OpenShift Container Platform 4.5

Pipelines

Configuring and using Pipelines in OpenShift Container Platform

Last Updated: 2021-01-06

OpenShift Container Platform 4.5 Pipelines

Configuring and using Pipelines in OpenShift Container Platform

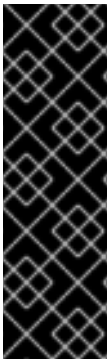
Legal Notice

Abstract

This document provides instructions for configuring and using pipelines in OpenShift Container Platform.

Table of Contents

CHAPTER 1. UNDERSTANDING OPENSHIFT PIPELINES



IMPORTANT

OpenShift Pipelines 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 https://access.redhat.com/support/offerings/techpreview/.

Red Hat OpenShift Pipelines is a cloud-native, continuous integration and continuous delivery (CI/CD) solution based on Kubernetes resources. It uses Tekton building blocks to automate deployments across multiple platforms by abstracting away the underlying implementation details. Tekton introduces a number of standard Custom Resource Definitions (CRDs) for defining CI/CD pipelines that are portable across Kubernetes distributions.

1.1. KEY FEATURES

- Red Hat OpenShift Pipelines is a serverless CI/CD system that runs Pipelines with all the required dependencies in isolated containers.
- Red Hat OpenShift Pipelines are designed for decentralized teams that work on microservice-based architecture.
- Red Hat OpenShift Pipelines use standard CI/CD pipeline definitions that are easy to extend and integrate with the existing Kubernetes tools, enabling you to scale on-demand.
- You can use Red Hat OpenShift Pipelines to build images with Kubernetes tools such as Source-to-Image (S2I), Buildah, Buildpacks, and Kaniko that are portable across any Kubernetes platform.
- You can use the OpenShift Container Platform Developer Console to create Tekton resources, view logs of Pipeline runs, and manage pipelines in your OpenShift Container Platform namespaces.

1.2. RED HAT OPENSHIFT PIPELINES CONCEPTS

Red Hat OpenShift Pipelines provide a set of standard Custom Resource Definitions (CRDs) that act as the building blocks from which you can assemble a CI/CD pipeline for your application.

Task

A Task is the smallest configurable unit in a Pipeline. It is essentially a function of inputs and outputs that form the Pipeline build. It can run individually or as a part of a Pipeline. A Pipeline includes one or more Tasks, where each Task consists of one or more Steps. Steps are a series of commands that are sequentially executed by the Task.

Pipeline

A Pipeline consists of a series of Tasks that are executed to construct complex workflows that automate the build, deployment, and delivery of applications. It is a collection of PipelineResources, parameters, and one or more Tasks. A Pipeline interacts with the outside world by using PipelineResources, which are added to Tasks as inputs and outputs.

PipelineRun

A PipelineRun is the running instance of a Pipeline. A PipelineRun initiates a Pipeline and manages the creation of a TaskRun for each Task being executed in the Pipeline.

TaskRun

A TaskRun is automatically created by a PipelineRun for each Task in a Pipeline. It is the result of running an instance of a Task in a Pipeline. It can also be manually created if a Task runs outside of a Pipeline.

Workspace

A Workspace is a storage volume that a Task requires at runtime to receive input or provide output. A Task or Pipeline declares the Workspace, and a TaskRun or PipelineRun provides the actual location of the storage volume, which mounts on the declared Workspace. This makes the Task flexible, reusable, and allows the Workspaces to be shared across multiple Tasks.

Trigger

A Trigger captures an external event, such as a Git pull request and processes the event payload to extract key pieces of information. This extracted information is then mapped to a set of predefined parameters, which trigger a series of tasks that may involve creation and deployment of Kubernetes resources. You can use Triggers along with Pipelines to create full-fledged CI/CD systems where the execution is defined entirely through Kubernetes resources.

Condition

A Condition refers to a validation or check, which is executed before a Task is run in your Pipeline. Conditions are like **if** statements which perform logical tests, with a return value of **True** or **False**. A Task is executed if all Conditions return **True**, but if any of the Conditions fail, the Task and all subsequent Tasks are skipped. You can use Conditions in your Pipeline to create complex workflows covering multiple scenarios.

1.3. DETAILED OPENSHIFT PIPELINE CONCEPTS

This guide provides a detailed view of the various Pipeline concepts.

1.3.1. Tasks

Tasks are the building blocks of a Pipeline and consist of sequentially executed Steps. Tasks are reusable and can be used in multiple Pipelines.

Steps are a series of commands that achieve a specific goal, such as building an image. Every Task runs as a pod and each Step runs in its own container within the same pod. Because Steps run within the same pod, they have access to the same volumes for caching files, ConfigMaps, and Secrets.

The following example shows the **apply-manifests** Task.

```
apiVersion: tekton.dev/v1beta1 1
kind: Task 2
metadata:
  name: apply-manifests 3
spec: 4
  params:
  - default: k8s
    description: The directory in source that contains yaml manifests
    name: manifest_dir
    type: string
  steps:
```

```
  - args:
    - |-
      echo Applying manifests in $(inputs.params.manifest_dir) directory
      oc apply -f $(inputs.params.manifest_dir)
      echo -----------------------------------
    command:
    - /bin/bash
    - -c
    image: quay.io/openshift/origin-cli:latest
    name: apply
    workingDir: /workspace/source
  workspaces:
  - name: source
```

1. Task API version **v1beta1**.
2. Specifies the type of Kubernetes object. In this example, **Task**.
3. Unique name of this Task.
4. Lists the parameters and Steps in the Task and the workspace used by the Task.

This Task starts the pod and runs a container inside that pod using the **maven:3.6.0-jdk-8-slim** image to run the specified commands. It receives an input directory called **workspace-git** that contains the source code of the application.

The Task only declares the placeholder for the Git repository, it does not specify which Git repository to use. This allows Tasks to be reusable for multiple Pipelines and purposes.

1.3.2. TaskRun

A *TaskRun* instantiates a Task for execution with specific inputs, outputs, and execution parameters on a cluster. It can be invoked on its own or as part of a PipelineRun.

A Task consists of one or more Steps that execute container images, and each container image performs a specific piece of build work. A TaskRun executes the Steps in a Task in the specified order, until all Steps execute successfully or a failure occurs.

The following example shows a TaskRun that runs the **apply-manifests** Task with the relevant input parameters:

```
apiVersion: tekton.dev/v1beta1 1
kind: TaskRun 2
metadata:
  name: apply-manifests-taskrun 3
spec: 4
  serviceAccountName: pipeline
  taskRef: 5
    kind: Task
    name: apply-manifests
  workspaces: 6
  - name: source
    persistentVolumeClaim:
      claimName: source-pvc
```

1. TaskRun API version **v1beta1**.
2. Specifies the type of Kubernetes object. In this example, **TaskRun**.
3. Unique name to identify this TaskRun.
4. Definition of the TaskRun. For this TaskRun, the Task and the required workspace are specified.
5. Name of the Task reference used for this TaskRun. This TaskRun executes the **apply-manifests** Task.
6. Workspace used by the TaskRun.

1.3.3. Pipelines

A *Pipeline* is a collection of Tasks arranged in a specific order of execution. You can define a CI/CD workflow for your application using Pipelines containing one or more Tasks.

A Pipeline definition consists of a number of fields or attributes, which together enable the Pipeline to accomplish a specific goal. Each Pipeline definition must contain at least one Task, which ingests specific inputs and produces specific outputs. The Pipeline definition can also optionally include Conditions, Workspaces, Parameters, or Resources depending on the application requirements.

The following example shows the **build-and-deploy** Pipeline, which builds an application image from a Git repository using the **buildah** ClusterTask:

```
apiVersion: tekton.dev/v1beta1 1
kind: Pipeline 2
metadata:
  name: build-and-deploy 3
spec: 4
  workspaces: 5
  - name: shared-workspace
  params: 6
  - 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: "release-tech-preview-2"
  - name: IMAGE
    type: string
    description: image to be build from the code
  tasks: 7
  - name: fetch-repository
    taskRef:
      name: git-clone
      kind: ClusterTask
    workspaces:
    - name: output
```

```
      workspace: shared-workspace
    params:
    - name: url
      value: $(params.git-url)
    - name: subdirectory
      value: ""
    - name: deleteExisting
      value: "true"
    - name: revision
      value: $(params.git-revision)
  - name: build-image 8
    taskRef:
      name: buildah
      kind: ClusterTask
    params:
    - name: TLSVERIFY
      value: "false"
    - name: IMAGE
      value: $(params.IMAGE)
    workspaces:
    - name: source
      workspace: shared-workspace
    runAfter:
    - fetch-repository
  - name: apply-manifests 9
    taskRef:
      name: apply-manifests
    workspaces:
    - name: source
      workspace: shared-workspace
    runAfter: 10
    - build-image
  - name: update-deployment
    taskRef:
      name: update-deployment
    workspaces:
    - name: source
      workspace: shared-workspace
    params:
    - name: deployment
      value: $(params.deployment-name)
    - name: IMAGE
      value: $(params.IMAGE)
    runAfter:
    - apply-manifests
```

1. Pipeline API version **v1beta1**.
2. Specifies the type of Kubernetes object. In this example, **Pipeline**.
3. Unique name of this Pipeline.
4. Specifies the definition and structure of the Pipeline.
5. Workspaces used across all the Tasks in the Pipeline.

6. Parameters used across all the Tasks in the Pipeline.
7. Specifies the list of Tasks used in the Pipeline.
8. Task **build-image**, which uses the **buildah** ClusterTask to build application images from a given Git repository.
9. Task **apply-manifests**, which uses a user-defined Task with the same name.
10. Specifies the sequence in which Tasks are run in a Pipeline. In this example, the **apply-manifests** Task is run only after the **build-image** Task is completed.

1.3.4. PipelineRun

A *PipelineRun* instantiates a Pipeline for execution with specific inputs, outputs, and execution parameters on a cluster. A corresponding TaskRun is created for each Task automatically in the PipelineRun.

All the Tasks in the Pipeline are executed in the defined sequence until all Tasks are successful or a Task fails. The **status** field tracks and stores the progress of each TaskRun in the PipelineRun for monitoring and auditing purpose.

The following example shows a PipelineRun to run the **build-and-deploy** Pipeline with relevant resources and parameters:

```
apiVersion: tekton.dev/v1beta1 1
kind: PipelineRun 2
metadata:
  name: build-deploy-api-pipelinerun 3
spec:
  pipelineRef:
    name: build-and-deploy 4
  params: 5
  - name: deployment-name
    value: vote-api
  - name: git-url
    value: http://github.com/openshift-pipelines/vote-api.git
  - name: IMAGE
    value: image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/vote-api
  workspaces: 6
  - name: shared-workspace
    persistentvolumeclaim:
      claimName: source-pvc
```

1. PipelineRun API version **v1beta1**.
2. Specifies the type of Kubernetes object. In this example, **PipelineRun**.
3. Unique name to identify this PipelineRun.
4. Name of the Pipeline to be run. In this example, **build-and-deploy**.
5. Specifies the list of parameters required to run the Pipeline.

6 Workspace used by the PipelineRun.

1.3.5. Workspaces

NOTE

It is recommended that you use Workspaces instead of PipelineResources in OpenShift Pipelines, as PipelineResources are difficult to debug, limited in scope, and make Tasks less reusable.

Workspaces declare shared storage volumes that a Task in a Pipeline needs at runtime. Instead of specifying the actual location of the volumes, Workspaces enable you to declare the filesystem or parts of the filesystem that would be required at runtime. You must provide the specific location details of the volume that is mounted into that Workspace in a TaskRun or a PipelineRun. This separation of volume declaration from runtime storage volumes makes the Tasks reusable, flexible, and independent of the user environment.

With Workspaces, you can:

- Store Task inputs and outputs
- Share data among Tasks
- Use it as a mount point for credentials held in Secrets
- Use it as a mount point for configurations held in ConfigMaps
- Use it as a mount point for common tools shared by an organization
- Create a cache of build artifacts that speed up jobs

You can specify Workspaces in the TaskRun or PipelineRun using:

- A read-only ConfigMaps or Secret
- An existing PersistentVolumeClaim shared with other Tasks
- A PersistentVolumeClaim from a provided VolumeClaimTemplate
- An emptyDir that is discarded when the TaskRun completes

The following example shows a code snippet of the **build-and-deploy** Pipeline, which declares a **shared-workspace** Workspace for the **build-image** and **apply-manifests** Tasks as defined in the Pipeline.

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
  workspaces: 1
  - name: shared-workspace
  params:
...
  tasks: 2
```

```
  - name: build-image
    taskRef:
      name: buildah
      kind: ClusterTask
    params:
    - name: TLSVERIFY
      value: "false"
    - name: IMAGE
      value: $(params.IMAGE)
    workspaces: 3
    - name: source 4
      workspace: shared-workspace 5
    runAfter:
    - fetch-repository
  - name: apply-manifests
    taskRef:
      name: apply-manifests
    workspaces: 6
    - name: source
      workspace: shared-workspace
    runAfter:
     - build-image
...
```

1. List of Workspaces shared between the Tasks defined in the Pipeline. A Pipeline can define as many Workspaces as required. In this example, only one Workspace named **shared-workspace** is declared.

2. Definition of Tasks used in the Pipeline. This snippet defines two Tasks, **build-image** and **apply-manifests**, which share a common Workspace.

3. List of Workspaces used in the **build-image** Task. A Task definition can include as many Workspaces as it requires. However, it is recommended that a Task uses at most one writable Workspace.

4. Name that uniquely identifies the Workspace used in the Task. This Task uses one Workspace named **source**.

5. Name of the Pipeline Workspace used by the Task. Note that the Workspace **source** in turn uses the Pipeline Workspace named **shared-workspace**.

6. List of Workspaces used in the **apply-manifests** Task. Note that this Task shares the **source** Workspace with the **build-image** Task.

Here is a code snippet of the **build-deploy-api-pipelinerun** PipelineRun, which uses a PersistentVolumeClaim for defining the storage volume for the **shared-workspace** Workspace used in the **build-and-deploy** Pipeline.

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: build-deploy-api-pipelinerun
spec:
  pipelineRef:
    name: build-and-deploy
```

```
  params:
...

  workspaces: (1)
  - name: shared-workspace (2)
    persistentvolumeclaim:
      claimName: source-pvc (3)
```

(1) Specifies the list of Pipeline Workspaces for which volume binding will be provided in the PipelineRun.

(2) The name of the Workspace in the Pipeline for which the volume is being provided.

(3) Specifies the name of a predefined PersistentVolumeClaim, which will be attached to the Workspace. In this example, an existing **source-pvc** PersistentVolumeClaim is attached with the **shared-workspace** Workspace.

1.3.6. Triggers

Use Triggers in conjunction with Pipelines to create a full-fledged CI/CD system where the Kubernetes resources define the entire CI/CD execution. Pipeline Triggers capture the external events and process them to extract key pieces of information. Mapping this event data to a set of predefined parameters triggers a series of tasks that can then create and deploy Kubernetes resources.

For example, you define a CI/CD workflow using Red Hat OpenShift Pipelines for your application. The PipelineRun must start for any new changes to take effect in the application repository. Triggers automate this process by capturing and processing any change events and by triggering a PipelineRun that deploys the new image with the latest changes.

Triggers consist of the following main components that work together to form a reusable, decoupled, and self-sustaining CI/CD system:

- *EventListeners* provide endpoints, or an event sink, that listen for incoming HTTP-based events with a JSON payload. The EventListener performs lightweight event processing on the payload using Event Interceptors, which identify the type of payload and optionally modify it. Currently, Pipeline Triggers support four types of Interceptors: Webhook Interceptors, GitHub Interceptors, GitLab Interceptors, and Common Expression Language (CEL) Interceptors.

- *TriggerBindings* extract the fields from an event payload and store them as parameters.

- *TriggerTemplates* specify how to use the parameterized data from the TriggerBindings. A TriggerTemplate defines a resource template that receives input from the TriggerBindings, and then performs a series of actions that result in creation of new PipelineResources and initiation of a new PipelineRun.

EventListeners tie the concepts of TriggerBindings and TriggerTemplates together. The EventListener listens for the incoming event, handles basic filtering using Interceptors, extracts data using TriggerBindings, and then processes this data to create Kubernetes resources using TriggerTemplates.

The following example shows a code snippet of the **vote-app-binding** TriggerBinding, which extracts the Git repository information from the received event payload:

```
apiVersion: triggers.tekton.dev/v1alpha1 (1)
kind: TriggerBinding (2)
```

```
metadata:
  name: vote-app 3
spec:
  params: 4
  - name: git-repo-url
    value: $(body.repository.url)
  - name: git-repo-name
    value: $(body.repository.name)
  - name: git-revision
    value: $(body.head_commit.id)
```

1. TriggerBinding API version **v1alpha1**.
2. Specifies the type of Kubernetes object. In this example, **TriggerBinding**.
3. Unique name to identify this TriggerBinding.
4. List of parameters which will be extracted from the received event payload and passed to the TriggerTemplate. In this example, the Git repository URL, name, and revision are extracted from the body of the event payload.

The following example shows a code snippet of a **vote-app-template** TriggerTemplate, which creates Pipeline Resources from the Git repository information received from the TriggerBinding:

```
apiVersion: triggers.tekton.dev/v1alpha1 1
kind: TriggerTemplate 2
metadata:
  name: vote-app 3
spec:
  params: 4
  - name: git-repo-url
    description: The git repository url
  - name: git-revision
    description: The git revision
    default: master
  - name: git-repo-name
    description: The name of the deployment to be created / patched

  resourcetemplates: 5
  - apiVersion: tekton.dev/v1beta1
    kind: PipelineRun
    metadata:
      name: build-deploy-$(tt.params.git-repo-name)-$(uid)
    spec:
      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
      persistentvolumeclaim:
        claimName: source-pvc
```

1. TriggerTemplate API version **v1alpha1**.
2. Specifies the type of Kubernetes object. In this example, **TriggerTemplate**.
3. Unique name to identify this TriggerTemplate.
4. Parameters supplied by the TriggerBinding or EventListerner.
5. List of Resource templates created for the Pipeline from the parameters received in the TriggerBinding or EventListener.

The following example shows an EventListener which uses **vote-app-binding** TriggerBinding and **vote-app-template** TriggerTemplate to process incoming events.

```
apiVersion: triggers.tekton.dev/v1alpha1 (1)
kind: EventListener (2)
metadata:
  name: vote-app (3)
spec:
  serviceAccountName: pipeline (4)
  triggers:
  - bindings: (5)
    - ref: vote-app
    template: (6)
      name: vote-app
```

1. EventListener API version **v1alpha1**.
2. Specifies the type of Kubernetes object. In this example, **EventListener**.
3. Unique name to identify this EventListener.
4. Service account name to be used.
5. Name of the TriggerBinding to be used for this EventListener.
6. Name of the Triggertemplate to be used for this Eventlistener.

1.4. ADDITIONAL RESOURCES

- For information on installing Pipelines, see Installing OpenShift Pipelines.
- For more details on creating custom CI/CD solutions, see Creating applications with CI/CD Pipelines.

CHAPTER 2. INSTALLING OPENSHIFT PIPELINES

Prerequisites

- You have access to an OpenShift Container Platform cluster using an account with **cluster-admin** permissions.
- You have installed **oc** CLI.
- You have installed OpenShift Pipelines (**tkn**) CLI on your local system.

2.1. INSTALLING THE RED HAT OPENSHIFT PIPELINES OPERATOR IN WEB CONSOLE

You can install Red Hat OpenShift Pipelines using the Operator listed in the OpenShift Container Platform OperatorHub. When you install the Red Hat OpenShift Pipelines Operator, the Custom Resources (CRs) required for the Pipelines configuration are automatically installed along with the Operator.

Procedure

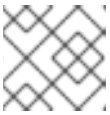
1. In the **Administrator** perspective of the web console, navigate to **Operators → OperatorHub**.
2. Use the **Filter by keyword** box to search for **Red Hat OpenShift Pipelines Operator** in the catalog. Click the **OpenShift Pipelines Operator** tile.

NOTE

Ensure that you do not select the **Community** version of the **OpenShift Pipelines Operator**.

3. Read the brief description about the Operator on the **Red Hat OpenShift Pipelines Operator** page. Click **Install**.
4. On the **Install Operator** page:
   a. Select **All namespaces on the cluster (default)**for the **Installation Mode**. This mode installs the Operator in the default **openshift-operators** namespace, which enables the Operator to watch and be made available to all namespaces in the cluster.
   b. Select **Automatic** for the **Approval Strategy**. This ensures that the future upgrades to the Operator are handled automatically by the Operator Lifecycle Manager (OLM). If you select the **Manual** approval strategy, OLM creates an update request. As a cluster administrator, you must then manually approve the OLM update request to update the Operator to the new version.
   c. Select an **Update Channel**.
      - The **ocp-<4.x>** channel enables installation of the latest stable release of the Red Hat OpenShift Pipelines Operator.
      - The **preview** channel enables installation of the latest preview version of the Red Hat OpenShift Pipelines Operator, which may contain features that are not yet available from the 4.x update channel.

5. Click **Install**. You will see the Operator listed on the **Installed Operators** page.



NOTE

The Operator is installed automatically into the **openshift-operators** namespace.

6. Verify that the **Status** is set to **Succeeded Up to date** to confirm successful installation of Red Hat OpenShift Pipelines Operator.

2.2. INSTALLING THE OPENSHIFT PIPELINES OPERATOR USING THE CLI

You can install Red Hat OpenShift Pipelines Operator from the OperatorHub using the CLI.

Procedure

1. Create a Subscription object YAML file to subscribe a namespace to the Red Hat OpenShift Pipelines Operator, for example, **sub.yaml**:

Example Subscription

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-pipelines-operator
  namespace: openshift-operators
spec:
  channel:  <channel name> 1
  name: openshift-pipelines-operator-rh 2
  source: redhat-operators 3
  sourceNamespace: openshift-marketplace 4
```

1. Specify the channel name from where you want to subscribe the Operator
2. Name of the Operator to subscribe to.
3. Name of the CatalogSource that provides the Operator.
4. Namespace of the CatalogSource. Use **openshift-marketplace** for the default OperatorHub CatalogSources.

2. Create the Subscription object:

```
$ oc apply -f sub.yaml
```

The Red Hat OpenShift Pipelines Operator is now installed in the default target namespace **openshift-operators**.

Additional Resources

- You can learn more about installing Operators on OpenShift Container Platform in the adding Operators to a cluster section.

CHAPTER 3. UNINSTALLING OPENSHIFT PIPELINES

Uninstalling the Red Hat OpenShift Pipelines Operator is a two-step process:

1. Delete the Custom Resources (CRs) that were added by default when you installed the Red Hat OpenShift Pipelines Operator.
2. Uninstall the Red Hat OpenShift Pipelines Operator.

Uninstalling only the Operator will not remove the Red Hat OpenShift Pipelines components created by default when the Operator is installed.

3.1. DELETING THE RED HAT OPENSHIFT PIPELINES COMPONENTS AND CUSTOM RESOURCES

Delete the Custom Resources (CRs) created by default during installation of the Red Hat OpenShift Pipelines Operator.

Procedure

1. In the **Administrator** perspective of the web console, navigate to **Administration → Custom Resource Definition**.
2. Type **config.operator.tekton.dev** in the **Filter by name** box to search for the Red Hat OpenShift Pipelines Operator CRs.
3. Click **CRD Config** to see the **Custom Resource Definition Details**page.
4. Click the **Actions** drop-down menu and select **Delete Custom Resource Definition**

NOTE

Deleting the CRs will delete the Red Hat OpenShift Pipelines components, and all the Tasks and Pipelines on the cluster will be lost.

5. Click **Delete** to confirm the deletion of the CRs.

3.2. UNINSTALLING THE RED HAT OPENSHIFT PIPELINES OPERATOR

Procedure

1. From the **Operators → OperatorHub** page, use the **Filter by keyword** box to search for **Red Hat OpenShift Pipelines Operator**.
2. Click the **OpenShift Pipelines Operator** tile. The Operator tile indicates it is installed.
3. In the **OpenShift Pipelines Operator** descriptor page, click **Uninstall**.

Additional Resources

- You can learn more about uninstalling Operators on OpenShift Container Platform in the deleting Operators from a cluster section.

CHAPTER 4. CREATING CI/CD SOLUTIONS FOR APPLICATIONS USING OPENSHIFT 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, you must perform the following tasks:

- Create custom Tasks, or install existing reusable Tasks.
- Create and define the delivery Pipeline for your application.
- Create a PersistentVolumeClaim attached to the Workspace to provide the volume or filesystem for Pipeline execution.
- Create a PipelineRun to instantiate and invoke the Pipeline.
- Add Triggers to capture any 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, **vote-ui**, with the source code in the **ui-repo** Git repository.
- A back-end interface, **vote-api**, with the source code in the **api-repo** Git repository.
- The **apply_manifest** and **update-deployment** Tasks in the **pipelines-tutorial** Git repository.

4.1. 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. Once installed, it is applicable to the entire cluster.
- You have installed OpenShift Pipelines CLI.
- You have forked the front-end **ui-repo** and back-end **api-repo** Git repositories using your GitHub ID, and have Administrator access to these repositories.
- Optional: You have cloned the **pipelines-tutorial** Git repository.

4.2. CREATING A PROJECT AND CHECKING YOUR PIPELINE SERVICEACCOUNT

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

```
$ oc get serviceaccount pipeline
```

4.3. CREATING PIPELINE TASKS

Procedure

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

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/release-tech-preview-2/01_pipeline/01_apply_manifest_task.yaml
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/release-tech-preview-2/01_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** Tasks 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 ClusterTasks, for example --**buildah** and **s2i-python-3**:

NOTE

You must use a privileged Pod container to run the **buildah** ClusterTask because it requires a privileged security context. To learn more about security context constraints (SCC) for pods, see the Additional resources section.

```
$ tkn clustertasks list
```

The output lists the Operator-installed ClusterTasks:

```
NAME                    DESCRIPTION   AGE
```

```
buildah                 1 day ago
git-clone               1 day ago
s2i-php                 1 day ago
tkn                   1 day ago
```

4.4. 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 **vote-api** and front-end application **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** ClusterTask.
- Pushes the image to the internal 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/v1beta1
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: "release-tech-preview-2"
  - name: IMAGE
    type: string
    description: image to be build from the code
```

```
  tasks:
  - name: fetch-repository
    taskRef:
      name: git-clone
      kind: ClusterTask
    workspaces:
    - name: output
      workspace: shared-workspace
    params:
    - name: url
      value: $(params.git-url)
    - name: subdirectory
      value: ""
    - name: deleteExisting
      value: "true"
    - name: revision
      value: $(params.git-revision)
  - name: build-image
    taskRef:
      name: buildah
      kind: ClusterTask
    params:
    - name: TLSVERIFY
      value: "false"
    - name: IMAGE
      value: $(params.IMAGE)
    workspaces:
    - name: source
      workspace: shared-workspace
    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
    workspaces:
    - name: source
      workspace: shared-workspace
    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/release-tech-preview-2/01_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   ---        ---       ---        ---
```

4.5. SPECIFYING PERSISTENTVOLUMECLAIMS AS VOLUMESOURCE IN WORKSPACES

Workspaces help Tasks share data, and allow you to specify one or more volumes that each Task in the Pipeline requires during execution.

In this section, you will create a PersistentVolumeClaim to provide data storage and bind it to the Workspace. This PersistentVolumeClaim provides the volumes or filesystem required for the Pipeline execution.

Procedure

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

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: source-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
```

2. Create the PersistentVolumeClaim, specifying the file you just created:

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

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

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/release-tech-preview-2/01_pipeline/03_persistent_volume_claim.yaml
```

4.6. RUNNING A PIPELINE

A PipelineRun 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 TaskRuns 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,claimName=source-pvc -p
deployment-name=vote-api -p git-url=http://github.com/openshift-pipelines/vote-api.git -p
IMAGE=image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/vote-api
```

   Note the PipelineRun ID returned in the command output.

2. Track the PipelineRun progress:

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

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

```
$ tkn pipeline start build-and-deploy -w name=shared-workspace,claimName=source-pvc -p
deployment-name=vote-api -p git-url=http://github.com/openshift-pipelines/vote-ui.git -p
IMAGE=image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/vote-ui
```

   Note the PipelineRun ID returned in the command output.

4. Track the PipelineRun progress:

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

5. After a few minutes, use **tkn pipelinerun list** command to verify that the Pipeline ran successfully by listing all the PipelineRuns:

```
$ tkn pipelinerun list
```

   The output lists the PipelineRuns:

```
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
```

6. Get the application route:

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

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

7. To rerun the last PipelineRun, using the PipelineResources and ServiceAccount of the previous Pipeline, run:

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

4.7. ADDING TRIGGERS TO A PIPELINE

Triggers enable Pipelines to respond to external GitHub events, such as push events and pull requests. After you have assembled and started the Pipeline for the application, add TriggerBindings, TriggerTemplates, and an EventListener to capture the GitHub events.

Procedure

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

```
apiVersion: triggers.tekton.dev/v1alpha1
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**:

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

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

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/release-tech-preview-2/03_triggers/01_binding.yaml
```

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

```
apiVersion: triggers.tekton.dev/v1alpha1
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: release-tech-preview-2
  - name: git-repo-name
    description: The name of the deployment to be created / patched

  resourcetemplates:
  - apiVersion: tekton.dev/v1beta1
    kind: PipelineRun
    metadata:
      name: build-deploy-$(params.git-repo-name)-$(uid)
```

```
    spec:
      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
        persistentvolumeclaim:
          claimName: source-pvc
```

4. Create the **TriggerTemplate**:

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

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

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/release-tech-preview-2/03_triggers/02_template.yaml
```

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

```
apiVersion: triggers.tekton.dev/v1alpha1
kind: EventListener
metadata:
  name: vote-app
spec:
  serviceAccountName: pipeline
  triggers:
  - bindings:
    - name: vote-app
    template:
      name: vote-app
```

6. Create the **EventListener**:

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

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

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/release-tech-preview-2/03_triggers/03_event_listener.yaml
```

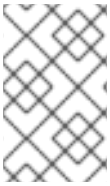
7. Expose the EventListener service as an OpenShift Container Platform route to make it publicly accessible:

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

4.8. CREATING WEBHOOKS

Webhooks are HTTP POST messages that are received by the EventListeners whenever a configured event occurs in your repository. The event payload is then mapped to TriggerBindings, and processed by TriggerTemplates. The TriggerTemplates eventually start one or more PipelineRuns, leading to the creation and deployment of Kubernetes resources.

In this section, you will configure a Webhook URL on your forked Git repositories **vote-ui** and **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:

```
$ 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 **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 **vote-api**.

4.9. TRIGGERING A PIPELINERUN

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 TriggerBindings and TriggerTemplates pair. The TriggerBinding extracts the parameters and the TriggerTemplate uses these parameters to create resources. This may rebuild and redeploy the application.

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

Procedure

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

```
$ git clone git@github.com:<your GitHub ID>/vote-ui.git -b release-tech-preview-2
```

2. Push an empty commit:

```
$ git commit -m "empty-commit" --allow-empty && git push origin release-tech-preview-2
```

3. Check if the PipelineRun was triggered:

```
$ tkn pipelinerun list
```

   Notice that a new PipelineRun was initiated.

4.10. ADDITIONAL RESOURCES

- For more details on pipelines in the **Developer** perspective, see the working with Pipelines in the **Developer** perspective section.
- To learn more about Security Context Constraints (SCCs), see 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.

CHAPTER 5. WORKING WITH RED HAT OPENSHIFT PIPELINES USING THE DEVELOPER PERSPECTIVE

You can use the **Developer** perspective of the OpenShift Container Platform web console to create CI/CD Pipelines for your software delivery process.

In the **Developer** perspective:

- Use the **Add → Pipeline → Pipeline Builder** option to create customized Pipelines for your application.
- Use the **Add → From Git** option to create Pipelines using operator-installed Pipeline templates and resources while creating an application on OpenShift Container Platform.

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 need to apply custom labels to a Pipeline created using the **Pipeline Builder** to see it in the **Topology** view.

Prerequisites

- You have access to an OpenShift Container Platform cluster and have switched to the Developer perspective in the web console.
- 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.

5.1. CONSTRUCTING PIPELINES USING THE PIPELINE BUILDER

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

- Construct a Pipeline flow using existing Tasks and ClusterTasks. When you install the OpenShift Pipelines Operator, it adds reusable Pipeline ClusterTasks to your cluster.
- 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.
- The parameters for a Task are prepopulated based on the specifications of the Task. If required, reference any additional parameters added to the Pipeline in the Task.

Procedure

1. In the **Add** view of the **Developer** perspective, click the **Pipeline** tile to see the **Pipeline Builder** page.
2. Enter a unique name for the Pipeline.

3. Select a Task from the **Select task** list to add a Task to the Pipeline. This example uses the **s2i-nodejs** Task.

   a. To add sequential Tasks to the Pipeline, click the plus icon to the right or left of the Task, and from the **Select task** list, select the Task you want to add to the Pipeline. For this example, use the plus icon to the right of the **s2i-nodejs** Task to add an **openshift-client** Task.

   b. To add a parallel Task to the existing Task, click the plus icon displayed below the Task, and from the **Select Task** list, select the parallel Task you want to add to the Pipeline.

   Figure 5.1. Pipeline Builder

4. Click **Add Resources** to specify the name and type of resources that the Pipeline Run will use. These resources are then used by the Tasks in the Pipeline as inputs and outputs. For this example:

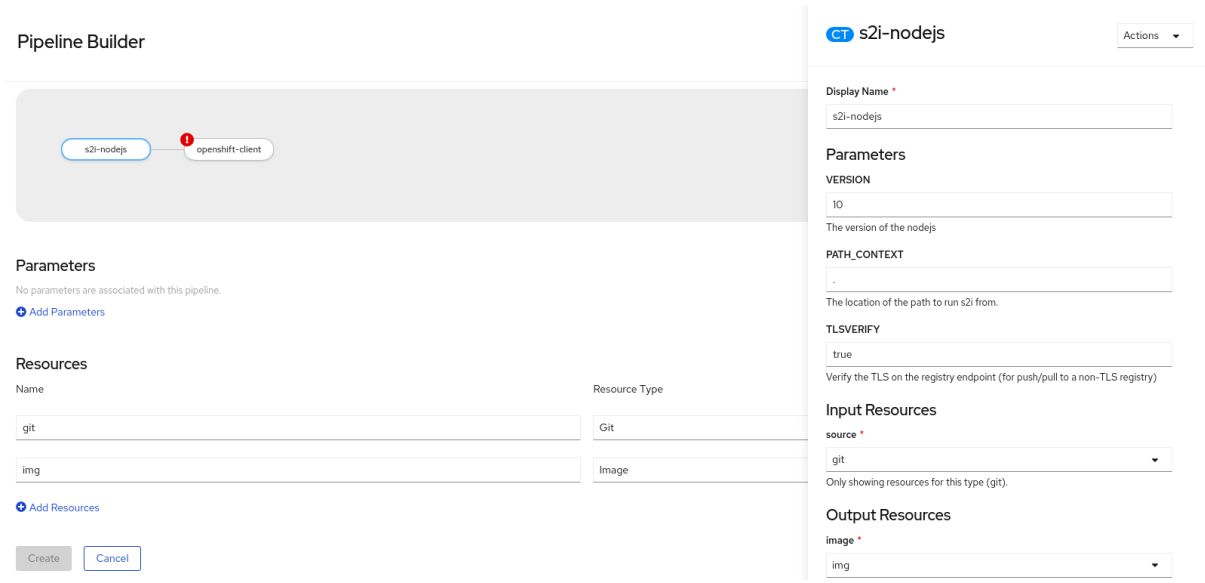
   a. Add an input resource. In the **Name** field, enter **Source**, and from the **Resource Type** drop-down list, select **Git**.

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

5. The **Parameters** for a Task are prepopulated based on the specifications of the Task. If required, use the **Add Parameters** link to add additional parameters.

6. A **Missing Resources** warning is displayed on a Task if the resources for the Task are not specified. Click the **s2i-nodejs** Task to see the side panel with details for the Task.

Figure 5.2. Tasks details in Pipelines Builder



7. In the Task side panel, specify the resources and parameters for it:
   a. In the **Input Resources → Source** section, the **Select Resources** drop-down list displays the resources that you added to the Pipeline. For this example, select **Source**.
   b. In the **Output Resources → Image** section, click the **Select Resources** list, and select **Img**.
   c. If required, in the **Parameters** section, add more parameters to the default ones, by using the **$(params.<param-name>)** syntax.
8. Similarly, add an input resource for the **openshift-client** Task.
9. Click **Create** to create the Pipeline. You are redirected to the **Pipeline Details** page that displays the details of the created Pipeline. You can now use the **Action** button to start the Pipeline.

Optionally, you can also use the **Edit YAML** link, on the upper right of the **Pipeline Builder** page, to directly modify a Pipeline YAML file in the console. You can also use the operator-installed, reusable snippets and samples to create detailed Pipelines.

5.2. CREATING APPLICATIONS WITH OPENSHIFT PIPELINES

To create Pipelines along with applications, use the **From Git** option in the **Add** view of the **Developer** perspective. For more information, see Creating applications using the Developer perspective .

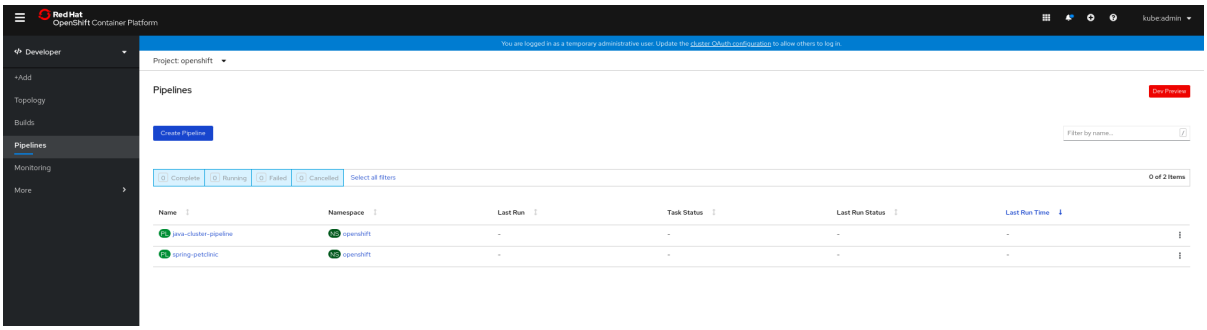
5.3. INTERACTING WITH PIPELINES USING THE DEVELOPER PERSPECTIVE

The **Pipelines** view in the **Developer** perspective lists all the Pipelines in a project along with details, such as the namespace in which the Pipeline was created, the last PipelineRun, the status of the Tasks in the PipelineRun, the status of the PipelineRun, and the time taken for the 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.

Figure 5.3. Pipelines view in the Developer perspective



2. Click the required Pipeline to see the **Pipeline Details** page. This page provides a visual representation of all the serial and parallel Tasks in the Pipeline. The Tasks are also listed at the lower right of the page. You can click the listed **Tasks** to view Task details.

3. Optionally, in the **Pipeline Details** page:

   - Click the **YAML** tab to edit the YAML file for the Pipeline.
   - Click the **Pipeline Runs** tab to see the completed, running, or failed runs for the Pipeline.

     You can 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 PipelineRun.
   - Click the **Parameters** tab to see the parameters defined in the Pipeline. You can also add or edit additional parameters as required.
   - Click the **Resources** tab to see the resources defined in the Pipeline. You can also add or edit additional resources as required.

5.4. STARTING PIPELINES

After you create a Pipeline, you need to start it to execute the included Tasks in the defined sequence. You can start a Pipeline Run from the **Pipelines** view, **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 templates.

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 credentials to authenticate the specified private Git server or Docker 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 Crendentials**, 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 for the **SSH Private Key** field.

      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 PipelineRun.

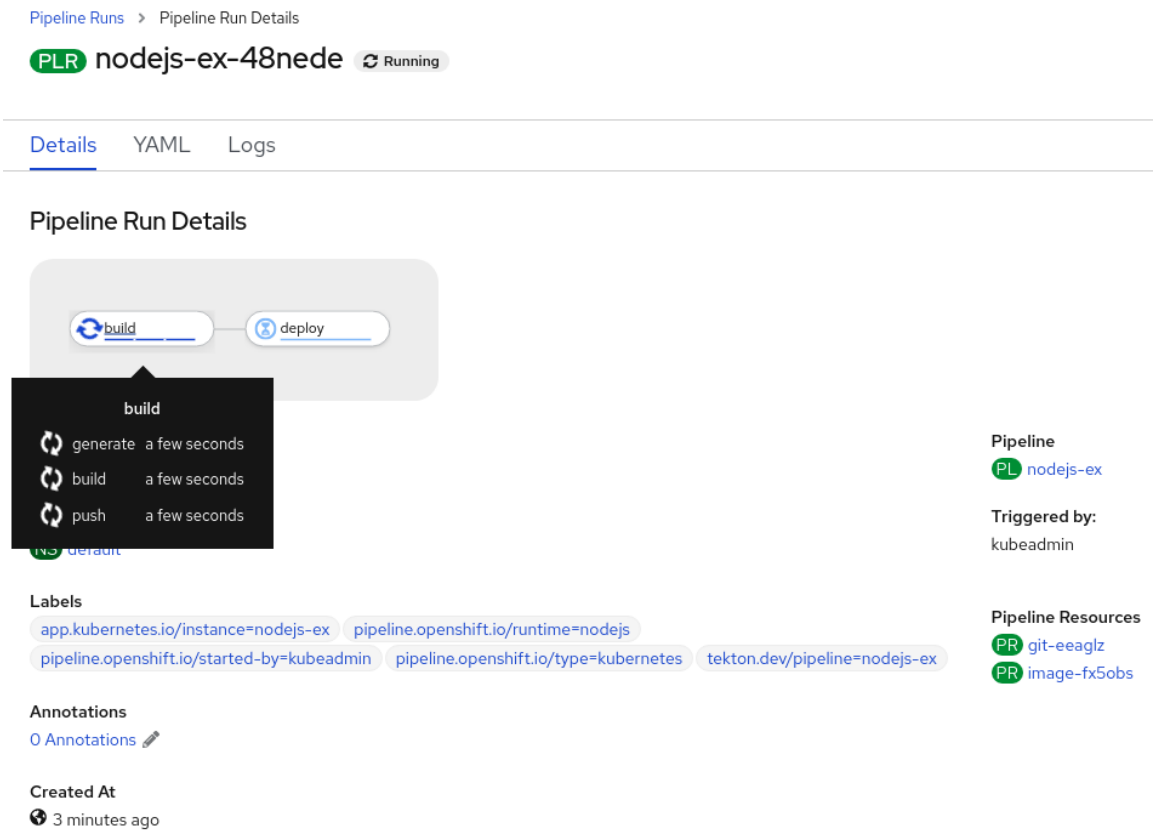
5. The **Pipeline Run Details** page displays the Pipeline being executed. After the Pipeline starts, the Tasks and Steps within each Task are executed. You can:

   - Hover over the Tasks to see the time taken for the execution of each Step.

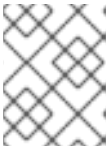
   - Click on a Task to see logs for each of the Steps in the Task.

   - Click the **Logs** tab to see the logs according to the execution sequence of the Tasks and use the **Download** button to download the logs to a text file.

Figure 5.4. Pipeline run



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



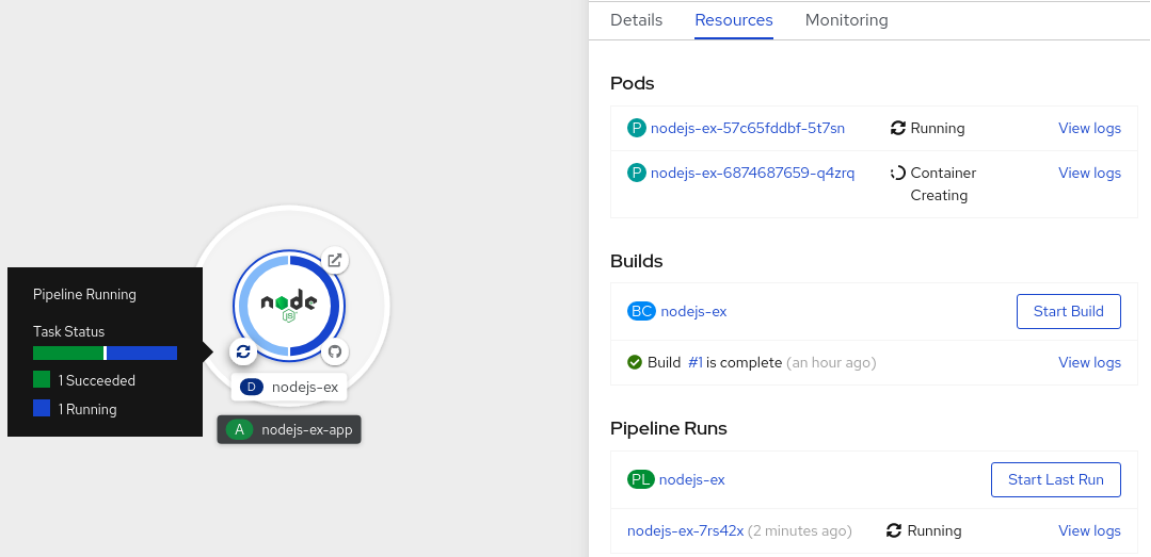
NOTE

To see Pipelines created using the **Pipeline Builder** in the **Topology** view, customize the Pipeline labels to link the Pipeline with the application workload.

   a. On the left navigation panel, click **Topology**, and click on the application to see the Pipeline Runs listed in the side panel.

   b. In the **Pipeline Runs** section, click **Start Last Run** to start a new Pipeline Run with the same parameters and resources as the previous ones. This option is disabled if a Pipeline Run has not been initiated.

Figure 5.5. Pipelines on the Topology view



c. In the **Topology** page, hover to the left of the application to see the status of the Pipeline Run for the application.

5.5. 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. In the **Pipeline Builder** page:
   - You can add additional Tasks, parameters, or resources to the Pipeline.
   - You can 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.

5.6. 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.

CHAPTER 6. RED HAT OPENSHIFT PIPELINES RELEASE NOTES

Red Hat OpenShift Pipelines is a cloud-native CI/CD experience based on the Tekton project which provides:

- Standard Kubernetes-native pipeline definitions (CRDs).
- Serverless pipelines with no CI server management overhead.
- Extensibility to build images using any Kubernetes tool, such as S2I, Buildah, JIB, and Kaniko.
- Portability across any Kubernetes distribution.
- Powerful CLI for interacting with pipelines.
- Integrated user experience with the Developer perspective of the OpenShift Container Platform web console.

For an overview of Red Hat OpenShift Pipelines, see Understanding OpenShift Pipelines.

6.1. GETTING SUPPORT

If you experience difficulty with a procedure described in this documentation, visit the Red Hat Customer Portal to learn more about Red Hat Technology Preview features support scope .

For questions and feedback, you can send an email to the product team at pipelines-interest@redhat.com.

6.2. RELEASE NOTES FOR RED HAT RED HAT OPENSHIFT PIPELINES TECHNOLOGY PREVIEW 1.1

6.2.1. New features

Red Hat OpenShift Pipelines Technology Preview (TP) 1.1 is now available on OpenShift Container Platform 4.5. Red Hat OpenShift Pipelines TP 1.1 is updated to support:

- Tekton Pipelines 0.14.3
- Tekton **tkn** CLI 0.11.0
- Tekton Triggers 0.6.1
- ClusterTasks based on Tekton Catalog 0.14

In addition to the fixes and stability improvements, here is a highlight of what's new in OpenShift Pipelines 1.1.

6.2.1.1. Pipelines

- Workspaces can now be used instead of PipelineResources. It is recommended that you use Workspaces in OpenShift Pipelines, as PipelineResources are difficult to debug, limited in scope, and make Tasks less reusable. For more details on Workspaces, see Understanding OpenShift Pipelines.

- Workspace support for VolumeClaimTemplates has been added:
    - The VolumeClaimTemplate for a PipelineRun and TaskRun can now be added as a volume source for Workspaces. The tekton-controller then creates a PersistentVolumeClaim (PVC) using the template that is seen as a PVC for all TaskRuns in the Pipeline. Thus you do not need to define the PVC configuration every time it binds a workspace that spans multiple tasks.
    - Support to find the name of the PersistentVolumeClaim when a VolumeClaimTemplate is used as a volume source is now available using variable substitution.
- Support for improving audits:
    - The **PipelineRun.Status** field now contains the status of every TaskRun in the Pipeline and the Pipeline specification used to instantiate a PipelineRun to monitor the progress of the PipelineRun.
    - Pipeline results have been added to the pipeline specification and **PipelineRun** status.
    - The **TaskRun.Status** field now contains the exact Task specification used to instantiate the **TaskRun**.
- Support to apply the default parameter to Conditions.
- A TaskRun created by referencing a ClusterTask now adds the **tekton.dev/clusterTask** label instead of the **tekton.dev/task** label.
- The **kubeconfigwriter** now adds the **ClientKeyData** and the **ClientCertificateData** configurations in the Resource structure to enable replacement of the pipeline resource type cluster with the kubeconfig-creator Task.
- The names of the **feature-flags** and the **config-defaults** ConfigMaps are now customizable.
- Support for HostNetwork in the PodTemplate used by TaskRun is now available.
- An Affinity Assistant is now available to support node affinity in TaskRuns that share workspace volume. By default, this is disabled on OpenShift Pipelines.
- The PodTemplate has been updated to specify **imagePullSecrets** to identify secrets that the container runtime should use to authorize container image pulls when starting a pod.
- Support for emitting warning events from the TaskRun controller if the controller fails to update the TaskRun.
- Standard or recommended k8s labels have been added to all resources to identify resources belonging to an application or component.
- The Entrypoint process is now notified for signals and these signals are then propagated using a dedicated PID Group of the Entrypoint process.
- The PodTemplate can now be set on a Task level at runtime using **TaskRunSpecs**.
- Support for emitting Kubernetes events:
    - The controller now emits events for additional TaskRun lifecycle events - **taskrun started** and **taskrun running**.
    - The PipelineRun controller now emits an event every time a Pipeline starts.

- In addition to the default Kubernetes events, support for CloudEvents for TaskRuns is now available. The controller can be configured to send any TaskRun events, such as create, started, and failed, as cloud events.
- Support for using the **$context.<task|taskRun|pipeline|pipelineRun>.name** variable to reference the appropriate name when in PipelineRuns and TaskRuns.
- Validation for PipelineRun parameters is now available to ensure that all the parameters required by the Pipeline are provided by the PipelineRun. This also allows PipelineRuns to provide extra parameters in addition to the required parameters.
- You can now specify Tasks within a Pipeline that will always execute before the pipeline exits, either after finishing all tasks successfully or after a Task in the Pipeline failed, using the **finally** field in the Pipeline YAML file.
- The **git-clone** ClusterTask is now available.

6.2.1.2. Pipelines CLI

- Support for embedded Trigger binding is now available to the **tkn evenlistener describe** command.
- Support to recommend subcommands and make suggestions if an incorrect subcommand is used.
- The **tkn task describe** command now auto selects the task if only one task is present in the Pipeline.
- You can now start a Task using default parameter values by specifying the **--use-param-defaults** flag in the **tkn task start** command.
- You can now specify a volumeClaimTemplate for PipelineRuns or TaskRuns using the **--workspace** option with the **tkn pipeline start** or **tkn task start** commands.
- The **tkn pipelinerun logs** command now displays logs for the final tasks listed in the **finally** section.
- Interactive mode support has now been provided to the **tkn task start** command and the **describe** subcommand for the following tkn resources: **pipeline**, **pipelinerun**, **task**, **taskrun**, **clustertask**, and **pipelineresource**.
- The **tkn version** command now displays the version of the Triggers installed in the cluster.
- The **tkn pipeline describe** command now displays parameter values and timeouts specified for Tasks used in the Pipeline.
- Support added for the **--last** option for the **tkn pipelinerun describe** and the **tkn taskrun describe** commands to describe the most recent PipelineRun or TaskRun, respectively.
- The **tkn pipeline describe** command now displays the conditions applicable to the Tasks in the Pipeline.
- You can now use the **--no-headers** and **--all-namespaces** flags with the **tkn resource list** command.

6.2.1.3. Triggers

- The following Common Expression Language (CEL) functions are now available:
  - **parseURL** to parse and extract portions of a URL
  - **parseJSON** to parse JSON value types embedded in a string in the **payload** field of the **deployment** webhook
- A new interceptor for webhooks from Bitbucket has been added.
- EventListeners now display the **Address URL** and the **Available status** as additional fields when listed with the **kubectl get** command.
- TriggerTemplate params now use the **$(tt.params.<paramName>)** syntax instead of **$(params.<paramName>)** to reduce the confusion between TriggerTemplate and ResourceTemplates params.
- You can now add **tolerations** in the EventListener CRD to ensure that EventListeners are deployed with the same configuration even if all nodes are tainted due to security or management issues.
- You can now add a Readiness Probe for EventListener Deployment at **URL/live**.
- Support for embedding TriggerBinding specifications in EventListener Triggers.
- Trigger resources are now annotated with the recommended **app.kubernetes.io** labels.

6.2.2. Deprecated features

The following items are deprecated in this release:

- The **--namespace** or **-n** flags for all cluster-wide commands, including the **clustertask** and **clustertriggerbinding** commands, are deprecated. It will be removed in a future release.
- The **name** field in **triggers.bindings** within an EventListener has been deprecated in favor of the **ref** field and will be removed in a future release.
- Variable interpolation in TriggerTemplates using **$(params)** has been deprecated in favor of using **$(tt.params)** to reduce confusion with the Pipeline variable interpolation syntax. The **$(params.<paramName>)** syntax will be removed in a future release.
- The **tekton.dev/task** label is deprecated on ClusterTasks.
- The **TaskRun.Status.ResourceResults.ResourceRef** field is deprecated and will be removed.
- The **tkn pipeline create**, **tkn task create**, and **tkn resource create -f** subcommands have been removed.
- Namespace validation has been removed from **tkn** commands.
- The default timeout of **1h** and the **-t** flag for the **tkn ct start** command have been removed.
- The **s2i** ClusterTask has been deprecated.

6.2.3. Known issues

- Conditions do not support Workspaces.

- The **--workspace** option and the interactive mode is not supported for the **tkn clustertask start** command.
- Support of backward compatibility for **$(params.<paramName>)** forces you to use TriggerTemplates with pipeline specific params as the Triggers webhook is unable to differentiate Trigger params from pipelines params.
- Pipeline metrics report incorrect values when you run a promQL query for **tekton_taskrun_count** and **tekton_taskrun_duration_seconds_count**.
- PipelineRuns and TaskRuns continue to be in the **Running** and **Running(Pending)** states respectively even when a non existing PVC name is given to a Workspace.

6.2.4. Fixed issues

- Previously, the **tkn task delete <name> --trs** command would delete both the Task and ClusterTask if the name of the Task and ClusterTask were the same. With this fix, the command deletes only the TaskRuns that are created by the Task **<name>**.
- Previously the **tkn pr delete -p <name> --keep 2** command would disregard the **-p** flag when used with the **--keep** flag and would delete all the PipelineRuns except the latest two. With this fix, the command deletes only the PipelineRuns that are created by the Pipeline **<name>**, except for the latest two.
- The **tkn triggertemplate describe** output now displays ResourceTemplates in a table format instead of YAML format.
- Previously the **buildah** ClusterTask failed when a new user was added to a container. With this fix, the issue has been resolved.

6.3. RELEASE NOTES FOR RED HAT RED HAT OPENSHIFT PIPELINES TECHNOLOGY PREVIEW 1.0

6.3.1. New features

Red Hat OpenShift Pipelines Technology Preview (TP) 1.0 is now available on OpenShift Container Platform 4.5. Red Hat OpenShift Pipelines TP 1.0 is updated to support:

- Tekton Pipelines 0.11.3
- Tekton **tkn** CLI 0.9.0
- Tekton Triggers 0.4.0
- ClusterTasks based on Tekton Catalog 0.11

In addition to the fixes and stability improvements, here is a highlight of what's new in OpenShift Pipelines 1.0.

6.3.1.1. Pipelines

- Support for v1beta1 API Version.

- Support for an improved LimitRange. Previously, LimitRange was specified exclusively for the TaskRun and the PipelineRun. Now there is no need to explicitly specify the LimitRange. The minimum LimitRange across the namespace is used.
- Support for sharing data between Tasks using TaskResults and TaskParams.
- Pipelines can now be configured to not overwrite the **HOME** environment variable and **workingDir** of Steps.
- Similar to Task Steps, **sidecars** now support script mode.
- You can now specify a different scheduler name in TaskRun **podTemplate**.
- Support for variable substitution using Star Array Notation.
- Tekton Controller can now be configured to monitor an individual namespace.
- A new description field is now added to the specification of Pipeline, Task, ClusterTask, Resource, and Condition.
- Addition of proxy parameters to Git PipelineResources.

6.3.1.2. Pipelines CLI

- The **describe** subcommand is now added for the following **tkn** resources: **eventlistener**, **condition**, **triggertemplate**, **clustertask**, and **triggerbinding**.
- Support added for **v1beta1** to the following commands along with backward comptibility for **v1alpha1**: **clustertask**, **task**, **pipeline**, **pipelinerun**, and **taskrun**.
- The following commands can now list output from all namespaces using the **--all-namespaces** flag option:
  - **tkn task list**
  - **tkn pipeline list**
  - **tkn taskrun list**
  - **tkn pipelinerun list**
    The output of these commands is also enhanced to display information without headers using the **--no-headers** flag option.
- You can now start a Pipeline using default parameter values by specifying **--use-param-defaults** flag in the **tkn pipelines start** command.
- Support for Workspace is now added to **tkn pipeline start** and **tkn task start** commands.
- A new **clustertriggerbinding** command is now added with the following subcommands: **describe**, **delete**, and **list**.
- You can now directly start a pipeline run using a local or remote **yaml** file.
- The **describe** subcommand now displays an enhanced and detailed output. With the addition of new fields, such as **description**, **timeout**, **param description**, and **sidecar status**, the command output now provides more detailed information about a specific **tkn** resource.

- The **tkn task log** command now displays logs directly if only one task is present in the namespace.

6.3.1.3. Triggers

- Triggers can now create both **v1alpha1** and **v1beta1** Pipeline resources.
- Support for new Common Expression Language (CEL) interceptor function - **compareSecret**. This function securely compares strings to secrets in CEL expressions.
- Support for authentication and authorization at the EventListener Trigger level.

6.3.2. Deprecated features

The following items are deprecated in this release:

- The environment variable **$HOME**, and variable **workingDir** in the Steps specification are deprecated and might be changed in a future release. Currently in a Step container, **HOME** and **workingDir** are overwritten to **/tekton/home** and **/workspace** respectively.
  In a later release, these two fields will not be modified, and will be set to values defined in the container image and Task YAML. For this release, use flags **disable-home-env-overwrite** and **disable-working-directory-overwrite** to disable overwriting of the **HOME** and **workingDir** variables.
- The following commands are deprecated and might be removed in the future release:
  - **tkn pipeline create**
  - **tkn task create**
- The **-f** flag with the **tkn resource create** command is now deprecated. It might be removed in the future release.
- The **-t** flag and the **--timeout** flag (with seconds format) for the **tkn clustertask create** command are now deprecated. Only duration timeout format is now supported, for example **1h30s**. These deprecated flags might be removed in the future release.

6.3.3. Known issues

- If you are upgrading from an older version of Red Hat OpenShift Pipelines, you must delete your existing deployments before upgrading to Red Hat OpenShift Pipelines version 1.0. To delete an existing deployment, you must first delete Custom Resources and then uninstall the Red Hat OpenShift Pipelines Operator. For more details, see the uninstalling Red Hat OpenShift Pipelines section.
- Submitting the same **v1alpha1** Tasks more than once results in an error. Use **oc replace** instead of **oc apply** when re-submitting a **v1alpha1** Task.
- The **buildah** ClusterTask does not work when a new user is added to a container.
  When the Operator is installed, the **--storage-driver** flag for the **buildah** ClusterTask is not specified, therefore the flag is set to its default value. In some cases, this causes the storage driver to be set incorrectly. When a new user is added, the incorrect storage-driver results in the failure of the **buildah** ClusterTask with the following error:

```
useradd: /etc/passwd.8: lock file already used
useradd: cannot lock /etc/passwd; try again later.
```

-

As a workaround, manually set the **--storage-driver** flag value to **overlay** in the **buildah-task.yaml** file:

1. Login to your cluster as a **cluster-admin**:

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

2. Use the **oc edit** command to edit **buildah** ClusterTask:

```
$ oc edit clustertask buildah
```

The current version of the **buildah** clustertask YAML file opens in the editor set by your **EDITOR** environment variable.

3. Under the **steps** field, locate the following **command** field:

```
command: ['buildah', 'bud', '--format=$(params.FORMAT)', '--tls-verify=$(params.TLSVERIFY)', '--layers', '-f', '$(params.DOCKERFILE)', '-t', '$(resources.outputs.image.url)', '$(params.CONTEXT)']
```

4. Replace the **command** field with the following:

```
command: ['buildah', '--storage-driver=overlay', 'bud', '--format=$(params.FORMAT)', '--tls-verify=$(params.TLSVERIFY)', '--no-cache', '-f', '$(params.DOCKERFILE)', '-t', '$(params.IMAGE)', '$(params.CONTEXT)']
```

5. Save the file and exit.

Alternatively, you can also modify the **buildah** ClusterTask YAML file directly on the web console by navigating to **Pipelines → Cluster Tasks → buildah**. Select **Edit Cluster Task** from the **Actions** menu and replace the **command** field as shown in the previous procedure.

6.3.4. Fixed issues

- Previously, the **DeploymentConfig** Task triggered a new deployment build even when an image build was already in progress. This caused the deployment of the Pipeline to fail. With this fix, the **deploy task** command is now replaced with the **oc rollout status** command which waits for the in-progress deployment to finish.
- Support for **APP_NAME** parameter is now added in Pipeline templates.
- Previously, the Pipeline template for Java S2I failed to look up the image in the registry. With this fix, the image is looked up using the existing image PipelineResources instead of the user provided **IMAGE_NAME** parameter.
- All the OpenShift Pipelines images are now based on the Red Hat Universal Base Images (UBI).
- Previously, when the Pipeline was installed in a namespace other than **tekton-pipelines**, the **tkn version** command displayed the Pipeline version as **unknown**. With this fix, the **tkn version** command now displays the correct Pipeline version in any namespace.
- The **-c** flag is no longer supported for the **tkn version** command.
- Non-admin users can now list the ClusterTriggerBindings.

- The EventListener CompareSecret function is now fixed for the CEL Interceptor.
- The **list**, **describe**, and **start** subcommands for **task** and **clustertask** now correctly display the output in case a Task and ClusterTask have the same name.
- Previously, the OpenShift Pipelines Operator modified the privileged security context constraints (SCCs), which caused an error during cluster upgrade. This error is now fixed.
- In the **tekton-pipelines** namespace, the timeouts of all TaskRuns and PipelineRuns are now set to the value of **default-timeout-minutes** field using the ConfigMap.
- Previously, the Pipelines section in the web console was not displayed for non-admin users. This issue is now resolved.