Adding GitHub Triggers

Skills Network

Estimated time needed: 30 minutes

Welcome to this hands-on lab for **Adding GitHub Triggers**.

Running a pipeline manually has limited uses. In this lab you will create a Tekton Trigger to cause a pipeline run from external events like changes made to a repo in GitHub.

Learning Objective

After completing this lab, you will be able to:

- Create an EventListener, a TriggerBinding and a TriggerTemplate
- State how to trigger a deployment when changes are made to github

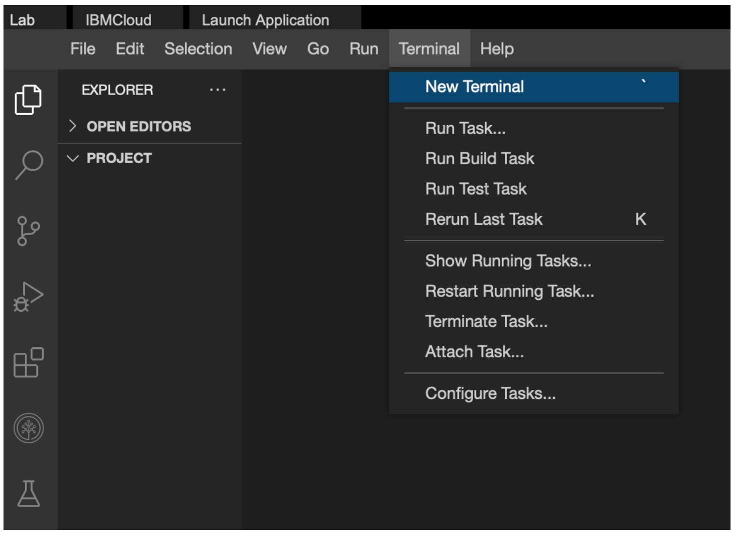
---

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!

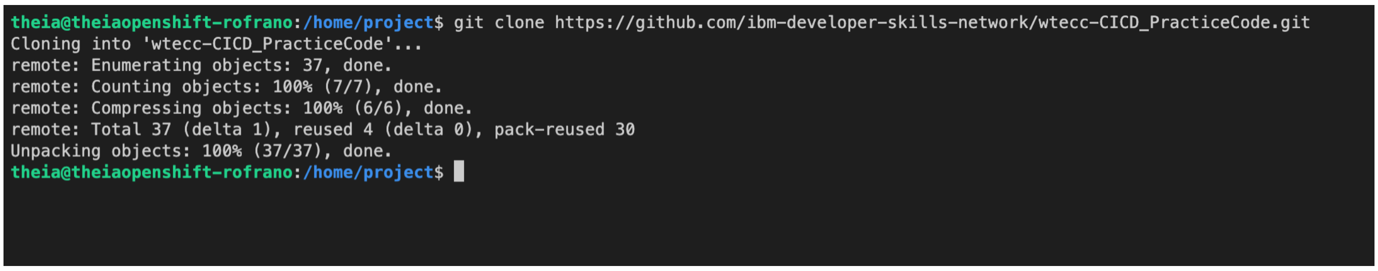
Clone the Code Repo

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:



```
theia@theiaopenshift-rofrano:/home/project$ git clone https://github.com/ibm-developer-skills-network/wtecc-CICD_PracticeCode.git
Cloning into 'wtecc-CICD_PracticeCode'...
remote: Enumerating objects: 37, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 37 (delta 1), reused 4 (delta 0), pack-reused 30
Unpacking objects: 100% (37/37), done.
theia@theiaopenshift-rofrano:/home/project$
```

Change to the Labs Directory

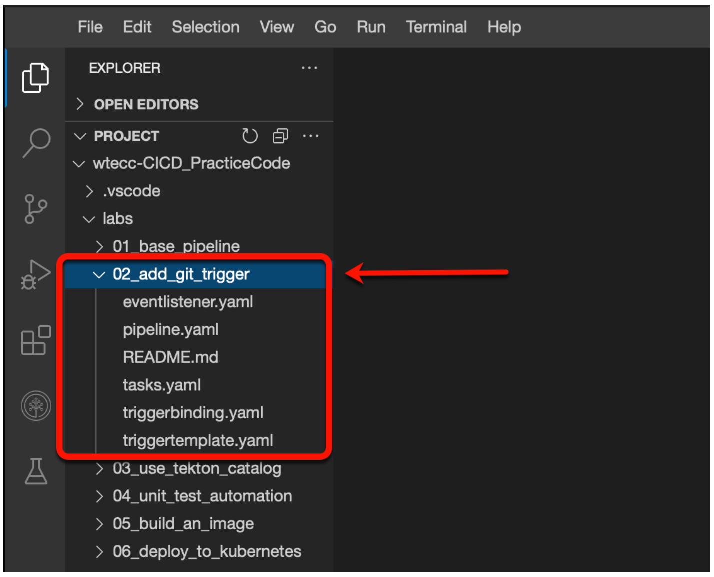
Once you have cloned the repository, change to the labs directory.

```
1. 1
1. cd wtecc-CICD_PracticeCode/labs/02_add_git_trigger/
```

Copied! Executed!

Navigate to the Lab Folder

Navigate to the `labs/02_add_git_trigger` folder in left explorer panel. All of your work will be with the files in this folder.



wtecc-CICD_PracticeCode
.vscode
labs
01_base_pipeline
02_add_git_trigger
eventlistener.yaml
pipeline.yaml
README.md
tasks.yaml
triggerbinding.yaml
triggertemplate.yaml
03_use_tekton_catalog
04_unit_test_automation
05_build_an_image
06_deploy_to_kubernetes

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:

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

Copied! Executed!

Prerequisites

This lab starts with the `cd-pipeline` pipeline and `checkout` and `echo` tasks from the previous lab.

If you did not complete the previous lab, you should apply them to your Kubernetes cluster before starting this lab:

Issue the following commands to install everything from the previous labs.

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

Copied! Executed!

Check that the tasks were created:

```
tkn task ls
```

Copied! Executed!

You should see output similar to this:

```
NAME       DESCRIPTION   AGE
checkout                 2 minute ago
echo                     2 minute ago
```

Copied!

Check that the pipeline was created:

```
tkn pipeline ls
```

Copied! Executed!

You should see output similar to this:

```
NAME          AGE             LAST RUN   STARTED   DURATION   STATUS
cd-pipeline   2 minutes ago   ---        ---       ---        ---
```

Copied!

You are now ready to continue with this lab.

Step 1: Create an EventListener

The first thing you need is an event listener that is listening for incoming events from GitHub.

You will update the `eventlistener.yaml` file to define an `EventListener` named `cd-listener` that references a TriggerBinding named `cd-binding` and a TriggerTemplate named `cd-template`.

Open **eventlistener.yaml** in IDE

It should initially look like this:

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

Copied!

Your Task

1. The first thing you want to do is give the EventListener a good name. Change `<place-name-Here>` to `cd-listener`.
2. The next thing is to add a service account. Add a `serviceAccountName:` with a value of `pipeline` to the spec section.
3. Now you need to define the triggers. Add a `triggers:` section under `spec:`. This is where you will define the bindings and template.
4. Add a `bindings:` section under the `triggers:` section with a `ref:` to `cd-binding`. Since there can be mutiple triggers, make sure you define `- bindings` as a list using the dash `-` prefix. Also since there can be multiple bindings, make sure you defne the `- ref:` with a dash `-` prefix as well.
5. Add a `template:` section at the same level as `bindings` with a `ref:` to `cd-template`.

Hint

▸ Click here for a hint.

Double-check that your work matches the solution below.

Solution

▾ Click here for the answer.

```
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: cd-listener
spec:
  serviceAccountName: pipeline
  triggers:
    - bindings:
      - ref: cd-binding
      template:
        ref: cd-template
```

Copied!

Apply the EventListener resource to the cluster:

```
kubectl apply -f eventlistener.yaml
```

Copied! Executed!

Check that it was created correctly.

```
tkn eventlistener ls
```

Copied! Executed!

You should see a reply similar to this:

```
NAME          AGE             URL                                                  AVAILABLE
cd-listener   9 seconds ago   http://el-cd-listener.default.svc.cluster.local:8080   True
```

Copied!

You will create the TriggerBinding named `cd-binding` and a TriggerTemplate named `cd-template` in the next steps.

Step 2: Create a TriggerBinding

The next thing you need is a way to bind the incoming data from the event to pass on to the pipeline. To accomplish this, you use a `TriggerBinding`.

Update the `triggerbinding.yaml` file to create a TriggerBinding named `cd-binding` that takes the `body.repository.url` and `body.ref` and binds them to the parameters `repository` and `branch`, respectively.

Open **triggerbinding.yaml** in IDE

It should initially look like this:

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

Copied!

Your Task

1. The first thing you want to do is give the TriggerBinding the same name that is referenced in the EventListener, which is `cd-binding`.
2. Next, you need to add a parameter named `repository` to the `spec:` section with a value that references `$(body.repository.url)`.
3. Finally, you need to add a parameter named `branch` to the `spec:` section with a value that references `$(body.ref)`.

Hint

▸ Click here for a hint.

Double-check that your work matches the solution below.

Solution

▾ Click here for the answer.

```
apiVersion: triggers.tekton.dev/v1beta1
kind: TriggerBinding
metadata:
  name: cd-binding
spec:
  params:
    - name: repository
      value: $(body.repository.url)
    - name: branch
      value: $(body.ref)
```

Copied!

Apply the new TriggerBinding definition to the cluster:

```
kubectl apply -f triggerbinding.yaml
```

Copied! Executed!

Step 3: Create a TriggerTemplate

The TriggerTemplate takes the parameters passed in from the TriggerBinding and creates a PipelineRun to start the pipeline.

Update the `triggertemplate.yaml` file to create a TriggerTemplate named `cd-template` that defines the parameters required, and create a PipelineRun that will run the `cd-pipeline` you created in the previous lab.

Open **triggertemplate.yaml** in IDE

It should initially look like this:

```
apiVersion: triggers.tekton.dev/v1beta1
kind: TriggerTemplate
metadata:
  name: <place-name-here>
spec:
  params:
  # Add parameters here
  resourcetemplates:
    - apiVersion: tekton.dev/v1beta1
      kind: PipelineRun
      metadata:
        generateName: cd-pipeline-run-
      spec:
      # Add pipeline definition here
```

Copied!

Your Task

You must update the parameter section of the TriggerTemplate and fill out the resourcetemplates section:

Update Name and Add Parameters

1. The first thing you want to do is give the TriggerTemplate the same name that is referenced in the EventListener, which is `cd-template`.
2. Next, you need to add a parameter named `repository` to the `spec:` section with a `description:` of *"The git repo"* and a `default:` of `" "`.
3. Then, you need to add a parameter named `branch` to the `spec:` section with a `description:` of *"the branch for the git repo"* and a `default:` of `master`.

Hint 1

▸ Click here for a hint.

Complete the Resource Template

Finish filling out the `resourcetemplates:` section by adding the following after the commented line `# Add pipeline definition here`.

1. Add a `serviceAccountName:` with a value of `pipeline`.
2. Add a `pipelineRef:` that refers to the `cd-pipeline` created in the last lab.
3. Add a parameter named `repo-url` with a value referencing the TriggerTemplate `repository` parameter above.
4. Add a second parameter named `branch` with a value referencing the TriggerTemplate`branch` parameter above.

Hint 2

▾ Click here for a hint.

The `resourcetemplates.spec:` section of your triggertemplate.yaml file structure should mirror this replacing the values in `{}` with the actual values:

```
spec:
  resourcetemplates:
      spec:
        # Add pipeline definition here
        serviceAccountName: {sa name goes here}
        pipelineRef:
          name: {pipeline name goes here}
        params:
          - name: {repository url parameter goes here}
            value: $(tt.params.repository)
          - name: {branch parameter goes here}
            value: $(tt.params.branch)
```

Copied!

Double-check that your work matches the solution below.

Solution

▾ Click here for the answer.

```
apiVersion: triggers.tekton.dev/v1beta1
kind: TriggerTemplate
metadata:
  name: cd-template
spec:
  params:
    - name: repository
      description: The git repo
      default: " "
    - name: branch
      description: the branch for the git repo
      default: master
  resourcetemplates:
    - apiVersion: tekton.dev/v1beta1
      kind: PipelineRun
      metadata:
        generateName: cd-pipeline-run-
      spec:
        serviceAccountName: pipeline
        pipelineRef:
          name: cd-pipeline
        params:
          - name: repo-url
            value: $(tt.params.repository)
          - name: branch
            value: $(tt.params.branch)
```

Copied!

> Note that while the parameter you bound from the event is `repository`, you pass it on as `repo-url` to the pipeline. This is to show that the names do not have to match, allowing you to use any pipeline to map parameters into.

Apply the new TriggerTemplate definition to the cluster:

```
kubectl apply -f triggertemplate.yaml
```

Copied! Executed!

Step 4: Start a Pipeline Run

Now it is time to call the event listener and start a PipelineRun. You can do this locally using the `curl` command to test that it works.

For this last step, you will need two terminal sessions.

Terminal 1

In one of the sessions, you need to run the `kubectl port-forward` command to forward the port for the event listener so that you can call it on `localhost`.

Use the `kubectl port-forward` command to forward port `8090` to `8080`.

```
kubectl port-forward service/el-cd-listener  8090:8080
```

Copied! Executed!

You will see the following output, but you will not get your cursor back.

```
Forwarding from 127.0.0.1:8090 -> 8080
Forwarding from [::1]:8090 -> 8080
```

Copied!

Terminal 2

Now you are ready to trigger the event listener by posting to the endpoint that it is listening on. You will now need to open a second terminal shell to issue commands.

1. Open a new Terminal shell wtih the menu item `Terminal > New Terminal`.
2. Use the `curl` command to send a payload to the event listener service.

```
curl -X POST http://localhost:8090 \
  -H 'Content-Type: application/json' \
  -d '{"ref":"main","repository":{"url":"https://github.com/ibm-developer-skills-network/wtecc-CICD_PracticeCode"}}'
```

Copied! Executed!

This should start a PipelineRun. You can check on the status with this command:

```
tkn pipelinerun ls
```

Copied! Executed!

You should see something like this come back:

```
NAME                    STARTED          DURATION   STATUS
cd-pipeline-run-hhkpm   10 seconds ago   ---        Running
```

Copied!

You can also examine the PipelineRun logs using this command (the -L means "latest" so that you do not have to look up the name for the last run):

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

Copied! Executed!

You should see:

```
[clone : checkout] Cloning into 'wtecc-CICD_PracticeCode'...

[lint : echo-message] Calling Flake8 linter...

[tests : echo-message] Running unit tests with PyUnit...

[build : echo-message] Building image for https://github.com/ibm-developer-skills-network/wtecc-CICD_PracticeCode ...

[deploy : echo-message] Deploying master branch of https://github.com/ibm-developer-skills-network/wtecc-CICD_PracticeCode ...
```

Copied!

Conclusion

Congratulations, you have successfully set up Tekton Triggers.

In this lab, you learned how to create a Tekton Trigger to cause a pipeline run from external events like changes made to a repo in GitHub. You learned how to create EventListerners, TriggerTemplates, TriggerBindings and how to start a Pipeline Run on a port.

Next Steps

Now that you know your triggers are working, you can expose the event listener service with an ingress and call it from a webhook in GitHub and have it run on changes to your GitHub repository.

Author(s)

Tapas Mandal
John J. Rofrano