vRealize Code Stream Trigger for Git

vRealize Code Stream 2.3

This document supports the version of each product listed and supports all subsequent versions until the document is replaced by a new edition. To check for more recent editions of this document, see http://www.vmware.com/support/pubs.

EN-002483-00

vmware®

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

http://www.vmware.com/support/

The VMware Web site also provides the latest product updates.

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

docfeedback@vmware.com

Copyright © 2017 VMware, Inc. All rights reserved. Copyright and trademark information.

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

Contents

Integrating vRealize Code Stream with GitHub or GitLab Enterprise

1

The vRealize Code Stream Trigger for Git integrates vRealize Code Stream with the Git lifecycle. The Trigger for Git enables events from GitHub Enterprise or GitLab Enterprise to trigger a pipeline.

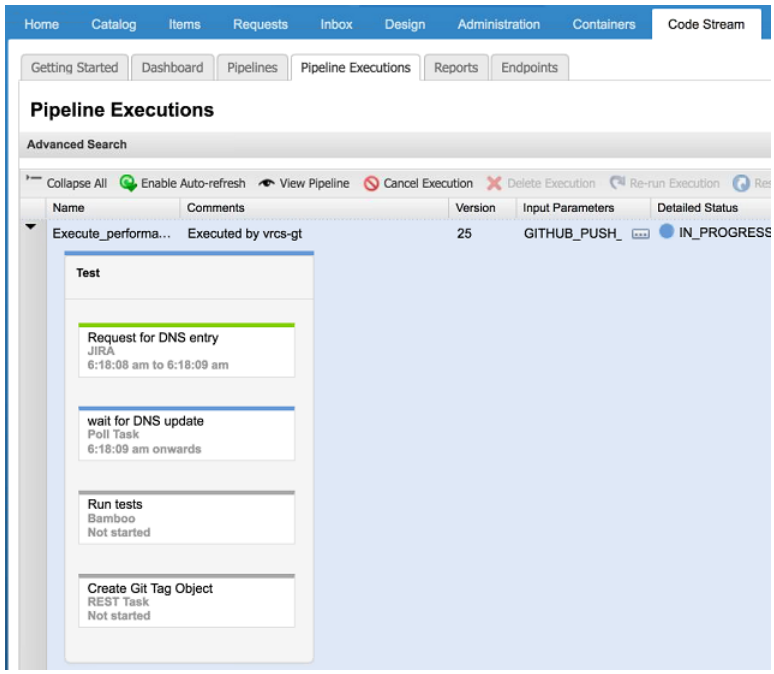
When developers make changes to code in a GitHub or GitLab repository, the change triggers an event. That event passes through a Webhook to the trigger for Git, which triggers the pipeline.

The events trigger the tasks in your pipeline. For example, your pipeline might include the following tasks.

Table 1-1. Tasks in a Pipeline

Task	Task type
Create a JIRA ticket to create a DNS entry and IP address.	JIRA
Wait for the ticket to create the DNS entry to be closed.	Poll
Run tests against the machine being added.	Bamboo
Create a Git tag object.	REST

These tasks appear as shown in the following pipeline:



You create the Webhook that allows the trigger for Git to receive events that occur in your repository. The following Webhook providers support the trigger for Git:

- GitHub provides version control, the ability to push artifacts to a repository, and the ability to release published artifacts.
- GitLab provides code testing and the ability to push artifacts to a repository.

This chapter includes the following topics:

Set Up and Run the Trigger for Git

You install and run the trigger for Git on a 32-bit or 64-bit Linux machine, or on your vRealize Code Stream appliance. Then, you create and run a configuration file to have the trigger for Git listen for events that occur. The configuration file defines the URL that the tool uses to listen for events.

To trigger a pipeline in vRealize Code Stream, GitHub and GitLab send events through Webhook providers to the trigger for Git. The tool monitors the URL and receives the events provided by the JSON payload from GitHub Enterprise, or from a standard Git server.

After you enter input properties in the pipeline configuration, the trigger for Git uses those properties, and the JSON payload, to call the pipeline.

Prerequisites

- Verify that GitHub Enterprise is installed on your development machine. See https://github.com/.
- Verify that you can access https://code.vmware.com to download the binary files.
- Familiarize yourself with the example configuration file. See "Example YAML Configuration File for Git Events," on page 7.

Procedure

1 Download the trigger for Git to any host, such as your Linux machine or vRealize Code Stream appliance.

   a Access https://code.vmware.com.

   b To locate the trigger for Git, search for vRealize Code Stream, and click the link to the latest version.

   c Download `vrcs-trigger.zip`, which contains the readme, binary, and sample files for the trigger for Git.

2 Install the trigger for Git on your Linux machine or vRealize Code Stream appliance.

   a Extract the ZIP file to a directory, such as `/opt/vrcs-trigger`.

   The top level directory includes a README file, which summarizes the tool commands, samples, and deployment. The directories named `/bin` and `/bin32` contain the binary files, and the directory named `samples` contains the sample YAML configuration file.

   b Export the PATH variable so that it includes the directory named `/bin`.

```
export PATH=$PATH:/opt/vrcs-trigger/bin
```

3 Generate your own configuration file.

```
vrcsgit generate /your_path/your_YAML-configuration-file
```

4 In the configuration file, update the sections named `gitListener`, `gitWebhook`, and `vrcsServer` with your values.

5 Validate the configuration file.

```
vrcsgit configure sample.yaml
```

The trigger for Git verifies the credentials for the vRealize Code Stream instance. It also encrypts the password, and the secret token if you provided one.

6 Run the configuration file for the Git trigger.

```
vrcsgit run sample.yaml
```

This command provides the URL with an IP address to use in your Webhook.

7 Copy the URL so that you can paste it in the Webhook for the trigger for Git.

You set up the trigger for Git to listen for events that occur in GitHub or GitLab.

When a developer modifies a file in your repository, the trigger for Git receives the list of change events on the URL. The tool then sends a request to trigger the pipeline.

What to do next

Create a Webhook for the trigger for Git to watch for GitHub or GitLab events that occur on your repository. See "Create a Webhook to Receive Events from GitHub Enterprise," on page 8.

Example YAML Configuration File for Git Events

You create a configuration file to listen for events from GitHub and GitLab. The configuration file defines the URL that the trigger for Git uses to listen for events.

In the YAML configuration file:

- The `gitListener` section generates the URL that the trigger for Git uses to listen for events from GitHub or GitLab. To enable HTTPS, you must provide the certificate and key file.
- The `gitWebHook` section allows you to trigger pipelines based on the GitHub or GitLab event. The `name` entry is the repository that contains the code to be pushed to the vRealize Code Stream pipeline. The `gitProvider` must be GitHub or GitLab.
- The `vrcsServer` section requires the location of vRealize Code Stream where the pipeline resides. When you enter the password the first time, it appears as clear text, but in following steps it is encrypted.

The configuration file includes the following code:

```
gitListener:
  name: git-listener-1
  # IP Address to listen on, for example 192.168.1.5
  IP: <SYSTEM_IP>
  # Port number to bind to. For example 1234
  portNumber: <PORT_NO>
  # Enable this to listen on https
  # Provide appropriate certificate, key file location below
  enableHTTPS: false
  certFileLocation: ""
  keyFileLocation: ""
gitWebhook:
  # Name of the project/repo
  name: project-name
  # Secret key passed from the Git system
  secret: ""
```

```
  # Provider name, supported are GitHub, GitLab (Enterprise versions)
  gitProvider: GitHub
  events:
  - pipelineName: preflight
    # Event push is supported on GitHub Enterprise and GitLab Enterprise
    event: push
  - pipelineName: postflight
    # Event release is supported only on GitHub Enterprise.
    event: release
vrcsServer:
  name: vrcs-server
  # Fully qualified URL of the vRCS server
  url: https://vrcs.example.com
  username: vrcs-user
  # Password in plain text. Will be encrypted during configure step
  password: vrcs-user-password
  tenant: tenant1
  # Time in minutes, the interval to poll for pipeline execution status
  pollIntervalInMinutes: 1
```

Create a Webhook to Receive Events from GitHub Enterprise

You must create a Webhook for the trigger for Git to receive GitHub events that occur on your repository. Triggers can include GitHub push and release events.

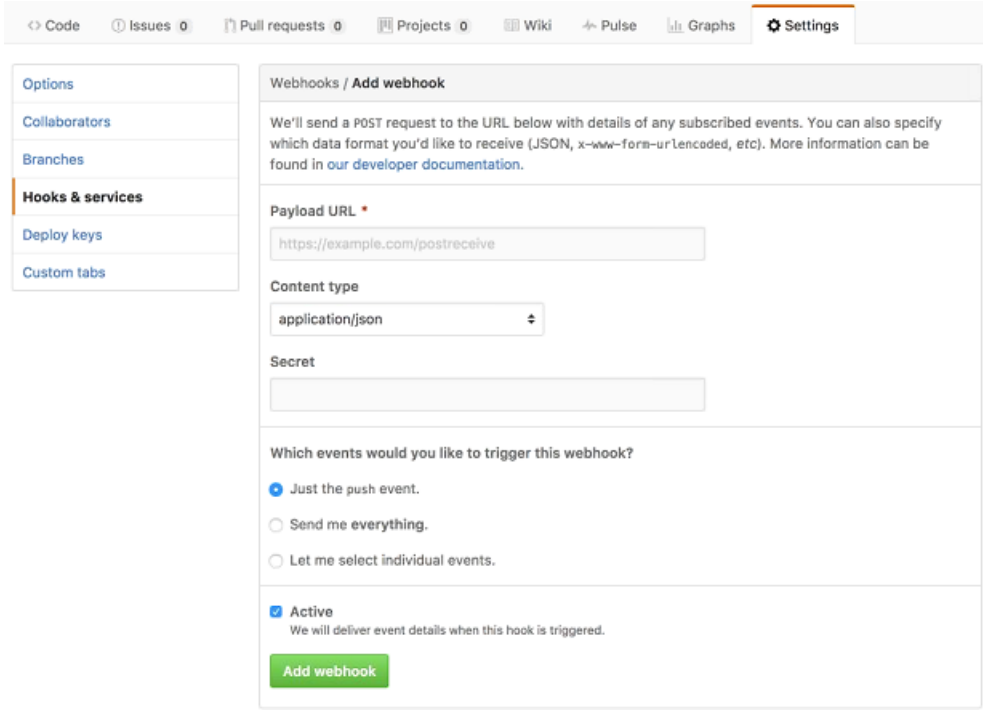
When you add a Webhook, you enter the URL for the trigger for Git to receive events from GitHub Enterprise. Then, you select the types of events to send.

Prerequisites

- Verify that you can access GitHub Enterprise to create a Webhook. See https://help.github.com/.

Procedure

1 In your GitHub Enterprise repository, click **Settings > Hooks & services**.



2 Click **Add webhook**.

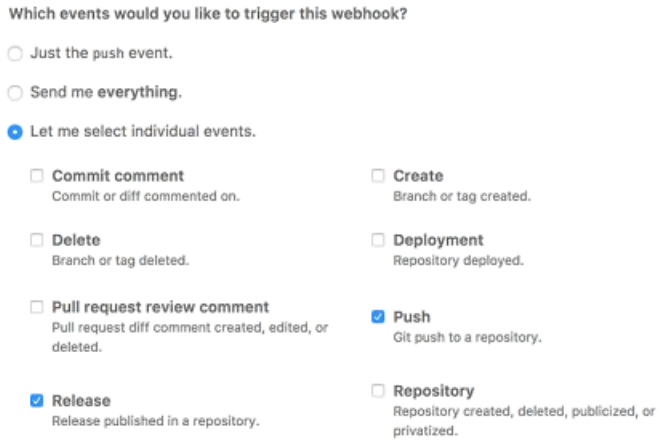
3 In the **Payload URL** text box, paste the Webhook URL that your configuration file generated.

The form of the payload URL resembles: `https://IP_address:port/webhook`

4 From the **Content type** drop-down menu, select **application/json**.

5 In the **Secret** text box, enter a token for the listener to use to verify the payload.

6 Select the type of events to trigger and send.

Option	Description
Just the push event.	Only trigger on GitHub Enterprise Push events.
Send me everything.	Not recommended. This option triggers and sends any event that occurs. Although GitHub supports multiple event types, the trigger for Git supports only Push and Release events. If you select multiple events that the trigger for Git does not support, the tool continues to log them, and indicates that the events are not supported.
Let me select individual Events.	Select Push events, Release events, or both, for the trigger.  ■ **Push.** A Git push to a repository. ■ **Release.** A release published in a repository.

You created a Webhook for the trigger for Git to receive Push and Release events that occur in your GitHub repository.

What to do next

Create a Webhook for the trigger for Git to receive GitLab events that occur on your repository. See "Create a Webhook to Receive Events from GitLab Enterprise," on page 11.

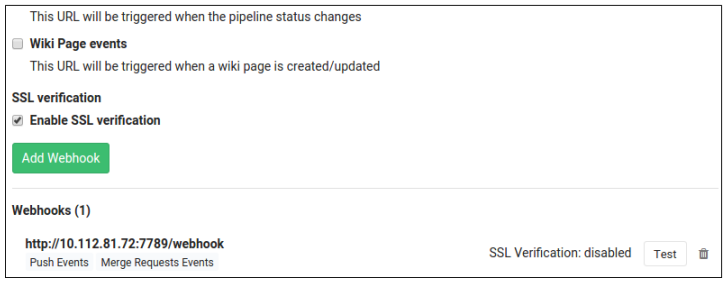
Create a Webhook to Receive Events from GitLab Enterprise

You must create a Webhook for the trigger for Git to receive GitLab events that occur on your repository. Triggers can include GitLab push events.

When you add a Webhook, you enter the URL that your configuration file created for the trigger for Git to receive events from GitLab Enterprise. Then, you select the types of Push events to send. The trigger for Git supports GitLab Push events.

Caution Before you test the Git connection from GitLab, and start the Git trigger, keep your pipeline in Draft state. If your pipeline is in the Activated state, a test of the Git trigger might cause the pipeline to trigger unexpectedly, which might result in unintended consequences. To test the connection from Gitlab, click the **Test** button. Then, start the Git trigger and check the logs. If the logs display the following message, the test connection worked. `WARNING: 2017/04/06 05:31:05 Pipeline <pipeline name> is not activated. Skipping event.`

To locate the Test button and test the connection, in GitLab click **Settings > Integrations**. The Test button appears next to your Webhook in the lower right area of the Webhook section, as shown here.



Prerequisites

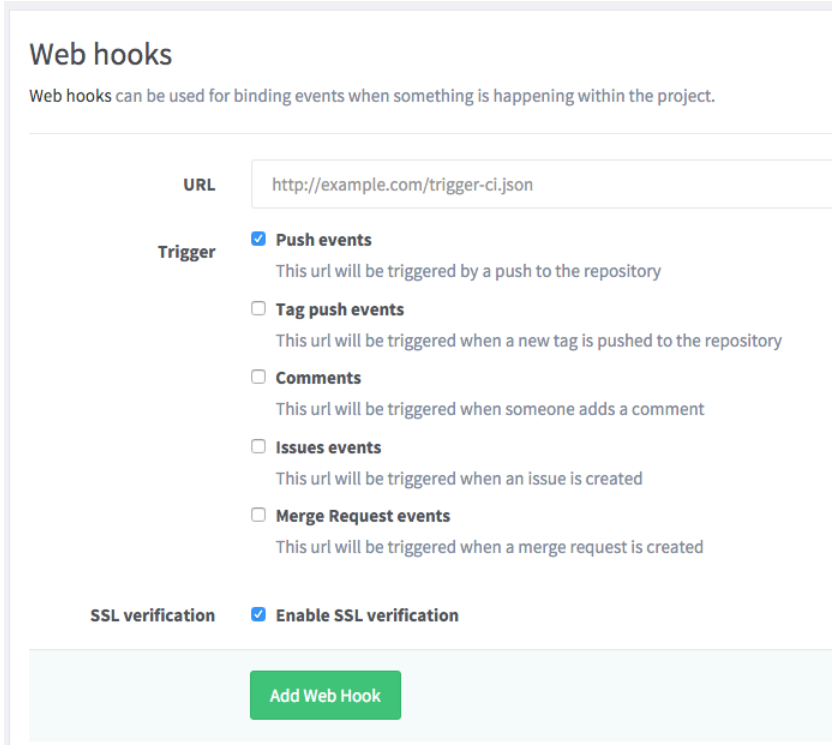
- Verify that you can access GitLab Enterprise to create a Webhook. See http://docs.buddybuild.com/docs/adding-a-gitlab-webhook.
- Verify that your pipeline is in Draft state before you test the Git trigger.

Procedure

1. In your GitLab Enterprise repository, click **Settings**.
2. From the menu, click **Webhooks**.
3. In the **URL** text box, paste the Webhook URL that your configuration file generated.

   The form of the payload URL resembles: **`https://IP_address:port/webhook`**
4. In the **Secret Token** text box, enter a token for the listener to use to verify the payload.

5 Select the type of Push events to trigger and send.



6 Click **Add Web Hook**.

You created a Webhook for the trigger for Git to receive events that occur in your GitLab repository.

What to do next

Configure a pipeline to examine the events. See Chapter 2, "Configure Your Pipeline to View Git Events," on page 15.

Integrate vRealize Code Stream with a Standard Git Server

If a standard Git server is installed, and you are not using a hosted Git provider such as GitHub or GitLab, you can integrate the trigger for Git with that server to trigger pipelines in vRealize Code Stream.

Follow this procedure only if you are not using a hosted Git provider such as GitHub or GitLab.

This procedure adds a script to your standard Git server. The script sends the following properties to the pipeline: `branch`, `refname`, `newrev`, and `oldrev`.

Prerequisites

- Verify that a standard Git server is available.

Procedure

1 In the Hooks folder of your standard Git server, create a script file named `post-receive`.

2 Add the following lines to the new script file or to an existing post-receive hook, and replace the variable names for your vRealize Code Stream server.

```
#!/bin/sh

username='vrcs-user'
password='vrcs-user-password'
```

```
tenant='tenant1'
pipeline_name='githook-pipeline'
server_host='vrcs.example.host.com'

read oldrev newrev refname
branch=${refname#refs/heads/}

#A SSO token is required to make any calls to the vRealize Code Stream server.
#A 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
#vRealize 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 $token" -k -d
'{"id":"","description":"from posthook","pipelineParams":
[{"name":"branch","type":"STRING","value":"'"$branch"'","description":""},
{"name":"refname","type":"STRING","value":"'"$refname"'","description":""},
{"name":"newrev","type":"STRING","value":"'"$newrev"'","description":""},
{"name":"oldrev","type":"STRING","value":"'"$oldrev"'","description":""}]}'
$execute_pipeline_url)

echo "Response to execute pipeline => $response"
```

The script sends the properties to vRealize Code Stream to trigger your pipeline.

You integrated the trigger for Git with a standard Git server so that you can trigger your pipeline.

What to do next

Configure your pipeline to receive the input properties. All properties are optional. See Chapter 2, "Configure Your Pipeline to View Git Events," on page 15.

Configure Your Pipeline to View Git Events

2

You can configure your pipeline in vRealize Code Stream, and examine the GitHub or GitLab events that occurred.

When the trigger for Git receives events from GitHub and GitLab through the Webhook you created, the tool sends a request to trigger the pipeline.

Prerequisites

- Run a configuration file to integrate the trigger for Git with GitHub and GitLab. See "Set Up and Run the Trigger for Git," on page 6.
- Create a Webhook for the trigger for Git to receive events that occur in GitHub or GitLab. See "Create a Webhook to Receive Events from GitHub Enterprise," on page 8.
- You are familiar with the pipeline configuration input properties to integrate with Git. See "Pipeline Input Properties for Git Integration," on page 16.

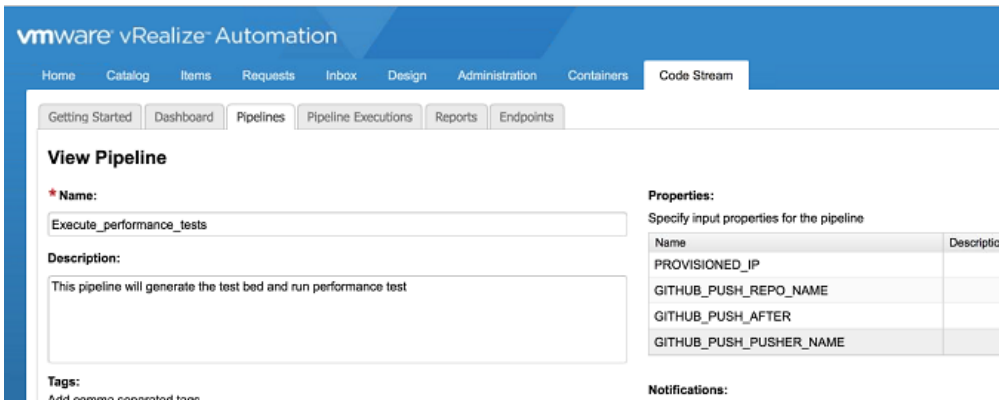
Procedure

1 In vRealize Code Stream, click **Pipelines**.

2 Configure the input properties to your pipeline so that it can receive events from GitHub or GitLab.

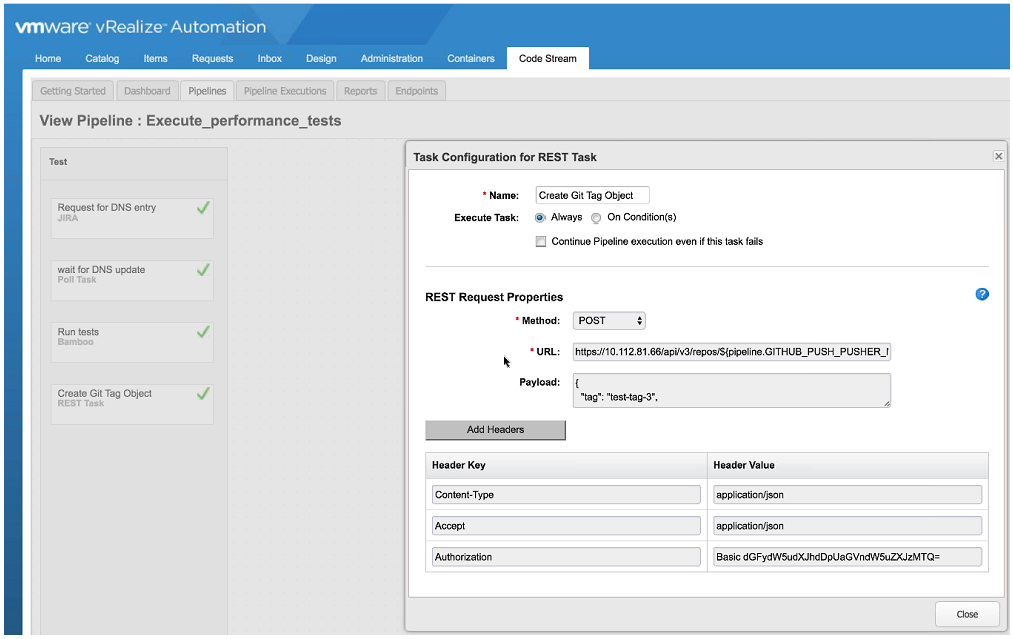
   a Select a pipeline.

   b Configure the input properties. For example:



c To have a pipeline task trigger when an event occurs in the Git repository, configure the task.

For example, you can have the trigger for Git send a POST command to the Git repository to create a tag on your check-in.



When the event occurs in the Git repository, the trigger for Git detects the change and triggers the pipeline. When the pipeline completes, the tool displays a message that indicates whether the pipeline succeeded or failed.

d Save the task and the pipeline.

3 Click **Pipeline Executions** and view the status of the pipeline that the events triggered.

You configured your pipeline and examined the Git events that triggered the pipeline.

Pipeline Input Properties for Git Integration

To integrate vRealize Code Stream with Git, you enter input properties in your pipeline configuration.

GitHub and GitLab Events

GitHub and GitLab send change events to the trigger for Git through the Webhook that you created. The events that the pipeline receives are based on the input properties in the pipeline configuration.

The following GitHub push events, GitHub release events, and GitLab push events are allowed.

Table 2-1. Input Properties for the Pipeline Configuration

Event Type	Events
GitHub push events	■ **GITHUB_PUSH_REF**. The full Git reference that was pushed. For example, `refs/heads/master`. ■ **GITHUB_PUSH_BEFORE**. The SHA of the most recent commit on the Git `ref` directory before the push. ■ **GITHUB_PUSH_AFTER**. The SHA on the Git reference after the push occurs, which is the comment ID included on the commit. ■ **GITHUB_PUSH_BASE_REF** ■ **GITHUB_PUSH_COMMITS_PAYLOAD**. An array of commit objects that describes the pushed commits. The array includes a maximum of 20 commits. ■ **GITHUB_PUSH_REPO_ID** ■ **GITHUB_PUSH_REPO_NAME** ■ **GITHUB_PUSH_REPO_URL** ■ **GITHUB_PUSH_PUSHER_NAME**. Name of the person who commits the code to be released. ■ **GITHUB_PUSH_PUSHER_EMAIL** ■ **GITHUB_PUSH_PAYLOAD**. The entire JSON payload that the trigger for Git received from the Webhook.
GitHub release events	■ **GITHUB_RELEASE_ACTION**. The action that was performed. The only action allowed is `PUBLISHED`. ■ **GITHUB_RELEASE_URL** ■ **GITHUB_RELEASE_NAME** ■ **GITHUB_RELEASE_ID** ■ **GITHUB_RELEASE_AUTHOR_LOGIN** ■ **GITHUB_RELEASE_AUTHOR_ID** ■ **GITHUB_RELEASE_AUTHOR_URL** ■ **GITHUB_RELEASE_SENDER_LOGIN** ■ **GITHUB_RELEASE_SENDER_ID** ■ **GITHUB_RELEASE_SENDER_URL** ■ **GITHUB_RELEASE_PAYLOAD**. The entire JSON payload that the trigger for Git received from the Webhook.
GitLab push events	■ **GITLAB_PUSH_USER_ID** ■ **GITLAB_PUSH_USER_NAME** ■ **GITLAB_PUSH_USER_EMAIL** ■ **GITLAB_PUSH_PROJECT_NAME** ■ **GITLAB_PUSH_PROJECT_URL** ■ **GITLAB_PUSH_COMMITS** ■ **GITLAB_PUSH_REPO_NAME** ■ **GITLAB_PUSH_REPO_URL** ■ **GITLAB_PUSH_REF**. The full Git reference that was pushed. For example, `refs/heads/master`.

Index