Lab (Option B: JavaScript): 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 eslint task
- Describe the parameters required to use the eslint task
- Use the eslint task in a Tekton pipeline to lint your JavaScript 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
```

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/ttwst-jhxyb-ci-cd-pipeline_js.git
```

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 ttwst-jhxyb-ci-cd-pipeline_js/labs/04_unit_test_automation/
```

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\[\033[00m\]]\$ "
```

Prerequisites

This lab requires the 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
```

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
```

Check that you have all of the previous tasks installed:

```
tkn task ls
```

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
```

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
```

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
```

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 that has been added to `tasks.yaml` file.

When a task causes a compilation of the JavaScript code, it leaves behind build files and node_modules 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 to the `pipeline.yaml` file, which runs every time before the `clone` task.

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

Check the updated cleanup task

▼ Click here.

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: cleanup
spec:
  description: This task will clean up a workspace by deleting all of the files.
  workspaces:
    - name: source
  steps:
    - name: remove
      image: alpine:3
      env:
        - name: WORKSPACE_SOURCE_PATH
          value: $(workspaces.source.path)
      workingDir: $(workspaces.source.path)
      securityContext:
        runAsNonRoot: false
        runAsUser: 0
      script: |
        #!/usr/bin/env sh
        set -eu
        echo "Removing all files from ${WORKSPACE_SOURCE_PATH} ..."
        # Delete any existing contents of the directory if it exists.
        #
        # We don't just "rm -rf ${WORKSPACE_SOURCE_PATH}" because ${WORKSPACE_SOURCE_PATH} might be "/"
        # or the root of a mounted volume.
        if [ -d "${WORKSPACE_SOURCE_PATH}" ] ; then
          # Delete non-hidden files and directories
          rm -rf "${WORKSPACE_SOURCE_PATH:?}"/*
          # Delete files and directories starting with . but excluding ..
          rm -rf "${WORKSPACE_SOURCE_PATH}"/.[!.]*
          # Delete files and directories starting with .. plus any other character
          rm -rf "${WORKSPACE_SOURCE_PATH}"/..?*
```

```
        fi
```

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
```

Step 1: Add the ESLint 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 `ESLint` to lint your JavaScript code. Since there isn't a pre-built ESLint task in Tekton Hub, you will create your own ESLint task.

First, let's add a custom ESLint task to your tasks.yaml file:

```
    # We'll add this task manually since it's not available in Tekton Hub
```

Step 2: Create ESLint task

Now, you will add a custom ESLint task to the `tasks.yaml` file.

Open the tasks.yaml file and add the following ESLint task:

Open **tasks.yaml** in IDE

Your task

1. Add a new task called eslint to the tasks.yaml file. Remember, each new task must be separated using three dashes— on a separate line.
2. The task should:
   - Have a workspace named source
   - Accept parameters for image (default: node:18-alpine) and args
   - Run ESLint with the specified arguments

Solution

▼ Click here for the answer.

```
---
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: eslint
spec:
  workspaces:
    - name: source
  steps:
    - name: install-dependencies
      image: node:18
      workingDir: $(workspaces.source.path)
      script: |
        npm install
    - name: run-eslint
      image: node:18
      workingDir: $(workspaces.source.path)
      script: |
        npx eslint .
```

Apply these changes to your cluster:

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

Step 3: Modify the pipeline to use ESLint

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

Open the pipeline.yaml file and modify the lint task:

Edit the `pipeline.yaml` file:

Open **pipeline.yaml** in IDE

Your task

1. Add the `workspaces:` keyword to the lint task after the task `name:` but before the `taskRef:`
2. Specify the workspace `name:` as `source`
3. Specify the `workspace:` reference as `pipeline-workspace`
4. Change the `taskRef:` from echo to reference the eslint task
5. Change the parameters to use image and args instead of message

Hint

▼ Click here for a hint.

```
- name: lint
  workspaces:
    - name: {workspace `name:`}
      workspace: {`workspace:` reference }
  taskRef:
    name: {task name}
  params:
  - name: {image goes here}
    value: {name of image}
  - name: {args goes here}
    value: ["{list}","{of}","{arguments}","{here}"]
```

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: eslint
  params:
  - name: image
    value: "node:18-alpine"
  - name: args
    value: ["--format", "stylish", "--ext", ".js", "."]
  runAfter:
    - clone
# Note: The remaining tasks are unchanged
```

Apply these changes to your cluster:

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

You should see the following output:

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

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/ttwst-jhxyb-ci-cd-pipeline_js" \
    -p branch="main" \
    -w name=pipeline-workspace,claimName=pipelinerun-pvc \
    --showlog
```

You should see the pipeline run completely 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 `Jest`.

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

Update the `tasks.yaml` file adding a new task called `jest` that uses the shared workspace for the pipeline and runs `npm test` in a `node:18-alpine`image.

Open **tasks.yaml** in IDE

Here is a bash script to install the Node.js dependencies and run the Jest tests. You can use this as the shell script in your new task:

```
#!/bin/sh
set -e
npm ci
npm test
```

Your task

1. Create a new task in the `tasks.yaml` file and name it `jest`. 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}
   ```

2. Next, you need to include the workspace that has the code that you want to test. Since Jest 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}
   ```

3. Create a parameter called args with a description, make the type: a string, and a default: with the verbose flag "–verbose" as the default.

   ▼ Click here for a hint.

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

4. Specify the steps with one step named test that runs in a node:18-alpine image, sets workingDir as the workspace path, and uses the script above.

   ▼ Click here for a hint.

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

Double-check that your work matches the solution below.

Solution

▼ Click here for the answer.

```
---
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: jest
spec:
  description: This task runs Jest tests for JavaScript applications
  workspaces:
    - name: source
  params:
    - name: args
      description: Arguments to pass to Jest
      type: string
      default: "--verbose"
  steps:
    - name: test
      image: node:18-alpine
      workingDir: $(workspaces.source.path)
      script: |
        #!/bin/sh
        set -e
        npm ci
        npm test -- $(params.args)
```

Apply these changes to your cluster:

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

You should see the following output:

```
task.tekton.dev/echo configured
task.tekton.dev/cleanup configured
task.tekton.dev/eslint configured
task.tekton.dev/jest configured
```

Step 6: Modify the pipeline to use Jest

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

Edit the `pipeline.yaml` file.

Open **pipeline.yaml** in IDE

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.

   ```
   - name: tests
     workspaces:
       - name: {name goes here}
         workspace: {workspace reference goes here}
     taskRef:
     ...
   ```

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

   ▼ Click here for a hint.

   ```
   - name: tests
     ...
     taskRef:
       name: {name goes here}
   ```

```
...
```

3. Change the `message` parameter to the `args` parameter and specify the arguments to pass to Jest as `--verbose --coverage`.

   ▼ Click here for a hint.

   ```
   - name: {name goes here}
     ...
     params:
     - name: {name goes here}
       value: "--verbose --coverage"
   ```

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: jest
  params:
  - name: args
    value: "--verbose --coverage"
  runAfter:
    - lint
```

Apply these changes to your cluster:

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

You should see the following output:

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

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/ttwst-jhxyb-ci-cd-pipeline_js.git" \
    -p branch="main" \
    -w name=pipeline-workspace,claimName=pipelinerun-pvc \
    --showlog
```

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

```
tkn pipelinerun ls
```

You should see:

```
$ tkn pipelinerun ls
NAME                   STARTED          DURATION   STATUS
cd-pipeline-run-6jdtk   3 minutes ago    3m11s      Succeeded
cd-pipeline-run-n9plp   15 minutes ago   1m42s      Succeeded
```

You can check the logs of the last run with:

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

Conclusion

Congratulations! You have just created custom ESLint and Jest tasks and integrated them into your Tekton pipeline for JavaScript development.

In this lab, you learned how to create custom tasks for JavaScript linting and testing. You learned how to use `ESLint` for code quality checks and Jest for unit testing. You also learned how to create your own tasks using shell scripts and how to pass parameters into your new tasks.

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)

Harsh Singh