ACTI 4850 – Lab 5 – Simple Python CI Pipeline

Instructor	Mike Mulder (mmulder10@bcit.ca)
Total Marks	10
Due Dates	Demo Due by End of Next Class: • Feb. 12th for Set C • Feb. 13th for Set B • Feb. 15th 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 (future).
- **SEC1020** – All web applications and API endpoints shall be encrypted (i.e., https endpoints). Note: We will use a signed certificate to better emulate a production-like environment.

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.

Step 1 – Upgrade Your Apache Reverse Proxy to a Self-Signed Certificate

Make sure your Apache VM is running. You are going to convert your self-signed certificate to a signed-certificate from LetsEncrypt.

SSH into the VM. First verify the following:

- Your 000-default.conf file has the correct ServerName defined. The ServerName should be set to the DNS of your Apache VM. Your signed certificate will apply to this domain.
- Your default-ssl.conf file should have the identical ServerName as the 000-default.conf

```
ServerName <Your Apache VM DNS Name>
```

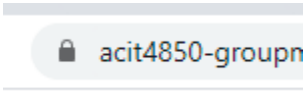
Run the following commands to install Certbot, which we will use to retrieve a LetsEncrypt signed certificate for us:

- sudo apt install certbot python3-certbot-apache

Run the following comments to object your signed SSL certificate for the domain defined by the ServerName in your conf files:

- sudo certbot --apache

    - If may ask you whether to redirect HTTP traffic to HTTPS. Since we've setup the redirect already, you can select No Redirect (option 1)
- Your signed certificate should now be downloaded, installed and loaded.
- Verify that you can access your sites through the reverse proxy. The Not Secure indicator next to the URL should now be a Padlock (in Chrome). If you click on it you should now have a valid certificate.



Reference: https://www.digitalocean.com/community/tutorials/how-to-secure-apache-with-let-s-encrypt-on-ubuntu-20-04

Step 2 – Update Your Point Project in GitLab

In Python, a requirements.txt defines the set of Python packages and their versions required by your program. Add a requirements.txt file to your Point Project with the following content:

```
SQLAlchemy==1.3.23
Flask==2.1.0
Xmlrunner==1.7.7
```

Add a folder to your project called 'ci'.

Create a Jenkinsfile in the ci folder as follows:

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                sh 'pip install -r requirements.txt --break-system-packages'
                sh 'pip install --upgrade flask --break-system-packages'
            }
        }
        stage('Unit Test') {
            steps {
                sh 'python3 -m unittest test_point_manager'
             }
        }
    }
}
```

Commit your changes to your Git project.

Step 3 – Create Your Jenkins Build Pipeline

In Jenkins, create a new Pipeline type job called PointPipeline (Dashboard -> New Item)

Under the Pipeline section of the job configuration:

- Selected Definition as 'Pipeline script from SCM'
- Select your SCM as Git and configure it for your Point repository in Git using valid credentials
- Set the Script Path as 'ci/Jenkinsfile'

Test your pipeline build job (using Build Now) and troubleshoot any issues until it builds successfully.

Step 4 – Add Test Results

Jenkins only supports Java junit based test results. Modify the Python unit tests in your GitLab project as follows to produce junit style test results:

- In test_point_manager.py, add "import xmlrunner"
- Add the following to the end of test_point_manager.py:

```
if __name__ == "__main__":
    runner = xmlrunner.XMLTestRunner(output='test-reports')
    unittest.main(testRunner=runner)
    unittest.main()
```

This will run the tests once to generate the test-reports and then again for the normal Python unit test output.

Update the Unit Test stage in your Jenkinsfile with the updated shell command and a post action as follows:

```
        stage('Unit Test') {
            steps {
                sh 'python3 test_point_manager.py'
             }
            post {
                always {
                    junit 'test-reports/*.xml'
                }
            }
        }
```

Commit your changes to your Git project. Re-run your build job in Jenkins and troubleshoot any issues until it builds successfully. Make sure the unit test results are reported. You should see a link to view the unit tests results and a graph showing the trends in unit tests results (i.e., pass vs failures).

Step 5 – Add Integration Tests

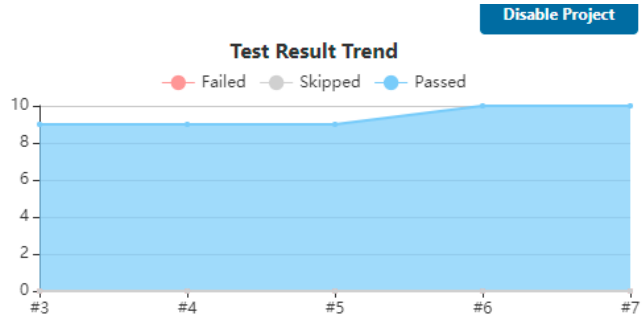
Add a new file called test_points_api.py to your GitLab project with the following structure:

```
import unittest
import points_api
import xmlrunner
import os
import inspect

from point_manager import PointManager
```

```
from sqlalchemy import create_engine
from base import Base

class TestPointApi(unittest.TestCase):

    def setUp(self):
        engine = create_engine('sqlite:///test_points.sqlite')

        # Creates all the tables
        Base.metadata.create_all(engine)
        Base.metadata.bind = engine

        points_api.point_mgr = PointManager('test_points.sqlite')

        points_api.app.testing = True
        self.app = points_api.app.test_client()

        self.logPoint()

    def tearDown(self):
        os.remove('test_points.sqlite')
        self.logPoint()

    def logPoint(self):
        currentTest = self.id().split('.')[-1]
        callingFunction = inspect.stack()[1][3]
        print('in %s - %s()' % (currentTest, callingFunction))

    def test_points_all(self):
        rv = self.app.get('/points/all')
        self.assertEqual(rv.status, '200 OK')

if __name__ == "__main__":
    runner = xmlrunner.XMLTestRunner(output='api-test-reports')
    unittest.main(testRunner=runner)
    unittest.main()
```

Add a new stage to your Jenkinsfile. It should be identical to the Unit Test stage but it is called "Integration Test" and runs the integration tests in test_points_api.py. Make sure the test results for these new tests show up after you run the job. It should get the results from api-test-reports (not test-reports) – so make sure to change 'test-reports/*.xml' to 'api-test-reports/*.xml'

Your PointPipeline should have a Test Results Trend graph that looks something like this:



And if you view the Test Results for the last build job (Job # -> Test Results), it should look something like this:

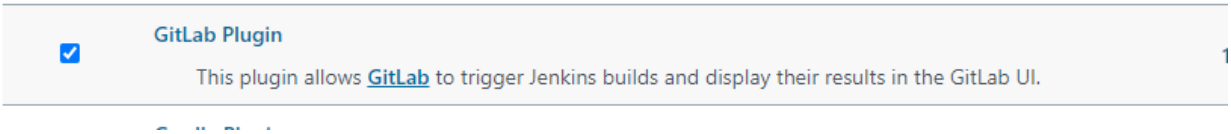
All Tests

Class	Duration	Fail	(diff)	Skip	(diff)	Pass	(diff)	Total	(diff)
TestPointApi	62 ms	0		0		1		1	
TestPointManager	0.41 sec	0		0		9		9	

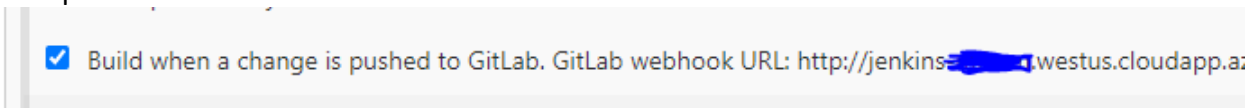
Step 6 – GitLab Triggers

On Jenkins:

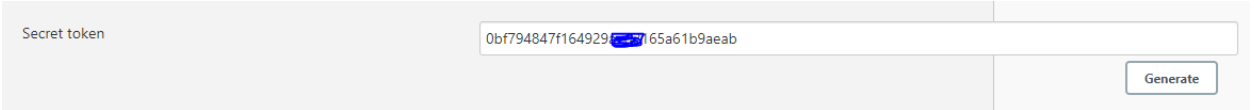
- Make sure you have the GitLab Plugin installed (Manage Jenkins -> Manage Plugins)
  - It should show up in the Installed plugins tab.



  - If not, go to the Available plugins tab, find it and install it.
- Go to the Configure page of your PointPipeline job.
  - Under Build Triggers sectio select the **Build when a change is pushed to GitLab** option**.**



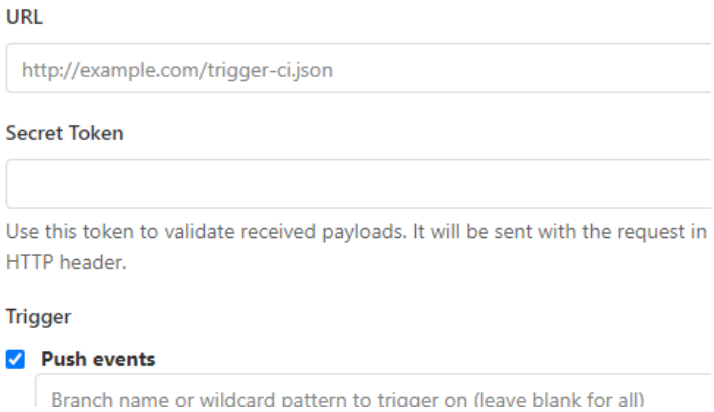
  - Make note of the URL value (you'll need this shortly).
  - Select the Advanced button for this option.
  - Generate a Secret Token. Make note of this value (you'll also need it soon)



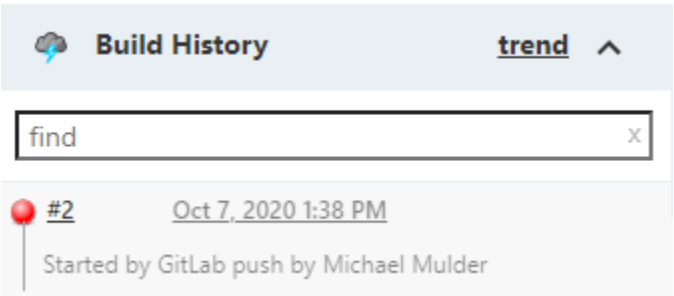
  - Save your configuration.

On GitLab:

- Go to the Point project
- On the left, select Settings -> Webhooks
- Use the URL and Secret Token from Jenkins to populate the URL and Secret Token fields



- Make sure Push events is checked
- Since we are now using a *signed certificate*, make sure the Enable SSL Verification option is checked.
- Select Add webhook and then test it out. A Jenkins job should be triggered (note: this may take a few seconds to show up).



Step 7 – Add Additional Integration Tests

Add the additional integration test below to test_points_api.py and verify that a Jenkins build is triggered each time you push your changes to your repository:

```
def test_points_all(self):
    rv = self.app.get('/points/all')
    self.assertEqual(rv.status, '200 OK')

def test_post_get_point(self):
    rv_post = self.app.post('/points', json={ "x": 5, "y": 5},
                            headers={"content-type":
                                     "application/json"})
    self.assertEqual(rv_post.status, '200 OK')

    rv_get = self.app.get('/points/all')
    self.assertEqual(rv_get.status, '200 OK')
```

The number of tests in the Test Result Trend graph and Test Results for the build job should have increased for the new integration test.

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	• Show your GitLab configuration for the integration with Jenkins (1 mark) • Show your Jenkins CI pipeline job configuration for integration with GitLab (1 mark) • Demonstrate a push to your GitLab project triggers your CI pipeline job in Jenkins (1 marks)	3

REQ1160	Yes	• Demonstrate your completed CI pipeline that includes the following stages (3 marks):
		  ○ Build
		  ○ Unit Test
		  ○ Integration Test
		• Your CI pipeline completes successfully (all stages) (1 mark)
		• Your Jenkinsfile is in your GitLab project and used by the CI Pipeline (1 mark)
		• The source code from GitLab is being used by your CI Pipeline (1 mark)
			6

SEC1020	No	• The SSL Certificate for your reverse-proxy is now signed and shows as valid in your browser.	1
Total			**10**

Submit your Jenkinsfile to the Lab 5 dropbox on D2L.