Devops Project

Abstract

In this project we are packaging a java file using Jenkins. We are using Git for taking the java file for version controlling. Continuous Integration is being used in the Jenkins to create an automatic pipeline for packaging the java file. Then we are deploying the java file into the docker image. At last the application is being tested using the selenium testing tool.

Introduction

We are working in a phase where we are having multiple developers working on the same code in an organization, the updation in the code can be made by any developer. This can arise the conflicts in the original code if everyone works on it. To overcome this issue and provide better usability, the platform Git is used for the versioning control and also provide the facilities to work on the code globally as well as locally.

The main problem at the organization is the flow of the work, like the flow of development and flow of deployment and flow of testing. As all of them are now done with the help of automation tools. But we can now automate the complete environment from the development to the testing and to the complete deployment to the production to managing the production.

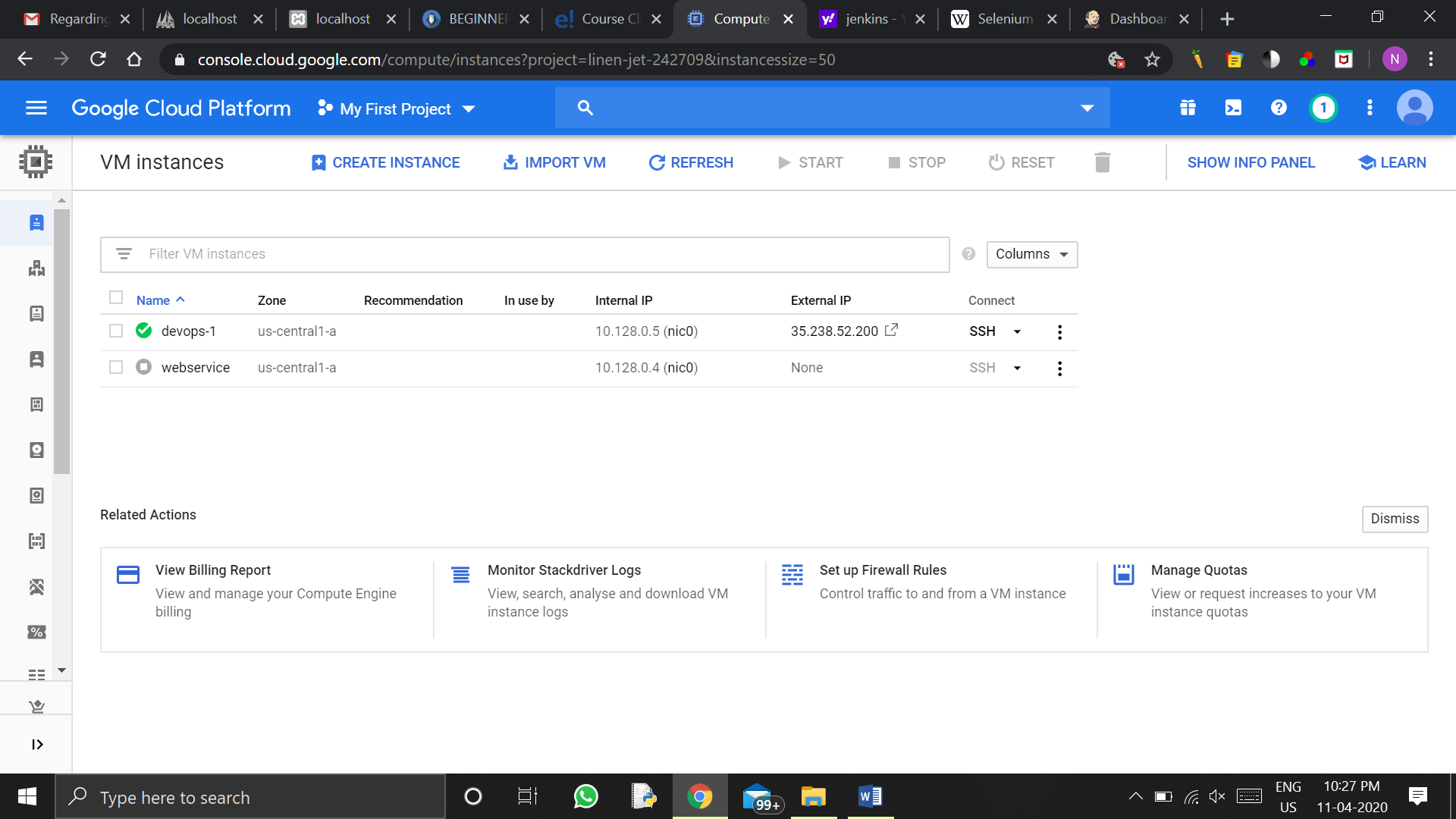
We are using Jenkins to automate the parts of the software development, implementing the continuous integration and continuous delivery. [Builds](https://en.wikipedia.org/wiki/Software_build) can be triggered by various means, for example by [commit](https://en.wikipedia.org/wiki/Commit_(version_control)) in a version control system, by scheduling via a [cron](https://en.wikipedia.org/wiki/Cron)-like mechanism and by requesting a specific build [URL](https://en.wikipedia.org/wiki/Uniform_Resource_Locator). It can also be triggered after the other builds in the queue have completed. Jenkins functionality can be extended with [plugins](https://en.wikipedia.org/wiki/Plug-in_(computing)).

We are using the docker which is a set of [platform as a service](https://en.wikipedia.org/wiki/Platform_as_a_service) (PaaS) products that uses [OS-level virtualization](https://en.wikipedia.org/wiki/OS-level_virtualization) to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, [libraries](https://en.wikipedia.org/wiki/Library_(computing)) and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single [operating system kernel](https://en.wikipedia.org/wiki/Kernel_(operating_system)) and therefore use fewer resources than [virtual machines](https://en.wikipedia.org/wiki/Virtual_machine).

Also we are using selenium provides a playback tool for authoring [functional tests](https://en.wikipedia.org/wiki/Functional_testing) without the need to learn a test [scripting language](https://en.wikipedia.org/wiki/Scripting_language) (Selenium IDE). It also provides a test [domain-specific language](https://en.wikipedia.org/wiki/Domain-specific_language) to write tests in a number of popular programming languages, including [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)), [Groovy](https://en.wikipedia.org/wiki/Groovy_(programming_language)), [Java](https://en.wikipedia.org/wiki/Java_(software_platform)), [Perl](https://en.wikipedia.org/wiki/Perl), [PHP](https://en.wikipedia.org/wiki/PHP), [Python](https://en.wikipedia.org/wiki/Python_(programming_language)), [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)) and [Scala](https://en.wikipedia.org/wiki/Scala_(programming_language)). The tests can then run against most modern [web browsers](https://en.wikipedia.org/wiki/Web_browser).

Procedure

We are using Google cloud platform for doing our project. First we will create an instance of Ubuntu 16.04 in the GCP make sure you allow the http and https request and create it, and then we will start the instance and open ssh window.



In the above image we have created a new instance named devops and open it in the ssh.

Now we will login as the root user by simple typing

sudo su –

we will now have to install Jenkins in our server. We can install it by following commands:

Adding Jenkins PPA

$wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add –

$sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

Installing Jenkins on Ubuntu 16.04

$sudo apt-get update

$sudo apt-get install jenkins

Then the Jenkins will be installed and we can access the Jenkins by the help of the public IP address of our devops server on the cloud.

As jenkins is a web browser it will be running on the 8080 port. And if any other web browser is already installed in your server than you have to change the ports of Jenkins by editing the file **/etc/default/Jenkins and changing the HTTP\_PORT.**

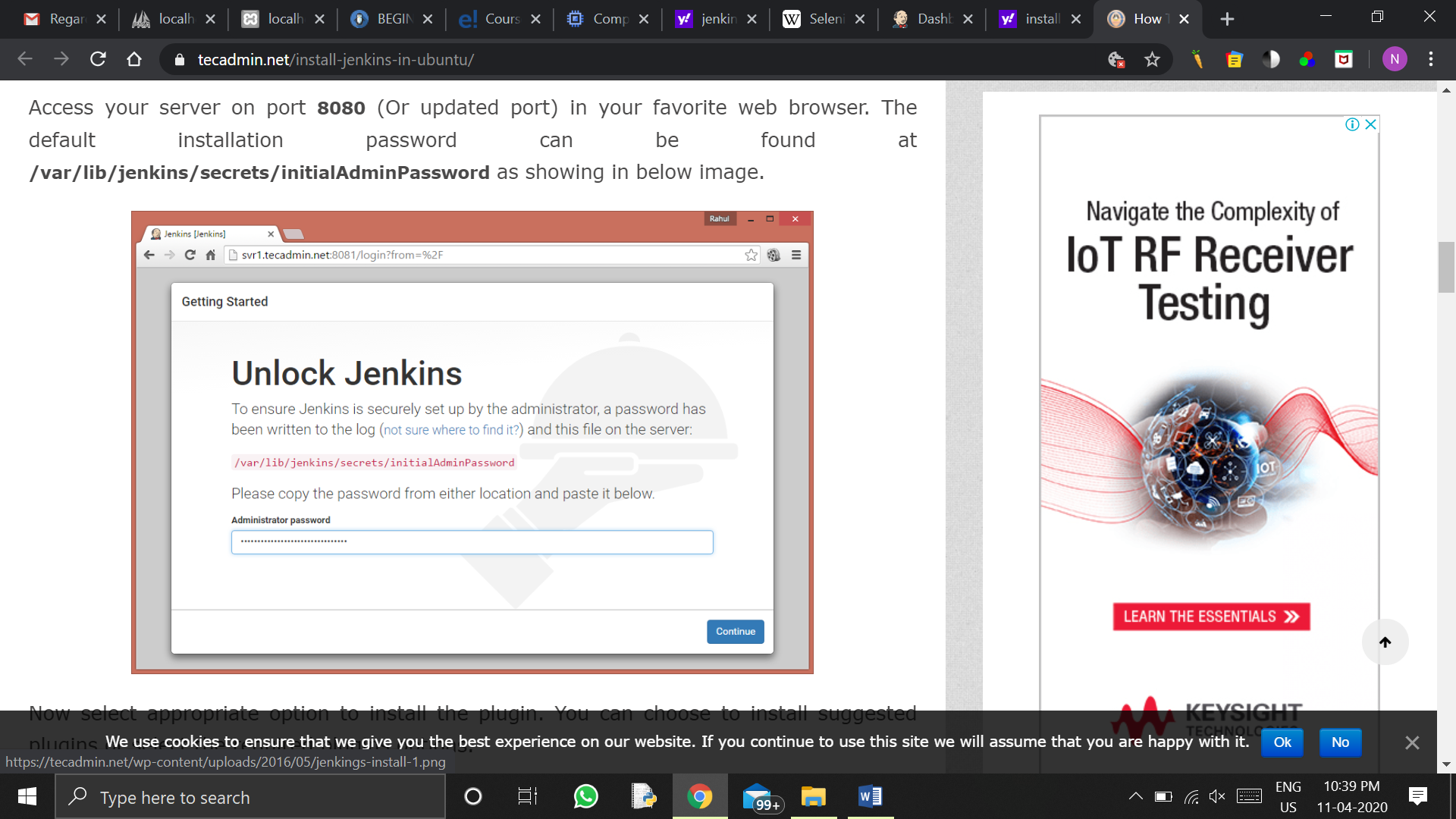
**Remember to rum this command after any updation in the file**

**$**sudo service jenkins restart

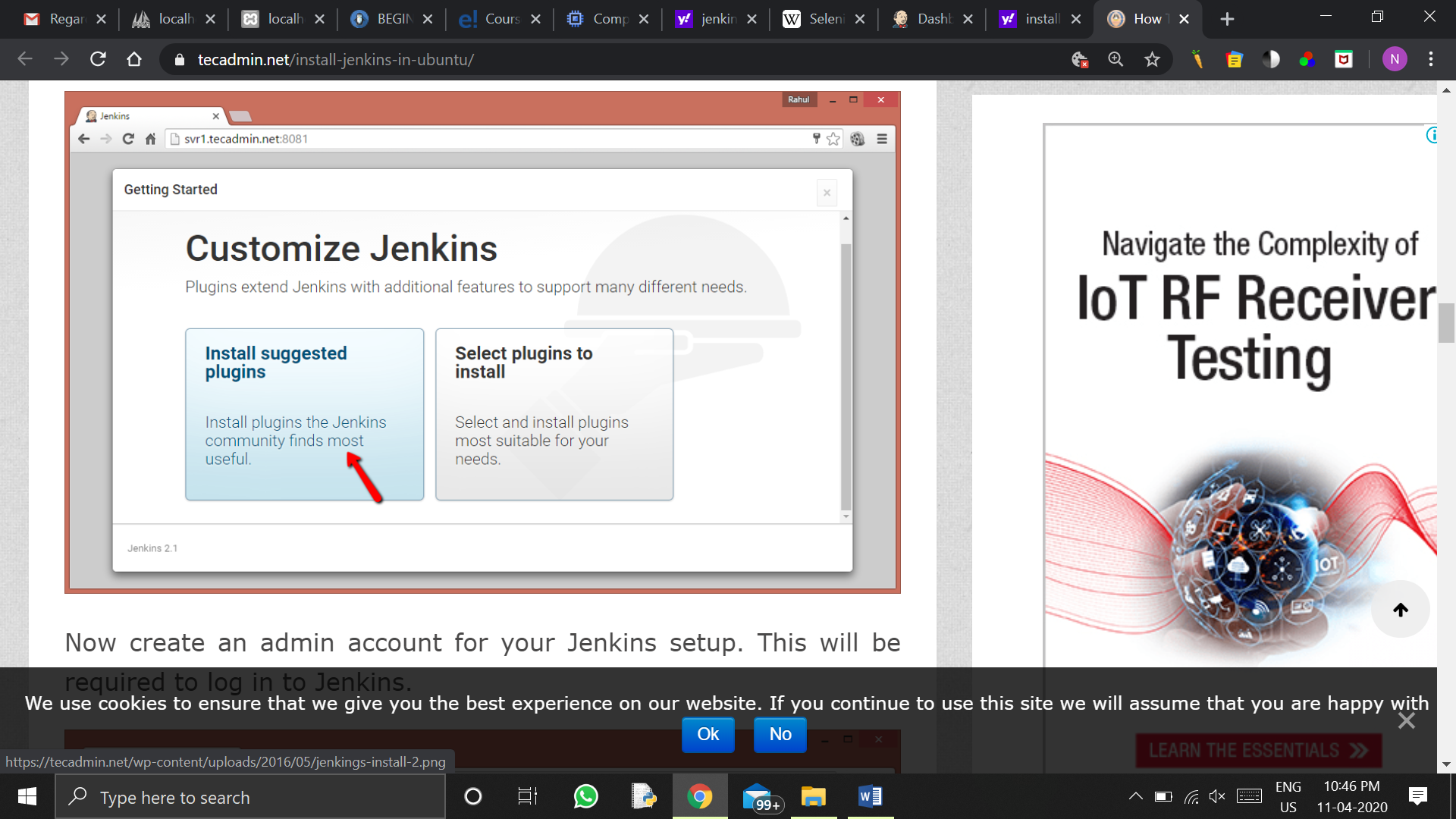
**We will open up it by pasting the public IP of server in a new tab and appending ‘:8080’ to it or the port number which you edit.**

**After the Jenkins window gets open it will ask you the password for the administrator.**

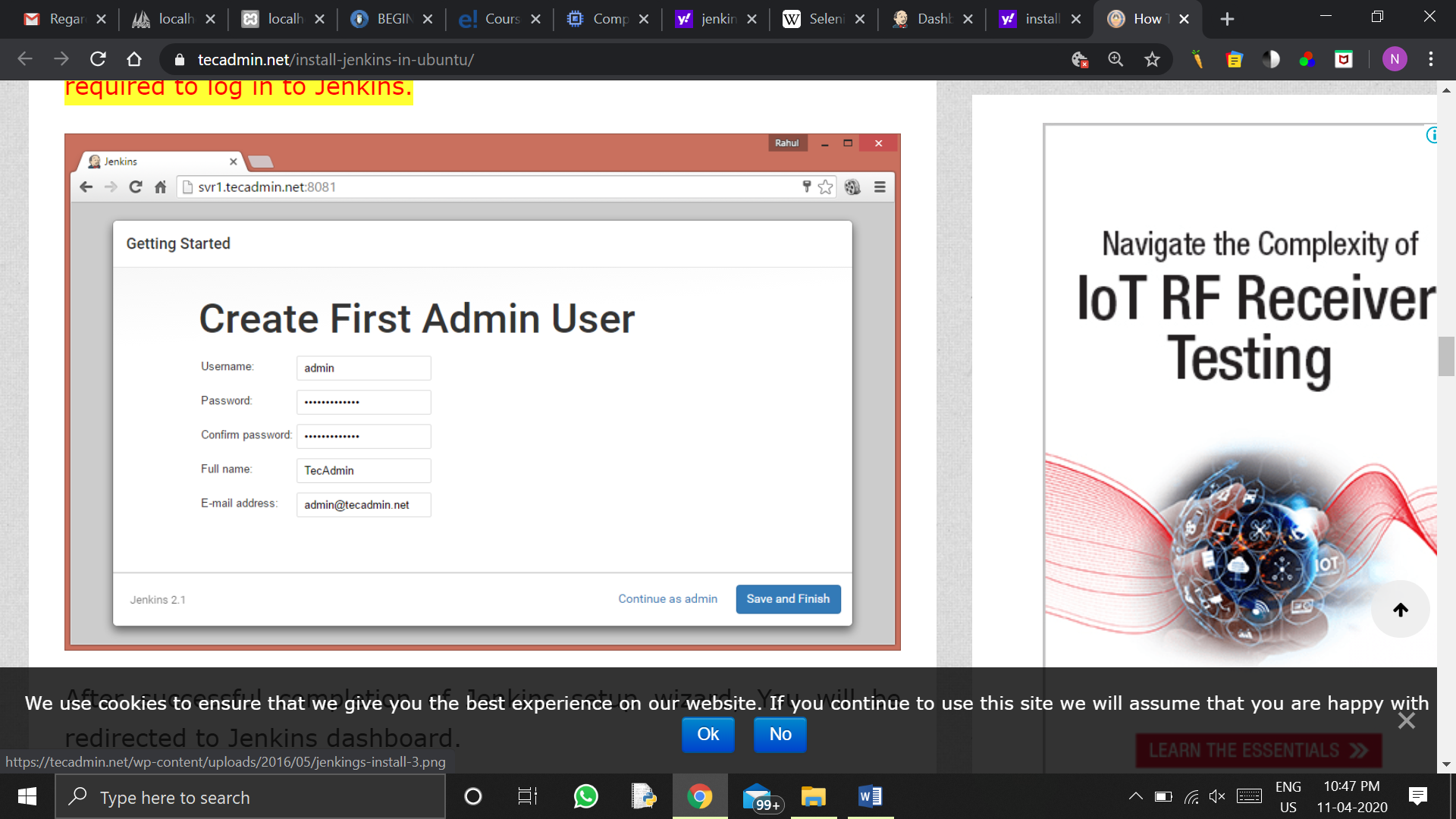
**You can find the password in the file /var/lib/jenkins/secrets/initialAdminPassword. Cat this file and paste the password in the Jenkins.**



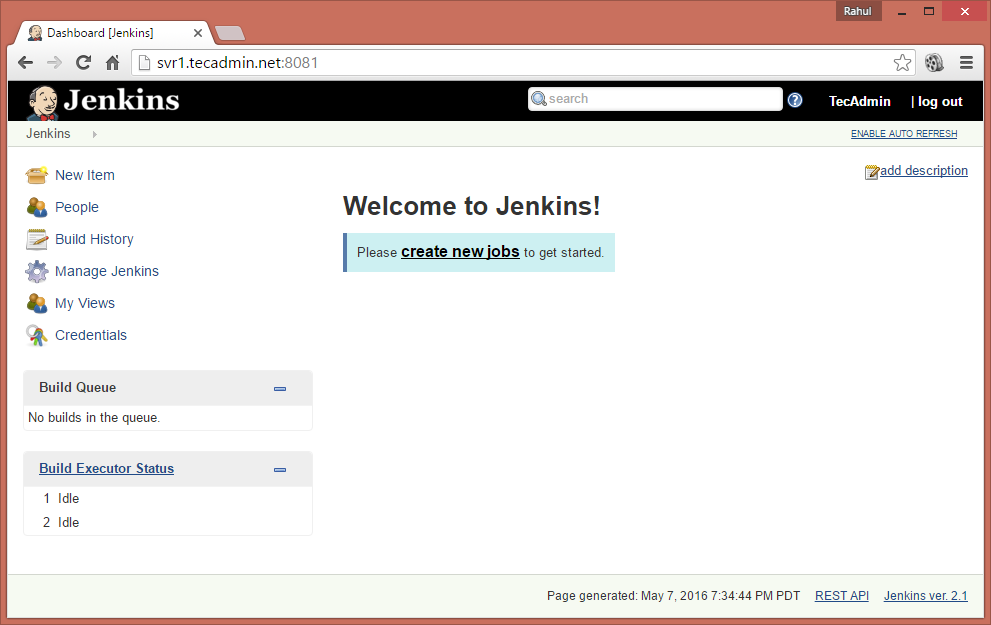
Now select appropriate option to install the plugin. You can choose to install suggested plugins or select the required plugins options.



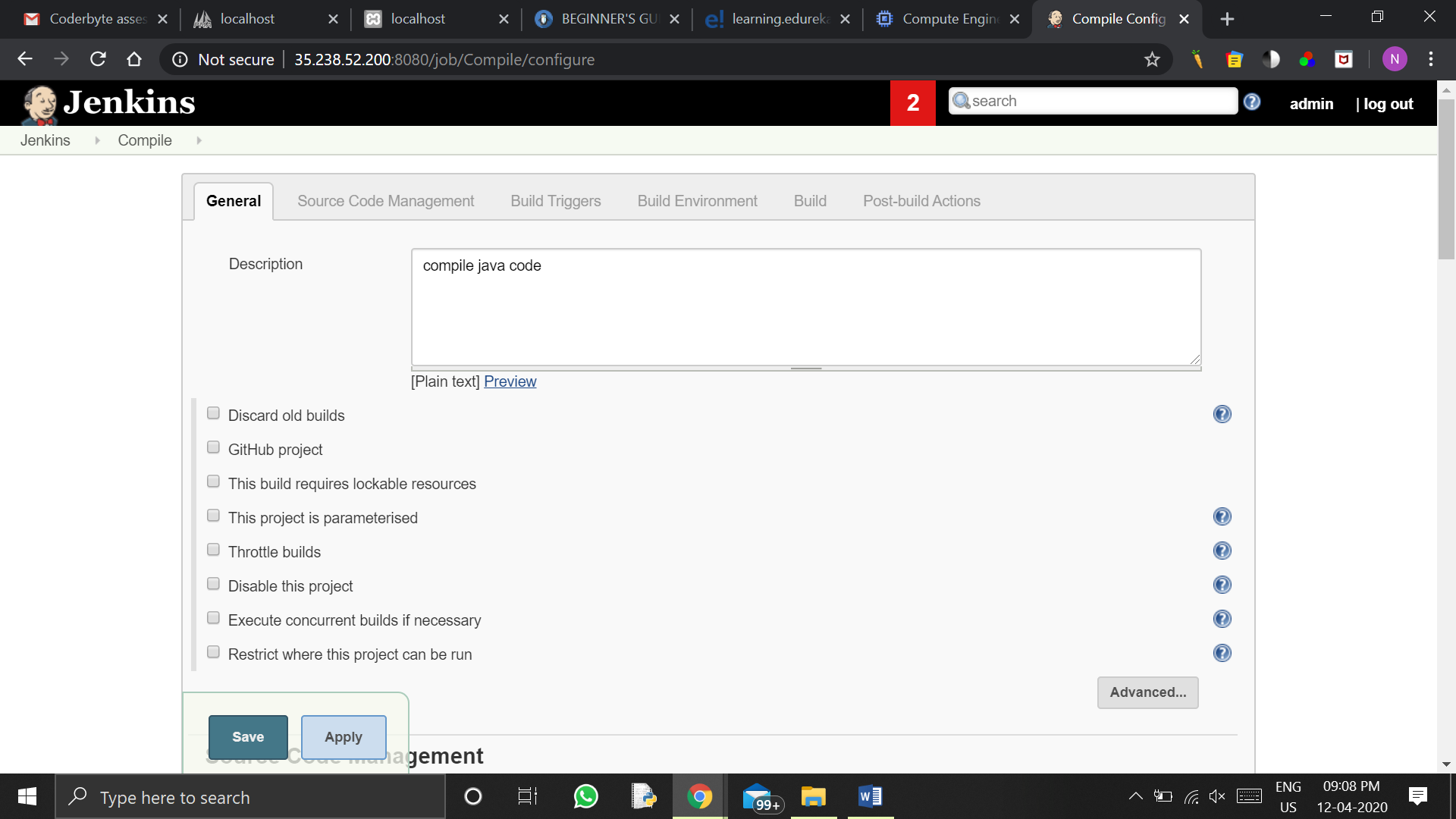
Now create an admin account for your Jenkins setup. This will be required to log in to Jenkins.



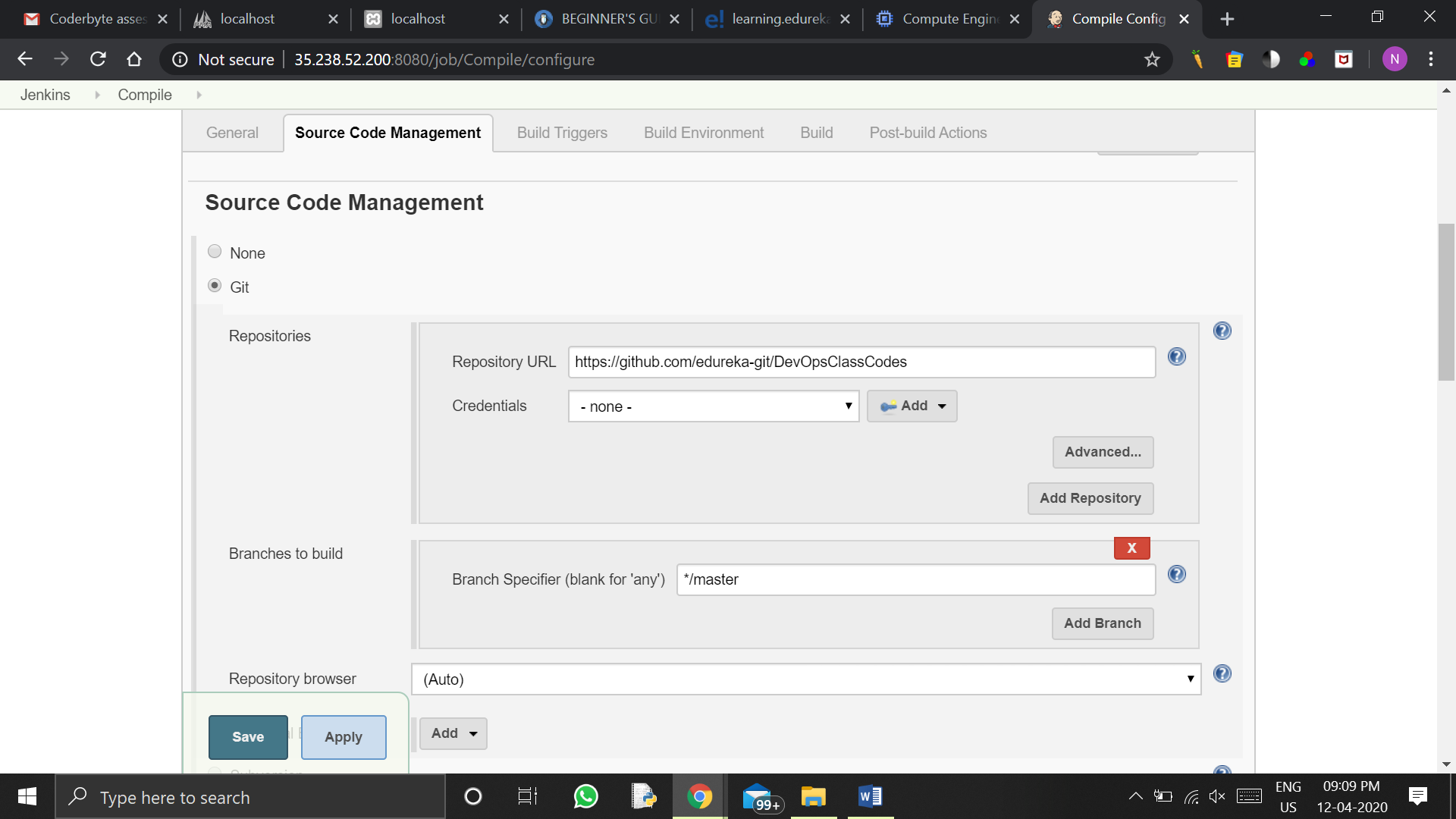
After successful completion of Jenkins setup wizard, You will be redirected to Jenkins dashboard..

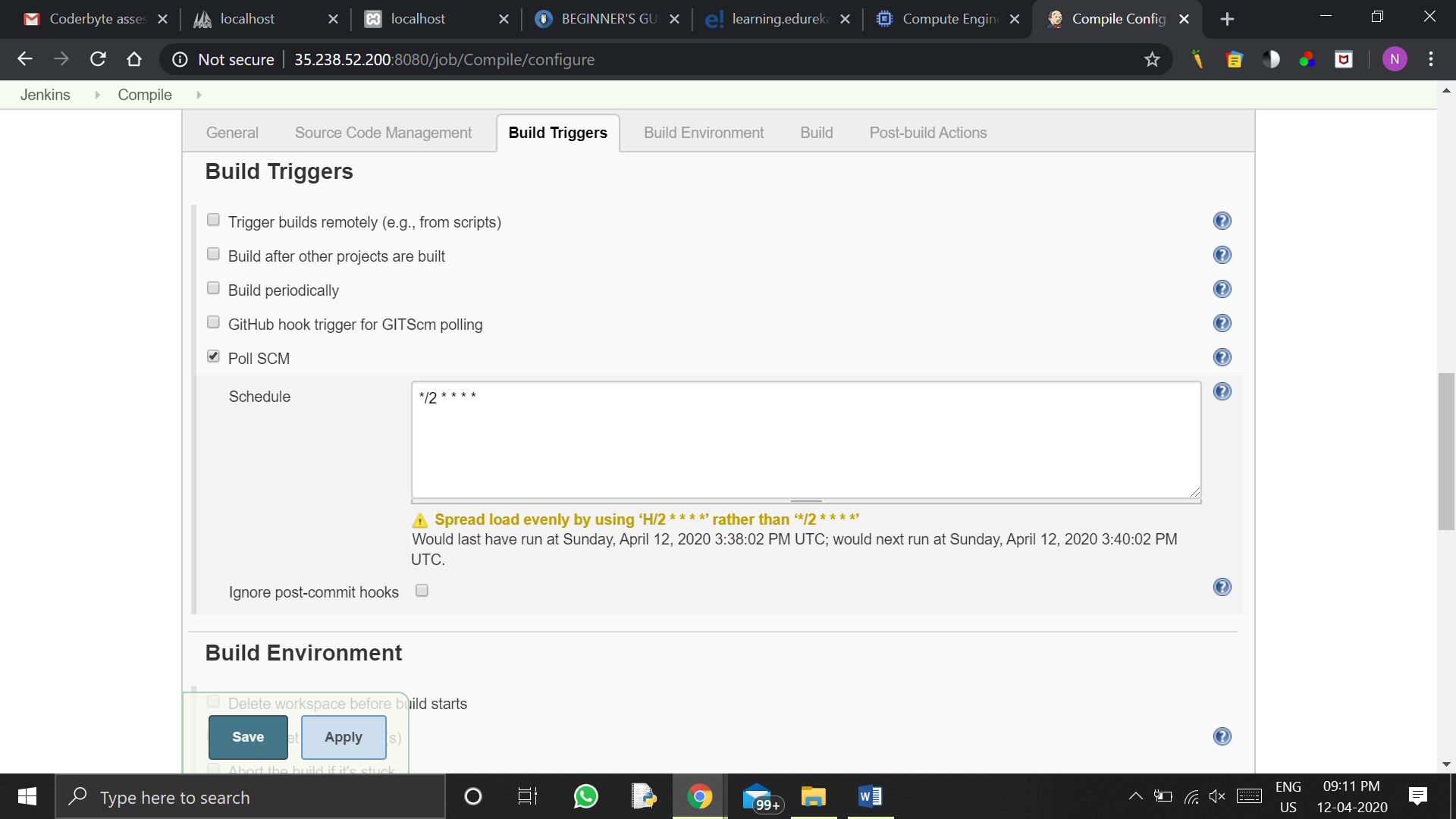
[](https://tecadmin.net/wp-content/uploads/2016/05/jenkings-install-4.png)

Now we will create the first job which is to compile the application

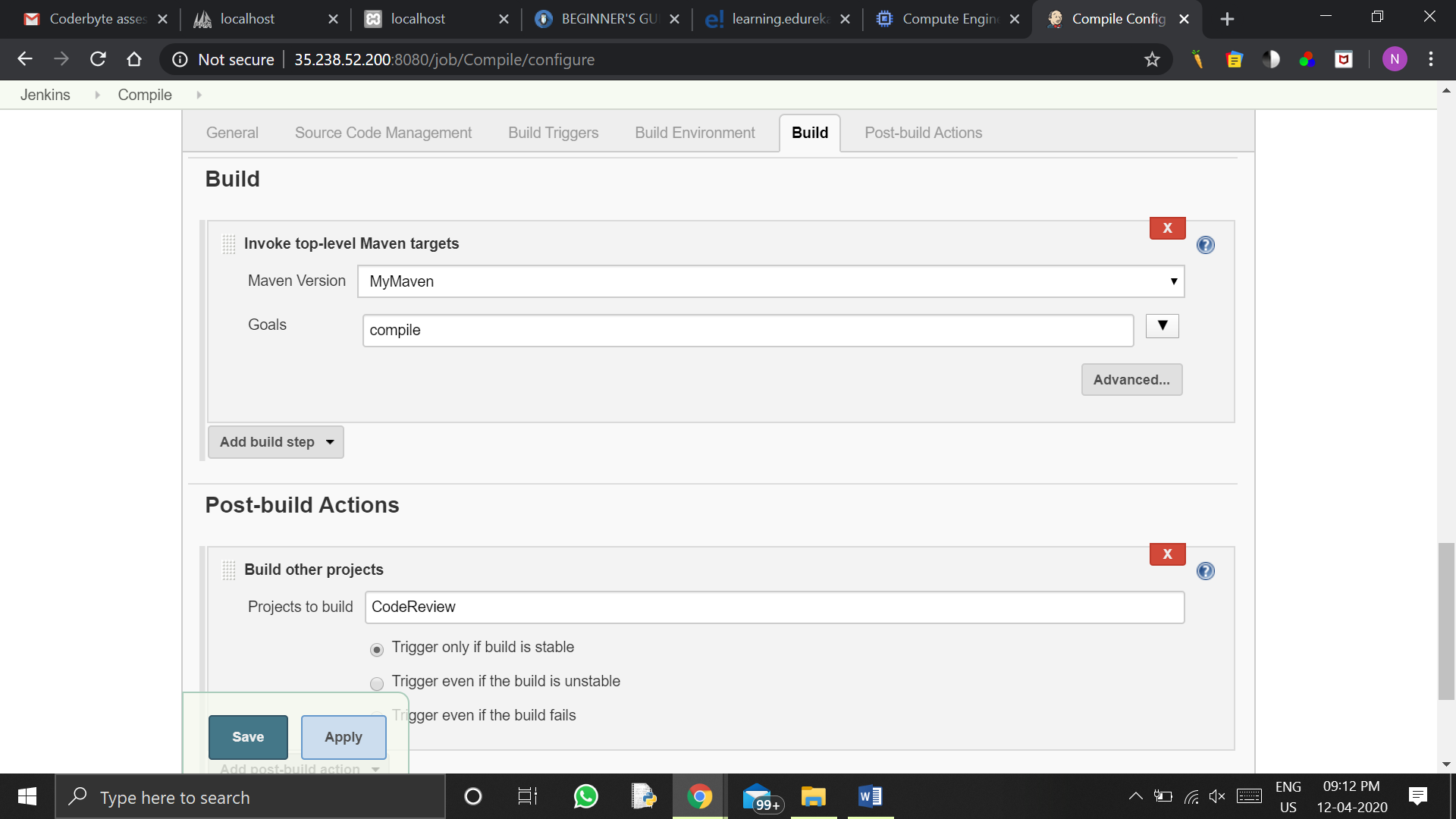


In source code management we will select the git and paste our git repository URL there.

In build triggers we are selecting the poll SCM and giving the schedule as triggered after every 2 min.



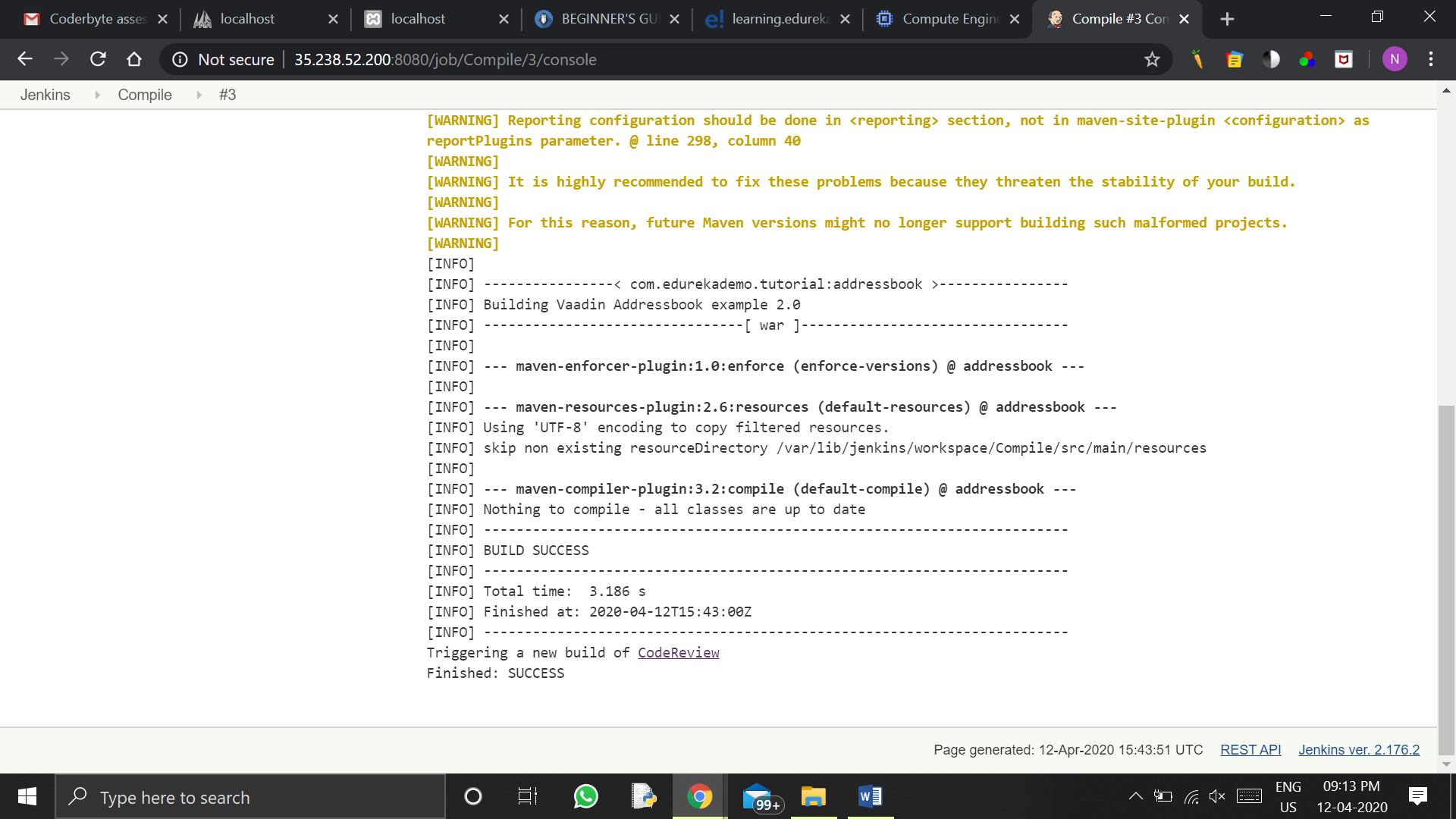
We can skip build environment for now and will give the maven version we created and the goals as compile



And then we save it

Now we will click on build now to run the build and check whether it is building success or failed

We find our compile to be build successfully.



Then we go to the Jenkins console and build new item that will be code review.

Again in source code management we will give the same URL of the git, and then in build we will give MyMaven as the maven version and ‘pmd:pmd’ as the goal.

In post build actions we will provide PMD results as ‘target/pmd.xml’. and save the configuration,

Again we will build it and find it a success build.

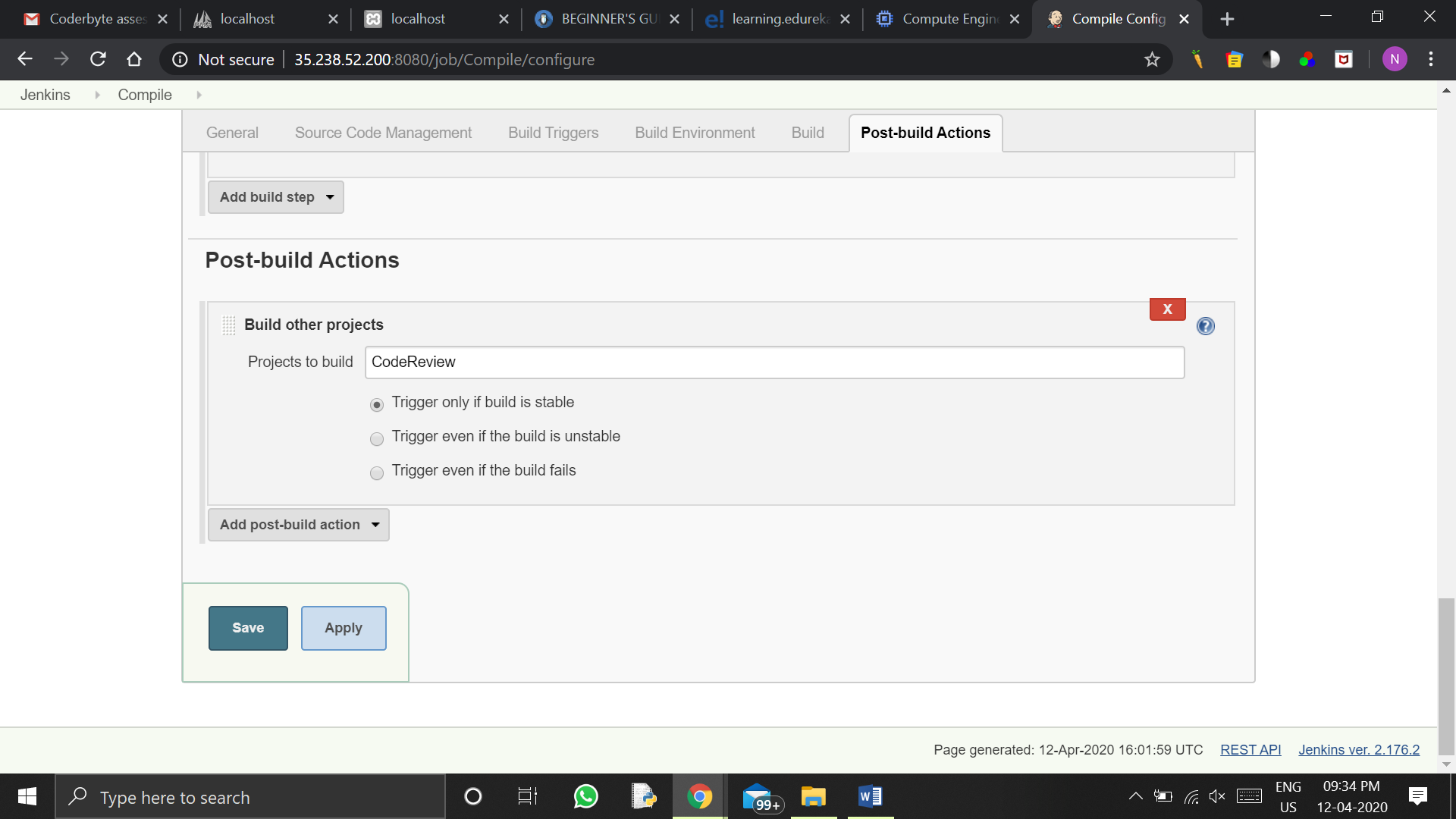
Now comes our third new job which is to test the code. We will give the same git repository in the source code management, and then in the build we give MyMaven and test as maven version and the goals respectively. In post build actions we will select publish JUnit test result report and provide ‘target/surefire-reports/\*.xml’ as test reports XMLs. Save the configuration and build it.

If it is the success then create the new job that is metrics check. Here we do the same as previous in the source code management and then in the build in goals we will give ‘cobertura:cobertura -Dcobertura.report.format=xml’ in the goals in build section.In post build actions we will select publish cobertura coverage report and give ‘target/site/cobertura/coverage.xml’ in cobertura XML report pattern.

And then save and build. After the success build we create our last job which is package.

In this we have to give the git URL in the source code management and will give the MyMaven and package in the build section and then save and build the job.

Then we have to create a triggering chain between these jobs go to the first job which is compile click on the configure and in the post build actions add the build other projects option and give code review to projects to build and click on the trigger only is build option.

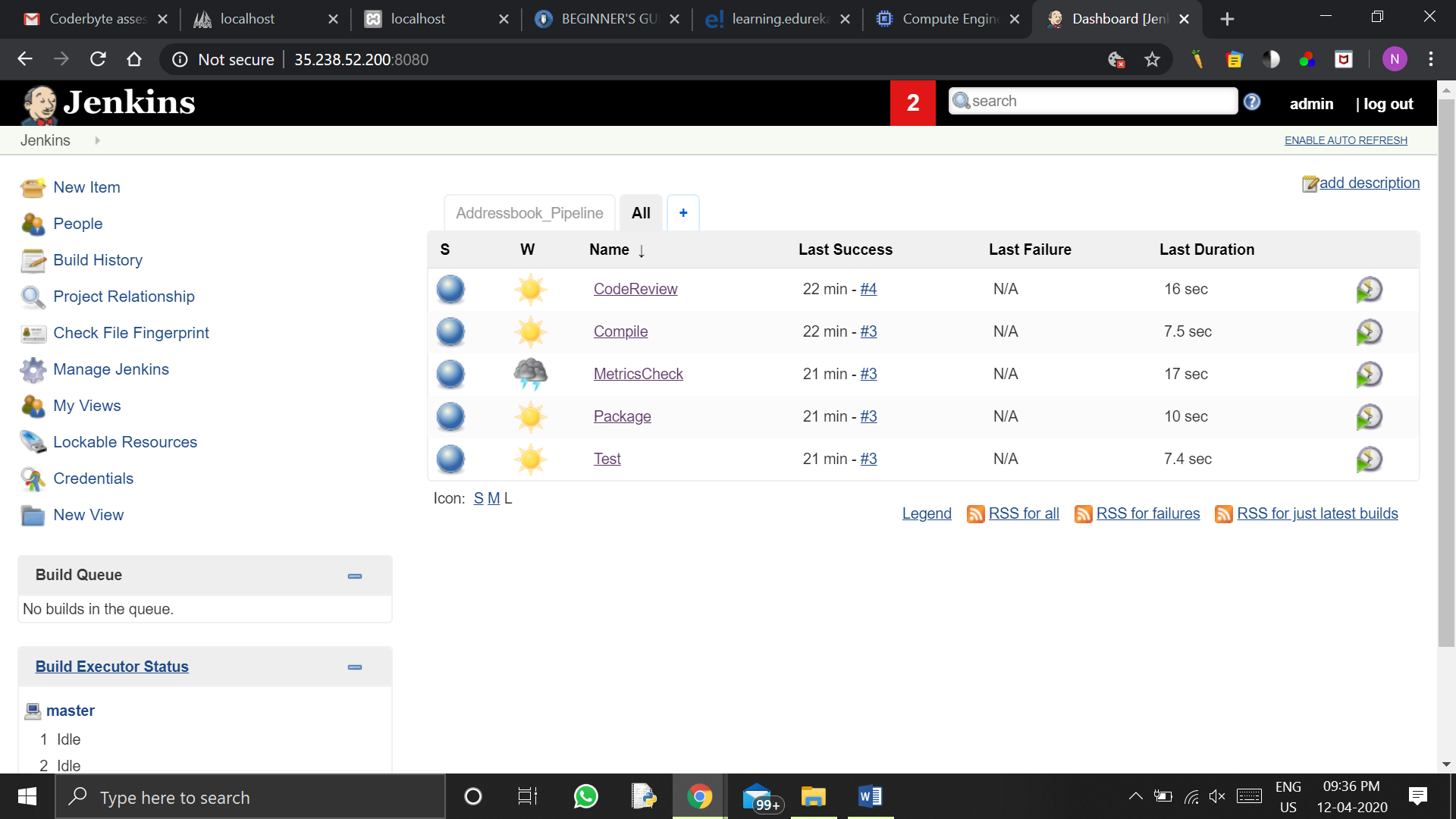


Do this for code review job and give test in it.

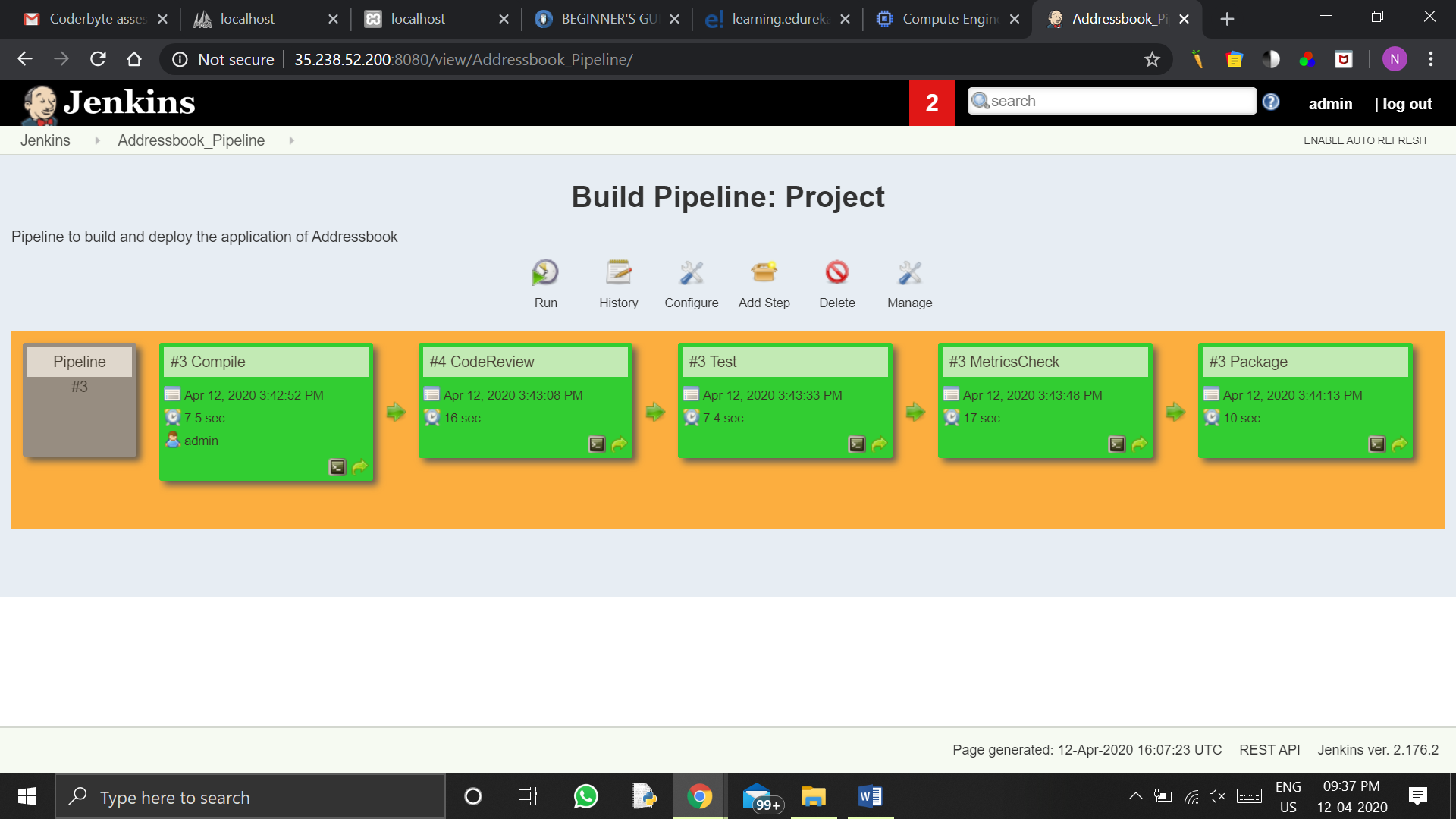
For test provide the metrics check job.

For metrics check provide the package job.

And now comes to the Jenkins console there we have to create a pipeline click on the ‘+’ sign above the jobs and create the new pipeline named ‘Addressbook\_Pipeline’.



Go to the pipeline and build it.



Hence our java file is now being packaged by continuous integration using Jenkins.