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

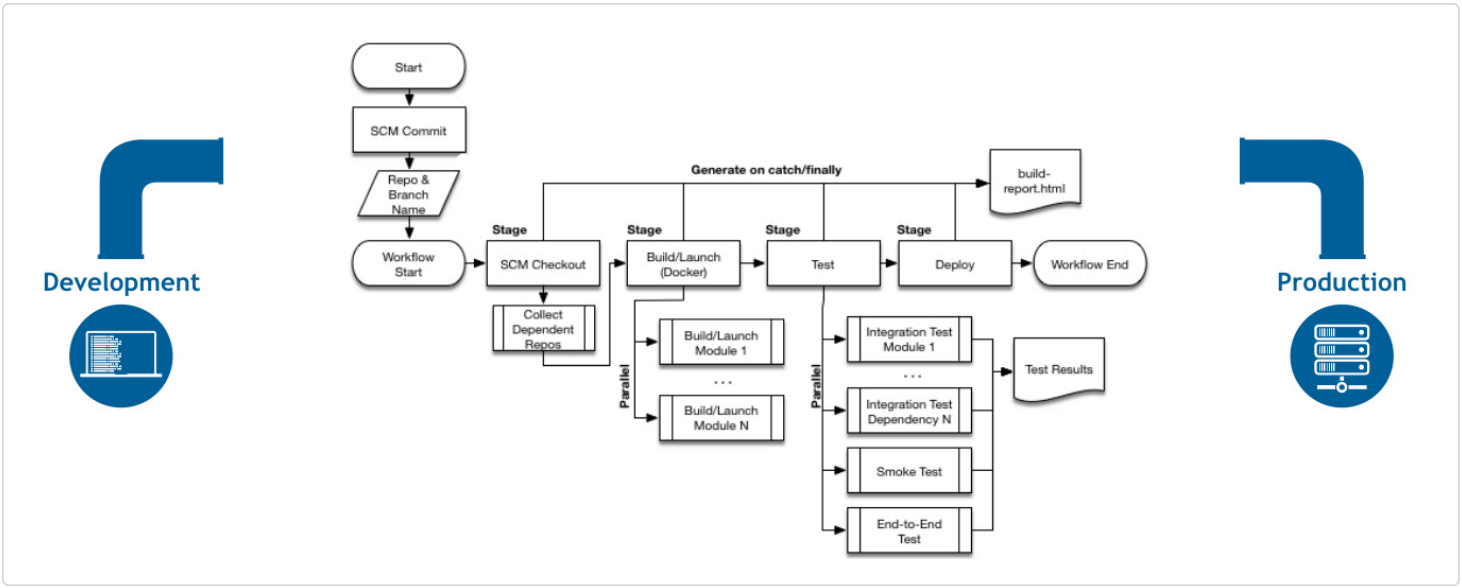
Automate Build and Deployment with Pipelines

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.

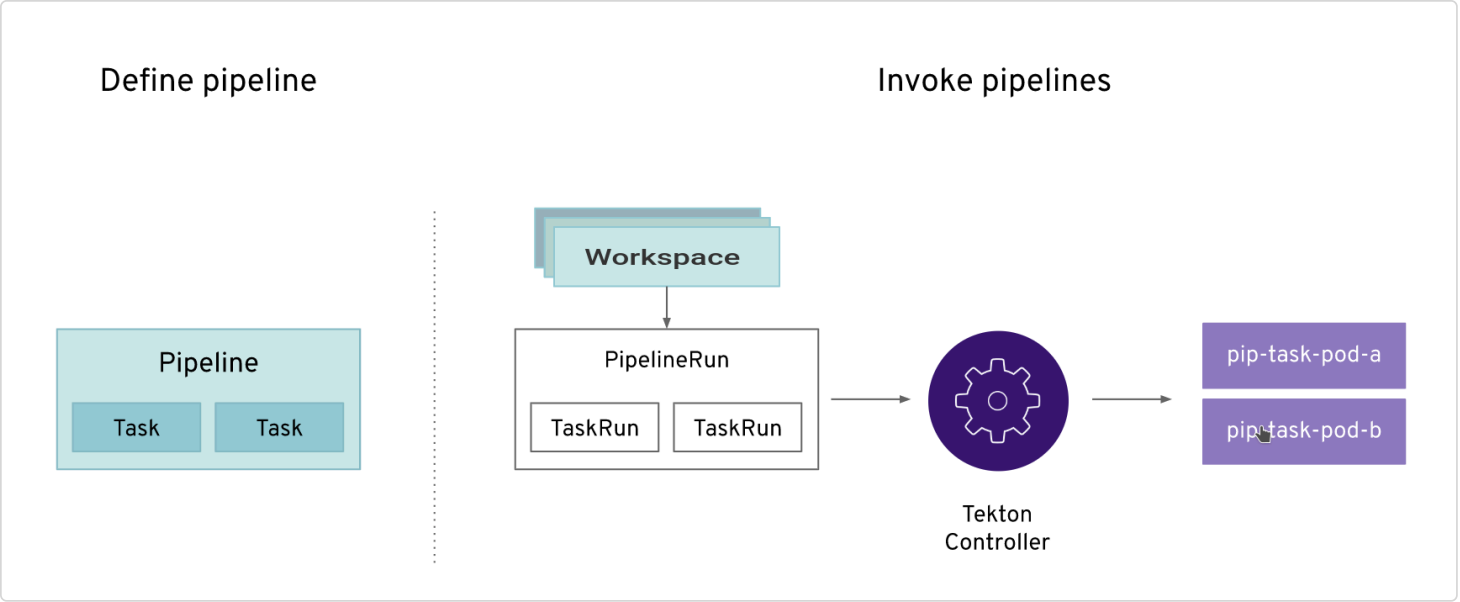


Understanding Tekton

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
```

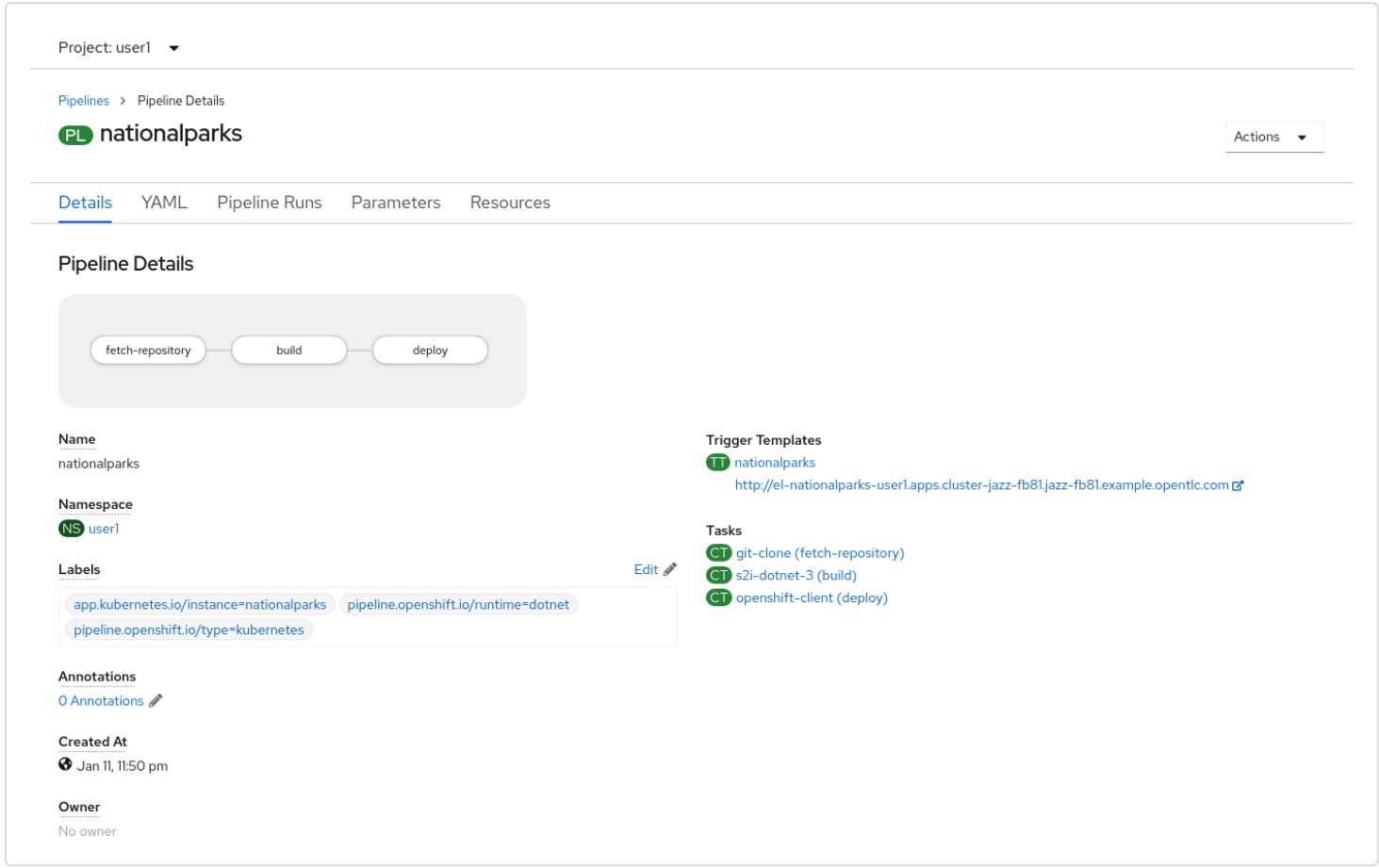
A `Pipeline` is a user-defined model of a CD pipeline. A Pipeline's code defines your entire build process, which typically includes stages for building an application, testing it and then delivering it.

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 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

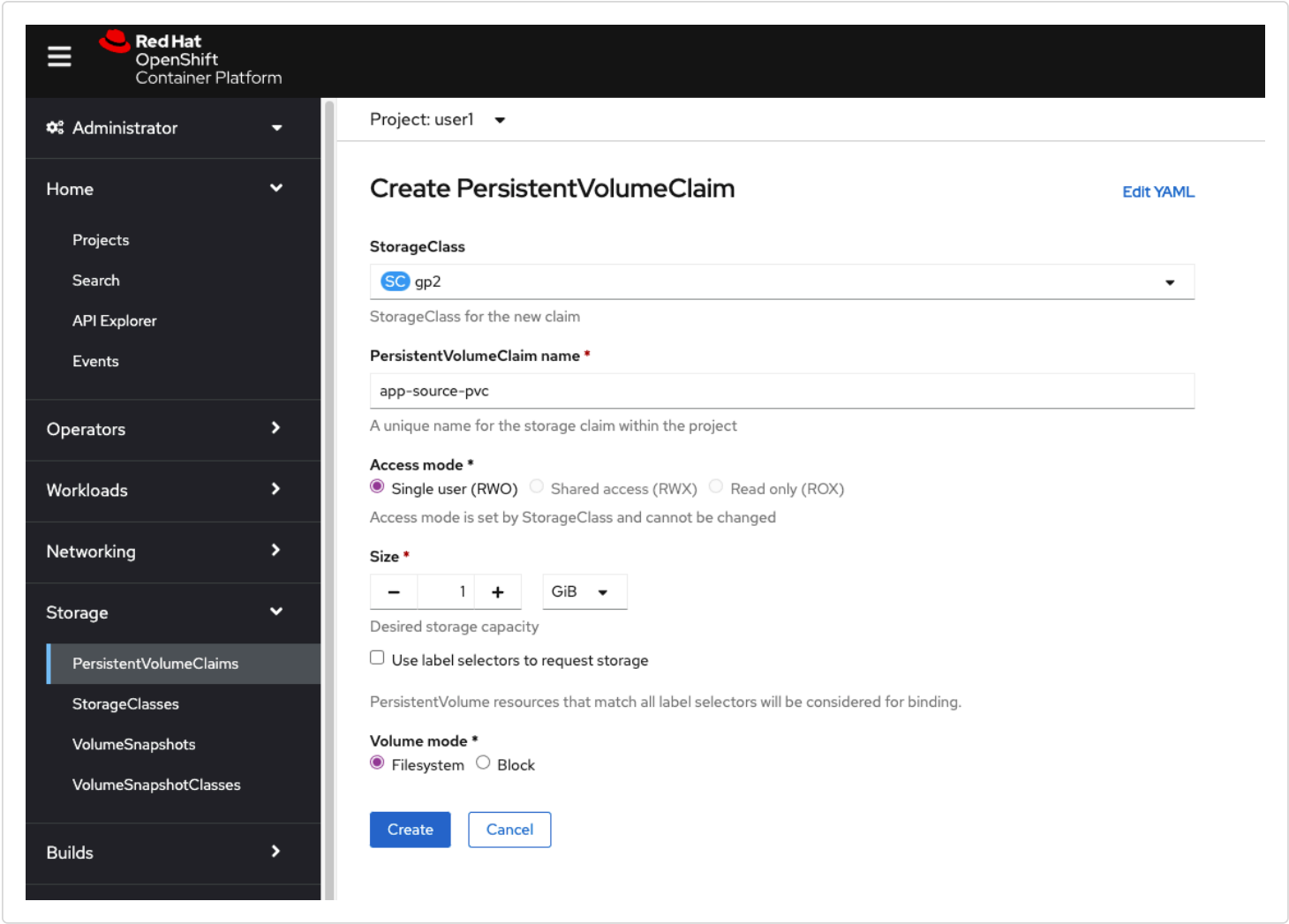
OpenShift manages Storage with Persistent Volumes to be attached to Pods running our applications through **Persistent Volume Claim** requests, and it also provides the capability to manage it at ease from Web Console. From **Administrator Perspective**, go to **Storage→ Persistent Volume Claims**.

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.

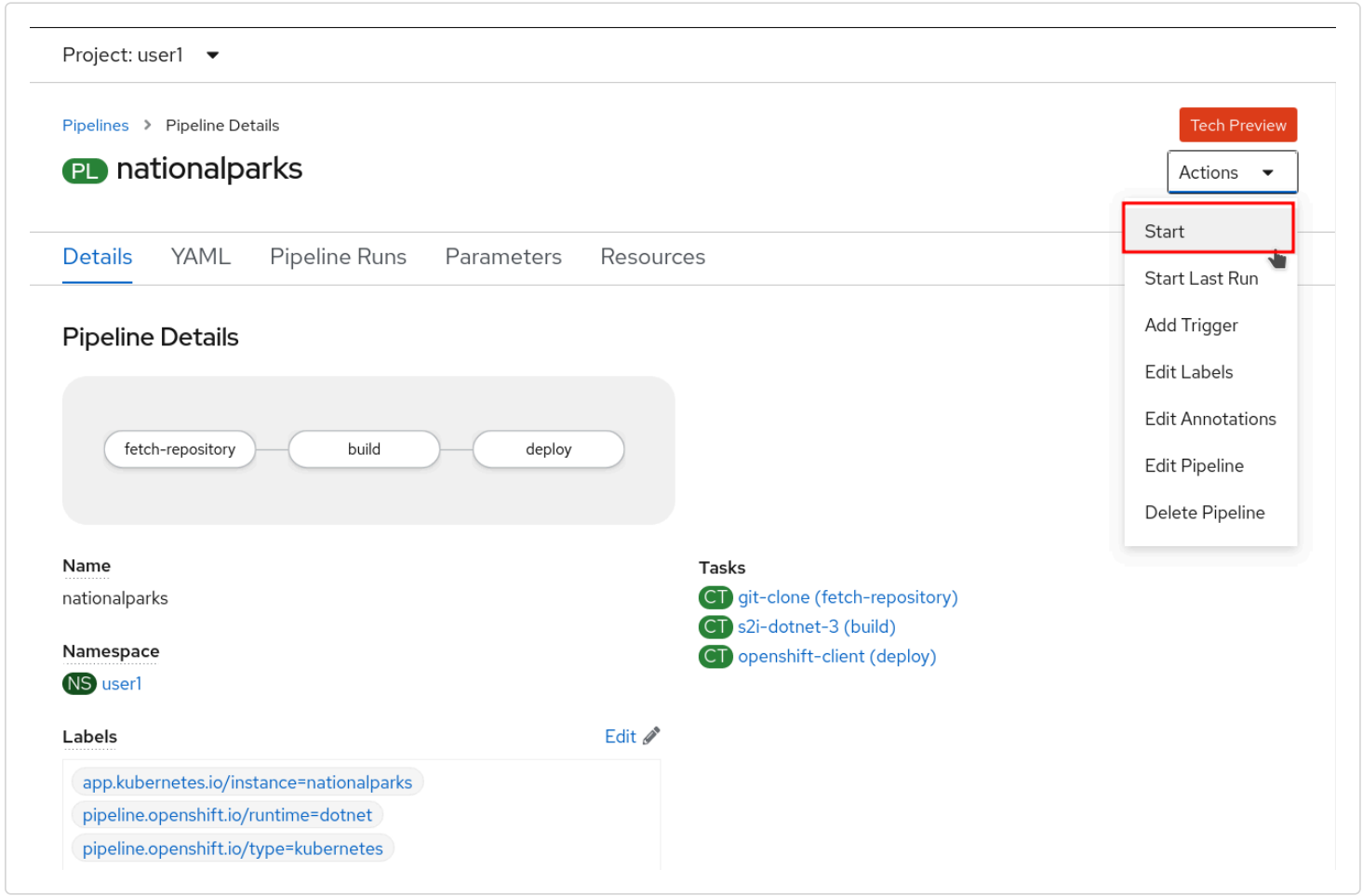
Leave all default settings, and click **Create**.



> The **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**.



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

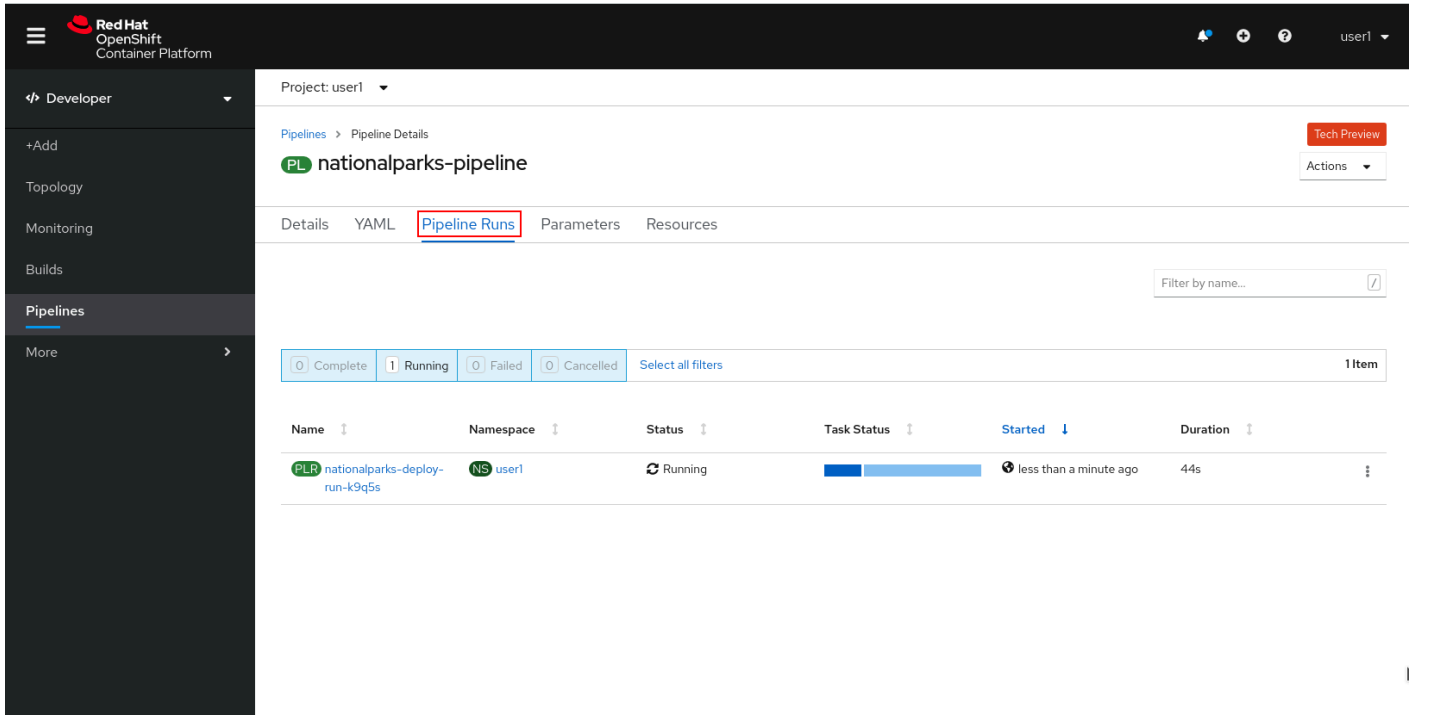
workspace *

PVC

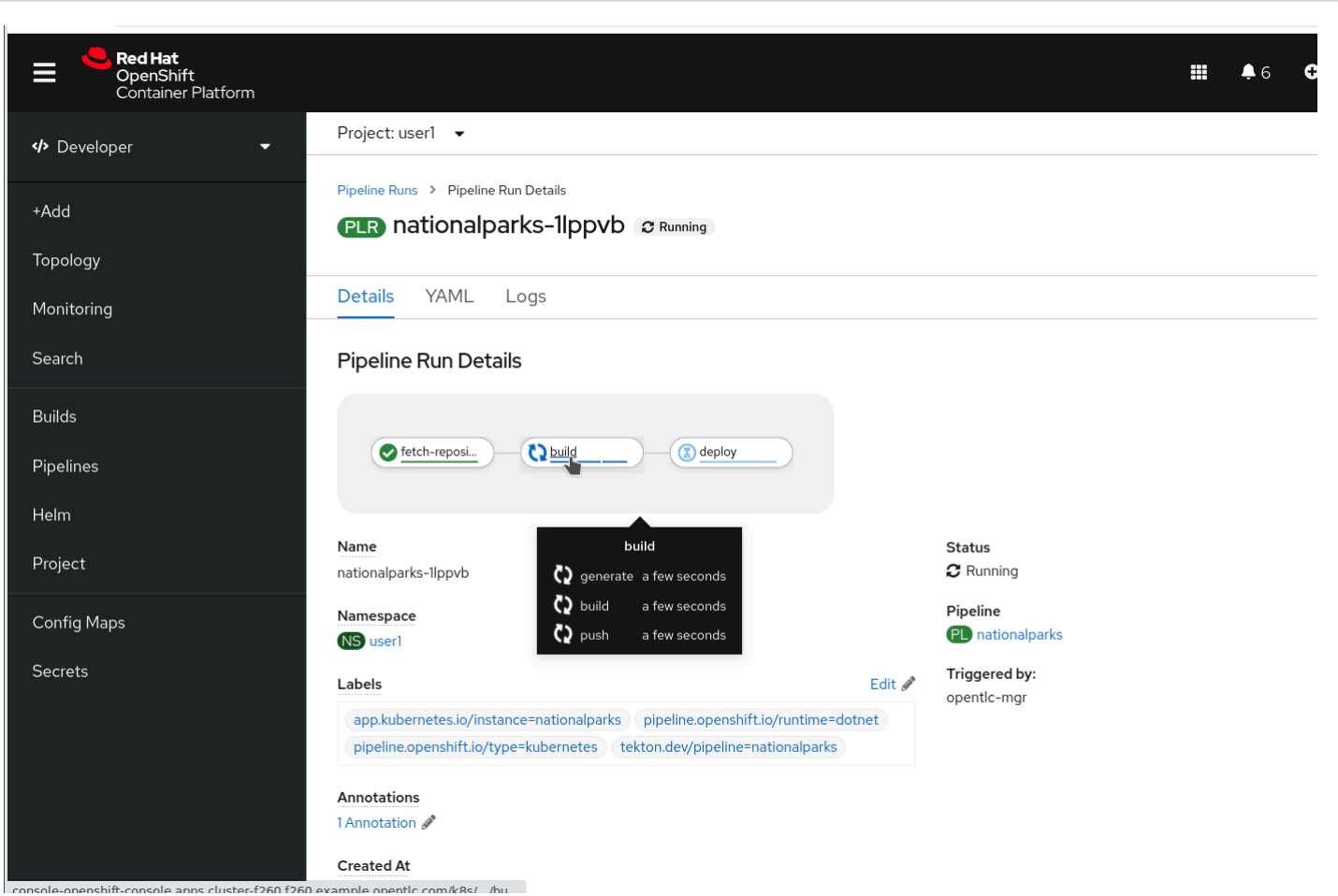
PVC app-source-pvc

Cancel Start

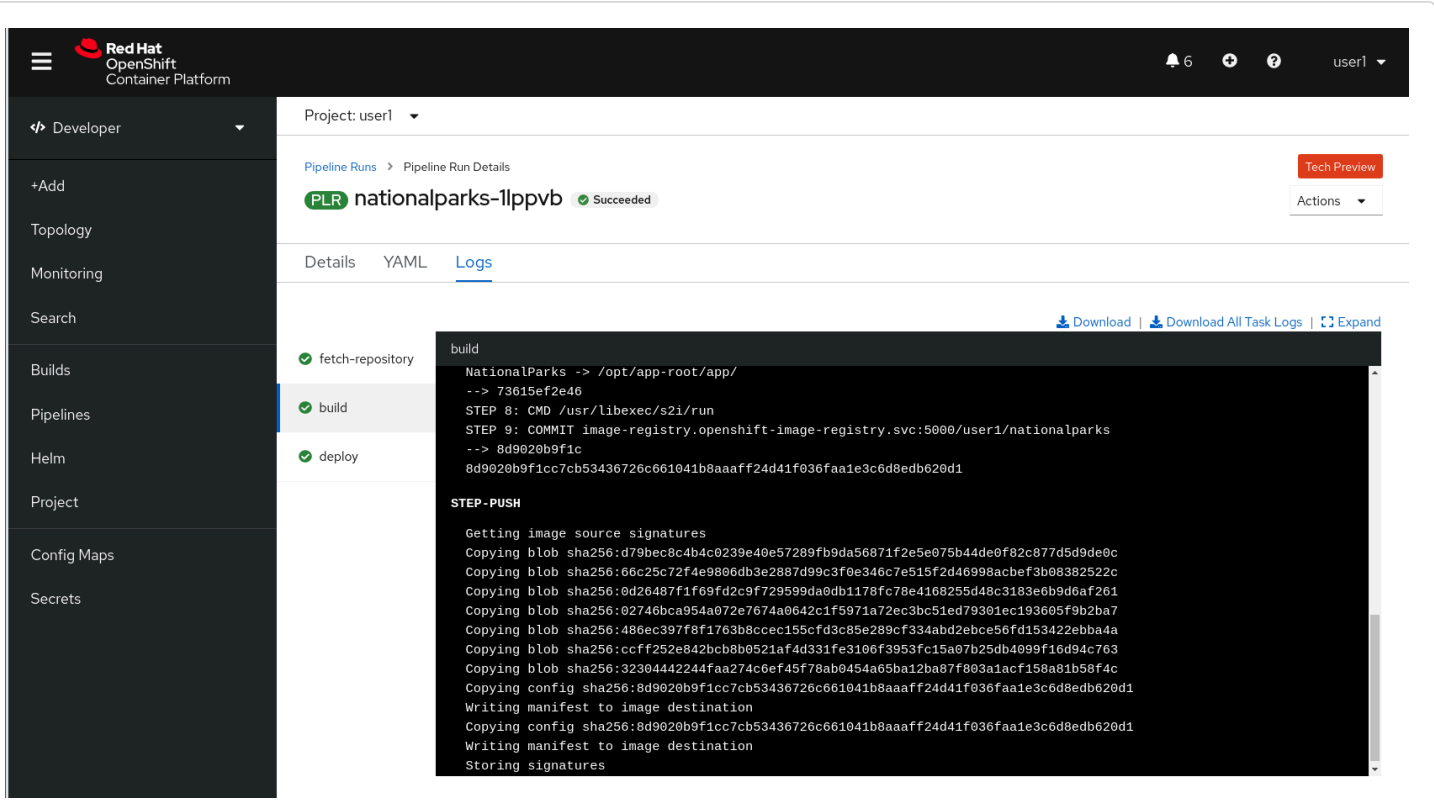
You can follow the Pipeline execution from **Pipeline** section, watching all the steps in progress. Click on **Pipeline Runs** tab to see it running:



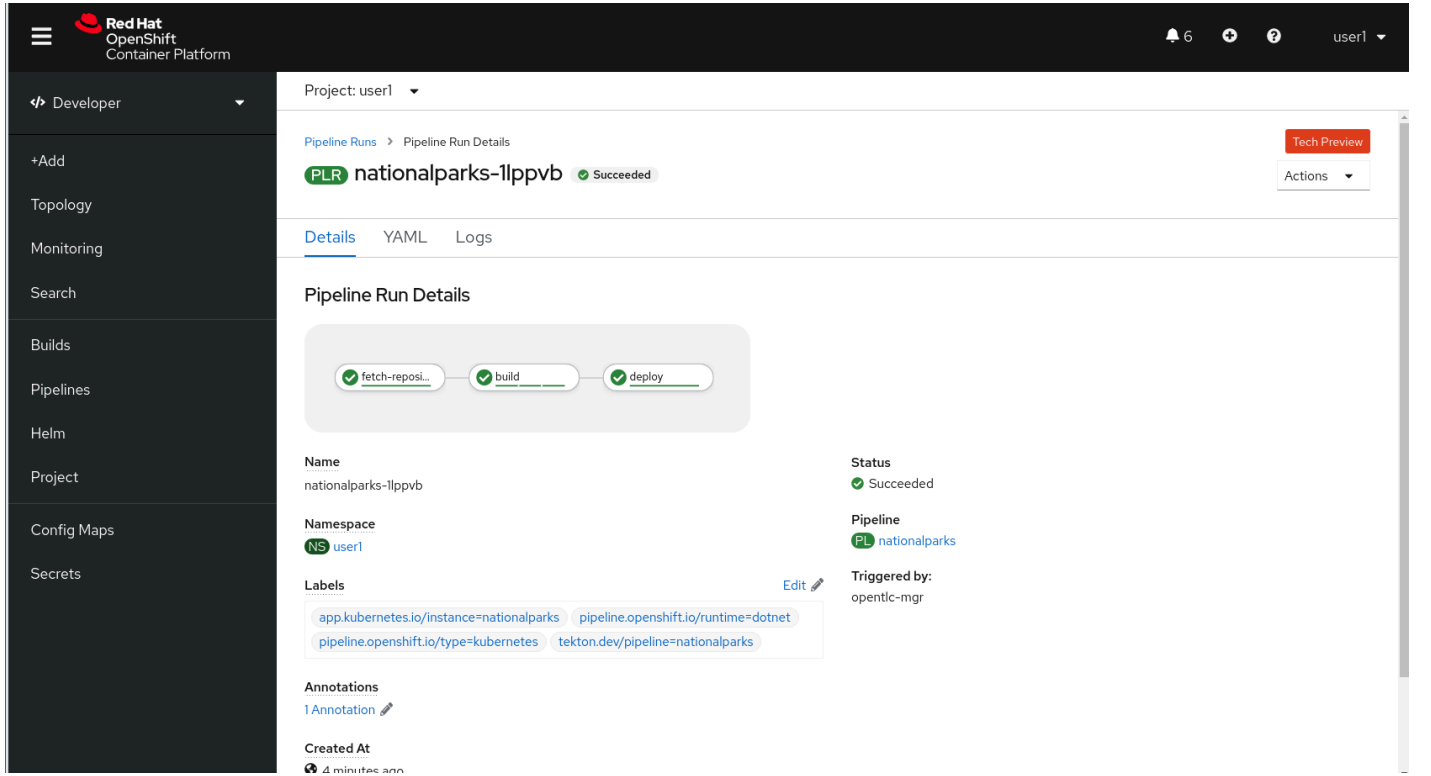
The click on the `PipelineRun` **national-parks-deploy-run-**:



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



Verify PipelineRun has been completed with success:


Red Hat OpenShift Container Platform
6
user1
Developer
+Add
Topology
Monitoring
Search
Builds
Pipelines
Helm
Project
Config Maps
Secrets
Project: user1
Pipeline Runs
Pipeline Run Details
Tech Preview
PLR nationalparks-1lppvb
Succeeded
Actions
Details
YAML
Logs
Pipeline Run Details
fetch-reposi...
build
deploy
Name
nationalparks-1lppvb
Namespace
NS user1
Labels
Edit
app.kubernetes.io/instance=nationalparks
pipeline.openshift.io/runtime=dotnet
pipeline.openshift.io/type=kubernetes
tekton.dev/pipeline=nationalparks
Annotations
1 Annotation
Created At
4 minutes ago
Status
Succeeded
Pipeline
PL nationalparks
Triggered by:
opentlc-mgr

Continue