SENG202 – Software Engineering Project Workshop 2020

Tutorial 5 – Continuous integration with GitLab

1. Introduction

By now you should be familiar with the rationale of automated testing in software engineering. Over time a project will accrue a large number of tests which together take a long time to run. While you should always locally run a subset of your organisation's tests before pushing code to the repo it will not be practical to run the full suite frequently (or perhaps at all) on your development machine. Continuous integration (CI) is a system whereby, after each push, all tests in the project are run to ensure that the changes have not broken something else in the project. For this we use GitLab CI. Your SENG202 project is unlikely to ever get large enough to reap the full benefits of a CI system but using the tool will give you experience that you will need in industry. It will also act as a safety net for the times when team members forget to run tests locally before pushing broken code.

Note: This lab can be completed by a single team member who has at least maintainer access to your eng-git repository, but your entire team should be present and paying attention for the setup process to avoid creating a single point of failure.

Each team will use their own server, this server is used for building your project – it will behave similar to your own workstations when building your project.

Objectives:

- Understand what GitLab CI is, and the problems it aims to solve
- Set up GitLab CI to work with your team's SENG202 repository (refer to tutorial 2 regarding git)
- Complete CI quiz on the Quiz Server. The quiz will be open in the second half of the tutorial time, and close one hour after the end.

At the end of this tutorial you should also be able to complete the project setup checklist. If you are having difficulty with any of these tasks, please contact the tutors.

2. Command line access via SSH

For this tutorial, you will require command line access to your server. For this we will use SSH. In a shell, enter the following command: **`ssh [username]@[host]`**

- Username: `sengstudent`
- Host: csse-s202g[team_number][1].canterbury.ac.nz
- Password: Ask one of the tutors for this
- Port: 22 (for those using other tools or curious minds)

If you are using Windows, you may want to use a third party program called Putty: http://www.putty.org. You can also use the MinGW shell bundled with the popular Git for Windows package: https://gitforwindows.org/ It should be noted that we recommend using the lab PCs running Linux, as opposed to using personal laptops. You're more than

[1] Starting on 1, not 01. For instance, Team **1**: csse-s202g**1**.canterbury.ac.nz; Team **2**: csse-s202g**2**.canterbury.ac.nz .....

welcome to use your own PCs, however your project must build and run on the lab machines, so it pays to test on these frequently.

3. SSH Public Key Authentication (optional, but useful)

SSH public key authentication is a more secure way of logging into a server using SSH as no passwords are exchanged. It works using a pair of RSA keys, one being a private key and the other a public key which are stored in plain text. The keypair is used to identify a user, and as naming conventions go the private key should be ***kept private***, and your public key can be made public. Keypairs are stored in your home directory under `~/.ssh` and are commonly named `id_rsa` for your private key and `id_rsa.pub` for your public key. If you have not got those files, then you must generate them before you can use public key authentication.

To generate an RSA keypair, type `` `ssh-keygen -b 2048` `` into a command line ***on the machine you wish to connect to your Virtual Machine from***. Press enter to use the default names and enter a passphrase if you want. The passphrase is used to encrypt the private key in your `~/.ssh` directory. You do not need a passphrase if you don't want to enter it every time you are connecting to your Virtual Machine. If all goes well, you should see an output similar to that shown in Figure 1 below.

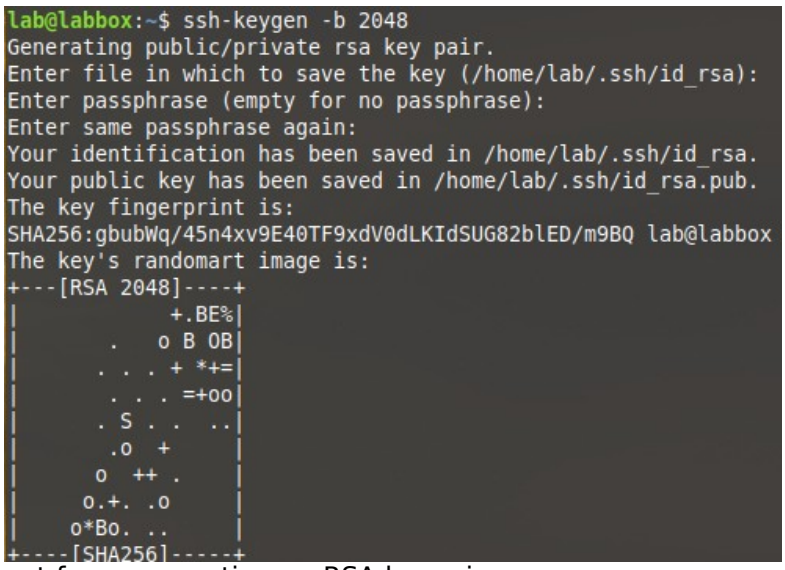

Figure 1: Output from generating an RSA keypair

Now that you have generated a keypair, you can use this to login to your Virtual Machine. But how does the authentication work? Well, when you try to login to your Virtual Machine the SSH server running on it will generate a challenge based on the public key it has available. Only the matching private key can decrypt the challenge and allow the connection to continue. Public keys which are authorised to login to the Virtual Machine are stored in the `~/.ssh` directory of the user you are trying to login as, in a file called `authorized_keys` with each public key being on a new line. In your case, this will be located at the following path `/home/sengstudent/.ssh/authorized_keys`.

Now that you know how authentication works, you need to share your ***public key*** with the Virtual Machine so it can authenticate you. To do this you can either copy the contents of your public key (on your machine) and paste them into the `authorized_keys` file on the Virtual Machine, or by using the `` `ssh-copy-id` `` command. We will use the command as it handles creating the `authorized_keys` file and adding multiple keypairs.

To do this type `ssh-copy-id sengstudent@csse-s202g[team number].canterbury.ac.nz` into the command line ***on the machine you wish to connect to your Virtual Machine from.*** You will need to replace `[team number]` with your team number so you are not connecting to the wrong Virtual Machine. You will need to enter the password for the Virtual Machine to initiate this connection as the public key is not available to it yet. If all goes well, you should see an output similar to Figure 2 below.

```
lab@labbox:~$ ssh-copy-id sengstudent@csse-s202g0.canterbury.ac.nz
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompt
ed now it is to install the new keys
sengstudent@csse-s202g0.canterbury.ac.nz's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'sengstudent@csse-s202g0.canterbu
ry.ac.nz'"
and check to make sure that only the key(s) you wanted were added.
```

Figure 2: Output of running the `ssh-copy-id` command

Now that your public key has been copied onto the Virtual Machine, you can try logging in as suggested in Figure 2 above. Note that we used seng202g0 as an example, the URL for your machine will be different.

4. Maven

Since your SENG202 projects will be built using Maven, you will need to check to ensure Maven is installed on your Virtual Machine. The VM's should all come with Maven pre-installed, but you shouldn't run into this process without verifying that the correct version is installed. To do so type `mvn -version` into the command line.

If everything went well, you should see an output similar to that shown in Figure 3 below and the maven and JDK versions should match what you have installed on the lab machines.

```
sengstudent@csse-s202g0:~$ mvn -version
Apache Maven 3.6.0
Maven home: /usr/share/maven
Java version: 11.0.8, vendor: Ubuntu, runtime: /usr/lib/jvm/java-11-openjdk-amd64
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.15.0-101-generic", arch: "amd64", family: "unix"
sengstudent@csse-s202g0:~$
```

Figure 3: Output of running `mvn –version` on Virtual Machine

5. Maven, GitLab CI, GitLab Runner – How it all fits together

Let's understand the architecture first. There are three components to be aware of, some of them you may already be familiar with: Maven, GitLab CI, and GitLab Runner.

GitLab CI is a component of GitLab (think eng-git). This is the first port of call when you push to your repository. GitLab CI examines what you have pushed and determines what (if any) pipelines need to be run. It then sends this information over to an available GitLab Runner that is associated with your project.

A GitLab Runner is a separate component that is purely responsible for running pipelines and reporting the outputs back to GitLab CI. GitLab Runner follows the orders of GitLab CI.

Maven is used by GitLab Runner to determine *how* to build your project. Maven builds your project according to your pom.xml file. A general overview of the architecture is shown in Figure 4 below.

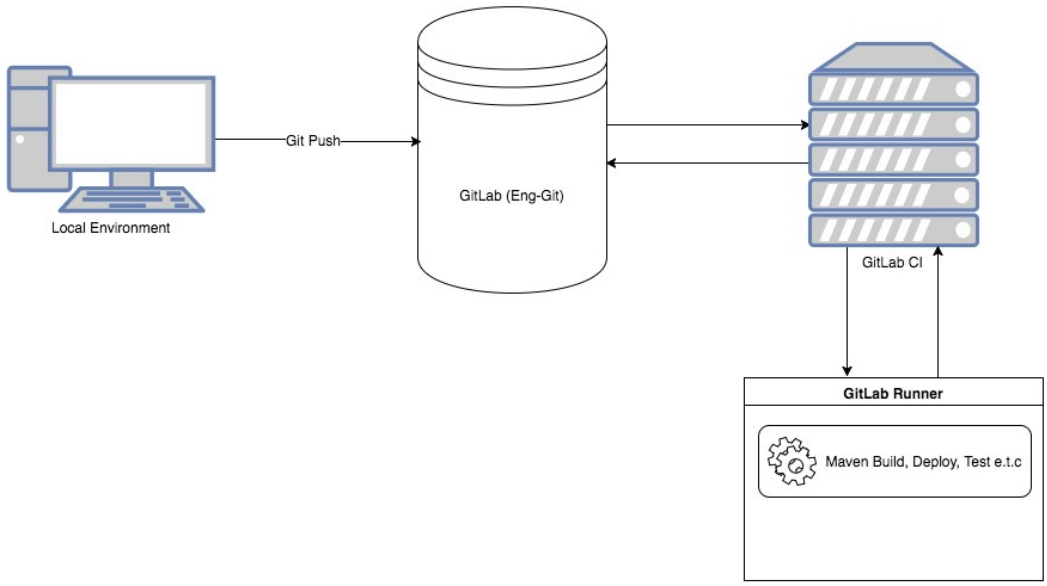

Figure 4: Architecture of GitLab CI

6. Setting up a GitLab Runner

Now that we have ensured Maven is correctly installed, it is time to register a GitLab Runner. Similar to Maven, GitLab Runner should be pre-installed on the Virtual Machine. Verify this by typing `` `sudo gitlab-runner status` `` into the command line. You should see an output similar to Figure 5 below, if not ask for help from a tutor.

```
sengstudent@csse-s202g0:~$ sudo gitlab-runner status
Runtime platform                                    arch=amd64 os=linux pid=13808 revision=21cb397c version=13.0.1
gitlab-runner: Service is running!
sengstudent@csse-s202g0:~$
```

Figure 5: Output of running `` `sudo gitlab-runner status` `` on Virtual Machine

Next, we need to tell it how to communicate with GitLab CI, and how to access your source code. For this, we require two pieces of information that you can retrieve from GitLab (https://eng-git.canterbury.ac.nz).

1. Log into Eng-Git and select your project repository
2. Click the **Settings** tab (At the bottom of the navigation bar on the left)
3. Select CI/CD
4. Expand the Runners section
5. Locate the Co-ordinator URL and Registration token as shown in Figure 6

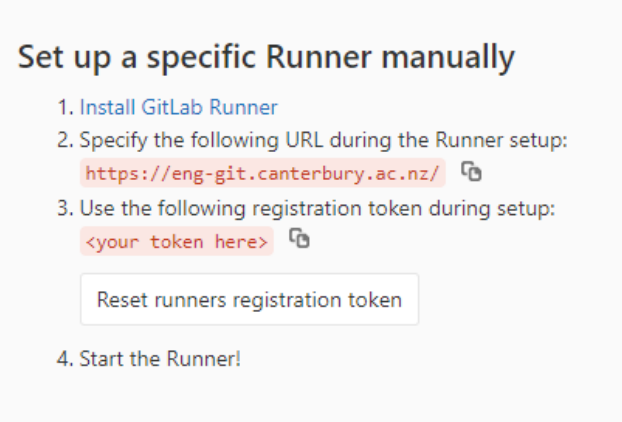

Figure 6: Runner Configuration Details

You can now proceed with registering your GitLab Runner with GitLab CI, by typing `**sudo gitlab-runner register**` into the command line.

When prompted, enter the following information:

- Co-ordinator URL: As above (from https://eng-git.canterbury.ac.nz)
- Token: As above (from https://eng-git.canterbury.ac.nz)
- Description: Anything sensible (e.g. `s202g[team number]-runner`)
- Tags: Skip this (just press enter)
- Executor: `shell`

You have now successfully configured a GitLab Runner. To verify, refresh the Runners page within GitLab – you should now see your runner listed, as shown in Figure 7.

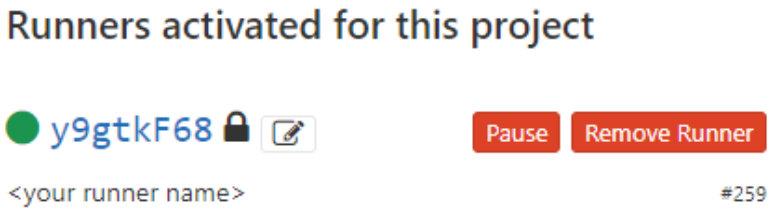

Figure 7: Newly created GitLab Runner showing within GitLab

7. Automatic Building

We are now finished working on your servers – go ahead and logout, either by typing logout, or just closing the terminal window.

Now we need to set up a GitLab CI pipeline configuration file. This tells GitLab CI what operations (jobs) to perform. To do this, create a new YML file `.gitlab-ci.yml` in the root of your repository (the same one we created/cloned in Tutorial 2). Copy the following code into your `.gitlab-ci.yml` file:

```
junit:
  stage: test
```

```
  script:
    - mvn -B clean test

generate_artifacts:
  stage: deploy
  script:
    - mvn -B clean package
  artifacts:
    paths:
    - target/[your_project_name]*.jar
```

Snippet 1: Basic `.gitlab_ci.yml` file specifying a junit and generate_artifacts job in their respective stages

This file defines a set of jobs with constraints stating when they should be run. The jobs are defined as top-level elements with a name (junit, and generate_artifacts in the above code), and always must contain at least the script attribute. When executing jobs, GitLab CI does exactly what you might think it does, it executes the command(s) in the script attribute.

Breaking it down further, you will also notice the “artifacts” attribute. The paths defined in here tell GitLab runner which files (if any) it should upload back to GitLab (eng-git in our case). You can add any files in here. Since the `.gitlab-ci.yml` file is located in the root directory of your repository, the paths in this file are relative to the root directory also (Maven will build from the root directory, which will create the target directory).

You will need to change this path to match your own build configuration by replacing [your_project_name] with the one you have specified in your pom.xml file under the `<artifactId>` tag. Note that the wildcard in the filename ensures that only the runnable jar is uploaded, not the original-* jar which does not contain any project dependencies or a manifest file.

Commit and push this file to git. Now go back into GitLab and go to the **Project** tab of your repository. You should see something similar to Figure 8.

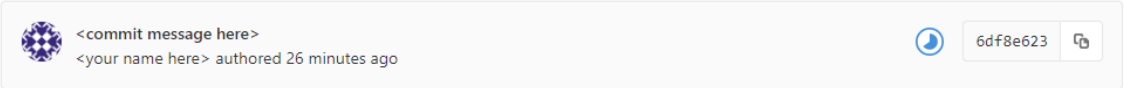

Figure 8: Build in progress

Clicking on **CI/CD** (left sidebar), is where you will find all the pipelines for your project. Clicking on an individual pipeline ID will take you to a page that shows you the status of the jobs in that pipeline. You should see two nodes in the pipeline “junit” and “generate_artifacts”, under their respective stages, as shown in Figure 9.

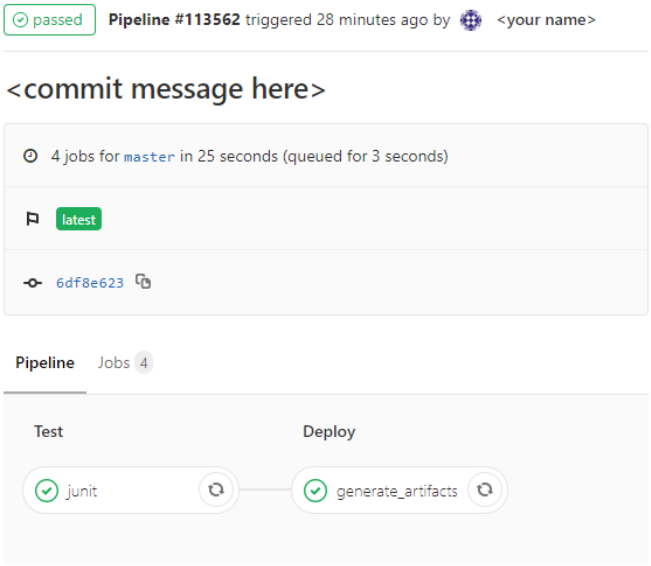

Figure 9: Pipeline page within GitLab

Clicking on one of the jobs will show us the job output, with the script being executed exactly as we defined in our `.gitlab-ci.yml` file for that job.

Remember the artifact path we specified? By clicking on the "generate_artifacts" job (at the bottom in Figure 9), you will see the results of this job. On the right-hand side of this page (as shown in Figure 10) you will see the "Job artifacts" section. This is where you can download the artifacts we specified in our `.gitlab-ci.yml` file. Download the .jar file and make sure you can run it.

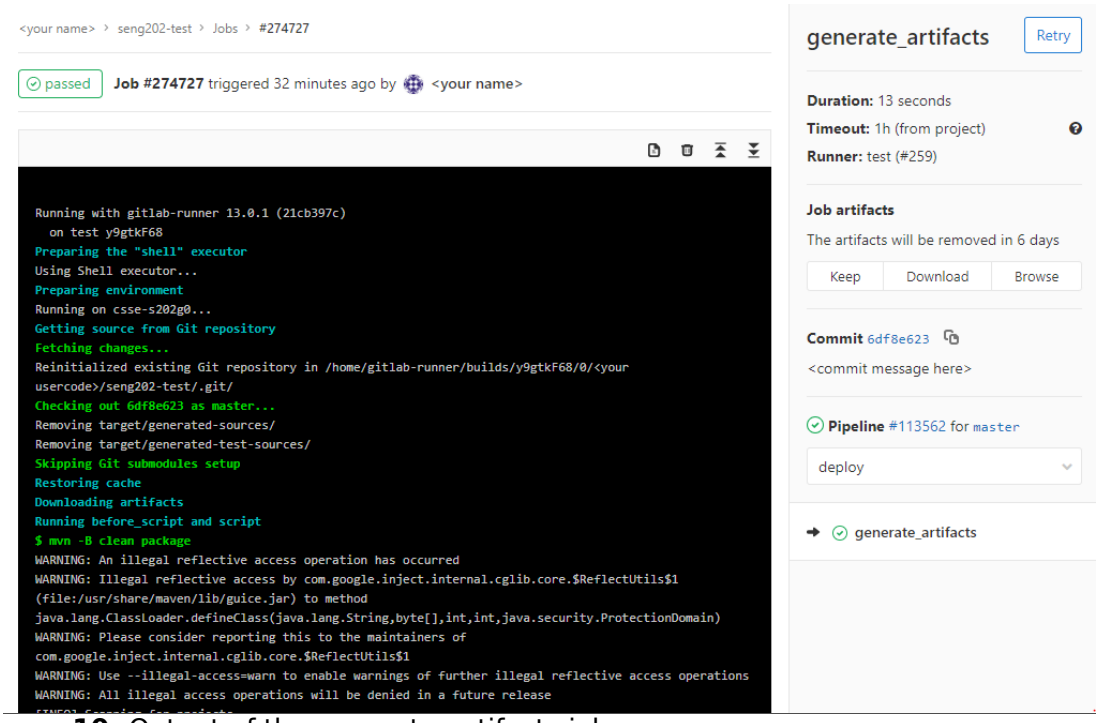

Figure 10: Output of the generate_artifacts job

8. Have a Play Around

Write a failing test, commit and push those changes.

What do you see now on GitLab? Hopefully you will see that the tests failed, looking further you should be able to see *which* tests failed.

9. Summary

So far, we have added a GitLab Runner, and set up automatic building for our project. Now, whenever someone in your team pushes to your repository, a new build will be started.

Make sure that your builds are passing under CI often. This is where the marking team will download your projects from. Your master branch should always pass, broken code shouldn't be committed to master.

10. Repository Access

In the second to last tutorial, you set up the Git repository for your project. You should have already added the teaching team to the project and have granted **Reporter** access (we need this for assessment and running your project). If you have not done so, then please do so now. The tutor's usernames are as follows (**note you need to add all**):

sjs227, lwa383 AND pid15