INDIAN INSTITUTE OF INFORMATION TECHNOLOGY

DevOps & its Applications (CS457)

Jenkins Assignment 2

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

Submitted by -

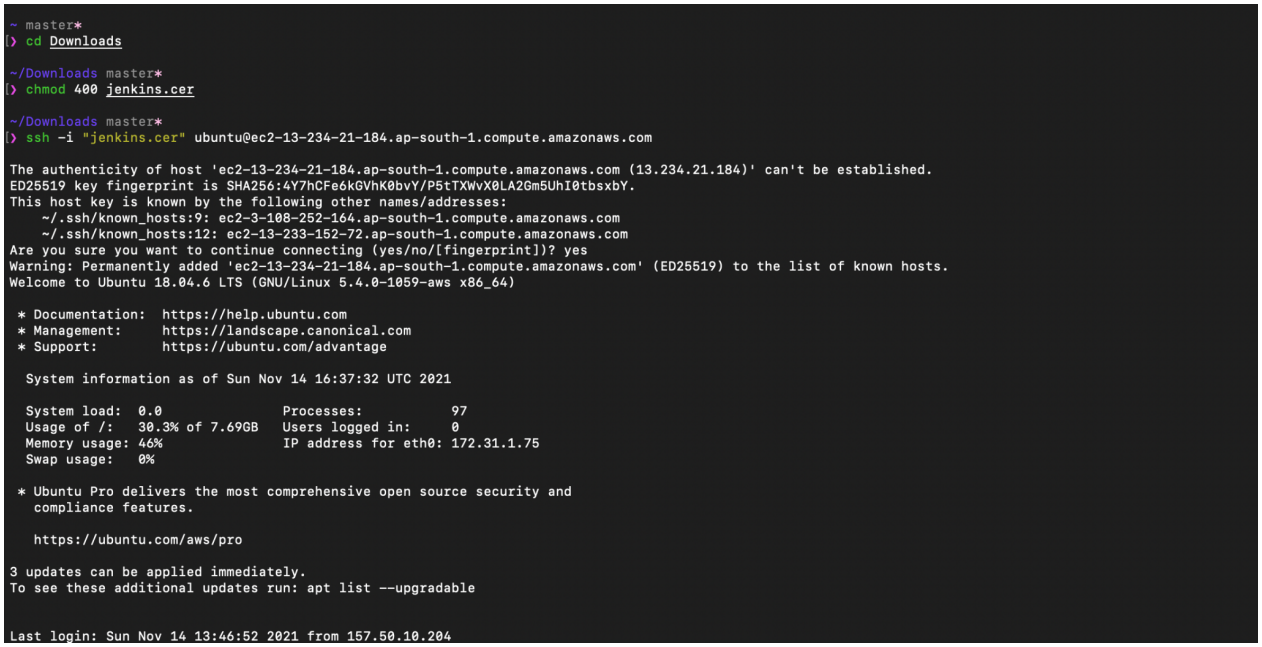
18BCS006-Anjuru Lokesh
18BCS021-Avinash c
18BCS025-Diddi Geethakrishna
18BCS028-Godina Pranav
18BCS035-Kancharla Gireesh Kumar
18BCS070-Pravalika

JENKINS MASTER - SLAVE PIPELINE

Tech Stack:

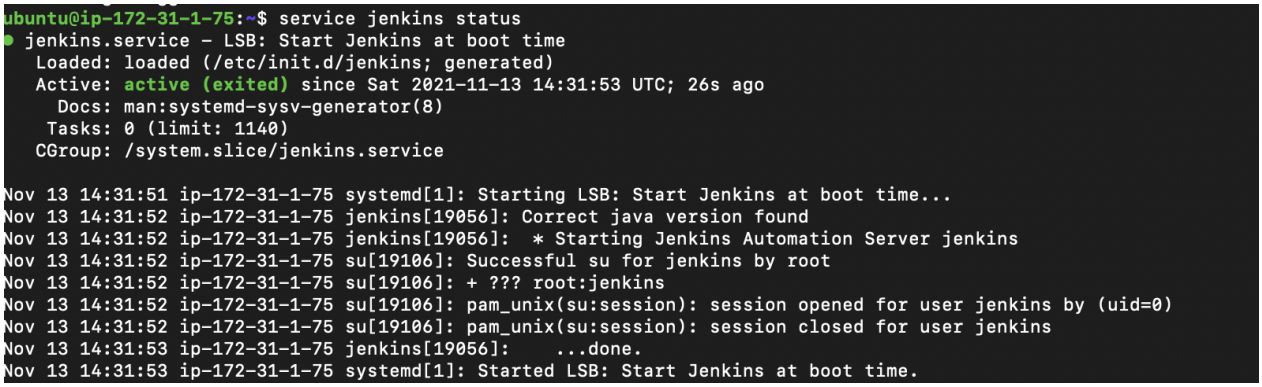
1. Git
2. Github
3. Docker
4. Jenkins
5. Three AWS EC2 Instances (t2.micro)

SETTING UP JENKINS MASTER

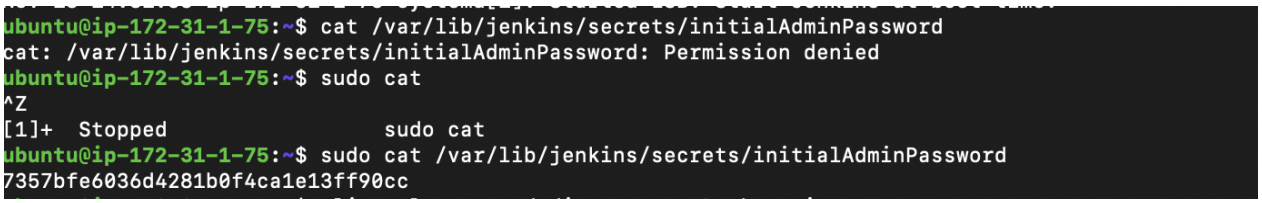


Step1: Install Jenkins on an EC2 instance with the commands mentioned below in the screenshot.

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



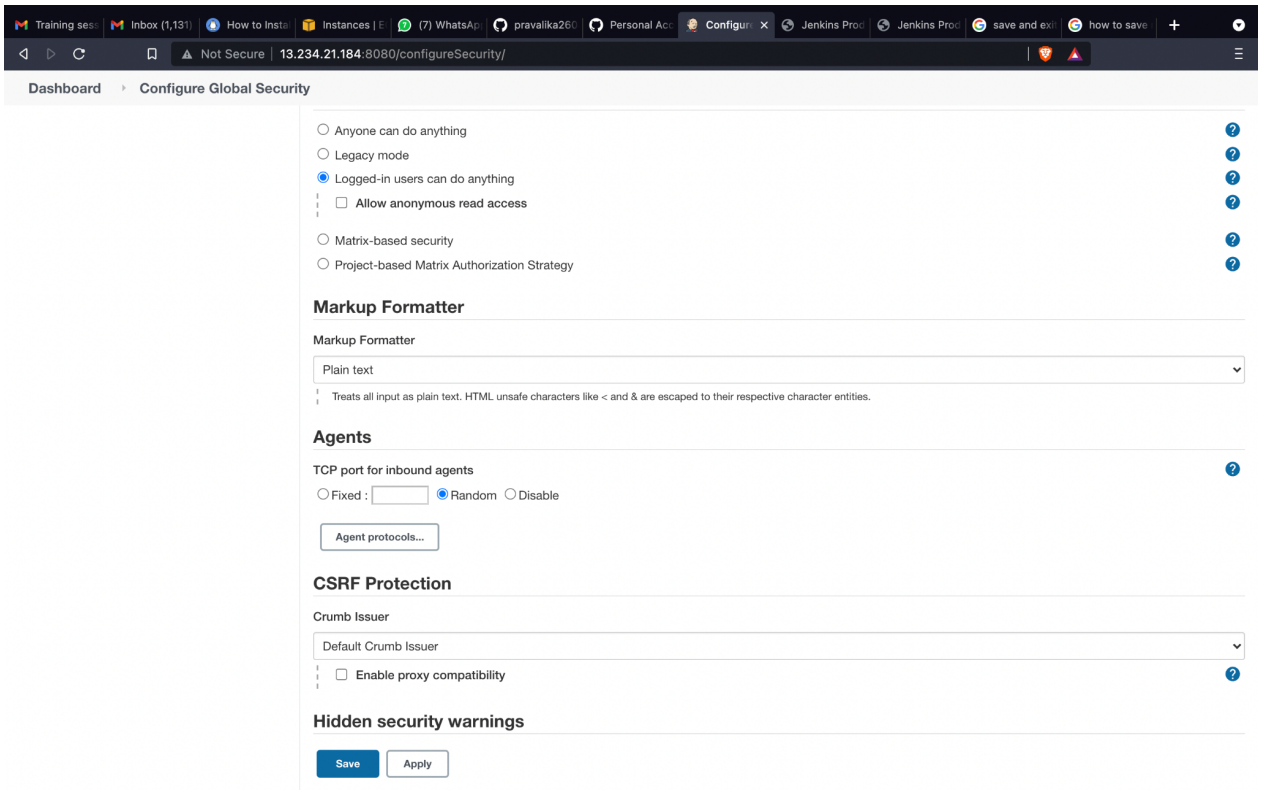
Step3: Now enter the public IP address of the EC2 instance followed by port number 8080 (by default, Jenkins runs on port 8080) in a browser to access the Jenkins Dashboard. Here, it will prompt you to enter the Admin Password to unlock Jenkins. The Admin Password can be found in the file whose absolute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

Step5: Name the other two EC2 instances as ‘slave1’ and ‘slave2’

Go to jenkins -> manage jenkins -> Configure global security. In agents, change TCP port for inbound agents to random and save.

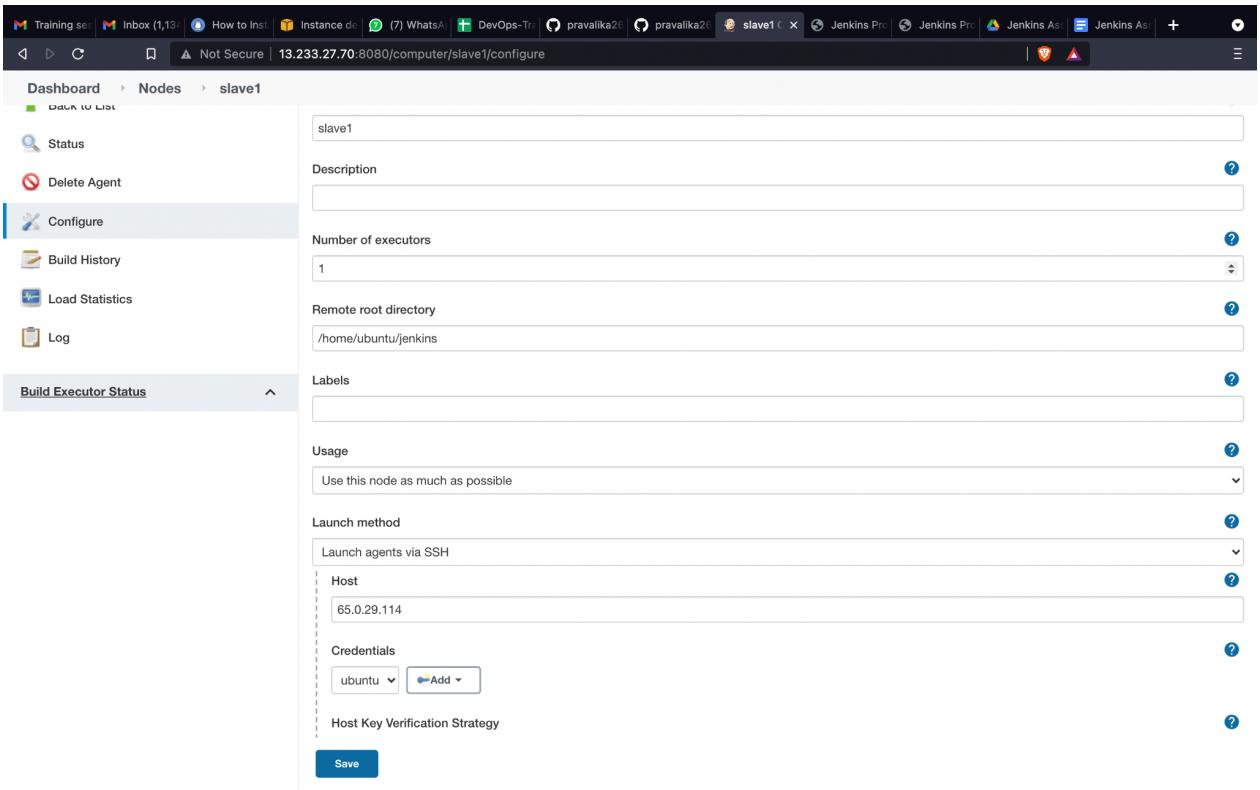


Step6: Connect slave1 and slave2 nodes to Jenkins master.

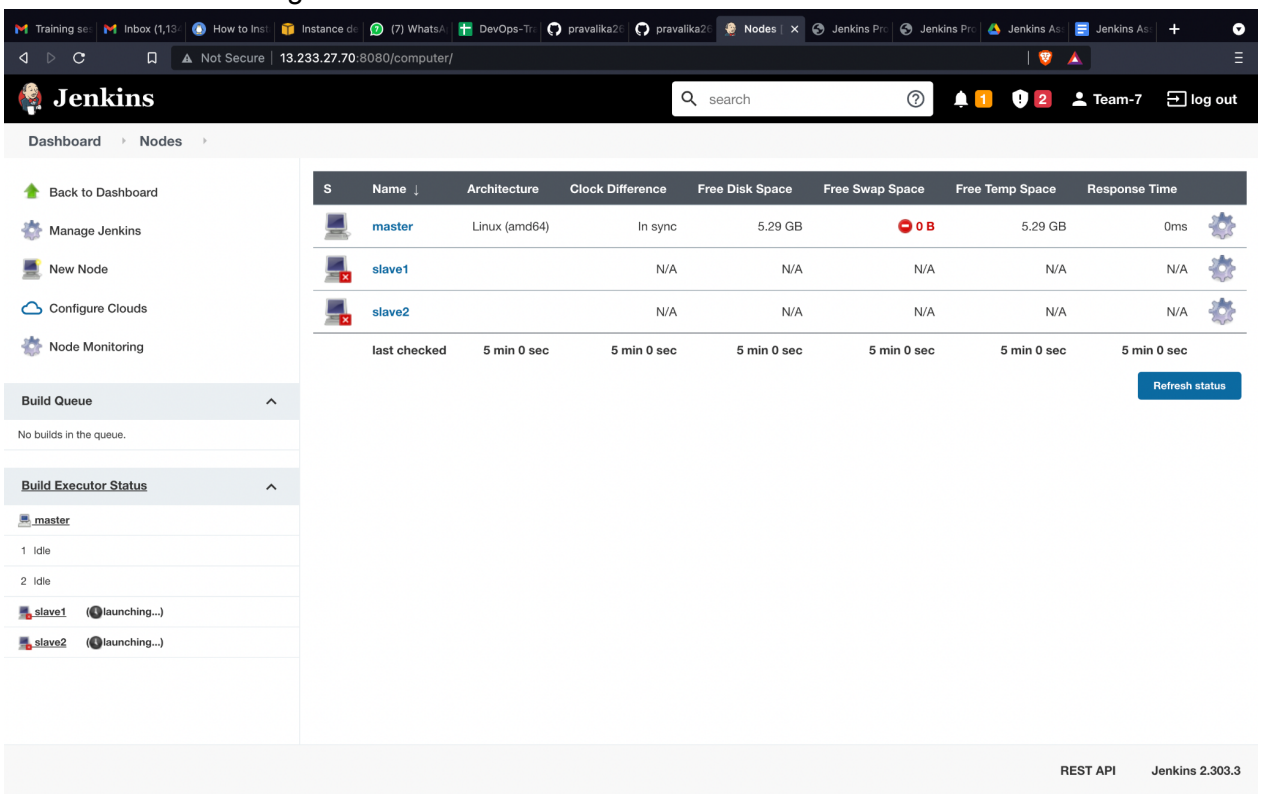
Go to Manage Jenkins -> Manage Nodes and Clouds -> New Node.

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’ and select ‘connect via SSH’. Add ip address of respective instance of the node and enter the credentials as mentioned in the screenshot below and click Save.



This is how the ‘Manage Nodes’ dashboard looks like:



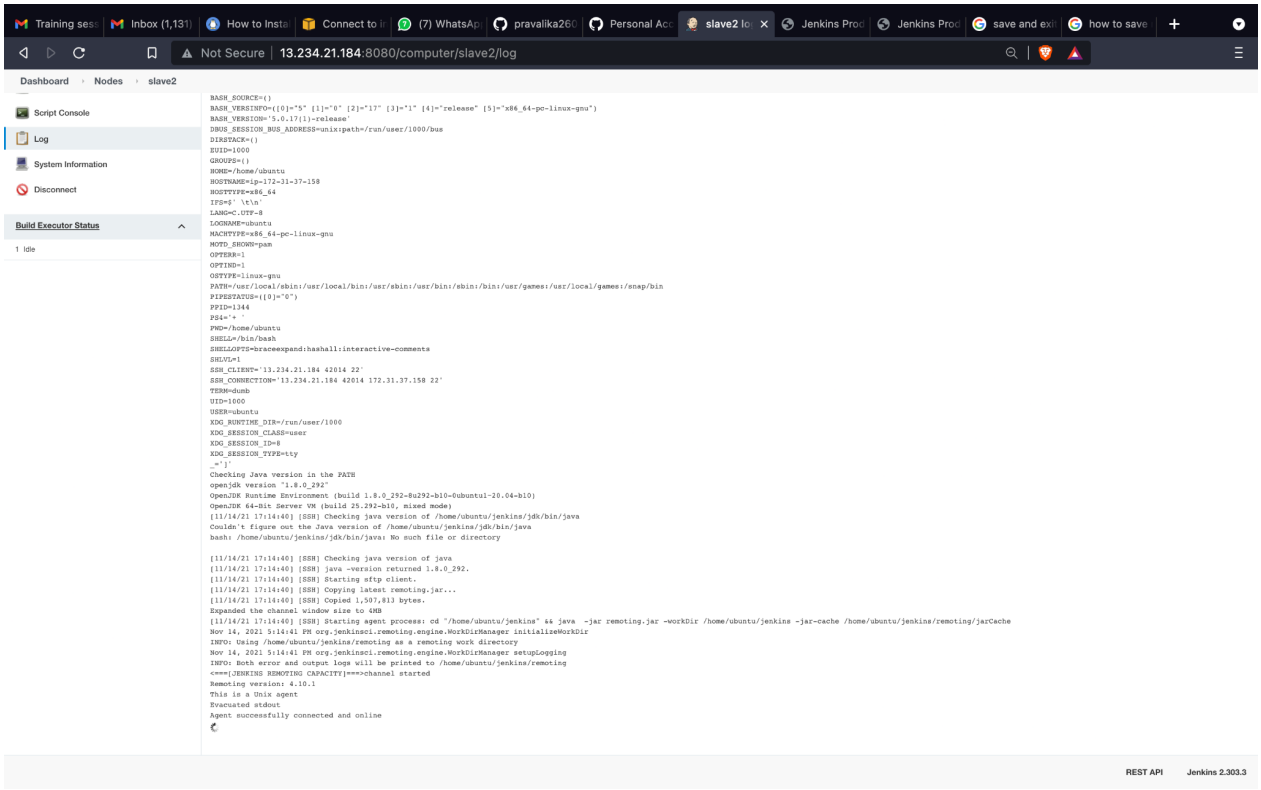
Step8: Go to master EC2 instance and run the following commands

- ssh-keygen -t rsa
  Generates the key pair
- cd ~/.ssh
- ls
- cat id_rsa.pub
  This is the public key

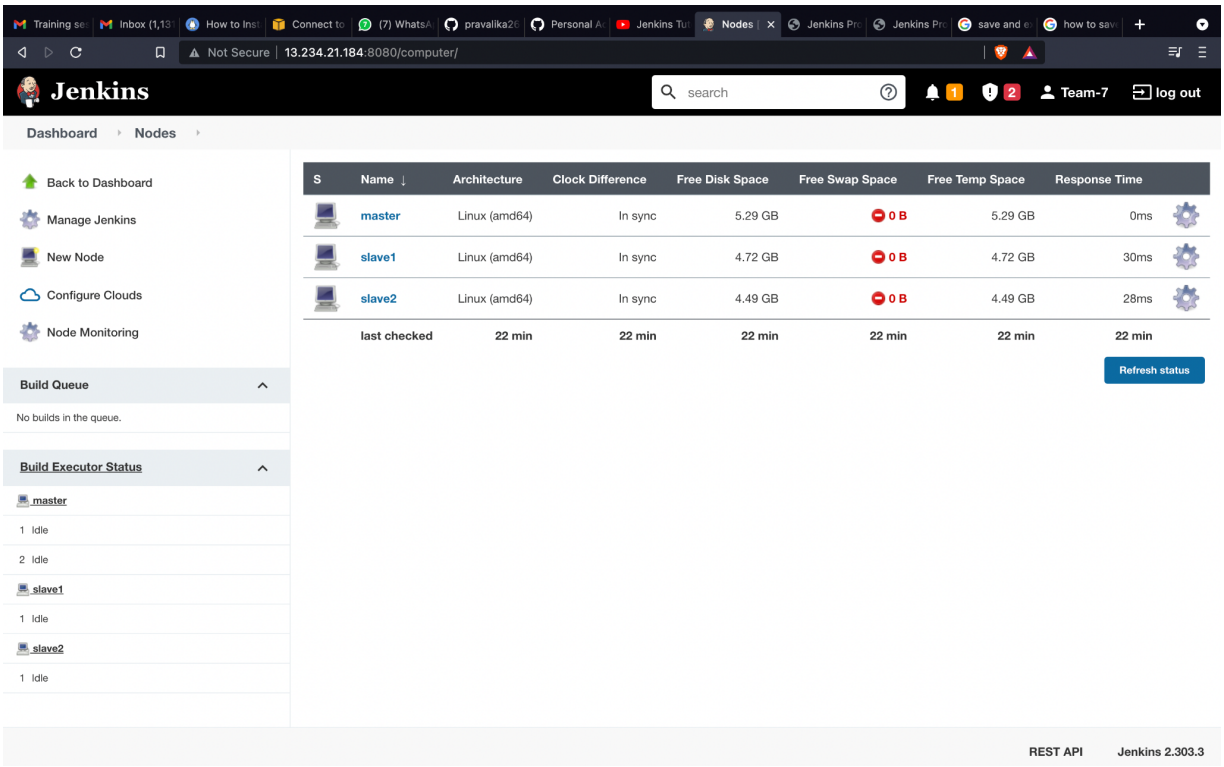
Go to both slave1 and salve2 Ec2 instances and run the command

- echo”Public key”>>authorized_keys
- sudo chmod 777 -R jenkins
- Cd ~/.ssh
- ls
- cat id_rsa
  The private key is copied and added in the credentials of salve1, salve2 configuration.

  Agent is successfully connected.



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



Step9: Node install JDK on both the slave nodes.

Step10: Now install docker on both the slave nodes.

```
Last login: Sun Nov 14 17:25:06 2021 from 157.50.10.204
ubuntu@ip-172-31-35-160:~$ sudo apt-get install docker.io
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base libidn11 pigz runc
  ubuntu-fan
Suggested packages:
  ifupdown aufs-tools cgroupfs-mount | cgroup-lite debootstrap docker-doc
  rinse zfs-fuse | zfsutils
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io libidn11 pigz
  runc ubuntu-fan
0 upgraded, 9 newly installed, 0 to remove and 24 not upgraded.
Need to get 74.5 MB of archives.
After this operation, 361 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 pigz amd64 2.4-1 [57.4 kB]
Get:2 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal/main amd64 bridge-utils amd64 1.6-2ubuntu1 [30.5 kB]
Get:3 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 runc amd64 1.0.1-0ubuntu2~20.04.1 [4155 kB]
Get:4 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 containerd amd64 1.5.5-0ubuntu3~20.04.1 [33.0 MB]
Get:5 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal/main amd64 dns-root-data all 2019052802 [5300 B]
Get:6 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal/main amd64 libidn11 amd64 1.33-2.2ubuntu2 [46.2 kB]
Get:7 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-updates/main amd64 dnsmasq-base amd64 2.80-1.1ubuntu1.4 [315 kB]
Get:8 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal-updates/universe amd64 docker.io amd64 20.10.7-0ubuntu5~20.04.2 [36.9 MB]
Get:9 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu focal/main amd64 ubuntu-fan all 0.12.13 [34.5 kB]
Fetched 74.5 MB in 18s (4144 kB/s)
Preconfiguring packages ...
Selecting previously unselected package pigz.
(Reading database ... 79377 files and directories currently installed.)
Preparing to unpack .../0-pigz_2.4-1_amd64.deb ...
Unpacking pigz (2.4-1) ...
Selecting previously unselected package bridge-utils.
Preparing to unpack .../1-bridge-utils_1.6-2ubuntu1_amd64.deb ...
Unpacking bridge-utils (1.6-2ubuntu1) ...
Selecting previously unselected package runc.
Preparing to unpack .../2-runc_1.0.1-0ubuntu2~20.04.1_amd64.deb ...
Unpacking runc (1.0.1-0ubuntu2~20.04.1) ...
Selecting previously unselected package containerd.
Preparing to unpack .../3-containerd_1.5.5-0ubuntu3~20.04.1_amd64.deb ...
Unpacking containerd (1.5.5-0ubuntu3~20.04.1) ...
Selecting previously unselected package dns-root-data.
Preparing to unpack .../4-dns-root-data_2019052802_all.deb ...
Unpacking dns-root-data (2019052802) ...
Selecting previously unselected package libidn11:amd64.
Preparing to unpack .../5-libidn11_1.33-2.2ubuntu2_amd64.deb ...
Unpacking libidn11:amd64 (1.33-2.2ubuntu2) ...
Selecting previously unselected package dnsmasq-base.
Preparing to unpack .../6-dnsmasq-base_2.80-1.1ubuntu1.4_amd64.deb ...
Unpacking dnsmasq-base (2.80-1.1ubuntu1.4) ...
Selecting previously unselected package docker.io.
Preparing to unpack .../7-docker.io_20.10.7-0ubuntu5~20.04.2_amd64.deb ...
Unpacking docker.io (20.10.7-0ubuntu5~20.04.2) ...
Selecting previously unselected package ubuntu-fan.
Preparing to unpack .../8-ubuntu-fan_0.12.13_all.deb ...
Unpacking ubuntu-fan (0.12.13) ...
Setting up runc (1.0.1-0ubuntu2~20.04.1) ...
Setting up dns-root-data (2019052802) ...
Setting up libidn11:amd64 (1.33-2.2ubuntu2) ...
Setting up bridge-utils (1.6-2ubuntu1) ...
Setting up pigz (2.4-1) ...
Setting up containerd (1.5.5-0ubuntu3~20.04.1) ...
```

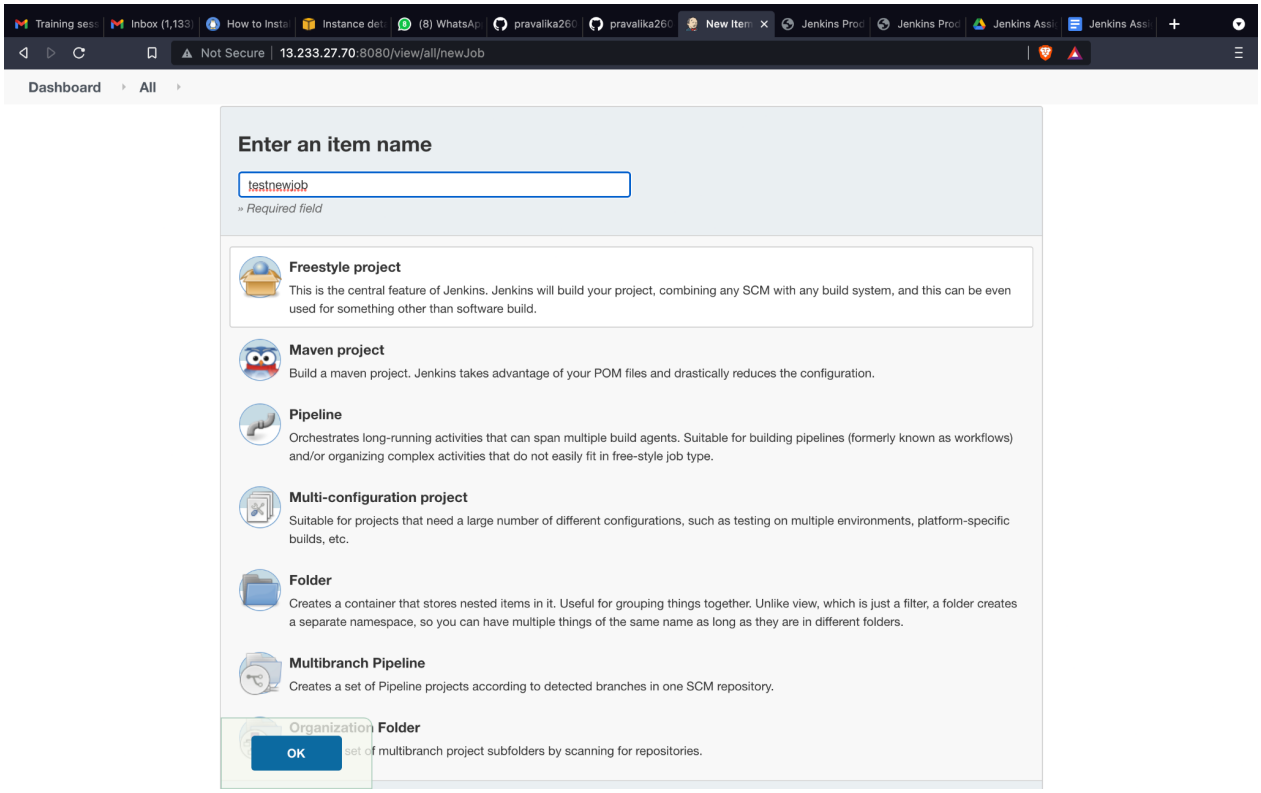
CI/CD JENKINS PIPELINE

We are treating salve1 as a testing server and slave2 as production server.

Configuring CICD pipeline in the way

- Configure webhook that triggers the slave1 job.
- When the slave1 job is successful, only then slave2 job is called.

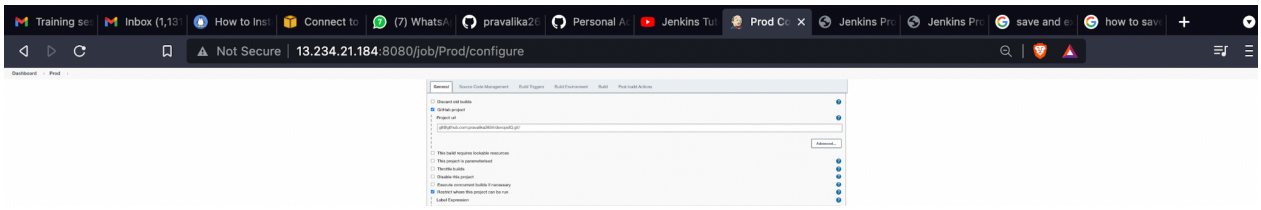
Step11: In Jenkins dashboard, create a new Job by going to Jenkins Dashboard ⇒ Create New Job ⇒ enter Job Name ⇒ select Freestyle Project ⇒ click OK.

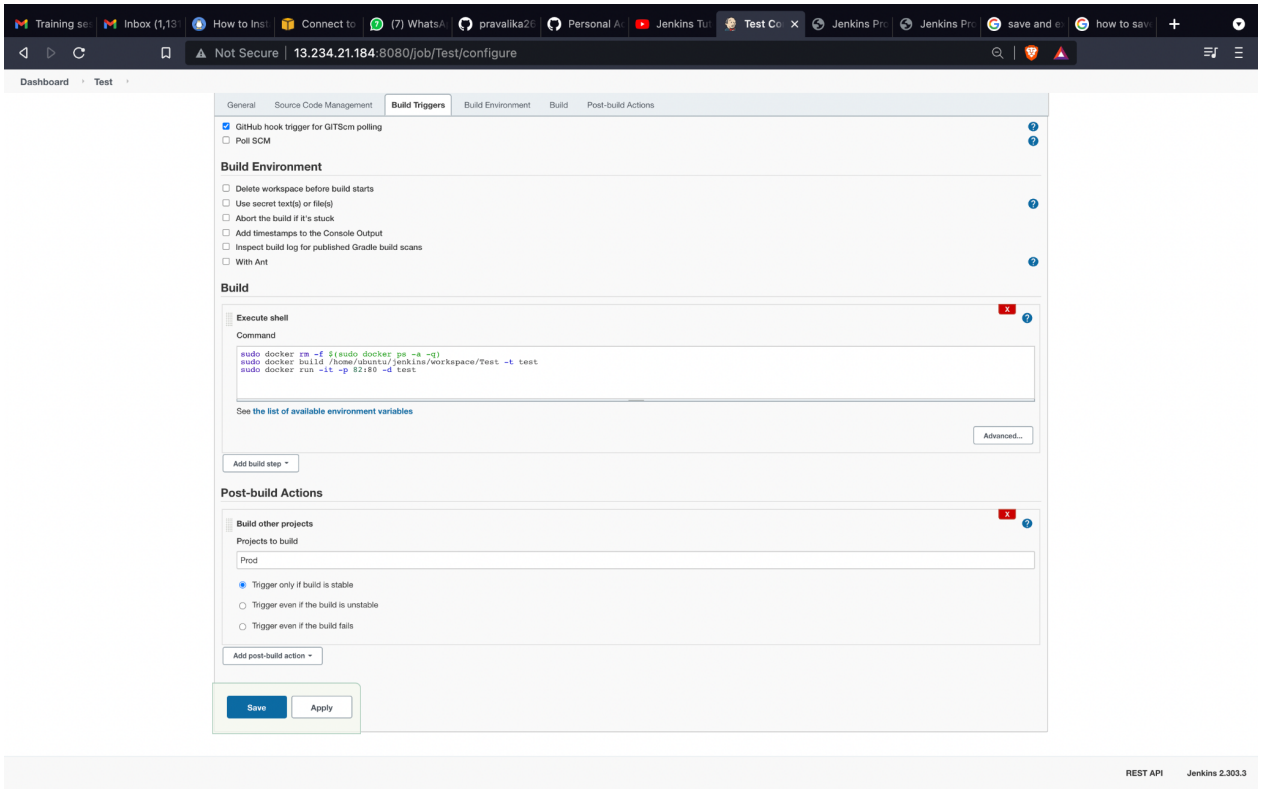


Step12: Now configure the new job by entering the details as follows:

- GitHub Project URL - git@github.com:pravalika2604/devopsIQ.git
- Select 'Restrict where this project can be run' and enter the appropriate node name.
- In 'Source Code Management' select Git and enter the repository URL mentioned above.
- Select 'GitHub hook trigger for GITScm polling' under 'Build Triggers'. (This step is only for the testing server).
- Under 'Build' select 'Execute shell' and enter the commands as shown in the screenshot below.
- Under 'Post-build Actions' select 'Build other projects' and enter the prod node name. Then select 'Trigger only if build is stable'. (This step should be done only on the testing server).
- Click Save.

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

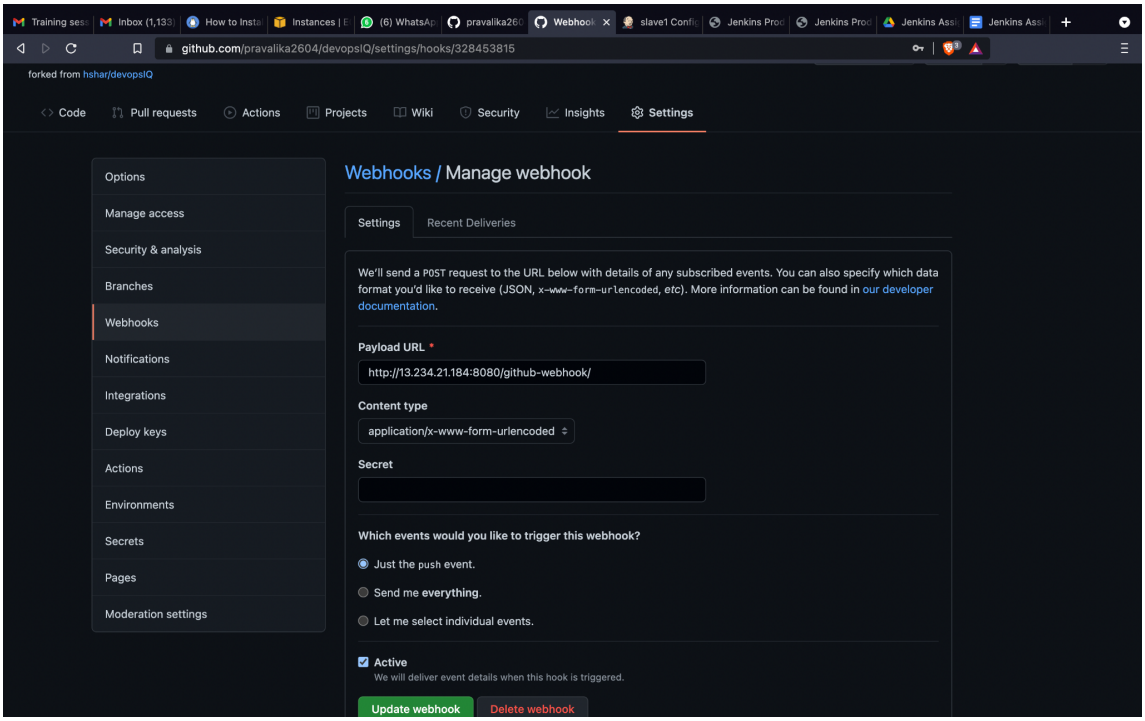




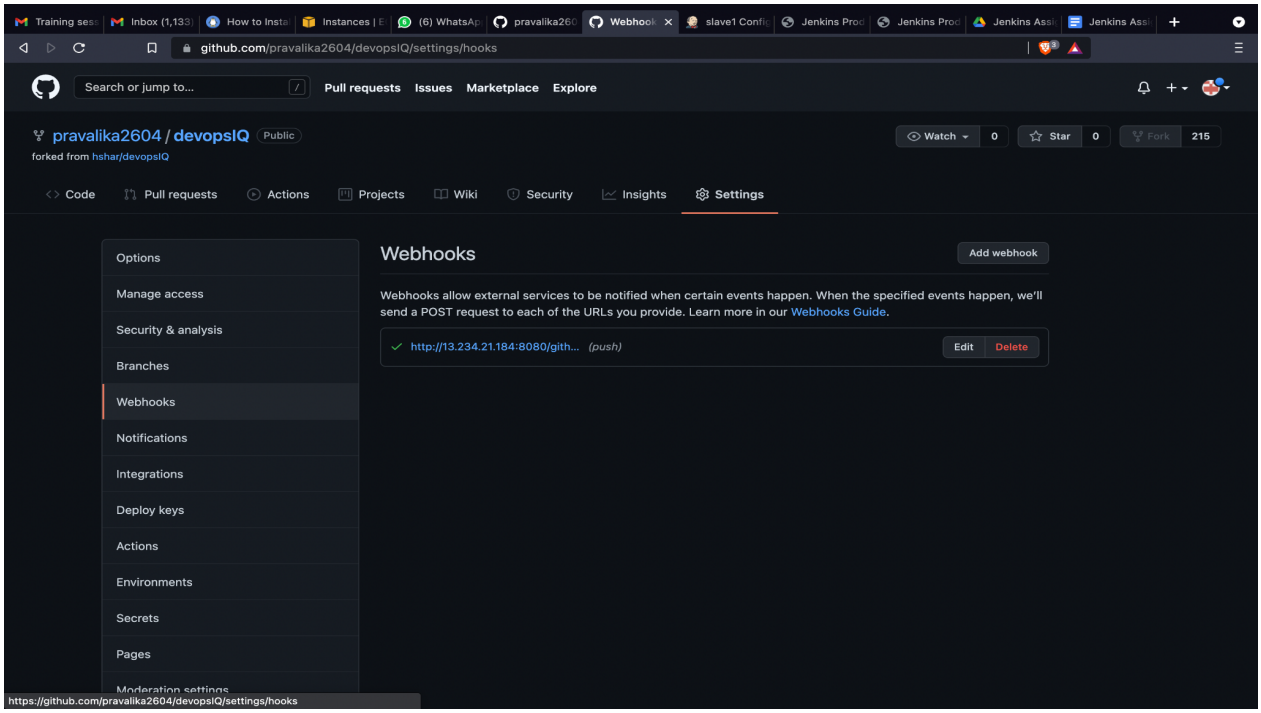
Step13: Now set up the Pipeline view. First install the 'build pipeline' plugin in Jenkins. Then go to Home and select the '+' icon beside 'All'. Then select 'Build Pipeline View' and give a name to the view (CICD here). Then click on OK.

Step14: Now under 'Pipeline Flow', select the initial job as 'Test'. Then click on OK.

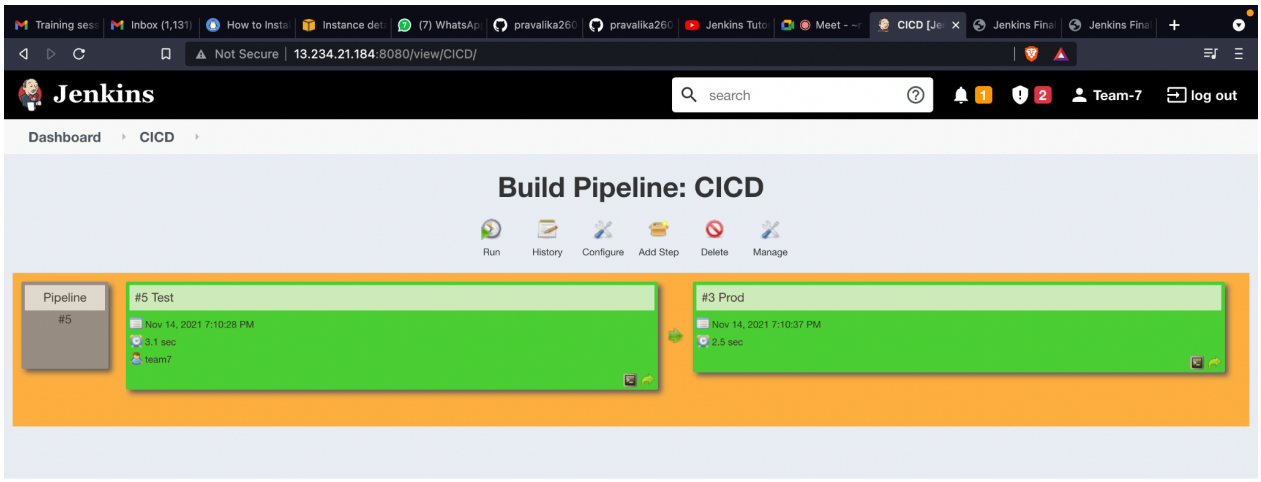
Step15: Now open the GitHub repository and go to 'Settings'. Then select 'Webhooks'. Click on 'Add webhook'. Then enter the 'Payload URL' as 'http://{ip-address-of-the-Jenkins-master-node}:8080/github-webhook/' and select 'Just the push event' and then click on 'Add webhook'.



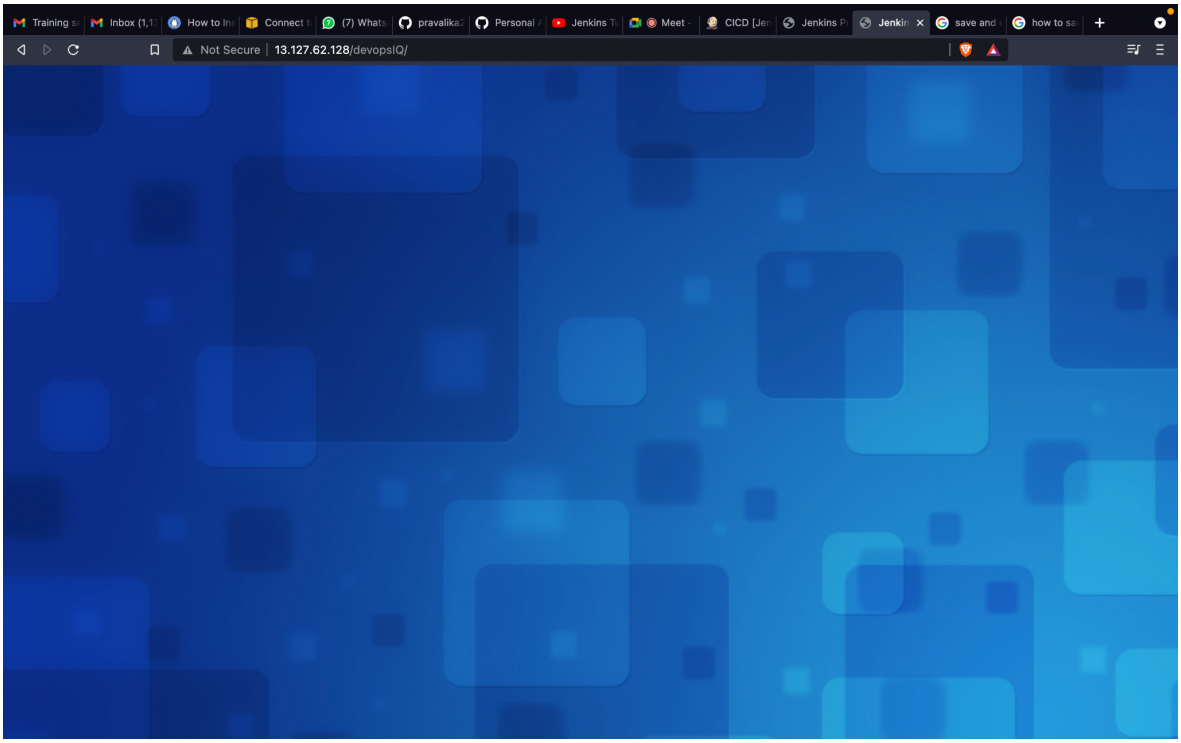
After successful ping, the webhook screen will look as follows:



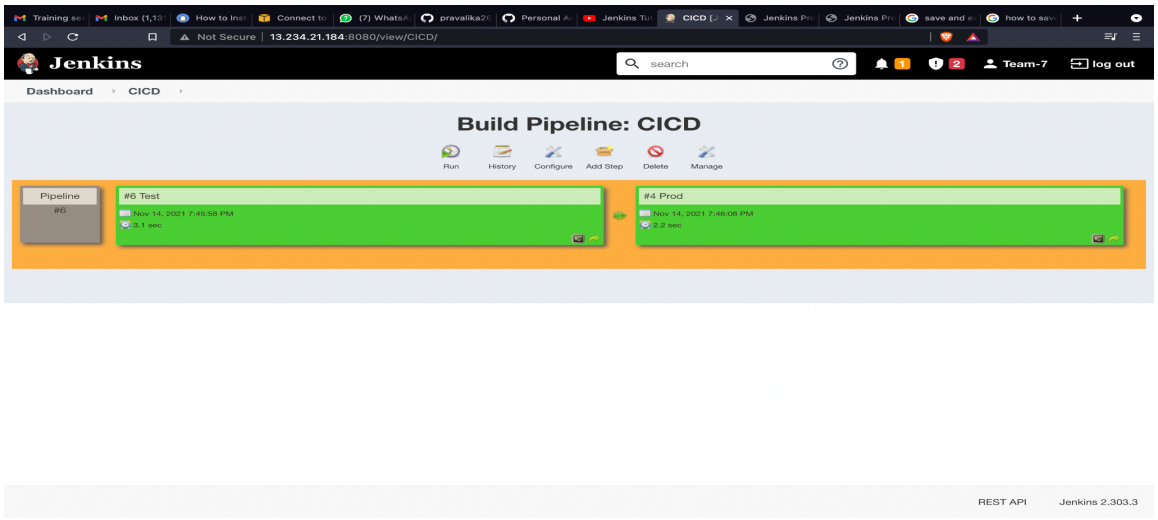
Step16: Now commit changes and push to the repository to trigger the webhook. This will trigger the Jenkins CICD pipeline and build the docker image on the test server first, and on successful build, it will build the same on the production server as well.
CICD pipeline looks as follows:

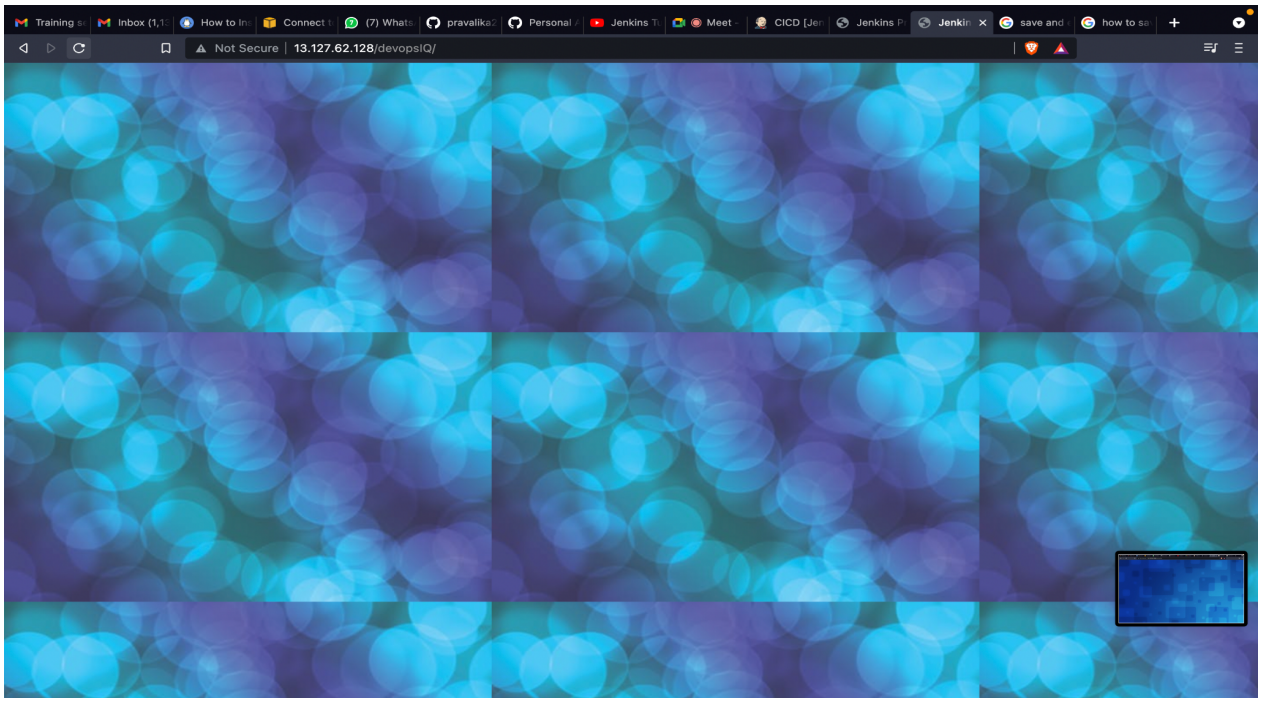


On successful build, the deployed Node application looks as follows on both test and prod servers:



Step17: Here we notice that the Website name is changed. And also the image is changed. We made the changes in the code and pushed it to the GitHub repository. This triggered the pipeline and changes reflect in the deployed website as shown below:





Changes reflected. Hence, Jenkins master slave pipeline has been successful.