



Red Hat OpenShift Pipelines 1.15

Creating CI/CD pipelines

Getting started with creating and running tasks and pipelines in OpenShift Pipelines

Red Hat OpenShift Pipelines 1.15 Creating CI/CD pipelines

Getting started with creating and running tasks and pipelines in OpenShift Pipelines

Legal Notice

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This document provides information about creating and running tasks and pipelines in OpenShift Pipelines.

Table of Contents

CHAPTER 1. CREATING CI/CD SOLUTIONS FOR APPLICATIONS USING OPENSIFT PIPELINES	4
1.1. PREREQUISITES	4
1.2. CREATING A PROJECT AND CHECKING YOUR PIPELINE SERVICE ACCOUNT	4
1.3. CREATING PIPELINE TASKS	5
1.4. ASSEMBLING A PIPELINE	6
1.5. MIRRORING IMAGES TO RUN PIPELINES IN A RESTRICTED ENVIRONMENT	8
1.6. RUNNING A PIPELINE	12
1.7. ADDING TRIGGERS TO A PIPELINE	13
1.8. CONFIGURING EVENT LISTENERS TO SERVE MULTIPLE NAMESPACES	17
1.9. CREATING WEBHOOKS	20
1.10. TRIGGERING A PIPELINE RUN	21
1.11. ENABLING MONITORING OF EVENT LISTENERS FOR TRIGGERS FOR USER-DEFINED PROJECTS	21
1.12. CONFIGURING PULL REQUEST CAPABILITIES IN GITHUB INTERCEPTOR	22
1.12.1. Filtering pull requests using GitHub Interceptor	23
1.12.2. Validating pull requests using GitHub Interceptors	24
1.13. ADDITIONAL RESOURCES	26
CHAPTER 2. WORKING WITH RED HAT OPENSIFT PIPELINES IN THE WEB CONSOLE	27
2.1. WORKING WITH RED HAT OPENSIFT PIPELINES IN THE DEVELOPER PERSPECTIVE	27
Prerequisites	27
2.1.1. Constructing pipelines using the Pipeline builder	27
2.1.2. Creating OpenShift Pipelines along with applications	30
2.1.3. Adding a GitHub repository containing pipelines	30
2.1.4. Interacting with pipelines using the Developer perspective	34
2.1.5. Starting pipelines from Pipelines view	36
2.1.6. Starting pipelines from Topology view	38
2.1.7. Interacting with pipelines from Topology view	39
2.1.8. Editing pipelines	39
2.1.9. Deleting pipelines	40
2.2. ADDITIONAL RESOURCES	40
2.3. CREATING PIPELINE TEMPLATES IN THE ADMINISTRATOR PERSPECTIVE	40
2.4. PIPELINE EXECUTION STATISTICS IN THE WEB CONSOLE	41
2.4.1. Enabling the OpenShift Pipelines console plugin	41
2.4.2. Viewing the statistics for all pipelines together	42
2.4.3. Viewing the statistics for a specific pipeline	42
CHAPTER 3. SPECIFYING REMOTE PIPELINES AND TASKS USING RESOLVERS	44
3.1. SPECIFYING A REMOTE PIPELINE OR TASK FROM A TEKTON CATALOG	44
3.1.1. Configuring the hub resolver	44
3.1.2. Specifying a remote pipeline or task using the hub resolver	45
3.2. SPECIFYING A REMOTE PIPELINE OR TASK FROM A TEKTON BUNDLE	48
3.2.1. Configuring the bundles resolver	48
3.2.2. Specifying a remote pipeline or task using the bundles resolver	48
3.3. SPECIFYING A REMOTE PIPELINE OR TASK FROM A GIT REPOSITORY	50
3.3.1. Configuring the Git resolver for anonymous Git cloning	51
3.3.2. Configuring the Git resolver for the authenticated SCM API	51
3.3.3. Specifying a remote pipeline or task using the Git resolver	53
3.4. SPECIFYING A REMOTE PIPELINE OR TASK FROM THE SAME CLUSTER	55
3.4.1. Configuring the cluster resolver	55
3.4.2. Specifying a remote pipeline or task using the cluster resolver	56
3.5. TASKS PROVIDED IN THE OPENSIFT PIPELINES NAMESPACE	58

buildah	58
git-cli	60
git-clone	63
kn	65
kn-apply	66
maven	67
openshift-client	68
s2i-dotnet	69
s2i-go	71
s2i-java	73
s2i-nodejs	75
s2i-perl	77
s2i-php	79
s2i-python	81
s2i-ruby	83
skopeo-copy	85
tkn	86
3.6. ADDITIONAL RESOURCES	87
CHAPTER 4. USING MANUAL APPROVAL IN OPENSIFT PIPELINES	88
4.1. ENABLING THE MANUAL APPROVAL GATE CONTROLLER	88
4.2. SPECIFYING A MANUAL APPROVAL TASK	89
4.3. APPROVING A MANUAL APPROVAL TASK	90
4.3.1. Approving a manual approval task by using the web console	90
4.3.2. Approving a manual approval task by using the command line	91
CHAPTER 5. USING RED HAT ENTITLEMENTS IN PIPELINES	93
5.1. PREREQUISITES	93
5.2. USING RED HAT ENTITLEMENTS BY MANUALLY COPYING THE ETC-PKI-ENTITLEMENT SECRET	94
5.3. USING RED HAT ENTITLEMENTS BY SHARING THE SECRET USING THE SHARED RESOURCES CSI DRIVER OPERATOR	95
5.4. ADDITIONAL RESOURCES	98
CHAPTER 6. MANAGING NON-VERSIONED AND VERSIONED CLUSTER TASKS	99
6.1. DIFFERENCES BETWEEN NON-VERSIONED AND VERSIONED CLUSTER TASKS	99
6.2. ADVANTAGES AND DISADVANTAGES OF NON-VERSIONED AND VERSIONED CLUSTER TASKS	99
6.3. DISABLING NON-VERSIONED AND VERSIONED CLUSTER TASKS	100

CHAPTER 1. CREATING CI/CD SOLUTIONS FOR APPLICATIONS USING OPENSIFT PIPELINES

With Red Hat OpenShift Pipelines, you can create a customized CI/CD solution to build, test, and deploy your application.

To create a full-fledged, self-serving CI/CD pipeline for an application, perform the following tasks:

- Create custom tasks, or install existing reusable tasks.
- Create and define the delivery pipeline for your application.
- Provide a storage volume or filesystem that is attached to a workspace for the pipeline execution, using one of the following approaches:
 - Specify a volume claim template that creates a persistent volume claim
 - Specify a persistent volume claim
- Create a **PipelineRun** object to instantiate and invoke the pipeline.
- Add triggers to capture events in the source repository.

This section uses the **pipelines-tutorial** example to demonstrate the preceding tasks. The example uses a simple application which consists of:

- A front-end interface, **pipelines-vote-ui**, with the source code in the [pipelines-vote-ui](#) Git repository.
- A back-end interface, **pipelines-vote-api**, with the source code in the [pipelines-vote-api](#) Git repository.
- The **apply-manifests** and **update-deployment** tasks in the [pipelines-tutorial](#) Git repository.

1.1. PREREQUISITES

- You have access to an OpenShift Container Platform cluster.
- You have installed [OpenShift Pipelines](#) using the Red Hat OpenShift Pipelines Operator listed in the OpenShift OperatorHub. After it is installed, it is applicable to the entire cluster.
- You have installed [OpenShift Pipelines CLI](#).
- You have forked the front-end [pipelines-vote-ui](#) and back-end [pipelines-vote-api](#) Git repositories using your GitHub ID, and have administrator access to these repositories.
- Optional: You have cloned the [pipelines-tutorial](#) Git repository.

1.2. CREATING A PROJECT AND CHECKING YOUR PIPELINE SERVICE ACCOUNT

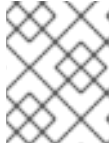
Procedure

1. Log in to your OpenShift Container Platform cluster:
 -


```
$ oc login -u <login> -p <password> https://openshift.example.com:6443
```

2. Create a project for the sample application. For this example workflow, create the **pipelines-tutorial** project:

```
$ oc new-project pipelines-tutorial
```



NOTE

If you create a project with a different name, be sure to update the resource URLs used in the example with your project name.

3. View the **pipeline** service account:
Red Hat OpenShift Pipelines Operator adds and configures a service account named **pipeline** that has sufficient permissions to build and push an image. This service account is used by the **PipelineRun** object.

```
$ oc get serviceaccount pipeline
```

1.3. CREATING PIPELINE TASKS

Procedure

1. Install the **apply-manifests** and **update-deployment** task resources from the **pipelines-tutorial** repository, which contains a list of reusable tasks for pipelines:

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.15/01_pipeline/01_apply_manifest_task.yaml
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.15/01_pipeline/02_update_deployment_task.yaml
```

2. Use the **tkn task list** command to list the tasks you created:

```
$ tkn task list
```

The output verifies that the **apply-manifests** and **update-deployment** task resources were created:

NAME	DESCRIPTION	AGE
apply-manifests		1 minute ago
update-deployment		48 seconds ago

3. Use the **tkn clustertasks list** command to list the Operator-installed additional cluster tasks such as **buildah** and **s2i-python**:



NOTE

To use the **buildah** cluster task in a restricted environment, you must ensure that the Dockerfile uses an internal image stream as the base image.

```
$ tkn clustertasks list
```

The output lists the Operator-installed **ClusterTask** resources:

NAME	DESCRIPTION	AGE
buildah		1 day ago
git-clone		1 day ago
s2i-python		1 day ago
tkn		1 day ago



IMPORTANT

In Red Hat OpenShift Pipelines 1.10, **ClusterTask** functionality is deprecated and is planned to be removed in a future release.

Additional resources

- [Managing non-versioned and versioned cluster tasks](#)

1.4. ASSEMBLING A PIPELINE

A pipeline represents a CI/CD flow and is defined by the tasks to be executed. It is designed to be generic and reusable in multiple applications and environments.

A pipeline specifies how the tasks interact with each other and their order of execution using the **from** and **runAfter** parameters. It uses the **workspaces** field to specify one or more volumes that each task in the pipeline requires during execution.

In this section, you will create a pipeline that takes the source code of the application from GitHub, and then builds and deploys it on OpenShift Container Platform.

The pipeline performs the following tasks for the back-end application **pipelines-vote-api** and front-end application **pipelines-vote-ui**:

- Clones the source code of the application from the Git repository by referring to the **git-url** and **git-revision** parameters.
- Builds the container image using the **buildah** task provided in the **openshift-pipelines** namespace.
- Pushes the image to the OpenShift image registry by referring to the **image** parameter.
- Deploys the new image on OpenShift Container Platform by using the **apply-manifests** and **update-deployment** tasks.

Procedure

1. Copy the contents of the following sample pipeline YAML file and save it:

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
```

```

workspaces:
- name: shared-workspace
params:
- name: deployment-name
  type: string
  description: name of the deployment to be patched
- name: git-url
  type: string
  description: url of the git repo for the code of deployment
- name: git-revision
  type: string
  description: revision to be used from repo of the code for deployment
  default: "pipelines-1.15"
- name: IMAGE
  type: string
  description: image to be built from the code
tasks:
- name: fetch-repository
  taskRef:
    resolver: cluster
    params:
      - name: kind
        value: task
      - name: name
        value: git-clone
      - name: namespace
        value: openshift-pipelines
  workspaces:
    - name: output
      workspace: shared-workspace
  params:
    - name: URL
      value: $(params.git-url)
    - name: SUBDIRECTORY
      value: ""
    - name: DELETE_EXISTING
      value: "true"
    - name: REVISION
      value: $(params.git-revision)
- name: build-image
  taskRef:
    resolver: cluster
    params:
      - name: kind
        value: task
      - name: name
        value: buildah
      - name: namespace
        value: openshift-pipelines
  workspaces:
    - name: source
      workspace: shared-workspace
  params:
    - name: IMAGE
      value: $(params.IMAGE)
  runAfter:

```

```

- fetch-repository
- name: apply-manifests
  taskRef:
    name: apply-manifests
  workspaces:
    - name: source
      workspace: shared-workspace
  runAfter:
    - build-image
- name: update-deployment
  taskRef:
    name: update-deployment
  params:
    - name: deployment
      value: $(params.deployment-name)
    - name: IMAGE
      value: $(params.IMAGE)
  runAfter:
    - apply-manifests

```

The pipeline definition abstracts away the specifics of the Git source repository and image registries. These details are added as **params** when a pipeline is triggered and executed.

2. Create the pipeline:

```
$ oc create -f <pipeline-yaml-file-name.yaml>
```

Alternatively, you can also execute the YAML file directly from the Git repository:

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.15/01_pipeline/04_pipeline.yaml
```

3. Use the **tkn pipeline list** command to verify that the pipeline is added to the application:

```
$ tkn pipeline list
```

The output verifies that the **build-and-deploy** pipeline was created:

```

NAME          AGE          LAST RUN   STARTED   DURATION   STATUS
build-and-deploy 1 minute ago ---       ---       ---       ---

```

1.5. MIRRORING IMAGES TO RUN PIPELINES IN A RESTRICTED ENVIRONMENT

To run OpenShift Pipelines in a disconnected cluster or a cluster provisioned in a restricted environment, ensure that either the Samples Operator is configured for a restricted network, or a cluster administrator has created a cluster with a mirrored registry.

The following procedure uses the **pipelines-tutorial** example to create a pipeline for an application in a restricted environment using a cluster with a mirrored registry. To ensure that the **pipelines-tutorial** example works in a restricted environment, you must mirror the respective builder images from the mirror registry for the front-end interface, **pipelines-vote-ui**; back-end interface, **pipelines-vote-api**; and the **cli**.

Procedure

1. Mirror the builder image from the mirror registry for the front-end interface, **pipelines-vote-ui**.
 - a. Verify that the required images tag is not imported:

```
$ oc describe imagestream python -n openshift
```

Example output

```
Name: python
Namespace: openshift
[...]
```

```
3.8-ubi9 (latest)
  tagged from registry.redhat.io/ubi9/python-38:latest
  prefer registry pullthrough when referencing this tag
```

Build and run Python 3.8 applications on UBI 8. For more information about using this builder image, including OpenShift considerations, see <https://github.com/sclorg/s2i-python-container/blob/master/3.8/README.md>.

```
Tags: builder, python
Supports: python:3.8, python
Example Repo: https://github.com/sclorg/django-ex.git
```

```
[...]
```

- b. Mirror the supported image tag to the private registry:

```
$ oc image mirror registry.redhat.io/ubi9/python-39:latest <mirror-registry>:
<port>/ubi9/python-39
```

- c. Import the image:

```
$ oc tag <mirror-registry>:<port>/ubi9/python-39 python:latest --scheduled -n openshift
```

You must periodically re-import the image. The **--scheduled** flag enables automatic re-import of the image.

- d. Verify that the images with the given tag have been imported:

```
$ oc describe imagestream python -n openshift
```

Example output

```
Name: python
Namespace: openshift
[...]
```

```
latest
  updates automatically from registry <mirror-registry>:<port>/ubi9/python-39
```

```
* <mirror-registry>:<port>/ubi9/python-39@sha256:3ee...
```

```
[...]
```

2. Mirror the builder image from the mirror registry for the back-end interface, **pipelines-vote-api**.

- a. Verify that the required images tag is not imported:

```
$ oc describe imagestream golang -n openshift
```

Example output

```
Name: golang
Namespace: openshift
[...]
```

```
1.14.7-ubi8 (latest)
tagged from registry.redhat.io/ubi8/go-toolset:1.14.7
prefer registry pullthrough when referencing this tag
```

Build and run Go applications on UBI 8. For more information about using this builder image, including OpenShift considerations, see <https://github.com/sclorg/golang-container/blob/master/README.md>.

```
Tags: builder, golang, go
Supports: golang
Example Repo: https://github.com/sclorg/golang-ex.git
```

```
[...]
```

- b. Mirror the supported image tag to the private registry:

```
$ oc image mirror registry.redhat.io/ubi9/go-toolset:latest <mirror-registry>:
<port>/ubi9/go-toolset
```

- c. Import the image:

```
$ oc tag <mirror-registry>:<port>/ubi9/go-toolset golang:latest --scheduled -n openshift
```

You must periodically re-import the image. The **--scheduled** flag enables automatic re-import of the image.

- d. Verify that the images with the given tag have been imported:

```
$ oc describe imagestream golang -n openshift
```

Example output

```
Name: golang
Namespace: openshift
[...]
```

```
latest
updates automatically from registry <mirror-registry>:<port>/ubi9/go-toolset
```

```
* <mirror-registry>:<port>/ubi9/go-
toolset@sha256:59a74d581df3a2bd63ab55f7ac106677694bf612a1fe9e7e3e1487f55c421
b37

[...]
```

3. Mirror the builder image from the mirror registry for the **cli**.

- a. Verify that the required images tag is not imported:

```
$ oc describe imagestream cli -n openshift
```

Example output

```
Name:          cli
Namespace:     openshift
[...]

latest
updates automatically from registry quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143551

* quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143551

[...]
```

- b. Mirror the supported image tag to the private registry:

```
$ oc image mirror quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143551
<mirror-registry>:<port>/openshift-release-dev/ocp-v4.0-art-dev:latest
```

- c. Import the image:

```
$ oc tag <mirror-registry>:<port>/openshift-release-dev/ocp-v4.0-art-dev cli:latest --
scheduled -n openshift
```

You must periodically re-import the image. The **--scheduled** flag enables automatic re-import of the image.

- d. Verify that the images with the given tag have been imported:

```
$ oc describe imagestream cli -n openshift
```

Example output

```
Name:          cli
Namespace:     openshift
[...]

latest
```

```
updates automatically from registry <mirror-registry>:<port>/openshift-release-dev/ocp-
v4.0-art-dev

* <mirror-registry>:<port>/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143551

[...]
```

Additional resources

- [Configuring Samples Operator for a restricted cluster](#)
- [Creating a cluster with a mirrored registry](#)

1.6. RUNNING A PIPELINE

A **PipelineRun** resource starts a pipeline and ties it to the Git and image resources that should be used for the specific invocation. It automatically creates and starts the **TaskRun** resources for each task in the pipeline.

Procedure

1. Start the pipeline for the back-end application:

```
$ tkn pipeline start build-and-deploy \
  -w name=shared-
workspace,volumeClaimTemplateFile=https://raw.githubusercontent.com/openshift/pipelines-
tutorial/pipelines-1.15/01_pipeline/03_persistent_volume_claim.yaml \
  -p deployment-name=pipelines-vote-api \
  -p git-url=https://github.com/openshift/pipelines-vote-api.git \
  -p IMAGE='image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/pipelines-
vote-api' \
  --use-param-defaults
```

The previous command uses a volume claim template, which creates a persistent volume claim for the pipeline execution.

2. To track the progress of the pipeline run, enter the following command::

```
$ tkn pipelinerun logs <pipelinerun_id> -f
```

The <pipelinerun_id> in the above command is the ID for the **PipelineRun** that was returned in the output of the previous command.

3. Start the pipeline for the front-end application:

```
$ tkn pipeline start build-and-deploy \
  -w name=shared-
workspace,volumeClaimTemplateFile=https://raw.githubusercontent.com/openshift/pipelines-
tutorial/pipelines-1.15/01_pipeline/03_persistent_volume_claim.yaml \
  -p deployment-name=pipelines-vote-ui \
  -p git-url=https://github.com/openshift/pipelines-vote-ui.git \
```



```
-p IMAGE='image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/pipelines-
vote-ui' \
--use-param-defaults
```

- To track the progress of the pipeline run, enter the following command:

```
$ tkn pipelinerun logs <pipelinerun_id> -f
```

The <pipelinerun_id> in the above command is the ID for the **PipelineRun** that was returned in the output of the previous command.

- After a few minutes, use **tkn pipelinerun list** command to verify that the pipeline ran successfully by listing all the pipeline runs:

```
$ tkn pipelinerun list
```

The output lists the pipeline runs:

NAME	STARTED	DURATION	STATUS
build-and-deploy-run-xy7rw	1 hour ago	2 minutes	Succeeded
build-and-deploy-run-z2rz8	1 hour ago	19 minutes	Succeeded

- Get the application route:

```
$ oc get route pipelines-vote-ui --template='http://{{.spec.host}}'
```

Note the output of the previous command. You can access the application using this route.

- To rerun the last pipeline run, using the pipeline resources and service account of the previous pipeline, run:

```
$ tkn pipeline start build-and-deploy --last
```

Additional resources

- [Authenticating pipelines with repositories using secrets](#)

1.7. ADDING TRIGGERS TO A PIPELINE

Triggers enable pipelines to respond to external GitHub events, such as push events and pull requests. After you assemble and start a pipeline for the application, add the **TriggerBinding**, **TriggerTemplate**, **Trigger**, and **EventListener** resources to capture the GitHub events.

Procedure

- Copy the content of the following sample **TriggerBinding** YAML file and save it:

```
apiVersion: triggers.tekton.dev/v1beta1
kind: TriggerBinding
metadata:
  name: vote-app
spec:
  params:
```

```

- name: git-repo-url
  value: $(body.repository.url)
- name: git-repo-name
  value: $(body.repository.name)
- name: git-revision
  value: $(body.head_commit.id)

```

2. Create the **TriggerBinding** resource:

```
$ oc create -f <triggerbinding-yaml-file-name.yaml>
```

Alternatively, you can create the **TriggerBinding** resource directly from the **pipelines-tutorial** Git repository:

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.15/03_triggers/01_binding.yaml
```

3. Copy the content of the following sample **TriggerTemplate** YAML file and save it:

```

apiVersion: triggers.tekton.dev/v1beta1
kind: TriggerTemplate
metadata:
  name: vote-app
spec:
  params:
    - name: git-repo-url
      description: The git repository url
    - name: git-revision
      description: The git revision
      default: pipelines-1.15
    - name: git-repo-name
      description: The name of the deployment to be created / patched

  resourcetemplates:
    - apiVersion: tekton.dev/v1
      kind: PipelineRun
      metadata:
        generateName: build-deploy-$(tt.params.git-repo-name)-
      spec:
        taskRunTemplate:
          serviceAccountName: pipeline
        pipelineRef:
          name: build-and-deploy
        params:
          - name: deployment-name
            value: $(tt.params.git-repo-name)
          - name: git-url
            value: $(tt.params.git-repo-url)
          - name: git-revision
            value: $(tt.params.git-revision)
          - name: IMAGE
            value: image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/$(tt.params.git-repo-name)
        workspaces:
          - name: shared-workspace

```

```

volumeClaimTemplate:
  spec:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 500Mi

```

The template specifies a volume claim template to create a persistent volume claim for defining the storage volume for the workspace. Therefore, you do not need to create a persistent volume claim to provide data storage.

4. Create the **TriggerTemplate** resource:

```
$ oc create -f <triggertemplate-yaml-file-name.yaml>
```

Alternatively, you can create the **TriggerTemplate** resource directly from the **pipelines-tutorial** Git repository:

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.15/03_triggers/02_template.yaml
```

5. Copy the contents of the following sample **Trigger** YAML file and save it:

```

apiVersion: triggers.tekton.dev/v1beta1
kind: Trigger
metadata:
  name: vote-trigger
spec:
  taskRunTemplate:
    serviceAccountName: pipeline
  bindings:
    - ref: vote-app
  template:
    ref: vote-app

```

6. Create the **Trigger** resource:

```
$ oc create -f <trigger-yaml-file-name.yaml>
```

Alternatively, you can create the **Trigger** resource directly from the **pipelines-tutorial** Git repository:

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.15/03_triggers/03_trigger.yaml
```

7. Copy the contents of the following sample **EventListener** YAML file and save it:

```

apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: vote-app
spec:
  taskRunTemplate:

```

```

serviceAccountName: pipeline
triggers:
  - triggerRef: vote-trigger

```

Alternatively, if you have not defined a trigger custom resource, add the binding and template spec to the **EventListener** YAML file, instead of referring to the name of the trigger:

```

apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: vote-app
spec:
  taskRunTemplate:
    serviceAccountName: pipeline
  triggers:
    - bindings:
      - ref: vote-app
    template:
      ref: vote-app

```

8. Create the **EventListener** resource by performing the following steps:

- To create an **EventListener** resource using a secure HTTPS connection:
 - a. Add a label to enable the secure HTTPS connection to the **Eventlistener** resource:

```
$ oc label namespace <ns-name> operator.tekton.dev/enable-annotation=enabled
```

- b. Create the **EventListener** resource:

```
$ oc create -f <eventlistener-yaml-file-name.yaml>
```

Alternatively, you can create the **EventListener** resource directly from the **pipelines-tutorial** Git repository:

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.15/03_triggers/04_event_listener.yaml
```

- c. Create a route with the re-encrypt TLS termination:

```
$ oc create route reencrypt --service=<svc-name> --cert=tls.crt --key=tls.key --ca-cert=ca.crt --hostname=<hostname>
```

Alternatively, you can create a re-encrypt TLS termination YAML file to create a secured route.

Example Re-encrypt TLS Termination YAML of the Secured Route

```

apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-passthrough-secured 1
spec:
  host: <hostname>

```

```

to:
  kind: Service
  name: frontend 2
tls:
  termination: reencrypt 3
  key: [as in edge termination]
  certificate: [as in edge termination]
  caCertificate: [as in edge termination]
  destinationCACertificate: |- 4
    -----BEGIN CERTIFICATE-----
    [...]
    -----END CERTIFICATE-----

```

- 1** **2** The name of the object, which is limited to 63 characters.
- 3** The **termination** field is set to **reencrypt**. This is the only required **tls** field.
- 4** Required for re-encryption. **destinationCACertificate** specifies a CA certificate to validate the endpoint certificate, securing the connection from the router to the destination pods. If the service is using a service signing certificate, or the administrator has specified a default CA certificate for the router and the service has a certificate signed by that CA, this field can be omitted.

See **oc create route reencrypt --help** for more options.

- To create an **EventListener** resource using an insecure HTTP connection:
 - a. Create the **EventListener** resource.
 - b. Expose the **EventListener** service as an OpenShift Container Platform route to make it publicly accessible:

```
$ oc expose svc el-vote-app
```

1.8. CONFIGURING EVENT LISTENERS TO SERVE MULTIPLE NAMESPACE



NOTE

You can skip this section if you want to create a basic CI/CD pipeline. However, if your deployment strategy involves multiple namespaces, you can configure event listeners to serve multiple namespaces.

To increase reusability of **EventListener** objects, cluster administrators can configure and deploy them as multi-tenant event listeners that serve multiple namespaces.

Procedure

1. Configure cluster-wide fetch permission for the event listener.
 - a. Set a service account name to be used in the **ClusterRoleBinding** and **EventListener** objects. For example, **el-sa**.

Example ServiceAccount.yaml

```

apiVersion: v1
kind: ServiceAccount
metadata:
  name: el-sa
---
```

- b. In the **rules** section of the **ClusterRole.yaml** file, set appropriate permissions for every event listener deployment to function cluster-wide.

Example ClusterRole.yaml

```

kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: el-sel-clusterrole
rules:
- apiGroups: ["triggers.tekton.dev"]
  resources: ["eventlisteners", "clustertriggerbindings", "clusterinterceptors",
"triggerbindings", "triggertemplates", "triggers"]
  verbs: ["get", "list", "watch"]
- apiGroups: [""]
  resources: ["configmaps", "secrets"]
  verbs: ["get", "list", "watch"]
- apiGroups: [""]
  resources: ["serviceaccounts"]
  verbs: ["impersonate"]
...
```

- c. Configure cluster role binding with the appropriate service account name and cluster role name.

Example ClusterRoleBinding.yaml

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: el-mul-clusterrolebinding
subjects:
- kind: ServiceAccount
  name: el-sa
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: el-sel-clusterrole
...
```

2. In the **spec** parameter of the event listener, add the service account name, for example **el-sa**. Fill the **namespaceSelector** parameter with names of namespaces where event listener is intended to serve.

Example EventListener.yaml

```

apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: namespace-selector-listener
spec:
  taskRunTemplate:
    serviceAccountName: el-sa
  namespaceSelector:
    matchNames:
      - default
      - foo
  ...

```

3. Create a service account with the necessary permissions, for example **foo-trigger-sa**. Use it for role binding the triggers.

Example ServiceAccount.yaml

```

apiVersion: v1
kind: ServiceAccount
metadata:
  name: foo-trigger-sa
  namespace: foo
  ...

```

Example RoleBinding.yaml

```

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: triggercr-rolebinding
  namespace: foo
subjects:
  - kind: ServiceAccount
    name: foo-trigger-sa
    namespace: foo
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tekton-triggers-eventlistener-roles
  ...

```

4. Create a trigger with the appropriate trigger template, trigger binding, and service account name.

Example Trigger.yaml

```

apiVersion: triggers.tekton.dev/v1beta1
kind: Trigger
metadata:
  name: trigger
  namespace: foo
spec:
  taskRunTemplate:

```

```

serviceAccountName: foo-trigger-sa
interceptors:
- ref:
  name: "github"
  params:
  - name: "secretRef"
    value:
      secretName: github-secret
      secretKey: secretToken
  - name: "eventTypes"
    value: ["push"]
bindings:
- ref: vote-app
template:
  ref: vote-app
...

```

1.9. CREATING WEBHOOKS

Webhooks are HTTP POST messages that are received by the event listeners whenever a configured event occurs in your repository. The event payload is then mapped to trigger bindings, and processed by trigger templates. The trigger templates eventually start one or more pipeline runs, leading to the creation and deployment of Kubernetes resources.

In this section, you will configure a webhook URL on your forked Git repositories **pipelines-vote-ui** and **pipelines-vote-api**. This URL points to the publicly accessible **EventListener** service route.



NOTE

Adding webhooks requires administrative privileges to the repository. If you do not have administrative access to your repository, contact your system administrator for adding webhooks.

Procedure

1. Get the webhook URL:

- For a secure HTTPS connection:

```
$ echo "URL: $(oc get route el-vote-app --template='https://{{.spec.host}}')"
```

- For an HTTP (insecure) connection:

```
$ echo "URL: $(oc get route el-vote-app --template='http://{{.spec.host}}')"
```

Note the URL obtained in the output.

2. Configure webhooks manually on the front-end repository:
 - a. Open the front-end Git repository **pipelines-vote-ui** in your browser.
 - b. Click **Settings** → **Webhooks** → **Add Webhook**
 - c. On the **Webhooks/Add Webhook** page:

- i. Enter the webhook URL from step 1 in **Payload URL** field
 - ii. Select **application/json** for the **Content type**
 - iii. Specify the secret in the **Secret** field
 - iv. Ensure that the **Just the push event** is selected
 - v. Select **Active**
 - vi. Click **Add Webhook**
3. Repeat step 2 for the back-end repository **pipelines-vote-api**.

1.10. TRIGGERING A PIPELINE RUN

Whenever a **push** event occurs in the Git repository, the configured webhook sends an event payload to the publicly exposed **EventListener** service route. The **EventListener** service of the application processes the payload, and passes it to the relevant **TriggerBinding** and **TriggerTemplate** resource pairs. The **TriggerBinding** resource extracts the parameters, and the **TriggerTemplate** resource uses these parameters and specifies the way the resources must be created. This may rebuild and redeploy the application.

In this section, you push an empty commit to the front-end **pipelines-vote-ui** repository, which then triggers the pipeline run.

Procedure

1. From the terminal, clone your forked Git repository **pipelines-vote-ui**:

```
$ git clone git@github.com:<your GitHub ID>/pipelines-vote-ui.git -b pipelines-1.15
```

2. Push an empty commit:

```
$ git commit -m "empty-commit" --allow-empty && git push origin pipelines-1.15
```

3. Check if the pipeline run was triggered:

```
$ tkn pipelinerun list
```

Notice that a new pipeline run was initiated.

1.11. ENABLING MONITORING OF EVENT LISTENERS FOR TRIGGERS FOR USER-DEFINED PROJECTS

As a cluster administrator, to gather event listener metrics for the **Triggers** service in a user-defined project and display them in the OpenShift Container Platform web console, you can create a service monitor for each event listener. On receiving an HTTP request, event listeners for the **Triggers** service return three metrics – **eventlistener_http_duration_seconds**, **eventlistener_event_count**, and **eventlistener_triggered_resources**.

Prerequisites

- You have logged in to the OpenShift Container Platform web console.

- You have installed the Red Hat OpenShift Pipelines Operator.
- You have enabled monitoring for user-defined projects.

Procedure

1. For each event listener, create a service monitor. For example, to view the metrics for the **github-listener** event listener in the **test** namespace, create the following service monitor:

```

apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app.kubernetes.io/managed-by: EventListener
    app.kubernetes.io/part-of: Triggers
    eventlistener: github-listener
  annotations:
    networkoperator.openshift.io/ignore-errors: ""
  name: el-monitor
  namespace: test
spec:
  endpoints:
    - interval: 10s
      port: http-metrics
  jobLabel: name
  namespaceSelector:
    matchNames:
      - test
  selector:
    matchLabels:
      app.kubernetes.io/managed-by: EventListener
      app.kubernetes.io/part-of: Triggers
      eventlistener: github-listener
  ...

```

2. Test the service monitor by sending a request to the event listener. For example, push an empty commit:

```
$ git commit -m "empty-commit" --allow-empty && git push origin main
```

3. On the OpenShift Container Platform web console, navigate to **Administrator** → **Observe** → **Metrics**.
4. To view a metric, search by its name. For example, to view the details of the **eventlistener_http_resources** metric for the **github-listener** event listener, search using the **eventlistener_http_resources** keyword.

Additional resources

- [Enabling monitoring for user-defined projects](#)

1.12. CONFIGURING PULL REQUEST CAPABILITIES IN GITHUB INTERCEPTOR

With GitHub Interceptor, you can create logic that validates and filters GitHub webhooks. For example, you can validate the webhook's origin and filter incoming events based on specified criteria. When you use GitHub Interceptor to filter event data, you can specify the event types that Interceptor can accept in a field. In Red Hat OpenShift Pipelines, you can use the following capabilities of GitHub Interceptor:

- Filter pull request events based on the files that have been changed
- Validate pull requests based on configured GitHub owners

1.12.1. Filtering pull requests using GitHub Interceptor

You can filter GitHub events based on the files that have been changed for push and pull events. This helps you to execute a pipeline for only relevant changes in your Git repository. GitHub Interceptor adds a comma delimited list of all files that have been changed and uses the CEL Interceptor to filter incoming events based on the changed files. The list of changed files is added to the **changed_files** property of the event payload in the top-level **extensions** field.

Prerequisites

- You have installed the Red Hat OpenShift Pipelines Operator.

Procedure

1. Perform one of the following steps:

- For a public GitHub repository, set the value of the **addChangedFiles** parameter to **true** in the YAML configuration file shown below:

```

apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: github-add-changed-files-pr-listener
spec:
  triggers:
  - name: github-listener
    interceptors:
    - ref:
        name: "github"
        kind: ClusterInterceptor
        apiVersion: triggers.tekton.dev
        params:
        - name: "secretRef"
          value:
            secretName: github-secret
            secretKey: secretToken
        - name: "eventTypes"
          value: ["pull_request", "push"]
        - name: "addChangedFiles"
          value:
            enabled: true
    - ref:
        name: cel
        params:
        - name: filter
          value: extensions.changed_files.matches('controllers/')
  ...

```

- For a private GitHub repository, set the value of the **addChangedFiles** parameter to **true** and provide the access token details, **secretName** and **secretKey** in the YAML configuration file shown below:

```

apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: github-add-changed-files-pr-listener
spec:
  triggers:
  - name: github-listener
    interceptors:
    - ref:
        name: "github"
        kind: ClusterInterceptor
        apiVersion: triggers.tekton.dev
        params:
        - name: "secretRef"
          value:
            secretName: github-secret
            secretKey: secretToken
        - name: "eventTypes"
          value: ["pull_request", "push"]
        - name: "addChangedFiles"
          value:
            enabled: true
            personalAccessToken:
              secretName: github-pat
              secretKey: token
    - ref:
        name: cel
        params:
        - name: filter
          value: extensions.changed_files.matches('controllers/')
    ...

```

- Save the configuration file.

1.12.2. Validating pull requests using GitHub Interceptors

You can use GitHub Interceptor to validate the processing of pull requests based on the GitHub owners configured for a repository. This validation helps you to prevent unnecessary execution of a **PipelineRun** or **TaskRun** object. GitHub Interceptor processes a pull request only if the user name is listed as an owner or if a configurable comment is issued by an owner of the repository. For example, when you comment **/ok-to-test** on a pull request as an owner, a **PipelineRun** or **TaskRun** is triggered.



NOTE

Owners are configured in an **OWNERS** file at the root of the repository.

Prerequisites

- You have installed the Red Hat OpenShift Pipelines Operator.

Procedure

1. Create a secret string value.
2. Configure the GitHub webhook with that value.
3. Create a Kubernetes secret named **secretRef** that contains your secret value.
4. Pass the Kubernetes secret as a reference to your GitHub Interceptor.
5. Create an **owners** file and add the list of approvers into the **approvers** section.
6. Perform one of the following steps:
 - For a public GitHub repository, set the value of the **githubOwners** parameter to **true** in the YAML configuration file shown below:

```

apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: github-owners-listener
spec:
  triggers:
  - name: github-listener
    interceptors:
    - ref:
      name: "github"
      kind: ClusterInterceptor
      apiVersion: triggers.tekton.dev
    params:
    - name: "secretRef"
      value:
        secretName: github-secret
        secretKey: secretToken
    - name: "eventTypes"
      value: ["pull_request", "issue_comment"]
    - name: "githubOwners"
      value:
        enabled: true
        checkType: none
  ...

```

- For a private GitHub repository, set the value of the **githubOwners** parameter to **true** and provide the access token details, **secretName** and **secretKey** in the YAML configuration file shown below:

```

apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: github-owners-listener
spec:
  triggers:
  - name: github-listener
    interceptors:
    - ref:
      name: "github"

```

```

kind: ClusterInterceptor
apiVersion: triggers.tekton.dev
params:
- name: "secretRef"
  value:
    secretName: github-secret
    secretKey: secretToken
- name: "eventTypes"
  value: ["pull_request", "issue_comment"]
- name: "githubOwners"
  value:
    enabled: true
    personalAccessToken:
      secretName: github-token
      secretKey: secretToken
    checkType: all

```

...

**NOTE**

The **checkType** parameter is used to specify the GitHub owners who need authentication. You can set its value to **orgMembers**, **repoMembers**, or **all**.

7. Save the configuration file.

1.13. ADDITIONAL RESOURCES

- To include Pipelines as Code along with the application source code in the same repository, see [About Pipelines as Code](#).
- For more details on pipelines in the **Developer** perspective, see the [Working with OpenShift Pipelines in the web console](#) section.
- To learn more about Security Context Constraints (SCCs), see the [Managing Security Context Constraints](#) section.
- For more examples of reusable tasks, see the [OpenShift Catalog](#) repository. Additionally, you can also see the Tekton Catalog in the Tekton project.
- To install and deploy a custom instance of Tekton Hub for reusable tasks and pipelines, see [Using Tekton Hub with Red Hat OpenShift Pipelines](#).
- For more details on re-encrypt TLS termination, see [Re-encryption Termination](#).
- For more details on secured routes, see the [Secured routes](#) section.

CHAPTER 2. WORKING WITH RED HAT OPENSIFT PIPELINES IN THE WEB CONSOLE

You can use the **Administrator** or **Developer** perspective to create and modify **Pipeline**, **PipelineRun**, and **Repository** objects from the **Pipelines** page in the OpenShift Container Platform web console. You can also use the **+Add** page in the **Developer** perspective of the web console to create CI/CD pipelines for your software delivery process.

2.1. WORKING WITH RED HAT OPENSIFT PIPELINES IN THE DEVELOPER PERSPECTIVE

In the **Developer** perspective, you can access the following options for creating pipelines from the **+Add** page:

- Use the **+Add** → **Pipelines** → **Pipeline builder** option to create customized pipelines for your application.
- Use the **+Add** → **From Git** option to create pipelines using pipeline templates and resources while creating an application.

After you create the pipelines for your application, you can view and visually interact with the deployed pipelines in the **Pipelines** view. You can also use the **Topology** view to interact with the pipelines created using the **From Git** option. You must apply custom labels to pipelines created using the **Pipeline builder** to see them in the **Topology** view.

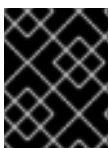
Prerequisites

- You have access to an OpenShift Container Platform cluster and have switched to [the Developer perspective](#).
- You have the [OpenShift Pipelines Operator installed](#) in your cluster.
- You are a cluster administrator or a user with create and edit permissions.
- You have created a project.

2.1.1. Constructing pipelines using the Pipeline builder

In the **Developer** perspective of the console, you can use the **+Add** → **Pipeline** → **Pipeline builder** option to:

- Configure pipelines using either the **Pipeline builder** or the **YAML view**.
- Construct a pipeline flow using existing tasks and cluster tasks. When you install the OpenShift Pipelines Operator, it adds reusable pipeline cluster tasks to your cluster.

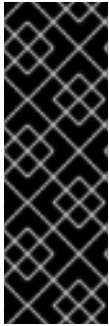


IMPORTANT

In Red Hat OpenShift Pipelines 1.10, **ClusterTask** functionality is deprecated and is planned to be removed in a future release.

- Specify the type of resources required for the pipeline run, and if required, add additional parameters to the pipeline.

- Reference these pipeline resources in each of the tasks in the pipeline as input and output resources.
- If required, reference any additional parameters added to the pipeline in the task. The parameters for a task are prepopulated based on the specifications of the task.
- Use the Operator-installed, reusable snippets and samples to create detailed pipelines.
- Search and add tasks from your configured local Tekton Hub instance.

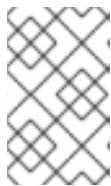


IMPORTANT

In the developer perspective, you can create a customized pipeline using your own set of curated tasks. To search, install, and upgrade your tasks directly from the developer console, your cluster administrator needs to install and deploy a local Tekton Hub instance and link that hub to the OpenShift Container Platform cluster. For more details, see *Using Tekton Hub with OpenShift Pipelines* in the *Additional resources* section. If you do not deploy any local Tekton Hub instance, by default, you can only access the cluster tasks, namespace tasks and public Tekton Hub tasks.

Procedure

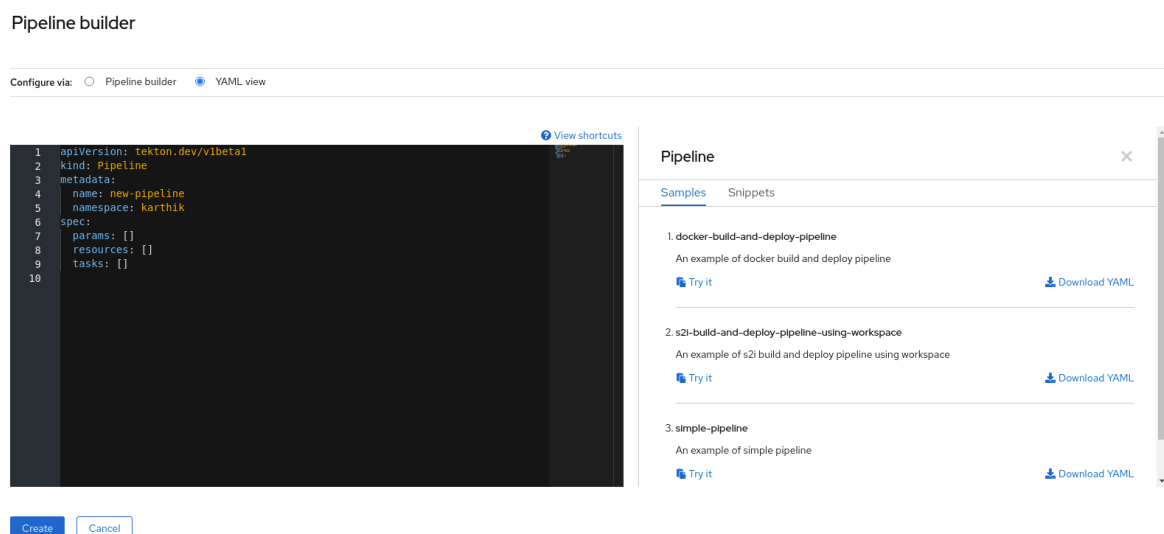
1. In the **+Add** view of the **Developer** perspective, click the **Pipeline** tile to see the **Pipeline builder** page.
2. Configure the pipeline using either the **Pipeline builder** view or the **YAML view**.



NOTE

The **Pipeline builder** view supports a limited number of fields whereas the **YAML view** supports all available fields. Optionally, you can also use the Operator-installed, reusable snippets and samples to create detailed pipelines.

Figure 2.1. YAML view



3. Configure your pipeline by using **Pipeline builder**:
 - a. In the **Name** field, enter a unique name for the pipeline.

b. In the **Tasks** section:

- i. Click **Add task**.
- ii. Search for a task using the quick search field and select the required task from the displayed list.
- iii. Click **Add** or **Install and add**. In this example, use the **s2i-nodejs** task.

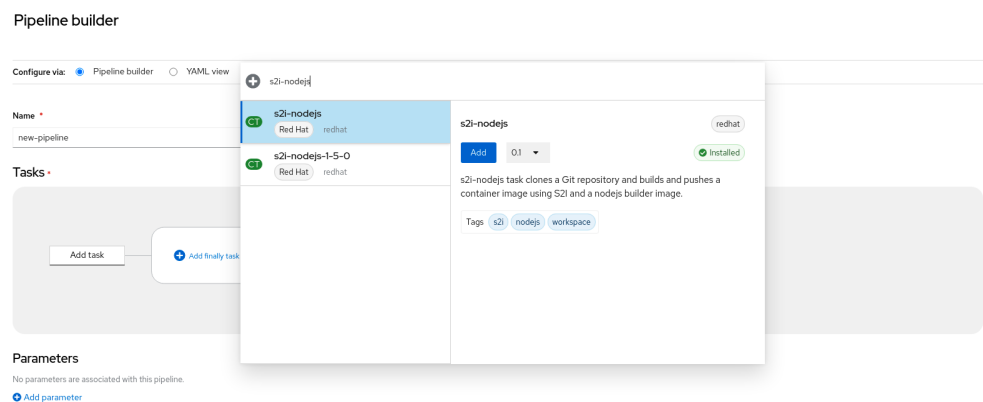


NOTE

The search list contains all the Tekton Hub tasks and tasks available in the cluster. Also, if a task is already installed it will show **Add** to add the task whereas it will show **Install and add** to install and add the task. It will show **Update and add** when you add the same task with an updated version.

- To add sequential tasks to the pipeline:
 - Click the plus icon to the right or left of the task → click **Add task**.
 - Search for a task using the quick search field and select the required task from the displayed list.
 - Click **Add** or **Install and add**.

Figure 2.2. Pipeline builder



- To add a final task:
 - Click the **Add finally task** → Click **Add task**.
 - Search for a task using the quick search field and select the required task from the displayed list.
 - Click **Add** or **Install and add**.
- c. In the **Resources** section, click **Add Resources** to specify the name and type of resources for the pipeline run. These resources are then used by the tasks in the pipeline as inputs and outputs. For this example:
- i. Add an input resource. In the **Name** field, enter **Source**, and then from the **Resource Type** drop-down list, select **Git**.

- ii. Add an output resource. In the **Name** field, enter **Img**, and then from the **Resource Type** drop-down list, select **Image**.

**NOTE**

A red icon appears next to the task if a resource is missing.

- d. Optional: The **Parameters** for a task are pre-populated based on the specifications of the task. If required, use the **Add Parameters** link in the **Parameters** section to add additional parameters.
 - e. In the **Workspaces** section, click **Add workspace** and enter a unique workspace name in the **Name** field. You can add multiple workspaces to the pipeline.
 - f. In the **Tasks** section, click the **s2i-nodejs** task to see the side panel with details for the task. In the task side panel, specify the resources and parameters for the **s2i-nodejs** task:
 - i. If required, in the **Parameters** section, add more parameters to the default ones, by using the `$(params.<param-name>)` syntax.
 - ii. In the **Image** section, enter **Img** as specified in the **Resources** section.
 - iii. Select a workspace from the **source** drop-down under **Workspaces** section.
 - g. Add resources, parameters, and workspaces to the **openshift-client** task.
4. Click **Create** to create and view the pipeline in the **Pipeline Details** page.
 5. Click the **Actions** drop-down menu then click **Start**, to see the **Start Pipeline** page.
 6. The **Workspaces** section lists the workspaces you created earlier. Use the respective drop-down to specify the volume source for your workspace. You have the following options: **Empty Directory**, **Config Map**, **Secret**, **PersistentVolumeClaim**, or **VolumeClaimTemplate**.

2.1.2. Creating OpenShift Pipelines along with applications

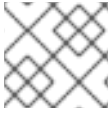
To create pipelines along with applications, use the **From Git** option in the **Add+** view of the **Developer** perspective. You can view all of your available pipelines and select the pipelines you want to use to create applications while importing your code or deploying an image.

The Tekton Hub Integration is enabled by default and you can see tasks from the Tekton Hub that are supported by your cluster. Administrators can opt out of the Tekton Hub Integration and the Tekton Hub tasks will no longer be displayed. You can also check whether a webhook URL exists for a generated pipeline. Default webhooks are added for the pipelines that are created using the **+Add** flow and the URL is visible in the side panel of the selected resources in the Topology view.

For more information, see [Creating applications using the Developer perspective](#).

2.1.3. Adding a GitHub repository containing pipelines

In the **Developer** perspective, you can add your GitHub repository containing pipelines to the OpenShift Container Platform cluster. This allows you to run pipelines and tasks from your GitHub repository on the cluster when relevant Git events, such as push or pull requests, are triggered.

**NOTE**

You can add both public and private GitHub repositories.

Prerequisites

- Ensure that your cluster administrator has configured the required GitHub applications in the administrator perspective.

Procedure

1. In the **Developer** perspective, choose the namespace or project in which you want to add your GitHub repository.
2. Navigate to **Pipelines** using the left navigation pane.
3. Click **Create** → **Repository** on the right side of the **Pipelines** page.
4. Enter your **Git Repo URL** and the console automatically fetches the repository name.
5. Click **Show configuration options**. By default, you see only one option **Setup a webhook**. If you have a GitHub application configured, you see two options:
 - **Use GitHub App**: Select this option to install your GitHub application in your repository.
 - **Setup a webhook**: Select this option to add a webhook to your GitHub application.
6. Set up a webhook using one of the following options in the **Secret** section:
 - Setup a webhook using **Git access token**
 - a. Enter your personal access token.
 - b. Click **Generate** corresponding to the **Webhook secret** field to generate a new webhook secret.

Project: openshift-pipelines ▾

Add Git Repository

Git Repo URL *

https://github.com/apps/pipelines-ci-clustername-ss-test

Name *

git-pipelines-ci-clustername-ss-test

▾ Hide configuration options

Secret


Git access token

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


Use your GitHub Personal token. Use this [link](#) to create a token with repo, public_repo & admin:repo_hook scopes and give your token an expiration, i.e 30d.

Git access token secret

Webhook secret

64bdd2115bab0219c2ac82fc13fbac63da3d9bb  Generate

▸ [See GitHub permissions](#)

[Read more about setting up webhook](#) 

Add Cancel



NOTE

You can click the link below the **Git access token** field if you do not have a personal access token and want to create a new one.

- Setup a webhook using **Git access token secret**
 - Select a secret in your namespace from the dropdown list. Depending on the secret you selected, a webhook secret is automatically generated.

Project: openshift-pipelines ▾

Add Git Repository

Git Repo URL *

https://github.com/apps/pipelines-ci-clustername-ss-test

Name *


git-pipelines-ci-clustername-ss-test

▾ Hide configuration options

Secret


Git access token

Git access token secret

 pipelines-as-code-secret ▾

Secret with the Git access token for pulling pipeline and tasks from your Git repository.

Webhook secret

64bdd2115bab0219c2ac82fc13fbbac63da3d9bb 

▸ [See GitHub permissions](#)

[Read more about setting up webhook](#)

7. Add the webhook secret details to your GitHub repository:
 - a. Copy the **webhook URL** and navigate to your GitHub repository settings.
 - b. Click **Webhooks → Add webhook**.
 - c. Copy the **Webhook URL** from the developer console and paste it in the **Payload URL** field of the GitHub repository settings.
 - d. Select the **Content type**.
 - e. Copy the **Webhook secret** from the developer console and paste it in the **Secret** field of the GitHub repository settings.
 - f. Select one of the **SSL verification** options.
 - g. Select the events to trigger this webhook.
 - h. Click **Add webhook**.
8. Navigate back to the developer console and click **Add**.
9. Read the details of the steps that you have to perform and click **Close**.
10. View the details of the repository you just created.

**NOTE**

When importing an application using **Import from Git** and the Git repository has a **.tekton** directory, you can configure **pipelines-as-code** for your application.

2.1.4. Interacting with pipelines using the Developer perspective

The **Pipelines** view in the **Developer** perspective lists all the pipelines in a project, along with the following details:

- The namespace in which the pipeline was created
- The last pipeline run
- The status of the tasks in the pipeline run
- The status of the pipeline run
- The creation time of the last pipeline run

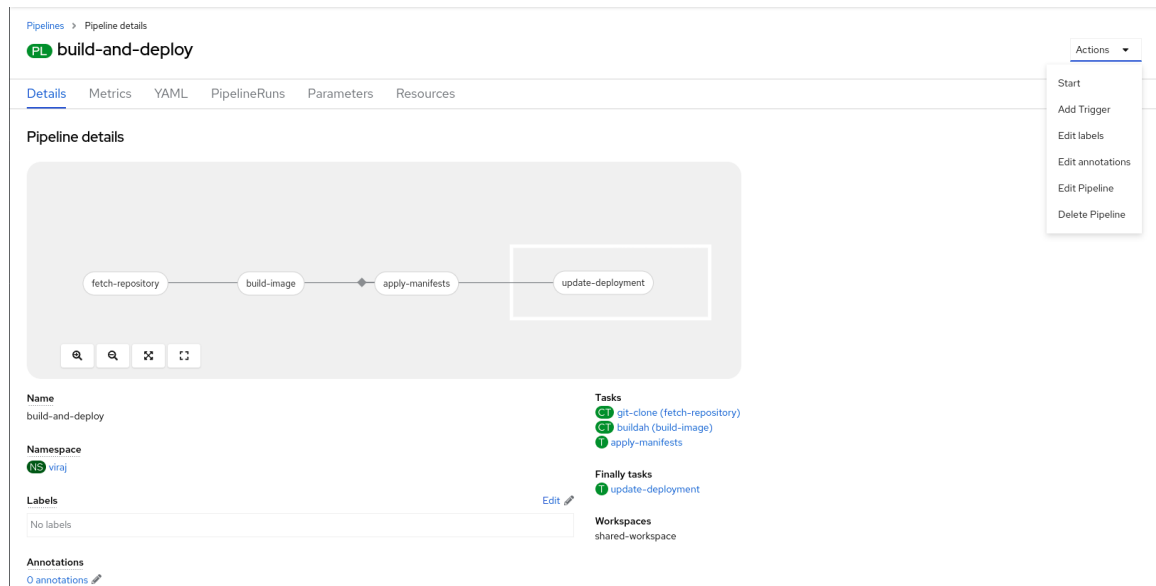
Procedure

1. In the **Pipelines** view of the **Developer** perspective, select a project from the **Project** drop-down list to see the pipelines in that project.
2. Click the required pipeline to see the **Pipeline details** page.
By default, the **Details** tab displays a visual representation of all the **serial** tasks, **parallel** tasks, **finally** tasks, and **when** expressions in the pipeline. The tasks and the **finally** tasks are listed in the lower right portion of the page.

To view the task details, click the listed **Tasks** and **Finally** tasks. In addition, you can do the following:

- Use the zoom in, zoom out, fit to screen, and reset view features using the standard icons displayed in the lower left corner of the **Pipeline details** visualization.
- Change the zoom factor of the pipeline visualization using the mouse wheel.
- Hover over the tasks and see the task details.

Figure 2.3. Pipeline details



3. Optional: On the **Pipeline details** page, click the **Metrics** tab to see the following information about pipelines:

- **Pipeline Success Ratio**
- **Number of Pipeline Runs**
- **Pipeline Run Duration**
- **Task Run Duration**

You can use this information to improve the pipeline workflow and eliminate issues early in the pipeline lifecycle.

4. Optional: Click the **YAML** tab to edit the YAML file for the pipeline.

5. Optional: Click the **Pipeline Runs** tab to see the completed, running, or failed runs for the pipeline.

The **Pipeline Runs** tab provides details about the pipeline run, the status of the task, and a link



to debug failed pipeline runs. Use the Options menu to stop a running pipeline, to rerun a pipeline using the same parameters and resources as that of the previous pipeline execution, or to delete a pipeline run.

- Click the required pipeline run to see the **Pipeline Run details** page. By default, the **Details** tab displays a visual representation of all the serial tasks, parallel tasks, **finally** tasks, and when expressions in the pipeline run. The results for successful runs are displayed under the **Pipeline Run results** pane at the bottom of the page. Additionally, you would only be able to see tasks from Tekton Hub which are supported by the cluster. While looking at a task, you can click the link beside it to jump to the task documentation.




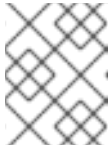
NOTE

The **Details** section of the **Pipeline Run Details** page displays a **Log Snippet** of the failed pipeline run. **Log Snippet** provides a general error message and a snippet of the log. A link to the **Logs** section provides quick access to the details about the failed run.

- On the **Pipeline Run details** page, click the **Task Runs** tab to see the completed, running, and failed runs for the task.

The **Task Runs** tab provides information about the task run along with the links to its task

and pod, and also the status and duration of the task run. Use the Options menu  to delete a task run.



NOTE

The **TaskRuns** list page features a **Manage columns** button, which you can also use to add a **Duration** column.

- Click the required task run to see the **Task Run details** page. The results for successful runs are displayed under the **Task Run results** pane at the bottom of the page.



NOTE

The **Details** section of the **Task Run details** page displays a **Log Snippet** of the failed task run. **Log Snippet** provides a general error message and a snippet of the log. A link to the **Logs** section provides quick access to the details about the failed task run.


6. Click the **Parameters** tab to see the parameters defined in the pipeline. You can also add or edit additional parameters, as required.
7. Click the **Resources** tab to see the resources defined in the pipeline. You can also add or edit additional resources, as required.

2.1.5. Starting pipelines from Pipelines view

After you create a pipeline, you need to start it to execute the included tasks in the defined sequence. You can start a pipeline from the **Pipelines** view, the **Pipeline Details** page, or the **Topology** view.

Procedure

To start a pipeline using the **Pipelines** view:

1. In the **Pipelines** view of the **Developer** perspective, click the **Options**  menu adjoining a pipeline, and select **Start**.
2. The **Start Pipeline** dialog box displays the **Git Resources** and the **Image Resources** based on the pipeline definition.



NOTE

For pipelines created using the **From Git** option, the **Start Pipeline** dialog box also displays an **APP_NAME** field in the **Parameters** section, and all the fields in the dialog box are prepopulated by the pipeline template.

- a. If you have resources in your namespace, the **Git Resources** and the **Image Resources** fields are prepopulated with those resources. If required, use the drop-downs to select or create the required resources and customize the pipeline run instance.

3. Optional: Modify the **Advanced Options** to add the credentials that authenticate the specified private Git server or the image registry.
 - a. Under **Advanced Options**, click **Show Credentials Options** and select **Add Secret**.
 - b. In the **Create Source Secret** section, specify the following:
 - i. A unique **Secret Name** for the secret.
 - ii. In the **Designated provider to be authenticated** section, specify the provider to be authenticated in the **Access to** field, and the base **Server URL**.
 - iii. Select the **Authentication Type** and provide the credentials:
 - For the **Authentication Type Image Registry Credentials**, specify the **Registry Server Address** that you want to authenticate, and provide your credentials in the **Username**, **Password**, and **Email** fields.
Select **Add Credentials** if you want to specify an additional **Registry Server Address**.
 - For the **Authentication Type Basic Authentication**, specify the values for the **UserName** and **Password or Token** fields.
 - For the **Authentication Type SSH Keys**, specify the value of the **SSH Private Key** field.



NOTE

For basic authentication and SSH authentication, you can use annotations such as:

- **tekton.dev/git-0: <https://github.com>**
- **tekton.dev/git-1: <https://gitlab.com>**.

- iv. Select the check mark to add the secret.

You can add multiple secrets based upon the number of resources in your pipeline.

4. Click **Start** to start the pipeline.
5. The **PipelineRun details** page displays the pipeline being executed. After the pipeline starts, the tasks and steps within each task are executed. You can:
 - Use the zoom in, zoom out, fit to screen, and reset view features using the standard icons, which are in the lower left corner of the **PipelineRun details** page visualization.
 - Change the zoom factor of the pipelinerun visualization using the mouse wheel. At specific zoom factors, the background color of the tasks changes to indicate the error or warning status.
 - Hover over the tasks to see the details, such as the time taken to execute each step, task name, and task status.
 - Hover over the tasks badge to see the total number of tasks and tasks completed.
 - Click on a task to see the logs for each step in the task.

- Click the **Logs** tab to see the logs relating to the execution sequence of the tasks. You can also expand the pane and download the logs individually or in bulk, by using the relevant button.
- Click the **Events** tab to see the stream of events generated by a pipeline run. You can use the **Task Runs**, **Logs**, and **Events** tabs to assist in debugging a failed pipeline run or a failed task run.

Figure 2.4. Pipeline run details

2.1.6. Starting pipelines from Topology view

For pipelines created using the **From Git** option, you can use the **Topology** view to interact with pipelines after you start them:



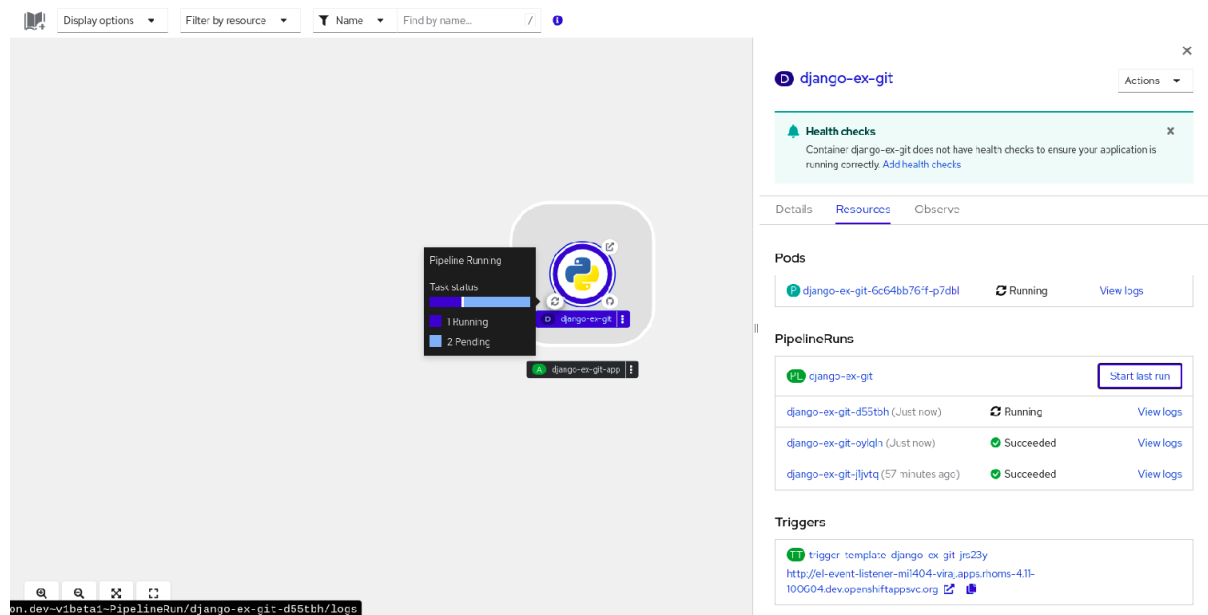
NOTE

To see the pipelines created using **Pipeline builder** in the **Topology** view, customize the pipeline labels to link the pipeline with the application workload.

Procedure

1. Click **Topology** in the left navigation panel.
2. Click the application to display **Pipeline Runs** in the side panel.
3. In **Pipeline Runs**, click **Start Last Run** to start a new pipeline run with the same parameters and resources as the previous one. This option is disabled if a pipeline run has not been initiated. You can also start a pipeline run when you create it.

Figure 2.5. Pipelines in Topology view



In the **Topology** page, hover to the left of the application to see the status of its pipeline run. After a pipeline is added, a bottom left icon indicates that there is an associated pipeline.

2.1.7. Interacting with pipelines from Topology view

The side panel of the application node in the **Topology** page displays the status of a pipeline run and you can interact with it.

- If a pipeline run does not start automatically, the side panel displays a message that the pipeline cannot be automatically started, hence it would need to be started manually.
- If a pipeline is created but the user has not started the pipeline, its status is not started. When the user clicks the **Not started** status icon, the start dialog box opens in the **Topology** view.
- If the pipeline has no build or build config, the **Builds** section is not visible. If there is a pipeline and build config, the **Builds section** is visible.
- The side panel displays a **Log Snippet** when a pipeline run fails on a specific task run. You can view the **Log Snippet** in the **Pipeline Runs** section, under the **Resources** tab. It provides a general error message and a snippet of the log. A link to the **Logs** section provides quick access to the details about the failed run.

2.1.8. Editing pipelines

You can edit the pipelines in your cluster using the **Developer** perspective of the web console:

Procedure


1. In the **Pipelines** view of the **Developer** perspective, select the pipeline you want to edit to see the details of the pipeline. In the **Pipeline Details** page, click **Actions** and select **Edit Pipeline**.
2. On the **Pipeline builder** page, you can perform the following tasks:
 - Add additional tasks, parameters, or resources to the pipeline.

- Click the task you want to modify to see the task details in the side panel and modify the required task details, such as the display name, parameters, and resources.
 - Alternatively, to delete the task, click the task, and in the side panel, click **Actions** and select **Remove Task**.
3. Click **Save** to save the modified pipeline.

2.1.9. Deleting pipelines

You can delete the pipelines in your cluster using the **Developer** perspective of the web console.

Procedure

1. In the **Pipelines** view of the **Developer** perspective, click the **Options**  menu adjoining a Pipeline, and select **Delete Pipeline**.
2. In the **Delete Pipeline** confirmation prompt, click **Delete** to confirm the deletion.

2.2. ADDITIONAL RESOURCES

- [Using Tekton Hub with OpenShift Pipelines](#)


2.3. CREATING PIPELINE TEMPLATES IN THE ADMINISTRATOR PERSPECTIVE

As a cluster administrator, you can create pipeline templates that developers can reuse when they create a pipeline on the cluster.

Prerequisites

- You have access to an OpenShift Container Platform cluster with cluster administrator permissions, and have switched to the **Administrator** perspective.
- You have installed the OpenShift Pipelines Operator in your cluster.

Procedure

1. Navigate to the **Pipelines** page to view existing pipeline templates.
2. Click the  icon to go to the **Import YAML** page.
3. Add the YAML for your pipeline template. The template must include the following information:

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
# ...
namespace: openshift 1
labels:
```

```
pipeline.openshift.io/runtime: <runtime> 2
pipeline.openshift.io/type: <pipeline-type> 3
# ...
```

- 1 The template must be created in the **openshift** namespace.
 - 2 The template must contain the **pipeline.openshift.io/runtime** label. The accepted runtime values for this label are **nodejs, golang, dotnet, java, php, ruby, perl, python, nginx**, and **httpd**.
 - 3 The template must contain the **pipeline.openshift.io/type:** label. The accepted type values for this label are **openshift, knative**, and **kubernetes**.
4. Click **Create**. After the pipeline has been created, you are taken to the **Pipeline details** page, where you can view information about or edit your pipeline.

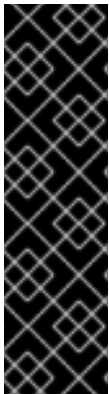
2.4. PIPELINE EXECUTION STATISTICS IN THE WEB CONSOLE

You can view statistics related to execution of pipelines in the web console.

To view the statistic information, you must complete the following steps:

- Install Tekton Results. For more information about installing Tekton Results, see *Using Tekton Results for OpenShift Pipelines observability* in the *Additional resources* section.
- Enable the OpenShift Pipelines console plugin.

Statistic information is available for all pipelines together and for each individual pipeline.



IMPORTANT

The OpenShift Pipelines console plugin is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

Additional resources

- [Using Tekton Results for OpenShift Pipelines observability](#)

2.4.1. Enabling the OpenShift Pipelines console plugin

To view the statistic information, you must first enable the OpenShift Pipelines console plugin.

Prerequisites

- You installed the Red Hat OpenShift Pipelines Operator in your cluster.
- You are logged on to the web console with cluster administrator permissions.



IMPORTANT

The OpenShift Pipelines console plugin requires OpenShift Container Platform version 4.15 or a later version.

Procedure

1. In the **Administrator** perspective of the web console, select **Operators → Installed Operators**.
2. Click **Red Hat OpenShift Pipelines** in the table of Operators.
3. In the right pane on the screen, check the status label under **Console plugin**. The label is either **Enabled** or **Disabled**.
4. If the label is **Disabled**, click this label. In the window that displays, select **Enable** and then click **Save**.

2.4.2. Viewing the statistics for all pipelines together

You can view consolidated statistic information related to all pipelines on the system.

Prerequisites

- You installed the Red Hat OpenShift Pipelines Operator in your cluster.
- You installed Tekton Results.
- You installed the OpenShift Pipelines web console plugin.

Procedure

1. In the **Administrator** perspective of the web console, select **Pipelines → Overview**.
A statistics overview displays. This overview includes the following information: **A graph reflecting the number and status of pipeline runs over a time period** The total, average, and maximum durations of pipeline execution over the same period. ****** The total number of pipeline runs over the same period.

A table of pipelines also displays. This table lists all pipelines that were run in the time period, showing their duration and success rate.

2. Optional: Change the settings of the statistics display as necessary:
 - **Project:** The project or namespace to display statistics for.
 - **Time range:** The time period to display statistics for.
 - **Refresh interval:** How often Red Hat OpenShift Pipelines must update the data in the window while you are viewing it.

2.4.3. Viewing the statistics for a specific pipeline

You can view statistic information related to a particular pipeline.

Prerequisites

- You installed the Red Hat OpenShift Pipelines Operator in your cluster.
- You installed Tekton Results.
- You installed the OpenShift Pipelines web console plugin.

Procedure

1. In the **Administrator** perspective of the web console, select **Pipelines → Pipelines**.
2. Click a pipeline in the list of pipelines. The **Pipeline details** view displays.
3. Click the **Metrics** tab.
A statistics overview displays. This overview includes the following information: **A graph reflecting the number and status of pipeline runs over a time period** The total, average, and maximum durations of pipeline execution over the same period. ****** The total number of pipeline runs over the same period.
4. Optional: Change the settings of the statistics display as necessary:
 - **Project:** The project or namespace to display statistics for.
 - **Time range:** The time period to display statistics for.
 - **Refresh interval:** How often Red Hat OpenShift Pipelines must update the data in the window while you are viewing it.

CHAPTER 3. SPECIFYING REMOTE PIPELINES AND TASKS USING RESOLVERS

Pipelines and tasks are reusable blocks for your CI/CD processes. You can reuse pipelines or tasks that you previously developed, or that were developed by others, without having to copy and paste their definitions. These pipelines or tasks can be available from several types of sources, from other namespaces on your cluster to public catalogs.

In a pipeline run resource, you can specify a pipeline from an existing source. In a pipeline resource or a task run resource, you can specify a task from an existing source.

In these cases, the *resolvers* in Red Hat OpenShift Pipelines retrieve the pipeline or task definition from the specified source at run time.

The following resolvers are available in a default installation of Red Hat OpenShift Pipelines:

Hub resolver

Retrieves a task or pipeline from the Pipelines Catalog available on Artifact Hub or Tekton Hub.

Bundles resolver

Retrieves a task or pipeline from a Tekton bundle, which is an OCI image available from any OCI repository, such as an OpenShift container repository.

Cluster resolver

Retrieves a task or pipeline that is already created on the same OpenShift Container Platform cluster in a specific namespace.

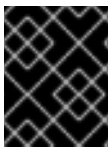
Git resolver

Retrieves a task or pipeline binding from a Git repository. You must specify the repository, the branch, and the path.

An OpenShift Pipelines installation includes a set of standard tasks that you can use in your pipelines. These tasks are located in the OpenShift Pipelines installation namespace, which is normally the **openshift-pipelines** namespace. You can use the cluster resolver to access the tasks.

3.1. SPECIFYING A REMOTE PIPELINE OR TASK FROM A TEKTON CATALOG

You can use the hub resolver to specify a remote pipeline or task that is defined either in a public Tekton catalog of [Artifact Hub](#) or in an instance of Tekton Hub.



IMPORTANT

The Artifact Hub project is not supported with Red Hat OpenShift Pipelines. Only the configuration of Artifact Hub is supported.

3.1.1. Configuring the hub resolver

You can change the default hub for pulling a resource, and the default catalog settings, by configuring the hub resolver.

Procedure

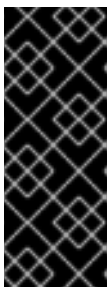
1. To edit the **TektonConfig** custom resource, enter the following command:


```
$ oc edit TektonConfig config
```

- In the **TektonConfig** custom resource, edit the **pipeline.hub-resolver-config** spec:

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    hub-resolver-config:
      default-tekton-hub-catalog: Tekton 1
      default-artifact-hub-task-catalog: tekton-catalog-tasks 2
      default-artifact-hub-pipeline-catalog: tekton-catalog-pipelines 3
      default-kind: pipeline 4
      default-type: tekton 5
      tekton-hub-api: "https://my-custom-tekton-hub.example.com" 6
      artifact-hub-api: "https://my-custom-artifact-hub.example.com" 7
```

- The default Tekton Hub catalog for pulling a resource.
- The default Artifact Hub catalog for pulling a task resource.
- The default Artifact Hub catalog for pulling a pipeline resource.
- The default object kind for references.
- The default hub for pulling a resource, either **artifact** for Artifact Hub or **tekton** for Tekton Hub.
- The Tekton Hub API used, if the **default-type** option is set to **tekton**.
- Optional: The Artifact Hub API used, if the **default-type** option is set to **artifact**.



IMPORTANT

If you set the **default-type** option to **tekton**, you must configure your own instance of the Tekton Hub by setting the **tekton-hub-api** value.

If you set the **default-type** option to **artifact** then the resolver uses the public hub API at <https://artifacthub.io/> by default. You can configure your own Artifact Hub API by setting the **artifact-hub-api** value.

3.1.2. Specifying a remote pipeline or task using the hub resolver

When creating a pipeline run, you can specify a remote pipeline from Artifact Hub or Tekton Hub. When creating a pipeline or a task run, you can specify a remote task from Artifact Hub or Tekton Hub.

Procedure

- To specify a remote pipeline or task from Artifact Hub or Tekton Hub, use the following reference format in the **pipelineRef** or **taskRef** spec:

```
# ...
  resolver: hub
  params:
  - name: catalog
    value: <catalog>
  - name: type
    value: <catalog_type>
  - name: kind
    value: [pipeline|task]
  - name: name
    value: <resource_name>
  - name: version
    value: <resource_version>
# ...
```

Table 3.1. Supported parameters for the hub resolver

Parameter	Description	Example value
catalog	The catalog for pulling the resource.	Default: tekton-catalog-tasks (for the task kind); tekton-catalog-pipelines (for the pipeline kind).
type	The type of the catalog for pulling the resource. Either artifact for Artifact Hub or tekton for Tekton Hub.	Default: artifact
kind	Either task or pipeline .	Default: task
name	The name of the task or pipeline to fetch from the hub.	golang-build
version	The version of the task or pipeline to fetch from the hub. You must use quotes (") around the number.	"0.5.0"

If the pipeline or task requires additional parameters, specify values for these parameters in the **params** section of the specification of the pipeline, pipeline run, or task run. The **params** section of the **pipelineRef** or **taskRef** specification must contain only the parameters that the resolver supports.

The following example pipeline run references a remote pipeline from a catalog:

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: hub-pipeline-reference-demo
spec:
  pipelineRef:
```

```

resolver: hub
params:
- name: catalog
  value: tekton-catalog-pipelines
- name: type
  value: artifact
- name: kind
  value: pipeline
- name: name
  value: example-pipeline
- name: version
  value: "0.1"
params:
- name: sample-pipeline-parameter
  value: test

```

The following example pipeline that references a remote task from a catalog:

```

apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: pipeline-with-cluster-task-reference-demo
spec:
  tasks:
- name: "cluster-task-reference-demo"
  taskRef:
    resolver: hub
    params:
- name: catalog
  value: tekton-catalog-tasks
- name: type
  value: artifact
- name: kind
  value: task
- name: name
  value: example-task
- name: version
  value: "0.6"
  params:
- name: sample-task-parameter
  value: test

```

The following example task run that references a remote task from a catalog:

```

apiVersion: tekton.dev/v1
kind: TaskRun
metadata:
  name: cluster-task-reference-demo
spec:
  taskRef:
    resolver: hub
    params:
- name: catalog
  value: tekton-catalog-tasks
- name: type

```

```

value: artifact
- name: kind
value: task
- name: name
value: example-task
- name: version
value: "0.6"
params:
- name: sample-task-parameter
value: test

```

3.2. SPECIFYING A REMOTE PIPELINE OR TASK FROM A TEKTON BUNDLE

You can use the bundles resolver to specify a remote pipeline or task from a Tekton bundle. A Tekton bundle is an OCI image available from any OCI repository, such as an OpenShift container repository.

3.2.1. Configuring the bundles resolver

You can change the default service account name and the default kind for pulling resources from a Tekton bundle by configuring the bundles resolver.

Procedure

1. To edit the **TektonConfig** custom resource, enter the following command:

```
$ oc edit TektonConfig config
```

2. In the **TektonConfig** custom resource, edit the **pipeline.bundles-resolver-config** spec:

```

apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    bundles-resolver-config:
      default-service-account: pipelines 1
      default-kind: task 2

```

1 The default service account name to use for bundle requests.

2 The default layer kind in the bundle image.

3.2.2. Specifying a remote pipeline or task using the bundles resolver

When creating a pipeline run, you can specify a remote pipeline from a Tekton bundle. When creating a pipeline or a task run, you can specify a remote task from a Tekton bundle.

Procedure

- To specify a remote pipeline or task from a Tekton bundle, use the following reference format in the **pipelineRef** or **taskRef** spec:

```
# ...
resolver: bundles
params:
- name: bundle
  value: <fully_qualified_image_name>
- name: name
  value: <resource_name>
- name: kind
  value: [pipeline|task]
# ...
```

Table 3.2. Supported parameters for the bundles resolver

Parameter	Description	Example value
serviceAccount	The name of the service account to use when constructing registry credentials.	default
bundle	The bundle URL pointing at the image to fetch.	gcr.io/tekton-releases/catalog/upstream/golang-build:0.1
name	The name of the resource to pull out of the bundle.	golang-build
kind	The kind of the resource to pull out of the bundle.	task

If the pipeline or task requires additional parameters, specify values for these parameters in the **params** section of the specification of the pipeline, pipeline run, or task run. The **params** section of the **pipelineRef** or **taskRef** specification must contain only the parameters that the resolver supports.

The following example pipeline run references a remote pipeline from a Tekton bundle:

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: bundle-pipeline-reference-demo
spec:
  pipelineRef:
    resolver: bundles
  params:
  - name: bundle
    value: registry.example.com:5000/simple/pipeline:latest
  - name: name
    value: hello-pipeline
  - name: kind
```

```
  value: pipeline
  params:
  - name: sample-pipeline-parameter
    value: test
  - name: username
    value: "pipelines"
```

The following example pipeline references a remote task from a Tekton bundle:

```
apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: pipeline-with-bundle-task-reference-demo
spec:
  tasks:
  - name: "bundle-task-demo"
    taskRef:
      resolver: bundles
      params:
      - name: bundle
        value: registry.example.com:5000/advanced/task:latest
      - name: name
        value: hello-world
      - name: kind
        value: task
    params:
    - name: sample-task-parameter
      value: test
```

The following example task run references a remote task from a Tekton bundle:

```
apiVersion: tekton.dev/v1
kind: TaskRun
metadata:
  name: bundle-task-reference-demo
spec:
  taskRef:
    resolver: bundles
    params:
    - name: bundle
      value: registry.example.com:5000/simple/new_task:latest
    - name: name
      value: hello-world
    - name: kind
      value: task
  params:
  - name: sample-task-parameter
    value: test
```

3.3. SPECIFYING A REMOTE PIPELINE OR TASK FROM A GIT REPOSITORY

You can use the Git resolver to specify a remote pipeline or task from a Git repository. The repository must contain a YAML file that defines the pipeline or task. The Git resolver can access a repository either by cloning it anonymously or by using the authenticated SCM API.

3.3.1. Configuring the Git resolver for anonymous Git cloning

If you want to use anonymous Git cloning, you can configure the default Git revision, fetch timeout, and default repository URL for pulling remote pipelines and tasks from a Git repository.

Procedure

1. To edit the **TektonConfig** custom resource, enter the following command:

```
$ oc edit TektonConfig config
```

2. In the **TektonConfig** custom resource, edit the **pipeline.git-resolver-config** spec:

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    git-resolver-config:
      default-revision: main 1
      fetch-timeout: 1m 2
      default-url: https://github.com/tektoncd/catalog.git 3
```

- 1** The default Git revision to use if none is specified.
- 2** The maximum time any single Git clone resolution may take, for example, **1m, 2s, 700ms**. Red Hat OpenShift Pipelines also enforces a global maximum timeout of 1 minute on all resolution requests.
- 3** The default Git repository URL for anonymous cloning if none is specified.

3.3.2. Configuring the Git resolver for the authenticated SCM API

For the authenticated SCM API, you must set the configuration for the authenticated Git connection.

You can use Git repository providers that are supported by the **go-scm** library. Not all **go-scm** implementations have been tested with the Git resolver, but the following providers are known to work:

- **github.com** and GitHub Enterprise
- **gitlab.com** and self-hosted Gitlab
- Gitea
- BitBucket Server
- BitBucket Cloud



NOTE

- You can configure only one Git connection using the authenticated SCM API for your cluster. This connection becomes available to all users of the cluster. All users of the cluster can access the repository using the security token that you configure for the connection.
- If you configure the Git resolver to use the authenticated SCM API, you can also use anonymous Git clone references to retrieve pipelines and tasks.

Procedure

1. To edit the **TektonConfig** custom resource, enter the following command:

```
$ oc edit TektonConfig config
```

2. In the **TektonConfig** custom resource, edit the **pipeline.git-resolver-config** spec:

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    git-resolver-config:
      default-revision: main 1
      fetch-timeout: 1m 2
      scm-type: github 3
      server-url: api.internal-github.com 4
      api-token-secret-name: github-auth-secret 5
      api-token-secret-key: github-auth-key 6
      api-token-secret-namespace: github-auth-namespace 7
      default-org: tektoncd 8
```

- 1 The default Git revision to use if none is specified.
- 2 The maximum time any single Git clone resolution may take, for example, **1m, 2s, 700ms**. Red Hat OpenShift Pipelines also enforces a global maximum timeout of 1 minute on all resolution requests.
- 3 The SCM provider type.
- 4 The base URL for use with the authenticated SCM API. This setting is not required if you are using **github.com**, **gitlab.com**, or BitBucket Cloud.
- 5 The name of the secret that contains the SCM provider API token.
- 6 The key within the token secret that contains the token.
- 7 The namespace containing the token secret, if not **default**.
- 8 Optional: The default organization for the repository, when using the authenticated API. This organization is used if you do not specify an organization in the resolver parameters.



NOTE

The **scm-type**, **api-token-secret-name**, and **api-token-secret-key** settings are required to use the authenticated SCM API.

3.3.3. Specifying a remote pipeline or task using the Git resolver

When creating a pipeline run, you can specify a remote pipeline from a Git repository. When creating a pipeline or a task run, you can specify a remote task from a Git repository.

Prerequisites

- If you want to use the authenticated SCM API, you must configure the authenticated Git connection for the Git resolver.

Procedure

1. To specify a remote pipeline or task from a Git repository, use the following reference format in the **pipelineRef** or **taskRef** spec:

```
# ...
resolver: git
params:
- name: url
  value: <git_repository_url>
- name: revision
  value: <branch_name>
- name: pathInRepo
  value: <path_in_repository>
# ...
```

Table 3.3. Supported parameters for the Git resolver

Parameter	Description	Example value
url	The URL of the repository, when using anonymous cloning.	https://github.com/tektoncd/catalog.git
repo	The repository name, when using the authenticated SCM API.	test-infra
org	The organization for the repository, when using the authenticated SCM API.	tektoncd
revision	The Git revision in the repository. You can specify a branch name, a tag name, or a commit SHA hash.	aeb957601cf41c012be462827053a21a420befca main v0.38.2

Parameter	Description	Example value
pathInRepo	The path name of the YAML file in the repository.	task/golang-build/0.3/golang-build.yaml

**NOTE**

To clone and fetch the repository anonymously, use the **url** parameter. To use the authenticated SCM API, use the **repo** parameter. Do not specify the **url** parameter and the **repo** parameter together.

If the pipeline or task requires additional parameters, specify values for these parameters in the **params** section of the specification of the pipeline, pipeline run, or task run. The **params** section of the **pipelineRef** or **taskRef** specification must contain only the parameters that the resolver supports.

The following example pipeline run references a remote pipeline from a Git repository:

```

apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: git-pipeline-reference-demo
spec:
  pipelineRef:
    resolver: git
    params:
      - name: url
        value: https://github.com/tektoncd/catalog.git
      - name: revision
        value: main
      - name: pathInRepo
        value: pipeline/simple/0.1/simple.yaml
  params:
    - name: name
      value: "testPipelineRun"
    - name: sample-pipeline-parameter
      value: test

```

The following example pipeline references a remote task from a Git repository:

```

apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: pipeline-with-git-task-reference-demo
spec:
  tasks:
    - name: "git-task-reference-demo"
      taskRef:
        resolver: git
        params:

```

```

- name: url
  value: https://github.com/tektoncd/catalog.git
- name: revision
  value: main
- name: pathInRepo
  value: task/git-clone/0.6/git-clone.yaml
params:
- name: sample-task-parameter
  value: test

```

The following example task run references a remote task from a Git repository:

```

apiVersion: tekton.dev/v1
kind: TaskRun
metadata:
  name: git-task-reference-demo
spec:
  taskRef:
    resolver: git
    params:
    - name: url
      value: https://github.com/tektoncd/catalog.git
    - name: revision
      value: main
    - name: pathInRepo
      value: task/git-clone/0.6/git-clone.yaml
  params:
  - name: sample-task-parameter
    value: test

```

3.4. SPECIFYING A REMOTE PIPELINE OR TASK FROM THE SAME CLUSTER

You can use the cluster resolver to specify a remote pipeline or task that is defined in a namespace on the OpenShift Container Platform cluster where Red Hat OpenShift Pipelines is running.

In particular, you can use the cluster resolver to access tasks that OpenShift Pipelines provides in its installation namespace, which is normally the **openshift-pipelines** namespace.

3.4.1. Configuring the cluster resolver

You can change the default kind and namespace for the cluster resolver, or limit the namespaces that the cluster resolver can use.

Procedure

1. To edit the **TektonConfig** custom resource, enter the following command:

```
$ oc edit TektonConfig config
```

2. In the **TektonConfig** custom resource, edit the **pipeline.cluster-resolver-config** spec:

```
apiVersion: operator.tekton.dev/v1alpha1
```

```

kind: TektonConfig
metadata:
  name: config
spec:
  pipeline:
    cluster-resolver-config:
      default-kind: pipeline 1
      default-namespace: namespace1 2
      allowed-namespaces: namespace1, namespace2 3
      blocked-namespaces: namespace3, namespace4 4

```

- 1** The default resource kind to fetch, if not specified in parameters.
- 2** The default namespace for fetching resources, if not specified in parameters.
- 3** A comma-separated list of namespaces that the resolver is allowed to access. If this key is not defined, all namespaces are allowed.
- 4** An optional comma-separated list of namespaces which the resolver is blocked from accessing. If this key is not defined, all namespaces are allowed.

3.4.2. Specifying a remote pipeline or task using the cluster resolver

When creating a pipeline run, you can specify a remote pipeline from the same cluster. When creating a pipeline or a task run, you can specify a remote task from the same cluster.

Procedure

- To specify a remote pipeline or task from the same cluster, use the following reference format in the **pipelineRef** or **taskRef** spec:

```

# ...
resolver: cluster
params:
  - name: name
    value: <name>
  - name: namespace
    value: <namespace>
  - name: kind
    value: [pipeline|task]
# ...

```

Table 3.4. Supported parameters for the cluster resolver

Parameter	Description	Example value
name	The name of the resource to fetch.	some-pipeline
namespace	The namespace in the cluster containing the resource.	other-namespace

Parameter	Description	Example value
kind	The kind of the resource to fetch.	pipeline

If the pipeline or task requires additional parameters, provide these parameters in **params**.

The following example pipeline run references a remote pipeline from the same cluster:

```

apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: cluster-pipeline-reference-demo
spec:
  pipelineRef:
    resolver: cluster
  params:
    - name: name
      value: some-pipeline
    - name: namespace
      value: test-namespace
    - name: kind
      value: pipeline
  params:
    - name: sample-pipeline-parameter
      value: test

```

The following example pipeline references a remote task from the same cluster:

```

apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: pipeline-with-cluster-task-reference-demo
spec:
  tasks:
    - name: "cluster-task-reference-demo"
      taskRef:
        resolver: cluster
        params:
          - name: name
            value: some-task
          - name: namespace
            value: test-namespace
          - name: kind
            value: task
  params:
    - name: sample-task-parameter
      value: test

```

The following example task run references a remote task from the same cluster:

```

apiVersion: tekton.dev/v1
kind: TaskRun

```

```

metadata:
  name: cluster-task-reference-demo
spec:
  taskRef:
    resolver: cluster
    params:
      - name: name
        value: some-task
      - name: namespace
        value: test-namespace
      - name: kind
        value: task
  params:
    - name: sample-task-parameter
      value: test

```

3.5. TASKS PROVIDED IN THE OPENSIFT PIPELINES NAMESPACE

An OpenShift Pipelines installation includes a set of standard tasks that you can use in your pipelines. These tasks are located in the OpenShift Pipelines installation namespace, which is normally the **openshift-pipelines** namespace. You can use the cluster resolver to access the tasks.

ClusterTask functionality is deprecated since OpenShift Pipelines 1.10 and is planned for removal in a future release. If your pipelines use **ClusterTasks**, you can re-create them with the tasks that are available from the OpenShift Pipelines installation namespace by using the cluster resolver. However, certain changes are made in these tasks compared to the existing **ClusterTasks**.

You cannot specify a custom execution image in any of the tasks available in the OpenShift Pipelines installation namespace. These tasks do not support parameters such as **BUILDER_IMAGE**, **gitnitImage**, or **KN_IMAGE**. If you want to use a custom execution image, create a copy of the task and replace the image by editing the copy.

buildah

The **buildah** task builds a source code tree into a container image and then pushes the image to a container registry.

Example usage of the buildah task

```

apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
  # ...
  tasks:
  # ...
  - name: build-image
    taskRef:
      resolver: cluster
      params:
        - name: kind
          value: task
        - name: name
          value: buildah
        - name: namespace

```

```

value: openshift-pipelines
params:
- name: IMAGE
  value: $(params.IMAGE)
workspaces:
- name: source
  workspace: shared-workspace
# ...

```

Table 3.5. Supported parameters for the `buildah` task

Parameter	Description	Type	Default value
IMAGE	Fully qualified container image name to be built by Buildah.	string	
DOCKERFILE	Path to the Dockerfile (or Containerfile) relative to the source workspace.	string	./Dockerfile
CONTEXT	Path to the directory to use as the context.	string	.
STORAGE_DRIVER	Set the Buildah storage driver to reflect the settings of the current cluster node settings.	string	vfs
FORMAT	The format of the container to build, either oci or docker .	string	oci
BUILD_EXTRA_ARGS	Extra parameters for the build command when building the image.	string	
PUSH_EXTRA_ARGS	Extra parameters for the push command when pushing the image.	string	
SKIP_PUSH	Skip pushing the image to the container registry.	string	false
TLS_VERIFY	The TLS verification flag, normally true .	string	true
VERBOSE	Turn on verbose logging; all commands executed are added to the log.	string	false

Table 3.6. Supported workspaces for the `buildah` task

Workspace	Description
source	Container build context, usually the application source code that includes a Dockerfile or Containerfile file.

Workspace	Description
dockerconfig	An optional workspace for providing a .docker/config.json file that Buildah uses to access the container registry. Place the file at the root of the workspace with the name config.json or .dockerconfigjson .
rhel-entitlement	An optional workspace for providing the entitlement keys that Buildah uses to access a Red Hat Enterprise Linux (RHEL) subscription. The mounted workspace must contain the entitlement.pem and entitlement-key.pem files.

Table 3.7. Results that the **buildah** task returns

Result	Type	Description
IMAGE_URL	string	The fully qualified name of the image that was built.
IMAGE_DIGEST	string	Digest of the image that was built.

Changes from the **buildah ClusterTask**

- The **VERBOSE** parameter was added.
- The **BUILDER_IMAGE** parameter was removed.

git-cli

The **git-cli** task runs the **git** command-line utility. You can pass the full Git command or several commands to run using the **GIT_SCRIPT** parameter. If the commands need authentication to a Git repository, for example, in order to complete a push, you must supply the authentication credentials.

Example usage of the **git-cli** task

```

apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: update-repo
spec:
  # ...
  tasks:
  # ...
  - name: push-to-repo
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: git-cli
      - name: namespace
        value: openshift-pipelines
    params:
    - name: GIT_SCRIPT

```



```

value: "git push"
- name: GIT_USER_NAME
  value: "Example Developer"
- name: GIT_USER_EMAIL
  value: "developer@example.com"
workspaces:
- name: ssh-directory
  workspace: ssh-workspace 1
- name: source
  workspace: shared-workspace
# ...

```

- 1** In this example, **ssh-workspace** must contain the contents of the **.ssh** directory with a valid key for authorization to the Git repository.

Table 3.8. Supported parameters for thegit-cli task

Parameter	Description	Type	Default value
CRT_FILENAME	Certificate Authority (CA) bundle filename in the ssl-ca-directory workspace.	string	ca-bundle.crt
HTTP_PROXY	HTTP proxy server (non-TLS requests).	string	
HTTPS_PROXY	HTTPS proxy server (TLS requests).	string	
NO_PROXY	Opt out of proxying HTTP/HTTPS requests.	string	
SUBDIRECTOR Y	Relative path to the source workspace where the git repository is present.	string	
USER_HOME	Absolute path to the Git user home directory in the pod.	string	/home/git
DELETE_EXISTING	Erase any existing contents of the source workspace before completing the git operations.	string	true
VERBOSE	Log all the executed commands.	string	false
SSL_VERIFY	The global http.sslVerify value. Do not use false unless you trust the remote repository.	string	true
GIT_USER_NAME	Git user name for performing Git operations.	string	

Parameter	Description	Type	Default value
GIT_USER_EMAIL	Git user email for performing Git operations.	string	
GIT_SCRIPT	The Git script to run.	string	git help

Table 3.9. Supported workspaces for the `git-cli` task

Workspace	Description
ssh-directory	A <code>.ssh</code> directory with the private key, <code>known_hosts</code> , <code>config</code> , and other files as necessary. If you provide this workspace, the task uses it for authentication to the Git repository. Bind this workspace to a Secret resource for secure storage of authentication information.
basic-auth	A workspace containing a <code>.gitconfig</code> and <code>.git-credentials</code> files. If you provide this workspace, the task uses it for authentication to the Git repository. Use a ssh-directory workspace for authentication instead of basic-auth whenever possible. Bind this workspace to a Secret resource for secure storage of authentication information.
ssl-ca-directory	A workspace containing CA certificates. If you provide this workspace, Git uses these certificates to verify the peer when interacting with remote repositories using HTTPS.
source	A workspace that contains the fetched Git repository.
input	An optional workspace that contains the files that need to be added to the Git repository. You can access the workspace from your script using <code>\$(workspaces.input.path)</code> , for example: <pre>cp \$(workspaces.input.path)/<file_that_i_want> . git add <file_that_i_want></pre>

Table 3.10. Results that the `git-cli` task returns

Result	Type	Description
COMMIT	string	The SHA digest of the commit that is at the HEAD of the current branch in the cloned Git repository.

Changes from the `git-cli` ClusterTask

- Several new parameters were added.
- The **BASE_IMAGE** parameter was removed.
- The **ssl-ca-directory** workspace was added.

- The default values for the **USER_HOME** and **VERBOSE** parameters were changed.
- The name of the result was changed from **commit** to **COMMIT**.

git-clone

The **git-clone** task uses Git to initialize and clone a remote repository on a workspace. You can use this task at the start of a pipeline that builds or otherwise processes this source code.

Example usage of the git-clone task

```

apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-source
spec:
  # ...
  tasks:
  - name: clone-repo
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: git-clone
      - name: namespace
        value: openshift-pipelines
    params:
    - name: URL
      value: "https://github.com/example/repo.git"
  workspaces:
  - name: output
    workspace: shared-workspace

```

Table 3.11. Supported parameters for the **git-clone** task

Parameter	Description	Type	Default value
CRT_FILENAME	Certificate Authority (CA) bundle filename in the ssl-ca-directory workspace.	string	ca-bundle.crt
HTTP_PROXY	HTTP proxy server (non-TLS requests).	string	
HTTPS_PROXY	HTTPS proxy server (TLS requests).	string	
NO_PROXY	Opt out of proxying HTTP/HTTPS requests.	string	
SUBDIRECTOR Y	Relative path in the output workspace where the task places the Git repository.	string	

Parameter	Description	Type	Default value
USER_HOME	Absolute path to the Git user home directory in the pod.	string	/home/git
DELETE_EXISTING	Delete the contents of the default workspace, if they exist, before running the Git operations.	string	true
VERBOSE	Log the executed commands.	string	false
SSL_VERIFY	The global http.sslVerify value. Do not set this parameter to false unless you trust the remote repository.	string	true
URL	Git repository URL.	string	
REVISION	The revision to check out, for example, a branch or tag.	string	main
REFSPEC	The refspec string for the repository that the task fetches before checking out the revision.	string	
SUBMODULES	Initialize and fetch Git submodules.	string	true
DEPTH	Number of commits to fetch, a "shallow clone" is a single commit.	string	1
SPARSE_CHECKOUT_DIRECTORIES	List of directory patterns, separated by commas, for performing a "sparse checkout".	string	

Table 3.12. Supported workspaces for thegit-clone task

Workspace	Description
ssh-directory	A .ssh directory with the private key, known_hosts , config , and other files as necessary. If you provide this workspace, the task uses it for authentication to the Git repository. Bind this workspace to a Secret resource for secure storage of authentication information.
basic-auth	A workspace containing a .gitconfig and .git-credentials files. If you provide this workspace, the task uses it for authentication to the Git repository. Use a ssh-directory workspace for authentication instead of basic-auth whenever possible. Bind this workspace to a Secret resource for secure storage of authentication information.

Workspace	Description
ssl-ca-directory	A workspace containing CA certificates. If you provide this workspace, Git uses these certificates to verify the peer when interacting with remote repositories using HTTPS.
output	A workspace that contains the fetched git repository, data will be placed on the root of the workspace or on the relative path defined by the SUBDIRECTORY parameter.

Table 3.13. Results that the **git-clone** task returns

Result	Type	Description
COMMIT	string	The SHA digest of the commit that is at the HEAD of the current branch in the cloned Git repository.
URL	string	The URL of the repository that was cloned.
COMMITTER_DATE	string	The epoch timestamp of the commit that is at the HEAD of the current branch in the cloned Git repository.

Changes from the **git-clone ClusterTask**

- All parameter names were changed to uppercase.
- All result names were changed to uppercase.
- The **gitInitImage** parameter was removed.

kn

The **kn** task uses the **kn** command-line utility to complete operations on Knative resources, such as services, revisions, or routes.

Example usage of the **kn** task

```

apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: kn-run
spec:
  pipelineSpec:
    tasks:
    - name: kn-run
      taskRef:
        resolver: cluster
        params:
        - name: kind
          value: task
        - name: name

```

```

value: kn
- name: namespace
  value: openshift-pipelines
params:
- name: ARGS
  value: [version]

```

Table 3.14. Supported parameters for thekn task

Parameter	Description	Type	Default value
ARGS	The arguments for the kn utility.	array	- help

Changes from thekn ClusterTask

- The **KN_IMAGE** parameter was removed.

kn-apply

The **kn-apply** task deploys a specified image to a Knative Service. This task uses the **kn service apply** command to create or update the specified Knative service.

Example usage of the kn-apply task

```

apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: kn-apply-run
spec:
  pipelineSpec:
    tasks:
    - name: kn-apply-run
      taskRef:
        resolver: cluster
        params:
        - name: kind
          value: task
        - name: name
          value: kn-apply
        - name: namespace
          value: openshift-pipelines
      params:
      - name: SERVICE
        value: "hello"
      - name: IMAGE
        value: "gcr.io/knative-samples/helloworld-go:latest"

```

Table 3.15. Supported parameters for thekn-apply task

Parameter	Description	Type	Default value
SERVICE	The Knative service name.	string	

Parameter	Description	Type	Default value
IMAGE	The fully qualified name of the image to deploy.	string	

Changes from the `kn-apply ClusterTask`

- The **KN_IMAGE** parameter was removed.

maven

The **maven** task runs a Maven build.

Example usage of the maven task

```

apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
  # ...
  tasks:
  # ...
  - name: build-from-source
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: maven
      - name: namespace
        value: openshift-pipelines
    workspaces:
      - name: source
        workspace: shared-workspace
  # ...

```

Table 3.16. Supported parameters for the `maven` task

Parameter	Description	Type	Default value
GOALS	The Maven goals to run.	array	- package
MAVEN_MIRROR_URL	The Maven repository mirror URL.	string	
SUBDIRECTORY	The subdirectory within the source workspace that the task runs the Maven build on.	string	.

Table 3.17. Supported workspaces for themaven task

Workspace	Description
source	The workspace that contains the Maven project.
server_secret	The workspace that contains the secrets for connecting to the Maven server, such as the user name and password.
proxy_secret	The workspace that contains the credentials for connecting to the proxy server, such as the user name and password.
proxy_configmap	The workspace that contains proxy configuration values, such as proxy_port , proxy_host , proxy_protocol , proxy_non_proxy_hosts .
maven_settings	The workspace that contains custom Maven settings.

Changes from the maven ClusterTask

- The parameter name **CONTEXT_DIR** was changed to **SUBDIRECTORY**.
- The workspace name **maven-settings** was changed to **maven_settings**.

openshift-client

The **openshift-client** task runs commands using the **oc** command-line interface. You can use this task to manage an OpenShift Container Platform cluster.

Example usage of the openshift-client task

```

apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: openshift-client-run
spec:
  pipelineSpec:
    tasks:
    - name: openshift-client-run
      taskRef:
        resolver: cluster
        params:
        - name: kind
          value: task
        - name: name
          value: openshift-client
        - name: namespace
          value: openshift-pipelines
      params:
        - name: SCRIPT
          value: "oc version"

```

Table 3.18. Supported parameters for the openshift-client task

Parameter	Description	Type	Default value
SCRIPT	The oc CLI arguments to run.	string	oc help
VERSION	The OpenShift Container Platform version to use.	string	latest

Table 3.19. Supported workspaces for the `openshift-client` task

Workspace	Description
manifest_dir	The workspace containing manifest files that you want to apply using the oc utility.
kubeconfig_dir	An optional workspace in which you can provide a .kube/config file that contains credentials for accessing the cluster. Place this file at the root of the workspace and name it kubeconfig .

Changes from the `openshift-client` ClusterTask

- The workspace name **manifest-dir** was changed to **manifest_dir**.
- The workspace name **kubeconfig-dir** was changed to **kubeconfig_dir**.

s2i-dotnet

The **s2i-dotnet** task builds the source code using the Source to Image (S2I) dotnet builder image, which is available from the OpenShift Container Platform registry as **image-registry.openshift-image-registry.svc:5000/openshift/dotnet**.

Example usage of the `s2i-dotnet` task

```

apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
  # ...
  tasks:
  # ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
        - name: kind
          value: task
        - name: name
          value: s2i-dotnet
        - name: namespace
          value: openshift-pipelines
    workspaces:

```

```
- name: source
  workspace: shared-workspace
# ...
```

Table 3.20. Supported parameters for `thes2i-dotnet` task

Parameter	Description	Type	Default value
IMAGE	The fully qualified name for the container image that the S2I process builds.	string	
IMAGE_SCRIPTS_URL	The URL containing the default assemble and run scripts for the builder image.	string	image:///usr/libexec/s2i
ENV_VARS	An array of values for environment variables to set in the build process, listed in the KEY=VALUE format.	array	
CONTEXT	Path to the directory within the source workspace to use as the context.	string	.
STORAGE_DRIVER	Set the Buildah storage driver to reflect the settings of the current cluster node settings.	string	vfs
FORMAT	The format of the container to build, either oci or docker .	string	oci
BUILD_EXTRA_ARGS	Extra parameters for the build command when building the image.	string	
PUSH_EXTRA_ARGS	Extra parameters for the push command when pushing the image.	string	
SKIP_PUSH	Skip pushing the image to the container registry.	string	false
TLS_VERIFY	The TLS verification flag, normally true .	string	true
VERBOSE	Turn on verbose logging; all commands executed are added to the log.	string	false
VERSION	The tag of the image stream, which corresponds to the language version.	string	latest

Table 3.21. Supported workspaces for `thes2i-dotnet` task

Workspace	Description
source	The application source code, which is the build context for the S2I workflow.
dockerconfig	An optional workspace for providing a .docker/config.json file that Buildah uses to access the container registry. Place the file at the root of the workspace with the name config.json or .dockerconfigjson .

Table 3.22. Results that the **s2i-dotnet** task returns

Result	Type	Description
IMAGE_URL	string	The fully qualified name of the image that was built.
IMAGE_DIGEST	string	Digest of the image that was built.

Changes from the **s2i-dotnet ClusterTask**

- Several new parameters were added.
- The **BASE_IMAGE** parameter was removed.
- The parameter name **PATH_CONTEXT** was changed to **CONTEXT**.
- The parameter name **TLS_VERIFY** was changed to **TLSVERIFY**.
- The **IMAGE_URL** result was added.

s2i-go

The **s2i-go** task builds the source code using the S2I Golang builder image, which is available from the OpenShift Container Platform registry as **image-registry.openshift-image-registry.svc:5000/openshift/golang**.

Example usage of the **s2i-go** task

```

apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
  # ...
  tasks:
  # ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: s2i-go
      - name: namespace

```

```

value: openshift-pipelines
workspaces:
- name: source
  workspace: shared-workspace
# ...

```

Table 3.23. Supported parameters for **thes2i-go** task

Parameter	Description	Type	Default value
IMAGE	The fully qualified name for the container image that the S2I process builds.	string	
IMAGE_SCRIPTS_URL	The URL containing the default assemble and run scripts for the builder image.	string	image:///usr/libexec/s2i
ENV_VARS	An array of values for environment variables to set in the build process, listed in the KEY=VALUE format.	array	
CONTEXT	Path to the directory within the source workspace to use as the context.	string	.
STORAGE_DRIVER	Set the Buildah storage driver to reflect the settings of the current cluster node settings.	string	vfs
FORMAT	The format of the container to build, either oci or docker .	string	oci
BUILD_EXTRA_ARGS	Extra parameters for the build command when building the image.	string	
PUSH_EXTRA_ARGS	Extra parameters for the push command when pushing the image.	string	
SKIP_PUSH	Skip pushing the image to the container registry.	string	false
TLS_VERIFY	The TLS verification flag, normally true .	string	true
VERBOSE	Turn on verbose logging; all commands executed are added to the log.	string	false
VERSION	The tag of the image stream, which corresponds to the language version.	string	latest

Table 3.24. Supported workspaces for **thes2i-go** task

Workspace	Description
source	The application source code, which is the build context for the S2I workflow.
dockerconfig	An optional workspace for providing a .docker/config.json file that Buildah uses to access the container registry. Place the file at the root of the workspace with the name config.json or .dockerconfigjson .

Table 3.25. Results that the **s2i-go** task returns

Result	Type	Description
IMAGE_URL	string	The fully qualified name of the image that was built.
IMAGE_DIGEST	string	Digest of the image that was built.

Changes from the **s2i-go ClusterTask**

- Several new parameters were added.
- The **BASE_IMAGE** parameter was removed.
- The parameter name **PATH_CONTEXT** was changed to **CONTEXT**.
- The parameter name **TLS_VERIFY** was changed to **TLSVERIFY**.
- The **IMAGE_URL** result was added.

s2i-java

The **s2i-java** task builds the source code using the S2I Java builder image, which is available from the OpenShift Container Platform registry as **image-registry.openshift-image-registry.svc:5000/openshift/java**.

Table 3.26. Supported parameters for the **s2i-java** task

Parameter	Description	Type	Default value
IMAGE	The fully qualified name for the container image that the S2I process builds.	string	
IMAGE_SCRIPTS_URL	The URL containing the default assemble and run scripts for the builder image.	string	image:///usr/libexec/s2i
ENV_VARS	An array of values for environment variables to set in the build process, listed in the KEY=VALUE format.	array	
CONTEXT	Path to the directory within the source workspace to use as the context.	string	.

Parameter	Description	Type	Default value
STORAGE_DRIVER	Set the Buildah storage driver to reflect the settings of the current cluster node settings.	string	vfs
FORMAT	The format of the container to build, either oci or docker .	string	oci
BUILD_EXTRA_ARGS	Extra parameters for the build command when building the image.	string	
PUSH_EXTRA_ARGS	Extra parameters for the push command when pushing the image.	string	
SKIP_PUSH	Skip pushing the image to the container registry.	string	false
TLS_VERIFY	The TLS verification flag, normally true .	string	true
VERBOSE	Turn on verbose logging; all commands executed are added to the log.	string	false
VERSION	The tag of the image stream, which corresponds to the language version.	string	latest

Table 3.27. Supported workspaces for the **s2i-java** task

Workspace	Description
source	The application source code, which is the build context for the S2I workflow.
dockerconfig	An optional workspace for providing a .docker/config.json file that Buildah uses to access the container registry. Place the file at the root of the workspace with the name config.json or .dockerconfigjson .

Table 3.28. Results that the **s2i-java** task returns

Result	Type	Description
IMAGE_URL	string	The fully qualified name of the image that was built.
IMAGE_DIGEST	string	Digest of the image that was built.

Changes from the **s2i-java** ClusterTask

- Several new parameters were added.

- The **BUILDER_IMAGE**, **MAVEN_ARGS_APPEND**, **MAVEN_CLEAR_REPO**, and **MAVEN_MIRROR_URL** parameters were removed. You can pass the **MAVEN_ARGS_APPEND**, **MAVEN_CLEAR_REPO**, and **MAVEN_MIRROR_URL** values as environment variables.
- The parameter name **PATH_CONTEXT** was changed to **CONTEXT**.
- The parameter name **TLS_VERIFY** was changed to **TLSVERIFY**.
- The **IMAGE_URL** result was added.

s2i-nodejs

The **s2i-nodejs** task builds the source code using the S2I NodeJS builder image, which is available from the OpenShift Container Platform registry as **image-registry.openshift-image-registry.svc:5000/openshift/nodejs**.

Example usage of the s2i-nodejs task

```

apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
# ...
  tasks:
# ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
        - name: kind
          value: task
        - name: name
          value: s2i-nodejs
        - name: namespace
          value: openshift-pipelines
    workspaces:
      - name: source
        workspace: shared-workspace
# ...

```

Table 3.29. Supported parameters for the **s2i-nodejs** task

Parameter	Description	Type	Default value
IMAGE	The fully qualified name for the container image that the S2I process builds.	string	
IMAGE_SCRIPTS_URL	The URL containing the default assemble and run scripts for the builder image.	string	image:///usr/libexec/s2i
ENV_VARS	An array of values for environment variables to set in the build process, listed in the KEY=VALUE format.	array	

Parameter	Description	Type	Default value
CONTEXT	Path to the directory within the source workspace to use as the context.	string	.
STORAGE_DRIVER	Set the Buildah storage driver to reflect the settings of the current cluster node settings.	string	vfs
FORMAT	The format of the container to build, either oci or docker .	string	oci
BUILD_EXTRA_ARGS	Extra parameters for the build command when building the image.	string	
PUSH_EXTRA_ARGS	Extra parameters for the push command when pushing the image.	string	
SKIP_PUSH	Skip pushing the image to the container registry.	string	false
TLS_VERIFY	The TLS verification flag, normally true .	string	true
VERBOSE	Turn on verbose logging; all commands executed are added to the log.	string	false
VERSION	The tag of the image stream, which corresponds to the language version.	string	latest

Table 3.30. Supported workspaces for thes2i-nodejs task

Workspace	Description
source	The application source code, which is the build context for the S2I workflow.
dockerconfig	An optional workspace for providing a .docker/config.json file that Buildah uses to access the container registry. Place the file at the root of the workspace with the name config.json or .dockerconfigjson .

Table 3.31. Results that the s2i-nodejs task returns

Result	Type	Description
IMAGE_URL	string	The fully qualified name of the image that was built.
IMAGE_DIGEST	string	Digest of the image that was built.

Changes from the s2i-nodejs ClusterTask

- Several new parameters were added.
- The **BASE_IMAGE** parameter was removed.
- The parameter name **PATH_CONTEXT** was changed to **CONTEXT**.
- The parameter name **TLS_VERIFY** was changed to **TLSVERIFY**.
- The **IMAGE_URL** result was added.

s2i-perl

The **s2i-perl** task builds the source code using the S2I Perl builder image, which is available from the OpenShift Container Platform registry as **image-registry.openshift-image-registry.svc:5000/openshift/perl**.

Example usage of the s2i-perl task

```

apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
  # ...
  tasks:
  # ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: s2i-perl
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: source
      workspace: shared-workspace
  # ...

```

Table 3.32. Supported parameters for the **s2i-perl** task

Parameter	Description	Type	Default value
IMAGE	The fully qualified name for the container image that the S2I process builds.	string	
IMAGE_SCRIPTS_URL	The URL containing the default assemble and run scripts for the builder image.	string	image:///usr/libexec/s2i
ENV_VARS	An array of values for environment variables to set in the build process, listed in the KEY=VALUE format.	array	

Parameter	Description	Type	Default value
CONTEXT	Path to the directory within the source workspace to use as the context.	string	.
STORAGE_DRIVER	Set the Buildah storage driver to reflect the settings of the current cluster node settings.	string	vfs
FORMAT	The format of the container to build, either oci or docker .	string	oci
BUILD_EXTRA_ARGS	Extra parameters for the build command when building the image.	string	
PUSH_EXTRA_ARGS	Extra parameters for the push command when pushing the image.	string	
SKIP_PUSH	Skip pushing the image to the container registry.	string	false
TLS_VERIFY	The TLS verification flag, normally true .	string	true
VERBOSE	Turn on verbose logging; all commands executed are added to the log.	string	false
VERSION	The tag of the image stream, which corresponds to the language version.	string	latest

Table 3.33. Supported workspaces for **thes2i-perl** task

Workspace	Description
source	The application source code, which is the build context for the S2I workflow.
dockerconfig	An optional workspace for providing a .docker/config.json file that Buildah uses to access the container registry. Place the file at the root of the workspace with the name config.json or .dockerconfigjson .

Table 3.34. Results that the **s2i-perl** task returns

Result	Type	Description
IMAGE_URL	string	The fully qualified name of the image that was built.
IMAGE_DIGEST	string	Digest of the image that was built.

Changes from the **s2i-perl** ClusterTask

- Several new parameters were added.
- The **BASE_IMAGE** parameter was removed.
- The parameter name **PATH_CONTEXT** was changed to **CONTEXT**.
- The parameter name **TLS_VERIFY** was changed to **TLSVERIFY**.
- The **IMAGE_URL** result was added.

s2i-php

The **s2i-php** task builds the source code using the S2I PHP builder image, which is available from the OpenShift Container Platform registry as **image-registry.openshift-image-registry.svc:5000/openshift/php**.

Example usage of the s2i-php task

```

apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
  # ...
  tasks:
  # ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: s2i-php
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: source
      workspace: shared-workspace
  # ...

```

Table 3.35. Supported parameters for the **s2i-php** task

Parameter	Description	Type	Default value
IMAGE	The fully qualified name for the container image that the S2I process builds.	string	
IMAGE_SCRIPTS_URL	The URL containing the default assemble and run scripts for the builder image.	string	image:///usr/libexec/s2i
ENV_VARS	An array of values for environment variables to set in the build process, listed in the KEY=VALUE format.	array	

Parameter	Description	Type	Default value
CONTEXT	Path to the directory within the source workspace to use as the context.	string	.
STORAGE_DRIVER	Set the Buildah storage driver to reflect the settings of the current cluster node settings.	string	vfs
FORMAT	The format of the container to build, either oci or docker .	string	oci
BUILD_EXTRA_ARGS	Extra parameters for the build command when building the image.	string	
PUSH_EXTRA_ARGS	Extra parameters for the push command when pushing the image.	string	
SKIP_PUSH	Skip pushing the image to the container registry.	string	false
TLS_VERIFY	The TLS verification flag, normally true .	string	true
VERBOSE	Turn on verbose logging; all commands executed are added to the log.	string	false
VERSION	The tag of the image stream, which corresponds to the language version.	string	latest

Table 3.36. Supported workspaces for thes2i-php task

Workspace	Description
source	The application source code, which is the build context for the S2I workflow.
dockerconfig	An optional workspace for providing a .docker/config.json file that Buildah uses to access the container registry. Place the file at the root of the workspace with the name config.json or .dockerconfigjson .

Table 3.37. Results that the s2i-php task returns

Result	Type	Description
IMAGE_URL	string	The fully qualified name of the image that was built.
IMAGE_DIGEST	string	Digest of the image that was built.

Changes from the s2i-php ClusterTask

- Several new parameters were added.
- The **BASE_IMAGE** parameter was removed.
- The parameter name **PATH_CONTEXT** was changed to **CONTEXT**.
- The parameter name **TLS_VERIFY** was changed to **TLSVERIFY**.
- The **IMAGE_URL** result was added.

s2i-python

The **s2i-python** task builds the source code using the S2I Python builder image, which is available from the OpenShift Container Platform registry as **image-registry.openshift-image-registry.svc:5000/openshift/python**.

Example usage of the s2i-python task

```

apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
  # ...
  tasks:
  # ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: s2i-python
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: source
      workspace: shared-workspace
  # ...

```

Table 3.38. Supported parameters for the **s2i-python** task

Parameter	Description	Type	Default value
IMAGE	The fully qualified name for the container image that the S2I process builds.	string	
IMAGE_SCRIPTS_URL	The URL containing the default assemble and run scripts for the builder image.	string	image:///usr/libexec/s2i
ENV_VARS	An array of values for environment variables to set in the build process, listed in the KEY=VALUE format.	array	

Parameter	Description	Type	Default value
CONTEXT	Path to the directory within the source workspace to use as the context.	string	.
STORAGE_DRIVER	Set the Buildah storage driver to reflect the settings of the current cluster node settings.	string	vfs
FORMAT	The format of the container to build, either oci or docker .	string	oci
BUILD_EXTRA_ARGS	Extra parameters for the build command when building the image.	string	
PUSH_EXTRA_ARGS	Extra parameters for the push command when pushing the image.	string	
SKIP_PUSH	Skip pushing the image to the container registry.	string	false
TLS_VERIFY	The TLS verification flag, normally true .	string	true
VERBOSE	Turn on verbose logging; all commands executed are added to the log.	string	false
VERSION	The tag of the image stream, which corresponds to the language version.	string	latest

Table 3.39. Supported workspaces for thes2i-python task

Workspace	Description
source	The application source code, which is the build context for the S2I workflow.
dockerconfig	An optional workspace for providing a .docker/config.json file that Buildah uses to access the container registry. Place the file at the root of the workspace with the name config.json or .dockerconfigjson .

Table 3.40. Results that thes2i-python task returns

Result	Type	Description
IMAGE_URL	string	The fully qualified name of the image that was built.
IMAGE_DIGEST	string	Digest of the image that was built.

Changes from thes2i-python ClusterTask

- Several new parameters were added.
- The **BASE_IMAGE** parameter was removed.
- The parameter name **PATH_CONTEXT** was changed to **CONTEXT**.
- The parameter name **TLS_VERIFY** was changed to **TLSVERIFY**.
- The **IMAGE_URL** result was added.

s2i-ruby

The **s2i-ruby** task builds the source code using the S2I Ruby builder image, which is available from the OpenShift Container Platform registry as **image-registry.openshift-image-registry.svc:5000/openshift/ruby**.

Example usage of the s2i-ruby task

```

apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
  # ...
  tasks:
  # ...
  - name: build-s2i
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: s2i-ruby
      - name: namespace
        value: openshift-pipelines
    workspaces:
    - name: source
      workspace: shared-workspace
  # ...

```

Table 3.41. Supported parameters for the s2i-ruby task

Parameter	Description	Type	Default value
IMAGE	The fully qualified name for the container image that the S2I process builds.	string	
IMAGE_SCRIPTS_URL	The URL containing the default assemble and run scripts for the builder image.	string	image:///usr/libexec/s2i
ENV_VARS	An array of values for environment variables to set in the build process, listed in the KEY=VALUE format.	array	

Parameter	Description	Type	Default value
CONTEXT	Path to the directory within the source workspace to use as the context.	string	.
STORAGE_DRIVER	Set the Buildah storage driver to reflect the settings of the current cluster node settings.	string	vfs
FORMAT	The format of the container to build, either oci or docker .	string	oci
BUILD_EXTRA_ARGS	Extra parameters for the build command when building the image.	string	
PUSH_EXTRA_ARGS	Extra parameters for the push command when pushing the image.	string	
SKIP_PUSH	Skip pushing the image to the container registry.	string	false
TLS_VERIFY	The TLS verification flag, normally true .	string	true
VERBOSE	Turn on verbose logging; all commands executed are added to the log.	string	false
VERSION	The tag of the image stream, which corresponds to the language version.	string	latest

Table 3.42. Supported workspaces for the **s2i-ruby** task

Workspace	Description
source	The application source code, which is the build context for the S2I workflow.
dockerconfig	An optional workspace for providing a .docker/config.json file that Buildah uses to access the container registry. Place the file at the root of the workspace with the name config.json or .dockerconfigjson .

Table 3.43. Results that the **s2i-ruby** task returns

Result	Type	Description
IMAGE_URL	string	The fully qualified name of the image that was built.
IMAGE_DIGEST	string	Digest of the image that was built.

Changes from the **s2i-ruby** ClusterTask

- Several new parameters were added.
- The **BASE_IMAGE** parameter was removed.
- The parameter name **PATH_CONTEXT** was changed to **CONTEXT**.
- The parameter name **TLS_VERIFY** was changed to **TLSVERIFY**.
- The **IMAGE_URL** result was added.

skopeo-copy

The **skopeo-copy** task executes the **skopeo copy** command.

Skopeo is a command-line tool for working with remote container image registries, which does not require a daemon or other infrastructure to load and run the images. The **skopeo copy** command copies an image from one remote registry to another, for example, from an internal registry to a production registry. Skopeo supports authorization on image registries using credentials that you provide.

Example usage of the skopeo-copy task

```

apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: build-deploy-image
spec:
  # ...
  tasks:
  - name: copy-image
    taskRef:
      resolver: cluster
      params:
      - name: kind
        value: task
      - name: name
        value: skopeo-copy
      - name: namespace
        value: openshift-pipelines
    params:
    - name: SOURCE_IMAGE_URL
      value: "docker://internal.registry/myimage:latest"
    - name: DESTINATION_IMAGE_URL
      value: "docker://production.registry/myimage:v1.0"
  workspaces:
  - name: output
    workspace: shared-workspace

```

Table 3.44. Supported parameters for the **skopeo-copy** task

Parameter	Description	Type	Default value
SOURCE_IMAGE_URL	Fully qualified name, including tag, of the source container image.	string	

Parameter	Description	Type	Default value
DESTINATION_IMAGE_URL	Fully qualified name, including tag, of the destination image to which Skopeo copies the source image.	string	
SRC_TLS_VERIFY	The TLS verification flag for the source registry, normally true .	string	true
DEST_TLS_VERIFY	The TLS verification flag for the destination registry, normally true	string	true
VERBOSE	Output debug information to the log.	string	false

Table 3.45. Supported workspaces for **theskopeo-copy** task

Workspace	Description
images_url	If you want to copy more than one image, use this workspace to provide the image URLs.

Table 3.46. Results that the **skopeo-copy** task returns

Result	Type	Description
SOURCE_DIGEST	string	The SHA256 digest of the source image.
DESTINATION_DIGEST	string	The SHA256 digest of the destination image.

Changes from the **theskopeo-copy** ClusterTask

- All parameter names were changed to uppercase.
- The **VERBOSE** parameter was added.
- The workspace name was changed from **images-url** to **images_url**.
- The **SOURCE_DIGEST** and **DESTINATION_DIGEST** results were added.

tkn

The **tkn** task performs operations on Tekton resources using **tkn**.

Example usage of the **tkn** task

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: tkn-run
```

```

spec:
  pipelineSpec:
    tasks:
    - name: tkn-run
      taskRef:
        resolver: cluster
        params:
        - name: kind
          value: task
        - name: name
          value: tkn
        - name: namespace
          value: openshift-pipelines
      params:
      - name: ARGS

```

Table 3.47. Supported parameters for the `tkn` task

Parameter	Description	Type	Default value
SCRIPT	The tkn CLI script to execute.	string	tkn \$@
ARGS	The tkn CLI arguments to run.	array	- --help

Table 3.48. Supported workspaces for the `tkn` task

Workspace	Description
kubeconfig_dir	An optional workspace in which you can provide a .kube/config file that contains credentials for accessing the cluster. Place this file at the root of the workspace and name it kubeconfig .

Changes from the `tkn` ClusterTask

- The **TKN_IMAGE** parameter was removed.
- The workspace name was changed from **kubeconfig** to **kubeconfig_dir**.

3.6. ADDITIONAL RESOURCES

- [Using Tekton Hub with OpenShift Pipelines](#)

CHAPTER 4. USING MANUAL APPROVAL IN OPENSIFT PIPELINES

You can specify a manual approval task in a pipeline. When the pipeline reaches this task, it pauses and awaits approval from one or several OpenShift Container Platform users. If any of the users chooses to reject the task instead of approving it, the pipeline fails. The manual approval gate controller provides this functionality.



IMPORTANT

The manual approval gate is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

4.1. ENABLING THE MANUAL APPROVAL GATE CONTROLLER

To use manual approval tasks, you must first enable the manual approval gate controller.

Prerequisites

- You installed the Red Hat OpenShift Pipelines Operator in your cluster.
- You are logged on to the cluster using the **oc** command-line utility.
- You have administrator permissions for the **openshift-pipelines** namespace.

Procedure

1. Create a file named **manual-approval-gate-cr.yaml** with the following manifest for the **ManualApprovalGate** custom resource (CR):

```
apiVersion: operator.tekton.dev/v1alpha1
kind: ManualApprovalGate
metadata:
  name: manual-approval-gate
spec:
  targetNamespace: openshift-pipelines
```

2. Apply the **ManualApprovalGate** CR by entering the following command:

```
$ oc apply -f manual-approval-gate-cr.yaml
```

3. Verify that the manual approval gate controller is running by entering the following command:

```
$ oc get manualapprovalgates.operator.tekton.dev
```

Example output

■

NAME	VERSION	READY	REASON
manual-approval-gate	v0.1.0	True	

Ensure that the **READY** status is **True**. If it is not **True**, wait for a few minutes and enter the command again. The controller might take some time to reach a ready state.

4.2. SPECIFYING A MANUAL APPROVAL TASK

You can specify a manual approval task in your pipeline. When the execution of a pipeline run reaches this task, the pipeline run stops and awaits approval from one or several users.

Prerequisites

- You enabled the manual approver gate controller.
- You created a YAML specification of a pipeline.

Procedure

- Specify an **ApprovalTask** in the pipeline, as shown in the following example:

```

apiVersion: tekton.dev/v1
kind: Pipeline
metadata:
  name: example-manual-approval-pipeline
spec:
  tasks:
# ...
  - name: example-manual-approval-task
    taskRef:
      apiVersion: openshift-pipelines.org/v1alpha1
      kind: ApprovalTask
    params:
      - name: approvers
        value:
          - user1
          - user2
          - user3
      - name: description
        value: Example manual approval task - please approve or reject
      - name: numberOfApprovalsRequired
        value: '2'
      - name: timeout
        value: '60m'
# ...

```

Table 4.1. Parameters for a manual approval task

Parameter	Type	Description
approvers	array	The OpenShift Container Platform users who can approve the task.

Parameter	Type	Description
description	string	Optional: The description of the approval task. OpenShift Pipelines displays the description to the user who can approve or reject the task.
numberOfApprovalsRequired	string	The number of approvals from different users that the task requires.
timeout	string	Optional: The timeout period for approval. If the task does not receive the configured number of approvals during this period, the pipeline run fails. The default timeout is 1 hour.

4.3. APPROVING A MANUAL APPROVAL TASK

When you run a pipeline that includes an approval task and the execution reaches the approval task, the pipeline run pauses and waits for user approval or rejection.

Users can approve or reject the task by using either the web console or the **opc** command line utility.

If any one of the approvers configured in the task rejects the task, the pipeline run fails.

If one user approves the task but the configured number of approvals is still not reached, the same user can change to rejecting the task and the pipeline run fails.

4.3.1. Approving a manual approval task by using the web console

You can approve or reject a manual approval task by using the OpenShift Container Platform web console.


If you are listed as an approver in a manual approval task and a pipeline run reaches this task, the web console displays a notification. You can view a list of tasks that require your approval and approve or reject these tasks.

Prerequisites

- You enabled the OpenShift Pipelines console plugin.

Procedure

1. View a list of tasks that you can approve by completing one of the following actions:
 - When a notification about a task requiring your approval displays, click **Go to Approvals tab** in this notification.

- In the **Administrator** perspective menu, select **Pipelines** → **Pipelines** and then click the **Approvals** tab.
 - In the **Developer** perspective menu, select **Pipelines** and then click the **Approvals** tab.
 - In the **PipelineRun details** window, in the **Details** tab, click the rectangle that represents the manual approval task. The list displays only the approval for this task.
 - In the **PipelineRun details** window, click the **ApprovalTasks** tab. The list displays only the approval for this pipeline run.
2. In the list of approval tasks, in the line that represents the task that you want to approve, click the  icon and then select one of the following options:
 - To approve the task, select **Approve**.
 - To reject the task, select **Reject**.
 3. Enter a message in the **Reason** field.
 4. Click **Submit**.

Additional resources

- [Enabling the OpenShift Pipelines console plugin](#)

4.3.2. Approving a manual approval task by using the command line

You can approve or reject a manual approval task by using the **opc** command-line utility. You can view a list of tasks for which you are an approver and approve or reject the tasks that are pending approval.

Prerequisites

- You downloaded and installed the **opc** command-line utility. This utility is available in the same package as the **tkn** command-line utility.
- You are logged on to the cluster using the **oc** command-line utility.

Procedure

1. View a list of manual approval tasks for which you are listed as an approver by entering the following command:

```
$ opc approvaltask list
```

Example output

```

NAME                                     NumberOfApprovalsRequired PendingApprovals Rejected
STATUS
manual-approval-pipeline-01w6e1-task-2 2                0                0        Approved
manual-approval-pipeline-6yvv82-task-2 2                2                0        Rejected
manual-approval-pipeline-90gyki-task-2 2                2                0        Pending
manual-approval-pipeline-jyrkb3-task-2 1                1                1        Rejected

```

- Optional: To view information about a manual approval task, including its name, namespace, pipeline run name, list of approvers, and current status, enter the following command:

```
$ opc approvtask describe <approval_task_name>
```

- Approve or reject a manual approval task as necessary:

- To approve a manual approval task, enter the following command:

```
$ opc approvtask approve <approval_task_name>
```

Optionally, you can specify a message for the approval by using the **-m** parameter:

```
$ opc approvtask approve <approval_task_name> -m <message>
```

- To reject a manual approval task, enter the following command:

```
$ opc approvtask reject <approval_task_name>
```

Optionally, you can specify a message for the rejection by using the **-m** parameter:

```
$ opc approvtask reject <approval_task_name> -m <message>
```

Additional resources

- [Installing tkn](#)

CHAPTER 5. USING RED HAT ENTITLEMENTS IN PIPELINES

If you have Red Hat Enterprise Linux (RHEL) entitlements, you can use these entitlements to build container images in your pipelines.

The Insights Operator automatically manages your entitlements after you import them into this operator from Simple Common Access (SCA). This operator provides a secret named **etc-pki-entitlement** in the **openshift-config-managed** namespace.

You can use Red Hat entitlements in your pipelines in one of the following two ways:

- Manually copy the secret into the namespace of the pipeline. This method is least complex if you have a limited number of pipeline namespaces.
- Use the Shared Resources Container Storage Interface (CSI) Driver Operator to share the secret between namespaces automatically.

5.1. PREREQUISITES

- You logged on to your OpenShift Container Platform cluster using the **oc** command line tool.
- You enabled the Insights Operator feature on your OpenShift Container Platform cluster. If you want to use the Shared Resources CSI Driver operator to share the secret between namespaces, you must also enable the Shared Resources CSI driver. For information about enabling features, including the Insights Operator and Shared Resources CSI Driver, see [Enabling features using feature gates](#).



NOTE

After you enable the Insights Operator, you must wait for some time to ensure that the cluster updates all the nodes with this operator. You can monitor the status of all nodes by entering the following command:

```
$ oc get nodes -w
```

To verify that the Insights Operator is active, check that the **insights-operator** pod is running in the **openshift-insights** namespace by entering the following command:

```
$ oc get pods -n openshift-insights
```

- You configured the importing of your Red Hat entitlements into the Insights Operator. For information about importing the entitlements, see [Importing simple content access entitlements with Insights Operator](#).



NOTE

To verify that the Insights Operator made your entitlements available, is active, check that the **etc-pki-entitlement** secret is present in the **openshift-config-managed** namespace by entering the following command:

```
$ oc get secret etc-pki-entitlement -n openshift-config-managed
```

5.2. USING RED HAT ENTITLEMENTS BY MANUALLY COPYING THE ETC-PKI-ENTITLEMENT SECRET

You can copy the **etc-pki-entitlement** secret from the **openshift-config-managed** namespace into the namespace of your pipeline. You can then configure your pipeline to use this secret for the Buildah task.

Prerequisites

- You installed the **jq** package on your system. This package is available in Red Hat Enterprise Linux (RHEL).

Procedure

- Copy the **etc-pki-entitlement** secret from the **openshift-config-managed** namespace into the namespace of your pipeline by running the following command:

```
$ oc get secret etc-pki-entitlement -n openshift-config-managed -o json | \
jq 'del(.metadata.resourceVersion)' | jq 'del(.metadata.creationTimestamp)' | \
jq 'del(.metadata.uid)' | jq 'del(.metadata.namespace)' | \
oc -n <pipeline_namespace> create -f - 1
```

- Replace **<pipeline_namespace>** with the namespace of your pipeline.
- In your Buildah task definition, use the **buildah** task provided in the **openshift-pipelines** namespace or a copy of this task and define the **rhel-entitlement** workspace, as shown in the following example.
- In your task run or pipeline run that runs the Buildah task, assign the **etc-pki-entitlement** secret to the **rhel-entitlement** workspace, as in the following example.

Example pipeline run definition, including the pipeline and task definitions, that uses Red Hat entitlements

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: buildah-pr-test
spec:
  workspaces:
    - name: shared-workspace
      volumeClaimTemplate:
        spec:
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
              storage: 1Gi
    - name: dockerconfig
      secret:
        secretName: regred
    - name: rhel-entitlement 1
      secret:
        secretName: etc-pki-entitlement
```

```

pipelineSpec:
  workspaces:
    - name: shared-workspace
    - name: dockerconfig
    - name: rhel-entitlement 2
  tasks:
# ...
    - name: buildah
      taskRef:
        resolver: cluster
        params:
          - name: kind
            value: task
          - name: name
            value: buildah
          - name: namespace
            value: openshift-pipelines
      workspaces:
        - name: source
          workspace: shared-workspace
        - name: dockerconfig
          workspace: dockerconfig
        - name: rhel-entitlement 3
          workspace: rhel-entitlement
      params:
        - name: IMAGE
          value: <image_where_you_want_to_push>

```

- 1 The definition of the **rhel-entitlement** workspace in the pipeline run, assigning the **etc-pki-entitlement** secret to the workspace
- 2 The definition of the **rhel-entitlement** workspace in the pipeline definition
- 3 The definition of the **rhel-entitlement** workspace in the task definition

5.3. USING RED HAT ENTITLEMENTS BY SHARING THE SECRET USING THE SHARED RESOURCES CSI DRIVER OPERATOR

You can set up sharing of the **etc-pki-entitlement** secret from the **openshift-config-managed** namespace to other namespaces using the Shared Resources Container Storage Interface (CSI) Driver Operator. You can then configure your pipeline to use this secret for the Buildah task.

Prerequisites

- You are logged on to your OpenShift Container Platform cluster using the **oc** command line utility as a user with cluster administrator permissions.
- You enabled the Shared Resources CSI Driver operator on your OpenShift Container Platform cluster.

Procedure

1. Create a **SharedSecret** custom resource (CR) for sharing the **etc-pki-entitlement** secret by running the following command:

```
$ oc apply -f - <<EOF
apiVersion: sharedresource.openshift.io/v1alpha1
kind: SharedSecret
metadata:
  name: shared-rhel-entitlement
spec:
  secretRef:
    name: etc-pki-entitlement
    namespace: openshift-config-managed
EOF
```

2. Create an RBAC role that permits access to the shared secret by running the following command:

```
$ oc apply -f - <<EOF
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: shared-resource-rhel-entitlement
  namespace: <pipeline_namespace> 1
rules:
  - apiGroups:
    - sharedresource.openshift.io
    resources:
    - sharedsecrets
    resourceName:
    - shared-rhel-entitlement
    verbs:
    - use
EOF
```

- 1** Replace **<pipeline_namespace>** with the namespace of your pipeline.

3. Assign the role to the **pipeline** service account by running the following command:

```
$ oc create rolebinding shared-resource-rhel-entitlement --role=shared-shared-resource-rhel-entitlement \
--serviceaccount=<pipeline-namespace>:pipeline 1
```

- 1** Replace **<pipeline-namespace>** with the namespace of your pipeline.



NOTE

If you changed the default service account for OpenShift Pipelines or if you define a custom service account in the pipeline run or task run, assign the role to this account instead of the **pipeline** account.

4. In your Buildah task definition, use the **buildah** task provided in the **openshift-pipelines** namespace or a copy of this task and define the **rhel-entitlement** workspace, as shown in the following example.

- In your task run or pipeline run that runs the Buildah task, assign the shared secret to the **rhel-entitlement** workspace, as in the following example.

Example pipeline run definition, including the pipeline and task definitions, that uses Red Hat entitlements

```

apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: buildah-pr-test-csi
spec:
  workspaces:
    - name: shared-workspace
      volumeClaimTemplate:
        spec:
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
              storage: 1Gi
    - name: dockerconfig
      secret:
        secretName: regred
    - name: rhel-entitlement 1
      csi:
        readOnly: true
        driver: csi.sharedresource.openshift.io
        volumeAttributes:
          sharedSecret: shared-rhel-entitlement
  pipelineSpec:
    workspaces:
      - name: shared-workspace
      - name: dockerconfig
      - name: rhel-entitlement 2
    tasks:
      # ...
      - name: buildah
        taskRef:
          resolver: cluster
          params:
            - name: kind
              value: task
            - name: name
              value: buildah
            - name: namespace
              value: openshift-pipelines
        workspaces:
          - name: source
            workspace: shared-workspace
          - name: dockerconfig
            workspace: dockerconfig
          - name: rhel-entitlement 3
            workspace: rhel-entitlement
        params:
          - name: IMAGE
            value: <image_where_you_want_to_push>

```

■

- 1 The definition of the **rhel-entitlement** workspace in the pipeline run, assigning the **shared-rhel-entitlement** CSI shared secret to the workspace
- 2 The definition of the **rhel-entitlement** workspace in the pipeline definition
- 3 The definition of the **rhel-entitlement** workspace in the task definition

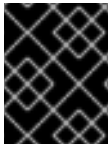
5.4. ADDITIONAL RESOURCES

- [Simple content access](#)
- [Using Insights Operator](#)
- [Importing simple content access entitlements with Insights Operator](#)
- [Shared Resource CSI Driver Operator](#)
- [Changing the default service account for OpenShift Pipelines](#)

CHAPTER 6. MANAGING NON-VERSIONED AND VERSIONED CLUSTER TASKS

As a cluster administrator, installing the Red Hat OpenShift Pipelines Operator creates variants of each default cluster task known as *versioned cluster tasks* (VCT) and *non-versioned cluster tasks* (NVCT). For example, installing the Red Hat OpenShift Pipelines Operator v1.7 creates a **buildah-1-7-0** VCT and a **buildah** NVCT.

Both NVCT and VCT have the same metadata, behavior, and specifications, including **params**, **workspaces**, and **steps**. However, they behave differently when you disable them or upgrade the Operator.



IMPORTANT

In Red Hat OpenShift Pipelines 1.10, **ClusterTask** functionality is deprecated and is planned to be removed in a future release.

6.1. DIFFERENCES BETWEEN NON-VERSIONED AND VERSIONED CLUSTER TASKS

Non-versioned and versioned cluster tasks have different naming conventions. And, the Red Hat OpenShift Pipelines Operator upgrades them differently.

Table 6.1. Differences between non-versioned and versioned cluster tasks

	Non-versioned cluster task	Versioned cluster task
Nomenclature	The NVCT only contains the name of the cluster task. For example, the name of the NVCT of Buildah installed with Operator v1.7 is buildah .	The VCT contains the name of the cluster task, followed by the version as a suffix. For example, the name of the VCT of Buildah installed with Operator v1.7 is buildah-1-7-0 .
Upgrade	When you upgrade the Operator, it updates the non-versioned cluster task with the latest changes. The name of the NVCT remains unchanged.	Upgrading the Operator installs the latest version of the VCT and retains the earlier version. The latest version of a VCT corresponds to the upgraded Operator. For example, installing Operator 1.7 installs buildah-1-7-0 and retains buildah-1-6-0 .

6.2. ADVANTAGES AND DISADVANTAGES OF NON-VERSIONED AND VERSIONED CLUSTER TASKS

Before adopting non-versioned or versioned cluster tasks as a standard in production environments, cluster administrators might consider their advantages and disadvantages.

Table 6.2. Advantages and disadvantages of non-versioned and versioned cluster tasks

Cluster task	Advantages	Disadvantages
Non-versioned cluster task (NVCT)	<ul style="list-style-type: none"> ● If you prefer deploying pipelines with the latest updates and bug fixes, use the NVCT. ● Upgrading the Operator upgrades the non-versioned cluster tasks, which consume fewer resources than multiple versioned cluster tasks. 	If you deploy pipelines that use NVCT, they might break after an Operator upgrade if the automatically upgraded cluster tasks are not backward-compatible.
Versioned cluster task (VCT)	<ul style="list-style-type: none"> ● If you prefer stable pipelines in production, use the VCT. ● The earlier version is retained on the cluster even after the later version of a cluster task is installed. You can continue using the earlier cluster tasks. 	<ul style="list-style-type: none"> ● If you continue using an earlier version of a cluster task, you might miss the latest features and critical security updates. ● The earlier versions of cluster tasks that are not operational consume cluster resources. ● * After it is upgraded, the Operator cannot manage the earlier VCT. You can delete the earlier VCT manually by using the oc delete clustertask command, but you cannot restore it.

6.3. DISABLING NON-VERSIONED AND VERSIONED CLUSTER TASKS

As a cluster administrator, you can disable cluster tasks that the OpenShift Pipelines Operator installed.

Procedure

1. To delete all non-versioned cluster tasks and latest versioned cluster tasks, edit the **TektonConfig** custom resource definition (CRD) and set the **clusterTasks** parameter in **spec.addon.params** to **false**.

Example TektonConfig CR

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  params:
    - name: createRbacResource
```

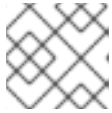


```

value: "false"
profile: all
targetNamespace: openshift-pipelines
addon:
  params:
    - name: clusterTasks
      value: "false"
...

```

When you disable cluster tasks, the Operator removes all the non-versioned cluster tasks and only the latest version of the versioned cluster tasks from the cluster.



NOTE

Re-enabling cluster tasks installs the non-versioned cluster tasks.

2. Optional: To delete earlier versions of the versioned cluster tasks, use any one of the following methods:
 - a. To delete individual earlier versioned cluster tasks, use the **oc delete clustertask** command followed by the versioned cluster task name. For example:

```
$ oc delete clustertask buildah-1-6-0
```

- b. To delete all versioned cluster tasks created by an old version of the Operator, you can delete the corresponding installer set. For example:

```
$ oc delete tektoninstallerset versioned-clustertask-1-6-k98as
```

CAUTION

If you delete an old versioned cluster task, you cannot restore it. You can only restore versioned and non-versioned cluster tasks that the current version of the Operator has created.