ACIT 4850 – Lab 12 – Extending Your GitLab CI Python Pipeline

Instructor	Mike Mulder (mmulder10@bcit.ca)
Total Marks	10
Due Dates	Demo and code/screenshots prior to the end of next class: • April 8th for Set C • April 9th for Set B • April 11th Set A

Purpose

This is a continuation of the experimental Lab 8. We are comparing GitLab CI with Jenkins and want to make the GitLab CI pipelines more equivalent to our current Jenkins Python pipelines for both the Point and Carlot projects.

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.

You will need the following three applications running for this lab:

- GitLab
- Jenkins/Apache (for comparison purposes)
- DockerHub (Cloud)

Prerequisites

- Docker is installed on your GitLab VM
- You have a GitLab CI runner for your Python pipelines:
    - (Option 1) You have a Shell Runner for your Python pipelines, with Python and Pylint dependencies installed on the host VM
    - (Option 2) You have a Docker Runner for your Python pipelines, with Python and PyLint dependencies installed on a custom Docker image and mapping of Docker commands to the host VM (like we did for the Jenkins image or using a DID base image).
- You have your existing Python pipeline for the Point project in your GitLab

Part 1 – Adding the Package and Deliver Stages

Add the following two stages (and associated jobs) to the gitlab-ci.yml file for your Point project:

- Package
    - Add a rule so it only runs on the main/master branch
    - Create CI/CD variables in your Point project (Settings -> CI/CD, Variables) for your DockerHub credentials (username and password). Ensure their values are hidden and masked in the job output.
    - Add your Docker commands to login to DockerHub, build the image and push to DockerHub. Note: Same as the Jenkins steps for the Package stage

- Deliver
    - Add a rule so it only runs on a deployment
    - Create a UI CI/CD variable that user can set when they manually run the pipeline. It should default to NOT run the deployment
    - Add your Docker commands to run your image. Note: Same as the Jenkins steps for the Deliver stage.

Verify that your updated pipeline works for the following cases:

- On a merge request
- On a manual deployment

Part 2 – Template for Reuse in Carlot

Create a GitLab CI job template in another project (i.e., ci_functions) that includes the key jobs for your Python pipeline, including at minimum the Package and Deliver jobs. You will need CI/CD variables to define the configuration aspects of these jobs.

Include the templated jobs in your Point GitLab CI pipeline. Create an identical GitLab CI pipeline for the Carlot project.

So you will have gitlab-ci.yml files for both the Point and Carlot projects in your GitLab installation. They should have the same stages and jobs and leverage your templated jobs. And they should be functionally equivalent to your Jenkins pipelines for these two projects.

Demonstration and Submission

Like Lab 11, demonstrate a Merge Request and Deployment on both the Point and Carlot projects. Make sure the correct stages/jobs are run based on your rules.

In addition:

- Show that your gitlab-ci.yml files use your CI/CD variables and templated jobs.
- Verify that your DockerHub credentials are hidden in the console output for the Package stage.

Submit the following files:

- gitlab-ci.yml file for the Point project
- Screenshot of the pipeline without the Package and Deploy stages run
- Screenshot of the pipeline with the Package and Deploy stages run