OpenShift Container Platform 4.4

Pipelines

Configuring and using Pipelines in OpenShift Container Platform

Last Updated: 2021-02-14

OpenShift Container Platform 4.4 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 3
1.1. KEY FEATURES 3
1.2. OPENSHIFT PIPELINES CONCEPTS 3

CHAPTER 2. INSTALLING OPENSHIFT PIPELINES 5
Prerequisites 5
2.1. INSTALLING THE OPENSHIFT PIPELINES OPERATOR IN WEB CONSOLE 5
2.2. INSTALLING THE OPENSHIFT PIPELINES OPERATOR USING THE CLI 6

CHAPTER 3. UNINSTALLING OPENSHIFT PIPELINES 7
3.1. DELETING THE OPENSHIFT PIPELINES COMPONENTS AND CUSTOM RESOURCES 7
3.2. UNINSTALLING THE OPENSHIFT PIPELINES OPERATOR 7

CHAPTER 4. CREATING APPLICATIONS WITH OPENSHIFT PIPELINES 8
Prerequisites 8
4.1. CREATING A PROJECT AND CHECKING YOUR PIPELINE SERVICEACCOUNT 8
4.2. ABOUT TASKS 9
4.3. CREATING PIPELINE TASKS 9
4.4. DEFINING AND CREATING PIPELINERESOURCES 11
4.5. ASSEMBLING A PIPELINE 12
4.6. RUNNING A PIPELINE 14
4.7. ABOUT TRIGGERS 15
4.8. ADDING TRIGGERS TO A PIPELINE 16
4.9. CREATING WEBHOOKS 18
4.10. TRIGGERING A PIPELINERUN 19

CHAPTER 5. WORKING WITH OPENSHIFT PIPELINES USING THE DEVELOPER PERSPECTIVE 20
5.1. INTERACTING WITH PIPELINES USING THE DEVELOPER PERSPECTIVE 20

CHAPTER 6. OPENSHIFT PIPELINES RELEASE NOTES 21
6.1. GETTING SUPPORT 21
6.2. RELEASE NOTES FOR RED HAT OPENSHIFT PIPELINES TECHNOLOGY PREVIEW 1.0 21
6.2.1. New features 21
6.2.1.1. Pipelines 21
6.2.1.2. Pipelines CLI 22
6.2.1.3. Triggers 22
6.2.2. Deprecated features 23
6.2.3. Known issues 23
6.2.4. Fixed issues 24

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/.

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

- OpenShift Pipelines is a serverless CI/CD system that runs Pipelines with all the required dependencies in isolated containers.
- OpenShift Pipelines are designed for decentralized teams that work on microservice-based architecture.
- 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 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. OPENSHIFT PIPELINES CONCEPTS

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.

PipelineResource

A PipelineResource is an object that is used as an input and output for Pipeline Tasks. For example, if an input is a Git repository and an output is a container image built from that Git repository, these are both classified as PipelineResources. PipelineResources currently support Git resources, Image resources, Cluster resources, Storage Resources and CloudEvent resources.

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.

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 OPENSHIFT PIPELINES OPERATOR IN WEB CONSOLE

You can install OpenShift Pipelines using the Operator listed in the OpenShift Container Platform OperatorHub. When you install the 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 **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 **OpenShift Pipelines Operator** page. Click **Install**.
4. On the **Create Operator Subscription** 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 OpenShift Pipelines Operator.
      - The **preview** channel enables installation of the latest preview version of the OpenShift Pipelines Operator, which may contain features that are not yet available from the 4.x update channel.
5. Click **Subscribe**. 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 OpenShift Pipelines Operator.

2.2. INSTALLING THE OPENSHIFT PIPELINES OPERATOR USING THE CLI

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

Procedure

1. Create a Subscription object YAML file to subscribe a namespace to the 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 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 OpenShift Pipelines Operator is a two-step process:

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

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

3.1. DELETING THE OPENSHIFT PIPELINES COMPONENTS AND CUSTOM RESOURCES

Delete the Custom Resources (CRs) created by default during installation of the 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 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 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 OPENSHIFT PIPELINES OPERATOR

Procedure

1. From the **Operators → OperatorHub** page, use the **Filter by keyword** box to search for **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 APPLICATIONS WITH OPENSHIFT PIPELINES

With 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 a Pipeline and PipelineResources to define the delivery Pipeline for your application.
- 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 **ui-repo** Git repository.
- A back-end interface **vote-api**, with the source code in **api-repo** Git repository.
- **apply_manifest** and **update-deployment** Tasks in **pipelines-tutorial** Git repository.

Prerequisites

- You have access to an OpenShift Container Platform cluster.
- You have installed OpenShift Pipelines using the 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.
- You have Administrator access to your repositories.

4.1. 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:
   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
   ```

4.2. ABOUT TASKS

Tasks are the building blocks of a Pipeline and consist of sequentially executed Steps. 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.

A Task uses **inputs** parameters, such as a Git resource, and **outputs** parameters, such as an image in a registry, to interact with other Tasks. They are reusable and can be used in multiple Pipelines.

Here is an example of a Maven Task with a single Step to build a Maven-based Java application.

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: maven-build
spec:
  resources:
    inputs:
    - name: workspace-git
      targetPath: /
      type: git
  steps:
  - name: build
    image: maven:3.6.0-jdk-8-slim
    command:
    - /usr/bin/mvn
    args:
    - install
```

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.

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-1/01_pipeline/01_apply_manifest_task.yaml
   $ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/release-tech-preview-1/01_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
   buildah-v0-11-3                          1 day ago
   git-clone                            1 day ago
   jib-maven                             1 day ago
   kn                                 1 day ago
   maven                               1 day ago
   openshift-client                        1 day ago
   openshift-client-v0-11-3                    1 day ago
   s2i                                1 day ago
   s2i-dotnet-3                           1 day ago
   s2i-dotnet-3-v0-11-3                       1 day ago
   s2i-go                              1 day ago
   s2i-go-v0-11-3                          1 day ago
   s2i-java-11                            1 day ago
   s2i-java-11-v0-11-3                        1 day ago
   s2i-java-8                             1 day ago
   s2i-java-8-v0-11-3                        1 day ago
   s2i-nodejs                             1 day ago
   s2i-nodejs-v0-11-3                        1 day ago
   s2i-perl                              1 day ago
   s2i-perl-v0-11-3                          1 day ago
   ```

```
s2i-php                    1 day ago
s2i-php-v0-11-3            1 day ago
s2i-python-3               1 day ago
s2i-python-3-v0-11-3       1 day ago
s2i-ruby                   1 day ago
s2i-ruby-v0-11-3           1 day ago
s2i-v0-11-3                1 day ago
tkn                        1 day ago
```

4.4. DEFINING AND CREATING PIPELINERESOURCES

PipelineResources are artifacts that are used as inputs or outputs of a Task.

After you create Tasks, create PipelineResources that contain the specifics of the Git repository and the image registry to be used in the Pipeline during execution:

NOTE

If you are not in the **pipelines-tutorial** namespace, and are using another namespace, ensure you update the front-end and back-end image resource to the correct URL with your namespace in the steps below. For example:

```
image-registry.openshift-image-registry.svc:5000/<namespace-name>/vote-api:latest
```

Procedure

1. Create a PipelineResource that defines the Git repository for the front-end application:

   ```
   $ tkn resource create
   ? Enter a name for a pipeline resource : ui-repo
   ? Select a resource type to create : git
   ? Enter a value for url : http://github.com/openshift-pipelines/vote-ui.git
   ? Enter a value for revision : release-tech-preview-1
   ```

   The output verifies that the **ui-repo** PipelineResource was created.

   ```
   New git resource "ui-repo" has been created
   ```

2. Create a PipelineResource that defines the OpenShift Container Platform internal image registry to where you want to push the front-end image:

   ```
   $ tkn resource create
   ? Enter a name for a pipeline resource : ui-image
   ? Select a resource type to create : image
   ? Enter a value for url : image-registry.openshift-image-registry.svc:5000/pipelines-
   tutorial/ui:latest
   ? Enter a value for digest :
   ```

   The output verifies that the **ui-image** PipelineResource was created.

   ```
   New image resource "ui-image" has been created
   ```

3. Create a PipelineResource that defines the Git repository for the back-end application:

   ```
   $ tkn resource create
   ? Enter a name for a pipeline resource : api-repo
   ? Select a resource type to create : git
   ? Enter a value for url : http://github.com/openshift-pipelines/vote-api.git
   ? Enter a value for revision : release-tech-preview-1
   ```

   The output verifies that the **api-repo** PipelineResource was created.

   ```
   New git resource "api-repo" has been created
   ```

4. Create a PipelineResource that defines the OpenShift Container Platform internal image registry to where you want to push the back-end image:

   ```
   $ tkn resource create
   ? Enter a name for a pipeline resource : api-image
   ? Select a resource type to create : image
   ? Enter a value for url : image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/api:latest
   ? Enter a value for digest :
   ```

   The output verifies that the **api-image** PipelineResource was created.

   ```
   New image resource "api-image" has been created
   ```

5. View the list of **resources** created:

   ```
   $ tkn resource list
   ```

   The output lists all the PipelineResource that were created.

   ```
   NAME        TYPE    DETAILS
   api-repo    git     url: http://github.com/openshift-pipelines/vote-api.git
   ui-repo     git     url: http://github.com/openshift-pipelines/vote-ui.git
   api-image   image   url: image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/api:latest
   ui-image    image   url: image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/ui:latest
   ```

4.5. ASSEMBLING A PIPELINE

A Pipeline represents a CI/CD flow and is defined by the Tasks to be executed. It specifies how the Tasks interact with each other and their order of execution using the **inputs** , **outputs**, and **runAfter** parameters. It is designed to be generic and reusable in multiple applications and environments.

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 repositories **api-repo** and **ui-repo**.

- Builds the container images **api-image** and **ui-image** using the **buildah** ClusterTask.
- Pushes the **api-image** and **ui-image** images to the internal image registry.
- Deploys the new images on OpenShift Container Platform 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:
  resources:
  - name: git-repo
    type: git
  - name: image
    type: image
  params:
  - name: deployment-name
    type: string
    description: name of the deployment to be patched
  tasks:
  - name: build-image
    taskRef:
      name: buildah
      kind: ClusterTask
    resources:
      inputs:
      - name: source
        resource: git-repo
      outputs:
      - name: image
        resource: image
    params:
    - name: TLSVERIFY
      value: "false"
  - name: apply-manifests
    taskRef:
      name: apply-manifests
    resources:
      inputs:
      - name: source
        resource: git-repo
    runAfter:
    - build-image
  - name: update-deployment
    taskRef:
      name: update-deployment
    resources:
      inputs:
      - name: image
        resource: image
    params:
    - name: deployment
      value: $(params.deployment-name)
    runAfter:
    - apply-manifests
```

Notice that the Pipeline definition abstracts away the specifics of the Git source repository and image registries to be used during the Pipeline execution.

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-1/01_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   ---        ---       ---        ---
```

4.6. 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 -r git-repo=api-repo -r image=api-image -p deployment-name=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 -r git-repo=ui-repo -r image=ui-image -p deployment-name=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. ABOUT 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 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, while then performing 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.

4.8. ADDING TRIGGERS TO A PIPELINE

After you have assembled and started the Pipeline for the application, add TriggerBindings, TriggerTemplates, and an EventListener to capture 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-1/03_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: master
  - name: git-repo-name
    description: The name of the deployment to be created / patched

  resourcetemplates:
  - apiVersion: tekton.dev/v1alpha1
  kind: PipelineResource
  metadata:
    name: $(params.git-repo-name)-git-repo-$(uid)
  spec:
    type: git
    params:
    - name: revision
      value: $(params.git-revision)
    - name: url
      value: $(params.git-repo-url)

- apiVersion: tekton.dev/v1alpha1
  kind: PipelineResource
  metadata:
    name: $(params.git-repo-name)-image-$(uid)
  spec:
    type: image
    params:
    - name: url
      value: image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/$(params.git-repo-name):latest

- apiVersion: tekton.dev/v1beta1
  kind: PipelineRun
  metadata:
    name: build-deploy-$(params.git-repo-name)-$(uid)
  spec:
    serviceAccountName: pipeline
    pipelineRef:
      name: build-and-deploy
    resources:
    - name: git-repo
      resourceRef:
        name: $(params.git-repo-name)-git-repo-$(uid)
    - name: image
      resourceRef:
        name: $(params.git-repo-name)-image-$(uid)
    params:
    - name: deployment-name
      value: $(params.git-repo-name)
```

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-1/03_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-1/03_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
   ```

4.9. 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**.

4.10. 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 will push an empty commit to the front-end **vote-api** repository, which will trigger the PipelineRun.

Procedure

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

```
$ git clone git@github.com:<your GitHub ID>/vote-api.git -b release-tech-preview-1
```

2. Push an empty commit:

```
$ git commit -m "empty-commit" --allow-empty && git push origin release-tech-preview-1
```

3. Check if the PipelineRun was triggered:

```
$ tkn pipelinerun list
```

   Notice that a new PipelineRun was initiated.

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 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 while creating an application on OpenShift Container Platform.

After you create Pipelines, you can view and visually interact with your deployed Pipelines.

<Discrete><title>Prerequisites</title>

- You have access to an OpenShift Container Platform cluster and have logged in to the web console.
- You have cluster administrator privileges to install Operators and have installed the OpenShift Pipelines Operator.
- You are in the Developer perspective.
- You have created a project.

</Discrete>

5.1. 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.
2. Click on the required Pipeline to see the **Pipeline Details** page. This 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 **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.

CHAPTER 6. OPENSHIFT PIPELINES RELEASE NOTES

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 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 OPENSHIFT PIPELINES TECHNOLOGY PREVIEW 1.0

6.2.1. New features

OpenShift Pipelines Technology Preview (TP) 1.0 is now available on OpenShift Container Platform 4.4. 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.

6.2.1.1. 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.

6.2.1.2. 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.

6.2.1.3. 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.

6.2.2. 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.

6.2.3. Known issues

- If you are upgrading from an older version of OpenShift Pipelines, you must delete your existing deployments before upgrading to OpenShift Pipelines version 1.0. To delete an existing deployment, you must first delete Custom Resources and then uninstall the OpenShift Pipelines Operator. For more details, see the uninstalling 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.

6.2.4. 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.