ACTI 4850 – Lab 6 – Shared Pipeline Libraries

Instructor	Mike Mulder (mmulder10@bcit.ca)
Total Marks	10
Due Dates	Demo Due by End of Next Class: • March 4th (or earlier). Can be demoed with Lab 7 • Feb. 20th for Set B • Feb. 22nd for Set A

Applicable Requirements

- **REQ1150 –** The Enterprise Development Environment shall automatically trigger a Continuous Integration Pipeline on a Software Project (i.e., repository) in Source Code Management when the source code changes. Note: A push to a project in GitLab should trigger the corresponding build job.
- **REQ1160 –** The Enterprise Development Environment shall support Continuous Integration Pipelines for Python projects. At minimum, the pipeline will include build, test, packaging and artifact storage. Note the Python steps in the Build and Test stages must run in a Virtual Environment.
- **REQ1170** – The Enterprise Development Environment shall support re-use of CI Pipeline definitions across similar projects. For example, similar Python projects (i.e., Flask applications) should be able to use the same pipeline definition.

Group Work

You will be working on the same Azure cloud environment as the previous lab, shared with your Lab partner. This lab will be done together with your partner.

Part 1 – Extend Your Point Project's Pipeline

Add the following stages to the Jenkinsfile for you Point Project (in the given order):

Note: Use the Jenkins Replay feature to test your changes rather than making a large number of Git commits.

Build – Existing

- Leave as is, except
- Create a Virtual Environment and run the pip install command in that virtual environment
    - Note 1: You may have to install the Virtual Environment module on your Jenkins image. If so, you'll need to update the dockerfile and stop/remove/build and start the image
    - Note 2: Delete the virtual environment if it already exists (we don't want old dependencies in the virtual environment.

Python Lint – New

- Runs Python Lint

- Note that you will need to install the following in your Jenkins Master docker image (i.e., update the dockerfile then stop/remove/build and restart the image:

apt-get install –y pylint

- You will need to run pylint --fail-under <Score> <Files> in your Lint Stage where the app will fail with a lint score under 5 and all .py files are scanned. See https://pylint.readthedocs.io/en/latest/user_guide/configuration/all-options.html#fail-under
- You can use wildcards to apply this to all .py files

Test and Coverage – Rename of Unit Test Stage

- Modify so it runs each file in the workspace that starts with "test" as a unittest
    - You will need to install the Pipeline Utilities Steps plugin in Jenkins to get the ability to get a filtered list of all files in a workspace (search on findFiles)
- Have a Post steps that processes both the test-results and api-test-results data
- Also add test coverage. See https://coverage.readthedocs.io/en/coverage-5.0.3/. Note that you will have to add the coverage library to your dockerfile and stop/build/restart the image.
- Hints on coverage (**see the slides for Week 6 on the pattern to use**):

Note: Run coverage in your virtual environment.

For each test file run:
coverage run --omit */site-packages/*,*/dist-packages/* <test file name>
The above should replace your test runs as it will run the tests and generate the test results as well as generating the coverage (otherwise you will get double the number of test results).

After all the test are run:
coverage report

Fixing the Jenkins Unit Test graph:

- Previously, most groups had a unit test graph in their Jenkins pipeline that kept on increasing each time the build was run. It should just show the number of units test run each build, not a running total. To fix this, we'll add this script code to the Test and Coverage stage:

```
// Remove any existing test results
script {
      def test_reports_exist = fileExists 'test-reports'
      if (test_reports_exist) {
          sh 'rm test-reports/*.xml || true'
      }

      def api_test_reports_exist = fileExists 'api-test-reports'
      if (api_test_reports_exist) {
          sh 'rm api-test-reports/*.xml || true'
      }
```

```
}

// Your Code to Run the Unit Tests with for loop

// Process the test results if they exist
script {
      def test_reports_exist = fileExists 'test-reports'
      if (test_reports_exist) {
          junit 'test-reports/*.xml'
      }

      def api_test_reports_exist = fileExists 'api-test-reports'
      if (api_test_reports_exist) {
          junit 'api-test-reports/*.xml'
      }
}
```

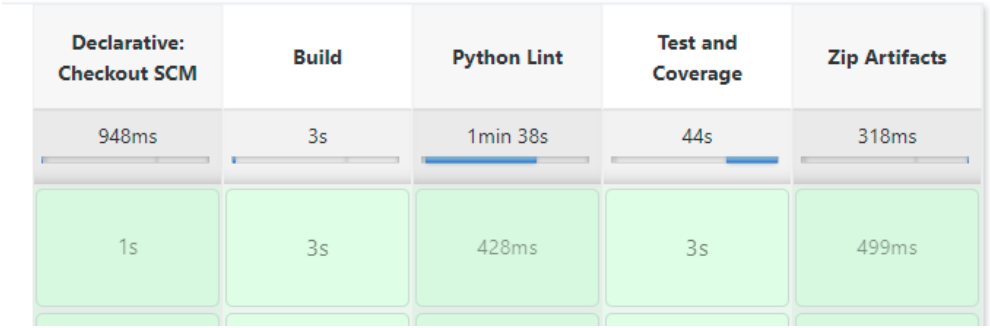
Integration Test – Remove

- Remove this stage altogether. It is replaced by the Test and Coverage Stage above, which will run all Python tests.

Zip Artifacts - New

- Zip up all the Python files (i.e., all files with a .py extension) into a zipfile called app.zip
    - You'll need to install zip on your Jenkins docker image
- Make that available as an artifact in the Jenkins build job for download
    - Look up the archiveArtifacts step in Jenkins

Your pipeline in Jenkins should look like this when successfully run:



Commit your changes to your Git project.

<u>Part 2 – Create a Shared Library for Your Point Project</u>

Create a new Project in GitLab called "ci_functions".

Create a subdirectory called "vars".

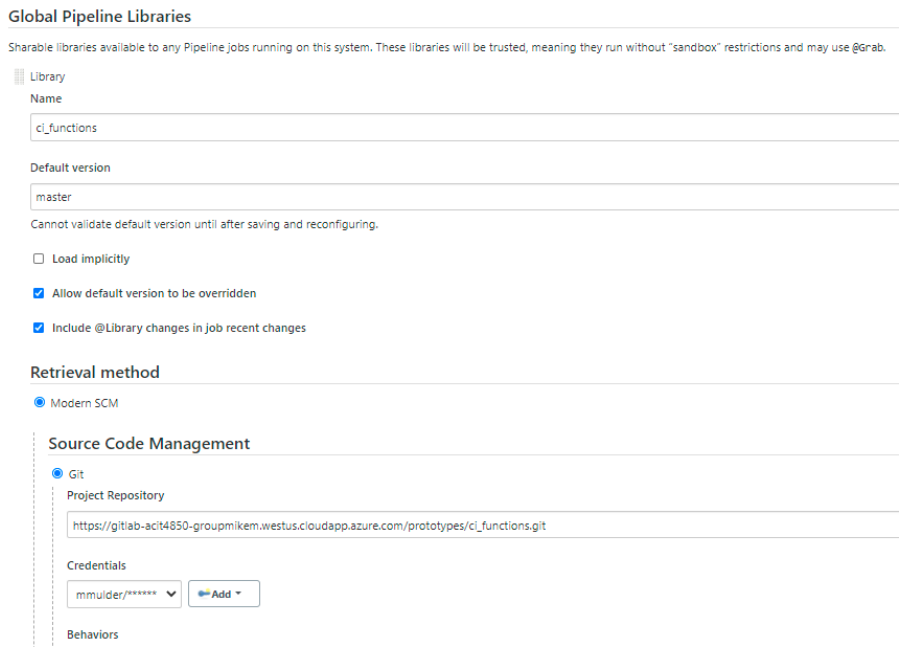
Create a file called python_build.groovy inside the vars directory.

Put your updated pipeline from Part 1 into the python_build.groovy file and make it callable.

```
1  def call() {
2
3      // Your Pipeline Here
4
5  }
```

Commit the code to the ci_functions project.

Update your Jenkins installation (i.e., Manage Jenkins -> Configure Jenkins -> Global Pipeline Libraries) so that ci_functions is a Global Pipeline Library. (on the master branch).



Update the Jenkinsfile in Point project to call the pipeline in python_build.groovy.

Test and make sure the Point project still builds.

Part 3 – Add a New Python Project Using the Same Pipeline Shared Library

Download the CarLotSample.zip file form Week 7 on D2L.

Create a new project in GitLab called CarLot and push the code into the repo.

- Add a requirements.txt file with specific versions of Flask, SQLAlchemy, Xmlrunner and requests. Push this file to the repo.
- Add a ci/Jenkinsfile that includes your ci_functions library and calls the pipeline in python_build.groovy.
- You will need to update the unit test to generate xml based unit test results.

Create a new CarLotPipeline pipeline build job in Jenkins for your new CarLot project that uses the ci/Jenkinsfile from the repo. Test out this build job and make sure it builds successfully for all the stages (there likely will be issues you need to fix). Fix any issues.

Make sure to setup a Webhook for this repo/job as well so it automatically builds on a push to the repo.

Make sure you shutdown any Azure resources (i.e., the VM) to conserve your credits and free tier usage.

Demo, Grading and Submission

A demo of your lab against the applicable requirements which will determine your grade on the lab. All mandatory requirements must be met otherwise you will receive zero on the lab. You can re-demo the lab if you haven't met the mandatory requirements, up to the last class before the midterm week, but you will lose 20% every week late.

Reqt.	Mandatory	Demo	Marks
REQ1150	Yes	• PointPipeline is triggered by pushes to the corresponding GitLab repo. • CarLotPipeline is triggered by pushes to the corresponding GitLab repo.	2

REQ1160	Yes	• PointPipeline
		• All four pipeline stages for PointPipeline:
		  ○ Build
		  ○ Lint
		  ○ Test (including Coverage)
		  ○ Package
		• **The Python steps are run in a Virtual Environment**
		• The app.zip file is an artifact
		• Uses the python_build.groovy shared library
		• Shows the correct output in the Test graph
			4

REQ1170	Yes	• CarLotPipeline
		• All four pipeline stages for PointPipeline:
		  ○ Build
		  ○ Lint
		  ○ Test (including Coverage)
		  ○ Package
		• **The Python steps are run in a Virtual Environment**
		• The app.zip file is an artifact
		• Uses the python_build.groovy shared library
		• Shows the correct output in the Test graph
			4

Total			**10**

Submit your python_build.groovy file to the Lab 6 dropbox on D2L.