UECS2363 SOFTWARE CONSTRUCTION AND CONFIGURATION

Practical 10 : Jenkins and Travis CI (Part II)

Jenkins - Plugins.

Jenkins uses plugins in order to execute continuous integration and feedback tasks. There are different categories of Jenkins plugins to enable the tasks such as;

1. Test: JUnit plugin
2. Reports: HTML Publisher plugin
3. Notification: Jenkins Build Notifications Plugin
4. Deployment: Deploy plugin
5. Compile: Maven, Gradle

For this practical, let's explore "Github Integration" plugin. Install the plugin through "Manage Plugins" configuration. By typing the plugin name into search tect input in "Available" menu, a series of search result inclusive of "Github Integration". Select "Github Integration" and click "Install without restart" as illustrated in Figure 1.

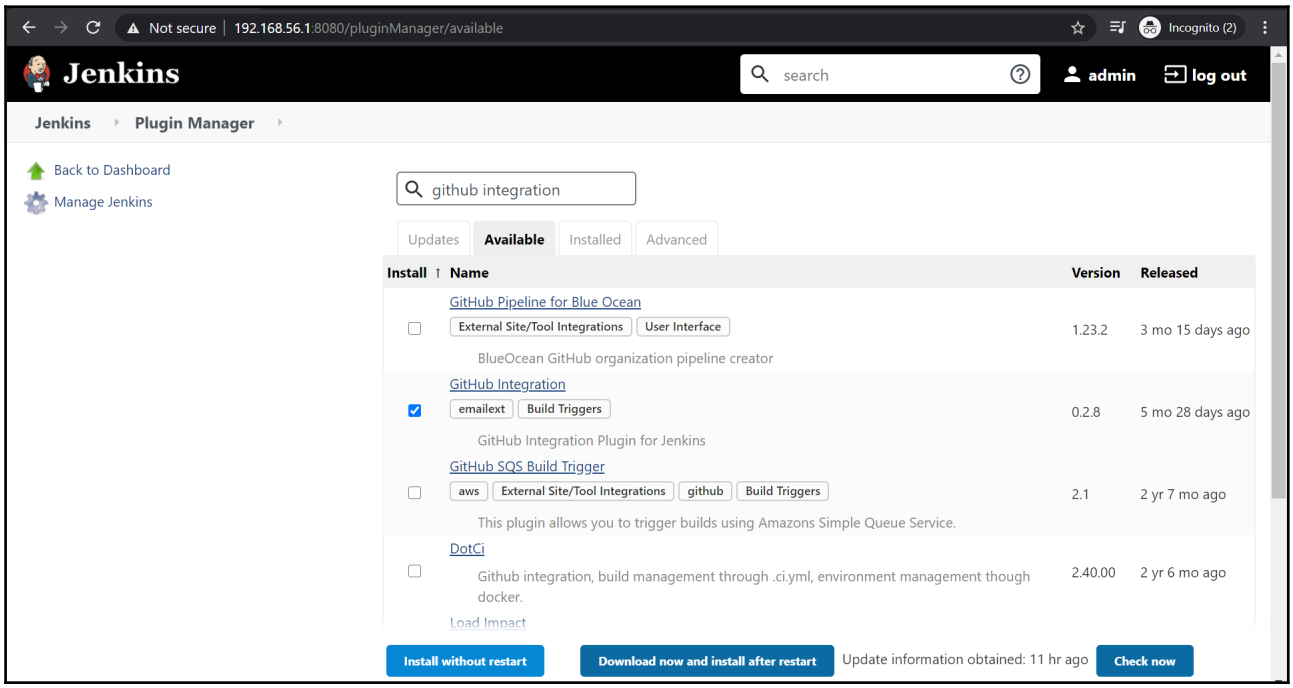


Figure 1: Install Github Integration plugin in Jenkins.

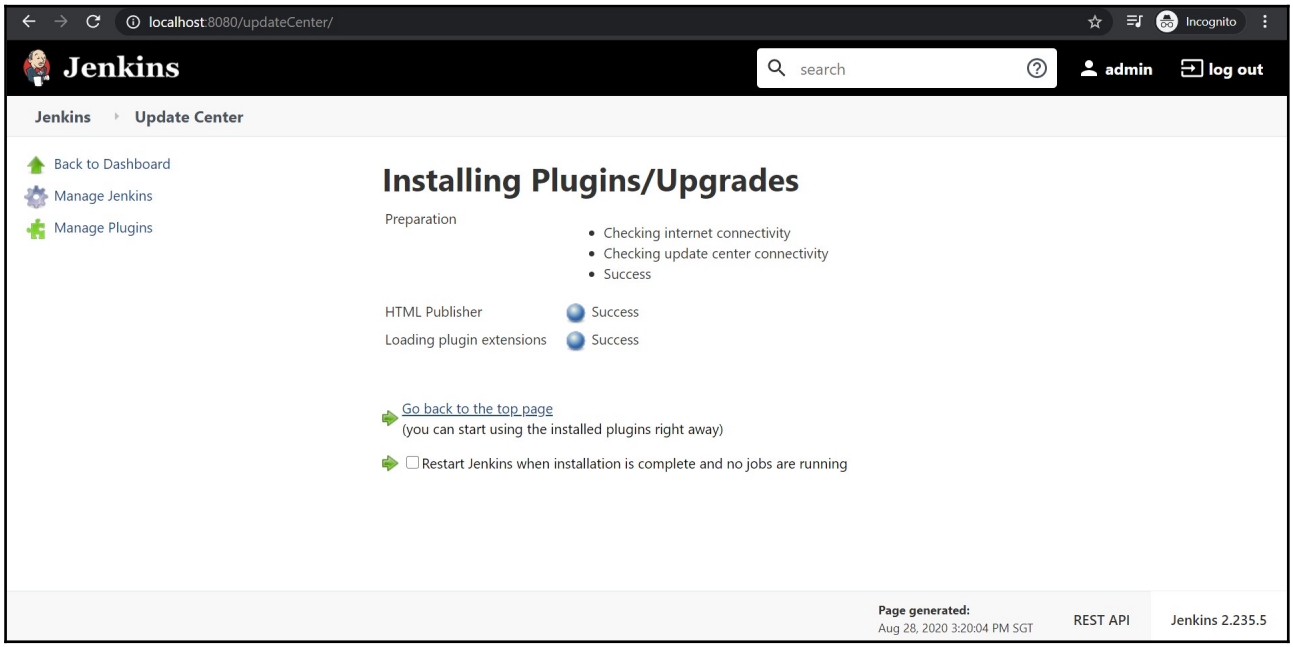


Figure 2: Successful installation of HTML Puiblisher plugin.

Upon successful installation as shown in Figure 2, click on “Back to Dashboard” button to go back to Jenkins dashboard.

Jenkins Integration with Github.

Configurations in Jenkins.

Jenkins need to connect to a shared source code repository server in order to perform CI/CD to the project files for each commit made to the project. In this practical, integration of Jenkins with Github is explored. In order to integrate Jenkins with Github, a “Webhook” need to be configured in your Github account.

Jenkins URL need to be setup in order for internet and Jenkins to find the correct server on internet for any pull and push activities to Github. One may configure Jenkins URL (by default localhost:8080), where the default URL is not recognized on internet, to a URL containing IP address of the host or domain name of the host.

For this practical learning purpose, you may just choose to include your host’s IP to be your Jenkins URL. Note that if you are unsure of the IP of your localhost, execute “ipconfig” command in Windows CLI; the first Ipv4 address indicates your localhost IP. Configure the IP address into Jenkins URL by first selecting “Manage Jenkins”, then select “Configure System” as shown in Figure 3.

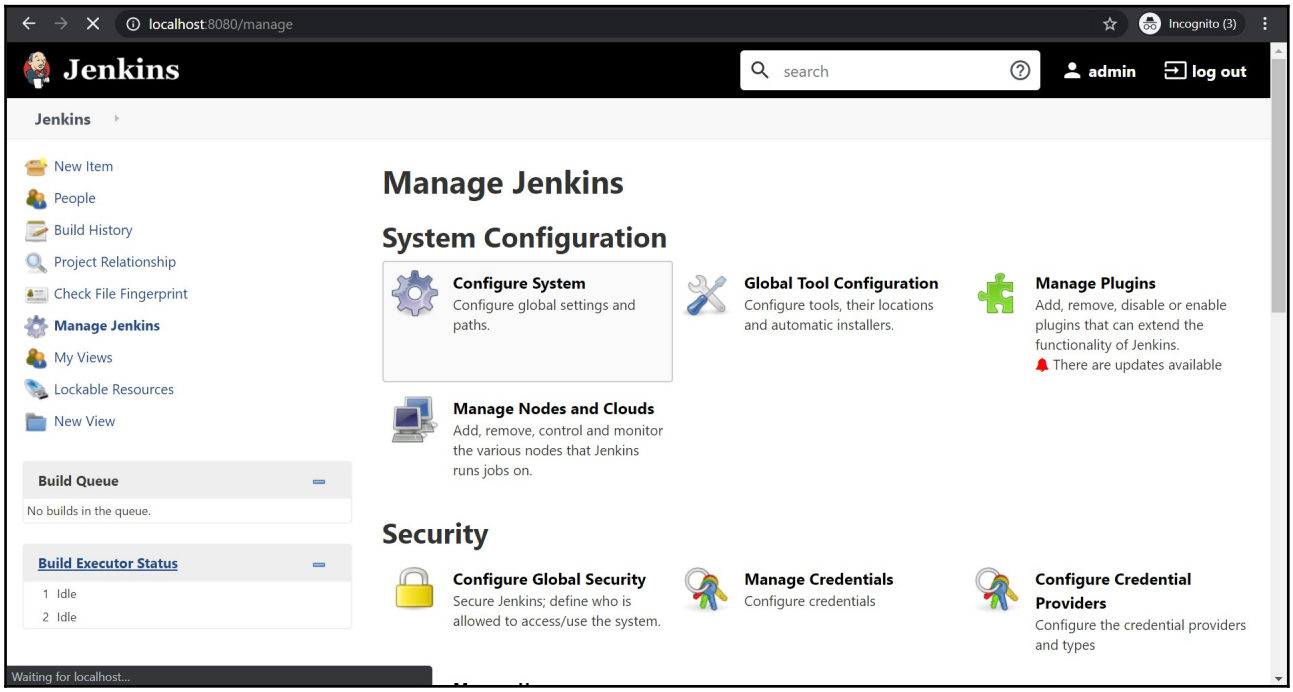


Figure 3: Manage Jenkins proxy server with Configure System menu.

Then, scroll to Jenkins Location and change the default Jenkins URL from “localhost:8080” to your host’s IP adress with port 8080 as shown in Figure 4.

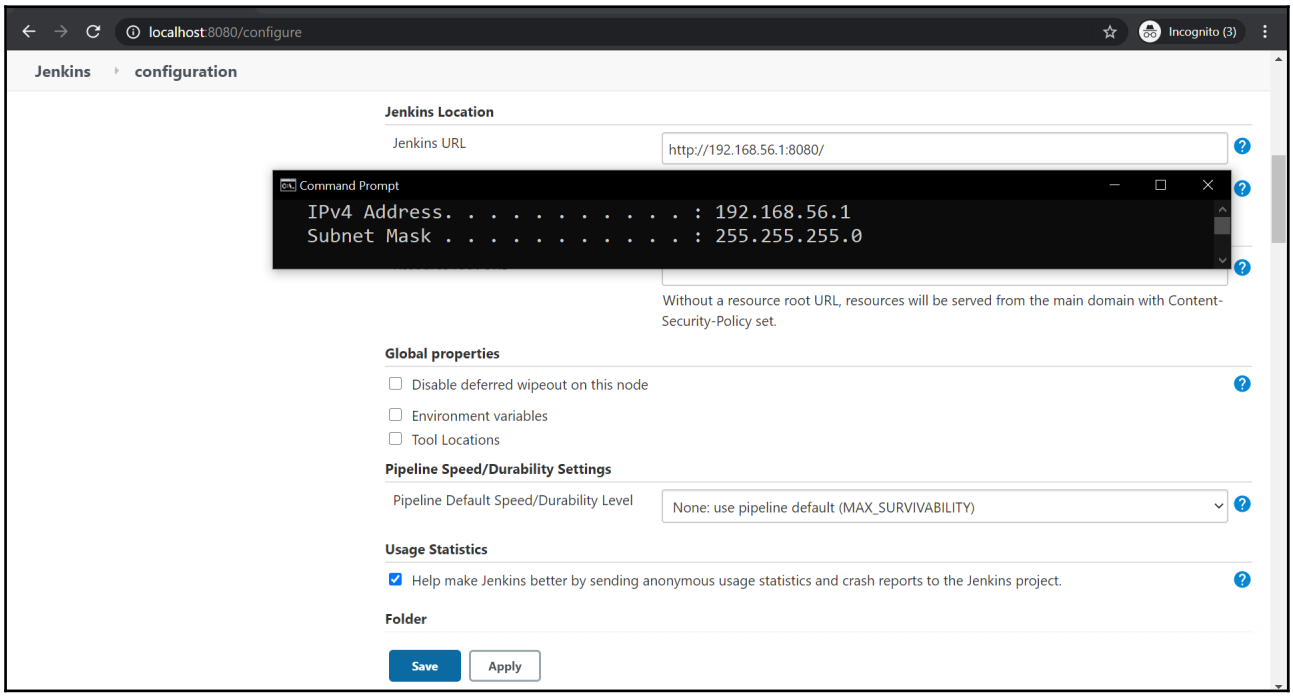


Figure 4: Change default Jenkins URL into recognizable URL address.

Configurations in Github.

For learning purpose, fork **https://github.com/saurabh0010/houselannister** Github repo into your Github account. Then go to the forked Github repo and click on "Settings" for the repository as shown in Figure 5.

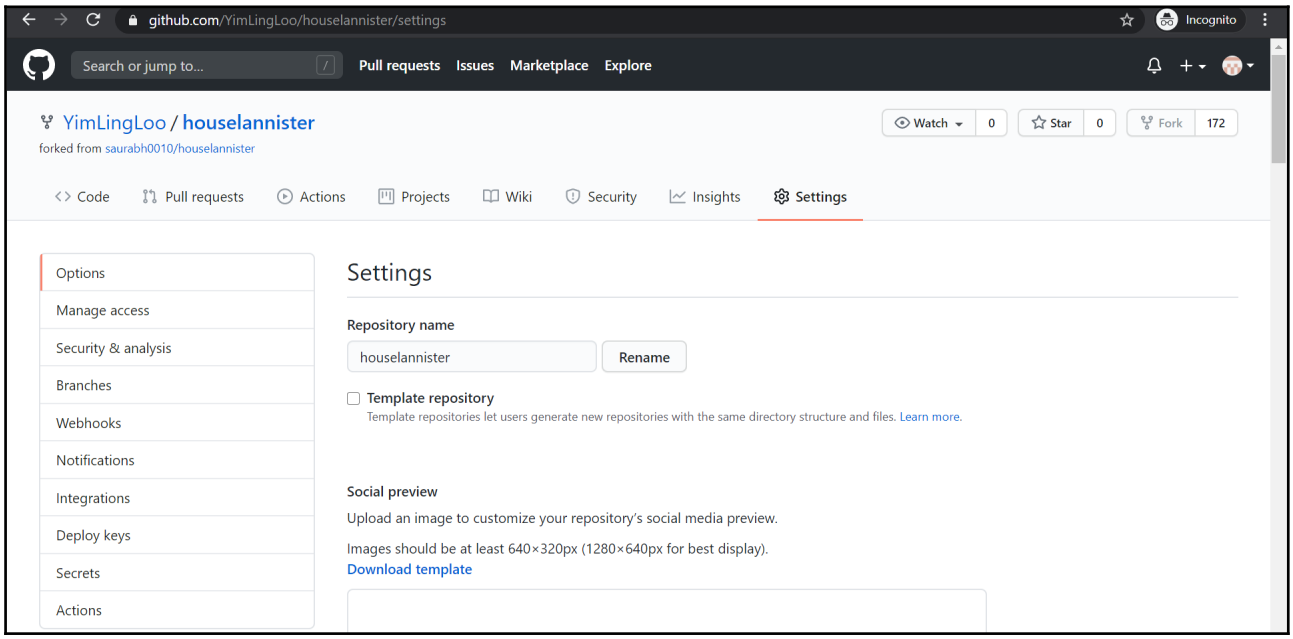


Figure 5: View "Settings" of the forked repo.

On Settings page, click on "Webhooks" on the left navigation pane then click "Add Webhook" as shown in Figure 6.

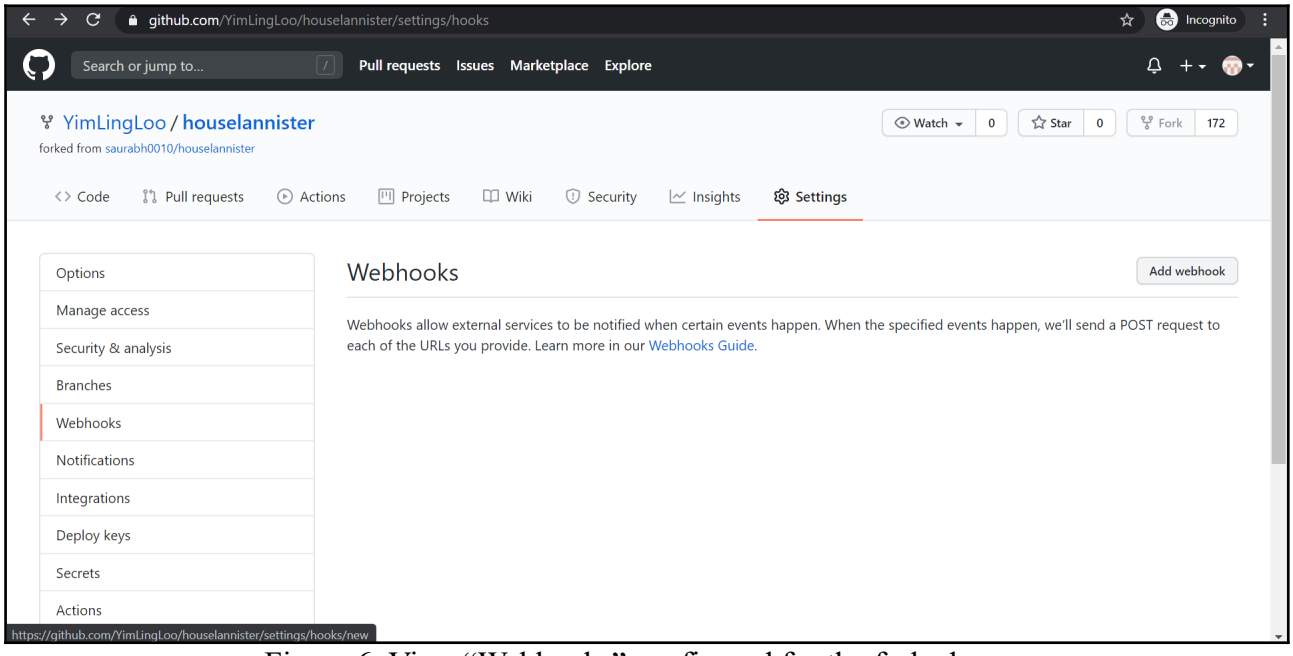


Figure 6: View "Webhooks" configured for the forked repo.

Upon selecting "Add Webhook", you'll be redirected to the page as shown in Figure 7. In the 'Payload URL' field, paste your Jenkins environment URL. At the end of this URL add "/github-webhook/". In the 'Content type' select 'application/json' and leave the 'Secret' field empty.

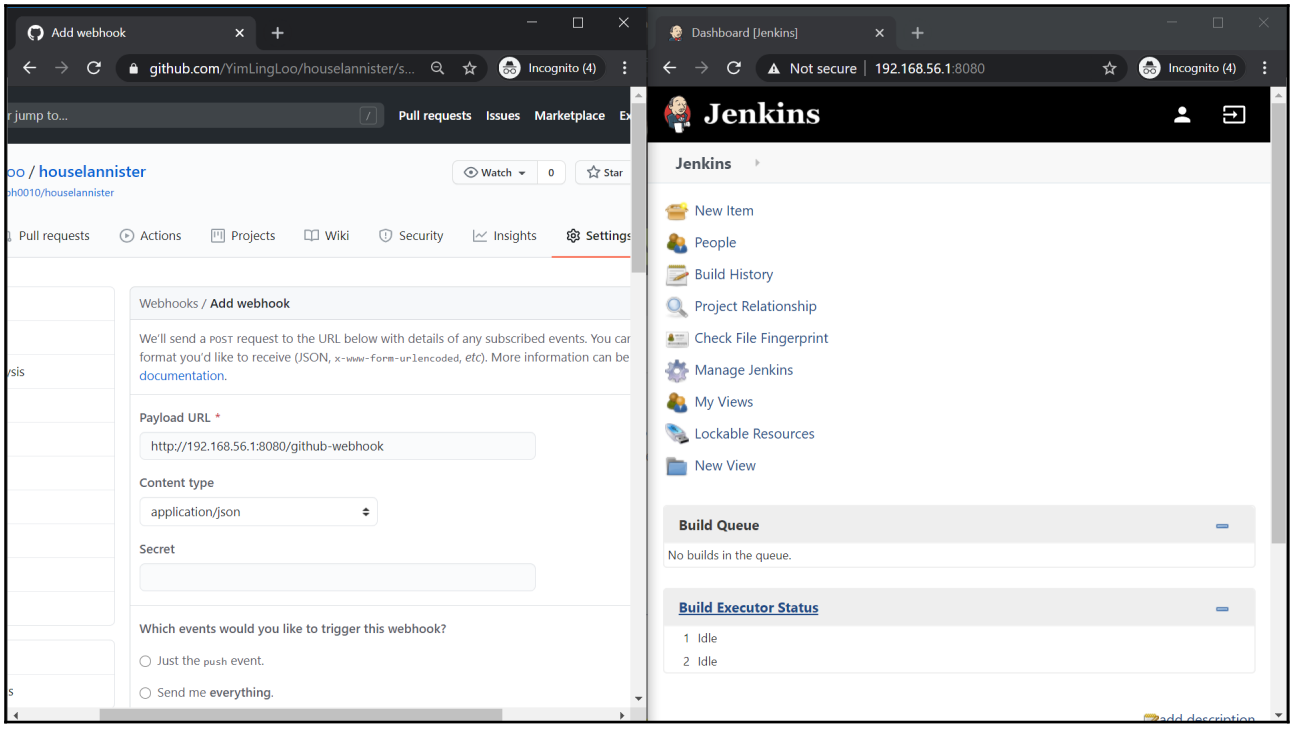


Figure 7: Enter webhook "Payload URL" and configure "Content Type".

Then, in the "Which events would you like to trigger this webhook?", choose "**Let me select individual events.**" Then, select "Pull Requests" and "Pushes". At the end of this option, make sure that "Active" is checked then finally click on "Add webhook" as shown in Figure 8.

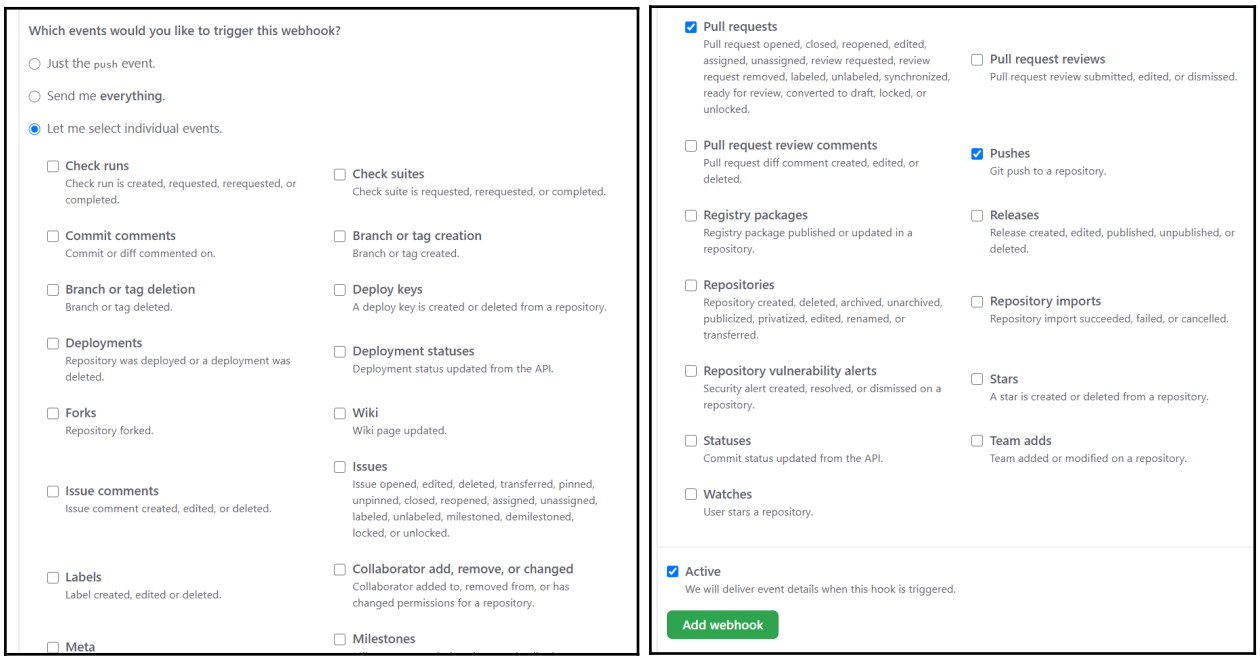


Figure 8: Select Pull and Pushes to be hooked on Jenkins URL.

Upon selecting "Add webhook", you've just finished configurating your repo to be always listened by Jenkins in order to have CI/CD.

Jenkins – A CI/CD Project Example.

Select "New Item" in the left navigation pane of Jenkins dashboard and create a new project as shown in Figure 9.

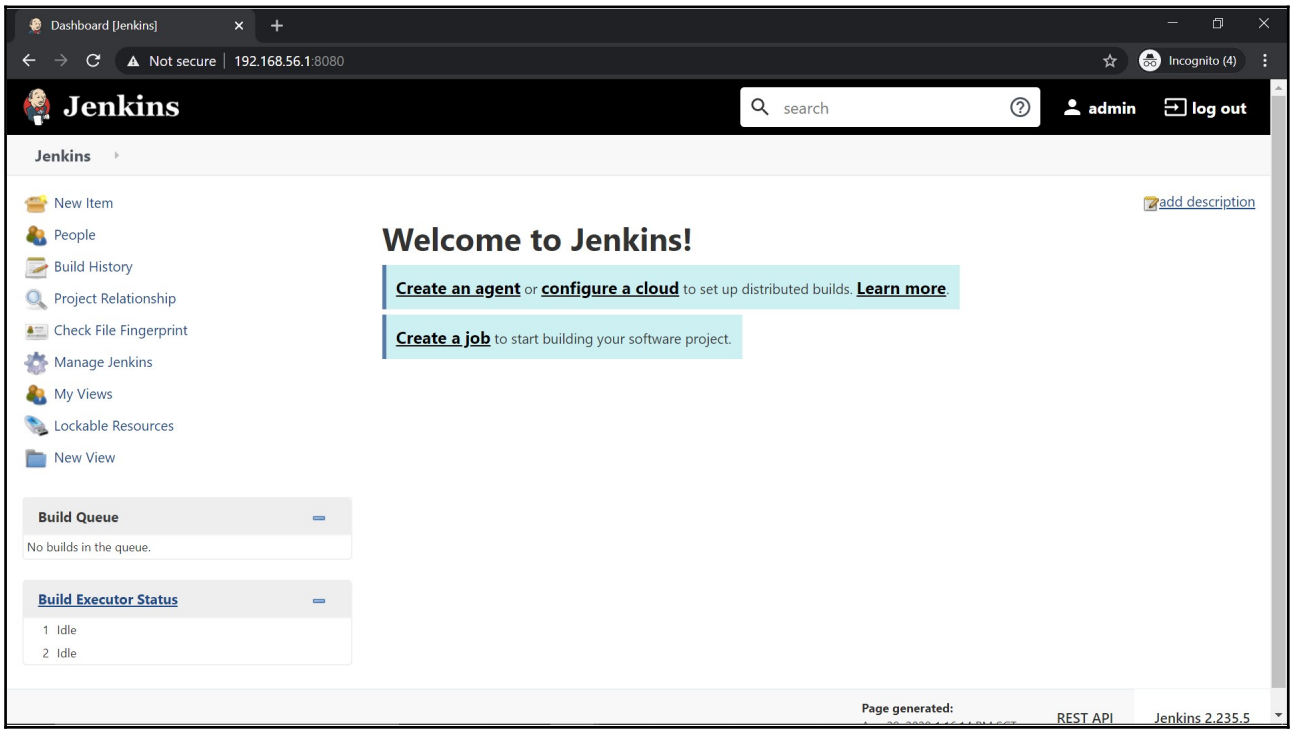


Figure 3: Select "New Item" to create a new project in Jenkins.

Then, give a name to this project and select "Freestyle project" as illustrated in Figure 10.

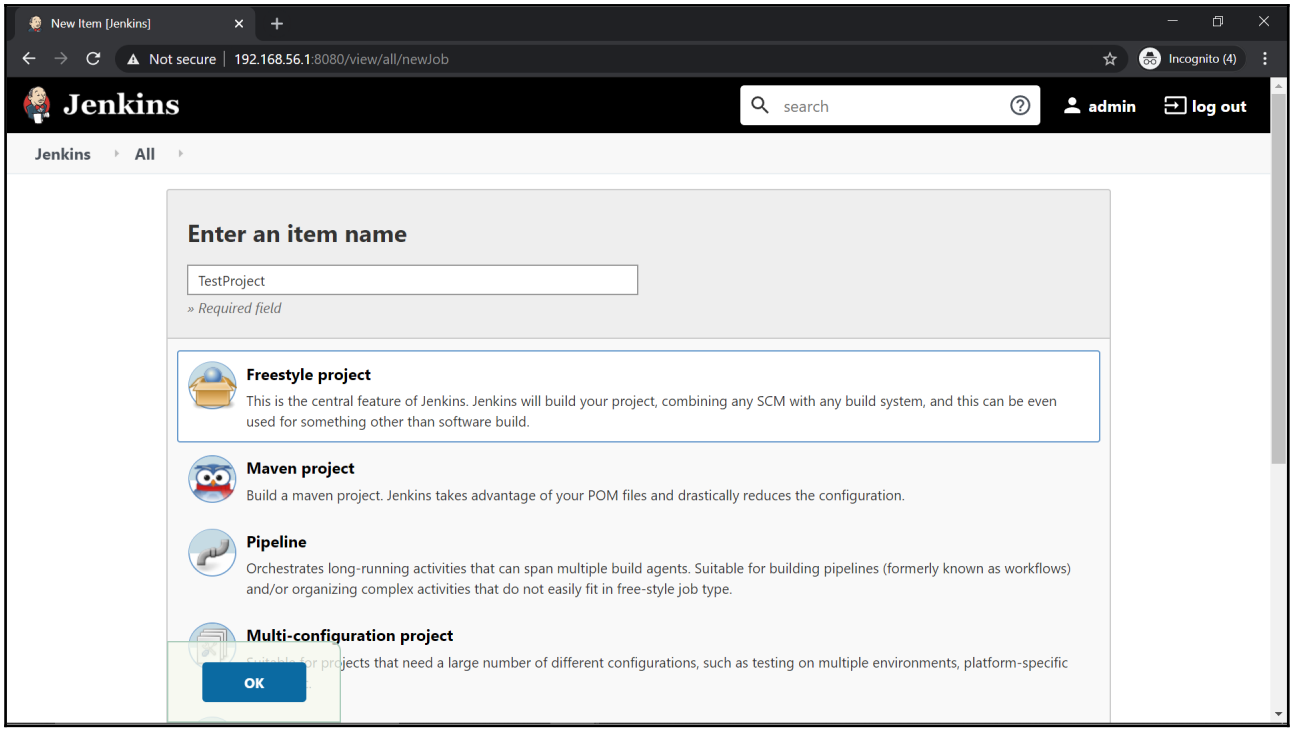
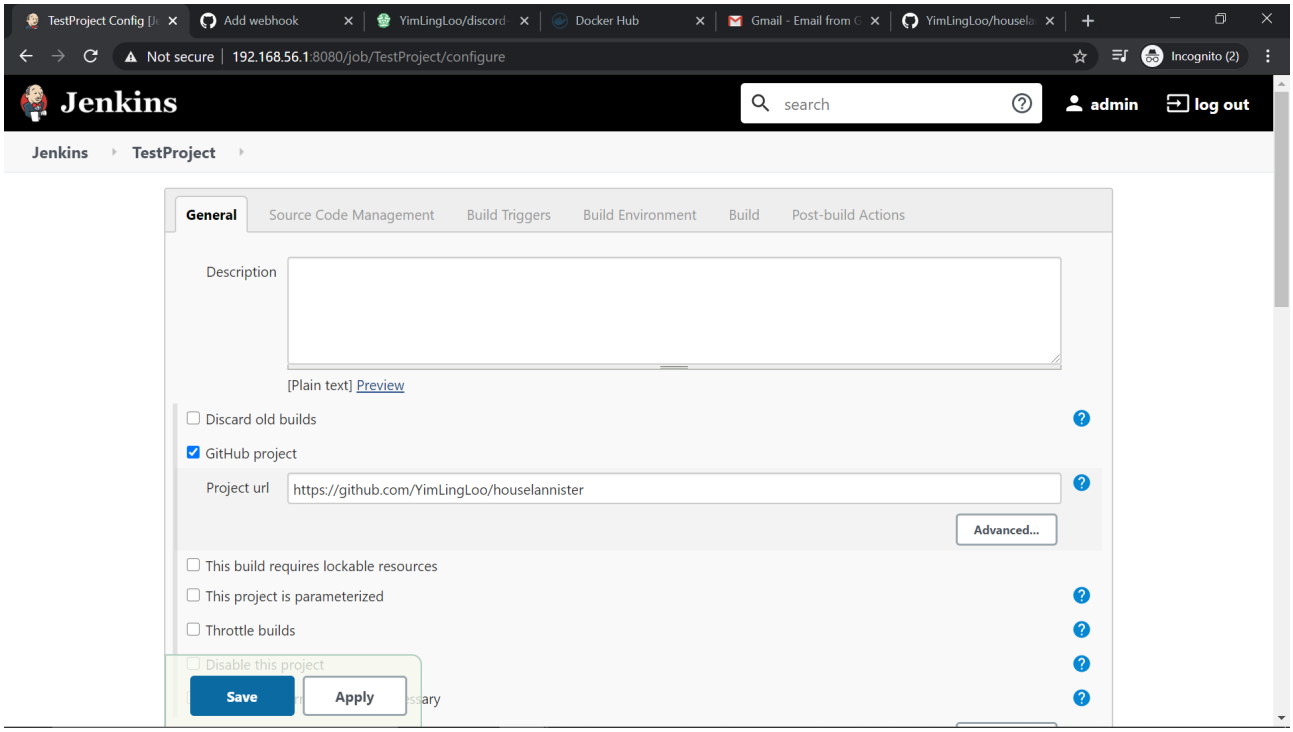


Figure 10: Name the project and select "Freestyle Project" as the project type.



The choice of "Freestyle project" enable a more flexible configuration and easy setup. After redirection to project configuration page upon creation of new project, select "Source Code Management" tab. Choose "Git" and enter a Github repository as illusttrated in Figure 11.

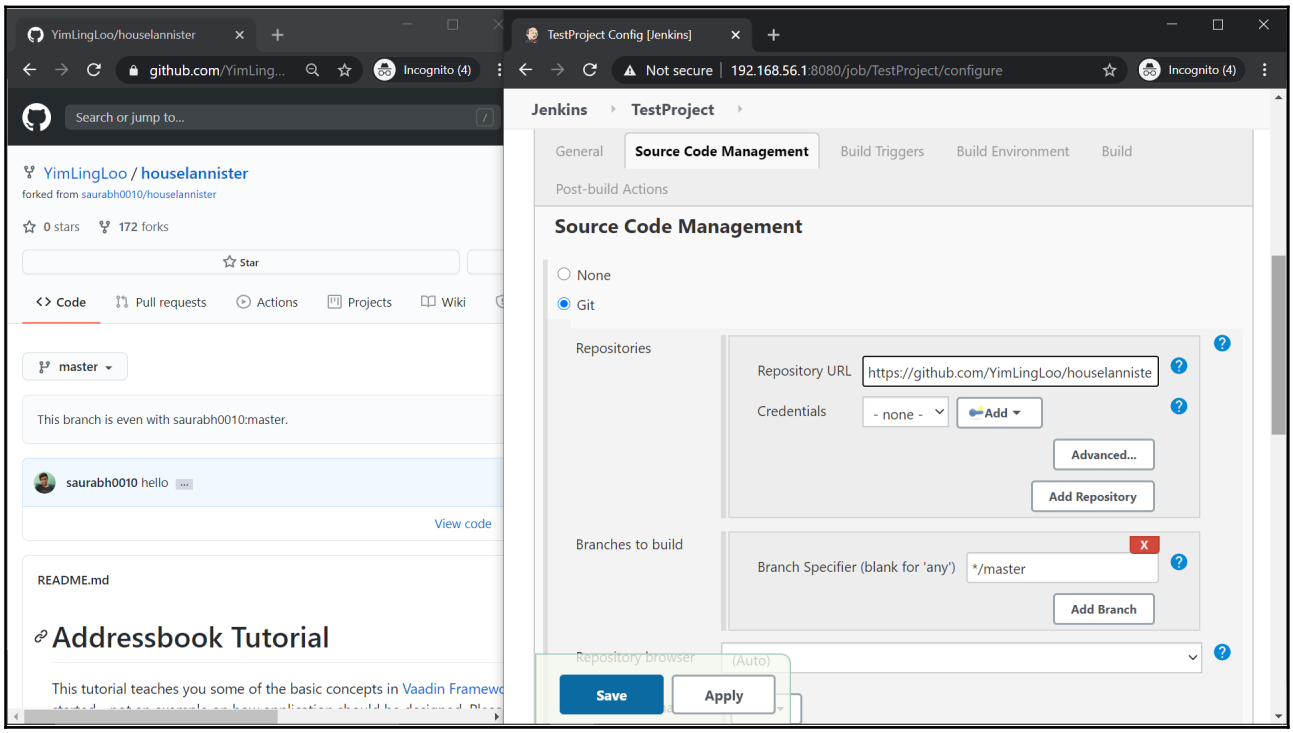


Figure 11: Enter Git repository into the source code management configuration.

Then, click on the "Build Triggers" tab and then select "GitHub hook trigger for GITScm polling" as shown in Figure 12.

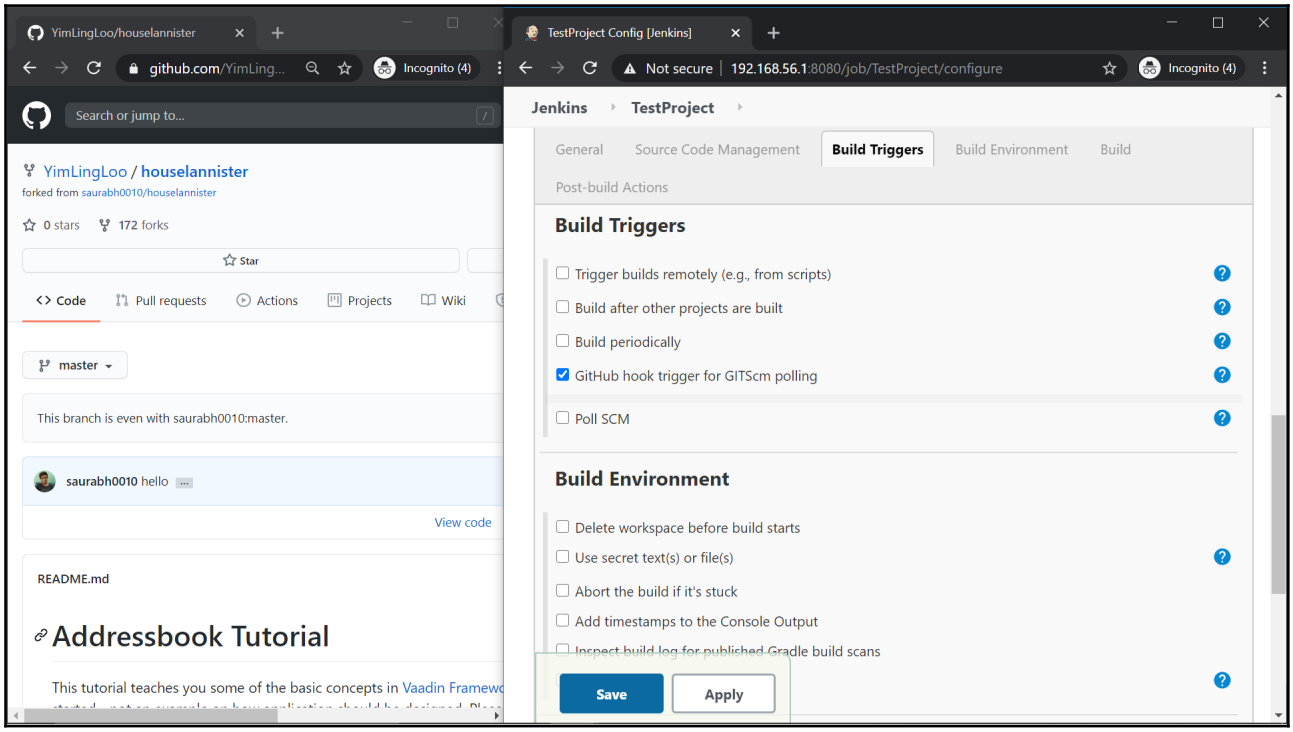


Figure 12: Configure Jenkins project to listen to trigger from Github hook.

In order to configure Jenkins job to run with every new commit to the Github repo, select “Build” tab. Choose “Invoke top-level Maven targets” and enter “compile” as Goals of the build in order to build up the project as shown in Figure 13.

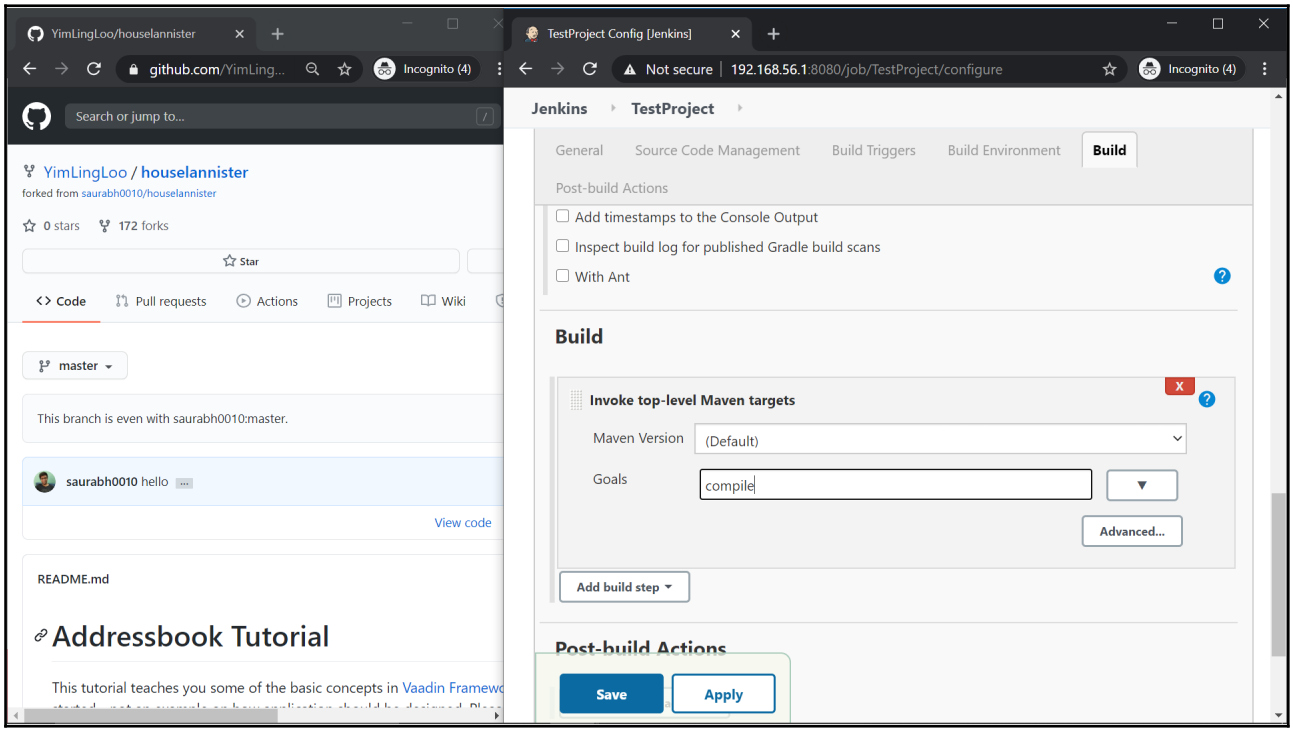


Figure 13: Use of Maven for build of Java project.

Upon selecting “Apply” and “Save”, Jenkins “TestProject” is now fully configured to listen to commits done in the Github repo that is hooked to it. Test the configured CI by committing a change in the Github repo and see that if Jenkins TestPorject is triggered for a build and report to Github as shown in Figure 14.

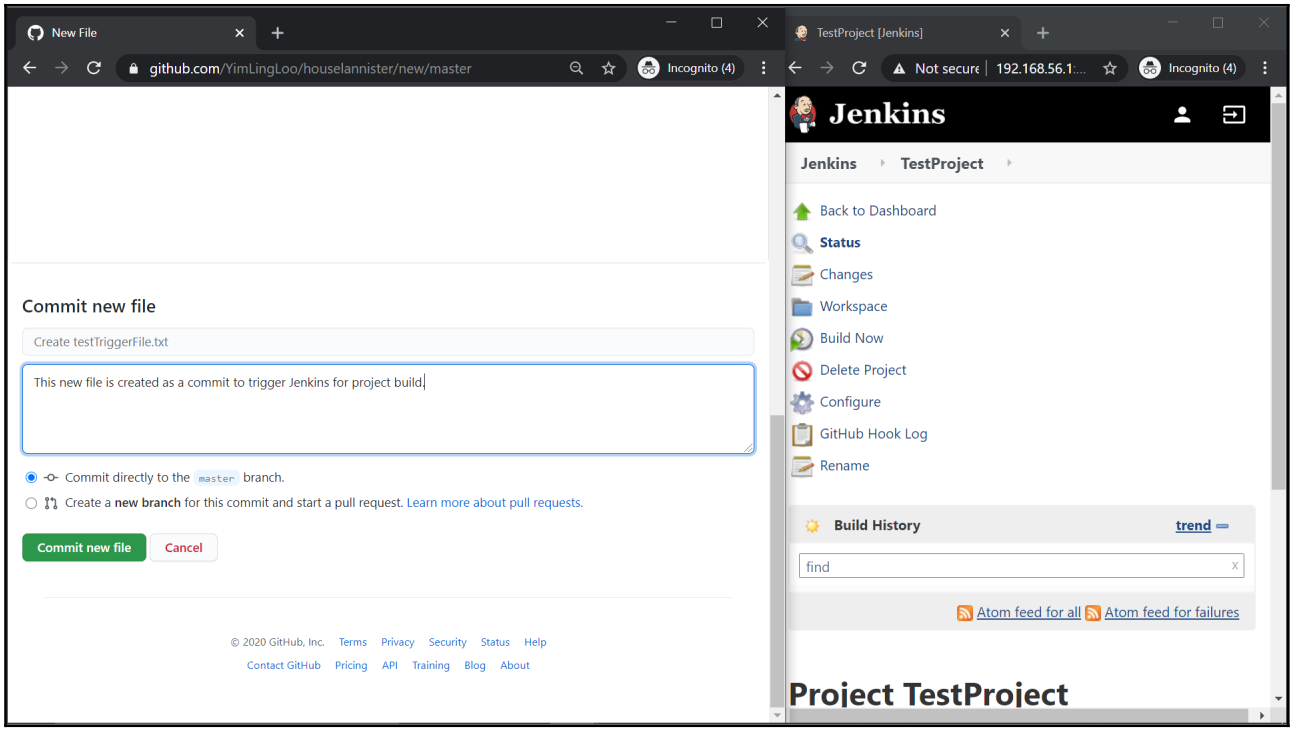


Figure 14: Create commit in Github and Jenkins for CI/CD.

Travis CI CI/CD.

Sign in to Travis CI account in https://travis-ci.com/. For this practical learning purpose, fork **https://github.com/bradmorg/discord-bot** Github repo into your Github account. Upon the successful fork of the repo, the same repo should be in your Github account, as illustrated in Figure 15.

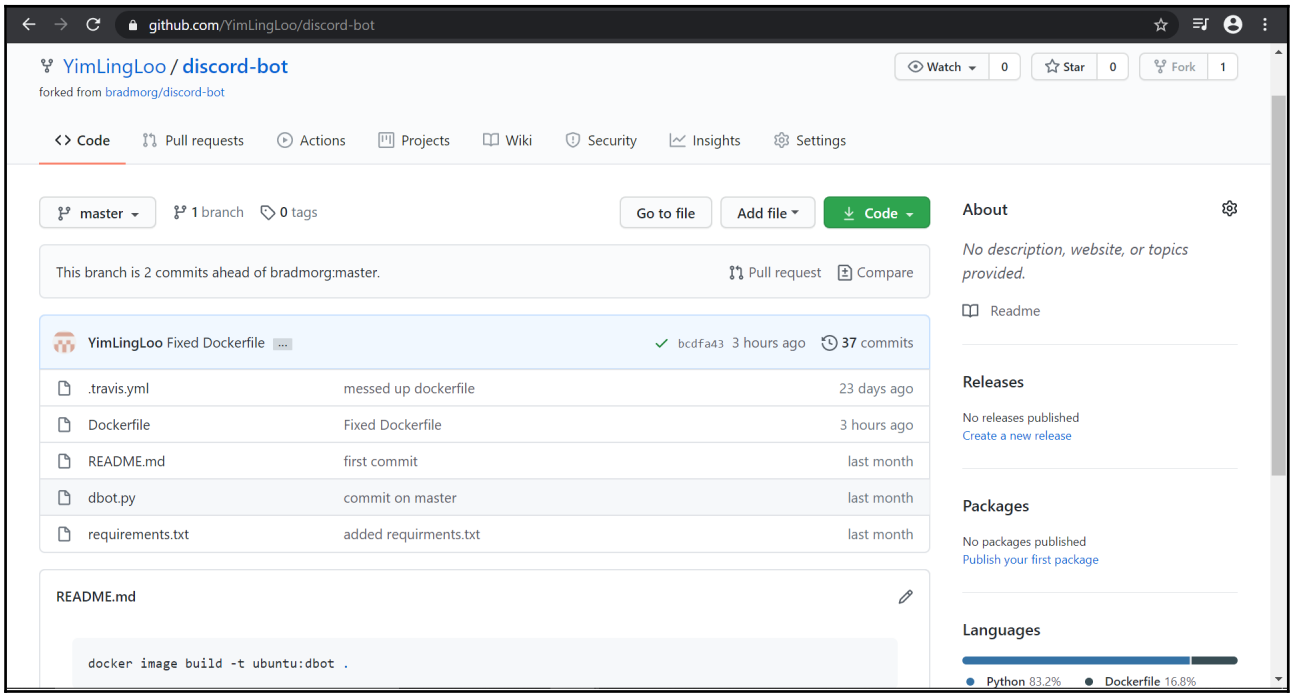


Figure 15: Fork the guest repo into own Github account.

The repo cotains a Discord bot python project. A quick overlook on the files in the project:

1. **dbot.py**: the Discord bot is written in python, which creates a chat bot in Discord application.
2. **Dockerfile**: the project is set to be deployed in a Docker container with all configurations setup for the project deployment and dependencies setup listed in the following file.
3. **requirements.txt**: in which the python will run and install all the listed dependencies
4. **travis.yml**: defines the architecture that Travis CI will build (arch), setting environment variables (to be detailed in Travis CI), configure service to Docker for Docker image build and deploy in container. The Docker image is built before deployment, after successful build, the Docker image is deployed in container. While Docker service is used, Docker username and password need to be configured in Travis CI.

In order to configure the environment variables so that Travis CI is able to run, build, test and deploy the project, go to Travis CI list of Github repositories (by now, you should've discord-bot repo linked to your Travis CI account). Select discord-bot repo then select "Settings" under the "More options" drop-down menu as illustrated in Figure 16.

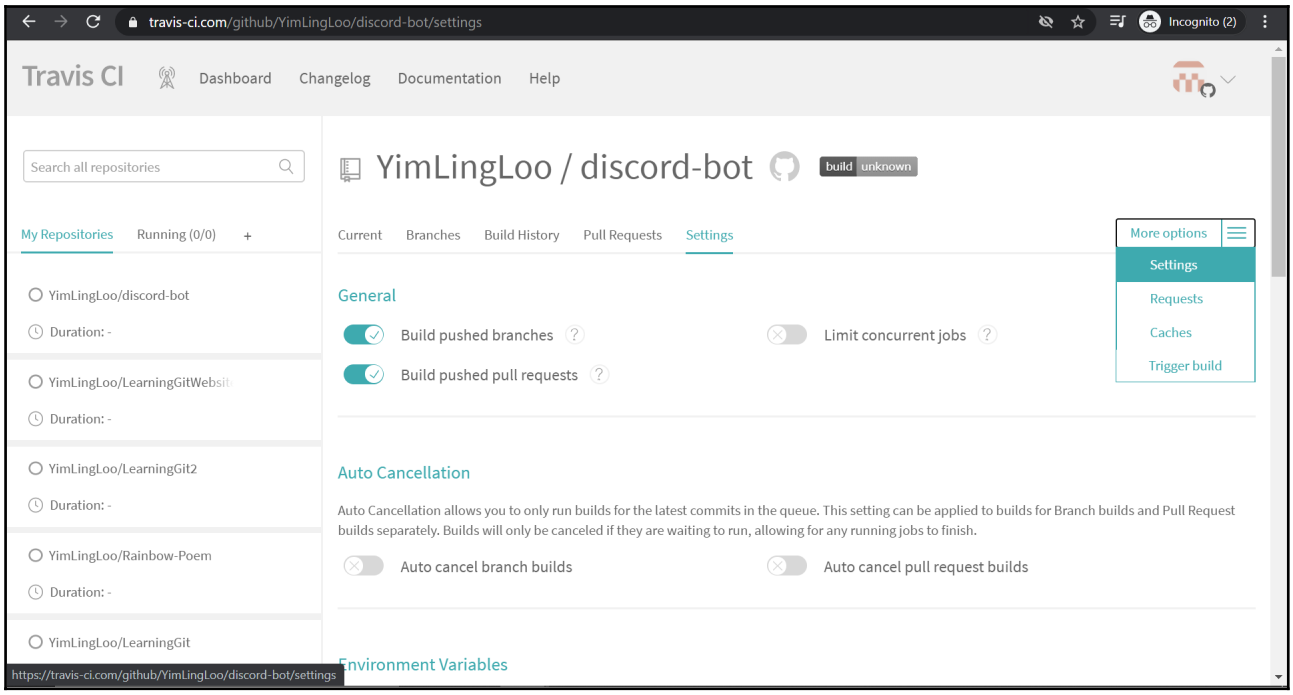


Figure 16: Setup the repo in Travis CI.

Then, scroll to Environment Variables section to add the environment variables as shown in Figure 17.

1. DHUB_PASS; <Your Docker Hub login password>
2. DHUB_USERNAME; <Your Docker Hub login username>
3. IMGNAME; any of your preferred name e.g. discordbot
4. INSTALL_LOCATION; /home/pi/discord-bot
5. IPADDR; Your DOCKER_HOST IP (found in Environment Variables) as shown in Figure 18.

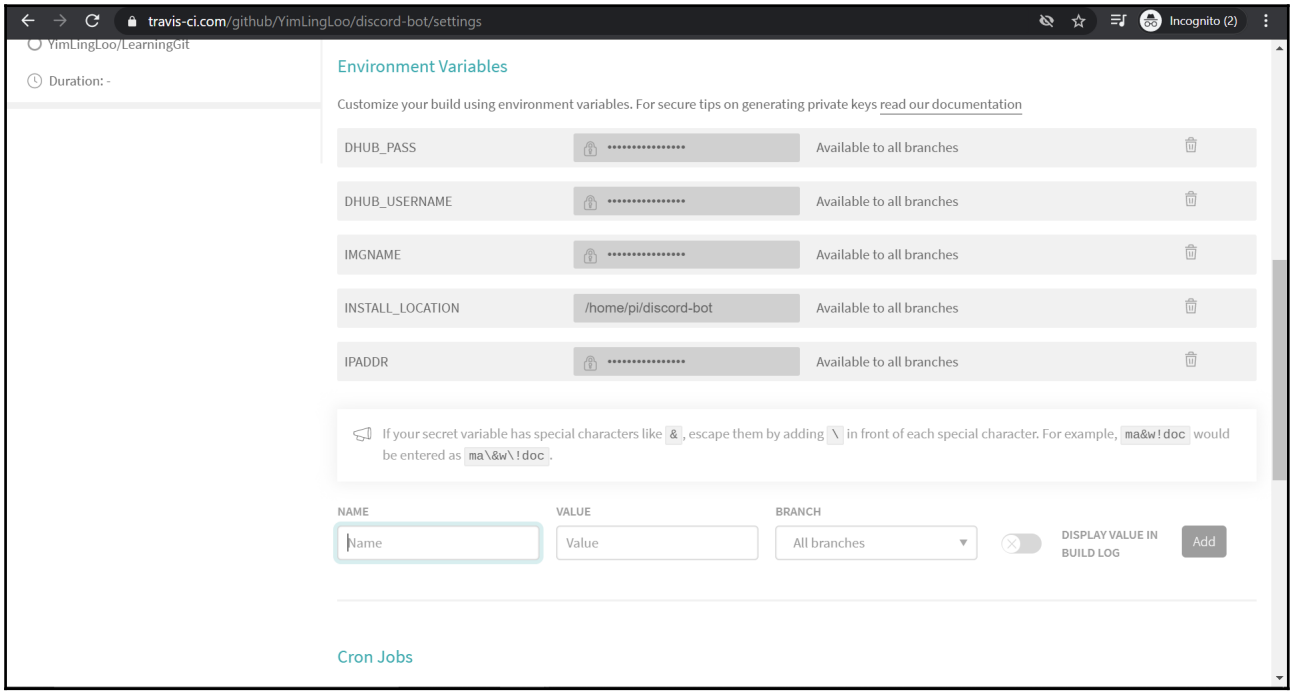


Figure 17: Travis CI Environment Setup.

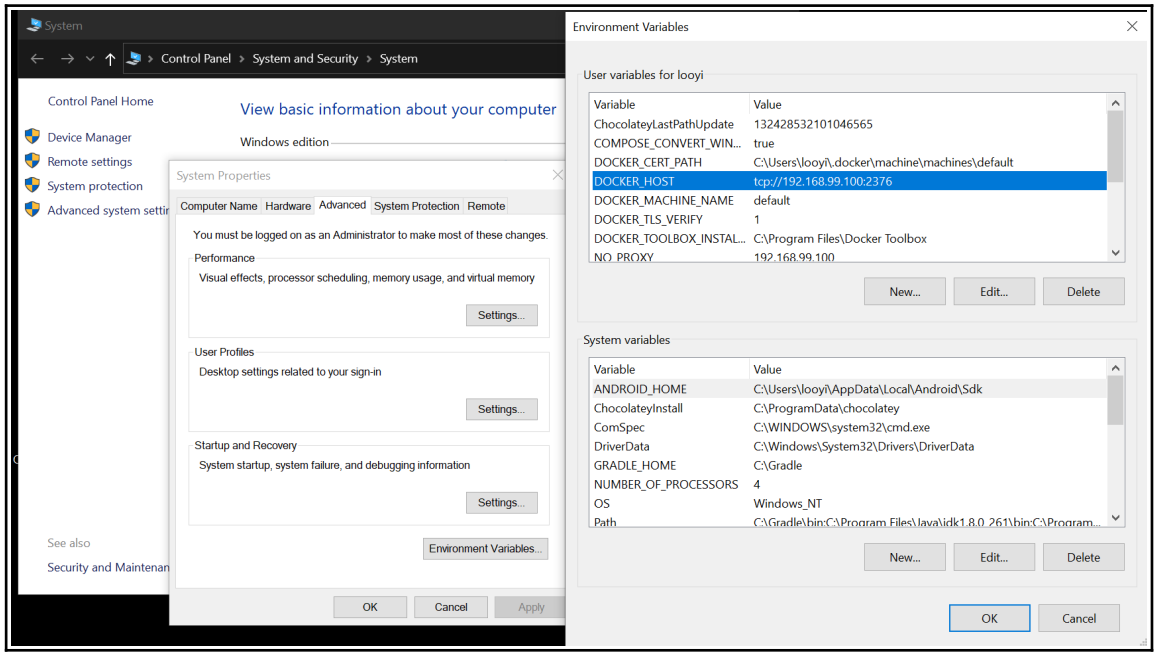


Figure 18: Docker Host for IPADDR.

After completing the add of environment vartiables, Travis CI is fully configured for CI/CD of the discord-bot project. Since travis.yml file is building the project into Docker image and deploying the project on Docker container, let's commit a change that will mess up the Dockerfile, and see that Travis CI is triggered for CI/CD of the project.

An example of commit made and trigger of Travis CI is illuastrated in Figure 19-22.

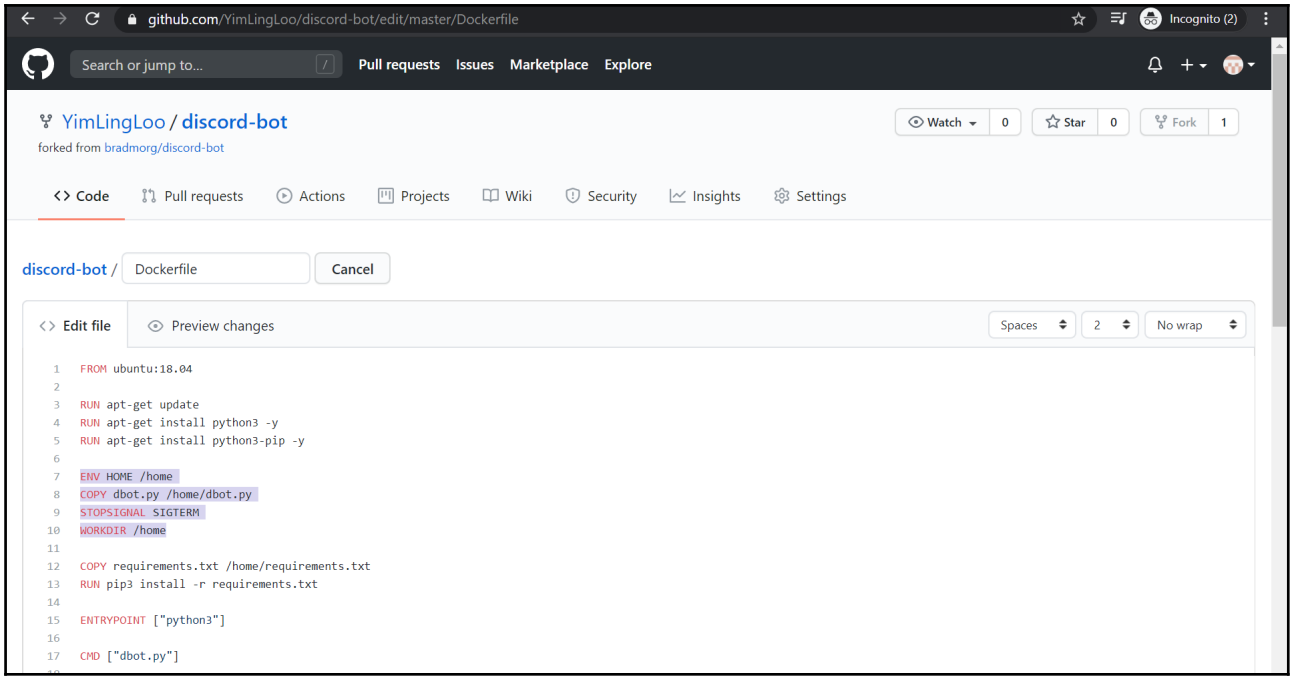


Figure 19: Delete the highlighted codes in Dockerfile.

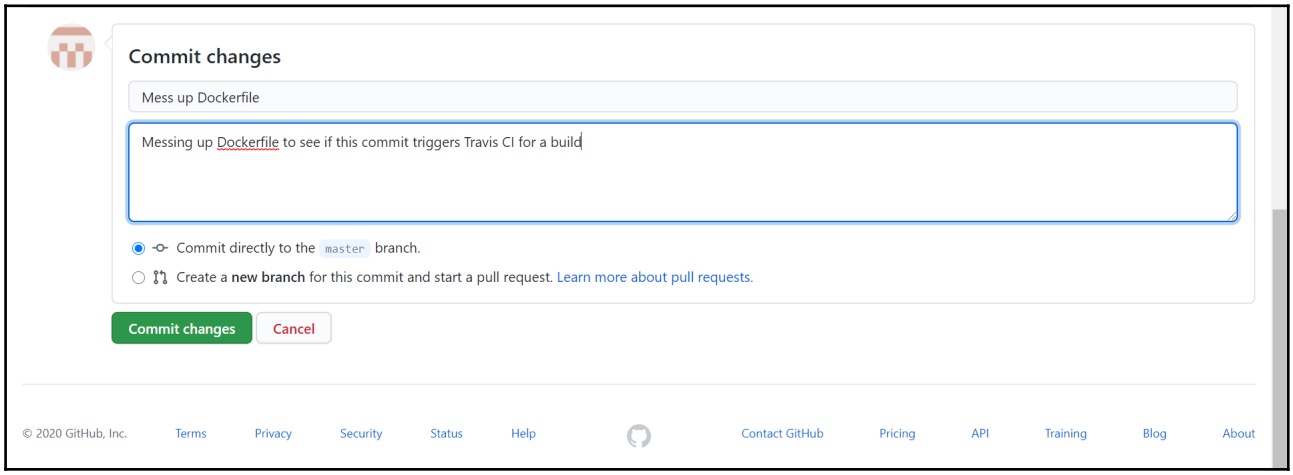


Figure 20: Commit the change in Github.

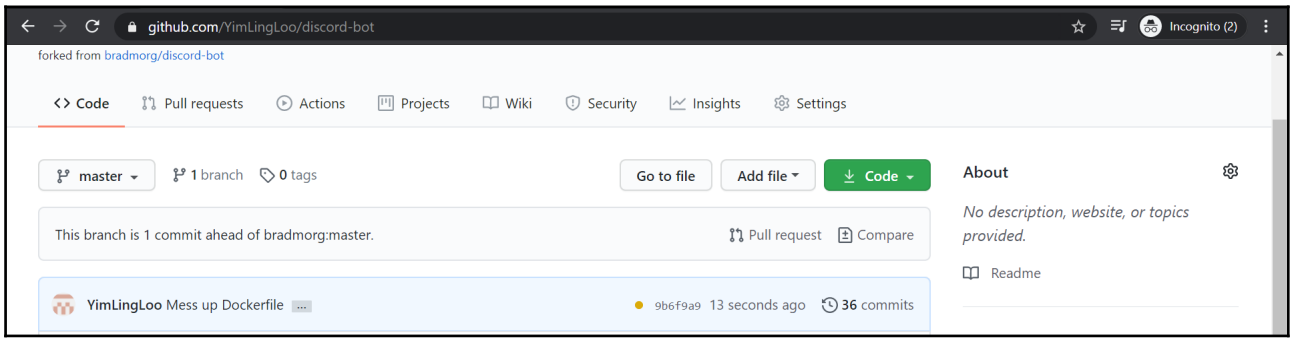


Figure 21: An orange-filled circle denoting that Travis CI is in execution for the commit made.

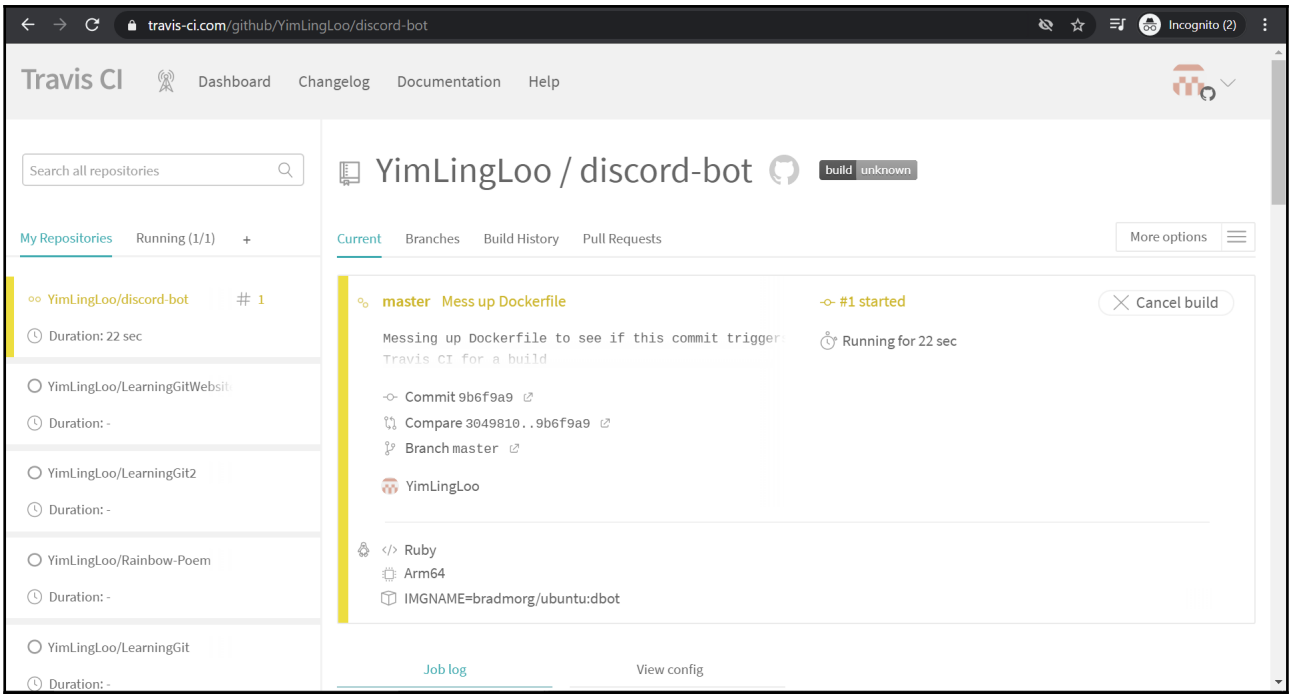


Figure 22: CI/CD started in Travis CI but end up in failure as the commit induced errors.
**Note: take a look at the details; the change commit description is included.*

Now, let's create a commit that will fix the erronous integration/deployment of the project. Fix the Dockerfile into its original workable file and see that Travis CI is triggered for CI/CD execution. An example of fix is illustrated in Figure 23-26.

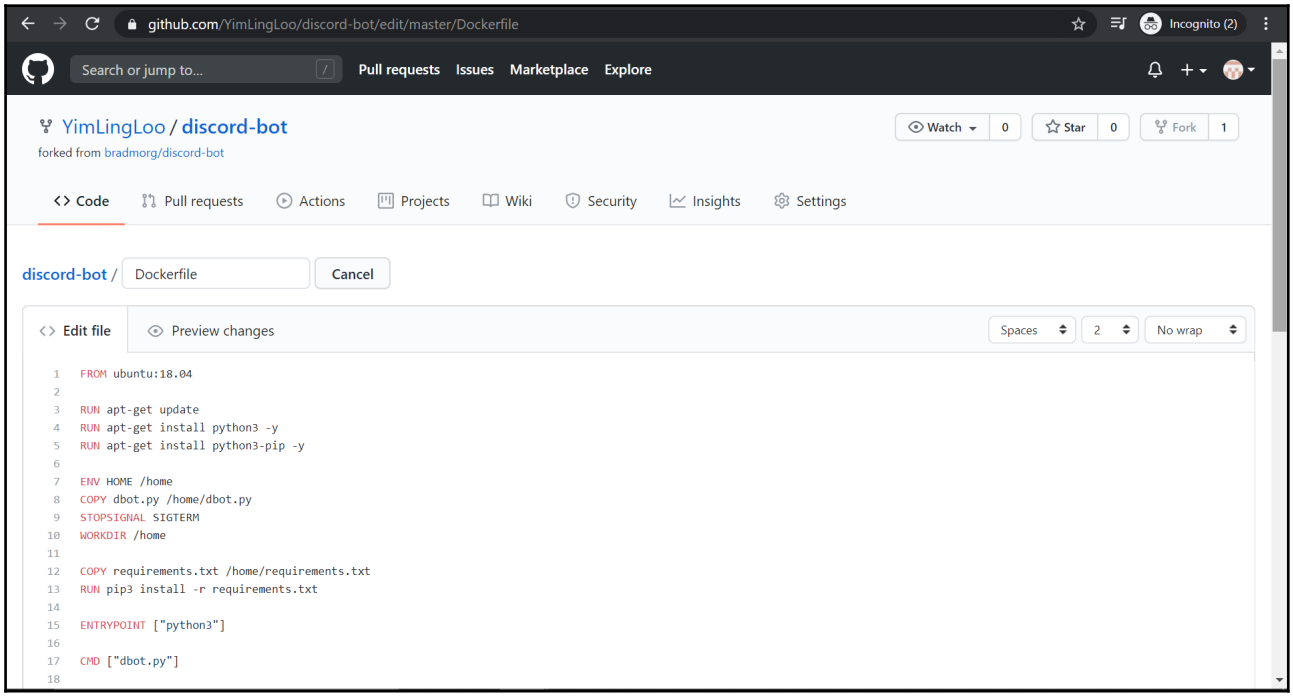


Figure 23: Fix Dockerfile and commit the change.

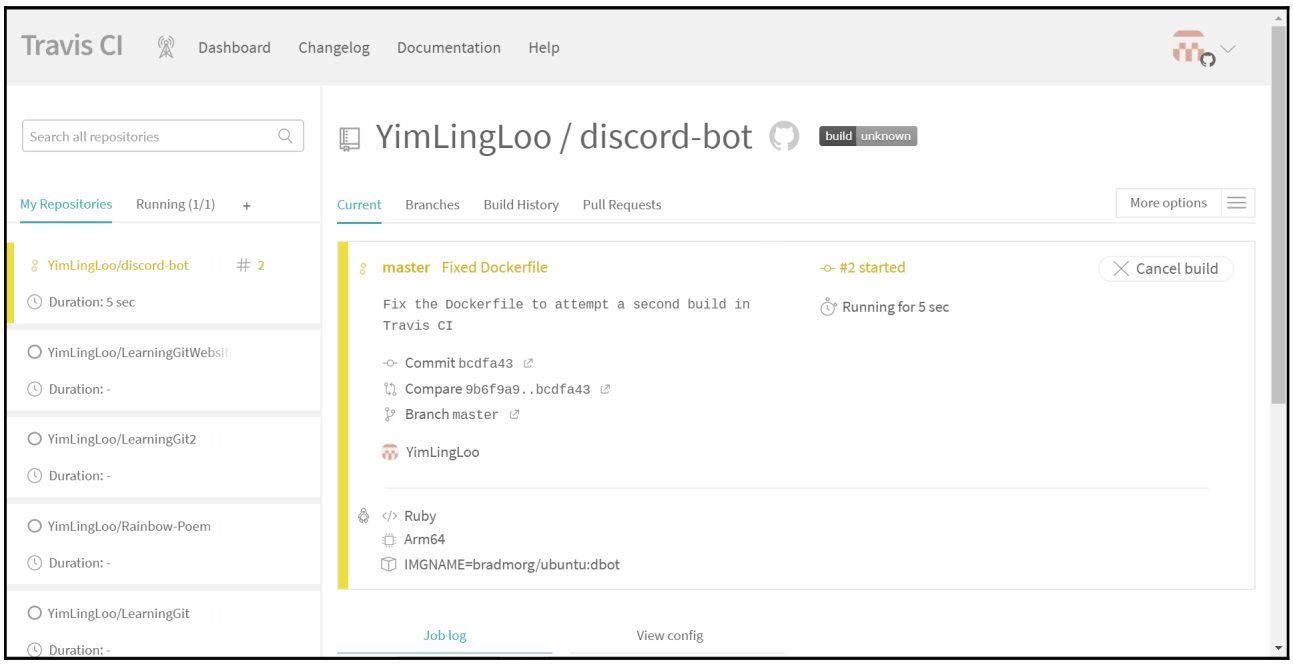


Figure 24: Travis CI is triggered for CI/CD execution (this will take a long while).

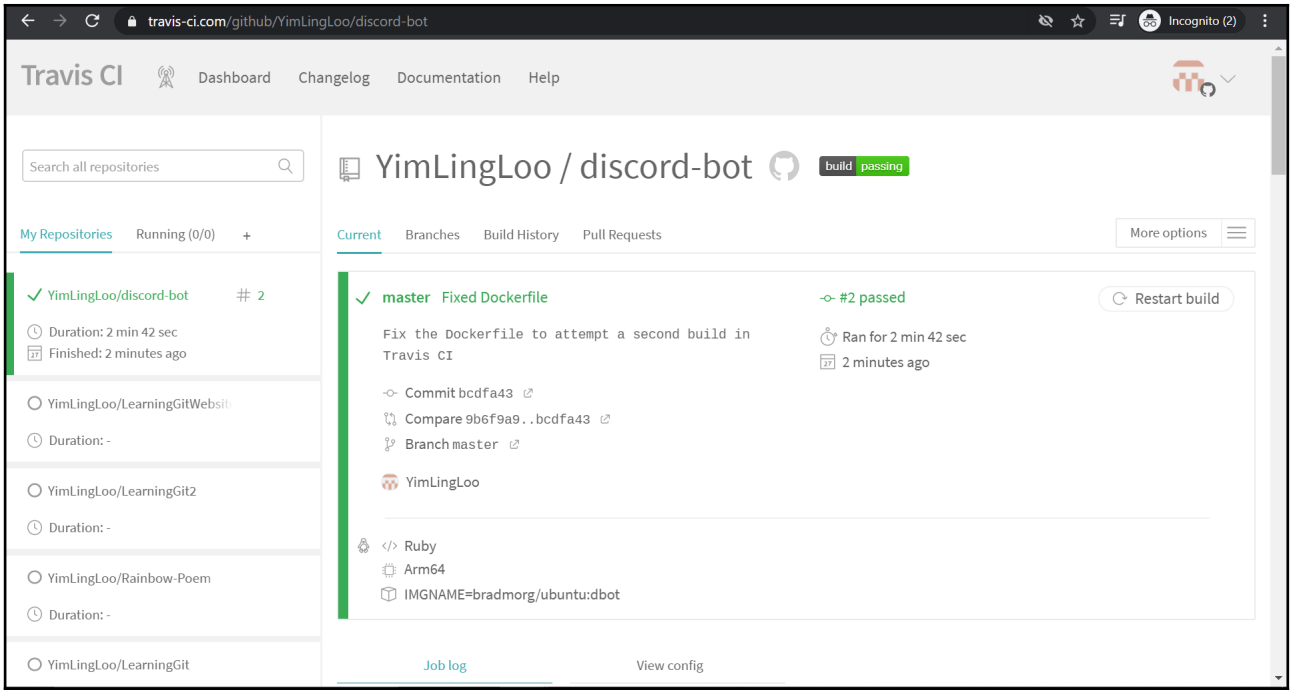


Figure 25: Successful CI/CD with the lable of "passed".
**Note: take a look at the details; the change commit description is included.*

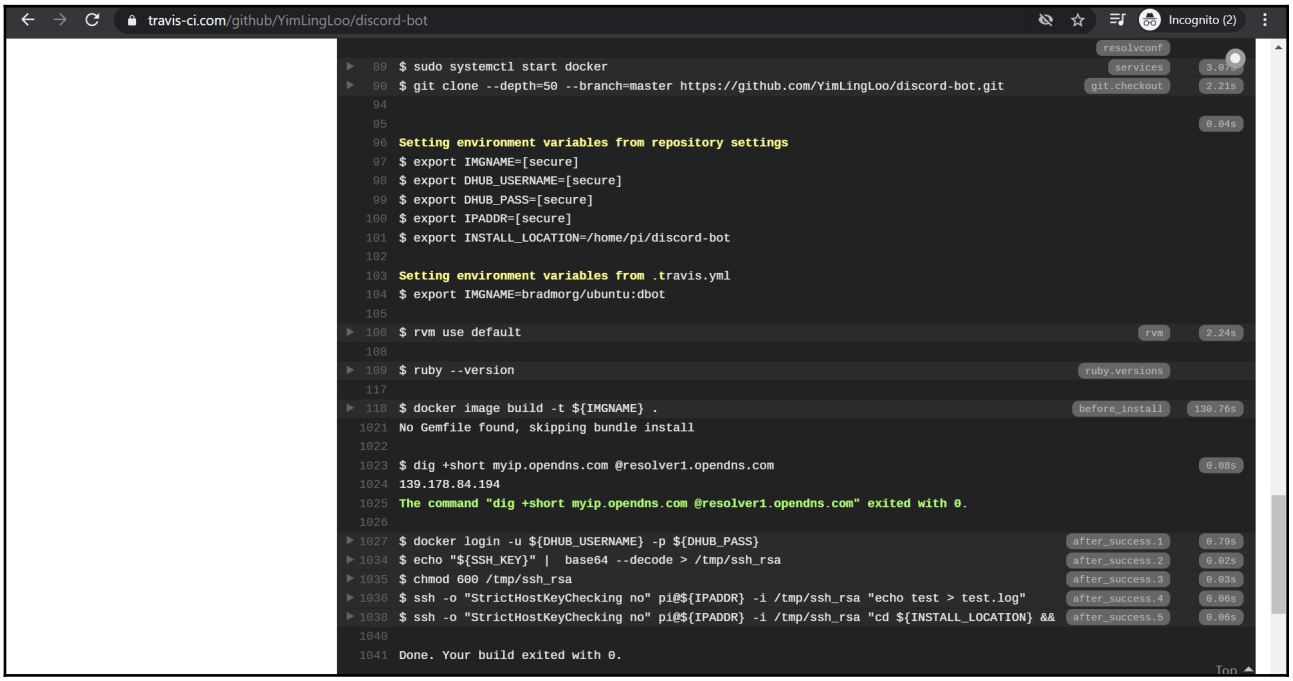


Figure 26: Detailed report on CI/CD execution in Travis CI.

Q: What are the differences of having development, integration, deployment, feedback with CI/CD?