Build an Automated CD DevOps Pipeline Using Tekton and OpenShift

Estimated time needed: 90 minutes

Welcome to the **Build an Automated CD DevOps Pipeline Using Tekton and OpenShift** hands-on lab. In this lab, you will create a parameterized Tekton pipeline that will allow you to reuse your pipeline-as-code artifacts, build your application, and deploy it to Kubernetes/OpenShift.

If you forget how any of these exercises work, refer to module 3 of the course Continuous Integration and Continuous Delivery (CI/CD) for detailed instructions on how to complete these tasks.

Objectives

In this lab, you will:

- Check out your capstone project code created in earlier modules
- Create a workspace and add a cd-pipeline in the workspace
- Create a lint task using flake8
- Create a test task using the Nose test runner
- Create a build task to build a container image
- Create a deploy task to deploy to an OpenShift cluster
- Apply the changes and run the cd-pipeline

Note: Important Security Information

Welcome to the Cloud IDE with OpenShift. This is where all your development will take place. It has all the tools you will need to use Docker for deploying a PostgreSQL database.

It is important to understand that the lab environment is **ephemeral**. It only lives for a short while and then it will be destroyed. This makes it imperative that you push all changes made to your own GitHub repository so that it can be recreated in a new lab environment any time it is needed.

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

Your Task

1. If you haven't generated 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. The environment may be recreated at any time so you may find that you have to perform the **Initialize Development Environment** each time the environment is created.

Note on Screenshots

Throughout this lab, you will be prompted to take screenshots and save them on your device. You will need these screenshots to either answer graded quiz questions or upload as your submission for peer review at the end of this course. Your screenshot must have either the .jpg or .png extension.

To take screenshots, you can use various free screen-capture tools or your operating system's shortcut keys. For example:

- **Mac:** you can use `Shift + Command + 3` (⇧ + ⌘ + 3) on your keyboard to capture your entire screen, or `Shift + Command + 4` (⇧ + ⌘ + 4) to capture a window or area. They will be saved as a file on your Desktop.

- **Windows:** you can capture your active window by pressing `Alt + Print Screen` on your keyboard. This command copies an image of your active window to the clipboard. Next, open an image editor, paste the image from your clipboard to the image editor, and save the image.

Initialize Development Environment

Because the Cloud IDE with OpenShift environment is ephemeral, it may be deleted at any time. The next time you come into the lab, a new environment may be created. Unfortunately, this means that you will need to initialize your development environment every time it is recreated.

This shouldn’t happen too often as the environment can last for several days at a time but when it is removed, 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:

```
1. 1
2. 2
3. 3
4. 4
```

```
1. git clone https://github.com/{your_github_account}/devops-capstone-project.git
2. cd devops-capstone-project
3. bash ./bin/setup.sh
4. exit
```

Copied!

Now, let's discuss each of 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 that contains the name of your GitHub account.

   Note: Substitute your real GitHub account for the `{your_github_account}` place holder below:

   ```
   1. 1
   1. export GITHUB_ACCOUNT={your_github_account}
   ```

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

   ```
   1. 1
   2. 2
   3. 3
   1. git clone https://github.com/$GITHUB_ACCOUNT/devops-capstone-project.git
   2. cd devops-capstone-project
   3. bash ./bin/setup.sh
   ```

   Copied! Executed!

   You should see the follow at the end of the setup execution:
4. Finally, use the `exit` command to close the current terminal. The environment won't be fully active until you open a new terminal in the next step.

   ```
   1. 1
   1. exit
   ```

   Copied! Executed!

Validate

In order to validate that your environment is working correctly, you must open a new terminal because the Python virtual environment will only activate when a new terminal is created. You should have ended the previous task by 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 python` command:

   Your prompt should look like this:

   Check which Python you are using:

```
1. 1
1. which python
```

Copied! Executed!

You should get back:

Check the Python version:

```
1. 1
1. python --version
```

Copied! Executed!

You should get back some patch level of Python 3.9:

This completes the setup of the development environment. Anytime your environment is recreated, you will need to follow this procedure.

You are now ready to start working.

Exercise 1: Pick Up the Next Story

The first thing you need to do is to go to your Zenhub kanban board in GitHub to get a story to work on. Take the first story from the top of the Sprint Backlog, move it to In Progress, assign it to yourself, and read the contents.

Your Task

1. Go to your kanban board and take the next story from the top of the `Sprint Backlog`. It should be titled: "*Create a CD pipeline to automate deployment to Kubernetes*".
2. Move the story to `In Progress`.
3. Open the story and assign it to *yourself*.
4. Read the contents of the story.

Results

The story should look like this:

Create a CD pipeline to automate deployment to Kubernetes

As a developer
I need to create a CD pipeline to automate deployment to Kubernetes
So that developers aren't wasting their time doing it manually

Assumptions

- Use Tekton to define the pipeline
- It should clone, lint, test, build, and deploy the service
- Deployment should be to OpenShift
- It can use a manual trigger for this MVP

Acceptance Criteria

```
1. 1
2. 2
3. 3
```

```
1. Given the CD pipeline has been created
2. When I trigger the pipeline run
3. Then I should see the accounts service deployed to OpenShift
```

Copied!

You are now ready to begin working on your story.

Exercise 2: Overview and Setup

In reading your story, you see that the assumptions state the following:

Assumptions

- Use Tekton to define the pipeline
- It should clone, lint, test, build, and deploy the service
- Deployment should be to OpenShift
- It can use a manual trigger for this MVP

It looks like you will need to create a pipeline task for each of these:

Task Name	Build After	Message
clone	-	Cloning the repo $(params.repo-url)
lint	clone	Calling Flake8 linter
tests	clone	Running unit tests with PyUnit
build	lint, tests	Building image for $(params.repo-url)
deploy	build	Deploying $(params.branch) branch of $(params.repo-url)

Luckily, the developer that worked on this before you started to create a pipeline but didn't get very far. They did, however, create a starter **Pipeline** in `pipeline.yaml` with a `clone` task, and defined a **PersistentVolumeClaim** in the `pvc.yaml` file to define a workspace.

Let's see if what we have already works.

Your Task

1. Change into your project directory: `cd devops-capstone-project`.
2. Create a new branch called `cd-pipeline` to work on in the development environment.
3. Run `nosetests` and make sure that all of the test cases are passing. Fix any that fail before proceeding.
4. Use the `oc create` command on the `tekton/pvc.yaml` file to create a workspace for your pipeline to use.

   ▼ Click here for the answer.

   ```
   1. 1
   1. oc create -f tekton/pvc.yaml
   ```
   Copied! Executed!

5. Use the `oc apply` command on the `tekton/tasks.yaml` file to create the tasks for your pipeline to use.

   ▼ Click here for the answer.

   ```
   1. 1
   1. oc apply -f tekton/tasks.yaml
   ```
   Copied! Executed!

6. Use the `oc apply` command on the `tekton/pipeline.yaml` file to create the initial pipeline.

   ▼ Click here for the answer.

   ```
   1. 1
   1. oc apply -f tekton/pipeline.yaml
   ```
   Copied! Executed!

7. If you edit the `pipeline.yaml` file, you will see that the `clone` task requires the `git-clone` task from Tekton Hub. Use the `tkn hub install task` command to install that now.

   ▼ Click here for the answer.

   ```
   1. 1
   1. tkn hub install task git-clone
   ```
   Copied! Executed!

Note: If the above command returns a error due to Tekton Version mismatch, please run the below command to fix this.

▼ Click here for the hint.

```
1
```

```
    kubectl apply -f https://raw.githubusercontent.com/tektoncd/catalog/main/task/git-clone/0.9/git-clone.yaml
```

Copied! Executed!

1. Use the `tkn pipeline start` command to start the pipeline and make sure the initial pipeline definitions work.

```
1
2
3
4
5
6
tkn pipeline start cd-pipeline \
    -p repo-url="https://github.com/$GITHUB_ACCOUNT/devops-capstone-project.git" \
    -p branch="main" \
    -w name=pipeline-workspace,claimName=pipelinerun-pvc \
    -s pipeline \
    --showlog
```

Copied! Executed!

Results

You should see the following output that proves that the initial pipeline is working. Now it is safe to start adding more tasks to it.

Just to be sure it ran successfully, use `tkn pipelinerun ls` and check the status column, which should show `Succeeded`.

```
1
```

```
tkn pipelinerun ls
```

Copied! Executed!

It should return:

Exercise 3: Create the Lint Task

The first task that you need to add is the lint task. You are going to use flake8 to lint your code. Luckily, Tekton Hub has a flake8 task that you can install and use:

You can use the following button to open the `pipeline.yaml` file to start editing it:

Open **pipeline.yaml** in IDE

Your Task

1. Use the `tkn hub install` command to install the `task` named `flake8` into your Kubernetes namespace.

   You should see output similar to this:

▼ Click here for the answer.

```
1
tkn hub install task flake8
```

Copied! Executed!

2. Now you will modify the `pipeline.yaml` file to use the new flake8 task. Start by copying and pasting the existing `clone` task and change the name to `lint`.

3. In reading the documentation for the flake8 task, you notice that it requires a workspace named `source`. You should change the workspace name to match. You also need to change the `taskRef` to `flake8`.

▼ Click here for a hint.

```
1
2
3
4
5
6
```

```
    - name: {name here}
      workspaces:
        - name: {workspace name here}
          workspace: pipeline-workspace
      taskRef:
        name: {task name here}
```

Copied!

4. You now need to set up the parameters. The flake8 task takes an optional `image` parameter that you will use, and the `args` parameter for flake8 to run. Change the existing parameters to use `python:3.9-slim` as the image, and pass the following `args` parameters to flake8: `--count --max-complexity=10 --max-line-length=127 --statistics`.

   ▼ Click here for a hint.
   Note: Replace "..." with each parameter you want to pass.

```
1
2
3
4
5
6
7
8
9
10
11
    - name: {name here}
      workspaces:
        - name: {workspace name here}
          workspace: pipeline-workspace
      taskRef:
        name: {task name here}
      params:
      - name: image
        value: {image name here}
      - name: args
        value: ["...","...","...","..."]
```

Copied!

5. Finally, you need to define when it should run. Add the `runAfter:` tag and indicate that it should run after the previous `clone` task.

   ▼ Click here for a hint.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
    - name: {name here}
      workspaces:
        - name: {workspace name here}
          workspace: pipeline-workspace
      taskRef:
        name: {task name here}
      params:
      - name: image
        value: {image name here}
      - name: args
        value: {arguments here}
      runAfter:
        - {task before}
```

Copied!

Check Your Answer

▼ Click here for the answer.

```
1
2
3
4
5
6
```

```
7
8
9
10
11
12
13
```

```
    - name: lint
      workspaces:
        - name: source
          workspace: pipeline-workspace
      taskRef:
        name: flake8
      params:
      - name: image
        value: "python:3.9-slim"
      - name: args
        value: ["--count","--max-complexity=10","--max-line-length=127","--statistics"]
      runAfter:
        - clone
```

Copied!

Start the Pipeline

Now it's time run the pipeline and make sure that the lint task works.

Apply the changes:

```
1
```

```
oc apply -f tekton/pipeline.yaml
```

Copied! Executed!

Start the pipeline:

```
1
2
3
4
5
6
```

```
tkn pipeline start cd-pipeline \
       -p repo-url="https://github.com/$GITHUB_ACCOUNT/devops-capstone-project.git" \
    -p branch="main" \
       -w name=pipeline-workspace,claimName=pipelinerun-pvc \
    -s pipeline \
    --showlog
```

Copied! Executed!

Check the Run Status

It may take a long time for the pipeline to run. The logs should display on the screen but if they don't, you can see the pipeline run status by opening a new terminal and listing the pipeline runs with:

```
1
```

```
tkn pipelinerun ls
```

Copied! Executed!

You can check the logs of the last run with:

```
1
```

```
tkn pipelinerun logs --last
```

Copied! Executed!

Commit Your Changes

Becasue the Cloud IDE is ephemeral, it may timeout and you will lose all of your work. It is strongly recommended that you commit your changes and push them to your remote branch so that you can pull them from GitHub if needed.

```
1. 1
2. 2
```

```
1. git commit -am 'added link task'
2. git push --set-upstream origin cd-pipeline
```

Copied! Executed!

Now you are ready to move on to the next excercise.

Exercise 4: Create a Test Task

The next stage of the pipeline you need to create is a test task. In your development environment, you use Nose to run your tests using the `nosetests` command. Unfortunately, there are no tasks in the **Tekton Hub** for nosetests, so you will write your own.

In this exercise, you will update the `tasks.yaml` file adding a new task called `nose` that uses the shared workspace for the pipeline and runs nosetests in a python:3.9-slim image as a shell script as seen in the CI/CD course video.

Here is a bash script to install the Python requirements and run the nosetests. You can use this as the shell script in your new task:

```
1. 1
2. 2
3. 3
4. 4
5. 5
```

```
1. #!/bin/bash
2. set -e
3. python -m pip install --upgrade pip wheel
4. pip install -qr requirements.txt
5. nosetests -v --with-spec --spec-color
```

Copied!

You can use the following button to open the `tasks.yaml` file to start editing it:

Open **tasks.yaml** in IDE

Your Task:

1. First, create a new task in the `tasks.yaml` file. Start by copying and pasting the existing `echo` task and change the name to `nose`.

   Remember, each new task must be separated using three dashes — on a separate line.

   ▼ Click here for a hint.
   Replace the `{here}` placeholders with the real values:

   ```
   1. 1
   2. 2
   3. 3
   4. 4
   5. 5
   6. 6
   1. ---
   2. apiVersion: tekton.dev/v1beta1
   3. kind: Task
   4. metadata:
   5.   name: {task name here}
   6. spec:
   ```
   Copied!

2. Next, it's a good idea to add a description to yor task so that others who may use it know what it does. Change the `description:` tag to be:
   `This task will run nosetests on the provided input.`

   ▼ Click here for a hint.
   Replace the `{here}` placeholders with the real values:

   ```
   1. 1
   2. 2
   3. 3
   4. 4
   5. 5
   6. 6
   7. 7
   1. ---
   2. apiVersion: tekton.dev/v1beta1
   ```

```
3. kind: Task
4. metadata:
5.   name: {task name here}
6. spec:
7.   description: {description here}
```

Copied!

3. Next, you need to include the workspace that has the code that you want to test. Since flake8 uses the name `source`, you can use that for consistency. Add a workspace named `source`.

▼ Click here for a hint.
Replace the `{here}` placeholders with the real values:

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
1. ---
2. apiVersion: tekton.dev/v1beta1
3. kind: Task
4. metadata:
5.   name: {task name here}
6. spec:
7.   description: {description here}
8.   workspaces:
9.     - name: {workspace name here}
```

Copied!

4. It might be a good idea to allow the passing in of different arguments to nosetests, so create a parameter called `args` just like the flake8 task has, and give it a `description:`, make the `type:` a **string**, and a `default:` with the verbose flag `-v` as the default.

▼ Click here for a hint.
Replace the `{here}` placeholders with the real values:

```
 1. 1
 2. 2
 3. 3
 4. 4
 5. 5
 6. 6
 7. 7
 8. 8
 9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
 1. ---
 2. apiVersion: tekton.dev/v1beta1
 3. kind: Task
 4. metadata:
 5.   name: {task name here}
 6. spec:
 7.   description: {description here}
 8.   workspaces:
 9.     - name: {workspace name here}
10.   params:
11.     - name: {argument name here}
12.       description: Arguments to pass to nose
13.       type: {type here}
14.       default: {default args here}
```

Copied!

5. Quite often, tests require a test database. It might be a good idea to allow the passing in of a database url. Add a new parameter called `database_uri` just like the args parameter. Give it a `description:`, make the `type:` a **string**, and a `default:` with SQLite (`"sqlite:///test.db"`) in case they don't have a separate database server.

▼ Click here for a hint.
Replace the `{here}` placeholders with the real values:

```
1. 1
2. 2
3. 3
4. 4
```

```
 5. 5
 6. 6
 7. 7
 8. 8
 9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
 1. ---
 2. apiVersion: tekton.dev/v1beta1
 3. kind: Task
 4. metadata:
 5.   name: {task name here}
 6. spec:
 7.   description: {description here}
 8.   workspaces:
 9.     - name: {workspace name here}
10.   params:
11.     - name: {argument name here}
12.       description: Arguments to pass to nose
13.       type: {type here}
14.       default: {default args here}
15.     - name: {database uri name here}
16.       description: Database connection string
17.       type: {type here}
18.       default: {default database uri here}
```

Copied!

6. Finally, you need to specify the `steps:`, and there is only one. Give it the name `nosetests`.

   ▶ Click here for a hint.

7. Have it run in a `python:3.9-slim` image.

   ▶ Click here for a hint.

8. Also, specify `workingDir` as the path to the workspace you defined (i.e., `$(workspaces.source.path)`).

   ▶ Click here for a hint.

9. Next, add an environment variable called `DATABASE_URI` that gets its value from the new parameter you just created: `$(params.database_uri)`.

   ▶ Click here for a hint.

10. Then, paste the script from above in the `script:` parameter and substitute `$(params.args)` for the parameters passed into `nosetests` instead of the hard-coded ones.

    ▶ Click here for a hint.

Check Your Answer

Your new task should look similar to this:

▼ Click here for the answer.

```
 1. 1
 2. 2
 3. 3
 4. 4
 5. 5
 6. 6
 7. 7
 8. 8
 9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
```

```
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
27. 27
28. 28
29. 29
30. 30
31. 31
32. 32
33. 33
34. 34
35. 35
```

```
 1. ---
 2. apiVersion: tekton.dev/v1beta1
 3. kind: Task
 4. metadata:
 5.   name: nose
 6. spec:
 7.   description: This task will run nosetests on the provided input.
 8.   workspaces:
 9.     - name: source
10.   params:
11.     - name: args
12.       description: Arguments to pass to nose
13.       type: string
14.       default: "-v"
15.     - name: database_uri
16.       description: Database connection string
17.       type: string
18.       default: "sqlite:///test.db"
19.   steps:
20.     - name: nosetests
21.       image: python:3.9-slim
22.       workingDir: $(workspaces.source.path)
23.       env:
24.         - name: DATABASE_URI
25.           value: $(params.database_uri)
26.       script: |
27.         #!/bin/bash
28.         set -e
29.
30.         echo "***** Installing dependencies *****"
31.         python -m pip install --upgrade pip wheel
32.         pip install -qr requirements.txt
33.
34.         echo "***** Running nosetests with: $(params.args)"
35.         nosetests $(params.args)
```

Copied!

Apply Your Changes

Apply these changes to your cluster:

```
1. 1
```

```
1. oc apply -f tekton/tasks.yaml
```

Copied! Executed!

You should see the following output:

Commit Your Changes

Becasue the Cloud IDE is ephemeral, it may timeout and you will lose all of your work. It is strongly recommended that you commit your changes and push them to your remote branch so that you can pull them from GitHub if needed.

```
1. 1
2. 2
```

```
1. git commit -am 'added nose task'
2. git push
```

Copied! Executed!

Now you are ready to move on to the next exercise.

Exercise 5: Add the Test Task to the Pipeline

Now it's time to update the pipeline to add the test task that uses the new `nose` task that you have created and then run the pipeline to see it work.

You can use the following button to open the `pipeline.yaml` file to start editing it:

Open **pipeline.yaml** in IDE

Your Task

1. Start by editing the `pipeline.yaml` file to add the new test task. Start by copying and pasting the `lint` task and change the name to `tests`.

   ▶ Click here for a hint.

2. Leave the workspace named `source` and change the `taskRef:` from lint to reference your new `nose` task.

   ▶ Click here for a hint.

3. Change the `image` parameter to `database_uri` with a value of `sqlite:///test.db`.

   ▶ Click here for a hint.

4. Change the value of the existing `args` parameter to specify the arguments to pass to the tests as `"-v --with-spec --spec-color"`.

   ▶ Click here for a hint.

5. Finally, specify that this task should `runAfter:` the `clone` step.

   > Notice that you are running lint and tests in parallel because they do not depend on each other. This will make the pipeline run faster.

   ▶ Click here for a hint.

Check Your Answer

Your new step should look similar to this:

▼ Click here for the answer.

```
 1. 1
 2. 2
 3. 3
 4. 4
 5. 5
 6. 6
 7. 7
 8. 8
 9. 9
10. 10
11. 11
12. 12
13. 13
```

```
 1.     - name: tests
 2.       workspaces:
 3.         - name: source
 4.           workspace: pipeline-workspace
 5.       taskRef:
 6.         name: nose
 7.       params:
 8.       - name: database_uri
 9.         value: "sqlite:///test.db"
10.       - name: args
11.         value: "-v --with-spec --spec-color"
12.       runAfter:
13.         - clone
```

Copied!

Commit Your Changes

Because the Cloud IDE is ephemeral, it may timeout and you will lose all of your work. It is strongly recommended that you commit your changes and push them to your remote branch so that you can pull them from GitHub if needed.

```
1. 1
2. 2
```

```
1. git commit -am 'added test pipeline task'
2. git push
```

Copied! Executed!

Now you are ready to move on to the next exercise.

Exercise 6: Start the Pipeline

Now it's time run the pipeline and make sure that the tests task works. The tests task will need a PostgreSQL database to run against. If it isn't running in your OpenShift cluster from the last lab, you must provision it again.

Your Task

1. Use the `oc get svc postgresql` command to see if the PostgreSQL service is defined.

   ```
   1. 1
   1. oc get svc postgresql
   ```
   Copied! Executed!

   If it is not found, use the `oc new-app` command to deploy the `postgresql-ephemeral` template.

   ```
   1. 1
   1. oc new-app postgresql-ephemeral
   ```
   Copied! Executed!

2. Use `oc get pods` to make sure that the postgres service is defined and the postgres pod is running.

   ```
   1. 1
   1. oc get pods
   ```
   Copied! Executed!

   You should see the following:

3. You are now ready to apply the changes to your pipeline.

   ```
   1. 1
   1. oc apply -f tekton/pipeline.yaml
   ```
   Copied! Executed!

   You should see the message: pipeline.tekton.dev/cd-pipeline configured

Start the Pipeline

Make sure that the `GITHUB_ACCOUNT` environment variable is set to the name of your GitHub account, and use the `tkn pipeline start` command to run the pipeline.

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
```

```
1. tkn pipeline start cd-pipeline \
2.        -p repo-url="https://github.com/$GITHUB_ACCOUNT/devops-capstone-project.git" \
3.     -p branch="main" \
4.        -w name=pipeline-workspace,claimName=pipelinerun-pvc \
5.     -s pipeline \
6.     --showlog
```

Copied! Executed!

Note that because you specified that `lint` and `tests` can run in parallel, you will see the log output of those two tasks intermixed.

Exercise 7: Create a Build Task

Before you can deploy your application, you need to build a Docker image and push it to an image registry. You search **Tekton Hub** for the word "build" and you see there is a task called `buildah` that will build images so you decide to use it in your pipeline to build your code.

Instead of installing it yourself, you first check the ClusterTasks in your cluster to see if it already exists. Luckily, the OpenShift environment you are using already has buildah installed as a **ClusterTask**. A ClusterTask is installed cluster-wide by an administrator, and anyone can use it in their pipelines without having to install it themselves.

You can always check what ClusterTasks are installed with this command:

```
1
```

```
tkn clustertask ls
```

Copied! Executed!

Now, you will update the `pipeline.yaml` file to use the new `buildah` task. You can use the following button to open the `pipeline.yaml` file to start editing it:

Open **pipeline.yaml** in IDE

Your Task

1. Start by editing the `pipeline.yaml` file to add the new build task. Start by copying and pasting the `tests` task and changing the name to `build`.

   ▼ Click here for a hint.

   ```
   1
       - name: {name here}
   ```
   Copied!

2. The workspace name `source` from the tests task is fine, but the `taskRef` needs to be changed to `buildah`, and you must add an additional line below it to indicate that it has a `kind:` of `ClusterTask`.

   ▼ Click here for a hint.

   ```
   1
   2
   3
   4
   5
   6
   7
       - name: {name here}
         workspaces:
           - name: source
             workspace: pipeline-workspace
         taskRef:
           name: {task name here}
           kind: {kind here}
   ```
   Copied!

3. Delete the `database_uri` parameter. It is not needed for this task.

4. Next, change the `args` parameter to `IMAGE` and set the value to `"$(params.build-image)"` to pick up the `build-image` name from the pipeline parameters.

   ▼ Click here for a hint.

   ```
   1
   2
   3
   4
   5
   6
   7
   8
   9
   10
       - name: {name here}
         workspaces:
           - name: source
             workspace: pipeline-workspace
         taskRef:
   ```

```
      name: {task name here}
      kind: {kind here}
    params:
    - name: {parameter name here}
      value: (params value here}
```

Copied!

5. Now, you need to add a parameter named `build-image` to the existing list of parameters at the top of the pipeline under `spec.params`.

▼ Click here for a hint.

```
1
2
3
4
5
6
spec:
  params:
    - name: repo-url
    - name: branch
      default: main
    - name: {new parameter name here}
```

Copied!

6. Finally, you must specify what task or tasks this should run after. Because both `tests` and `lint` ran in parallel, you want to wait until both complete before bulding the image. Change the `taskRef` to specify both `tests` and `lint`.

▼ Click here for a hint.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
    - name: {name here}
      workspaces:
        - name: source
          workspace: pipeline-workspace
      taskRef:
        name: {task name here}
        kind: {kind here}
      params:
      - name: {parameter name here}
        value: (params value here}
      runAfter:
        - {first task here}
        - {second task here}
```

Copied!

Check Your Answer

Check that your step looks like this:

▼ Click here for the answer.

```
1
2
3
4
```

```
spec:
  params:
    ... other parameters ...
    - name: build-image
```

Copied!

```
1
2
3
4
5
```

```
6
7
8
9
10
11
12
13
```

```
    - name: build
      workspaces:
        - name: source
          workspace: pipeline-workspace
      taskRef:
        name: buildah
        kind: ClusterTask
      params:
      - name: IMAGE
        value: "$(params.build-image)"
      runAfter:
        - tests
        - lint
```

Copied!

Apply the Changes

You are now ready to apply the changes to your pipeline.

```
1
```

```
oc apply -f tekton/pipeline.yaml
```

Copied! Executed!

You should see the message: pipeline.tekton.dev/cd-pipeline configured

Commit Your Changes

Because the Cloud IDE is ephemeral, it may timeout and you will lose all of your work. It is strongly recommended that you commit your changes and push them to your remote branch so that you can pull them from GitHub if needed.

```
1
2
```

```
git commit -am 'added build task'
git push
```

Copied! Executed!

Start the Pipeline

When you start the pipeline, you need to pass in the build-image parameter, which is the name of the image to build.

This will be different for every learner that uses this lab. Here is the format:

```
1
```

```
image-registry.openshift-image-registry.svc:5000/$SN_ICR_NAMESPACE/accounts:latest
```

Copied!

Notice the variable `$SN_ICR_NAMESPACE` in the image name. This is automatically set to point to your container namespace. There is nothing for you to do here.

Make sure that the `GITHUB_ACCOUNT` environment variable is set to the name of your GitHub account, and use the `tkn pipeline start` command to run the pipeline.

```
1
2
3
4
5
6
7
```

```
tkn pipeline start cd-pipeline \
```

```
    -p repo-url="https://github.com/$GITHUB_ACCOUNT/devops-capstone-project.git" \
    -p branch=main \
    -p build-image=image-registry.openshift-image-registry.svc:5000/$SN_ICR_NAMESPACE/accounts:1 \
    -w name=pipeline-workspace,claimName=pipelinerun-pvc \
    -s pipeline \
    --showlog
```

Copied! Executed!

Check that everything ran to completion (this may take a while).

Exercise 8: Add the Deploy Task to the Pipeline

You are now at the deploy step, which is the last step in your CD pipeline. For this step, you will use the OpenShift client to deploy your Docker image to an OpenShift cluster.

Knowing that you want to deploy to OpenShift, you search Tekton Hub for "openshift" and you see there is a task called `openshift-client` that will execute OpenShift commands on your cluster. You decide to use the openshift-client task in your pipeline to deploy your image.

First, you need to update the pipeline.yaml file to reference the new openshift-client ClusterTask that you want to use in the deploy pipeline. Then you will need to update your `deploy/deployment.yaml` file to substitute the name of the image to deploy.

In the previous steps, you simply changed the name of the reference to the task, but since the openshift-client task is installed as a **ClusterTask**, you need to add the statement `kind: ClusterTask` under the name so that Tekton knows to look for a **ClusterTask** and not a regular **Task**.

Your Task

1. Start by editing the `pipeline.yaml` file to add the new deploy task. Start by copying and pasting the `build` task and changing the name to `deploy`.

   ▼ Click here for a hint.

   ```
   1
       - name: {name here}
   ```
   Copied!

2. Next, change the `name:` of the `workspaces:` to `manifest-dir` as specified in the documentation of the `openshift-client` on Tekton Hub.

   ▼ Click here for a hint.

   ```
   1
   2
   3
   4
       - name: {name here}
         workspaces:
           - name: {workspace name here}
             workspace: pipeline-workspace
   ```
   Copied!

3. Then, change the `taskRef` from buildah to `openshift-client` and leave the `kind:` as `ClusterTask` to indicate that this is a ClusterTask.

   ▼ Click here for a hint.

   ```
   1
   2
   3
   4
   5
   6
   7
       - name: {name here}
         workspaces:
           - name: {workspace name here}
             workspace: pipeline-workspace
         taskRef:
           name: {task name here}
           kind: ClusterTask
   ```
   Copied!

4. Now, change the `IMAGE` parameter to the name `SCRIPT` and specify the `oc` command to apply your Kubernetes manifests from the previous lab. Before you do, you will need to modify your `deployment.yaml` file to have an image name placeholder and use `sed` to change it to the image name for the pipeline before execution.

   Here is the script to use (you will modify the deployment.yaml file in a later step):

```
1
2
3
4
5
6
```

```
echo "Updating manifest..."
sed -i "s|IMAGE_NAME_HERE|$(params.build-image)|g" deploy/deployment.yaml
cat deploy/deployment.yaml
echo "Deploying to OpenShift..."
oc apply -f deploy/
oc get pods -l app=accounts
```

Copied!

▶ Click here for a hint.

5. Finally, specify that this task should `runAfter:` the `build` task.

▶ Click here for a hint.

Check Your Answer

If you changed everything correctly, the full deploy task in the pipeline should look like this:

▼ Click here for the answer.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
```

```
    - name: deploy
      workspaces:
        - name: manifest-dir
          workspace: pipeline-workspace
      taskRef:
        name: openshift-client
        kind: ClusterTask
      params:
      - name: SCRIPT
        value: |
          echo "Updating manifest..."
          sed -i "s|IMAGE_NAME_HERE|$(params.build-image)|g" deploy/deployment.yaml
          cat deploy/deployment.yaml
          echo "Deploying to OpenShift..."
          oc apply -f deploy/
          oc get pods -l app=accounts
      runAfter:
        - build
```

Copied!

Apply the Pipeline

Apply the same changes you just made to `pipeline.yaml` to your cluster:

```
1
```

```
kubectl apply -f tekton/pipeline.yaml
```

Copied! Executed!

If it applied correctly, you can commit your changes. That doesn't mean it will run without errors, but it's a good idea to commit what you have before you start debugging.

Update Your Deployment Manifest

There is one last thing that you need to do before you can start the pipeline. You must edit your `deploy/deployment.yaml` file to specify a placeholder for the name of the image that you just built in the `build` step.

Open **deployment.yaml** in IDE

Find the `image:` tag and change its value to `IMAGE_NAME_HERE`.

For example:

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
```

```
1. spec:
2.   template:
3.     spec:
4.       containers:
5.       - image: IMAGE_NAME_HERE
6.         name: accounts
```

Copied!

Commit Your Changes

Because the pipeline is going to clone your Git repository and you changed `deploy/deployment.yaml`, you must push all of your code changes back to your remote branch in GitHub. You will also need to use this branch name when you run the pipeline. If you followed the instructions, the current branch should be: `cd-pipeline`.

```
1. 1
2. 2
```

```
1. git commit -am 'added deploy task'
2. git push
```

Copied! Executed!

Exercise 9: Apply Changes and Run the Pipeline

Now that you have built out the entire pipeline, it is time to run it. If you were in a persistent environment you would set up a webhook in GitHub so that whenever your code changes, the CD pipeline would run after all the tests and checks had passed.

Unfortunately, the lab environnement is ephemeral and the best we can do is start the pipeline manually.

Start the Pipeline

When you start the pipeline, you will need to use the name of the branch that you are working on, which should be `cd-pipeline`.

Now, start the pipeline to see your new deploy task run. Use the Tekton CLI pipeline start command to run the pipeline, passing in the parameters `repo-url`, `branch`, and `build-image` using the `-p` option. Specify the workspace `pipeline-workspace` and persistent volume claim `pipelinerun-pvc` using the `-w option:

Your Task

1. First, make sure that you have set the environment variable `GITHUB_ACCOUNT` to contain the name of your GitHub account.
2. Use the `tkn pipeline start` command to start the pipeline and see if your deploy task works.

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
```

```
1. tkn pipeline start cd-pipeline \
2.     -p repo-url="https://github.com/$GITHUB_ACCOUNT/devops-capstone-project.git" \
3.     -p branch=cd-pipeline \
```

```
    -p build-image=image-registry.openshift-image-registry.svc:5000/$SN_ICR_NAMESPACE/accounts:1 \
    -w name=pipeline-workspace,claimName=pipelinerun-pvc \
    -s pipeline \
    --showlog
```

Copied! Executed!

Check the Deployment

Now, check to see if the deployment is running. Use the `oc` command to check that your deployment is in a running state.

```
1
```

```
oc get all -l app=accounts
```

Copied! Executed!

Evidence

For the evidence, you will capture the logs of the piepline run in a file called `pipelinerun.txt`

1. Open your terminal and issue the command `tkn pipelinerun logs -L` and redirect it to a file called: `pipelinerun.txt`.

```
1
tkn pipelinerun logs -L > pipelinerun.txt
```

Copied! Executed!

2. Open this file in the Cloud IDE editor and copy and paste it into a file of the same name on your computer so that you can upload it as evidence.

Exercise 10: Make a Pull Request

Now that you have a working deployment, it's time to push the Kubernetes manifests up to GitHub and make a pull request. Then, you will merge the request and move your story to Done.

Your Task

1. Use `git status` to make sure that you have committed your changes locally in the development environment.
2. If there are uncommitted changes, commit your changes using the message `Final CD pipeline`.
3. Push your local changes to a remote branch.

   Note: Use your GitHub **Personal Access Token** as your password in the Cloud IDE environment. Also, you may have to configure Git the first time you use it.
4. Make a pull request.
5. Once the test cases pass, merge your pull request.
6. Move your story to the Done column on your kanban board.
7. Pull the last code down to your development environment and delete your old branch.

```
1
2
3
git checkout main
git pull
git branch -d cd-pipeline
```

Copied!

Evidence

For the evidence, you will take a screenshot of your kanban board to show that the story is done.

1. Open your kanban board and take a screenshot of the board with your story in the Done column, saving it as `cd-pipeline-done.jpg` (or `cd-pipeline-done.png`).

Conclusion

Congratulations! You have built a CD pipeline using Tekton to clone, test, lint, build, and deploy your microservice to an OpenShift/Kubernetes cluster using YAML manifests.

Next Steps

You have completed the DevOps Capstone Project. It's now time to upload all of your evidence for the entire project.

Author

Tapas Mandal
John J. Rofrano

Other Contributor(s)

Change Log

Date	Version	Changed by	Change Description
2022-10-13	0.1	Tapas Mandal	Initial version created
2022-10-14	0.2	John Rofrano	Added additional content
2022-11-2	0.3	Beth Larsen	QA pass
2022-11-17	0.4	Lavanya Rajalingam	Updated instructions
2023-03-16	0.5	Lavanya Rajalingam	Updated SN Logo