Using vRealize Code Stream

vRealize Code Stream 2.0

vmware®

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

https://docs.vmware.com/

The VMware Web site also provides the latest product updates.

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

docfeedback@vmware.com

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

Contents

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.

VMware Technical Publications Glossary

VMware Technical Publications provides a glossary of terms that might be unfamiliar to you. For definitions of terms as they are used in VMware technical documentation, go to http://www.vmware.com/support/pubs.

1 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 being driven by lines of business and development operation teams. Application teams can independently use vRealize Code Stream to automate and streamline their software release process while continuing 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 where each stage is composed of multiple tasks and environments that the software has to 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.

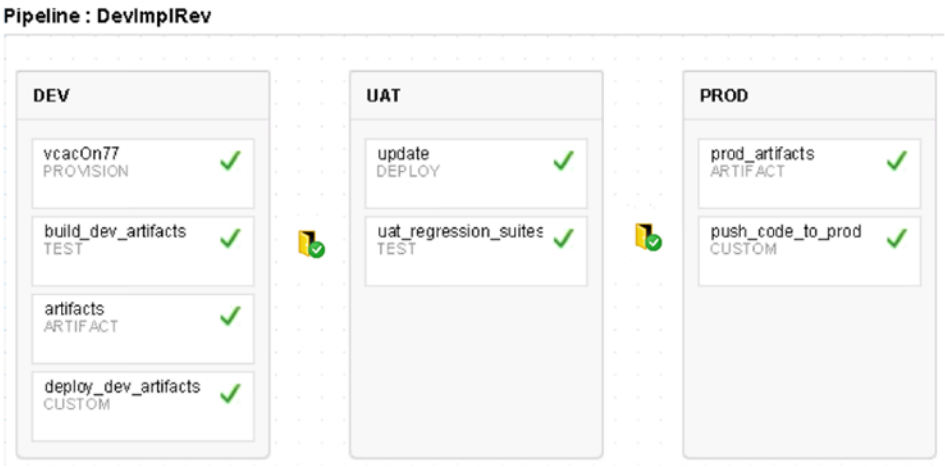
In each stage, teams might also use different kinds of development and management tools. Using different tools results in needing to build a product that is extensible and that can integrate easily with a variety of existing tools. To satisfy this need for flexibility, vRealize Code Stream offers a model-driven, open, and extensible architecture. With its catalog of plug-ins, you can integrate 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 such as provisioning a machine, retrieving an artifact, deploying software, running a test, creating a manual task, or running a custom workflow or script. The software changes are promoted to the next stage in the pipeline when they satisfy a set of rules called gating rules. 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. The environments do not need to be aware of these gating rules.

Modeling a Release Pipeline Checklist

A release pipeline is a collection of stages where each stage represents a deployment environment. For example, the development, test, user acceptance test (UAT), load test (LT), systems integration testing (SIT), and staging environments that a software change has to pass through independently before it is released.

Sample Release Pipeline with Stages



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

Modeling a Release Pipeline Template (http://bcove.me/zgy163u6).

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

1 Create a Release Pipeline on page 9

You can create, edit, view, copy, and delete the release pipeline. You can have multiple stages and build IDs for a single release pipeline.

2 Request a Machine Blueprint on page 10

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

3 Add an Artifact Task on page 11

An artifact task lets you search the library of binaries in different Artifactory repositories that you can deploy on a virtual machine.

4 Add a Custom Script Task on page 16

You can use a custom script task to run a script that resides on a remote host for any task. After the script runs, you can monitor the script progress and capture the script response, which you can pass as input to other release pipeline tasks.

5 Add a Custom Task on page 21

With a custom task, you can start a custom activity with a vRealize Orchestrator workflow. This activity can also be a manual approval.

6 Add a Custom Service Blueprint Task on page 22

A service blueprint task lets 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.

7 Configure a Team Foundation Server Task on page 25

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

8 Add a Deploy Task on page 27

A deploy task deploys or updates the existing artifacts on the machines.

9 Add a Provision Task on page 28

A provision task provisions machines. When you register a vRealize Automation, vRealize Code Stream starts a local or remote vRealize Automation 6.1. or 6.2 instance to provision infrastructure in a specific environment.

10 Add a Test Task on page 31

You can use a test task to test a deployment. When you register a Jenkins Server endpoint, you connect to a Jenkins server to start a build job from a release pipeline.

11 Configure Gating Rules on page 33

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.

12 Activate and Run a Release Pipeline on page 34

You can run a release pipeline that is activated. After you activate the release pipeline, pipeline modeling is complete and you can run it.

Create a Release Pipeline

You can create, edit, view, copy, and delete the release pipeline. You can have multiple stages and build IDs for a single release pipeline.

These pipelines form a single application or a module. You can model a pipeline with multiple stages and tasks in a stage. You can use each to depict the release cycle for an application or a module.

Plug-ins can use constant variables as part of input configurations. These variables are replaced at runtime with the values where they are placed. When users implement the plug-in spec API, it displays the ReleasePluginServiceException exception.

```
public ResponseResult execute(ExecutionContext executionContext)
```

The executionContext variable contains the getTrackingProperties() method, which is populated by default with a unique UUID combination. The combination contains the executionId of vRealize Orchestrator and the task ID that can be used in the external systems to tie their execution with the execution of the release pipeline.

The shared pipeline variable is always preceded by the constant variable. For example, ${pipeline.*PipelineVariable*}.

Prerequisites

- Verify that you have available 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 prepopulated 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 Log in to the vRealize Automation appliance.

2 Click **Add** to create a pipeline.

3 Enter a name and description for the pipeline.

4 Click **Add** to define the input properties for the pipeline.

These properties are required only if you want 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
Name	Enter a property name. The name cannot include an underscore.
Description	Enter a description for the property.
Value	Enter a property value. For example, ABC-876541.

5 (Optional) Select the check box if you do not want this pipeline to run concurrently.

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

6 Add the email addresses of recipients who receive event notifications during the pipeline run.

The email addresses are for the users who have access to the same vRealize Automation appliance.

7 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.

8 Click **Stages** to continue pipeline creation.

9 Click **Add Stage** to add multiple stages to this pipeline.

10 Double-click the default stage name and enter a name.

For example, you can create development, test, QE, and production stages.

11 (Optional) Select a stage and drag it to a different place in this pipeline.

For example, if your pipeline consists of development, test, and QE stages, you can move the test stage after the QE stage.

12 Click **Save as Draft** to save the pipeline.

13 Create tasks for every pipeline stage.

You can add multiple tasks to a stage and model tasks within a stage to run in parallel. You can drag and drop 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 are run sequentially or in parallel depending on how you have modelled the tasks within a stage. You can configure a task's input to depend on the output from an earlier task. When you add parallel tasks in a single group and configure the task's input to depend on the output from an earlier task, the variables from the output of the earlier task belonging to the previous group is always considered.

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 Select an existing pipeline to configure from the **Pipeline** tab.

3 Select **Edit** > **Stages**.

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 an Artifact Task

An artifact task lets you search the library of binaries in different Artifactory repositories that you can deploy on a virtual machine.

When you include an artifact task in a 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, which includes a repository name, a download URL, and size information, if available.

Prerequisites

- Verify that a pipeline is available. See "Create a Release Pipeline," on page 9.
- Verify that an Artifactory server is registered. See the *Installation and Configuration* guide.

Procedure

1 Click the **Code Stream** tab.

2 Select an existing pipeline to configure from the **Pipeline** tab.

3 Select **Edit** > **Stages**.

4 Select **Add Task**.

5 Select **Artifact** from the **Category** drop-down menu.

6 Select the **VMware Repository Solution** from the **Provider** drop-down menu.

7 Enter an artifact task name and click **OK**.

8 Select the new task from the stage column.

9 Confirm to save the pipeline.

10 Select an Execute Task for the release pipeline.

Option	Description
Always	Runs the release pipeline task without conditions.
On Condition(s)	Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped. A boolean expression using the following operands and operators is supported. ■ Pipeline variables such as, ${pipeline.*variableName*}. Curly brackets are reserved for specifying pipeline variables. ■ Task output variables such as, ${Stage1.task1.machines.value.hostIp} ■ Default pipeline constant variables such as, ${releasePipelineName} ■ Case insensitive Boolean values such as, true, false, 'true', 'false' ■ Integer or decimal values without quotes ■ String values used with single or double quotes such as, "test", 'test' ■ String and Numeric types of values such as, == Equals and != Not Equals ■ Relational operators such as, > , >=, <, and <= ■ Boolean logic such as, && and || ■ Arithmetic operators such as, +, -, *, and / ■ Nested expressions using round brackets ■ Strings with literal value ABCD is evaluated as false and the task is skipped. Unary operators are not supported. A sample condition, ${Stage1.task1.output} == “Passed” || ${pipeline.*variableName*} == 39

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 Artifcatory repository.

Prerequisites

Verify that you have an existing artifact task. See “Add an Artifact Task,” on page 11.

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
gavc	Provide a group ID, artifact ID, version ID, or classifier parameters for the search on local repositories. **NOTE** On remote repositories the search requires a group ID, artifact ID, and version ID. You can not use the wild card character asterisk (*) as a search parameter. ■ The group ID is the organization that published the artifact. For example, org.springframework. ■ The artifact ID is the identifier of the artifact. For example, travel. ■ The version ID parameter supports the LATEST and version-SNAPSHOT keywords. For example, enter **`4.1-SNAPSHOT`** to get the snapshot of a version. The classifier is optional and supports the wild card character asterisk (*). For example, release*, *release*, or *. When you start a gavc search type from an Artifactory repository, the search result displays all of the files with the same artifact name and different extensions. You can narrow the search results with the pattern search type instead.
pattern	Provide the artifact name or path parameters for the search on local repositories. **NOTE** On remote repositories the search requires the complete artifact path. The artifact name is optional. ■ The name parameter supports the wild card character asterisk (*). For example, test*, test.*, *.jar, or *. ■ The path parameter supports the wild card character asterisk (*) for local repositories. For example, path/*/release searches under the path directory in the /test/release and /dev/release folders for the artifacts only in those folders. Each wildcard (*) represents one level in the folder structure.

Search Type	Description and Sample Search Parameters
properties	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. For example, the **Property** field can be artifactory.licenses and the **Value** field can be BSD . **Note** The properties search function does not work on remote repositories. You can cache artifacts or locally tag the artifacts to use this search function.
build	You can not create in the Artifactory user interface. You must push a build from your CI server to the Artifactory user interface. ■ The build Name is the name of the job that is run on your CI system. For example, Jenkins-release. ■ The build Number supports the LATEST or a build status keyword . For example, you can search for builds with the Prod status. ■ The name lets you filter an artifact from the list of artifacts in a specific build. For example, public-API.jar or public-*.jar **Note** The build search function does not work on remote repositories.

5 Enter a temporary value in the **Value** text box if you use a variable as a search parameter and click **Search**.

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

For example, for a gavc search type, if you use ${LATEST} as the version variable, which could be different when you run the pipeline, you must add the current version, such as 2.1 in the temporary value .

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 Click **Validate** to verify that the artifact exists in the repository.

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 Select one or more artifact output properties to expose to other tasks in the pipeline.

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 an Artifact Task," on page 11.

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
gavc	Provide a group ID, artifact ID, version ID, or classifier parameters for the search on local repositories. **Note** On remote repositories the search requires a group ID, artifact ID, and version ID. You can not use the wild card character asterisk (*) as a search parameter. ■ The group ID is the organization that published the artifact. For example, org.springframework. ■ The artifact ID is the identifier of the artifact. For example, travel. ■ The version ID parameter supports the LATEST and version-SNAPSHOT keywords. For example, enter **`4.1-SNAPSHOT`** to get the snapshot of a version. The classifier is optional and supports the wild card character asterisk (*). For example, release*, *release*, or *. When you start a gavc search type from an Artifactory repository, the search result displays all of the files with the same artifact name and different extensions. You can narrow the search results with the pattern search type instead.
pattern	Provide the artifact name or path parameters for the search on local repositories. **Note** On remote repositories the search requires the complete artifact path. The artifact name is optional. ■ The name parameter supports the wild card character asterisk (*). For example, test*, test.*, *.jar, or *. ■ The path parameter supports the wild card character asterisk (*) for local repositories. For example, path/*/release searches under the path directory in the /test/release and /dev/release folders for the artifacts only in those folders. Each wildcard (*) represents one level in the folder structure.

Search Type	Description and Sample Search Parameters
properties	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. For example, the **Property** field can be artifactory.licenses and the **Value** field can be BSD . **Note** The properties search function does not work on remote repositories. You can cache artifacts or locally tag the artifacts to use this search function.
build	You can not create in the Artifactory user interface. You must push a build from your CI server to the Artifactory user interface. ■ The build Name is the name of the job that is run on your CI system. For example, Jenkins-release. ■ The build Number supports the LATEST or a build status keyword . For example, you can search for builds with the Prod status. ■ The name lets you filter an artifact from the list of artifacts in a specific build. For example, public-API.jar or public-*.jar **Note** The build search function does not work on remote repositories.

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

6 Click **Validate** to verify that the artifact exists in the repository.

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 Select one or more artifact output properties to expose to other tasks in the pipeline.

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

8 Click **Save**.

Add a Custom Script Task

You can use a custom script task to run a script that resides on a remote host for any task. After the script runs, you can monitor the script progress and capture the script response, which you can pass as input to other release pipeline tasks.

Prerequisites

Verify that you defined and configured the plug-ins. An instance of this plug-in must be created in vRealize Orchestrator to enable the plug-ins in the pipeline. See the *Installation and Configuration* guide.

Procedure

1 Click the **Code Stream** tab.

2 Select an existing pipeline to configure from the **Pipeline** tab.

3 Select **Edit** > **Stages**.

4 Select **Add Task**.

5 Select **Custom** from the **Category** drop-down menu.

6 Select **Custom Script** from the **Provider** drop-down menu.

7 Enter a name for this custom task and click **OK.**

8 Select the new task from the stage column.

9 Confirm to save the pipeline.

10 Select an Execute Task for the release pipeline.

Option	Description
Always	Runs the release pipeline task without conditions.
On Condition(s)	Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped. A boolean expression using the following operands and operators is supported. ■ Pipeline variables such as, ${pipeline.*variableName*}. Curly brackets are reserved for specifying pipeline variables. ■ Task output variables such as, ${Stage1.task1.machines.value.hostIp} ■ Default pipeline constant variables such as, ${releasePipelineName} ■ Case insensitive Boolean values such as, true, false, 'true', 'false' ■ Integer or decimal values without quotes ■ String values used with single or double quotes such as, "test", 'test' ■ String and Numeric types of values such as, == Equals and != Not Equals ■ Relational operators such as, > , >=, <, and <= ■ Boolean logic such as, && and || ■ Arithmetic operators such as, +, -, *, and / ■ Nested expressions using round brackets ■ Strings with literal value ABCD is evaluated as false and the task is skipped. Unary operators are not supported. A sample condition, ${Stage1.task1.output} == “Passed” || ${pipeline.*variableName*} == 39

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.

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 should 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. The status variable is set by default to be shared by all of the tasks in the release pipeline. When these variables need to 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. The script must have executable permission for a remote user to run it.
- Verify that the folder where the script exists has permission to allow a file to be created.
- Familiarize yourself with using task output variables. See "Create a Release Pipeline," on page 9.

Procedure

1 Open a release pipeline.

2 Select the new task from the stage column.

3 Confirm to save the pipeline.

4 Select an Execute on Host type and add the applicable details.

Option	Description
User defined hosts	Enter one or more IP Addresses, DNS hostname, or release pipeline input variables as one of the valid hosts. For example, the input variable machine_ip can be 198.51.100.13. The user defined host can also have $machine_ip with static hostname and IP Address.
Read From Pipeline or Task Property	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. The pipeline variables are specified on the page before the modeling page. The run time values are applied when the release pipeline runs. For example, the task output variable ${*StageName*.*TaskName*.status.outputConfig.host} is bound and shared by all of the hosts. The status variable is set by default to be shared by all of the tasks in the release pipeline. To differentiate between a pipeline variable and a task output variable, the pipeline variable is always preceded by the constant variable. For example, ${pipeline.*PipelineVariable*}.
Filter Criteria	Set one of the following criteria from the drop-down menu. ■ ALL HOSTS. Run the script on all of the hosts that the host group variable gives. ■ STARTS WITH. From the host group, filter machines whose names start with the prefix as given by the user. The filter search criteria is case sensitive. ■ ENDS WITH. Filter machines whose names end with the prefix as given by the user. ■ EXACT MATCH. Run the script only on the machine whose name exactly matches the DNS hostname or IP Address.

5 Type the host user name and password for a remote connection to the host.

6 Specify the script file name and the path that already resides on the host.

An example file path for a Bash script is `/home/test/template/test.sh`.

7 Select a **Bash** or **PowerShell** script from the drop-down menu.

8 Configure the command-line arguments.

The command-line arguments can be variables or constant 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.

9 (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.

10 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 to enable the script to 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 the artifact output property is shared. See "Add an Artifact Task," on page 11.

Procedure

1 Open a terminal.

2 Download artifacts within the script task.

```
wget $artifactDownloadUrl
```

The $artifactDownloadUrl script variable is mapped to the artifact.

3 (Optional) Bypass the certificate check for a self-signed SSL certificate.

- `wget $artifactDownloadUrl --no-check-certificate`
- `curl -O $artifactDownloadUrl --insecure`

4 Close the terminal.

5 Open a release pipeline.

6 Select the script task from the stage column.

7 Click the **Advanced** tab.

8 Type the artifact group variable.

${*StageName*.*TaskName*.ARTIFACT_OUTPUT}

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

An example of an artifact group variable, ${Dev.Artifact.ARTIFACT_OUTPUT}.

9 Click **Add** to define the Artifact input parameters.

Option	Description
Parameter Name	Name of the Shell variable referring to an artifact in the script. An example parameter name is artifact_config.
Parameter Value	Artifact name defined in the artifact group. An example parameter value is CodeStream_JAR.

10 Select **Download URL** or **Repository URL** from the drop-down menu.

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

11 Click **Add** to define the Other input.

Option	Description
Parameter Name	Name of the other Shell variable defined in the script. An example parameter name is portNumber or buidID.
Parameter Value	Value for the Shell variable. An example parameter name is 800.

12 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: $APLICATION_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 VCAC_APP_DOWNLOAD_URL environment variable to determine what version of the VCAC artifact to download from the repository. The artifact input parameter to the script should be the VCAC_APP_DOWNLOAD_URL parameter.

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

For the script to be able to share data with 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 Custom Task

With a custom task, you can start a custom activity with a vRealize Orchestrator workflow. This activity can also be a manual approval.

Prerequisites

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

Procedure

1 Click the **Code Stream** tab.

2 Select an existing pipeline to configure from the **Pipeline** tab.

3 Select **Edit** > **Stages**.

4 Select **Add Task**.

5 Select **Custom** from the **Category** drop-down menu.

6 Select **Custom Workflow** from the **Provider** drop-down menu.

7 Enter a name for the custom task and click **OK**.

8 Select the new task from the stage column.

9 Confirm to save the pipeline.

10 Select an Execute Task for the release pipeline.

Option	Description
Always	Runs the release pipeline task without conditions.
On Condition(s)	Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped. A boolean expression using the following operands and operators is supported. ■ Pipeline variables such as, ${pipeline.*variableName*}. Curly brackets are reserved for specifying pipeline variables. ■ Task output variables such as, ${Stage1.task1.machines.value.hostIp} ■ Default pipeline constant variables such as, ${releasePipelineName} ■ Case insensitive Boolean values such as, true, false, 'true', 'false' ■ Integer or decimal values without quotes ■ String values used with single or double quotes such as, "test", 'test' ■ String and Numeric types of values such as, == Equals and != Not Equals ■ Relational operators such as, > , >=, <, and <= ■ Boolean logic such as, && and || ■ Arithmetic operators such as, +, -, *, and / ■ Nested expressions using round brackets ■ Strings with literal value ABCD is evaluated as false and the task is skipped. Unary operators are not supported. A sample condition, ${Stage1.task1.output} == “Passed” || ${pipeline.*variableName*} == 39

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 workflow name from the drop-down menu.

13 Enter the input parameters.

For example, enter **`vsphere.local\administrators`** for the Task Group DN Name and enter the task details.

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

15 Click **Save**.

Example: Task Output Format for a Custom Workflow Task

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

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

Add a Custom Service Blueprint Task

A service blueprint task lets 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 Select an existing pipeline to configure from the **Pipeline** tab.

3 Select **Edit** > **Stages**.

4 Select **Add Task**.

5 Select **Custom** from the **Category** drop-down menu.

You can also select the Deploy, Provision, or Test categories for the service blueprint task.

6 Select **vRA Service Blueprints** from the **Provider** drop-down menu.

7 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.

8 Enter a name for the custom task and click **OK.**

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 from the stage column.

3 Confirm to save the pipeline.

4 Select an Execute Task for the release pipeline.

Option	Description
Always	Runs the release pipeline task without conditions.
On Condition(s)	Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped. A boolean expression using the following operands and operators is supported. ■ Pipeline variables such as, ${pipeline.*variableName*}. Curly brackets are reserved for specifying pipeline variables. ■ Task output variables such as, ${Stage1.task1.machines.value.hostIp} ■ Default pipeline constant variables such as, ${releasePipelineName} ■ Case insensitive Boolean values such as, true, false, 'true', 'false' ■ Integer or decimal values without quotes ■ String values used with single or double quotes such as, "test", 'test' ■ String and Numeric types of values such as, == Equals and != Not Equals ■ Relational operators such as, > , >=, <, and <= ■ Boolean logic such as, && and || ■ Arithmetic operators such as, +, -, *, and / ■ Nested expressions using round brackets ■ Strings with literal value ABCD is evaluated as false and the task is skipped. Unary operators are not supported. A sample condition, ${Stage1.task1.output} == “Passed” || ${pipeline.*variableName*} == 39

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
- Hiera 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"
  }
]
```


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. Select an existing pipeline to configure from the **Pipeline** tab.
3. Select **Edit** > **Stages**.
4. Select **Add Task**.
5. Select **Custom** from the **Category** drop-down menu.
6. Select **TFS** from the **Provider** drop-down menu.
7. Select the Team Foundation Server endpoint you registered from the drop-down menu.
8. Enter a name for the custom task and click **OK.**

9 Select the new task from the stage column.

10 Confirm to save the pipeline.

11 Select an Execute Task for the release pipeline.

Option	Description
Always	Runs the release pipeline task without conditions.
On Condition(s)	Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped. A boolean expression using the following operands and operators is supported. ■ Pipeline variables such as, ${pipeline.*variableName*}. Curly brackets are reserved for specifying pipeline variables. ■ Task output variables such as, ${Stage1.task1.machines.value.hostIp} ■ Default pipeline constant variables such as, ${releasePipelineName} ■ Case insensitive Boolean values such as, true, false, 'true', 'false' ■ Integer or decimal values without quotes ■ String values used with single or double quotes such as, "test", 'test' ■ String and Numeric types of values such as, == Equals and != Not Equals ■ Relational operators such as, > , >=, <, and <= ■ Boolean logic such as, && and || ■ Arithmetic operators such as, +, -, *, and / ■ Nested expressions using round brackets ■ Strings with literal value ABCD is evaluated as false and the task is skipped. Unary operators are not supported. A sample condition, ${Stage1.task1.output} == “Passed” || ${pipeline.*variableName*} == 39

12 (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.

13 Configure the Team Foundation Server properties.

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

14 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:///Buil
d/Build/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:///Buil
d/Build/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 Deploy Task

A deploy task deploys or updates the existing artifacts on the machines.

For example, if the plug-in provider for this task is a script, then the script can be a Bash or PowerShell script.

Prerequisites

Verify that you have a valid script available.

Procedure

1 Click the **Code Stream** tab.

2 Select an existing pipeline to configure from the **Pipeline** tab.

3 Select **Edit** > **Stages**.

4 Select **Add Task**.

5 Select **Deploy** from the **Category** drop-down menu.

6 Select **Custom Script** from the **Provider** drop-down menu.

7 Enter a name for the deploy task and click **OK**.

8 Select the new task from the stage column.

9 Confirm to save the pipeline.

10 Select an Execute Task for the release pipeline.

Option	Description
Always	Runs the release pipeline task without conditions.
On Condition(s)	Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped. A boolean expression using the following operands and operators is supported. ■ Pipeline variables such as, ${pipeline.*variableName*}. Curly brackets are reserved for specifying pipeline variables. ■ Task output variables such as, ${Stage1.task1.machines.value.hostIp} ■ Default pipeline constant variables such as, ${releasePipelineName} ■ Case insensitive Boolean values such as, true, false, 'true', 'false' ■ Integer or decimal values without quotes ■ String values used with single or double quotes such as, "test", 'test' ■ String and Numeric types of values such as, == Equals and != Not Equals ■ Relational operators such as, > , >=, <, and <= ■ Boolean logic such as, && and || ■ Arithmetic operators such as, +, -, *, and / ■ Nested expressions using round brackets ■ Strings with literal value ABCD is evaluated as false and the task is skipped. Unary operators are not supported. A sample condition, ${Stage1.task1.output} == “Passed” || ${pipeline.*variableName*} == 39

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.

Add a Provision Task

A provision task provisions machines. When you register a vRealize Automation, vRealize Code Stream starts a local or remote vRealize Automation 6.1. or 6.2 instance 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.
  - 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 Select an existing pipeline to configure from the **Pipeline** tab.

3 Select **Edit** > **Stages**.

4 Select **Add Task**.

5 Select **Provision** from the **Category** drop-down menu.

6 Select **vRA Provisioning** from the **Provider** drop-down menu.

7 Select from the drop-down menu the vRealize Automation server endpoint you registered.

8 Enter a name for the provision task and click **OK**.

9 Select the new task from the stage column.

10 Confirm to save the pipeline.

11 Select an Execute Task for the release pipeline.

Option	Description
Always	Runs the release pipeline task without conditions.
On Condition(s)	Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped. A boolean expression using the following operands and operators is supported. ■ Pipeline variables such as, ${pipeline.*variableName*}. Curly brackets are reserved for specifying pipeline variables. ■ Task output variables such as, ${Stage1.task1.machines.value.hostIp} ■ Default pipeline constant variables such as, ${releasePipelineName} ■ Case insensitive Boolean values such as, true, false, 'true', 'false' ■ Integer or decimal values without quotes ■ String values used with single or double quotes such as, "test", 'test' ■ String and Numeric types of values such as, == Equals and != Not Equals ■ Relational operators such as, > , >=, <, and <= ■ Boolean logic such as, && and || ■ Arithmetic operators such as, +, -, *, and / ■ Nested expressions using round brackets ■ Strings with literal value ABCD is evaluated as false and the task is skipped. Unary operators are not supported. A sample condition, ${Stage1.task1.output} == “Passed” || ${pipeline.*variableName*} == 39

12 (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.

13 Enter the machine properties.

Option	Description
Blueprint	Select the published vRealize Automation blueprint from the drop-down menu.
Number of Machines	Enter the number of machines to be provisioned and the configuration details.
(Optional) Configuration Details	Enter the Lease (Days), Number of CPUs, Memory, Storage values, and Description. **Note** An existing machine blueprint configuration can overwrite the values specified here.
Owner	Select **Logged In User** to allow the user currently logged in to run the release pipeline. Select **Enter Owner or Bind** 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.

14 (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.

15 Click **Save**.

Example: Task Output Format for a Provisioning 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"
    }
]
```

Add a Test Task

You can use a test task to test a deployment. When you register a Jenkins Server endpoint, you connect to a Jenkins server to start a build job from a release pipeline.

Note If you have existing artifacts in the Jenkins server you cannot access them from vRealize Code Stream.

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

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.

Procedure

1 Click the **Code Stream** tab.

2 Select an existing pipeline to configure from the **Pipeline** tab.

3 Select **Edit** > **Stages**.

4 Select **Add Task**.

5 Select **Test** from the **Category**drop-down menu.

6 Select **Jenkins** from the **Provider** drop-down menu.

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

8 Enter a name for the test task and click **OK**.

9 Select the new task from the stage column.

10 Confirm to save the pipeline.

11 Select an Execute Task for the release pipeline.

Option	Description
Always	Runs the release pipeline task without conditions.
On Condition(s)	Runs the release pipeline task only if the defined condition is evaluated as true. If the condition is false, the task is skipped. A boolean expression using the following operands and operators is supported. ■ Pipeline variables such as, ${pipeline.*variableName*}. Curly brackets are reserved for specifying pipeline variables. ■ Task output variables such as, ${Stage1.task1.machines.value.hostIp} ■ Default pipeline constant variables such as, ${releasePipelineName} ■ Case insensitive Boolean values such as, true, false, 'true', 'false' ■ Integer or decimal values without quotes ■ String values used with single or double quotes such as, "test", 'test' ■ String and Numeric types of values such as, == Equals and != Not Equals ■ Relational operators such as, > , >=, <, and <= ■ Boolean logic such as, && and || ■ Arithmetic operators such as, +, -, *, and / ■ Nested expressions using round brackets ■ Strings with literal value ABCD is evaluated as false and the task is skipped. Unary operators are not supported. A sample condition, ${Stage1.task1.output} == “Passed” || ${pipeline.*variableName*} == 39

12 (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.

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

14 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*}.

15 (Optional) Select one or more Jenkins job output properties to expose to other tasks in the pipeline.

For example, you can expose the Jenkins job name and the test results in a custom task.

16 Click **Save**.

Example: Output Format for a Test 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 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.

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 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.

Table 1-1. Default Workflows

Workflow	Description
Approval	This workflow is tagged with a 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 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 Select an existing pipeline to configure from the **Pipeline** tab.

3 Select **Edit** > **Stages**.

4 Click the Gating Rule icon () next to a stage.

5 Select **If outcome of a vRealize Orchestrator workflow is successful** to create the gating rule.

6 Select the appropriate workflow from the drop-down menu.

  a 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.

  b Enter the threshold percentage and the test result value in the **Test Acceptance Threshold** text box.

7 Click **Save**.

Activate and Run a Release Pipeline

You can run a release pipeline that is activated. After you activate the release pipeline, pipeline modeling is complete and you can run it.

You can view a release pipeline run on the **Pipeline Executions** tab. The draft state signifies that the Release Manager is still modeling the release pipeline.

Multilevel information is provided during the release pipeline run.

- Level 1.Status at release pipeline level which is the description of the current activity.
- Level 2. Status at the task or stage level that displays the current task progress and the executed task status.
- Level 3. Provides detailed information at task level.

Prerequisites

- Verify that a release pipeline is created in a draft state. See "Create a Release Pipeline," on page 9.
- Verify that predefined vRealize Automation blueprints, workflows, and scripts are created, and test jobs that perform tasks that the pipeline can trigger.
- Verify that artifacts are available in the Artifactory server repository to use the vRealize Code Stream Artifact Management capabilities.

Procedure

1 Click the **Code Stream** tab.

2 Select an existing pipeline to configure from the **Pipeline** tab.

3 Click **Activate** to create a vRealize Orchestrator workflow.

Every time a pipeline is activated, a new version of the pipeline is created.

A list of the available pipelines appears.

4 Select the pipeline.

5 Click **Execute**.

The Execute Pipeline dialog box appears.

6 Enter the description and properties.

7 Click **OK** to run a release 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 cancelled, but the configured build in the TFS server will still display the status of the pipeline as completed.

After a custom script task runs, the EXECUTION_ID_FOLDER run folder is created in the *Script_Name*_RUN folder. This folder contains the logFile, responseFile, exitStatusFile files, and a clone of the actual script that is being run. For example, `script_params.txt`, `script_log.txt`, `script_response.txt`.

Table 1-2. Run Folder Locations

Run Folder	Location
Bash script	/var/tmp/*Script_Name*_RUN/EXECUTION_ID_FOLDER
Powershell Script	C:\Users*User_Name*\AppData\Local\Temp*Script_Name*_RUN/EXECUTION_ID_FOLDER

What to do next

You can track the status of the pipeline and troubleshoot failed pipelines. See Chapter 4, "Using the Experimental Release Dashboard Feature," on page 41.

Managing Release Automation 2

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

This chapter includes the following topics:

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," on page 9.

Procedure

1 Click the **Code Stream** tab.

2 Select an existing pipeline to configure from the **Pipeline** tab.

3 Select **Edit** > **Stages**.

4 Click the gear icon ( ) next to the task in a stage.

5 Click **Clone**.

A clone task appears in the stage.

6 Drag the cloned task to another stage or place it in the 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," on page 9.

Procedure

1 Click the **Code Stream** tab.

2 Select an existing pipeline to configure from the **Pipeline** tab.

3 Select **Edit** > **Stages**.

4 Click the gear icon ( ) next to the task in a 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," on page 9.

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," on page 9.

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.

Viewing Pipeline Execution Reports 3

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

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 pipelines from the **Pipeline** drop-down menu.

4 Select the duration of time from the **Time** drop-down menu to view the reports for the selected time period.

5 Click **View** to view a pie chart with the details of the executed pipelines .

You can click the green sector to view the successfully completed pipelines. 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 failed stage and task along with the failure message is also displayed in the corresponding table.

View Failed Pipelines

You can analyze and determine the details of the top 5 most failed tasks corresponding to top 5 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 duration of time from the **Time** drop-down menu to view the reports for the selected time period.

4 Click **View** to view bar charts and the reasons for failure .

5 Click the bar representing the failed stage to view the corresponding top 5 failed tasks.

6 Click the bar representing the failed task to view the time of failure and corresponding failure messages.

7 Click the **Clear Selection** link to refresh your selection .

Using the Experimental Release Dashboard Feature

4

When a release pipeline execution is complete, the data for the artifacts, machines, and pipelines is available in the release dashboard, which is an experimental feature. The release dashboard home page summarizes the total number of artifacts, pipelines, and machines. It also lists the number of successful, in progress, or failed pipelines.

As a Release Dashboard user you can view artifacts, machines, and pipelines, and track the cross dependencies between these resource types to facilitate troubleshooting when deployments fail or exceed the expected deployment time threshold.

The Artifacts resource section lists all of the artifacts that the executed pipelines deploy. When you select an artifact, the associated details of the artifact and the details of the parent pipelines' execution instances that deployed this artifact instance appear. The Machines resource section uses the vRealize Automation machine provisioning to list all of the machines that are deployed through an executed pipeline. The Pipeline resource section lists each instance of a successful and failed pipeline execution.

On the right side of the home page, you can view the recently updated resources, a summary of the pipeline details, and the status of the related machines and artifacts.

You can navigate to the Pipelines page and click the failed pipeline or navigate to the Machines page to identify the related machines that might have caused the deployment to fail. A Release Dashboard user can view each artifact in the pipeline and associated pipelines to verify that the pipeline did not fail because of that artifact.

Troubleshooting Release Pipeline Failures

vRealize Code Stream provides logs to aid in troubleshooting. You can retrieve these logs from the user interface, perform a comparison between a successful and failed pipeline instance, and correct some of the problems.

If you are asked for log files while you are troubleshooting a failed pipeline instance, you can retrieve them from the file system of the vRealize Automation appliance, a remote machine, or a local machine.

When the log storage is full, the pipelines fail to continue with the execution of tasks. In such a scenario, delete the log files or move the logs to another location for future reference.

Compare a Pipeline Execution

You can open a failed pipeline execution in one column and a successful pipeline execution in the adjoining column in the comparison view and expand the pipeline instance details to identify the component that might have caused the release pipeline deployment to fail.

Prerequisites

- Log in to the vRealize Code Stream console as a **release dashboard user**, **release manager**, or **release engineer**.
- Verify that the parent release pipeline includes multiple instances of a completed pipeline execution.

Procedure

1. Select **Code Stream > Pipeline Executions**.
2. Locate a parent release pipeline and click **Compare**.

   The compare view opens on the **Release Dashboard** tab.
3. Click in the **Compare With** text box and select a pipeline execution instance from the drop-down menu.

   A list of all dates and times of when the selected pipeline instance was run appears.
4. View the various elements that were changed, deleted, or added between the two pipeline execution instances.

   Modified elements are highlighted in yellow. Deleted elements are highlighted in red. New elements added are highlighted in green.
5. Identify the element marked as failed during the pipeline execution.
6. Compare the failed element with the element that passed in the adjoining instance.

   This comparison provides details about the failure and how the problem was resolved.

Assign a File Server Repository for Release Pipeline Log Files

You can store log files in a file server repository and retrieve them from the vRealize Code Stream user interface to troubleshoot a release pipeline failure.

Important You cannot configure a remote file server repository.

Prerequisites

- Log in to the vRealize Automation console as a **system administrator** or **release manager**.
- Verify that you have a local file server repository configured.

Procedure

1. Select **Administration > Artifact Management**.
2. Select a local file server repository from the drop-down menu to store the release pipeline log files.
3. Click **Test Connection** to verify that vRealize Code Stream is connected to the file server.

Collect Log Files from a Remote Machine

You can retrieve log files from a remote machine to troubleshoot a failure in a release pipeline.

Prerequisites

- Log in to the vRealize Automation console as a **release dashboard user**.

- Verify that a remote file server is configured. See "Assign a File Server Repository for Release Pipeline Log Files," on page 42.

Procedure

1 Select **Code Stream > Pipeline Executions**.

2 Locate the failed release pipeline.

3 Expand the failed release pipeline to view the details.

4 Click the troubleshoot icon ().

The troubleshooting view opens.

5 Expand the Input and Output sections to view the details of the failed task.

6 Click **Add Log** to upload the log files from the remote machine by using SSH.

7 Enter the remote machine information.

Option	Description
Host Name	Name of the remote machine.
User credentials	Enter the user name and password.
Path	Specify the log file path in the remote machine. To retrieve all of the logs matching a file extension, add the wild card character asterisk (*) to the file name. For example, /path/test/*.jar.

8 Click **Upload** to retrieve the logs.

9 Click **Refresh** to view the status of the log retrieval.

If the log fails to upload, verify that the file server is accurate and add the log again.

The log file appears in the Logs section.

Collect Logs Files from a Local Machine

You can retrieve log files from your local machine to troubleshoot a failure in a release pipeline.

Prerequisites

- Log in to the vRealize Automation console as a **release dashboard user**.
- Verify that the log file size is less than 20 MB.

Procedure

1 Select **Code Stream > Pipeline Executions**.

2 Locate the failed release pipeline.

3 Expand the failed release pipeline to view the details.

4 Click the troubleshoot icon ().

The troubleshooting view opens.

5 Expand the Input and Output sections to view the details of the failed task.

6 Click **Add Log** to upload the logs from the local machine.

7 Select to upload from your local machine.

8 Click **Browse** to locate the log file.

9 Click **Upload** to retrieve the logs.

10 Click **Refresh** to view the status of the log retrieval.

If the log fails to upload, verify that the file server is accurate and add the log again.

The log file appears in the Logs section.

Collect Logs from the vRealize Automation Appliance

You can access the `catalina.out` log file from the vRealize Automation appliance.

Prerequisites

Verify that you have credentials for logging in to the Linux-based virtual machine with root privileges.

Procedure

1 Log in to the virtual machine.

2 Navigate to the `/var/lib/vcac/server/logs` directory.

3 Retrieve the logs from the `catalina.out` file and troubleshoot the problem.

vRealize Code Stream Execution Engine and Plug-in Framework

5

The vRealize Code Stream execution engine enables integrations with various external systems through a plug-in framework.

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

Troubleshooting the Execution Engine

During modelling of a pipeline task, you can access execution engine, plug-in logs and runtime data.

You can access the execution engine and plug-in logs from `/var/log/vmware/tekton`. If you want to backup and restore vRealize Code Stream runtime data, use the location `/storage/db/tekton/9000`.

If you are unable to view the providers in the **Provider** drop-down menu, perform the following:

Procedure

1 Ensure that the execution engine is running.

2 Execute the following on the console of the server:

   a `service tekton-server status` to check the status of the service.

   b `service tekton-server restart` to restart the service.

vRealize Code Stream REST API and Example Scripts

6

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

You can also use the provided example scripts to run a release pipeline.

This chapter includes the following topics:

Exporting a Release Pipeline

You can export a release pipeline model between vRealize Code Stream 2.0 appliances so that users do not have to recreate an existing release pipeline.

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

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.

```
{
     "modelVersion": 1,
     "metadata": {
          "pluginInstances": {
               "instance": []
          }
```

```
        },
        "model": {
             "name": "Sample Pipeline",
             "notificationList": "xyz@company.com",
             "version": 1,
             "tags": {
                  "tag": ["abc"]
             },
             "pipelineParams": {
                  "property": []
             },
             "stages": {
                  "stage": [{
                       "name": "DEV",
                       "index": 0,
                       "tasks": {
                            "task": [{
                                 "index": 0,
                                 "name": "Test task",
                                 "plugin": {
                                      "provider": "vrcs.jenkins:build_job",
                                      "category": "TEST",
                                      "instanceId": 1
                                 },
                                 "inputProperties": {
                                      "property": [
                                           {
                                                     "name": "jenkinsServer",
                                                     "type": "vrcs.jenkins:JenkinsServer",
                                                     "value": "$
{racks['vrcs.jenkins:JenkinsServer@sample_jenkins_endpoint']}",
                                                     "description": "jenkinsServer",
                                                     "additional": false
                                           }
                                      ]
                                 },
                                 "outputProperties": {
                                      "property": []
                                 },
                                 "conditionalExpression": "",
                                 "continueOnFailure": false,
                                 "dependsOn": []
                            }]
                       }
                  }]
             },
             "concurrentExecutionsSupported": true
        }
   }
```

Import a Release Pipeline

You can import a release pipeline model between vRealize Code Stream 2.0 appliances to avoid recreating a release pipeline.

When you use the REST API to upgrade from vRealize Code Stream 1.0 to 2.0, the release pipeline models are migrated to the latest release version.

If the release pipeline is using configured endpoint, configured them on the appliance before you import the release pipeline. If an endpoint is not configured on the appliance, you receive an error message.

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>
```

Missing Endpoint Response

The endpoint is not configured for the imported release pipeline model.

```
<errors>
    <error code="18142">
        <message>Importing Release pipeline failed.</message>
        <systemMessage>Instance Jenkins_Sample of plugin RPTestJenkins not found. Please create
the plugin instance in the associated vCO</systemMessage>
    </error>
</errors>
```

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"
```

Index