

# Using vRealize Code Stream

11 APRIL 2019

vRealize Code Stream 2.4



vmware®

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

<https://docs.vmware.com/>

If you have comments about this documentation, submit your feedback to

[docfeedback@vmware.com](mailto:docfeedback@vmware.com)

**VMware, Inc.**  
3401 Hillview Ave.  
Palo Alto, CA 94304  
[www.vmware.com](http://www.vmware.com)

Copyright © 2019 VMware, Inc. All rights reserved. [Copyright and trademark information.](#)

# Contents

Using vRealize Code Stream	4	
<b>1</b>	<b>Introducing vRealize Code Stream</b>	<b>5</b>
	Core Architectural Principles	7
	Roles and Responsibilities of Personas	8
	Integrating vRealize Code Stream with External Systems	9
	Key Release Automation Concepts	10
<b>2</b>	<b>Using Release Automation</b>	<b>12</b>
	Modeling a Release Pipeline Checklist	12
<b>3</b>	<b>Managing Release Automation</b>	<b>78</b>
	Clone a Task	78
	Delete a Task or a Cloned Task	79
	Copy a Release Pipeline	79
	Delete a Release Pipeline	79
	Resume From Failure	80
<b>4</b>	<b>Viewing Pipeline Execution Reports</b>	<b>81</b>
	View the Success Rate of the Executed Pipelines	81
	View Failed Pipelines	81
<b>5</b>	<b>Working with the Release Dashboard</b>	<b>83</b>
	View the Release Status	84
<b>6</b>	<b>vRealize Code Stream Execution Engine and Plug-In Framework</b>	<b>85</b>
	Troubleshooting the Execution Engine	85
	Troubleshooting PowerShell Task Failures Due to Concurrent Operations	86
	Troubleshooting PowerShell Task Failures Due to Memory Constraints	86
	Troubleshooting Failed Script Tasks Due to MaxSessions Setting	87
<b>7</b>	<b>vRealize Code Stream REST API and Example Scripts</b>	<b>88</b>
	Using a REST API to Export a Release Pipeline	88
	Using a REST API to Import a Release Pipeline	90
	Java Code to Run a Release Pipeline	91
	Example Script to Run a Release Pipeline	97

# Using vRealize Code Stream

*Using vRealize Code Stream* provides information about how to automate the release of applications, frequently while using existing tools in the build, development, test, provisioning, and monitoring environments.

## Intended Audience

This information is intended for anyone who wants to automate the release of applications in various development environments. The information is written for experienced developers and operation teams who are familiar with release automation.

# Introducing vRealize Code Stream

1

vRealize Code Stream automates the software release process by modeling all the necessary tasks in pipeline templates.

## Overview of vRealize Code Stream

A release pipeline is a sequence of stages. Each stage is composed of multiple tasks and environments that the software must complete before it is released to production. The stages can include development, functional testing, user acceptance test (UAT), load testing (LT), systems integration testing (SIT), staging, and production. Release managers can model and build pipelines, and release engineers can run pipelines.

Each stage in a pipeline includes a set of activities. The activities can provision a machine, retrieve an artifact, deploy software, run a test, create a manual task, or run a custom workflow or script, and so on.

When the software changes satisfy a set of rules called gating rules, they are promoted to the next stage in the pipeline. The gating rules include testing rules and compliance rules. Gating rules that are associated with a pipeline are specific to an organization or an application. Users can define gating rules when a pipeline template is created.

## Key Features

This release improves the performance and stability of the core platform, and the integrations with vRealize Automation and Cloud Foundry.

## Key Features in Previous Releases

The previous release of vRealize Code Stream simplified licensing, provided pipeline triggering utilities to enhance developer use cases, included new plug-ins to integrate with external systems over REST, and provided easier pipeline import and export capability. With simplified licensing, you can enable your vRealize Code Stream license from the VMware Application Management Interface.

The vRealize Code Stream Trigger for Git integrates vRealize Code Stream with the Git lifecycle. The Trigger for Git enables events from GitHub Enterprise or GitLab Enterprise to trigger a pipeline. When developers change their code in a GitHub or GitLab repository, the change triggers an event. That event passes through a Webhook to the trigger for Git, which triggers the pipeline. For more information about the vRealize Code Stream Trigger for Git, see <https://code.vmware.com>.

The vRealize Code Stream Trigger for Gerrit integrates vRealize Code Stream with the Gerrit code review lifecycle. The Trigger for Gerrit enables reviews and merges of developer code to trigger a pipeline based on events. When developers create a patch set for a code change, or merge code changes, Gerrit sends the change events to the trigger for Gerrit. The events trigger the tasks in your pipeline. For more information about the vRealize Code Stream Trigger for Gerrit, see <https://code.vmware.com>.

The following video explains these features.



What's New in the Previous Release of vRealize Code Stream

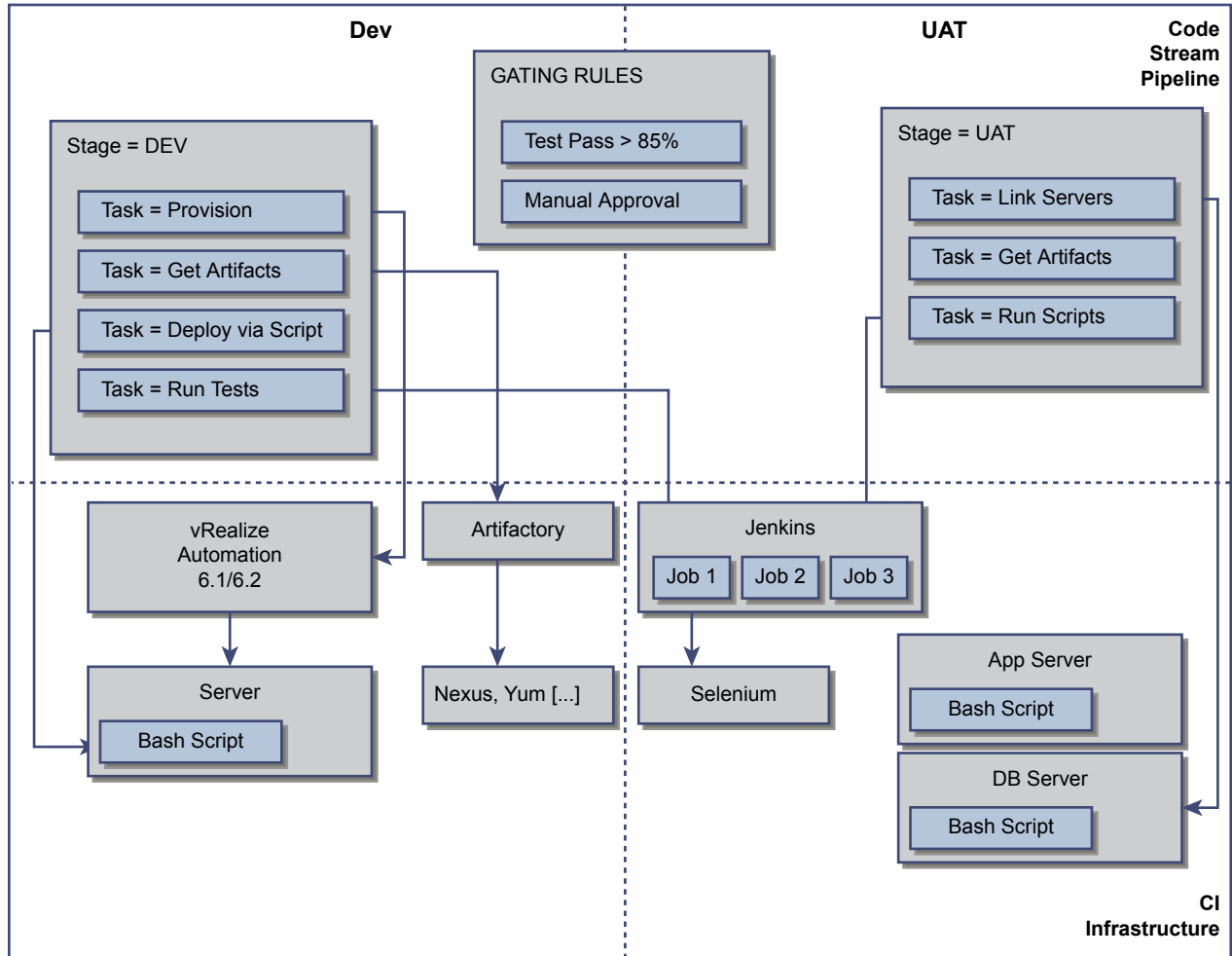
([http://link.brightcove.com/services/player/bcpid2296383276001?bctid=ref:video\\_whats\\_new\\_code\\_stream\\_2.3](http://link.brightcove.com/services/player/bcpid2296383276001?bctid=ref:video_whats_new_code_stream_2.3))

Also a previous release of vRealize Code Stream extended Role-Based Access Control (RBAC) to pipeline templates, offered the ability to resume a failed pipeline execution from the point of failure, and supported integration with remote jFrog Artifactory instances. It also introduced a Plug-in SDK to build custom plug-ins for vRealize Code Stream.

## Components of vRealize Code Stream

The following diagram shows the primary components included in vRealize Code Stream.

Figure 1-1. Main Components of vRealize Code Stream



This chapter includes the following topics:

- [Core Architectural Principles](#)
- [Roles and Responsibilities of Personas](#)
- [Integrating vRealize Code Stream with External Systems](#)
- [Key Release Automation Concepts](#)

## Core Architectural Principles

vRealize Code Stream works with deployment engines such as vRealize Automation, vCenter Server, and scripts. Interaction with Chef and Puppet is also possible through vRealize Orchestrator workflows or through Advanced Service Designer (ASD).

vRealize Code Stream provides extensibility through plug-ins. Both of the approval and extensibility components are embedded in the vRealize Automation virtual appliance.

For the supported vRealize Code Stream integrations, see [Integrating vRealize Code Stream with External Systems](#).

## Deployment Engines

vRealize Code Stream integrates with various provisioning and deployment solutions including vRealize Automation. It can also trigger scripts or vRealize Orchestrator workflows. Plug-ins that VMware, partners, or users publish support other provisioning solutions.

## Testing Frameworks

vRealize Code Stream integrates with Jenkins to trigger Jenkins jobs, including test routines through the plug-in framework.

A Jenkins job can run test cases that are configured for an application. The Test Acceptance Threshold workflow in the gating rule verifies the results of the Jenkins job and returns the response to the vRealize Code Stream server. Based on the results of the test and the gating rules that were defined, the build either proceeds to the next stage of the release pipeline or it fails.

vRealize Code Stream also integrates with Bamboo, Jenkins, and Team Foundation Server.

## Approval Systems

vRealize Code Stream uses vRealize Orchestrator plug-ins to integrate with approval systems. Manual approval tasks can be created within the vRealize Automation inbox. vRealize Code Stream can also integrate with BMC Remedy ITSM, HP Service Manager, ServiceNow, and other ticketing systems. The approval systems integration requires downloading and installing the appropriate vRealize Orchestrator plug-in from the VMware Solution Exchange.

## Roles and Responsibilities of Personas

A tenant administrator can assign the release manager and release engineer roles, which are a part of release automation.

These roles have various responsibilities when they interact with the product. For information on configuring more tenants, see the *Installation and Configuration* guide.

The following table lists the roles and responsibilities of the personas.



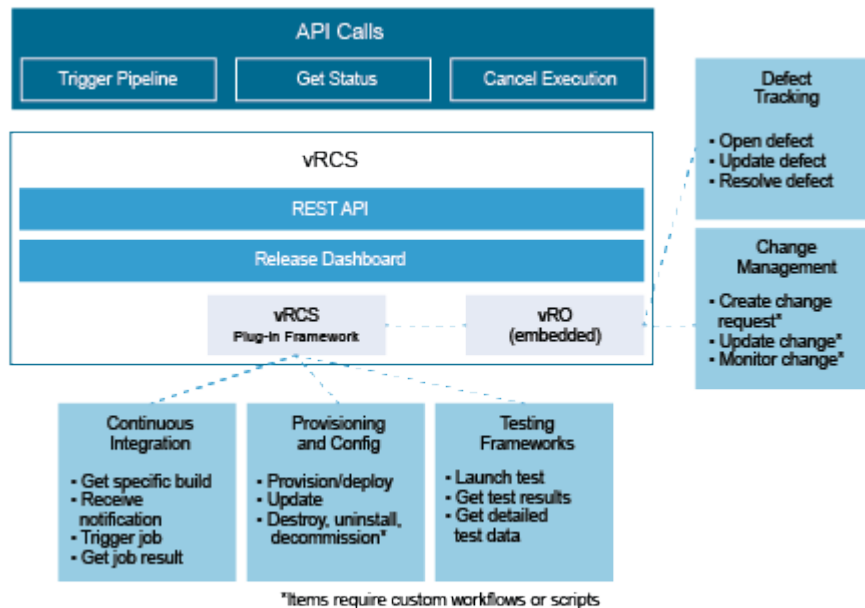
**Table 1-1. Roles and Responsibilities in vRealize Code Stream**

Role	Responsibilities
Release Manager	<ul style="list-style-type: none"> <li>■ View, create, delete, update, and publish a dashboard</li> <li>■ Set default dashboard for all users</li> <li>■ View, create, delete, and update an endpoint</li> <li>■ View, create, execute, update, and delete a release pipeline</li> <li>■ Modify and trigger a release pipeline</li> <li>■ View release pipeline details</li> <li>■ Perform artifact management administration tasks</li> </ul>
Release Engineer	<ul style="list-style-type: none"> <li>■ View, create, update, and delete a private dashboard</li> <li>■ Execute a release pipeline</li> <li>■ View endpoint</li> <li>■ Trigger a release pipeline</li> <li>■ View the details of a release pipeline</li> <li>■ View release pipeline execution</li> <li>■ Perform artifact management administration tasks</li> </ul>

## Integrating vRealize Code Stream with External Systems

vRealize Code Stream includes an extensibility framework that supports modular integrations with external systems, without changing the core platform.

Based on the type of external system, different mechanisms are recommended.

**Figure 1-2. Supported Integration with External Systems**

## Release Pipeline Integrations

Release pipeline templates support various tasks that can trigger actions in a wide category of systems such as continuous integration, testing frameworks, or defect tracking systems.

Integrations such as Atlassian Bamboo, Jenkins, Microsoft Team Foundation Server, Pivotal Cloud Foundry, and Bugzilla are supported natively. You can integrate other plug-ins by creating custom workflows using vRealize Orchestrator.

System Category	Integration Mechanism
Repository	External JFrog Artifactory
Continuous Integration	<ul style="list-style-type: none"> <li>■ Atlassian Bamboo</li> <li>■ Jenkins</li> <li>■ Pipelines</li> <li>■ Microsoft Team Foundation Server</li> </ul>
Provisioning and configuration management	<ul style="list-style-type: none"> <li>■ Pivotal Cloud Foundry</li> <li>■ Script</li> <li>■ vRealize Automation 6</li> <li>■ vRealize Automation 7</li> </ul>
Testing frameworks	<ul style="list-style-type: none"> <li>■ Atlassian Bamboo</li> <li>■ Jenkins</li> <li>■ Microsoft Team Foundation Server</li> </ul>
Defect tracking systems	<ul style="list-style-type: none"> <li>■ Bugzilla</li> <li>■ JIRA</li> </ul>

vRealize Orchestrator custom workflow is offered as a vRealize Code Stream plug-in. You can include other integrations such as Chef, Puppet, and JetBrains TeamCity using a vRealize Orchestrator workflow.

You can download vRealize Orchestrator plug-ins from the VMware Solution Exchange.

## Key Release Automation Concepts

Use the following definitions to help you understand the release pipeline modeling and the artifact management workflow.

### Artifact

A script or the output of a build process. The artifact can be deployed or upgraded in a given stage.

Artifact types can be configuration files, application bits, or third-party software.

### Artifact management

A service that manages the artifacts over a range of local and remote repositories.

For example, managing a WAR file stored in the Maven repository.

**Gating rule**

A set of rules that must be completed before the software changes are promoted and the next set of tasks starts in the subsequent stage.

The gating rules include testing rules and compliance rules. Gating rules that are associated with a pipeline are specific to an organization and applications.

Gating rules allow you to request approvals. You can set an expiration on the approval for the task. After the pipeline runs, you can examine the pipeline results to see who approved the task and when. If the approval is not granted in the time allowed, the pipeline fails.

**Endpoint**

A plug-in scenario that captures the specific configurations of a provider.

**Pipeline**

A collection of all the stages or environments in which a software change must pass through independently before it is released into production.

For example, development, test, user acceptance test, load test, staging, and production.

**Provider**

Service providers such as Jenkins, Bamboo, Bugzilla, Pivotal Cloud Foundry, and JIRA, which support various tasks.

**Stage**

Every stage in the pipeline defines a set of activities.

For example, deploy, test, approval through gating rules, and custom tasks.

**Task**

An activity in a given stage.

For example, an activity can provision the machines. It can also resolve the artifact, deploy the artifact, run the test, and so on. A manual task might be to open the port in a firewall.

**Reports**

Reports display the pipeline success and failure rate over time.

## Using Release Automation

The software development life cycle includes work phases before it moves to production. As the software changes move closer to production, the quality checks and approval policies become stringent. This process is enforced to ensure that no disruptions occur in the production environment.

vRealize Code Stream enables central IT to host and manage new application workloads that lines of business and development operation teams drive. Application teams can independently use vRealize Code Stream to automate and streamline their software release process while they continue to use their preferred provisioning and deployment tools.

vRealize Code Stream also enables applications or operations teams to model their software release process in a release pipeline. A release pipeline is a sequence of stages. Each stage is composed of multiple tasks and environments that the software must pass through before it is released to production. The stages can include development, functional testing, user acceptance test, load testing, systems integration testing, staging, and production.

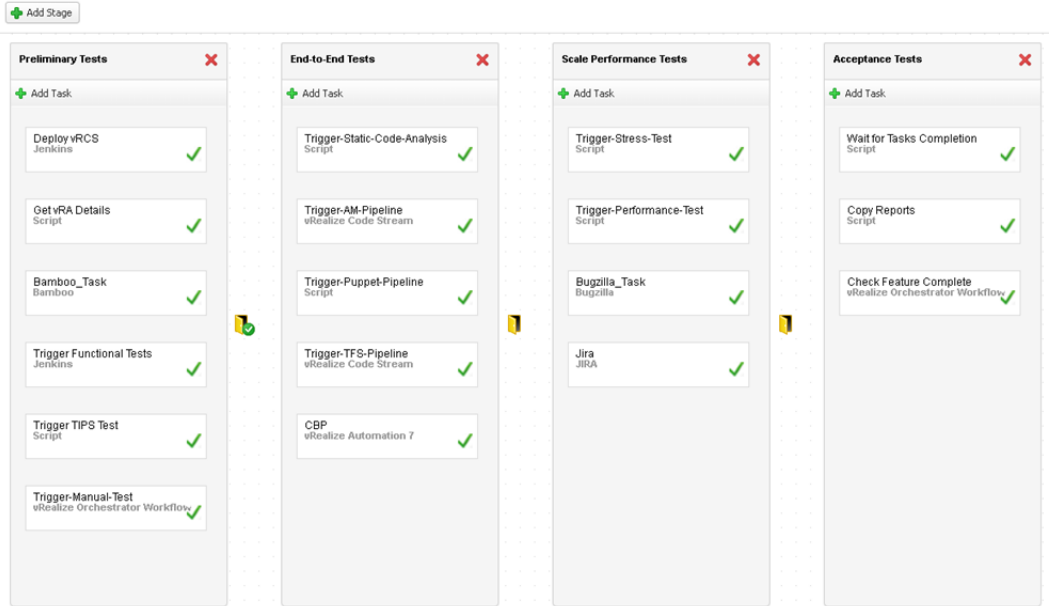
In each stage, teams might use different kinds of development and management tools. Using different tools results in the need to build a product that is extensible and that can integrate easily with various existing tools. To satisfy this need for flexibility, vRealize Code Stream offers a model-driven, open, and extensible architecture. With its catalog of plug-ins, you can integrate vRealize Code Stream with existing tools, including build and integration systems, testing frameworks, provisioning, deployment engines, change management systems, and so on.

Each stage in a pipeline includes a set of activities. The activities can provision a machine, retrieve an artifact, deploy software, run a test, create a manual task, or run a custom workflow or script, and so on. When the software changes satisfy a set of rules called gating rules, they are promoted to the next stage in the pipeline. The gating rules include testing rules and compliance rules. Gating rules that are associated with a pipeline are specific to an organization or an application. You can define gating rules when you create a pipeline template.

### Modeling a Release Pipeline Checklist

A release pipeline is a collection of stages where each stage represents a deployment environment. You can use stages to represent the stages that a software change must pass through independently before it is released. For example, stages might include development, test, user acceptance test, load test, systems integration testing, and staging environments.

The following diagram shows a sample release pipeline with several stages.

**Pipeline : E2E\_vRCSPipeLine**

The number of stages and the configuration of each stage varies based on the application, and whether the release is major, minor, patch, or based on organization release policies.

To observe how to model a release process in vRealize Code Stream, watch the following video.



See Modeling a Pipeline in vRealize Code Stream at

<http://www.youtube.com/watch?>

[v=SErodVvlnA4&list=PLrFo2o1FG9n7Pr4Fq-7exPvN6ThDYRcPV&index=1](http://www.youtube.com/watch?v=SErodVvlnA4&list=PLrFo2o1FG9n7Pr4Fq-7exPvN6ThDYRcPV&index=1).

To track your work as you complete the modeling tasks, complete the tasks in the order they are given.

### 1 Create a Release Pipeline

Pipelines comprise a single application or a module. You can model a pipeline using multiple stages, and each stage can include multiple tasks. Each stage represents a part of the release cycle for the application or module. You can create, edit, view, copy, and delete the release pipeline.

### 2 Importing Pipelines

To import a pipeline, you can enter the pipeline data directly. Or, you can import a JSON file or a TXT file that contains the pipeline data, from your local file system to import a previously saved pipeline. The import adds the pipeline to your vRealize Code Stream instance.

### 3 Exporting Pipelines

You can export an existing pipeline from your vRealize Code Stream instance to a JSON file. The exported pipeline file on your local file system uses the same name as the pipeline name. You can share the JSON file with other users so that they can import the same pipeline into their vRealize Code Stream instance.

#### 4 [Assign Role-Based Access Control Permissions at the Pipeline Level](#)

When you create or edit a pipeline, you can give permissions to users and groups so that they can modify and trigger the pipeline. The access permissions apply only to the users and groups that you add to the permissions lists.

#### 5 [Activate and Run a Release Pipeline](#)

After you create a release pipeline, you must activate it to complete the pipeline modeling. After you activate the pipeline, you can run it. A pipeline that is in draft state indicates that the Release Manager is still modeling the pipeline.

#### 6 [Configure a Bamboo Task](#)

When you register a Bamboo Server endpoint, you connect to a Bamboo server to start a build plan from a release pipeline. You can configure multiple artifacts from a particular plan.

#### 7 [Configure a Bugzilla Task](#)

You can execute a pipeline task which creates a Bugzilla task. You can use the output properties of the Bugzilla task in subsequent tasks in a release pipeline.

#### 8 [Configure a Cloud Foundry Task](#)

Use the Cloud Foundry task to model a pipeline task that connects to a Cloud Foundry instance. The task can deploy, scale, start, stop, and delete an application. You can also bind and unbind services to an application.

#### 9 [Configure a Jenkins Task](#)

You can use a test task to test a deployment. When you register a Jenkins Server endpoint, vRealize Code Stream connects to a Jenkins server and starts a build job from a release pipeline.

#### 10 [Configure a JIRA Task](#)

You can create and update JIRA tasks and issues in a release pipeline.

#### 11 [Configure a Poll Task](#)

You can use the vRealize Code Stream Poll Plug-in to listen for the status on a specific API to detect when a particular task completes. You configure the interval for the Poll plug-in so that it continues to poll the API until the task meets the exit criteria that you define.

#### 12 [Configure a REST Task](#)

You can use the REST Plug-In with vRealize Code Stream to invoke a REST API, which sends and receives data between external and internal applications, such as your Git repository, JIRA, and so on. The REST Plug-In enables applications that use a REST API to interact with other plug-ins to support continuous delivery of your development applications through the pipeline.

#### 13 [Configure a Team Foundation Server Task](#)

You can use the Team Foundation Server task to connect to the Team Foundation Server and invoke configured build and test jobs.

**14 Add a Custom Service Blueprint Task**

You can add a Custom Service Blueprint task to add a XaaS blueprint. A service blueprint task allows you invoke a plug-in available in the vRealize Automation service catalog from a release pipeline. You can publish a predefined workflow from vRealize Orchestrator to the vRealize Automation service catalog.

**15 Add a jFrog Artifactory Task**

An artifact task lets you search for artifacts from Artifactory. When you search for artifacts, Artifactory interfaces with various repositories such as Nexus and Yum.

**16 Add a Script Task**

You can use a custom script task to configure bash or PowerShell scripts and run scripts on any Linux or Windows machine. After the script runs, you can monitor the script progress and capture the script response. You can pass the script response as input to the other release pipeline tasks.

**17 Add a vRealize Automation Task**

You use a provision task to provision machines. You can register a vRealize Automation 6.x or 7.x instance with vRealize Code Stream to provision infrastructure in a specific environment.

**18 Add a vRealize Orchestrator Workflow Task**

With a vRealize Orchestrator task, you can configure a vRealize Orchestrator to be triggered as part of the release pipeline process.

**19 Create a Nested Pipeline**

You can model a complex pipeline and execute the pipeline in a modular manner by invoking pipelines within pipelines. You can also view the result of the nested pipeline execution.

**20 Configure Gating Rules**

Gating rules are a set of criteria that each stage must pass to proceed to the subsequent stage. You can configure gating rules based on your requirement for a pipeline.

## Create a Release Pipeline

Pipelines comprise a single application or a module. You can model a pipeline using multiple stages, and each stage can include multiple tasks. Each stage represents a part of the release cycle for the application or module. You can create, edit, view, copy, and delete the release pipeline.

Plug-ins can use binding variables as part of input configurations. These variables are replaced at runtime with the values where they are placed. The binding variable always precedes the shared pipeline variable. For example, `${pipeline.PipelineVariable}`.

Various pipeline execution variables are also available by default. These variables can be used in pipeline logic like conditional task skipping or, for example, a task that updates an external ticketing, issue tracking, or auditing system. The default pipeline execution variables include:

- `${releasePipelineName}`
- `${stageName}`
- `${nextStageName}`

- `${taskName}`
- `${nextTaskName}`
- `${requestBy}`
- `${executionId}`

### Prerequisites

- Verify that the following items are available to you: predefined vRealize Automation blueprints, workflows, scripts, or test jobs that perform tasks that the pipelines trigger.
- Verify that the artifacts in the Artifactory server repository are already populated so that you can use the artifact management capabilities.
- Verify that the applicable plug-ins and endpoints are registered. See the *Installation and Configuration* guide.
- Familiarize yourself with the input and output details required to create tasks.

### Procedure

- 1 Select **Code Stream > Pipelines**.
- 2 To create a pipeline, click **Add**.
- 3 Enter a name and description for the pipeline.
- 4 To define input properties for the pipeline, click **Add**.

These properties are only required to pass certain parameters at the time of triggering a pipeline run. You can reference these parameter values across all stages during modeling. The run time values are applied during the pipeline run.

Option	Description
<b>Name</b>	Enter a property name.
<b>Description</b>	Enter a description for the property.
<b>Value</b>	Enter a property value. For example, ABC-876541.

- 5 (Optional) To run this pipeline concurrently, select the check box named **Allow concurrent pipeline execution**.

The pipeline model reuses the same set of virtual machines to deploy a software change. The concurrent run overwrites the deployed change.



- 6 (Optional) To send notifications about the pipeline execution to Socialcast, select the check box named **Post updates to Socialcast**. Then, enter the **Socialcast Webhook Name** and **Socialcast Webhook URL**.

Socialcast updates are useful to collaborate among team members who work on the releases. The updates notify users about the pipeline execution status such as the pipeline execution start, pause, resume, cancelation, failure, and completion. A new pipeline execution creates a status message on the Socialcast group, and posts the subsequent statuses of the execution as comments to the message. You can add comments to individual threads, and track the progress of a pipeline execution.

The name of the Webhook you provide is the name of the group created for communication in Socialcast.

To locate the Webhook URL, navigate to **Socialcast > Configured Group > Integrations > Codestream > Add > Step 5**.

---

**Note** To add the integration and copy the Webhook URL, you must have administrator privileges for the group configured on Socialcast.

---

- 7 To send event notifications to users during the pipeline execution, add the email addresses of the recipients.

The users whose emails you enter must have access to the same vRealize Automation appliance.

- 8 Enter the applicable tags.

A tag is useful in grouping pipeline models or runs.

For example, you can use a tag to filter a pipeline model or run view.

- 9 To continue pipeline creation, click **Stages >**.
- 10 To add multiple stages to the pipeline, click **Add Stage**.
- 11 Double-click the default stage name and enter a name.

For example, you can create stages named Development, Test, QE, and Production.

- 12 (Optional) Select a stage and drag it to a different place in the pipeline.

For example, if your pipeline consists of stages named Development, Test, and QE, you can move the stage named Test after the stage named QE.

- 13 To save the pipeline, click **Save as Draft**.

## 14 Create tasks for every stage in the pipeline.

You can add multiple tasks to a stage. You can model tasks in a stage to run in parallel, and drag those tasks to run in parallel. Running a parallel task is limited only within a stage.

The Artifact, Custom, Deploy, Provision, and Test tasks are supported. Certain tasks might depend on tasks that precede them in the workflow. You can drag tasks up or down depending on the workflow. Tasks run either sequentially or in parallel depending on how you modeled the tasks in the stage.

You can configure the input for a task to depend on the output of an earlier task. You can add parallel tasks in a single group, and configure the input for the task to depend on the output of an earlier task. In these cases, vRealize Code Stream always considers the variables from the output of the earlier task, which belong to the previous group.

## Importing Pipelines

To import a pipeline, you can enter the pipeline data directly. Or, you can import a JSON file or a TXT file that contains the pipeline data, from your local file system to import a previously saved pipeline. The import adds the pipeline to your vRealize Code Stream instance.

This version of vRealize Code Stream supports importing pipelines from version 2.2 and greater.

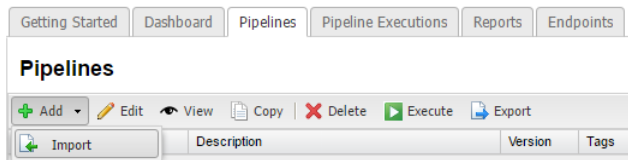
When you import a file from your local file system, vRealize Code Stream adds the pipeline to your list of pipelines.

### Prerequisites

- Verify that you have a working instance of vRealize Code Stream.
- On your local file system, verify that you have a JSON or TXT file available, which contains the modeling information for a pipeline.

### Procedure

- 1 To import a pipeline JSON file, click **Pipelines > Add > Import**.



- 2 In the Import Pipeline dialog box, enter the pipeline data or optionally select a pipeline JSON file.

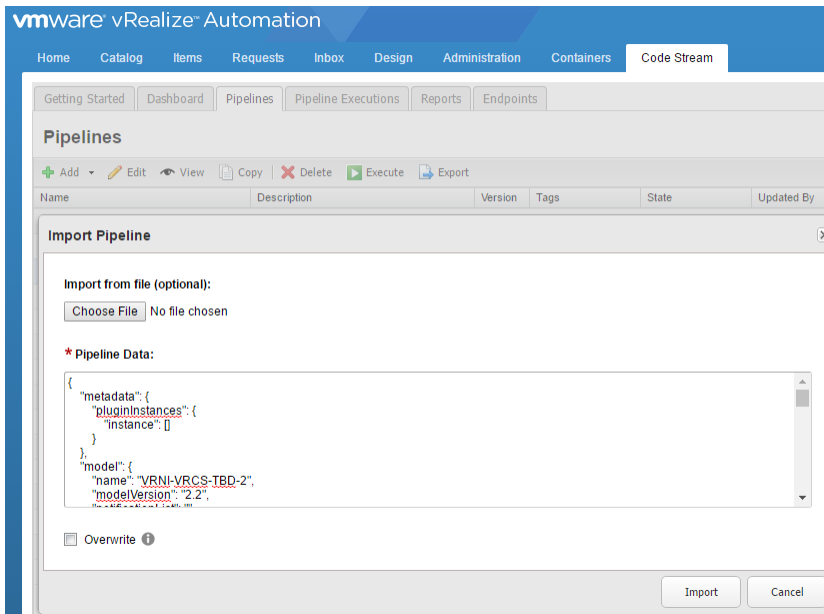
- a In the Pipeline Data text box, enter the pipeline data.

You can copy and paste the pipeline data directly into the text area. The text area allows you to edit the pipeline data easily. For example, you can change the name of a pipeline, task, or stage, or change other information about the pipeline.

- b (Optional) Click **Choose File**, and select a JSON file that you saved on your local file system.

When you select a file, the content of the file appears in the text box.

- c To allow the import to overwrite an existing pipeline that has the same name that you entered, select **Overwrite**.
- d Click **Import**, and wait for the import to complete.



You imported a pipeline to your vRealize Code Stream instance.

When the import succeeds, a message appears at the top of the Pipelines tab. The imported pipeline appears as the first row in the grid of the existing pipelines in the instance.

### What to do next

Activate and run the imported pipeline. For the pipelines to run, you must have endpoints defined for the task providers.

## Exporting Pipelines

You can export an existing pipeline from your vRealize Code Stream instance to a JSON file. The exported pipeline file on your local file system uses the same name as the pipeline name. You can share the JSON file with other users so that they can import the same pipeline into their vRealize Code Stream instance.

This version of vRealize Code Stream supports exporting pipelines from version 2.2 and greater.

You can export a pipeline an unlimited number of times. The export uses the latest version of the pipeline.

The export stores the pipeline information in a JSON file that you can save to use later, or to share with other users across multiple instances of vRealize Code Stream. If a JSON file exists and has the same name as the pipeline that you export, the browser determines how the file is saved. Depending on the browser that you use, it either saves the pipeline with a new name, or prompts you to save it in the default download location.

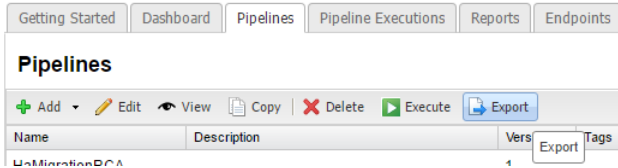
## Prerequisites

- You have one or more pipelines modeled in your instance of vRealize Code Stream.

## Procedure

- 1 To export a pipeline to a JSON file, click **Pipelines**.
- 2 Select a pipeline, and click **Export**.

The export saves the pipeline information to a JSON file that has the same name as the pipeline. For example, vRCSE2E.json.



- 3 To verify that the export succeeded, view the JSON file on your local file system.

You exported an existing pipeline to a JSON file.

## What to do next

Share the exported pipeline file with other users.

# Assign Role-Based Access Control Permissions at the Pipeline Level

When you create or edit a pipeline, you can give permissions to users and groups so that they can modify and trigger the pipeline. The access permissions apply only to the users and groups that you add to the permissions lists.

By default, all Release Managers can create a pipeline. When you add a group to the list of Release Managers or Release Engineers, all users in that group can modify or trigger the pipeline. When you do not add any users or groups, the following permissions are available by default:

- All vRealize Code Stream Release Managers can modify and trigger the pipeline.
- All vRealize Code Stream Release Engineers can trigger the pipeline.
- A user who has the Release Manager or Release Engineer role and Tenant Administrator role has implicit access to modify and trigger pipeline templates even when not explicitly added to the permissions list.

## Procedure

- 1 In vRealize Automation, click **Code Stream > Pipelines**.
- 2 To add or edit a pipeline so that you can apply permissions to it, either click **Add**, or select a pipeline and click **Edit** to modify it.

- 3 To add permissions to the pipeline, enter the names of users or groups in the Permissions area.
  - a In the Release Managers area, enter users or groups who can modify and trigger a pipeline.
  - b In the Release Engineers area, enter users or groups who can trigger a pipeline.

---

**Caution** A user who creates a pipeline can lose control of that pipeline. For example, if User 1 creates a pipeline, and only adds User 2 to the Release Manager permissions list, User 1 loses control on the pipeline being created. To retain access permission on the pipeline, User 1 must add their own user ID and group to the permissions list.

---

- 4 On the taskbar, click **Administration > Users & Groups > Directory Users and Groups**.
- 5 Search for the user.
- 6 In the list of users, click the user name.
- 7 On the **User Details** page, on the **General** tab, scroll the list of roles to add the user.
  - a To give the user permissions to modify and trigger a pipeline, select the **Release Manager** check box.
  - b To give the user permissions to trigger a pipeline, select the **Release Engineer** check box.

You have assigned permissions to users and groups to modify and trigger a pipeline.

#### What to do next

Create other tasks for the pipeline. See [Modeling a Release Pipeline Checklist](#).

## Activate and Run a Release Pipeline

After you create a release pipeline, you must activate it to complete the pipeline modeling. After you activate the pipeline, you can run it. A pipeline that is in draft state indicates that the Release Manager is still modeling the pipeline.

To view a release pipeline as it runs, click the **Pipeline Executions** tab.

The release pipeline run provides multiple levels of information.

- Level 1. Detailed status at the release pipeline level describes the current activity.
- Level 2. Status at the task or stage level displays the current task progress and the status of the task that ran.
- Level 3. Detailed information at the task level.

If the pipeline execution fails, you can resume execution. For example, a pipeline run might fail if artifacts do not exist for a task in the pipeline. For failed or canceled pipeline runs, you can retrigger the pipeline execution run again. For failed pipeline runs, you can also resume the pipeline execution from the last failed task or tasks in a stage.

To observe how to activate and run a release pipeline in vRealize Code Stream, watch the following video.



See Executing and Tracking Pipelines in vRealize Code Stream at <http://www.youtube.com/watch?v=2PPJlhxB3Lc&list=PLrFo2o1FG9n7Pr4Fq-7exPvN6ThDYRcPV&index=2>.

### Prerequisites

- Verify that a release pipeline is created in a draft state. See [Create a Release Pipeline](#).
- Verify that predefined vRealize Automation blueprints, workflows, and scripts are created, and that test jobs that perform tasks that the pipeline can trigger are available.
- Verify that artifacts are available in the Artifactory server repository. See [Search Artifacts from the Artifactory Repository](#).

### Procedure

- 1 Click the **Code Stream** tab.
- 2 On the **Pipelines** tab, select an existing pipeline, and click **Edit** to configure it.
- 3 Click **Activate** to create a vRealize Orchestrator workflow.  
A list of available pipelines appears.
- 4 Select the pipeline, and click **Execute**.  
The Execute Pipeline dialog box appears.
- 5 Enter the description and properties, and click **OK** to run the pipeline.

---

**Note** If you are running a pipeline with Team Foundation Server 2015 (TFS), you can cancel the pipeline execution and view the status of the configured build in the server. If you are running a pipeline with TFS 2013 and cancel the pipeline execution, the pipeline is marked as canceled. Although the pipeline is marked as canceled, the configured build in the TFS server still displays the status of the pipeline as completed.

---

- 6 If a pipeline execution fails, you can resume execution.
  - a Select the failed pipeline execution.
  - b Expand the arrow next to the pipeline name, and review the stages and tasks to determine which task failed.  
  
Tasks that failed display a red line at the top of the task in the stage, and the top of the stage displays a red line.
  - c Click the task that failed, and read the status message to understand the failure.
  - d On the **Result And Input** tab, review the information about the artifacts.

- e If needed, click the **Raw Output Properties** tab, and review the values of the output parameters. Then, click the **Raw Input Properties** tab, and from the values of external system verify whether the external system is available. Then, click **Close**.

An external system that is not available might cause the transient failure.

- f Click **Resume Execution**.

The pipeline execution runs again.

You have activated and run a pipeline, and resumed any failed execution that occurred.

## Configure a Bamboo Task

When you register a Bamboo Server endpoint, you connect to a Bamboo server to start a build plan from a release pipeline. You can configure multiple artifacts from a particular plan.

The location of the artifacts produced by a Bamboo build can be exposed as an output parameter to allow other tasks to execute tests on the build and then deploy it.

### Prerequisites

- Verify that the Bamboo server endpoint is registered. See the *Installation and Configuration* guide.
- Verify that the Bamboo server version is 5.9.7 or later.

### Procedure

- 1 Click the **Code Stream** tab.
- 2 On the **Pipelines** tab, select an existing pipeline, and click **Edit** to configure it.
- 3 Click the **Stages >** button.
- 4 Click **Add Task**.
- 5 Select **Bamboo** from the **Provider** drop-down menu.
- 6 Enter a name for the task and click **OK**.
- 7 Select the new task in the stage.
- 8 Confirm to save the pipeline.

## 9 Select when to run the task in the release pipeline.

Option	Description
<b>Always</b>	Runs the release pipeline task without conditions.
<b>On Condition(s)</b>	<p>Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped.</p> <p>A boolean expression, which uses the following operands and operators, is supported.</p> <ul style="list-style-type: none"> <li>■ Pipeline variables such as, <code>\${pipeline.variableName}</code>. Curly brackets are reserved for entering pipeline variables.</li> <li>■ Task output variables such as, <code>\${Stage1.task1.machines[0].value.hostIp[0]}</code></li> <li>■ Default pipeline binding variables such as, <code>\${releasePipelineName}</code></li> <li>■ Case insensitive Boolean values such as, true, false, 'true', 'false'</li> <li>■ Integer or decimal values without quotes</li> <li>■ String values used with single or double quotes such as, "test", 'test'</li> <li>■ String and Numeric types of values such as, == Equals and != Not Equals</li> <li>■ Relational operators such as, &gt;, &gt;=, &lt;, and &lt;=</li> <li>■ Boolean logic such as, &amp;&amp; and   </li> <li>■ Arithmetic operators such as, +, -, *, and /</li> <li>■ Nested expressions using round brackets</li> <li>■ Strings with literal value ABCD are evaluated as false and the task is skipped.</li> </ul> <p>Unary operators are not supported.</p> <p>A sample condition, <code>\${Stage1.task1.output} == "Passed"    \${pipeline.variableName} == 39</code></p>

## 10 (Optional) Select the **Continue Pipeline execution even if this task fails** option.

This option allows the release pipeline to bypass the failed task and continue with the execution process.

## 11 Select the endpoint, project, and the Bamboo plan.

## 12 Enter the Bamboo plan input properties.

## 13 Click **Save**.

## Example: Output Format for a Bamboo Task

The following task output format is an example for a Bamboo task.

```
[
  {
    "name": "planKey",
    "type": "String",
    "value": "SAM-S2P1"
  },
  {
    "name": "resultUrl",
    "type": "String",
    "value": "http://xx.xxx.xx.xx:xxxx/browse/SAM-S2P1-101"
  },
  {

```



```

    "name": "buildNumber",
    "type": "String",
    "value": 101
  },
  {
    "name": "buildTestSummary",
    "type": "String",
    "value": "4 passed"
  },
  {
    "name": "triggerReason",
    "type": "String",
    "value": "Manual run by <USER>"
  },
  {
    "name": "successfulTestCount",
    "type": "String",
    "value": 0
  },
  {
    "name": "failedTestCount",
    "type": "String",
    "value": 0
  },
  {
    "name": "skippedTestCount",
    "type": "String",
    "value": 0
  }
}
]

```

## Configure a Bugzilla Task

You can execute a pipeline task which creates a Bugzilla task. You can use the output properties of the Bugzilla task in subsequent tasks in a release pipeline.

You can export and import a pipeline model containing a Bugzilla task.

### Prerequisites

- Verify that the Bugzilla server endpoint is registered. See the *Installation and Configuration* guide.
- Verify that the Bugzilla server version is 5.0.1 or later.

### Procedure

- 1 Click the **Code Stream** tab.
- 2 On the **Pipelines** tab, select an existing pipeline, and click **Edit** to configure it.
- 3 Click the **Stages >** button.
- 4 Click **Add Task**.
- 5 Select **Bugzilla** from the **Provider** drop-down menu.

- 6 Enter a name for the task and click **OK**.
- 7 Select the new task in the stage.
- 8 Confirm to save the pipeline.
- 9 Select when to run the task in the release pipeline.

Option	Description
<b>Always</b>	Runs the release pipeline task without conditions.
<b>On Condition(s)</b>	<p>Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped.</p> <p>A boolean expression, which uses the following operands and operators, is supported.</p> <ul style="list-style-type: none"> <li>■ Pipeline variables such as, <code>\${pipeline.variableName}</code>. Curly brackets are reserved for entering pipeline variables.</li> <li>■ Task output variables such as, <code>\${Stage1.task1.machines[0].value.hostIp[0]}</code></li> <li>■ Default pipeline binding variables such as, <code>\${releasePipelineName}</code></li> <li>■ Case insensitive Boolean values such as, true, false, 'true', 'false'</li> <li>■ Integer or decimal values without quotes</li> <li>■ String values used with single or double quotes such as, "test", 'test'</li> <li>■ String and Numeric types of values such as, == Equals and != Not Equals</li> <li>■ Relational operators such as, &gt;, &gt;=, &lt;, and &lt;=</li> <li>■ Boolean logic such as, &amp;&amp; and   </li> <li>■ Arithmetic operators such as, +, -, *, and /</li> <li>■ Nested expressions using round brackets</li> <li>■ Strings with literal value ABCD are evaluated as false and the task is skipped.</li> </ul> <p>Unary operators are not supported.</p> <p>A sample condition, <code>\${Stage1.task1.output} == "Passed"    \${pipeline.variableName} == 39</code></p>

- 10 (Optional) Select the **Continue Pipeline execution even if this task fails** option.

This option allows the release pipeline to bypass the failed task and continue with the execution process.

- 11 Select the Bugzilla endpoint from the **Endpoint** drop-down menu.

- 12 Select the action you want to perform.

You can create or update a bug.

## Example: Output Format for a Bugzilla Task

The following task output format is an example for a Bugzilla task.

```
[
  {
    "name": "id",
    "type": "INTEGER",
    "description": "Bug ID",
    "value": "153"
```

```

    },
    {
      "name": "bugURL",
      "type": "STRING",
      "description": "BugUrl",
      "value": "http://bugzilla.example.com/show_bug.cgi?id=153"
    },
    {
      "name": "status",
      "type": "STRING",
      "description": "Bug Status",
      "value": "RESOLVED"
    },
    {
      "name": "product",
      "type": "STRING",
      "description": "Product the bug is filed under",
      "value": "vRealize Code Stream"
    },
    {
      "name": "assigned_to",
      "type": "String",
      "description": "Bug Assignee",
      "value": "user@example.com"
    }
  ]

```

## Create Bug

You can execute a release pipeline model containing a Bugzilla task for creating a bug.

### Procedure

- 1 Click **Create Bug** and select the product from the **Product** drop-down menu to create a bug.
- 2 Select the component from the **Component** drop-down menu.
- 3 Enter the **Summary**.
  - a Click the text box and enter `${` .  
The associated variables appear in the drop-down menu.
  - b Select the variable.
  - c (Optional) Enter a period next to the variable to view the task name in the drop-down menu.
  - d (Optional) Enter a period next to the task name to view the task output or pipeline parameters in the drop-down menu.
- 4 Select the severity, version and enter the description.
- 5 Select the optional fields that you want to add to the bug from the **Optional** tab.
- 6 Click **Save**.

## Update Bug

You can edit a bug by providing the Bug ID.

### Procedure

- 1 Click **Update Bug** and enter the **Bug ID** to update a bug.
- 2 Update the assignee if you want to change the assignee and enter your comments.
- 3 Selecting the resolution from the **Resolution** drop-down menu to resolve the bug.
- 4 Select the **Verified** option to verify the bug.

## Configure a Cloud Foundry Task

Use the Cloud Foundry task to model a pipeline task that connects to a Cloud Foundry instance. The task can deploy, scale, start, stop, and delete an application. You can also bind and unbind services to an application.

### Prerequisites

- Verify that the Cloud Foundry Server endpoint is registered. See the *Installation and Configuration Guide*.

### Procedure

- 1 Click the **Code Stream** tab.
- 2 On the **Pipelines** tab, select an existing pipeline, and click **Edit** to configure it.
- 3 Click the **Stages >** button.
- 4 Click **Add Task**.
- 5 From the **Provider** drop-down menu, select **Cloud Foundry**.
- 6 Enter a name for the task, and click **OK**.
- 7 Select the new task in the stage.
- 8 Confirm to save the pipeline.

## 9 Select when to run the task in the release pipeline.

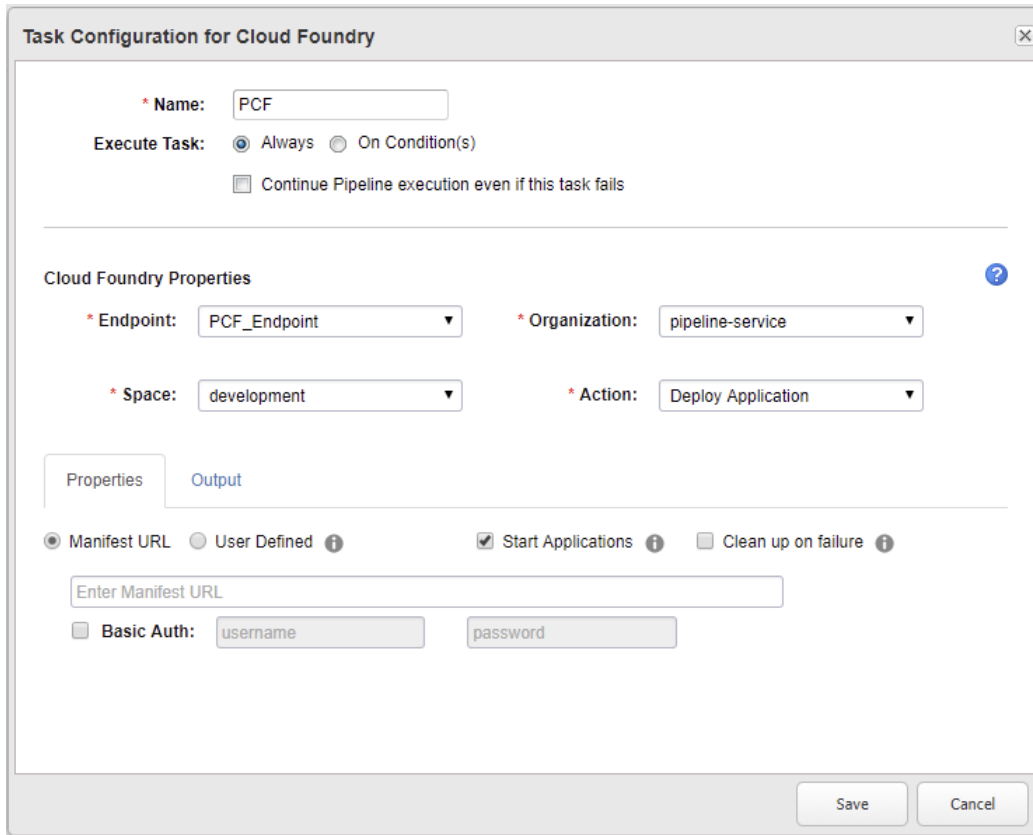
Option	Description
<b>Always</b>	Runs the release pipeline task without conditions.
<b>On Condition(s)</b>	<p>Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped.</p> <p>A boolean expression, which uses the following operands and operators, is supported.</p> <ul style="list-style-type: none"> <li>■ Pipeline variables such as, <code>\${pipeline.variableName}</code>. Curly brackets are reserved for entering pipeline variables.</li> <li>■ Task output variables such as, <code>\${Stage1.task1.machines[0].value.hostIp[0]}</code></li> <li>■ Default pipeline binding variables such as, <code>\${releasePipelineName}</code></li> <li>■ Case insensitive Boolean values such as, true, false, 'true', 'false'</li> <li>■ Integer or decimal values without quotes</li> <li>■ String values used with single or double quotes such as, "test", 'test'</li> <li>■ String and Numeric types of values such as, == Equals and != Not Equals</li> <li>■ Relational operators such as, &gt;, &gt;=, &lt;, and &lt;=</li> <li>■ Boolean logic such as, &amp;&amp; and   </li> <li>■ Arithmetic operators such as, +, -, *, and /</li> <li>■ Nested expressions using round brackets</li> <li>■ Strings with literal value ABCD are evaluated as false and the task is skipped.</li> </ul> <p>Unary operators are not supported.</p> <p>A sample condition, <code>\${Stage1.task1.output} == "Passed"    \${pipeline.variableName} == 39</code></p>

## 10 (Optional) Select the **Continue Pipeline execution even if this task fails** option.

This option allows the release pipeline to bypass the failed task and continue with the execution process.

## 11 Enter the Cloud Foundry properties.

Option	Description
<b>Endpoint</b>	Provides a list of registered endpoints for Cloud Foundry.
<b>Organization</b>	Lists the existing organizations.
<b>Space</b>	Lists the spaces in the selected organization.
<b>Action</b>	Lists the tasks such as deploy, scale, start, stop, delete an application in a Cloud Foundry instance.



**Task Configuration for Cloud Foundry**

\* Name:

Execute Task: ☒ Always ☐ On Condition(s)

☐ Continue Pipeline execution even if this task fails

---

**Cloud Foundry Properties** ?

\* Endpoint:  \* Organization:

\* Space:  \* Action:

Properties Output

☒ Manifest URL ☐ User Defined ⓘ

☒ Start Applications ⓘ ☐ Clean up on failure ⓘ

☐ Basic Auth:

Save Cancel

- a Select **Bind/Unbind Services > Properties**.

Option	Description
<b>Read from Pipeline/Task property</b>	Enter the name of the application or provide variable binding to read from the pipeline or task property.
<b>Existing Applications</b>	Select an existing application.

You can view the name, URL, and the status of the application you have selected from the **Output** tab.

- b Select **Delete an Application > Properties**.

Option	Description
<b>Read from Pipeline/Task property</b>	Enter the name of the application.
<b>Existing Applications</b>	Select the name of an existing application.
<b>Delete Route</b>	Delete the path or route associated with the application. If the route is not linked to any other application, it is deleted.

You can view the name, URL, and status of the application that you selected from the **Output** tab.

c Select **Deploy Application > Properties**.

Option	Description
<b>Manifest URL</b>	<p>To authenticate the repository that contains the manifest file, enter the manifest file URL, and click the <b>Basic Auth</b> link.</p> <p><b>Note</b> Authenticate the URL if you download the artifact from hosting servers such as an Artifactory server.</p>
<b>User Defined</b>	<p>Enter the application name, click <b>Add</b>, and enter the configurable properties such as disk and memory information, and the number of instances of the application.</p> <p>Enter the Artifact URL, which is the path where the application is located in the repository. For example: <i>http://artifactoryserver.com/artifactory/deb-local/SampleArtifactory.war</i>.</p> <p>To authenticate the repository that contains the downloadable artifact, click the <b>Basic Auth</b> link.</p> <p>You can also deploy an application by using advanced options where you:</p> <ul style="list-style-type: none"> <li>■ Provide the domain name.</li> <li>■ Select services.</li> <li>■ Select the stack.</li> <li>■ Enter a command. For example: <i>bundle exec rake VERBOSE=true</i>.</li> <li>■ Enter the host URL. For example: <i>http://sample-host.domain</i>.</li> <li>■ Enter the buildpack URL. For example: <i>https://github.com/cloudfoundry/java-buildpack.git</i>.</li> </ul>
<b>Start Applications</b>	To start the applications on the Cloud Foundry instance after deployment, select the check box.
<b>Clean up on failure</b>	To delete the application from the Cloud Foundry instance when the application deployment fails, select the check box.

You can view the name, URL, and status of the application that you selected from the **Output** tab.

d Select **Scale an Application > Properties**.

Option	Description
<b>Read from Pipeline/Task property</b>	Enter the name of the application. Or, to read from the pipeline or task property, provide variable binding. Enter the memory and disk size. Enter the number of instances of the application.
<b>Existing Applications</b>	Select the name of an existing application, enter the disk and memory information, and number of instances of the application.

**Note** You can view information about the memory, disk, and the number of instances from a pipeline or task property.

You can view the name, URL, and status of the application that you selected from the **Output** tab.

e Select **Start an Application > Properties**.

Option	Description
<b>Read from Pipeline/Task property</b>	Enter the name of the application, or provide variable binding to read from the pipeline or task property.
<b>Existing Applications</b>	Select the name of an existing application.

You can view the name, URL, and the status of the application that you selected from the **Output** tab.

To stop an application, and select to read from the pipeline/task property or select an existing application, select **Stop an Application**.

12 Click **Save**.**Example: Task Output Format for a Cloud Foundry Instance**

The following output format is an example for a Cloud Foundry task that is used to scale, bind, unbind, stop, start, and delete an application. The output contains the status of one application.

```
[
  {
    "name": "detailedStatus",
    "type": "JSON",
    "value": {
      "sample-iot-app": {
        "status": "SUCCESS",
        "displayStatus": "Success",
        "appName": "sample-iot-app",
        "appUrl": "sample-iot-app.cf.vca10.pivotal.io",
      }
    }
  }
]
```



The following output format is an example for a Cloud Foundry task that is used to deploy an application. The output contains the status of more than one application.

```
[
  {
    "name": "detailedStatus",
    "type": "JSON",
    "value": {
      "sample-iot-app": {
        "status": "SUCCESS",
        "displayStatus": "Success",
        "appName": "sample-iot-app",
        "appUrl": "sample-iot-app.cf.vca10.pivotal.io"
      },
      "sample-schedule-app": {
        "status": " SUCCESS ",
        "displayStatus": " Success ",
        "appName": "sample-schedule-app",
        "appUrl": " sample-schedule-app.cf.vca10.pivotal.io"
      }
    }
  }
]
```

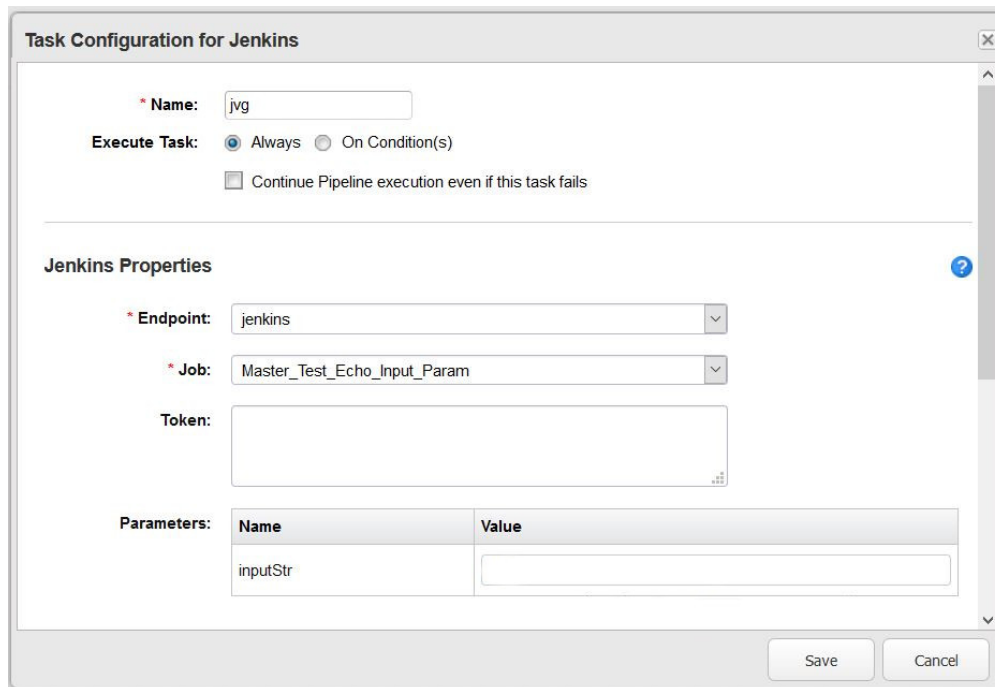
## Configure a Jenkins Task

You can use a test task to test a deployment. When you register a Jenkins Server endpoint, vRealize Code Stream connects to a Jenkins server and starts a build job from a release pipeline.

If you configure a Jenkins test job to fail when test failures occur, the release pipeline also fails.

Be aware that if you have existing artifacts in the Jenkins server you cannot access them from vRealize Code Stream.

When you configure a Jenkins server endpoint, the configuration allows you to enter the path to the folder that groups your Jenkins jobs. You can optionally add a token parameter in case it is configured in the Jenkins job.



**Task Configuration for Jenkins**

\* **Name:**

**Execute Task:** ☒ Always ☐ On Condition(s)

☐ Continue Pipeline execution even if this task fails

---

**Jenkins Properties** ?

\* **Endpoint:**

\* **Job:**

**Token:**

**Parameters:**

Name	Value
inputStr	<input type="text"/>

Save Cancel

### Prerequisites

- Verify that the Jenkins server endpoint is registered. See the *Installation and Configuration* guide.
- Verify that the Jenkins server version is 1.561 or later.
- Verify that the Jenkins jobs are created in the Jenkins server with the input string parameter, `vRCSTestExecutionId`.

### Procedure

- 1 Click the **Code Stream** tab.
- 2 On the **Pipelines** tab, select an existing pipeline, and click **Edit** to configure it.
- 3 Click the **Stages >** button.
- 4 Click **Add Task**.
- 5 Select **Jenkins** from the **Provider** drop-down menu.
- 6 Enter a name for the test task and click **OK**.
- 7 Select the new task in the stage.
- 8 Confirm to save the pipeline.

## 9 Select when to run the task in the release pipeline.

Option	Description
<b>Always</b>	Runs the release pipeline task without conditions.
<b>On Condition(s)</b>	<p>Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped.</p> <p>A boolean expression, which uses the following operands and operators, is supported.</p> <ul style="list-style-type: none"> <li>■ Pipeline variables such as, <code>\${pipeline.variableName}</code>. Curly brackets are reserved for entering pipeline variables.</li> <li>■ Task output variables such as, <code>\${Stage1.task1.machines[0].value.hostIp[0]}</code></li> <li>■ Default pipeline binding variables such as, <code>\${releasePipelineName}</code></li> <li>■ Case insensitive Boolean values such as, true, false, 'true', 'false'</li> <li>■ Integer or decimal values without quotes</li> <li>■ String values used with single or double quotes such as, "test", 'test'</li> <li>■ String and Numeric types of values such as, == Equals and != Not Equals</li> <li>■ Relational operators such as, &gt;, &gt;=, &lt;, and &lt;=</li> <li>■ Boolean logic such as, &amp;&amp; and   </li> <li>■ Arithmetic operators such as, +, -, *, and /</li> <li>■ Nested expressions using round brackets</li> <li>■ Strings with literal value ABCD are evaluated as false and the task is skipped.</li> </ul> <p>Unary operators are not supported.</p> <p>A sample condition, <code>\${Stage1.task1.output} == "Passed"    \${pipeline.variableName} == 39</code></p>

## 10 (Optional) Select the **Continue Pipeline execution even if this task fails** option.

This option allows the release pipeline to bypass the failed task and continue with the execution process.

## 11 Select the Jenkins job from the **Jobs** drop-down menu.

## 12 Select the Jenkins server endpoint you registered from the drop-down menu.

## 13 Enter the Jenkins job input properties.

- a Click the text box and enter `${`.

The associated variables appear in the drop-down menu.

- b Select the variable.

- c (Optional) Enter a period next to the variable to view the task name in the drop-down menu.

- d (Optional) Enter a period next to the task name to view the task output or pipeline parameters in the drop-down menu.

For example, the input property variable can be `${StageName.test.JobName}`.

## 14 Click **Save**.

## Example: Output Format for a Jenkins Task

The following task output format is an example for a test task.

```
[
  {
    "name": "buildId",
    "type": "STRING",
    "description": "Build Id",
    "value": "4"
  },
  {
    "name": "jobUrl",
    "type": "STRING",
    "description": "Job Url",
    "value": "http://198.51.100.13:8080/job/dummy/4"
  },
  {
    "name": "estimatedDuration",
    "type": "NUMBER",
    "description": "Estimation time to complete the build",
    "value": "2332343"
  },
  {
    "name": "jobName",
    "type": "STRING",
    "description": "Job Name",
    "value": "dummy"
  },
  {
    "name": "testResult",
    "type": "JSON",
    "description": "Job Name",
    "value": {
      "totalCount": 40,
      "skipCount": 0,
      "failureCount": 0,
      "successCount": 40
    }
  }
]
```

## Configure a JIRA Task

You can create and update JIRA tasks and issues in a release pipeline.

When you register a JIRA server endpoint, you connect vRealize Code Stream to a JIRA server so that the pipeline can use the JIRA server to create and update JIRA tasks and issues.

### Prerequisites

- Verify that the JIRA server endpoint is registered. See the *Installation and Configuration* guide.

- Verify that the JIRA server version is 6.3 or later.

### Procedure

- 1 Click the **Code Stream** tab.
- 2 On the **Pipelines** tab, select an existing pipeline, and click **Edit** to configure it.
- 3 Click the **Stages >** button.
- 4 Click **Add Task**.
- 5 Select **JIRA** from the **Provider** drop-down menu.
- 6 Enter a name for the task and click **OK**.
- 7 Select the new task in the stage.
- 8 Confirm to save the pipeline.
- 9 Click the task you created.
- 10 Select when to run the task in the release pipeline.

Option	Description
<b>Always</b>	Runs the release pipeline task without conditions.
<b>On Condition(s)</b>	<p>Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped.</p> <p>A boolean expression, which uses the following operands and operators, is supported.</p> <ul style="list-style-type: none"> <li>■ Pipeline variables such as, <code>\${pipeline.variableName}</code>. Curly brackets are reserved for entering pipeline variables.</li> <li>■ Task output variables such as, <code>\${Stage1.task1.machines[0].value.hostIp[0]}</code></li> <li>■ Default pipeline binding variables such as, <code>\${releasePipelineName}</code></li> <li>■ Case insensitive Boolean values such as, <code>true</code>, <code>false</code>, <code>'true'</code>, <code>'false'</code></li> <li>■ Integer or decimal values without quotes</li> <li>■ String values used with single or double quotes such as, <code>"test"</code>, <code>'test'</code></li> <li>■ String and Numeric types of values such as, <code>==</code> Equals and <code>!=</code> Not Equals</li> <li>■ Relational operators such as, <code>&gt;</code>, <code>&gt;=</code>, <code>&lt;</code>, and <code>&lt;=</code></li> <li>■ Boolean logic such as, <code>&amp;&amp;</code> and <code>  </code></li> <li>■ Arithmetic operators such as, <code>+</code>, <code>-</code>, <code>*</code>, and <code>/</code></li> <li>■ Nested expressions using round brackets</li> <li>■ Strings with literal value ABCD are evaluated as false and the task is skipped.</li> </ul> <p>Unary operators are not supported.</p> <p>A sample condition, <code>\${Stage1.task1.output} == "Passed"    \${pipeline.variableName} == 39</code></p>

- 11 (Optional) Select the **Continue Pipeline execution even if this task fails** option.

This option allows the release pipeline to bypass the failed task and continue with the execution process.

- 12 Select the JIRA endpoint from the **Endpoint** drop-down menu.

**13** Select the action you want to perform.

Option	Description
<b>Create Issue</b>	To create a JIRA issue, run a release pipeline model that contains a JIRA task.
<b>Update Issue</b>	Edit the issue by providing the Issue ID.

**14** Select the project and issue type.

Based on the project and issue type that you select, the input properties in the required and optional tabs vary.

**15** Enter the description, summary, and the reporter name.**16** Click **Save**.**17** Click **Execute** to run the release pipeline.**18** Click the executed pipeline.

Task Result displays the Issue number, Assignee, Issue status, Project, and URL. The URL redirects you to the issue in detail.

**19** Enter the JIRA issue number to update the existing issue.

You can also select assignee, transition, resolution (if configured), and enter your comments.

**20** Click **Save**.**Example: Output Format for a JIRA Task**

The following task output format is an example for a test task.

```
[
  {
    "name": "issueNumber",
    "type": "String",
    "description": "Issue Number",
    "value": "VSD-1"
  },
  {
    "name": "assignee",
    "type": "String",
    "description": "Assignee",
    "value": "jiraadmin"
  },
  {
    "name": "issueStatus",
    "type": "String",
    "description": "Issue Status",
    "value": "Resolved"
  },
  {
    "name": "project",
    "type": "String",
```

```

    "description": "Project",
    "value": "Project-1"
  }
]

```

## Configure a Poll Task

You can use the vRealize Code Stream Poll Plug-in to listen for the status on a specific API to detect when a particular task completes. You configure the interval for the Poll plug-in so that it continues to poll the API until the task meets the exit criteria that you define.

With the Poll plug-in, you can include an exit criteria for the poll task to stop the polling, and mark the task as completed or failed. The exit criteria can be a success criteria that is mandatory, and an optional failure criteria. The criteria can refer to the JSON response of the API call, and bind to a property value in the JSON response.

You can add and configure a Poll task anywhere in your pipeline. With a Poll task, you can have any other task in the pipeline wait until the status changes on a REST API.

In the task, you enter the URL of any REST endpoint, such as the server that hosts the appliance. In the URL, you include a variable that reflects the output property from the previous task. To verify that the previous task completes, the output must match the criteria that you include in the Poll plug-in task configuration.

You can enter a complex JSON body to search for a specific value. When the value matches the exit condition in the Poll task configuration, the Poll task indicates success or failure. You can include a single exit condition in the Poll task, which must be an On-Success value condition.

### Prerequisites

- You modeled a pipeline, included stages, and added a task to a stage. See [Modeling a Release Pipeline Checklist](#).

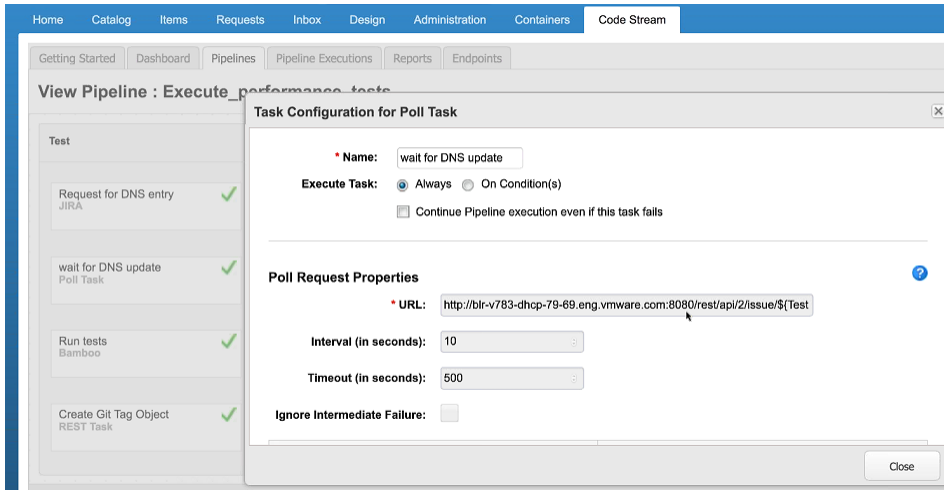
### Procedure

- 1 In your modeled pipeline, click **Add Task** to add a task to a stage.
- 2 Add a Poll task.
  - a In the Add Task dialog box, click the **Provider** drop-down menu.
  - b Click **Poll Task**.
  - c Enter a name for the task, and click **OK**.

### 3 Configure the Poll task.

- a In the Poll Task, click the gear icon, and click **Configure**.
- b In the Task Configuration for Poll Task, enter the URL of any REST endpoint, and include the variable that reflects the output of the previous task.

You can have the Poll task listen for the status of a JIRA task, and ensure that the Poll task reflects the JIRA issue number. For example, you might enter: `http://<URL>:8080/rest/api/2/issue/${Stage0.Jira_Test_Job.issueNumber}`. A similar example appears as follows:



- c In the Interval text box, enter the number of seconds for the Poll task to poll.  
Enter a desired number. A common poll value is 10 seconds, which is the default value.
- d In the Timeout text box, enter the number of seconds for the Poll task to wait before the task times out.  
Enter a desired number. An adequate timeout might be 500 seconds. The default is 5 seconds.
- e To continue to poll if the response indicates an error, such as a 404 when the resource is not available, select **Ignore Immediate Failure**.
- f To provide authorization for the task, click **Add Headers** and enter a header key and value.



- g In the section named Add exit criteria, enter the JSON key path.

The screenshot shows the VMware vRealize Automation interface with the 'Code Stream' tab selected. A 'View Pipeline' dialog is open, displaying the 'Task Configuration for Poll Task' for a task named 'Wait for approval'.

**Task Configuration for Poll Task**

**Name:** Wait for approval

**Execute Task:** ☒ Always ☐ On Condition(s)

☐ Continue Pipeline execution even if this task fails

**Poll Request Properties**

**URL:** http://10.110.14.186:32774/restapi/2/issue/\${Staging.Raise Ticket.Issu

**Interval (in seconds):** 6

**Timeout (in seconds):** 3600000

☐ Ignore Intermediate Failure:

Header Key	Header Value
Content-Type	application/json
Accept	application/json
Authorization	Basic YWRtaW46Vk13YXJIMSE=

**Exit Criteria**

JSON key path	On-Success value	On-Failure value
fields.resolution.name	Done	empty

**Output Properties:**

Close

< Previous Stages > Close

- h At the end of the JSON key path, include the value for the On-Success or On-Failure for the Poll task to detect the matching status.

For example, in your pipeline, you might have a task to create a JIRA issue. You also have a Poll task to cause the pipeline to wait until the ticket is resolved. The Poll task polls the JSON output from the API, which includes the key path. After the JIRA issue is resolved, the pipeline run succeeds. The JSON output includes the key path components, such as fields, status and name. The JSON code for fields, status and name resembles:

```
{
  "expand":
  "renderedFields,names,schema,operations,editmeta,changelog,versionedRepresentations",
  "id": "19982",
  "self": "http://10.110.205.137:8080/rest/api/2/issue/19982",
  "key": "VSD-8479",
  "fields": {
    "issuetype": {
  ...
    "status": {
      "self": "http://10.110.205.137:8080/rest/api/2/status/10000",
      "description": "",
      "iconUrl": "http://10.110.205.137:8080/",
      "name": "Done",
      "id": "10000",
      "statusCategory": {
  ...
```

- i Click **Save**.
- 4 Activate and run the pipeline, and view the task execution details.
    - a Click **Pipeline Executions**, and click the Poll task in the pipeline stage.
    - b In the Task Result dialog box, click the **Result** drop-down arrow.
    - c Verify that the details display the response code, response body, and the response header keys and values.
 

If the response data size exceeds 5 MB, the REST task might fail.
    - d Click the **Input** drop-down arrow.
    - e View the input details, which include the name and value pairs, the header key and value pairs, JSON key path, and success criteria.

You configured a Poll task to listen for the matching status to detect when a particular task completes, and viewed the task results.

## Configure a REST Task

You can use the REST Plug-In with vRealize Code Stream to invoke a REST API, which sends and receives data between external and internal applications, such as your Git repository, JIRA, and so on. The REST Plug-In enables applications that use a REST API to interact with other plug-ins to support continuous delivery of your development applications through the pipeline.

The REST Plug-In works with any REST API. It supports GET, POST, PUT, PATCH, and DELETE HTTP methods. When you add a REST task to your pipeline, the task calls a URL. For POST, PUT, and PATCH actions, you include a payload. In the payload, you can bind your pipeline and task properties when the pipeline runs.

You can add REST tasks to your pipeline to communicate information between applications, and provide status to the next task in the pipeline stage. For example, if you add a REST task to create a tag on a Git commit for a build, the task could post a request to get the check-in ID from your repository. Then, the task would send a payload to your repository to create a tag for build, and your repository would return the response with the tag.

You can use a Poll task to poll the API used in the REST task until the task results meet the exit criteria. See [Configure a Poll Task](#).

### Prerequisites

- Verify that your instance of vRealize Code Stream includes one or more pipelines with stages, in Draft state. See [Create a Release Pipeline](#).

### Procedure

- 1 In your pipeline, click **Add Task** to add a task to a stage.
- 2 Add a REST task.
  - a Click **Add Task**.
  - b In the Add Task dialog box, click the **Provider** drop-down menu.
  - c Click **REST Task**.
  - d Enter a name for the task, and click **OK**.
- 3 Configure the REST task.
  - a In the REST task, click the gear, and click **Configure**.
  - b In the REST Request Properties area, select the HTTP method.  
 To have the REST task import information from another application, you select the **POST** method. To export information to another application, you select the **GET** method.
  - c Enter the URL that identifies the REST API endpoint.

- d For a task to import information from another application, include the payload variable.

For example, for an import task, you would enter `${Stage0.export.responseBody}`. If the response data size exceeds 5 MB, the REST task might fail.

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

**Task Configuration for REST Task**

\* Name:

Execute Task: ☒ Always ☐ On Condition(s)

☐ Continue Pipeline execution even if this task fails

---

**REST Request Properties**

\* Method:

\* URL:

Payload:

Add Headers

Header Key	Header Value
Content-Type	application/json
Accept	application/json
Authorization	Bearer MTQwMDUyMzE0MTIzNDU2Nzg5MTA=

Close

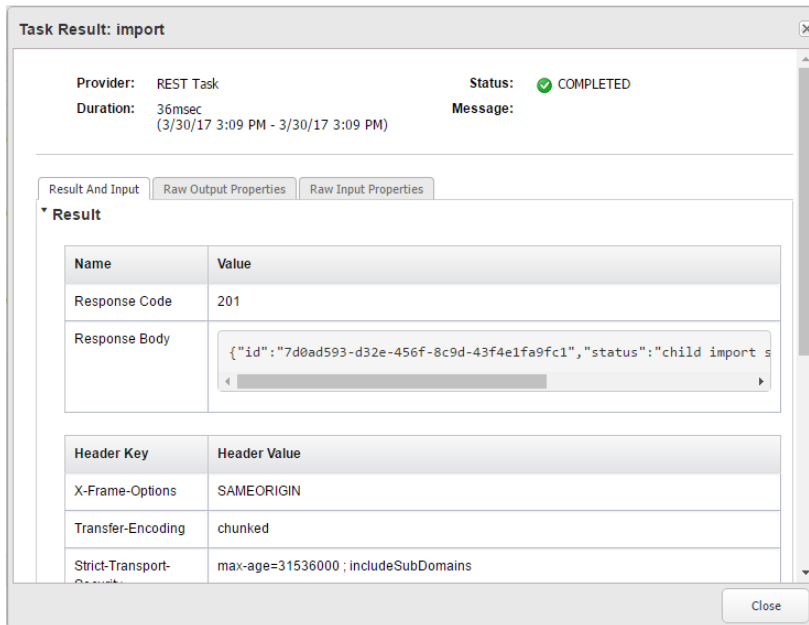
- f Click **Save**.

- 4** Activate and run the pipeline, and view the task execution details.

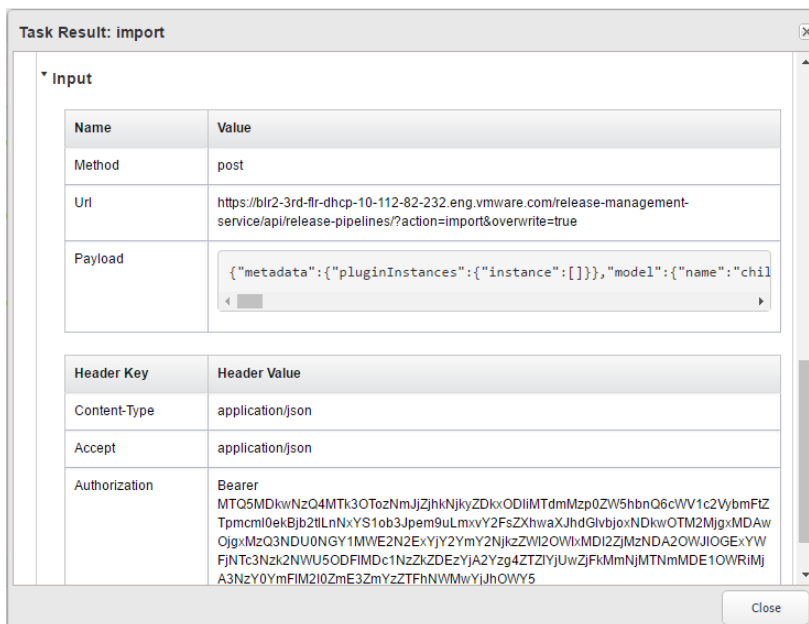
- a In your pipeline, click **Activate**, then on the toolbar click **Execute**.

- b After the pipeline runs, click **Pipeline Executions**, and click the REST task in the pipeline stage.

- c In the Task Result dialog box, click the **Result** drop-down arrow, and verify that the details display the response code, response body, and the response header keys and values.



- d Click the **Input** drop-down arrow, and view the input details, which include the name and value pairs, and the header key and value pairs.



You configured a REST task to send information between applications and plug-ins in your instance of vRealize Code Stream.

## Configure a Team Foundation Server Task

You can use the Team Foundation Server task to connect to the Team Foundation Server and invoke configured build and test jobs.

### Prerequisites

- Verify that the Team Foundation Server endpoint is registered. See the *Installation and Configuration* guide.
- Verify that the Team Foundation Server project collection, team projects, and build definitions are configured.

### Procedure

- 1 Click the **Code Stream** tab.
- 2 On the **Pipelines** tab, select an existing pipeline, and click **Edit** to configure it.
- 3 Click the **Stages >** button.
- 4 Click **Add Task**.
- 5 Select **Team Foundation Server** from the **Provider** drop-down menu.
- 6 Enter a name for the custom task and click **OK**.
- 7 Select the new task in the stage.
- 8 Confirm to save the pipeline.

## 9 Select when to run the task in the release pipeline.

Option	Description
<b>Always</b>	Runs the release pipeline task without conditions.
<b>On Condition(s)</b>	<p>Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped.</p> <p>A boolean expression, which uses the following operands and operators, is supported.</p> <ul style="list-style-type: none"> <li>■ Pipeline variables such as, <code>\${pipeline.variableName}</code>. Curly brackets are reserved for entering pipeline variables.</li> <li>■ Task output variables such as, <code>\${Stage1.task1.machines[0].value.hostIp[0]}</code></li> <li>■ Default pipeline binding variables such as, <code>\${releasePipelineName}</code></li> <li>■ Case insensitive Boolean values such as, true, false, 'true', 'false'</li> <li>■ Integer or decimal values without quotes</li> <li>■ String values used with single or double quotes such as, "test", 'test'</li> <li>■ String and Numeric types of values such as, == Equals and != Not Equals</li> <li>■ Relational operators such as, &gt;, &gt;=, &lt;, and &lt;=</li> <li>■ Boolean logic such as, &amp;&amp; and   </li> <li>■ Arithmetic operators such as, +, -, *, and /</li> <li>■ Nested expressions using round brackets</li> <li>■ Strings with literal value ABCD are evaluated as false and the task is skipped.</li> </ul> <p>Unary operators are not supported.</p> <p>A sample condition, <code>\${Stage1.task1.output} == "Passed"    \${pipeline.variableName} == 39</code></p>

## 10 (Optional) Select the **Continue Pipeline execution even if this task fails** option.

This option allows the release pipeline to bypass the failed task and continue with the execution process.

## 11 Configure the Team Foundation Server properties.

Option	Description
<b>Endpoint</b>	Provides the registered Team Foundation Server endpoint.
<b>Project Collection</b>	Provides a list of existing Team Foundation Server project collections.
<b>Team Project</b>	Lists the existing team projects on the Team Foundation Server.
<b>Build Priority</b>	Defines the level of priority on when the build runs.
<b>Build Definition</b>	<p>Provides a search capability for build definitions with automatic complete.</p> <p>When you select the build definition the associated arguments and properties are populated from Team Foundation Server.</p>

## 12 Select one or more Team Foundation Server job output properties to expose to other tasks in the release pipeline.

For example, you can expose the Team Foundation Server build ID and build URL in a deploy task.

## Example: Task Output Format for a Team Foundation Server

The following task output format is an example for a Team Foundation Server task.

```
[
  {
    "name": "buildId",
    "displayName": "Build Id",
    "value": "823",
    "displayValue": null
  },
  {
    "name": "dropLocation",
    "displayName": "Drop Location",
    "value": "#/70869/drop",
    "displayValue": null
  },
  {
    "name": "logUrl",
    "displayName": "Build Summary Url",
    "value":
"http://19.126.10.35:8080/tfs/TFSTestCollection/ConsoleApplication/_build#buildUri=vstfs:///Build/Buil
d/893&_a=summary",
    "displayValue": "http://19.126.10.35:8080/tfs/TFSTestCollection/ConsoleApplication/summary"
  },
  {
    "name": "buildUrl",
    "displayName": "Build Summary Url",
    "value":
"http://19.126.10.35:8080/tfs/TFSTestCollection/ConsoleApplication/_build#buildUri=vstfs:///Build/Buil
d/893&_a=summary",
    "displayValue": "http://19.126.10.35:8080/tfs/TFSTestCollection/ConsoleApplication/summary"
  },
  {
    "name": "failedTestsCount",
    "displayName": "Number Of Failed Tests",
    "value": "3",
    "displayValue": null
  }
]
```

## Add a Custom Service Blueprint Task

You can add a Custom Service Blueprint task to add a XaaS blueprint. A service blueprint task allows you invoke a plug-in available in the vRealize Automation service catalog from a release pipeline. You can publish a predefined workflow from vRealize Orchestrator to the vRealize Automation service catalog.

The service blueprint task has the following limitations.

- When there are multiple tabs in the service blueprint task form only the active tab is validated, when you save the task.
- String type output parameter values of published catalog items are supported.



- String, SecureString, and boolean string input parameter values are supported in the task execution input parameter text box.
- Auto-complete is supported for text boxes that have the gwt-TextBox css property value.
- Form field layout and validation logic cannot be overridden.
- Plug-ins in the service catalog that are only from local vRealize Automation instances can be invoked.

#### Prerequisites

- Verify that a service blueprint is created and published. See the *Installation and Configuration* guide.
- Familiarize yourself with the parameter values that are required to configure and use the vRealize Orchestrator plug-in for Puppet. See the *Using the vRealize Orchestrator Puppet Plug-In 1.0* guide.

#### Procedure

- 1 Click the **Code Stream** tab.
- 2 On the **Pipelines** tab, select an existing pipeline, and click **Edit** to configure it.
- 3 Click the **Stages >** button.
- 4 Click **Add Task**.
- 5 Select **Execute Service Blueprints** from the **Provider** drop-down menu.
- 6 Select a published service blueprint from the drop-down menu.  
  
For example, if you are using the published Puppet plug-in workflow, the endpoint would be Add a Puppet Master.
- 7 Enter the **Tagged workflow**, **Tag**, **Value**, and **Global tag**.
- 8 Click **Save**.

### Configure the Custom Service Blueprint Task

You can configure the parameter values of the published service blueprints based on Puppet plug-in or other workflows.

#### Prerequisites

- Verify that a custom service blueprint task is available in a release pipeline.
- Familiarize yourself with the parameter values that are required to configure and use the vRealize Orchestrator plug-in for Puppet. See the *Using the vRealize Orchestrator Puppet Plug-In 1.0* guide.

#### Procedure

- 1 Open a release pipeline.
- 2 Select the new task in the stage.
- 3 Confirm to save the pipeline.

#### 4 Select when to run the task in the release pipeline.

Option	Description
<b>Always</b>	Runs the release pipeline task without conditions.
<b>On Condition(s)</b>	<p>Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped.</p> <p>A boolean expression, which uses the following operands and operators, is supported.</p> <ul style="list-style-type: none"> <li>■ Pipeline variables such as, <code>\${pipeline.variableName}</code>. Curly brackets are reserved for entering pipeline variables.</li> <li>■ Task output variables such as, <code>\${Stage1.task1.machines[0].value.hostIp[0]}</code></li> <li>■ Default pipeline binding variables such as, <code>\${releasePipelineName}</code></li> <li>■ Case insensitive Boolean values such as, true, false, 'true', 'false'</li> <li>■ Integer or decimal values without quotes</li> <li>■ String values used with single or double quotes such as, "test", 'test'</li> <li>■ String and Numeric types of values such as, == Equals and != Not Equals</li> <li>■ Relational operators such as, &gt;, &gt;=, &lt;, and &lt;=</li> <li>■ Boolean logic such as, &amp;&amp; and   </li> <li>■ Arithmetic operators such as, +, -, *, and /</li> <li>■ Nested expressions using round brackets</li> <li>■ Strings with literal value ABCD are evaluated as false and the task is skipped.</li> </ul> <p>Unary operators are not supported.</p> <p>A sample condition, <code>\${Stage1.task1.output} == "Passed"    \${pipeline.variableName} == 39</code></p>

#### 5 (Optional) Select the **Continue Pipeline execution even if this task fails** option.

This option allows the release pipeline to bypass the failed task and continue with the execution process.

#### 6 Configure the workflow parameters in the release pipeline.

The parameter values for a plug-in workflow depends on the type of the workflow.

For example, you can configure the following Puppet plug-in workflows.

- Add a Puppet Master
- Validate a Puppet Master
- Update a Puppet Master
- Remove a Puppet Master
- Node Management Workflows
- Hieria Workflows
- Manifest Workflows
- Samples Workflows
- Experimental Puppet Plug-In Rake Workflows

- 7 (Optional) Select the script task output to share the output with other tasks in the release pipeline.

The output contains the runtime data that is written to the \$SCRIPT\_RESPONSE\_FILE file.

- 8 Click **Save**.

### Example: Task Output Format for a Custom Service Blueprint

The following task output format is an example for a custom service blueprint task.

```
[
  {
    "description": "master",
    "name": "master",
    "value": "",
    "type": "Puppet:Master"
  },
  {
    "description": "workflowExecutionId",
    "name": "workflowExecutionId",
    "value": "8af0d1274f72d384014fb05beec144a9",
    "type": "STRING"
  },
  {
    "description": "workflowId",
    "name": "workflowId",
    "value": "0ddc5db0-2c43-46af-93cd-b3507fb0fc5b",
    "type": "STRING"
  },
  {
    "description": "__asd_requestedBy",
    "name": "__asd_requestedBy",
    "value": "test@test.com",
    "type": "STRING"
  },
  {
    "description": "__asd_requestedFor",
    "name": "__asd_requestedFor",
    "value": "test@test.com",
    "type": "STRING"
  },
  {
    "description": "workflowName",
    "name": "workflowName",
    "value": "Add a Puppet Master",
    "type": "STRING"
  },
  {
    "description": "__asd_tenantRef",
    "name": "__asd_tenantRef",
    "value": "qe",
    "type": "STRING"
  },
  {
    "description": "__asd_subtenantRef",
```

```

    "name": "__asd_subtenantRef",
    "value": "4f9adef5-f09e-408b-8427-57cbc18e8e90",
    "type": "STRING"
  },
  {
    "description": "__asd_catalogRequestId",
    "name": "__asd_catalogRequestId",
    "value": "8af0d1274f72d384014fb05bd2bb44a3_96bdad96-91b3-4306-a255-be8049fbe2f2",
    "type": "STRING"
  }
]

```

## Add a jFrog Artifactory Task

An artifact task lets you search for artifacts from Artifactory. When you search for artifacts, Artifactory interfaces with various repositories such as Nexus and Yum.

When you include an Artifactory task in a pipeline stage, you can run a pipeline execution every time you develop new code that affects that artifact. The search output parameter from varied source repositories is always the same, and includes a repository name, a download URL, and size information, if available.

---

**Note** The open source version of Artifactory 4.14.2 exposes an issue in the Artifactory REST API. When you click **Validate** on the artifact that you add to a task, the REST API should validate the artifact. Even though vRealize Code Stream resolves the artifact, the Artifactory REST API cannot locate the artifact.

---

### Prerequisites

- Verify that a pipeline is available. See [Create a Release Pipeline](#).
- Verify that an Artifactory server endpoint is registered. See the *Installation and Configuration* guide.

### Procedure

- 1 Click the **Code Stream** tab.
- 2 On the **Pipelines** tab, select an existing pipeline, and click **Edit** to configure it.
- 3 Click the **Stages >** button.
- 4 Click **Add Task**.
- 5 From the **Provider** drop-down menu, select **Artifactory**.
- 6 Enter a task name and click **OK**.
- 7 Select the new task in the stage.

## 8 Configure the task.

- a In the Artifact Properties area, click the drop-down menu, select an endpoint, and select the method to add the endpoint.

**Table 2-1. Add an Endpoint**

Option	Description
Add By Search	Locate artifacts in the Artifactory repository by entering search parameters. Select the repository and the search type. Enter values for the name and path, and click <b>Search</b> .
Add By Name	Enter the name of an artifact and provide the property values. Select the repository, select the search type, and enter the name of the artifact. Enter values for the name and path.

- b To use a simple JSON structure to output your artifacts, click **Add**, and enter an Artifact key of your choice, with no dot or space in the key.
- c Click **Save**.

## 9 Confirm to save the pipeline.

## 10 Select when to run the task in the release pipeline.

Option	Description
<b>Always</b>	Runs the release pipeline task without conditions.
<b>On Condition(s)</b>	<p>Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped.</p> <p>A boolean expression, which uses the following operands and operators, is supported.</p> <ul style="list-style-type: none"> <li>■ Pipeline variables such as, <code>\${pipeline.variableName}</code>. Curly brackets are reserved for entering pipeline variables.</li> <li>■ Task output variables such as, <code>\${Stage1.task1.machines[0].value.hostIp[0]}</code></li> <li>■ Default pipeline binding variables such as, <code>\${releasePipelineName}</code></li> <li>■ Case insensitive Boolean values such as, true, false, 'true', 'false'</li> <li>■ Integer or decimal values without quotes</li> <li>■ String values used with single or double quotes such as, "test", 'test'</li> <li>■ String and Numeric types of values such as, == Equals and != Not Equals</li> <li>■ Relational operators such as, &gt;, &gt;=, &lt;, and &lt;=</li> <li>■ Boolean logic such as, &amp;&amp; and   </li> <li>■ Arithmetic operators such as, +, -, *, and /</li> <li>■ Nested expressions using round brackets</li> <li>■ Strings with literal value ABCD are evaluated as false and the task is skipped.</li> </ul> <p>Unary operators are not supported.</p> <p>A sample condition, <code>\${Stage1.task1.output} == "Passed"    \${pipeline.variableName} == 39</code></p>

## 11 (Optional) Select the **Continue Pipeline execution even if this task fails** option.

This option allows the release pipeline to bypass the failed task and continue with the execution process.

## Search Artifacts from the Artifactory Repository

You can enter a few search parameters to locate and filter the pertinent artifacts from the Artifactory repository.

### Prerequisites

Verify that you have an existing artifact task. See [Add a jFrog Artifactory Task](#).

### Procedure

- 1 Open an existing artifact task.
- 2 Select **Add By Search**.
- 3 Select a repository from the drop-down menu.

#### 4 Determine a search type and add the search parameters.

The search type depends on your repository and artifact.

For example, you can use the `gavc` search type to search for the Maven artifacts.

Search Type	Description and Sample Search Parameters
<b>gavc</b>	<p>Provide a group ID, artifact ID, version ID, or classifier parameters for the search on local repositories.</p> <hr/> <p><b>Note</b> On remote repositories, the search requires a group ID, artifact ID, and version ID. You cannot use the asterisk (*) as a search parameter.</p> <hr/> <ul style="list-style-type: none"> <li>■ The group ID is the organization that published the artifact. For example, <code>org.springframework</code>.</li> <li>■ The artifact ID is the identifier of the artifact. For example, <code>travel</code>.</li> <li>■ The version ID parameter supports the keywords: <code>LATEST</code> and <code>version-SNAPSHOT</code>. For example, enter <code>4.1-SNAPSHOT</code> to get the snapshot of a version.</li> </ul> <p>The classifier is optional and supports the asterisk (*). For example, <code>release*</code>, <code>*release*</code>, or <code>*</code>.</p> <p>When you start a <code>gavc</code> search type from an Artifactory repository, the search result displays all the files with the same artifact name and different extensions. You can narrow the search results with the pattern search type instead.</p>
<b>pattern</b>	<p>Provide the artifact name or path parameters for the search on local repositories.</p> <hr/> <p><b>Note</b> On remote repositories, the search requires the complete artifact path. The artifact name is optional.</p> <hr/> <ul style="list-style-type: none"> <li>■ The name parameter supports the asterisk (*). For example, <code>test*</code>, <code>test.*</code>, <code>*.jar</code>, or <code>*</code>.</li> <li>■ The path parameter supports the asterisk (*) for local repositories. For example, <code>path/*/release</code> searches under the path directory in the <code>/test/release</code> and <code>/dev/release</code> folders for the artifacts only in those folders. Each wildcard (*) represents one level in the folder structure.</li> </ul>

Search Type	Description and Sample Search Parameters
<b>properties</b>	<p>Defined in the Artifactory server to tag artifacts with custom user properties. These properties can be any string values. An artifact can have multiple properties and these properties can have multiple values. You can use these properties instead of the actual group ID, version, or specific path to locate an artifact in the repository.</p> <p>For example, the <b>Property</b> field can be artifactory.licenses and the <b>Value</b> field can be BSD.</p> <hr/> <p><b>Note</b> The properties search function does not work on remote repositories. You can cache artifacts or locally tag the artifacts to use this search function.</p>
<b>build</b>	<p>You cannot create an artifact in the Artifactory user interface. You must push a build from your CI server to the Artifactory user interface.</p> <ul style="list-style-type: none"> <li>■ The build Name is the name of the job that is run on your CI system. For example, Jenkins-release.</li> <li>■ The build Number supports the keyword named LATEST, or a build status. For example, you can search for builds with the Prod status.</li> <li>■ The name lets you filter an artifact from the list of artifacts in a specific build. For example, public-API.jar or public-*.jar</li> </ul> <hr/> <p><b>Note</b> The build search function does not work on remote repositories.</p>

- 5 If you use a variable as a search parameter, enter a temporary value in the **Value** text box, and click **Search**.

The variable is applied during the artifact configuration for the property, and the temporary value is ignored.

For example, if you use `${LATEST}` as the version variable for a gavc search type, you must add the current version, such as 2.1, in the temporary value. It is necessary to enter this value, because the version might be different when you run the pipeline.

If the search shows that an artifact does not exist, verify that the search parameters are accurate.

- 6 Select one or more artifacts from the search results and click **Add**.

- 7 To verify that the artifact exists in the repository, click **Validate**.

If a variable is assigned as an artifact property, then the validation fails. You can replace the variable with an artifact property value and validate.

- 8 To expose to other tasks in the pipeline, select one or more artifact output properties.

For example, you can select to expose the artifact output and the status results in a custom task.

- 9 Click **Save**.

## Add Artifacts by Name to the Artifact Task

You can specify the name of an artifact and provide the property values for the artifact task.



## Prerequisites

Verify that you have an existing artifact task. See [Add a jFrog Artifactory Task](#).

## Procedure

- 1 Open an existing artifact task.
- 2 Select **Add By Name**.
- 3 Select a repository from the drop-down menu.
- 4 Determine a search type and add the search parameters.

The search type depends on your repository and artifact.

For example, you can use the `gavc` search type to search for the Maven artifacts.

Search Type	Description and Sample Search Parameters
<b>gavc</b>	<p>Provide a group ID, artifact ID, version ID, or classifier parameters for the search on local repositories.</p> <p><b>Note</b> On remote repositories, the search requires a group ID, artifact ID, and version ID. You cannot use the asterisk (*) as a search parameter.</p> <ul style="list-style-type: none"> <li>■ The group ID is the organization that published the artifact. For example, <code>org.springframework</code>.</li> <li>■ The artifact ID is the identifier of the artifact. For example, <code>travel</code>.</li> <li>■ The version ID parameter supports the keywords: <code>LATEST</code> and <code>version-SNAPSHOT</code>. For example, enter <code>4.1-SNAPSHOT</code> to get the snapshot of a version.</li> </ul> <p>The classifier is optional and supports the asterisk (*). For example, <code>release*</code>, <code>*release*</code>, or <code>*</code>.</p> <p>When you start a <code>gavc</code> search type from an Artifactory repository, the search result displays all the files with the same artifact name and different extensions. You can narrow the search results with the pattern search type instead.</p>
<b>pattern</b>	<p>Provide the artifact name or path parameters for the search on local repositories.</p> <p><b>Note</b> On remote repositories, the search requires the complete artifact path. The artifact name is optional.</p> <ul style="list-style-type: none"> <li>■ The name parameter supports the asterisk (*). For example, <code>test*</code>, <code>test.*</code>, <code>*.jar</code>, or <code>*</code>.</li> <li>■ The path parameter supports the asterisk (*) for local repositories. For example, <code>path/*/release</code> searches under the path directory in the <code>/test/release</code> and <code>/dev/release</code> folders for the artifacts only in those folders. Each wildcard (*) represents one level in the folder structure.</li> </ul>

Search Type	Description and Sample Search Parameters
<b>properties</b>	<p>Defined in the Artifactory server to tag artifacts with custom user properties. These properties can be any string values. An artifact can have multiple properties and these properties can have multiple values. You can use these properties instead of the actual group ID, version, or specific path to locate an artifact in the repository.</p> <p>For example, the <b>Property</b> field can be artifactory.licenses and the <b>Value</b> field can be BSD.</p> <hr/> <p><b>Note</b> The properties search function does not work on remote repositories. You can cache artifacts or locally tag the artifacts to use this search function.</p>
<b>build</b>	<p>You cannot create an artifact in the Artifactory user interface. You must push a build from your CI server to the Artifactory user interface.</p> <ul style="list-style-type: none"> <li>■ The build Name is the name of the job that is run on your CI system. For example, Jenkins-release.</li> <li>■ The build Number supports the keyword named LATEST, or a build status. For example, you can search for builds with the Prod status.</li> <li>■ The name lets you filter an artifact from the list of artifacts in a specific build. For example, public-API.jar or public-*.jar</li> </ul> <hr/> <p><b>Note</b> The build search function does not work on remote repositories.</p>

5 Enter the artifact name and click **Add**.

6 To verify that the artifact exists in the repository, click **Validate**.

If a variable is assigned as an artifact property, then the validation fails. You can replace the variable with an artifact property value and validate.

7 To expose to other tasks in the pipeline, select one or more artifact output properties.

For example, you can select to expose the artifact output and the status results in a custom task.

8 Click **Save**.

## Add a Script Task

You can use a custom script task to configure bash or PowerShell scripts and run scripts on any Linux or Windows machine. After the script runs, you can monitor the script progress and capture the script response. You can pass the script response as input to the other release pipeline tasks.

### Prerequisites

- Verify the following for Bash script orchestration:
  - The SSH service is configured on the Linux host.
  - Verify that SSHD configuration `MaxSessions` is 50.
- Verify the following for PowerShell script orchestration:
  - The service named `winrm` is configured on the Windows host.

- Verify *winrm* is configured for *MaxShellsPerUser* and *MaxMemoryPerShellMB*.
- Verify that the number of concurrent operations per user is acceptable. See [Troubleshooting PowerShell Task Failures Due to Concurrent Operations](#).
- Verify that memory for the *winrm* service is not constrained. See [Troubleshooting PowerShell Task Failures Due to Memory Constraints](#).

## Procedure

- 1 Click the **Code Stream** tab.
- 2 On the **Pipelines** tab, select an existing pipeline, and click **Edit** to configure it.
- 3 Click the **Stages >** button.
- 4 Click **Add Task**.
- 5 Select **Script** from the **Provider** drop-down menu.
- 6 Enter a name for this custom task and click **OK**.
- 7 Select the new task in the stage.
- 8 Confirm to save the pipeline.
- 9 Select when to run the task in the release pipeline.

Option	Description
<b>Always</b>	Runs the release pipeline task without conditions.
<b>On Condition(s)</b>	<p>Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped.</p> <p>A boolean expression, which uses the following operands and operators, is supported.</p> <ul style="list-style-type: none"> <li>■ Pipeline variables such as, <code>\${pipeline.variableName}</code>. Curly brackets are reserved for entering pipeline variables.</li> <li>■ Task output variables such as, <code>\${Stage1.task1.machines[0].value.hostIp[0]}</code></li> <li>■ Default pipeline binding variables such as, <code>\${releasePipelineName}</code></li> <li>■ Case insensitive Boolean values such as, <code>true</code>, <code>false</code>, <code>'true'</code>, <code>'false'</code></li> <li>■ Integer or decimal values without quotes</li> <li>■ String values used with single or double quotes such as, <code>"test"</code>, <code>'test'</code></li> <li>■ String and Numeric types of values such as, <code>==</code> Equals and <code>!=</code> Not Equals</li> <li>■ Relational operators such as, <code>&gt;</code>, <code>&gt;=</code>, <code>&lt;</code>, and <code>&lt;=</code></li> <li>■ Boolean logic such as, <code>&amp;&amp;</code> and <code>  </code></li> <li>■ Arithmetic operators such as, <code>+</code>, <code>-</code>, <code>*</code>, and <code>/</code></li> <li>■ Nested expressions using round brackets</li> <li>■ Strings with literal value ABCD are evaluated as false and the task is skipped.</li> </ul> <p>Unary operators are not supported.</p> <p>A sample condition, <code>\${Stage1.task1.output} == "Passed"    \${pipeline.variableName} == 39</code></p>

**10 (Optional) Select the **Continue Pipeline execution even if this task fails** option.**

This option allows the release pipeline to bypass the failed task and continue with the execution process.

After a custom script task runs, the EXECUTION\_ID\_FOLDER run folder that contains logFile and exitStatusFile is created in the following paths:

<b>Bash Script</b>	<b>Location</b>
Remote script	<CODESTREAM_FOLDER_PATH>/<ScriptName> Example: /var/tmp/codestream/demo/f8e27157-792f-428e-8bbd-f88ac106ddd2
Inline script	<CODESTREAM_FOLDER_PATH>/user_defined_script Example: /var/tmp/codestream/user_defined_script/e05606b8-8acb-43c7-9293-a465e86cc3fe

<b>PowerShell Script</b>	<b>Location</b>
Remote script	<CODESTREAM_FOLDER_PATH>/<ScriptName> Example: C:\Users\fritz\AppData\Local\Temp\codestream\demo\f8e27157-792f-428e-8bbd-f88ac106ddd2
Inline script	<CODESTREAM_FOLDER_PATH>/user_defined_script Example: C:\Users\fritz\AppData\Local\Temp\codestream\user_defined_script\e05606b8-8acb-43c7-9293-a465e86cc3fe

Following are the script plug-in system properties, which can be configured on the vRealize Code Stream appliance:

**Table 2-2. Bash Script**

<b>Property Name</b>	<b>Description</b>	<b>Default Value</b>
fms.bundle.script.bash.codestream.folder	Configure path for the codestream folder where the script execution folder is created.	/var/tmp/codestream/
fms.bundle.script.bash.log.snippet.lines	Configure number of lines for the log snippet displayed in script response.	50
fms.bundle.script.bash.log.snippet.line.length	Configure the maximum characters per log line after which the log is truncated.	1024
fms.bundle.script.bash.upload.chunk.bytes	For inline script, the data is uploaded in chunks and this property is used to configure chunk size.	1024 * 50

**Table 2-3. PowerShell Script**

Property Name	Description	Default Value
fms.bundle.script.powershell.code stream.folder	Configure path for the codestream folder where the script execution folder is created.	C:\Users\<User_Name>\AppData\Local\Temp\codestream\
fms.bundle.script.powershell.log.snippet.lines	Configure number of lines for the log snippet displayed in script response.	50
fms.bundle.script.powershell.log.snippet.line.length	Configure the maximum characters per log line after which the log is truncated.	1024
fms.bundle.script.powershell.upload.chunk.bytes	For inline script, the data is uploaded in chunks and this property is used to configure chunk size.	1024 * 50

## Configure the General Tab

A script task allows you to configure input variables for a script as a simple key-value pair. The variable name must match the variable being used in the script when you configure a general input variable to customize the script.

Pipeline variables are runtime variables that are output from a previous task and are available for consumption by other tasks. The jayway/JsonPath property format is supported.

You can use the `${VAR}` notation to have a task consume parameters that are available as pipeline runtime variables or the output produced from other tasks. You can select task output variables when you configure the release pipeline. By default, the status variable is set as shared by all the tasks in the release pipeline. When these variables must be consumed as a direct input for a field in the user interface, you can set them as `BUILD_ID = 123`, where 123 is the value of `${BUILD_ID}`. The variable must be consumed as `${BUILD_ID}`.

When this variable is consumed as `vcac-123.war`, where 123 is pulled from `${BUILD_ID}`, it must be consumed as `vcac-${BUILD_ID}.war`.

### Prerequisites

- Verify that a script task is available in a release pipeline.
- Verify that your script exists on a remote host that is configured in the script task. For a remote user to run it, the script must have the executable permission.
- Verify that the folder where the remote script exists has the permission to allow a file to be created.
- Familiarize yourself with using task output variables. See [Create a Release Pipeline](#).

### Procedure

- 1 Open a release pipeline.
- 2 Select the new task in the stage.

- 3 Confirm to save the pipeline.
- 4 Select an Execute on Host type and add the applicable details.

Option	Description
<b>User-defined hosts</b>	<p>Enter one or more IP Addresses, DNS hostname, or release pipeline input variables as one of the valid hosts.</p> <p>For example, the input variable <code>machine_ip</code> can be <code>198.51.100.13</code>. The user-defined host can also have <code>\$machine_ip</code> with static hostname and IP Address.</p>
<b>Read From Pipeline or Task Property</b>	<p>Provide a task output variable shared by another task or pipeline variable and apply filter criteria to restrict the script to run only on selected hosts.</p> <p>The pipeline variables are specified on the page before the modeling page. The run time values are applied when the release pipeline runs.</p> <p>For example, the task output variable <code>\${StageName.TaskName.status.outputConfig.host}</code> is bound and shared by all the hosts. By default, the status variable is set as shared by all the tasks in the release pipeline.</p> <p>To differentiate between a pipeline variable and a task output variable, the constant variable always precedes the pipeline variable. For example, <code>\${pipeline.PipelineVariable}</code>.</p>
<b>Filter Criteria</b>	<p>Set one of the following criteria from the drop-down menu.</p> <ul style="list-style-type: none"> <li>■ <b>ALL HOSTS.</b> Run the script on all the hosts that the host group variable gives.</li> <li>■ <b>STARTS WITH.</b> From the host group, filter machines whose names start with the prefix as given by the user. The filter search criteria is case-sensitive.</li> <li>■ <b>ENDS WITH.</b> Filter machines whose names end with the suffix as given by the user.</li> <li>■ <b>EXACT MATCH.</b> Run the script only on the machine whose name exactly matches the DNS hostname or IP Address.</li> </ul>

- 5 Type the host user name and password for a remote connection to the host.
- 6 Select a **Bash** or **PowerShell** script from the drop-down menu.
- 7 Select the script type.

Option	Description
<b>Remote Script File</b>	<p>Specify the script filename and the path that already resides on the remote host.</p> <p>An example file path for a Bash script is <code>/home/test/template/test.sh</code>.</p>
<b>User-Defined Script</b>	<p>Specify the inline Bash or PowerShell script to be run on the remote host.</p> <p>Inline scripts enable dynamic creation, and the user-defined script to run on the remote host at the time the pipeline runs.</p> <p>An example of Bash inline script:</p> <pre>message="Hello World" echo \$message</pre>

- 8 Enter the working directory path from where the script must be run.

## 9 Configure the command-line arguments.

The command-line arguments can be variables or binding variables. If the argument value has a space, enclose the value in double quotation marks.

```
${pipeline.buildNo} 10 2000 300 "test application"
```

You can view the resolved values of the command-line arguments in the task results after the custom script task runs in the release pipeline.

## 10 (Optional) To share the output with other tasks in the release pipeline, select the script task output.

The output contains the runtime data that is written to the \$SCRIPT\_RESPONSE\_FILE file.

## 11 Click **Save**.

### Example: Task Output Format for a Custom Script Task

The following task output format is an example for a custom script task.

```
[
  {
    "description": "Host Response",
    "name": "198.51.100.13",
    "value": "Sample Response of Linux Host 01. Sample Response of Linux Host 01.",
    "type": "STRING"
  },
  {
    "description": "Host Response",
    "name": "198.51.100.13",
    "value": "Sample Response of Linux Host 02. Sample Response of Linux Host 02.",
    "type": "STRING"
  }
]
```

## Configure the Advanced Tab

You use the artifact parameters to map artifacts to a script task so that the script task can download artifacts from a repository to the host.

### Prerequisites

- Verify that a script task is available in a release pipeline.
- Verify that an artifact task is configured, and that the artifact output property is shared. See [Add a jFrog Artifactory Task](#).

### Procedure

- 1 Open a release pipeline.
- 2 Select the script task from the stage column.
- 3 Click the **Advanced** tab.

#### 4 Enter the artifact group variable.

Use the form: `${StageName.TaskName.ARTIFACT_OUTPUT}`

This variable is generated as part of the artifact output from the existing artifact task.

For example, an artifact group variable might be: `${Dev.Artifact.ARTIFACT_OUTPUT}`

An incorrect form, which does not work as an artifact group variable is: `$`

`{StageName.TaskName.artifacts}`

#### 5 Click **Add** to define the Artifact input parameters.

Option	Description
<b>Parameter Name</b>	Name of the Shell variable that refers to an artifact in the script. For example: artifact_config
<b>Parameter Value</b>	Artifact name defined in the artifact group. For example: CodeStream_JAR

#### 6 From the drop-down menu, select **Download URL** or **Repository URL**.

The script calls the download URL or the repository URL for an artifact by mapping an artifact input property.

#### 7 Click **Add** to define the Other input.

Option	Description
<b>Parameter Name</b>	Name of the other Shell variable defined in the script. For example: portNumber or buildID
<b>Parameter Value</b>	Value for the Shell variable. For example: 800

#### 8 Click **Save**.

### Example: Example Scripts for a Custom Script Task

The following example shows a script that configures a vRealize Automation application.

```
$ cat configureAppServer.sh
echo "Configure app server";
echo "VCAC Application Download URL: $VCAC_APP_DOWNLOAD_URL";

wget -O $VCAC_APP_DOWNLOAD_URL
echo "Configuring VCAC application";

echo "Starting application on port: $APPLICATION_PORT";

MACHINE_IP = ifconfig | sed -En 's/127.0.0.1//;s/.*inet (addr:)?((([0-9]*\.)(3)[0-9]*).*\/2/p'
printf "Application URL: $MACHINE_IP:$APPLICATION_PORT/vcac/" > $SCRIPT_RESPONSE_FILE
```



The script refers to the environment variable named `VCAC_APP_DOWNLOAD_URL` to determine the version of the VCAC artifact to download from the repository. The artifact input parameter to the script must be the parameter named `VCAC_APP_DOWNLOAD_URL`.

```
wget -O VCAC_APP_DOWNLOAD_URL
echo "Configuring VCAC application";
```

To enable the script to share data with the other release pipeline tasks, the contents must be written in a response file. The response file contents are stored in the script output variable.

```
MACHINE_IP = ifconfig | sed -En 's/127.0.0.1//;s/.*inet (addr:)?((([0-9]*\.){3}[0-9]*).*\/2/p'
printf "Application URL: $MACHINE_IP/vcac/" > $SCRIPT_RESPONSE_FILE
```

## Add a vRealize Automation Task

You use a provision task to provision machines. You can register a vRealize Automation 6.x or 7.x instance with vRealize Code Stream to provision infrastructure in a specific environment.

vRealize Code Stream can also start multiple vRealize Automation instances. You can create machines from a single machine blueprint for a provision task. The task output is an array of machines.

### Prerequisites

- Verify that the vRealize Automation server endpoint is registered. See the *Installation and Configuration* guide.
- Verify that you completed the following tasks in vRealize Automation 6.2.x or 7.x to add an endpoint in vRealize Code Stream.
  - Choosing an Endpoint Scenario see *IaaS Configuration for Virtual Platforms*.
  - Create a Fabric Group see *IaaS Configuration for Virtual Platforms*.
  - Create a Business Group see *IaaS Configuration for Virtual Platforms*.
  - Create a Reservation see *IaaS Configuration for Virtual Platforms*.
  - Create a Reservation Policy see *IaaS Configuration for Physical Machines*.
  - Create a Network Profile see *IaaS Integration for Multi-Machine Services*.
  - Create a Blueprint see *IaaS Configuration for Virtual Platforms*.
  - Publish a Blueprint see *IaaS Configuration for Virtual Platforms*.
- Verify that the vRealize Automation catalog item is assigned to a service.

The service must be added to the entitlement for the Business group proxy user.

### Procedure

- 1 Click the **Code Stream** tab.
- 2 On the **Pipelines** tab, select an existing pipeline, and click **Edit** to configure it.

- 3 Click the **Stages >** button.
- 4 Click **Add Task**.
- 5 Select **vRealize Automation 6** from the **Provider** drop-down menu.
- 6 Select the vRealize Automation server endpoint you registered.
- 7 Enter a name for the provision task and click **OK**.
- 8 Select the new task in the stage.
- 9 Confirm to save the pipeline.
- 10 Select when to run the task in the release pipeline.

Option	Description
<b>Always</b>	Runs the release pipeline task without conditions.
<b>On Condition(s)</b>	<p>Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped.</p> <p>A boolean expression, which uses the following operands and operators, is supported.</p> <ul style="list-style-type: none"> <li>■ Pipeline variables such as, <code>\${pipeline.variableName}</code>. Curly brackets are reserved for entering pipeline variables.</li> <li>■ Task output variables such as, <code>\${Stage1.task1.machines[0].value.hostIp[0]}</code></li> <li>■ Default pipeline binding variables such as, <code>\${releasePipelineName}</code></li> <li>■ Case insensitive Boolean values such as, true, false, 'true', 'false'</li> <li>■ Integer or decimal values without quotes</li> <li>■ String values used with single or double quotes such as, "test", 'test'</li> <li>■ String and Numeric types of values such as, == Equals and != Not Equals</li> <li>■ Relational operators such as, &gt;, &gt;=, &lt;, and &lt;=</li> <li>■ Boolean logic such as, &amp;&amp; and   </li> <li>■ Arithmetic operators such as, +, -, *, and /</li> <li>■ Nested expressions using round brackets</li> <li>■ Strings with literal value ABCD are evaluated as false and the task is skipped.</li> </ul> <p>Unary operators are not supported.</p> <p>A sample condition, <code>\${Stage1.task1.output} == "Passed"    \${pipeline.variableName} == 39</code></p>

- 11 (Optional) Select the **Continue Pipeline execution even if this task fails** option.  
This option allows the release pipeline to bypass the failed task and continue with the execution process.
- 12 Select the endpoint from the **Endpoint** drop-down menu.
- 13 Select the published vRealize Automation blueprint from the **Blueprint** drop-down menu.

## 14 Enter the machine properties.

Option	Description
<b>Number of Machines</b>	Enter the number of machines to be provisioned and the configuration details. This property allows variable binding.
<b>Lease (Days) and other machine details</b>	Enter the Lease (Days), Number of CPUs, Memory, Storage values, and Description.  <b>Note</b> An existing machine blueprint configuration can overwrite the values specified here.
<b>Owner</b>	Select <b>Logged In User</b> to allow the user currently logged in to run the release pipeline.  Select <b>Enter Owner or Bind</b> to allow a different user to log in, configure the machine, and provision it when the release pipeline is run. You must enter the login credential information of that user.

## 15 (Optional) Select to share the machine output information with other tasks during the release pipeline execution.

This output contains details of the machines successfully provisioned by this task. You can use this variable in succeeding tasks to retrieve the machine details.

## 16 Click **Save**.

## Example: Task Output Format for a vRealize Automation Task

The following task output format is an example for a provisioning task.

```
[
  {
    "name": "vcac-prov01",
    "value": {
      "memory": 1024,
      "machineId": "f9ee3f71-c5d0-4138-9520-24e15e376d13",
      "hostIp": ["10.72.12.56"],
      "cpu": 1,
      "endLease": 1410478627000,
      "storageSize": 4,
      "startLease": 1410392227000,
      "blueprintId": "f9ee3f71-c5d0-4138-3476-24e15e376f36",
      "provider": "iaas-service"
    },
    "type": "MACHINE"
  }
]
```

## Request a Machine Blueprint

You can request blueprints and customize machine components to provision a machine or a stack of machines.

You can modify the properties for blueprints configured in vRealize Automation and view the task results.

## Prerequisites

You need to install and configure the vRealize Automation IaaS feature and create a blueprint for your vSphere machine component.

---

**Note** vRealize Automation 6.x and 7.x are supported and you can select separate tasks and configure distinct endpoints in the vRealize Automation Properties pane when you configure the task.

---

## Procedure

- 1 Click the **Code Stream** tab.
- 2 On the **Pipelines** tab, select an existing pipeline, and click **Edit** to configure it.
- 3 Click the **Stages >** button.
- 4 Select a task and select **vRealize Automation 7** from the **Provider** drop-down menu.
- 5 Select the endpoint and the blueprint to view the components.

The catalog API provides all component information, which you can modify after logging in to vRealize Automation. You can set constraints or machine resource information such as number of CPUs and storage. You can also choose to display or hide any custom property in the request form and submit the changes to the blueprint. If you modify any information in vRealize Automation, a message displays to indicate that you modified the blueprint.

- 6 Click the **General** tab and provide the number of deployments and the number of lease days.
- 7 Click the **Advanced** tab to provision the selected blueprint.

The vRealize Automation API provides an instance of the blueprint. You can view all components of the blueprint, configuration items, and data nodes. The input property of the available fields such as machine components is validated.

- 8 Select **Save**, activate the pipeline, and click **Execute**.

You can view the status and details of the request on the vRealize Automation **Request** tab while the server processes your request.

- 9 After the pipeline execution is complete, select the task to view the output property.

The **Machines** output property displays all the output properties of the machines provisioned in the blueprint, and the **Resources** output property displays all resources provisioned by the request.

## Add a vRealize Orchestrator Workflow Task

With a vRealize Orchestrator task, you can configure a vRealize Orchestrator to be triggered as part of the release pipeline process.

## Prerequisites

Verify that you created a workflow and tagged it with the vRCS\_CUSTOM keyword in the vRealize Orchestrator Workflow Designer.

## Procedure

- 1 Click the **Code Stream** tab.
- 2 On the **Pipelines** tab, select an existing pipeline, and click **Edit** to configure it.
- 3 Click the **Stages >** button.
- 4 Click **Add Task**.
- 5 Select **vRealize Orchestrator Workflow** from the **Provider** drop-down menu.
- 6 Enter a name for the custom task and click **OK**.
- 7 Select the new task in the stage.
- 8 Confirm to save the pipeline.
- 9 Select when to run the task in the release pipeline.

Option	Description
<b>Always</b>	Runs the release pipeline task without conditions.
<b>On Condition(s)</b>	<p>Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped.</p> <p>A boolean expression, which uses the following operands and operators, is supported.</p> <ul style="list-style-type: none"> <li>■ Pipeline variables such as, <code>\${pipeline.variableName}</code>. Curly brackets are reserved for entering pipeline variables.</li> <li>■ Task output variables such as, <code>\${Stage1.task1.machines[0].value.hostIp[0]}</code></li> <li>■ Default pipeline binding variables such as, <code>\${releasePipelineName}</code></li> <li>■ Case insensitive Boolean values such as, true, false, 'true', 'false'</li> <li>■ Integer or decimal values without quotes</li> <li>■ String values used with single or double quotes such as, "test", 'test'</li> <li>■ String and Numeric types of values such as, == Equals and != Not Equals</li> <li>■ Relational operators such as, &gt;, &gt;=, &lt;, and &lt;=</li> <li>■ Boolean logic such as, &amp;&amp; and   </li> <li>■ Arithmetic operators such as, +, -, *, and /</li> <li>■ Nested expressions using round brackets</li> <li>■ Strings with literal value ABCD are evaluated as false and the task is skipped.</li> </ul> <p>Unary operators are not supported.</p> <p>A sample condition, <code>\${Stage1.task1.output} == "Passed"    \${pipeline.variableName} == 39</code></p>

- 10 (Optional) Select the **Continue Pipeline execution even if this task fails** option.

This option allows the release pipeline to bypass the failed task and continue with the execution process.

- 11 Select the workflow name from the drop-down menu.

For example: **Manual Task** or other custom workflows.

## 12 Enter the input parameters.

**Table 2-4. Example Input Parameters for vRealize Orchestrator Workflow Task**

Workflow	Input Values
Manual task	<p>Enter the input parameters for the vRealize Orchestrator task to run:</p> <ul style="list-style-type: none"> <li>■ <b>taskDetail.</b> Enter the message for users to review in the task detail. The message can indicate whether the pipeline was paused, and how to approve the pipeline to continue to run. It can also include the state of the tasks and stages in the pipeline, where to find approvals, and how to resume or reject the pipeline run. For example:  <code>\${releasePipelineName} paused at \${taskName} in \${stageName}. To continue the pipeline run, you must approve the request.</code></li> <li>■ <b>taskGroup.</b> Enter the common name for the taskGroup who must receive and approve the request for the manual task. For example, enter the task group in the form of <code>my-team@example.org</code> or <code>example-corp\my-team</code>. This entry is case-sensitive.</li> <li>■ <b>expiryInDays.</b> Amount of time before the task expires. The default is 15 days.</li> </ul> <p>To approve the request, in vRealize Automation, click <b>Inbox</b>, read the approval message, enter a comment, and click <b>Approve</b> or <b>Reject</b>.</p>

## 13 Click **Save**.

## 14 After the pipeline runs, you can view the results of the task to determine who responded to the task and when.

- Click **Pipeline Executions**, and open the pipeline.
- In the stage, click a task, and examine the results.

**Task Result: s**

**Provider:** vRealize Orchestrator  
**Duration:** 7m 16s 201msec  
 (11/3/17 7:02 AM - 11/3/17 7:09 AM)

**Status:** ✔ COMPLETED  
**Message:** Workflow completed successfully.

Result And Input | Raw Output Properties | Raw Input Properties

**Result**

Name	Value
workflowExecutionId	{ "name": "result", "type": "STRING", "description": "Result of workflow run.", "value": "" }
respondedOn	2017-11-03 13:09:32.06
respondedBy	Administrator, Local
comments	Manual task has been completed by the assignee for continuing the pipeline execution.
completed	true

**Input**

Workflow Name: Manual Task

Name	Value
taskDetail	The execution of workflow has been paused at s in Stage0. Approval can be performed by navigating to Inbox -> Manual User Action. Please click "Submit" to resume execution or "Reject" to stop further execution of this pipeline. Resuming execution will start next task in Stage0. If you Reject, the execution will stop at s in Stage0.

Close

The task results indicate who responded, and when.

## Example: Task Output Format for a vRealize Orchestrator Workflow Task

The following task output format is an example for a workflow task.

```
[{
  "name": "result",
  "type": "STRING",
  "description": "Result of workflow run.",
  "value": ""
},
{
  "name": "message",
  "type": "STRING",
  "description": "Message",
  "value": ""
}]
```

## Add a vRealize Orchestrator External Task

You can create a vRealize Orchestrator task to connect to a remote instance of vRealize Orchestrator and run workflows residing in the external orchestrator.

### Prerequisites

- Verify that the vRealize Orchestrator external endpoint is registered.
- Verify that you created a workflow and tagged it with the VRCS\_CUSTOM keyword in the vRealize Orchestrator Workflow Designer.

### Procedure

- 1 Click the **Code Stream** tab.
- 2 On the **Pipelines** tab, select an existing pipeline, and click **Edit** to configure it.
- 3 Click the **Stages >** button.
- 4 Click **Add Task**.
- 5 Select **vRealize Orchestrator External** from the **Provider** drop-down menu.
- 6 Enter a name for the custom task and click **OK**.
- 7 Select the new task in the stage.
- 8 Confirm to save the pipeline.



## 9 Select when to run the task in the release pipeline.

Option	Description
<b>Always</b>	Runs the release pipeline task without conditions.
<b>On Condition(s)</b>	<p>Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped.</p> <p>A boolean expression, which uses the following operands and operators, is supported.</p> <ul style="list-style-type: none"> <li>■ Pipeline variables such as, <code>\${pipeline.variableName}</code>. Curly brackets are reserved for entering pipeline variables.</li> <li>■ Task output variables such as, <code>\${Stage1.task1.machines[0].value.hostIp[0]}</code></li> <li>■ Default pipeline binding variables such as, <code>\${releasePipelineName}</code></li> <li>■ Case insensitive Boolean values such as, true, false, 'true', 'false'</li> <li>■ Integer or decimal values without quotes</li> <li>■ String values used with single or double quotes such as, "test", 'test'</li> <li>■ String and Numeric types of values such as, == Equals and != Not Equals</li> <li>■ Relational operators such as, &gt;, &gt;=, &lt;, and &lt;=</li> <li>■ Boolean logic such as, &amp;&amp; and   </li> <li>■ Arithmetic operators such as, +, -, *, and /</li> <li>■ Nested expressions using round brackets</li> <li>■ Strings with literal value ABCD are evaluated as false and the task is skipped.</li> </ul> <p>Unary operators are not supported.</p> <p>A sample condition, <code>\${Stage1.task1.output} == "Passed"    \${pipeline.variableName} == 39</code></p>

## 10 (Optional) Select the **Continue Pipeline execution even if this task fails** option.

This option allows the release pipeline to bypass the failed task and continue with the execution process.

## 11 Select the endpoint from the **Endpoint** drop-down menu.

## 12 Select the **Workflow** from the drop-down menu and provide the values for the input parameters.

## 13 Click **Save**.

### Example: Task Output Format for a vRealize Orchestrator External Task

The following task output format is an example for a workflow task.

```
[{
    "name": "result",
    "type": "STRING",
    "description": "Result of workflow run.",
    "value": ""
},
{
    "name": "message",
```

```

        "type": "STRING",
        "description": "Message",
        "value": ""
    }
]
```

## Create a Nested Pipeline

You can model a complex pipeline and execute the pipeline in a modular manner by invoking pipelines within pipelines. You can also view the result of the nested pipeline execution.

The nested pipelines allow you to model complex release pipelines by managing pipelines as tasks.

### Prerequisites

Verify that you have available predefined release pipelines.

### Procedure

- 1 Create a master release pipeline.
- 2 Add a stage or add multiple stages to this pipeline and include tasks.
- 3 Add a new task, select **vRealize Code Stream** as the provider and enter the name of the task.
- 4 Select the new task in the stage.
- 5 Confirm to save the pipeline.
- 6 Select when to run the task in the release pipeline.

Option	Description
<b>Always</b>	Runs the release pipeline task without conditions.
<b>On Condition(s)</b>	<p>Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped.</p> <p>A boolean expression, which uses the following operands and operators, is supported.</p> <ul style="list-style-type: none"> <li>■ Pipeline variables such as, <code>\${pipeline.variableName}</code>. Curly brackets are reserved for entering pipeline variables.</li> <li>■ Task output variables such as, <code>\${Stage1.task1.machines[0].value.hostIp[0]}</code></li> <li>■ Default pipeline binding variables such as, <code>\${releasePipelineName}</code></li> <li>■ Case insensitive Boolean values such as, true, false, 'true', 'false'</li> <li>■ Integer or decimal values without quotes</li> <li>■ String values used with single or double quotes such as, "test", 'test'</li> <li>■ String and Numeric types of values such as, == Equals and != Not Equals</li> <li>■ Relational operators such as, &gt;, &gt;=, &lt;, and &lt;=</li> <li>■ Boolean logic such as, &amp;&amp; and   </li> <li>■ Arithmetic operators such as, +, -, *, and /</li> <li>■ Nested expressions using round brackets</li> <li>■ Strings with literal value ABCD are evaluated as false and the task is skipped.</li> </ul> <p>Unary operators are not supported.</p> <p>A sample condition, <code>\${Stage1.task1.output} == "Passed"    \${pipeline.variableName} == 39</code></p>

- 7 (Optional) Select the **Continue Pipeline execution even if this task fails** option.

This option allows the release pipeline to bypass the failed task and continue with the execution process.

- 8 Click the task you created.

- 9 Select the pipeline from the **Pipeline** drop-down menu to configure the selected pipeline as a task.

You can override the input parameters to enable the child pipeline to run with the new values instead of the configured values.

- 10 Click the **Output** tab, select the output properties that you want to publish and provide the custom alias name.

Only the values that you select for publishing are considered in the master pipeline execution.

- 11 Click **Save**.

- 12 Repeat steps 3-5 to configure pipelines for each task.

- 13 Click **Execute** to run the master release pipeline.

- 14 Click the **Pipeline Executions** tab.

The status of each invoked pipeline is displayed.

- 15 Select the master release pipeline and click the task to view the details of the pipeline task.

- 16 Click the **Pipeline Detail** button to view the details of the task modeled in the pipeline.

You can also view the details in the **Dashboard** tab.

## Configure Gating Rules

Gating rules are a set of criteria that each stage must pass to proceed to the subsequent stage. You can configure gating rules based on your requirement for a pipeline.

Gating rules allow you to request approvals. You can set an expiration on the approval for the task. After the pipeline runs, you can examine the pipeline results to see who approved the task and when. If the approval is not granted in the time allowed, the pipeline fails.

If the gating rule is not configured, then the process proceeds to the next stage regardless of the outcome of tasks in the current stage. The workflows are tagged as `vRCS_GATING_RULE` and are listed in the drop-down menu.

A workflow is depicted as a sequence of operations. The workflow consists of an orchestrated and repeatable pattern of business activity enabled by the systematic organization of resources into processes that transform materials, provide services, or process information.

The default workflows are Approval and Test Acceptance Threshold. For information on how you can register the vRO workflow for a gating rule, see *Installation and Configuration guide*.

**Table 2-5. Default Workflows**

Workflow	Description
Approval	This workflow is tagged with a vRCS_GATING_RULE tag. You must provide the approval group DN value and the approval message for a stage to continue or stop. The appropriate user receives an email to approve or reject this workflow. After the user submits or rejects the notification that appears in My Inbox, the workflow either continues or stops.
Test Acceptance Threshold	This workflow is tagged with a vRCS_GATING_RULE tag. You must provide the threshold percentage value and the test result for a task to succeed the gating rule to the next level. This workflow accepts the test result from a Jenkins test task. Based on the configured threshold for passed tests criteria, it allows the pipeline run to proceed to the subsequent stage.


**Prerequisites**

- Verify that the last task of a stage is a test, custom script, or custom workflow.
- Verify that a custom script output is converted to create a JSON and then passed to the gating rule.

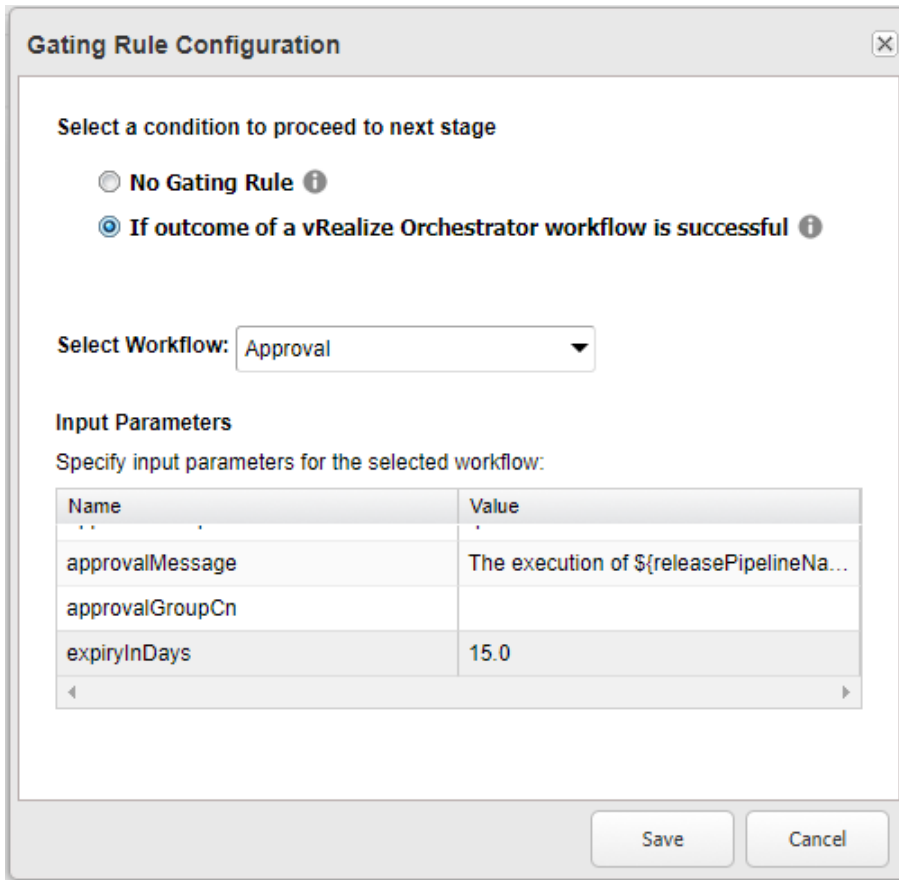
The Jenkins output can be passed directly to the gating rule, as the following example shows.

```
{
  "totalCount": 40,
  "skipCount": 0,
  "failureCount": 0,
  "successCount": 40
}
```

**Procedure**

- 1 Click the **Code Stream** tab.
- 2 On the **Pipelines** tab, select an existing pipeline, and click **Edit** to configure it.
- 3 Click the **Stages >** button.
- 4 Click the Gating Rule icon (  ) next to a stage.

- 5 To create the gating rule, select **If outcome of a vRealize Orchestrator workflow is successful**.



**Gating Rule Configuration**

Select a condition to proceed to next stage

☐ No Gating Rule ⓘ  
☒ If outcome of a vRealize Orchestrator workflow is successful ⓘ

Select Workflow: Approval ▼

**Input Parameters**  
Specify input parameters for the selected workflow:

Name	Value
approvalMessage	The execution of \${releasePipelineNa...}
approvalGroupCn	
expiryInDays	15.0

Save Cancel

- 6 Select the appropriate workflow from the drop-down menu.
- Enter the approval group DN value, **vsphere.local\administrators**, in the **Approval** text box.  
You can also edit the approval message. DN is any valid DN name that is configured in the identity store tied to vRealize Automation.
  - Enter the threshold percentage and the test result value in the **Test Acceptance Threshold** text box.
  - To set an expiration, enter the number of days. For example, 15.0.
- 7 Click **Save**.

# Managing Release Automation

After you execute a release pipeline, you can monitor the status from the Dashboard home page. You can also delete a task, duplicate a pipeline, or delete a pipeline from the **Code Stream** tab.

This chapter includes the following topics:

- [Clone a Task](#)
- [Delete a Task or a Cloned Task](#)
- [Copy a Release Pipeline](#)
- [Delete a Release Pipeline](#)
- [Resume From Failure](#)

## Clone a Task


You can include multiple tasks with the same configuration in your pipeline by creating a task and cloning it. You can place the cloned task in the same stage or move it to a different stage of the pipeline.

You cannot have multiple cloned tasks with the same name in a stage.

### Prerequisites

Verify that you have an existing task. See [Create a Release Pipeline](#).

### Procedure

- 1 Click the **Code Stream** tab.
- 2 On the **Pipelines** tab, select an existing pipeline, and click **Edit** to configure it.
- 3 Click the **Stages >** button.
- 4 Click the gear icon (  ) next to the task in the stage.
- 5 Click **Clone**.  
A clone task appears in the stage.
- 6 Drag the cloned task to another stage or place it in the same stage.  
You can reorganize the order of the cloned task in a stage.
- 7 Confirm to save the pipeline.


## Delete a Task or a Cloned Task

You can delete any task or cloned task in a stage that is not required in the pipeline. Deleting a task might affect the subsequent tasks that refer to the output of that task.

### Prerequisites

Verify that you have an existing task. See [Create a Release Pipeline](#).

### Procedure

- 1 Click the **Code Stream** tab.
- 2 On the **Pipelines** tab, select an existing pipeline, and click **Edit** to configure it.
- 3 Click the **Stages >** button.
- 4 Click the gear icon (  ) next to the task in the stage.
- 5 Click **Delete**.

## Copy a Release Pipeline

Copying or cloning a release pipeline creates the replica of a release pipeline with a different release pipeline name. This replication creates a release pipeline without affecting the existing release pipeline model. The configuration of the source and target release pipelines does not change.

### Prerequisites

Verify that you have created a release pipeline. See [Create a Release Pipeline](#).

### Procedure

- 1 Select the pipeline to copy on the **Pipelines** tab.
- 2 Click **Copy**.  
The Copy Pipeline window appears.
- 3 Enter an appropriate name for the release pipeline replica.
- 4 Enter the description for the release pipeline .
- 5 Click **OK**.

A copy of the release pipeline is created and is listed on the **Pipelines** tab. You can edit this release pipeline to make further modifications.

## Delete a Release Pipeline

Deleting a release pipeline removes the pipeline from the pipeline model.

## Prerequisites

Verify that you have an existing release pipeline. See [Create a Release Pipeline](#).

## Procedure

- 1 Select the pipeline to delete on the **Pipelines** tab.
- 2 Click **Delete**.  
A confirmation message appears.
- 3 Click **Yes** to delete the selected release pipeline.

# Resume From Failure

If the pipeline execution fails, you can resume execution. A pipeline run might fail if artifacts do not exist for a task in the pipeline.

For a failed pipeline execution run, you can resume the pipeline execution from the last failed task(s) in the failed stage.

## Prerequisites

Verify that you have created a release pipeline. See [Create a Release Pipeline](#).

## Procedure

- 1 Select the failed pipeline execution.
- 2 Expand the arrow next to the pipeline name, and review the stages and tasks to determine which task failed.  
  
Tasks that failed display a red line at the top of the task in the stage, and the top of the stage displays a red line.
- 3 Click the task that failed, and read the status message to understand the failure.
- 4 On the **Result And Input** tab, review the information about the artifacts.
- 5 If needed, verify the raw input and output properties.
  - a Click the **Raw Output Properties** tab, and review the values of the output parameters.
  - b Click the **Raw Input Properties** tab, and from the values of external system verify whether the external system is available.  
  
An external system that is not available might cause the transient failure.
  - c Click **Close**.
- 6 Click **Resume Execution**.  
  
The pipeline execution runs again.

You have resumed a failed pipeline.



# Viewing Pipeline Execution Reports

# 4

You can determine key business metrics for your application release by viewing the reports of the successfully executed release pipelines and analyzing the failed pipelines.

This chapter includes the following topics:

- [View the Success Rate of the Executed Pipelines](#)
- [View Failed Pipelines](#)

## View the Success Rate of the Executed Pipelines

You can view a consolidated report of release pipeline executions displayed at predefined intervals to indicate the tasks that fail and make informed decisions about the release progress.

### Procedure

- 1 Select **Code Stream > Reports**.
- 2 Select **Pipeline success Rate** from the **Report** drop-down menu.
- 3 Select the executed pipeline from the **Pipeline** drop-down menu.
- 4 To view the reports for the selected time period, select the duration of time from the **Time** drop-down menu.
- 5 Click **View** to view a pie chart with execution details of the selected pipeline.

You can click the green sector to view the pipelines successfully executed. You can also view the input parameters, status, and execution start time for each execution displayed in the table. You can click the red sector of the pie chart to determine the failed pipeline executions. The failure message containing failed stage, failed task and reason for failure is displayed in the corresponding table.

- 6 To reset your selection, click the **Clear Selection** link.

## View Failed Pipelines

You can analyze and determine the details of the top five failed tasks or five most failed tasks corresponding to top five failed stages in a release pipeline for the selected time period. You can also determine the frequency of failures.

## Procedure

- 1 Select **Code Stream > Reports**.
- 2 Select **Pipeline Failure Analysis** from the **Reports** drop-down menu.
- 3 Select the pipeline from the **Pipeline** drop-down menu.
- 4 To view the reports for the selected time period, select the duration of time from the **Time** drop-down menu.
- 5 Click **View** to view bar charts and the reasons for failure.
- 6 Click the bar representing the failed stage to view the corresponding top five failed tasks.
- 7 Click the bar representing the failed task to view the time of failure and corresponding failure messages.
- 8 To reset your selection, click the **Clear Selection** link.

# Working with the Release Dashboard

# 5

The release dashboard provides an end-to-end status summary of executions of one or more master pipelines. You can customize your selection and save your preference to view the status on subsequent logins.

The dashboard provides a summary of the pipeline details, and the status of the related machines and artifacts. You can view each artifact in the pipeline and associated pipelines to verify that the pipeline did not fail because of that artifact.

You can select one or more input parameters for the selected pipelines on the dashboard to understand the context of a specific execution. The dashboard remains private until you publish your dashboard.

To publish a dashboard and save it as a default dashboard for other users, you must have the role of Release Manager.

## Release Engineer

As a Release Engineer, you can:

- Trigger pipelines.
- Create a dashboard that appears in **My dashboard**.
- Set a default dashboard.
- Modify or delete dashboards.

## Release Manager

As a Release Manager, you can:

- Create pipelines.
- Modify and trigger pipelines.
- Create a dashboard that is displayed in **My dashboard**.
- Set a default dashboard.
- Modify or delete dashboards.
- Publish a dashboard to other users in the tenant, which appears under **Published dashboard**. You can also make the dashboard as the default dashboard for the user.

## View the Release Status

You can view the details of execution of any task from the dashboard, and then execute failed or canceled pipelines from the pipeline execution.

The dashboard displays the execution date, detailed status, overall status, execution ID, and comments of a release pipeline.

### Procedure

- 1 Select **Code Stream > Dashboard > NEW**.
- 2 Enter the **Name** of the dashboard.
- 3 Select the pipeline from the **Add pipeline to Dashboard** drop-down menu.
- 4 Click **Save**.

You can now view the dashboard and publish it to other users based on your role.

- 5 Click the status button in the **Detailed Status** column to view the task result.

Each button represents the task in a stage. If you have multiple stages, the buttons are separated by a gating rule icon.

- 6 Click **MODIFY** and select the **Publish** check box to publish the dashboard for all users to view the dashboard.

You can select multiple pipelines and add them to your dashboard using the **Add pipeline to Dashboard** option and delete the pipelines using the **REMOVE GRID** option. You can also make the published dashboard a default dashboard by selecting the **Tenant Default** check box. If you select the **Tenant Default** check box, the selected dashboard is published.

- 7 (Optional) Click the drop-down arrow ▼ in any column header to customize your view for each release pipeline.
- 8 Click **Save** to save your dashboard.

# vRealize Code Stream Execution Engine and Plug-In Framework

# 6

The vRealize Code Stream execution engine enables integrations with various external systems through a plug-in framework. The execution engine saves endpoint settings, and is used during the pipeline modeling to query external systems for available options. It is also used during pipeline executions to run commands on external systems.

The execution engine runs as a Linux service named `tekton-server` on the virtual appliance (VA).

This chapter includes the following topics:

- [Troubleshooting the Execution Engine](#)
- [Troubleshooting PowerShell Task Failures Due to Concurrent Operations](#)
- [Troubleshooting PowerShell Task Failures Due to Memory Constraints](#)
- [Troubleshooting Failed Script Tasks Due to MaxSessions Setting](#)

## Troubleshooting the Execution Engine

You can access plug-in logs, perform backups, and restore the execution engine runtime data and endpoint settings.

You can access the execution engine and plug-in logs from `/var/log/vmware/tekton`.

To back up and restore the vRealize Code Stream execution engine runtime data and endpoint settings, use the location `/storage/db/tekton/9000`.

If you cannot view the providers in the **Provider** drop-down menu, use the following procedure.

### Procedure

- 1 Verify that the execution engine is running.
- 2 On the server console, run the following commands:
  - a To check the status of the service, run the command:  

```
service tekton-server status
```
  - b To restart the service, run the command:  

```
service tekton-server restart
```

## Troubleshooting PowerShell Task Failures Due to Concurrent Operations

When concurrent operations for a user exceed the acceptable value, a PowerShell task fails.

### Problem

One or more pipelines fail when you run concurrent pipelines that exceed the maximum number.

### Cause

This problem occurs when the pipeline consists of a PowerShell task, and concurrent operations exceed the acceptable value for a user. As a result, vRealize Code Stream displays the following error: The WS-Management service cannot process the request. The maximum number of concurrent operations for this user has been exceeded. Close existing operations for this user, or raise the quota for this user.

To resolve the problem, use the following procedure.

### Solution

- 1 Modify the configuration for the service named winrm to set a higher value for MaxShellsPerUser.

The acceptable value is 500 for 50 concurrent pipelines, with 5 PowerShell tasks for each pipeline.

- 2 Run the following command:

```
winrm set winrm/config/winrs '@{MaxShellsPerUser="500"}'
```

- 3 Verify the configuration by running the following command:

```
winrm get winrm/config
```

- 4 Restart the winrm service.

## Troubleshooting PowerShell Task Failures Due to Memory Constraints

When the memory is constrained on the service named winrm, a PowerShell task might fail to respond.

### Problem

When you run concurrent pipelines that contain one or more PowerShell tasks, some pipelines might fail.

### Cause

This problem occurs when the memory is constrained on the service named winrm.

To resolve the problem, use the following procedure.

**Solution**

- 1 Modify the configuration for the service named winrm to set a higher value for MaxMemoryPerShellMB.

The acceptable value is 2048.

- 2 Run the following command:

```
winrm set winrm/config/winrs '@{MaxMemoryPerShellMB="2048"}'
```

- 3 Verify the configuration by running the following command:

```
winrm get winrm/config
```

- 4 Restart the winrm service.

## Troubleshooting Failed Script Tasks Due to MaxSessions Setting

Script tasks that run in parallel in a vRealize Code Stream pipeline on a Linux machine fail when the MaxSessions value is set too low.

**Problem**

When a pipeline is run with multiple script tasks, and you attempt to run a script on the same Linux host simultaneously, several tasks might fail with the following message: Error: channel is not open.

**Cause**

The SSH MaxSessions configuration, which defines the maximum number of open sessions allowed for a network connection, is set to 10 by default. This value limits the number of script tasks that can run simultaneously on the Linux host.

To resolve the problem, you must configure SSH MaxSessions to use a higher value. The acceptable value is 50. You calculate the value based on the number of script tasks that you expect to run simultaneously on that particular Linux host. To configure the value, follow this procedure.

**Solution**

- 1 On your Linux machine, edit the file /etc/ssh/sshd\_config.

- 2 Configure the value for MaxSessions to **50**, and save the file.

- 3 Restart the SSH service by running the following command:

```
service sshd restart
```

- 4 Verify the configured value by running the following command:

```
sshd -T | grep -i maxsessions
```

- 5 Run the pipeline again, and verify that the tasks succeed.

# vRealize Code Stream REST API and Example Scripts

# 7

You can use REST APIs to import and export release pipeline models between vRealize Code Stream appliances.

Pipelines that were previously exported from vRealize Code Stream 2.3 and 2.2 can be imported to the current version of vRealize Code Stream. You can also use the example scripts provided to run a release pipeline.

For more information on supported API, open the URL *https://<VA-IP>/component-registry/services/docs* and select **release-management** from the drop-down menu.

This chapter includes the following topics:

- [Using a REST API to Export a Release Pipeline](#)
- [Using a REST API to Import a Release Pipeline](#)
- [Java Code to Run a Release Pipeline](#)
- [Example Script to Run a Release Pipeline](#)

## Using a REST API to Export a Release Pipeline

You can export a release pipeline model, and import it later, so that users do not have to recreate an existing release pipeline.

vRealize Code Stream supports exports of pipelines from version 2.2 and greater for importing later.

When you use a REST API to export a release pipeline model, the latest version of the model is exported.

To use the vRealize Code Stream user interface to export pipelines, see [Exporting Pipelines](#).

---

**Note** You can export the release pipeline model by specifying the release pipeline name.

---

## HTTP Method

GET

## URI Syntax

```
/release-management-service/api/release-pipelines/{ReleasePipelineName}?action=export
```



## Response

The release pipeline model can be exported to an XML format by specifying the HTTP Accept Header as application/xml. JSON (application/json) is the default response format.

The password value is removed during the export process.

```
{
  "metadata": {
    "pluginInstances": {
      "instance": [

      ]
    }
  },
  "model": {
    "name": "export",
    "modelVersion": "2.1",
    "notificationList": "",
    "version": 1,
    "tags": {
      "tag": [

      ]
    },
    "pipelineParams": {
      "property": [

      ]
    },
    "stages": {
      "stage": [{
        "name": "Stage0",
        "index": 0,
        "tasks": {
          "task": [{
            "index": 0,
            "name": "sample-task",
            "plugin": {
              "provider": "vracs.jenkins:build_job",
              "category": "ALL"
            },
            "inputProperties": {
              "property": [{
                "name": "jobName",
                "type": "String",
                "value": "sample",
                "description": "jobName",
                "additional": false
              }, {
                "name": "jobParameters",
                "type": "JSON[]",
                "value": "[]"
              }
            ]
          }
        ]
      }
    ]
  }
}
```

```

        "description": "jobParameters",
        "additional": false
    }, {
        "name": "jenkinsServer",
        "type": "vracs.jenkins:JenkinsServer",
        "value": "${racks['vracs.jenkins:JenkinsServer@sample-jenkins-
endpoint']}",
        "description": "jenkinsServer",
        "additional": false
    }]
},
"outputProperties": {
    "property": []
},
"conditionalExpression": "",
"continueOnFailure": false,
"dependsOn": [
]
}]
}
}],
"concurrentExecutionsSupported": true
}
}

```

## Using a REST API to Import a Release Pipeline

You can import a release pipeline model to avoid recreating a release pipeline.

This version of vRealize Code Stream supports importing pipelines that were previously exported from version 2.2 and greater.

If the release pipeline is using a configured endpoint, configure the endpoint on the appliance before you import the release pipeline. If an endpoint is not configured on the appliance, an error message occurs.

To use the vRealize Code Stream user interface to import pipelines, see [Importing Pipelines](#).

## HTTP Method

POST

## URI Syntax

```
/release-management-service/api/release-pipelines?action=import&overwrite=false
```

The overwrite value is set to false by default, and is optional. When the overwrite value is set to true, the imported release pipeline model overwrites the existing pipeline with the same name.

## Response

The imported release pipeline model can be in the XML or JSON format depending on whether the HTTP Content-Type header is set to application/json or application/xml.

The imported pipeline is always in the draft state.

```
<release-pipeline-import-response>
  <id>da6019bf-eee2-483d-87b3-2d7752289dda</id>
  <status>rp01 import successful</status>
</release-pipeline-import-response>
```

## Existing Release Pipeline Name Response

The imported release pipeline model name exists.

```
<errors>
  <error code="18142">
    <message>Importing Release pipeline failed.</message>
    <systemMessage>The pipeline name Sample Pipeline already exists. Please rename the pipeline or
    use the overwrite option</systemMessage>
  </error>
</errors>
```

## Java Code to Run a Release Pipeline

You can use Java code to query and run release pipelines programmatically. The Java code must be packaged with open source dependencies.

## Example Java Script

```
package samples;

import com.google.gson.Gson;
import org.apache.http.Header;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.conn.ClientConnectionManager;
import org.apache.http.conn.scheme.Scheme;
import org.apache.http.conn.scheme.SchemeRegistry;
import org.apache.http.conn.ssl.SSLSocketFactory;
import org.apache.http.conn.ssl.TrustStrategy;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.impl.conn.BasicClientConnectionManager;
import org.apache.http.message.BasicHeader;
import org.apache.http.protocol.HTTP;
```

```

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import java.util.ArrayList;
import java.util.List;

/**
 * Sample Java class to invoke REST APIs on Release pipeline
 *
 */
public class PipelineApiHelper {

    private String username;
    private String password;
    private String tenant;
    private String serverHost;

    private Gson gson = new Gson();

    private static String FETCH_TOKEN_URL_TEMPLATE = "https://%s/identity/api/tokens";
    private static String TRIGGER_EXECUTION_URL_TEMPLATE = "https://%s/release-management-
service/api/release-pipelines/%s/executions";
    private static String FETCH_PIPELINE_INFO_URL_TEMPLATE = "https://%s/release-management-
service/api/release-pipelines?name=%s";

    private static ClientConnectionManager connectionManager;

    /**
     * Sample helper class to fetch pipeline info and trigger an execution
     *
     * @param serverhost Code Stream server host name
     * @param username login credentials
     * @param password login credentials
     * @param tenant tenant id for the user
     */
    public PipelineApiHelper(String serverhost, String username, String password, String tenant) {
        this.username = username;
        this.password = password;
        this.tenant = tenant;
        this.serverHost = serverhost;
    }

    /**
     * Trigger a pipeline execution
     *
     * @param pipelineName name of the pipeline to execute
     * @param pipelineParamsJson JSON string for the pipeline execution
     * pipeline param JSON is the input required for the pipeline execution.
     * for a single pipeline parameter 'token', the JSON input would look like:
     * Eg: {"description":"test run","pipelineParams":[{"name":"token","type":"STRING","value":"4321"}]}
     */
    public void triggerPipelineExecution(String pipelineName, String pipelineParamsJson) {

```

```

//A SSO token is required to make any calls to the Code Stream server. Token can be obtained easily by
passing the credentials as follows
final String token = fetchToken();
List<Header> headers = getCommonHeaders();
headers.add( new BasicHeader("Authorization", "Bearer " + token));
try {
//fetch the pipeline id
InputStream pipelineRespStream = getRequest(String.format(FETCH_PIPELINE_INFO_URL_TEMPLATE,
serverHost, pipelineName),
headers);
BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(pipelineRespStream));
//response contains the pipeline info (containing the id)
PipelineInfoWrapper pInfo = gson.fromJson(bufferedReader, PipelineInfoWrapper.class);

//trigger the pipeline execution with the pipeline id and pipeline params
InputStream is = postRequest(String.format(TRIGGER_EXECUTION_URL_TEMPLATE, serverHost,
pInfo.getContent().get(0).getId()),
headers,
pipelineParamsJson);
bufferedReader.close();
bufferedReader = new BufferedReader(new InputStreamReader(is));

String response;
System.out.println("Response:");
while ((response = bufferedReader.readLine()) != null) {
System.out.print(response);
}
System.out.println("=====");
} catch (Exception ex) {
System.out.println("Error triggering pipeline: " + ex.getMessage());
}
System.out.println("Pipeline execution triggered !");
}

/**
 * Fetch SSO Token from Identity server
 * @return token
 */
private String fetchToken() {
//POST username,password,tenant to fetch token API
TokenRequest tokenRequest = new TokenRequest();
tokenRequest.setUsername(username);
tokenRequest.setPassword(password);
tokenRequest.setTenant(tenant);
String token = null;

try {
InputStream is = postRequest(String.format(FETCH_TOKEN_URL_TEMPLATE, serverHost),
getCommonHeaders(),
gson.toJson(tokenRequest));
BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(is));
TokenResponse tokenResponse = gson.fromJson(bufferedReader, TokenResponse.class);
if (tokenResponse != null) {
token = tokenResponse.getId();
}
}

```

```

    } catch (Exception ex) {
        System.out.println("Error fetching token: " + ex.getMessage());
    }
    return token;
}

/**
 * Construct the common headers required for the API calls
 *
 * @return
 */
private List<Header> getCommonHeaders() {
    List<Header> headers = new ArrayList<Header>(){
        add(new BasicHeader(HTTP.CONTENT_TYPE, "application/json"));
        add(new BasicHeader("Accept", "application/json"));
    };

    return headers;
}

/**
 * Helper method for Http POST request
 * @param url
 * @param headers
 * @param requestBody
 * @return
 * @throws Exception
 */
private InputStream postRequest(String url, List<Header> headers, String requestBody) throws Exception
{
    HttpClient httpClient = new DefaultHttpClient(getConnManager());
    HttpPost post = new HttpPost(url);

    post.setEntity(new StringEntity(requestBody));
    Header[] headersArray = headers.toArray(new Header[headers.size()]);
    post.setHeaders(headersArray);
    HttpResponse response = httpClient.execute(post);
    if (null != response && response.getStatusLine().getStatusCode()/100 == 2) {
        return response.getEntity().getContent();
    } else {
        return null;
    }
}

/**
 * Helper method for http GET request
 * @param url
 * @param headers
 * @return
 * @throws Exception
 */
private InputStream getRequest(String url, List<Header> headers) throws Exception {
    HttpClient httpClient = new DefaultHttpClient(getConnManager());
    HttpGet get = new HttpGet(url);

```

```

Header[] headersArray = headers.toArray(new Header[headers.size()]);
get.setHeaders(headersArray);
HttpResponse response = httpClient.execute(get);
if (null != response && response.getStatusLine().getStatusCode()/100 == 2) {
return response.getEntity().getContent();
} else {
return null;
}
}

/**
 * Construct a connection manager
 * Note: This sample method ignores the SSL certificates. Ignoring them may not be something that you
intend.
 * @return
 * @throws Exception
 */
private static ClientConnectionManager getConnManager() throws Exception {
if (connectionManager == null) {
SSLSocketFactory sslSocketFactory = new SSLSocketFactory(new TrustStrategy() {
public boolean isTrusted(
final X509Certificate[] chain, String authType) throws CertificateException {
return true;
}
});
Scheme httpsScheme = new Scheme("https", 443, sslSocketFactory);
SchemeRegistry schemeRegistry = new SchemeRegistry();
schemeRegistry.register(httpsScheme);

connectionManager = new BasicClientConnectionManager(schemeRegistry);
}
return connectionManager;
}

public static void main(String[] args) {
PipelineApiHelper helper = new PipelineApiHelper("test.com", "test@test.com", "password", "sample");
helper.triggerPipelineExecution("PipelineName", "{\"id\":\"\", \"description\":\"test
run\", \"pipelineParams\": [{\"name\":\"token\", \"type\":\"STRING\", \"value\":\"4321\"}]}");
}
}

class TokenResponse {
String expires;
String id;
String tenant;

public String getId() {
return id;
}

public String getExpires() {
return expires;
}
}

```

```

public String getTenant() {
    return tenant;
}

public void setExpires(String expires) {
    this.expires = expires;
}

public void setId(String id) {
    this.id = id;
}

public void setTenant(String tenant) {
    this.tenant = tenant;
}
}

class TokenRequest {
    String username;
    String password;
    String tenant;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getTenant() {
        return tenant;
    }

    public void setTenant(String tenant) {
        this.tenant = tenant;
    }
}

class PipelineInfoWrapper {
    List<PipelineInfo> content;

    public List<PipelineInfo> getContent() {
        return content;
    }

    public void setContent(List<PipelineInfo> content) {

```



```

this.content = content;
}
}

class PipelineInfo {
String id;

public String getId() {
return id;
}

public void setId(String id) {
this.id = id;
}
}

```

## Example Script to Run a Release Pipeline

You can use the example script to query and run release pipelines programmatically. The script code has no dependencies.

### Example Script

```

#!/bin/bash

echo "#### Sample script to query and trigger a pipeline execution ####"
#Server host address refers to the host on which Code Stream server is setup. Eg: codestream.abc.com
read -p "vRealize Code Stream Server Host: " server_host

#user name and password with which you login on Code Stream server Eg: jane.doe@abc.com
read -p "Username: " username
read -p "Password: " password

#tenant name can be obtained from your system administrator if not known already
read -p "Tenant: " tenant
echo "-----"

#fetch the pipeline details and subsequently trigger an execution
#enter the pipeline name for which you want to trigger an execution
read -p "Release pipeline name:" pipeline_name

#pipeline param JSON is the input required for the pipeline execution. for a single pipeline parameter
'token', the JSON input would look like:
# Eg: {"description":"test run","pipelineParams":[{"name":"token","type":"STRING","value":"4321"}]}
read -p "Enter the pipeline param JSON:" pipeline_params

#A SSO token is required to make any calls to the Code Stream server. Token can be obtained easily by
passing the credentials as follows
host_url="https://$server_host/identity/api/tokens"
response=$(curl -s -X POST -H 'Content-Type: application/json' -H 'Accept: application/json' --
insecure -d '{"username": "'$username'", "password": "'$password'", "tenant": "'$tenant'"}'
$host_url)
#token can be extracted from the JSON response as follows

```

```

token=`echo $response | sed -n 's/.*"id":"\([^"]*\)",.*}/\1/p'`

#with the token obtained, subsequent calls can be made to the code stream server (a token has an
expiry so renewal might be required if the same token is reused beyond expiry)
pipeline_fetch_url="https://$server_host/release-management-service/api/release-pipelines?name=
$pipeline_name"
response=$(curl -s -X GET -H "Content-Type: application/json" -H "Accept: application/json" -H
"Authorization: Bearer $token" -k $pipeline_fetch_url)
pipeline_id=`echo $response | sed -n 's/.*"id":"\([^"]*\)",.*stages.*}/\1/p'`
#echo "pipeline id: $pipeline_id"

#with the pipeline id, an execution can be triggered as follows
execute_pipeline_url="https://$server_host/release-management-service/api/release-
pipelines/$pipeline_id/executions"
echo "executing pipeline:$pipeline_name :[$pipeline_id]"
response=$(curl -s -X POST -H "Content-Type: application/json" -H "Accept: application/json" -H
"Authorization: Bearer $token" -k -d "$pipeline_params" $execute_pipeline_url)
echo "Response to execute pipeline => $response"

```