

# Final Project Lab (Option B: JavaScript): Continuous Integration and Continuous Delivery (CI/CD)

Estimated duration: 60 Minutes



Welcome to the **Continuous Integration and Continuous Delivery (CI/CD) Final Project** development environment. Now it's time to apply all that you have learned in the previous modules of this course. This lab environment will provide you with a sample application and an OpenShift Cluster, which will enable you to carry out the following objectives:

## Objectives

- Create a CI pipeline in GitHub Actions with steps for **linting** and **unit testing**.
- Use Tekton to create tasks for **linting**, **unit testing**, and **building an image**.
- Create an OpenShift CI Pipeline that uses the previously created Tekton steps.
- Add the deploy step to the OpenShift pipeline that deploys the code to the lab OpenShift cluster.

You should complete all the work in the final project in this lab environment.

## Prerequisite

### Important security information

Welcome to the Cloud IDE with OpenShift. This lab environment is where all of your development will take place. It has all the tools you will need, including an OpenShift cluster.

It is essential to understand that the lab environment is **ephemeral**. It only lives for a short while and then will be destroyed. Hence, you must push all changes made to your own GitHub repository to recreate it in a new lab environment, whenever required.

Also, note that this environment is shared and, therefore, not secure. You should not store personal information, usernames, passwords, or access tokens in this environment for any purpose.

## Your task

1. If you still need to generate a **GitHub Personal Access Token**, you should do so now. You will need it to push code back to your repository. It should have `repo` and `write` permissions and set to expire in **60 days**. When Git prompts you for a password in the Cloud IDE environment, use your Personal Access Token instead.
2. You can recreate this environment by performing **Initialize Development Environment** each time.
3. Create a repository from the GitHub template provided for this lab in the next step.

## Create your own GitHub repository

You will need your repository to complete the final project. We have provided a GitHub Template to create your repository in your own GitHub account. **Do not Fork the repository as it's already a template**. This will avoid confusion when making Pull Requests in the future.

## Your task

1. In a browser, visit this GitHub repository:  
<https://github.com/ibm-developer-skills-network/hdjgq-ci-cd-final-project-template>
2. From the GitHub **Code** tab, use the green **Use this template** to create your repository from this template.
3. Select **Create a new repository** from the dropdown menu. On the next screen, fill out these prompts following the screenshot:

A screenshot of a GitHub repository page. The URL in the address bar is 'github.com/ibm-developer-skills-network/hdjgq-ci-cd-final-project-template'. The repository name is 'hdjgq-ci-cd-final-project-template', generated from 'ibm-developer-skills-network/coding-project-template'. The 'Code' tab is selected. At the top, there is a dropdown for branches ('main'), showing '1 Branch' and '0 Tags'. A search bar says 'Go to file'. Below the header, there is a commit by 'harsh-skillup' with the message 'feat: initialize Node.js counter service with API endpoints and testing'. The repository structure shows '.github/workflows', '.tekton', 'bin', and 'src' folders. Each folder has a commit message 'feat: initialize Node.js counter service with API endpoints and testing'. The bottom of the screenshot lists two numbered steps: 'Select your GitHub account from the dropdown list.' and 'Name the repository ci-cd-final-project and ensure it is set to Public so it can be accessed during AI evaluation and by your peers.'

4. Select your GitHub account from the dropdown list.

5. Name the repository `ci-cd-final-project` and ensure it is set to Public so it can be accessed during AI evaluation and by your peers.

6. (Optional) Add a description to let people know what this repo is for.

7. Use the [Create repository from template](#) to create the repository in your GitHub account.



## New repository

# Create a new repository

A repository contains all project files, including the revision history elsewhere? [Import a repository](#).

*Required fields are marked with an asterisk (\*).*

### Repository template



ibm-developer-skills-network/hdjgq-ci-cd-final-project-te

Start your repository with a template repository's contents.

**Include all branches**

Copy all branches from ibm-developer-skills-network/hdjgq-ci-cd-final-project-te branch.

**Owner \***



harsh-skillup ▾

**Repository name \***

ci-cd-final-project

ci-cd-final-project is available.

Great repository names are short and memorable. Need inspiration?

**Description (optional)**

Final Project on CI/CD project

Public

Anyone on the internet can see this repository. You choose who

Private

You choose who can see and commit to this repository.

You are creating a public repository in your personal account.

Note: These steps only need to be done once. Whenever you re-enter this lab, you should start from the next page, [Initialize Development Environment](#)

## Initialize Development Environment

As previously covered, the Cloud IDE with OpenShift environment is ephemeral, and might delete at any time. The Cloud IDE with OpenShift environment will create a new environment the next time you enter the lab. Unfortunately, you will need to initialize your development environment every time. This shouldn't happen too often as the environment can last for several days at a time, but when it is gone, this is the procedure to recreate it.

### Overview

Each time you need to set up your lab development environment, you will need to run three commands.

Each command will be explained in further detail, one at a time, in the following section.

{your\_github\_account} represents your GitHub account username.

The commands include:

```
git clone https://github.com/{your_github_account}/ci-cd-final-project.git
cd ci-cd-final-project
bash ./bin/setup.sh
exit
```

Now, let's discuss these commands and explain what needs to be done.

## Task details

Initialize your environment using the following steps:

1. Open a terminal with Terminal -> New Terminal if one isn't open already.
2. Next, use the `export GITHUB_ACCOUNT=` command to export an environment variable containing your GitHub account.

Note: Substitute your real GitHub account that you used to create the repository for the `{your_github_account}` placeholder:

```
export GITHUB_ACCOUNT={your_github_account}
```

3. Then use the following commands to clone your repository, change it into the `devops-capstone-project` directory, and execute the `./bin/setup.sh` command.

```
git clone https://github.com/$GITHUB_ACCOUNT/ci-cd-final-project.git
cd ci-cd-final-project
bash ./bin/setup.sh
```

You should see the following at the end of the setup execution:

```
theia@theiaopenshift-harshsingh15: /home/project/ci-cd-final-project ✘ theia@theiaopenshift-harshsingh15
```

### 3 high severity vulnerabilities

To address all issues (including breaking changes), run:  
`npm audit fix --force`

Run `npm audit` for details.

```
npm notice
npm notice New major version of npm available! 10.8.2 -> 11.4.2
npm notice Changelog: https://github.com/npm/cli/releases/tag/v11.4.2
npm notice To update run: npm install -g npm@11.4.2
npm notice
*** Setting up development environment...
*****
Node.js Counter Service Environment Setup Complete
*****
```

Use 'npm start' to run the service  
Use 'npm run dev' for development with auto-reload  
Use 'npm test' to run tests

```
theia@theiaopenshift-harshsingh15:/home/project/ci-cd-final-project$
```

4. Finally, close the current terminal using the `exit` command. The environment won't be fully active until you open a new terminal in the next step.

```
exit
```

## Validate

In order to validate that your environment is working correctly, you must open a new terminal because the Node.js environment will only activate when a new terminal is present. You should have ended the previous task using the `exit` command to exit the terminal.

1. Open a terminal with Terminal -> New Terminal and check that everything worked correctly by using the `which node` command:

Your prompt should look like this:

```
theia@theiaopenshift-harshsingh15:/home/project/ci-cd-final-project$ theia@theia
theia@theiaopenshift-harshsingh15:/home/project$ cd ci-cd-final-project/
theia@theiaopenshift-harshsingh15:/home/project/ci-cd-final-project$ which node
```

Check which Node you are using:

```
which node
```

You should get back:

```
theia@theiaopenshift-harshsingh15:/home/project$ cd ci-cd-final-project/
theia@theiaopenshift-harshsingh15:/home/project/ci-cd-final-project$ which node
/usr/bin/node
theia@theiaopenshift-harshsingh15:/home/project/ci-cd-final-project$ █
```

Check the Node version:

```
node --version
```

You should get back some patch level of Node 20.19.3:

```
theia@theiaopenshift-harshsingh15:/home/project/ci-cd-final-project$ which node
/usr/bin/node
theia@theiaopenshift-harshsingh15:/home/project/ci-cd-final-project$ node --version
v20.19.3
theia@theiaopenshift-harshsingh15:/home/project/ci-cd-final-project$ █
```

This completes the setup of the development environment.

You are now ready to start working.

## Final project scenario

You're part of a team responsible for building an innovative microservice, a RESTful API that allows users to manage and track counters. Another team has already developed the user interface (UI) for this microservice, and it's now your turn to ensure the reliability and efficiency of the backend services.

### Continuous Integration (CI) with GitHub Actions

Your first task is to set up CI pipelines using GitHub Actions. The codebase comes with unit tests for the provided endpoints. Your goal is to automate the linting and testing processes. You will create a GitHub Actions workflow that triggers whenever changes are pushed to the repository.

### Continuous Deployment (CD) with OpenShift Pipelines

In the second phase, establish CD pipelines within OpenShift Pipelines. These pipelines should include linting, testing, building an image, and the seamlessly deploying the microservice to an OpenShift cluster.

You need to provide the URL for your repository with the GitHub workflow and tekton yaml files in addition to other terminal outputs and screenshots as evidence of your work. Your evidence will be essential for either peer project evaluation or AI graded evaluation. Best of luck with your project!

## Final Project

### Note:

1. Please ensure that all updates to the files are properly saved.
  2. Save and Push your project files to Github and capture screenshots as mentioned in the respective tasks.
  3. You can submit your project deliverables through either Option 1: AI-Graded Submission and Evaluation or Option 2: Peer-Graded Submission and Evaluation.
- 4. For Option 1: AI-Graded Submission and Evaluation:**
- Submission requires the Github URL of **workflow.yaml**, **tasks.yaml**, and **Readme.md** files along with Text response for **Task 7** and **Task 10** and screenshots for **Task 6**, **Task 8** and **Task 9**.
- 5. For Option 2: Peer-Graded Submission and Evaluation:**
- Submission requires screenshots for **Tasks 6 to 10** along with Github URL of **workflow.yaml**, **tasks.yaml** files and **Github repo** URL also.

### Exercise 0 - Push CI code to GitHub

To test the workflow and the CI pipeline, you need to commit the changes and push your branch back to the GitHub repository. Each new push to the main branch should trigger the workflow.

#### Your task

1. Configure the Git account with your email and name using the `git config --global user.email` and `git config --global user.name` commands.

▼ Click here for a hint.

Open the terminal and configure your email:

```
git config --global user.email "you@example.com"
```

Open the terminal and configure your user name

```
git config --global user.name "Your Name"
```

2. Update your **Readme.md** file in your GitHub repository and add the Project name as **ci-cd-final-project** under **CI/CD Tools and Practices Final Project Template**.

3. Make sure to stage all the changes you made in the exercises and push them to your forked repo on GitHub.

▼ Click here for a hint.

You can use the following commands to commit your changes to staging and then push to your forked repository:

```
git add -A  
git commit -m "COMMIT MESSAGE"  
git push
```

Your output should look similar to the following image:

#### Solution

```
theia@theiaopenshift-harhsingh15:/home/project/ci-cd-final-project$ git push  
Username for 'https://github.com': harsh-skillup  
Password for 'https://harsh-skillup@github.com':  
Enumerating objects: 9, done.  
Counting objects: 100% (9/9), done.  
Delta compression using up to 16 threads  
Compressing objects: 100% (4/4), done.  
Writing objects: 100% (5/5), 658 bytes | 658.00 KiB/s, done.  
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0  
remote: Resolving deltas: 100% (1/1), completed with 1 local object.  
To https://github.com/harsh-skillup/ci-cd-final-project.git  
 ff61375..7cb6e57 main -> main
```

#### Assessment:

**For Option 1 - AI Graded Submission and Evaluation:** Copy and paste the the public Github URL of `Readme.md` file and save it for the Submission later.

**For Option 2 - Peer Graded Submission and Evaluation:** Copy and paste the the public Github URL of `ci-cd-final-project` and save it for the Peer review Submission later.

## Exercise 1: Create basic workflow

Your GitHub repository has an empty workflow file, `.github/workflows/workflow.yml`. You will create the CI workflow by writing several steps in this workflow file.

[Open workflow.yml in IDE](#)

#### Your task

Open the `.github/workflows/workflow.yml` file and add the following:

```
1. name: CI workflow  
2. workflow triggers: push on main branch and pull_request on main branch  
3. Jobs  
    o runs-on: ubuntu-latest  
4. Checkout step:  
    o name: Checkout  
    o uses: actions/checkout@v3  
5. Setup Node.js step:  
    o name: Setup Node.js  
    o uses: actions/setup-node@v3  
    o with node-version: 20  
6. Install Dependencies step:  
    o name: Install dependencies  
    o run npm ci commands
```

You can also refer to the videos and labs in the module 2 of the course in case you want to familiarize yourself with the concepts before proceeding further.

#### Hint

▼ Click here for a hint.

You can use the following file as a template for this exercise:

```
name: {name of workflow}  
on:  
  push:  
    branches: {array of branches}  
  pull_request:  
    branches: {array of branches}  
jobs:  
  build:  
    runs-on: {placeholder}  
    steps:  
      - name: {placeholder}  
        uses: {placeholder}  
      - name: {placeholder}  
        uses: {placeholder}  
        with:  
          node-version: {placeholder}  
      - name: {placeholder}  
        run: {placeholder}
```

## Exercise 2: Add the linting step to CI workflow

Next, you will add the Lint step to the GitHub workflow. You will use ESLint module for linting. Open the `.github/workflows/workflow.yml` file and complete the following tasks.

[Open workflow.yml in IDE](#)

#### Your task

Add a linting task with the following details:

```
1. name: Lint with ESLint  
2. commands:  
    o npm run lint
```

You can refer to the videos and labs in the module 2 for help.

## Hint

▼ Click here for a hint.

You can use the following file as a template for this exercise:

```
- name: {placeholder}
  run: {placeholder}
```

## Exercise 3: Add the test step to CI workflow

Next, you will add the test step to the GitHub workflow. You will use the Jest module for running the tests. Open the .github/workflows/workflow.yml file and complete the following tasks.

[Open workflow.yml in IDE](#)

## Your task

Add a test step with the following details:

1. name: Run unit tests with Jest
2. command:
  - o npm test

You can refer to the videos and labs in the module 2 for help.

## Hint

▼ Click here for a hint.

You can use the following file as a template for this exercise:

```
- name: Run unit tests with Jest
  run: {insert command here}
```

Make sure to stage all the changes you made in the exercises and push them to your forked repo on GitHub.

## Assessment:

For Option 1 - AI Graded Submission and Evaluation and For Option 2 - Peer Graded Submission and Evaluation: Copy and paste the the public Github URL of workflow.yaml and save it for the Submission later.

## Exercise 4: Validate GitHub Actions Workflow

1. To validate that your workflow ran and was successful, simply go to your version of the repository on GitHub and click Actions.

The screenshot shows a GitHub repository page for 'harsh-skillup / ci-cd-final-project'. The 'Actions' tab is highlighted with a red box. Below the header, the repository name 'ci-cd-final-project' is shown, along with its status as 'Public' and a note that it was generated from 'ibm-developer-skills-network/hdjgq-ci-cd-final-project-template'. The repository summary shows 'main' branch, '1 Branch', and '0 Tags'. The 'Actions' section displays a table of workflow runs. The first run, by user 'harsh-skillup', is labeled 'COMMIT MESSAGE' with a green checkmark. It shows three steps: '.github/workflows' (Commit message), '.tekton' (Initial commit), and 'bin' (Initial commit). A 'Go to file' button is visible in the top right corner of the table area.

Workflow	Step	Status
harsh-skillup / ci-cd-final-project	COMMIT MESSAGE	✓
	.github/workflows	Commit message
	.tekton	Initial commit
	bin	Initial commit

You can click the CI Workflow to see more details.

# All workflows

Showing runs from all workflows



## Help us improve GitHub Actions

Tell us how to make GitHub Actions work better for you with three quick questions.

2 workflow runs

### ✓ COMMIT MESSAGE

CI workflow #5: Commit [8ca00ca](#) pushed by harsh-skillup

main

### ✗ Initial commit

CI workflow #1: Commit [ff61375](#) pushed by harsh-skillup

main

2. Finally, you can drill into the action to confirm all the steps succeeded.

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#)

← CI workflow

## ✓ COMMIT MESSAGE #5

Summary

Jobs

✓ build

Run details

⌚ Usage

⤷ Workflow file

### build

succeeded 6 minutes ago in 14s

- > ✓ Set up job
- > ✓ Checkout
- > ✓ Setup Node.js
- > ✓ Install dependencies
- > ✓ Lint with ESLint
- > ✓ Run unit tests with Jest
- > ✓ Post Setup Node.js
- > ✓ Post Checkout
- > ✓ Complete job

#### Assessment:

**For Option 2 Peer Graded Submission and Evaluation:** Take a screenshot of the action workflow page and name the file `cicd-github-validate(.png)` and save it for the Peer review submission later.

3. To get the list of the action workflow of your GitHub repository in the terminal, Run the following command:-

```
gh run list  
gh run view <run-id>  
gh run view <run-id> verbose
```

*Replace <run-id> with the ID displayed in the output from the gh run list command.*

Note: Ensure you are inside the correct repository directory and have successfully authenticated using `gh auth login` before running these commands.

Your output should appear similar to the image below:

```
theia@theiaopenshift-alimaahter:/home/project/ci-cd-final-project$ gh run view 18366848200 --verbose  
✓ main CI workflow · 18366848200  
Triggered via push about 1 day ago  
  
JOBS  
✓ build in 5s (ID 52321211487)  
✓ Set up job  
✓ Checkout  
✓ Setup Node.js  
✓ Simulated Failing Test  
✓ Post Setup Node.js  
✓ Post Checkout  
✓ Complete job  
  
ANNOTATIONS  
X Process completed with exit code 1.  
build: .github#7
```

#### Assessment:

**For Option 1 - AI Graded Submission and Evaluation:** Copy and paste the terminal output of action workflow in the text file named `cicd-github-validate` and save it for the Submission later.

## Exercise 5: Create cleanup Tekton task

Congratulations on successfully creating the GitHub CI workflow to checkout, lint, and test your code. The next step is to create the CD pipeline in OpenShift. Before you can do that, create the cleanup task that will clean the output workspace so that the CD pipeline can start fresh.

Open the `.tekton/tasks.yaml` file and complete the following tasks.

[Open tasks.yaml in IDE](#)

### Your task

Add a cleanup task with the following details:

```
1. apiVersion: tekton.dev/v1beta1  
2. kind: Task  
3. name: cleanup  
4. spec.workspaces.name: source
```

This task will have a single step called `remove` as follows:

```
1. name: remove  
2. image: alpine:3  
3. env:  
   o name: WORKSPACE_SOURCE_PATH  
   o value: ${workspaces.source.path}  
4. workingDir: ${workspaces.source.path}  
5. securityContext  
   o runAsNonRoot: false  
   o runAsUser: 0  
6. 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 ${WORKSPACE_SOURCE_PATH} might be "/"  
   # on the root of a mounted volume.  
   if [ -d "${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
```

You can also refer to the videos and labs in the module 3 of the course in case you want to familiarize yourself with the concepts before proceeding further.

### Hint

▼ Click here for a hint.

You can use the following file as a template for this exercise:

```
---  
apiVersion: {placeholder}  
kind: {placeholder}  
metadata:  
  name: {placeholder}  
spec:  
  description: This task will clean up a workspace by deleting all the files.  
  workspaces:  
    - name: {placeholder}  
  steps:  
    - name: {placeholder}  
      image: {placeholder}  
      env:  
        - name: {placeholder}  
          value: {placeholder}
```

```
workingDir: {placeholder}
securityContext:
  runAsNonRoot: {placeholder}
  runAsUser: {placeholder}
script: |
{placeholder}
```

## Exercise 6: Create lint Tekton task

You have added the `cleanup` task to the tekton file. Next, add the test task called `eslint` right under the `cleanup` task.

Open the `.tekton/tasks.yaml` file and complete the following tasks.

[Open tasks.yaml in IDE](#)

### Your Task

Add a testing task with the following details:

```
1. apiVersion: tekton.dev/v1beta1
2. kind: Task
3. name: eslint
4. spec.workspaces.name: source
5. params:
  o name: args
  o description: Arguments to pass to ESLint
  o type: string
  o default: "src/ tests/ --ext *.js"
```

This task will have a single step called `lint` as follows:

```
1. name: lint
2. image: node:20-alpine
3. workingDir: ${workspaces.source.path}
4. script:
    #!/bin/sh
    set -e
    echo "Installing dependencies..."
    npm ci
    echo "Running ESLint..."
    npm run lint ${params.args}
```

### Hint

▼ Click here for a hint.

You can use the following file as a template for this exercise:

```
---
apiVersion: {placeholder}
kind: {placeholder}
metadata:
  name: {placeholder}
spec:
  workspaces:
    - name: {placeholder}
  params:
    - name: {placeholder}
      description: {placeholder}
      type: {placeholder}
      default: {placeholder}
  steps:
    - name: {placeholder}
      image: {placeholder}
      workingDir: {placeholder}
      script: |
{placeholder}
```

## Exercise 7: Create test Tekton task

You have added the `cleanup` task to the tekton file. Next, add the test task called `jest-test` right under the `cleanup` task.

Open the `.tekton/tasks.yaml` file and complete the following tasks.

[Open tasks.yaml in IDE](#)

### Your Task

Add a testing task with the following details:

```
1. apiVersion: tekton.dev/v1beta1
2. kind: Task
3. name: jest-test
4. spec.workspaces.name: source
5. params:
  o name: args
  o description: Arguments to pass to jest
  o type: string
  o default: "-v"
```

This task will have a single step called `jest` as follows:

```
1. name: jest
2. image: node:20-alpine
3. workingDir: ${workspaces.source.path}
4. script:
    #!/bin/sh
    set -e
```

```
echo "Installing dependencies..."  
npm ci  
echo "Running Jest tests..."  
npm test ${params.args}
```

You can also refer to the videos and labs in the module 3 of the course in case you want to familiarize yourself with the concepts before proceeding further.

#### Hint

▼ Click here for a hint.

You can use the following file as a template for this exercise:

```
---  
apiVersion: {placeholder}  
kind: {placeholder}  
metadata:  
  name: {placeholder}  
spec:  
  workspaces:  
    - name: {placeholder}  
  params:  
    - name: {placeholder}  
      description: {placeholder}  
      type: {placeholder}  
      default: {placeholder}  
  steps:  
    - name: {placeholder}  
      image: {placeholder}  
      workingDir: {placeholder}  
      script: |  
        {placeholder}
```

Make sure to stage all the changes you made in the exercises and push them to your forked repo on GitHub.

#### Assessment:

**For Option 1 - AI Graded Submission and Evaluation and For Option 2 - Peer Graded Submission and Evaluation:** Copy and paste the public GitHub URL of `.tekton/tasks.yaml` and save it for the Submission later.

## Exercise 8: Create OpenShift pipeline

You are almost done with the final project. Now that you have the tasks created, you will need to:

- Install the tasks in the lab OpenShift cluster
- Create CD pipeline

Please follow the process mentioned in the Hands-on Lab: CI/CD with OpenShift in Module 4 Pipelines for doing the following tasks.

You can refer to the videos and other content in the Module 4 [Lab: CI/CD with OpenShift](#) of the course in case you want to familiarize yourself with the concepts before proceeding further.

#### Your task

1. In the terminal, install the `cleanup`, `eslint`, and `jest-test` tasks by applying the `tasks.yaml` file with `kubectl apply -f .tekton/tasks.yaml` command.
2. Open the OpenShift console from the lab environment.
3. Create a PVC through terminal as mentioned in the previous lab or either from the `Administrator` perspective with
  - storageclass: `skills-network-learner`
  - select a PVC: `oc-lab-pvc`
  - size: 1GB
4. Create a new pipeline and a workspace called `output`
5. Add the following steps in this order:
  - `cleanup`
  - `git clone`
  - `eslint linting`
  - `jest tests`
  - `buildah task`
6. Test the pipeline works. Take a screenshot as described in this exercise's `Solutions` section.
7. Add the final step of deploying the application to the lab openshift cluster using the `OpenShift client` task and the `oc deploy` command.
  - `oc create deployment ${params.app-name} --image=${params.build-image} --dry-run=client -o yaml | oc apply -f -`

You can refer to the videos and other content in the module 4 of the course in case you want to familiarize yourself with the concepts before proceeding further.

#### Hint

The PVC options should look as follows:

The screenshot shows the OpenShift pipeline console interface. On the left, there's a sidebar with various navigation items like Home, Operators, Workloads, Networking, Storage, Pipelines, User Management, and Administration. The 'PersistentVolumeClaims' item under the Storage section is currently selected and highlighted in blue.

The main content area displays the details for a PersistentVolumeClaim named 'oc-lab-pvc'. At the top, it says 'Project: sn-labs-harshsingh15'. Below that, it shows the 'PVC' status as 'oc-lab-pvc' with a 'Pending' status indicator. There are tabs for 'Details', 'YAML', 'Events', and 'VolumeSnapshots', with 'Details' being the active tab.

The 'PersistentVolumeClaim details' section contains the following fields:

- Name**: oc-lab-pvc (This field is highlighted with a red box.)
- Namespace**: ns sn-labs-harshsingh15
- Labels**: No labels
- Annotations**: 0 annotations
- Label selector**: No selector
- Created at**: 29 Jun 2025, 17:12
- Owner**: No owner

Below these details, there's a section titled 'Conditions'.

#### Assessment:

**For Option 1 - AI Graded Submission and Evaluation and For Option 2 - Peer Graded Submission and Evaluation:** Take a screenshot of this page and save it as `oc-pipelines-console-pvc-details.png` for the submission later.

At the end of this exercise, you can validate the solution as follows:

#### Solution

1. Confirm the pipeline has the following steps:

 **ci-cd-pipeline**
[Details](#)   [YAML](#)   [PipelineRuns](#)   [Parameters](#)   [Metrics](#)
**Pipeline details****Name**

ci-cd-pipeline

**Namespace**
 sn-labs-harshsingh15
**Labels**

No labels

[Edit](#) 
**Tasks**

-  cleanup
-  git-clone
-  eslint
-  jest-test
-  buildah
-  openshift-client

**Workspaces**

output

**Annotations**0 annotations **Created at**
 29 Jun 2025, 17:22
**Owner**

No owner

**Assessment:**
**For Option 1 - AI Graded Submission and Evaluation and For Option 2 - Peer Graded Submission and Evaluation:** Take a screenshot of this page and save it as oc-pipelines-oc-final(.png) for the submission later.

1. Confirm the pipeline runs as shown:

Project: sn-labs-harhsingh15 ▾

PipelineRuns > PipelineRun details

**PLR ci-cd-pipeline-vxt6mz** Succeeded

Details YAML TaskRuns Parameters **Logs** Events

The screenshot shows the PipelineRun details page for a run named 'ci-cd-pipeline-vxt6mz'. The 'Logs' tab is selected. On the left, a list of steps is shown with green checkmarks: 'cleanup', 'git-clone', 'eslint', 'jest-test', 'buildah', and 'deploy'. The 'deploy' step is currently active, indicated by a blue selection bar. The log output for this step shows: 'deployment.apps/cicd-app created'. The rest of the log area is blacked out.

Assessment:

For Option 1 - AI Graded Submission and Evaluation and For Option 2 - Peer Graded Submission and Evaluation: Take a screenshot of this page and save it as oc-pipelines-oc-green(.png) for the submission later.

2. Confirm you can see the application logs in the OpenShift console:

Project: sn-labs-harhsingh15 ▾

Pods > Pod details

**P cicd-app-5dfff976cc-nx7dn** Running

Details YAML Environment **Logs** Events Terminal

The screenshot shows the Pod details page for a pod named 'cicd-app-5dfff976cc-nx7dn'. The 'Logs' tab is selected. At the top, there are controls for 'Log streaming...', a dropdown for 'tekton-lab', a 'Current log' dropdown, and a search bar. The log output area shows 9 lines of text, starting with command-line entries like 'counter-service@1.0.0 start' and 'node src/app.js', followed by a timestamp '[2025-06-29T12:13:52.959Z] [INFO] Logging handler established', a separator line '\*\*\*\*\*', the text 'S E R V I C E R U N N I N G', another separator line '\*\*\*\*\*', and finally 'Server running on port 8000'.

Assessment:

For Option 1 - AI Graded Submission and Evaluation: Copy and paste the the output of SERVICERUNNING and save it in a text file named as oc-pipeline-app-logs for the Final Project Submission and Evaluation.

For Option 2 - Peer Graded Submission and Evaluation: Take a screenshot of the SERVICERUNNING and save it as oc-pipeline-app-logs.png for Peer Assignment.

## Submission

Commit the code to your Github repository

1. Use `git status` to ensure that you have committed your changes locally in the development environment.

2. Use the `git add` command to update the staging area's code.

3. Commit your changes using `git commit -m <commit message>`

4. Push your local changes to a remote branch using the `git push` command

Note: Use your GitHub **Personal Access Token** as your password in the Cloud IDE environment. You may also need to configure Git the first time you use it with:

```
git config --local user.email "you@example.com"  
git config --local user.name "Your Name"
```

Submit the link to your GitHub repository when completed.

## Evaluation

Follow the checklist below to verify that your project meets all requirements before submission.

**Submit your work through either Option 1: AI-Graded Submission and Evaluation or Option 2: Peer-Graded Submission and Evaluation, depending on the submission path you choose for project evaluation.**

### Option 1: AI-Graded Submission

1. Submit the GITHUB URL of the `README.md` file that contains the Project name details. (2 points)
2. Provide the GitHub URL of `.github/workflows/workflow.yml` showing the code snippet for the Lint with flake8 step or ESLint and the code snippet for the Run unit tests with nose step or Jest-test (4 points)
3. Provide the GitHub URL of `.tekton/tasks.yml` showing the code snippet for the cleanup task and the code snippet for the nose task or Jest-test (4 points)
4. Provide the screenshot of the OpenShift PersistentVolumeClaim details in a file named `oc-pipelines-console-pvc-details.png` (2 points)
5. Provide the text of terminal output named `cicd-github-validate` showing details of GitHub actions running successfully in the actions workflow containing all the steps.(2 points)
6. Provide the screenshot showing details of the OpenShift Pipeline `oc-pipelines-oc-final.png`(2 points)
7. Provide the screenshot showing details of the OpenShift Pipeline running successfully in a file named `oc-pipelines-oc-green.png`(2 points)
8. Provide the text response saved in a text file named `oc-pipelines-app-logs` showing details of the OpenShift application logs (2 points)

### Option 2: Peer-Graded Submission

1. The GitHub repo URL that you pushed your changes to. Should be of the format [https://github.com/{your\\_github\\_account}/ci-cd-final-project.git](https://github.com/{your_github_account}/ci-cd-final-project.git)
2. Provide the GitHub URL of the `.github/workflows/workflow.yml` file showing the code snippet for the linting step.
3. Provide the GitHub URL of the `.github/workflows/workflow.yml` file showing the code snippet for the test step.
4. Provide the GitHub URL of the `.tekton/tasks.yml` file showing the code snippet for the cleanup task.
5. Provide the GitHub URL of the `.tekton/tasks.yml` file showing the code snippet for the nose test task.
6. Screenshot showing OpenShift PVC details. Name this file `oc-pipelines-console-pvc-details(.png)`
7. Screenshot showing GitHub actions running successfully. Name this file `cicd-github-validate(.png)`
8. Screenshot showing details of the OpenShift Pipeline. Name this file `oc-pipelines-oc-final(.png)`
9. Screenshot showing details of the OpenShift Pipeline running successfully. Name this file `oc-pipelines-oc-green(.png)`
10. Screenshot of the running application logs from OpenShift console. Name this file `oc-pipelines-app-logs(.png)`

## Sample Files for Tasks 6-10

▼ Click here for a hint.

1. Task 6: `oc-pipelines-console-pvc-details(.png/jpeg)`

Administrator

Home

Operators

Workloads

Networking

Storage

PersistentVolumeClaims

StorageClasses

VolumeSnapshots

VolumeSnapshotClasses

Builds

Pipelines

User Management

Administration

Project: sn-labs-harshsingh15

PersistentVolumeClaims > PersistentVolumeClaim details

**PVC** **oc-lab-pvc** Pending

[Details](#) [YAML](#) [Events](#) [VolumeSnapshots](#)

**PersistentVolumeClaim details**

**Name** **oc-lab-pvc**

**Namespace** **NS** [sn-labs-harshsingh15](#)

**Labels**  
No labels

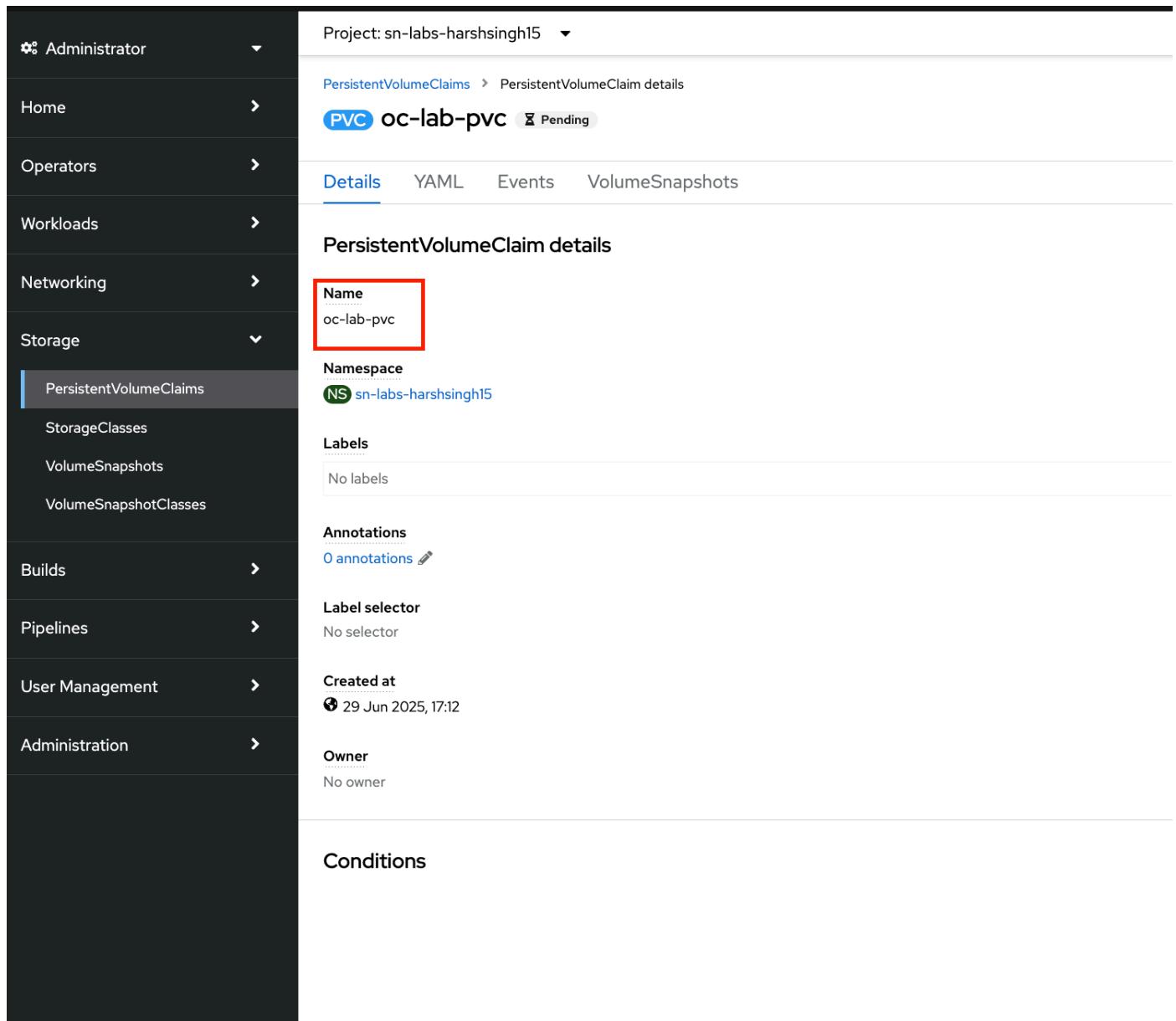
**Annotations**  
0 annotations

**Label selector**  
No selector

**Created at**  
 29 Jun 2025, 17:12

**Owner**  
No owner

**Conditions**



2. Task 7: cicd-github-validate(.png/jpeg)

← CI workflow

## ✓ COMMIT MESSAGE #2

The image shows a screenshot of a Continuous Integration (CI) pipeline interface. On the left, there's a summary page with a sidebar containing links like 'Jobs', 'Run details', 'Usage', and 'Workflow file'. A large red box highlights the 'Jobs' section, which lists a single job named 'build' with a green checkmark icon. On the right, a large red box highlights the detailed history of the 'build' job. The job summary shows it succeeded 5 minutes ago in 24s. Below this, a list of steps is shown, each with a checkmark icon indicating success:

- > Set up job
- > Initialize containers
- > Checkout
- > Install dependencies
- > Lint with flake8
- > Run unit tests with nose
- > Post Checkout
- > Stop containers
- > Complete job

A large, semi-transparent watermark reading 'SAMPLE' is overlaid across the entire image.

3. Task 8: oc-pipelines-oc-final(.png/jpeg)

## PL ci-cd-pipeline

[Details](#)   [YAML](#)   [PipelineRuns](#)   [Parameters](#)   [Metrics](#)

### Pipeline details

**Name**

ci-cd-pipeline

**Namespace**

NS sn-labs-harshsingh15

**Labels**

No labels

[Edit](#) **Tasks**

- T cleanup
- CT git-clone
- T eslint
- T jest-test
- CT buildah
- CT openshift-client

**Workspaces**  
output**Annotations**

0 annotations

**Created at**

29 Jun 2025, 17:22

**Owner**

No owner

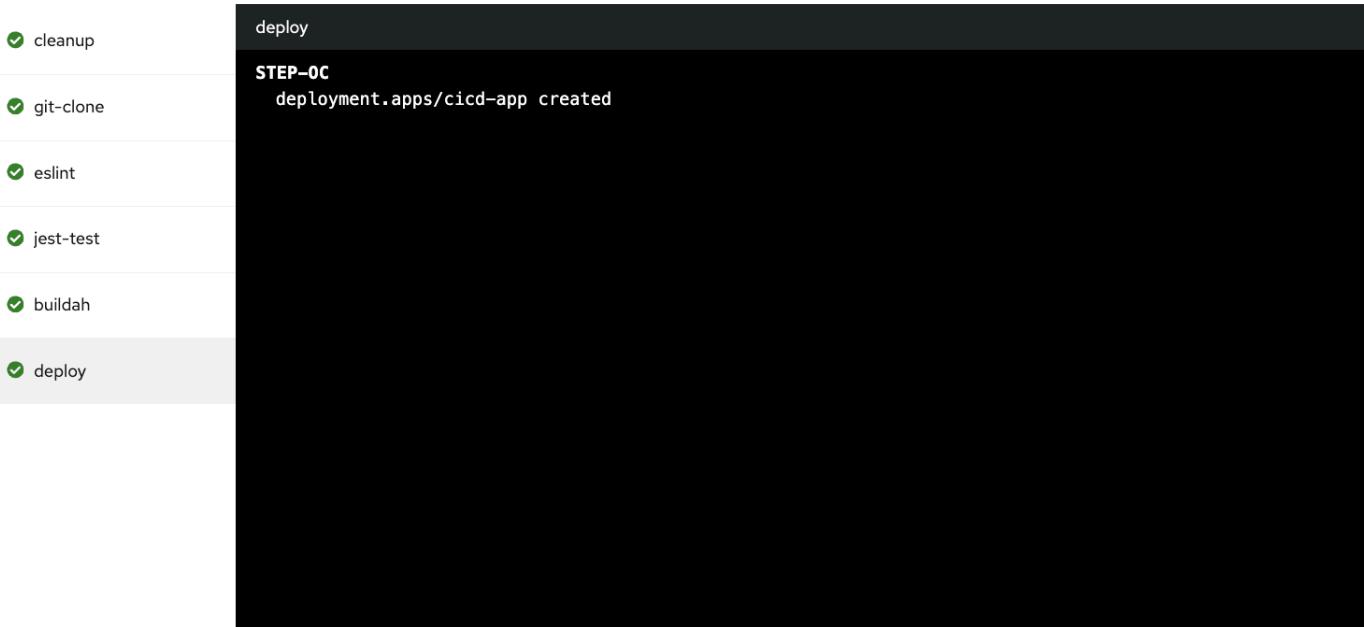
4. Task 9: oc-pipelines-oc-green(.png/jpeg)

Project: sn-labs-harhsingh15 ▾

PipelineRuns > PipelineRun details

PLR ci-cd-pipeline-vxt6mz ✓ Succeeded

Details YAML TaskRuns Parameters Logs Events

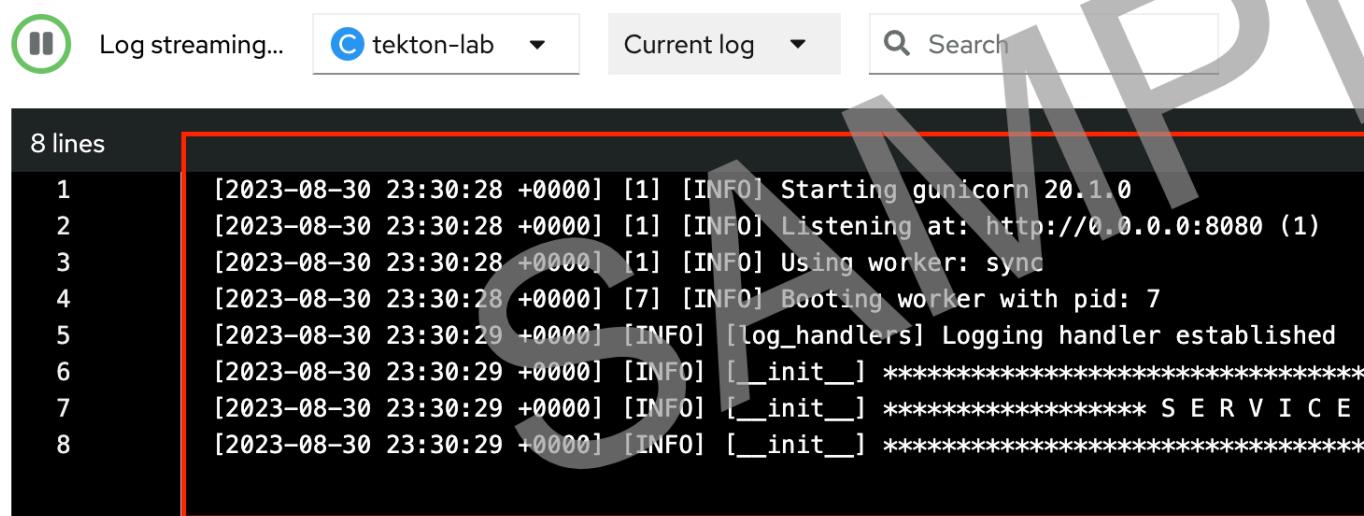


5. Task 10: oc-pipelines-app-logs(.png/jpeg)

Pods > Pod details

P cicd-app-7b7cf6b5d5-h4mx5 ⚡ Running

Details YAML Environment Logs Events Terminal



## Conclusion

Congratulations on completing the CI/CD Final Project. Now you understand how to create continuous integration and continuous deployment pipelines with best practices in mind.

## Next Steps

Incorporate these new practices into your projects at home and work. Write the test cases for the code you "wish you had," then write the code to make those tests pass. Describe the behavior of your system from the outside in and then prove that it behaves that way by automating those tests with Cucumber.js.

## Author(s)

Harsh Singh

