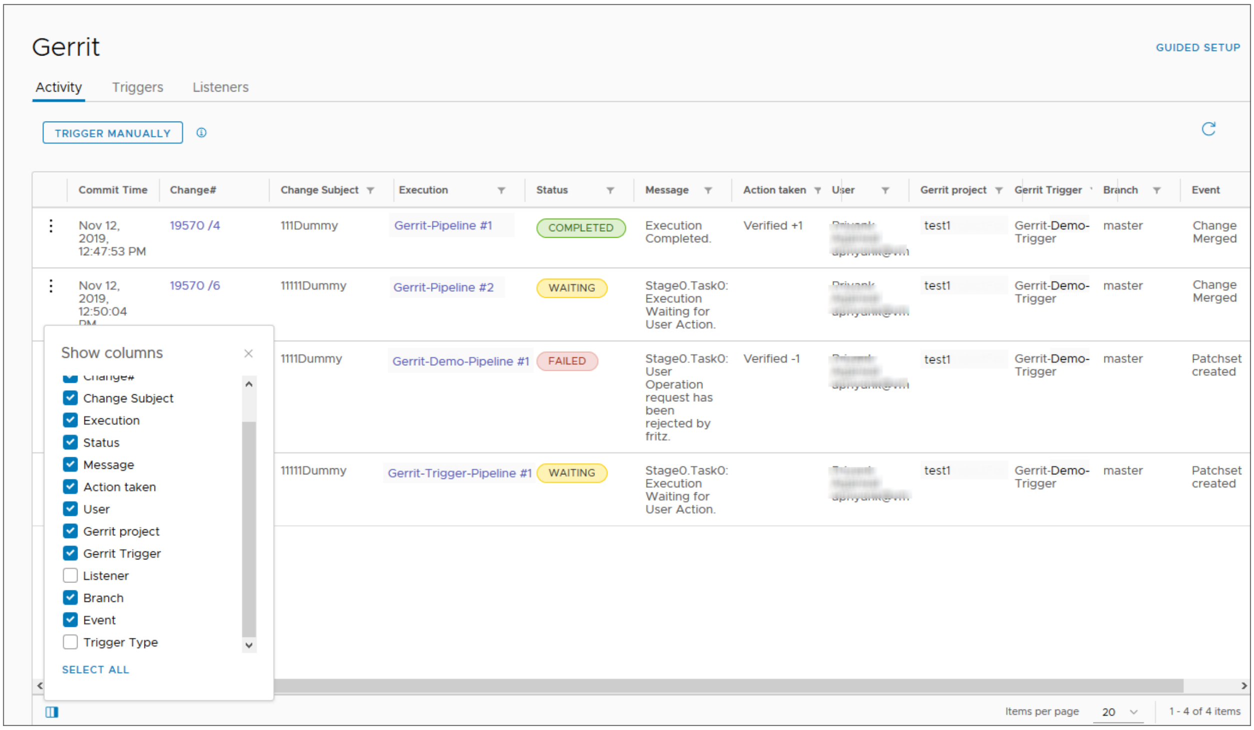
What to do next

After you commit a code change in the Gerrit project, observe the **Activity** tab for the Gerrit event in Code Stream. Verify that the list of activities includes entries that correspond to every pipeline execution in the trigger configuration.

When an event occurs, only pipelines in the Gerrit trigger that relate to the particular type of event can run. In this example, if a patch set is created, only the **Gerrit-Trigger-Pipeline** and the **Gerrit-Demo-Pipeline** will run.

Information in the columns on the **Activity** tab describe each Gerrit trigger event. You can select the columns that appear by clicking the column icon that appears below the table.

- The **Change Subject** and **Execution** columns are empty when the trigger was a direct Git push.
- The **Gerrit Trigger** column displays the trigger that created the event.
- The **Listener** column is turned off by default. When you select it, the column displays the Gerrit listener that received the event. A single listener can appear as associated with multiple triggers.
- The **Trigger Type** column is turned off by default. When you select it, the column displays the type of trigger as AUTOMATIC or MANUAL.
- Other columns include **Commit Time**, **Change#**, **Status**, **Message**, **Action taken**, **User**, **Gerrit project**, **Branch**, and **Event**.



Gerrit

GUIDED SETUP

Activity | Triggers | Listeners

TRIGGER MANUALLY

	Commit Time	Change#	Change Subject	Execution	Status	Message	Action taken	User	Gerrit project	Gerrit Trigger	Branch	Event
⋮	Nov 12, 2019, 12:47:53 PM	19570 /4	111Dummy	Gerrit-Pipeline #1	COMPLETED	Execution Completed.	Verified +1	[redacted]	test1	Gerrit-Demo-Trigger	master	Change Merged
⋮	Nov 12, 2019, 12:50:04	19570 /6	11111Dummy	Gerrit-Pipeline #2	WAITING	Stage0.Task0: Execution Waiting for User Action.		[redacted]	test1	Gerrit-Demo-Trigger	master	Change Merged
			1111Dummy	Gerrit-Demo-Pipeline #1	FAILED	Stage0.Task0: User Operation request has been rejected by fritz.	Verified -1	[redacted]	test1	Gerrit-Demo-Trigger	master	Patchset created
			11111Dummy	Gerrit-Trigger-Pipeline #1	WAITING	Stage0.Task0: Execution Waiting for User Action.		[redacted]	test1	Gerrit-Demo-Trigger	master	Patchset created

Show columns

- ☑ Change#
- ☑ Change Subject
- ☑ Execution
- ☑ Status
- ☑ Message
- ☑ Action taken
- ☑ User
- ☑ Gerrit project
- ☑ Gerrit Trigger
- □ Listener
- ☑ Branch
- ☑ Event
- □ Trigger Type

SELECT ALL

Items per page 20 | 1 - 4 of 4 items

To control the activity for a completed or failed pipeline run, click the three dots at the left of any entry on the Activity screen.

- If the pipeline fails to run because of a mistake in the pipeline model or another problem, correct the mistake and select **Re-run**, which runs the pipeline again.
- If the pipeline fails to run because of a network connectivity issue or another problem, select **Resume**, which restarts the same pipeline execution, and saves run time.
- Use **View Execution**, which opens the pipeline execution view. See How do I run a pipeline and see results.
- Use **Delete** to delete the entry from the Activity screen.

If a Gerrit event fails to trigger a pipeline, you can click **Trigger Manually**, then select the Gerrit trigger, enter the Change-Id, and click **Run**.

Monitoring pipelines in Code Stream

8

As a Code Stream administrator or developer, you need insight about the performance of your pipelines in Code Stream. You need to know how effectively your pipelines release code from development, through testing, and to production.

To gain insight, you use Code Stream dashboards to monitor the trends and results of a pipeline execution. You can use the default pipeline dashboards to monitor a single pipeline, or create custom dashboards to monitor multiple pipelines.

- Pipeline metrics include statistics such as mean times, which are available on the pipeline dashboard.
- To see metrics across multiple pipelines, use the custom dashboards.

This chapter includes the following topics:

What does the pipeline dashboard show me in Code Stream

A pipeline dashboard is a view of the results for a specific pipeline that ran, such as trends, top failures, and successful changes. Code Stream creates the pipeline dashboard when you create a pipeline.

The dashboard contains the widgets that display pipeline execution results.

Pipeline Execution Status Counts Widget

You can view the total number of executions of a pipeline over a period of time grouped by status: Completed, Failed, or Canceled. To see how the pipeline execution status has changed over longer or shorter periods of time, change the duration on the display.

Pipeline Execution Statistics Widget

The pipeline execution statitstics include the mean times to recover, deliver, or fail a pipeline over time.

The following states apply to all pipeline executions:

- Completed
- Failed
- Waiting
- Running
- Canceled
- Queued
- Not Started
- Rolling Back
- Rollback Completed
- Rollback Failed
- Paused

Table 8-1. Measuring mean times

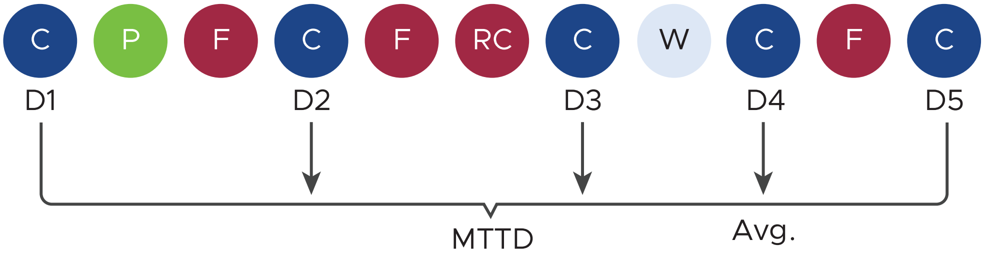
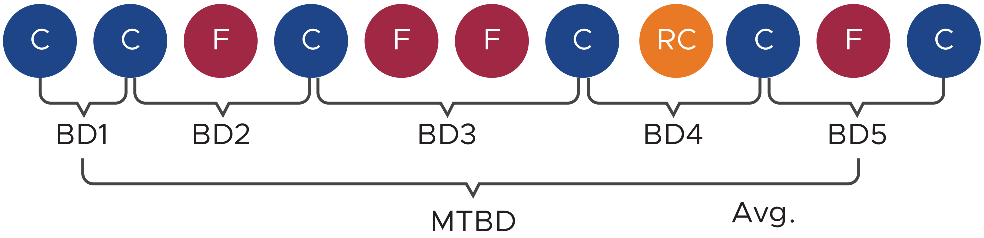
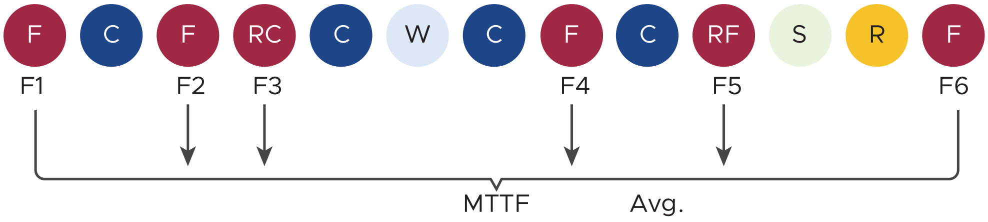
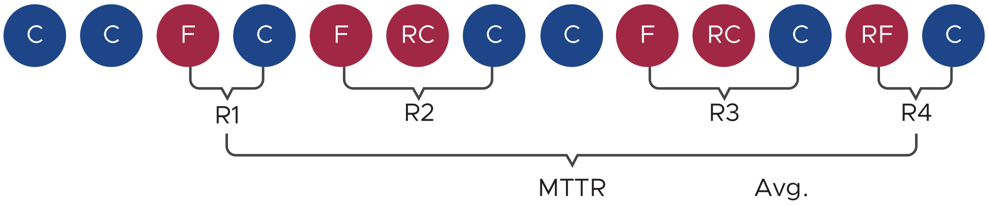
What gets measured...	What it means...
Average CI	Average time spent in the continuous integration phase, measured by time in the CI task type.
Mean time to delivery (MTTD)	Average duration of all COMPLETED runs over a period of time. D1, D2, and so forth is the amount of time to deliver each COMPLETED run.  C P F C F RC C W C F C D1 D2 D3 D4 D5 MTTD Avg.
Mean time between deliveries (MTBD)	Average time elapsed between successful deliveries over a period of time. The time elapsed between two consecutive COMPLETED runs is the time between successful deliveries, such as BD1, BD2 and so forth. MTBD indicates how often a production environment updates.  C C F C F F C RC C F C BD1 BD2 BD3 BD4 BD5 MTBD Avg.

Table 8-1. Measuring mean times (continued)

What gets measured...	What it means...
Mean time to failure (MTTF)	Average duration of runs that end in FAILED, ROLLBACK_COMPLETED or ROLLBACK_FAILED states over a period of time. F1, F2, and so forth is the amount of time for a run to end in FAILURE, ROLLBACK_COMPLETED, or ROLLBACK FAILED.  F C F RC C W C F C RF S R F F1 F2 F3 F4 F5 F6 MTTF Avg.
Mean time to recovery (MTTR)	Average time to recovery from a failure over a period of time. The time to recovery from a failure is the time elapsed between a run with a final status of FAILED, ROLLBACK_COMPLETED, or ROLLBACK_FAILED and the next immediate successful run with a COMPLETED status. R1, R2 and so forth, is the amount of time to recovery after each FAILED or ROLLBACK_FAILED run. 

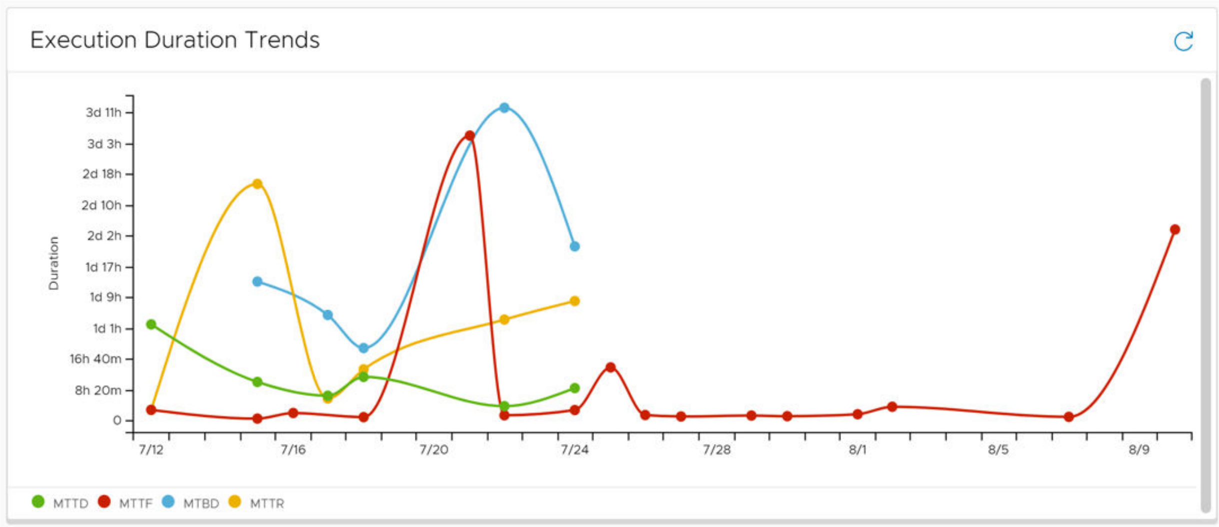
Top Failed Stages and Tasks Widgets

Two widgets display the top failed stages and tasks in a pipeline. Each measurement reports the number and percentage of failures for development and post-development environments for each pipeline and project, averaged over a week or month. You view the top failures to troubleshoot problems in the release automation process.

For example, you can configure the display for a particular duration such as the last seven days and note the top failed tasks during that period of time. If you make a change in your environment or pipeline and run the pipeline again, then check the top failed tasks over a longer duration such as the last 14 days, the top failed tasks may have changed. With that result, you will know that the change in your release automation process improved the success rate of your pipeline execution.

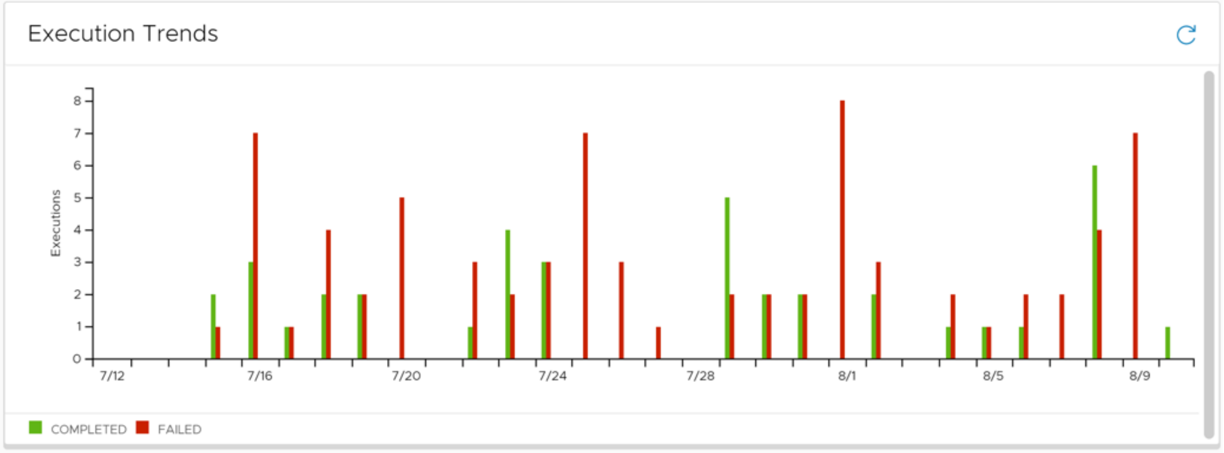
Pipeline Execution Duration Trends Widget

Pipeline execution duration trends show the MTTD, MTTF, MTBD, and MTTR, over a period of time.



Pipeline Execution Trends Widget

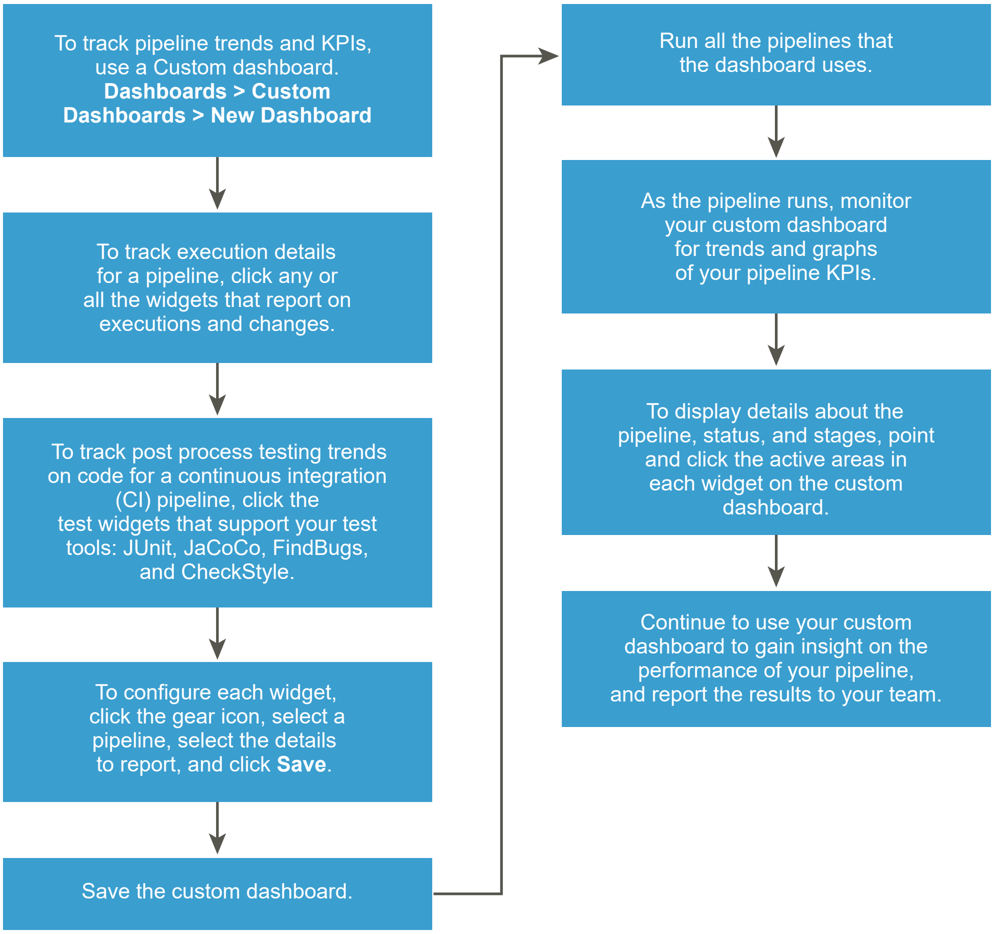
Pipeline execution trends show the total daily runs of a pipeline, grouped by status over a period of time. Except for the current day, most daily aggregation counts only show COMPLETED and FAILED runs.



How do I use custom dashboards to track key performance indicators for my pipeline in Code Stream

As a Code Stream administrator of developer, you create the custom dashboard to display the results you want to see for one or more pipelines that ran. For example, you can create a project-wide dashboard with KPIs and metrics gathered from multiple pipelines. If an execution warning or failure is reported, you can use the dashboard to troubleshoot the failure.

To track trends and key performance indicators for your pipelines by using a custom dashboard, you add widgets to the dashboard, and configure them to report on your pipelines.


To track pipeline trends and KPIs, use a Custom dashboard. **Dashboards > Custom Dashboards > New Dashboard**
To track execution details for a pipeline, click any or all the widgets that report on executions and changes.
To track post process testing trends on code for a continuous integration (CI) pipeline, click the test widgets that support your test tools: JUnit, JaCoCo, FindBugs, and CheckStyle.
To configure each widget, click the gear icon, select a pipeline, select the details to report, and click **Save**.
Save the custom dashboard.
Run all the pipelines that the dashboard uses.
As the pipeline runs, monitor your custom dashboard for trends and graphs of your pipeline KPIs.
To display details about the pipeline, status, and stages, point and click the active areas in each widget on the custom dashboard.
Continue to use your custom dashboard to gain insight on the performance of your pipeline, and report the results to your team.

Prerequisites

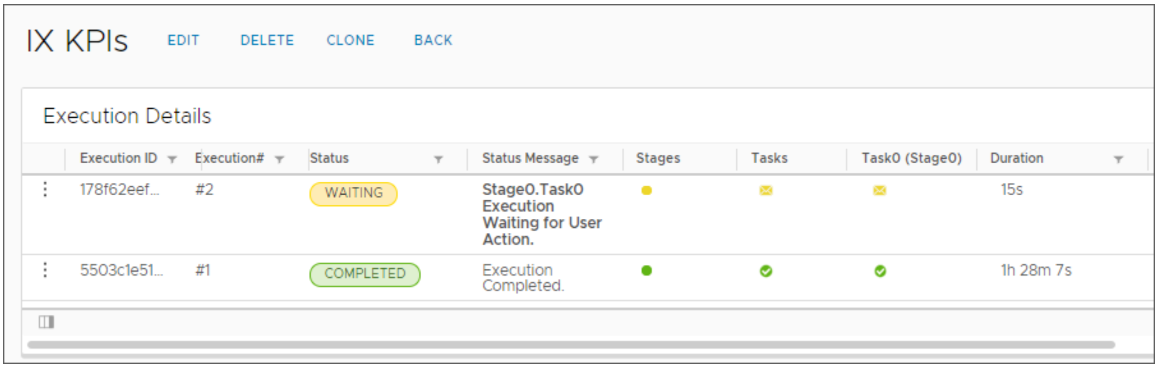
- Verify that one or more pipelines exist. In the user interface, click **Pipelines**.
- For the pipelines that you intend to monitor, verify that they ran successfully. Click **Executions**.

Procedure

1 To create a custom dashboard, click **Dashboards > Custom Dashboards > New Dashboard**.

2 To customize the dashboard so that it reports on specific trends and key performance indicators for your pipeline, click a widget.

For example, to display details about the pipeline status, stages, tasks, how long it ran, and who ran it, click the **Execution Details** widget. Or, for a continuous integration (CI) pipeline, you can track the trends on post-processing by using the widgets for JUnit, JaCoCo, FindBugs, and CheckStyle.



Execution Details

Execution ID	Execution#	Status	Status Message	Stages	Tasks	Task0 (Stage0)	Duration
178f62eef...	#2	WAITING	Stage0.Task0 Execution Waiting for User Action.	●	✉	✉	15s
5503c1e51...	#1	COMPLETED	Execution Completed.	●	✓	✓	1h 28m 7s

3 Configure each widget that you add.

a On the widget, click the gear icon.

b Select a pipeline, set the available options, and select the columns to display.

c To save the widget configuration, click **Save**.

d To save the custom dashboard, click **Save**, and click **Close**.

4 To display more information about the pipeline, click the active areas on the widgets.

For example, in the **Execution Details** widget, click an entry in the Status column to display more information about the pipeline execution. Or, on the **Latest Successful Change** widget, to display a summary of the pipeline stage and task, click the active link.

Results

Congratulations! You created a custom dashboard that monitors trends and KPIs for your pipelines.

What to do next

Continue to monitor the performance of your pipelines in Code Stream, and share the results with your manager and teams to continue to improve the process to release your applications.

Learn more about Code Stream 9

There are many ways for Code Stream administrators and developers to learn more about Code Stream and what it can do for you.

You can use this documentation to learn more about pipelines and their executions, how to add endpoints, how to add projects, and more.

Understand the permissions that roles provide. Learn how to use restricted resources, and require approvals on pipelines. See How do I manage user access and approvals in Code Stream.

See the value of search by discovering where specific jobs or components are located in your pipelines, executions, or endpoints.

This chapter includes the following topics:

What is Search in Code Stream

You use search to find where specific items or other components are located. For example, you might want to search for activated or deactivated pipelines. Because if a pipeline is deactivated, it cannot run.

What can I search

You can search in:

- Projects
- Endpoints
- Pipelines
- Executions
- Pipeline Dashboards, Custom Dashboards
- Gerrit Triggers and Servers
- Git Webhooks
- Docker Webhooks

You can perform column-based filter search in:

- User Operations
- Variables
- Trigger Activity for Gerrit, Git, and Docker

You can perform grid-based filter search on the **Activity** page for each trigger.

How does search work

The criteria for search varies depending on the page you are on. Each page has different search criteria.

Where you search	Criteria to use for search
Pipeline Dashboards	Project, Name, Description, Tags, Link
Custom Dashboards	Project, Name, Description, Link (UUID of an item on the dashboard)
Executions	Name, Comments, Reason, Tags, Index, Status, Project, Show, Executed by, Executed by me, Link (UUID of the execution), and Input parameters, Output parameters, or Status message by using this format: `<key>:<value>`
Pipelines	Name, Description, State, Tags, Created by, Created by me, Updated by, Updated by me, Project
Projects	Name, Description
Endpoints	Name, Description, Type, Updated by, Project
Gerrit triggers	Name, Status, Project
Gerrit servers	Name, Server URL, Project
Git Webhooks	Name, Server Type, Repo, Branch, Project

Where:

- Link is the UUID of a pipeline, execution, or widget on a dashboard.
- Input parameter, Output parameter, and Status message notation and examples include:
  - Notation: `input.<inputKey>:<inputValue>`

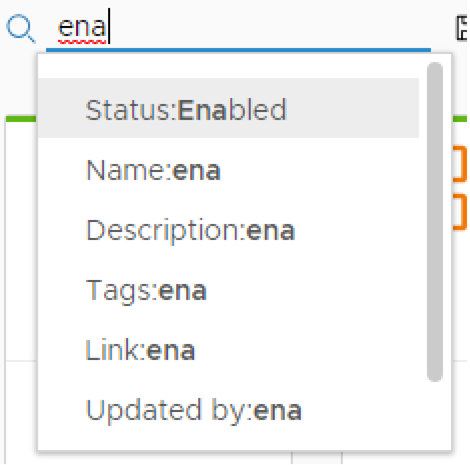
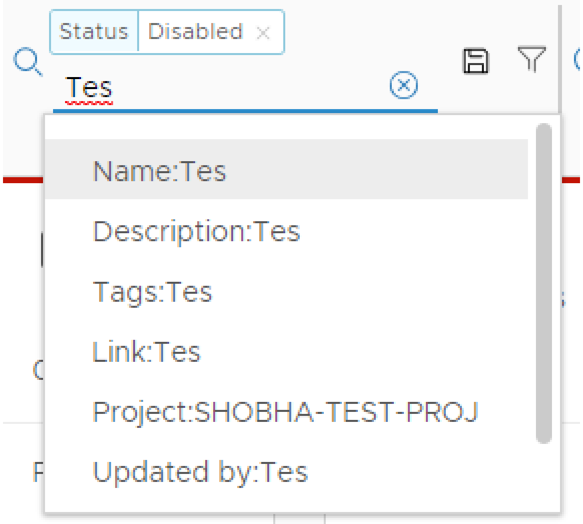
    Example: **`input.GERRIT_CHANGE_OWNER_EMAIL:joe_user`**
  - Notation: `output.<outputKey>:<outputValue>`

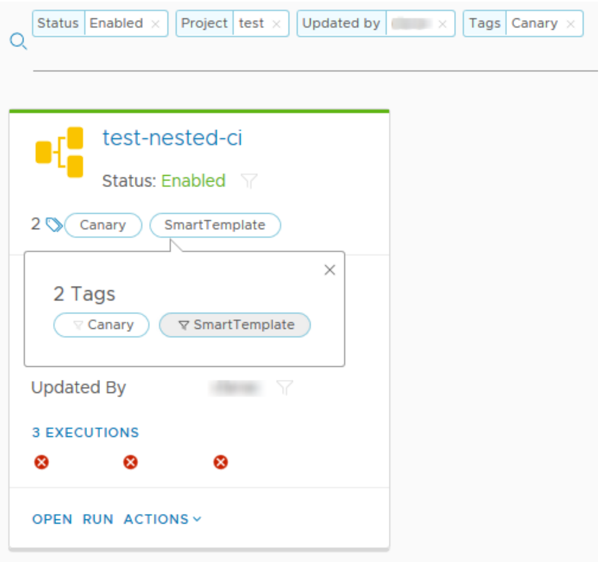
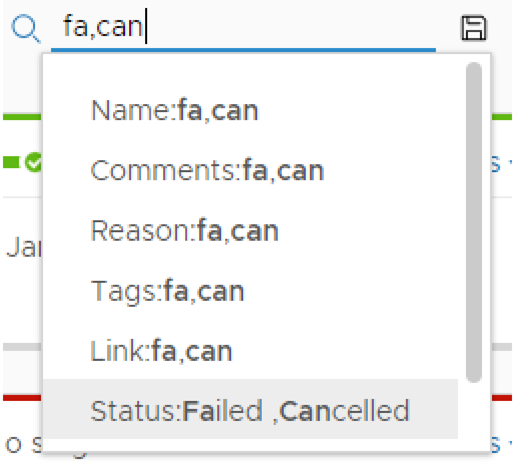
    Example: **`output.BuildNo:29`**
  - Notation: `statusMessage:<value>`

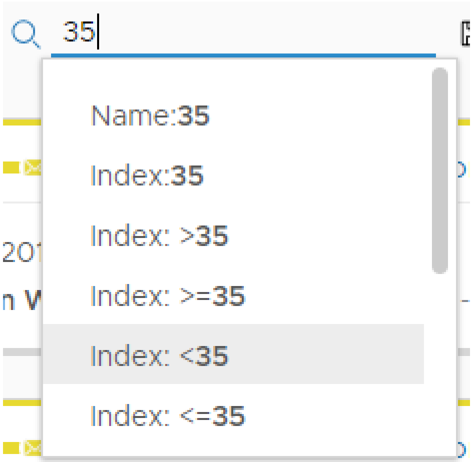
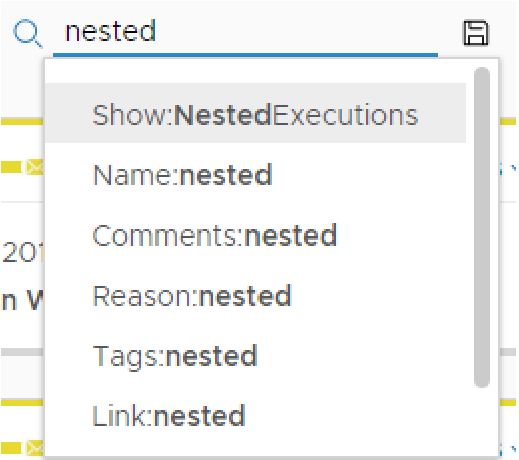
    Example: **`statusMessage:Execution failed`**

- Status or state depends on the search page.
  - For executions, possible values include: completed, failed, rollback_failed, or canceled.
  - For pipelines, possible state values include: enabled, disabled, or released.
  - For triggers, possible status values include: enabled or disabled.
- Executed, Created, or Updated by me refers to me, the logged in user.

Search appears at the upper right of every valid page. When you start typing into the search blank, Code Stream knows the context of the page and suggests options for the search.

Methods you can use to search	How to enter it
Type a portion of the search parameter. For example, to add a status filter that lists all the enabled pipelines, type **`ena`**.	 ena Status:**Ena**bled Name:**ena** Description:**ena** Tags:**ena** Link:**ena** Updated by:**ena**
To reduce the number of items found, add a filter. For example, type **`Tes`** to add a name filter. The filter works as an `AND` with the existing **Status:disabled** filter to show only the deactivated pipelines with **`Tes`** in the name. When you add another filter, the remaining options appear: **Name**, **Description**, **Tags**, **Link**, **Project**, and **Updated by**.	 Status Disabled Tes Name:Tes Description:Tes Tags:Tes Link:Tes Project:SHOBHA-TEST-PROJ Updated by:Tes

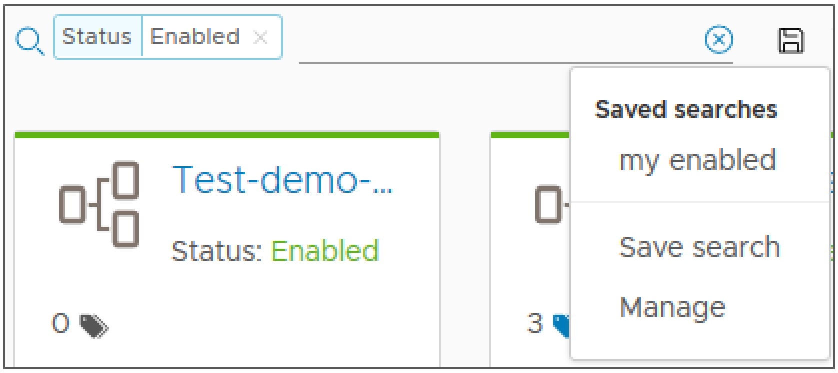
Methods you can use to search	How to enter it
To reduce the number of items displayed, click the filter icon on properties of a pipeline or a pipeline execution. ■ For pipelines, **Status**, **Tags**, **Project**, and **Updated by** each have a filter icon. ■ For executions, **Tags**, **Executed by**, and **Status Message** each have a filter icon. For example on the pipeline card, click the icon to add the filter for the **SmartTemplate** tag to the existing filters for: **Status:Enabled**, **Project:test**, **Updated by:user** and **Tags:Canary**.	 Status Enabled × Project test × Updated by × Tags Canary × test-nested-ci Status: Enabled 2 Canary SmartTemplate 2 Tags Canary SmartTemplate Updated By 3 EXECUTIONS OPEN RUN ACTIONS
Use a comma separator to include all items in two execution states. For example, type `fa,can` to create a status filter that works as an `OR` to list all failed or canceled executions.	 fa,can Name:**fa,can** Comments:**fa,can** Reason:**fa,can** Tags:**fa,can** Link:**fa,can** Status:**Failed** ,**Cancelled**

Methods you can use to search	How to enter it
Type a number to include all items within an index range. For example, type **35** and select **<** to list all executions with an index number less than 35.	 35 Name:35 Index:35 Index: >35 Index: >=35 Index: <35 Index: <=35
Pipelines that are modeled as tasks become nested executions and are not listed with all executions by default. To show nested executions, type `nested` and select the **Show** filter.	 nested Show:NestedExecutions Name:nested Comments:nested Reason:nested Tags:nested Link:nested

How do I save a favorite search

You can save favorite searches to use on each page by clicking the disk icon next to the search area.

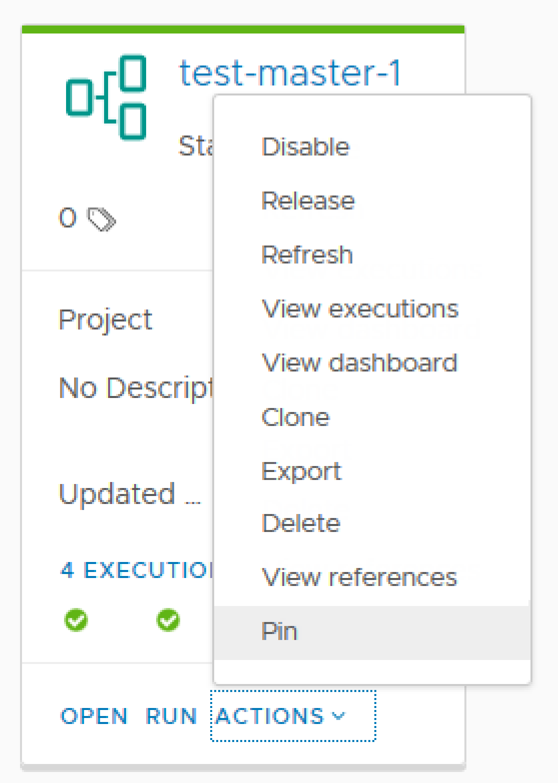
- You save a search by typing the parameters for search and clicking the icon to give the search a name such as `my enabled`.
- After saving a search, you click the icon to access the search. You can also select **Manage** to rename, delete, or move the search in the list of saved searches.



Searches are tied to your user name and only appear on the pages for which the search applies. For example, if you saved a search named `my enabled` for **Status:enabled** on the pipelines page, the `my enabled` search is not available on the Gerrit triggers page, even though **Status:enabled** is a valid search for a trigger.

Can I save a favorite pipeline

If you have a favorite pipeline or dashboard, you can pin it so that it always appears at the top of your pipelines or dashboards page. On the pipeline card, click **Actions > Pin**.



More resources for Code Stream Administrators and Developers

As a Code Stream administrator or developer, you can learn more about Code Stream.

Table 9-1. More resources for administrators

To learn about...	See these resources...
Other ways administrators can use Code Stream: - Configure pipelines to automate the testing and release of cloud native applications. - Automate and test developer source code, through testing, to production. - Configure pipelines for developers to test changes before they commit them to the primary branch. - Track key pipeline metrics.	Code Stream - vRealize Automation Documentation - vRealize Automation product website VMware Hands On - Use the vRealize Automation Community. - Use the VMware Learning Zone. - Search the VMware Blogs. - Try the VMware Hands On Labs.

Table 9-2. More resources for developers

To learn about...	See these resources...
Other ways developers can use Code Stream: - Use public and private registry images to build environments for new applications or services. - Set up development environments so that you can create branches from the latest stable build. - Update development environments with the latest code changes and artifacts. - Test uncommitted code changes against the latest stable builds of other dependent services. - Receive a notification when a change committed to a primary CICD pipeline breaks other services.	Code Stream - vRealize Automation Documentation - vRealize Automation product website VMware Hands On - Use the vRealize Automation Community. - Use the VMware Learning Zone. - Search the VMware Blogs. - Try the VMware Hands On Labs.