SENG 426

Software Quality Engineering

Summer 2021

University of Victoria

Delivery 2: Continuous Integration Setup

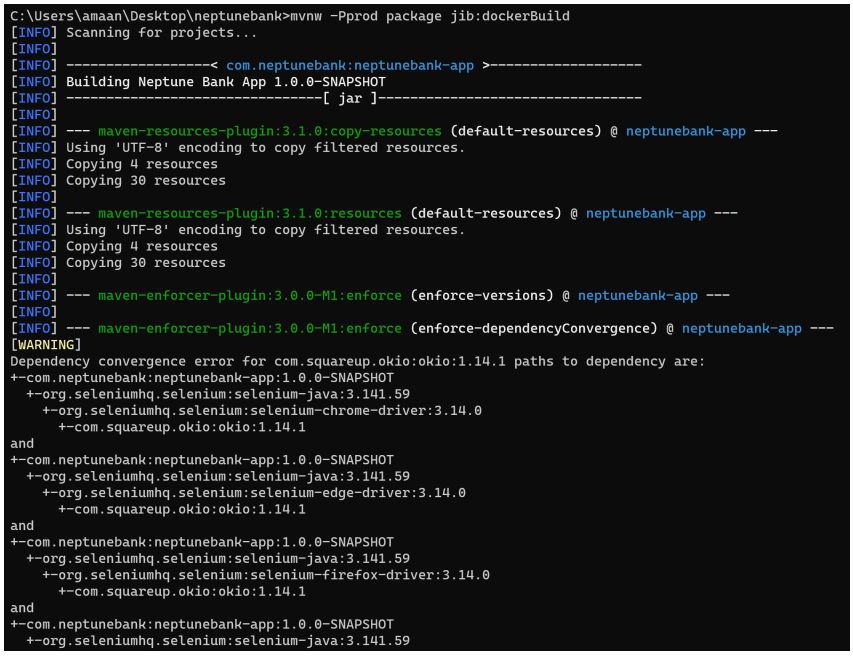
June 13th, 2021

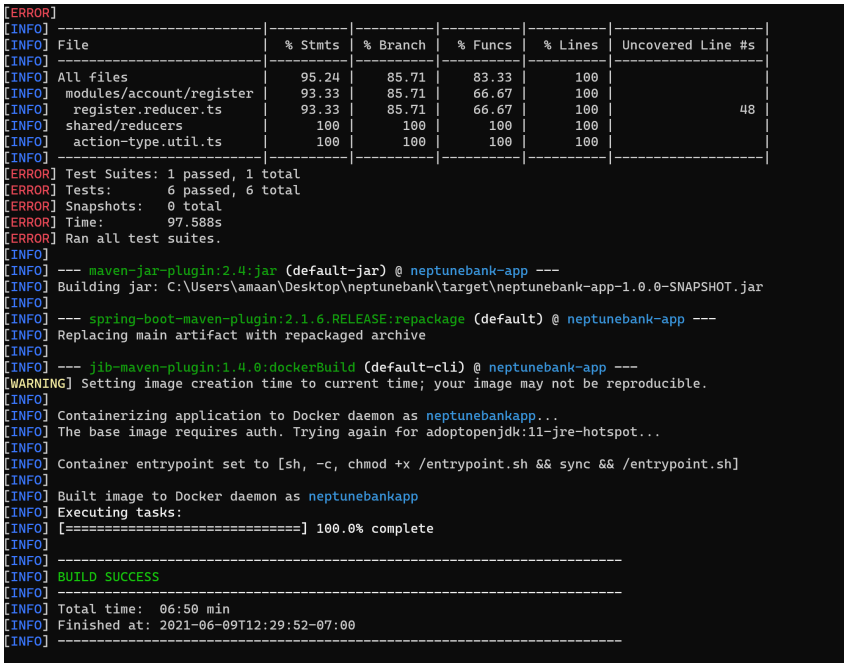
David Bishop	V00884250
Amaan Makhani	V00883520
Ben Austin	V00892013
Nolan Kurylo	V00893175
Liam Franke	V00887604
Evan Roubekas	V00891470

Table of Contents

Demonstration of the Docker Container Deployed

Since Docker cannot be used due to IT restrictions a step by step demonstration of it's deployment is shown below. Maven with Google's *Jib* is used to containerize the application. Jib allows us to build our local Docker installation in order to get our docker image. We use the command mvnw -Pprod package jib:dockerBuild to do this.





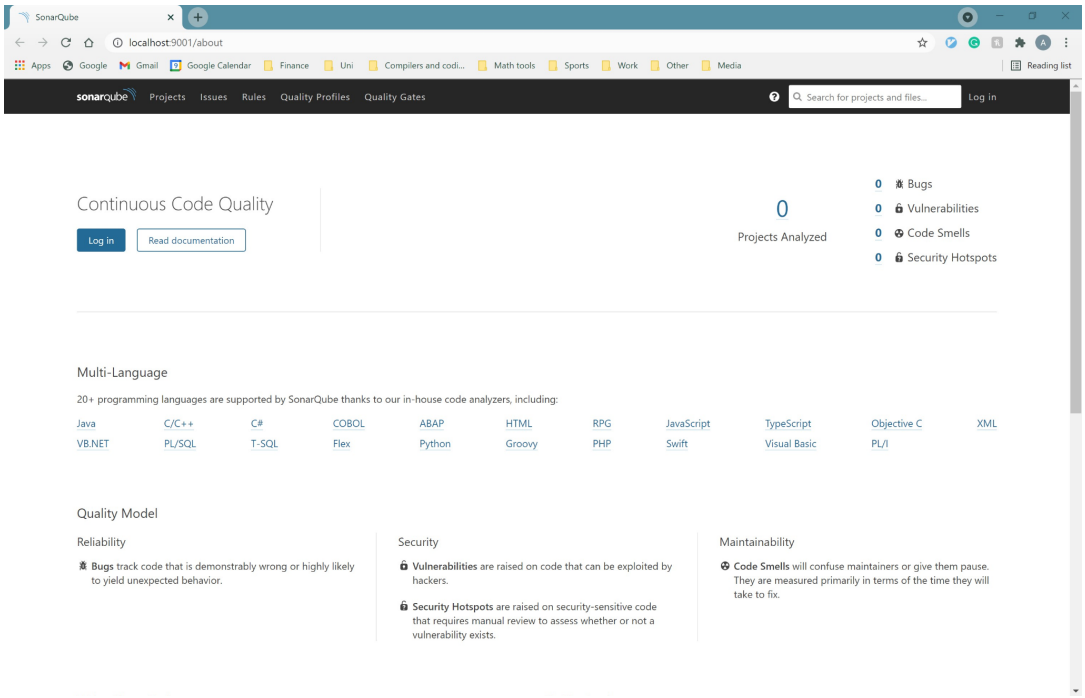
Using the image it can be launched using docker-compose as follows: *docker-compose -f src/main/docker/app.yml up -d*. This is shown below.

```
C:\Users\amaan\Desktop\neptunebank>docker-compose -f src/main/docker/app.yml up -d
Docker Compose is now in the Docker CLI, try `docker compose up`

WARNING: Found orphan containers (docker_neptunebank-sonar_1) for this project. If you removed or renamed this service in your compose file, you can run t
his command with the --remove-orphans flag to clean it up.
docker_neptunebank-mysql_1 is up-to-date
Creating docker_neptunebank-app_1 ... done
```

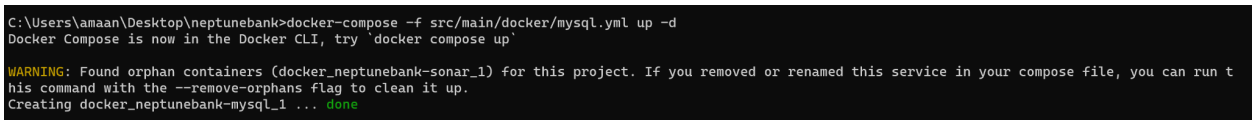
Docker can also be used to create versions of the sonarqube and mysql needed for the production environment. The sonarqube container can be created using the dockerfile in our application. This is shown below using the command docker-compose -f src/main/docker/sonar.yml up -d.

```
C:\Users\amaan\Desktop\neptunebank>docker-compose -f src/main/docker/sonar.yml up -d
Docker Compose is now in the Docker CLI, try `docker compose up`

Pulling neptunebank-sonar (sonarqube:7.9.1-community)...
7.9.1-community: Pulling from library/sonarqube
000eee12ec04: Pull complete
2f1dc2bdcfe1: Pull complete
eec880363624: Pull complete
8992c959a11d: Pull complete
25ff10872c55: Pull complete
9a775036a9e6: Pull complete
10640c614c1e: Pull complete
0a5459c4b518: Pull complete
b68f6850bf39: Pull complete
Digest: sha256:64d3a0e6fc899542d9171b0a94135ea7c4b51fbb25842ca782baf0de00f66535
Status: Downloaded newer image for sonarqube:7.9.1-community
Creating docker_neptunebank-sonar_1 ... done
```

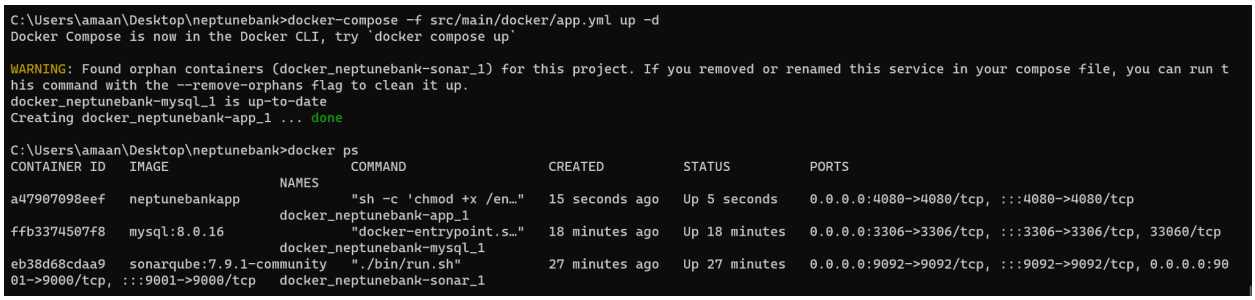
It can then be launched on the local host as shown below.



The mysql container can be created using the dockerfile in our application. This is shown below using the command docker-compose -f src/main/docker/mysql.yml up -d.



To view all these containers the docker ps command can be used and as shown below all three containers are running as expected.



Analysis of Docker

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. Docker provides the ability to package and run an application in a loosely isolated environment called a container. Containers are lightweight and contain everything needed to run the application, regardless of what is installed on the host.

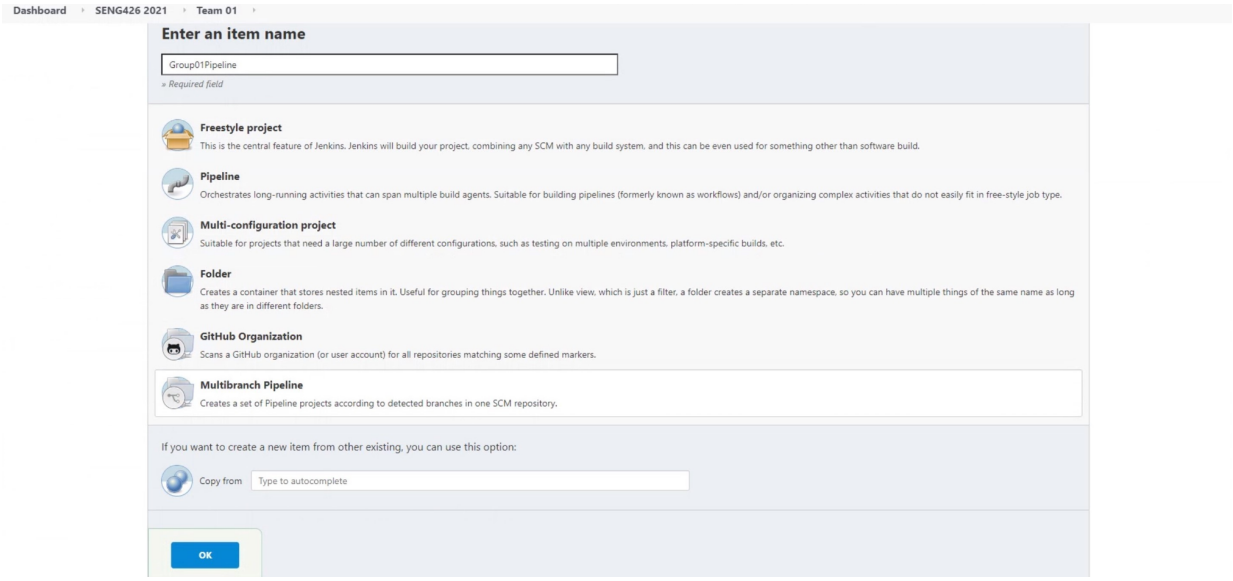
Docker *images* are read-only templates with an instruction set for creating a docker *container*. Docker images can be customized for a user's specific needs by using a *Dockerfile*, with custom instruction sets. A container is a runnable instance of an image. Containers can be connected to networks or storage devices. A container is defined by its image and configuration options provided when creating it.

There are two tools often used when managing Docker. The usual Docker tool and Dockerfile are used by a user to describe and assemble an image to their configuration. It simply takes the Dockerfile and returns a corresponding Docker image. On the other hand, the Docker compose tool can be used to define and run multi-container Docker applications. A user can define the services that make up the structure of their application in a docker-compose.yml file so that they can be run together in an isolated environment. This allows you to get an app running with a single command. Docker compose essentially works as a script to manage and issue a large amount of Docker commands and configuration.

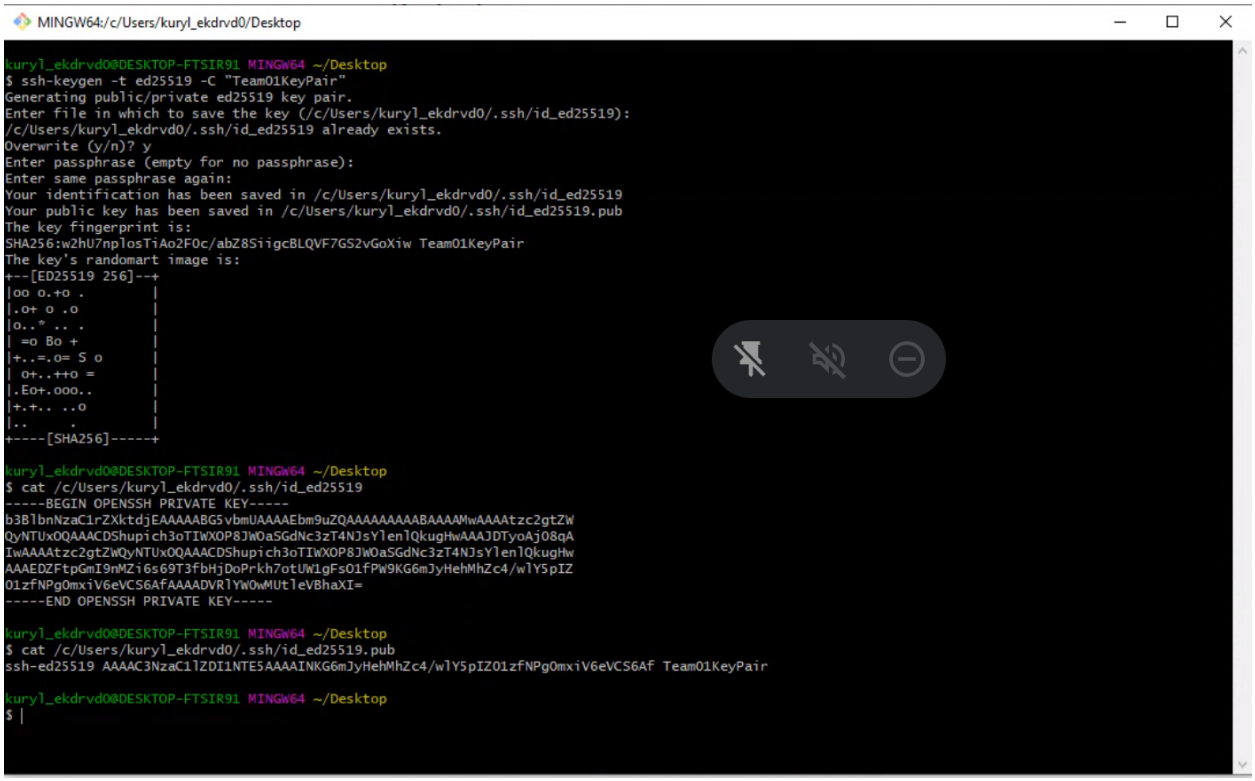
The Neptune Banking application image is created using a variety of steps and tools in the process. Within the repository, there are a few docker-compose configurations available to launch the services needed for the application to function correctly. Firstly, the correct version of Maven is fetched as specified in the maven-wrapper.properties file. Next, Google's *Jib* is added to our pom.xml and used to containerize the application. Jib allows one to build containers without using a Dockerfile or Docker installation. In our case, Jib uses an option (jib:dockerBuild) that allows us to build to our local Docker installation in order to get our image. Jhipster is then run in its production spring profile which focuses on performance and scalability by providing Maven with the *-Pprod package* option, packaging it as an executable JAR file. Finally, our production image is created, and can be launched using docker-compose as follows: *docker-compose -f src/main/docker/app.yml up -d*.

Jenkins and GitLab Setup

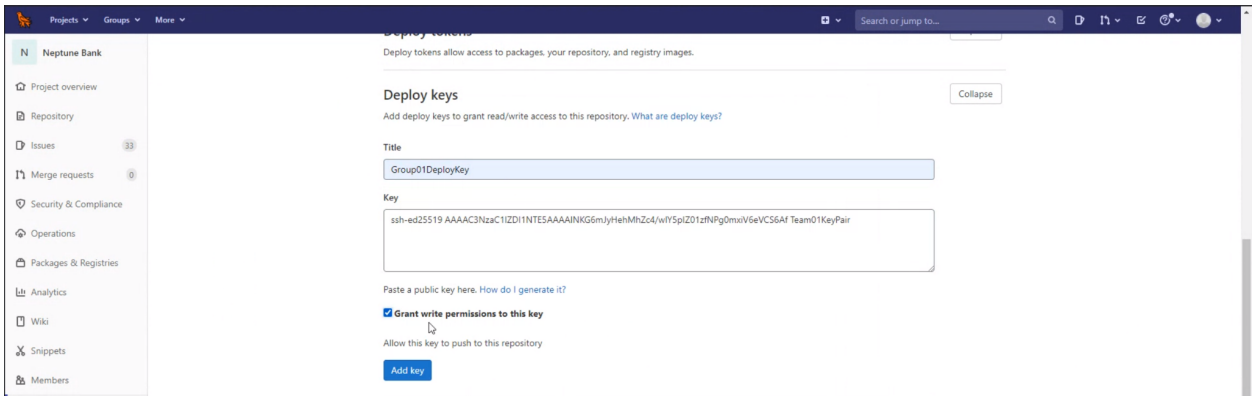
First a new project is started. We select a multibranch pipeline and assign it a name. We use a multibranch pipeline so when we develop we can each use branches and each branch will have it's own dedicated pipeline. These pipelines are created automatically by Jenkins when a branch is created.



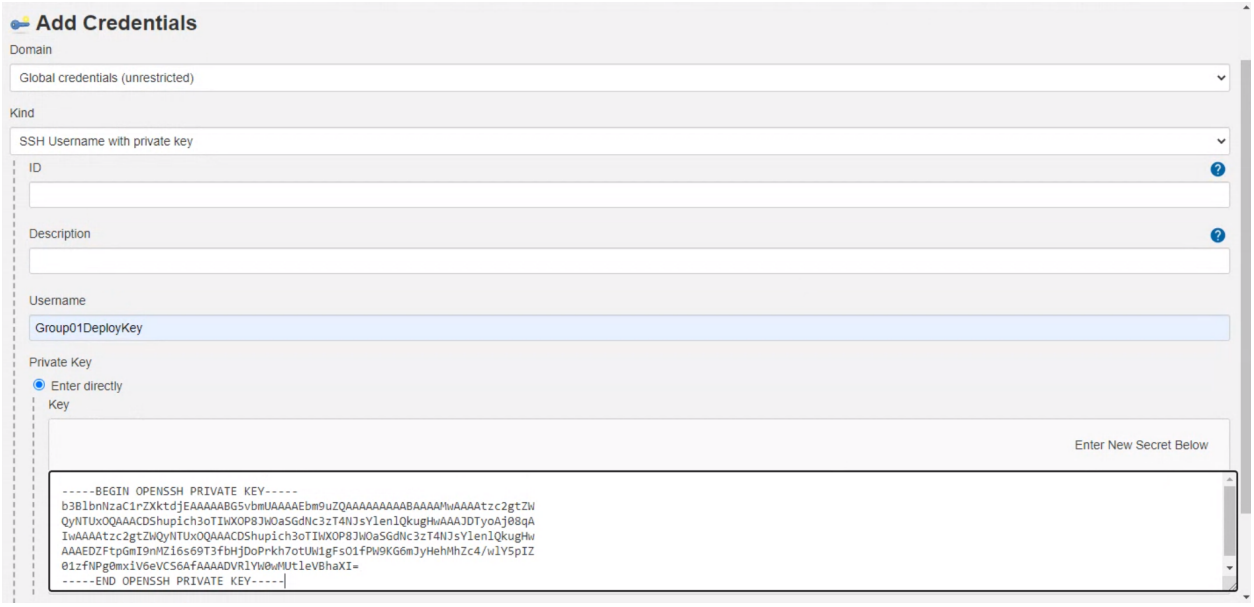
To allow for the communication of Jenkins and Gitlab we need to generate a ssh keygen pair using the first command shown in the image below. To grab the keypair we use the ~/.ssh, it is important to have both keys, and the cat command is used to print out the two keys into the terminal.



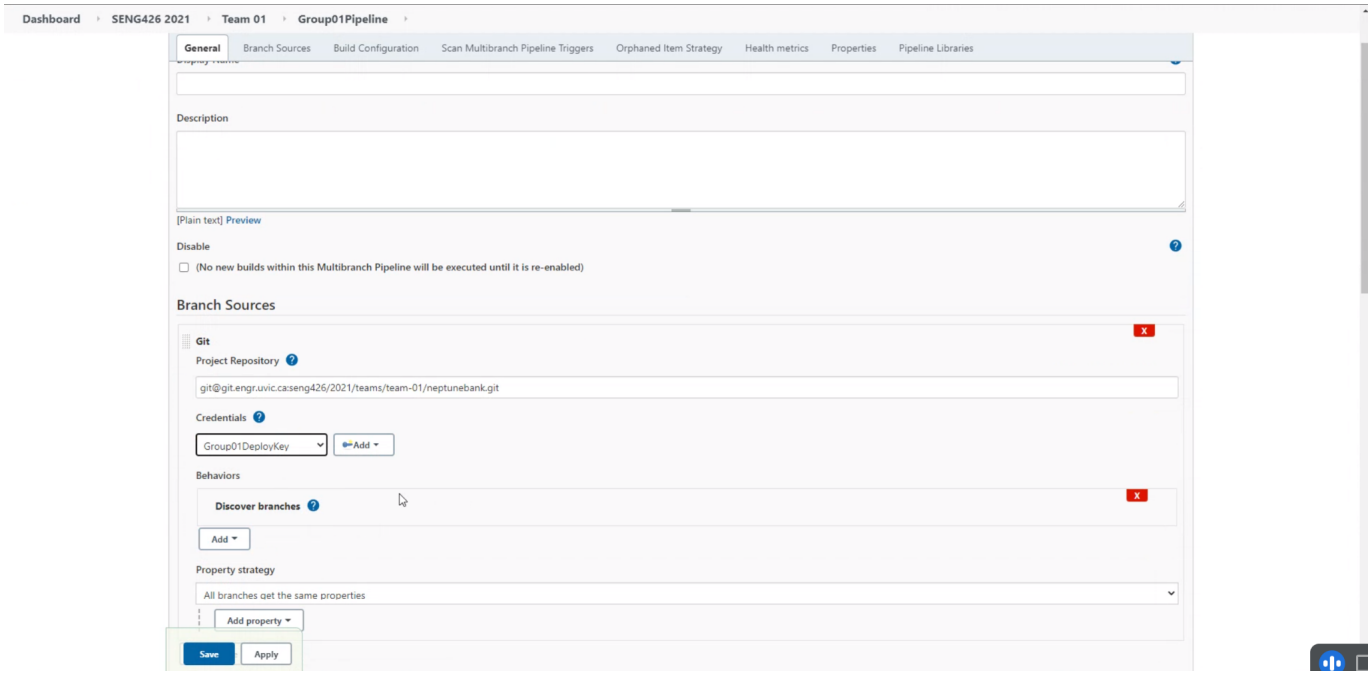
To use the keys we copy the public key (first printed out key in the terminal image) and add it as an deploy ssh key in gitlab (see image below). This page can be navigated to by viewing the repository settings and going to security. This key is given a name to uniquely identify it from others the account may have.



We then go to Jenkins and add the private key as shown below. The credentials can be added through pipeline settings and within the general page. This key is given a name to uniquely identify it from others the account may have.

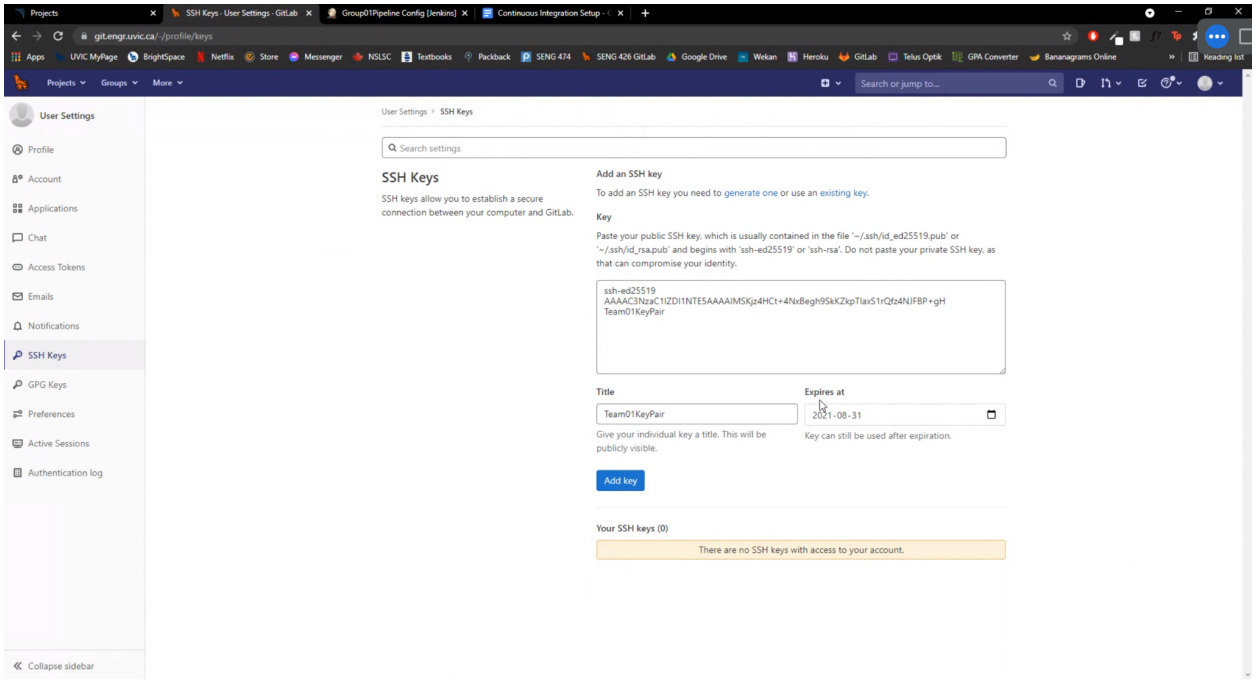


The account that the credentials are used for are also given after the credentials are added. This is done by giving the repo URL as shown below. Then the changes are saved.

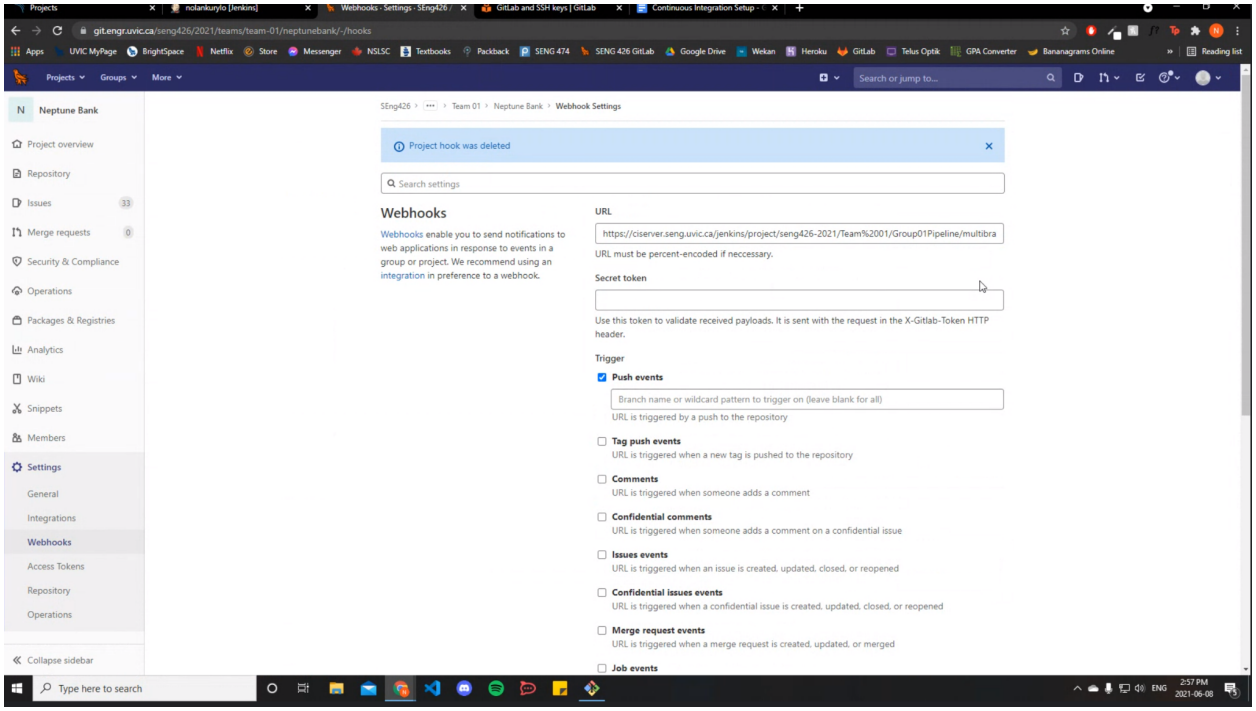


The public key is also added as an SSH key in gitlab through the repository settings. This is shown below. This key is given a name to uniquely identify it from others the

account may have. This key will be used for Jenkins to pull your code when the pipeline is triggered.



To trigger your pipeline a webhook must first be added. To do this navigate to repository settings then webhooks. The URL for the Jenkins pipeline must be copied and pasted here. Then you select all the events you want a notification for. In our case this is push events as we want the pipeline to verify our changes. Now when you commit a change an event is sent to this URL triggering the pipeline.



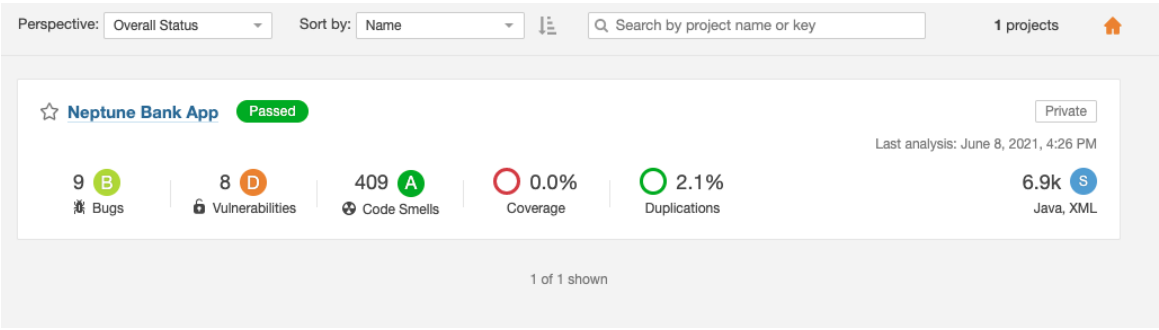
Changes Made to Support CI

To support the continuous integration process, several changes needed to be made to the application's files as well as the Jenkinsfile. No modifications were necessary to the dockerfiles to complete the docker demonstration. Firstly, to support the application for production, the configuration file *config/application-prod.yml* was re-configured with updated properties for datasource's url, username, password, and server port to ensure the application can successfully deploy from the Jenkins server.

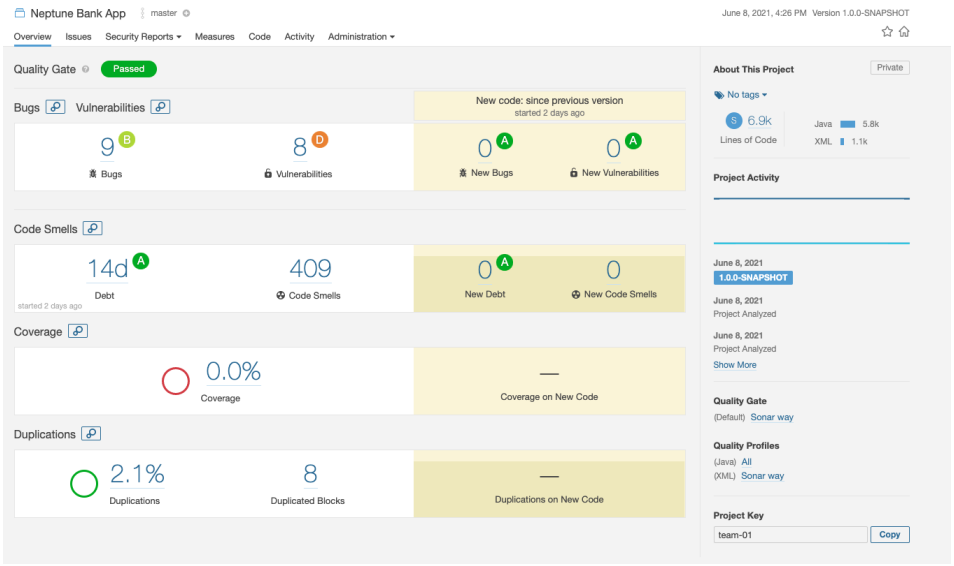
Next, to support our continuous integration process, a *Jenkinsfile* was created and placed in the root of the application repository. We utilized a declarative pipeline syntax with three main stages. To start, the Jenkinsfile defines a *Build* stage to clean and package the application for production, with an extra flag to ensure that tests are not run during this stage. Next, there is a *Sonar* stage to run the sonar analysis of our source code. Finally, a *Deploy and Test* stage concurrently deploys the application and starts the selenium integration test suite. With a declarative post-action, we added a functionality to notify a team member via email if a build fails.

SonarQube

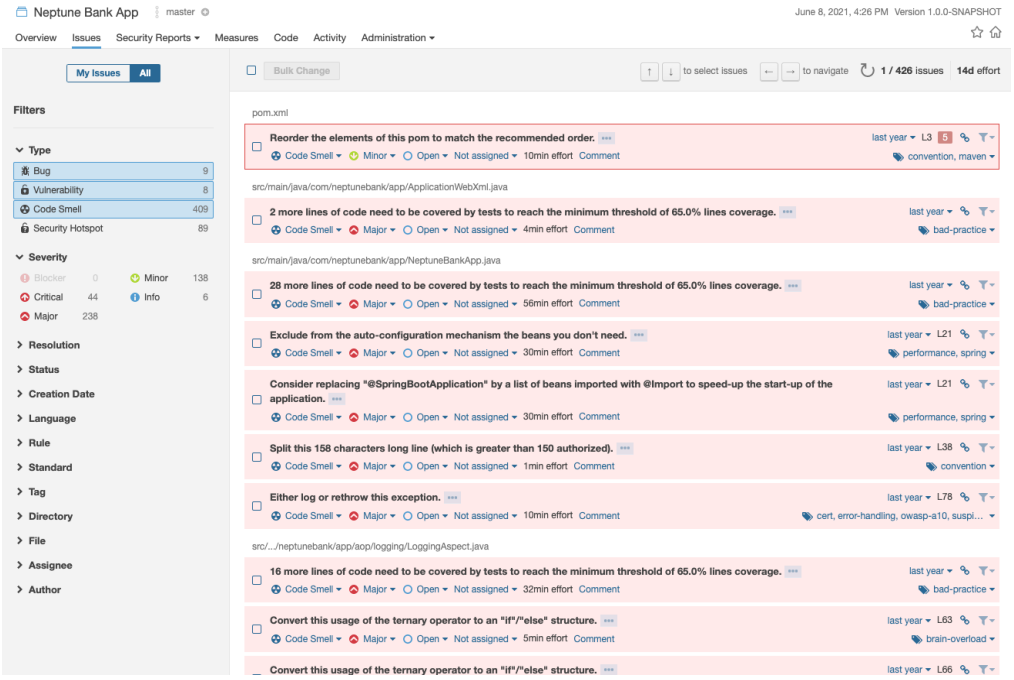
During one stage of our build pipeline, SonarQube is used to analyze the application's source code. Below are screenshots of the analysis after the code has been scanned during a build.



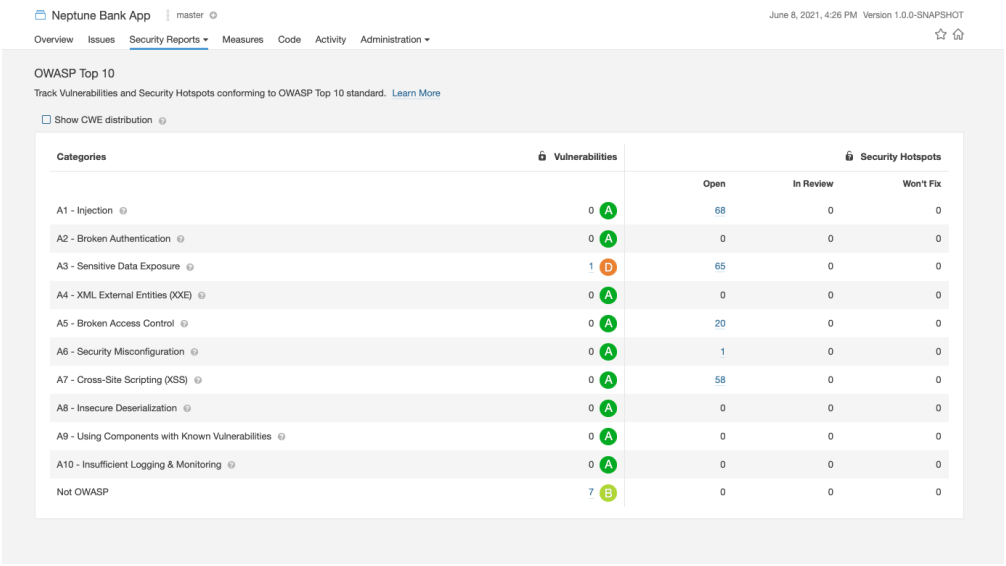
Firstly, we can see a general overview of the project with some high level information such as bugs, vulnerability, code smells, coverage and duplications.

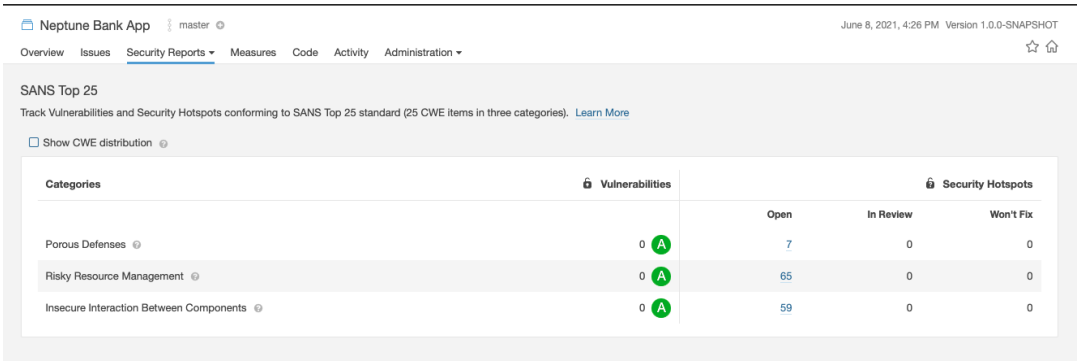


In the project page, a more detailed overview can be seen, with a helpful “new code” section to compare new builds to previous builds.



Next, the issues page gives an overview of all application issues with effort estimates and priority indicators, along with other metrics.

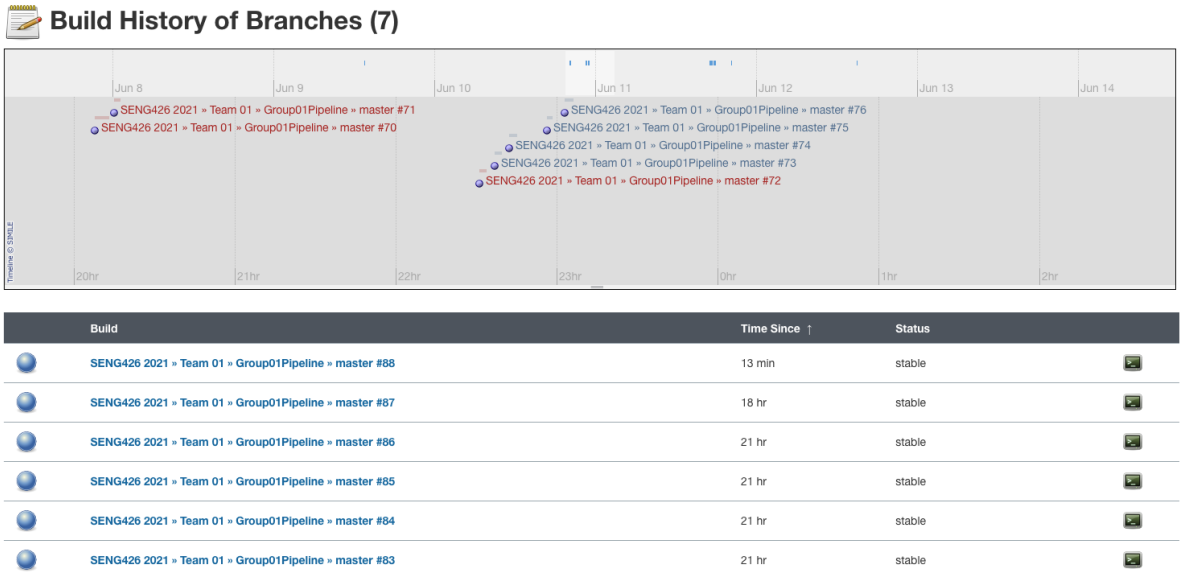




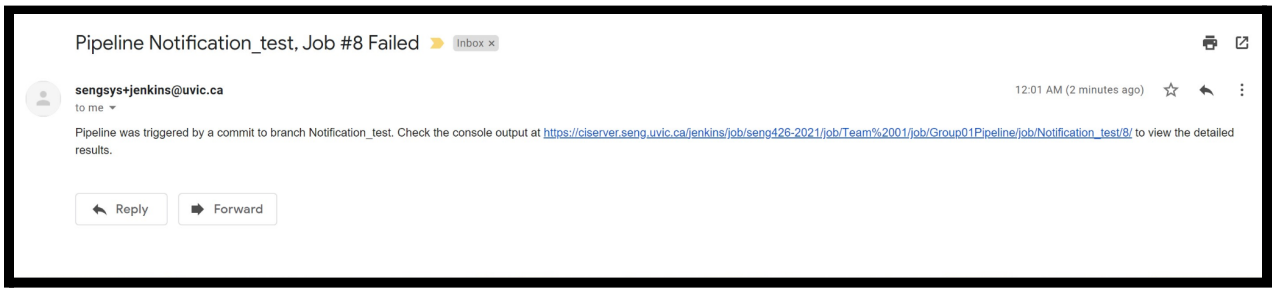
Finally, we can see two security reports. Both OWASP Top 10 and SANS Top 25 reports are available from the navigation menu.

Jenkins History and Reports

The build history of the branches of our repository can be seen below:



Notifications



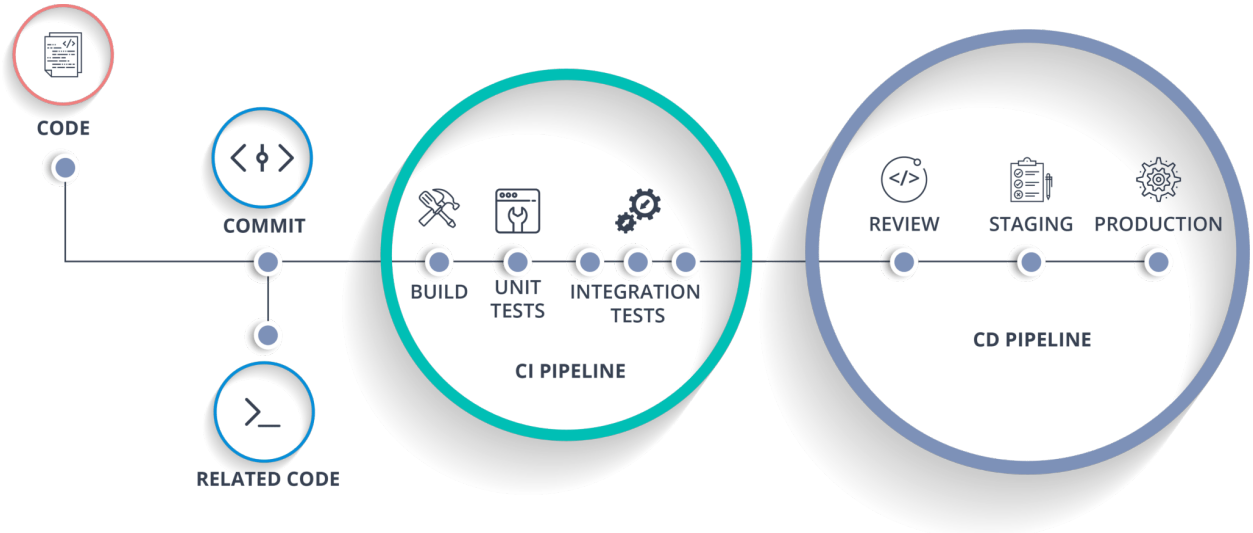
The notifications were sent using the extended email extension plugin enabled in Jenkins. The email notification is sent to any developer who caused a change in the change set. The email title indicates the pipeline run(which has the same name as the branch) and the job number that failed. In this notification body the branch that triggered the pipeline is documented and the link to the Jenkins job is included.

We did wish to include other information like the commit ID and the commit author but various details were not set by Jenkins and Gitlab. These variables are detailed below:

- CHANGE_ID: For a multibranch project corresponding to some kind of change request, this will be set to the change ID, such as a pull request number, if supported; else unset.
- CHANGE_URL: For a multibranch project corresponding to some kind of change request, this will be set to the change URL, if supported; else unset.
- CHANGE_TITLE: For a multibranch project corresponding to some kind of change request, this will be set to the title of the change, if supported; else unset.
- CHANGE_AUTHOR: For a multibranch project corresponding to some kind of change request, this will be set to the username of the author of the proposed change, if supported; else unset.
- CHANGE_AUTHOR_DISPLAY_NAME: For a multibranch project corresponding to some kind of change request, this will be set to the human name of the author, if supported; else unset.

CI in Software Development

Continuous integration (CI) or continuous development (CD) is a core concept to modern software development. The goal of CI/CD is to streamline development and to automate a pipeline that takes a load off developers so they are free to do other tasks. \



N. Balajee, "What is CI/CD Pipeline?," *Medium*, 07-Jan-2020. [Online]. Available: https://nanduribalajee.medium.com/what-is-ci-cd-pipeline-e2f25db99bbe. [Accessed: 08-Jun-2021].

See the above image for a visualized start to finish look at what this pipeline might entail. The picture follows closely to what we are doing with this step of the project. By pushing or merging code a series of systems are in place to automatically build, test, and eventually deploy the commit. By having these systems in place there is a consistency

across all developers in how code is made into development. This way we avoid any discrepancies between environments or local setups that the developers have, this is also a good way to emulate how code will change the production system. The pipeline is also set up so that it stops once any errors or tests fail, there is a notification system (not shown in the image) that will then report this back to the developer. As mentioned previously a big benefit of having these systems in place is to save the developer time in having to do all these tasks manually, as well as making a consistent process that alleviates any human error from interfering with the deployment process.

As companies and development teams mature it becomes more obvious how necessary this development operation is to producing quality code, in some instances there are whole roles (devop engineers) whose job is dedicated to streamlining and creating this pipeline. This pipeline is also a good way for QA to better be established throughout development of the code, as the tests are built right in and the pipeline will not continue unless everything passes successfully.

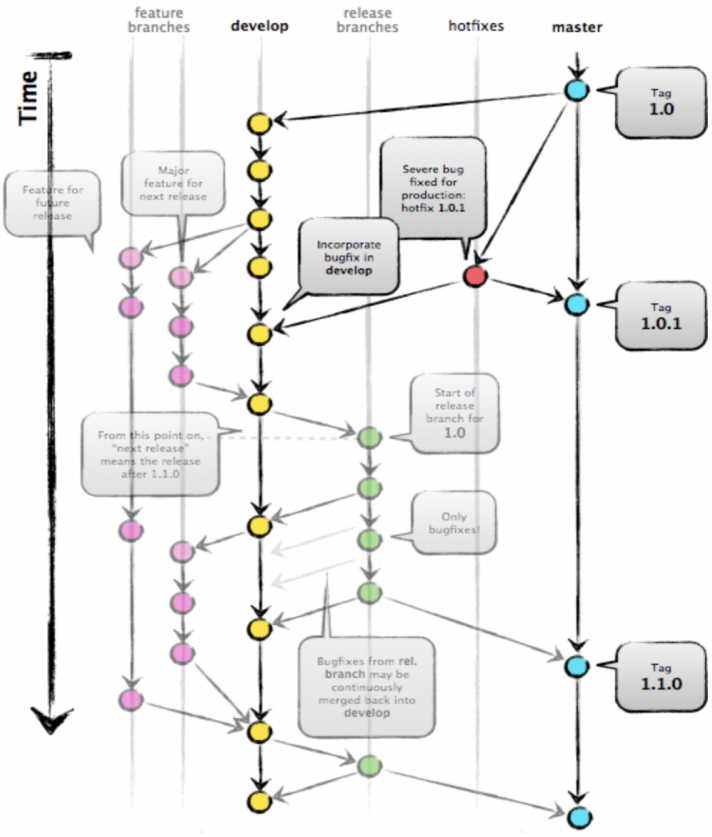
CI Improvements

In the current pipeline that was designed according to project specifications testing is done after the project has successfully deployed. In our opinion this is backwards, ideally we would like to test the new changes and have all the tests pass successfully before deploying. This way if there are any errors or bugs caught by the tests we do not need to roll back changes or quickly find a fix to the buggy deployed code. *Figure Y* below shows the results of the test script and the errors being faced after deployment, when in reality the app should never have been deployed while the test script is failing. Furthermore, testing scripts should be written to test different browsers in parallel to verify proper functionality among all browsers.

Another alternative to help streamline the pipeline process is to use Gitlabs built in CI/CD functionality. Since Gitlab is already used for source control, using it for CI/CD would remove some moving parts and potentially remove sources of error. This would also be beneficial, as members of our team are well versed in working with Gitlabs CI/CD functionality. Constant issues accessing the Jenkins server and the sonarqube servers often prevented our group from working on this project. One major improvement that has to be made is the availability of these servers. The servers would not be able to be reached by some members even if correctly on the VPN. Troubleshooting tasks like restarting the VPN could work but were unreliable and often very unsuccessful.

Git Flow Model

The GitFlow model is the branching model for Git, and has massively grown in popularity since its inception in 2005, due to its scalability and collaborative nature. The basic project flow is simple; Developers are able to branch off from the master branch, creating feature branches. These feature branches are completely isolated from the master branch, which is the main body of code, and a pull request is created and must be approved by other developers prior to being merged back into the main body of code. This poses many benefits - developers are able to work in parallel, focusing on different features at the same time, and without interrupting or preventing other developers from creating features or fixing bugs. Furthermore, multiple developers are able to work on the same feature branch simultaneously, each making the changes required to get the feature running. Support for Emergency Fixes also exists - hotfix branches can be created if an emergency bug needs to be fixed, without posing any risk that any new unfinished features may be merged into the master branch.

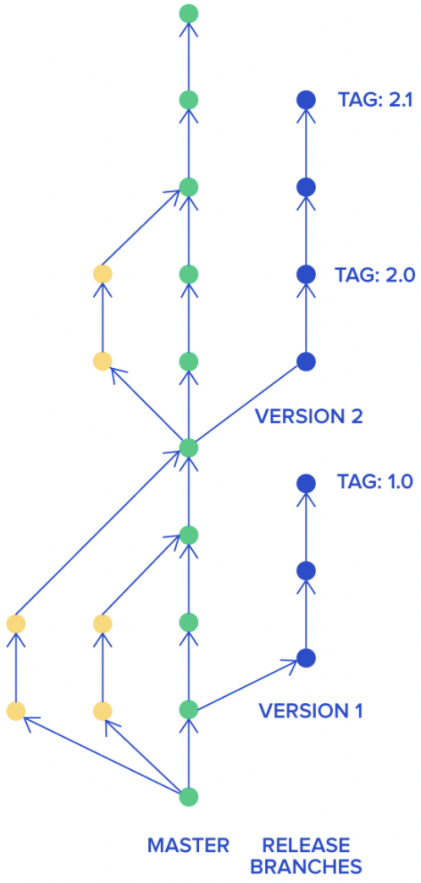


GitFlow Development Model

Ultimately, the GitFlow model allows multiple developers to create branches off of a working body of code to fix bugs and create new features, and to merge back to the

master once the new branch is complete. It is very useful for collaboration, does not limit developers to focus on one issue at a time, is extremely fast and scalable, and is fully distributed, which is why it has become immensely popular amongst developers.

Subversion, or SVN, is a version control system that embodies the Trunk-based Development Workflow. Not completely dissimilar to the GitFlow, all developers have access to the master branch, and build on that master branch over time. However, rather than developers creating a large number of feature branches, creating pull requests, and merging back into master as seen in GitFlow, developers commit code directly to the master branch and run it - some cases feature branches will be created, but these branches ideally are merged to master within a few days. Trunk-based development truly embodies continuous integration, as the omission of lengthy feature branches prevents difficulties caused by resolving merge conflicts to arise.



Trunk-based Development Model

Both GitFlow and Trunk-based Development are popular version control systems. Both are used for continuous integration of projects for developers, for both teams and solo

projects alike. GitFlow is widely regarded to be faster, less resource heavy, and allows for better auditing of pull requests when features are completed. This is largely due to the fact that it is a distributed control system, which also allows each developer to have full local control. Trunk-based Development is a centralized version control system that prevents complexities from resolving merge conflicts and maintains fewer development lines, as all developers generally commit directly to master. This allows for continuous code integrations and continuous code review amongst developers.

References

https://git.wiki.kernel.org/index.php/GitSvnComparsion
https://toptal.com/software/trunk-based-development-git-flow
https://cloud.google.com/architecture/devops/devops-tech-trunk-based-development


```
// Part 1
------------------------------------------------------------------------
------------------------------------------------------------------


    public void register(String username, String email, String password,
String repeatPassword) {

driver.findElement(By.xpath("/html/body/div/div/div[2]/div[2]/nav/div/ul/l
i[3]/a/span")).click();

driver.findElement(By.xpath("/html/body/div/div/div[2]/div[3]/div[1]/div/d
iv/div[2]/div/form/div[1]/input")).sendKeys(username);

driver.findElement(By.xpath("/html/body/div/div/div[2]/div[3]/div[1]/div/d
iv/div[2]/div/form/div[2]/input")).sendKeys(email);

driver.findElement(By.xpath("/html/body/div/div/div[2]/div[3]/div[1]/div/d
iv/div[2]/div/form/div[3]/input")).sendKeys(password);

driver.findElement(By.xpath("/html/body/div/div/div[2]/div[3]/div[1]/div/d
iv/div[2]/div/form/div[5]/input")).sendKeys(repeatPassword);
```

```
driver.findElement(By.xpath("/html/body/div/div/div[2]/div[3]/div[1]/div/div/div[2]/div/form/button")).click();
    }
    public void addPayee(String username, String firstname, String lastname, String telephone) {

driver.findElement(By.xpath("/html/body/div/div/div[2]/div[2]/nav/div/ul/li[5]/a/span")).click();

driver.findElement(By.xpath("/html/body/div/div/div[2]/div[3]/div[1]/div/div/div/h2/a")).click();

driver.findElement(By.xpath("/html/body/div/div/div[2]/div[3]/div[1]/div/div/div/div[2]/div/form/div[1]/div/input")).sendKeys(username);

driver.findElement(By.xpath("/html/body/div/div/div[2]/div[3]/div[1]/div/div/div/div[2]/div/form/div[2]/div/input")).sendKeys(firstname);

driver.findElement(By.xpath("/html/body/div/div/div[2]/div[3]/div[1]/div/div/div/div[2]/div/form/div[3]/div/input")).sendKeys(lastname);

driver.findElement(By.xpath("/html/body/div/div/div[2]/div[3]/div[1]/div/div/div/div[2]/div/form/div[4]/div/input")).sendKeys(telephone);

driver.findElement(By.xpath("/html/body/div/div/div[2]/div[3]/div[1]/div/div/div/div[2]/div/form/button[2]")).click();
    }

    @Test
    public void createsUserAccountWithCorrectCredentials() {
        String username = "test" + RandomStringUtils.randomAlphanumeric(5);
        String email = "test" + RandomStringUtils.randomAlphanumeric(8) + "@gmail.com";
        String password = "123456";
        String repeatPassword = "123456";
        assertFalse(isElementPresent(By.id("account-menu"))); // starts logged out
        register(username, email, password, repeatPassword);
```

```
        assertTrue(isElementPresent(By.xpath("//*[contains(text(),'Your
account has successfully been created.')]")));


    }

    @Test
    public void doesNotCreateUserAccountGivenMismatchingPasswords() {
        String username = "test" +
RandomStringUtils.randomAlphanumeric(5);
        String email = "test" + RandomStringUtils.randomAlphanumeric(8) +
"@gmail.com";
        String password = "12345";
        String repeatPassword = "123456";
        assertFalse(isElementPresent(By.id("account-menu"))); // starts
logged out
        register(username, email, password, repeatPassword);
        assertFalse(isElementPresent(By.xpath("//*[contains(text(),'Your
account has successfully been created.')]")));

    }



    // Part 2
--------------------------------------------------------------------------
-----------------------------------------------------------------



    @Test
    public void correctEmailFormatForAddingPayee() { // bug fix #5

        assertFalse(isElementPresent(By.id("account-menu"))); // starts
logged out
        login(validUsername, validPassword);
        assertTrue(isElementPresent(By.xpath("//*[contains(text(),'Manage
Payees')]")));
        String email = "test@gmail.com";
        String firstname = "hello";
```

```
        String lastname = "world";
        String telephone = "123456798";
        addPayee(email, firstname, lastname, telephone);
        assertFalse(isElementPresent(By.xpath("//*[contains(text(),'This
field is invalid')]")));

    }

    @Test
    public void incorrectEmailFormatForAddingPayee() { // bug fix #5

        assertFalse(isElementPresent(By.id("account-menu"))); // starts
logged out
        login(validUsername, validPassword);
        assertTrue(isElementPresent(By.xpath("//*[contains(text(),'Manage
Payees')]")));
        String email = "testATgmail.com";
        String firstname = "hello";
        String lastname = "world";
        String telephone = "123456798";
        addPayee(email, firstname, lastname, telephone);
        assertTrue(isElementPresent(By.xpath("//*[contains(text(),'This
field is invalid')]")));

    }
```