INDIAN INSTITUTE OF INFORMATION TECHNOLOGY

DevOps & its Applications (CS457)

Jenkins Assignment 2

Under the Guidance of - **Dr. Uma S**

Submitted by -

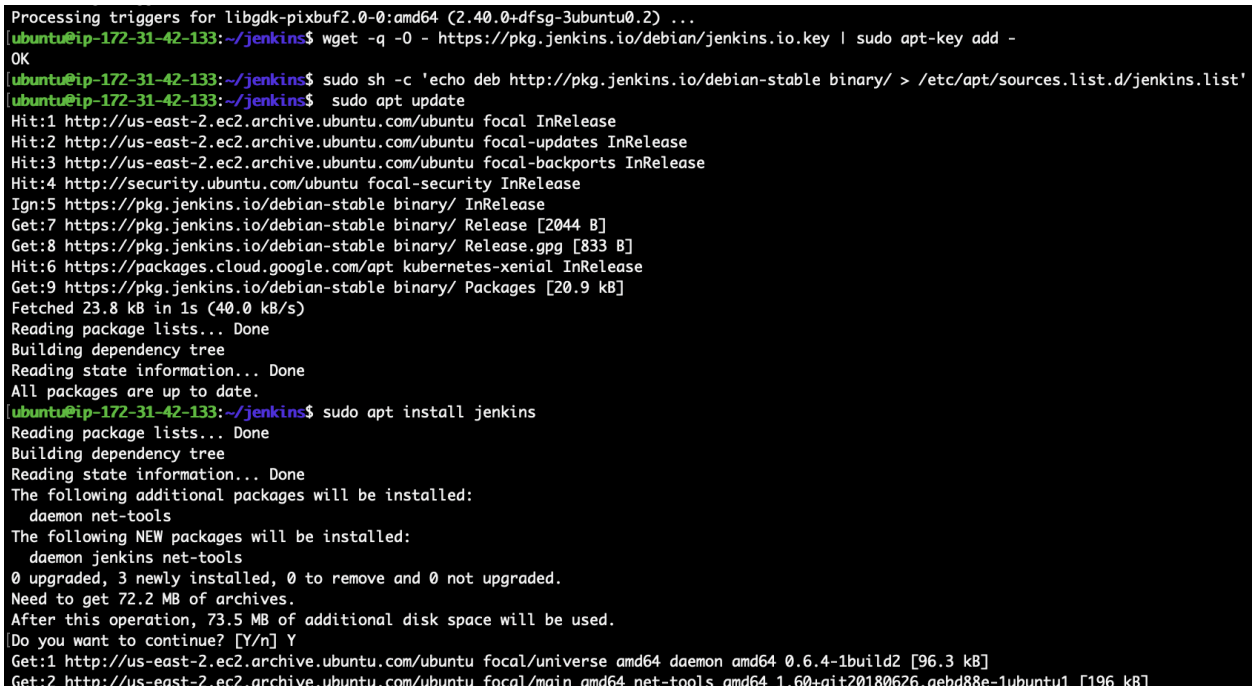
Advay Aggarwal (18BCS002)
Meghna Hadimani (18BCS052)
Perumalla Tushar (18BCS065)
Rahul Priyadarshi (18BCS074)
Samana B S (18BCS088)

JENKINS MASTER - SLAVE PIPELINE

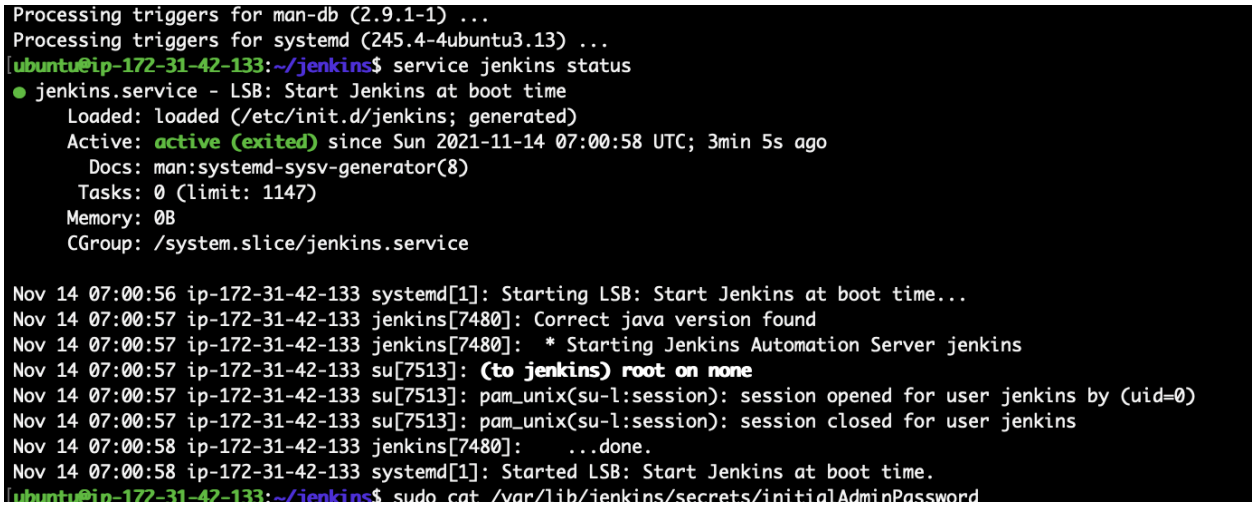
Tech Stack:

1. Git
2. Docker
3. Jenkins
4. Node
5. Three AWS EC2 Instances (t2.micro)

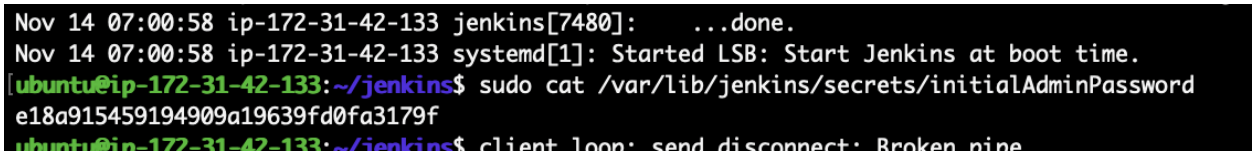
Step1: Install Jenkins on an EC2 instance with the commands mentioned below in the screenshot.



Step2: After installation, we can see the status of the jenkins service. It should be “active” as mentioned in the screenshot below.



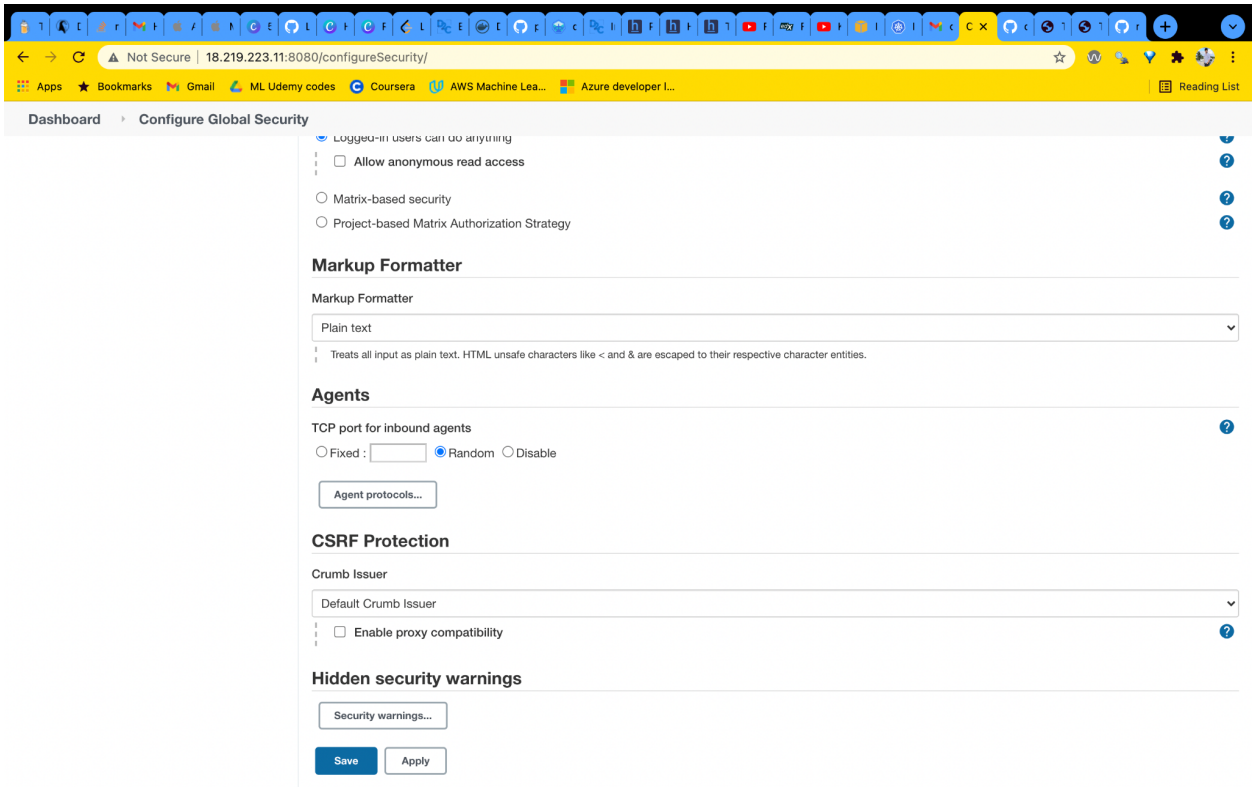
Step3: Now enter the public IP address of the EC2 instance followed by port number 8080 (by default, Jenkins runs on port 8080) in a browser to access the Jenkins Dashboard. Here, it will prompt to enter the Admin Password to unlock Jenkins. The Admin Password can be found in the file whose absolute path will be mentioned in the prompt. Read the content of the file as mentioned in the screenshot below, and enter that in the Admin Password field.



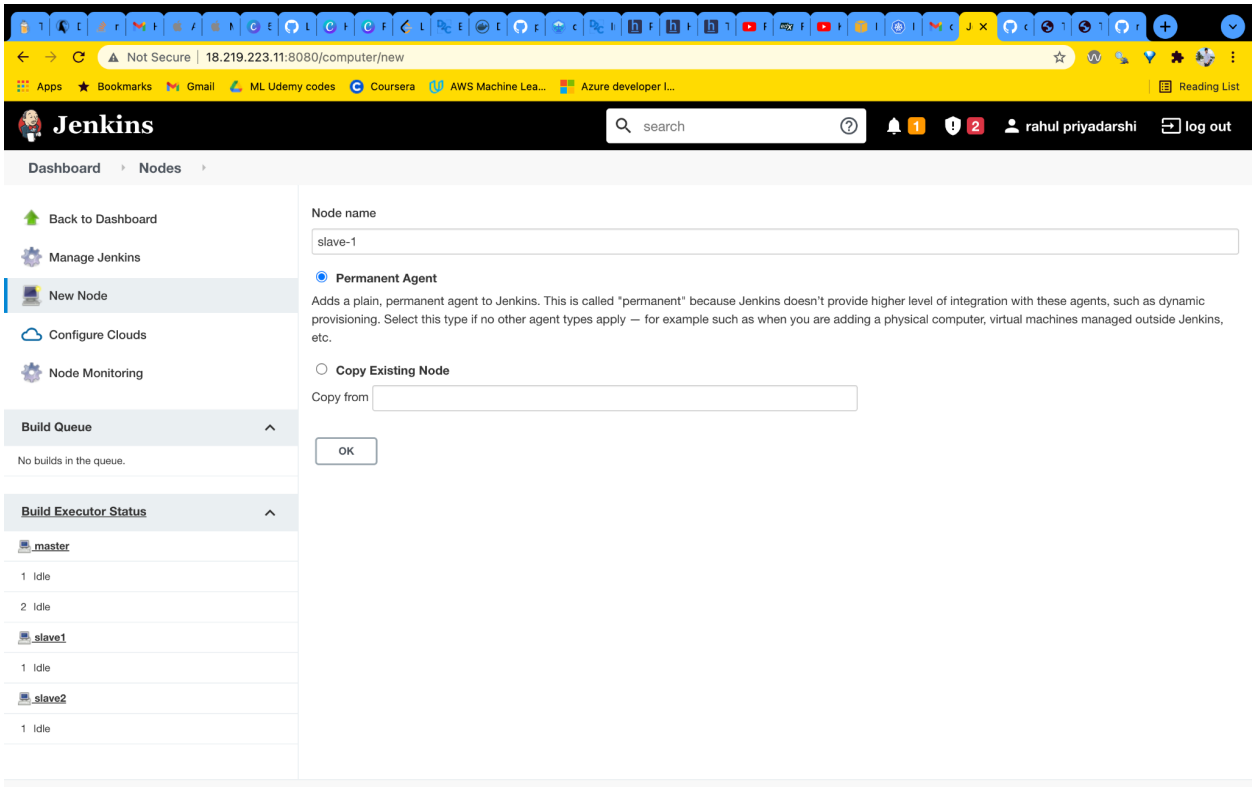
Step4: After entering the correct password, Jenkins will prompt to create a new user by entering username, password, and email. Enter the same and login.

Jenkins is now ready to use.

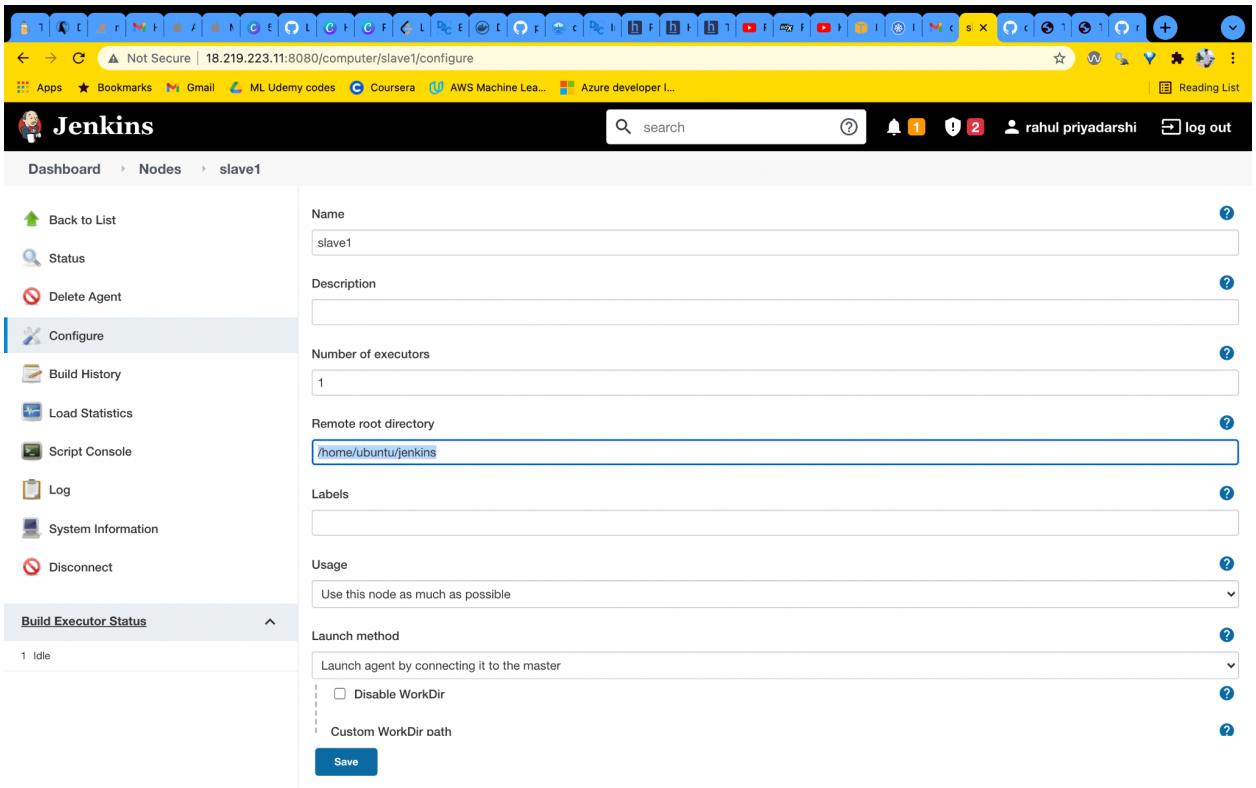
Step5: Name the other two EC2 instances as ‘slave1’ and ‘slave2’ to be set-up as test and production servers. Then, from the Jenkins Dashboard, go to Manage Jenkins ⇒ Configure global security. In ‘Agents’, change the ‘TCP port for inbound agents’ to ‘Random’, and click ‘Save’.



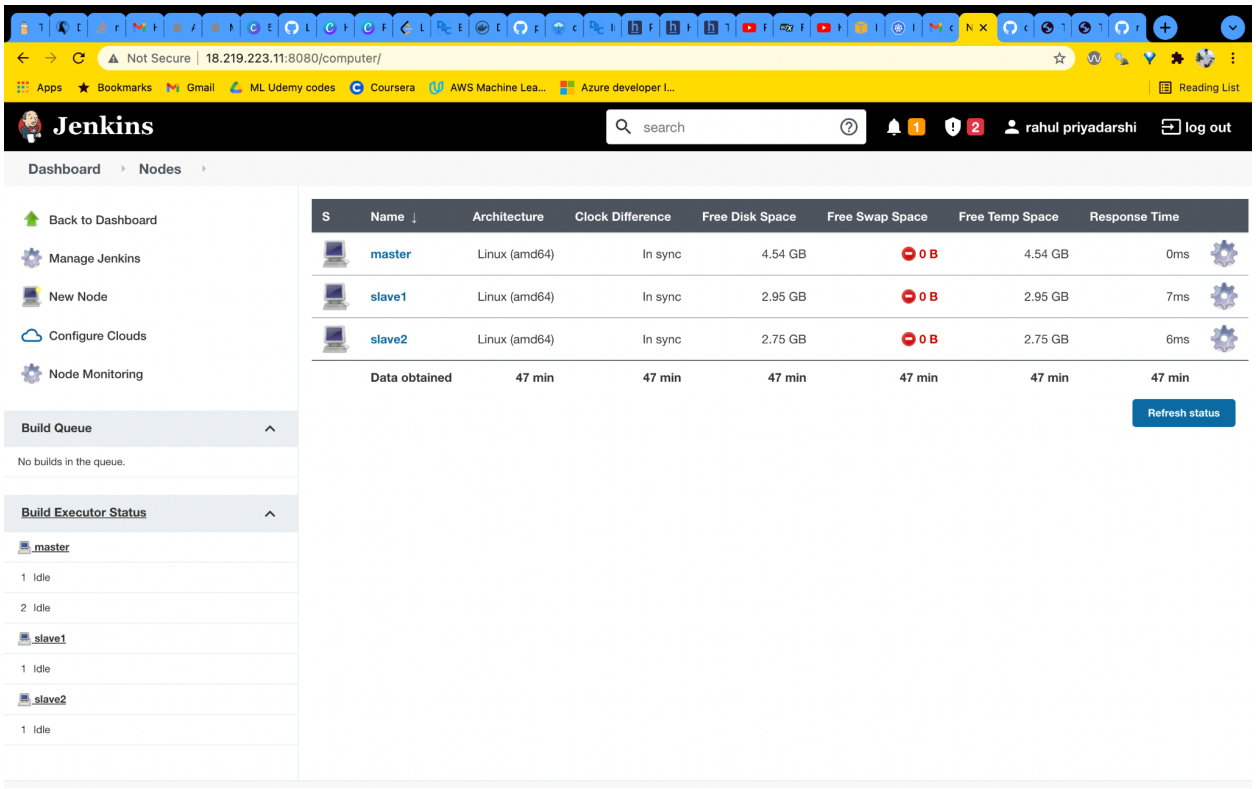
Step6: Add slave1 and slave2 nodes to Jenkins master. Go to Manage Jenkins ⇒ Manage Nodes and Clouds ⇒ New Node. Now enter the name as slave1/slave2 and enable ‘Permanent Agent’. Then click ‘OK’.



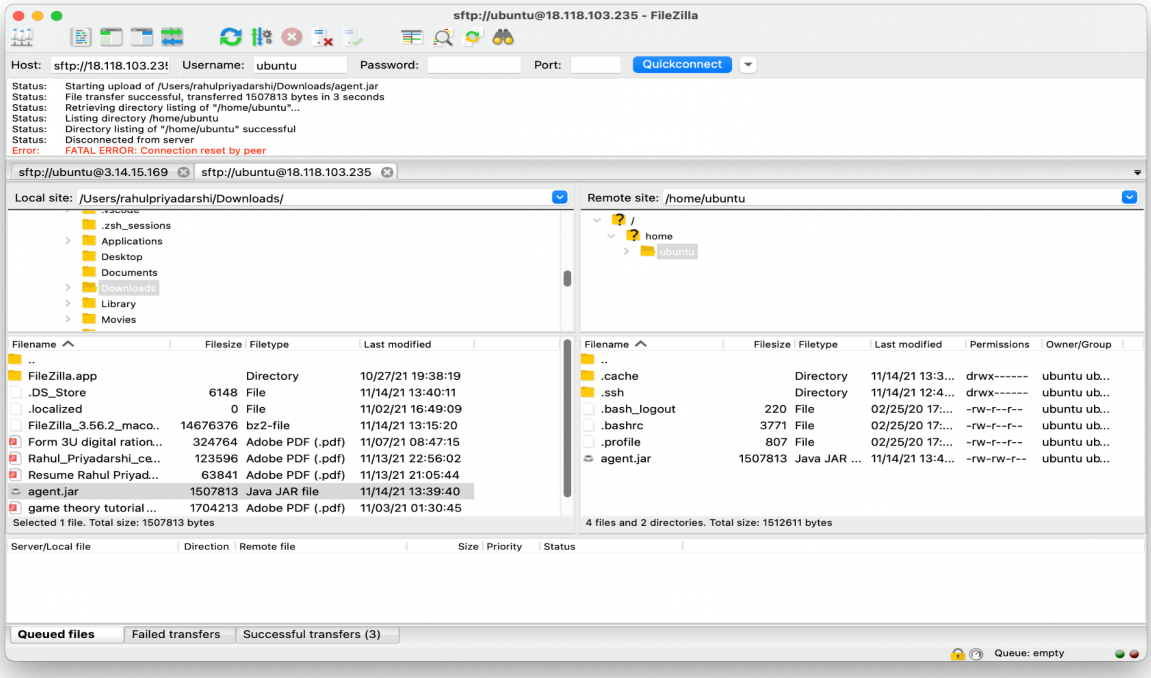
Step7: Now configure the created Nodes and enter '/home/ubuntu/jenkins' in the 'Remote root directory' as mentioned in the screenshot below and click Save.



This is how the 'Manage Nodes' dashboard looks like:



Step8: Go to slave1 and download the agent.jar file. Now open FileZilla application and connect it to slave1 EC2 node. Then transfer the agent.jar file to the node using SFTP.



Repeat the step for both the nodes.

Step9: Node install JDK on both the slave nodes.

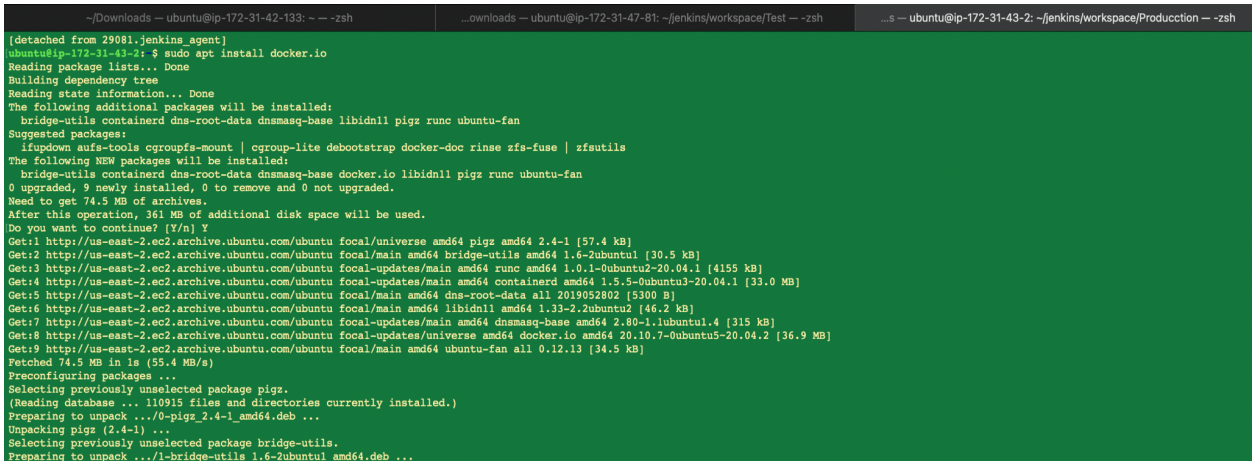
```
Found linux image: /boot/vmlinuz-5.11.0-1020-aws
Found initrd image: /boot/microcode.cpio /boot/initrd.img-5.11.0-1020-aws
Found Ubuntu 20.04.3 LTS (20.04) on /dev/xvda1
done
ubuntu@ip-172-31-43-2:~$ sudo apt install openjdk-8-jdk
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  adwaita-icon-theme at-spi2-core ca-certificates-java fontconfig fontconfig-
  libasyncns0 libatk-bridge2.0-0 libatk-wrapper-java libatk-wrapper-java-jni
  libcups2 libdatrie1 libdrm-amdgpu1 libdrm-intel1 libdrm-nouveau2 libdrm-rad
  libgdk-pixbuf2.0-common libgif7 libgl1 libgl1-mesa-dri libgl1-mesa-glx libg
  libice6 libjbig0 libjpeg-turbo8 libjpeg8 liblcms2-2 libllvm12 libpango-1.0-
  librsvg2-common libsensors-config libsensors5 libsm-dev libsm6 libsndfile1
  libxcb-dri2-0 libxcb-dri3-0 libxcb-glx0 libxcb-present0 libxcb-randr0 libxc
  libxdmcp-dev libxfixes3 libxft2 libxi6 libxinerama1 libxkbfile1 libxmu6 lib
  openjdk-8-jdk-headless openjdk-8-jre openjdk-8-jre-headless ubuntu-mono x11
Suggested packages:
  default-jre cups-common gvfs libice-doc liblcms2-utils pcscd pulseaudio lib
  libnss-mdns fonts-ipafont-gothic fonts-ipafont-mincho fonts-wqy-microhei fo
The following NEW packages will be installed:
  adwaita-icon-theme at-spi2-core ca-certificates-java fontconfig fontconfig-
  libasyncns0 libatk-bridge2.0-0 libatk-wrapper-java libatk-wrapper-java-jni
  libcups2 libdatrie1 libdrm-amdgpu1 libdrm-intel1 libdrm-nouveau2 libdrm-rad
```

Step10: Now copy the command from the slave nodes on Jenkins, and execute them on the EC2 instances as shown below, This should be done for both the nodes.

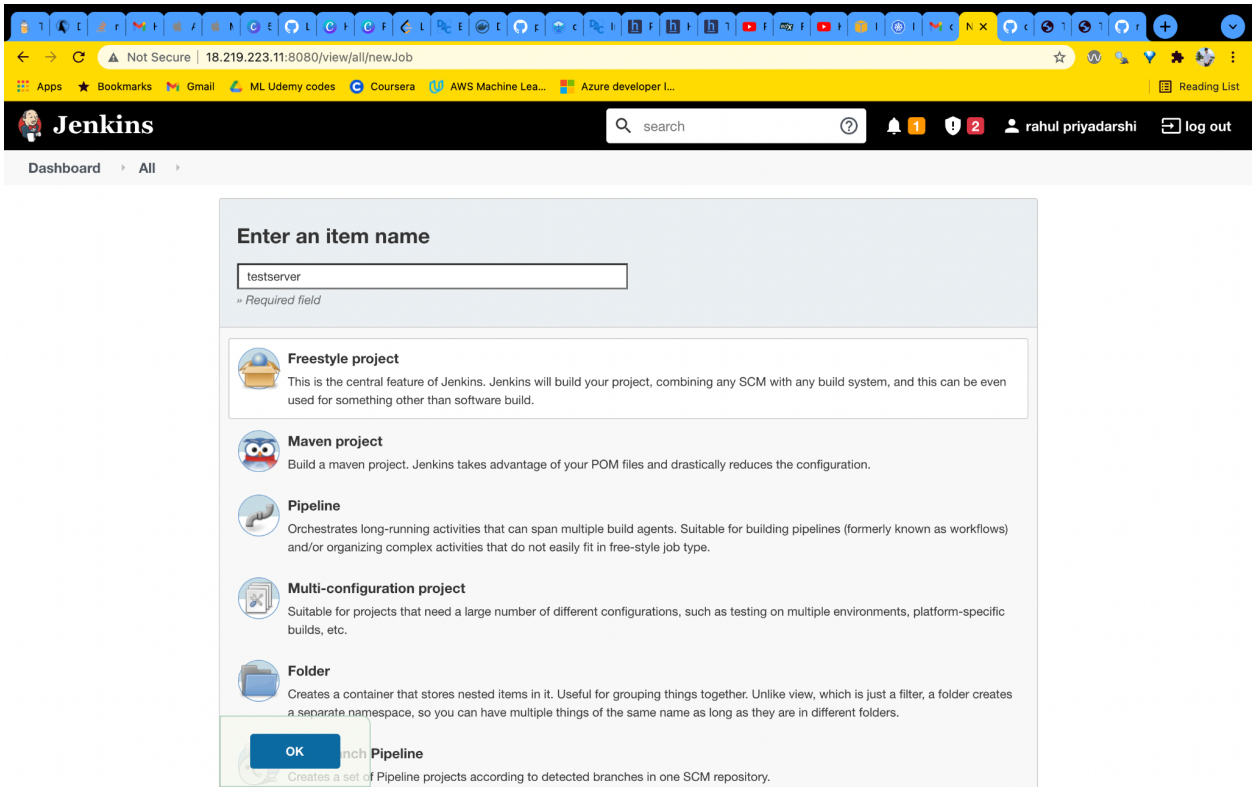
```
Processing triggers for libgdk-pixbuf2.0-0:amd64 (2.40.0+dfsg-3ubuntu0.2) ...
ubuntu@ip-172-31-43-2: $ java -jar agent.jar -jnlpUrl http://18.219.223.11:8080/computer/slave2/jenkins-agent.jnlp -secret d4176c315e2adc68332ee61a139ff6968d43a06418690c47b6a8a0edfaab6c10 -workDir "/home/
ubuntu/jenkins"
Nov 14, 2021 8:21:31 AM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFO: Using /home/ubuntu/jenkins/remoting as a remoting work directory
Nov 14, 2021 8:21:31 AM org.jenkinsci.remoting.engine.WorkDirManager setupLogging
INFO: Both error and output logs will be printed to /home/ubuntu/jenkins/remoting
Nov 14, 2021 8:21:31 AM hudson.remoting.jnlp.Main createEngine
INFO: Setting up agent: slave2
Nov 14, 2021 8:21:31 AM hudson.remoting.jnlp.Main$CuiListener <init>
INFO: Jenkins agent is running in headless mode.
Nov 14, 2021 8:21:31 AM hudson.remoting.Engine startEngine
INFO: Using Remoting version: 4.10.1
Nov 14, 2021 8:21:31 AM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFO: Using /home/ubuntu/jenkins/remoting as a remoting work directory
Nov 14, 2021 8:21:31 AM hudson.remoting.jnlp.Main$CuiListener status
INFO: Locating server among [http://18.219.223.11:8080/]
Nov 14, 2021 8:21:31 AM org.jenkinsci.remoting.engine.JnlpAgentEndpointResolver resolve
INFO: Remoting server accepts the following protocols: [JNLP4-connect, Ping]
Nov 14, 2021 8:21:31 AM hudson.remoting.jnlp.Main$CuiListener status
INFO: Agent discovery successful
  Agent address: 18.219.223.11
  Agent port:    34293
  Identity:      0e:3e:1a:f6:5a:cd:a1:67:5e:2a:72:62:2e:56:83:0d
Nov 14, 2021 8:21:31 AM hudson.remoting.jnlp.Main$CuiListener status
INFO: Handshaking
Nov 14, 2021 8:21:31 AM hudson.remoting.jnlp.Main$CuiListener status
INFO: Connecting to 18.219.223.11:34293
Nov 14, 2021 8:21:31 AM hudson.remoting.jnlp.Main$CuiListener status
INFO: Trying protocol: JNLP4-connect
Nov 14, 2021 8:21:32 AM org.jenkinsci.remoting.protocol.impl.BIONetworkLayer$Reader run
INFO: Waiting for ProtocolStack to start.
Nov 14, 2021 8:21:32 AM hudson.remoting.jnlp.Main$CuiListener status
INFO: Remote identity confirmed: 0e:3e:1a:f6:5a:cd:a1:67:5e:2a:72:62:2e:56:83:0d
Nov 14, 2021 8:21:33 AM hudson.remoting.jnlp.Main$CuiListener status
INFO: Connected
```

Both the slave nodes will now be in sync with Jenkins master.

Step11: Now install docker on both the slave nodes.



Step12: In Jenkins dashboard, create a new Job by going to Jenkins Dashboard ⇒ Create New Job ⇒ enter Job Name ⇒ select Freestyle Project ⇒ click OK.

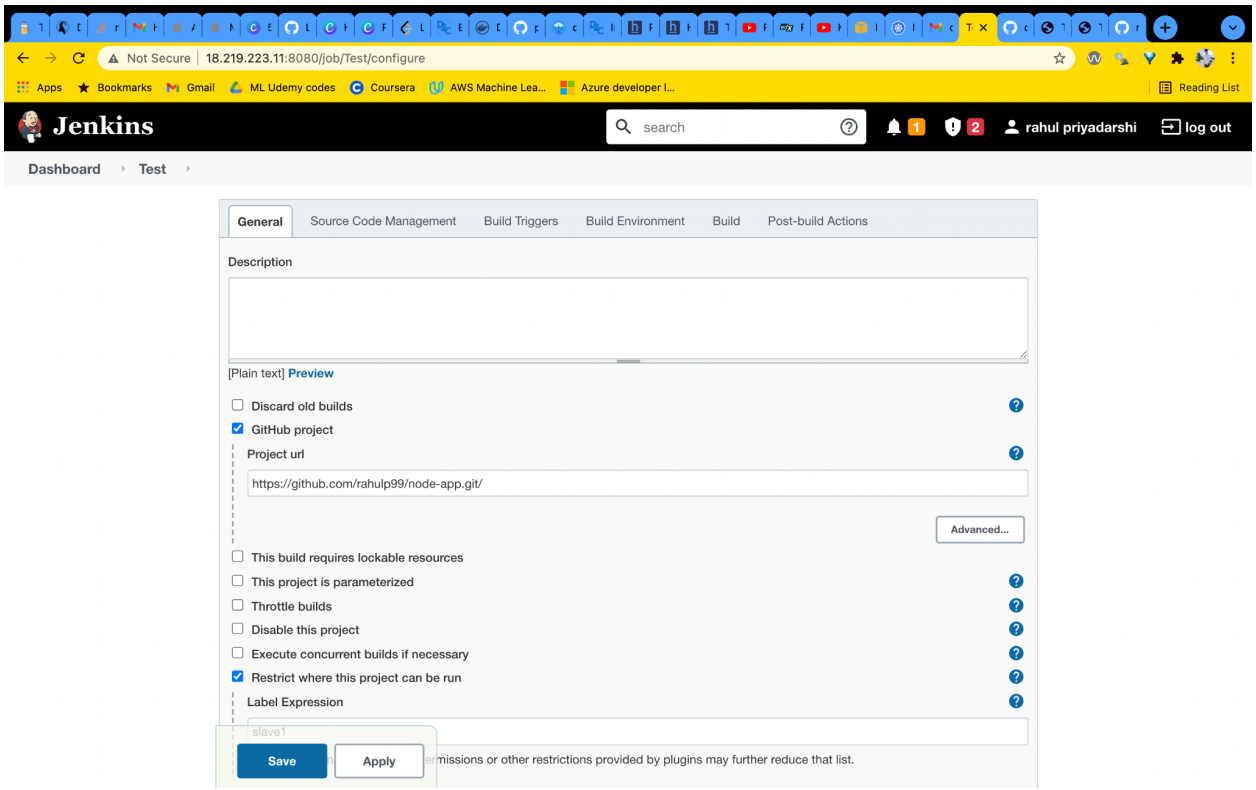


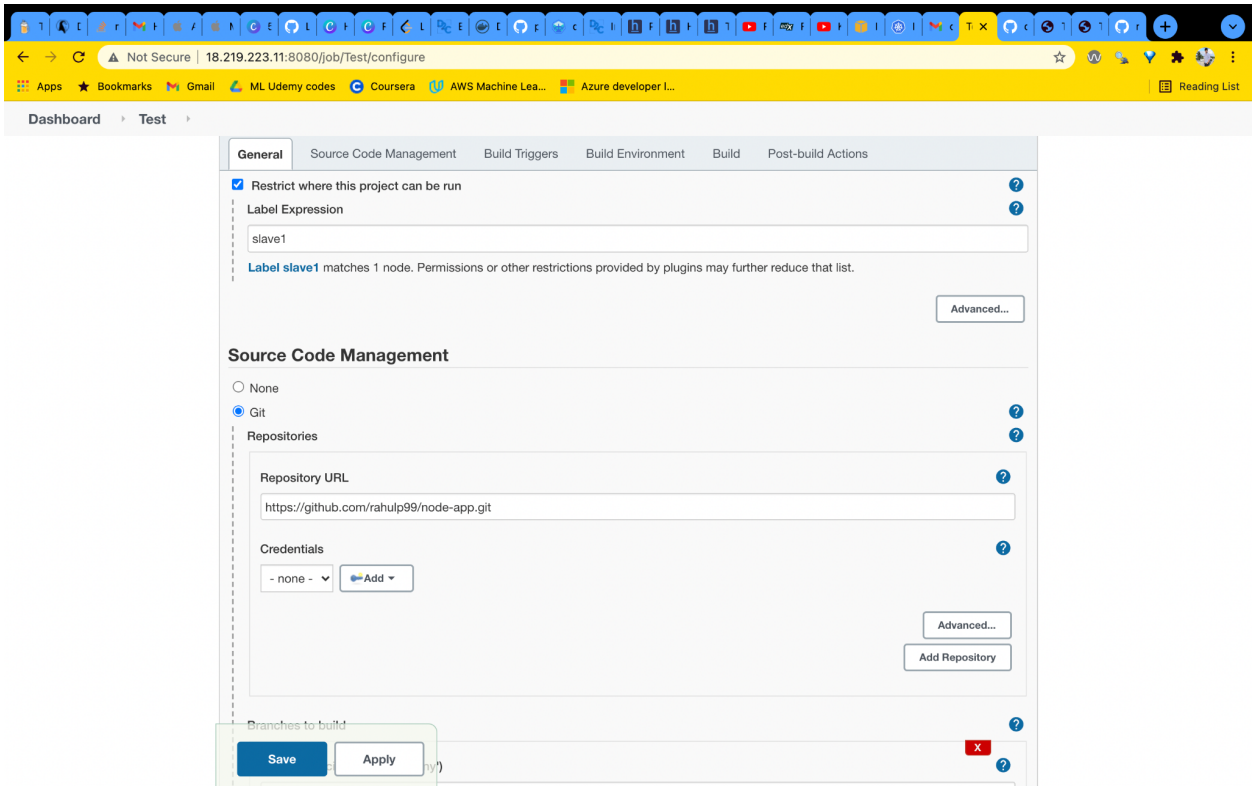
Step13: Now configure the new job by entering the details as follows:

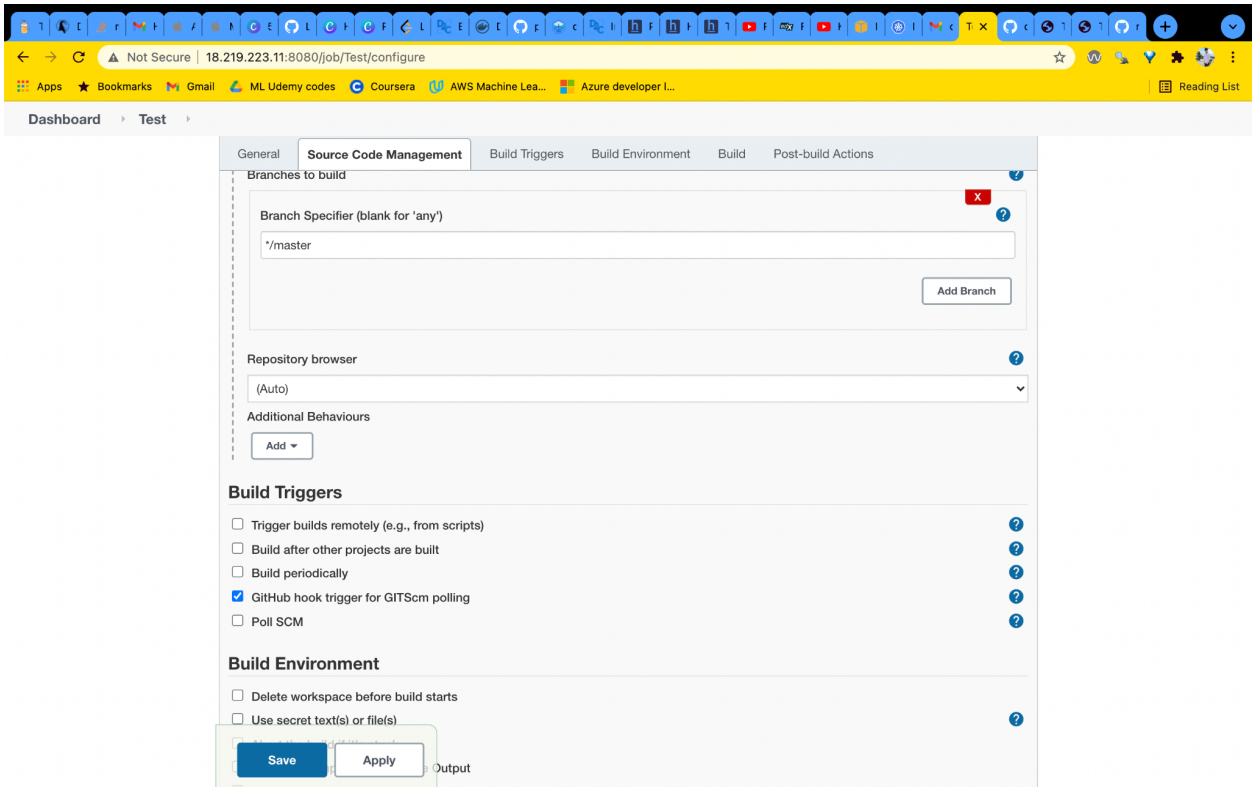
a. GitHub Project URL - https://github.com/rahulp99/node-app.git/
b. Select 'Restrict where this project can be run' and enter the appropriate node name.
c. In 'Source Code Management' select Git and enter the repository URL mentioned above.
d. Select 'GitHub hook trigger for GITScm polling' under 'Build Triggers'. (This step is only for the testing server).

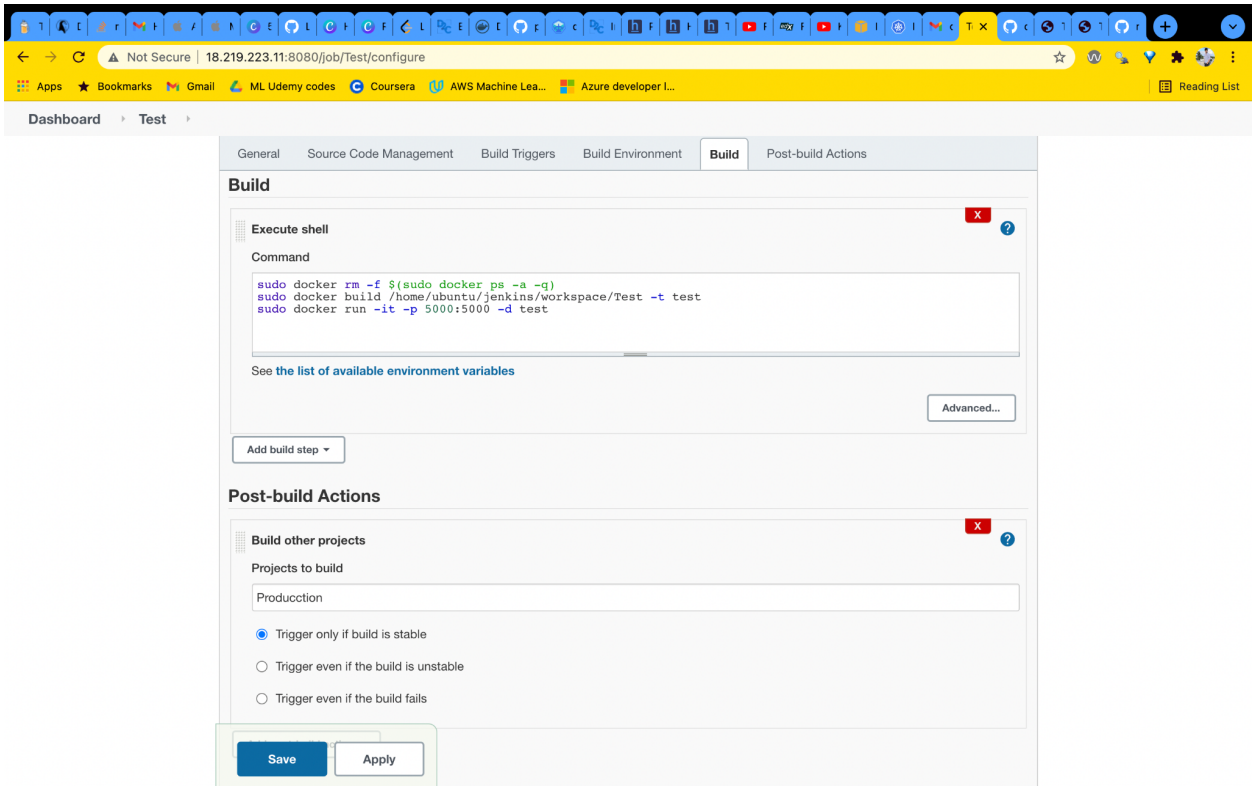
e. Under ‘Build’ select ‘Execute shell’ and enter the commands as shown in the screenshot below.
f. Under ‘Post-build Actions’ select ‘Build other projects’ and enter the production node name. Then select ‘Trigger only if build is stable’. (This step should be done only on the testing server).
g. Click Save.

This step should be done for both testing and production servers.

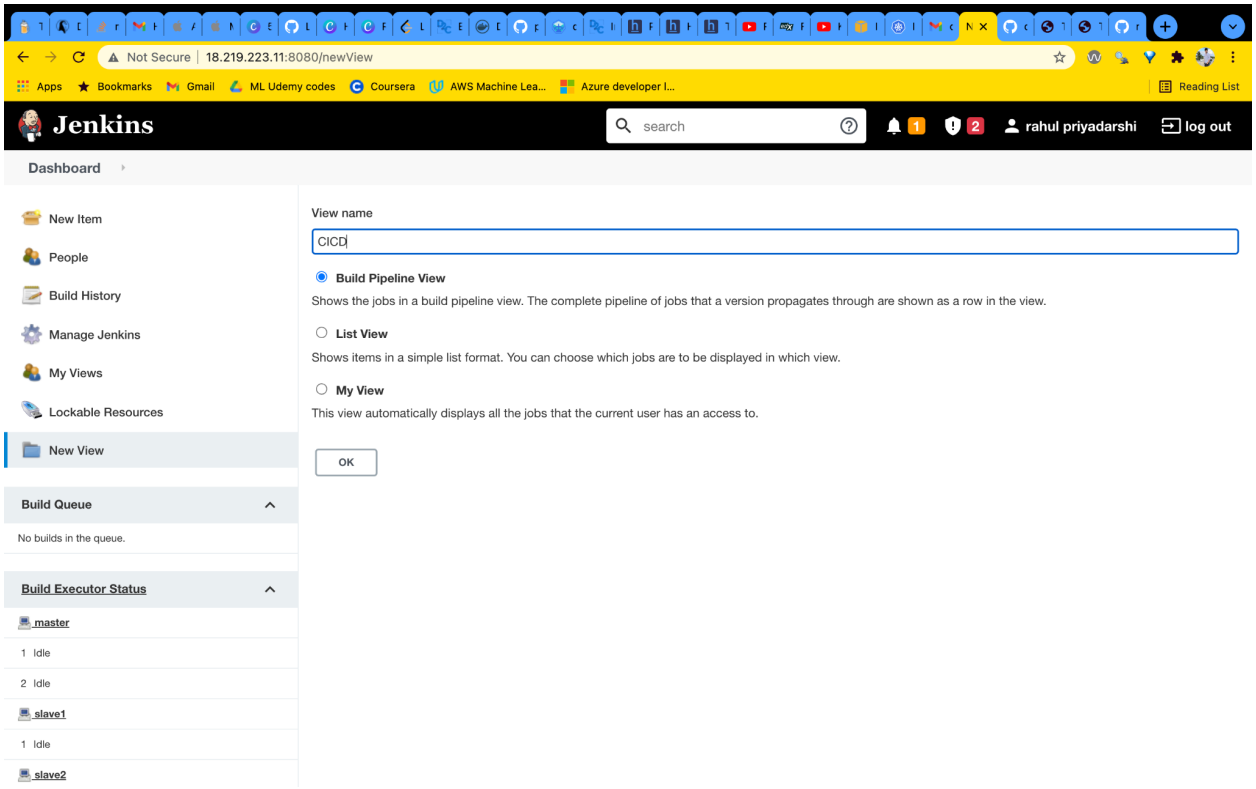



Not Secure | 18.219.223.11:8080/job/Test/configure
Apps
Bookmarks
Gmail
ML Udemy codes
Coursera
AWS Machine Lea...
Azure developer I...
Reading List
Dashboard
Test
General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
Restrict where this project can be run
Label Expression
slave1
Label slave1 matches 1 node. Permissions or other restrictions provided by plugins may further reduce that list.
Advanced...
Source Code Management
None
Git
Repositories
Repository URL
https://github.com/rahulp99/node-app.git
Credentials
- none -
Add
Advanced...
Add Repository
Branches to build
Save
Apply

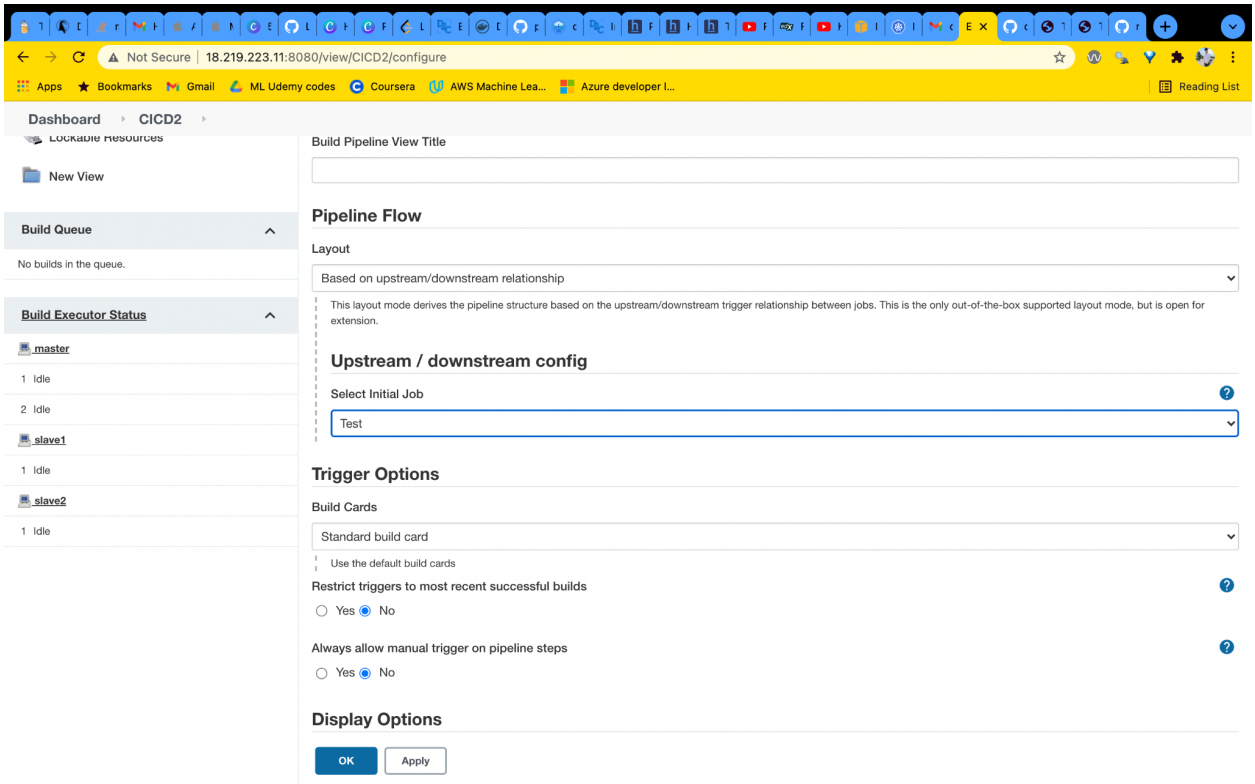

Not Secure | 18.219.223.11:8080/job/Test/configure
Apps
Bookmarks
Gmail
ML Udemy codes
Coursera
AWS Machine Lea...
Azure developer I...
Reading List
Dashboard
Test
General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
Branches to build
Branch Specifier (blank for 'any')
*/master
Add Branch
Repository browser
(Auto)
Additional Behaviours
Add
Build Triggers
Trigger builds remotely (e.g., from scripts)
Build after other projects are built
Build periodically
GitHub hook trigger for GITScm polling
Poll SCM
Build Environment
Delete workspace before build starts
Use secret text(s) or file(s)
Save
Apply


Not Secure | 18.219.223.11:8080/job/Test/configure
Apps
Bookmarks
Gmail
ML Udemy codes
Coursera
AWS Machine Lea...
Azure developer l...
Reading List
Dashboard
Test
General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
Build
Execute shell
Command
sudo docker rm -f $(sudo docker ps -a -q)
sudo docker build /home/ubuntu/jenkins/workspace/Test -t test
sudo docker run -it -p 5000:5000 -d test
See the list of available environment variables
Advanced...
Add build step
Post-build Actions
Build other projects
Projects to build
Producction
Trigger only if build is stable
Trigger even if the build is unstable
Trigger even if the build fails
Save
Apply

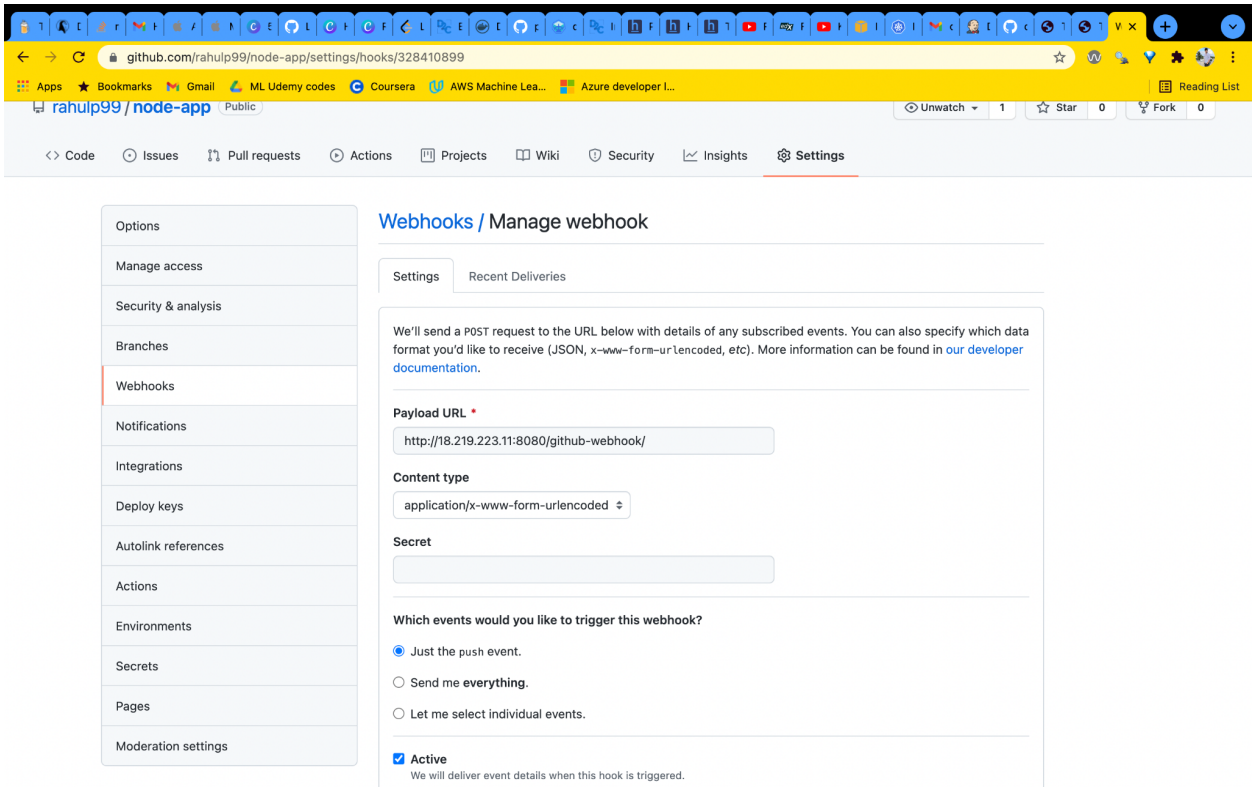
Step14: Now setup the Pipeline view. First install the 'build pipeline' plugin in Jenkins. Then go to Home and select the '+' icon beside 'All'. Then select 'Build Pipeline View' and give a name to the view (CICD here). Then click on OK.



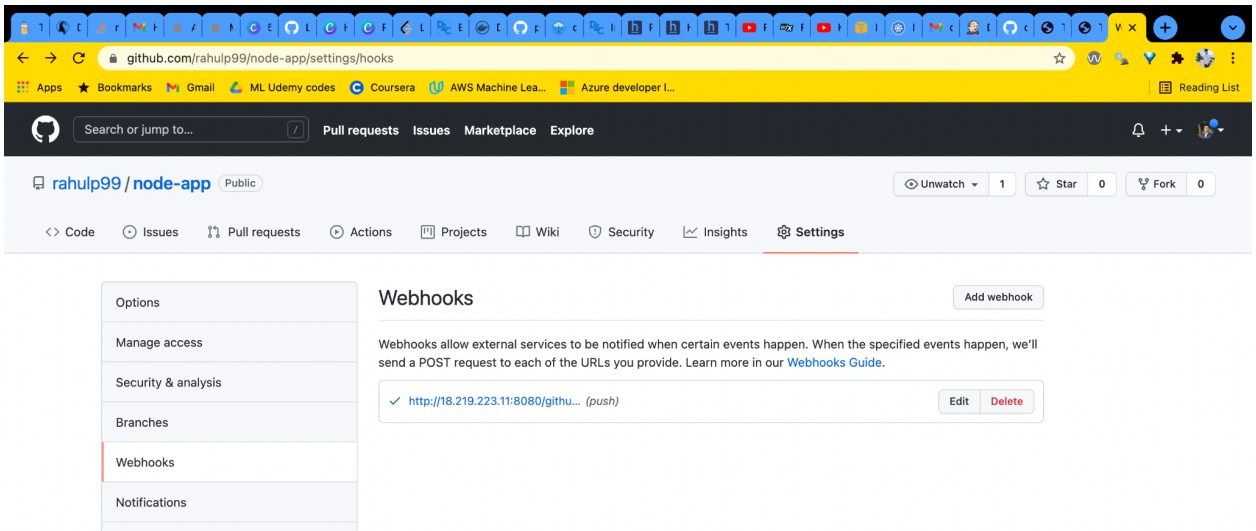
Step15: Now under ‘Pipeline Flow’, select initial job as ‘Test’. Then click on OK.



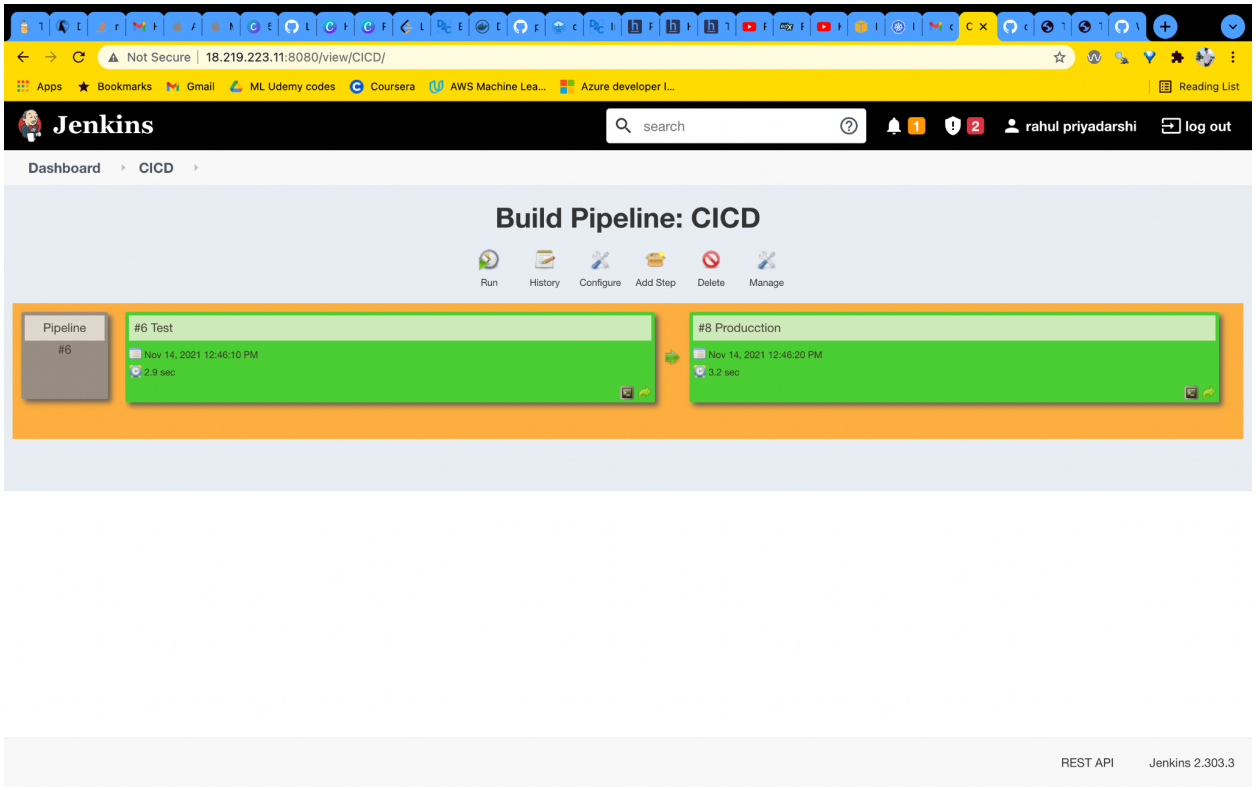
Step16: Now open the GitHub repository and go to ‘Settings’. Then select ‘Webhooks’. Click on ‘Add webhook’. Then enter the ‘Payload URL’ as ‘http://{ip-address-of-the-Jenkins-master-node}:8080/github-webhook/’ and select ‘Just the push event’ and then click on ‘Add webhook’.



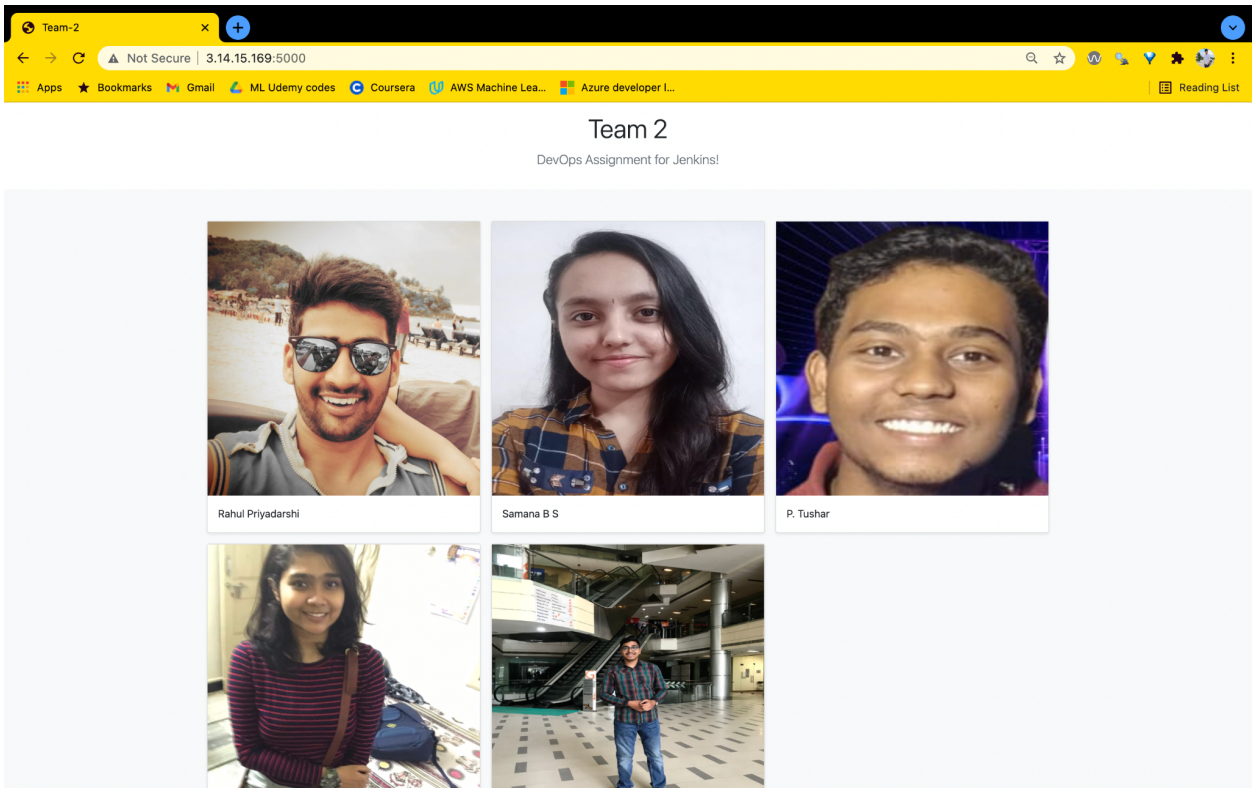
After successful ping, the webhook screen will look as follows:



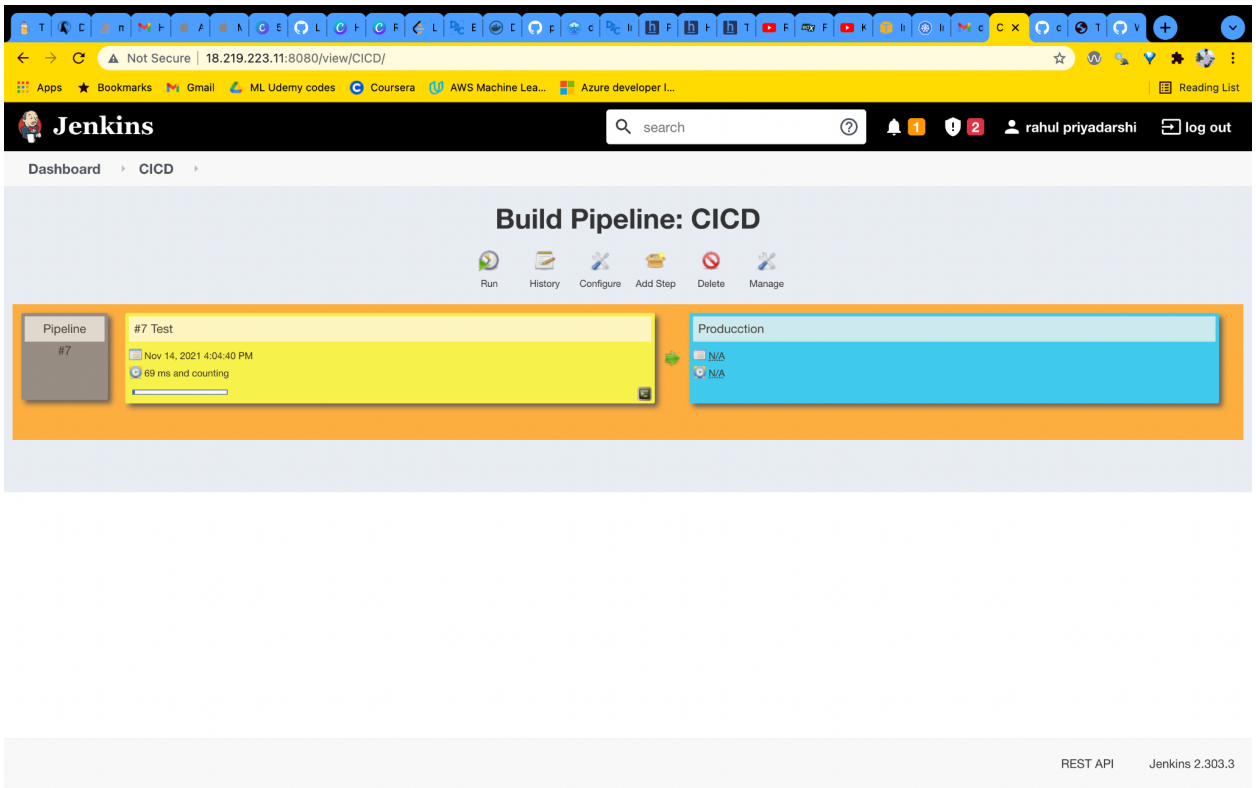
Step17: Now commit changes and push to the repository to trigger the webhook. This will trigger the Jenkins CICD pipeline and build the docker image on the test server first, and on successful build, it will build the same on the production server as well.
CICD pipeline looks as follows:

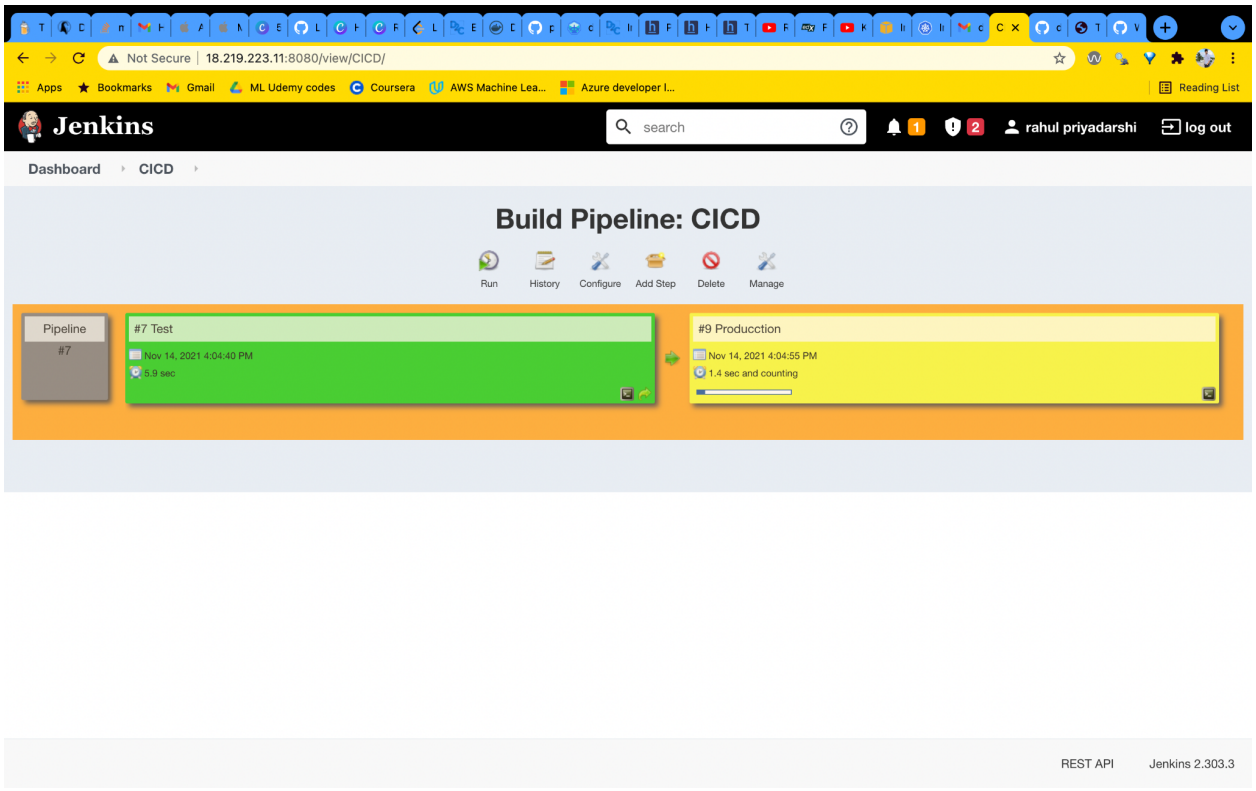


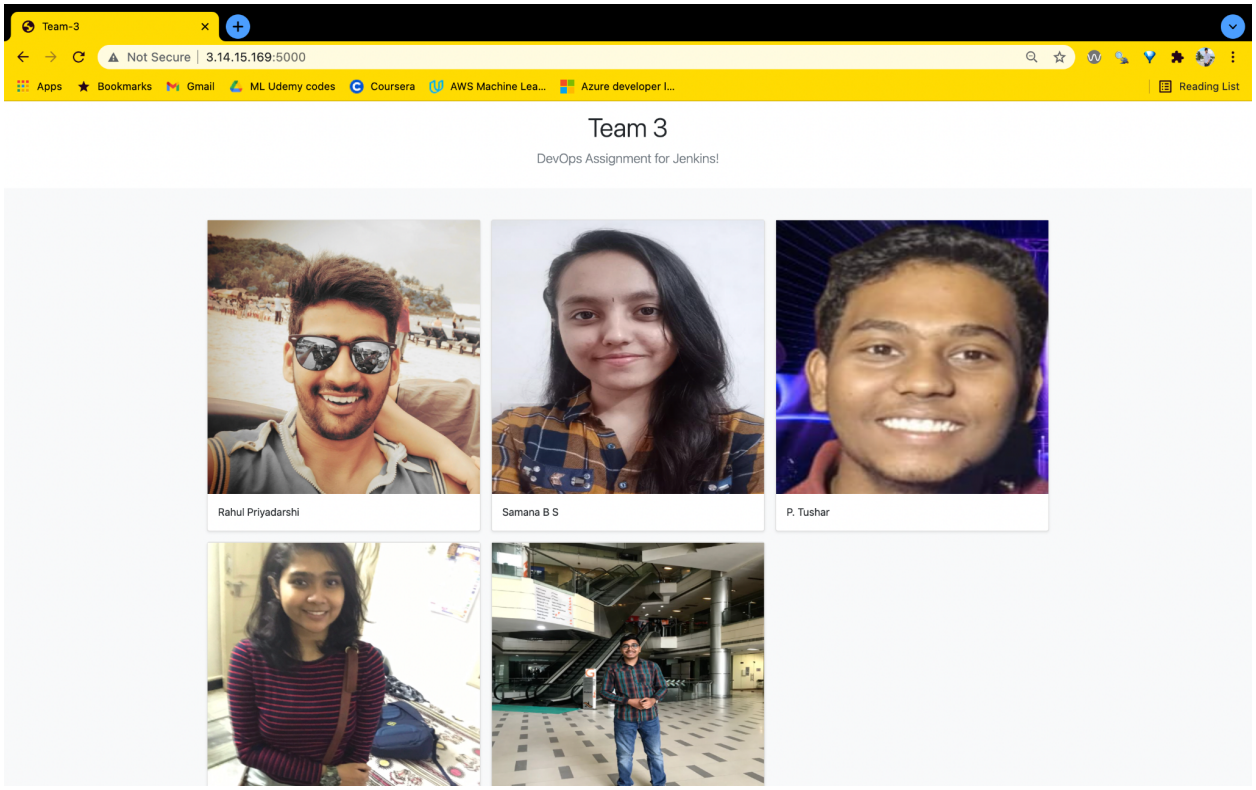
On successful build, the deployed Node application looks as follows on both testing and production servers:



Step18: Here we notice that the Team number entered in the website is Incorrect. It should be ‘Team 3’ and not ‘Team 2’. So we make the correction in the code and push it to the GitHub repository. This triggered the pipeline and changes reflect in the deployed website as shown below:







Changes reflected. Hence, we created a CI/CD pipeline successfully with Jenkins.