Building a Tekton Pipeline

Skills Network

Estimated time needed: 40 minutes

Welcome to the hands-on lab for **Building a Tekton Pipeline**. In this lab, you will create a simple Tekton pipeline with one task in Step 1 and then add a parameter to it in Step 4. You will learn best practices for structuring a Tekton pipeline project and how to author Tekton pipelines and tasks so that they are easy to use and parameterize. You will see that Tekton allows you to reuse your pipeline-as-code artifacts, and you will look at practical approaches to publishing your pipeline and task definitions to a Git repository.

Learning Objectives

After completing this lab, you will be able to:

- Create a base pipeline and task to echo a message.
- Apply parameters to the task and pipeline.
- Apply additional parameters to a pipeline to clone a Git repository.

Prerequisites

You will need the following to complete the exercises in this lab:

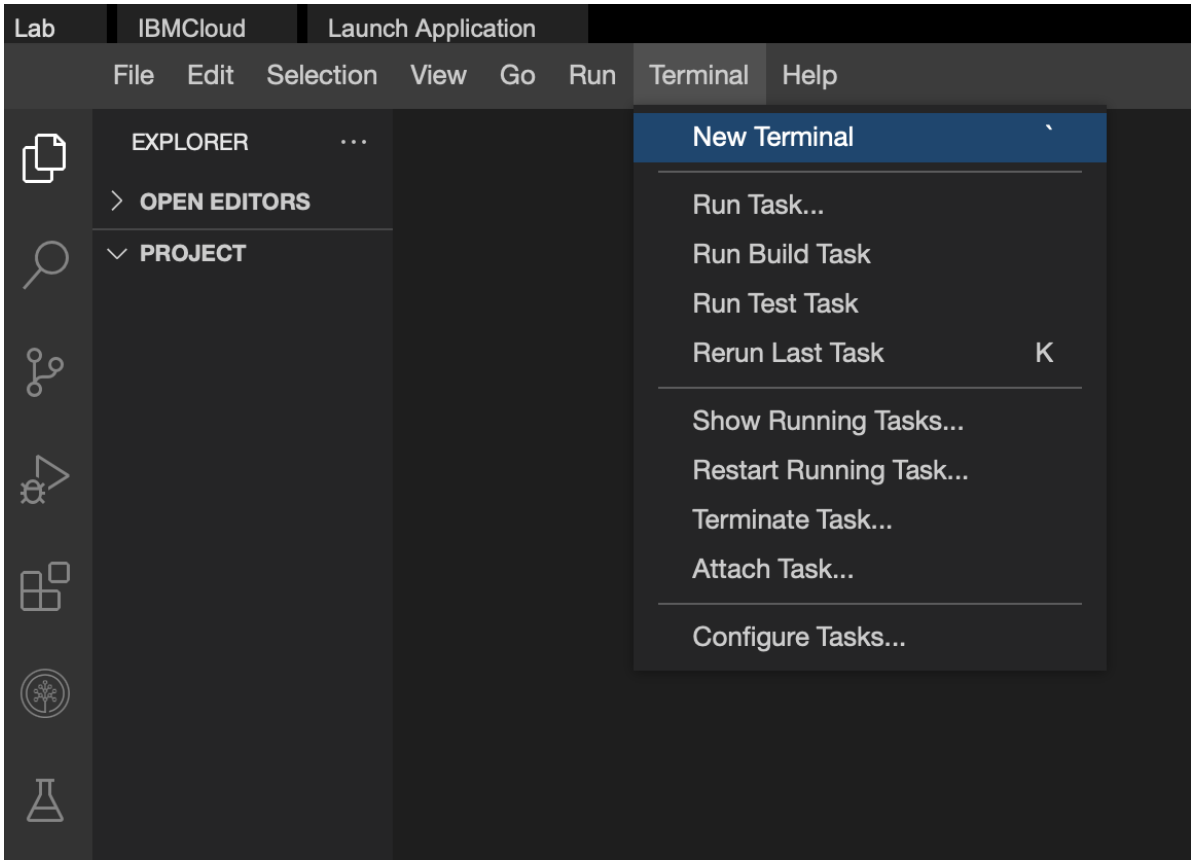
- A basic understanding of YAML
- A GitHub account
- An intermediate-level knowledge of CLIs

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.

```
1. 1
```

```
1. cd /home/project
```

Copied! Executed!

Clone the Code Repo

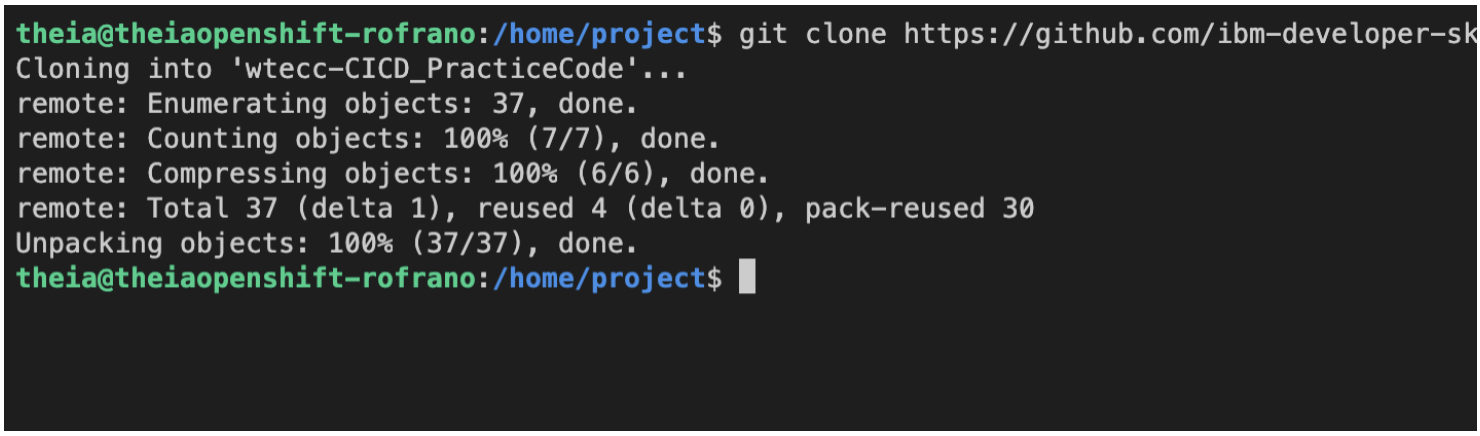
Now, get the code that you need to test. To do this, use the `git clone` command to clone the Git repository:

```
1. 1
```

```
1. git clone https://github.com/ibm-developer-skills-network/wtecc-CICD_PracticeCode.git
```

Copied! Executed!

Your output should look similar to the image below:



Change to the Labs Directory

Once you have cloned the repository, change to the labs directory.

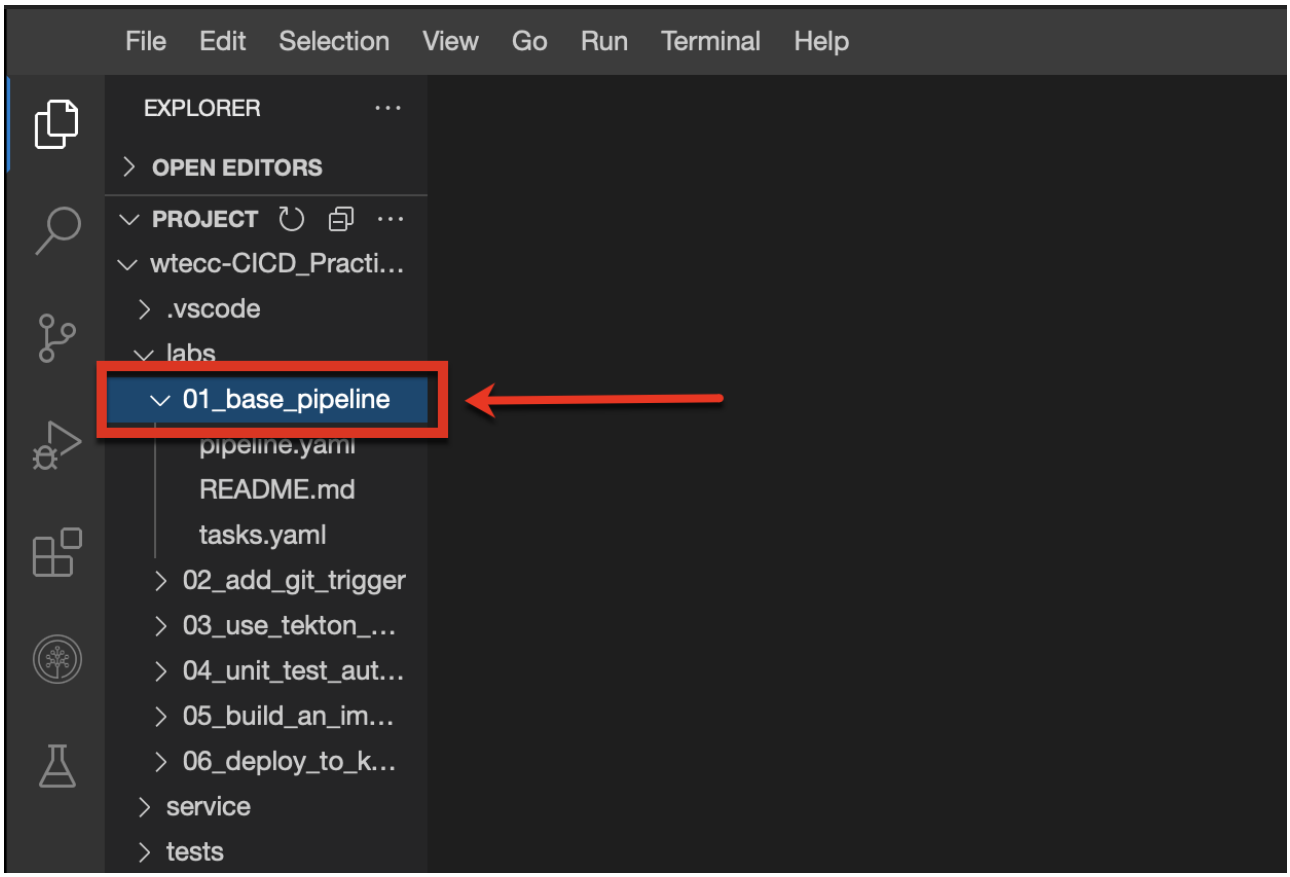
```
1
```

```
cd wtecc-CICD_PracticeCode/labs/01_base_pipeline/
```

Copied! Executed!

Navigate to the Labs Folder

Navigate to the `labs/01_base_pipeline` folder in left explorer panel. All of your work will be completed with the files in this folder.



You are now ready to start the lab.

Optional

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

```
1
```

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

Copied! Executed!

Step 1: Create an echo Task

In true computer programming tradition, the first task you create will echo “Hello World!” to the console.

There is starter code in the `labs/01_base_pipeline` folder for a task and a pipeline. Navigate to this folder in left explorer panel, and open the `tasks.yaml` file to edit it:

Open **tasks.yaml** in IDE

It should look like this:

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

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: <place-name-here>
spec:
  steps:
```

Copied!

Using the same steps that you learned in the video to create the **checkout** task, you will now create a **hello-world** task.

Your Task

1. The first thing you want to do is give the task a good name. Change `<place-name-here>` to `hello-world`.
2. The next thing is to add a step. Remember from the video that steps that run a single command need to include `name`, `image`, `command`, and `args`. Make the name `echo`, use the image `alpine:3`, have the command be `[/bin/echo]` and the args be `["Hello World"]`.

Hint

▼ Click here for a hint.
Add a step under the `steps:` tag. Since there can be multiple steps, each one starts with a dash `-` to identify it as an item in a `yaml` list.

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

```
spec:
  steps:
    - name: {name here}
      image: {image here}
      command: {command here}
      args: {args here}
```

Copied!

Double-check that your work matches the solution below.

Solution

▼ Click here for the answer.

```
1
2
3
4
5
6
7
8
9
10
```

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: hello-world
spec:
  steps:
    - name: echo
      image: alpine:3
      command: [/bin/echo]
      args: ["Hello World!"]
```

Copied!

Apply it to the cluster:

```
1
```

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

Copied! Executed!

Step 2: Create a hello-pipeline Pipeline

Next, you will create a very simple pipeline that only calls the `hello-world` task that you just created. Navigate to this folder in left explorer panel, and open the `pipeline.yaml` file to edit it:

Open **pipeline.yaml** in IDE

It should look like this:

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

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: <place-name-here>
spec:
  tasks:
```

Copied!

Using the same steps that you learned in the video to create the **ci-pipeline** pipeline, you will now create a **hello-pipeline** pipeline.

Your Task

1. The first thing you want to do is give the pipeline a good name. Change `<place-name-here>` to `hello-pipeline`.
2. The next thing is to add a reference to the hello-world task you just created. Remember from the video that task needs a `name:` for the pipeline task, and a `taskRef:`, with a `name:` tag under it set to the name of the task you are referencing. Set the name of the pipeline task to `hello` and the name of the task you are referencing as `hello-world`.

Hint

▼ Click here for a hint.
Add a task under the `tasks:` tag. Since there can be multiple tasks, each one starts with a dash `-` to identify it as an item in a `yaml` list.

```
1
2
3
4
5
```

```
spec:
  tasks:
    - name: {name here}
      taskRef:
        name: {task name here}
```

Copied!

Double-check that your work matches the solution below.

Solution

▼ Click here for the answer.

```
1
2
3
4
5
6
7
8
9
```

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: hello-pipeline
spec:
  tasks:
    - name: hello
      taskRef:
        name: hello-world
```

Copied!

Apply it to the cluster:

```
1
```

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

Copied! Executed!

You are now ready to run your pipeline and see if it works.

Step 3: Run the hello-pipeline

Run the pipeline using the Tekton CLI:

```
1
```

```
tkn pipeline start --showlog hello-pipeline
```

Copied! Executed!

You should see the output:

```
1
2
3
```

```
PipelineRun started: hello-pipeline-run-9vkbb
Waiting for logs to be available...
[hello : echo] Hello World!
```

Copied!

Congratulations! You just ran your first pipeline from a pipeline and task that you created.

Step 4: Add a parameter to the task

Hopefully the hello-world task has given you a sense for how pipelines call tasks. Now it is time to make that task a little more useful by making it print any message that you want, not just “Hello World”.

To do this, you will add a parameter called `message` to the task and use that parameter as the message that it echoes. You will also rename the task to `echo`.

Edit the `tasks.yaml` file to add the parameter to both the input and the echo command:

Open **tasks.yaml** in IDE

Your Task

1. Change the name of the task from `hello-world` to `echo` to more acurately reflect its new functionality, by changing the `name:` in the `metadata:` section.
2. Add a `params:` section to the task with a parameter that has a `name:` of “message”, a `type:` of “string”, and a `description` of “The message to echo”.
3. Change the name of the step from `echo` to `echo-message` to better describe its new functionality.
4. Modify the `args:` tag to use the message parameter you just created.

Hint

▼ Click here for a hint.
Add a `params:` section under the `spec:` tag. Since there can be multiple parameters, each one starts with a dash `-` to identify it as an item in a `yaml` list.

The reference to the message parameter should be `$(params.message)`.

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

```
metadata:
  name: {change the task name}
spec:
  params:
    - name: {name here}
      description: {description here}
      type: {type here}
  steps:
    - name: echo-message
      image: alpine:3
      command: [/bin/echo]
      args: {place parameter reference here}
```

Copied!

Double-check that your work matches the solution below.

Solution

▼ Click here for the answer.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: echo
spec:
  params:
    - name: message
      description: The message to echo
      type: string
  steps:
    - name: echo-message
      image: alpine:3
      command: [/bin/echo]
      args: ["$(params.message)"]
```

Copied!

Apply the new task definition to the cluster:

```
1
```

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

Copied! Executed!

Step 5: Update the hello-pipeline

You now need to update the pipeline to pass the message that you want to send to the `echo` task so that it can echo the message to the console.

Edit the `pipeline.yaml` file to add the parameter:

Open **pipeline.yaml** in IDE

Your Task

1. Add a `params:` section to the pipeline under `spec:`, with a parameter that has a `name:` of “message”.
2. Change the `name:` of the `taskRef:` from `hello-world` to the new `echo` task.
3. Add a `params:` section to the task, with a parameter that has a `name:` of “message” and a `value:` that is a reference to the pipeline parameter for `params.message`.

Hint

▼ Click here for a hint.
Specify the value of message params in tasks as "$(params.message)".

```
1
2
3
4
5
6
7
8
9
10
```

```
spec:
  params:
    - name: {parameter name here}
  tasks:
    - name: hello
      taskRef:
```

```
        name: {change parameter value here}
      params:
        - name: {task parameter name here}
          value: "$(params.message)"
```

Copied!

Double-check that your work matches the solution below.

Solution

▼ Click here for the answer.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: hello-pipeline
spec:
  params:
    - name: message
  tasks:
    - name: hello
      taskRef:
        name: echo
      params:
        - name: message
          value: "$(params.message)"
```

Copied!

Apply it to the cluster:

```
1
```

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

Copied! Executed!

Step 6: Run the message-pipeline

Run the pipeline using the Tekton CLI:

```
1
2
3
```

```
tkn pipeline start hello-pipeline \
    --showlog  \
    -p message="Hello Tekton!"
```

Copied! Executed!

You should see the output:

```
1
2
3
```

```
PipelineRun started: hello-pipeline-run-9qf42
Waiting for logs to be available...
[hello : echo-message] Hello Tekton!
```

Copied!

Congratulations! You just created and ran a pipeline that requires a parameter.

Step 7: Create a checkout Task

In this step, you will combine your knowledge of running a command in a container with your knowledge of passing parameters, to create a task that checks out your code from GitHub as the first step in a CD pipeline.

Create checkout task

You can have multiple definitions in a single yaml file by separating them with three dashes `---` on a single line. In this step, you will add a new task to `tasks.yaml` that uses the `bitnami/git:latest` image to run the `git` command passing in the branch name and URL of the repo you want to clone.

Open the `tasks.yaml` file to create a new task:

Open **tasks.yaml** in IDE

Add three dashes on a separate line:

```
1
```

```
---
```

Copied!

You are now ready to add your new task.

Your Task

Your new task will create a Tekton task that accepts a repository URL and a branch name and calls `git clone` to clone your source code.

1. Create a new task and name it `checkout`.
2. Add a parameter named `repo-url` with a `type:` of string and a `description:` of "The URL of the git repo to clone".

3. Add a second parameter named `branch` with a `type:` of string and a `description:` of “The branch to clone”.

4. Add a step with the `name:` “checkout” that uses the `bitnami/git:latest` image to run the `git` command by specifying `clone` and `--branch` parameters and passing both the params created in spec as the arguments.

Hint

▼ Click here for a hint.
Pass arguments to the checkout task as ["clone", "--branch", "$(params.branch)", "$(params.repo-url)"].

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

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: {name of task here}
spec:
  params:
    - name: {1st parameter name here}
      description: {1st parameter name here}
      type: string
    - name: {2nd parameter name here}
      description: {2nd parameter name here}
      type: string
  steps:
    - name: {step name here}
      image: {image name here}
      command: [{command name here}]
      args: [{arguments here}]
```

Copied!

Double-check that your work matches the solution below.

Solution

▼ Click here for the answer.

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

```
---
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: checkout
spec:
  params:
    - name: repo-url
      description: The URL of the git repo to clone
      type: string
    - name: branch
      description: The branch to clone
      type: string
  steps:
    - name: checkout
      image: bitnami/git:latest
      command: [git]
      args: ["clone", "--branch", "$(params.branch)", "$(params.repo-url)"]
```

Copied!

Apply it to the cluster:

```
1
```

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

Copied! Executed!

Your output should look like this:

```
1
2
```

```
task.tekton.dev/echo configured
task.tekton.dev/checkout created
```

Copied!

The `echo` task was unchanged and the `checkout` task has been created.

Step 8: Create the cd-pipeline Pipeline

Finally, you will create a pipeline called `cd-pipeline` to be the starting point of your Continuous Delivery pipeline.

Open the `pipeline.yaml` file to create a new pipeline called `cd-pipeline`:

Open **pipeline.yaml** in IDE

You can use `---` on a separate line to separate your new pipeline, or you can modify the existing pipeline to look like the new one.

Your Task

1. Create a new pipeline and name it `cd-pipeline`.
2. Add two parameters named `repo-url` and `branch`.
3. Set the `default:` for **branch** to “master”.
4. Add a task with the `name:` “clone” that has a `taskRef:` to the `checkout` task that you just created.
5. Add the two parameters `repo-url` and `branch` to the task, mapping them back to the pipeline parameters of the same name.

Hint

▼ Click here for a hint.
Add params of name `repo-url` and `branch` with values as "$(params.repo-url)" and "$(params.branch)" to the `clone` task.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

```
spec:
  params:
    - name: {1st parameter name here}
    - name: {2nd parameter name here}
      default: {default value here}
  tasks:
    - name: {pipeline task name here}
      taskRef:
        name: {Task name here}
      params:
      - name: {1st parameter name here}
        value: "$(params.repo-url)"
      - name: {2nd parameter name here}
        value: "$(params.branch)"
```

Copied!

Double-check that your work matches the solution below.

Solution

▼ Click here for the answer.

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

```
---
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: cd-pipeline
spec:
  params:
    - name: repo-url
    - name: branch
      default: "master"
  tasks:
    - name: clone
      taskRef:
        name: checkout
      params:
      - name: repo-url
        value: "$(params.repo-url)"
      - name: branch
        value: "$(params.branch)"
```

Copied!

Apply it to the cluster:

```
1
```

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

Copied! Executed!

Step 9: Run the cd-pipeline

Run the pipeline using the Tekton CLI:

```
1
2
3
4
```

```
tkn pipeline start cd-pipeline \
    --showlog \
    -p repo-url="https://github.com/ibm-developer-skills-network/wtecc-CICD_PracticeCode.git" \
    -p branch="main"
```

Copied! Executed!

The output should look like this:

```
1
2
3
```

```
PipelineRun started: cd-pipeline-run-rf6zp
Waiting for logs to be available...
[clone : checkout] Cloning into 'wtecc-CICD_PracticeCode'...
```

Copied!

Step 10: Fill Out cd-pipeline with Placeholders

In this final step, you will fill out the rest of the pipeline with calls to the `echo` task to simply display a message for now. You will replace these "placeholder" tasks with real ones in future labs.

Update the pipeline.yaml file to include four placeholder tasks.

Open **pipeline.yaml** in IDE

Now you will add four tasks to the pipeline to lint, unit test, build, and deploy. All of these pipeline tasks will reference the `echo` task for now.

Your Task

Create a pipeline task for each of these:

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

Hint

▼ Click here for a hint.
Add params of name `repo-url` and `branch` with values as "$(params.repo-url)" and "$(params.branch)" to the `clone` task.

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

```
spec:
  tasks:
    - name: {pipeline task name here}
      taskRef:
        name: echo
      params:
      - name: message
        value: {message to display here}
      runAfter:
        - {name of previous task}

    ...
```

Copied!

You now have a base pipeline to build the rest of your tasks into.

Double-check that your work matches the solution below.

Solution

▼ Click here for the answer.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
```

```
36. 36
37. 37
38. 38
39. 39
40. 40
41. 41
42. 42
43. 43
44. 44
45. 45
46. 46
47. 47
48. 48
49. 49
50. 50
51. 51
52. 52
53. 53
54. 54
55. 55
```

```
 1. ---
 2. apiVersion: tekton.dev/v1beta1
 3. kind: Pipeline
 4. metadata:
 5.   name: cd-pipeline
 6. spec:
 7.   params:
 8.     - name: repo-url
 9.     - name: branch
10.       default: "master"
11.   tasks:
12.     - name: clone
13.       taskRef:
14.         name: checkout
15.       params:
16.       - name: repo-url
17.         value: "$(params.repo-url)"
18.       - name: branch
19.         value: "$(params.branch)"
20.
21.     - name: lint
22.       taskRef:
23.         name: echo
24.       params:
25.       - name: message
26.         value: "Calling Flake8 linter..."
27.       runAfter:
28.         - clone
29.
30.     - name: tests
31.       taskRef:
32.         name: echo
33.       params:
34.       - name: message
35.         value: "Running unit tests with PyUnit..."
36.       runAfter:
37.         - lint
38.
39.     - name: build
40.       taskRef:
41.         name: echo
42.       params:
43.       - name: message
44.         value: "Building image for $(params.repo-url) ..."
45.       runAfter:
46.         - tests
47.
48.     - name: deploy
49.       taskRef:
50.         name: echo
51.       params:
52.       - name: message
53.         value: "Deploying $(params.branch) branch of $(params.repo-url) ..."
54.       runAfter:
55.         - build
```

Copied!

Apply it to the cluster:

```
1. 1
```

```
1. kubectl apply -f pipeline.yaml
```

Copied! Executed!

Step 11: Run the cd-pipeline

Run the pipeline using the Tekton CLI:

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

```
1. tkn pipeline start cd-pipeline \
2.     --showlog \
3.     -p repo-url="https://github.com/ibm-developer-skills-network/wtecc-CICD_PracticeCode.git" \
4.     -p branch="main"
```

Copied! Executed!

The output will look like this:

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

```
1. PipelineRun started: cd-pipeline-run-wvfzx
2. Waiting for logs to be available...
3. [clone : checkout] Cloning into 'wtecc-CICD_PracticeCode'...
4.
5. [lint : echo-message] Calling Flake8 linter...
```

```
6.
7. [tests : echo-message] Running unit tests with PyUnit...
8.
9. [build : echo-message] Building image for https://github.com/ibm-developer-skills-network/wtecc-CICD_PracticeCode.git ...
10.
11. [deploy : echo-message] Deploying main branch of https://github.com/ibm-developer-skills-network/wtecc-CICD_PracticeCode.git ...
```

Copied!

Conclusion

Congratulations! You are now able to create a Tekton pipeline and pass parameters to a pipeline.

In this lab, you learned how to create a base pipeline, specify and pass parameteres to a task and pipeline. You learned how to modify your pipeline to reference the task and configure its parameters. You also learned how to pass additional parameters to a pipeline and how to run it to echo and clone a Git repository.

Next Steps

You will learn and use GitHub Triggers in the next lab.

Author(s)

Tapas Mandal
John J. Rofrano

Other Contributor(s)

Change Log

Date	Version	Changed by	Change Description
2022-07-24	0.1	Tapas Mandal	Initial version created
2022-07-28	0.2	John Rofrano	Added additional instructions
2022-07-31	0.3	Steve Ryan	ID review
2022-08-01	0.4	Beth Larsen	QA review
2023-03-15	0.5	Lavanya Rajalingam	Updated SN Logo
