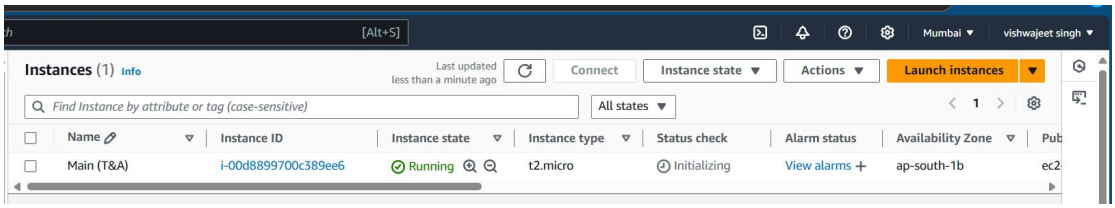
---you are hired as a DevOps engineer for Analytics Pvt Ltd. This company is a product based organization which uses Docker for their containerization needs within the company. The final product received a lot of traction in the first few weeks of launch. Now with the increasing demand, the organization needs to have a platform for automating deployment, scaling, and operations of application containers across clusters of hosts, As a DevOps engineer, you need
implement a DevOps life cycle, such that all the requirements are implemented without any change in the Docker containers in the testing environment. Up until now, this organization used to follow a monolithic architecture with just 2 developers. The product is present on *https://github.com/hshar/website.git*

Following are the specifications of life-cycle:

1. Git workflow should be implemented. Since the company follows monolithic architecture of Development you need to take care of version control. The release should happen only on 25th of every month.
2. Code build should be triggered once the commits are made in the master Branch.
3. The code should be containerized with the help of the Docker file, The Dockerfile should be built every time if there is a push to Git-Hub. Create a custom Docker image using a Dockerfile.
4. As per the requirement in the production server, you need to use the Kubernetes cluster and the containerized code from Docker hub should be deployed with 2 replicas. Create a NodePort service and configure the same for port 30008.
5. Create a Jenkins pipeline script to accomplish the above task.
6. For configuration management of the infrastructure, you need to deploy the configuration on the servers to install necessary software and configurations.
7. Using Terraform accomplish the task of infrastructure creation in the AWS cloud provider.

Step1: lets launch instance and name it as main. We will install ansible and terraform on this instance.



We named instance as Main(T&A), T stands for terraform and A stands for ansible. Now lets install teraform on this instance, we will use this to launch different machine/architecture.

Step2: Install Terraform

As we are using Ubuntu type instance. Check official documentation for installing the terraform.

https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli

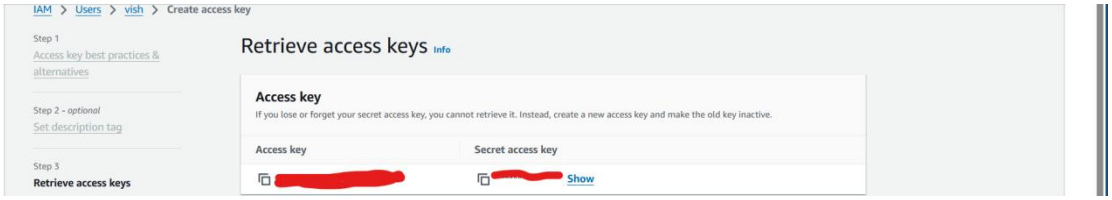
```
sudo apt-get update && sudo apt-get install -y gnupg software-properties-common
wget -O- https://apt.releases.hashicorp.com/gpg | \
gpg --dearmor | \
sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg > /dev/null
gpg --no-default-keyring \
--keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg \
--fingerprint
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | \
sudo tee /etc/apt/sources.list.d/hashicorp.list
sudo apt update
sudo apt-get install terraform
```

Use above command one by one for installing terraform

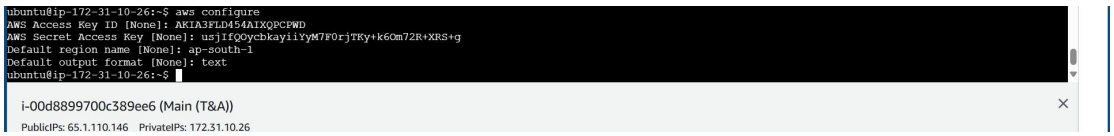
We have installed terraform successfully

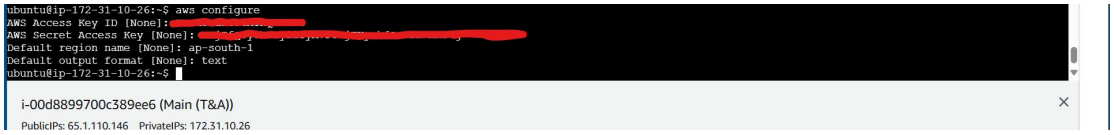
Step3: lets create directory and create .tf file. We will use this to launch three additional instance and which will be our kubernetes master and 2 slave instance where we will run the container

We will set aws configuration with the help of iam user



Run command aws configure to add the configuration



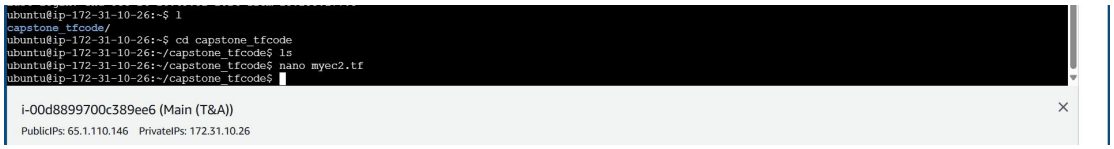


Now we have setup the aws configuration, we will use this in out terraform file

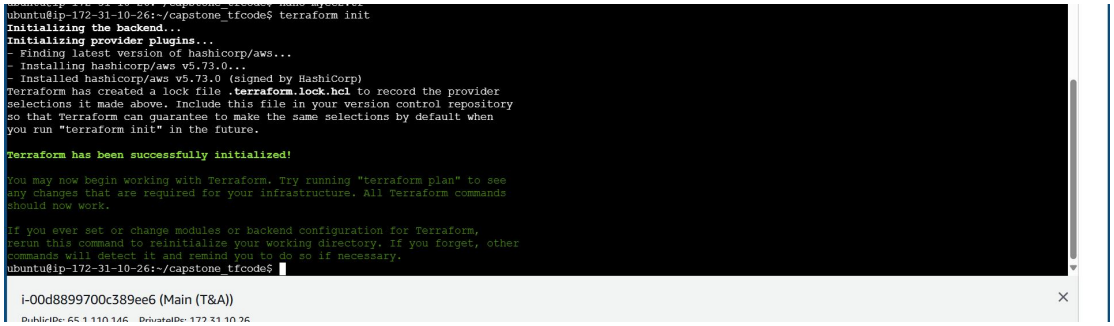
```
provider "aws" {
    profile= "default"
    region = "ap-south-1"
}
resource "aws_instance" "Kubernetes_Master" {
  ami          = "ami-09b0a86a2c84101e1"
```

```
  instance_type = "t2.medium"
  subnet_id = "subnet-0b4518708d215c36a"
  key_name = "capstone"
  tags = {
    Name = "k8s_master"
  }
}
resource "aws_instance" "Kubernetes_Slave1" {
  ami           = "ami-09b0a86a2c84101e1"
  instance_type = "t2.micro"
  subnet_id = "subnet-0b4518708d215c36a"
  key_name = "capstone"
  tags = {
    Name = "k8s_slave1"
  }
}
resource "aws_instance" "Kubernetes_Slave2" {
  ami           = "ami-09b0a86a2c84101e1"
  instance_type = "t2.micro"
  subnet_id = "subnet-0b4518708d215c36a"
  key_name = "capstone"
  tags = {
    Name = "k8s_slave2"
   }
}
```

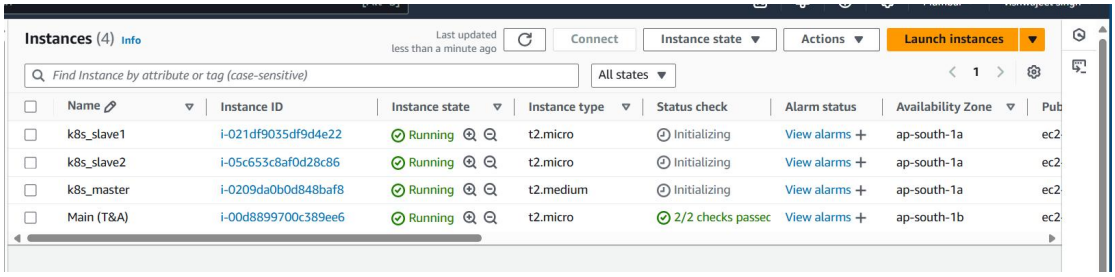
We will paste the above code in myec2.tf file and we will launch



Run terraform init , it will download required plugged in.



Run terraform apply to deploy it



We have successfully launched the three instance. Now we will work installing ansible on main machine which will help us to install required.

<u>Step 4 : connect with main machine and install ansible on it</u>

sudo apt update
sudo apt install software-properties-common
sudo add-apt-repository --yes --update ppa:ansible/ansible
sudo apt install ansible

Lets install by running above command one by one to install ansible

-------> we will set password-less ssh connection between between the instances,
Generate public key with the command “ssh-keygen -t rsa”

After running the above command we will have public key, cat the public key and copy it

Now this copied key will be pasted in authorized_keys of instance

```
ubuntu@ip-172-31-32-127:~$ nano .ssh/authorized_keys
ubuntu@ip-172-31-32-127:~$ cat .ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC5PRKtIiPktE8jzTPT07Gd135UzFx4iYj8herV4CmCT3VO07CgWjTJdggcfVMJZ+qnLarP05v0P8A0nKAoUbHnxHqY6ctlpnK6r9jNJC4Nh+x5CWZCcOXHxTo+4up
8gaohITvuDLizofpZyMZoDvCOTR5ZDTZkFjvRI0saPVw3d+/aSAvmPEisHKDlNs5gc4H97RcsPAUVpiDbFyp8os3RWAtbvu4s6aM3/JvMu4kk/vDcI2enaXt0wAIzJjUEVlyupp57TeoM+G6Tc5vtGhvfrNTIwpJUrP
ZVJ4sSDQ6ov2WZepzlZeZ1+Qlrrhx8VasclMcrHAif2BOIwj2iQBaD capstone
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQChVdKFifs5Q0LvcYi3OKjblN2XYd3zugdFS46z7phEpd+zTz2MtmEuMWCzpc/z4ld7MhVx20WUImqQt4F4ORRldKRFF6VEZdhgfShsP9Wwp/1sFRYelCWEVN53ZAL
R2P8QYwNX1AZRwL5fi8KKL0jJ5dgFTvVFd3hkvTAlBHfzdh+4+GFe6c527h8ozNduOuFrFX2YK4NLvMXMoneZcqppRSJJDLana32cweI8WBDcIiuWMxERaWnocz+R6DmM4coerniazFBRbKYWF46dAyC4+kHM0zALaw
Kv4Wr9MNOJyHDr4lKNZb+wxmi9HOGiwn/QB9Psp4SoP2dKt9OwyJjWvMbArEtwlWWAhWQ86vt3jflDR5+q6kemvUO940WfYrqWDi3fBlXJkmWCb8W+KtoQtRF6B6Qeo3NOtEfAjHG4RVBqDqZyZFVpsE6+hhHo9jiDW
t8CJsy47wC4jzwsE4NO0zuICRlu8/z2bAO1e2yKygwu85k4WUf6neY9+qlBZVM- ubuntu@ip-172-31-10-26
ubuntu@ip-172-31-32-127:~$
```

i-05c653c8af0d28c86 (k8s_slave2)

PublicIPs: 3.111.149.23 PrivateIPs: 172.31.32.127

Next we will add the private ip of the instances to complete the setup of ansible

Cd to /etc/ansible/hosts

```
[master]
172.31.47.55

[slave1]
172.31.34.103

[slave2]
172.31.32.127

^G Help      ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark   M-] To Bracket  M-Q Previous
^X Exit      ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy       ^Q Where Was    M-W Next
```

i-00d8899700c389ee6 (Main (T&A))

PublicIPs: 3.109.121.248 PrivateIPs: 172.31.10.26

<u>Step 5: as per above project we will install below tools on nodes</u>

Main (T&A) - Jenkins and java (for automation)
K8_master- Kubernetes, docker, java
K8slave1 - docker kuberntes
K8slave2- docker kubernetes

We will be creating K8s cluster of 3 nodes.

Lets create script so we can bash it using absile, according to above requirement, we will need three script

<u>Main.sh (jenkins and java)</u>

```
#!/bin/bash
sudo apt update
sudo apt install fontconfig openjdk-17-jre -y
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" \
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
/etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt update
sudo apt install jenkins -y
```

```
ubuntu@ip-172-31-8-213:~$ nano main.sh
ubuntu@ip-172-31-8-213:~$
```

i-034b8f66d8aaa0d3f (Main(T&A))

PublicIPs: 52.66.247.54 PrivateIPs: 172.31.8.213

Lets create playbook which will run aand install these script file

```
---
- name: Installation of Java and Jenkins
  hosts: localhost
  become: yes
  tasks:
    - name: Clean apt cache
      apt:
        autoclean: yes
      become: yes

    - name: Run the main.sh script
      script: main.sh
      become: yes
```

Now lets run it

We can see java and jenkins has been installed sucessfully on our main file. Now lets setup k8s cluser

<u>Step 6: create script file on master and add the below command and bash it. We are doing this manually as k8s will not be installed via ansible</u>

```
#!/bin/bash
set -euxo pipefail

# Kuernetes Variable Declaration

KUBERNETES_VERSION="1.29.0-1.1"

# disable swap
sudo swapoff -a

# keeps the swaf off during reboot
(crontab -l 2>/dev/null; echo "@reboot /sbin/swapoff -a") | crontab - || true
sudo apt-get update -y


# Install CRI-O Runtime

OS="xUbuntu_22.04"

VERSION="1.28"

# Create the .conf file to load the modules at bootup
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
```

```
br_netfilter
EOF

sudo modprobe overlay
sudo modprobe br_netfilter

# sysctl params required by setup, params persist across reboots
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF

# Apply sysctl params without reboot
sudo sysctl --system

cat <<EOF | sudo tee /etc/apt/sources.list.d/devel:kubic:libcontainers:stable.list
deb https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/ /
EOF
cat <<EOF | sudo tee /etc/apt/sources.list.d/devel:kubic:libcontainers:stable:cri-o:$VERSION.list
deb http://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-
o:/$VERSION/$OS/ /
EOF

curl -L https://download.opensuse.org/repositories/devel:kubic:libcontainers:stable:cri-
o:$VERSION/$OS/Release.key | sudo apt-key --keyring /etc/apt/trusted.gpg.d/libcontainers.gpg add -
curl -L
https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/Release.key |
sudo apt-key --keyring /etc/apt/trusted.gpg.d/libcontainers.gpg add -

sudo apt-get update
sudo apt-get install cri-o cri-o-runc -y

sudo systemctl daemon-reload
sudo systemctl enable crio --now

echo "CRI runtime installed susccessfully"

# Install kubelet, kubectl and Kubeadm

sudo apt-get update -y
sudo apt-get install -y apt-transport-https ca-certificates curl gpg

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | sudo gpg --dearmor -o
/etc/apt/keyrings/kubernetes-1-28-apt-keyring.gpg
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-1-28-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes-1.28.list

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo gpg --dearmor -o
/etc/apt/keyrings/kubernetes-1-29-apt-keyring.gpg
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-1-29-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes-1.29.list

sudo apt-get update -y
sudo apt-get install -y kubelet="$KUBERNETES_VERSION" kubectl="$KUBERNETES_VERSION"
kubeadm="$KUBERNETES_VERSION"
sudo apt-get update -y
```

```
sudo apt-mark hold kubelet kubeadm kubectl

sudo apt-get install -y jq

local_ip="$(ip --json addr show eth0 | jq -r '.[0].addr_info[] | select(.family == "inet") | .local')"
cat > /etc/default/kubelet << EOF
KUBELET_EXTRA_ARGS=--node-ip=$local_ip
EOF


sudo bash install.sh
```

```
## Execute ONLY on "Master Node" ( one-by-one ) :

#!/bin/bash
# Setup for Control Plane (Master) servers

set -euxo pipefail

# If you need public access to API server using the servers Public IP adress, change PUBLIC_IP_ACCESS
to true.

PUBLIC_IP_ACCESS="true"
NODENAME=$(hostname -s)
POD_CIDR="192.168.0.0/16"

# Pull required images

sudo kubeadm config images pull

# Initialize kubeadm based on PUBLIC_IP_ACCESS

if [[ "$PUBLIC_IP_ACCESS" == "false" ]]; then

   MASTER_PRIVATE_IP=$(ip addr show eth0 | awk '/inet / {print $2}' | cut -d/ -f1)
   sudo kubeadm init --apiserver-advertise-address="$MASTER_PRIVATE_IP" --apiserver-cert-extra-
sans="$MASTER_PRIVATE_IP" --pod-network-cidr="$POD_CIDR" --node-name "$NODENAME" --
ignore-preflight-errors Swap

elif [[ "$PUBLIC_IP_ACCESS" == "true" ]]; then

   MASTER_PUBLIC_IP=$(curl ifconfig.me && echo "")
   sudo kubeadm init --control-plane-endpoint="$MASTER_PUBLIC_IP" --apiserver-cert-extra-
sans="$MASTER_PUBLIC_IP" --pod-network-cidr="$POD_CIDR" --node-name "$NODENAME" --ignore-
preflight-errors Swap

else
   echo "Error: MASTER_PUBLIC_IP has an invalid value: $PUBLIC_IP_ACCESS"
   exit 1
fi

# Configure kubeconfig

mkdir -p "$HOME"/.kube
sudo cp -i /etc/kubernetes/admin.conf "$HOME"/.kube/config
sudo chown "$(id -u)":"$(id -g)" "$HOME"/.kube/config
```

Install Claico Network Plugin Network

kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml

```
ubuntu@ip-172-31-38-48:~$ sudo nano  install.sh
ubuntu@ip-172-31-38-48:~$ sudo bash install.sh
+ KUBERNETES_VERSION=1.29.0-1.1
+ sudo swapoff -a
+ crontab -l
```

i-03fb281f6ea68cd90 (K8s_master)

PublicIPs: 3.108.196.157 PrivateIPs: 172.31.38.48

```
No VM guests are running outdated hypervisor (qemu) binaries on this host.
++ ip --json addr show eth0
++ jq -r '.[0].addr_info[] | select(.family == "inet") | .local'
+ local_ip=172.31.38.48
+ cat
ubuntu@ip-172-31-38-48:~$
```

i-03fb281f6ea68cd90 (K8s_master)

PublicIPs: 3.108.196.157 PrivateIPs: 172.31.38.48

Similary we will run these commands on slave nodes also

Step7: Now we have to run few command manually, run below command on master node one by one to setup k8s

Execute ONLY on "Master Node" (one-by-one) :

```
#!/bin/bash
# Setup for Control Plane (Master) servers

set -euxo pipefail

# If you need public access to API server using the servers Public IP adress, change PUBLIC_IP_ACCESS to true.

PUBLIC_IP_ACCESS="true"
NODENAME=$(hostname -s)
POD_CIDR="192.168.0.0/16"

# Pull required images

sudo kubeadm config images pull

# Initialize kubeadm based on PUBLIC_IP_ACCESS

if [[ "$PUBLIC_IP_ACCESS" == "false" ]]; then

   MASTER_PRIVATE_IP=$(ip addr show eth0 | awk '/inet / {print $2}' | cut -d/ -f1)
   sudo kubeadm init --apiserver-advertise-address="$MASTER_PRIVATE_IP" --apiserver-cert-extra-sans="$MASTER_PRIVATE_IP" --pod-network-cidr="$POD_CIDR" --node-name "$NODENAME" --ignore-preflight-errors Swap

elif [[ "$PUBLIC_IP_ACCESS" == "true" ]]; then

   MASTER_PUBLIC_IP=$(curl ifconfig.me && echo "")
   sudo kubeadm init --control-plane-endpoint="$MASTER_PUBLIC_IP" --apiserver-cert-extra-sans="$MASTER_PUBLIC_IP" --pod-network-cidr="$POD_CIDR" --node-name "$NODENAME" --ignore-preflight-errors Swap

else
   echo "Error: MASTER_PUBLIC_IP has an invalid value: $PUBLIC_IP_ACCESS"
   exit 1
```

fi

Configure kubeconfig

mkdir -p "$HOME"/.kube
sudo cp -i /etc/kubernetes/admin.conf "$HOME"/.kube/config
sudo chown "$(id -u)":"$(id -g)" "$HOME"/.kube/config

Install Claico Network Plugin Network

kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml

Once we run all the commands we will have join commands

```
kubeadm join 3.108.196.157:6443 --token b3wkxl.oa6x3p8ejcktxyo0 \
        --discovery-token-ca-cert-hash sha256:06f518e9c107a89dbf292f396f26d2845b7e0e9f27a7854b87e7948329f1d3b0
ubuntu@ip-172-31-38-48:~$ mkdir -p "$HOME"/.kube
sudo cp -i /etc/kubernetes/admin.conf "$HOME"/.kube/config
sudo chown "$(id -u)":"$(id -g)" "$HOME"/.kube/config
+ mkdir -p /home/ubuntu/.kube
```

i-03fb281f6ea68cd90 (K8s_master)
PublicIPs: 3.108.196.157 PrivateIPs: 172.31.38.48

We will run this join commands on slave/worker node

Now go to worker node and run below command
sudo kubeadm reset pre-flight checks

After that we will

```
ubuntu@ip-172-31-34-193:~$ sudo su
root@ip-172-31-34-193:/home/ubuntu# kubeadm join 3.108.196.157:6443 --token b3wkxl.oa6x3p8ejcktxyo0 \
        --discovery-token-ca-cert-hash sha256:06f518e9c107a89dbf292f396f26d2845b7e0e9f27a7854b87e7948329f1d3b0 --v=5
I1026 14:36:37.635441     6292 join.go:413] [preflight] found NodeName empty; using OS hostname as NodeName
I1026 14:36:37.635709     6292 initconfiguration.go:122] detected and using CRI socket: unix:///var/run/crio/crio.sock
[preflight] Running pre-flight checks
I1026 14:36:37.635821     6292 preflight.go:93] [preflight] Running general checks
```

i-0155b5b6ecc29200a (K8s_slave1)
PublicIPs: 13.235.23.121 PrivateIPs: 172.31.34.193

```
This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

root@ip-172-31-34-193:/home/ubuntu#
```

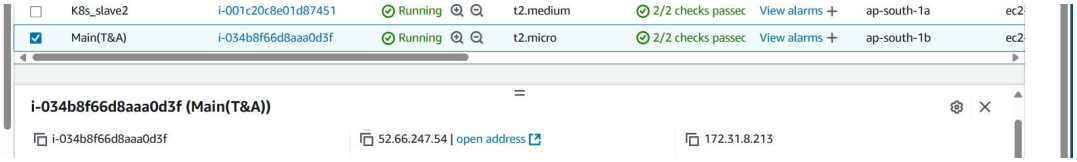
i-0155b5b6ecc29200a (K8s_slave1)
PublicIPs: 13.235.23.121 PrivateIPs: 172.31.34.193

```
This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

root@ip-172-31-36-52:/home/ubuntu#
```

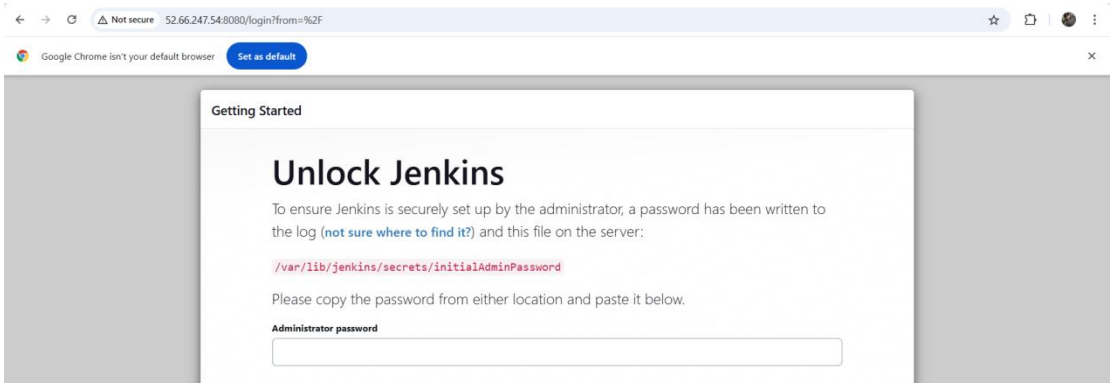
i-001c20c8e01d87451 (K8s_slave2)
PublicIPs: 15.206.145.43 PrivateIPs: 172.31.36.52

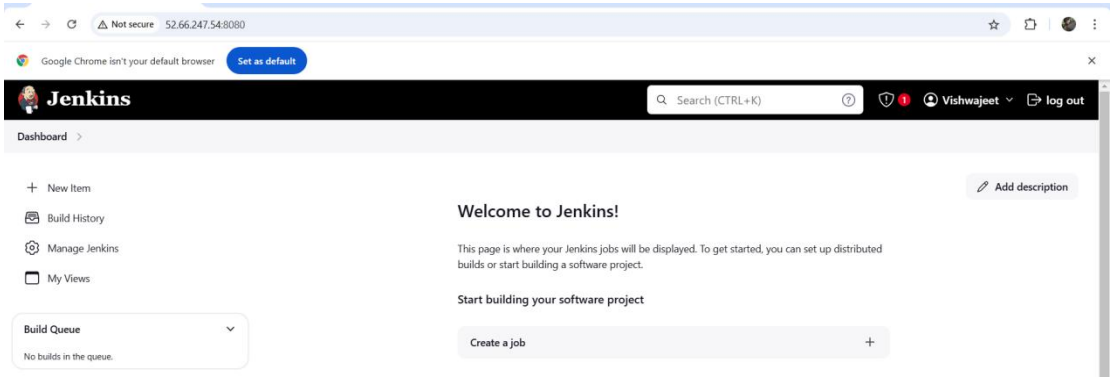
```
Last login: Sat Oct 26 13:59:35 2024 from 13.233.177.5
ubuntu@ip-172-31-38-48:~$ kubectl get nodes
NAME                STATUS   ROLES           AGE   VERSION
ip-172-31-34-193    Ready    <none>          17m   v1.29.0
ip-172-31-36-52     Ready    <none>          76s   v1.29.0
ip-172-31-38-48     Ready    control-plane   36m   v1.29.0
ubuntu@ip-172-31-38-48:~$
```

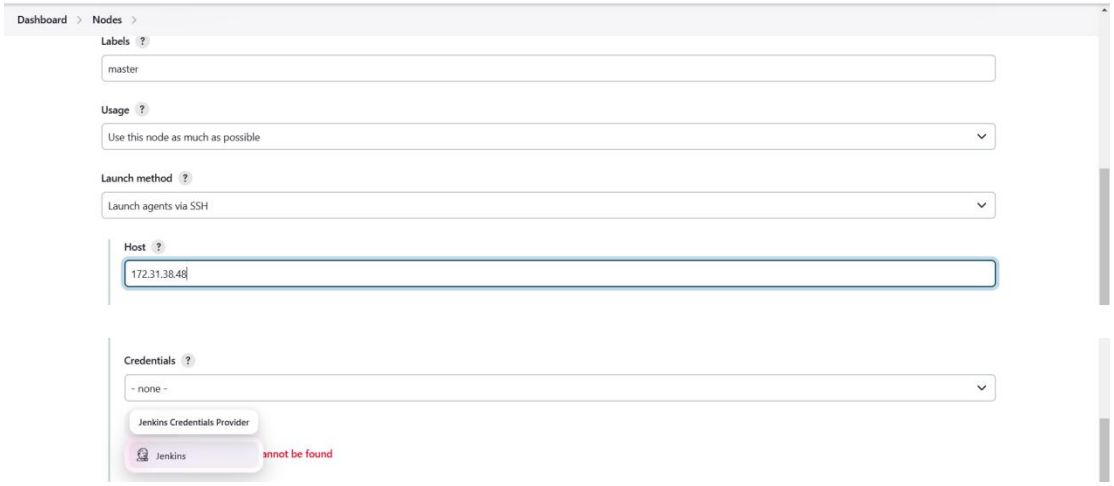
i-03fb281f6ea68cd90 (K8s_master)
PublicIPs: 3.108.196.157 PrivateIPs: 172.31.38.48

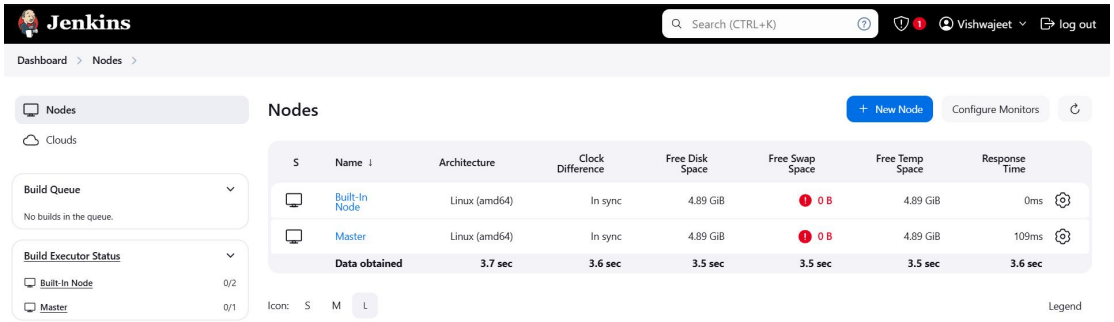


Step7: lets set the jenkins





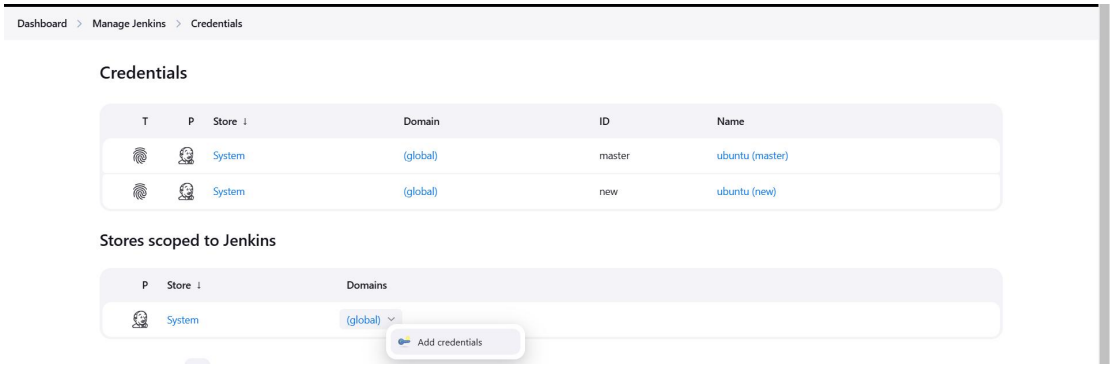


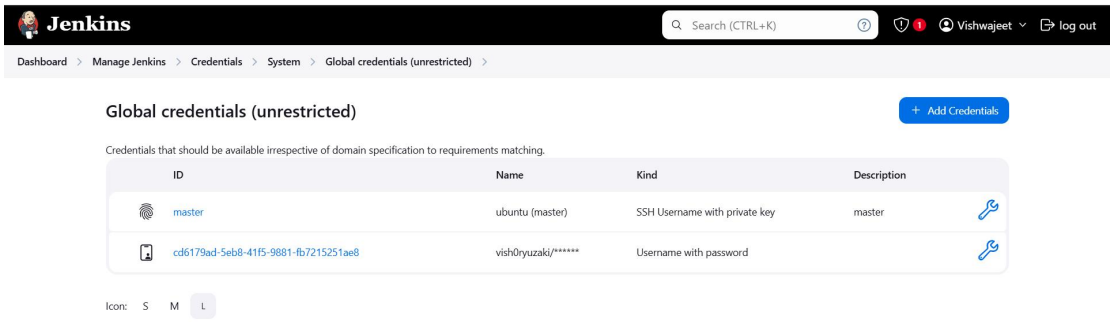


Node has been added succesfully.

Step 8: lets add dockerhub cred

Go to manage jenkins and then credentials

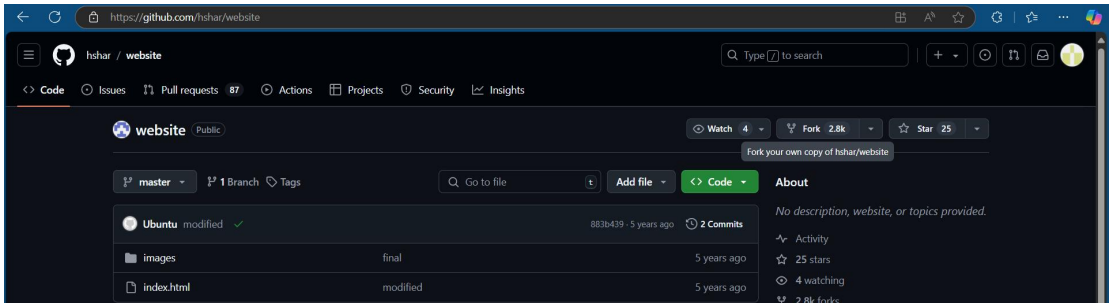


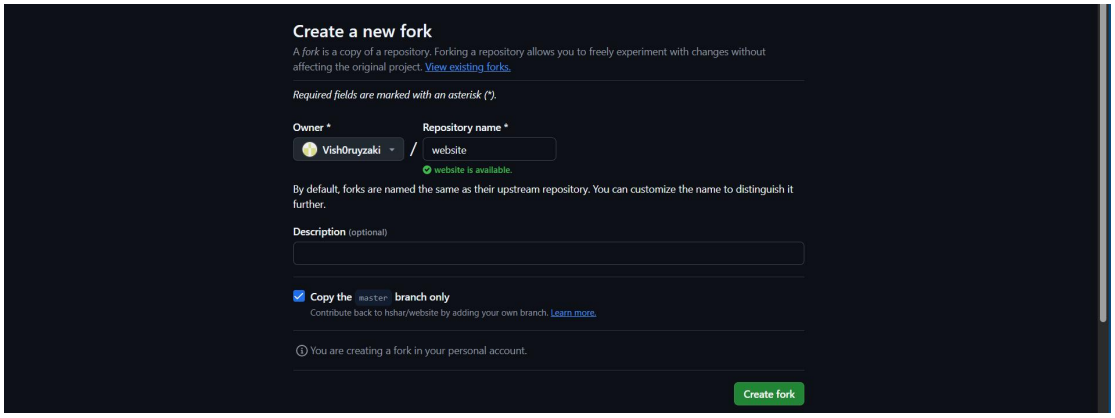


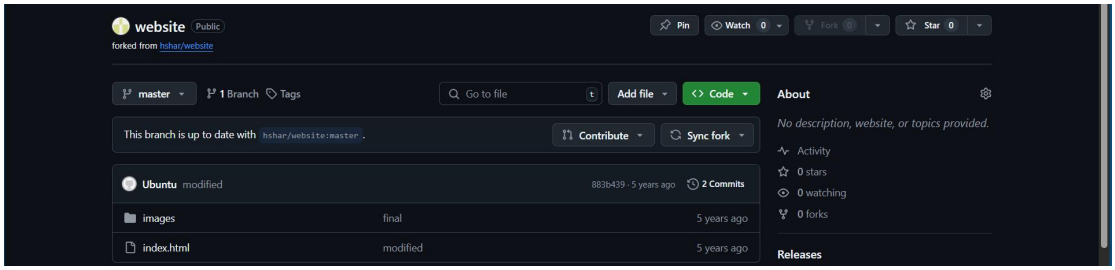
Added the credentails

Step 9: lets fork the repo, go to below repo

https://github.com/hshar/website.git

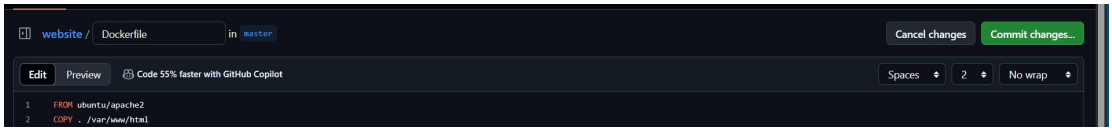


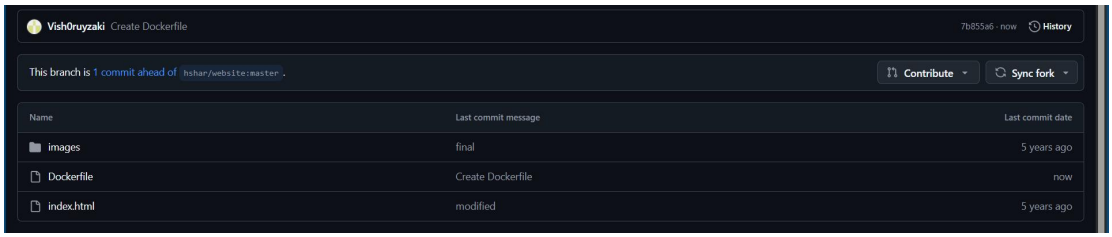




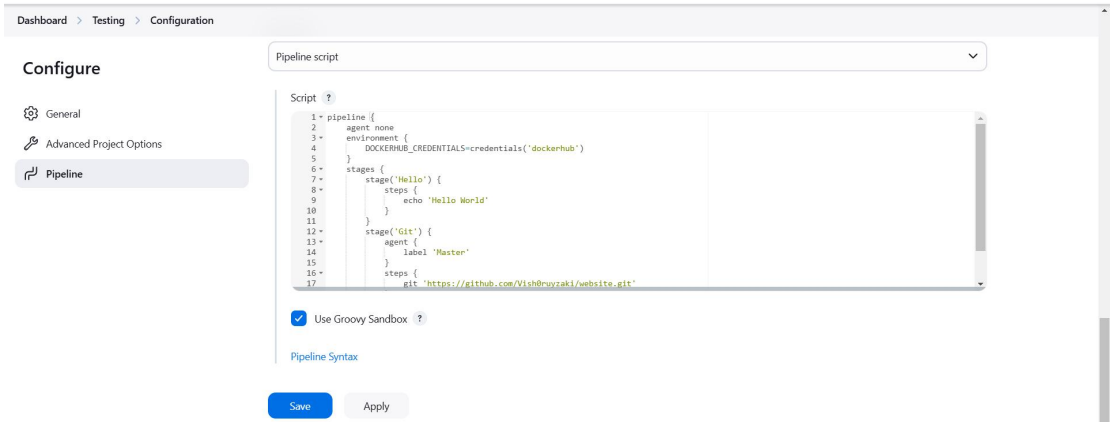
Step 10: lets creeate dockerfile inside the forked repo

FROM ubuntu/apache2
COPY . /var/www/html

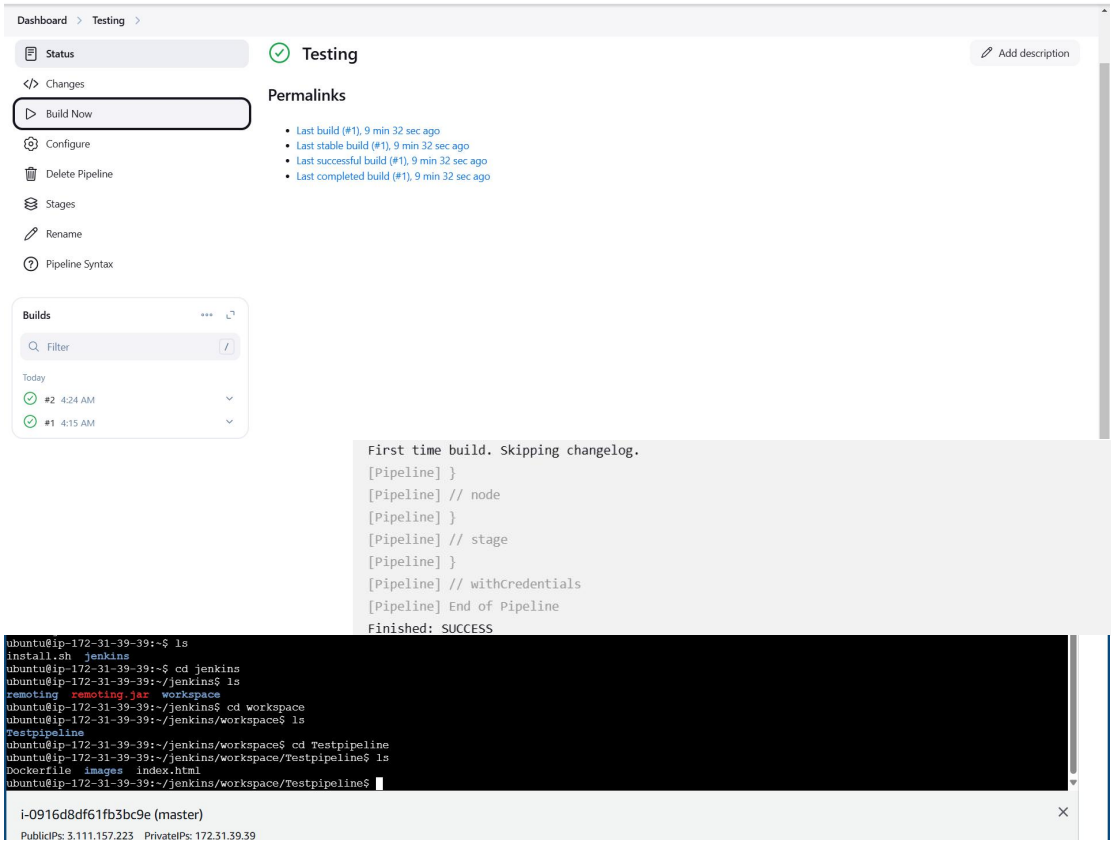




Step 11: lets create the job and pipline

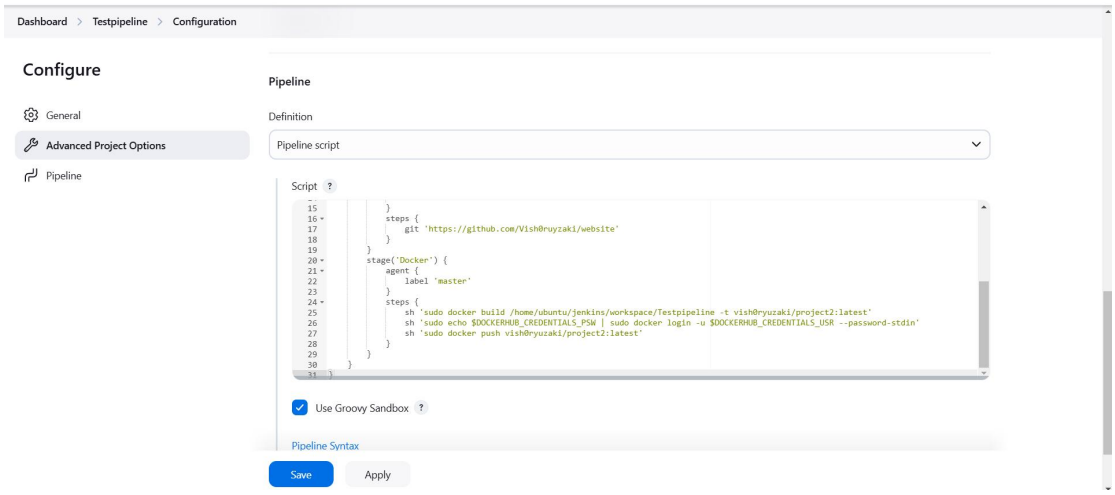


Lets run and check

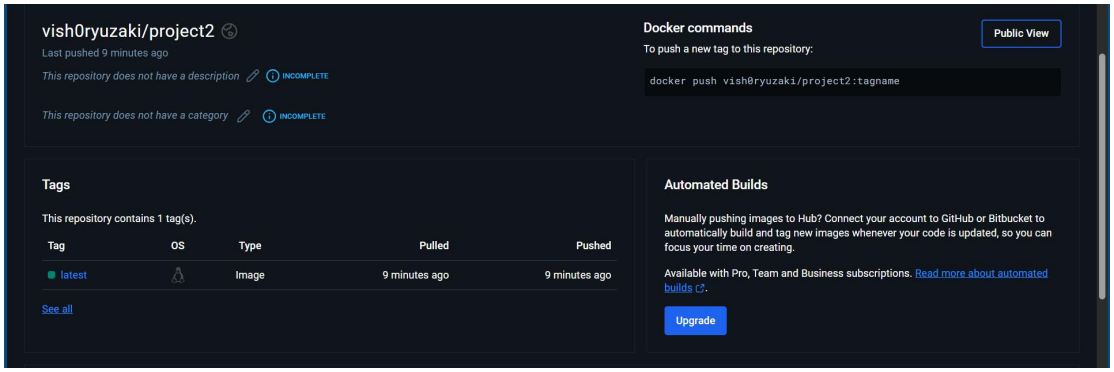


We can see the files has been imported from github to the master instance

Now we will create image and push it to docker hub



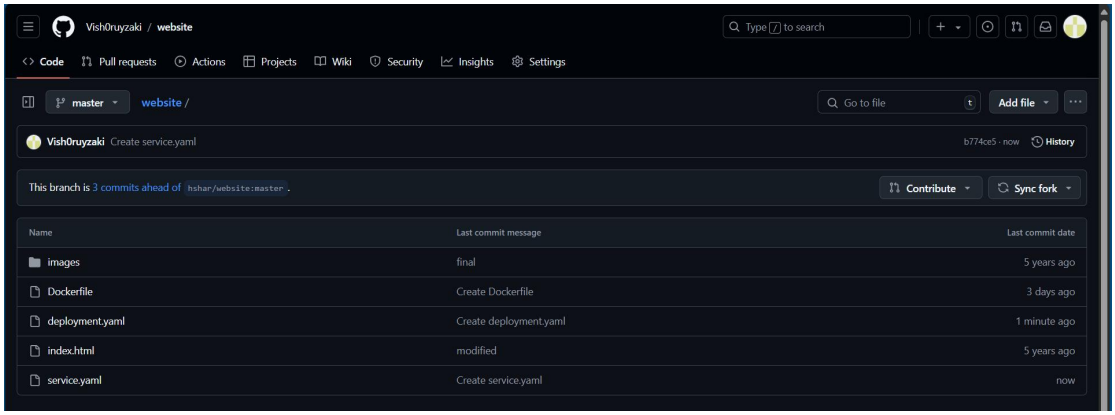

```
pipeline {
  agent none
  environment {
    DOCKERHUB_CREDENTIALS=credentials('cd6179ad-5eb8-41f5-9881-fb7215251ae8')
  }
  stages {
    stage('Hello') {
      steps {
        echo 'Hello World'
      }
    }
    stage('Git') {
      agent {
        label 'master'
      }
      steps {
        git 'https://github.com/Vish0ruyzaki/website'
      }
    }
    stage('Docker') {
      agent {
        label 'master'
      }
      steps {
        sh 'sudo docker build /home/ubuntu/jenkins/workspace/Testpipeline -t
vish0ryuzaki/project2:latest'
        sh 'sudo echo $DOCKERHUB_CREDENTIALS_PSW | sudo docker login -u
$DOCKERHUB_CREDENTIALS_USR --password-stdin'
        sh 'sudo docker push vish0ryuzaki/project2:latest'
      }
    }
  }
}
```



Step10: now lets create the yaml file to deply the container and nodeport. We will add this on github

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: vish0ruyzaki/project2:latest
        ports:
        - containerPort: 80
```

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30008
```

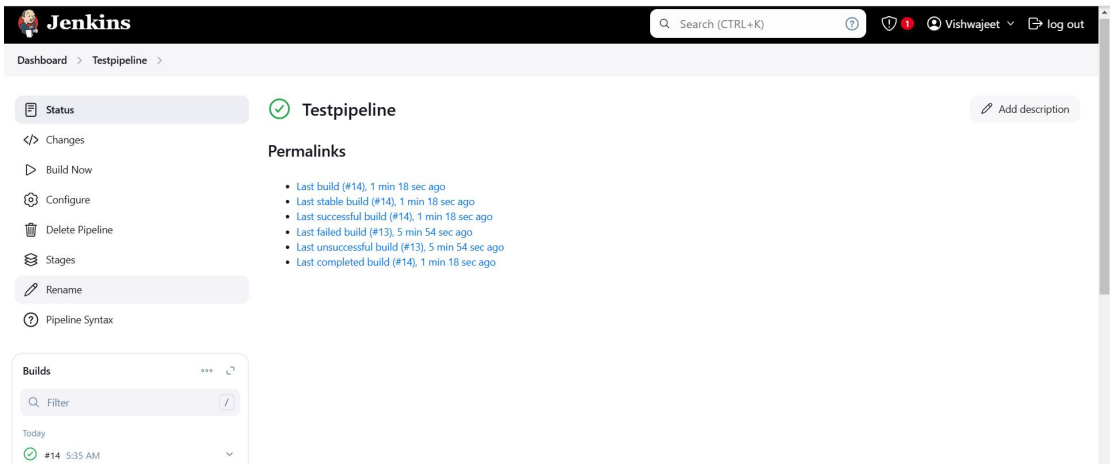


Step11: now we will create next pipeline script which will be below

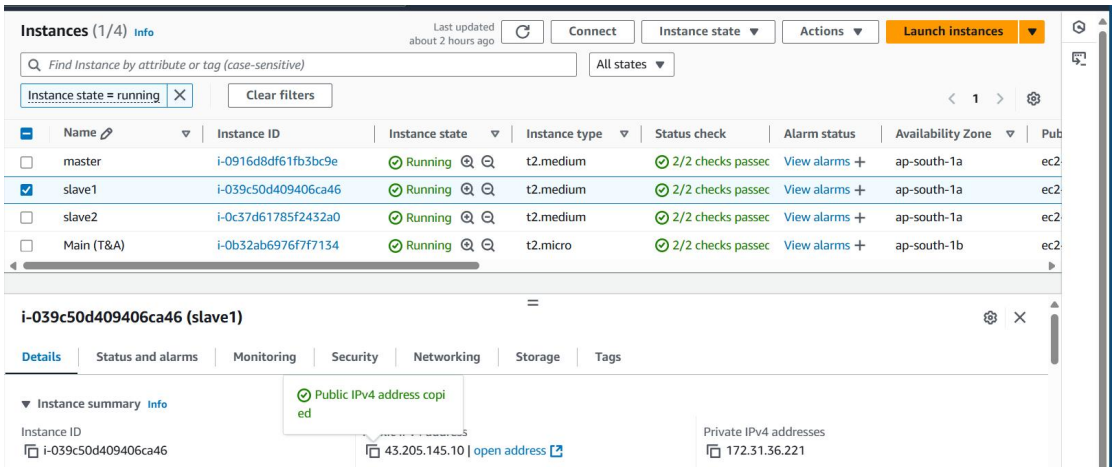
```
pipeline {
  agent none
  environment {
    DOCKERHUB_CREDENTIALS=credentials('cd6179ad-5eb8-41f5-9881-fb7215251ae8')
  }
  stages {
    stage('Hello') {
      steps {
        echo 'Hello World'
      }
    }
    stage('Git') {
      agent {
        label 'master'
      }
      steps {
        git 'https://github.com/Vish0ruyzaki/website'
      }
    }
    stage('Docker') {
      agent {
        label 'master'
      }
      steps {
        sh 'sudo docker build /home/ubuntu/jenkins/workspace/Testpipeline -t
vish0ryuzaki/project2:latest'
        sh 'sudo echo $DOCKERHUB_CREDENTIALS_PSW | sudo docker login -u
$DOCKERHUB_CREDENTIALS_USR --password-stdin'
        sh 'sudo docker push vish0ryuzaki/project2:latest'
      }
    }
    stage('K8s') {
      agent {
        label 'master'
      }
      steps {
        sh 'kubectl apply -f deployment.yaml'
        sh 'kubectl apply -f service.yaml'
      }
    }
  }
```

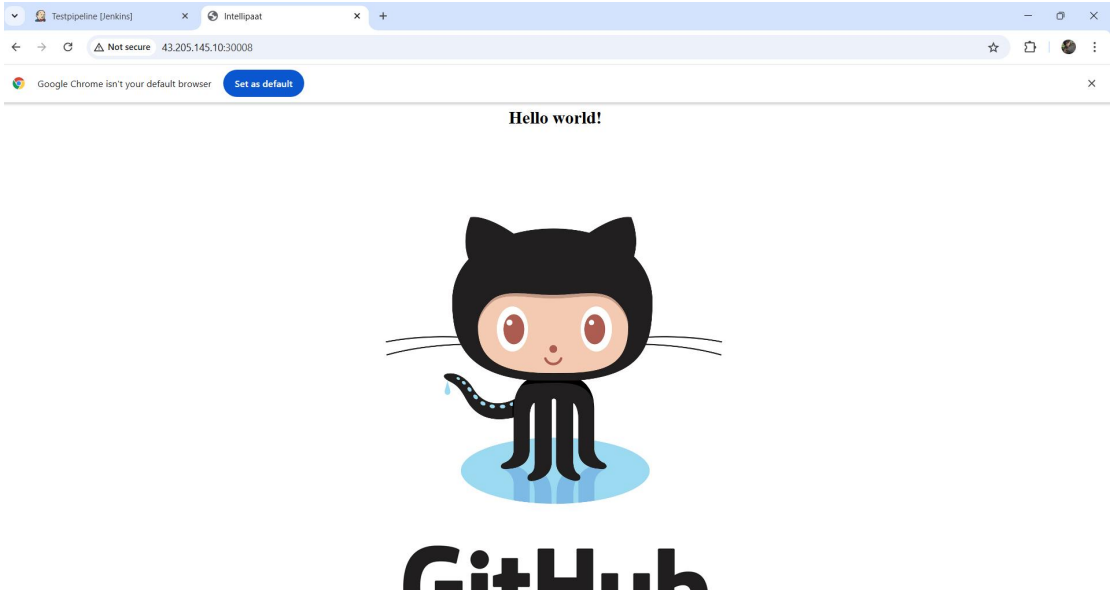
}

Lets run it

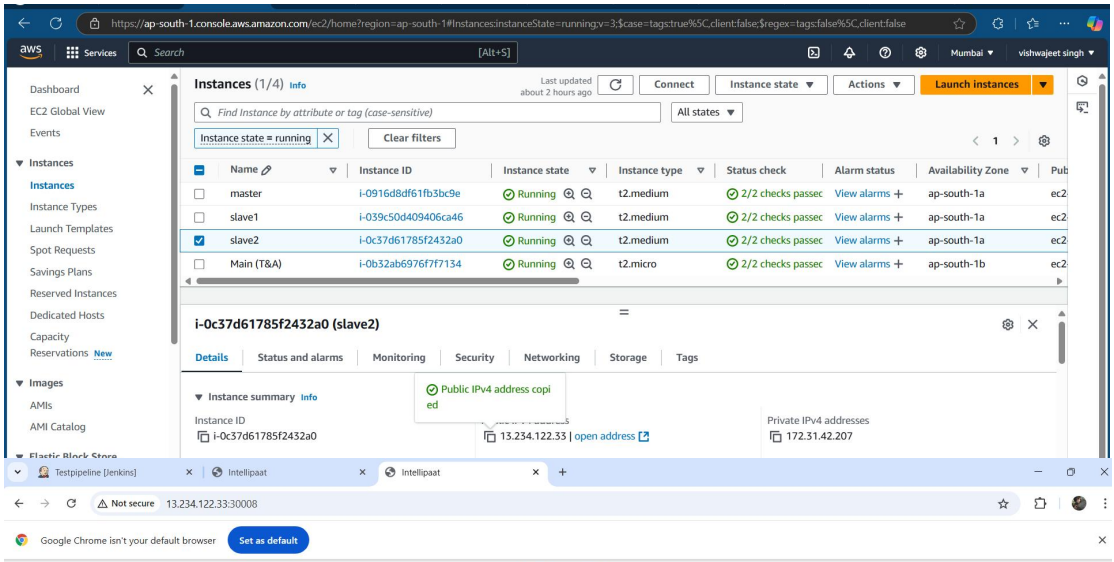


Now lets check if we can see the project on the slave ip

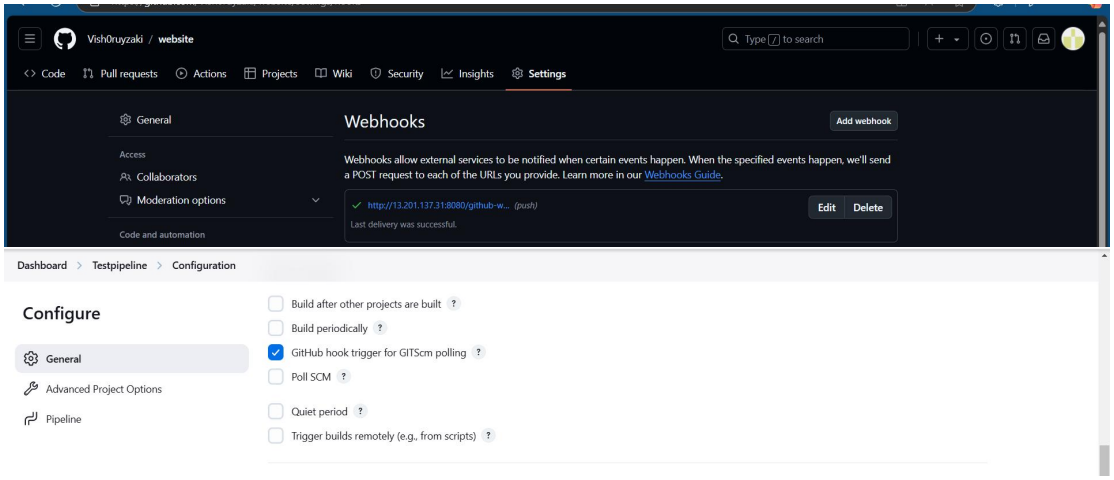




Lets check the second slave



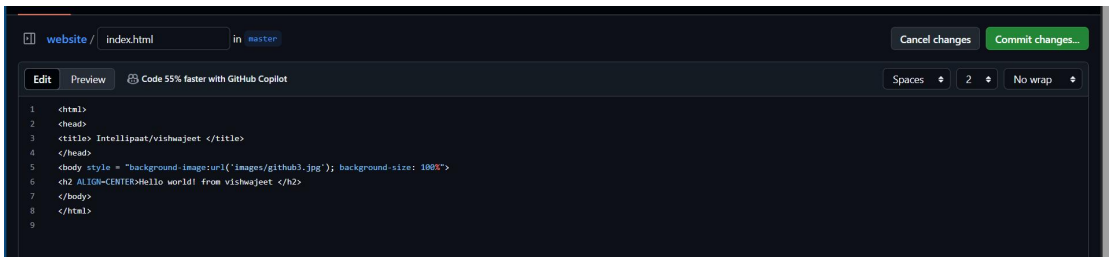
Step12: to automate the pipeline, we will steup webhook on the gitub



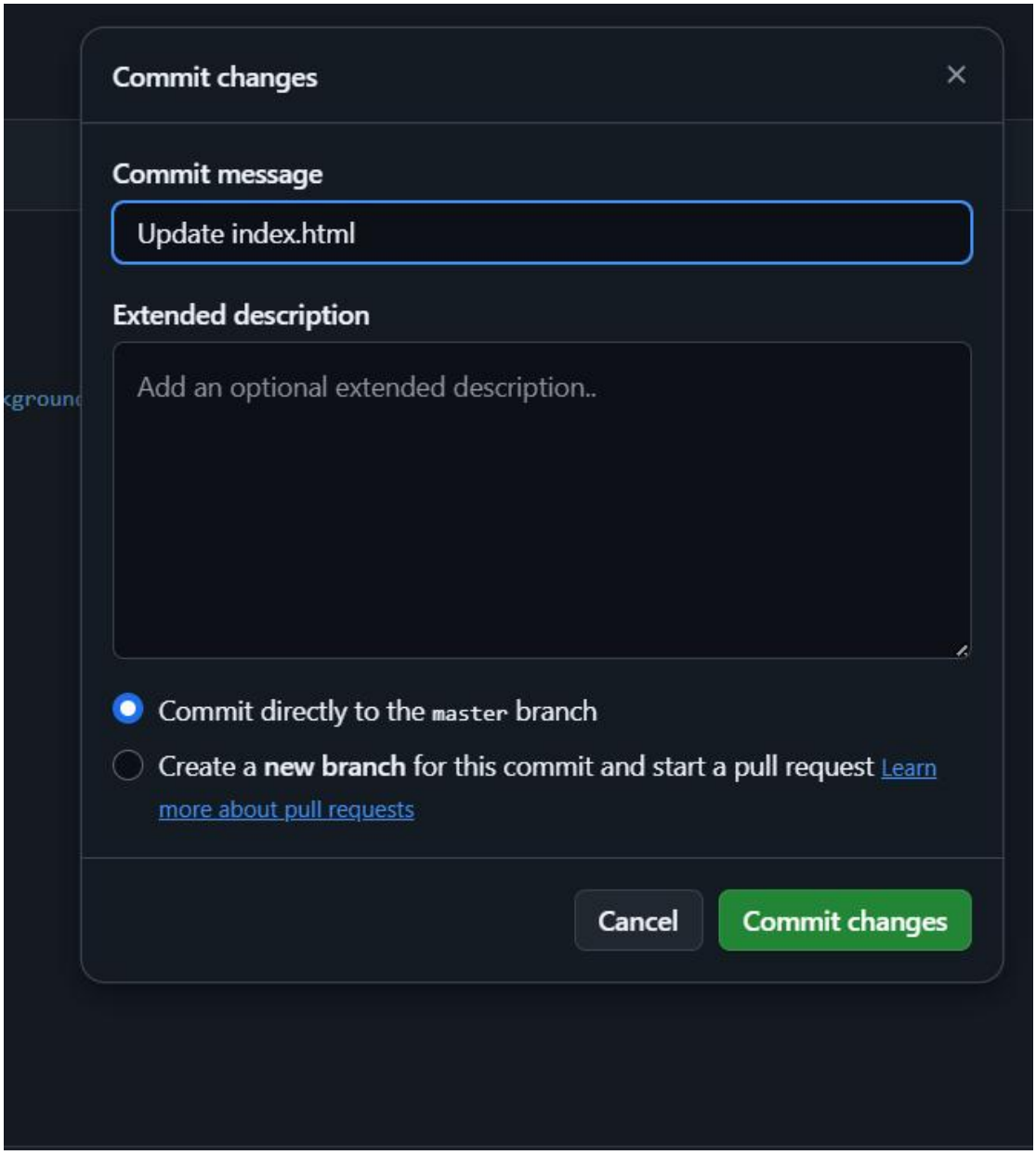
Now we will modify the jenkins scrips as well because the port will be occupied and previous pods needs to be deleted in order to create new one.. hence we will and below command

Sh ‘kubectl delete deploy nginx-deployment’

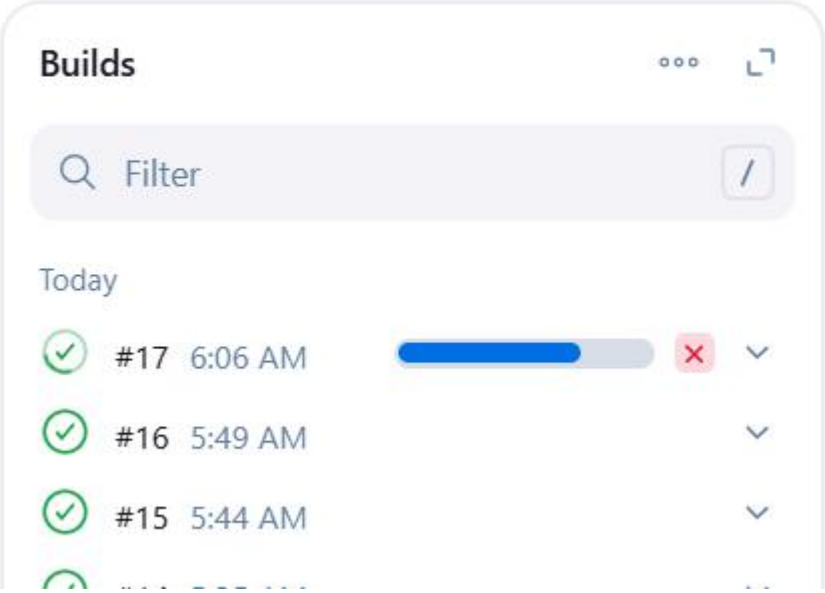
Now we will make chnages on git to run the pipeline automatctially

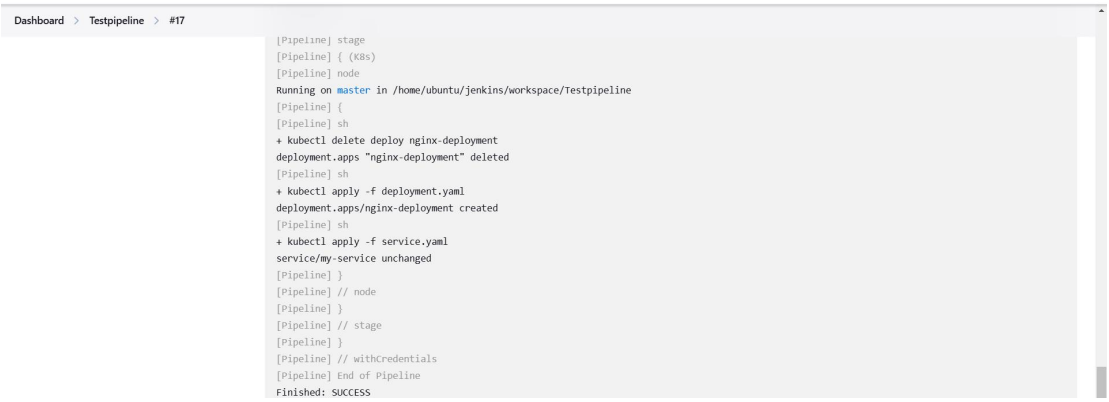


We have chnaged the heaader and message



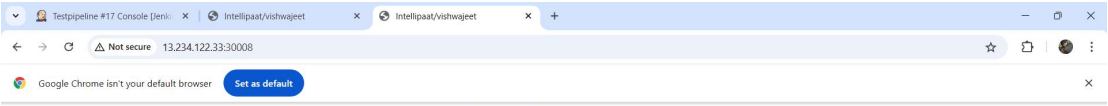
We can see below piplein has been running on its own





Now lets check this on browser

Hello world! from vishwajeet



Hello world! from vishwajeet