

Starter Labs (Python)

WORKSHOP MODULES

Workshop Summary

Environment Overview

Using Homeroom

Architecture Overview of the
ParksMap Application

Exploring the CLI and Web Console

Deploying Your First Container Image

Scaling and Self Healing

Exposing Your Application to the
Outside World

Exploring OpenShift's Logging
Capabilities

Role-Based Access Control

Remote Access to Your Application

Deploying Python Code

Adding a Database (MongoDB)

Application Health

**Automate Build and Deployment
with Pipelines**

Automation for Your Application on
Code Changes

Further Resources

Workshop Links

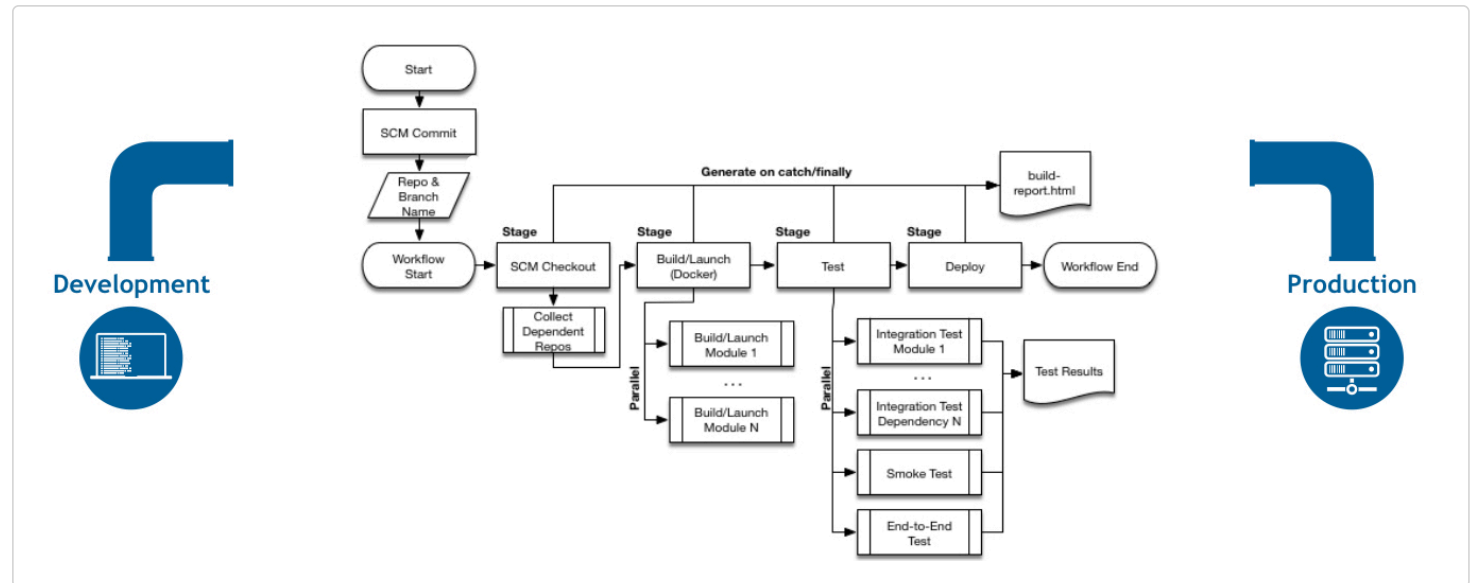
Automate Build and Deployment with Pipelines

In this lab you will learn about pipelines and how to configure a pipeline in OpenShift so that it will take care of the application lifecycle.

Background: Continuous Integration and Pipelines

A continuous delivery (CD) pipeline is an automated expression of your process for getting software from version control right through to your users and customers. Every change to your software (committed in source control) goes through a complex process on its way to being released. This process involves building the software in a reliable and repeatable manner, as well as progressing the built software (called a "build") through multiple stages of testing and deployment.

OpenShift Pipelines is a cloud-native, continuous integration and delivery (CI/CD) solution for building pipelines using [Tekton](#). Tekton is a flexible, Kubernetes-native, open-source CI/CD framework that enables automating deployments across multiple platforms (Kubernetes, serverless, VMs, etc) by abstracting away the underlying details.

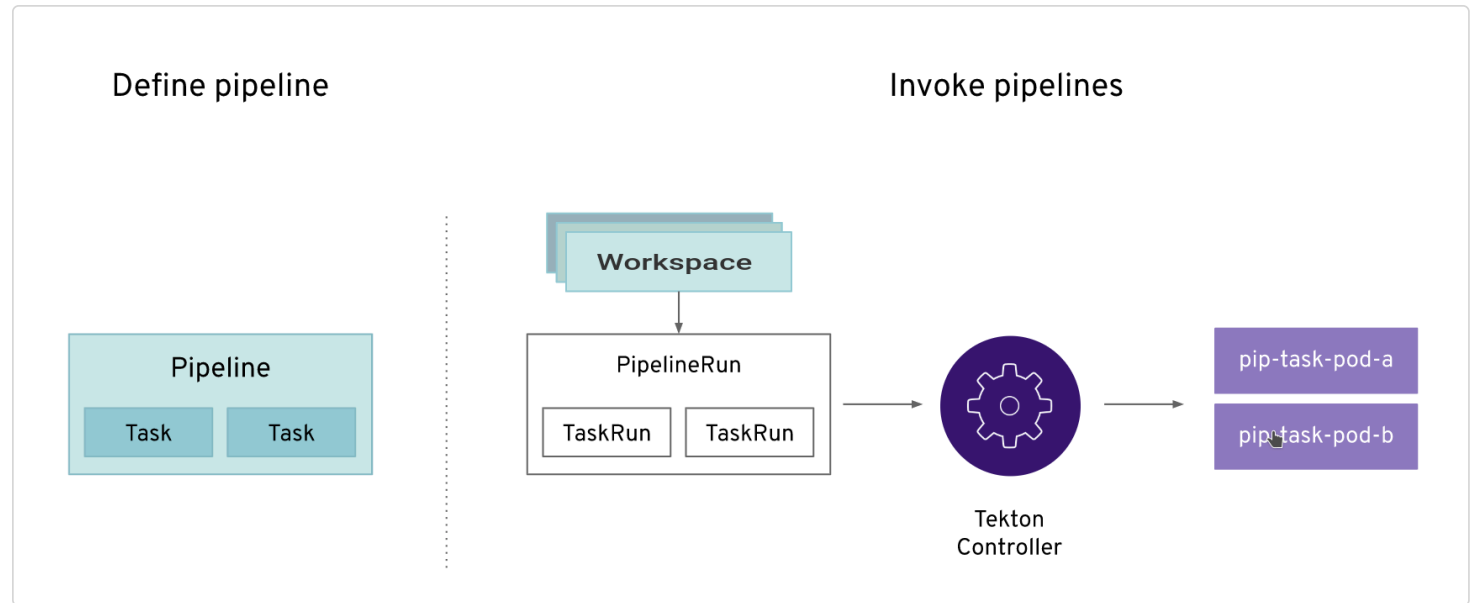


Understanding Tekton

Tekton defines a number of [Kubernetes custom resources](#) as building blocks in order to standardize pipeline concepts and provide a terminology that is consistent across CI/CD solutions.

The custom resources needed to define a pipeline are listed below:

- **Task** : a reusable, loosely coupled number of steps that perform a specific task (e.g. building a container image)
- **Pipeline** : the definition of the pipeline and the **Tasks** that it should perform
- **TaskRun** : the execution and result of running an instance of task
- **PipelineRun** : the execution and result of running an instance of pipeline, which includes a number of **TaskRuns**



In short, in order to create a pipeline, one does the following:

- Create custom or install [existing](#) reusable `Tasks`
- Create a `Pipeline` and `PipelineResources` to define your application's delivery pipeline
- Create a `PersistentVolumeClaim` to provide the volume/filesystem for pipeline execution or provide a `VolumeClaimTemplate` which creates a `PersistentVolumeClaim`
- Create a `PipelineRun` to instantiate and invoke the pipeline

For further details on pipeline concepts, refer to the [Tekton documentation](#) that provides an excellent guide for understanding various parameters and attributes available for defining pipelines.

Explore your Pipeline

As pipelines provide the ability to promote applications between different stages of the delivery cycle, Tekton, which is our Continuous Integration server that will execute our pipelines, will be deployed on a project with a Continuous

Integration role. Pipelines executed in this project will have permissions to interact with all the projects modeling the different stages of our delivery cycle.

For this example, we're going to deploy our pipeline that we created automatically from Developer Console together with `nationalparks` backend.

Verify Tasks already available in the OpenShift cluster (ClusterTasks):

```
oc get clustertasks -n user4
```

You should see something similar:

NAME	AGE
....	
s2i-python-3	4h58m
s2i-python-3-pr	4h58m
....	

Verify the Pipeline we created:

```
oc get pipelines -n user4
```

You should see something like this:

NAME	AGE
nationalparks	8s

Now let's review our Tekton Pipeline:

```
---
apiVersion: tekton.dev/v1beta1
kind: Pipeline
```

```
metadata:
  name: nationalparks
spec:
  params:
    - default: nationalparks
      name: APP_NAME
      type: string
    - default: >-
      https://github.com/openshift-roadshow/nationalparks-py.git
      name: GIT_REPO
      type: string
    - default: master
      name: GIT_REVISION
      type: string
    - default: 'image-registry.openshift-image-registry.svc:5000/user1/nationalparks'
      name: IMAGE_NAME
      type: string
    - default: .
      name: PATH_CONTEXT
      type: string
    - default: '1'
      name: MINOR_VERSION
      type: string
  tasks:
    - name: fetch-repository
      params:
        - name: url
          value: ${params.GIT_REPO}
        - name: revision
          value: ${params.GIT_REVISION}
        - name: subdirectory
          value: ''
        - name: deleteExisting
          value: 'true'
      taskRef:
        kind: ClusterTask
        name: git-clone
      workspaces:
        - name: output
          workspace: workspace
```

```
- name: build
  params:
    - name: IMAGE
      value: $(params.IMAGE_NAME)
    - name: TLSVERIFY
      value: 'false'
    - name: PATH_CONTEXT
      value: $(params.PATH_CONTEXT)
    - name: MINOR_VERSION
      value: $(params.MINOR_VERSION)
  runAfter:
    - fetch-repository
  taskRef:
    kind: ClusterTask
    name: s2i-python
  workspaces:
    - name: source
      workspace: workspace
- name: deploy
  params:
    - name: SCRIPT
      value: oc rollout status deploy/$(params.APP_NAME)
  runAfter:
    - build
  taskRef:
    kind: ClusterTask
    name: openshift-client
  workspaces:
    - name: workspace
```

A **Pipeline** is a user-defined model of a CD pipeline. A Pipeline's code defines your entire build process, which typically includes stages for building an application, testing it and then delivering it.

A **Task** and a **ClusterTask** contain some step to be executed. **ClusterTasks** are available to all user within a cluster where OpenShift Pipelines has been installed, while **Tasks** can be custom.

This pipeline has 3 Tasks defined:

- **fetch-repository:** this is a `ClusterTask` that will clone our source repository for nationalparks and store it to a `Workspace` `app-source` which will use the PVC created for it `app-source-workspace`
- **build:** will build and test our Python application, generate and push a container image automatically with compiled binaries inside OpenShift Container Registry.
- **deploy:** it will deploy the created image on OpenShift using the Deployment named `nationalparks` we created in the previous lab.

From left-side menu, click on **Pipeline**, then click on **nationalparks** to see the pipeline you just created.

The screenshot displays the OpenShift Pipelines console for the 'nationalparks' pipeline. At the top, the project is set to 'user1'. The breadcrumb navigation shows 'Pipelines > Pipeline Details'. The pipeline name 'nationalparks' is prominently displayed with a 'PL' icon and an 'Actions' dropdown menu. Below this, a tabbed interface shows 'Details' as the active tab, with other tabs for 'YAML', 'Pipeline Runs', 'Parameters', and 'Resources'. The 'Pipeline Details' section features a visual pipeline diagram with three steps: 'fetch-repository', 'build', and 'deploy'. To the right of the diagram, the 'Trigger Templates' section lists 'nationalparks' with its URL. The 'Tasks' section lists three tasks: 'git-clone (fetch-repository)', 's2i-dotnet-3 (build)', and 'openshift-client (deploy)'. On the left side, the 'Name' is 'nationalparks', the 'Namespace' is 'user1', and the 'Labels' section contains two labels: 'app.kubernetes.io/instance=nationalparks' and 'pipeline.openshift.io/runtime=dotnet'. The 'Annotations' section shows '0 Annotations'. The 'Created At' timestamp is 'Jan 11, 11:50 pm', and the 'Owner' is listed as 'No owner'.

The Pipeline is parametric, with default value on the one we need to use.

It is using one **Workspace**:

- **app-source**: this need to be linked to a **PersistentVolumeClaim** since will be used to store the code and the compiled binary to be used in different **Tasks**

Exercise: Add Storage for your Pipeline

OpenShift manages Storage with [Persistent Volumes](#) to be attached to Pods running our applications through **Persistent Volume Claim** requests, and it also provides the capability to manage it at ease from Web Console. From **Administrator Perspective**, go to **Storage**→ **Persistent Volume Claims**.

Go to top-right side and click **Create Persistent Volume Claim** button.

Inside **Persistent Volume Claim name** insert **app-source-pvc**.

In **Size** section, insert **1** as we are going to create 1 GiB Persistent Volume for our Pipeline, using RWO Single User access mode.

Leave all default settings, and click **Create**.

The screenshot shows the Red Hat OpenShift Container Platform dashboard. On the left is a dark sidebar with navigation links: Administrator, Home, Projects, Search, API Explorer, Events, Operators, Workloads, Networking, Storage (selected), PersistentVolumeClaims (selected), StorageClasses, VolumeSnapshots, VolumeSnapshotClasses, and Builds. The main content area is titled 'Project: user1' and 'Create PersistentVolumeClaim' with an 'Edit YAML' link. The form includes a 'StorageClass' dropdown set to 'gp2', a 'PersistentVolumeClaim name' field with 'app-source-pvc', 'Access mode' set to 'Single user (RWX)', 'Size' set to '1 GiB', and 'Volume mode' set to 'Filesystem'. There are 'Create' and 'Cancel' buttons at the bottom.

Red Hat OpenShift Container Platform

Project: user1

Create PersistentVolumeClaim [Edit YAML](#)

StorageClass

SC gp2

StorageClass for the new claim

PersistentVolumeClaim name *

app-source-pvc

A unique name for the storage claim within the project

Access mode *

☒ Single user (RWX) ☐ Shared access (RWX) ☐ Read only (ROX)

Access mode is set by StorageClass and cannot be changed

Size *

1 GiB

Desired storage capacity

☐ Use label selectors to request storage

PersistentVolume resources that match all label selectors will be considered for binding.

Volume mode *

☒ Filesystem ☐ Block

Create Cancel

The **Storage Class** is the type of storage available in the cluster.

Run the Pipeline

We can start now the Pipeline from the Web Console. Switch to **Developer Perspective**, and from left-side menu, click on **Pipeline**, then click on **nationalparks**. From top-right **Actions** list, click on **Start**.

The screenshot shows the OpenShift Pipelines web console for a project named 'user1'. The breadcrumb navigation is 'Pipelines > Pipeline Details'. The pipeline name is 'nationalparks' with a 'PL' icon. The 'Details' tab is selected, showing a pipeline diagram with three tasks: 'fetch-repository', 'build', and 'deploy'. Below the diagram, the pipeline's metadata is displayed: Name 'nationalparks', Namespace 'user1' (with 'NS' icon), and a list of labels including 'app.kubernetes.io/instance=nationalparks', 'pipeline.openshift.io/runtime=dotnet', and 'pipeline.openshift.io/type=kubernetes'. An 'Edit' button with a pencil icon is next to the labels. On the right side, the 'Actions' menu is open, showing options like 'Start', 'Start Last Run', 'Add Trigger', 'Edit Labels', 'Edit Annotations', 'Edit Pipeline', and 'Delete Pipeline'. The 'Start' option is highlighted with a red box. A 'Tech Preview' badge is visible in the top right corner of the console.

You will be prompted with parameters to add the Pipeline, showing default ones.

In **Workspaces** → **app-source** select **PVC** from the list, then select **app-source-pvc**. This is the shared volume used by Pipeline Tasks in your Pipeline containing the source code and compiled artifacts.

Click on **Start** to run your Pipeline.

Start Pipeline

Parameters

APP_NAME *

nationalparks

GIT_REPO *

http://gogs-lab.apps.cluster-jazz-fb81.jazz-fb81.example.opentlc.com/user1/nationalparks-dotnet.git

GIT_REVISION *

master

IMAGE_NAME *

image-registry.openshift-image-registry.svc:5000/user1/nationalparks

PATH_CONTEXT *

.

MINOR_VERSION *

1

Workspaces

workspace *

PVC

 app-source-pvc

Cancel

Start

You can follow the Pipeline execution from **Pipeline** section, watching all the steps in progress. Click on **Pipeline Runs** tab to see it running:

The screenshot shows the Red Hat OpenShift Container Platform dashboard. The left sidebar contains navigation links: Developer, +Add, Topology, Monitoring, Builds, Pipelines (highlighted), and More. The main content area is titled 'Project: user1' and shows 'Pipelines > Pipeline Details' for 'nationalparks-pipeline'. The 'Pipeline Runs' tab is selected and highlighted with a red box. Below the tabs, there is a filter bar with '0 Complete', '1 Running', '0 Failed', and '0 Cancelled', along with a 'Select all filters' button and a '1 Item' count. A table displays the pipeline run details:

Name	Namespace	Status	Task Status	Started	Duration
PLR nationalparks-deploy-run-k9q5s	NS user1	Running	<div><div></div></div>	less than a minute ago	44s

The click on the PipelineRun **nationalparks-deploy-run-**

The screenshot displays the Red Hat OpenShift Container Platform dashboard. The left sidebar contains navigation links: Developer, +Add, Topology, Monitoring, Search, Builds, Pipelines, Helm, Project, Config Maps, and Secrets. The main content area shows the 'Pipeline Run Details' for a pipeline named 'nationalparks-1lppvb' in the 'user1' namespace. The pipeline is currently 'Running'. A diagram shows the pipeline steps: 'fetch-reposi...' (completed), 'build' (current step), and 'deploy' (pending). A tooltip for the 'build' step shows its sub-tasks: 'generate' (a few seconds), 'build' (a few seconds), and 'push' (a few seconds). The 'Labels' section lists several labels including 'app.kubernetes.io/instance=nationalparks', 'pipeline.openshift.io/runtime=dotnet', 'pipeline.openshift.io/type=kubernetes', and 'tekton.dev/pipeline=nationalparks'. The 'Status' section indicates the pipeline is 'Running' and was 'Triggered by: opentlc-mgr'.

Project: user1

Pipeline Runs > Pipeline Run Details

PLR nationalparks-1lppvb Running

[Details](#) [YAML](#) [Logs](#)

Pipeline Run Details

fetch-reposi... build deploy

Name
nationalparks-1lppvb

Namespace
NS user1

Labels
[app.kubernetes.io/instance=nationalparks](#) [pipeline.openshift.io/runtime=dotnet](#)
[pipeline.openshift.io/type=kubernetes](#) [tekton.dev/pipeline=nationalparks](#)

Annotations
[1 Annotation](#)

Created At
console-openshift-console.apps.cluster-f260-f260.example.opentlc.com/k8s/ /bu

Status
Running

Pipeline
PL nationalparks

Triggered by:
opentlc-mgr

Then click on the **Task** running to check logs:

The screenshot displays the Red Hat OpenShift Container Platform dashboard. The left sidebar contains navigation links: Developer, +Add, Topology, Monitoring, Search, Builds, Pipelines, Helm, Project, Config Maps, and Secrets. The main content area shows the 'Project: user1' dropdown and a breadcrumb trail 'Pipeline Runs > Pipeline Run Details'. A green status bar indicates 'PLR nationalparks-1lppvb' is 'Succeeded'. Below this, tabs for 'Details', 'YAML', and 'Logs' are visible, with 'Logs' selected. The log content shows the 'build' step, including commands like 'NationalParks -> /opt/app-root/app/' and 'STEP 8: CMD /usr/libexec/s2i/run'. The 'STEP - PUSH' section lists image source signatures and copying operations for blobs and configs, ending with 'Writing manifest to image destination' and 'Storing signatures'. A 'Tech Preview' badge and an 'Actions' dropdown are also present.

Red Hat OpenShift Container Platform

Project: user1

Pipeline Runs > Pipeline Run Details

PLR nationalparks-1lppvb Succeeded

Tech Preview

Actions

Details YAML Logs

Download Download All Task Logs Expand

build

NationalParks -> /opt/app-root/app/
--> 73615ef2e46
STEP 8: CMD /usr/libexec/s2i/run
STEP 9: COMMIT image-registry.openshift-image-registry.svc:5000/user1/nationalparks
--> 8d9020b9f1c
8d9020b9f1cc7cb53436726c661041b8aaaff24d41f036faae3c6d8edb620d1

STEP - PUSH

Getting image source signatures
Copying blob sha256:d79bec8c4b4c0239e40e57289fb9da56871f2e5e075b44de0f82c877d5d9de0c
Copying blob sha256:66c25c72f4e9806db3e2887d99c3f0e346c7e515f2d46998acbef3b08382522c
Copying blob sha256:0d26487f1f69fd2c9f729599da0db1178fc78e4168255d48c3183e6b9d6af261
Copying blob sha256:02746bca954a072e7674a0642c1f5971a72ec3bc51ed79301ec193605f9b2ba7
Copying blob sha256:486ec397f8f1763b8ccec155cfd3c85e289cf334abd2ebce56fd153422ebba4a
Copying blob sha256:ccff252e842bcb8b0521af4d331fe3106f3953fc15a07b25db4099f16d94c763
Copying blob sha256:32304442244faa274c6ef45f78ab0454a65ba12ba87f803a1acf158a81b58f4c
Copying config sha256:8d9020b9f1cc7cb53436726c661041b8aaaff24d41f036faae3c6d8edb620d1
Writing manifest to image destination
Copying config sha256:8d9020b9f1cc7cb53436726c661041b8aaaff24d41f036faae3c6d8edb620d1
Writing manifest to image destination
Storing signatures

Verify PipelineRun has been completed with success:

The screenshot displays the Red Hat OpenShift Container Platform dashboard. The left sidebar contains navigation links: Developer, +Add, Topology, Monitoring, Search, Builds, Pipelines, Helm, Project, Config Maps, and Secrets. The main content area shows the 'Pipeline Run Details' for a pipeline named 'nationalparks-1lppvb' in the 'user1' namespace. The pipeline run is marked as 'Succeeded' and consists of three steps: 'fetch-reposi...', 'build', and 'deploy', all of which are completed. The 'Labels' section lists several key-value pairs related to the pipeline and its execution. The 'Annotations' section shows one annotation. The 'Created At' timestamp is '4 minutes ago'. A 'Tech Preview' badge is visible in the top right corner of the dashboard.

Red Hat OpenShift Container Platform

Project: user1

Pipeline Runs > Pipeline Run Details

PLR nationalparks-1lppvb Succeeded

Tech Preview

Actions

Details YAML Logs

Pipeline Run Details

fetch-reposi... build deploy

Name
nationalparks-1lppvb

Namespace
NS user1

Labels
app.kubernetes.io/instance=nationalparks pipeline.openshift.io/runtime=dotnet
pipeline.openshift.io/type=kubernetes tekton.dev/pipeline=nationalparks

Status
Succeeded

Pipeline
PL nationalparks

Triggered by:
opentic-mgr

Annotations
1 Annotation

Created At
4 minutes ago

[Continue](#)