ACIT 4850 – Lab 10 – Build Parameters, Continuous Delivery and Merge Request

Instructor	Mike Mulder (mmulder10@bcit.ca)
Total Marks	10
Due Dates	Demo and code/screenshots prior to next class: • April 8th (or earlier) for Set C • April 2nd for Set B • April 4th for Set A

Applicable Requirements

- **(Existing) REQ1200** – The Enterprise Development Environment shall use containerization to package applications for deployment to a target environment.
- **(Existing) REQ1230** – The Enterprise Development Environment shall hide and encrypt any passwords or secrets used in the pipeline builds.
- **(New) REQ1210** – The Enterprise Development Environment shall require code reviews of all new features prior to integration into the main branch.
- **(New) REQ1220** – The Enterprise Development Environment shall have build pipelines that allow container artifacts to optionally be deployed to enable testing by the test team.

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:

- Apache
- GitLab
- Jenkins
- SonarQube (can skip but you need to modify your Java build so it doesn't report analysis to SonarQube)
- DockerHub (Cloud)

Part 1 – Java Sample – Add Parameter and Update Delivery Stage

Update the Jenkinsfile for your Java Sample project as follows:

- Add a boolean parameter to enable/disable the Delivery stage of the pipeline. This should go between the agent and stages keywords.

```
agent any

parameters {
    booleanParam(defaultValue: false, description: 'Deploy the App', name:
'DEPLOY')
}
```

```
stages {
```

The defaultValue is used when no parameter value is specified, such as when the build is triggered by a Webhook in GitLab. However, when the user manually triggers the build in Jenkins, they will be prompted to set the value of the DEPLOY parameter.

Note that the "Build Now" option in Jenkins will change to "Build with Parameters".

- Add a when block to the Deliver stage of the pipeline with the following expression:

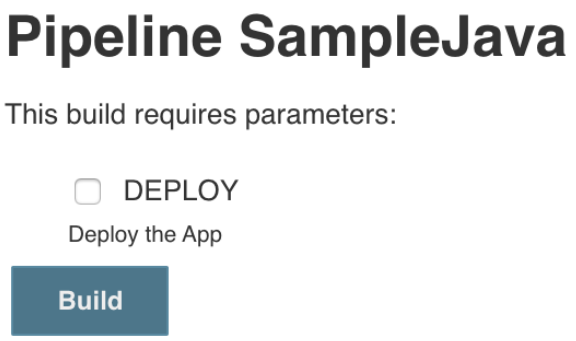
```
expression { params.DEPLOY }
```

The Deliver stage will now only run if the DEPLOY parameter is set to true.

- Update the Deliver stage to run your Docker image as a container rather than the deliver.sh script.

```
sh "docker run myapp:latest"
```

- Run your build in Jenkins using Build Now. It will take one run of the build before Jenkins automatically picks of up the parameters. When it does, "Build Now" changes to "Build with Parameters".



- Run the build with DEPLOY set to true and with DEPLOY set to false to make sure the Deliver stage is only run with DEPLOY is set to true.
    - When the DEPLOY is set to true, it should show the output of the Docker run of the myapp:latest container (i.e., Hello World!) in the Console Output
- Take a screenshot of the build pipeline stages in Jenkins for consecutive runs with the DEPLOY false and then the DEPLOY set to true. This is the green stage blocks on the main job pages. (**Make sure to show the URL of your Jenkins server in the screenshot**)

Part 2 – Python Shared Library Updates (Point and CarLot Pipelines)

In your shared pipeline (python_build.groovy), make the same changes as the Java Sample for:

- Adding the DEPLOY parameter as above in Part 1
- Add a new function parameter called portNum to the call function:

```
def call(dockerRepoName, imageName, portNum)
```

Add a new final stage to the pipeline called Deliver that has the following functionality:

- Is only run if the DEPLOY parameter is true
- Runs the following two docker commands as shell commands

```
docker stop ${dockerRepoName} || true && docker rm
${dockerRepoName} || true
```

```
docker run -d –p ${portNum}:${portNum} --name ${dockerRepoName}
${dockerRepoName}:latest
```

The first command tries to stop and remove an instance of the container named ${repo} if it is running. Since a shell command will fail the build if the command returns an error, we return true if the command fails because the container is not already running (i.e., if this is the first time run or and container fails).

The second command runs the ${dockerRepoName}:latest image as a Docker container and names it as ${dockerRepoName} on port ${portNum}

Update the Jenkinsfiles for the CarLot and Point Pipelines with the correct port number argument to python_build:

- Port for CarLot will be 5000, i.e., python_build('carlot', 'carlotapp', 5000)
- Port for Point will be 3000 (we will adjust the code in Part 3)

In the car_api.py file of the CarLot repo, update the app.run() to as follows:

```
app.run(host='0.0.0.0', port=5000)
```

Test your CarLot Pipeline build and verify the following:

- Note: You will have to run it twice for the parameter to be picked up
- The DockerHub password (i.e., access token) is hidden in the console output
- The Deliver stage is only run with the DEPLOY parameter is set to true (i.e., checked)
- The Docker container is running on your Jenkins server. SSH into the Jenkins server and run “docker ps” to verify that they are running.

Take screenshots of the following for CarLot Python pipeline:

- The build pipeline stages in Jenkins for consecutive runs with the DEPLOY false and then the DEPLOY set to true. This is the green stage blocks on the main job pages. (**Make sure to show the URL of your Jenkins server in the screenshot**)
- Your console output showing the hidden password for the docker login command.

- The output of “docker ps” showing the Docker container running.

Make sure the following wget command on your Jenkins VM returns a 200 OK response from the CarLot application:

```
wget http://localhost:5000/cars/all
```

Part 3 – Merge Requests (Point Pipeline)

Update your Point repository and Jenkins job to build branches associated with Merge Requests.

In GitLab:

- Update your Webhook to be triggered by Merge Requests (there is a checkbox in the Webhook to enable this).
- Also filter the Push Requests trigger to only apply to main (put main in the text box under the Push events checkbox).

In Jenkins:

- Select Configure for the PointPipeline job in Jenkins
- Under Pipeline -> Definition -> SCM find the Branches to Build section.
- Replace ***/main** with ***/*** so that this build with work on any branch.
- Save the Configuration

With your partner exercise a Merge Request as per the instructions from GitLab:

- https://docs.gitlab.com/ee/user/project/merge_requests/

The two changes you want to make for your merge requests are as follows:

1. In the points_api.py file of the Point repo, update the app.run() to as follows:

```
app.run(host='0.0.0.0', port=3000)
```

2. Update the create_tables.py as follows:

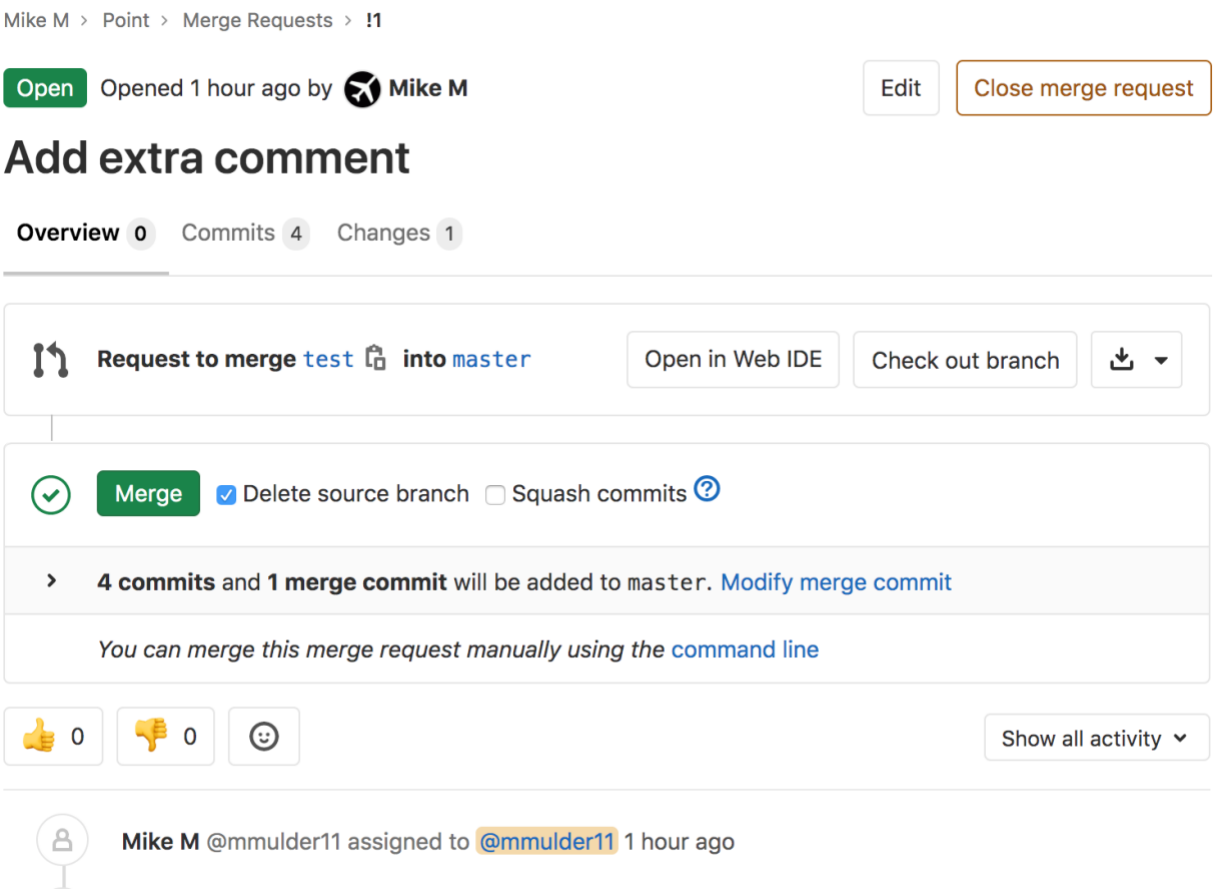
create_tables.py 263 Bytes

```
1  import sqlite3
2
3  conn = sqlite3.connect('points.sqlite')
4
5  c = conn.cursor()
6  c.execute('''
7            CREATE TABLE point
8            (id INTEGER PRIMARY KEY ASC,
9             x INTEGER NOT NULL,
10            y INTEGER NOT NULL)
11           ''')
12
13 conn.commit()
14 conn.close()
```

If you use Git on the command line, the basic steps are as follows:

- Go to the folder of your Point repo
- Make sure you are on main
- Perform the following:
    - git checkout –b <branch name>
        - Note select a unique branch name
    - Make a change to a file
    - git add .
        - This add all changed files to a staged commit
    - git commit -m “<Comment Message>”
        - This commits the staged files locally
    - git push origin
        - This pushes the commit to the GitLab server
        - Note: If this is the first commit on the branch, GitLab will ask you perform a different git push. Just follow the instructions it provides on the command line.

Once your branch is on the GitLab server, create a Merge Request.



Have your partner (if applicable) review the changes and merge them into main (select the Merge option).

Make sure the following wget command on your Jenkins VM returns a 200 OK response from the Point application:

```
wget http://localhost:3000/points/all
```

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

Submission

Post the following to the Lab 10 dropbox on the Learning Hub:

- (4 marks) Two screenshots of the Java and CarLot points (showing the URL containing your Group #)
- (3 mark) Updated Jenkinsfile for the Java Sample pipeline
- (3 mark) Updated python_build.groovy for the Python pipelines
- (5 marks) **Demo the following:**
    - A merge request on the Point pipeline (using **command line Git** to make your code changes)
    - Show that your carlot and point applications are running as Docker containers on your Jenkins VM
    - You get a 200 OK response on an endpoint for each:
        - `wget http://localhost:3000/points/all`
        - `wget http://localhost:5000/cars/all`