Automated CI Project for Maven Using Jenkins Server on AWS EC2 instance and Github Web-hook

Author: Shashwat Kumar
Medium Link:
https://shashwat9kumar.medium.com/automated-ci-project-for-maven-using-a-remote-jenkins-server-on-aws-ec2-instance-and-github-509220e191f3

A CI project that automatically triggers a remote Jenkins server when changes are pushed to github for a maven project.
The project triggers a pipeline on a remote Jenkins server to run the clean, compile, test and install stage for the maven project.

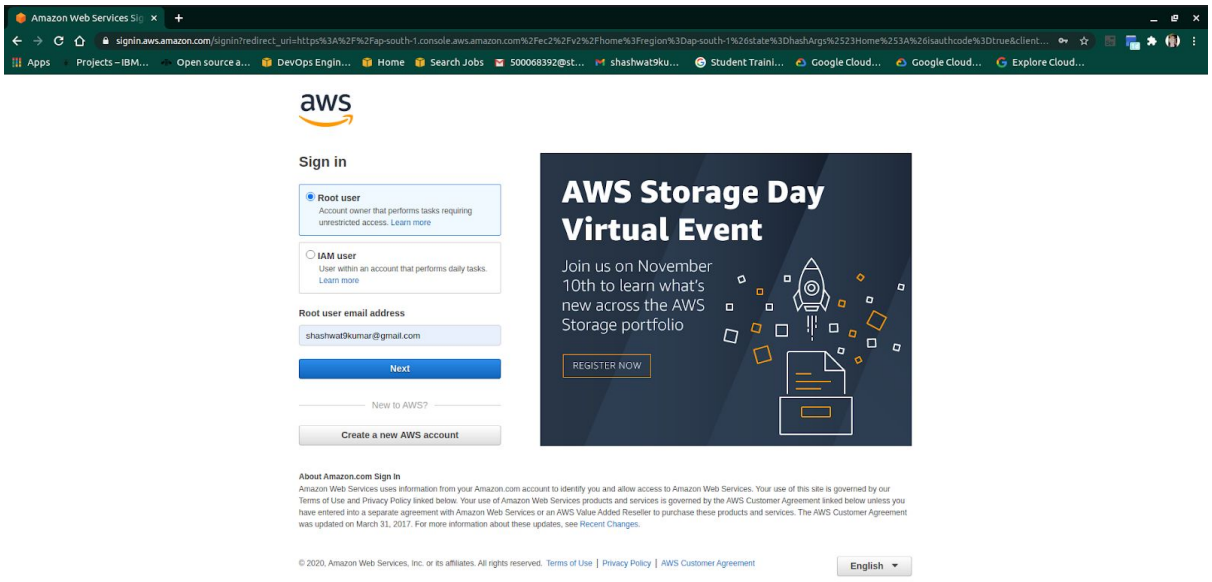
Aim

- Create an Ubuntu EC2 instance on AWS
- Connect to the virtual cloud machine via SSH, and install required dependencies
- Install and run jenkins server on a virtual cloud machine
- Activate github webhooks for automated build triggering
- Creating the automated pipeline for Continuous Integration
- Testing the pipeline via a test push/commit

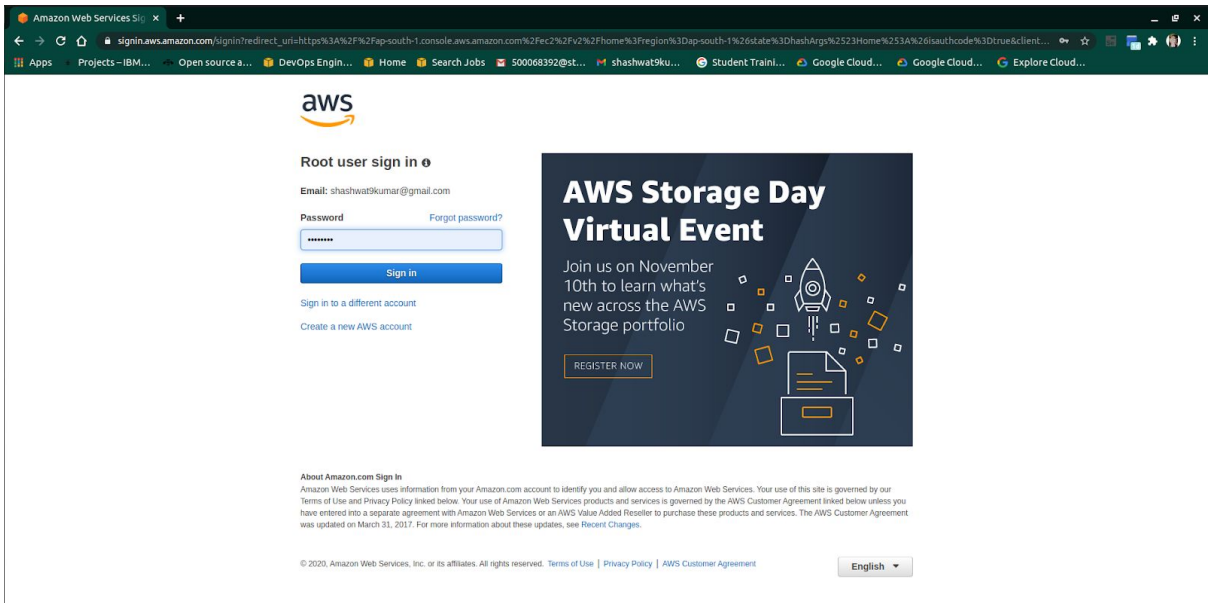
Step 1

Create an Ubuntu EC2 instance on AWS

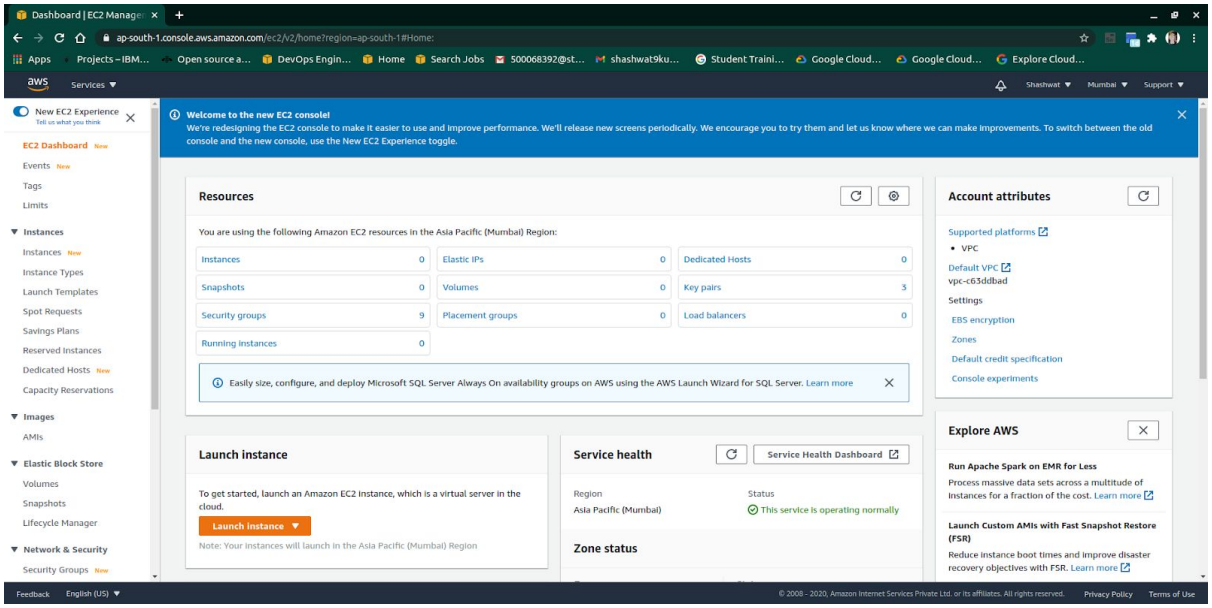
Visit the AWS sign in to console at: https://signin.aws.amazon.com/
And sign in using your root user account
Enter your root user username and click next



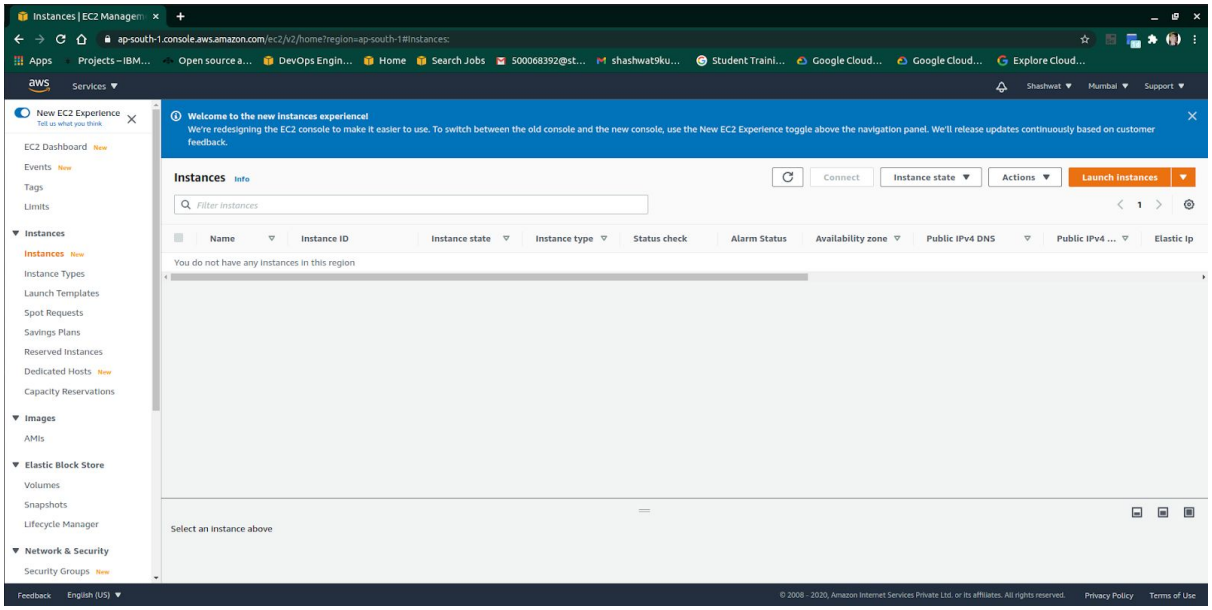
Enter your password and click Sign In



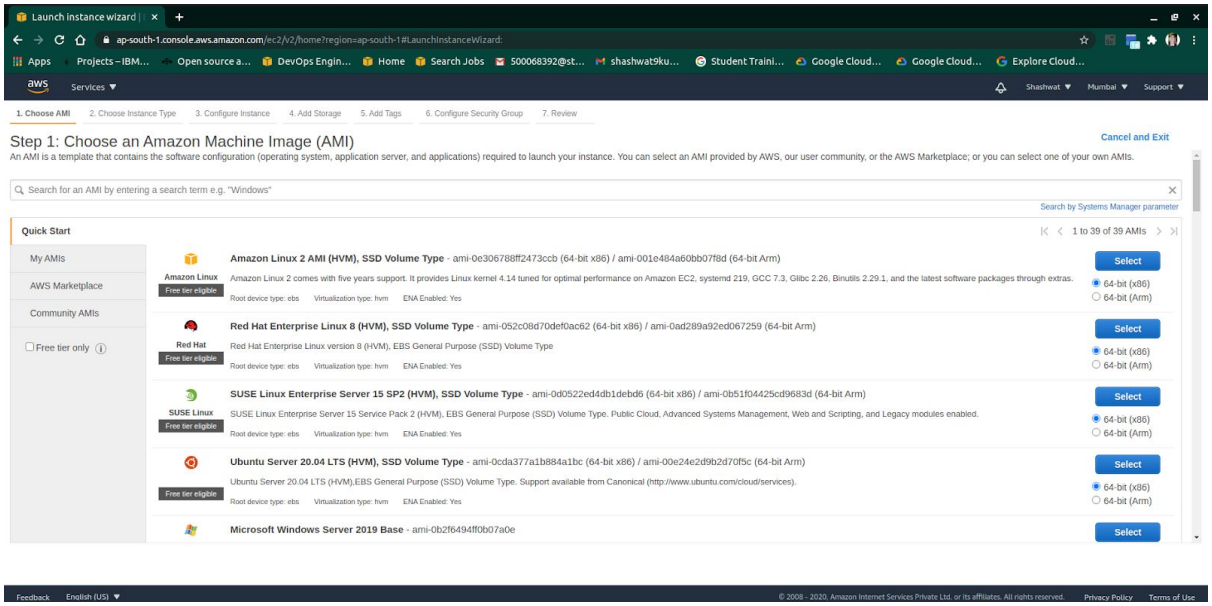
Upon sign in you'll land on the Amazon EC2 dashboard.
On the resources section, click on instances



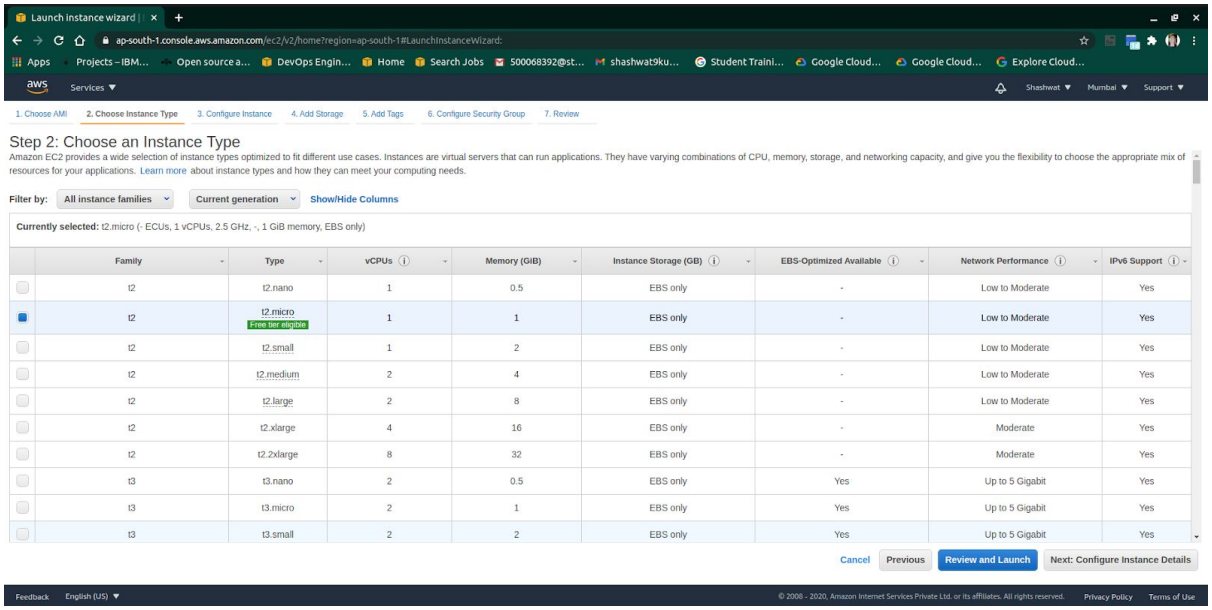
You'll be directed to a page similar to this.
To create a new instance, click on the Launch Instance tab on the top right



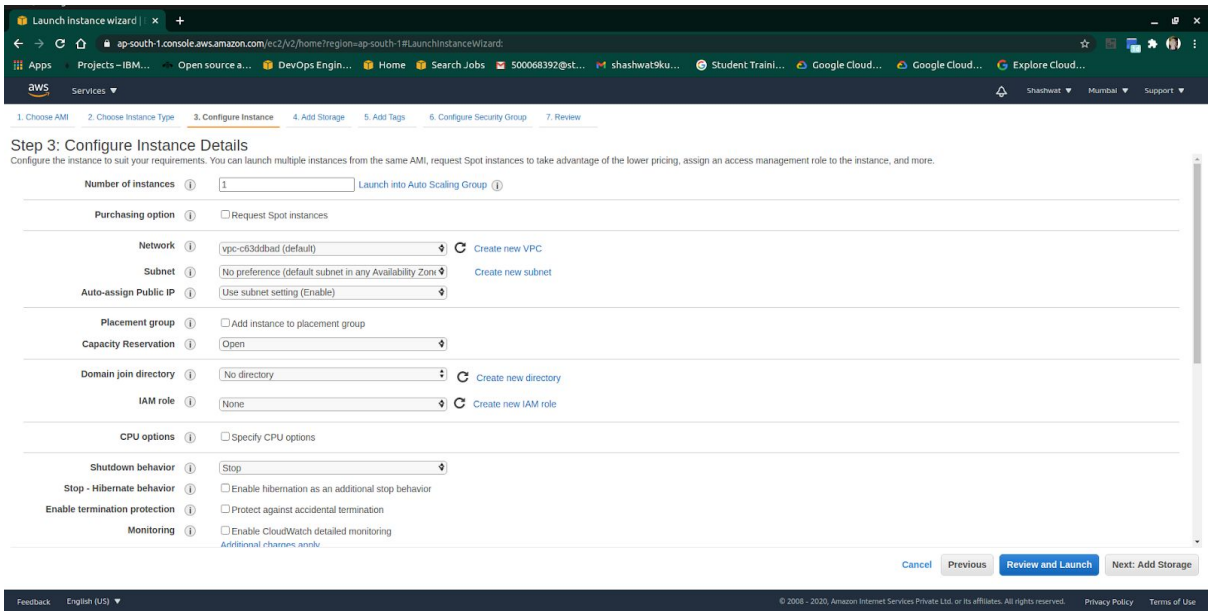
You'll now scroll down to find the Ubuntu Server 20.04 LTS and click on select.



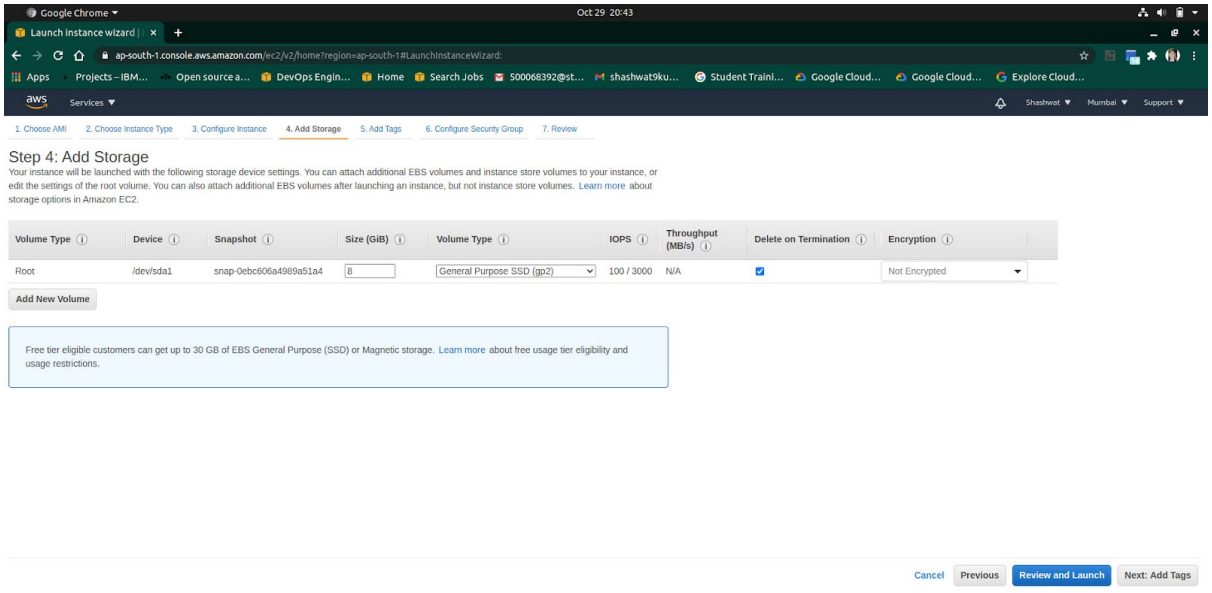
By default, you'll have a t2 micro machine selected. No need to change it.
Click the Next: Configure Instance Details tab on the bottom right



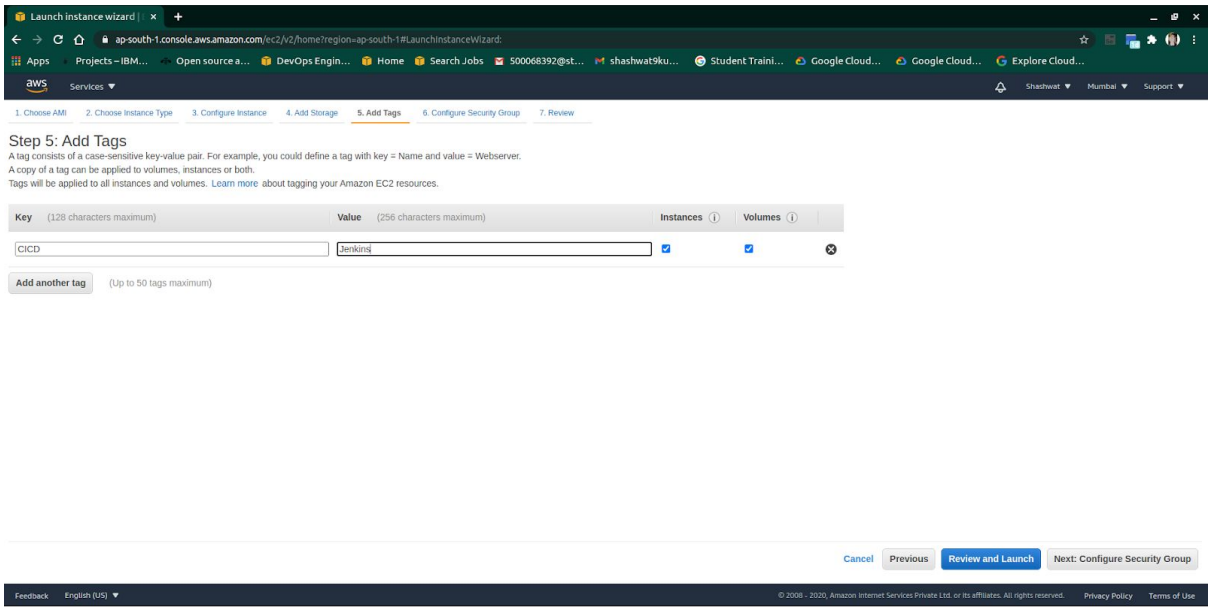
Leave this page as default.(Though you may tweak around with some settings if you wish)
Click on the Next: Add Storage tab on the bottom right



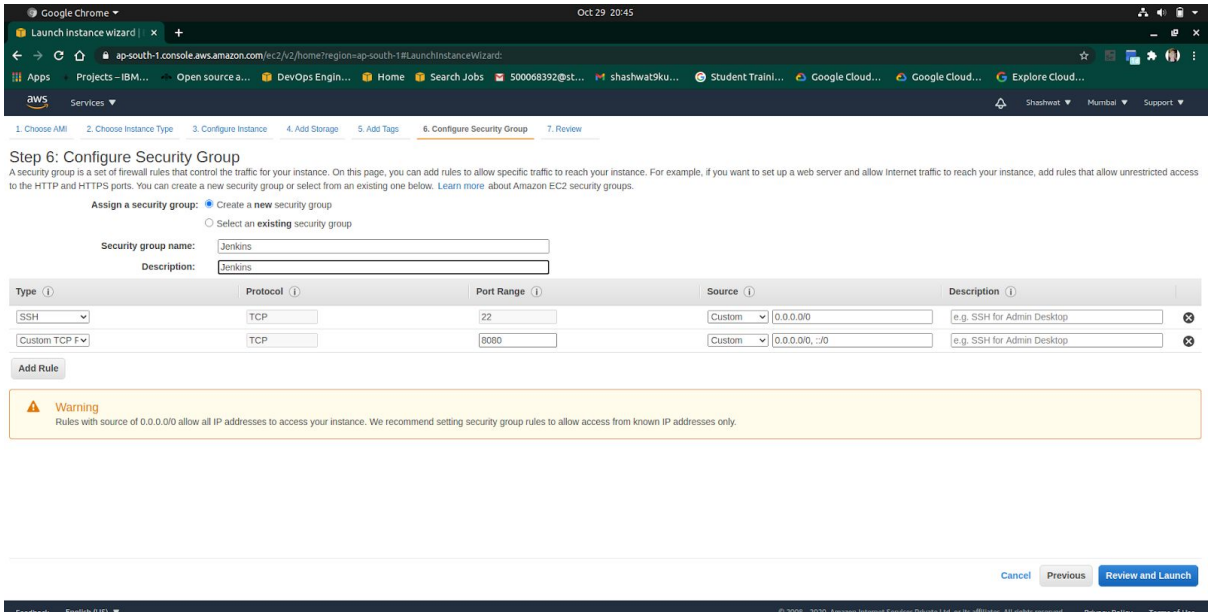
Because we are just going to use a jenkins server so we need not add any extra storage. You can leave everything as default again.
Click on Add Tags tab on the bottom right



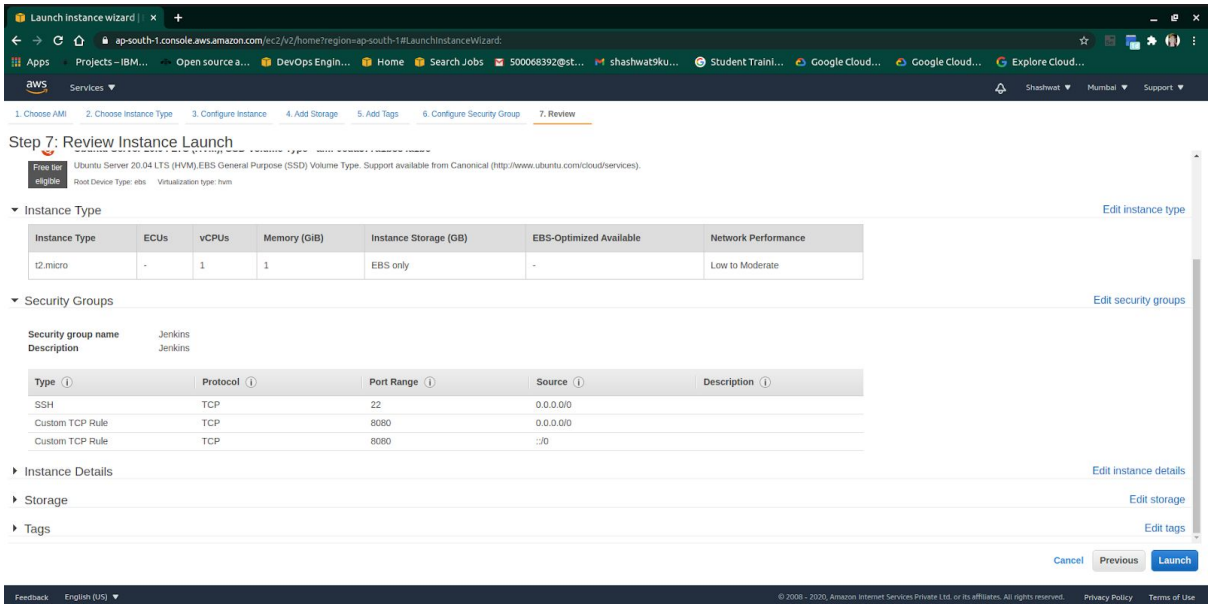
You can add a tag as a key-value pair.
Once added, click on the Next: Configure Security Groups tab



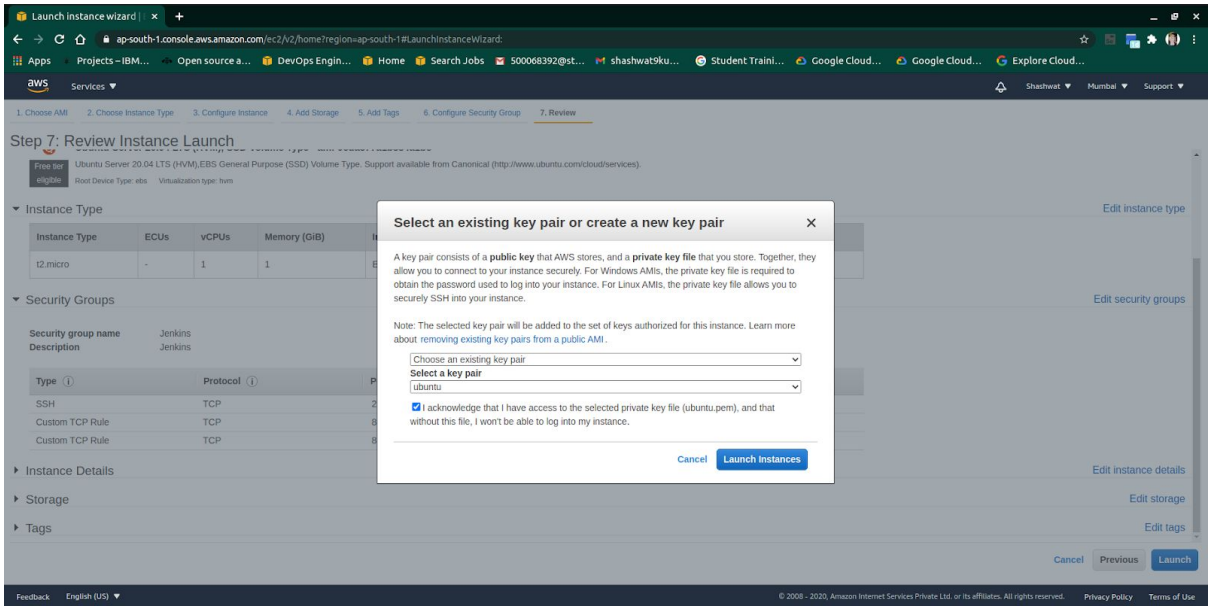
You can give your security group a name and add a description to it as well.
Also, importantly, Jenkins runs on port 8080, so we have to open that port.
Click on the Add Rule tab
For type: select the custom TCP option
For Protocol, select TCP
For Port Range, type 8080
Now click on Review and Launch tab on the bottom right



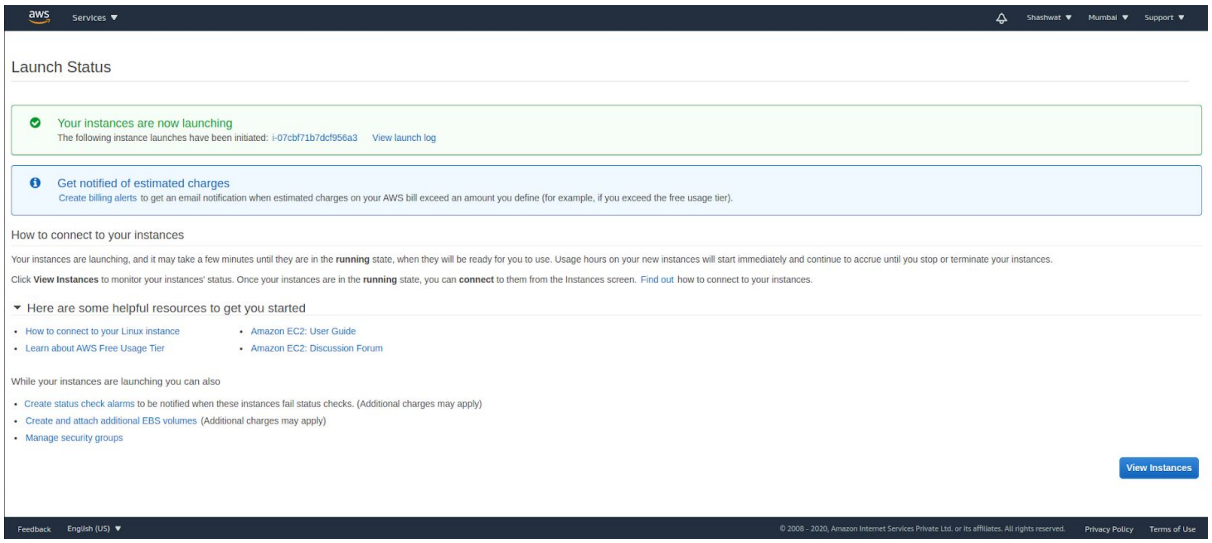
Once satisfied with all the settings, click on the Launch tab on the bottom right



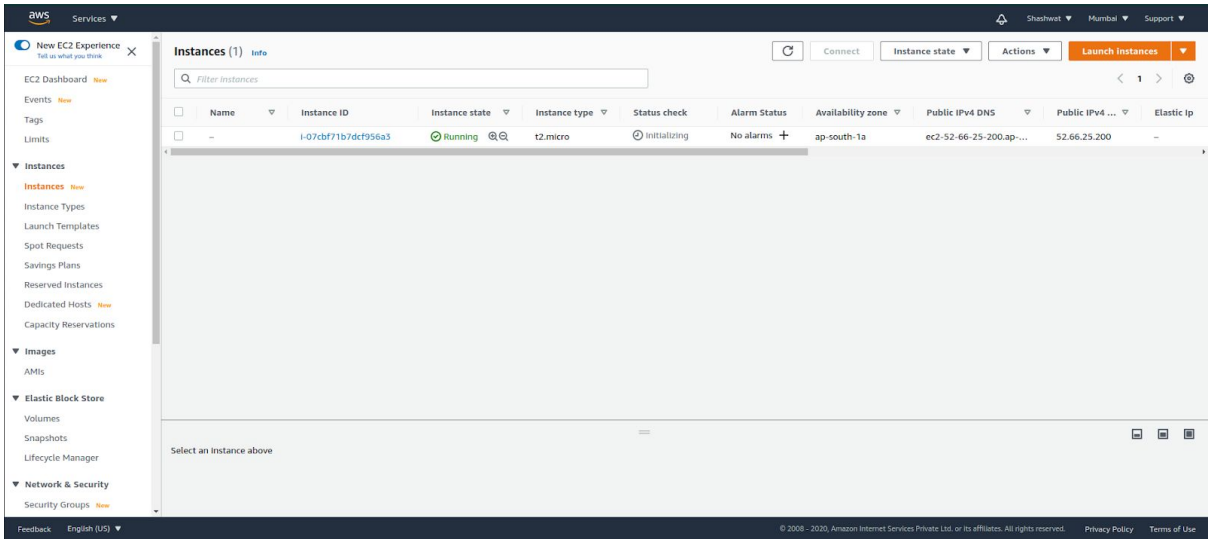
On the dialog box that appears, you can select an existing key pair (like done in this example), or create a new one. In either case, make sure the key pair file is downloaded and remember the download location. Click Launch Instance



The instance will take some time to launch.
You can click the View Instance tab to see the instance



If you see a green tick next to your instance state with its state as “Running”, your instance is up and running.
Congratulations, you've now launched an EC2 Ubuntu instance on AWS.

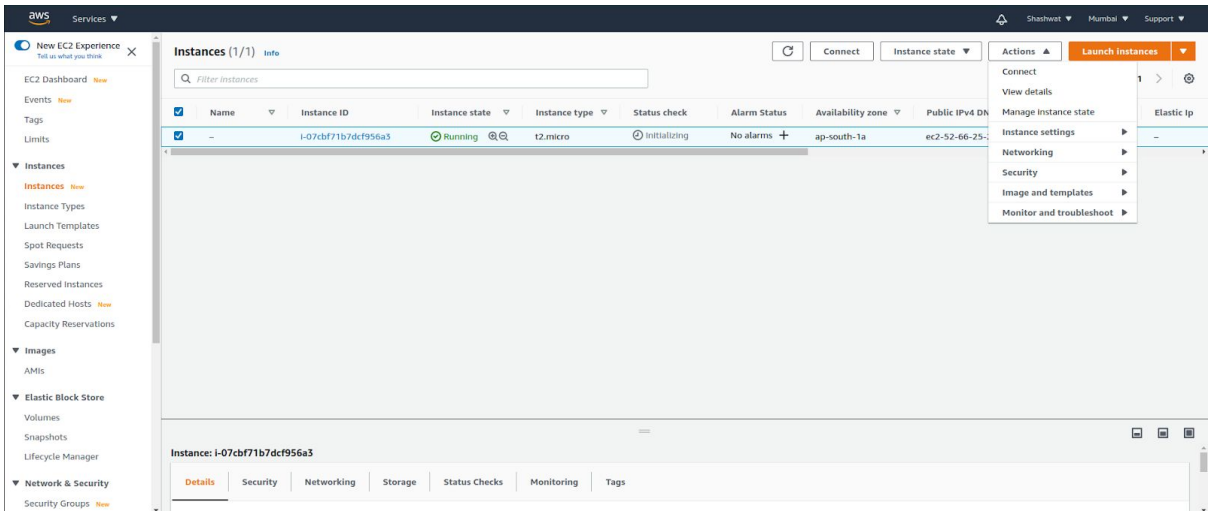


Step 2

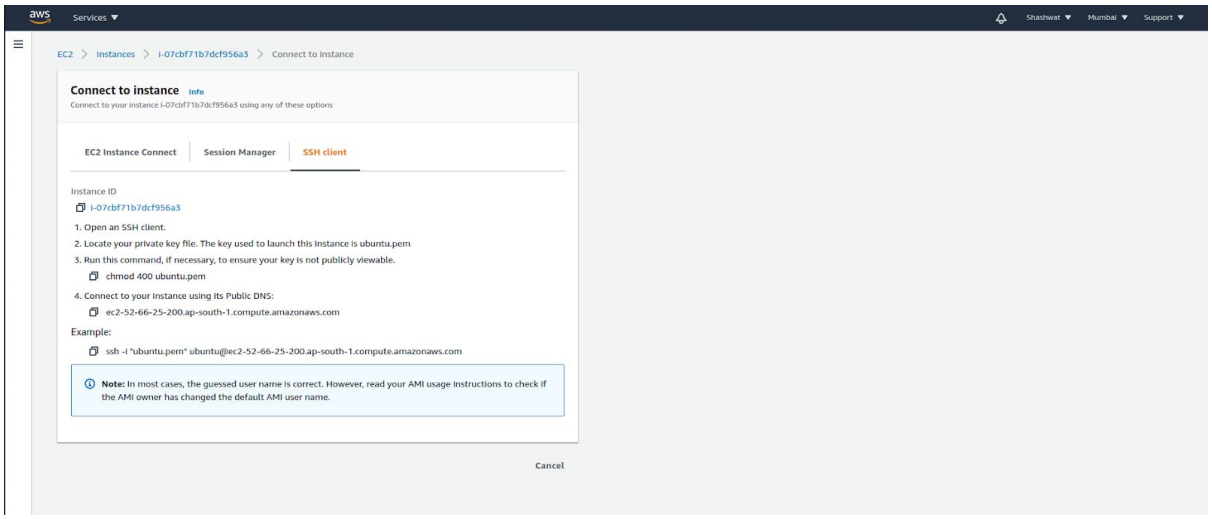
Connect to the virtual cloud machine via SSH, and install required dependencies

We’ll now connect to the virtual machine using the SSH manager. It would be appropriate to use a Linux machine or Machine on a virtual box. (If not, you may also install a SSH manager for Windows)
Select the running instance by checking the checkbox next to it.

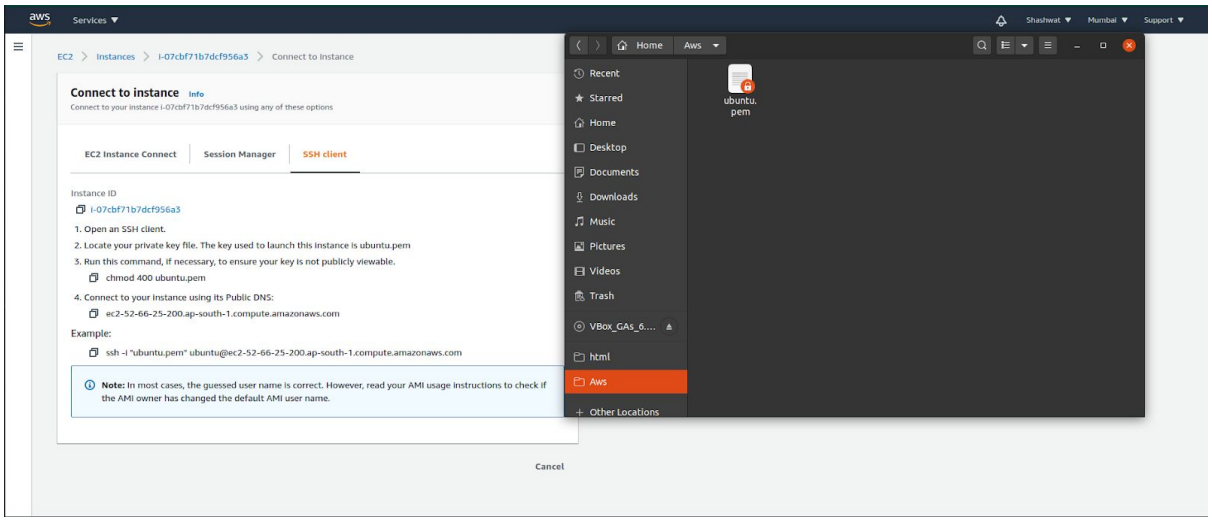
Click on the Actions tab on the top right, and select Connect



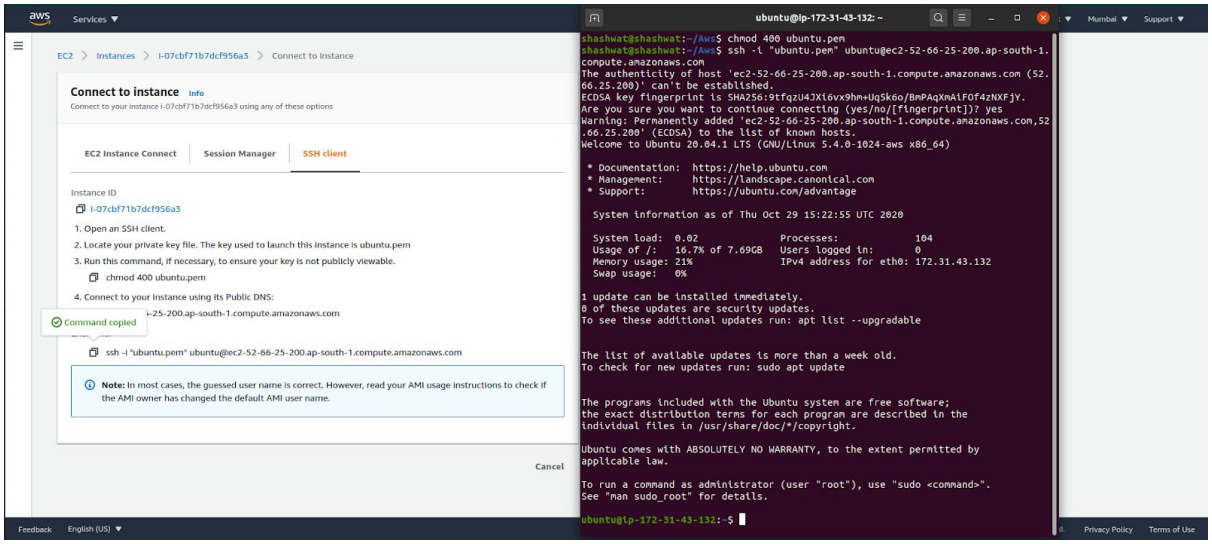
On the Connect to Instance page, select the SSH Client Tab



Open the location where you downloaded/saved the private key. Open a terminal at that file location.

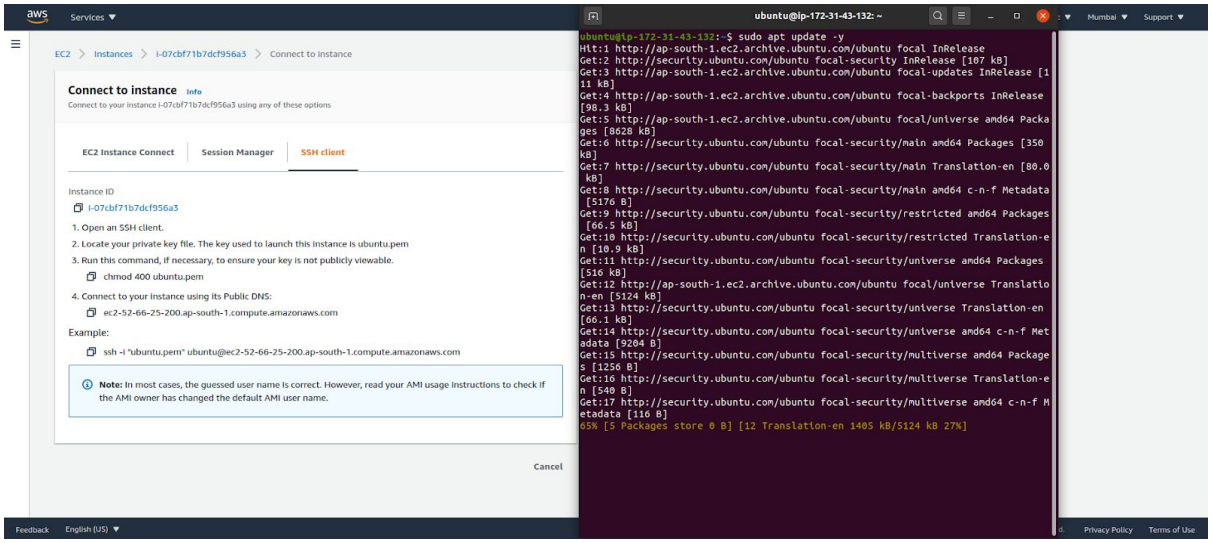


Copy the command in the point 3 and run it in the terminal:
The command should be something like “chmod 400 <filename.pem>”
This will make the private key file accessible only to admin.
Now, copy the command under the example and run it in the terminal. Enter “Y” in the terminal when prompted.
This will help you connect remotely to the created instance.
Once done, you’ll be able to see that your terminal shows ubuntu@<ip-address>:~$.
This confirms that you've successfully connected to the instance.

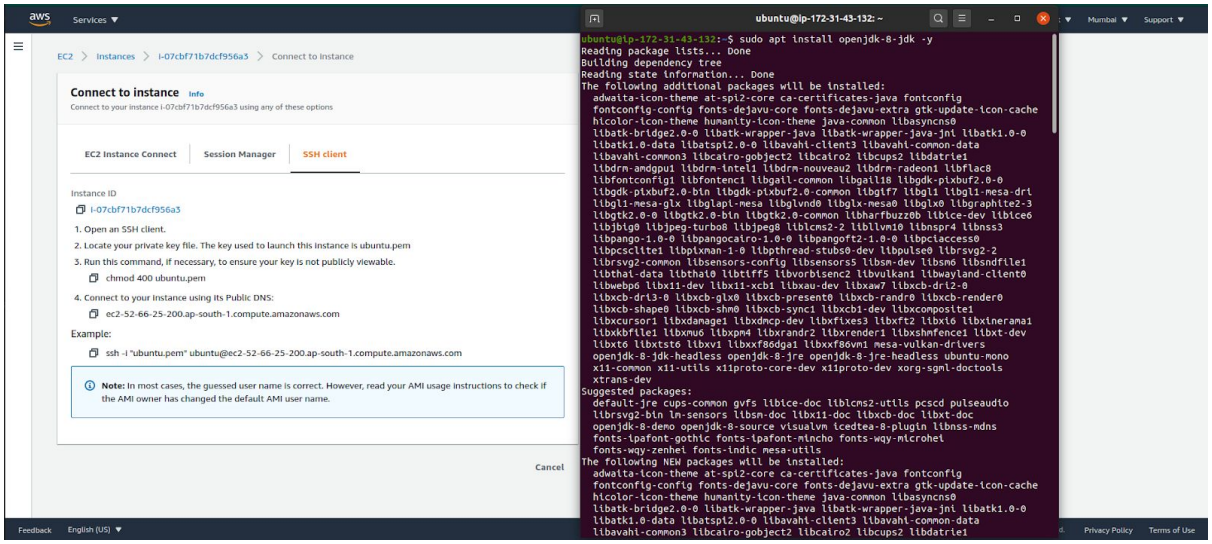


Now that you have connected to the instance, we will install the required dependencies for Jenkins.
Run the following commands in the terminal:

- sudo apt update -y ## to update the existing dependencies, if any

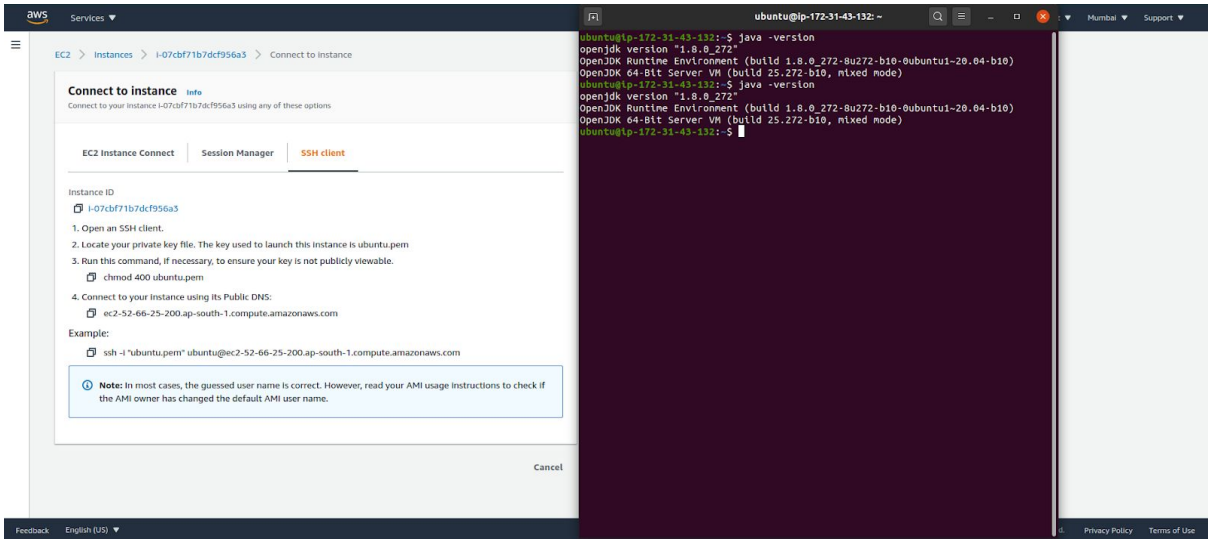


- sudo apt install openjdk-8-jdk -y ## to install JDK 8. Jenkins uses Java to run



Confirm the java installation using the following commands:

- java -version
- javac -version



We'll now set the JAVA_HOME and export it into the system PATH variable.
Run the following commands:

- export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64 ##your java installation path may be different. Check before you set the JAVA_HOME variable

Now, we'll install maven
Run:

- sudo apt install maven

Enter "Y" when prompted to.

```
ubuntu@ip-172-31-43-132:~$ sudo apt install maven
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libaopalliance-java libapache-pom-java libatinject-jsr330-api-java
  libcdi-api-java libcommons-cli-java libcommons-io-java libcommons-lang3-java
  libcommons-parent-java libgeronimo-annotation-1.3-spec-java
  libgeronimo-interceptor-3.0-spec-java libguava-java libguice-java
  libhawtjni-runtime-java libjansi-java libjansi-native-java libjsr305-java
  libmaven-parent-java libmaven-resolver-java libmaven-shared-utils-java
  libmaven3-core-java libplexus-cipher-java libplexus-classworlds-java
  libplexus-component-annotations-java libplexus-interpolation-java
  libplexus-sec-dispatcher-java libplexus-utils2-java libsisu-inject-java
  libsisu-plexus-java libslf4j-java libwagon-file-java
  libwagon-http-shaded-java libwagon-provider-api-java
Suggested packages:
  libaopalliance-java-doc libatinject-jsr330-api-java-doc libservlet3.1-java
  libcommons-io-java-doc libcommons-lang3-java-doc libasm-java libcglib-java
  libjsr305-java-doc libmaven-shared-utils-java-doc liblogback-java
  libplexus-cipher-java-doc libplexus-classworlds-java-doc
  libplexus-sec-dispatcher-java-doc libplexus-utils2-java-doc junit4 testng
  libcommons-logging-java liblog4j1.2-java
The following NEW packages will be installed:
  libaopalliance-java libapache-pom-java libatinject-jsr330-api-java
  libcdi-api-java libcommons-cli-java libcommons-io-java libcommons-lang3-java
  libcommons-parent-java libgeronimo-annotation-1.3-spec-java
  libgeronimo-interceptor-3.0-spec-java libguava-java libguice-java
  libhawtjni-runtime-java libjansi-java libjansi-native-java libjsr305-java
  libmaven-parent-java libmaven-resolver-java libmaven-shared-utils-java
  libmaven3-core-java libplexus-cipher-java libplexus-classworlds-java
  libplexus-component-annotations-java libplexus-interpolation-java
  libplexus-sec-dispatcher-java libplexus-utils2-java libsisu-inject-java
  libsisu-plexus-java libslf4j-java libwagon-file-java
  libwagon-http-shaded-java libwagon-provider-api-java maven
0 upgraded, 33 newly installed, 0 to remove and 82 not upgraded.
Need to get 9209 kB of archives.
After this operation, 12.1 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 libap
ache-pom-java all 18-1 [4720 B]
Get:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal/universe amd64 libat
inject-jsr330-api-java all 1.0+ds1-5 [5348 B]
```

We set the MAVEN_HOME variable similar to the JAVA_HOME variable.
Run the following commands:

- MAVEN_HOME=/usr/share/maven ## your maven installation may exist at a different location. Check before you set this variable
- PATH=$PATH:$MAVEN_HOME/bin
- export MAVEN_HOME
- export PATH

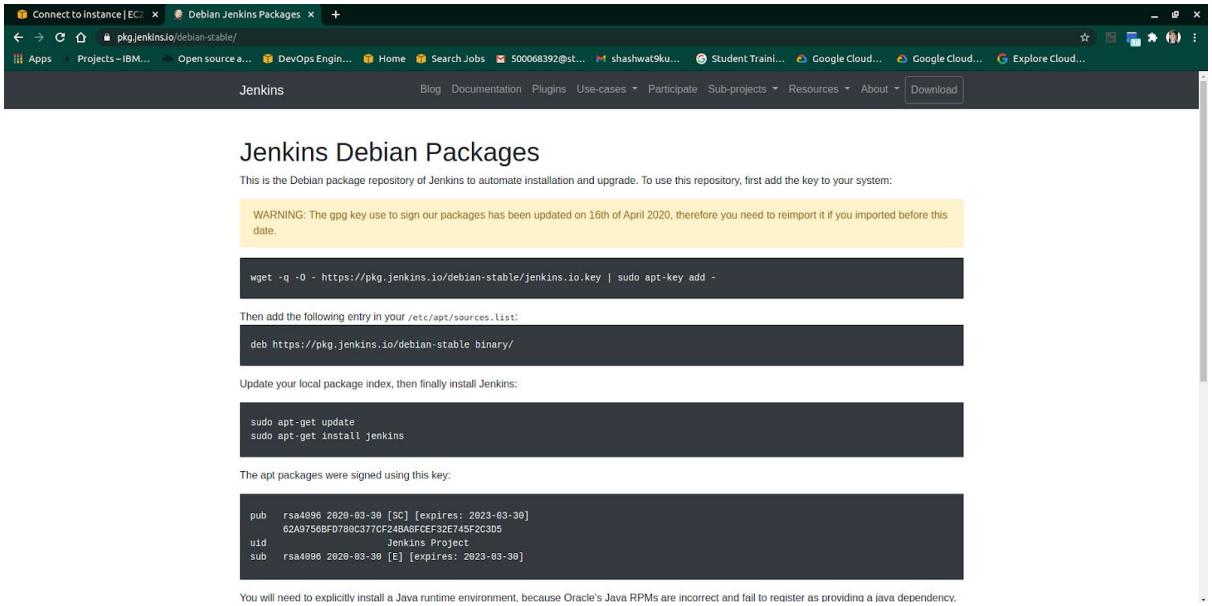
```
ubuntu@ip-172-31-43-132:~$ echo $MAVEN_HOME

ubuntu@ip-172-31-43-132:~$ echo $M2_HOME

ubuntu@ip-172-31-43-132:~$ MAVEN_HOME=/usr/share/maven
ubuntu@ip-172-31-43-132:~$ PATH=$PATH:$MAVEN_HOME/bin
ubuntu@ip-172-31-43-132:~$ export MAVEN_HOME
ubuntu@ip-172-31-43-132:~$ export PATH
ubuntu@ip-172-31-43-132:~$
```

Congratulations. You've set all the dependencies now for running JEnkins.

Step 3

Install and run jenkins server on a virtual cloud machine

For installation of jenkins visit the website: https://pkg.jenkins.io/debian-stable/
Here you'll find the list of all the commands to install Jenkins.



While still connected to the virtual instance, run the first command in the terminal.
The command: "wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -"

```
ubuntu@ip-172-31-43-132:~$ wget -q -O - https://pkg.jenkins.io/debian-stable/jen
kins.io.key | sudo apt-key add -
OK
ubuntu@ip-172-31-43-132:~$ sudo nano /etc/apt/sources.list
ubuntu@ip-172-31-43-132:~$ sudo apt-get update -y
Ign:1 https://pkg.jenkins.io/debian-stable binary/ InRelease
Get:2 https://pkg.jenkins.io/debian-stable binary/ Release [2044 B]
Get:3 https://pkg.jenkins.io/debian-stable binary/ Release.gpg [833 B]
Get:4 https://pkg.jenkins.io/debian-stable binary/ Packages [18.5 kB]
Hit:5 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:6 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal InRelease
Hit:7 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:8 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease
Fetched 21.4 kB in 1s (34.8 kB/s)
Reading package lists... Done
ubuntu@ip-172-31-43-132:~$ ls
ubuntu@ip-172-31-43-132:~$ sudo apt-get install jenkins
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  daemon net-tools
The following NEW packages will be installed:
  daemon jenkins net-tools
0 upgraded, 3 newly installed, 0 to remove and 82 not upgraded.
Need to get 67.1 MB of archives.
After this operation, 68.6 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Now run: “sudo nano /etc/apt/sources.list”
This will open an editor to edit the sources.list file
Add the following line to the end of the file:
“deb https://pkg.jenkins.io/debian-stable binary/”
Once done click CTRL+O and Enter. Click CTRL+X to exit the editor.

```
  GNU nano 4.8                  /etc/apt/sources.list
## review or updates from the Ubuntu security team.
deb http://ap-south-1.ec2.archive.ubuntu.com/ubuntu/ focal universe
# deb-src http://ap-south-1.ec2.archive.ubuntu.com/ubuntu/ focal universe
deb http://ap-south-1.ec2.archive.ubuntu.com/ubuntu/ focal-updates universe
# deb-src http://ap-south-1.ec2.archive.ubuntu.com/ubuntu/ focal-updates univer>

## N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu
## team, and may not be under a free licence. Please satisfy yourself as to
## your rights to use the software. Also, please note that software in
## multiverse WILL NOT receive any review or updates from the Ubuntu
## security team.
deb http://ap-south-1.ec2.archive.ubuntu.com/ubuntu/ focal multiverse
# deb-src http://ap-south-1.ec2.archive.ubuntu.com/ubuntu/ focal multiverse
deb http://ap-south-1.ec2.archive.ubuntu.com/ubuntu/ focal-updates multiverse
# deb-src http://ap-south-1.ec2.archive.ubuntu.com/ubuntu/ focal-updates multiv>

## N.B. software from this repository may not have been tested as
## extensively as that contained in the main release, although it includes
## newer versions of some applications which may provide useful features.
## Also, please note that software in backports WILL NOT receive any review
## or updates from the Ubuntu security team.
deb http://ap-south-1.ec2.archive.ubuntu.com/ubuntu/ focal-backports main restr>
# deb-src http://ap-south-1.ec2.archive.ubuntu.com/ubuntu/ focal-backports main>

## Uncomment the following two lines to add software from Canonical's
## 'partner' repository.
## This software is not part of Ubuntu, but is offered by Canonical and the
## respective vendors as a service to Ubuntu users.
# deb http://archive.canonical.com/ubuntu focal partner
# deb-src http://archive.canonical.com/ubuntu focal partner

deb http://security.ubuntu.com/ubuntu focal-security main restricted
# deb-src http://security.ubuntu.com/ubuntu focal-security main restricted
deb http://security.ubuntu.com/ubuntu focal-security universe
# deb-src http://security.ubuntu.com/ubuntu focal-security universe
deb http://security.ubuntu.com/ubuntu focal-security multiverse
# deb-src http://security.ubuntu.com/ubuntu focal-security multiverse
deb https://pkg.jenkins.io/debian-stable binary/
```

Run the following command to install jenkins:

- sudo apt-get update
- sudo apt-get install jenkins

Enter “Y” when prompted to.

```
ubuntu@ip-172-31-43-132:~$ wget -q -O - https://pkg.jenkins.io/debian-stable/jen
kins.io.key | sudo apt-key add -
OK
ubuntu@ip-172-31-43-132:~$ sudo nano /etc/apt/sources.list
ubuntu@ip-172-31-43-132:~$ sudo apt-get update -y
Ign:1 https://pkg.jenkins.io/debian-stable binary/ InRelease
Get:2 https://pkg.jenkins.io/debian-stable binary/ Release [2044 B]
Get:3 https://pkg.jenkins.io/debian-stable binary/ Release.gpg [833 B]
Get:4 https://pkg.jenkins.io/debian-stable binary/ Packages [18.5 kB]
Hit:5 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:6 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal InRelease
Hit:7 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:8 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease
Fetched 21.4 kB in 1s (34.8 kB/s)
Reading package lists... Done
ubuntu@ip-172-31-43-132:~$ ls
ubuntu@ip-172-31-43-132:~$ sudo apt-get install jenkins
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  daemon net-tools
The following NEW packages will be installed:
  daemon jenkins net-tools
0 upgraded, 3 newly installed, 0 to remove and 82 not upgraded.
Need to get 67.1 MB of archives.
After this operation, 68.6 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

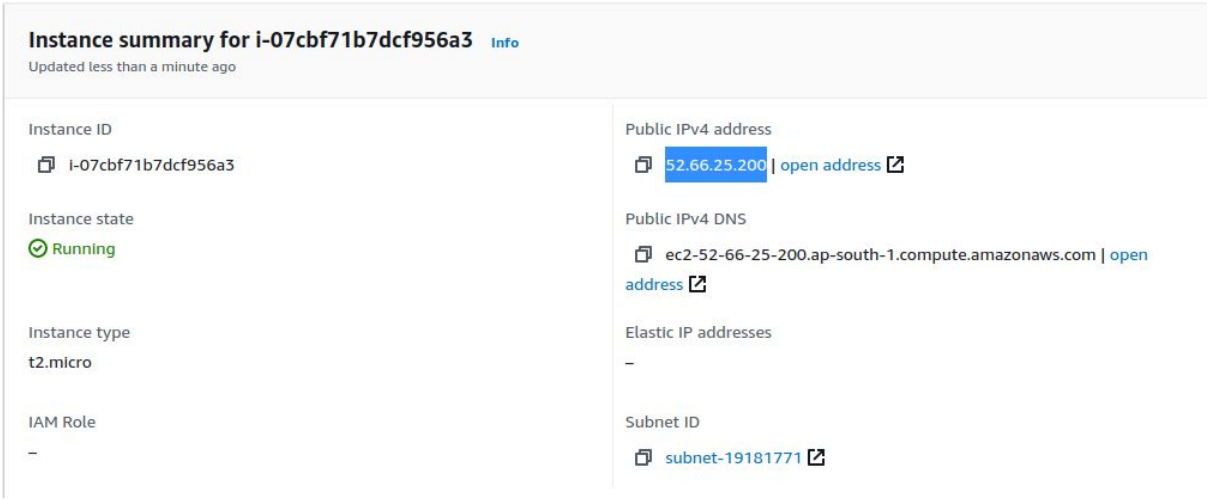
Now that Jenkins is installed. We must run it.
Run the following commands:

- sudo systemctl start jenkins ## to manually start jenkins
- sudo systemctl enable jenkins ## to enable jenkins to start at system restart
- sudo systemctl status jenkins ## to check the status of jenkins- if it is running or not

If you see, an output screen similar to this, then congratulations, JEnkins server is up and running.

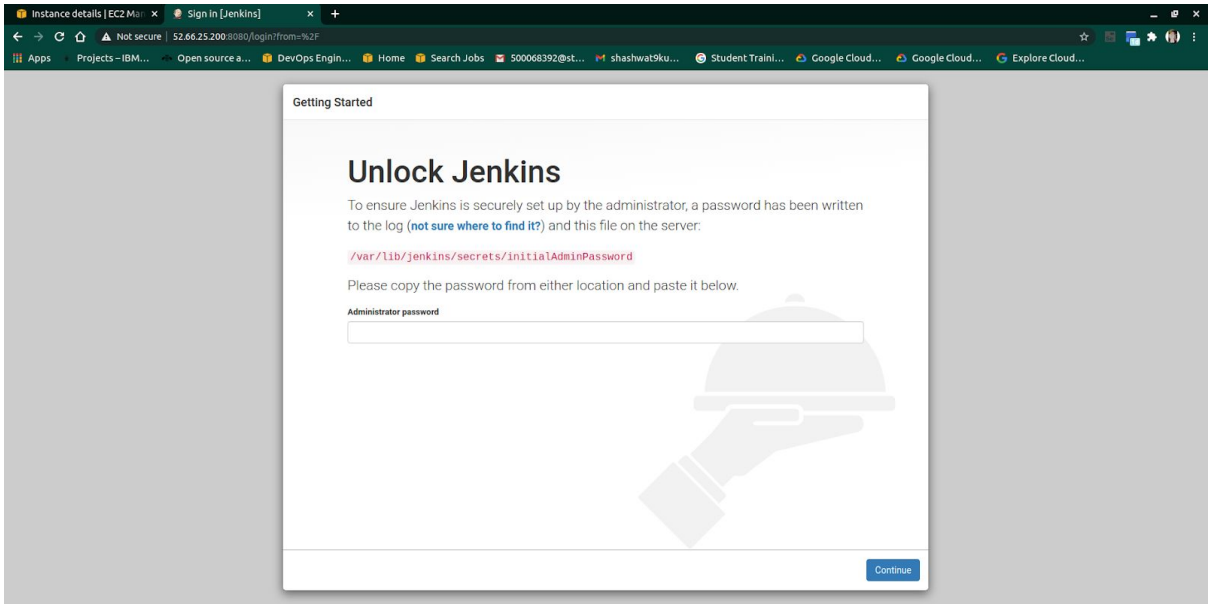
```
ubuntu@ip-172-31-43-132:~$ sudo systemctl start jenkins
ubuntu@ip-172-31-43-132:~$ sudo systemctl enable jenkins
jenkins.service is not a native service, redirecting to systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable jenkins
ubuntu@ip-172-31-43-132:~$ sudo systemctl status jenkins
● jenkins.service - LSB: Start Jenkins at boot time
     Loaded: loaded (/etc/init.d/jenkins; generated)
     Active: active (exited) since Thu 2020-10-29 15:48:34 UTC; 1min 8s ago
       Docs: man:systemd-sysv-generator(8)
      Tasks: 0 (limit: 1164)
     Memory: 0B
     CGroup: /system.slice/jenkins.service

Oct 29 15:48:32 ip-172-31-43-132 systemd[1]: Starting LSB: Start Jenkins at boo>
Oct 29 15:48:33 ip-172-31-43-132 jenkins[7481]: Correct java version found
Oct 29 15:48:33 ip-172-31-43-132 jenkins[7481]:  * Starting Jenkins Automation >
Oct 29 15:48:33 ip-172-31-43-132 su[7520]: (to jenkins) root on none
Oct 29 15:48:33 ip-172-31-43-132 su[7520]: pam_unix(su-l:session): session open>
Oct 29 15:48:33 ip-172-31-43-132 su[7520]: pam_unix(su-l:session): session clos>
Oct 29 15:48:34 ip-172-31-43-132 jenkins[7481]:    ...done.
Oct 29 15:48:34 ip-172-31-43-132 systemd[1]: Started LSB: Start Jenkins at boot>
lines 1-16/16 (END)
```

Now, it's time to visit the Jenkins dashboard.
On the AWS instance page, you'll find the public IP for your instance. Copy it.



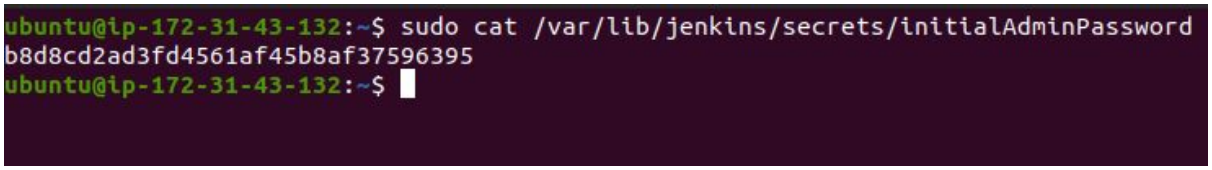
Jenkins runs on port 8080. Hence your jenkins dashboard will run on:
http://<public-ip-of instance>:8080
Example: http://35.88.70.01:8080

Once visited, you'll find Jenkins asking for the administrator password.
You'll find The dashboard telling you the location of the administrator password file. Copy that location.

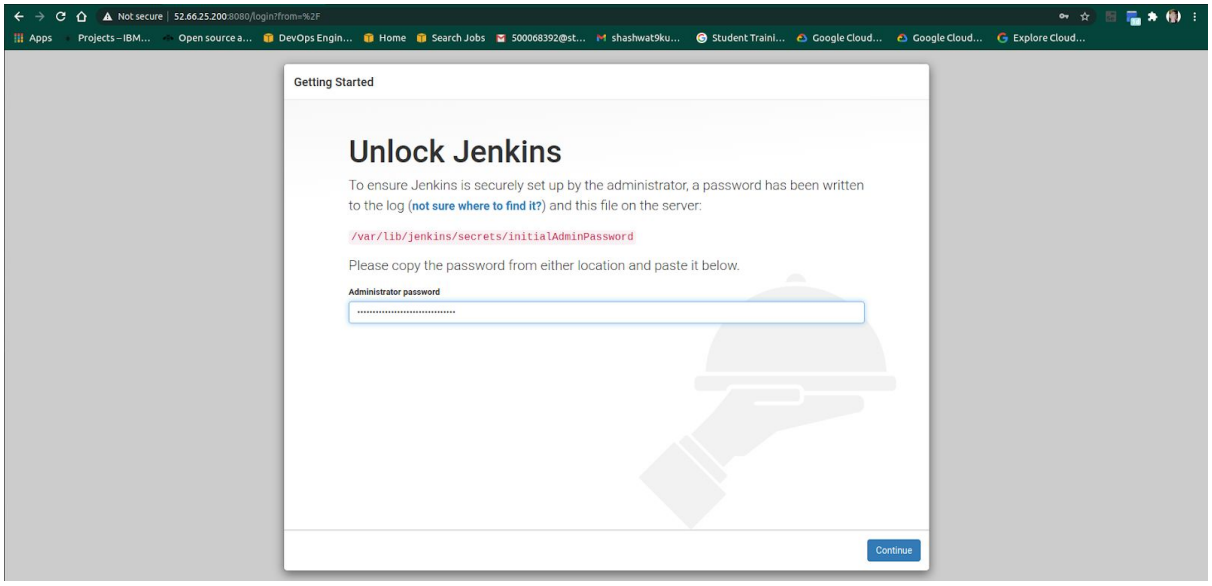


Run the following command to see the admin password:

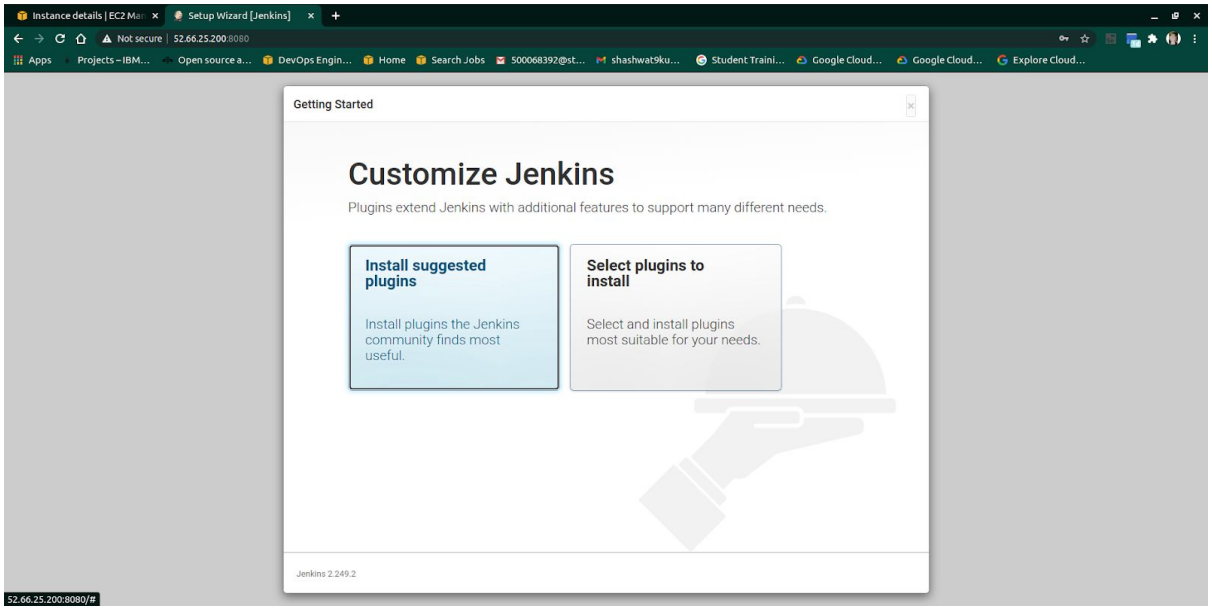
- sudo cat /var/lib/jenkins/secrets/initialAdminPassword



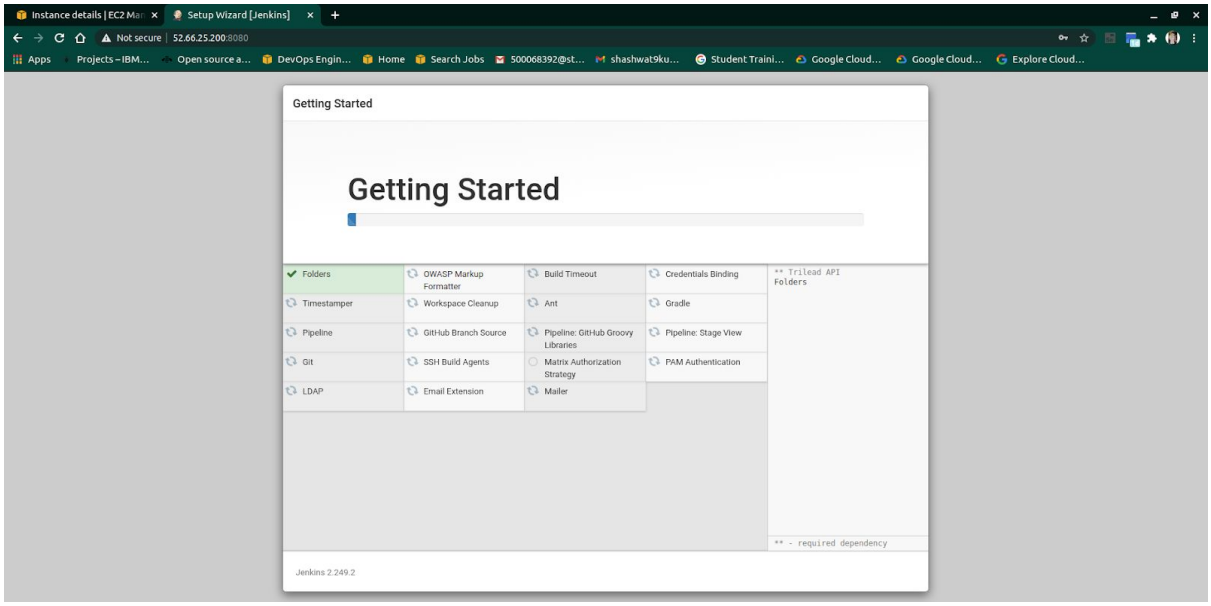
Copy the output and paste it in the space given on the Jenkins dashboard.
Click Continue tab



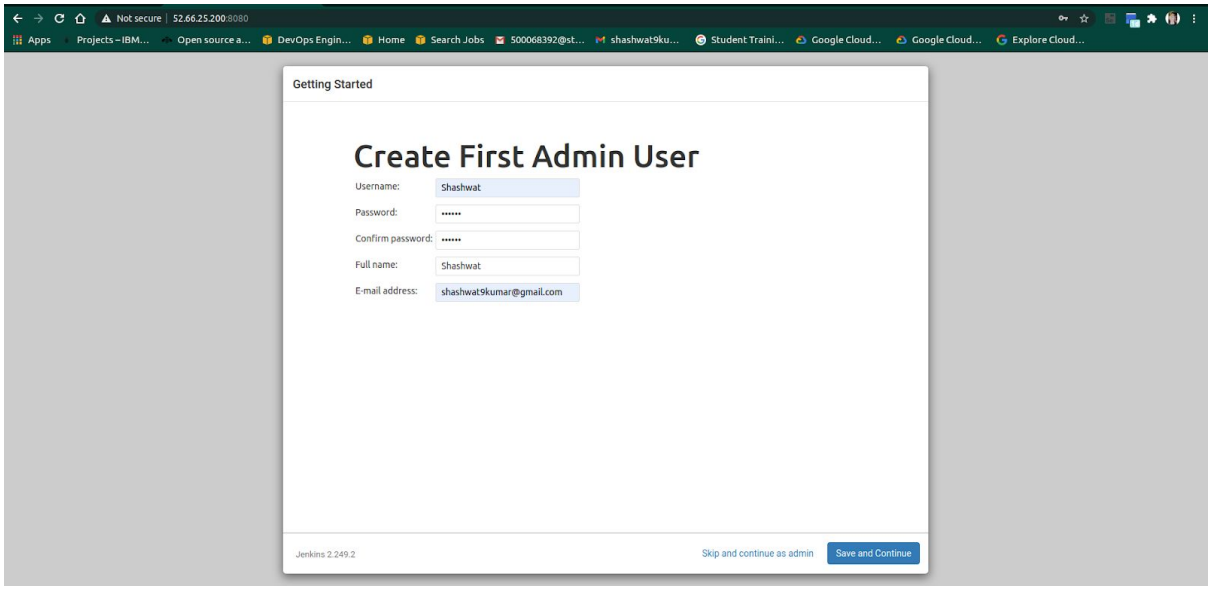
On Customize Jenkins page, select the “Install Suggested Plugins” option



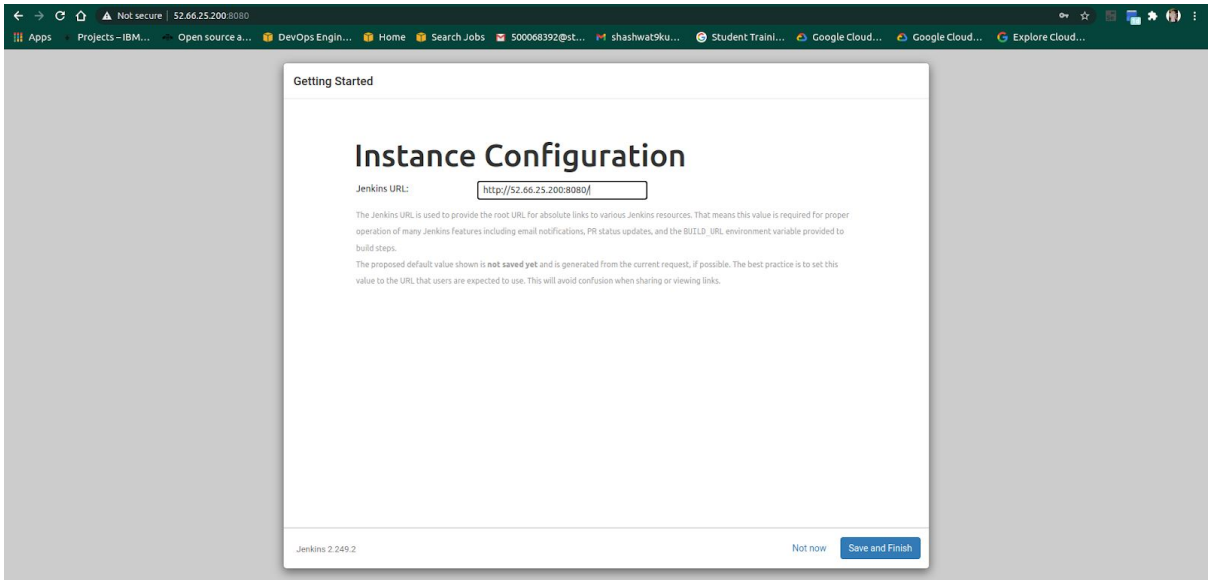
The suggested plugins take some time to get installed.



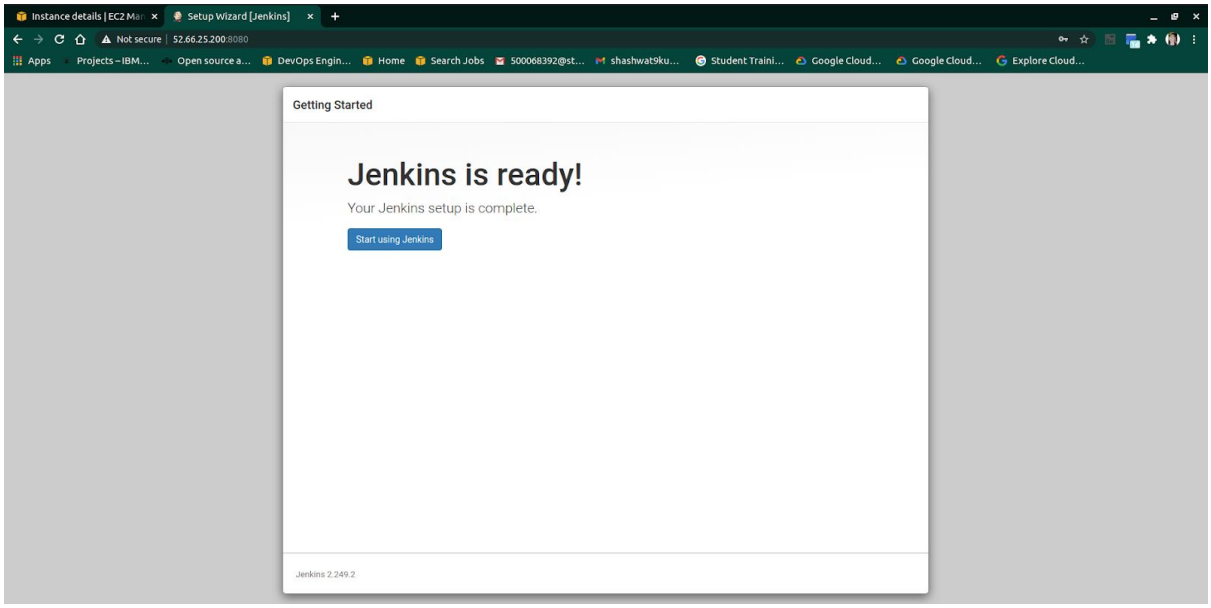
After installation, Jenkins prompts you to create an Admin user.
Fill in the necessary details and remember the username and password you enter.
Click on Save and Continue



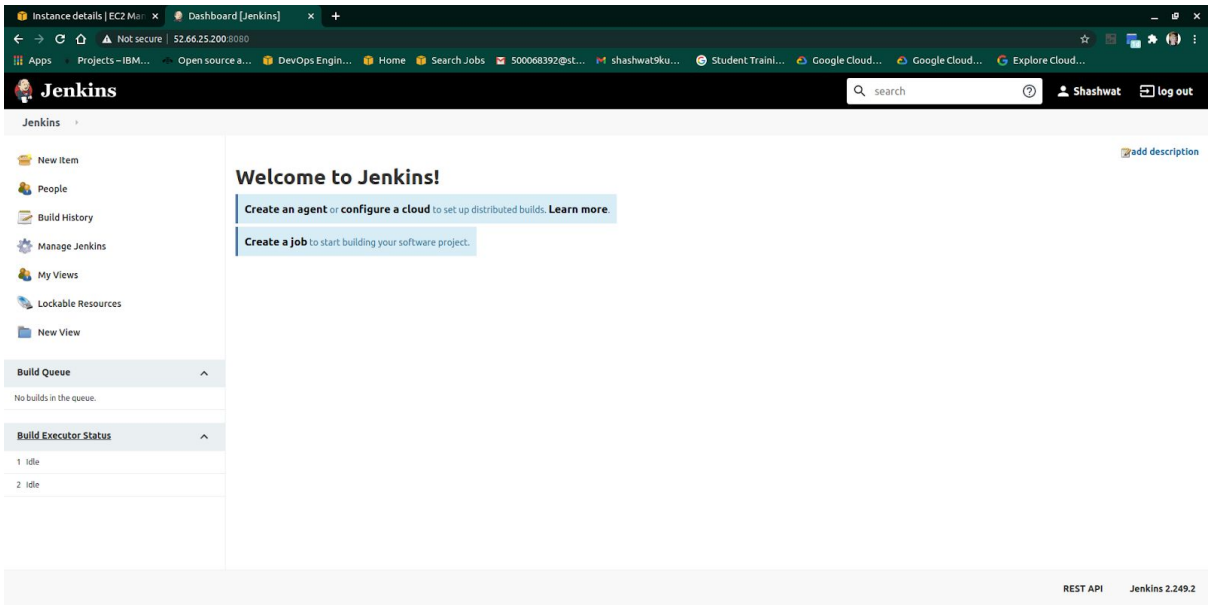
Jenkins now shows you the URL of your Jenkins server.
Click Save and Finish tab



You are now ready to use Jenkins.
Click on Start Using Jenkins tab

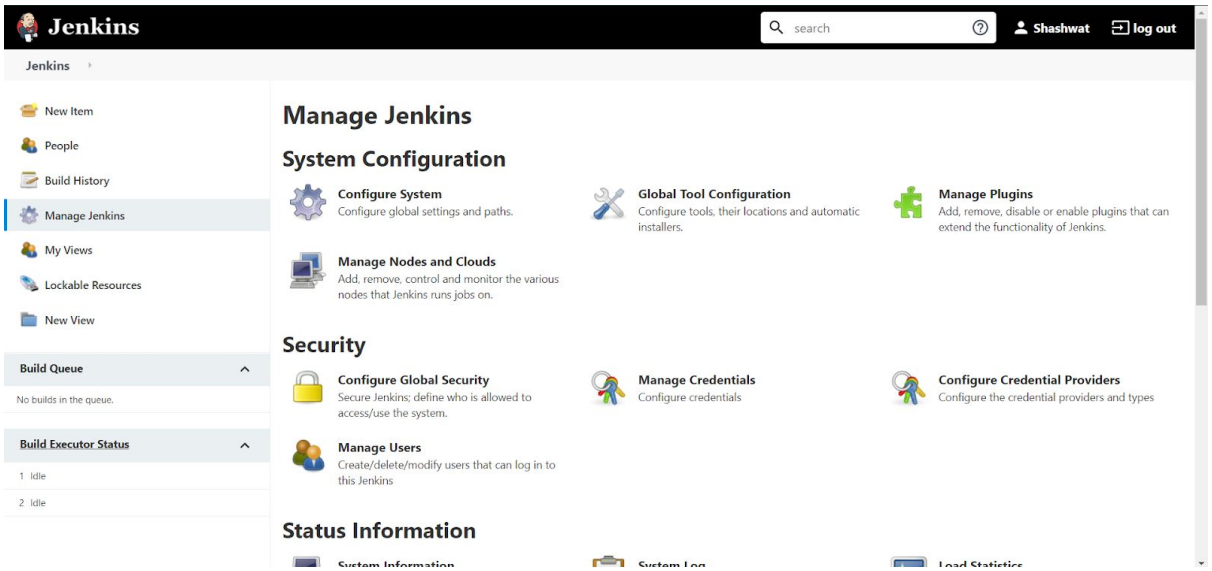


You'll now see the Jenkins dashboard.

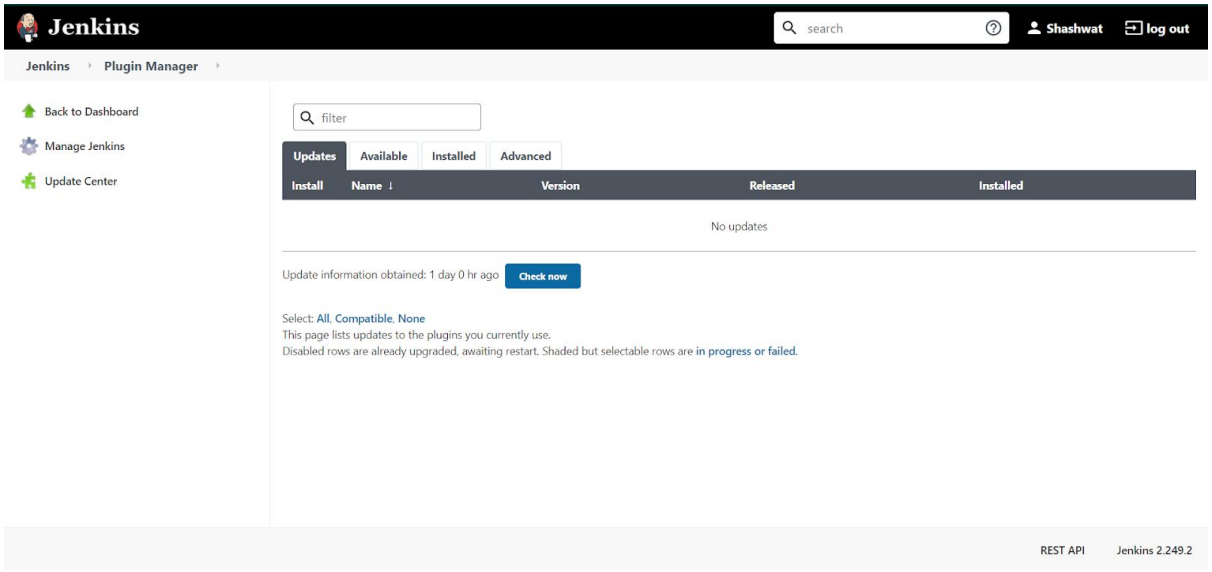


Congratulations. You've now run your very own Jenkins Server. You can now access the Jenkins from any browser and any computer system using URL and the login details.

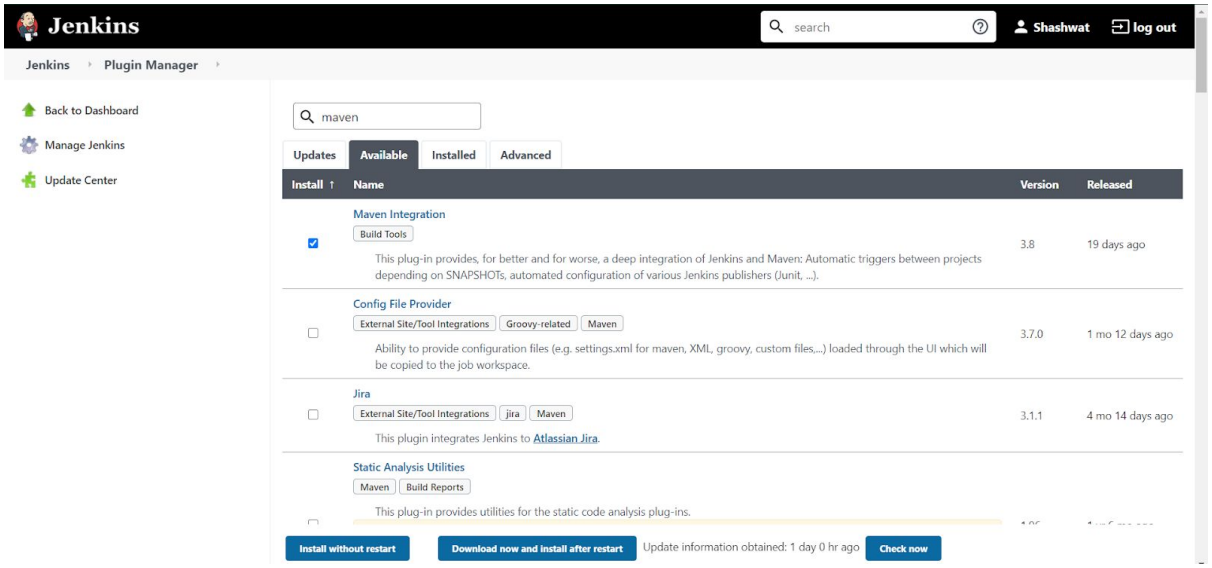
While still on the JEnkins Dashboard page, click on Manage Jenkins Tab on the left
From this Page click on the Manage Plugins option



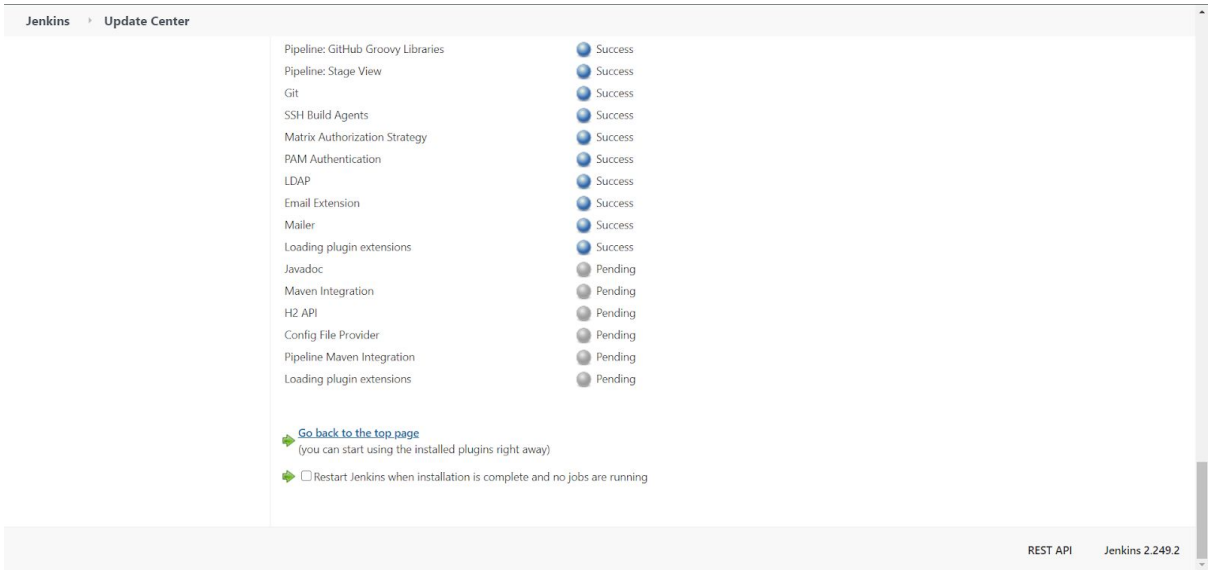
You'll reach the install plugins page. Click on the Available tab



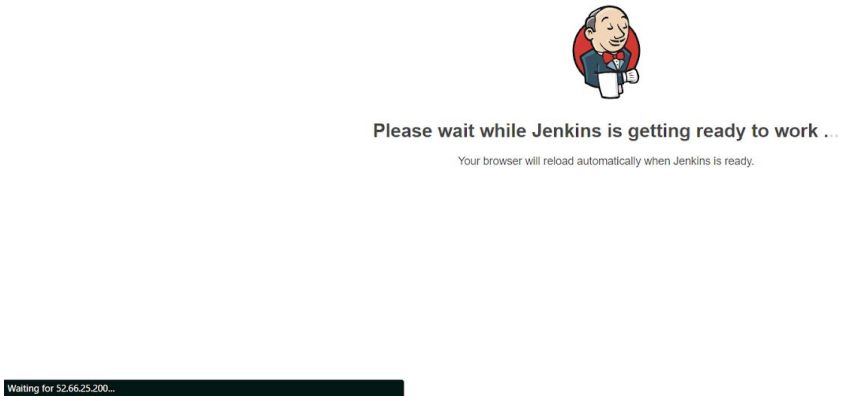
Search for the Maven Integration plugin, Git plugin and Blue Ocean plugin(optional).
Click on the Download and Install after restart option on the bottom of the page



Your plugins will get installed. It will take some time to do so. You can also check the Restart jenkins when installation is complete option

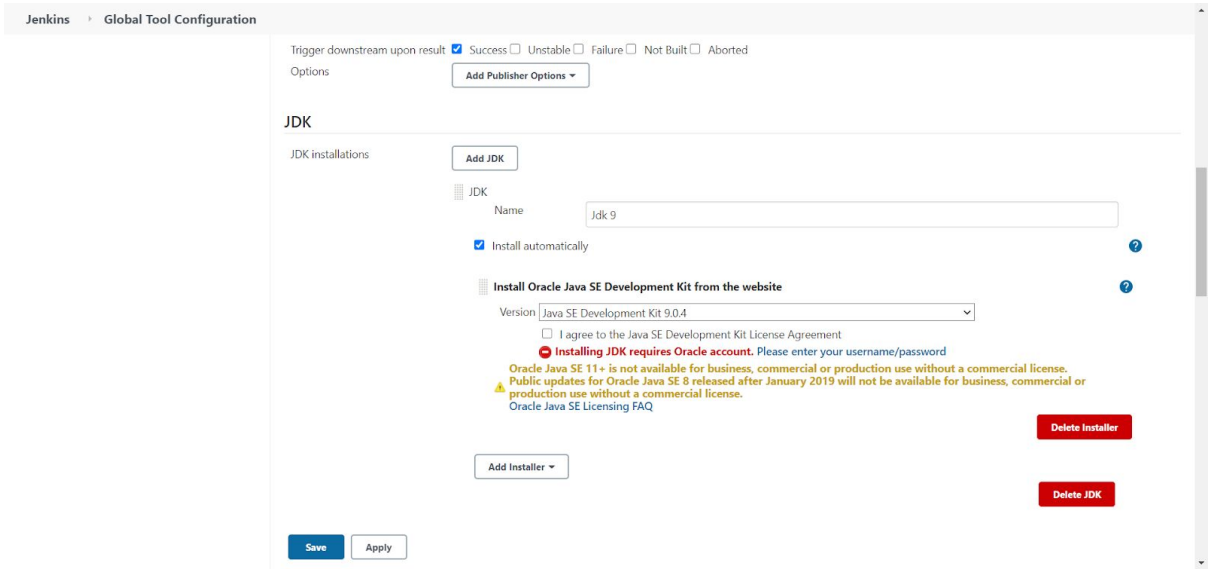


When your plugins get installed, Jenkins will restart itself.

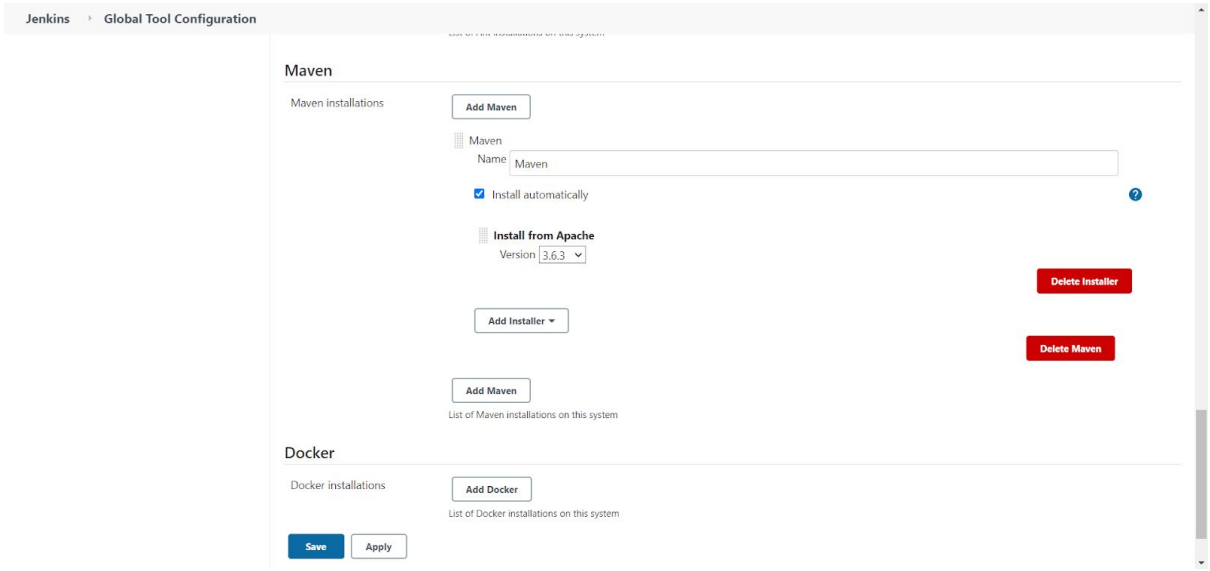


After Jenkins starts again, Click Manage Jenkins Option and select the Global Tool Configuration option

In the JDK section, name the JDK with a name you wish, and either check the “install automatically” or give the path to the installed jdk on the machine. The JAVA_HOME path you stored in the previous step should be provided here.



In the MAven section, name the MAven with a name you wish, and either check the “install automatically” or give the path to the installed maven on the machine. The MAVEN_HOME path you stored in the previous step should be provided here.



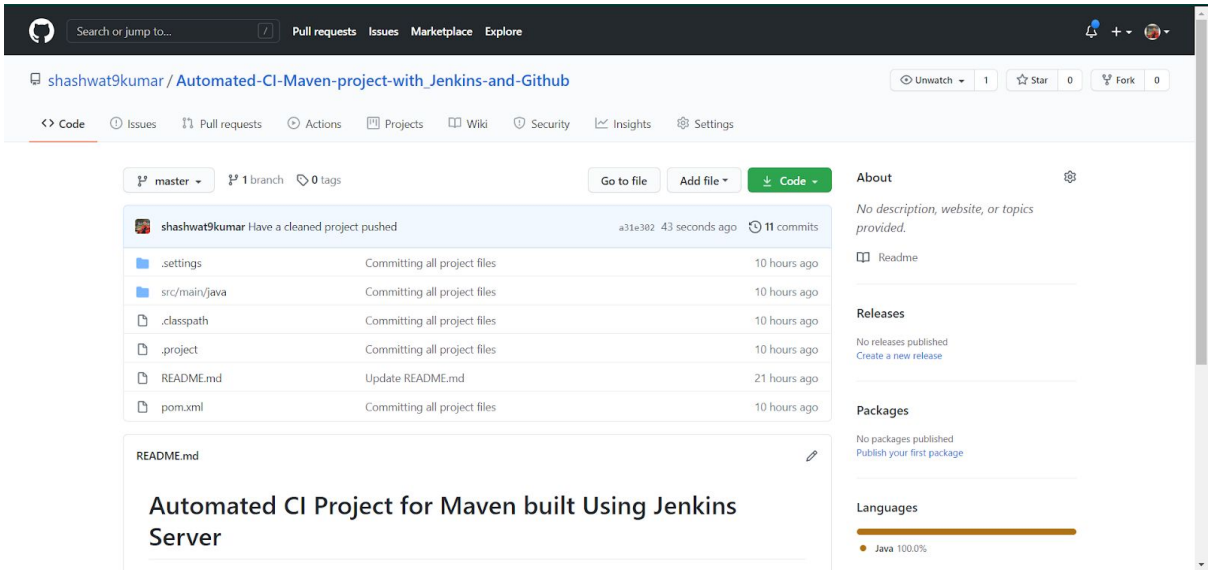
Once done, Click Save option

Step 4

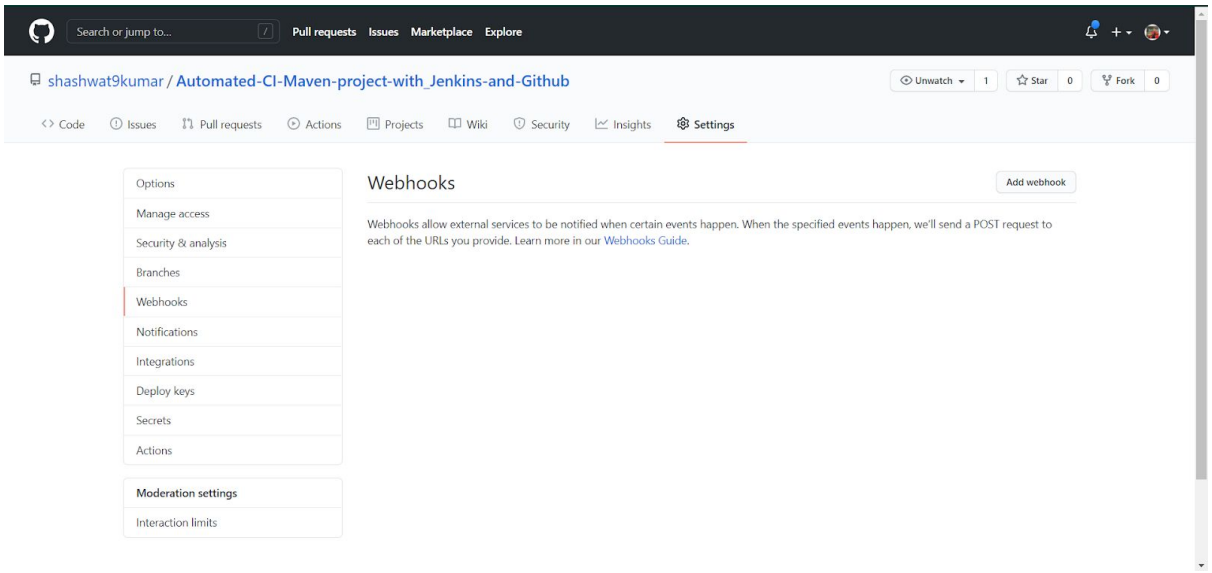
Activate github webhooks for automated build triggering

Go to the Maven project on Github for which you want to set up the automated CI pipeline. (If you do not have an existing maven project you may push a demo maven project a github repository to follow along)

Click on the Setting tab

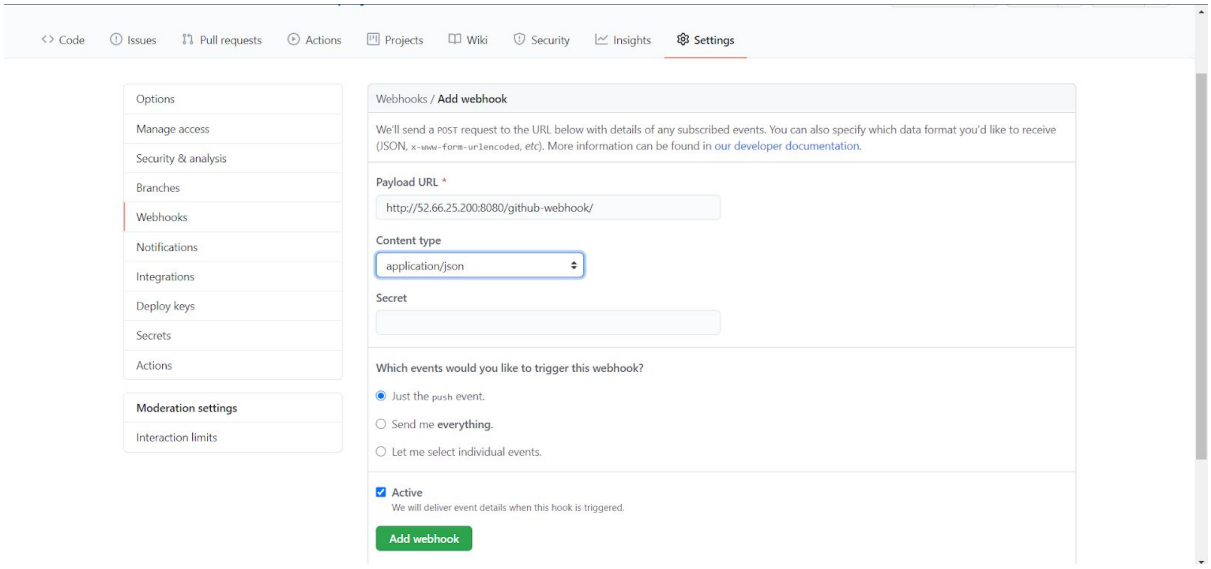


Now, click on the Webhooks option on the left panel.
Click the Add webhook button

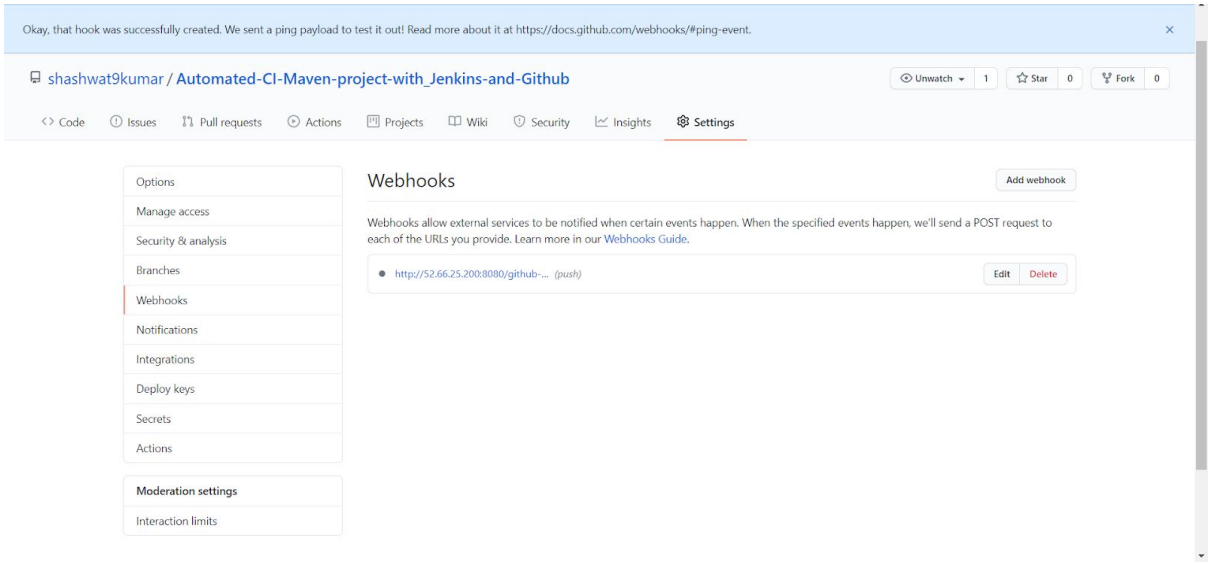


In the payload url option, paste the jenkins server url and append “/github-webhook/”to it.
Example: http://23.44.55.66:8080/github-webhook/

In the Content Type select the application/JSON option
Check the “Just the push event”
Also check the “Active”
Now click on Add webhook button



Congratulations, you've enabled a webhook. Now github will inform Jenkins server whenever a push even occurs.

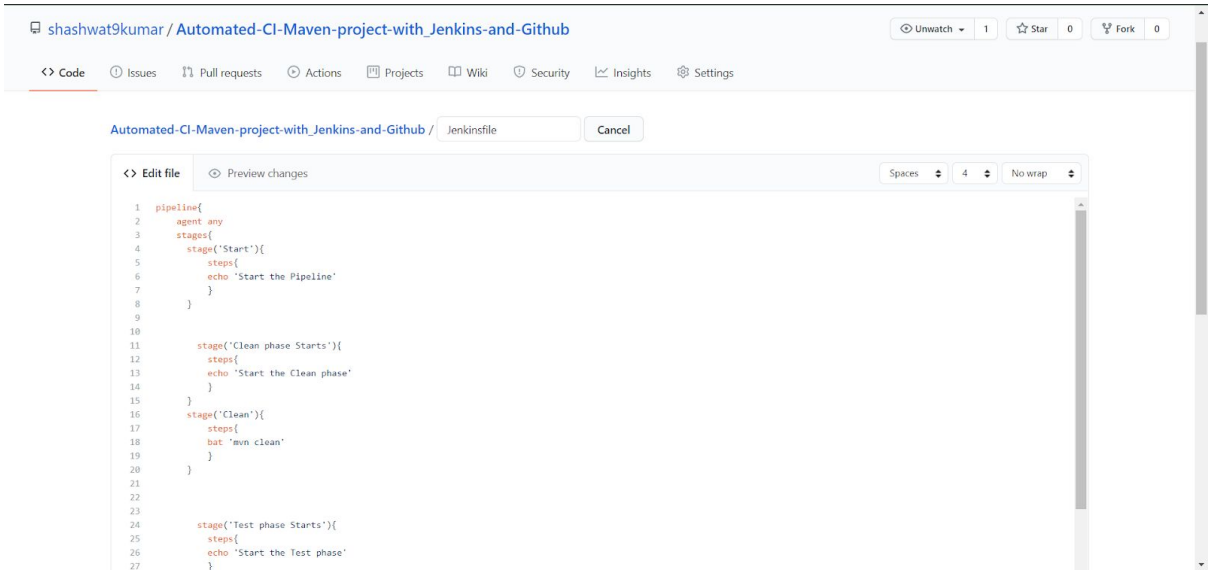


Additionally, for more automation, we define the “Jenkinsfile” in the github project.
Create a new Jenkinsfile in the github project if you don't have one already.

(The Jenkinsfile used in this example is as follows:

```
pipeline{
    agent any
    stages{
      stage('Start'){
         steps{
         echo 'Start the Pipeline'
         }
      }

        stage('Clean phase Starts'){
         steps{
         echo 'Start the Clean phase'
         }
      }
      stage('Clean'){
         steps{
         sh 'mvn clean'
         }
      }

        stage('Compile phase Starts'){
         steps{
         echo 'Start the Compile phase'
         }
      }
      stage('Compile'){
         steps{
         sh 'mvn compile'
         }
      }

        stage('Test phase Starts'){
         steps{
         echo 'Start the Test phase'
         }
      }
        stage('Test'){
         steps{
         sh 'mvn test'
      }
}

        stage('Install phase Starts'){
         steps{
         echo 'Start the Instll phase'
         }
      }
        stage('Install'){
         steps{
         sh 'mvn install'
         }
      }



        stage('End'){
         steps{
         echo 'End the pipeline'
         }
      }
    }
}
```

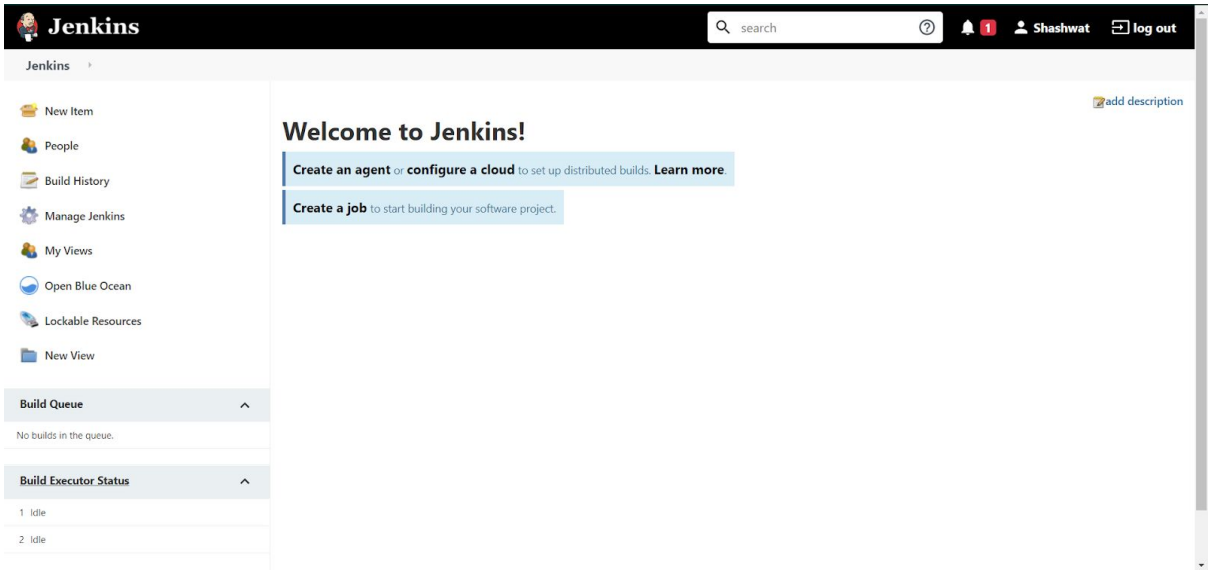
)



Step 5

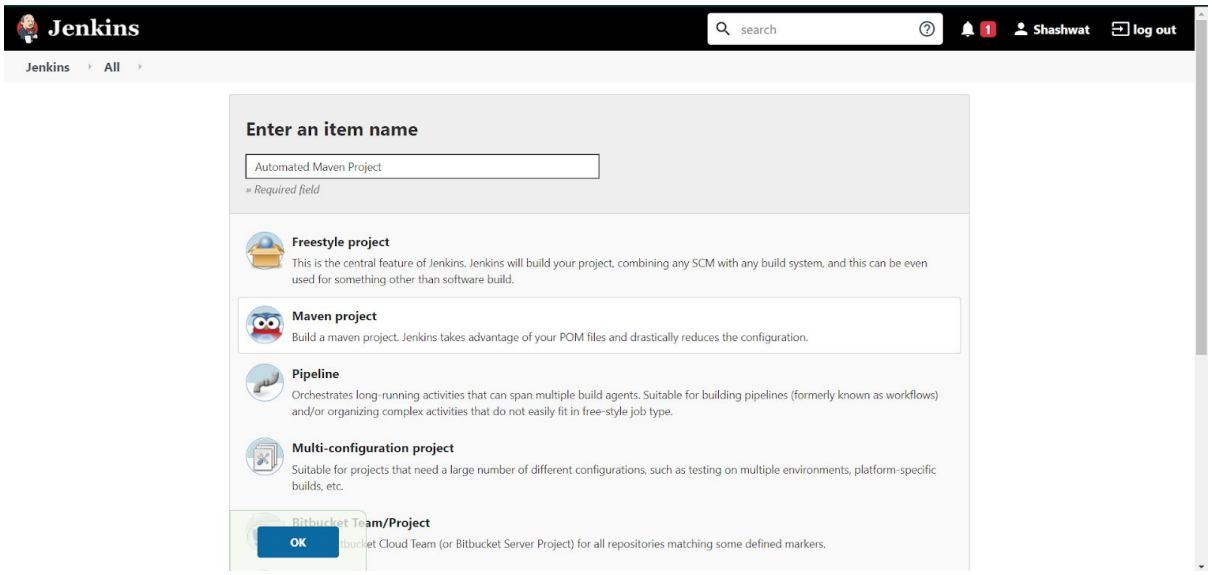
Creating the automated pipeline for Continuous Integration

Return to the Jenkins Dashboard.
Click on New Item option to create a new Jenkins pipeline

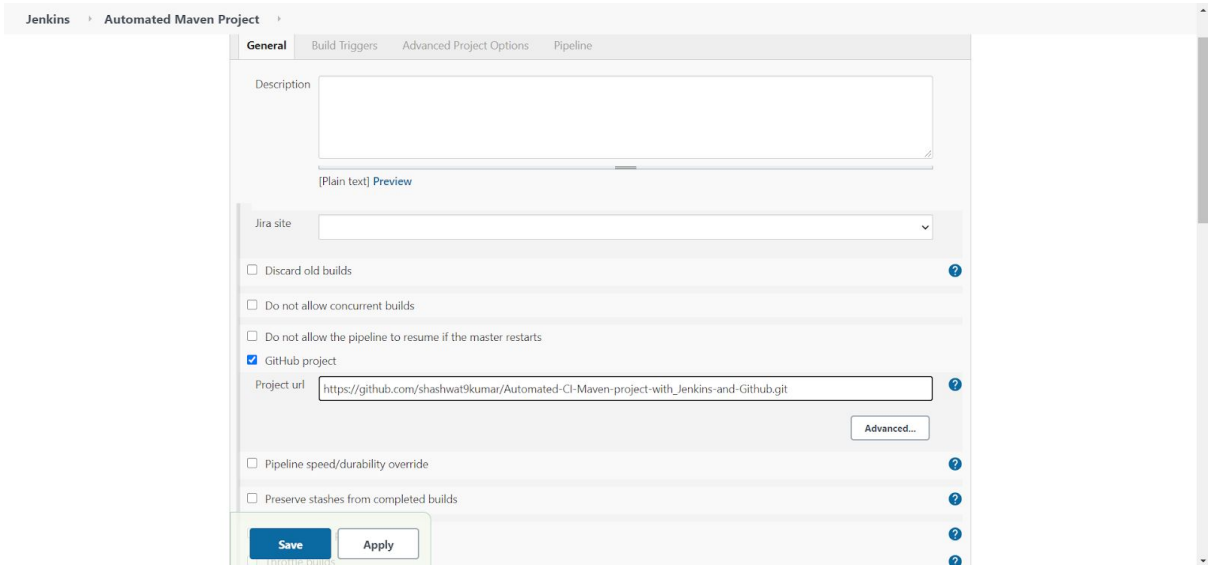


Here on this page, name the project with a name you find suitable.

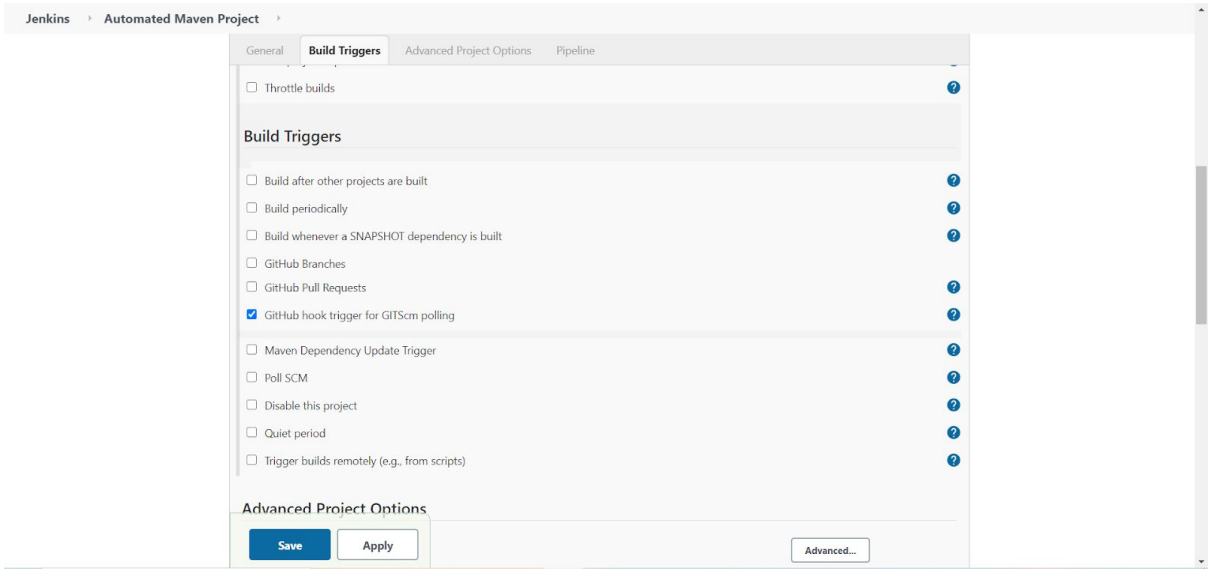
Select the project type as “Pipeline”
Click OK tab



In the General Section, check the Github Project option and give the location to the project repository

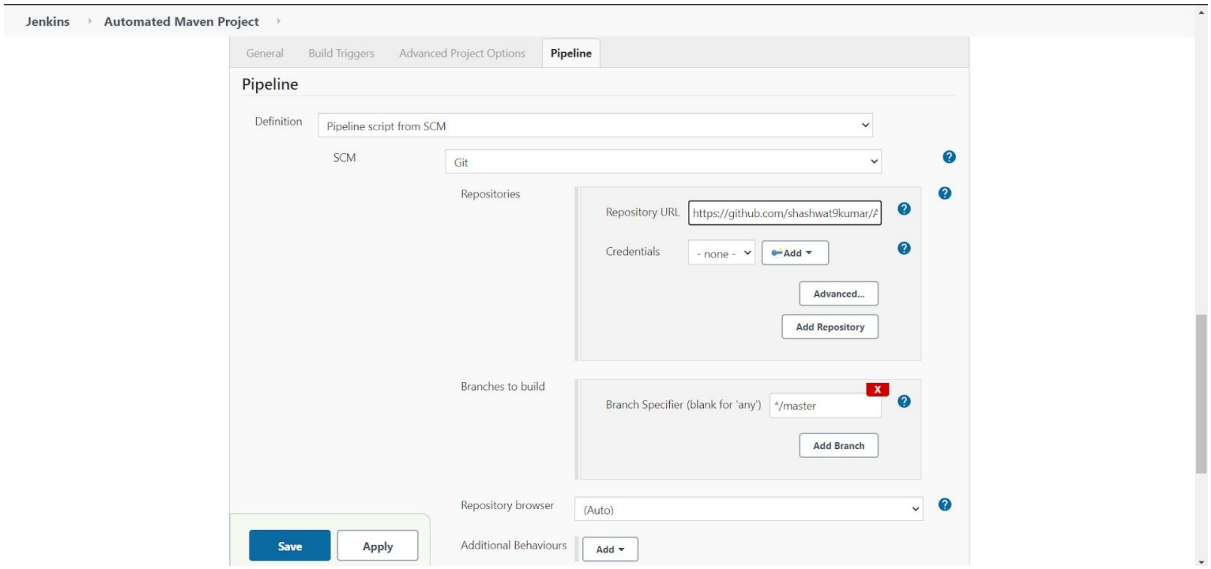


In the Build Triggers section, select the GitHub hook trigger for GITScm polling option

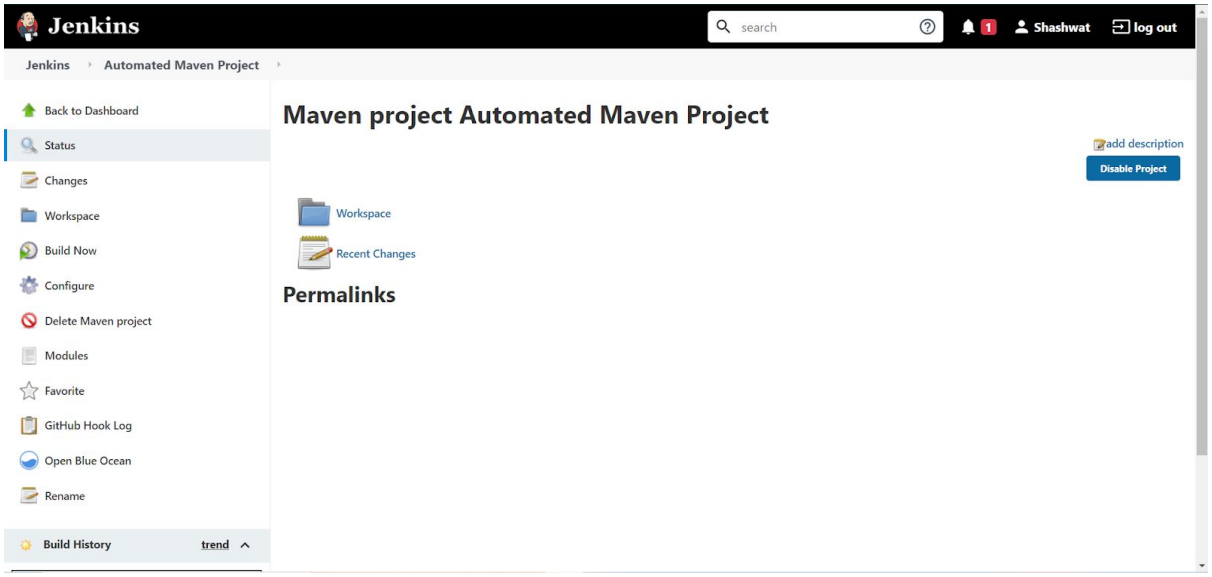


In the Pipeline section, select Pipeline Script from SCM option in the Definition box
Add SCM as Git
And give the repository link to the github repository

Click the **Save** button at the bottom



Once done, you'll reach a page similar to this.



Congratulations. You've now created an automated CI pipeline that triggers the Jenkins server whenever you push on github.

Step 6

Testing the pipeline via a test push/commit

And now that an automated pipeline is built, the only thing that's left to do is test it.
Open a terminal or Command line (depending upon your OS)
Move to the project repository on your local machine

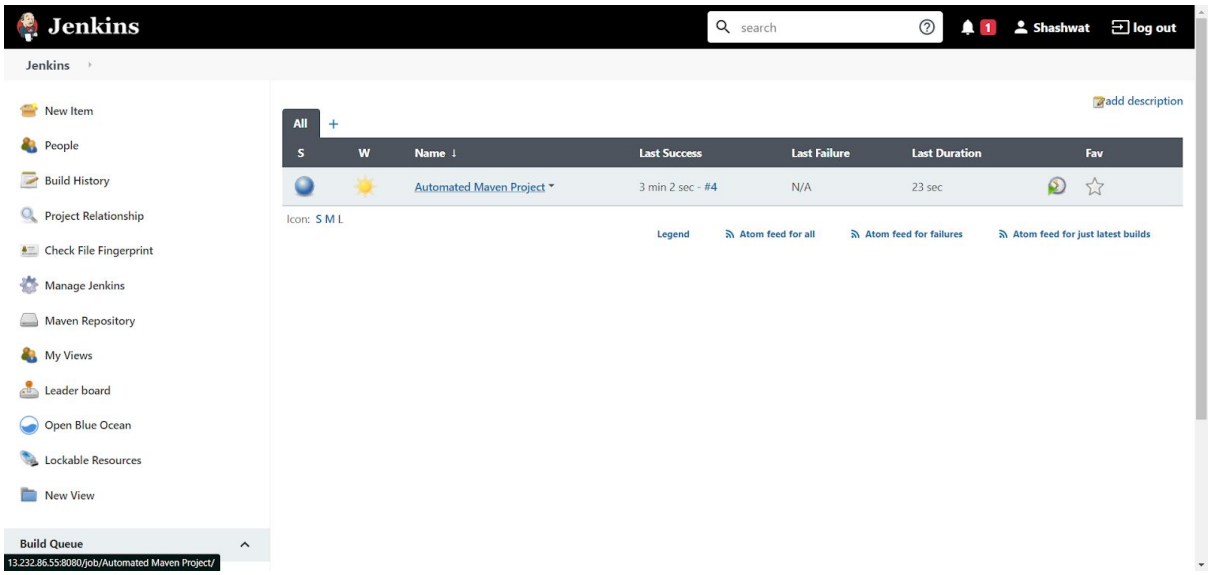
Run the following commands:

- git pull origin master ## best practice is to pull before you push
- <Make some changes to the project, however small. Eg: change the ReadMe.md file>
- git add .
- git commit -m "Commit message"
- git push -u origin master

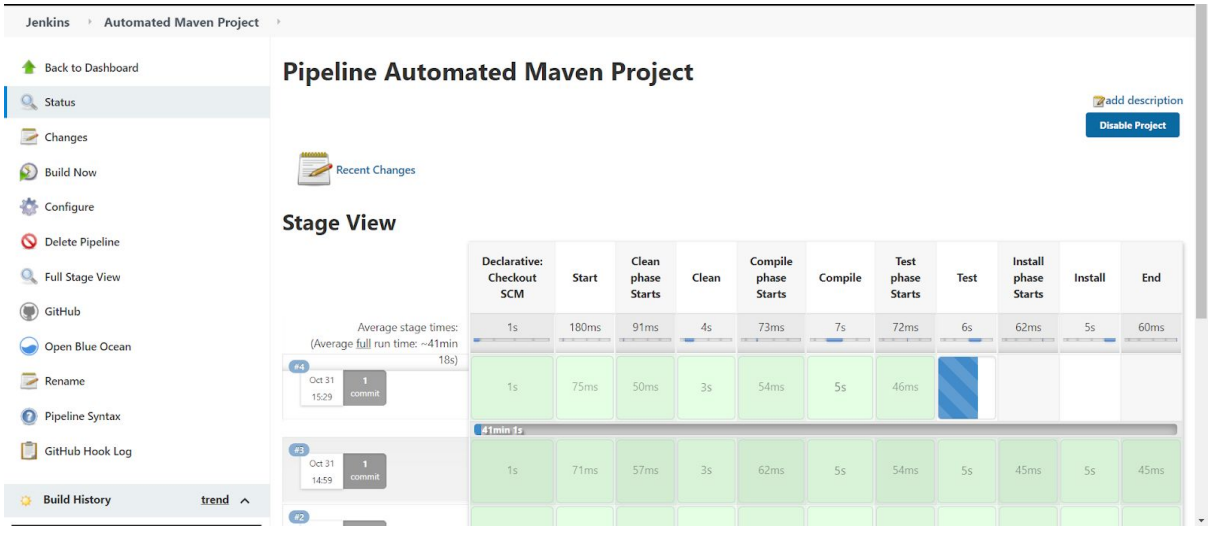
You'll see your changes pushed to the remote repository on Github.

```
D:\Project\Automated-CI-Maven-project-with_Jenkins-and-Github>git pull origin master
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 843 bytes | 6.00 KiB/s, done.
From https://github.com/shashwat9kumar/Automated-CI-Maven-project-with_Jenkins-and-Github
 * branch            master     -> FETCH_HEAD
   a31e302..959e68b  master     -> origin/master
Updating a31e302..959e68b
Fast-forward
 Jenkinsfile | 55 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 1 file changed, 55 insertions(+)
 create mode 100644 Jenkinsfile

D:\Project\Automated-CI-Maven-project-with_Jenkins-and-Github>git add .

D:\Project\Automated-CI-Maven-project-with_Jenkins-and-Github>git commit -m "Test Commit"
[master 12e314c] Test Commit
 1 file changed, 1 insertion(+), 1 deletion(-)

D:\Project\Automated-CI-Maven-project-with_Jenkins-and-Github>git push -u origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 322 bytes | 322.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/shashwat9kumar/Automated-CI-Maven-project-with_Jenkins-and-Github.git
   959e68b..12e314c  master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.

D:\Project\Automated-CI-Maven-project-with_Jenkins-and-Github>
```

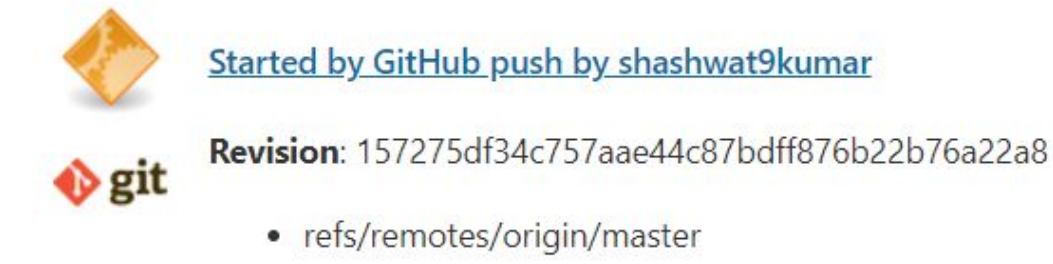
Additionally see your pipeline on Jenkins server. It starts building too.
Click on your project on the Jenkins Dashboard.



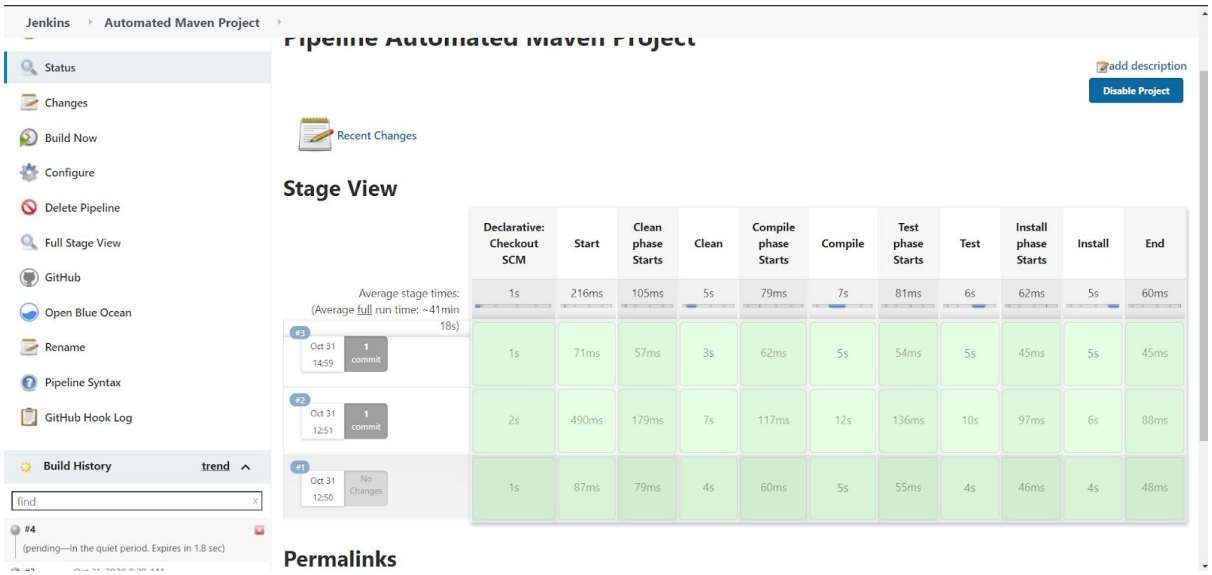
On clicking, you'll find your project has started building on its own. This was because github notifies Jenkins of the push event that occurred on it.



The status of the build also mentions that



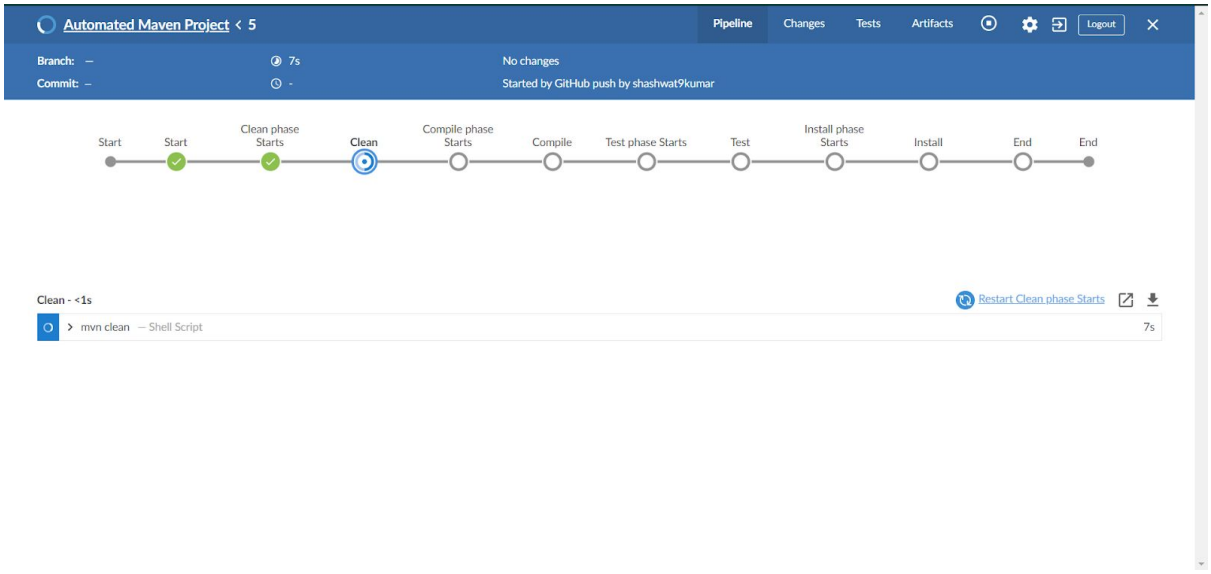
If the project has no errors all the stages complete successfully. And the pipeline runs successfully.

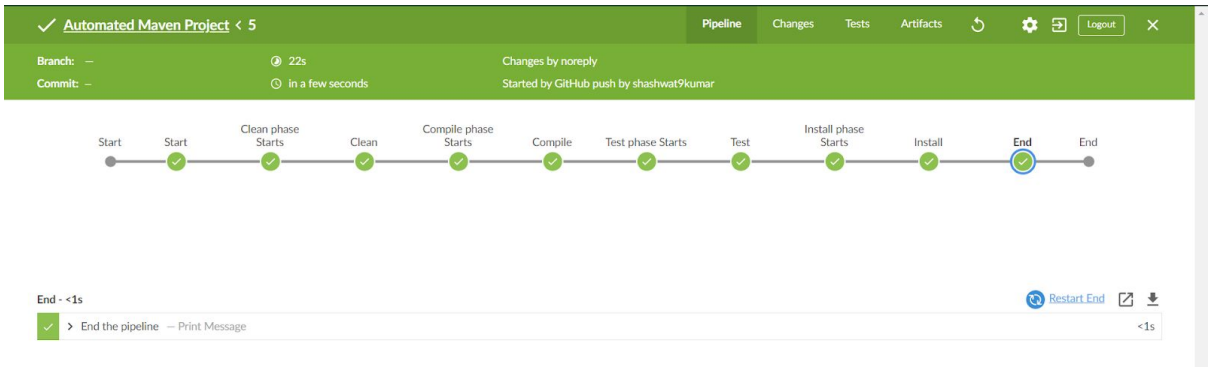


Also the console output shows BUILD SUCCESS

For a more visual pipeline and better representation, you may additionally click on the Blue Ocean option on the left panel (if you installed the plugin before)

Blue Ocean provides a better visual of the rather mundane CI pipeline.





Congratulations. Now you've created, enabled and tested an automated CI pipeline.