CI/CD with OpenShift Pipelines

Skills Network

Estimated time needed: 45 minutes

Welcome to the hands-on lab for **CI/CD with OpenShift Pipelines**. In this lab, you will create a CI/CD workflow using the OpenShift Pipelines.

Learning Objectives

After completing this lab, you will be able to:

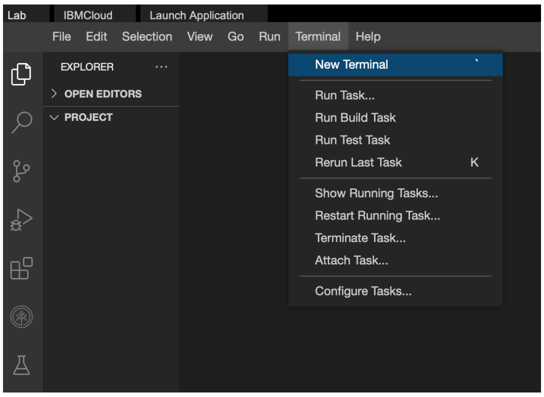
- Create a CI/CD workflow using the OpenShift Pipelines
- Add parameters to tasks created using OpenShift Pipelines
- Add a workspace and persistant volume claim in the OpenShift UI
- Add tasks that clone the GitHub repository, lint the source code, run unit tests and finally deploy the application to the OpenShift cluster

Set Up the Lab Environment

You have a little preparation to do before you can start the lab.

Open a Terminal

Open a terminal window by using the menu in the editor: Terminal > New Terminal.



In the terminal, if you are not already in the `/home/project` folder, change to your project folder now.

```
cd /home/project
```

You can use the following command to ensure you are connected to an OpenShift cluster:

```
oc config current-context
```

You should see something like:

```
captainfedo1-context
```

You are now ready to continue installing the **Prerequisites**.

Optional

If working in the terminal becomes difficult because the command prompt is very long, you can shorten the prompt using the following command:

```
export PS1="[\[\033[01;32m\]\u\[\033[00m\]: \[\033[01;34m\]\W\[\033[00m\]]\$ "
```

Prerequisites

This lab requires installation of the tasks introduced in the previous labs. To be sure, apply the previous tasks to your cluster before proceeding. Reissue these commands:

Establish the Tasks

First create an empty file called `tasks.yaml` in the root folder:

```
touch tasks.yaml
```

Open the `tasks.yaml` file and add the following yaml content.

Open **tasks.yaml** in IDE

```
---
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: cleanup
spec:
  description: This task will clean up a workspace by deleting all the files.
  workspaces:
    - name: source
  steps:
    - name: remove
      image: alpine:3
      env:
        - name: WORKSPACE_SOURCE_PATH
          value: $(workspaces.source.path)
      workingDir: $(workspaces.source.path)
      securityContext:
        runAsNonRoot: false
        runAsUser: 0
      script: |
        #!/usr/bin/env sh
        set -eu
        echo "Removing all files from ${WORKSPACE_SOURCE_PATH} ..."
        # Delete any existing contents of the directory if it exists.
        #
        # We don't just "rm -rf ${WORKSPACE_SOURCE_PATH}" because ${WORKSPACE_SOURCE_PATH} might be "/"
        # or the root of a mounted volume.
        if [ -d "${WORKSPACE_SOURCE_PATH}" ] ; then
          # Delete non-hidden files and directories
          rm -rf "${WORKSPACE_SOURCE_PATH:?}"/*
          # Delete files and directories starting with . but excluding ..
          rm -rf "${WORKSPACE_SOURCE_PATH}"/.[!.]*
          # Delete files and directories starting with .. plus any other character
          rm -rf "${WORKSPACE_SOURCE_PATH}"/..?*
        fi
---
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: nose
spec:
  workspaces:
    - name: source
  params:
    - name: args
      description: Arguments to pass to nose
      type: string
      default: "-v"
  steps:
    - name: nosetests
      image: python:3.9-slim
      workingDir: $(workspaces.source.path)
      script: |
        #!/bin/bash
        set -e
        python -m pip install --upgrade pip wheel
        pip install -r requirements.txt
        nosetests $(params.args)
```

Make sure you save the file. Next, apply the tasks to your OpenShift Cluster:

```
kubectl apply -f tasks.yaml
```

Check that you have all of the previous tasks installed:

```
oc get tasks
```

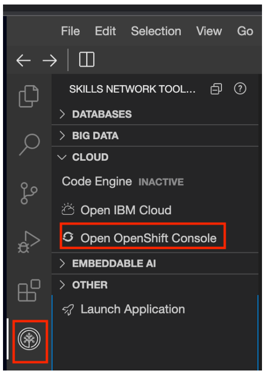
You should see the output similar to this:

```
NAME      AGE
cleanup   5h42m
nose      5h42m
```

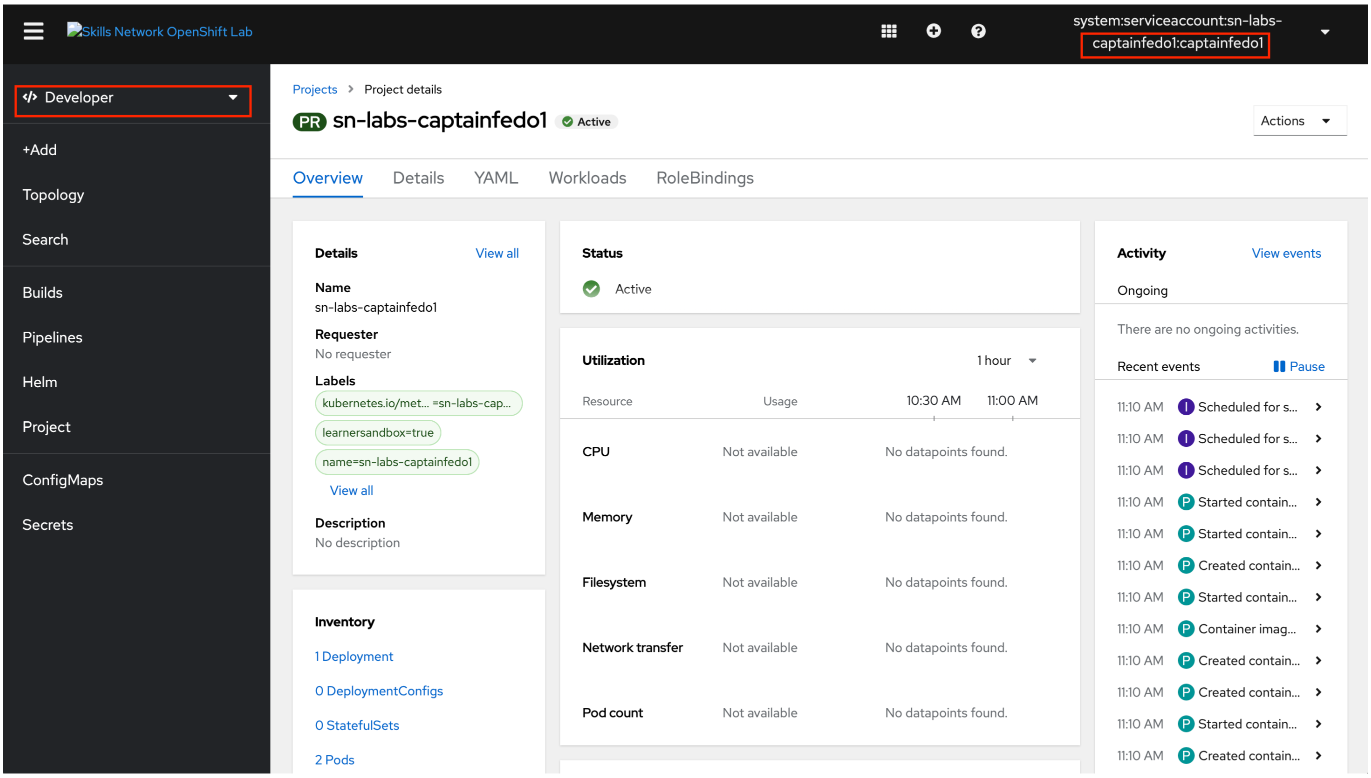
Step 1: Create PersistentVolumeClaim

You also need a PersistentVolumeClaim (PVC) to use as a workspace. You can use the OpenShift Administrator perspective to create the PVC.

Open the OpenShift console using the **Open OpenShift Console** under the **Skills Network Toolbox** menu.



The lab should open the **Developer** perspective for the OpenShift console in a new tab.



PR sn-labs-captainfedo1 Active

Details

Name
sn-labs-captainfedo1

Requester
No requester

Labels
kubernetes.io/met... =sn-labs-cap...
learnersandbox=true
name=sn-labs-captainfedo1
View all

Description
No description

Inventory

1 Deployment
0 DeploymentConfigs
0 StatefulSets
2 Pods

Status

Active

Utilization 1 hour

Resource	Usage	10:30 AM 11:00 AM
CPU	Not available	No datapoints found.
Memory	Not available	No datapoints found.
Filesystem	Not available	No datapoints found.
Network transfer	Not available	No datapoints found.
Pod count	Not available	No datapoints found.

Activity

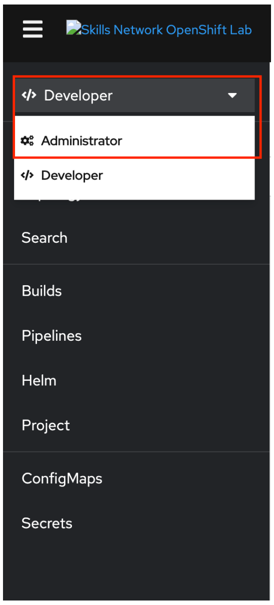
Ongoing

There are no ongoing activities.

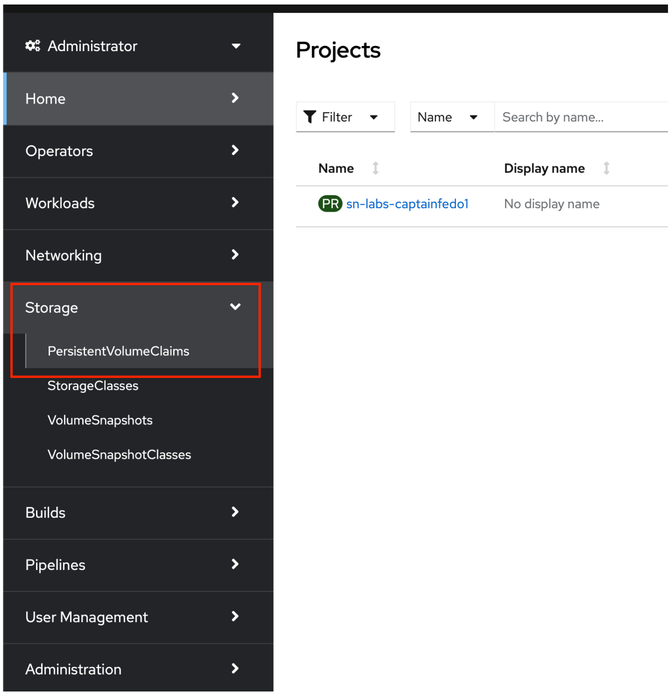
Recent events

11:10 AM I Scheduled for s...
11:10 AM I Scheduled for s...
11:10 AM I Scheduled for s...
11:10 AM P Started contain...
11:10 AM P Started contain...
11:10 AM P Created contain...
11:10 AM P Started contain...
11:10 AM P Container imag...
11:10 AM P Created contain...
11:10 AM P Created contain...
11:10 AM P Started contain...
11:10 AM P Created contain...

Open the **Administrator** perspective using the drop down on the left side of the screen.

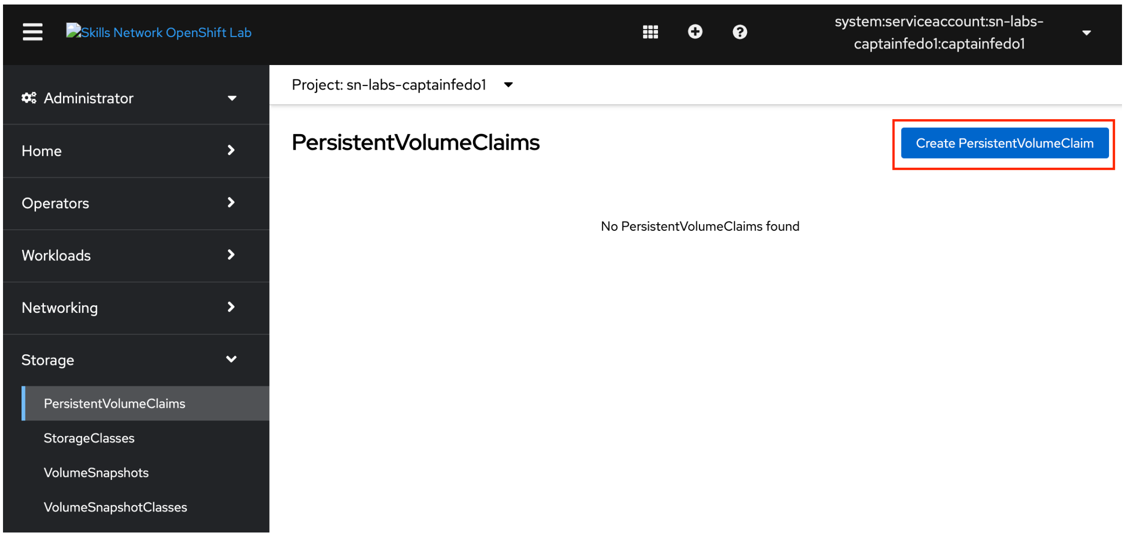


Once the page switches to the Administrator view, click **Storage** and **PersistantVolumeClaims**.



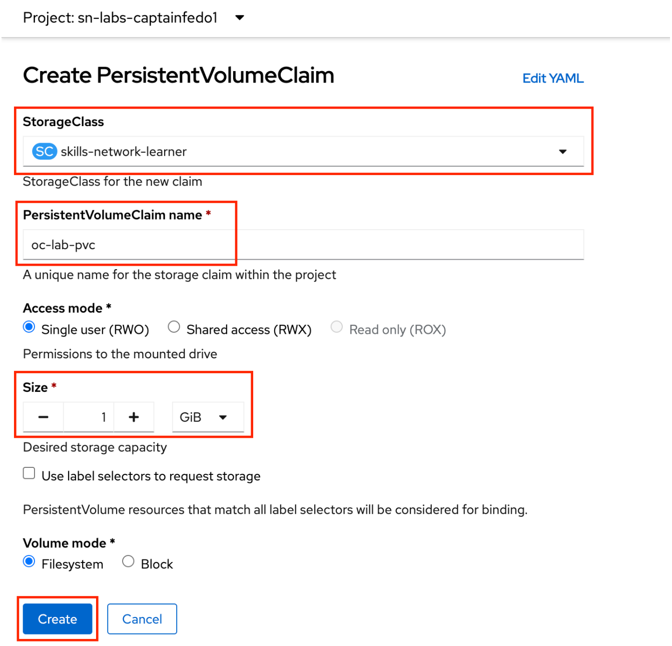
Note: *If you encounter an error when opening OpenShift and accessing the Persistent Claim value, please close the OpenShift window and then reopen it.*

Click `Create PersistentVolumeClaim` to create a new PVC:


Skills Network OpenShift Lab
system:serviceaccount:sn-labs-captainfedo1:captainfedo1
Project: sn-labs-captainfedo1
Administrator
Home
Operators
Workloads
Networking
Storage
PersistentVolumeClaims
StorageClasses
VolumeSnapshots
VolumeSnapshotClasses
PersistentVolumeClaims
Create PersistentVolumeClaim
No PersistentVolumeClaims found

Next, fill out the form as follows:

- StorageClass: `skills-network-learner`
- PersistantVolumenClaim name: `oc-lab-pvc`
- Size: `1GB`


Project: sn-labs-captainfedo1

Create PersistentVolumeClaim

Edit YAML

StorageClass
SC skills-network-learner
StorageClass for the new claim

PersistentVolumeClaim name *
oc-lab-pvc
A unique name for the storage claim within the project

Access mode *
● Single user (RWO) ○ Shared access (RWX) ○ Read only (ROX)
Permissions to the mounted drive

Size *
− 1 + GiB
Desired storage capacity

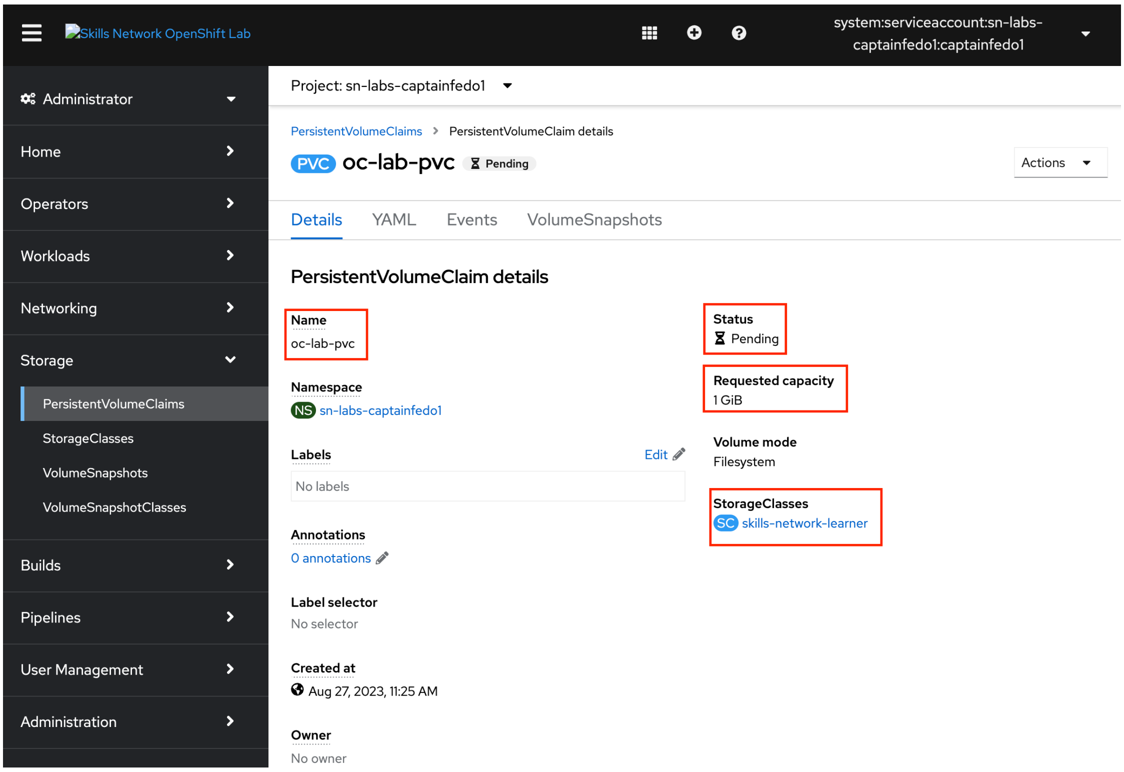
□ Use label selectors to request storage

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

Volume mode *
● Filesystem ○ Block

Create Cancel

Finally, click `Create` to create the PVC. Once the PVC is created, you should see the details. Notice the **Status** is **Pending**. It takes a few minutes for the PVC to complete. You don't have to wait for this to finish as it will most likely be in place by the time you need it in the pipeline.


Skills Network OpenShift Lab
system:serviceaccount:sn-labs-captainfedo1:captainfedo1
Administrator
Home
Operators
Workloads
Networking
Storage
PersistentVolumeClaims
StorageClasses
VolumeSnapshots
VolumeSnapshotClasses
Builds
Pipelines
User Management
Administration
Project: sn-labs-captainfedo1
PersistentVolumeClaims > PersistentVolumeClaim details
PVC oc-lab-pvc Pending
Actions
Details YAML Events VolumeSnapshots
Edit

PersistentVolumeClaim details

Name
oc-lab-pvc

Namespace
NS sn-labs-captainfedo1

Labels
No labels

Annotations
0 annotations

Label selector
No selector

Created at
Aug 27, 2023, 11:25 AM

Owner
No owner

Status
Pending

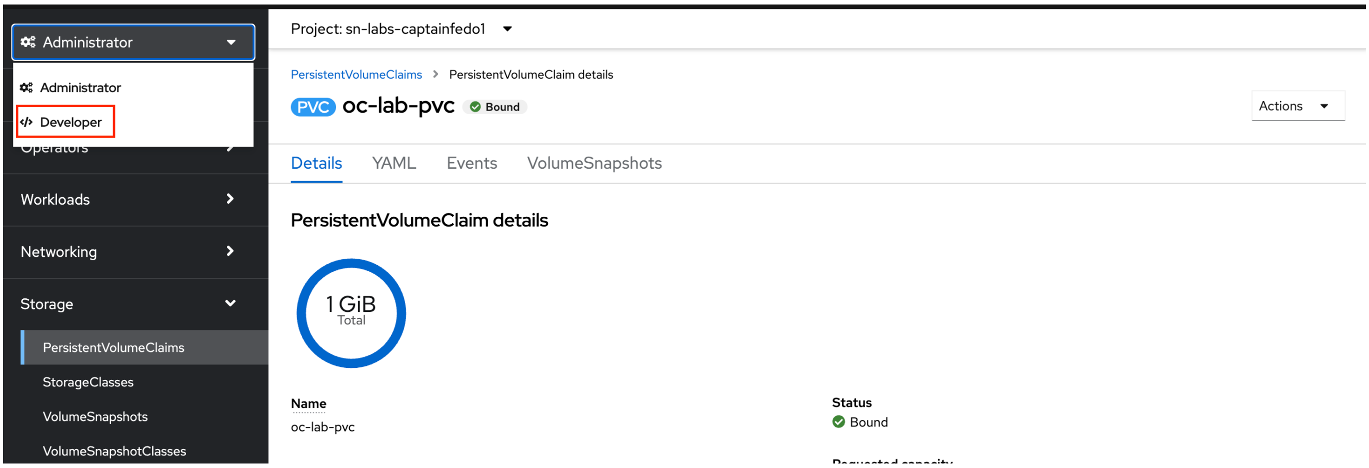
Requested capacity
1 GiB

Volume mode
Filesystem

StorageClasses
SC skills-network-learner

Step 2: Create a new Pipeline

Now that you have a PVC in place, the next step is start working on the pipeline. First, go back to the **Developer** perspective.

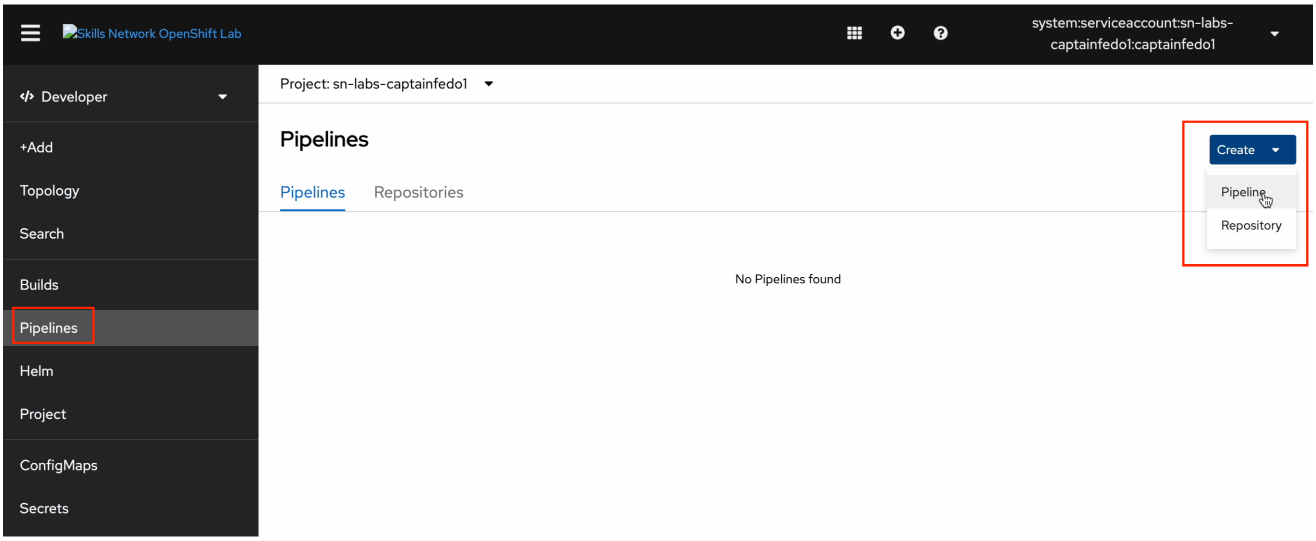


PersistentVolumeClaim details

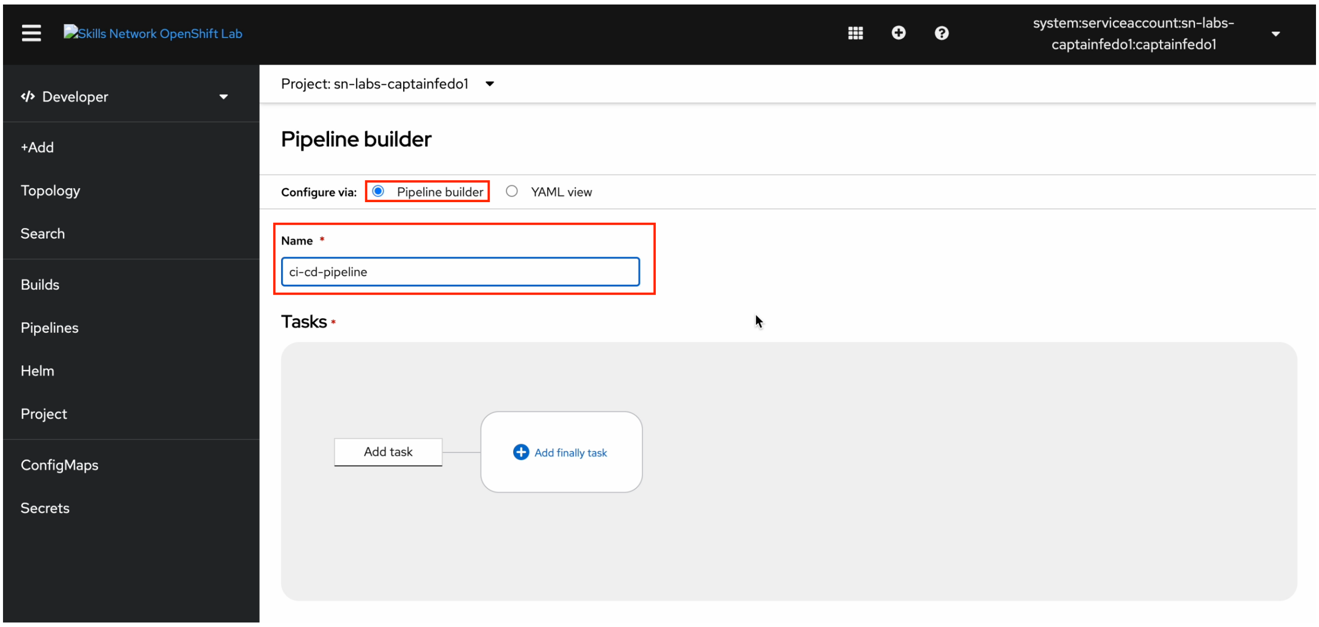
Name
oc-lab-pvc

Status
Bound

Next, click **Pipelines** on the left panel and create a new pipeline.



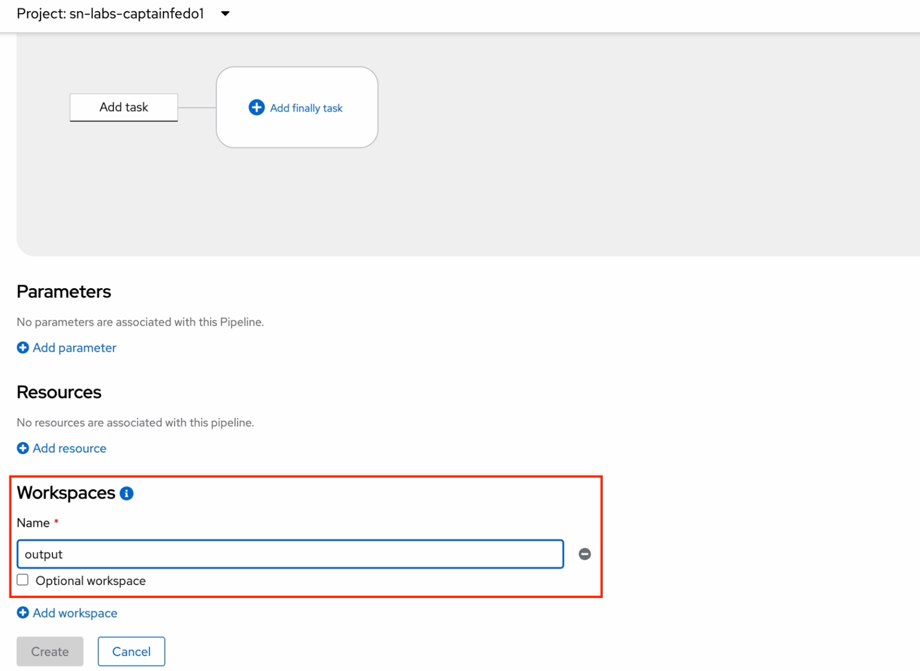
You are presented with the pipeline builder. Ensure you have **Pipeline Builder** selected in **Configure Via** and enter **ci-cd-pipeline** as the name of your pipeline.



Configure via: ● Pipeline builder ○ YAML view

Name *: ci-cd-pipeline

Before you create your first task, let's add a workspace to your pipeline. Scroll to the bottom of the page and add a new workspace with the name `output`. This workspace will be used to clone the code.


Project: sn-labs-captainfedo1
Add task
Add finally task
Parameters
No parameters are associated with this Pipeline.
Add parameter
Resources
No resources are associated with this pipeline.
Add resource
Workspaces
Add workspace
Create
Cancel

Name *: output

□ Optional workspace

Great! We can now start adding tasks to your pipeline.

Step 3: Add the cleanup task

You were asked to apply a `tasks.yaml` file that contained the `cleanup` and the `nose` tasks. You can confirm the tasks are installed by using the following command:

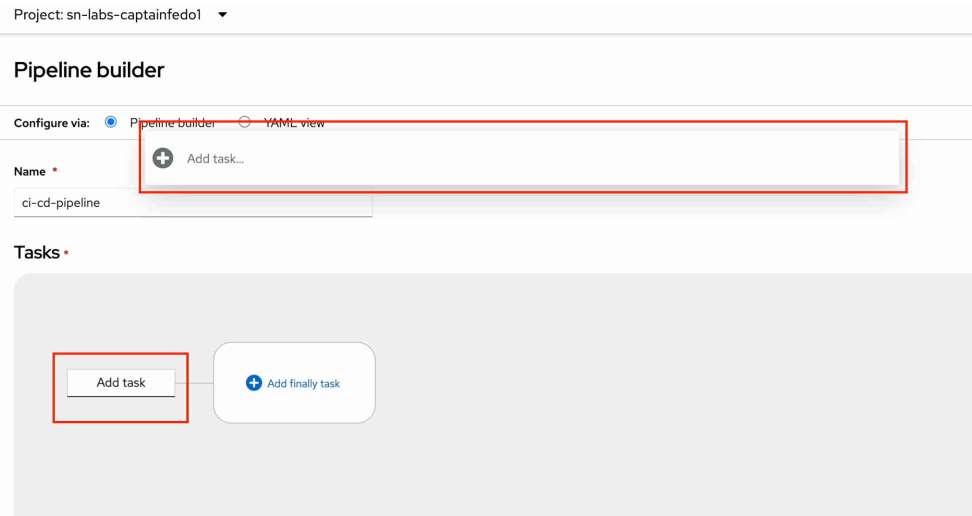
```
oc get tasks
```

You should see the output similar to this:

```
NAME      AGE
cleanup   5h42m
nose      5h42m
```

If you don't see both of these tasks, go back to the `Preqrequisites` step and make sure you apply the tasks.yaml file.

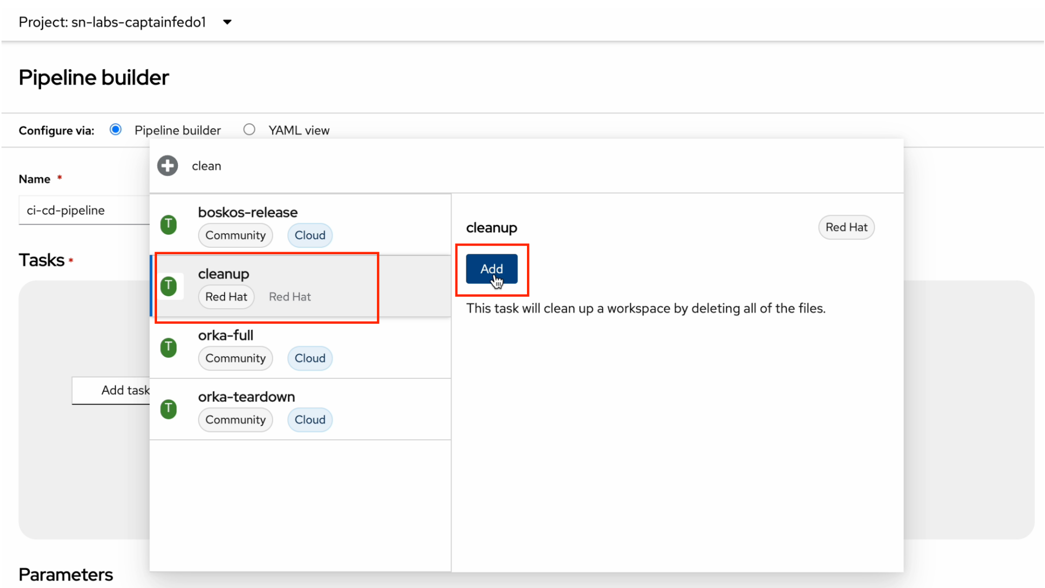
You will create the first task in this step. Click **Add Task** in the builder UI to open the `Add task ...` dialog.



Configure via: ● Pipeline builder ○ YAML view

Name *: ci-cd-pipeline

Type `cleanup` to see the task you installed earlier from the yaml file. Click **Add** to use the task in the builder.



Project: sn-labs-captainfedo1

Pipeline builder

Configure via: ● Pipeline builder ○ YAML view

Name * ci-cd-pipeline

Tasks *

clean

boskos-release Community Cloud

cleanup Red Hat Red Hat

orka-full Community Cloud

orka-teardown Community Cloud

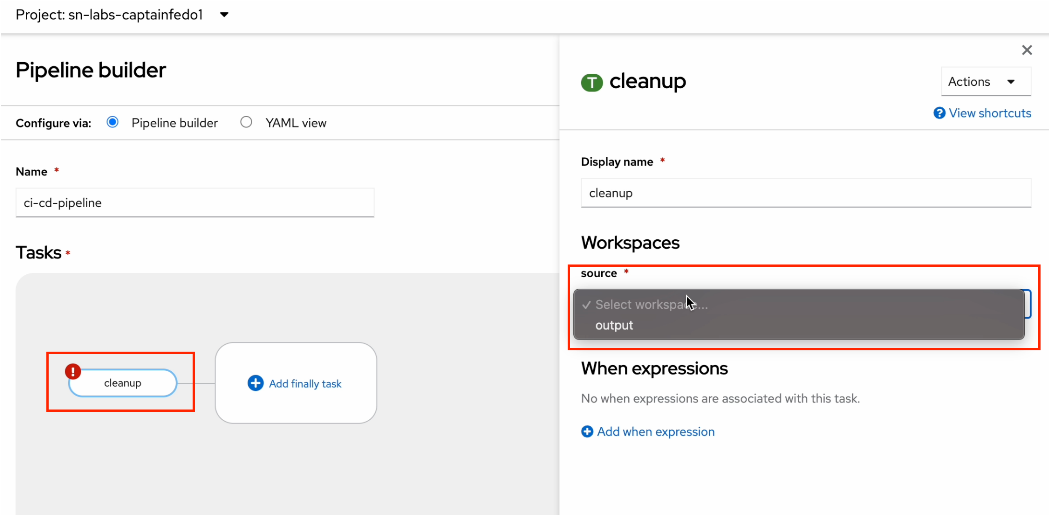
cleanup Red Hat

Add

This task will clean up a workspace by deleting all of the files.

Parameters

This should install your first task. You will notice the red exclamation mark on the task. This means the task has not been completely configured yet. Click on the task to open the task flyout. Change the workspace to **output**.



Project: sn-labs-captainfedo1

Pipeline builder

Configure via: ● Pipeline builder ○ YAML view

Name * ci-cd-pipeline

Tasks *

cleanup

Add finally task

cleanup Actions

View shortcuts

Display name * cleanup

Workspaces

source *

✓ Select workspace...

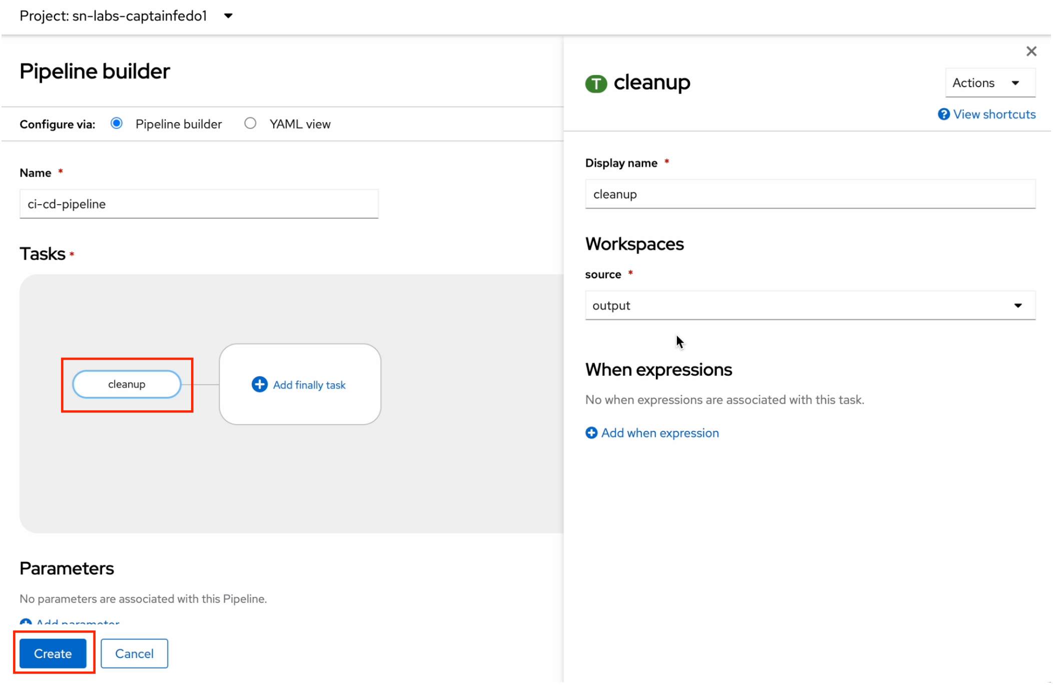
output

When expressions

No when expressions are associated with this task.

Add when expression

You should see the exclamation mark go away and **Create** enabled. Click **Create** to finish creating the task in the pipeline.



Project: sn-labs-captainfedo1

Pipeline builder

Configure via: ● Pipeline builder ○ YAML view

Name * ci-cd-pipeline

Tasks *

cleanup

Add finally task

Parameters

No parameters are associated with this Pipeline.

Create Cancel

cleanup Actions

View shortcuts

Display name * cleanup

Workspaces

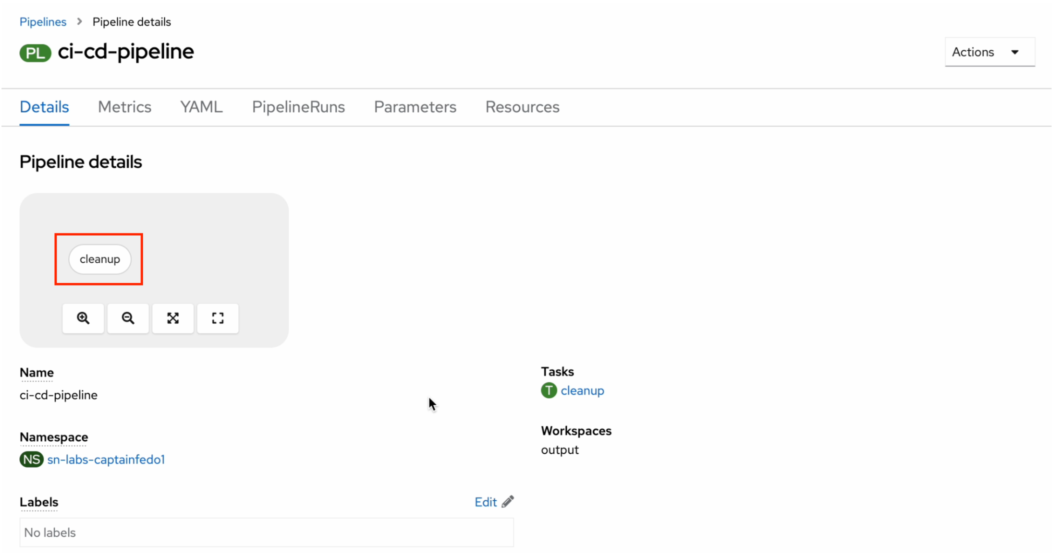
source * output

When expressions

No when expressions are associated with this task.

Add when expression

You should now see your pipeline with the one task you just added.



Name: ci-cd-pipeline

Namespace: NS sn-labs-captainfedo1

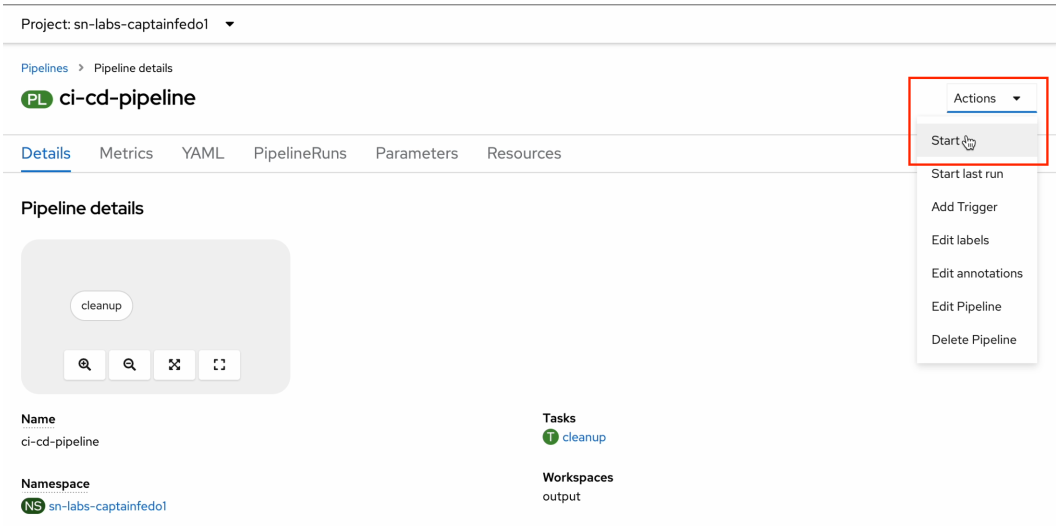
Labels: No labels

Tasks: T cleanup

Workspaces: output

Step 4: Run the Pipeline

Now that you have a pipeline with the one cleanup step, let's see how you can run this pipeline. Click **Pipelines** on the left bar, if you are not already on the pipelines page. Click on **ci-cd-pipeline** pipeline. You can now use the **Actions** dropdown on the left to run the pipeline.



Name: ci-cd-pipeline

Namespace: NS sn-labs-captainfedo1

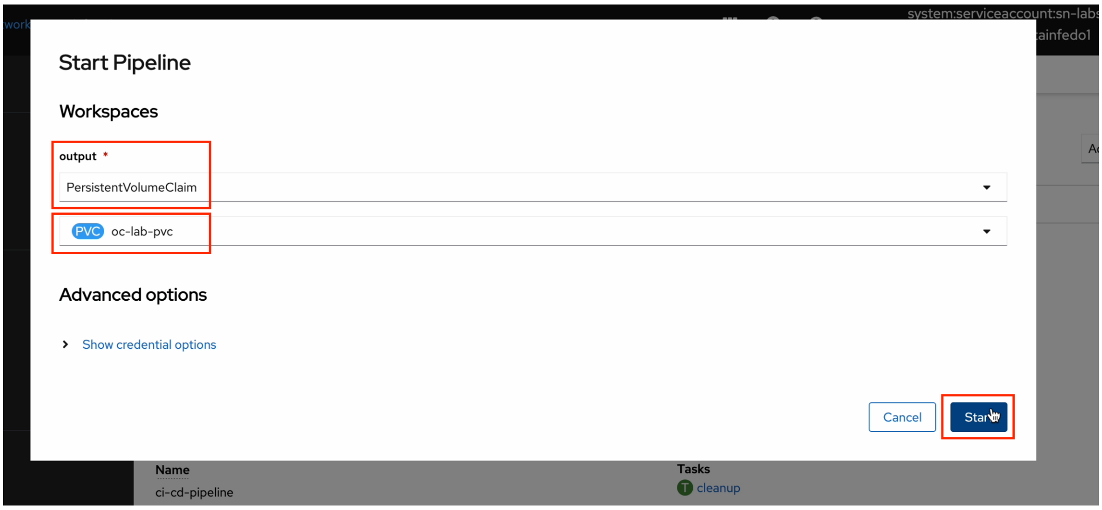
Tasks: T cleanup

Workspaces: output

OpenShift brings up the **Start Pipeline** dialog box. Ensure that you pick the following:

- output: `PersistentVolumeClaim`
- select a PVC: `oc-lab-pvc`

Click **Start** after you have filled out the form.



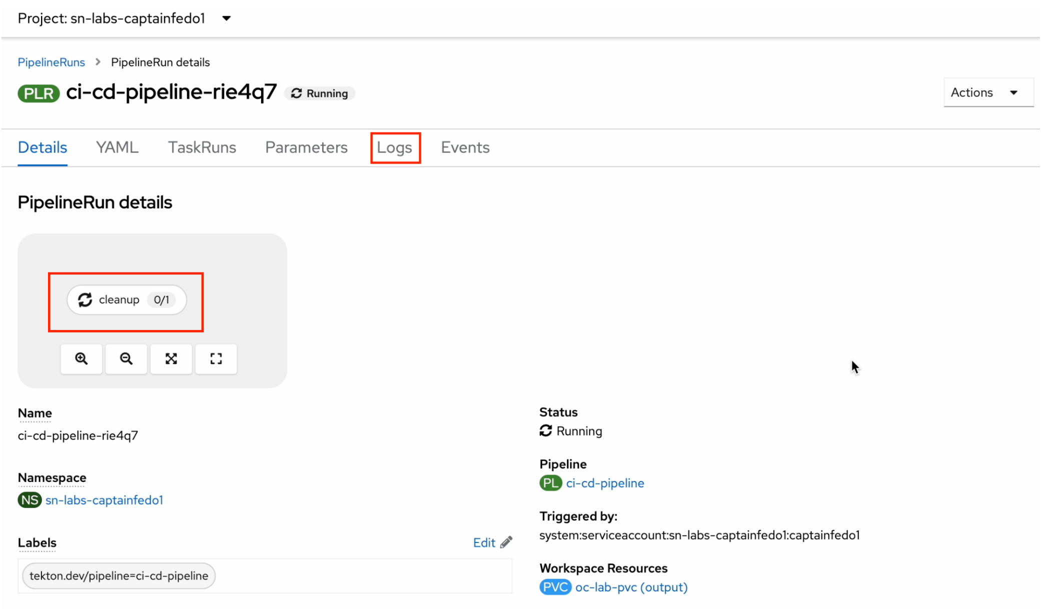
output *: PersistentVolumeClaim

PVC oc-lab-pvc

Name: ci-cd-pipeline

Tasks: T cleanup

You should see the pipeline running on the next page. You can click on the task name to see the logs for a particular task. Alternatively, you can click on the **Logs** tab:


Project: sn-labs-captainfedo1
PipelineRuns > PipelineRun details
PLR ci-cd-pipeline-rie4q7 Running
Actions
Details
YAML
TaskRuns
Parameters
Logs
Events
cleanup 0/1

PipelineRun details

Name
ci-cd-pipeline-rie4q7

Namespace
NS sn-labs-captainfedo1

Labels
tekton.dev/pipeline=ci-cd-pipeline

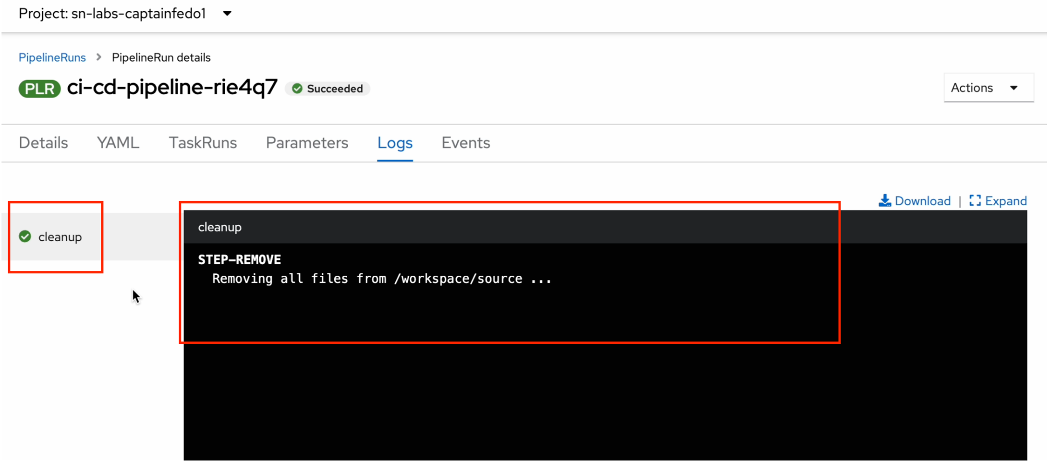
Status
Running

Pipeline
PL ci-cd-pipeline

Triggered by:
system:serviceaccount:sn-labs-captainfedo1:captainfedo1

Workspace Resources
PVC oc-lab-pvc (output)

You can see the detailed logs and also have an option to download them. You will see the task on the right turn green, if it completes successfully.


Project: sn-labs-captainfedo1
PipelineRuns > PipelineRun details
PLR ci-cd-pipeline-rie4q7 Succeeded
Actions
Details
YAML
TaskRuns
Parameters
Logs
Events
Download | Expand
cleanup

```
cleanup

STEP-REMOVE
  Removing all files from /workspace/source ...
```

Congratulations! You created a pipeline from scratch and added the cleanup tasks on it. You then ran the pipeline and viewed the logs. This first task was explained in detail as an example. The lab is now asking you to finish the rest of the tasks in this pipeline on your own. Don't worry! You will be given appropriate hints and you have the option of reaching out to your peers or course TAs in the forums if you get stuck. Good luck!

Step 5: Add the Git Clone task

You are asked to use the `git-clone` in-built task to clone the GitHub code into your pipeline.

Your Task

1. Open the pipeline in edit mode. Select Pipeline from the left menu, select the pipeline name, and the go to Actions -> Edit Pipeline. See hint for a screenshot.
2. Add a new task after the cleanup task in the pipeline from previous step. Hover over the step to display the `+` buttons. Use the `+` button on the right of the task to add a task instead of using the `Add finally task` link. See hint for a screenshot.
3. Look for the `RedHat git-clone` task and add it to the placeholder task.
4. Click on the red exclaimation on the task or the task card to open the configure task flyout. Configure the `git-clone` task as follows:
   - url: `https://github.com/ibm-developer-skills-network/wtecc-CICD_PracticeCode`
   - workspace.output: `output`
5. Save the pipeline.
6. Run the pipeline.
7. Check the logs to see if there are issues with the pipeline.

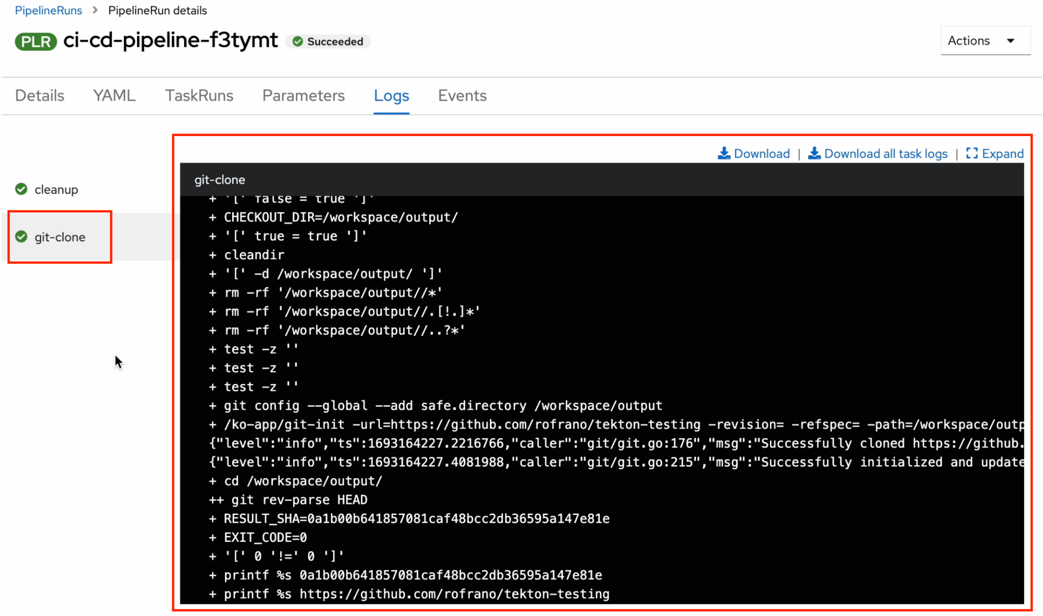
Hint

▸ Click here for a hint.

Check your Solution

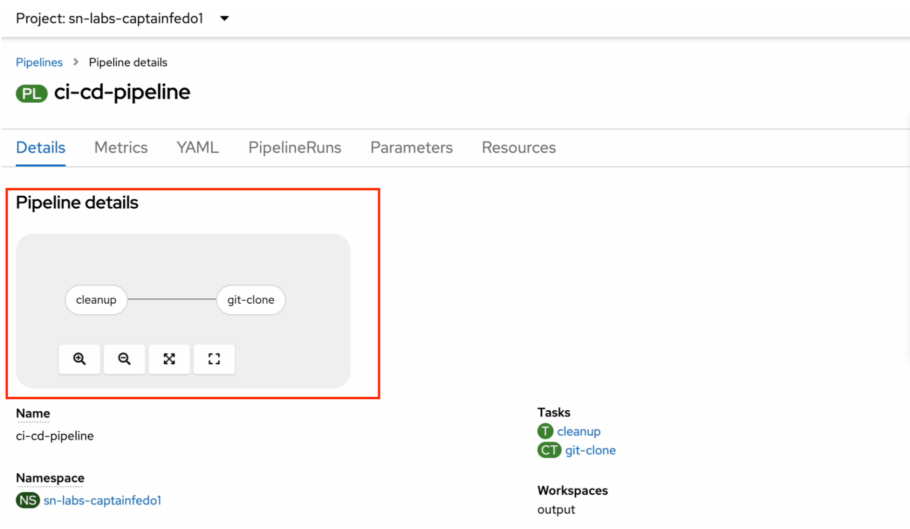
▾ Click here for the solution.

If the pipeline ran successfully, you should see both tasks in green in the logs tab:


PipelineRuns > PipelineRun details
PLR ci-cd-pipeline-f3tymt Succeeded
Actions
Details
YAML
TaskRuns
Parameters
Logs
Events
cleanup
git-clone
Download | Download all task logs | Expand

```
git-clone
  + '[' false = true ']'
  + CHECKOUT_DIR=/workspace/output/
  + '[' true = true ']'
  + cleandir
  + '[' -d /workspace/output/ ']'
  + rm -rf '/workspace/output//*'
  + rm -rf '/workspace/output//.[!.]*'
  + rm -rf '/workspace/output//..?*'
  + test -z ''
  + test -z ''
  + test -z ''
  + git config --global --add safe.directory /workspace/output
  + /ko-app/git-init -url=https://github.com/rofrano/tekton-testing -revision= -refspec= -path=/workspace/outp
  {"level":"info","ts":1693164227.2216766,"caller":"git/git.go:176","msg":"Successfully cloned https://github.
  {"level":"info","ts":1693164227.4081988,"caller":"git/git.go:215","msg":"Successfully initialized and update
  + cd /workspace/output/
  ++ git rev-parse HEAD
  + RESULT_SHA=0a1b00b641857081caf48bcc2db36595a147e81e
  + EXIT_CODE=0
  + '[' 0 '!=' 0 ']'
  + printf %s 0a1b00b641857081caf48bcc2db36595a147e81e
  + printf %s https://github.com/rofrano/tekton-testing
```

If you complete the task successfully, your pipeline should look as follows:



Name
ci-cd-pipeline

Namespace
NS sn-labs-captainfedo1

Tasks
T cleanup
CT git-clone

Workspaces
output

Step 6: Add the Flake8 task

You are asked to use the `Flake8` in-built task to lint the source code. As part of this task, you will configure the task with specific arguments.

Your Task

1. Open the pipeline in edit mode.
2. Add a new task after the git-clone task in the pipeline from previous step.
3. Look for the `Flake8` task from the community.
4. Install and add it to the placeholder task.
5. Click on the red exclaimation on the task or the task card to open the configure task flyout. Configure the `Flake8` task as follows:
    - image: `python:3.9-slim`
    - arg: `--count`
    - arg: `--max-complexity=10`
    - arg: `--max-line-length=127`
    - workspace.source: `output`
6. Save the pipeline.
7. Run the pipeline.
8. Check the logs to see if there are issues with the pipeline.

Hint

▸ Click here for a hint.

Check your Solution

▸ Click here for the solution.

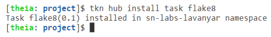
Step 7: Add the Nose task

The next step is to add the `nose` task for unit testing the source code of the application.

Your Task

1. Open the pipeline in edit mode.
2. Add a new task after the Flake8 task in the pipeline from previous step.
3. Look for the `nose` task.
4. Install and add it to the placeholder task.

   Note: If you encounter an error stating: missing workspaces, kindly run the following command in the terminal window to install flake8:
   `tkn hub install task flake8`
   You should then see this:



```
[theia: project]$ tkn hub install task flake8
Task flake8(0.1) installed in sn-labs-lavanyar namespace
[theia: project]$ █
```

5. Click on the red exclaimation on the task or the task card to open the configure task flyout. Configure the `nose` task as follows:
    - workspace.source: `output`
6. Save the pipeline.
7. Run the pipeline.
8. Check the logs to see if there are issues with the pipeline.

Check your Solution

▸ Click here for the solution.

Step 8: Add the buildah task

The next step is to add a task to create an image from the GitHub source code. You will use the `buildah` in-built task to perform this action.

Your Task

1. Open the pipeline in edit mode.
2. Add a new task after the nose task in the pipeline from previous step.
3. Look for the `buildah` task from RedHat.
4. Install and add it to the placeholder task.
5. You will need the namespace of your lab environment for one of the arguments. You can obtain this by using the command line terminal and using the `echo $SN_ICR_NAMESPACE` command in the lab terminal.

   ```
   echo $SN_ICR_NAMESPACE
   ```

6. Click on the red exclaimation on the task or the task card to open the configure task flyout. Configure the `buildah` task as follows:
    - image: `$(params.build-image)`
    - workspace.source: `output`
7. Click on the main page to close the flyout. Add the following parameter and the default value to the pipeline:
    - parameter.name: `build-image`
    - parameter.default: `image-registry.openshift-image-registry.svc:5000/SN_ICR_NAMESPACE/tekton-lab:latest`.
    - Replace `SN_ICR_NAMESPACE` with the value above.
8. Save the pipeline.
9. Run the pipeline.
10. Check the logs to see if there are issues with the pipeline.

Hint

▸ Click here for a hint.

Check your Solution

▸ Click here for the solution.

Step 9: Deploy Application

Next, you will create a task to deploy the image you created to the lab OpenShift cluster. You will use the `OpenShift client` task to execute the `oc deploy` command with the image you built in the previous step.

Your Task

1. Open the pipeline in edit mode.
2. Add a new task after the buildah task in the pipeline from the previous step.
3. Look for the `openshift-client` task from RedHat.
4. Install and add it to the placeholder task.
5. Click on the red exclaimation on the task or the task card to open the configure task flyout. Configure the task with the following
    - display name: `deploy`
    - SCRIPT: `oc create deployment $(params.app-name) --image=$(params.build-image) --dry-run=client -o yaml | oc apply -f -`
6. Click on the main page to close the flyout. Add the following parameter and the default value to the pipeline:
    - parameter.name: `app-name`
    - parameter.default: `cicd-app`.
7. Save the pipeline.
8. Run the pipeline.
9. Check the logs to see if there are issues with the pipeline.

Hint

▸ Click here for a hint.

Check your Solution

▾ Click here for the solution.

If the pipeline ran successfully, you should see all tasks in green in the logs tab. You should also see the output from the `deploy` task indicating all tests have passed successfully.