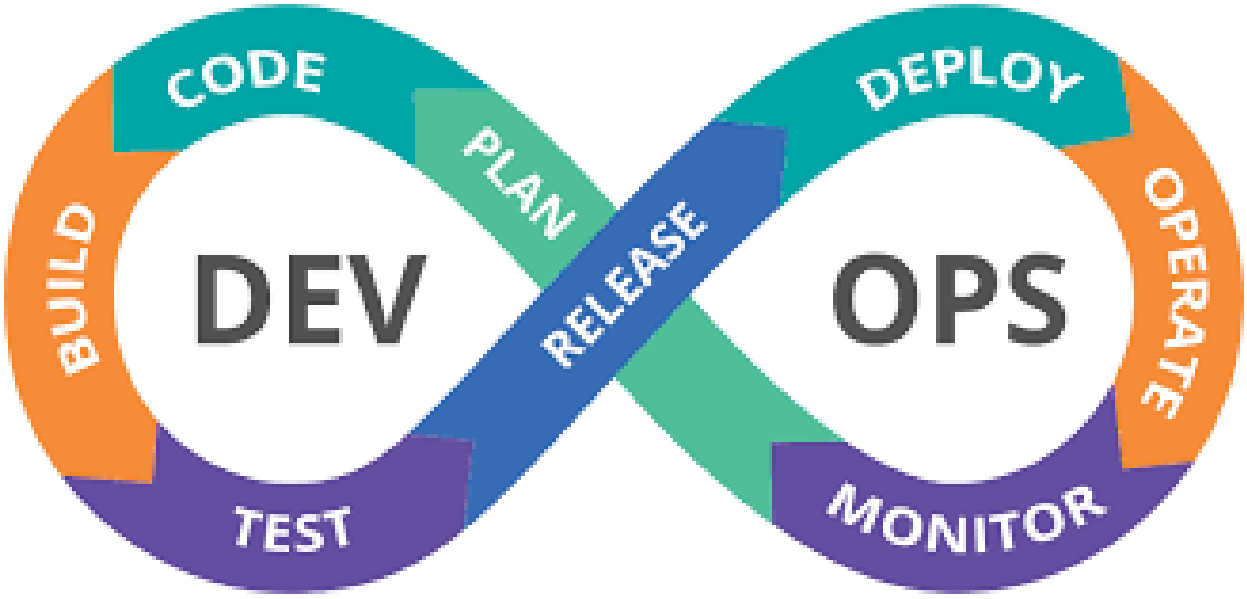


DevOps Jenkins Assignment

—

Submitted to:

Dr. Uma S

Submitted by:

Team 8

Akshaya Reddy 18BCS017

Bhavya Tripathi 18BCS019

Harshitha M 18BCS032

Meghana N 18BCS053

S Namratha 18BCS083

Vinita Yadav 18BCS109

Deepanshu Sachdeva 18BCS114

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY

Dharwad

ज्ञानेन विकासः

TASK 1

Setting up CI/CD Jenkins pipeline for kubernetes

Tools and Technologies used :

- Github
- Docker and Docker hub
- Jenkins
- Kubernetes Cluster

System Requirements :

3 AWS Ec2 Ubuntu instances of size t2.medium and 15 GB of volumes attached.

Step - 1 : Setting Up kubernetes Cluster

We have used the kubeadm tool to set up the kubernetes cluster.

Setting up a kubernetes cluster containing 2 nodes master and worker.

1.1 : Commands to run on both nodes

To update the system packages

```
$    sudo apt-get update
```

1.1.1 : Installing docker

```
$   sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

```
$    curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

```
$    sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
```

```
$   apt-cache policy docker-ce
```

```
$   sudo apt-get install docker-ce docker-ce-cli containerd.io -y
```

To Confirm docker installation

```
$  sudo docker version
```

To add docker daemon

```
$ cat <<EOF | sudo tee /etc/docker/daemon.json
{
"exec-opts": ["native.cgroupdriver=systemd"],
"log-driver": "json-file",
"log-opts": {
"max-size": "100m"
},
"storage-driver": "overlay2"
}
EOF
```

Enabling and starting docker

```
$    sudo systemctl enable docker
$    sudo systemctl daemon-reload
$    sudo systemctl restart docker
```

1.1.2 : Installing kubernetes

Install Kubeadm,Kubelet and Kubectl

```
$     sudo apt-get update
$      sudo apt-get install -y apt-transport-https ca-certificates
curl
$     sudo curl -fsSLo
/usr/share/keyrings/kubernetes-archive-keyring.gpg
https://packages.cloud.google.com/apt/doc/apt-key.gpg
$     echo "deb
[signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg]
https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee
/etc/apt/sources.list.d/kubernetes.list
$      sudo apt-get update -y
$     sudo apt-get install -y kubelet kubeadm kubectl
```

To confirm kubectl installation

```
$      sudo kubectl version
```

To freeze versions of Kubeadm,Kubelet and Kubectl

```
$      sudo apt-mark hold kubelet kubeadm kubectl
```

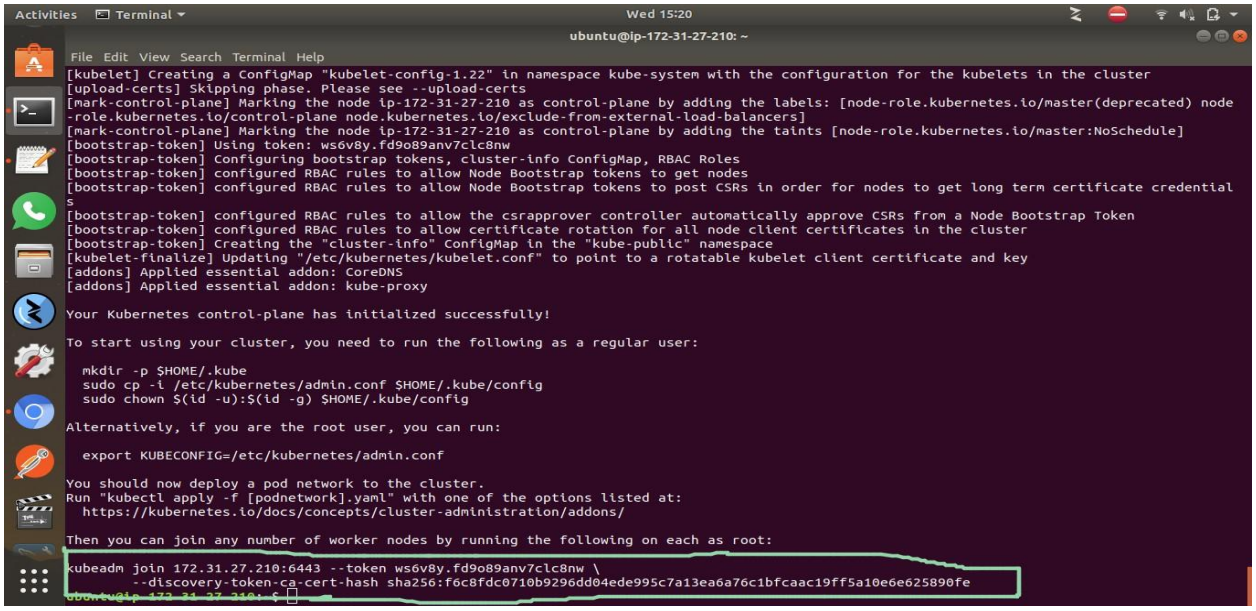
1.2 : Commands to run only on master node

Initializing the master node using kubeadm

```
$     sudo kubeadm init --pod-network-cidr 10.0.0.0/16
```

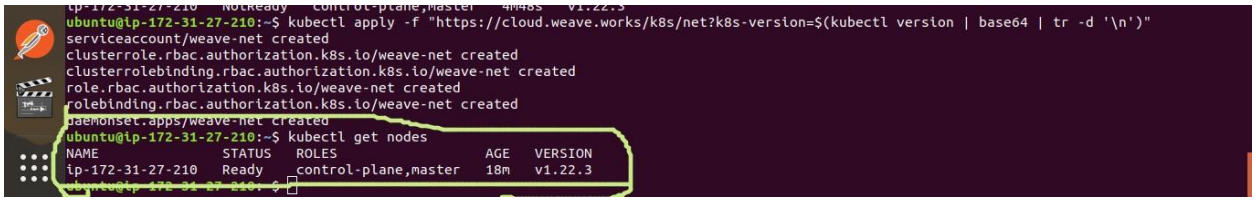
Output of this command would be a key , through which worker nodes can join the kubernetes cluster.

Copy the token and save it somewhere.




```
$     mkdir -p $HOME/.kube
$      sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
$     sudo chown $(id -u):$(id -g) $HOME/.kube/config
$     kubeadm version
$     kubectl apply -f
"https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version |
base64 | tr -d '\n')"
$ kubectl get nodes
```

Gives the nodes and status of nodes present in the cluster

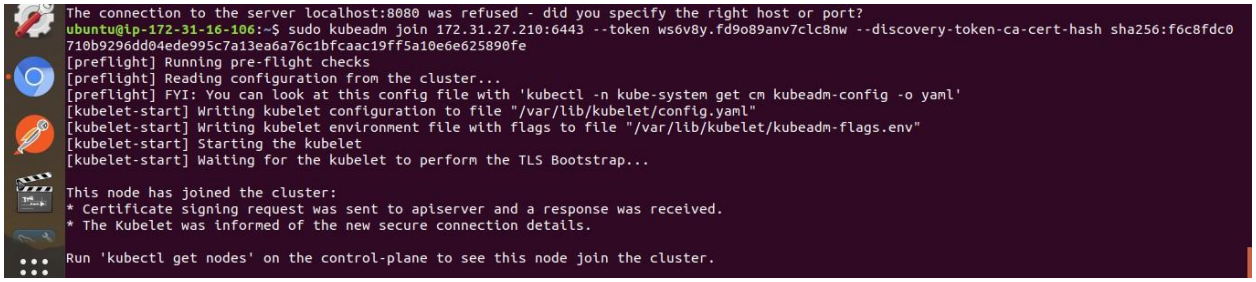


1.3 : Commands to run only on worker node

Using the token copied earlier

$ sudo kubeadm join 172.31.27.210:6443 --token ws6v8y.fd9o89anv7clc8nw --discovery-token-ca-cert-hash sha256:f6c8fdc0710b9296dd04ede995c7a13ea6a76c1bfcaac19ff5a10e6e625890fe

Output of this command



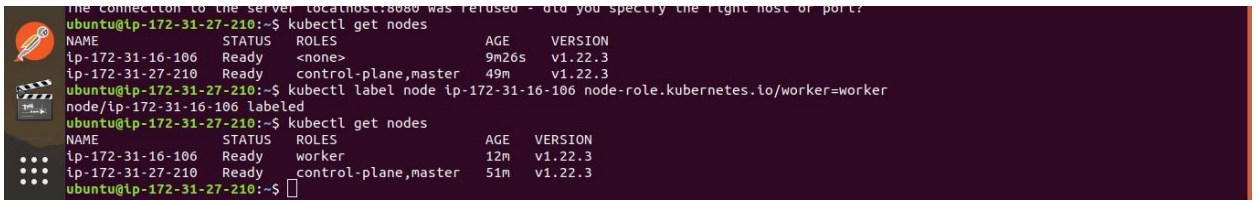
1.4 : Commands to run only on master node

$ kubectl get nodes

Now there would be 2 nodes one master and one newly joined worker node

Newly joined node's role name would be <none> to label it as worker ,

$ kubectl label node ip-@(ip address nuder name section of node) node-role.kubernetes.io/worker=worker

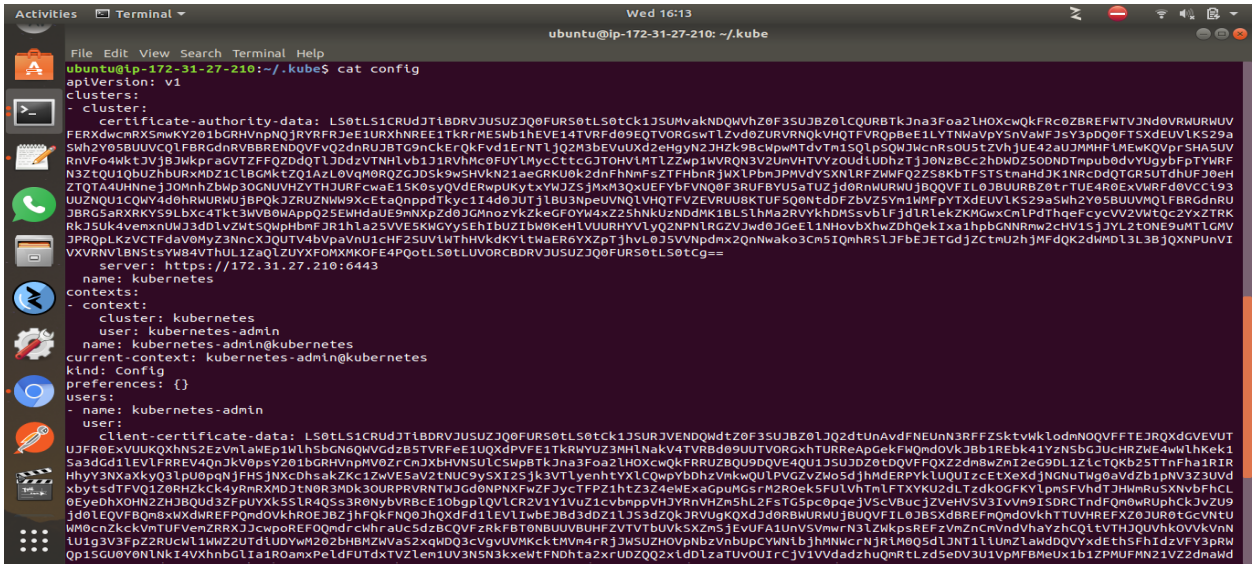


To check configurations of kubernetes cluster

$ cd .kube

$ cat config

Output :



Step - 2 : Setting up jenkins On aws ec2 ubuntu

2.1 : Installing java and jenkins

```
$ sudo apt-get update -y

$  sudo apt install openjdk-8-jdk

$ java -version

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

$ sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > \

    /etc/apt/sources.list.d/jenkins.list'

$ sudo apt-get update

$ sudo apt-get install jenkins
```

Open tcp custom port 8080 on ec2 machine , to verify jenkins installation open http://public-ip-address-of-ec2-instance:8080 like http://34.216.41.126:8080 on browser.

Output :

We need to provide Default jenkins(initialAdminPassword) administrator password, which is present at given path (in the page)

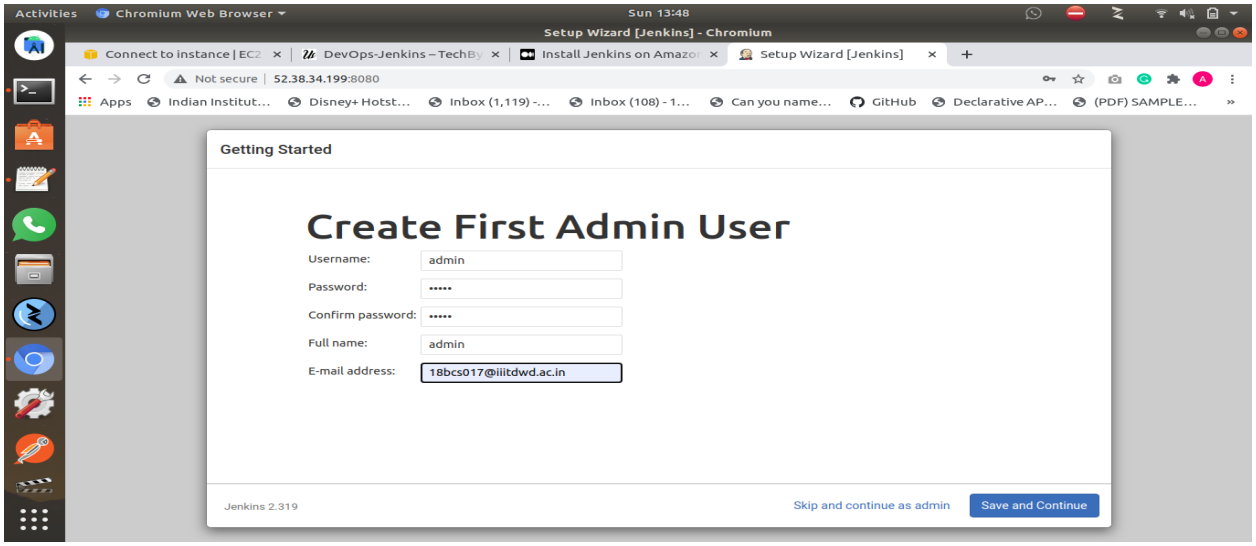
```
$  cat  given-path
```

For example

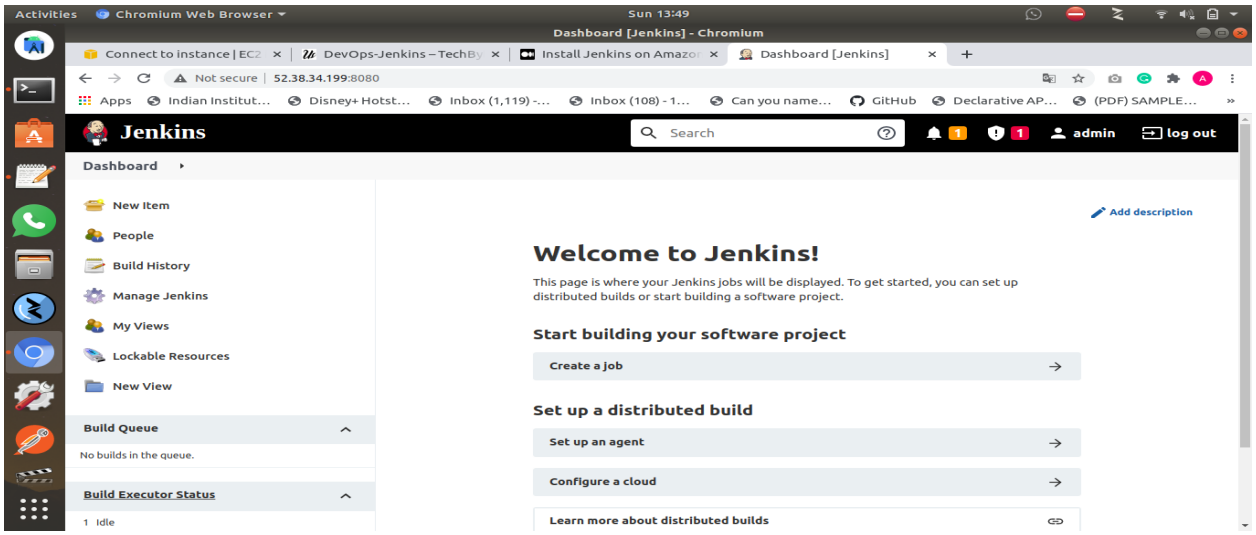
```
$  cat /var/lib/jenkins/secrets/initialAdminPassword
```

In the next page select on Install suggested plugins

After you have installed all the suggested default plugins, it will prompt you for setting up admin username and password -



After creating user credentials, jenkins is ready to use.



2.2 : Installing docker on jenkins server

Same steps mentioned in step 1.1.1 to install docker

To add current ubuntu usr and jenkins to docker group

```
$ sudo usermod -aG docker $USER
```

```
$ sudo usermod -aG docker jenkins
```

Step - 3: Setting up code

We made a simple java application, code is on github.

Code link : [github url]

code also includes the Dockerfile for building the docker image

```
FROM openjdk:11

ARG JAR_FILE

COPY ${JAR_FILE}

ENTRYPOINT  "java" "-jar" "/app.jar"
```

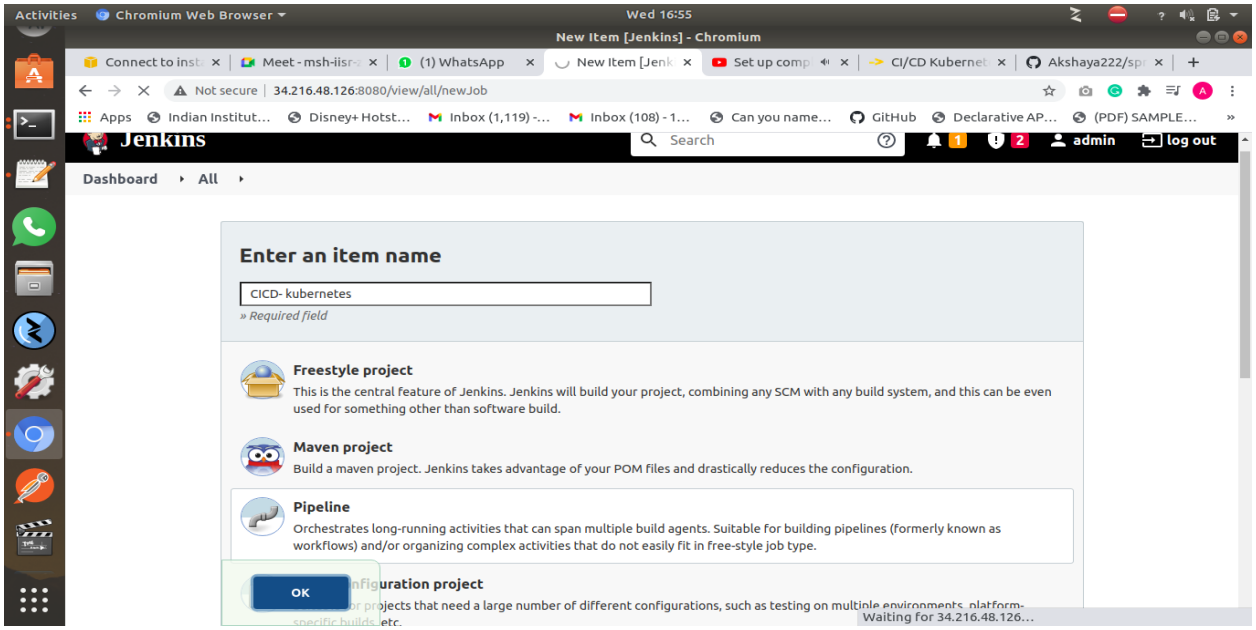
Deployment.yml file is present in the github repo.

Step - 4: Building the pipeline

4.1 : Creating a pipeline

Go to Jenkins dashboard -> new items -> enter item name:CICD-pipeline -> select pipeline -> click ok

After creating a pipeline , select pipeline.



4.2 : Cloning the github repository

First stage in the pipeline is to clone the code from github repository , first we need to setup the github credentials

4.2.1 : setup github credentials

Go to jenkins dashboard -> manage jenkins -> manage credentials -> stores scope to jenkins -> global -> add credentials

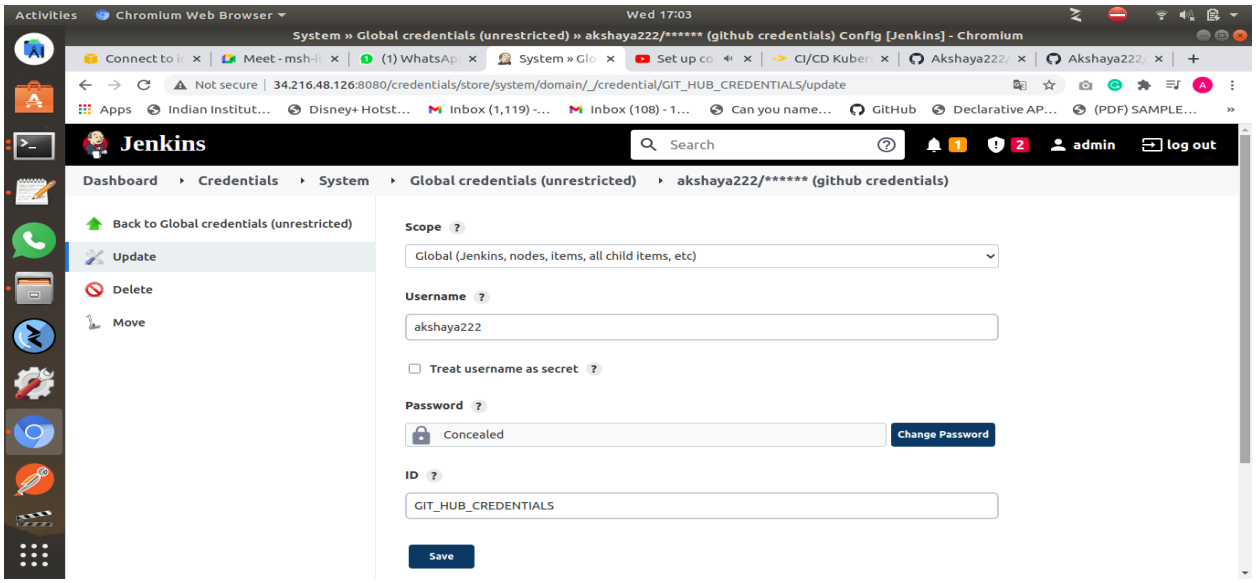
Select username and password from select menu

Scope : select global scope

For username : Github username

For password : Github Personal Access Token (PAT)

For ID : GIT_HUB_CREDENTIALS

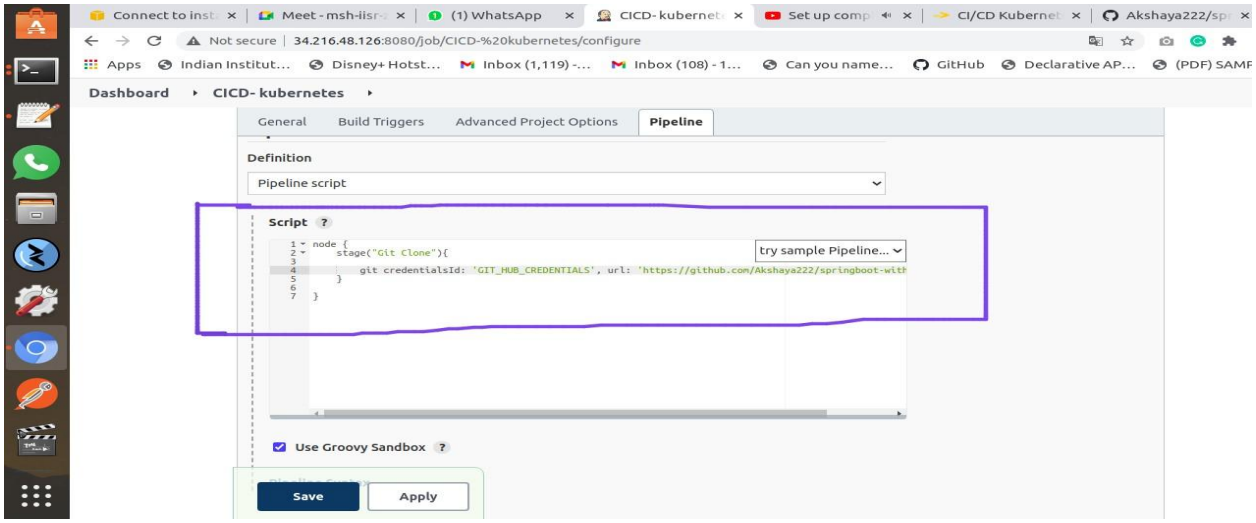


4.2.2 : adding stage in pipeline

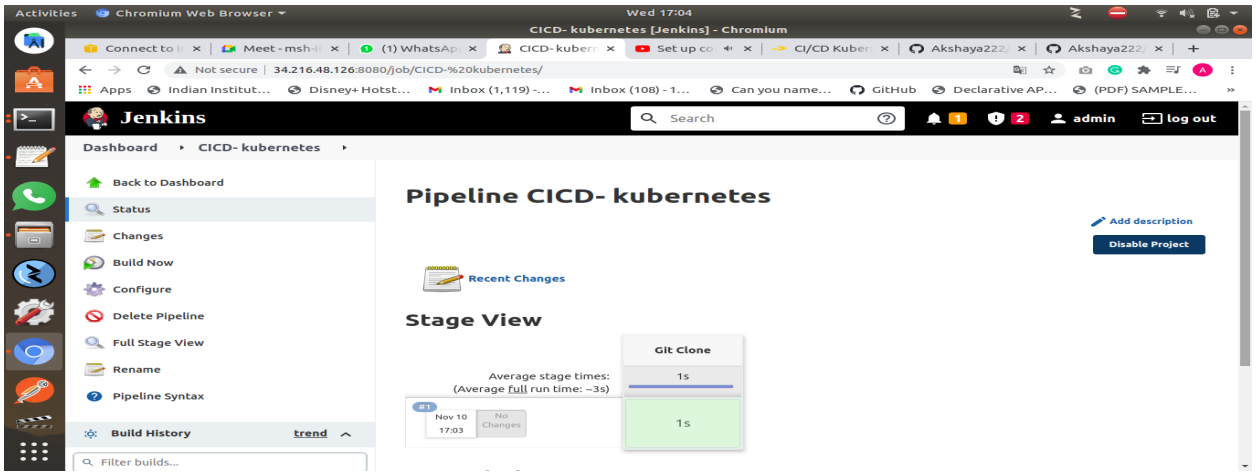
git credentialsId: Id of the github credentials- GIT_HUB_CREDENTIALS is the id of credentials created just before this

url : url of the github repository

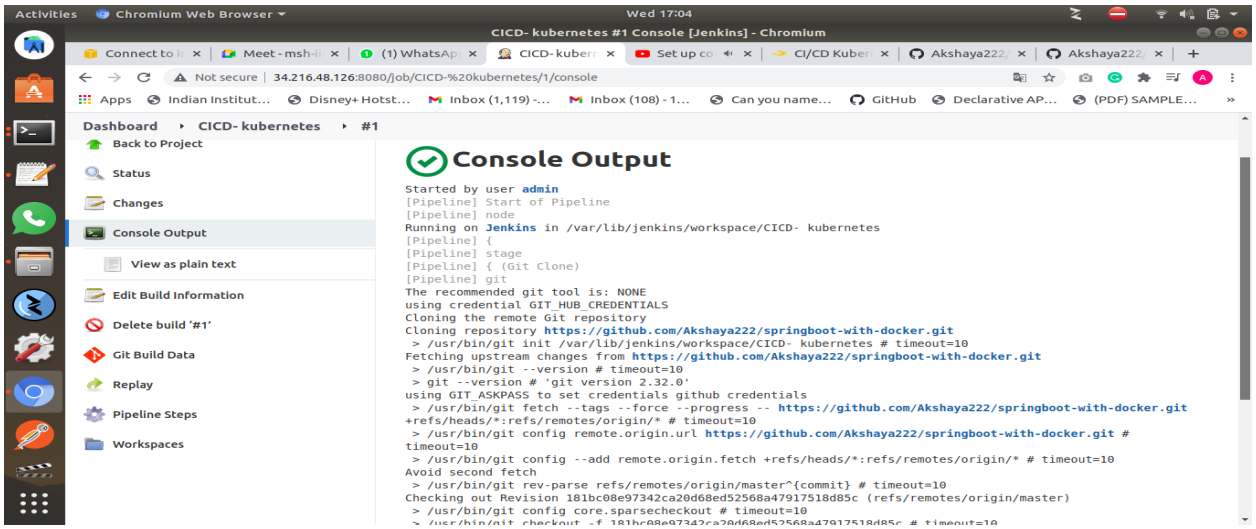
```
node {
stage("Git Clone") {
  git credentialsId: 'GIT_HUB_CREDENTIALS',
url:'https://github.com/Akshaya222/springboot-with-docker.git'
    }
}
```



Build status of first stage



Logs of the build:

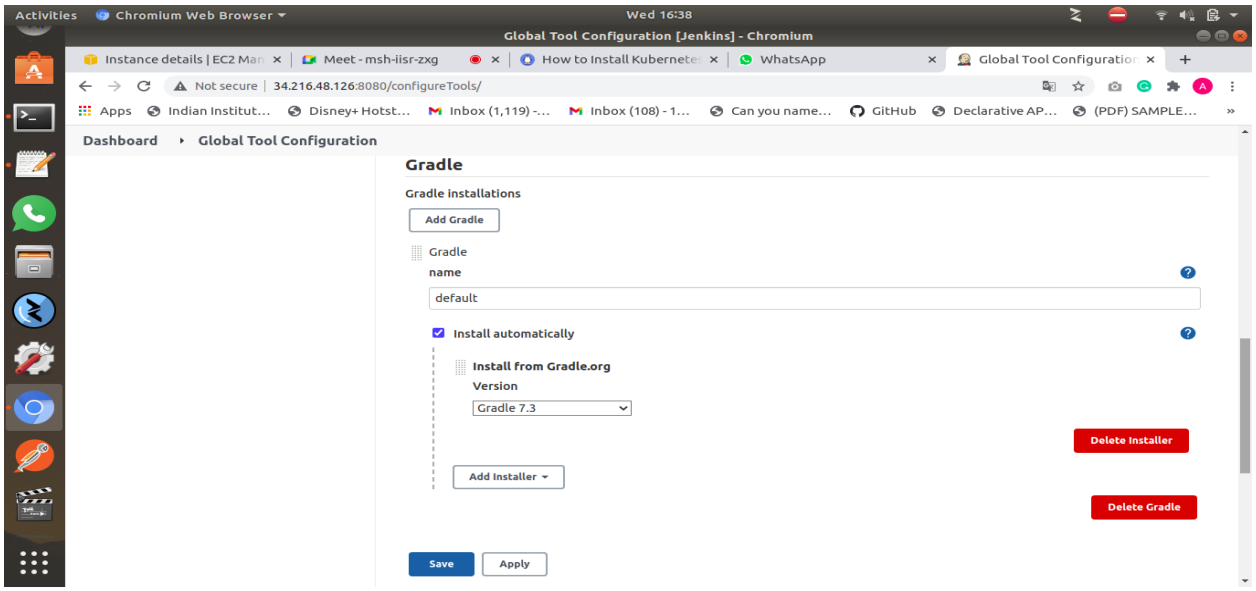


4.3 : Building the spring boot application using gradle

4.3.1 : setup gradle for building application

Go to jenkins dashboard -> manage jenkins -> global tool configuration -> gradle -> add gradle

Enter name as default ,Select latest version from select menu and enable install automatically.



4.3.2 : adding stage for building application using gradle

```
stage('Gradle Build') {
    sh './gradlew build'
  }
```

4.4 : Build the docker image and tag it

4.4.1 : Create a repository in the docker hub

I have created a repository named akshaya-docker-hub-repos.

4.4.2 adding stage for building docker image and tagging it.

```
stage("Docker build"){
  sh 'docker version'
   sh 'docker build -t new-image-name.'
   sh 'docker image list'
```

```
  sh 'docker tag new-image-name
docker_id/repository_name:new-image-name'
    }
```

4.5 : Adding docker login stage

4.5.1 : create docker credentials in jenkins

Go to jenkins dashboard -> manage jenkins -> manage credentials -> stored scoped to jenkins -> global -> add credentials

Select secret text from the kind select menu

Select: scope as global scope

Secret :docker hub password

Id : DOCKER_HUB_PASSWORD

4.5.2 : adding stage for docker login for jenkins

credentialsId : ID of docker credentials

```
stage("Docker Login"){
        withCredentials([string(credentialsId: 'DOCKER_HUB_PASSWORD',
variable: 'PASSWORD')]) {
            sh 'docker login -u akshaya1dockerlid -p $PASSWORD'
        }
    }
```

4.6 : Pushing the tagged docker image into docker hub

```
stage("Push Image to Docker Hub"){
        sh 'docker push
akshaya1dockerlid/akshaya-docker-hub-repos:akshaya-docker-image'
    }
```

4.7 : add stage for login in to the kubernetes master node from jenkins server

We do this by through ssh

4.7.1 : add ssh pipeline plugin

Go to jenkins dashboard -> manage jenkins -> manage plugins -> available -> in the search box type ssh pipeline steps ->

select ssh pipeline plugin -> install without restart.

4.7.2 : create password for kubernetes master server (ec2 ubuntu)

```
$ sudo vi /etc/ssh/sshd_config
```

Change the line passwordAuthentication from “no” to “yes”

Change the line permitRooLogin from “prohibit-password” to “yes”

```
$ sudo passwd ubuntu
```

It will prompt you to enter a password.

```
$ sudo service sshd restart
```

4.7.3 : adding the stage for ssh-ing into k8s server

remote.name= any name

remote.host : public ip of kubernetes master

remote.user:ubuntu remote.password:password of ec2 instance

```
stage("SSH Into k8s Server") {
        def remote = [:]
        remote.name = 'kubernetes-master'
        remote.host = '35.86.87.241'
        remote.user = 'ubuntu'
        remote.password = 'ubuntu'
        remote.allowAnyHosts = true    }
```

4.8 : add stage for deploying the application on kubernetes

4.8.1 : copy deployment.yml file to kubernetes master

Adding stage for copying deployment.yml to kubernetes master server

deployment.yml name : k8s-spring-boot-deployment.yml

This copies k8s-spring-boot-deployment.yml to the root directory of kubernetes master.

```
stage('Put deployment.yml onto kubernetes master') {
            sshPut remote: remote, from:
'k8s-spring-boot-deployment.yml', into: '.'
        }
```

Checking the k8s-spring-boot-deployment.yml on kubernetes server

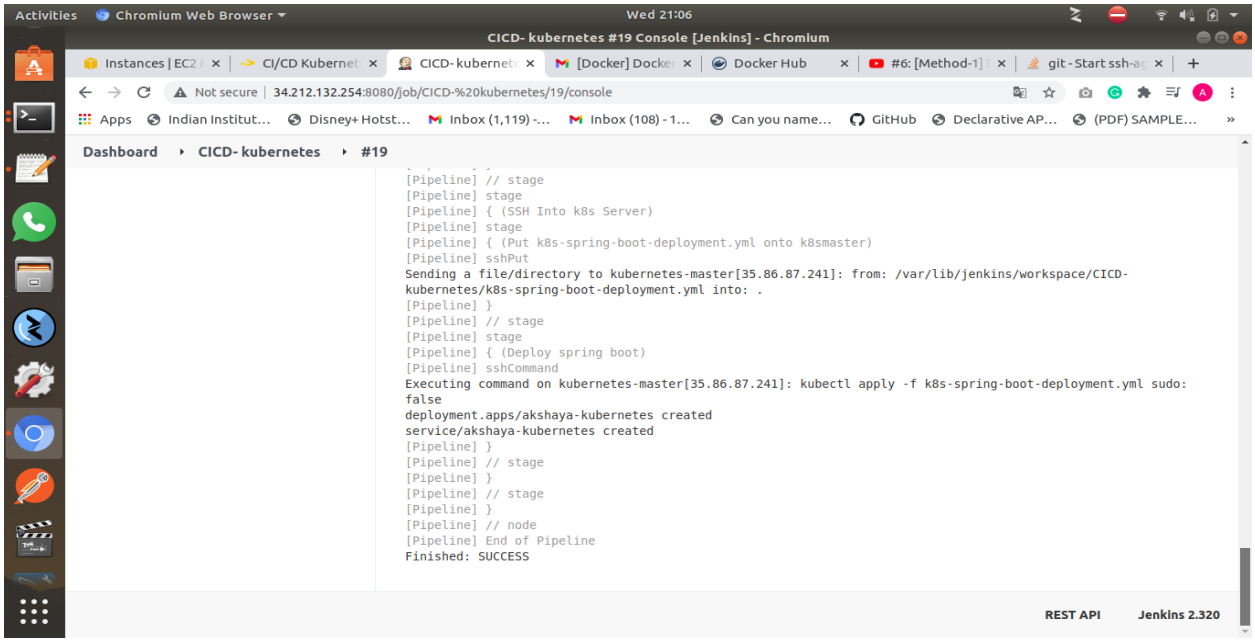
```
Last login: Wed Nov 10 15:00:46 2021 from 103.217.239.132
ubuntu@ip-172-31-27-210:~$ kubectl get nodes
NAME               STATUS     ROLES                  AGE     VERSION
ip-172-31-16-106   NotReady   worker                 5h6m    v1.22.3
ip-172-31-27-210   Ready      control-plane,master   5h45m   v1.22.3
ubuntu@ip-172-31-27-210:~$ ls
k8s-spring-boot-deployment.yml
ubuntu@ip-172-31-27-210:~$
```

4.8.2 : Creating the deployment and service on kubernetes

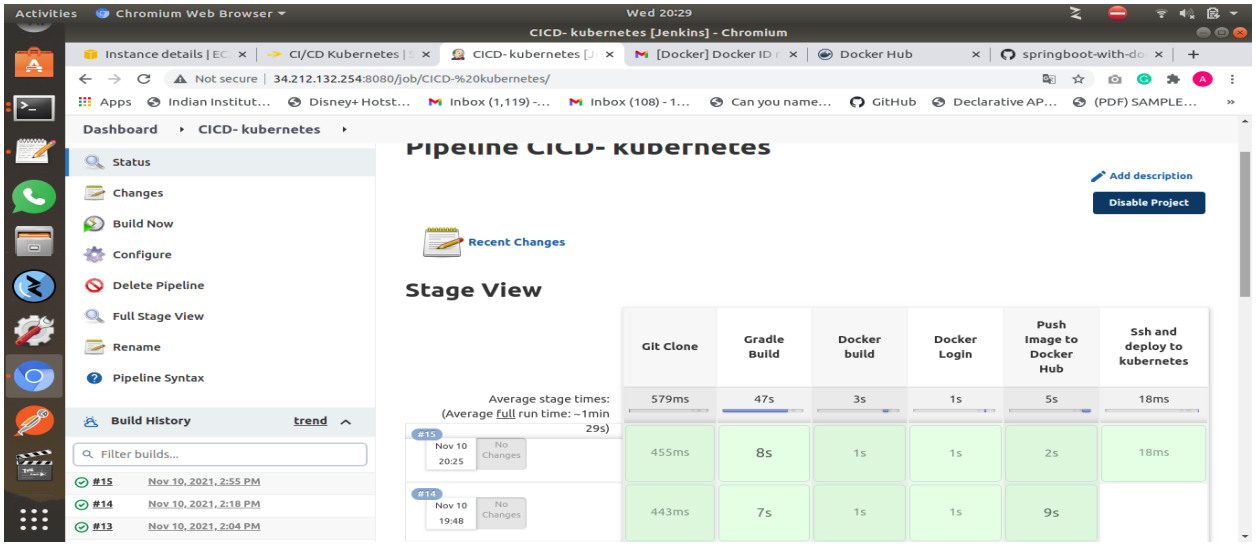
This command creates deployment and exposes the service on nodeport

```
stage('Deploy spring boot') {
        sshCommand remote: remote, command: "kubectl apply -f k8s-spring-boot-deployment.yml"
    }
```

Logs of deployment on jenkins :



Build view of all stages



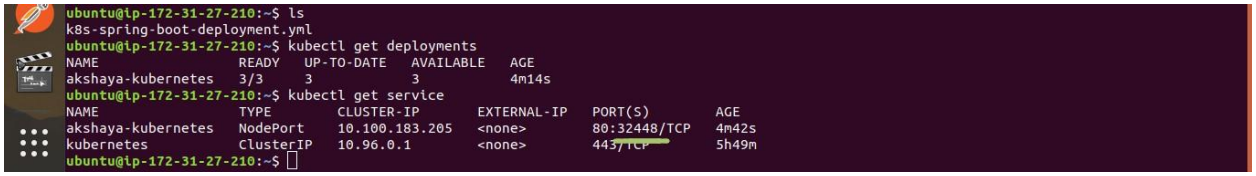
Checking the deployment on kubernetes server

To get list of deployments on kubernetes server

```
$  kubectl get deployments
```

To get list of services on kubernetes server

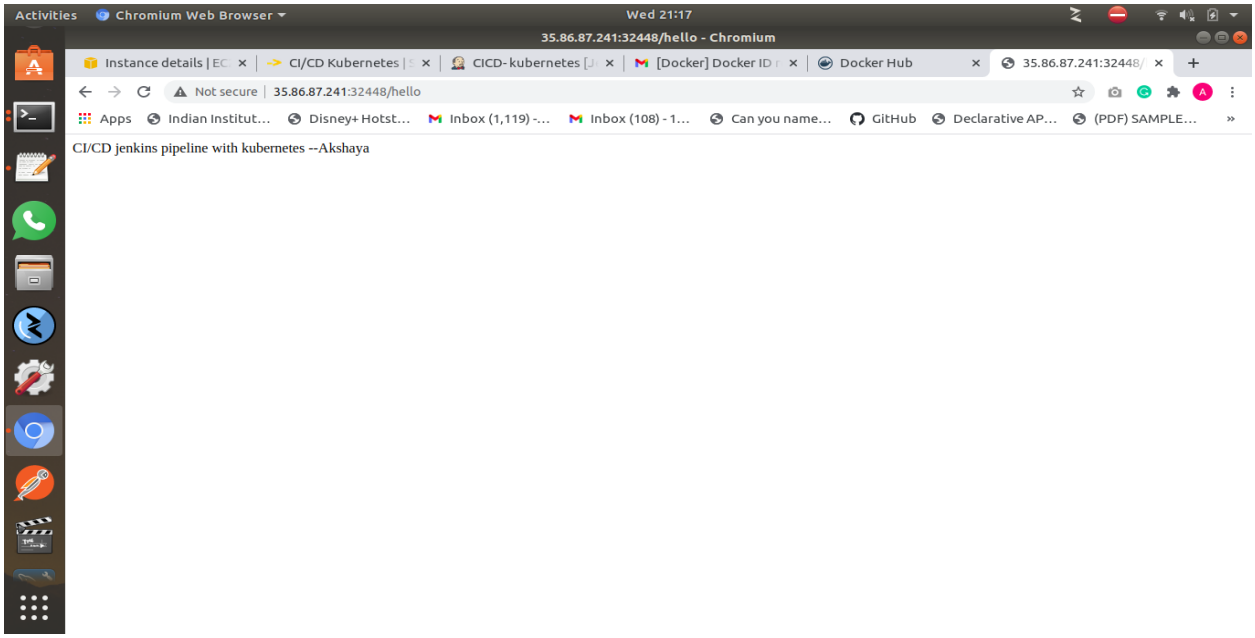
```
$ kubectl get service
```



Akshaya-kubernetes service is running on Nodeport, so Deployment on kubernetes is successful.

To verify the service running open http://public-ip-of-kubernetes-server:port_of_service/hello

From the terminal output, akshaya-kubernetes service is running on port number 37448, so open tcp custom port 37448 on kubernetes server instance.



Application is successfully deployed on to kubernetes through jenkins CICD pipeline.

Task - 2 : Jenkins Master Slave pipeline

Tools and Technologies used :

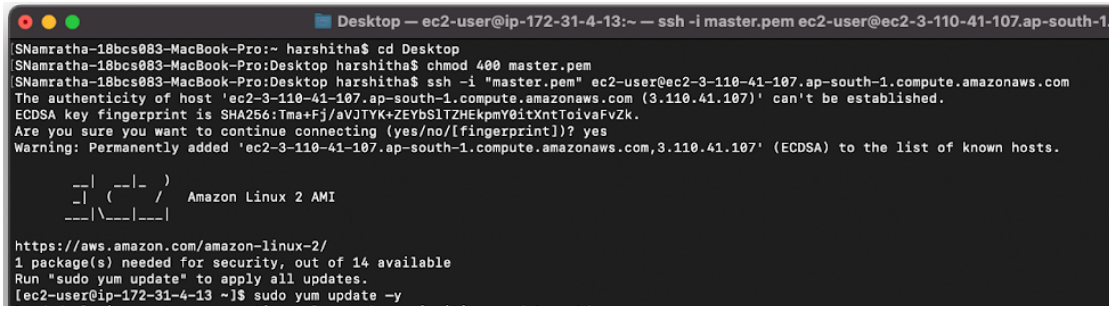
- Github
- Docker
- Jenkins

System Requirements :

3 AWS Ec2 Linux instances of size t2.micro.

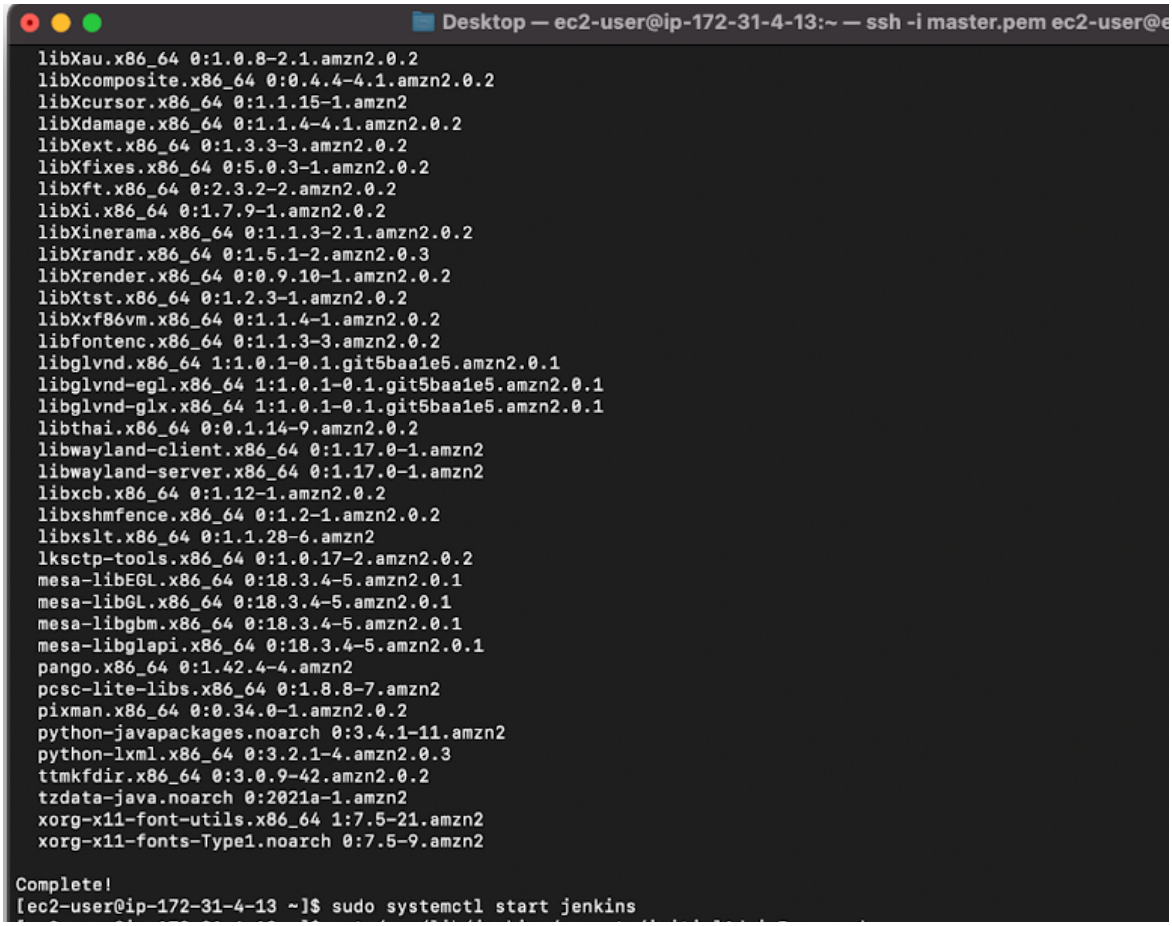
Step - 1 : Setting up Jenkins Master

1.1 : Connect to ec2 linux instance through ssh



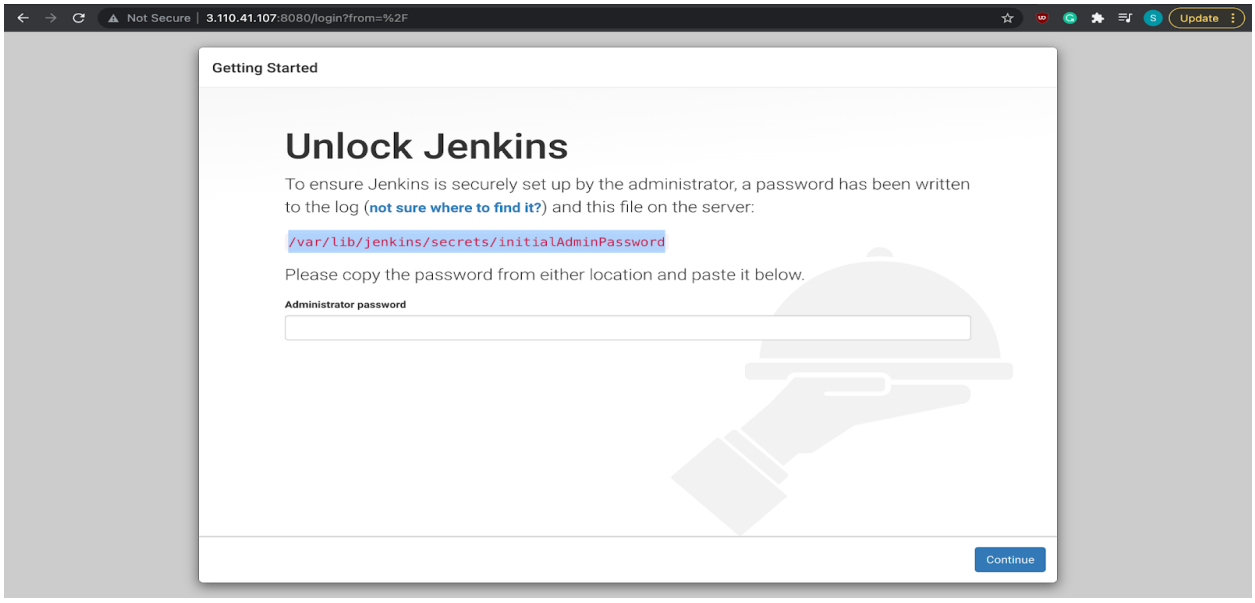
1.2 : Install Java and jenkins

```
$     sudo yum update -y

$     sudo yum install java-1.8.0

$     sudo wget -O /etc/yum.repos.d/jenkins.repo
http://pkg.jenkins-ci.org/redhat/jenkins.repo

$     sudo rpm — import https://pkg.jenkins.io/redhat/jenkins.io.key

$     amazon-linux-extras install epel -y

$      sudo yum install jenkins -y

$   sudo service jenkins start
```



Open custom tcp port 8080 on this ec2 instance

Go to browser and hit http://public-ip-of-instance:8080



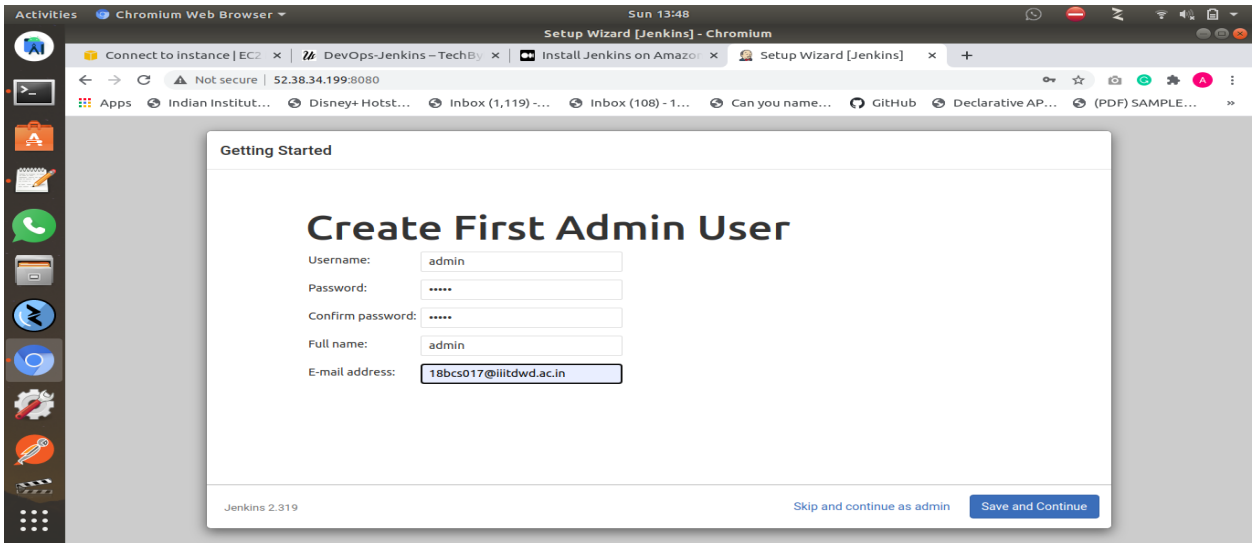
We need to provide Default jenkins(initialAdminPassword) administrator password, which is present at given path (in the page)

$ cat given-path For example

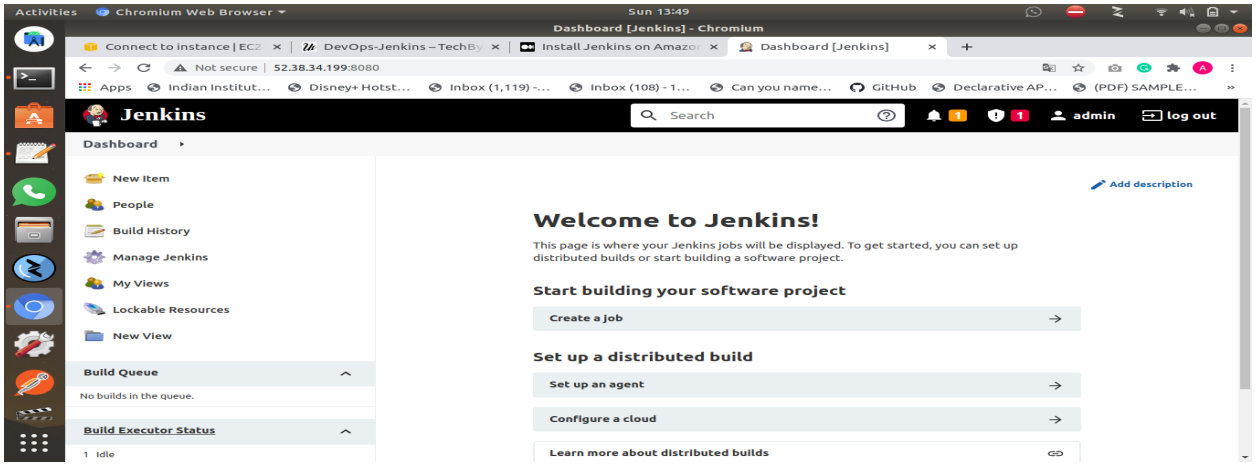
```
$  cat /var/lib/jenkins/secrets/initialAdminPassword
```

In the next page select on Install suggested plugins

After you have installed all the suggested default plugins, it will prompt you for setting up admin username and password -



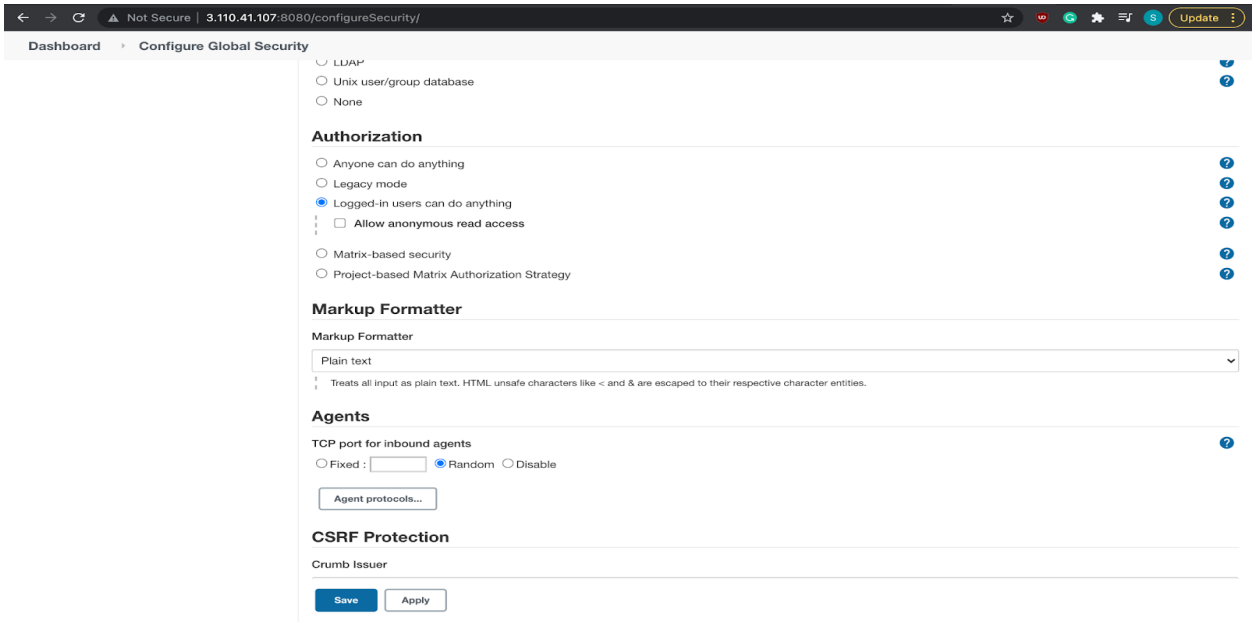
After creating user credentials, jenkins is ready to use.



1.3 : Configuring this server as jenkins master

1.3.1 : Name other 2 instances as slave-1 and slave-2

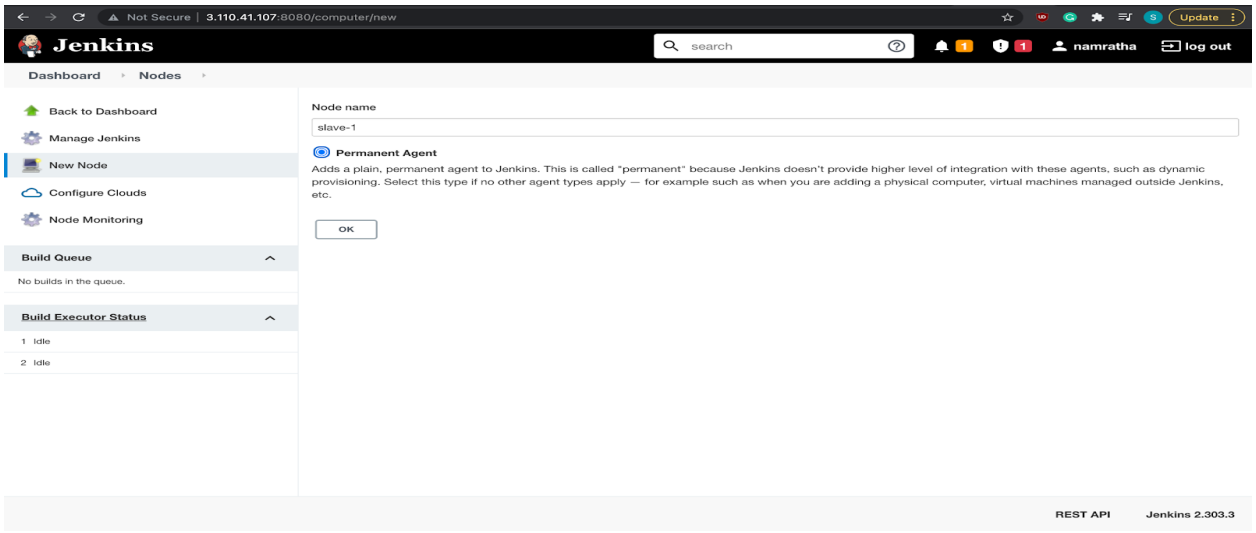
1.3.2 : Go to jenkins -> Manage jenkins -> Configure global security. In agents, change TCP port for inbound agents to random and save.



1.3.3 connect slave-1 and slave-2 to this jenkins master (do same for both slaves)

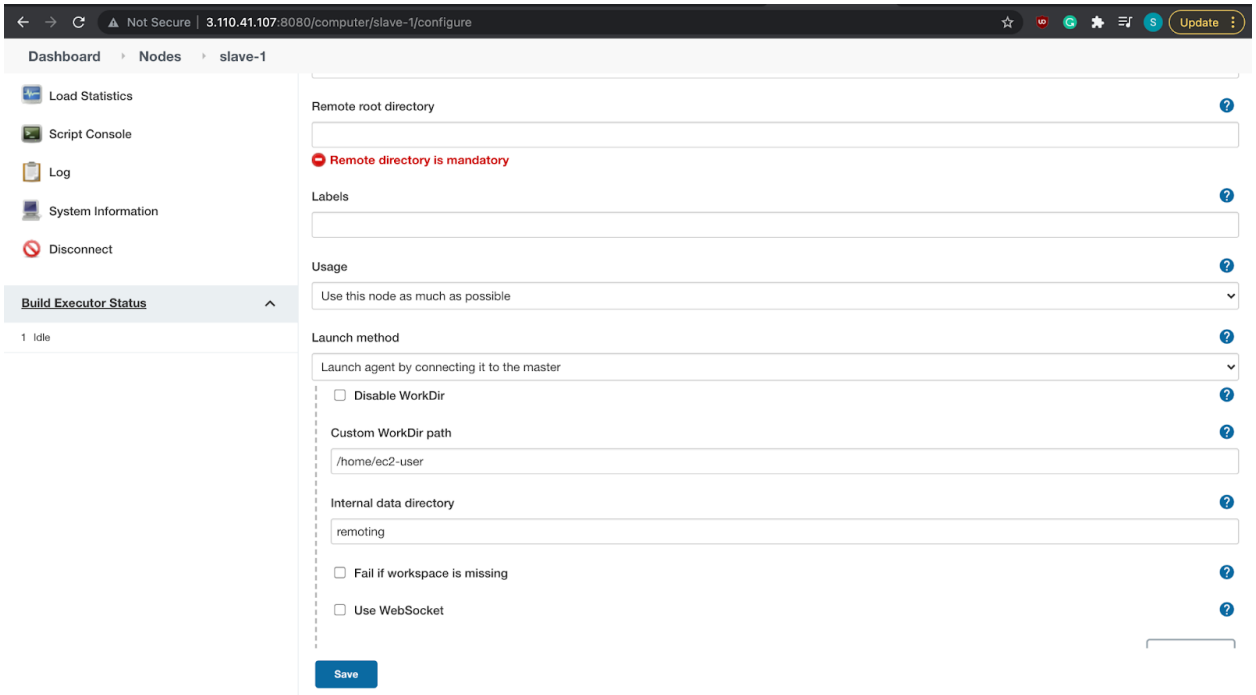
Go to jenkins dashboard -> manage nodes -> add new node

Enter name as slave -1 for slave -1 and slave -2 for slave-2 and enable permanent agent and click ok.

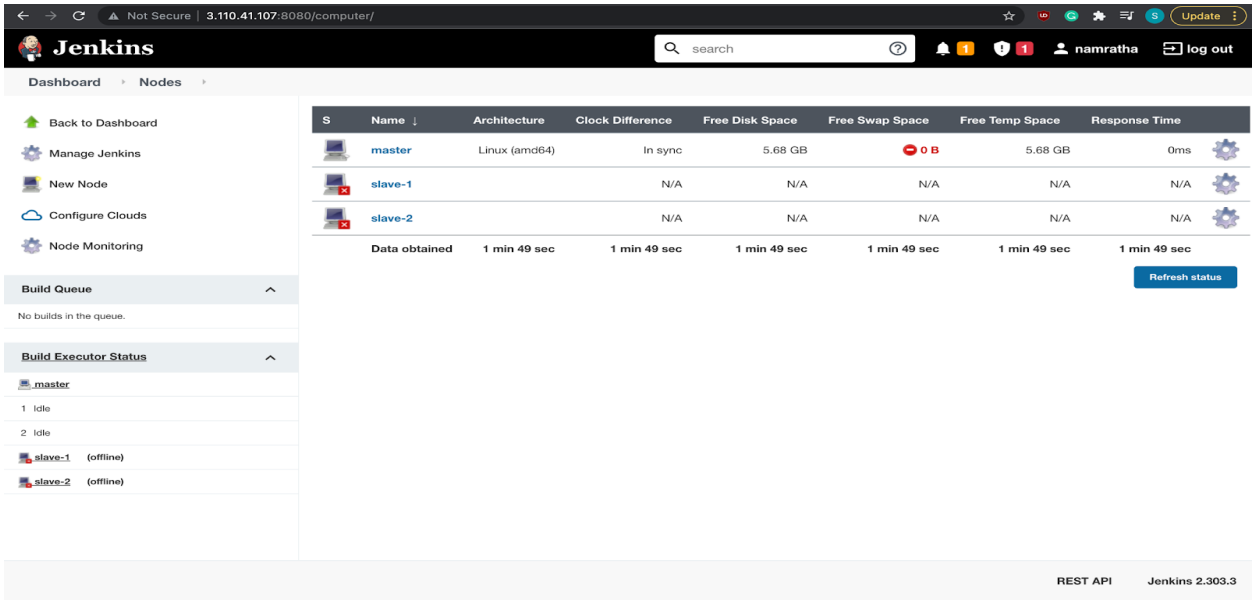


In the launch method select launch by connecting to master from select menu.

For root directory -> /home/ec2-user



Manage nodes page looks like :

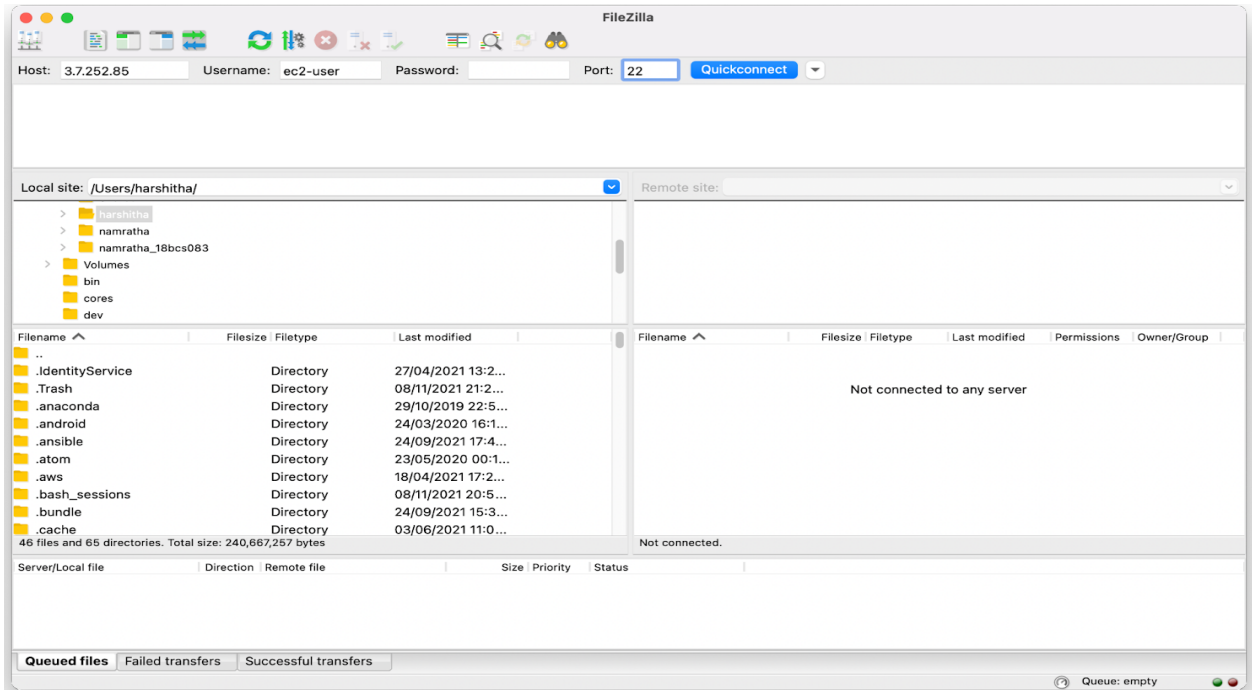


Go to slave-1 and download the agent.jar file.

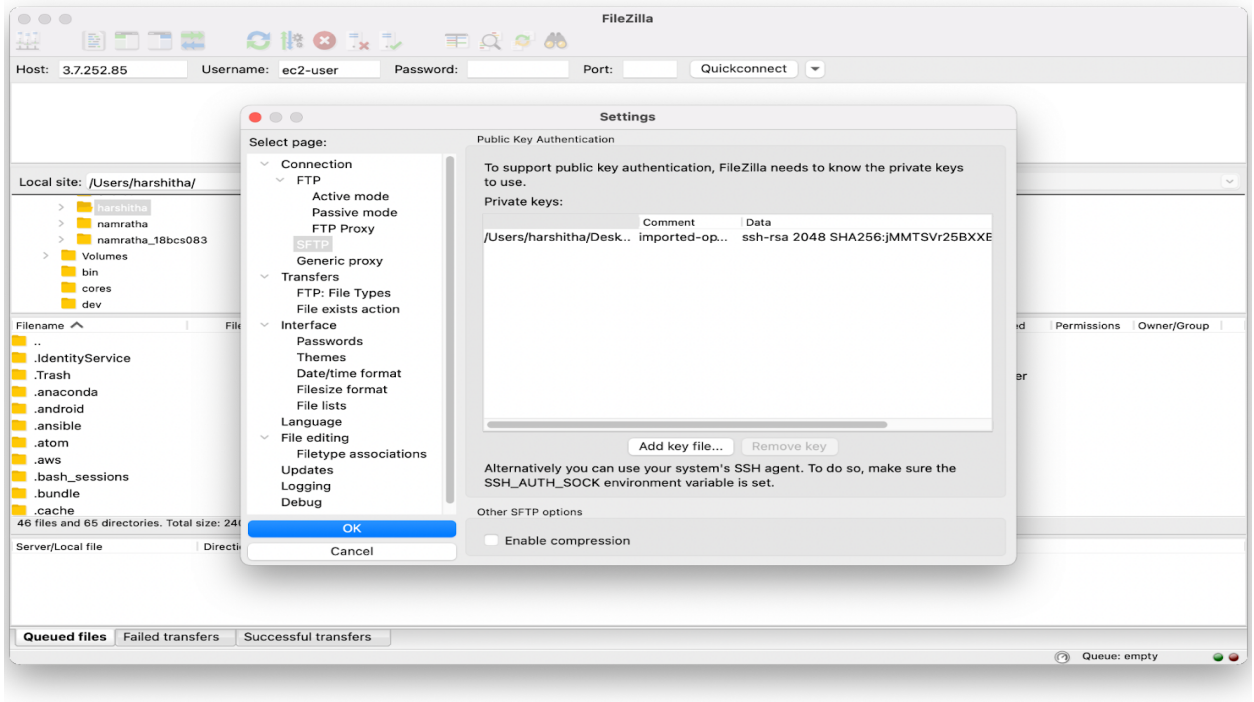
Go to slave1 EC2 instance and select public ip address.

Open filezilla.

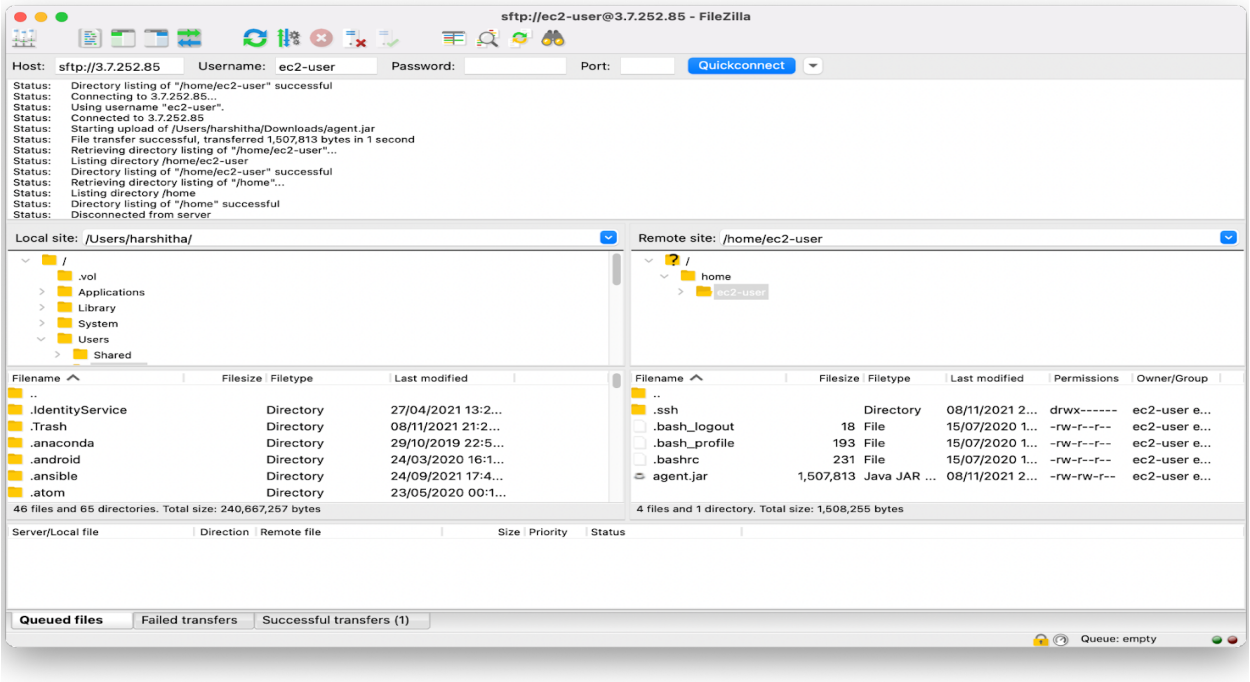
Paste the ip address, change username to ec2-user, and port number 22.



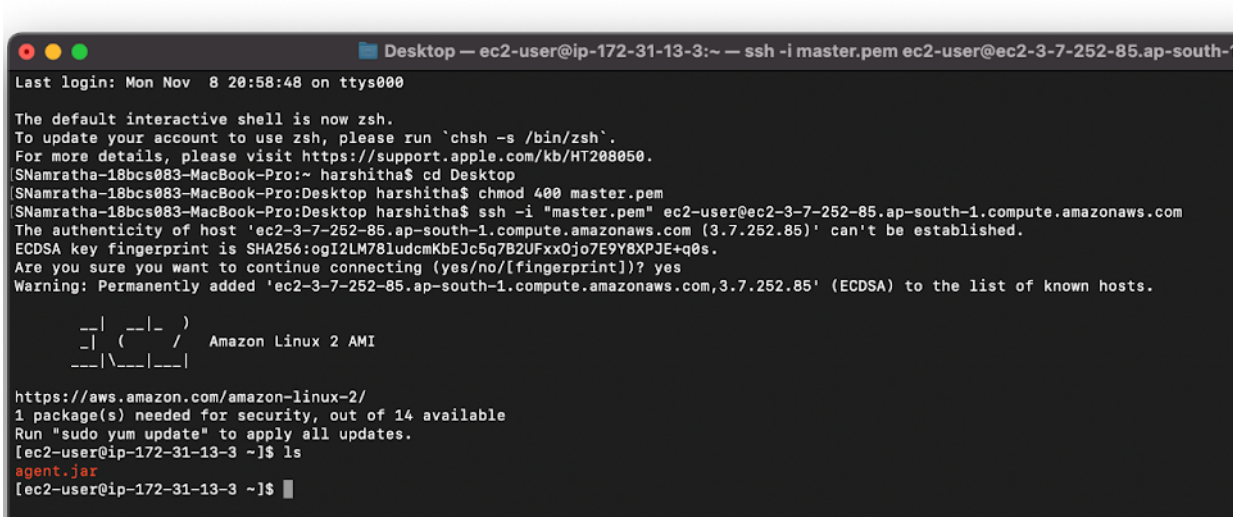
In setting -> SFTP, add the pem file.



After hosting, add the agent.jar file in ec2-user.



To verify, connect to slave-1 and type command - ls. Agent.jar is present.

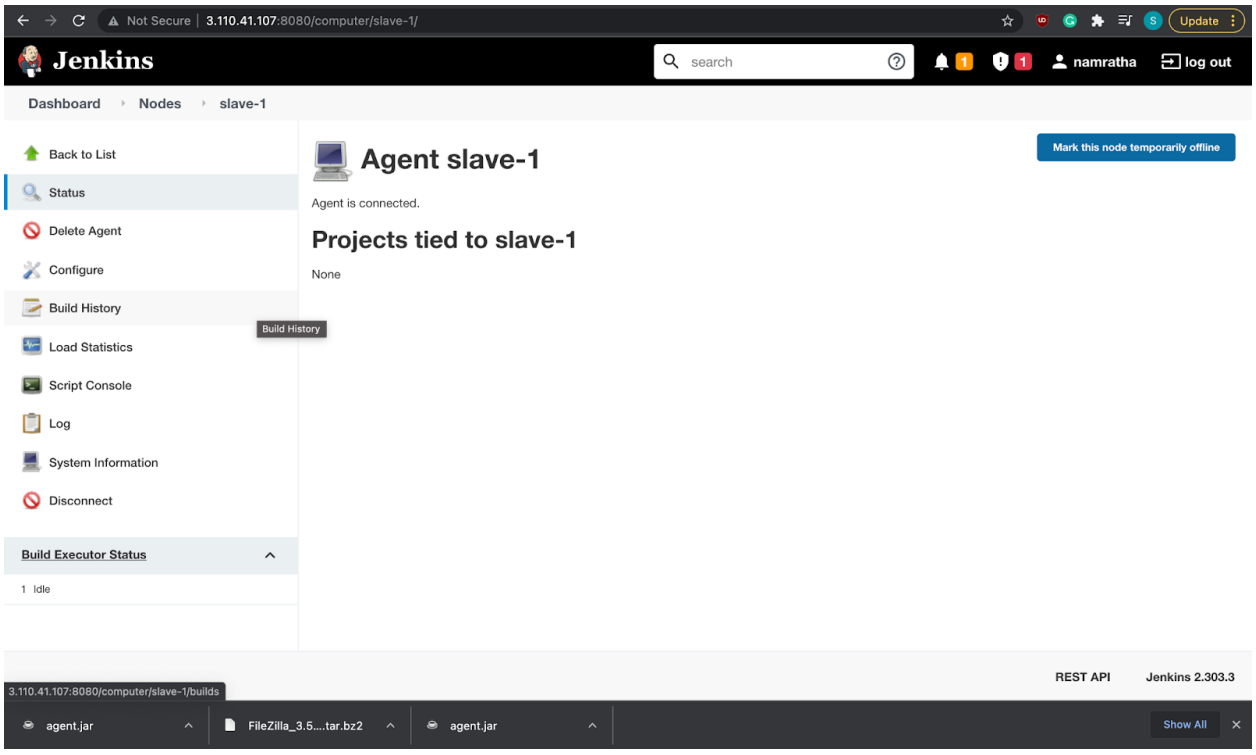


Install java on slaves

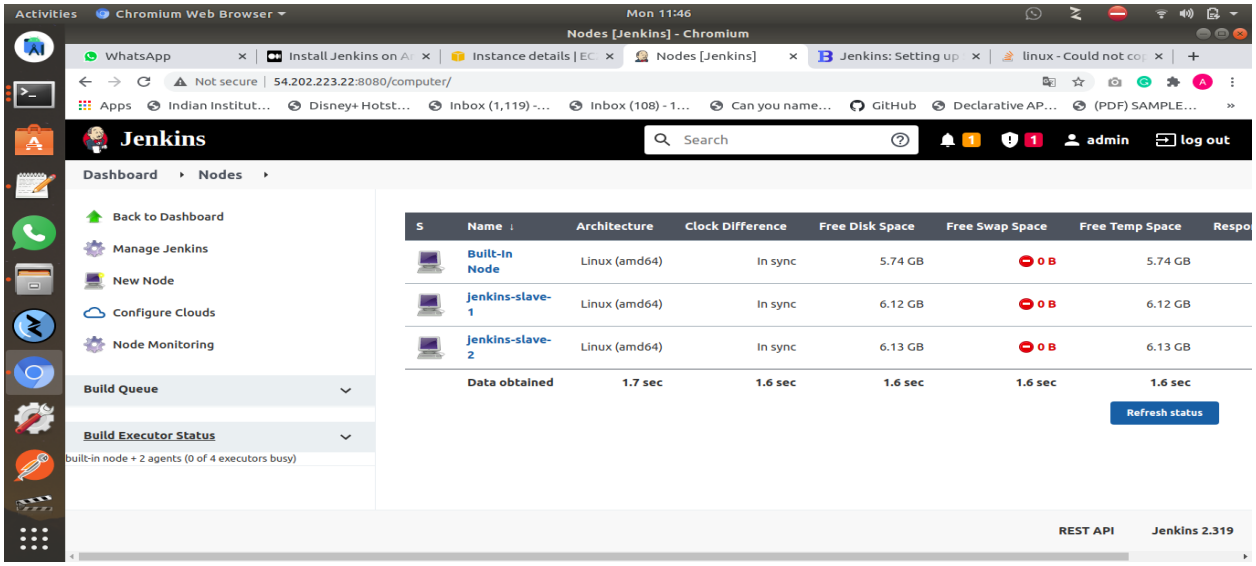
```
$     sudo yum update -y
$     sudo yum install java-1.8.0
```

```
[ec2-user@ip-172-31-13-3 ~]$ java -jar agent.jar -jnlpUrl http://3.110.41.107:8080/computer/slave-1/jenkins-agent.jnlp -secret 176b64fb55870df5236c5b93c0f0fa0fdd43e1fa579e1bdf53ccf81b36e13d4d -workDir "/h
ome/ec2-user"
Nov 08, 2021 4:58:18 PM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFO: Using /home/ec2-user/remoting as a remoting work directory
Nov 08, 2021 4:58:19 PM org.jenkinsci.remoting.engine.WorkDirManager setupLogging
INFO: Both error and output logs will be printed to /home/ec2-user/remoting
Nov 08, 2021 4:58:19 PM hudson.remoting.jnlp.Main createEngine
INFO: Setting up agent: slave-1
Nov 08, 2021 4:58:19 PM hudson.remoting.jnlp.Main$CuiListener <init>
INFO: Jenkins agent is running in headless mode.
Nov 08, 2021 4:58:19 PM hudson.remoting.Engine startEngine
INFO: Using Remoting version: 4.10.1
Nov 08, 2021 4:58:19 PM org.jenkinsci.remoting.engine.WorkDirManager initializeWorkDir
INFO: Using /home/ec2-user/remoting as a remoting work directory

Nov 08, 2021 4:58:19 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Locating server among [http://3.110.41.107:8080/]
Nov 08, 2021 4:58:19 PM org.jenkinsci.remoting.engine.JnlpAgentEndpointResolver resolve
INFO: Remoting server accepts the following protocols: [JNLP4-connect, Ping]
Nov 08, 2021 4:58:19 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Agent discovery successful
  Agent address: 3.110.41.107
  Agent port:    42959
  Identity:      78:8d:f7:95:e6:e8:cd:22:8a:7c:69:e5:d5:f4:00:92
Nov 08, 2021 4:58:19 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Handshaking
Nov 08, 2021 4:58:19 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Connecting to 3.110.41.107:42959
Nov 08, 2021 4:58:19 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Trying protocol: JNLP4-connect
Nov 08, 2021 4:58:19 PM org.jenkinsci.remoting.protocol.impl.BIONetworkLayer$Reader run
INFO: Waiting for ProtocolStack to start.
Nov 08, 2021 4:58:19 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Remote identity confirmed: 78:8d:f7:95:e6:e8:cd:22:8a:7c:69:e5:d5:f4:00:92
Nov 08, 2021 4:58:21 PM hudson.remoting.jnlp.Main$CuiListener status
INFO: Connected
```

Slave -1 looks like this, Agent is connected.



Slaves are in sync



Step - 2 : Installing docker on slave machines

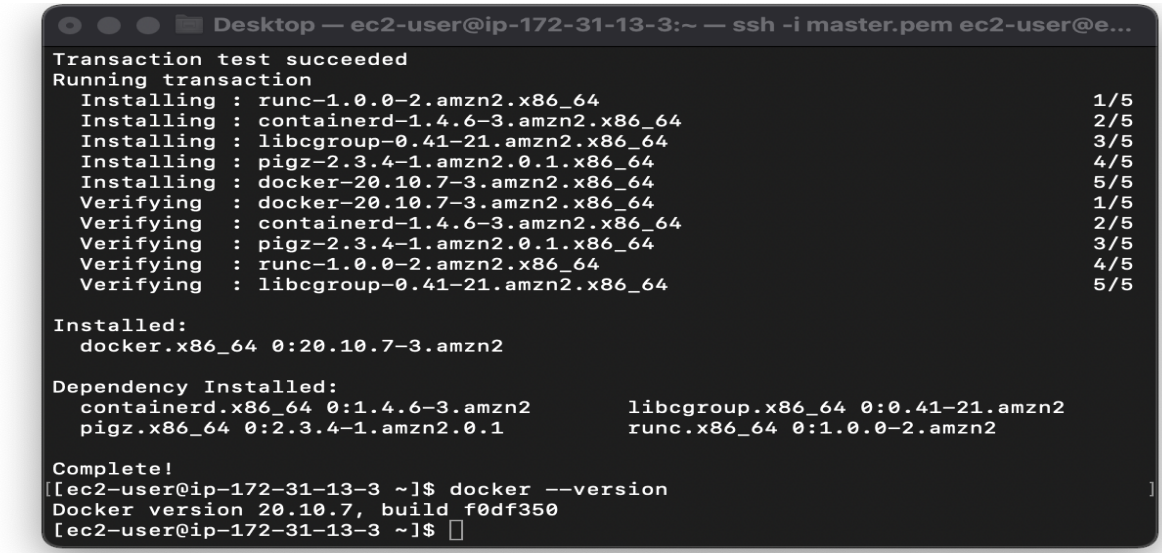
```
$ sudo yum update -y
```

```
$ sudo amazon-linux-extras install docker
```

```
$ sudo yum install docker
```

```
$ sudo service docker start
```

```
$ sudo usermod -a -G docker ec2-user
```




```
$ sudo mkdir jenkins
```

```
$ sudo chmod 777 -R jenkins
```

Step - 3 : CICD jenkins pipeline

Code is in the github repository : github_link

We are treating slave 1 as testing server and slave 2 as production server.

Configuring CICD pipeline in the way

- Configure web hook that triggers the slave-1's job.
- When the slave-1's job is successful, only then slave-2's job is called.

3.1 Configuring the github web hook

Go to github repo -> settings -> select webhook -> add webhook ->

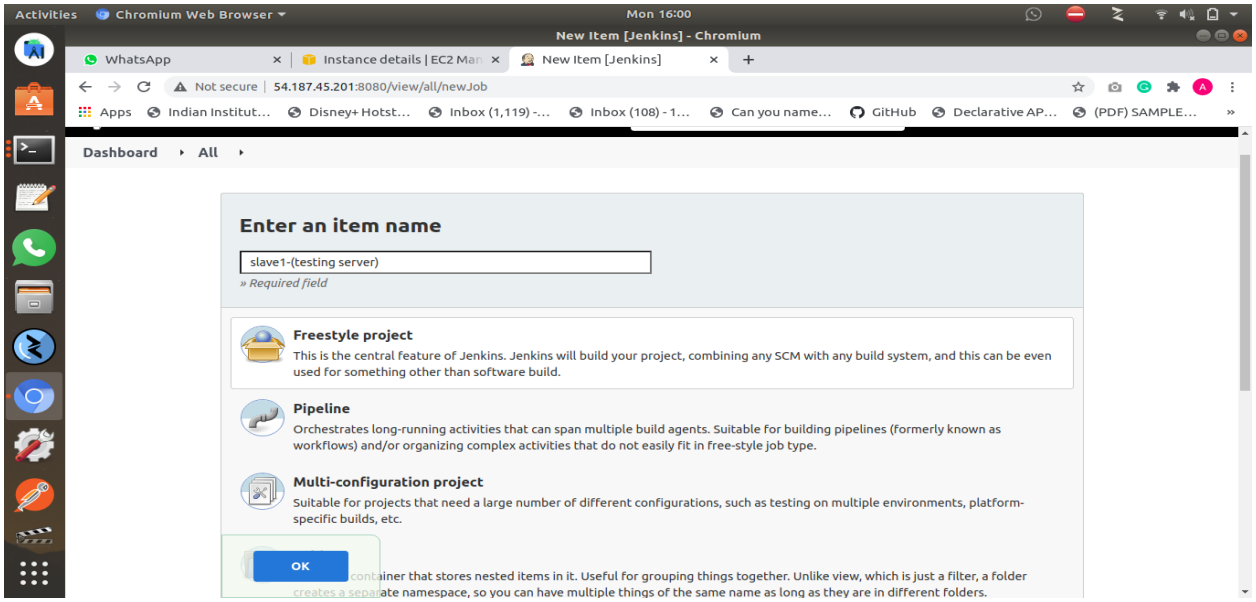
Under payload url http://ip-address-of-jenkins-master:8080/github-webhook/

Select push event

3.2 : Setting up job of slave-1

3.2.1 : Creating new job

Go to jenkins dashboard -> create new job -> enter job name -> select freestyle project -> click ok



3.1.2 : Configuring job

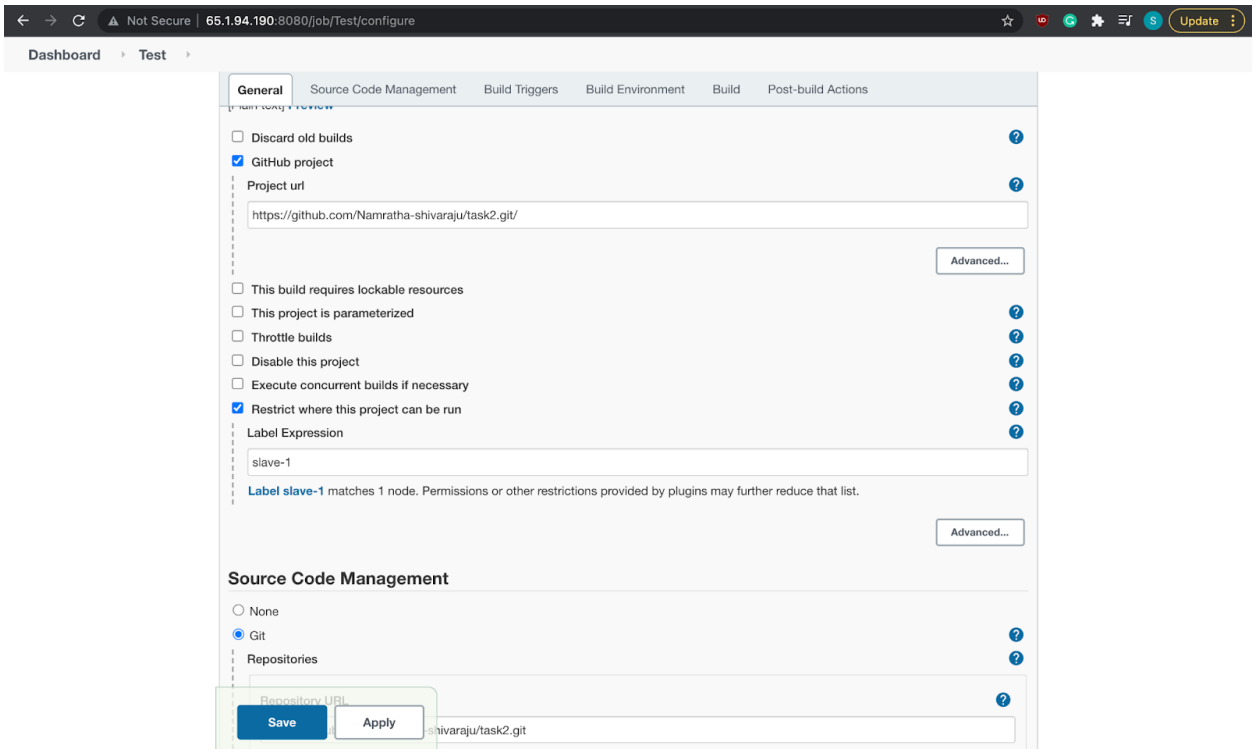
1.Enable under github project

For github url add https://github.com/Akshaya222/devopsIQ.git

2.Enable Restrict where the project can run

Under label expressions add slave-1

3.under source code management select git



When the job executes it creates a directory called workspace in root directory /home/ec2-user.

Jenkins clones the github repository into /home/ec2-user/workspace/job-name, to verify it

Select execute shell put the commands for

delete any existing docker containers , build docker container and open port 82 for it.

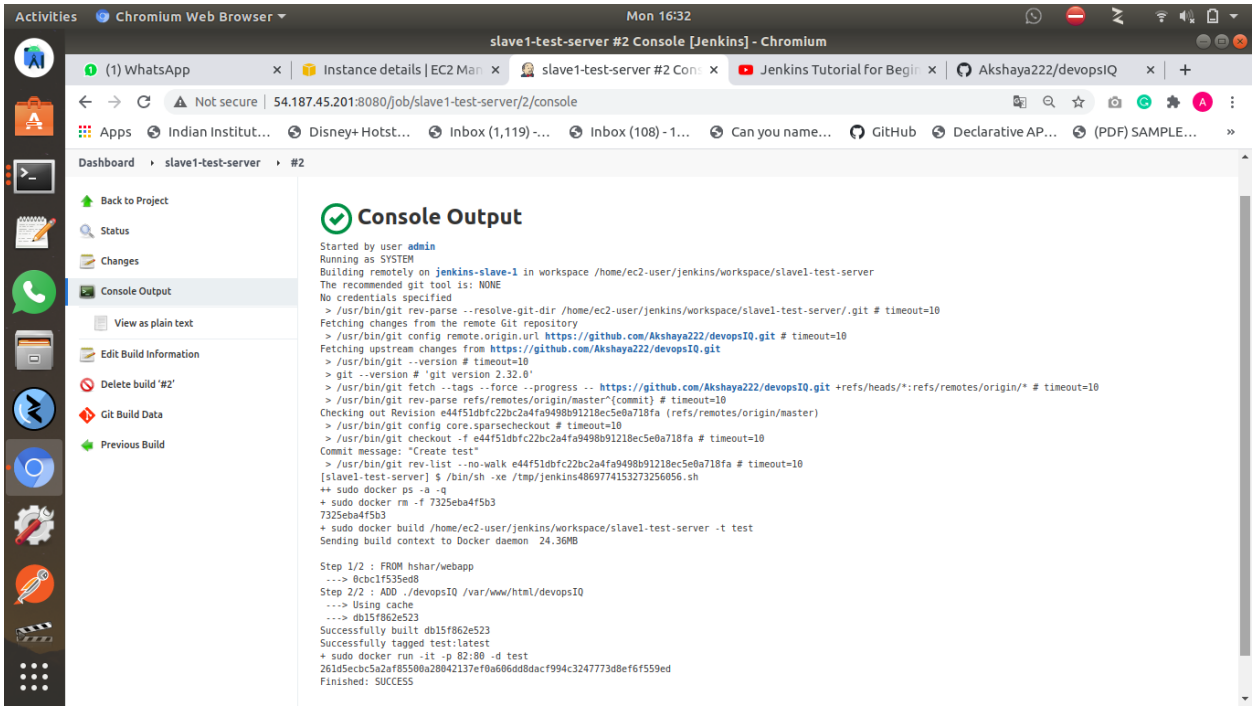
Put these commands under execute shell

```
$ sudo docker rm -f $(sudo docker ps -a -q)
$ sudo docker build  path_of_slave-1_job_under-workspace -t test
```

```
$ sudo docker run -it -p 82:80 -d test
```

Build the job.

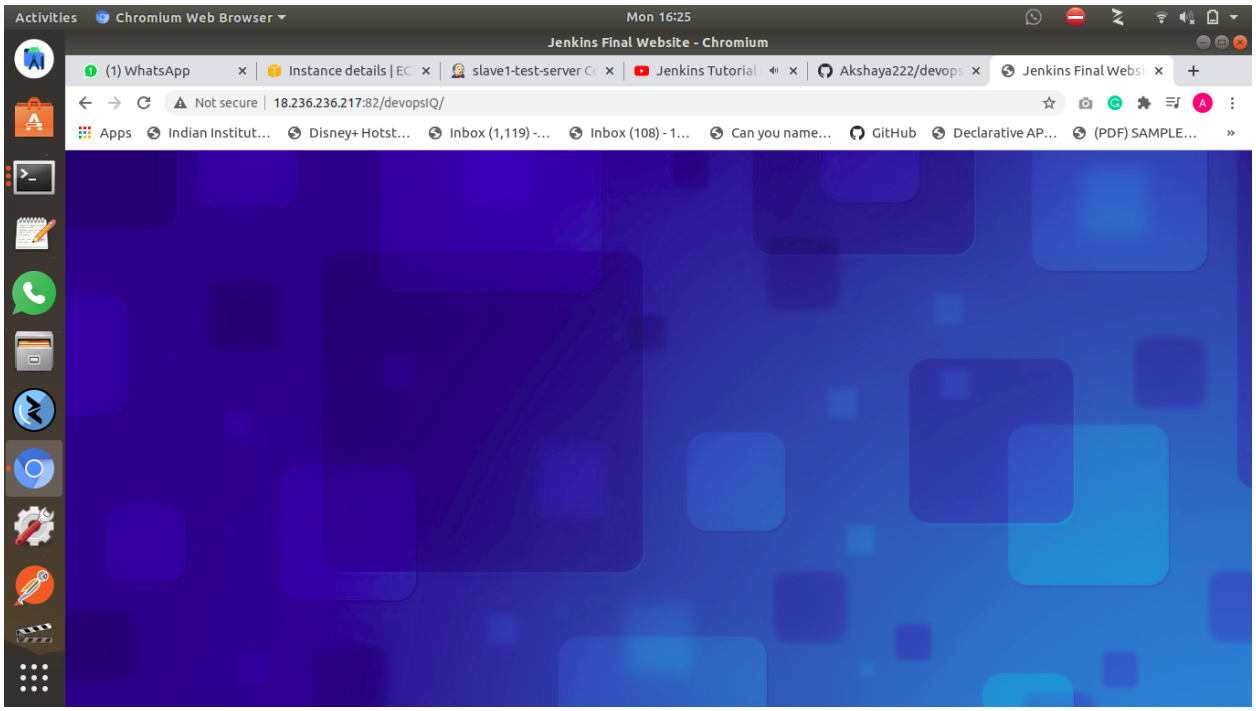
To check if the build is successful



To check if build is successful on testing server (on slave -1)

Open custom port 82 on slave-1 instance

Go to browser and hit http://ipaddress_of_slave-1:82/devopsIQ



3.3 : Setting up job of slave-2

Go to jenkins dashboard -> create new job -> enter job name -> select freestyle project -> click ok

Configure this job same as job of slave-1 and under execute shell,open port 80 for docker

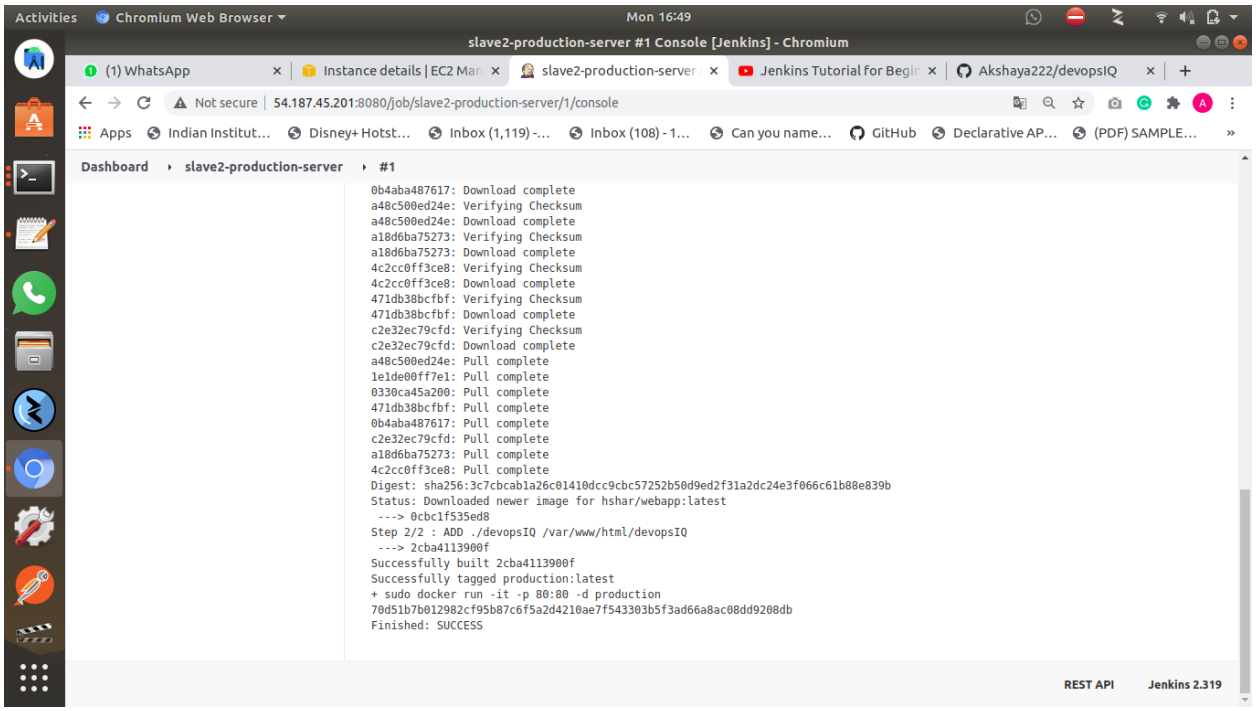
Put these commands under execute shell

```
$ sudo docker rm -f $(sudo docker ps -a -q)
```

```
$ sudo docker build  path_of_slave-2_job_under-workspace -t production
```

```
$ sudo docker run -it -p 80:80 -d production
```

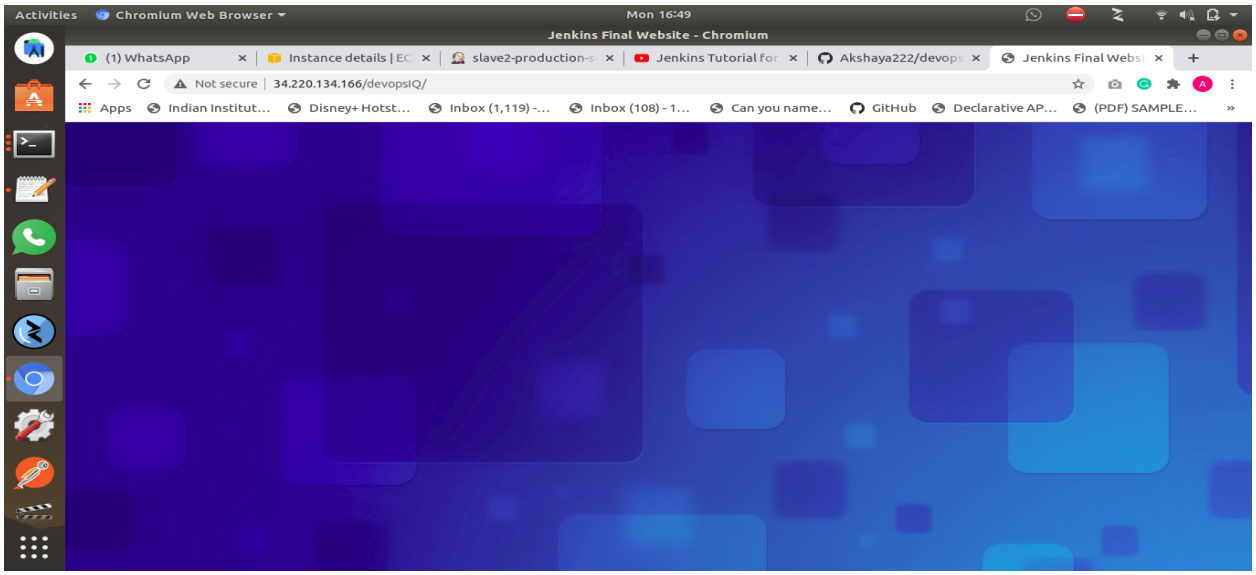
To check if the build is successful



To check if build is successful on production server (on slave -2)

Open custom port 80 on slave-2 instance

Go to browser and hit http://ipaddress_of_slave-2:80/devopsIQ



3.4 : Configuring pipeline to trigger production job (job of slave-2) only if testing job(job of slave-1) is successful.

Go to jenkins dashboard -> select testing job(job of slave 1) -> configure

Under post build actions -> **select build other projects** -> enter production job name (name of slave-2's job) ->

select trigger only if build is stable. -> click on save.

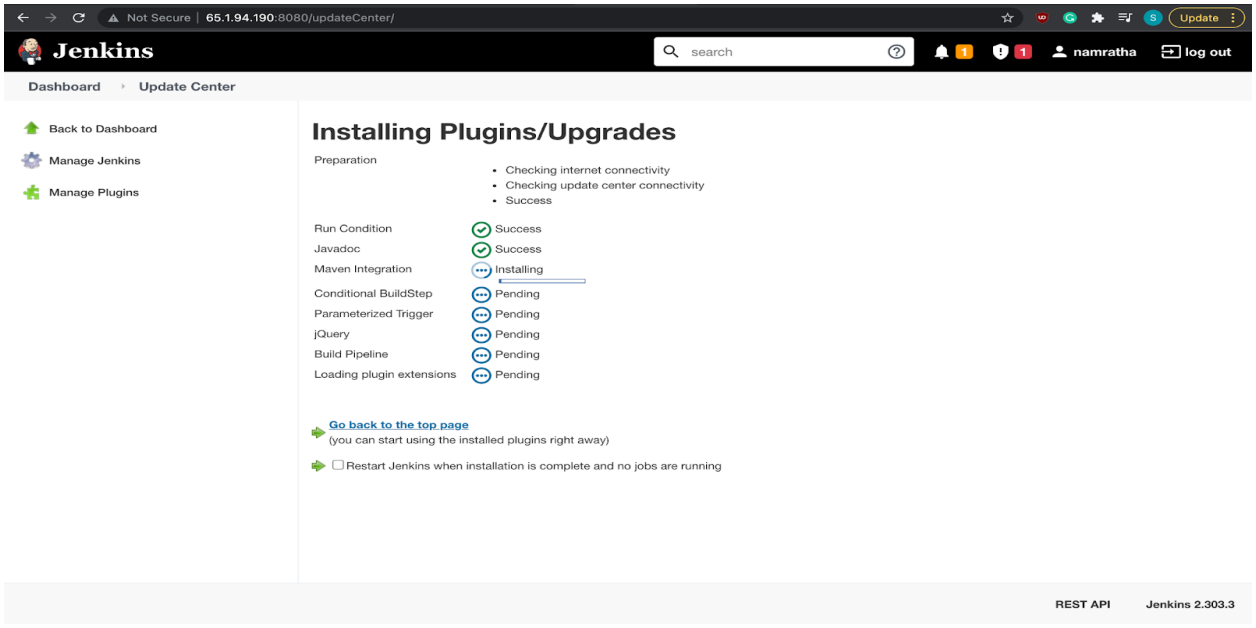
We can run the job of testing (job of slave 1) using pipeline view.

3.4.1 : Setting up the pipeline view

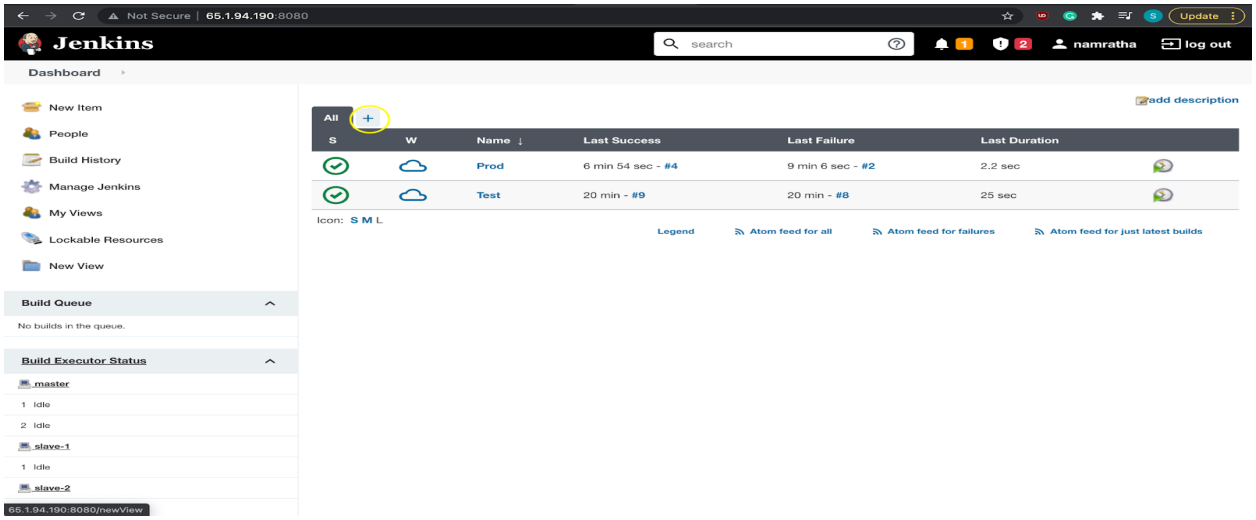
Installing build pipeline for enable pipeline view

Go to jenkins dashboard -> manage jenkins -> manage plugins -> available -> search "build pipeline" ->

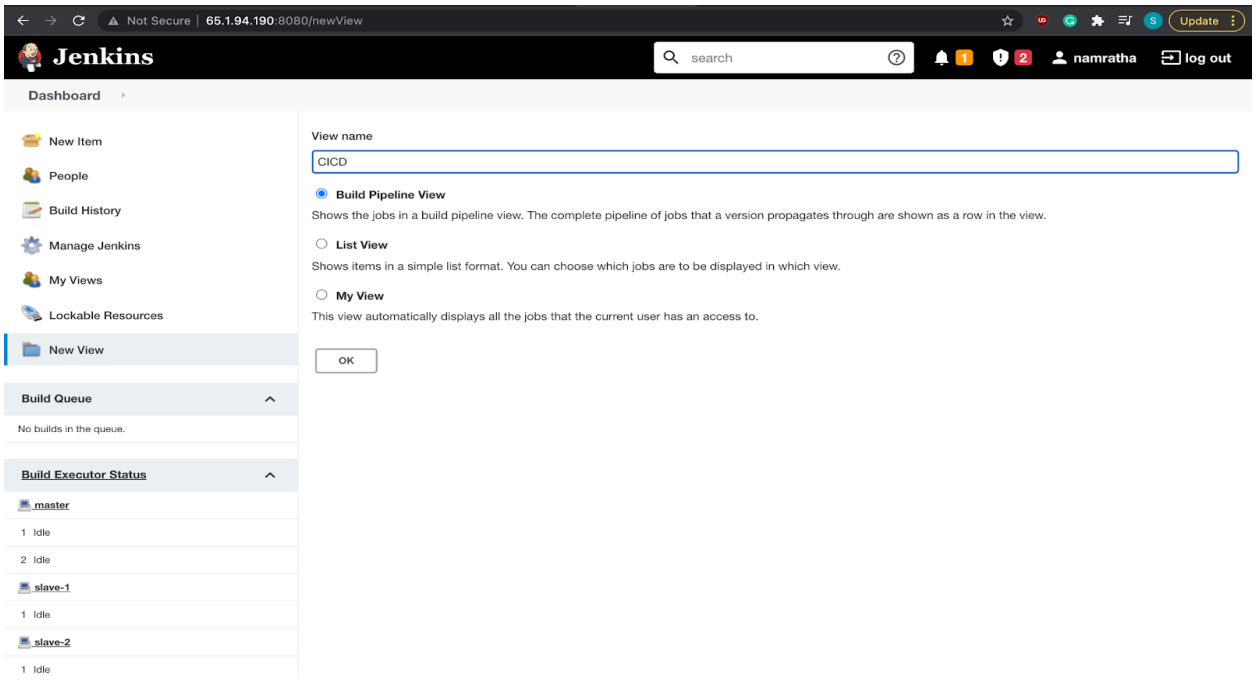
Select "install without restart"



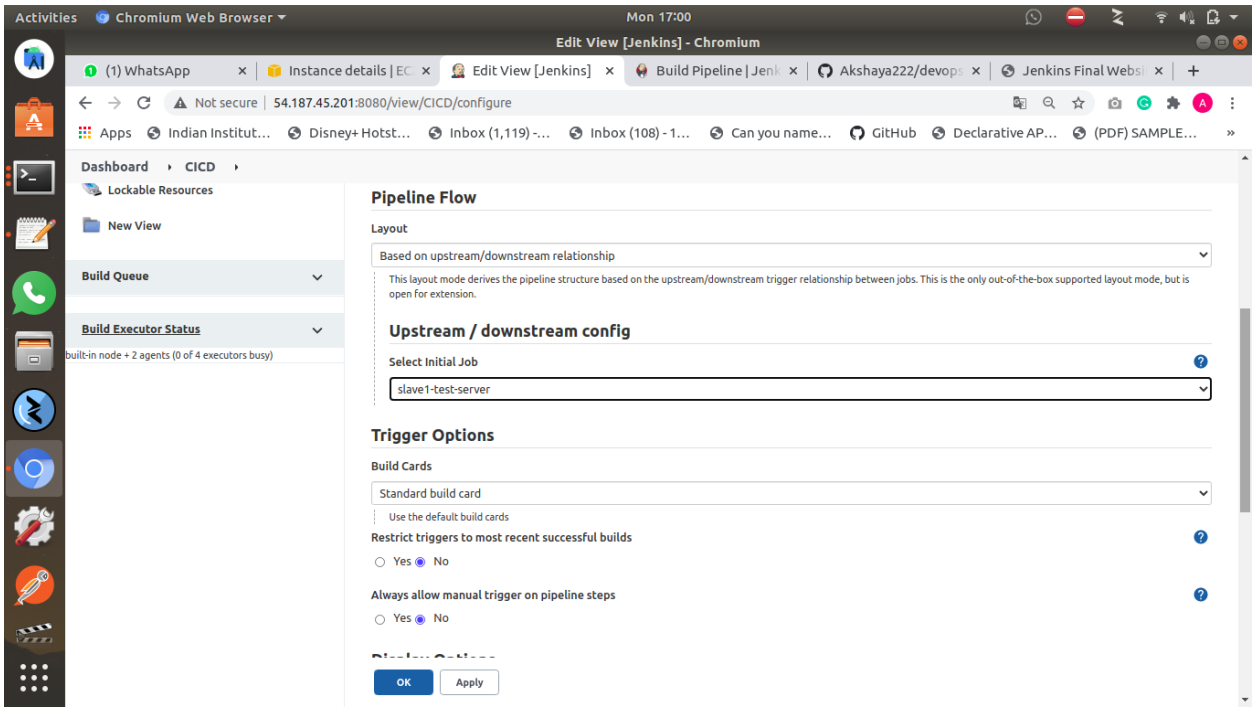
After installation, go back to the dashboard and click on "+".



Select build pipeline view and name the view name as CICD.
Enter the name- CICD
Build pipeline view title- CICD.
Select initial job - Test
Click Ok



Under pipeline options -> under select initial job -> select testing job -> click on apply -> click ok.



To trigger the pipeline we need to commit new changes and push the code to github.

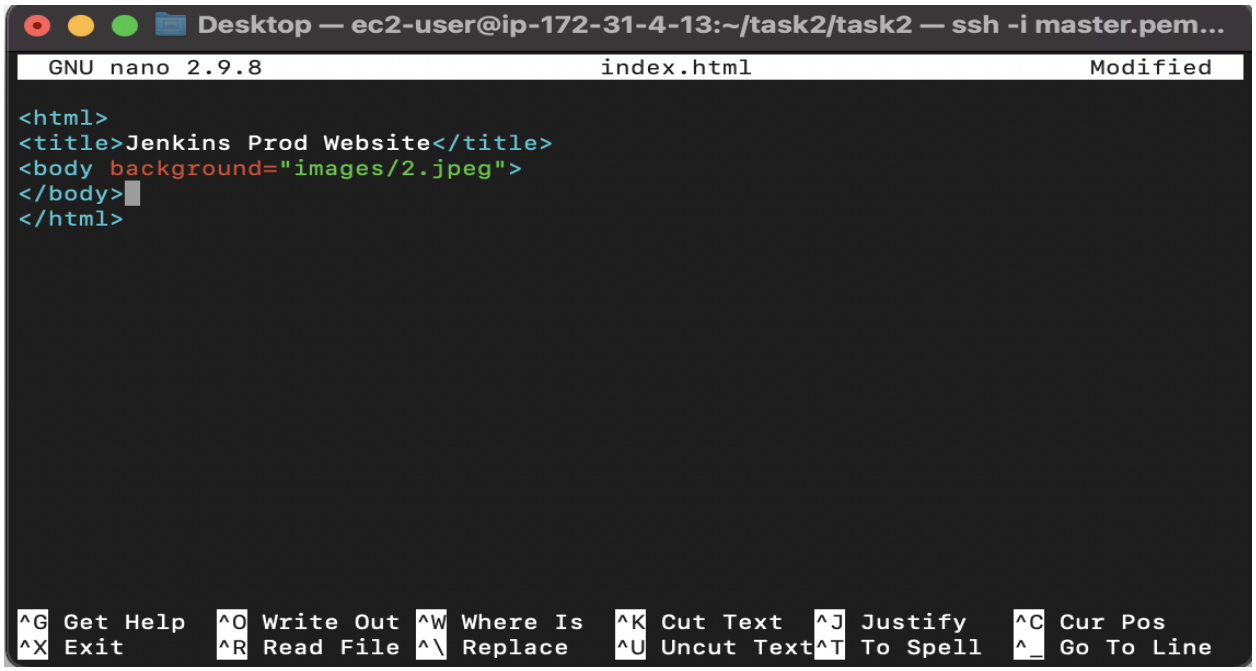
When the new push event occurs on github, the testing job automatically gets triggered with the help of webhook.

Go to jenkins master server to clone and make changes in the github.

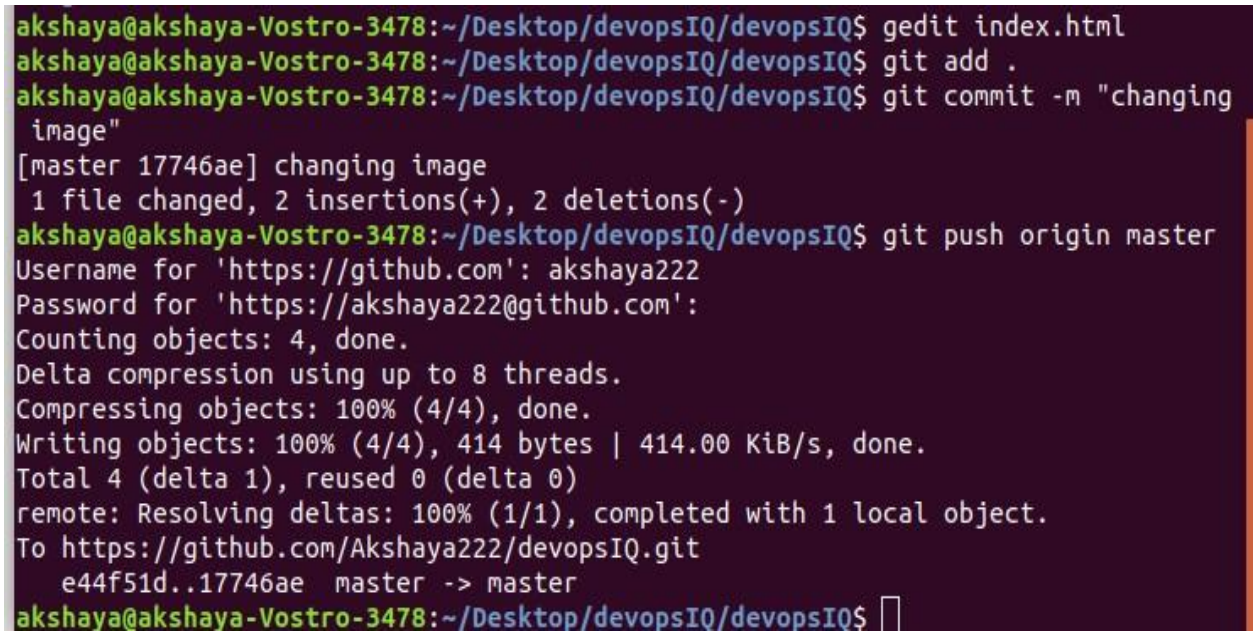
Cloning the repository

```
[[ec2-user@ip-172-31-4-13 ~]$ git clone https://github.com/Namratha-shivaraju/tas]
k2.git
Cloning into 'task2'...
remote: Enumerating objects: 28, done.
remote: Counting objects: 100% (28/28), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 28 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (28/28), 11.45 MiB | 7.56 MiB/s, done.
Resolving deltas: 100% (1/1), done.
[ec2-user@ip-172-31-4-13 ~]$
```

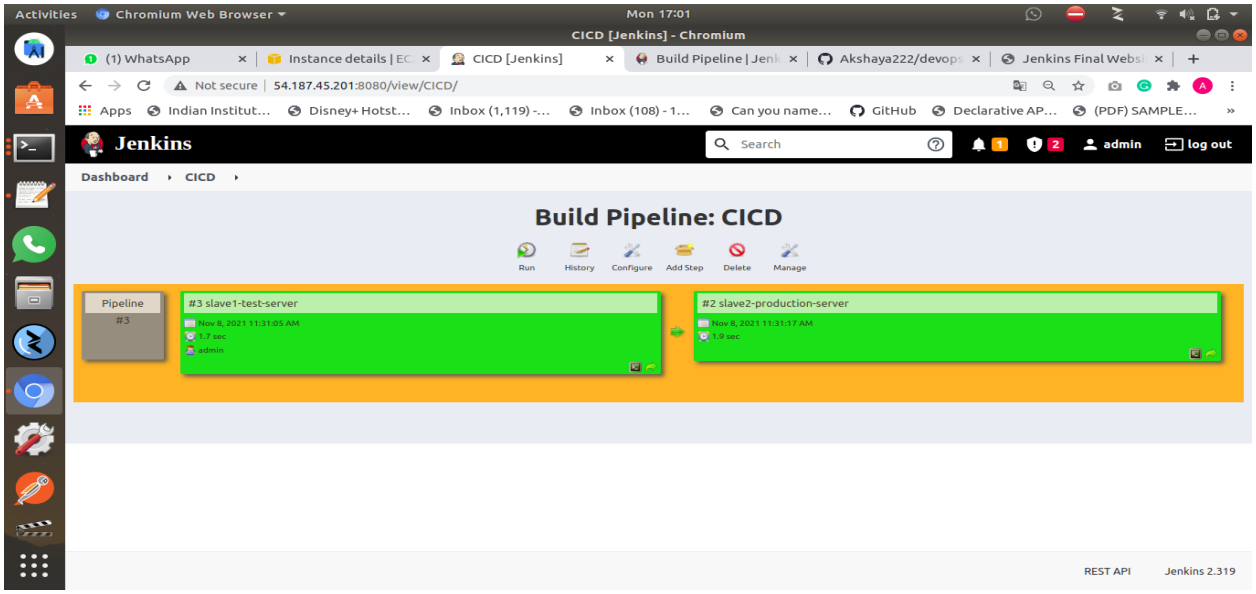
Editing index.html



Committing and pushing the changes to github repository



Go to pipeline view to see the jobs status



Both the jobs ran successfully.
To verify if the changes have been deployed on testing server

Go to browser and hit http://ipaddress_of_slave-1:82/devopsIQ

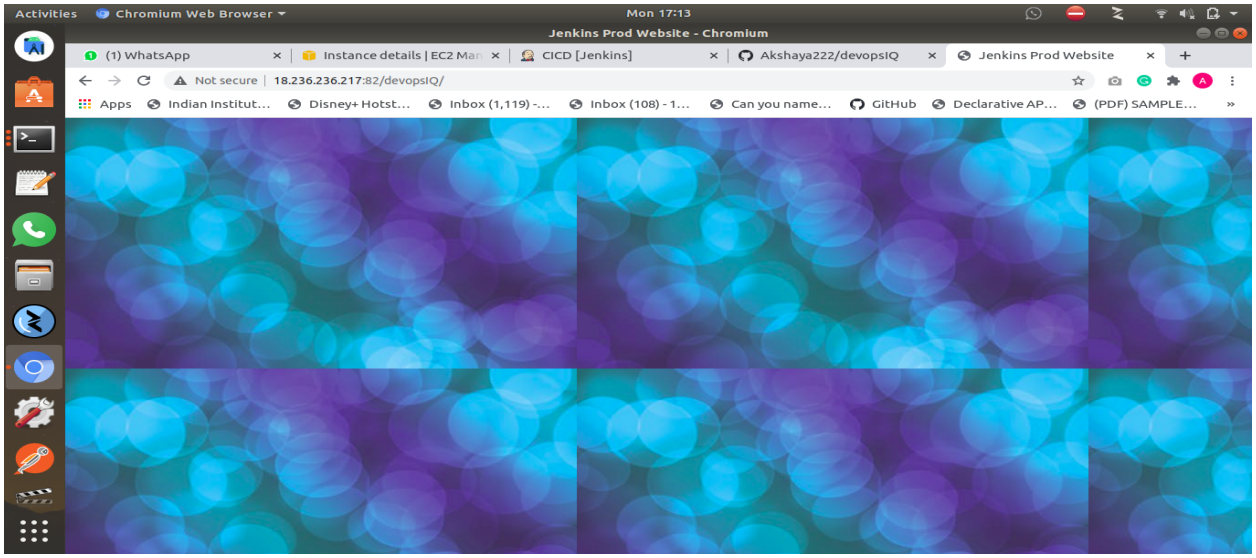
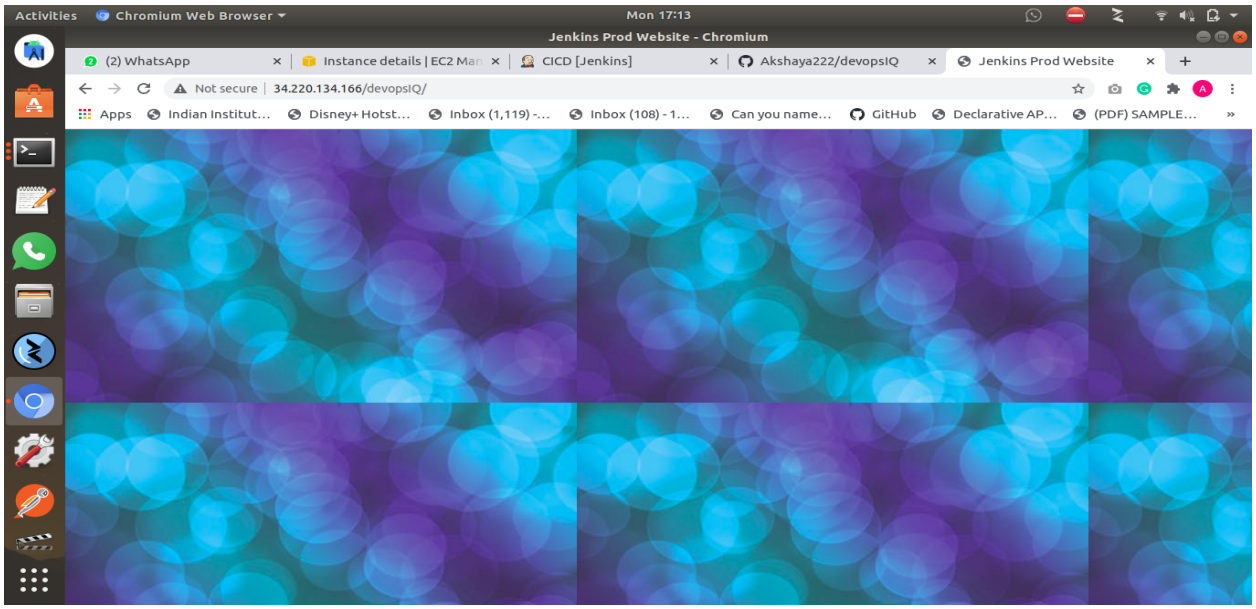


Image has been changed that means changes deployed successfully.So the testing job build was successful.

So the production job must have been triggered as the testing job was successful.

To verify if the changes have been deployed on production server

Go to browser and hit http://ipaddress_of_slave-2:80/devopsIQ



Changes have been successfully deployed to the production server.
Jenkins master slave pipeline has been successful.