

Using and Managing vRealize Automation Code Stream

14 December 2022

vRealize Automation 8.2

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

Copyright © 2022 VMware, Inc. All rights reserved. [Copyright and trademark information.](#)

Contents

1	What is vRealize Automation Code Stream and how does it work	5
2	Setting up to model my release process	10
	How do I add a project	14
	How do I manage user access and approvals	15
	What are user operations and approvals	23
3	Creating and using pipelines	25
	How do I run a pipeline and see results	28
	What task types are available	33
	How do I do use variable bindings in pipelines	36
	How do I use variable bindings in a condition task to run or stop a pipeline	46
	What variables and expressions can I use when binding pipeline tasks	49
	How do I send notifications about my pipeline	66
	How do I create a Jira ticket when a pipeline task fails	68
	How do I roll back my deployment	71
4	Planning to natively build, integrate, and deliver your code	77
	Planning a CICD native build before using the smart pipeline template	77
	Planning a CI native build before using the smart pipeline template	82
	Planning a CD native build before using the smart pipeline template	83
	Planning a CICD native build before manually adding tasks	84
	Planning for rollback	91
5	Tutorials	94
	How do I continuously integrate code from my GitHub or GitLab repository into my pipeline	95
	How do I automate the release of an application that I deploy from a YAML cloud template	99
	How do I automate the release of an application to a Kubernetes cluster	107
	How do I deploy my application to my Blue-Green deployment	115
	How do I integrate my own build, test, and deploy tools	119
	How do I use a REST API to integrate with other applications	128
6	Connecting to endpoints	133
	What are Endpoints	133
	How do I integrate with Jenkins	135
	How do I integrate with Git	141
	How do I integrate with Gerrit	143
	How do I integrate with vRealize Orchestrator	145

7 Triggering pipelines 151

How do I use the Docker trigger to run a continuous delivery pipeline 151

How do I use the Git trigger to run a pipeline 159

How do I use the Gerrit trigger to run a pipeline 166

8 Monitoring pipelines 174

How do I track key performance indicators for my pipeline 174

9 Learn more 178

What is Search 178

More resources for Administrators and Developers 183

What is vRealize Automation Code Stream and how does it work

1

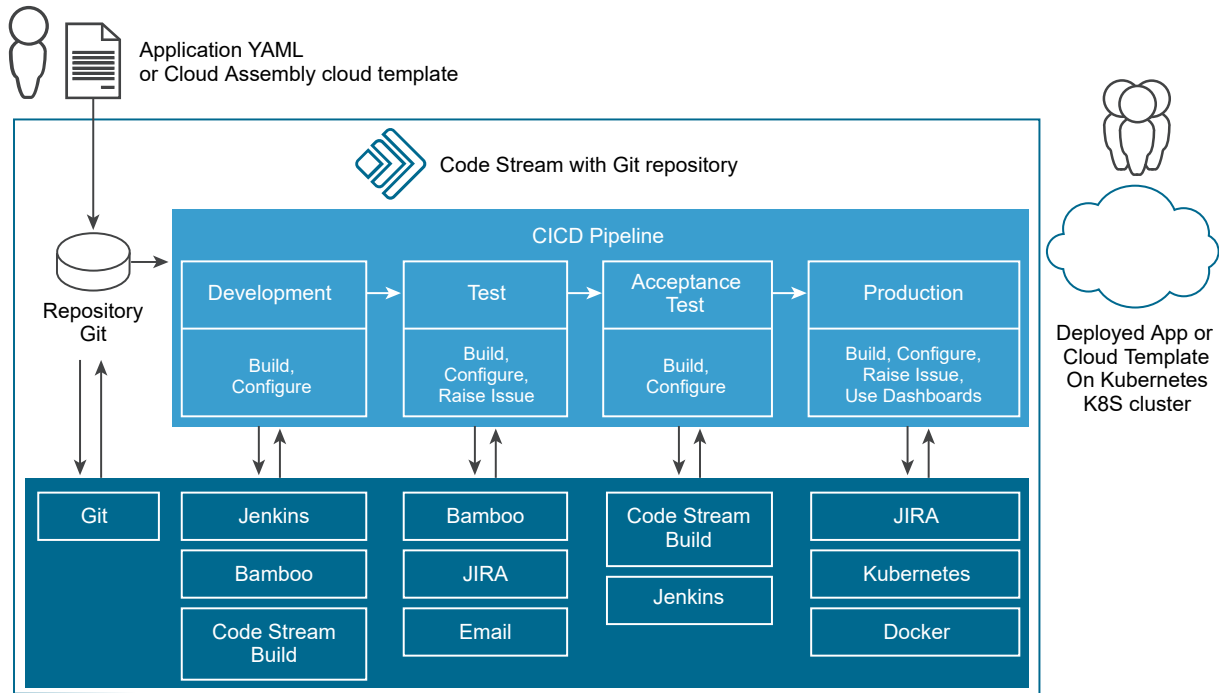
vRealize Automation Code Stream™ is a continuous integration and continuous delivery (CI/CD) tool that you use to build pipelines that model the software release process in your DevOps lifecycle. By creating pipelines, you build the code infrastructure that delivers your software rapidly and continuously.



When you use vRealize Automation 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 vRealize Automation 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. vRealize Automation 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 vRealize Automation Code Stream](#).

How vRealize Automation Code Stream Administrators use vRealize Automation 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 vRealize Automation Code Stream](#).

Table 1-1. How vRealize Automation Code Stream Administrators support developers

To support developers...	Here's what you can do...
Provide and manage environments.	<p>Create environments for developers to test and deploy their code.</p> <ul style="list-style-type: none"> ■ Track status and send email notifications. ■ Keep your developers productive by ensuring that their environments continuously work. <p>To find out more, see More resources for vRealize Automation Code Stream Administrators and Developers.</p> <p>Also see Chapter 5 Tutorials for using vRealize Automation Code Stream.</p>
Provide endpoints.	<p>Ensure that developers have working instances of endpoints that can connect to their pipelines.</p>
Provide integrations with other services.	<p>Ensure that integrations to other services are working.</p> <p>To find out more, see vRealize Automation documentation.</p>

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

To support developers...	Here's what you can do...
Create pipelines.	<p>Create pipelines that model release processes.</p> <p>To find out more, see Chapter 3 Creating and using pipelines in vRealize Automation Code Stream.</p>
Trigger pipelines.	<p>Ensure that pipelines run when events occur.</p> <ul style="list-style-type: none"> ■ 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. <p>To find out more, see Chapter 7 Triggering pipelines in vRealize Automation Code Stream.</p>
Manage pipelines and approvals.	<p>Stay up-to-date on pipelines.</p> <ul style="list-style-type: none"> ■ View pipeline status, and see who ran the pipelines. ■ View approvals on pipeline executions, and manage approvals for active and inactive pipeline executions. <p>To find out more, see What are user operations and approvals in vRealize Automation Code Stream.</p> <p>Also, see How do I track key performance indicators for my pipeline in vRealize Automation Code Stream.</p>
Monitor developer environments.	<p>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.</p> <p>You can also see:</p> <ul style="list-style-type: none"> ■ 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. <p>To find out more, see Chapter 8 Monitoring pipelines in vRealize Automation Code Stream.</p>
Troubleshoot problems.	<p>Troubleshoot and resolve pipeline failures in developer environments.</p> <ul style="list-style-type: none"> ■ 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 vRealize Automation Code Stream. <p>Also, see Chapter 2 Setting up vRealize Automation Code Stream to model my release process.</p>

vRealize Automation Code Stream is part of vRealize Automation. vRealize Automation Code Stream integrates with:

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

For other things you can do, see [VMware vRealize Automation Documentation](#).

How Developers Use vRealize Automation Code Stream

As a developer, you use vRealize Automation 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 vRealize Automation Code Stream](#).

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


- Did my code succeed through all stages of the pipeline? Look at the results in **Executions**.
- What do I do if the pipeline failed, and what caused the failure? Look at the top errors that occurred in **Dashboards**.

Table 1-2. Developers who use vRealize Automation 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 to get 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 vRealize Automation 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 to release your software. vRealize Automation Code Stream then automates the process that builds, tests, approves, and deploys your code.

Now that you have everything in place to model your software release process, here's how you do it in vRealize Automation Code Stream.

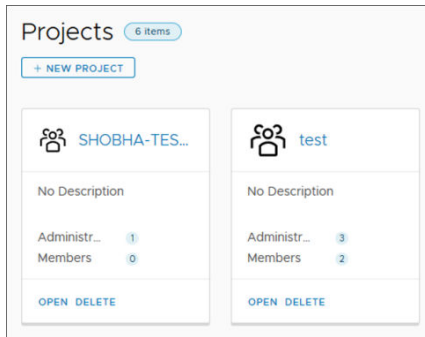
Prerequisites

- Verify whether any endpoints are already available. In vRealize Automation Code Stream, click **Endpoints**.
- Learn about native ways to build and deploy your code. See [Chapter 4 Planning to natively build, integrate, and deliver your code in vRealize Automation 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 viewer role instead of the administrator role, determine who is the administrator for your vRealize Automation Code Stream instance.

Procedure

- 1 Examine the projects available in vRealize Automation Code Stream and select one that is right for you.
 - If no projects are listed, ask a vRealize Automation Code Stream administrator to create a project and make you a member of the project. See [How do I add a project in vRealize Automation Code Stream](#).

- If you are not a member of any projects listed, ask a vRealize Automation Code Stream administrator to 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.

Use restricted variables to constrain the resources used in your pipelines, such as a host machine. 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 might be defined as `HostIPAddress`. Then, 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 vRealize Automation Code Stream needs so that it can natively build, integrate, test, and deploy your code.

How to create a vRealize Automation Code Stream native build...

Use one of the smart pipeline templates.

Results in this build strategy...

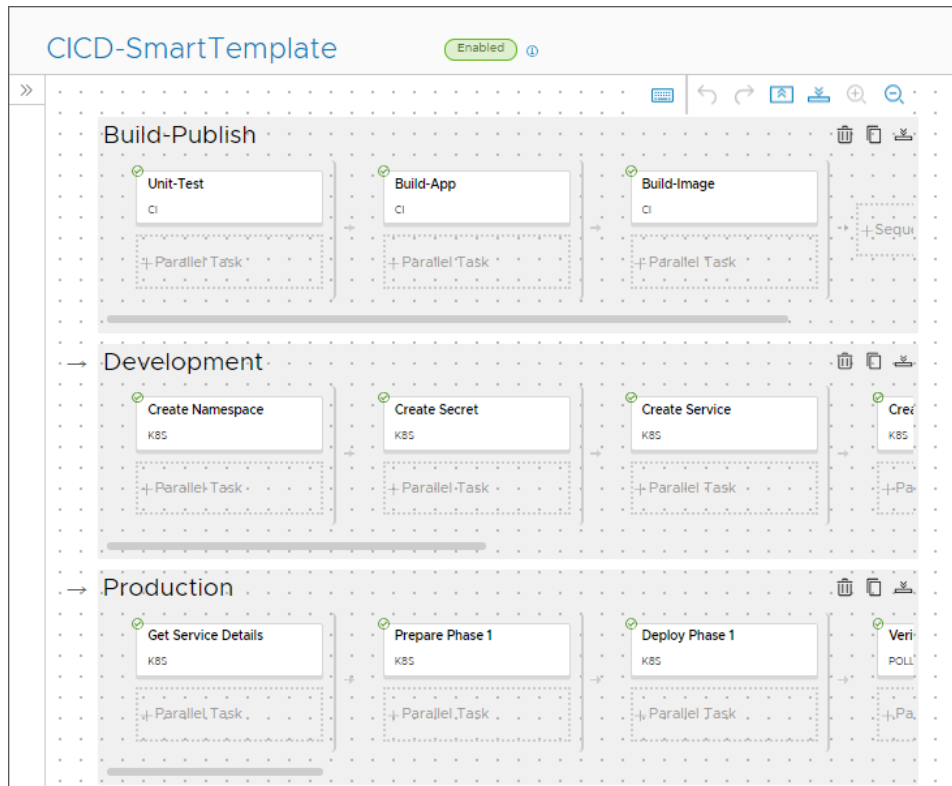
- Builds all the stages and tasks for you.
- Clones the source repository.
- Builds and tests your code.
- Containerizes your code for deployment.
- Populates the pipeline task steps based on your selections.

Add stages and tasks manually.

You add stages and 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](#)

Icon	Name	Status	Stages	Actions	User	Time	Input	Output
	Demo-Jenkins... #95	COMPLETED	<div><div></div></div>	ACTIONS	kr	09/11/2018 10:32 AM	8df0d9a1d365299f2...	NA
	Demo-Jenkins... #94	COMPLETED	<div><div></div></div>	ACTIONS	kr	09/11/2018 9:17 AM	6d82d079a8b8921a9...	NA
	Demo-CICD-S... #51	COMPLETED	<div><div></div></div>	ACTIONS	dk	09/11/2018 7:13 AM	NA	NA
	Demo-CICD-S... #50	FAILED	<div><div></div></div>	ACTIONS	dk	09/11/2018 5:51 AM	NA	NA

Execution failed on task 'Production.Deploy Phase 1'. deployments...

9 To track status and 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	<div><div></div></div>	<div><div></div></div>	<div><div></div></div>
#45	<div><div></div></div>	<div><div></div></div>	<div><div></div></div>
#44	<div><div></div></div>	<div><div></div></div>	<div><div></div></div>
#43	<div><div></div></div>	<div><div></div></div>	<div><div></div></div>
#42	<div><div></div></div>	<div><div></div></div>	<div><div></div></div>
#41	<div><div></div></div>	<div><div></div></div>	<div><div></div></div>
#40	<div><div></div></div>	<div><div></div></div>	<div><div></div></div>
#39	<div><div></div></div>	<div><div></div></div>	<div><div></div></div>
#38	<div><div></div></div>	<div><div></div></div>	<div><div></div></div>
#37	<div><div></div></div>	<div><div></div></div>	<div><div></div></div>

● Completed ● Failed ● Running ● Waiting

Results

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

You can also export your pipeline YAML to import and reuse 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 vRealize Automation Code Stream](#).

How do I add a project in vRealize Automation Code Stream

You create a project to which you add administrators and members so that the 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 vRealize Automation Code Stream administrator.

A project must exist before you can create a pipeline. When you create a pipeline, you select the project to associate it with so that all your pipeline information is grouped together. Endpoint and variable definitions also depend on an existing project.

Prerequisites

- Verify that you have the vRealize Automation Code Stream administrator role. See [What are Roles in vRealize Automation Code Stream](#).

If you do not have the vRealize Automation Code Stream administrator role, but you have vRealize Automation Cloud Assembly administrator role, you can create, update, or delete projects in the vRealize Automation Cloud Assembly UI. See "How do I add a project for my vRealize Automation 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 edit group role assignments in vRealize Automation" 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 with assigned 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 vRealize Automation Code Stream](#).
- 6 Click **Save**.

What to do next

Add endpoints and pipelines that use the project. See [Chapter 6 Connecting vRealize Automation Code Stream to endpoints](#) and [Chapter 3 Creating and using pipelines in vRealize Automation Code Stream](#).

How do I manage user access and approvals in vRealize Automation Code Stream

vRealize Automation 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 vRealize Automation Code Stream

Depending on your role in vRealize Automation 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 vRealize Automation Code Stream

vRealize Automation Code Stream Roles					
Access levels	Code Stream Administrator	Code Stream Developer	Code Stream Executor	Code Stream Viewer	Code Stream User
vRealize Automation 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 vRealize Automation Code Stream (continued)

vRealize Automation 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. vRealize Automation 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				
	Update pipelines	Yes	Yes			

Table 2-2. vRealize Automation 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
Pipeline Executions	Delete pipelines	Yes	Yes			
	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				
Dashboards	View dashboards	Yes	Yes	Yes	Yes	
	Create dashboards	Yes	Yes			
	Update dashboards	Yes	Yes			
	Delete dashboards	Yes	Yes			

Custom roles and permissions in vRealize Automation Code Stream

You can create custom roles in vRealize Automation Cloud Assembly that extend privileges to users who work with pipelines. When you create a custom role for vRealize Automation 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		
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	<ul style="list-style-type: none"> ■ 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	<ul style="list-style-type: none"> ■ 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	<ul style="list-style-type: none"> ■ Create and update custom integrations. ■ Version and release custom integrations. ■ Delete and deprecate custom integration versions. ■ Delete custom integrations.
Execute Pipelines	<ul style="list-style-type: none"> ■ 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	<ul style="list-style-type: none"> ■ 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	<ul style="list-style-type: none"> ■ 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 vRealize Automation Code Stream.

For example, if you create a custom role and include the permission called **Manage Restricted Pipelines**, users who have the vRealize Automation 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 vRealize Automation 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 vRealize Automation 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 vRealize Automation 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 vRealize Automation Code Stream administrator or project administrator must resume the pipeline.

If you have the User role

You can access vRealize Automation 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.

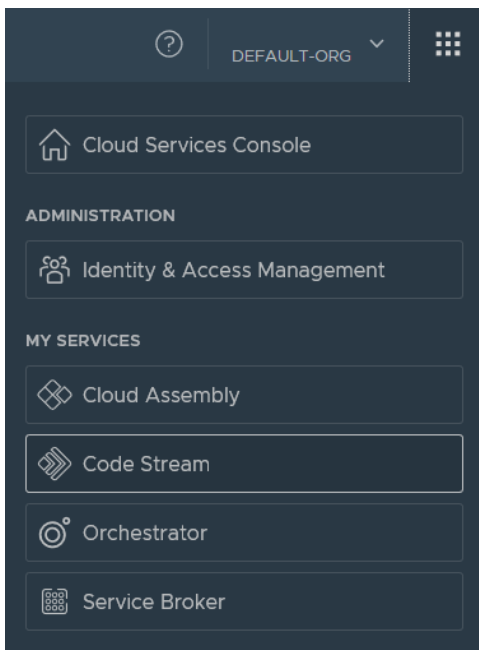
If you have the Executor role

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**.



- 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 vRealize Automation Code Stream

The User Operations area displays pipeline runs that need approval. The user who is required as an 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.

The required approver for a pipeline task must have the permission and expertise to determine whether to approve it.

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.

After the required user approves the user operation 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 GUIDED SETUP

Active Items Inactive Items

✓ APPROVE ✗ REJECT

<input type="checkbox"/>	Index#	Execution	Summary	Requested By	Request Date	Approvers
<input type="checkbox"/>	> c07b12	Demo2-Jenkins-K8s#7	Testing	fritz	Nov 13, 2019, 11:32:31 AM	fritz@vmware.com
<input type="checkbox"/>	> a0a990	Demo2-Jenkins-K8s#6	Testing	fritz	Nov 11, 2019, 1:34:11 PM	kevin@vmware.com, fritz@vmware.com
<input checked="" type="checkbox"/>	▼	User Operation #8f1728 <hr/> Request Details Execution: Demo-Jenkins-K8s #5 Summary: Testing Approvers: kevin@vmware.com, fritz@vmware.com Requested By: fritz Requested On: Nov 11, 2019, 1:22:21 PM Expires On: Nov 14, 2019, 1:22:21 PM				

1 items Items per page: 20 1 - 7 of 7 items

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 to 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 vRealize Automation 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 actions required to deliver your software application through the stages.

What are Pipelines in vRealize Automation Code Stream

A pipeline is a continuous integration and delivery model of your software release process. It releases your software from source code, through testing, to production. It includes a sequence of stages with 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 CI/CD native build in vRealize Automation 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 vRealize Automation 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 to build, test, deploy, and release 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	<p>In a development stage, you can provision a machine, retrieve an artifact, add a build task to create a Docker host to use for continuous integration of your code, and more.</p> <p>For example:</p> <ul style="list-style-type: none"> ■ 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 CI native build in vRealize Automation Code Stream before using the smart pipeline template.
Test	<p>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.</p> <p>For example:</p> <ul style="list-style-type: none"> ■ 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 vRealize Automation 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 vRealize Automation Code Stream. ■ Track trends on post-processing for a continuous integration (CI) pipeline. See How do I track key performance indicators for my pipeline in vRealize Automation Code Stream.
Production	<p>In a production stage, you can integrate with cloud template in vRealize Automation Cloud Assembly to provision your infrastructure, deploy your software to a Kubernetes cluster, and more.</p> <p>For example:</p> <ul style="list-style-type: none"> ■ 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 vRealize Automation 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 vRealize Automation 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 vRealize Automation 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 vRealize Automation 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 vRealize Automation Code Stream when a pipeline task fails.](#)

Triggering pipelines

Pipelines can trigger when developers check in or review code, or when a build artifact is created or updated.

- 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 vRealize Automation 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 vRealize Automation 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 vRealize Automation 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 vRealize Automation Code Stream.](#)

This chapter includes the following topics:

- [How do I run a pipeline and see results](#)
- [What task types are available in vRealize Automation Code Stream](#)
- [How do I use variable bindings in vRealize Automation Code Stream pipelines](#)
- [How do I use variable bindings in a condition task to run or stop a pipeline in vRealize Automation Code Stream](#)
- [What variables and expressions can I use when binding pipeline tasks in vRealize Automation Code Stream](#)
- [How do I send notifications about my pipeline in vRealize Automation Code Stream](#)
- [How do I create a Jira ticket in vRealize Automation Code Stream when a pipeline task fails](#)
- [How do I roll back my deployment in vRealize Automation Code Stream](#)

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 vRealize Automation 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 vRealize Automation 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 can have Jenkins run a pipeline after a Jenkins build finishes.

For more information about using the triggers, see [Chapter 7 Triggering pipelines in vRealize Automation 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

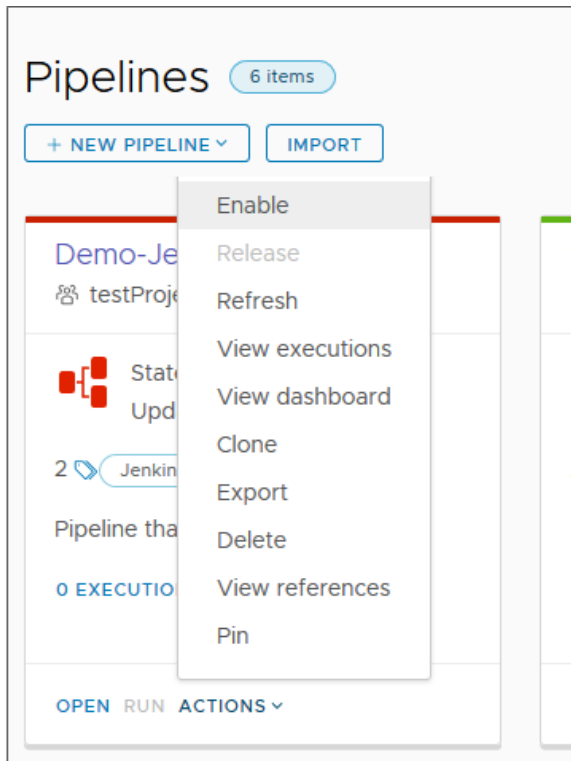
- Verify that one or more pipelines are created. See the examples in [Chapter 5 Tutorials for using vRealize Automation Code Stream](#).

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**.



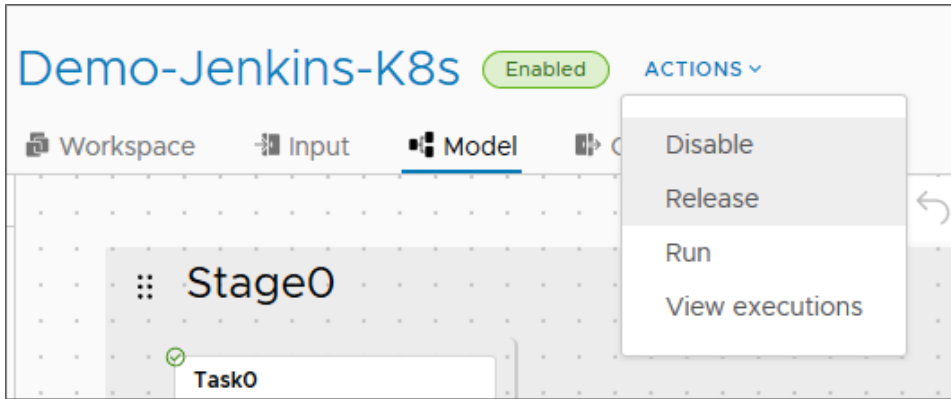
You can also enable your pipeline while you are in the pipeline. If your pipeline is already enabled, **Run** is active, and the **Actions** menu displays **Disable**.

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 vRealize Automation 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 vRealize Automation Service Broker to add the pipeline as a catalog item and run it. See how to add vRealize Automation Code Stream pipelines to the vRealize Automation Service Broker catalog in *Using and Managing VMware Service Broker*.

Note If the pipeline requires more than 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 vRealize Automation Service Broker as administrator and select **Infrastructure > Projects**. Click your project name and then click **Provisioning**.

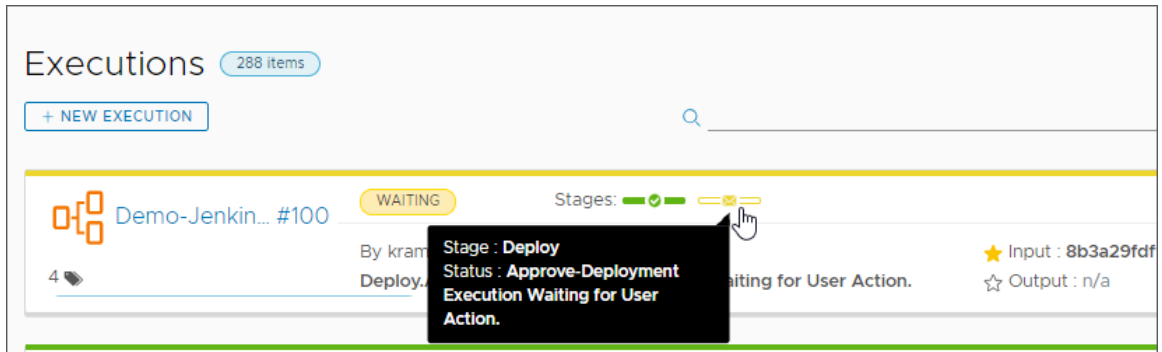
If the request timeout value is not set, an execution that requires more than 120 minutes to run appears as failed with a callback timeout request error. However, the pipeline execution is not affected.

- 3 On the pipeline card, click **Run**.
- 4 To view the pipeline as it runs, click **Executions**.

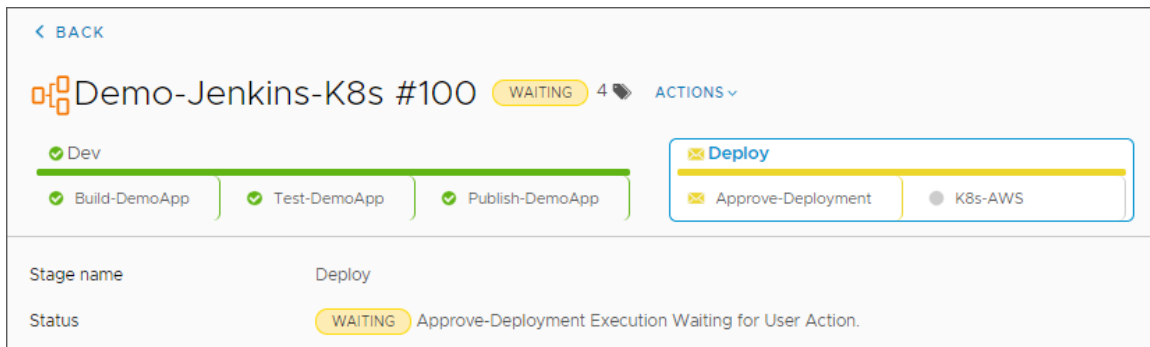
The pipeline runs each stage in sequence, and the pipeline execution displays a status icon for each stage. If the pipeline includes a user operation task, a user must approve the task for the pipeline to continue to run. When a user operation task is used, the pipeline stops running and waits for the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



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

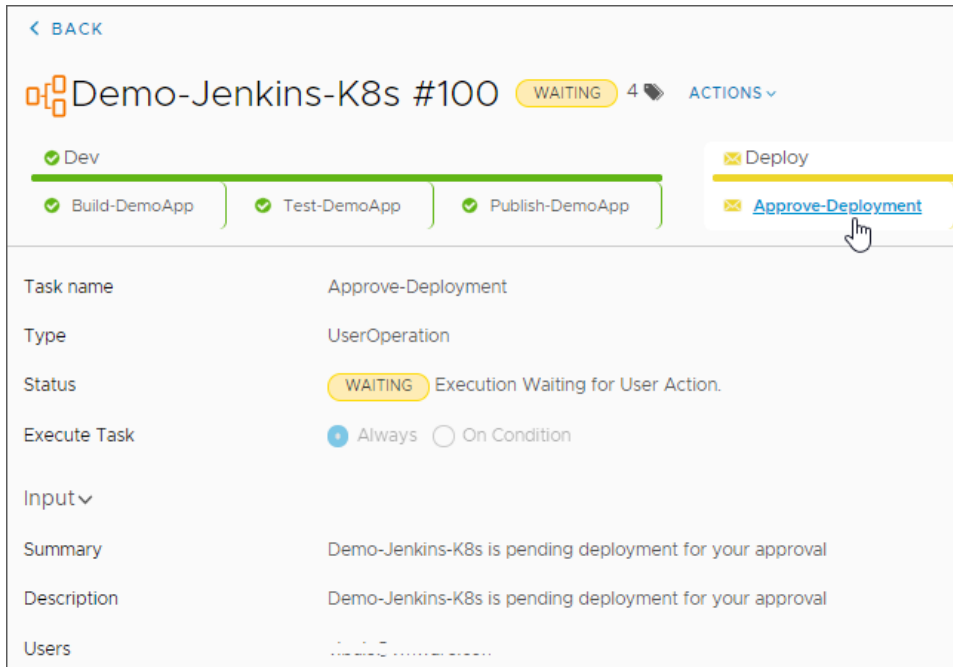


- To see the details for the task, click the task.

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 vRealize Automation 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.



- 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 administrator 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 execution. Then, click the tab at the top of the execution, and select an execution.



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 vRealize Automation Code Stream to automate your software release cycle, see [Chapter 5 Tutorials for using vRealize Automation Code Stream](#).

What task types are available in vRealize Automation Code Stream

You can configure your pipeline to perform certain actions by adding specific task types to it. Each task type integrates with another application to enable your pipeline to accomplish what you design it to deliver.

Whether you need to pull artifacts from a repository for deployment, run a remote script, or require approval from a team member for your pipeline to run, vRealize Automation Code Stream has the task type for you!

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

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

Task type...	What it does...	Examples and details...
User Operation	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 vRealize Automation 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 vRealize Automation Code Stream.

Table 3-3. Automate continuous integration and deployment

Task type...	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.	<p>See How do I automate the release of an application that I deploy from a YAML cloud template in vRealize Automation Code Stream.</p> <p>When you select Create or Update in the cloud template task, then select Cloud Template and Version, the cloud template parameters appear. You can add these elements, which accommodate variable bindings, to the input text areas in the cloud template task:</p> <ul style="list-style-type: none"> ■ Integer ■ Enumeration string ■ Boolean ■ Array variable <p>When you use variable binding in the input fields, 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.</p> <p>The cloud template parameter appears in the cloud template task when a cloud template in vRealize Automation Cloud Assembly includes input variables. For example, if a cloud template has an input type of <code>Integer</code>, you can enter the integer directly or as a variable by using variable binding.</p>
CI	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.	See Planning a CICD native build in vRealize Automation Code Stream before using the smart pipeline template .
Custom	Integrates vRealize Automation Code Stream with your own build, test, and deploy tools.	See How do I integrate my own build, test, and deploy tools with vRealize Automation Code Stream .

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

Task type...	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 vRealize Automation Code Stream to a Kubernetes cluster.
Pipeline	<p>Nests a pipeline in a primary pipeline. When a pipeline is nested, it behaves as a task in the primary pipeline.</p> <p>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.</p>	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 vRealize Automation 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.	vRealize Automation Code Stream supports Team Foundation Server 2013 and 2015.
vRO	Extends the capability of vRealize Automation Code Stream by running predefined or custom workflows in vRealize Orchestrator.	See How do I integrate vRealize Automation Code Stream with vRealize Orchestrator.

Table 3-5. Integrate other applications through an API

Task type...	What it does...	Examples and details...
REST	Integrates vRealize Automation 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 vRealize Automation Code Stream with other applications.
Poll	Invokes a REST API and polls it until the pipeline task meets the exit criteria and completes.	See How do I use a REST API to integrate vRealize Automation Code Stream with other applications.

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

Task type...	What it does...	Examples and details...
PowerShell	<p>Allows the PowerShell script task type to run script commands on a remote host. For example, a script can automate test tasks, and run administrative types of commands.</p> <p>The script can be remote or user-defined. It can connect over HTTP or HTTPS, and can use TLS.</p> <p>The service named <code>winrm</code> must be configured on the Windows host, and <code>winrm</code> must be configured for <code>MaxShellsPerUser</code> and <code>MaxMemoryPerShellMB</code>.</p>	<p>When you configure <code>MaxShellsPerUser</code> and <code>MaxMemoryPerShellMB</code>:</p> <ul style="list-style-type: none"> ■ The acceptable value for <code>MaxShellsPerUser</code> is 500 for 50 concurrent pipelines, with 5 PowerShell tasks for each pipeline. To set the value, run: <code>winrm set winrm/config/winrs '@{MaxShellsPerUser="500"}'</code> ■ The acceptable memory value for <code>MaxMemoryPerShellMB</code> is 2048. To set the value, run: <code>winrm set winrm/config/winrs '@{MaxMemoryPerShellMB="2048"}'</code> <p>The script writes the output to a response file that another pipeline can consume.</p>
SSH	<p>Allows the Bash shell script task type to run script commands on a remote host. For example, a script can automate test tasks, and run administrative types of commands.</p> <p>The script can be remote or user-defined. It can connect over HTTP or HTTPS, and requires a private key or password.</p> <p>The SSH service must be configured on the Linux host, and the SSHD configuration of <code>MaxSessions</code> must be set to 50.</p>	<p>The script can be remote or user-defined. For example, a script might resemble:</p> <pre>message="Hello World" echo \$message</pre> <p>The script writes the output to a response file that another pipeline can consume.</p>

How do I use variable bindings in vRealize Automation 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 vRealize Automation Code Stream pipeline cloud template task. The way you modify the variables in vRealize Automation Code Stream depends on how the variable properties are coded in the cloud template.

If you need to 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 vRealize Automation 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 property types that the vRealize Automation 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 vRealize Automation 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 vRealize Automation Code Stream Executor role and models the pipeline, the developer must also have the same vRealize Automation Cloud Assembly role of the person who deployed the cloud template. For example, the required role might be vRealize Automation Cloud Assembly Administrator.
- Only the person who models the pipeline has the permission to create the pipeline and create the deployment.

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 vRealize Automation Code Stream Executor role. Then, when the Executor runs the pipeline, it will use the API token and the credentials that the API token creates.
- When a user enters the an API token in the cloud template task, it creates the credentials needed for the pipeline to run.
- 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`. The image properties are 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 vRealize Automation Code Stream pipeline, and add dollar bindings to it, here's what you'll do.

- 1 In vRealize Automation 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 can provide credentials for the pipeline to run, and that you can use **Create Variable** to create a variable that encrypts the API token in the cloud template task.
- 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 you need to change the cloud template in vRealize Automation Cloud Assembly. For example, you:
 - a Set the `flavor` to use a property of type `array`. vRealize Automation 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 will 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 vRealize Automation Code Stream, select the new version of the cloud template, and enter values in the **Parameter and Value** table. Cloud templates support the following parameter types and the ability for vRealize Automation Code Stream to allow dollar variable bindings to them, which results in slight differences in the vRealize Automation Code Stream cloud template task user interface and vRealize Automation Cloud Assembly cloud template interface. Depending on how a cloud template is coded, you might or might not be allowed to enter values in the task.
 - 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

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

Output Parameter

- name
- description
- Stage0
- status
- deploymentid
- deploymentname
- deploymenturl
- deploymentversion

- 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 vRealize Automation 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 vRealize Automation Code Stream pass them to the pipeline. Then, when the pipeline runs, a user must enter the value for the input parameter. You can add output parameters to your pipeline so that pipeline tasks can use the output value from a task. vRealize Automation 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 needed to use a default value for the Git server URL, you would 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**.

The screenshot shows the configuration window for a task named 'Task3'. The interface includes tabs for 'Task :Task3', 'Notifications', and 'Rollback', with a 'VALIDATE TASK' button in the top right. The configuration fields are as follows:

- Task name:** Task3
- Type:** REST
- Continue on failure:** ☐
- Execute task:** ☒ Always ☐ On condition
- REST Request:**
 - Action:** GET
 - URL:** \${input.GIT_SERVER_URL} (with a dropdown menu showing various Git variables like GIT_BRANCH_NAME, GIT_CHANGE_SUBJECT, etc., and GIT_SERVER_URL selected)
 - Agent endpoint:**
 - Headers:**
- Output Parameters:** status, responseHeaders, responseBody, responseJson, responseCode

The entry resembles: `${input.GIT_SERVER_URL}`

- To verify the integrity of the variable binding for the task, click **Validate Task**.
vRealize Automation Code Stream indicates the task validated successfully.
- 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 and output parameters. Here's how.

Let's say you need your pipeline to call a URL in a REST task, and output a response. To do this, you include both input and output parameters in your REST task. You also need a user to approve the task, so you also include a User Operations task for another user to 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.

rest-ix-1 Enabled ACTIONS ▾

Workspace **Input** Model Output

Input Parameters ⓘ

Auto inject parameters ☐ Gerrit ☐ Git ☐ Docker ☒ None

ADD ADD/REMOVE INJECTED PARAMETERS

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

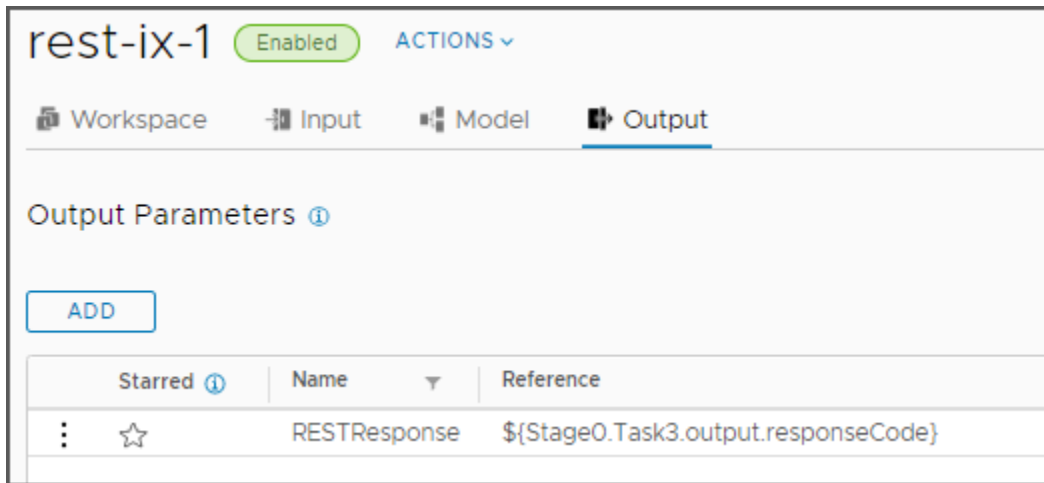
Name *

Reference \$ *

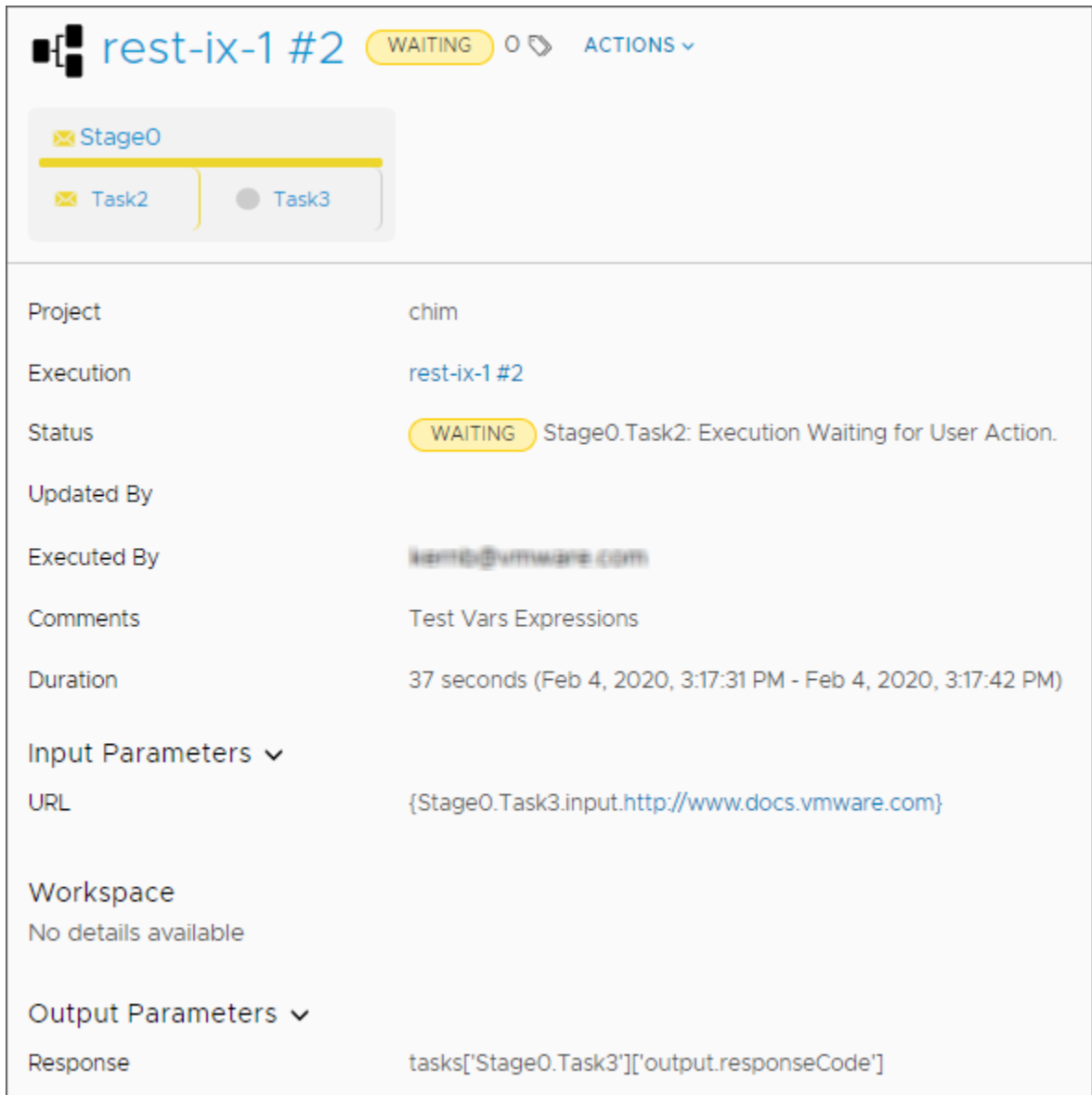
- responseHeaders
- responseBody
- responseJson
- responseCode

- 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**.



- 5 Save your pipeline.
- 6 From the **Actions** menu, click **Run**.
- 7 Click **Actions > View executions**.
- 8 Click the execution, and see the input and output parameters that you defined.



rest-ix-1 #2 WAITING 0 **ACTIONS** ▾

✉ **Stage0**

✉ **Task2** Task3

Project	chim
Execution	rest-ix-1 #2
Status	WAITING Stage0.Task2: Execution Waiting for User Action.
Updated By	
Executed By	kern@vmware.com
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']

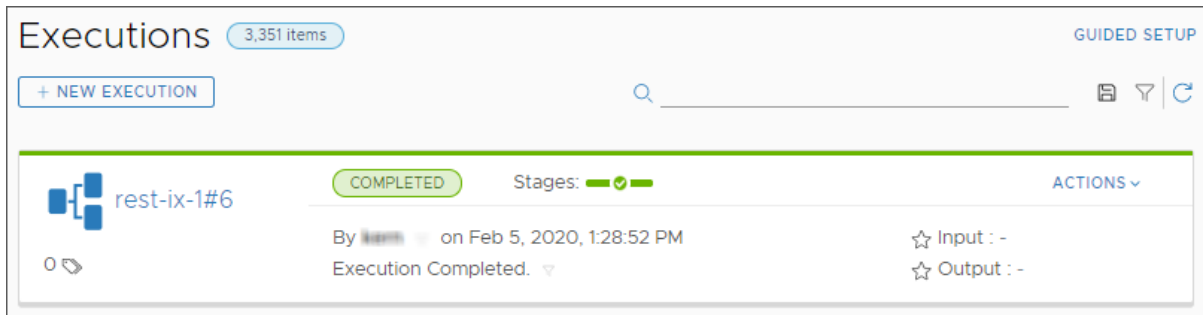
- 9 To approve the pipeline, click **User Operations**, and view the list of approvals on the **Active Items** tab. Or, stay in the Executions, click the task, and click **Approve**.
- 10 To enable the **Approve** and **Reject** buttons, click the check box next to the execution.
- 11 To see the details, expand the drop-down arrow.
- 12 To approve the task, click **APPROVE**, enter a reason, and click **OK**.

The screenshot shows the 'User Operations' interface. At the top right is a 'GUIDED SETUP' link. Below the title are tabs for 'Active Items' and 'Inactive Items'. There are two buttons: a green '✓ APPROVE' button and a red '✗ REJECT' button. A refresh icon is on the right. Below these is a table with columns for selection, 'Index#', and 'Execution'. One item is listed: 'User Operation #f0d252'. Below the table, the 'Request Details' are shown in a light blue box. The details include: Execution (rest-ix-1 #2), Summary (hello), Approvers (kern@vmware.com, f0f2@vmware.com), Requested By (kern@vmware.com), Requested On (Feb 4, 2020, 3:17:40 PM), and Expires On (Feb 7, 2020, 3:17:40 PM). At the bottom of the details box are three buttons: a green 'APPROVE' button, a red 'REJECT' button, and a blue 'VIEW DASHBOARD' link.

- 13 Click **Executions** and watch the pipeline continue to run.

The screenshot shows the 'Executions' interface. At the top right is a 'GUIDED SETUP' link. Below the title is a '3,347 items' badge. There is a '+ NEW EXECUTION' button, a search bar, and icons for save, filter, and refresh. Below these is a list of executions. The first execution is 'rest-i... #3' with a 'RUNNING' status badge. It shows a progress bar for 'Stages' and an 'ACTIONS' dropdown. Below the execution name, it says 'By kern on Feb 4, 2020, 3:41:05 PM'. There are two star icons: 'Input : -' and 'Output : -'. Below that, it says 'RUNNING' with a dropdown arrow and 'Comments: Testing' with a dropdown arrow.

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



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 vRealize Automation 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 vRealize Automation Code Stream](#).

How do I use variable bindings in a condition task to run or stop a pipeline in vRealize Automation 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 type.

You use the **Condition** task type 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 to continue to run.
- 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 vRealize Automation Code Stream pipelines](#).

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

Condition task type...	What it affects...	What it does...
Condition task	Pipeline	The Condition task type 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	<p>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 might require the Condition task output status to be Completed, or to use a build number of 74.</p> <p>The condition expression appears on the Task tab in the Condition task type.</p>

Task : Task2 Notifications Rollback

Task name ⓘ * **Task2**

Type * **Condition** ▾

Condition Task

Condition \$ ⓘ

Output

response

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

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

The screenshot shows the 'Task : Deploy Phase 1' configuration window. It has tabs for 'Task : Deploy Phase 1', 'Notifications', and 'Rollback'. A 'VALIDATE TASK' button is in the top right. The configuration fields are:

- Task name: Deploy Phase 1
- Type: K8S
- Continue on failure: ☐
- Execute task: ☐ Always, ☒ On condition
- Condition: An empty text box with an information icon on the right.

In other task types, the **On Condition** setting:

- 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.
- 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 pass or fail status.
- 3 Add a condition expression to the Condition task.

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

The screenshot shows the vRealize Automation interface. On the left, the 'Model' tab displays a pipeline diagram with 'Stage0' containing 'Task1' (a Condition task) and a '+Parallel Task'. On the right, the 'Task : Task1' configuration window is open. It has tabs for 'Task : Task1', 'Notifications', and 'Rollback'. A 'VALIDATE TASK' button is in the top right. The configuration fields are:

- 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 to run, or stops at the Condition task.

What to do next

You can also use the Condition task type if you decide to roll back a pipeline deployment. For example, in a rollback pipeline, the Condition task helps vRealize Automation 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 vRealize Automation Code Stream](#).

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

With variables and expressions, you can use input 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.

When you bind pipeline tasks together, you can include default and complex expressions so that 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 be resolved 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 vRealize Automation 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>}`.

In each expression, `SCOPE` is the context that vRealize Automation Code Stream uses to determine the behavior of a pipeline task. 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 to use them in your pipeline.

Table 3-8. Using SCOPE and KEY

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

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

SCOPE	Purpose of expression and example	KEY	How to use SCOPE and KEY in your pipeline
task input	Input to a task: \$ {MY_STAGE.MY_TASK.input. SOMETHING}	Indicates the input of a task in a notification	<p>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:</p> <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>
task output	Output of a task: \$ {MY_STAGE.MY_TASK.output .SOMETHING}	Indicates the output of a task in a subsequent task	<p>To refer to the output of pipeline task 1 in task 2, use this format:</p> <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>

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: <code>\${var.myVariable}</code>	Refer to variable in an endpoint	To refer to a secret variable in an endpoint for a password, use this format: <pre>--- project: MyProject kind: ENDPOINT name: MyJenkinsServer type: jenkins properties: url: https:// jenkins.example.com username: jenkinsUser password: \$ {var.jenkinsPassword}</pre>
var	Variable: <code>\${var.myVariable}</code>	Refer to variable in a pipeline	To refer to variable in a pipeline URL, use this format: <pre>tasks: task1: type: REST input: action: get url: \$ {var.MY_SERVER_URL}</pre>
task status	Status of a task: <pre>\$ {MY_STAGE.MY_TASK.status } \$ {MY_STAGE.MY_TASK.status Message}</pre>		
stage status	Status of a stage: <pre>\${MY_STAGE.status} \$ {MY_STAGE.statusMessage}</pre>		

Default Expressions

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

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

Using SCOPE and KEY in pipeline task types

You can use expressions with any of the supported pipeline task types. Use these examples as a reference to see how to define the `SCOPE` and `KEY`, and confirm the syntax. These code examples use `MY_STAGE` and `MY_TASK` as the pipeline stage and task names.

To find out more about available task types, see [What task types are available in vRealize Automation Code Stream](#).

Table 3-9. Gating tasks

Task type	Scope	Key	How to use SCOPE and KEY in the task
User Operation			
	Input	<p>summary: Summary of the request for the User Operation</p> <p>description: Description of the request for the User Operation</p> <p>approvers: List of approver email addresses, where each entry can be a variable with a comma, or use a semi-colon for separate emails</p> <p>approverGroups: List of approver group addresses for the platform and identity</p> <p>sendemail: Optionally sends an email notification upon request or response when set to true</p> <p>expirationInDays: Number of days that represents the expiry time of the request</p>	<pre> \${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} </pre>
	Output	<p>index: Six-digit hexadecimal string that represents the request</p> <p>respondedBy: Account name of the person who approved/rejected the User Operation</p> <p>respondedByEmail: Email address of the person who responded</p> <p>comments: Comments provided during response</p>	<pre> \${MY_STAGE.MY_TASK.output.index} \${MY_STAGE.MY_TASK.output.respondedBy} \$ {MY_STAGE.MY_TASK.output.respondedByEmail} \${MY_STAGE.MY_TASK.output.comments} </pre>
Condition			
	Input	<p>condition: Condition to evaluate. When the condition evaluates to true, it marks the task as complete, whereas other responses fail the task</p>	<pre> \${MY_STAGE.MY_TASK.input.condition} </pre>
	Output	<p>result: Result upon evaluation</p>	<pre> \${MY_STAGE.MY_TASK.output.response} </pre>

Table 3-10. Pipeline tasks

Task type	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	<pre> \${MY_STAGE.MY_TASK.input.name} \${MY_STAGE.MY_TASK.input.inputProperties} # Refer to all properties \$ {MY_STAGE.MY_TASK.input.inputProperties.inpu t1} # Refer to value of input1 </pre>
	Output	executionStatus: Status of the pipeline execution executionIndex: Index of the pipeline execution outputProperties: Output properties of a pipeline execution	<pre> \${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.ou tput1} # Refer to value of output1 </pre>

Table 3-11. Automate continuous integration tasks

Task type	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	<pre> \${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 </pre>
	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	<pre> \${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.processRespons e} \$ {MY_STAGE.MY_TASK.output.processRespons e[0].result} # Result of the first process configuration </pre>
Custom			

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

Task type	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	<pre> \${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 </pre>
	Output	properties: Output properties from the custom integration response	<pre> \${MY_STAGE.MY_TASK.output.properties} #Refer to all properties \$ {MY_STAGE.MY_TASK.output.properties.property1} #Refer to value of property1 </pre>

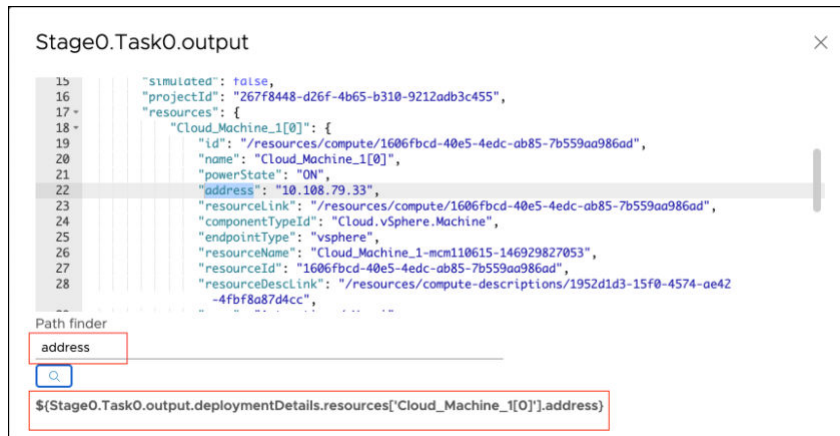
Table 3-12. Automate continuous deployment tasks: Cloud template

Task type	Scope	Key	How to use SCOPE and KEY in the task
Cloud template			
	Input	<p>action: One of createDeployment, updateDeployment, deleteDeployment, rollbackDeployment</p> <p>blueprintInputParams: Used for the create deployment and update deployment actions</p> <p>allowDestroy: Machines can be destroyed in the update deployment process.</p> <p>CREATE_DEPLOYMENT</p> <ul style="list-style-type: none"> ■ <code>blueprintName</code>: Name of the cloud template ■ <code>blueprintVersion</code>: Version of the cloud template <p>OR</p> <ul style="list-style-type: none"> ■ <code>fileUrl</code>: URL of the remote cloud template YAML, after selecting a GIT server. <p>UPDATE_DEPLOYMENT</p> <p>Any of these combinations:</p> <ul style="list-style-type: none"> ■ <code>blueprintName</code>: Name of the cloud template ■ <code>blueprintVersion</code>: Version of the cloud template <p>OR</p> <ul style="list-style-type: none"> ■ <code>fileUrl</code>: URL of the remote cloud template YAML, after selecting a GIT server. <p>-----</p> <ul style="list-style-type: none"> ■ <code>deploymentId</code>: ID of the deployment <p>OR</p> <ul style="list-style-type: none"> ■ <code>deploymentName</code>: Name of the deployment <p>-----</p> <p>DELETE_DEPLOYMENT</p> <ul style="list-style-type: none"> ■ <code>deploymentId</code>: ID of the deployment 	

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

Task type	Scope	Key	How to use SCOPE and KEY in the task
		<p>OR</p> <ul style="list-style-type: none"> ■ <code>deploymentName</code>: Name of the deployment <p>ROLLBACK_DEPLOYMENT</p> <p>Any of these combinations:</p> <ul style="list-style-type: none"> ■ <code>deploymentId</code>: ID of the deployment <p>OR</p> <ul style="list-style-type: none"> ■ <code>deploymentName</code>: Name of the deployment <p>-----</p> <ul style="list-style-type: none"> ■ <code>blueprintName</code>: Name of the cloud template ■ <code>rollbackVersion</code>: Version to roll back to 	
	Output		<p>Parameters that can bind to other tasks or to the output of a pipeline:</p> <ul style="list-style-type: none"> ■ Deployment Name can be accessed as <code>\${Stage0.Task0.output.deploymentName}</code> ■ Deployment Id can be accessed as <code>\${Stage0.Task0.output.deploymentId}</code> ■ Deployment Details is a complex object, and internal details can be accessed by using the JSON results. <p>To access any property, use the dot operator to follow the JSON hierarchy. For example, to access the address of resource <code>Cloud_Machine_1[0]</code>, the \$ binding is:</p> <pre>\$ {Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].address}</pre> <p>Similarly, for the flavor, the \$ binding is:</p> <pre>\$ {Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].flavor}</pre> <p>In the vRealize Automation Code Stream user interface, you can obtain the \$ bindings for any property.</p> <ol style="list-style-type: none"> 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:



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 type	Scope	Key	How to use SCOPE and KEY in the task
Kubernetes			
	Input	<p>action: One of GET, CREATE, APPLY, DELETE, ROLLBACK</p> <ul style="list-style-type: none"> ■ timeout: Overall timeout for any action ■ filterByLabel: Additional label to filter on for action GET using K8S labelSelector <p>GET, CREATE, DELETE, APPLY</p> <ul style="list-style-type: none"> ■ 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. <p>ROLLBACK</p> <ul style="list-style-type: none"> ■ 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 	<p><code>\${MY_STAGE.MY_TASK.input.action}</code> #Determines the action to perform.</p> <p><code>\${MY_STAGE.MY_TASK.input.timeout}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.filterByLabel}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.yaml}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.parameters}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.filePath}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.scmConstants}</code></p> <p><code>\$</code> <code>{MY_STAGE.MY_TASK.input.continueOnConflict}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.resourceType}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.resourceName}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.namespace}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.revision}</code></p>
	Output	<p>response: Captures the entire response</p> <p>response.<RESOURCE>: Resource corresponds to configMaps, deployments, endpoints, ingresses, jobs, namespaces, pods, replicaSets, replicationControllers, secrets, services, statefulSets, nodes, loadBalancers.</p> <p>response.<RESOURCE>.<KEY>: The key corresponds to one of apiVersion, kind, metadata, spec</p>	<p><code>\${MY_STAGE.MY_TASK.output.response}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.response.}</code></p>

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

Task type	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	<code>\${MY_STAGE.MY_TASK.input.plan}</code> <code>\${MY_STAGE.MY_TASK.input.planKey}</code> <code>\${MY_STAGE.MY_TASK.input.variables}</code> <code>\${MY_STAGE.MY_TASK.input.parameters}</code> # Refer to all parameters <code>\${MY_STAGE.MY_TASK.input.parameters.param1}</code> # 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	<code>\${MY_STAGE.MY_TASK.output.resultUrl}</code> <code>\${MY_STAGE.MY_TASK.output.buildResultKey}</code> <code>\${MY_STAGE.MY_TASK.output.buildNumber}</code> <code>\${MY_STAGE.MY_TASK.output.buildTestSummary}</code> # Refer to all results <code>\${MY_STAGE.MY_TASK.output.successfulTestCount}</code> # Refer to the specific test count <code>\${MY_STAGE.MY_TASK.output.buildNumber}</code>
Jenkins			
	Input	job: Name of the Jenkins job parameters: Parameters to be passed to the job	<code>\${MY_STAGE.MY_TASK.input.job}</code> <code>\${MY_STAGE.MY_TASK.input.parameters}</code> # Refer to all parameters <code>\${MY_STAGE.MY_TASK.input.parameters.param1}</code> # 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	<code>\${MY_STAGE.MY_TASK.output.job}</code> <code>\${MY_STAGE.MY_TASK.output.jobId}</code> <code>\${MY_STAGE.MY_TASK.output.jobStatus}</code> <code>\${MY_STAGE.MY_TASK.output.jobResults}</code> # Refer to all results <code>\${MY_STAGE.MY_TASK.output.jobResults.junitResponse}</code> # Refer to JUnit results <code>\${MY_STAGE.MY_TASK.output.jobResults.jacocoResponse}</code> # Refer to JaCoCo results <code>\${MY_STAGE.MY_TASK.output.jobUrl}</code>
TFS			

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

Task type	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	<code>\${MY_STAGE.MY_TASK.input.projectCollection}</code> <code>\${MY_STAGE.MY_TASK.input.teamProject}</code> <code>\${MY_STAGE.MY_TASK.input.buildDefinitionId}</code>
	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	<code>\${MY_STAGE.MY_TASK.output.buildId}</code> <code>\${MY_STAGE.MY_TASK.output.buildUrl}</code> <code>\${MY_STAGE.MY_TASK.output.logUrl}</code> <code>\${MY_STAGE.MY_TASK.output.dropLocation}</code>
vRO			
	Input	workflowId: ID of the workflow to be run parameters: Parameters to be passed to the workflow	<code>\${MY_STAGE.MY_TASK.input.workflowId}</code> <code>\${MY_STAGE.MY_TASK.input.parameters}</code>
	Output	workflowExecutionId: ID of the workflow execution properties: Output properties from the workflow execution	<code>\${MY_STAGE.MY_TASK.output.workflowExecutionId}</code> <code>\${MY_STAGE.MY_TASK.output.properties}</code>

Table 3-15. Integrate other applications through an API

Task type	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	<code>\${MY_STAGE.MY_TASK.input.url}</code> <code>\${MY_STAGE.MY_TASK.input.action}</code> <code>\${MY_STAGE.MY_TASK.input.headers}</code> <code>\${MY_STAGE.MY_TASK.input.payload}</code> <code>\${MY_STAGE.MY_TASK.input.fingerprint}</code> <code>\${MY_STAGE.MY_TASK.input.allowAllCerts}</code>

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

Task type	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	<pre> \${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 </pre>
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 pollIntervalSeconds: Number of seconds to wait between each iteration 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	<pre> \${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} </pre>
	Output	responseCode: HTTP response code responseBody: String format of response received responseJson: Traversable response if the content-type is application/json	<pre> \${MY_STAGE.MY_TASK.output.responseCode} \${MY_STAGE.MY_TASK.output.responseBody} \${MY_STAGE.MY_TASK.output.responseJson} # Refer to response as JSON </pre>

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

Task type	Scope	Key	How to use SCOPE and KEY in the task
PowerShell			
	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	<pre> \${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} </pre>
	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	<pre> \${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} </pre>
SSH			

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

Task type	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	<pre> \${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} </pre>
	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	<pre> \${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} </pre>

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	<code>\${<Stage Key>.<Task Key>.output.<Task output key>}</code>
To refer to the previous task output value as an input for the current task	<code>\${<Previous/Current Stage key>.<Previous task key not in current Task group>.output.<task output key>}</code>

To learn more

To learn more about how to bind variables, see:

- [How do I use variable bindings in vRealize Automation Code Stream pipelines](#)
- [How do I use variable bindings in a condition task to run or stop a pipeline in vRealize Automation Code Stream](#)

- [What task types are available in vRealize Automation Code Stream](#)

How do I send notifications about my pipeline in vRealize Automation Code Stream

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

You can configure vRealize Automation Code Stream to send notifications when a pipeline runs, based on the status of the entire pipeline, stage, or task.

- An email notification sends an email on:
 - Pipeline completion, waiting, failure, cancellation, 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, cancellation, 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.

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

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 vRealize Automation 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 vRealize Automation 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 based on...	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 Enter the information to configure the Slack notification.
 - 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 Enter the information to configure the Jira notification.
 - 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

CANCEL **SAVE**

Results

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

What to do next

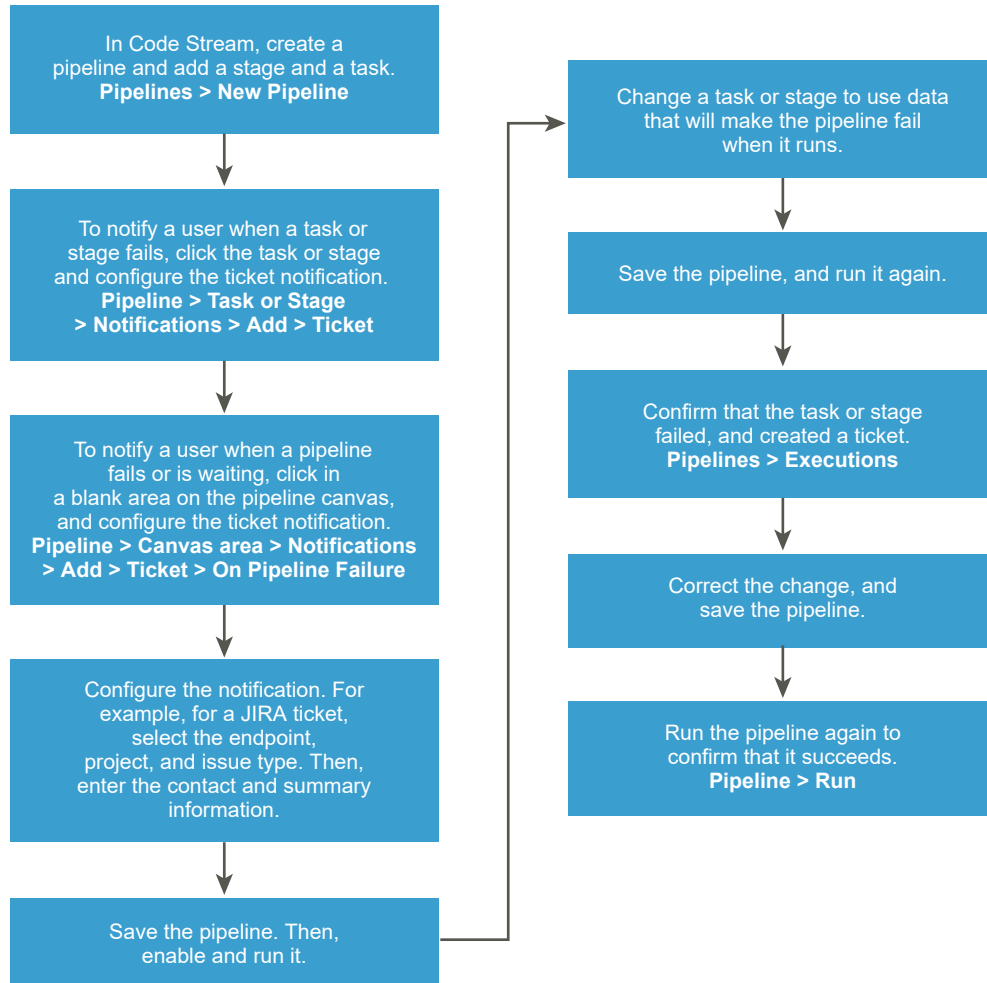
For a detailed example of how to create a notification, see [How do I create a Jira ticket in vRealize Automation Code Stream when a pipeline task fails](#).

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

If a stage or task in your pipeline fails, you can have vRealize Automation 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. vRealize Automation 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 vRealize Automation 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.



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 issue type.
 - 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 \$

- 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 vRealize Automation 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 vRealize Automation 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 vRealize Automation 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. You attach the rollback pipeline to tasks or stages that you want to roll back in the event of a failure.

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

- As a release engineer, I want vRealize Automation 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 vRealize Automation Code Stream](#).

Prerequisites

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

- Verify that you have a Kubernetes YAML file to apply to the deployment.
- Familiarize yourself with the CD smart pipeline template. See [Planning a CD native build in vRealize Automation Code Stream before using the smart pipeline template](#).

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 Dev and Prod environments.
 - 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 * [ⓧ](#)

Pipeline name *

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	<input type="text" value="bgreen-596788"/>
Production	Prod-VKE-Cluster	<input type="text" value="bgreen"/>

Image source * ☐ Docker trigger ☒ Pipeline runtime input

Deployment model * ☐ Canary ☒ Rolling upgrade ☐ Blue-Green

Rollback ☒

Health check URL *



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

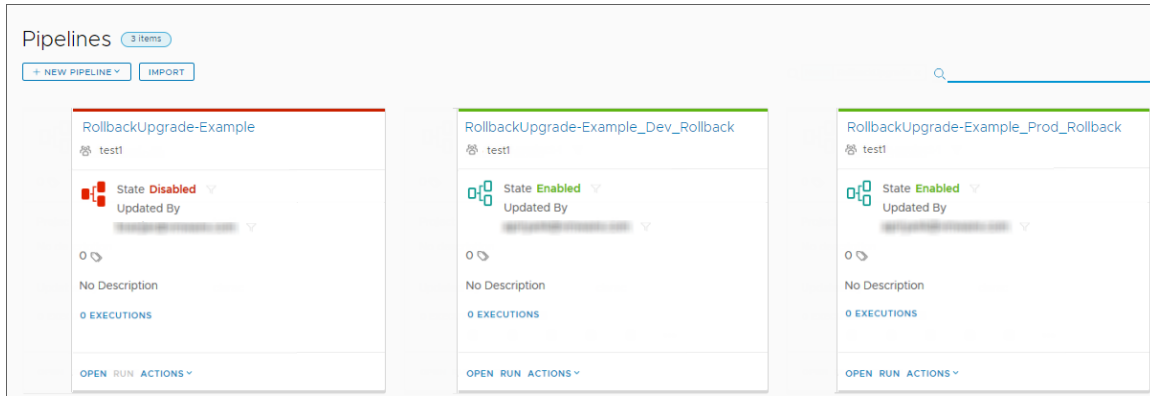
The RollbackUpgrade-Example pipeline appears with the rollback icon on tasks in the Development and Production stages that can be rolled back.

[illegible]

4 Close the pipeline.

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

- RollingUpgrade-Example. vRealize Automation Code Stream deactivates the pipeline that you created by default, which ensures that you review it before you run it.
- RollingUpgrade-Example_Dev_Rollback. This rollback development pipeline is invoked on the failure of tasks in the development stage, such as Create service, Create secret, Create deployment, Verify deployment. The rollback development pipeline is enabled by default to ensure the rollback of development tasks.
- RollingUpgrade-Example_Prod_Rollback. This rollback production pipeline is invoked on the failure of tasks in the production stage, such as Deploy phase 1, Verify phase 1, Deploy Rollout phase, Finish Rollout phase, Verify rollout phase. The rollback production pipeline is enabled by default to ensure the rollback of production tasks.

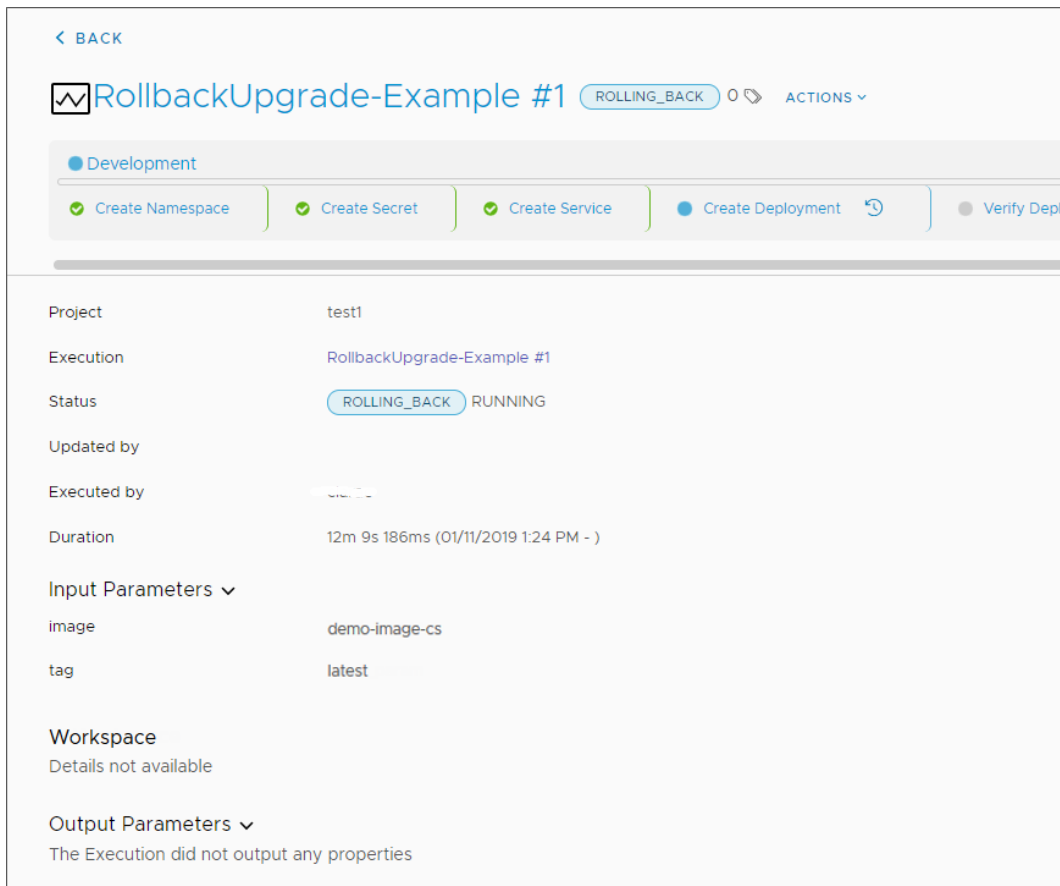


5 Enable and run the pipeline you created.

When you start the run, you are prompted 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** to 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 is triggered to roll back the deployment and the pipeline status changes to **ROLLING_BACK**.



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

- The pipeline you created that was rolled back shows **ROLLBACK_COMPLETED**.
- The rollback development pipeline that was triggered to perform the rollback shows **COMPLETED**.

The screenshot shows the 'Executions' page with a search bar and a '+ NEW EXECUTION' button. Below the header, there are two execution entries:

Icon	Name	Status	Stages	By	On	Time	Comments
	RollbackUpgrade-Example_Dev... #1	COMPLETED	Stages:	By Cloud	on 01/11/2019	1:36 PM	Execution Completed. Comments: Triggered to rollback Development. Create Deployment of RollbackUpgrade-Example#1
	RollbackUpgrade-Example#1	ROLLBACK_COMPLETED	Stages:	By Cloud	on 01/11/2019	1:24 PM	Create Deployment ROLLBACK_COMPLETED

Results

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

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

4

Before you have vRealize Automation 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.

We've provided several examples that show you how to plan for your continuous integration and delivery build. These plans describe the prerequisites you'll need to do, and overviews to help you prepare to use the native build capability effectively to build your pipelines.

This chapter includes the following topics:

- [Planning a CICD native build in vRealize Automation Code Stream before using the smart pipeline template](#)
- [Planning a CI native build in vRealize Automation Code Stream before using the smart pipeline template](#)
- [Planning a CD native build in vRealize Automation Code Stream before using the smart pipeline template](#)
- [Planning a CICD native build in vRealize Automation Code Stream before manually adding tasks](#)
- [Planning for rollback in vRealize Automation Code Stream](#)

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

To create a continuous integration and continuous delivery (CICD) pipeline in vRealize Automation Code Stream, you can use the CICD smart pipeline template. To plan your CICD native build, you'll gather the information you need to fill out the smart pipeline template before you use it to create the pipeline in this example plan.

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 where to deploy your image based on the environment types you select, such as Dev and Prod. The pipeline will publish your Docker image, and perform the actions required to run it. After your pipeline runs, you can monitor trends across the pipeline executions.

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

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

Planning the Continuous Integration (CI) stage

To plan the CI stage of your pipeline, you'll set up the external and internal requirements, and determine the information to enter in the CI portion of the smart pipeline template. Here is a summary.

Endpoints and repositories that you'll need:

- A Git source code repository where your developers check in code. vRealize Automation 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 vRealize Automation 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 vRealize Automation Code Stream. If you are not, ask a vRealize Automation Code Stream administrator to add you as a member of a project. See [How do I add a project in vRealize Automation Code Stream](#).

You'll need a Git webhook that enables vRealize Automation 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 vRealize Automation 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 first 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'll 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 vRealize Automation Code Stream can deploy your image to a Kubernetes cluster.

Environment types and files:

- All the environment types where vRealize Automation 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 4-1. 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.

To apply the file, you click **Select**, select the Kubernetes YAML file, and 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 ☐

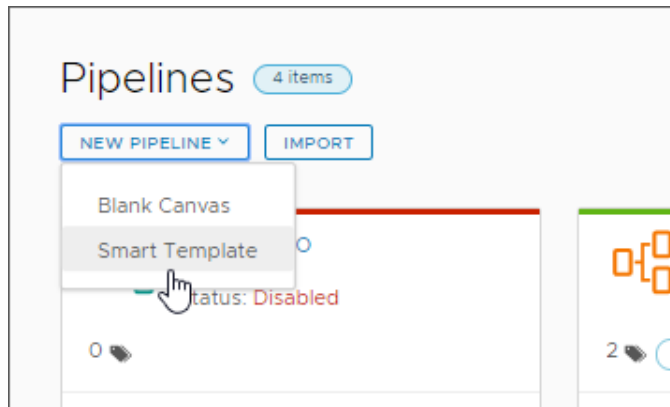
Health check URL *

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 vRealize Automation Code Stream to my Blue-Green deployment.](#)

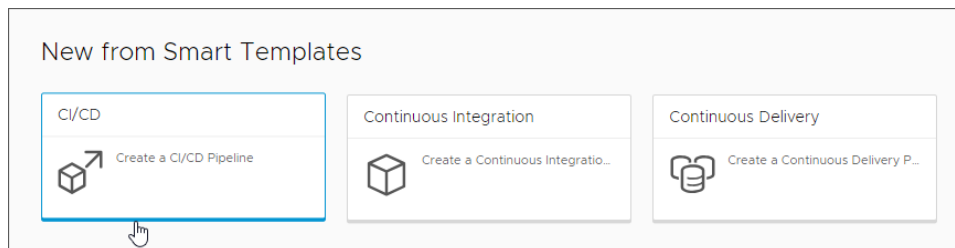
How you'll create the CICD pipeline by using the smart pipeline template

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.

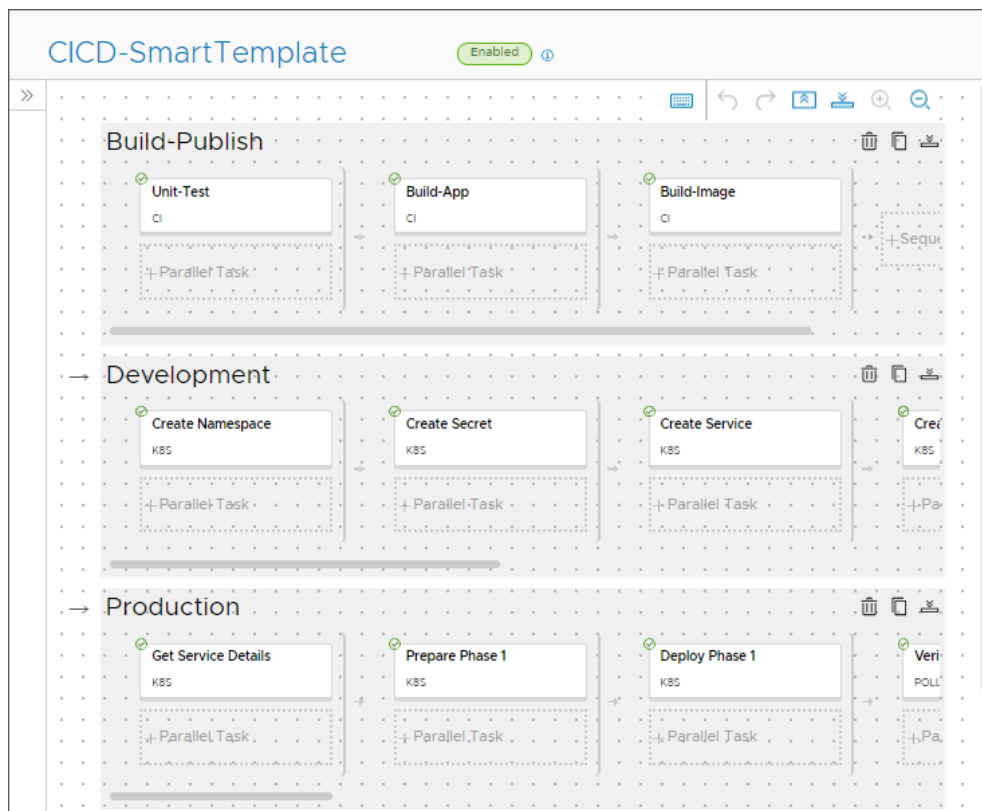
In Pipelines, you'll select **New Pipeline > Smart Templates**.



You'll select the CI/CD 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 vRealize Automation Code Stream](#).

Planning a CI native build in vRealize Automation Code Stream before using the smart pipeline template

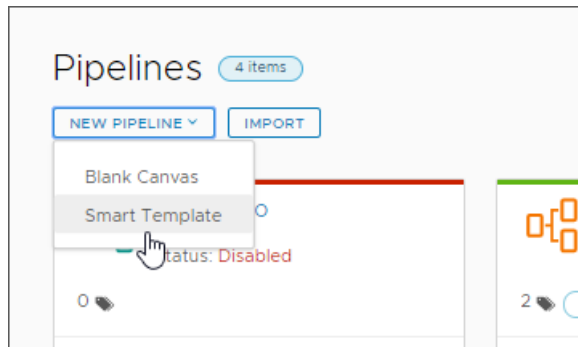
To create a continuous integration (CI) pipeline in VMware Code Stream, you can use the CI smart pipeline template. To plan your CI native build, you'll gather the information you need to fill out the smart pipeline template before you use it to create the pipeline in this example plan.

When you fill out the smart pipeline template, it creates a CI pipeline in your repository, and performs the actions required to run it. After your pipeline runs, you can monitor trends across the pipeline executions.

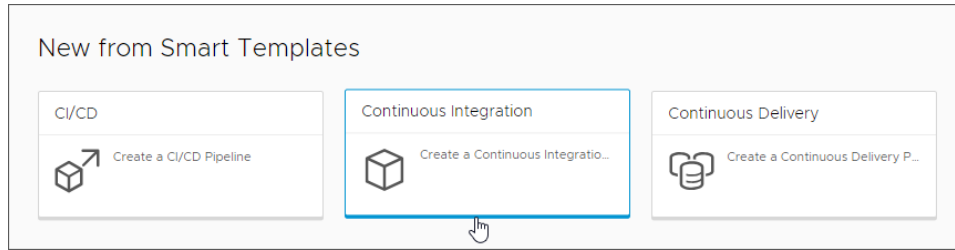
To plan your build before you use the CI smart pipeline template, you'll gather the information for your build, then follow the CI portion of [Planning a CICD native build in vRealize Automation Code Stream before using the smart pipeline template](#).

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

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



You'll select the CI smart pipeline template.



You will fill out the template, and click **Create** to save the pipeline with the stages that it creates.

You can edit the pipeline to make any final changes that you might need. Then, you can enable the pipeline and run it. After the pipeline runs, here are some things to 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 vRealize Automation Code Stream](#).

Planning a CD native build in vRealize Automation Code Stream before using the smart pipeline template

To create a continuous delivery (CD) pipeline in vRealize Automation Code Stream, you can use the CD smart pipeline template. To plan your CD native build, you'll gather the information you need to fill out the smart pipeline template before you use it to create the pipeline in this example plan.

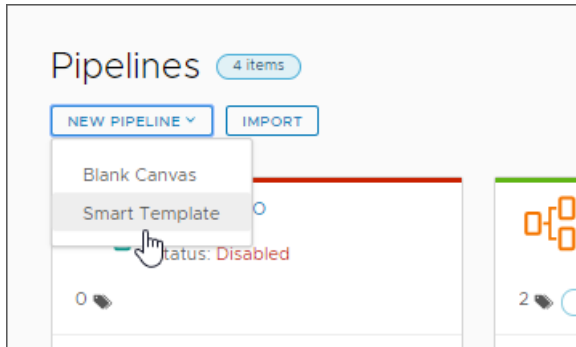
When you fill out the smart pipeline template, it creates a CD pipeline in your repository, and performs the actions required to run it. After your pipeline runs, you can monitor trends across the pipeline executions.

To plan your build before you use the CD smart pipeline template, you will:

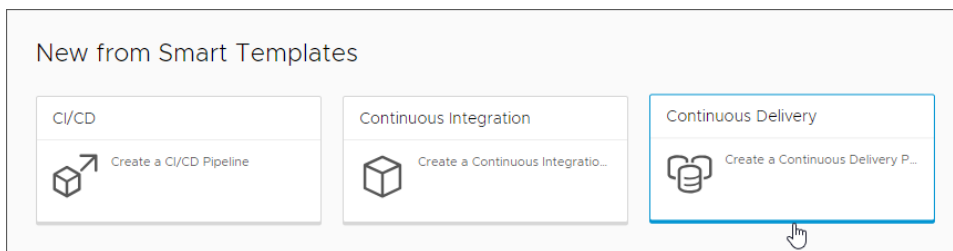
- Gather the information for your build, then follow the CD portion of [Planning a CI/CD native build in vRealize Automation Code Stream before using the smart pipeline template](#).
- Add a Kubernetes endpoint where vRealize Automation Code Stream will deploy the container.
- Identify a project that will group all your work, including your pipeline, endpoints, and dashboards.

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

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



You'll select the CD smart pipeline template.



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

You can edit the pipeline to make any final changes that you might need. Then, you can enable the pipeline and run it. After the pipeline runs, here are some things to 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 vRealize Automation Code Stream](#).

Planning a CI/CD native build in vRealize Automation Code Stream before manually adding tasks

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

You will need to 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.

To plan the CI and CD stages of your pipeline, you'll verify that all the requirements are met before you create your pipeline.

Planning the external and internal requirements

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 your developers check in code. vRealize Automation 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 Kubernetes endpoint so that vRealize Automation 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.

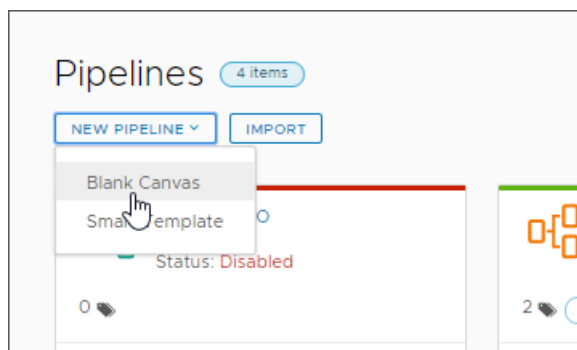
You'll need access to a project. The project groups all your work, including your pipeline, endpoints, and dashboards. Verify that you are a member of a project in vRealize Automation Code Stream. If you are not, ask a vRealize Automation Code Stream administrator to add you as a member of a project.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

You'll need a Git webhook that enables vRealize Automation 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 vRealize Automation 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.

The screenshot shows the 'Workspace' tab in the vRealize Automation interface. The tab is highlighted with a blue border. Below the tab, there are several configuration fields:

- Build host ***: A dropdown menu with the text '--Select Host Endpoint--' and a downward arrow.
- Builder image URL ⓘ ***: A text input field.
- Image registry ⓘ**: A dropdown menu with the text '--Select Container Registry Endpoint--' and a downward arrow.
- Working directory ⓘ**: A text input field.
- Git clone**: A text input field with a tooltip that reads: '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'.
- Cache ⓘ**: A text input field with a blue plus icon to its right.

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.

Pipeline
Input
Output
Notifications
Workspace ⓘ

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_DESCRIPTOR...		
⋮	☆	GIT_EVENT_OWNER...		
⋮	☆	GIT_EVENT_TIMESTAMP		
⋮	☆	GIT_REPO_NAME		
⋮	☆	GIT_SERVER_URL		

Configure the pipeline to test your code:

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

Task : *Unit-Test*
Notifications
VALIDATE TASK

Task name ⓘ *
Unit-Test

Type *
CI

Continue On Failure
☐

Execute Task
☒ Always ☐ On Condition

Continuous Integration

Steps *

```

1 cd demo-project
2 mvn test

```

Preserve Artifacts ⓘ
+

Export

Process
ADD

JUnit
JUnit /demo-project +

JaCoCo
Jacoco /demo-project +

Checkstyle
Checkstyle /demo-project +

FindBugs
Findbugs /demo-project +

Output Parameters
status exports

Configure the pipeline to build your code:

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

Task : *Build-App*
Notifications
VALIDATE TASK

Task name ⓘ *
Build-App

Type *
CI

Continue On Failure
☐

Execute Task
☒ Always ☐ On Condition

Continuous Integration

Steps *

```

1 cd demo-project
2 mvn clean install -DskipTests

```

Preserve Artifacts ⓘ
+

Export

Process
ADD

JUnit
JUnit /demo-project +

JaCoCo
Jacoco /demo-project +

Checkstyle
Checkstyle /demo-project +

FindBugs
Findbugs /demo-project +

Output Parameters
status exports

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.

The screenshot shows the configuration for a task named 'Build-Image'. The 'Type' is set to 'CI'. The 'Execute Task' option is set to 'Always'. The 'Continuous Integration' section contains a code block with the following steps:

```

1 cd demo-pro
2 export IMAGE=automation/demo-cicd-smart-template:${executionIndex}
3 export DOCKER_HOST=tcp://10.10.10.10:2376
4 docker login --username=auto --password=VM
5 docker build -t $IMAGE --file ./docker/Dockerfile .
6 docker push $IMAGE

```

The 'Export' field is set to 'IMAGE'. The 'Process' field has an 'ADD' button. The 'Output Parameters' section has buttons for 'status' and 'exports'.

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 vRealize Automation Code Stream continuously integrates your code, see [How do I continuously integrate code from my GitHub or GitLab repository into my pipeline in vRealize Automation Code Stream](#).

Planning for rollback in vRealize Automation Code Stream

In the event of a pipeline execution failure, you can use rollback to return your environment to a previously stable state. To use rollback, you should 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 vRealize Automation 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 vRealize Automation 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, vRealize Automation 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 vRealize Automation 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 4-2. 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.

RollbackUpgrade-Example Enabled

Workspace Input **Model** Output

Task: *UPD Deploy US* Notifications **Rollback**

Pipeline *RB_Deploy_US*

Production

- UPD Deploy US (Kubernetes)
- UPD Deploy UK (Kubernetes)
- UPD Deploy AU (Kubernetes)
- +Parallel Task
- +Stage

SAVE **RUN** **CLOSE** Last saved 12 days ago

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 vRealize Automation Code Stream

5

Use vRealize Automation Code Stream to model and support your DevOps release lifecycle and continuously test and release your applications.

You already set up everything you need so that you can use vRealize Automation Code Stream. See [Chapter 2 Setting up vRealize Automation 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 vRealize Automation Code Stream deploy container-based or traditional applications.

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

Using features...	Examples of what you can do...
Use the native build capability in vRealize Automation Code Stream.	Create CI/CD, CI, and CD pipelines that continuously integrate, containerize, and deliver your code. <ul style="list-style-type: none">■ Use a smart pipeline template to create 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. <ul style="list-style-type: none">■ Continuously integrate your code in GitHub or GitLab into your pipeline.■ 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 vRealize Automation Code Stream with your own build, test, and deploy tools.■ Use a REST API that integrates vRealize Automation Code Stream with other applications.
Track trends, metrics, and KPIs.	Create custom dashboards and gain insight about the performance of your pipelines.
Resolve problems.	When a pipeline execution fails, have vRealize Automation Code Stream create a JIRA ticket.

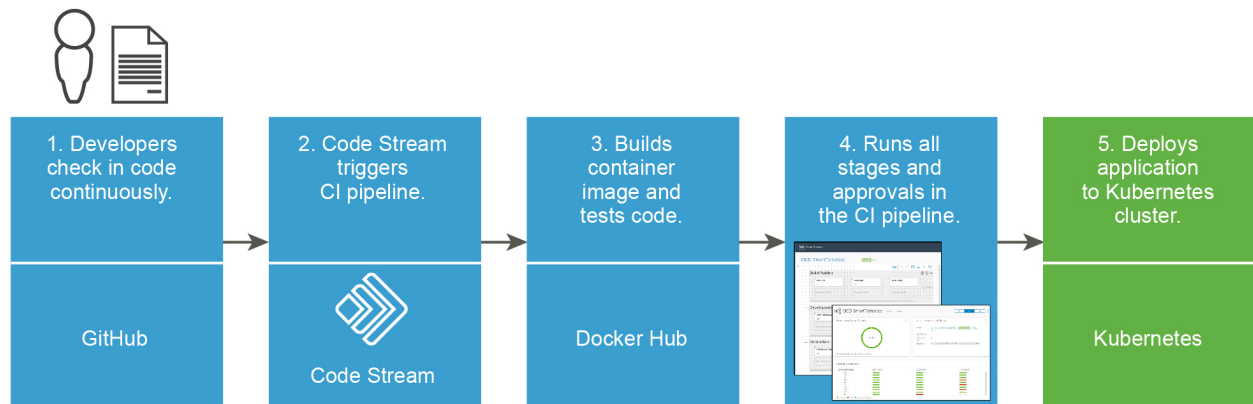
This chapter includes the following topics:

- [How do I continuously integrate code from my GitHub or GitLab repository into my pipeline in vRealize Automation Code Stream](#)
- [How do I automate the release of an application that I deploy from a YAML cloud template in vRealize Automation Code Stream](#)

- How do I automate the release of an application in vRealize Automation Code Stream to a Kubernetes cluster
- How do I deploy my application in vRealize Automation Code Stream to my Blue-Green deployment
- How do I integrate my own build, test, and deploy tools with vRealize Automation Code Stream
- How do I use a REST API to integrate vRealize Automation Code Stream with other applications

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

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



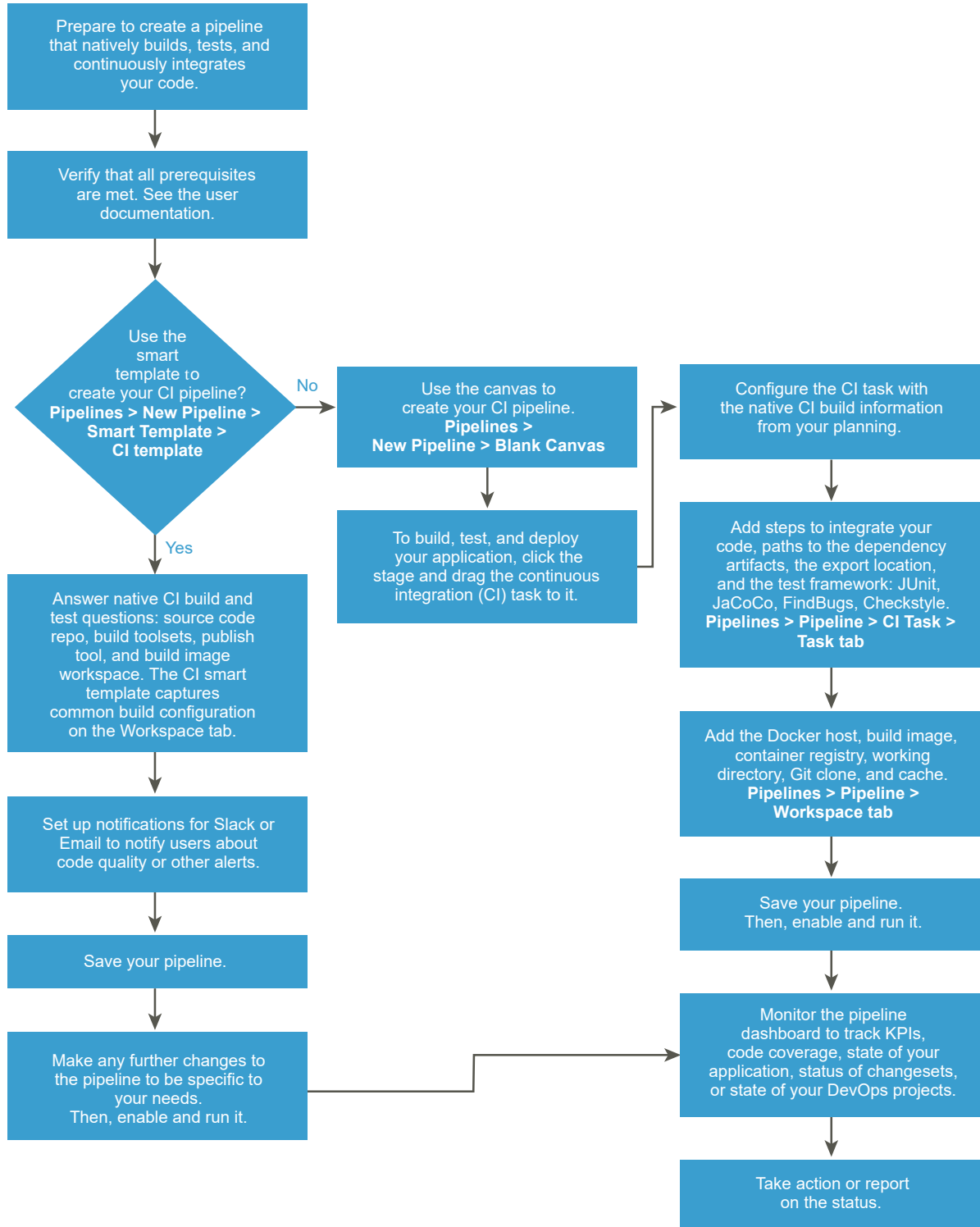
To have vRealize Automation Code Stream trigger your pipeline on code changes, you will use the Git trigger. vRealize Automation Code Stream will trigger your pipeline every time you commit changes to your code.

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

Then you will use the continuous integration (CI) smart pipeline template to create a CI pipeline that builds, tests, and deploys your code to your project team's Kubernetes cluster on AWS. You will use a cache to store the code dependency artifacts for your CI task, which will save time in code builds.

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

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



In this example, you'll use the continuous integration (CI) smart pipeline template to create a CI pipeline so that you can continuously integrate your code into your pipeline.

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 vRealize Automation Code Stream before manually adding tasks](#).

Prerequisites

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

Procedure

- 1 Follow the prerequisites.
- 2 To create the pipeline by using the smart pipeline template, open the CI 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 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 have your native CI build information ready to configure the continuous integration (CI) task.
 - a Click **Pipelines > New Pipeline > Blank Canvas**.
 - b Click the stage, then drag the several CI tasks from the navigation pane to the stage.
 - c To configure the CI 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 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.

The screenshot displays the vRealize Automation Code Stream interface. On the left, a sidebar contains navigation links: Dashboards, Executions, User Operations, Pipelines, Manage (with sub-links for Endpoints, Variables, and Triggers), and Triggers (with sub-links for Gerrit and Git). The main area shows the execution details for 'CICD-SmartTemplate #51', which is marked as 'COMPLETED'. A progress bar at the top indicates the status of various tasks: Build-Publish (Unit-Test, Build-App, Build-Image) and Development (Create Namespace, Create Secret, Create Service, Create Deployment). The 'Build-Image' task is highlighted. Below the progress bar, the task details for 'Build-Image' are shown, including its type (CI), status (COMPLETED), duration (5s), and execution time (09/11/2018 7:16 AM - 09/11/2018 7:16 AM). The 'Result' section shows the output of the task, which includes a successful login and build process. The 'Exports' section shows the output of the task, which is the 'IMAGE' value 'automation/cicd-smart-template:51'.

- 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 or GitLab repository into your pipeline, and deploys your build image.

What to do next

To learn more, see [More resources for vRealize Automation Code Stream Administrators and Developers](#).

How do I automate the release of an application that I deploy from a YAML cloud template in vRealize Automation 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. vRealize Automation 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 vRealize Automation 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.

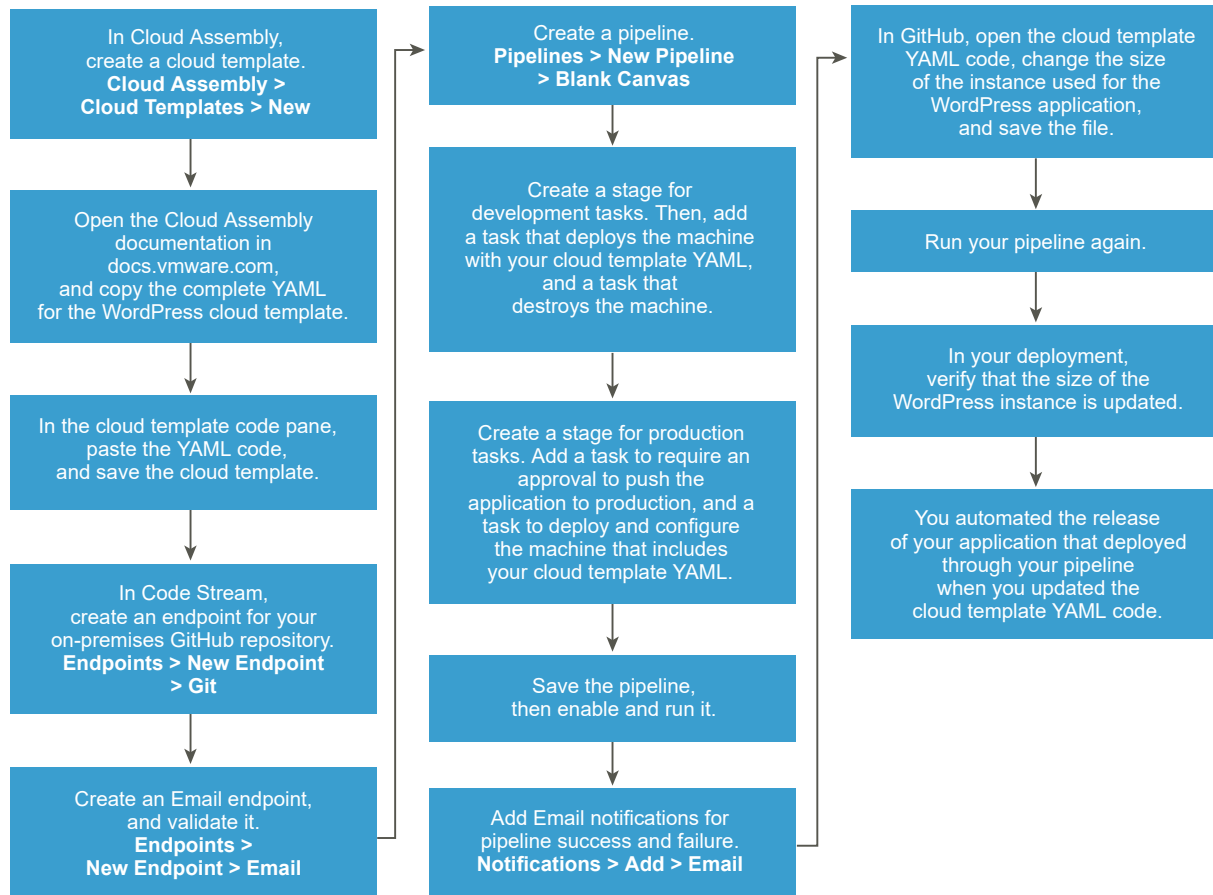
If you have a YAML cloud template in your GitHub repository, and want to use that cloud template in your pipeline, here's what you'll need to do.

- 1 In vRealize Automation Cloud Assembly, push the cloud template to your GitHub repository.
- 2 In vRealize Automation 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 vRealize Automation Cloud Assembly, you'll add a cloud account, add a cloud zone, and create the cloud template.
- In vRealize Automation 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.

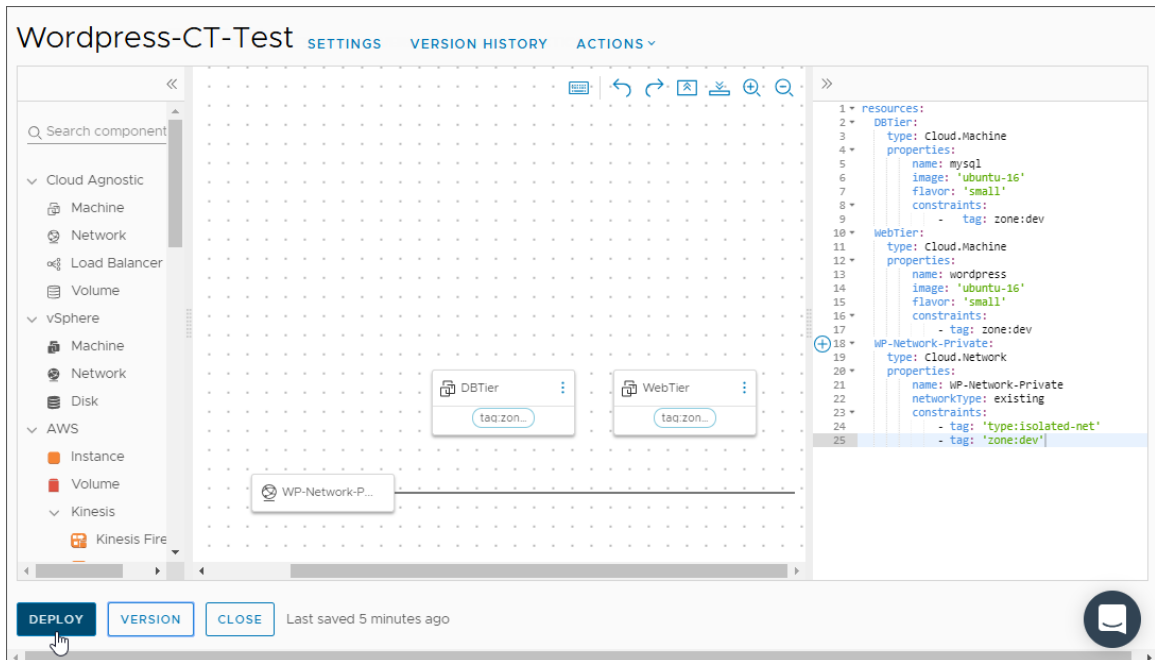


Prerequisites

- 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 vRealize Automation Code Stream, click **Triggers > Git > Webhooks for Git**.
- To work with a cloud template task, you must have any of the vRealize Automation Cloud Assembly roles.

Procedure

- 1 In vRealize Automation 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.



- 2 In vRealize Automation 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 to notify users about the pipeline status when it runs.

Add Endpoint

Project *	Codestream ▾
Type *	Email ▾
Name *	Enter value here
Description	
Mark as restricted	<input type="checkbox"/> 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

CREATEVALIDATECANCEL

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

Notification

Send notification type

When pipeline

Email server ⓘ *

Send Email

To ⓘ \$ *

Subject \$ *

Body ⓘ \$ *

☒ Email ☐ Ticket ☐ Webhook

☒ Completes ☐ Is Waiting ☐ Fails ☐ Is cancelled ☐ Starts to run

--Select Email server-- ▾

Email IDs of recipients

Email Subject

1	
---	--

CANCEL

SAVE

- 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, vRealize Automation Code Stream will assign it a unique random name.

Here's what you need to 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 vRealize Automation Code Stream, rollback will not be allowed.

When you log in to vRealize Automation 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 vRealize Automation 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.

6 Run the pipeline.

To verify that each task completed successfully, click the task in the execution, and check 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 vRealize Automation Code Stream, see [Chapter 5 Tutorials for using vRealize Automation Code Stream](#).

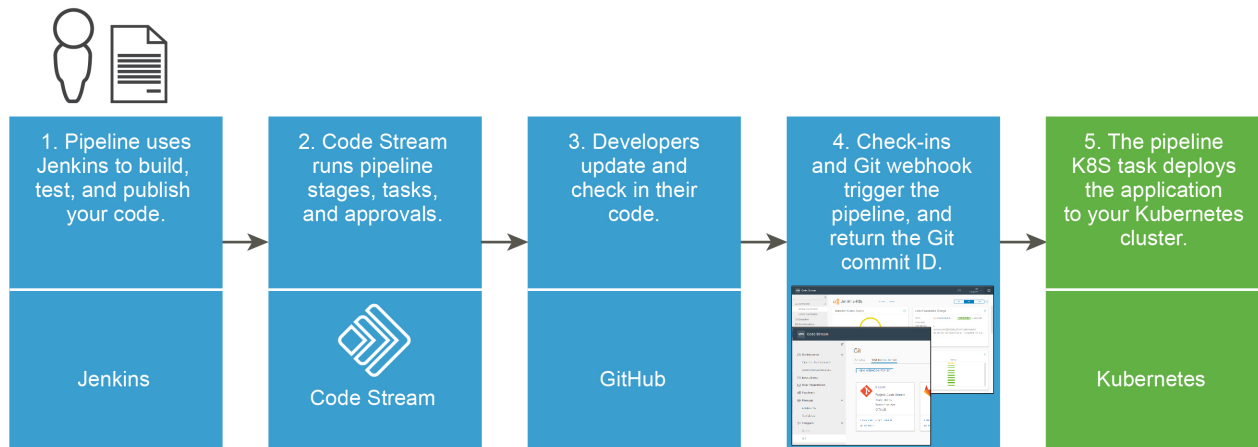
For additional references, see [More resources for vRealize Automation Code Stream Administrators and Developers](#).

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

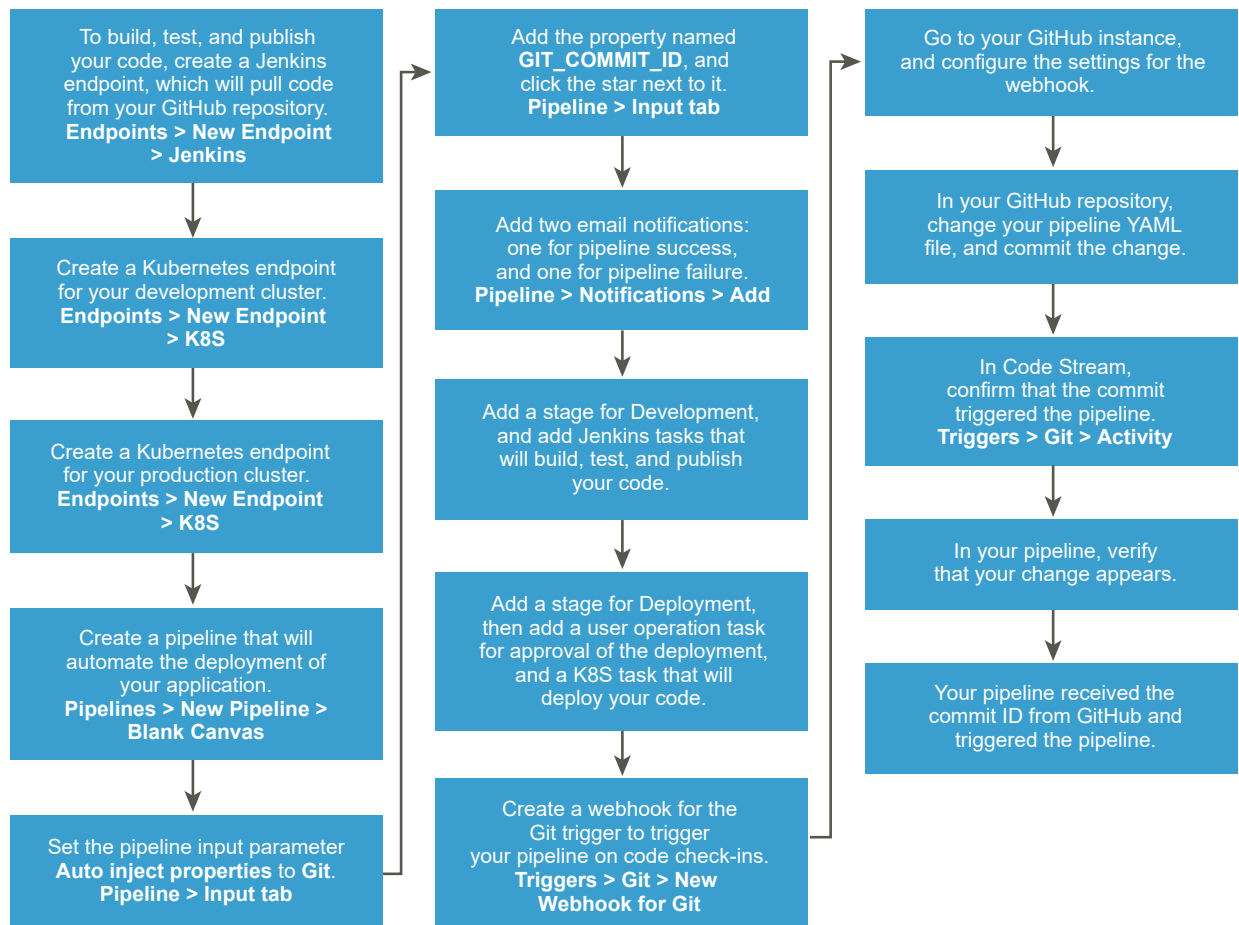
As a vRealize Automation Code Stream administrator or developer, you can use vRealize Automation Code Stream and VMware Tanzu Kubernetes Grid Integrated Edition (formerly known as VMware Enterprise PKS) to automate the deployment of your software applications to a Kubernetes cluster. This use case mentions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



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.



Other methods that automate the release of your application:

- Instead of using Jenkins to build your application, you can use the vRealize Automation Code Stream native build capability and a Docker build host.

- Instead of deploying your application to a Kubernetes cluster, you might deploy it to an Amazon Web Services (AWS) cluster.

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

- [Planning a CICD native build in vRealize Automation Code Stream before using the smart pipeline template](#)
- [Planning a CICD native build in vRealize Automation Code Stream before manually adding tasks](#)

Prerequisites

- Verify that the application code to be deployed resides in a working GitHub repository.
- Verify that you have a working instance of Jenkins.
- Verify that you have a working email server.
- In vRealize Automation 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 vRealize Automation 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.

- 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 to it.

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

The screenshot shows the Jenkins-K8s pipeline editor. The pipeline is titled "Jenkins-K8s" and is in the "Enabled" state. The pipeline is divided into two stages: "Dev" and "Deploy".

The "Dev" stage contains three tasks: "Build-DemoApp" (Jenkins), "Test-DemoApp" (Jenkins), and "Publish-DemoApp" (Jenkins). Each task is followed by a "Parallel Task" block.

The "Deploy" stage contains two tasks: "Approve-Deployment" (UserOperation) and "tpm-K8s-AWS" (K8S). Each task is followed by a "Parallel Task" block. A "Sequential Task" block is also present in the Deploy stage.

The "Pipeline Input Parameters" panel is open on the right side of the editor. It shows the "Auto inject properties" section with the "Git" radio button selected. Below this, there is a table of input parameters:

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		

At the bottom of the panel, it says "8 input parameters".

At the bottom of the editor, there are buttons for "SAVE", "RUN", and "CLOSE". A status bar indicates "Last saved 5 days ago".

- 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

CANCEL

SAVE

- 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.

The screenshot displays the Jenkins-K8s pipeline editor. The pipeline is titled "Jenkins-K8s" and is in an "Enabled" state. It features a "Dev" stage containing three tasks: "Build-DemoApp", "Test-DemoApp", and "Publish-DemoApp", each using the "Jenkins" provider. Below the Dev stage is a "Deploy" stage. The right-hand panel shows the configuration for the "Build-DemoApp" task. The task name is "Build-DemoApp", the type is "Jenkins", and it is configured to "Always" execute. The Jenkins Job endpoint is "Jenkins", the job is "Build-DemoApp", and the branchName is set to the input parameter `${input.GIT_BRANCH_NAME}`. At the bottom, there are buttons for "status", "job", "jobid", "jobResults", and "joburi". The bottom of the editor shows "SAVE", "RUN", and "CLOSE" buttons, along with a "Last saved 5 days ago" timestamp.

- 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 vRealize Automation Code Stream to use the Git trigger, which triggers your pipeline when developers commit their code.

The screenshot shows the 'Git' configuration page in vRealize Automation. The 'Webhooks for Git' tab is active. The form includes the following fields and options:

- Webhook URL:** `https://[redacted].vmware.com/pipeline/api/git-webhook-listeners/d4c4b02804780`
- Project:** Code Stream
- Name:** muser-Demo-WH
- Description:** (empty text box)
- Endpoint:** tpm-GitHub
- Branch:** master
- Secret token:** (masked with dots) and a **GENERATE** button.
- File:** (empty text box)
- Inclusions:** --Select-- (dropdown) and Value (text box) with a **+** button.
- Exclusions:** --Select-- (dropdown) and Value (text box) with a **+** button.
- Prioritize Exclusion:** (toggle switch, currently off)
- Trigger:**
 - For Git:** ☒ PUSH ☐ PULL REQUEST
- API token:** (masked with dots) and **CREATE VARIABLE** and **GENERATE TOKEN** buttons.
- Pipeline:** Jenkins-K8s (dropdown menu)
- Comments:** (empty text box)

- 8 To test your pipeline, go to your GitHub repository, update your application YAML file, and commit the change.
 - a In vRealize Automation 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:
eyJzeWlwag9ueS10YW5nbyliZXRhMi5qZnJvZy5pbyI6eyJlc2VybmFtZSI6InRhbmdvLWJldGEyIiwicGFzc3dvcmQiOi
JhRGstcmVOLWlUQi1IejciLCJlbWVpbCI6InRhbmdvLWJldGEyQHZtd2FyZS5jb20iLCJhdXRoIjoizEdGdVoyOHRZbVYw
WVRJN1lVUnJMWEPsVGkxdFZFSXRTSG8zIn19
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 vRealize Automation Code Stream with Jenkins, see [How do I integrate vRealize Automation Code Stream with Jenkins](#).

How do I deploy my application in vRealize Automation 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 vRealize Automation 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 vRealize Automation Code Stream represents your Blue-Green deployment model in two stages. One stage is for development, and the other stage is for production.

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.

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

Task type	Task
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 vRealize Automation 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 vRealize Automation 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 vRealize Automation 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 vRealize Automation 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 vRealize Automation 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 and Prod environments.
 - 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 *

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 *



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. 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 vRealize Automation Code Stream, see [Chapter 5 Tutorials for using vRealize Automation Code Stream](#).

To roll back a deployment, see [How do I roll back my deployment in vRealize Automation Code Stream](#).

For additional references, see [More resources for vRealize Automation Code Stream Administrators and Developers](#).

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

As a DevOps administrator or developer, you can create custom scripts that extend the capability of vRealize Automation Code Stream. With your script, you can integrate vRealize Automation 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 to integrate with your build, test, and deploy tools. For example, it can work with the workspace in your pipeline to support CI tasks that build and test your application, and CD tasks that deploy your application. It can send a message to Slack when a pipeline finishes, and much more.

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 vRealize Automation 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.	<p>The custom task:</p> <ul style="list-style-type: none"> ■ 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 vRealize Automation Code Stream.
After your pipeline runs, look at the executions.	Verify that the pipeline delivered the results you expected.

This example creates a custom integration that connects vRealize Automation 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 these shell languages: Bash, sh, or zsh.
- Generate a container image by using the Node.js or Python runtime that is installed.

Procedure

1 Create the custom integration.

- Click **Custom Integrations > New**, and enter a relevant name.
- Select the preferred runtime environment.
- 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 vRealize Automation Code Stream runs the code, which can be one of these case-insensitive strings: <ul style="list-style-type: none"> ■ 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: <ul style="list-style-type: none"> ■ 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

Custom task YAML input keys	Description	Required
labelMessage	String that acts as a help tooltip for users. Adds a tooltip icon i next to the input title.	No
enum	<p>Takes in an array of values that displays the select input property options. Only supported only for certain input property types.</p> <p>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.</p> <p>For example, the value 2015.</p> <ul style="list-style-type: none"> ■ 2015 ■ 2016 ■ 2017 ■ 2018 ■ 2019 ■ 2020 	No
options	<p>Takes in an array of objects by using optionKey and optionValue.</p> <ul style="list-style-type: none"> ■ optionKey. Value propagated to the code section of the task. ■ optionValue. String that displays the option in the user interface. <p>Only supported only for certain input property types.</p> <p>Options:</p> <p>optionKey: key1. When selected and saved for the custom task, the value of this inputProperty would correspond to key1 in the code section.</p> <p>optionValue: 'Label for 1'. Display value for key1 in the user interface, and does not appear anywhere else for the custom task.</p> <p>optionKey: key2</p> <p>optionValue: 'Label for 2'</p> <p>optionKey: key3</p> <p>optionValue: 'Label for 3'</p>	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 5-5. Supported data types and meta data for custom scripts

Supported data types	Supported meta data for input
<ul style="list-style-type: none"> 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 	<ul style="list-style-type: none"> 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 vRealize Automation 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
  });

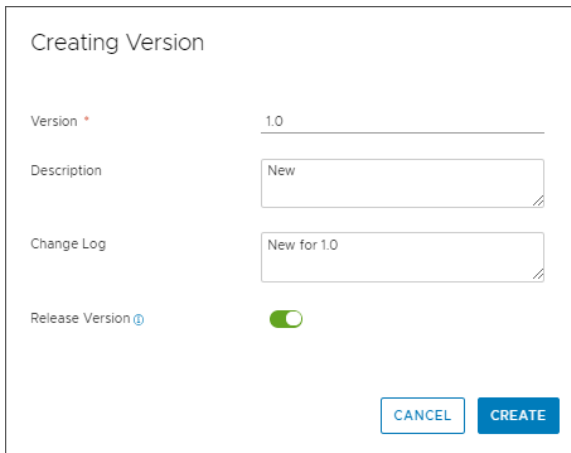
const options = {
  hostname: 'hooks.slack.com',
  port: 443,
  path: '/YOUR_SLACK_WEBHOOK_PATH',
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'Content-Length': Buffer.byteLength(slackPayload)
  }
};

// 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 cursor 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**.



The image shows a 'Creating Version' dialog box with the following fields and controls:

- Version ***: A text input field containing '1.0'.
- Description**: A text input field containing 'New'.
- Change Log**: A text input field containing 'New for 1.0'.
- Release Version**: A toggle switch that is currently turned on (green).
- At the bottom right, there are two buttons: **CANCEL** and **CREATE**.

- 8 To save the script, click **Save**.

- 9 In your pipeline, configure the workspace.
 - a Click the **Workspace** tab.
 - b Select the Docker host and the builder image URL.

The screenshot shows the 'Workspace' tab for a pipeline named 'Demo-customTask-nodejs' (status: Enabled). The configuration fields are as follows:

- Host:** Docker-saas (dropdown menu)
- Builder image URL:** node:latest
- Image registry:** --Select Container Registry Endpoint-- (dropdown menu)
- Working directory:** (empty text field)
- Cache:** (empty text field with a blue plus icon to the right)
- Git clone:** A text box containing the instruction: "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:

The screenshot shows the 'Model' tab for a pipeline named 'CustomTask-IX' (status: Enabled). The interface is divided into a visual editor on the left and a configuration panel on the right.

Visual Editor: Shows a stage named 'Stage0' containing a task named 'Task0' of type 'Custom'. Below the task is a '+ Parallel Task' button. At the bottom, there is a '+ Stage' button.

Configuration Panel (Task: Task0):

- Task name:** Task0
- Type:** Custom (dropdown menu)
- Continue on failure:** ☐
- Execute task:** ☒ Always ☐ On condition
- Custom Task:**
 - Task:** sample1 (dropdown menu)
 - Version:** version-shell (dropdown menu)
- Shell Message:** Hello Slack
- Output Parameters:** responseCode, properties, log, status (buttons)

At the top right of the configuration panel are tabs for 'Task: Task0', 'Notifications', and 'Rollback', along with a 'VALIDATE TASK' button.

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:

The screenshot displays the vRealize Automation interface for a pipeline execution. At the top, a navigation bar shows a back arrow and the text '< BACK'. Below this, the pipeline name 'custom-int-demo #5' is displayed next to a green 'COMPLETED' status badge and a '0' icon. An 'ACTIONS' dropdown menu is visible. Underneath, a progress bar shows 'Stage0' with a green checkmark, and 'Task0' and 'Task1' are listed with green checkmarks. The main section shows the details for 'Task1', including a 'VIEW OUTPUT JSON' link. The task is of type 'Custom' and has a status of 'COMPLETED' with the message 'Execution Completed.'. The duration is '6s (12/21/2018 3:04 AM - 12/21/2018 3:04 AM)'. The 'Continue on failure' checkbox is unchecked. The 'Execute task' option is set to 'Always' (selected) with a radio button, and 'On condition' is unselected. The 'Output' section shows 'statusCode' as 200 and 'Response code' as 0. The 'Logs' section shows a snippet of a command: '1 + node -r ./context.js app.js', '2', and '3 |'. A 'View Full Log' link is at the bottom.

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 vRealize Automation 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 vRealize Automation Code Stream in the automation of your software release lifecycle.

How do I use a REST API to integrate vRealize Automation Code Stream with other applications

vRealize Automation Code Stream provides a REST plug-in, which allows you to integrate vRealize Automation 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 vRealize Automation Code Stream and another application.

With the REST plug-in, you can:

- Integrate external REST API-based systems into a vRealize Automation Code Stream pipeline.
- Integrate a vRealize Automation 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 vRealize Automation Code Stream and other applications.

Table 5-6. Preparing a pipeline to communicate over the REST API

What you do...	What happens...
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:

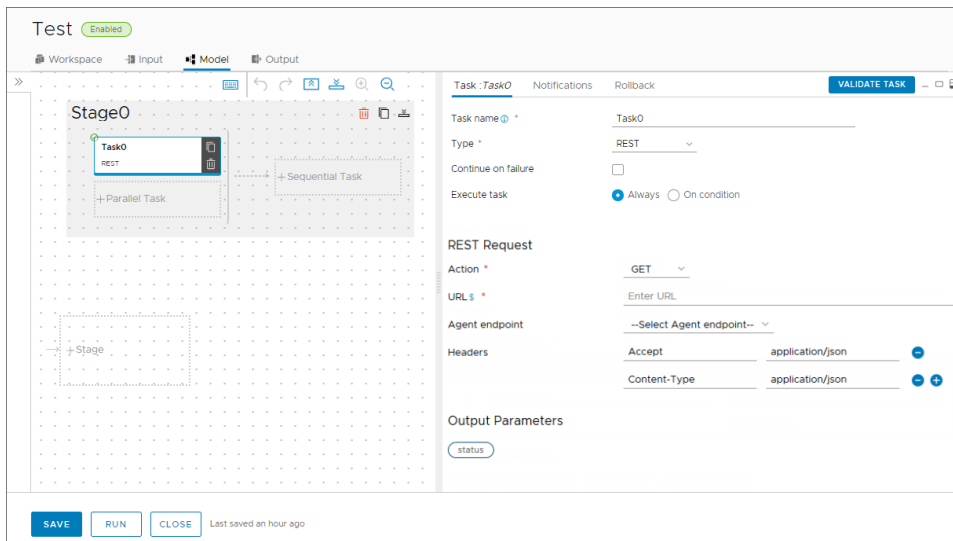
- a Enter a name for the task.
- b In the Type drop-down menu, select **REST**.
- c In the REST Request area, select **GET**.

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`.

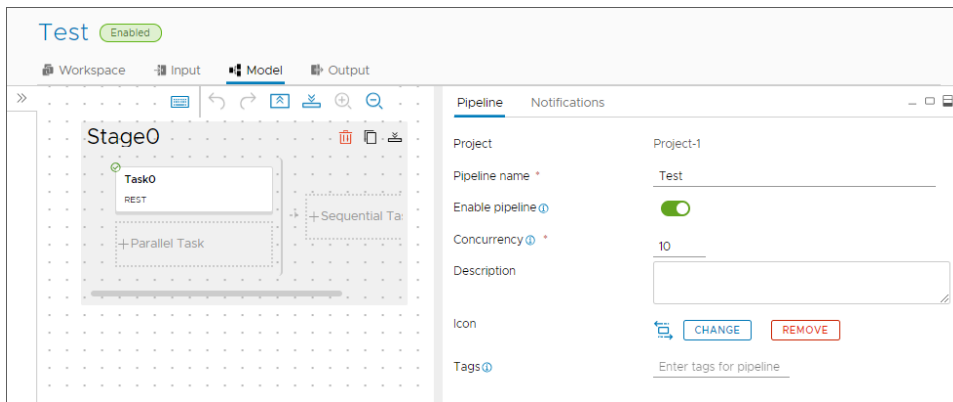
For a REST task to import data from another application, you can include the payload variable. For example, for an import action, you can enter `{Stage0.export.responseBody}`. If the response data size exceeds 5 MB, the REST task might fail.

e To provide authorization for the task, click **Add Headers** and enter a header key and value.

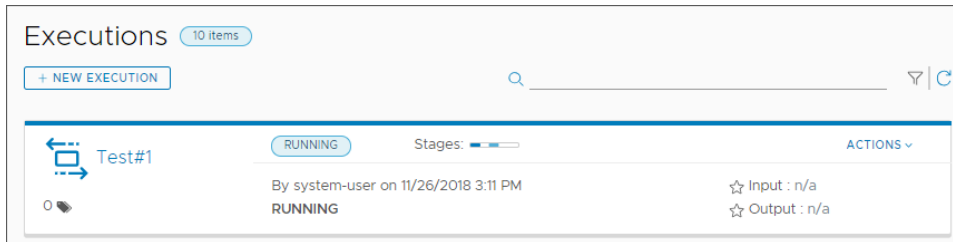


4 To save your pipeline, click **Save**.

- 5 On the pipeline tab, click **Enable pipeline**.



- 6 Click **Save**, then click **Close**.
- 7 Click **Run**.
- 8 To watch the pipeline run, click **Executions**.



- 9 To verify that the REST plug-in 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, look at the task details, and verify that the REST plug-in returned the expected results.

The task details display the response code, body, header keys, and values.

[Back](#)

Test #2

COMPLETED
ACTIONS

Stage0

Task0

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

200

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/google/gm/gm_standard_color_128dp.png" itemprop="image"><title>Google</title><script nonce="aMmwYdugKGr9CHU6QQGz"=>(function(){window.google=({E1:"cfn86KpIeVwXx-aLoDA",kEXPI:"0,1353747,57,50,1150,454,303,1017,1120,286,698,527,730,142,184,293,132,278,420,350,30,524,27,275,401,457,110,114,56,164,2336158,235,32,45,23,6,1,329219,1294,12383,4855,19577,13114,8163,7085,867,605,6,636,2239,3232,5281,1100,3335,2,2,4605,2196,369,1212,2102,4133,1372,224,887,1331,260,1028,2714,1367,573,835,284,2,57,9,727,612,1820,58,2,2,2,189,1108,1712,28,2584,402,1693,664,630,8,300,1270,773,276,1230,609,134,978,430,2487,850,525,2,2,599,5,2,2,1963,528,3,1959,105,465,556,905,1378,966,942,108,334,130,1190,154,386,8,1003,81,7,3,25,463,620,29,989,406,458,1847,93,676,536,427,269,1456,1,2833,313,876,412,2,557,73,1483,698,59,318,273,108,167,323,744,101,1119,38,363,557,438,135,145,155,497,2,718,383,978,487,47,1080,901,387,422,659,359,8,59,32,416,283,9,1,211,2,460,25,60,386,282,528,307,2,67,30,13,1,255,122,143,217,37,628,255,1,1125,264,28,7,2,479,241,129,43,200,188,481,709,29,57,201,337,65,97,167,82,24,7,109,1049,14,758,7,127,179,9,21,261,1413,5977597,12,1861,681,134,43,5997424,90,2800095,4,1572,549,332,445,1,2,80,1,90,0,583,6,307,1,8,1,2,2132,1,1,1,1,1,414,1,748,141,297,169,301,24,2,8,96,50,2,47,22307501",authuser:0,kscs:"c9c918f0_cnf8n6KpIeVwXx-aLoDA",kGL:"IN"};google.kHL="en-IN");})();google.time=function(){return(new Date).getTime();})();function(){google.lc=[{};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

10 To see the JSON output, click **VIEW OUTPUT JSON**.

```

1  {
2    "responseHeaders": {
3      "X-Frame-Options": "SAMEORIGIN",
4      "Transfer-Encoding": "chunked",
5      "Cache-Control": "private, max-age=0",
6      "Server": "gws",
7      "Alt-Svc": "quic=\":443\"; ma=2592000; v=\"44,43,39,35\"",
8      "Set-Cookie": "NID=148
          =RTUkvjVhyg9KVAZRI58yCCSEw8WosYfn9MwDfQ1N5fndSDaVrXUM5Bj38PyKXK1Z_rNp3usxttMpd7YiqRUOSfHKTC7cTERbd
          UMonj3cTppHe3PHIXjPGHNTSZEweb3cxtjVihVo1S85ezVXaTSRYfcg0B_XIHZBkq8Buw1aE; expires=Tue, 28-May-2019
          22:45:06 GMT; path=/; domain=.google.com; HttpOnly",
9      "Expires": "-1",
10     "P3P": "CP-\"This is not a P3P policy! See g.co/p3phelp for more info.\"\"",
11     "X-XSS-Protection": "1; mode=block",
12     "Date": "Mon, 26 Nov 2018 22:45:06 GMT",
13     "Content-Type": "text/html; charset=ISO-8859-1"
14   },
15   "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/google/1x/google_standard_color_128dp.png\" itemprop=\"image\"><title>Google</title><script
          nonce=\"8WwM/ydugKGrSCHU6QQG==\">(function(){window.google={kEI:'cnf8W6KpJieVkwXx-ALoDA',kEXPI:'0
          ,1353747,57,50,1150,454,303,1017,1120,286,690,527,730,142,104,293,132,278,420,350,30,524,27,275,401,457
          ,110,114,56,164,2336158,235,32,45,23,6,1,329219,1294,12303,4855,19977,13114,8163,7085,067,6056,636,2239
          ,3232,5281,1100,3335,2,2,4605,2196,369,1212,2102,4133,1372,224,867,1331,260,1028,2714,1367,573,835,204
          ,2,579,727,612,1820,58,2,2,189,1108,1712,28,2584,402,1693,664,630,8,300,1270,773,276,1230,609,134,978
          ,430,2487,850,525,22,599,5,2,2,1963,528,3,1959,105,465,556,905,1370,966,942,108,334,130,1190,154,386,8
          ,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

```

Path finder Enter key

Results

Congratulations! You configured a REST task that invoked a REST API and sent information between vRealize Automation 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 vRealize Automation Code Stream with other applications so that you can develop and deliver your software applications. Consider using poll tasks to poll the API until it completes and the pipeline task meets the exit criteria.

Connecting vRealize Automation Code Stream to endpoints

6

vRealize Automation 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 to integrate vRealize Automation Code Stream with other development applications.

To integrate vRealize Automation Code Stream with JIRA, an external plug-in is not needed, because the JIRA ticket creation capability is built into vRealize Automation Code Stream 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 vRealize Automation Code Stream](#)
- [How do I integrate vRealize Automation Code Stream with Jenkins](#)
- [How do I integrate vRealize Automation Code Stream with Git](#)
- [How do I integrate vRealize Automation Code Stream with Gerrit](#)
- [How do I integrate vRealize Automation Code Stream with vRealize Orchestrator](#)

What are Endpoints in vRealize Automation Code Stream

An endpoint is an instance of a DevOps application that connects to vRealize Automation Code Stream to provide data for your pipelines to run, such as a data source, repository, or notification system.

Your role in vRealize Automation 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 vRealize Automation Code Stream](#).

To connect vRealize Automation 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 vRealize Automation 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.

Table 6-1. Endpoints that vRealize Automation 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 embedded before you run the pipeline. When the pipeline runs, vRealize Automation Code Stream downloads a binary file that uses cURL 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.	1.9.*	
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	
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 vRealize Automation Code Stream with Jenkins

vRealize Automation 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 vRealize Automation Code Stream. Then, you create a pipeline and add a Jenkins task to it.

Prerequisites

- Set up a Jenkins server that runs version 1.561 or later.
- Verify that you are a member of a project in vRealize Automation Code Stream. If you are not, ask a vRealize Automation Code Stream administrator to add you as a member of a project. See [How do I add a project in vRealize Automation 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 endpoint type select **Jenkins**. Then, enter a name and 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	<p>Path for the folder that groups your jobs. Jenkins can run all jobs in the folder. You can create sub folders. For example:</p> <ul style="list-style-type: none">■ <code>folder_1</code> can include <code>job_1</code>■ <code>folder_1</code> can include <code>folder_2</code>, which can include <code>job_2</code> <p>When you create an endpoint for <code>folder_1</code>, the folder path is <code>job/folder_1</code>, and the endpoint only lists <code>job_1</code>.</p> <p>To obtain the list of jobs in the child folder named <code>folder_2</code>, you must create another endpoint that uses the folder path as <code>/job/folder_1/job/folder_2/</code>.</p>
URL	<p>Host URL of the Jenkins server. Enter the URL in the form of <code>protocol://host:port</code>. For example: <code>http://192.10.121.13:8080</code></p>
Polling Interval	<p>Interval duration for vRealize Automation Code Stream to poll the Jenkins server for updates.</p>

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 vRealize Automation 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 ✖ [CREATE VARIABLE](#)

Folder Path: /job/DevFolder/

Poll Interval (sec) *: 15

Request Retries *: 5

Retry Wait Time (sec) *: 60

[SAVE](#) [VALIDATE](#) [CANCEL](#)

- 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.

The screenshot shows the 'Build and Deploy' task configuration in vRealize Automation. The task is named 'Build' and is of type 'Jenkins'. It is configured to 'Always' execute. The Jenkins endpoint is 'aa' and the job is 'add_numbers'. Two input parameters, 'Num1' and 'Num2', are both set to '22'. A 'Token' field is present but empty. Under 'Output Parameters', there are buttons for 'status', 'job', 'jobId', 'jobResults', and 'jobUrl'. The left pane shows a visual workflow with 'Stage0' containing 'Build' and 'Test' tasks. The bottom of the interface has 'SAVE', 'RUN', and 'CLOSE' buttons, along with a status message 'Last saved a month ago' and a chat icon.

Build and Deploy Enabled

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

Stage0

Build Jenkins

Test Jenkins

Parallel Task *

Stage

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 ACTIONS

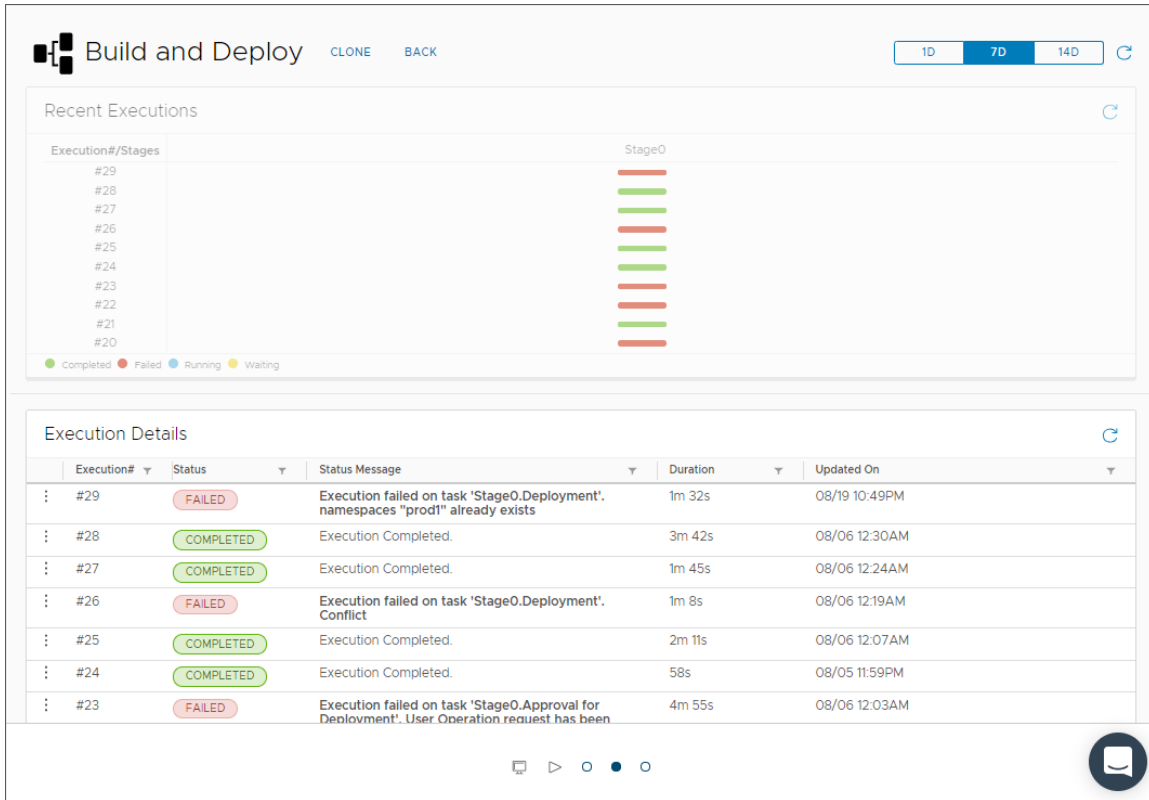
Stage0

✓ 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	<input type="checkbox"/>														
Execute Task	<input checked="" type="radio"/> Always <input type="radio"/> On Condition														
Jenkins Job															
Endpoint	aa														
Job Name	add_numbers														
Job ID	1428														
Job URL	http://localhost:8080/job/add_numbers/1428/														
Job Result	<table> <thead> <tr> <th>Key</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>junitResponse.failCount</td> <td>0</td> </tr> <tr> <td>junitResponse.skipCount</td> <td>0</td> </tr> <tr> <td>junitResponse.totalCount</td> <td>0</td> </tr> <tr> <td>junitResponse.successCount</td> <td>0</td> </tr> <tr> <td>jacocoResponse.lineCoverage</td> <td>0</td> </tr> <tr> <td>jacocoResponse.classCoverage</td> <td>0</td> </tr> </tbody> </table>	Key	Value	junitResponse.failCount	0	junitResponse.skipCount	0	junitResponse.totalCount	0	junitResponse.successCount	0	jacocoResponse.lineCoverage	0	jacocoResponse.classCoverage	0
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.



Results

Congratulations! You integrated vRealize Automation 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 vRealize Automation Code Stream to endpoints](#).

How do I integrate vRealize Automation Code Stream with Git

vRealize Automation 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. vRealize Automation Code Stream connects to the Git endpoint through a webhook.

To define a Git endpoint in vRealize Automation 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 vRealize Automation 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 vRealize Automation Code Stream. If you are not, ask a vRealize Automation Code Stream administrator to add you as a member of a project. See [How do I add a project in vRealize Automation 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 vRealize Automation Code Stream](#)
- [Chapter 2 Setting up vRealize Automation 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, enter **`https://api.github.com/vmware-example/repo-example`**.
- 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 password, private token, or private key that goes with the user name.
 - Password. Your password provides complete access to the repository. You can also create a variable for the password.

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.
 - Private key. This SSH key is a private key that provides access to a specific repository. When a Git event occurs, vRealize Automation Code Stream uses this key to clone a repository. See <https://help.github.com/articles/reviewing-your-ssh-keys/>.

- Click **Validate**, and verify that the endpoint connects to vRealize Automation 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

ACCEPT CERTIFICATE

Branch *

master

Authentication Type *

Password

Username *

ExampleUser

Password *

.....

CREATE VARIABLE

CREATE

VALIDATE

CANCEL

What to do next

To learn more, review the other sections. See [How do I use the Git trigger in vRealize Automation Code Stream to run a pipeline](#).

How do I integrate vRealize Automation Code Stream with Gerrit

vRealize Automation Code Stream provides a way to trigger a pipeline if a code review occurs in your Gerrit project. The Gerrit trigger definition includes the Gerrit project and the pipelines to run for different event types.

The Gerrit trigger uses a Gerrit listener on the Gerrit server that you want to monitor. To define a Gerrit endpoint in vRealize Automation 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.

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 vRealize Automation Code Stream. If you are not, ask a vRealize Automation Code Stream administrator to add you as a member of a project. See [How do I add a project in vRealize Automation Code Stream](#).

Procedure

1 Define a Gerrit endpoint.

- a Click **Configure > Endpoints** and click **New Endpoint**.
- b Select a project and for the endpoint type, select Gerrit. Then, enter a name and 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 you want the password to be encrypted, click **Create Variable** and select the type:

- Secret. The password is resolved at the time of execution by a user with any role.
- Restricted. The password is resolved at the time of execution by a user with the Admin role.

For value, enter the password that you want to 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 located in the .ssh directory.

- g (Optional) If a passphrase is associated with the private key, enter the passphrase.

If you want the passphrase to be encrypted, click **Create Variable** and select the type:

- Secret. The passphrase is resolved at the time of execution by a user with any role.
- Restricted. The passphrase is resolved at the time of execution by a user with the Admin role.

For value, enter the passphrase that you want to be secure, such as the passphrase for an SSH server.

- 2 Click **Validate**, and verify that the Gerrit endpoint in vRealize Automation Code Stream connects to the Gerrit server.

If it does not connect, correct any errors, then attempt to validate it 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 *: CREATE VARIABLE

Private Key *:

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-128-CBC,F00CE0B6526AF67DC77ADCD0962DBF92
```

CREATE VARIABLE

Pass Phrase ⓘ: CREATE VARIABLE

CREATE VALIDATE CANCEL

- 3 Click **Create**.

What to do next

To learn more, review the other sections. See [How do I use the Gerrit trigger in vRealize Automation Code Stream to run a pipeline](#).

How do I integrate vRealize Automation Code Stream with vRealize Orchestrator

vRealize Automation 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 vRealize Automation Code Stream. If you are not, ask a vRealize Automation Code Stream administrator to add you as a member of a project. See [How do I add a project in vRealize Automation Code Stream](#).
- In vRealize Automation 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 vRealize Automation 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 vRealize Automation 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 vRealize Automation Code Stream, and add a `vRO` task to your pipeline, the tagged workflow appears in the workflow list.

- 2 In vRealize Automation 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://cava-n-01-234.eng.vmware.com:8281`

- Do not use this format: `https://cava-n-01-234.eng.vmware.com:8281/vco/api`

- 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.

- 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**.

The screenshot shows the configuration window for a vRO workflow task. The window has tabs for 'Task : vRO workflow', 'Notifications', and 'Rollback'. A 'VALIDATE TASK' button is in the top right. The 'Task name' is 'vRO workflow'. The 'Type' is 'vRO'. The 'Duration' is 'NaNms (-)'. The 'Continue on failure' checkbox is unchecked. In the 'Execute task' section, the 'On condition' radio button is selected. Below it is a text box for the condition, which is currently empty and highlighted with a blue border. The 'Workflow Properties' section shows 'Endpoint' as 'vROEP', 'Workflow' as 'Test', and 'Greeting' as 'Hello!'. At the bottom, there are 'status' and 'properties' buttons.

Task name ⓘ *	vRO workflow
Type *	vRO ▾
Duration	NaNms (-)
Continue on failure	<input type="checkbox"/>
Execute task	<input type="radio"/> Always <input checked="" type="radio"/> On condition
Condition ⓘ	<div></div> ⓘ
Workflow Properties	
Endpoint	vROEP ▾
Workflow	Test ▾
Greeting	Hello!
Output Parameters	
<div>status properties</div>	

- f Enter the conditions that apply when the pipeline runs.

When to run pipeline...	Select conditions...
On Condition	<p>Runs the pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped.</p> <p>The vRO task allows you to include a boolean expression, which uses the following operands and operators.</p> <ul style="list-style-type: none"> ■ Pipeline variables such as <code>\${pipeline.variableName}</code>. Only use curly brackets when entering variables. ■ Task output variables such as <code>\${Stage1.task1.machines[0].value.hostIp[0]}</code>. ■ Default pipeline binding variables such as <code>\${releasePipelineName}</code>. ■ Case insensitive Boolean values such as, <code>true</code>, <code>false</code>, <code>'true'</code>, <code>'false'</code>. ■ Integer or decimal values without quotation marks. ■ String values used with single or double quotation marks such as <code>"test"</code>, <code>'test'</code>. ■ String and Numeric types of values such as <code>==</code> Equals and <code>!=</code> Not Equals. ■ Relational operators such as <code>></code>, <code>>=</code>, <code><</code>, and <code><=</code>. ■ Boolean logic such as <code>&&</code> and <code> </code>. ■ Arithmetic operators such as <code>+</code>, <code>-</code>, <code>*</code>, and <code>/</code>. ■ Nested expressions using round brackets. ■ Strings that include the literal value <code>ABCD</code> are evaluated as false, and the task is skipped. ■ Unary operators are not supported. <p>An example condition might be <code>\${Stage1.task1.output} == "Passed" \${pipeline.variableName} == 39</code></p>
Always	If you select Always , the pipeline runs the task without conditions.

- 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.
- Click **Executions**.
 - Click the pipeline.
 - Click the task.
 - 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 vRealize Automation Code Stream, and added a vRO task in your vRealize Automation 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 vRealize Automation Code Stream

7

You can have vRealize Automation Code Stream trigger a pipeline when certain events occur.

For example, you can:

- Use the Docker trigger to run a pipeline when a new artifact is created or updated.
- Use the Git trigger to trigger a pipeline when developers update code.
- Use the Gerrit trigger to trigger a pipeline when developers review code.
- Use the `curl` command to have Jenkins trigger the pipeline after a build completes.

This chapter includes the following topics:

- [How do I use the Docker trigger in vRealize Automation Code Stream to run a continuous delivery pipeline](#)
- [How do I use the Git trigger in vRealize Automation Code Stream to run a pipeline](#)
- [How do I use the Gerrit trigger in vRealize Automation Code Stream to run a pipeline](#)

How do I use the Docker trigger in vRealize Automation Code Stream to run a continuous delivery pipeline

As a vRealize Automation Code Stream administrator or developer, you can use the Docker trigger in vRealize Automation 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 vRealize Automation 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 vRealize Automation Code Stream.

Table 7-1. How to use the Docker trigger

What you do...	More information about this action...
Create a Docker registry endpoint.	<p>For vRealize Automation 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.</p> <p>The Docker registry endpoint includes the URL to the Docker Hub repository.</p>
Add input parameters to the pipeline that auto inject Docker parameters when the pipeline runs.	<p>You can inject Docker parameters into the pipeline. Parameters can include the Docker event owner name, image, repository name, repository namespace, and tag.</p> <p>In your CD pipeline, you include input parameters that the Docker webhook passes to the pipeline before the pipeline triggers.</p>
Create a Docker webhook.	<p>When you create the Docker webhook in vRealize Automation Code Stream, it also creates a corresponding webhook in Docker Hub. The Docker webhook in vRealize Automation Code Stream connects to Docker Hub through the URL that you include in the webhook.</p> <p>The webhooks communicate with each other, and trigger the pipeline when an artifact is created or updated in Docker Hub.</p> <p>If you update or delete the Docker webhook in vRealize Automation Code Stream, the webhook in Docker Hub is also updated or deleted.</p>
Add and configure a Kubernetes task in your pipeline.	<p>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.</p>
Include a local YAML definition in the task.	<p>The YAML definition that you apply to the deployment task includes the Docker container image and any secret keys that are required to pull the image from the repository for deployment.</p>

When an artifact is created or updated in the Docker Hub repository, the webhook in Docker Hub notifies the webhook in vRealize Automation Code Stream, which triggers the pipeline. The following actions occur:

- 1 Docker Hub sends a POST request to the URL in the webhook.
- 2 vRealize Automation 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 vRealize Automation 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 vRealize Automation 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 vRealize Automation 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 vRealize Automation 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 vRealize Automation Code Stream. If you are not, ask a vRealize Automation Code Stream administrator to add you as a member of a project. See [How do I add a project in vRealize Automation Code Stream](#).

Procedure

- 1 Create a Docker registry endpoint.
 - a Click **Endpoints**.
 - b Click **New Endpoint**.
 - c Enter a relevant name.
 - d Select the server type as **Docker Hub**.

- e Enter the URL to the Docker Hub repository.
- f Enter the name and password that is used to access the repository.

Edit Endpoint

Project	AWS_PGProj
Type	Docker Registry
Name *	dockerhub-endpoint
Description	<div></div>
Mark as restricted	<input type="checkbox"/> non-restricted
Server Type *	DockerHub
Repo URL *	https://cloud.docker.com/repository/docker/autom:
Authentication Type *	Password
Username *	admin
Password *	<div>.....</div> <div>CREATE VARIABLE</div>

SAVE
VALIDATE
CANCEL

- 2 In your CD pipeline, set the input properties to auto inject Docker parameters when the pipeline runs.

sm-1 Enabled

Workspace **Input** Model Output

Input Parameters ⓘ

Auto inject parameters ☐ Gerrit ☐ Git ☒ Docker ☐ None

ADD
ADD/REMOVE INJECTED PARAMETERS

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 CI/CD pipeline and configure the workspace](#).

- g Enter the API Token.

The CSP API token authenticates you for external API connections with vRealize Automation 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

Activity **Webhooks for 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 * `[redacted]` [?] CREATE VARIABLE GENERATE TOKEN

Image [?] `Image`

Tag [?] `Tags`

SAVE CANCEL

- 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#\n#\u0009\u0009docker run -t -i -rm -e http_proxy http://192.168.1.2:3142/debian bash\n#\nFROM\u0009\u0009ubuntu\n#\nVOLUME\u0009\u0009[/var/cache/apt-cacher-ng]\nRUN\u0009\u0009apt-get update ; apt-get install -yq apt-cacher-ng\n#\nEXPOSE\n\u0009\u000993142\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 vRealize Automation 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`.

vRealize Automation 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

GUIDED SETUP

Activity
Webhooks for Docker

↻

Commit Time	Webhook	Image	Tag	Owner	Repository	Pipeline	Execution Status
01/09/2019 10:59 AM	dt1l-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	dt1l-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 track your pipeline as it runs.

Executions

417 items
GUIDED SETUP

+ NEW EXECUTION

🔍

📄
🔼
🔽
↻

sm-1-IX#1

0 🐞

RUNNING

Stages:

ACTIONS ▾

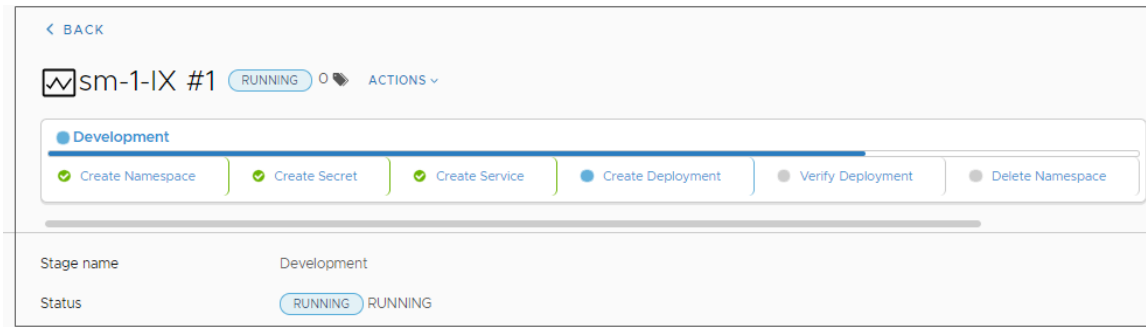
By k on 01/09/2019 2:41 PM

RUNNING

☆ Input : n/a

☆ Output : n/a

- 8 Click the running stage and view the tasks as the pipeline runs.



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 vRealize Automation Code Stream to run a pipeline

As a vRealize Automation Code Stream administrator or developer, you can use the Git trigger to integrate vRealize Automation Code Stream with the Git lifecycle. When you make a code change in GitHub, GitLab, or Bitbucket Enterprise, the event communicates with vRealize Automation Code Stream through a webhook and triggers a pipeline to run. The webhook works with GitLab, GitHub, and Bitbucket on-premises enterprise versions when both vRealize Automation Cloud Assembly and the enterprise version are reachable on the same network.

When you add the webhook for Git in vRealize Automation Code Stream, it also creates a webhook in the GitHub, GitLab, or the Bitbucket repository. If you update or delete the webhook later, the webhook in GitHub, GitLab, or Bitbucket is also updated or deleted.

Your webhook definition must include a Git endpoint on the branch of the repository that you want to monitor. vRealize Automation Code Stream uses the Git endpoint to create the webhook. 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.

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.

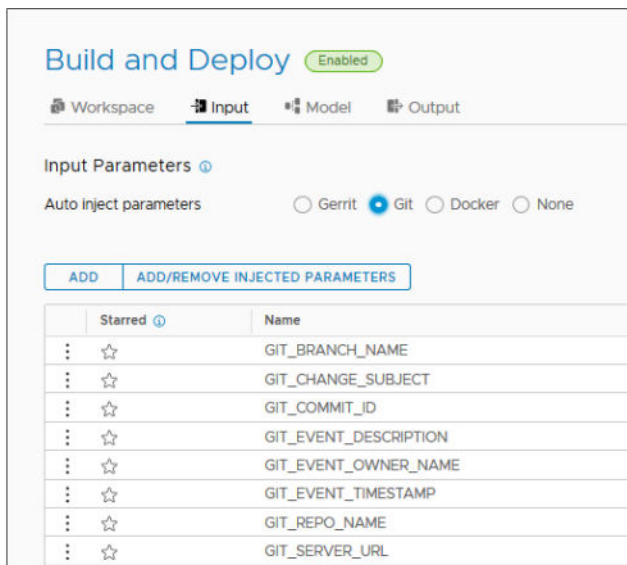
This example shows you how to use the Git trigger with a GitHub repository, but the prerequisites include preparations required if another Git server type is used.

Prerequisites

- Verify that you are a member of a project in vRealize Automation Code Stream. If you are not, ask a vRealize Automation Code Stream administrator to add you as a member of a project. See [How do I add a project in vRealize Automation Code Stream](#).
- Verify that you have a Git endpoint on the GitHub branch you want to monitor. See [How do I integrate vRealize Automation 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.



For information about input parameters, see [How you'll create the CI/CD pipeline and configure the workspace](#).

Procedure

- 1 In vRealize Automation 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 password in the endpoint, the webhook will continue to work.

See [How do I integrate vRealize Automation Code Stream with Git](#).

- d (Optional) Enter the branch that you want the webhook to monitor.

If left unspecified, the webhook monitors the branch that is configured for the Git endpoint.

- e (Optional) Generate a secret token for the webhook.

If used, vRealize Automation 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. vRealize Automation 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.

- You provide file inclusions so that when any of the files in a commit match the files specified in the inclusion paths or regex, pipelines are triggered. With a regex specified, vRealize Automation 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.
- You provide file exclusions so that when all the files in a commit match the specified files in the exclusion paths or regex, pipelines are not triggered.
- When toggled on, Prioritize Exclusion ensures that pipelines are not triggered even if any of the files in a commit match the specified files in the exclusion paths or regex. The default setting is off.

If conditions for both inclusion and exclusion are met, pipelines are not triggered.

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	<input type="checkbox"/>		

- 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 vRealize Automation 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 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 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

Activity **Webhooks for Git**

Webhook URL ⓘ `https://ca[REDACTED]om/codestream/api/git-webhook-listeners/963b2287-527f-4e9b`

Project `test`

Name * `test-webhook`

Description

Endpoint `DemoApp-Git`

Branch ⓘ `master`

Secret token ⓘ * `GYH0cBWZx4dUn47Y/KA8H/BOKts=` GENERATE

File ⓘ

Inclusions `--Select--` `Value` +

Exclusions `--Select--` `Value` +

Prioritize Exclusion ☐

Trigger

For Git ☒ PUSH ☐ PULL REQUEST

API token * `.....` ✖ CREATE VARIABLE GENERATE TOKEN

Pipeline * `CICD-2` ⓘ

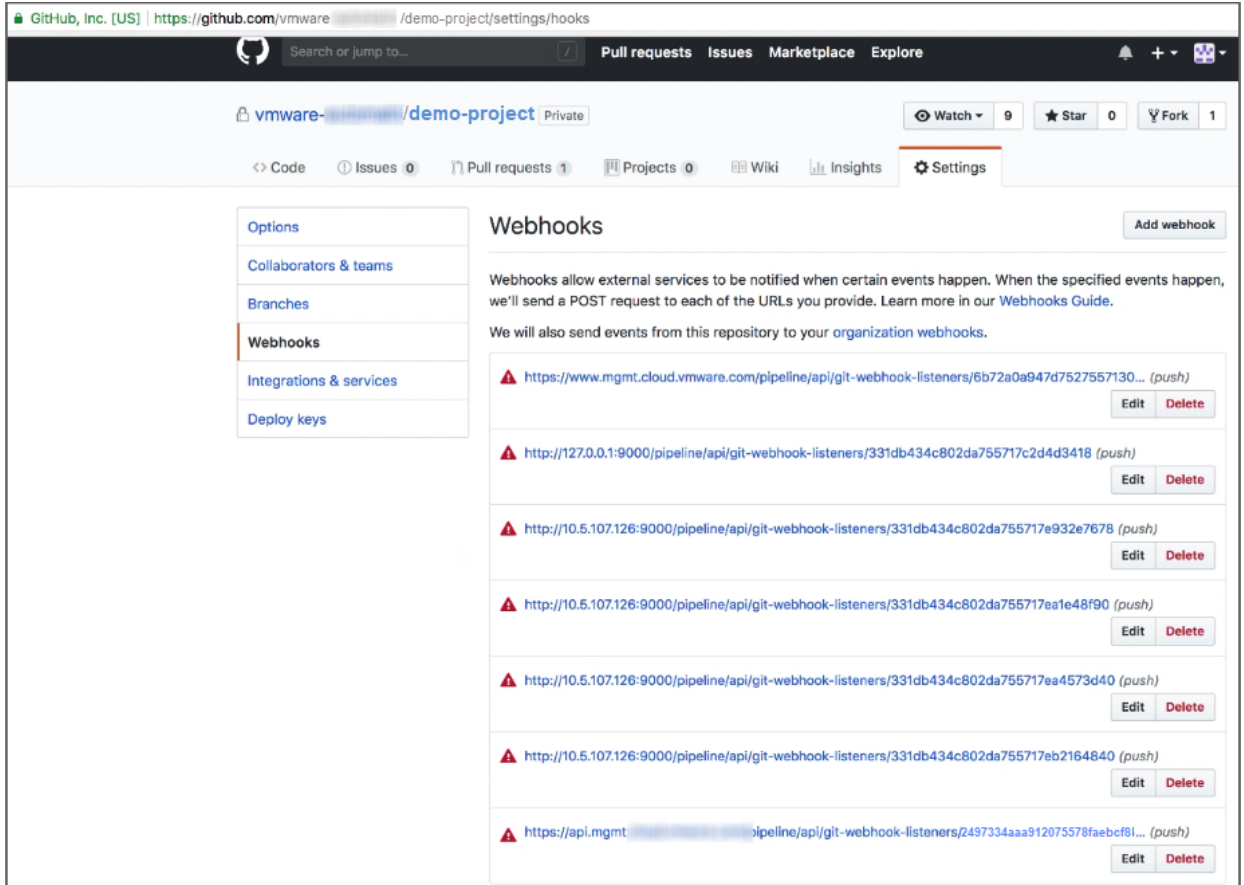
Comments

Execution trigger delay ⓘ `1`

SAVE CANCEL

- 4 In a new browser window, open the GitHub repository that is connected through the webhook.
 - a To see the webhook that you added in vRealize Automation Code Stream, click the **Settings** tab and select **Webhooks**.

At the bottom of the webhooks list, you see the same webhook URL.



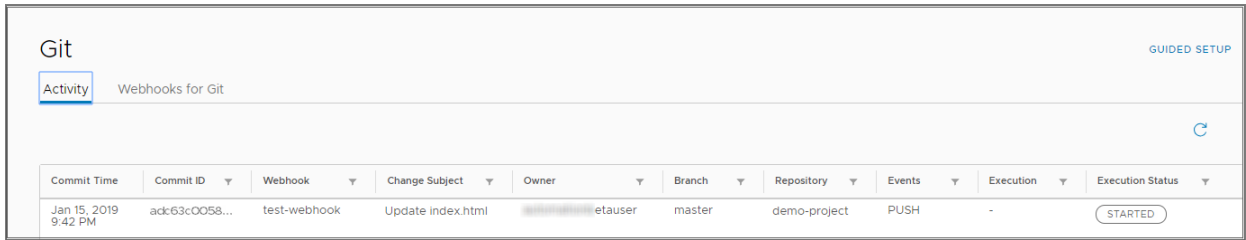
- b To make a code change, click the **Code** tab and select a file to edit on the branch to edit. Commit the change.
 - c To verify that the webhook URL is working, click the **Settings** tab and select **Webhooks** again.

At the bottom of the webhooks list, a green checkmark appears next to the webhook URL.



- Return to vRealize Automation Code Stream to view the activity on the Git webhook. Click **Triggers > Git > Activity**.

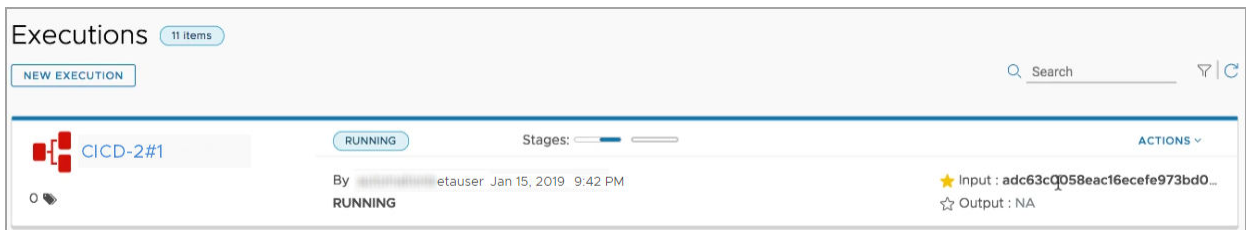
Under Execution Status, verify that the pipeline run has started.



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	etauser	master	demo-project	PUSH	-	STARTED

- Click **Executions** to track your pipeline as it runs.

You can press the refresh button to watch the pipeline execution.



Execution Name	Status	Stages	By	Time	Input	Output
CICD-2#1	RUNNING	Stages: 0/1	etauser	Jan 15, 2019 9:42 PM	Input : adc63c0058eac16ecef973bd0...	Output : NA

Results

Congratulations! You have successfully used a Git trigger to run a pipeline.

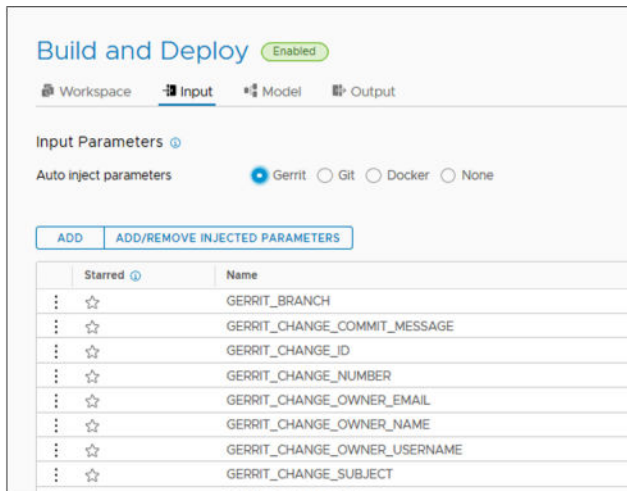
How do I use the Gerrit trigger in vRealize Automation Code Stream to run a pipeline

As a vRealize Automation Code Stream administrator or developer, you can use the Gerrit trigger to integrate vRealize Automation Code Stream with the Gerrit code review lifecycle. 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 trigger for Gerrit, you select a Gerrit listener, a Gerrit project on the server, and configure Gerrit events. In this example your 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 vRealize Automation Code Stream. If you are not, ask a vRealize Automation Code Stream administrator to add you as a member of a project. See [How do I add a project in vRealize Automation Code Stream](#).
- Verify that you have a Gerrit endpoint configured in vRealize Automation Code Stream. See [How do I integrate vRealize Automation Code Stream with Gerrit](#).
- For pipelines to trigger, verify that you set the input properties to inject Gerrit parameters when the pipeline runs.



For information about input parameters, see [How you'll create the CI/CD pipeline and configure the workspace](#).

Procedure

- 1 In vRealize Automation Code Stream, click **Triggers > Gerrit**.
- 2 (Optional) Click the **Listeners** tab, then click **New Listener**.

Note Skip this step if the Gerrit listener that you plan to use for the Gerrit trigger is already defined.

- 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 vRealize Automation 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. If you copied the token, the API token displays the masked token.

The screenshot shows the 'Gerrit' configuration window with the 'Listeners' tab selected. The form contains the following fields and controls:

- Project ***: A text input field containing 'test1' with a clear (X) button.
- Name ***: A text input field containing 'Gerrit-Demo-Listener'.
- Endpoint ***: A dropdown menu showing 'corporate-gerrit'.
- API token ***: A text input field containing '\${var.CSuser API Token}' with a green checkmark icon to its right.

At the bottom right of the form are two buttons: 'CREATE VARIABLE' and 'GENERATE TOKEN'. At the bottom left are three buttons: 'CREATE' (highlighted in dark blue), 'VALIDATE', and 'CANCEL'.

- 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 Gerrit 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.

vRealize Automation Code Stream uses the Gerrit listener selection to provide 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 is to be monitored.

- 9 (Optional) Provide file inclusions or exclusions as conditions for the trigger.

- You provide file inclusions to have trigger pipelines. When any of the files in a commit match the files specified in the inclusion paths or regex, pipelines are triggered. With a regex specified, vRealize Automation 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 to keep pipelines from triggering. When all the files in a commit match the files specified in the exclusion paths or regex, the pipelines are not triggered.
- When toggled on, Prioritize Exclusion ensures that pipelines are not triggered. The pipelines will 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 for both inclusion and exclusion are met, pipelines are not triggered.

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		<input type="checkbox"/>	

- 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.

10 Click **New Configuration**.

- 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**.
- Select the pipeline to trigger.

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 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.
- Default values used to configure the trigger will not be updated if the parameter values in the pipeline model are changed.

For information about Auto inject input parameters for Gerrit triggers, see the [Prerequisites](#).

- 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.
- 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, then **Gerrit-Pipeline** is triggered.
- If a **Patchset Created** event occurs in the Gerrit project, then **Gerrit-Trigger-Pipeline** and **Gerrit-Demo-Pipeline** are triggered.

Gerrit

GUIDED SETUP

Activity
Triggers
Listeners

Project * ⓧ

Name * Gerrit-Demo-Trigger

Gerrit Listener * Gerrit-Demo-Listener ▾

Gerrit project * Gerrit-Demo-Project ▾

Branch * master

File ⓘ

Inclusions -- Select Type -- ▾

Exclusions -- Select Type -- ▾

Prioritize Exclusion ☐

value +

value +

+ NEW CONFIGURATION

	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**.

When the trigger is enabled, it uses the Gerrit listener to start monitoring events that occur on the branch of the Gerrit project.

When you create the trigger, you include the repository where the code commit occurs. If you want to create a trigger with the same file inclusion or exclusion conditions but with a different repository, for example, you can click **Actions > Clone** on the trigger card. Then click **Open** on the new trigger and change parameters.

Results

Congratulations! You have 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, check the Activity tab for the Gerrit event in vRealize Automation Code Stream. Verify that the list of activities includes entries corresponding to every pipeline execution configured in the trigger. When an event occurs, only pipelines in the Gerrit trigger that are related to the particular event type will 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 choose the columns to display.

- The Change Subject and Execution columns are empty if the trigger was a direct git push.
- The Gerrit Trigger column shows the trigger that created the event.
- The Listener is off by default. When selected, it shows the Gerrit listener that received the event. One listener can be associated with multiple triggers.
- The Trigger Type is off by default. When selected, it shows if the trigger was triggered manually or automatically.

The screenshot displays the 'Gerrit' section of the vRealize Automation Code Stream interface. The 'Activity' tab is selected, showing a table of activity entries. A 'Show columns' dialog is open, allowing users to select which columns to display. The table shows three entries: a completed execution, a waiting execution, and a failed execution. The 'Show columns' dialog lists various columns with checkboxes to toggle their visibility.

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	1111Dummy	Gerrit-Pipeline #1	COMPLETED	Execution Completed.	Verified +1	Orlando...	test1	Gerrit-Demo-Trigger	master	Change Merged
Nov 12, 2019, 12:50:04 PM	19570 /6	11111Dummy	Gerrit-Pipeline #2	WAITING	Stage0.Task0: Execution Waiting for User Action.		Orlando...	test1	Gerrit-Demo-Trigger	master	Change Merged
		11111Dummy	Gerrit-Demo-Pipeline #1	FAILED	Stage0.Task0: User Operation request has been rejected by Fritz.	Verified -1	Orlando...	test1	Gerrit-Demo-Trigger	master	Patchset created
		11111Dummy	Gerrit-Trigger-Pipeline #1	WAITING	Stage0.Task0: Execution Waiting for User Action.		Orlando...	test1	Gerrit-Demo-Trigger	master	Patchset created

The 'Show columns' dialog lists the following columns with checkboxes:

- ☒ Change#
- ☒ Change Subject
- ☒ Execution
- ☒ Status
- ☒ Message
- ☒ Action taken
- ☒ User
- ☒ Gerrit project
- ☒ Gerrit Trigger
- ☐ Listener
- ☒ Branch
- ☒ Event
- ☐ Trigger Type

Buttons: TRIGGER MANUALLY, SELECT ALL, ITEMS per page: 20, 1 - 4 of 4 items

To control the activity for a completed or failed execution, 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 other problem, correct the mistake and select **Re-run** to run it again.
- If the pipeline fails to run because of a network connectivity issue or other problem, select **Resume** to restart the same pipeline execution. Doing so saves run time.
- Use **View Execution** to transfer to the Execution screen. 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 the **Trigger Manually** button and enter the name of the Gerrit trigger and Change-Id.

Monitoring pipelines in vRealize Automation Code Stream



As a vRealize Automation Code Stream Administrator or developer, you use vRealize Automation 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.

What are pipeline dashboards

A pipeline dashboard is a view of the results for a specific pipeline that ran, such as trends, top failures, and successful changes. vRealize Automation Code Stream creates the pipeline dashboard when you create a pipeline.

What are custom dashboards

A custom dashboard is a view of the results for one or more pipelines that ran. You create the custom dashboard and add widgets to display the results you want to see. 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.

This chapter includes the following topics:

- [How do I track key performance indicators for my pipeline in vRealize Automation Code Stream](#)

How do I track key performance indicators for my pipeline in vRealize Automation Code Stream

As a vRealize Automation Code Stream administrator or developer, you need insight about the performance of your pipelines in vRealize Automation Code Stream. You need to know how effectively your pipelines release code from development, through testing, and to production.

To gain insight, you can use the default dashboard for your pipeline, or use a custom dashboard.

- 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.

You can have vRealize Automation Code Stream measure the mean times to recover, deliver, or fail a pipeline over time, and display the trends for those mean times.

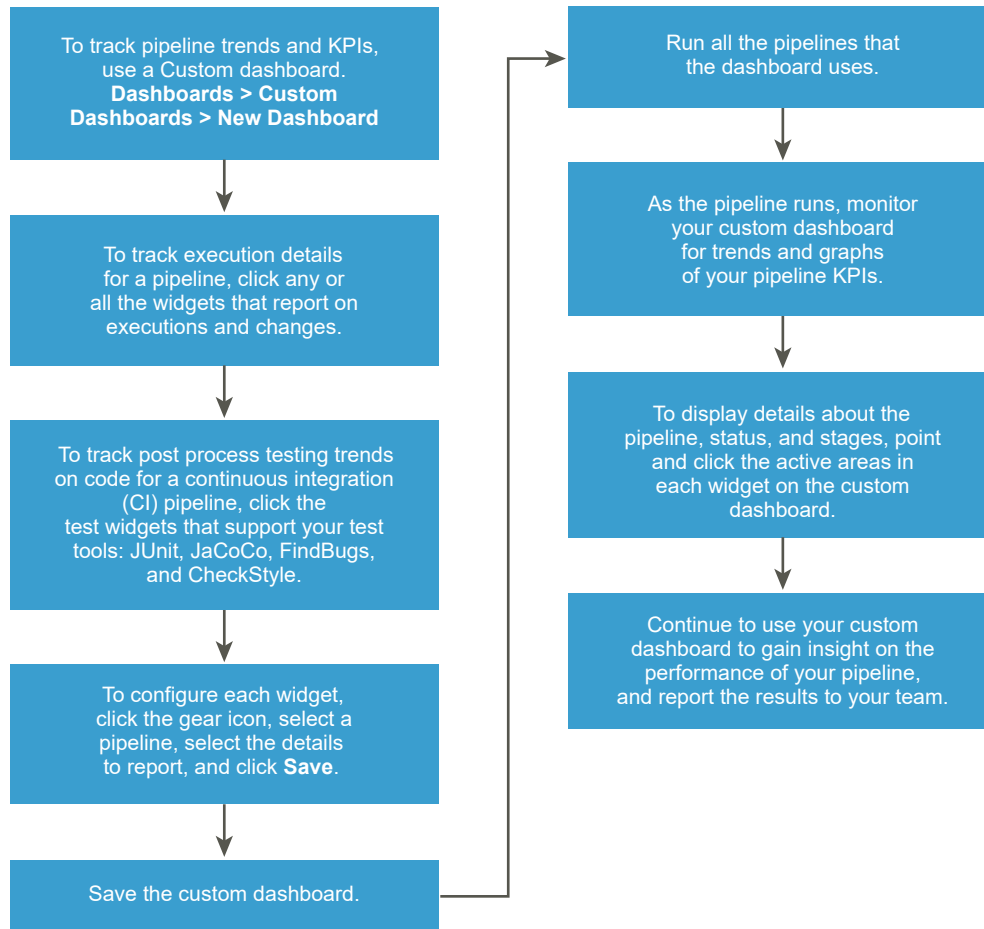
Table 8-1. Measuring mean times

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 deliver a pipeline	Average time required by the pipeline to deliver an update after it triggers.
Mean time between successful deployments	Time between successful deployments, which indicates how often a production environment updates.
Mean time to fail a pipeline	Time it takes for a pipeline to fail after it triggers.
Mean time to recover from a pipeline failure	Average time to deliver a successful pipeline after the pipeline fails. Measures time between the failure of a build or test criteria and the next build that produces a successful pipeline run, averaged over a week or month.

You can also have vRealize Automation Code Stream display the top failed tasks and stages in a pipeline. This 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.

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.



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.

IX KPIs

EDITDELETECLONEBACK

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 vRealize Automation 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 vRealize Automation Code Stream administrators and developers to learn more about vRealize Automation 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 vRealize Automation 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 vRealize Automation Code Stream](#)
- [More resources for vRealize Automation Code Stream Administrators and Developers](#)

What is Search in vRealize Automation 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.

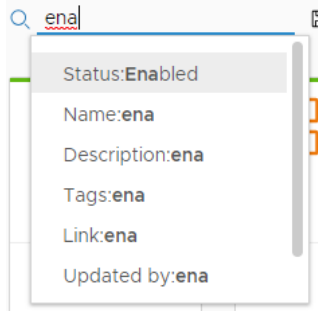
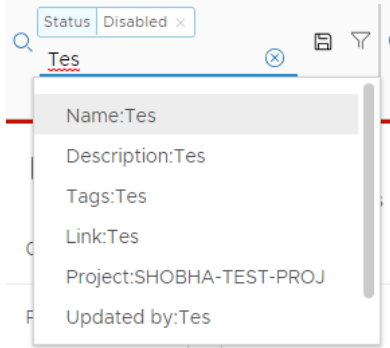
Where you search	Criteria to use for search
Pipeline Dashboards	Project, Name, Description, Tags, Link
Custom Dashboards	Project, Name, Description, Link (UUID of an item on the dashboard)
Executions	Name, Comments, Reason, Tags, Index, Status, Project, Show, Executed by, Executed by me, Link (UUID of 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

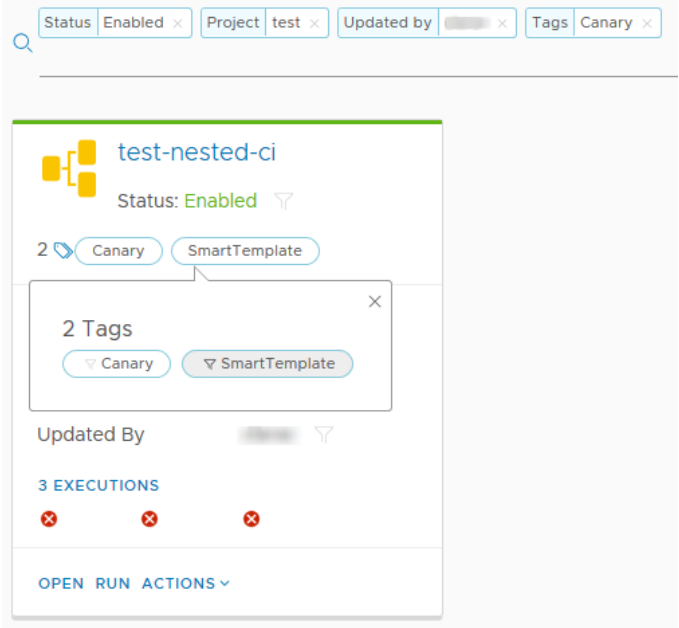
Where:

- 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.

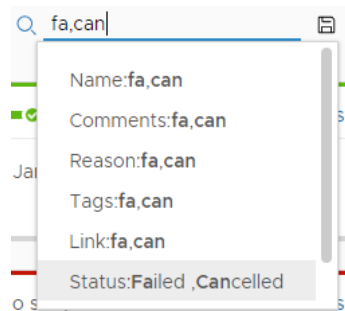
Search appears at the upper right of every valid page. When you start typing into the search blank, vRealize Automation Code Stream knows the context of the page and suggests options for the search.

Methods you can use to search	How to enter it
<p>Type a portion of the search parameter.</p> <p>For example, type ena to add a status filter that lists all the enabled pipelines.</p>	 <p>The screenshot shows a search bar with the text 'ena' entered. A dropdown menu is displayed below the search bar, listing several search suggestions: 'Status:Enabled', 'Name:ena', 'Description:ena', 'Tags:ena', 'Link:ena', and 'Updated by:ena'.</p>
<p>Add a filter to reduce the number of items found.</p> <p>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.</p>	 <p>The screenshot shows a search bar with the text 'Tes' entered. Above the search bar, there is a filter tag that says 'Status Disabled'. A dropdown menu is displayed below the search bar, listing several search suggestions: 'Name:Tes', 'Description:Tes', 'Tags:Tes', 'Link:Tes', 'Project:SHOBHA-TEST-PROJ', and 'Updated by:Tes'.</p>

Methods you can use to search	How to enter it
<p>Click the filter icon on properties of a pipeline or execution to reduce the number of items displayed.</p> <ul style="list-style-type: none"> 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. <p>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.</p>	

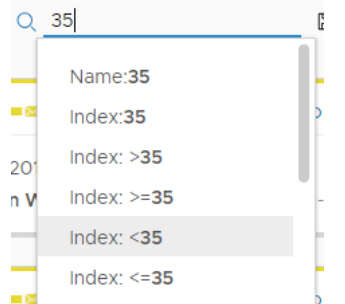
Use a comma separator to include all items in two execution states.

For example, type **fa,can** to create a status filter that works as an OR to list all failed or canceled executions.

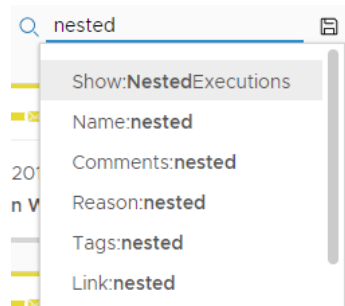


Type a number to include all items within an index range.

For example, type **35** and select **<** to list all executions with an index number less than 35.



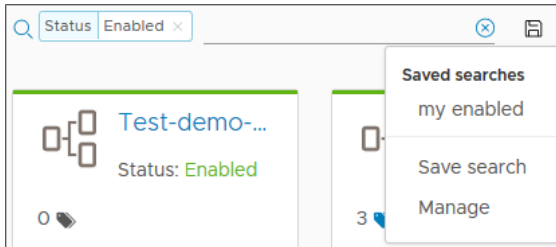
Pipelines that are modeled as tasks become nested executions and are not listed with all executions by default. To show nested executions, type **nested** and select the **Show** filter.



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.

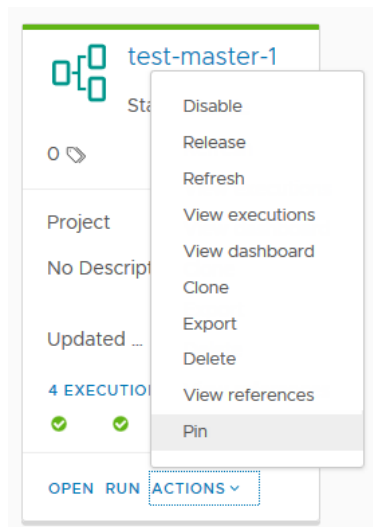
- You save a search by typing the parameters for search and clicking the icon to give the search a name such as **my enabled**.
- After saving a search, you click the icon to access the search. You can also select **Manage** to rename, delete, or move the search in the list of saved searches.



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 vRealize Automation Code Stream Administrators and Developers

As a vRealize Automation Code Stream administrator or developer, you can learn more about vRealize Automation Code Stream.

Table 9-1. More resources for administrators

To learn about...	See these resources...
<p>Other ways administrators can use vRealize Automation Code Stream:</p> <ul style="list-style-type: none"> ■ 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. 	<p>vRealize Automation Code Stream</p> <ul style="list-style-type: none"> ■ vRealize Automation Documentation ■ vRealize Automation product website <p>VMware Hands On</p> <ul style="list-style-type: none"> ■ 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...
<p>Other ways developers can use vRealize Automation Code Stream:</p> <ul style="list-style-type: none"> ■ 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 CI/CD pipeline breaks other services. 	<p>vRealize Automation Code Stream</p> <ul style="list-style-type: none"> ■ vRealize Automation Documentation ■ vRealize Automation product website <p>VMware Hands On</p> <ul style="list-style-type: none"> ■ Use the vRealize Automation Community. ■ Use the VMware Learning Zone. ■ Search the VMware Blogs. ■ Try the VMware Hands On Labs.