Integrating Unit Test Automation

Estimated time needed: 30 minutes

Welcome to the hands-on lab for **Integrating Unit Test Automation**. In this lab, you will take the cloned code from the previous pipeline step and run linting and unit tests against it to ensure it is ready to be built and deployed.

Learning Objectives

After completing this lab, you will be able to:

- Use the Tekton CD catalog to install the flake8 task
- Describe the parameters required to use the flake8 task
- Use the flake8 task in a Tekton pipeline to lint your code
- Create a test task from scratch and use it in your pipeline

Set Up the Lab Environment

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

Open a Terminal

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

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

```
cd /home/project
```

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:

```
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.

```
cd wtecc-CICD_PracticeCode/labs/04_unit_test_automation/
```

Copied! Executed!

Navigate to the Labs Folder

Navigate to the `labs/04_unit_test_automation` folder in the left explorer panel. All of your work will be with the files in this folder.

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

Optional

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

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

Copied! Executed!

Prerequisites

This lab requires installation of the tasks introduced in previous labs. To be sure, apply the previous tasks to your cluster before proceeding. Reissuing these commands will not hurt anything:

Establish the Tasks

```
kubectl apply -f tasks.yaml
tkn hub install task git-clone
```

Copied! Executed!

Note: If the above command for installing git-clone task returns a error due to Tekton Version mismatch, please run the below command to fix this.

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

Copied! Executed!

Check that you have all of the previous tasks installed:

```
tkn task ls
```

Copied! Executed!

You should see the output similar to this:

```
NAME        DESCRIPTION              AGE
checkout                             2 minutes ago
echo                                 2 minutes ago
git-clone   These Tasks are Git...   2 minutes ago
```

Copied!

Establish the Workspace

You also need a PersistentVolumeClaim (PVC) to use as a workspace. Apply the following `pvc.yaml` file to establish the PVC:

```
kubectl apply -f pvc.yaml
```

Copied! Executed!

You should see the following output:

> Note: if the PVC already exists, the output will say **unchanged** instead of **created**. This is fine.

```
persistentvolumeclaim/pipelinerun-pvc created
```

Copied!

You can now reference this persistent volume claim by its name `pipelinerun-pvc` when creating workspaces for your Tekton tasks.

You are now ready to continue with this lab.

Step 0: Check for cleanup

Please check as part of Step 0 for the new `cleanup` task which has been added to `tasks.yaml` file.

When a task that causes a compilation of the Python code, it leaves behind .pyc files that are owned by the specific user. For consecutive pipeline runs, the git-clone task tries to empty the directory but needs privileges to remove these files and this `cleanup` task takes care of that.

The `init` task is added `pipeline.yaml` file which runs everytime before the `clone` task.

Check the `tasks.yaml` file which has the new `cleanup` task updated.

Check the updated cleanup task

▶ Click here.

Check the `pipeline.yaml` file which is updated with `init` that uses the `cleanup` task.

Check the updated init task

▼ Click here.

```
tasks:
    - name: init
      workspaces:
        - name: source
          workspace: pipeline-workspace
      taskRef:
        name: cleanup
```

Copied!

Step 1: Add the flake8 Task

Your pipeline has a placeholder for a `lint` step that uses the `echo` task. Now it is time to replace it with a real linter.

You are going to use `flake8` to lint your code. Luckily, Tekton Hub has a flake8 task that you can install and use:

Use the following Tekton CLI command to install the flake8 task into your namespace.

```
tkn hub install task flake8
```

Copied! Executed!

This will install the `flake8` task in your Kubernetes namespace.

You should see output similar to this:

> Note: Your Kubernetes namespace will be listed.

```
Task flake8(0.1) installed in sn-labs-rofrano namespace
```

Copied!

Step 2: Modify the Pipeline to Use flake8

Now you will modify the `pipeline.yaml` file to use the new `flake8` task.

In reading the documentation for the flake8 task, you notice that it requires a workspace named `source`. Add the workspace to the lint task after the `name:`, but before the `taskRef:`.

Open **pipeline.yaml** in IDE

Your Task

1. Scroll down to the `lint` task.
2. Add the `workspaces:` keyword to the lint task after the task `name:` but before the `taskRef:`.
3. Specify the workspace `name:` as `source`.
4. Specify the `workspace:` reference as `pipeline-workspace`, which was created in the previous lab.

   ▶ Click here for a hint.

5. Change the `taskRef:` from `echo` to reference the `flake8` task.

   ▼ Click here for a hint.

   ```
       - name: lint
         ...
         taskRef:
           name: {task reference to flake}
   ```

   Copied!

Check that your new edits match the solution up to this point.

Solution

▼ Click here for the answer.

```
    - name: lint
      workspaces:
```

```
        - name: source
          workspace: pipeline-workspace
      taskRef:
        name: flake8
```

Copied!

Next, you will modify the parameters passed into the task.

Step 3: Modify the Parameters for flake8

Now that you have added the workspace and changed the task reference to **flake8**, you need to modify the `pipeline.yaml` file to change the parameters to what flake8 is expecting.

In reading the documentation for the flake8 task, you see that it accepts an optional `image` parameter that allows you to specify your own container image. Since you are developing in a Python 3.9-slim container, you want to use `python:3.9-slim` as the image.

The flake8 task also allows you to specify arguments to pass to flake8 using the `args` parameter. These arguments are specified as a list of strings where each string is a parameter passed to flake8. For example, the arguments `--count --statistics` would be specified as: `["--count", "--statistics"]`.

Edit the `pipeline.yaml` file:

Open **pipeline.yaml** in IDE

Your Task

1. Change the `message` parameter to the `image` parameter to specify the value of `python:3.9-slim`.
2. Add a new parameter called `args` to specify the arguments as a list `[]` with the values `--count --max-complexity=10 --max-line-length=127 --statistics` to pass to flake8.

   The documentation tells you that this must be passed as a list, so be sure to pass each argument as a separate string in the list, delimited by commas.

Hint

▼ Click here for a hint.

```
    - name: lint
      ...
      params:
      - name: {image goes here}
        value: {name of image}
      - name: {args goes here}
        value: ["{list}","{of}","{arguments}","{here}"]
```

Copied!

Double-check that your work matches the solution below.

Solution

▼ Click here for the answer.

```
    - 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

    # Note: The remaining tasks are unchanged
```

Copied!

Apply these changes to your cluster:

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

Copied! Executed!

You should see the following output:

```
pipeline.tekton.dev/cd-pipeline configured
```

Copied!

Step 4: Run the Pipeline

You are now ready to run the pipeline and see if your new lint task is working properly. You will use the Tekton CLI to do this.

Start the pipeline using the following command:

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

Copied! Executed!

You should see the pipeline run complete successfully. If you see errors, go back and check your work against the solutions provided.

Step 5: Create a Test Task

Your pipeline also has a placeholder for a `tests` task that uses the `echo` task. Now you will replace it with real unit tests. In this step, you will replace the `echo` task with a call to a unit test framework called `nosetests`.

There are no tasks in the Tekton Hub for `nosetests`, so you will write your own.

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 course video.

Open **tasks.yaml** in IDE

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:

```
#!/bin/bash
set -e
python -m pip install --upgrade pip wheel
pip install -r requirements.txt
nosetests -v --with-spec --spec-color
```

Copied!

Your Task

1. Create a new task in the `tasks.yaml` file and name it `nose`. Remember, each new task must be separated using three dashes `---` on a separate line.

   ▼ Click here for a hint.

   ```
   ---
   apiVersion: tekton.dev/v1beta1
   kind: Task
   metadata:
     name: {name goes here}
   ```

   Copied!

2. 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.

   ```
     workspaces:
       - name: {name goes here}
   ```

   Copied!

3. 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.

```
  params:
    - name: {name goes here}
      description: {description goes here}
      type: {type goes here}
      default: "-v"
```

Copied!

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

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

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

7. Then, paste the script from above in the `script` parameter.

▼ Click here for a hint.

```
  steps:
    - name: {name goes here}
      image: {image goes here}
      workingDir: $(workspaces.source.path)
      script: |
        {paste bash script here}
```

Copied!

Double-check that your work matches the solution below.

Solution

▼ Click here for the answer.

```
---
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: nose
spec:
  workspaces:
    - name: source
  params:
    - name: args
      description: Arguments to pass to nose
      type: string
      default: "-v"
  steps:
    - name: nosetests
      image: python:3.9-slim
      workingDir: $(workspaces.source.path)
      script: |
        #!/bin/bash
        set -e
        python -m pip install --upgrade pip wheel
        pip install -r requirements.txt
        nosetests $(params.args)
```

Copied!

Apply these changes to your cluster:

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

Copied! Executed!

You should see the following output:

```
task.tekton.dev/echo configured
task.tekton.dev/cleanup configured
task.tekton.dev/nose created
```

Copied!

Step 6: Modify the Pipeline to Use nose

The final step is to use the new `nose` task in your existing pipeline in place of the `echo` task placeholder.

Edit the `pipeline.yaml` file.

Open **pipeline.yaml** in IDE

Add the workspace to the `tests` task after the name but before the `taskRef:`, change the `taskRef` to reference your new `nose` task, and change the `message` parameter to pass in your new `args` parameter.

Your Task

Scroll down to the `tests` task definition.

1. Add a workspace named `source` that references `pipeline-workspace` to the `tests` task after the `name:` but before the `taskRef:`.

   ▶ Click here for a hint.

2. Change the `taskRef:` from `echo` to reference your new `nose` task.

   ▶ Click here for a hint.

3. Change the `message` parameter to the `args` parameter and specify the arguments to pass to the tests as `-v --with-spec --spec-color`.

   ▶ Click here for a hint.

Double-check that your work matches the solution below.

Solution

▼ Click here for the answer.

```
    - name: tests
      workspaces:
        - name: source
          workspace: pipeline-workspace
      taskRef:
        name: nose
      params:
      - name: args
        value: "-v --with-spec --spec-color"
      runAfter:
        - lint
```

Copied!

Apply these changes to your cluster:

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

Copied! Executed!

You should see the following output:

```
pipeline.tekton.dev/cd-pipeline configured
```

Copied!

Step 7: Run the Pipeline Again

Now that you have your `tests` task complete, run the pipeline again using the Tekton CLI to see your new test tasks run:

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

Copied! Executed!

You can see the pipeline run status by listing the PipelineRun with:

```
tkn pipelinerun ls
```

Copied! Executed!

You should see:

```
$ tkn pipelinerun ls
NAME                     STARTED         DURATION     STATUS
cd-pipeline-run-fbxbx    1 minute ago    59 seconds   Succeeded
```

Copied!

You can check the logs of the last run with:

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

Copied! Executed!

Conclusion

Congratulations! You have just added a task from the Tekton catalog and used a familiar tool to write your own custom task for testing your code.

In this lab, you learned how to use the `flake8` task from the Tekton catalog. You learned how to install the task locally using the Tekton CLI and how to modify your pipeline to reference the task and configure its parameters. You also learned how to create your own task using a shell script that you already have and how to pass parameters into your new task.

Next Steps

In the next lab, you will learn how to build a container image and push it to a local registry in preparation for final deployment. In the meantime, try to set up a pipeline to build an image with Tekton from one of your own code repositories.

If you are interested in continuing to learn about Kubernetes and containers, you can get your own free Kubernetes cluster and your own free IBM Container Registry.

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
2022-08-01	0.2	Tapas Mandal	Added additional instructions
2022-08-08	0.3	John Rofrano	Added more detailed instructions
2022-08-09	0.4	Steve Ryan	ID Review
2022-08-09	0.5	Beth Larsen	QA review
2022-11-22	0.6	Lavanya Rajalingam	Updated Instructions to include Cleanup Task
2023-03-15	1.5	Lavanya Rajalingam	Updated SN Logo