Capstone Project 2

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 to implement a DevOps lifecycle 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 the lifecycle:

1. Git workflow should be implemented. Since the company follows a monolithic architecture of development, you need to take care of version control. The release should happen only on the 25th of every month.

2. CodeBuild should be triggered once the commits are made in the master branch.

3. The code should be containerized with the help of the Dockerfile. The Dockerfile should be built every time if there is a push to GitHub. 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.

Architectural Advice:

Softwares to be installed on the respective machines using configuration management.

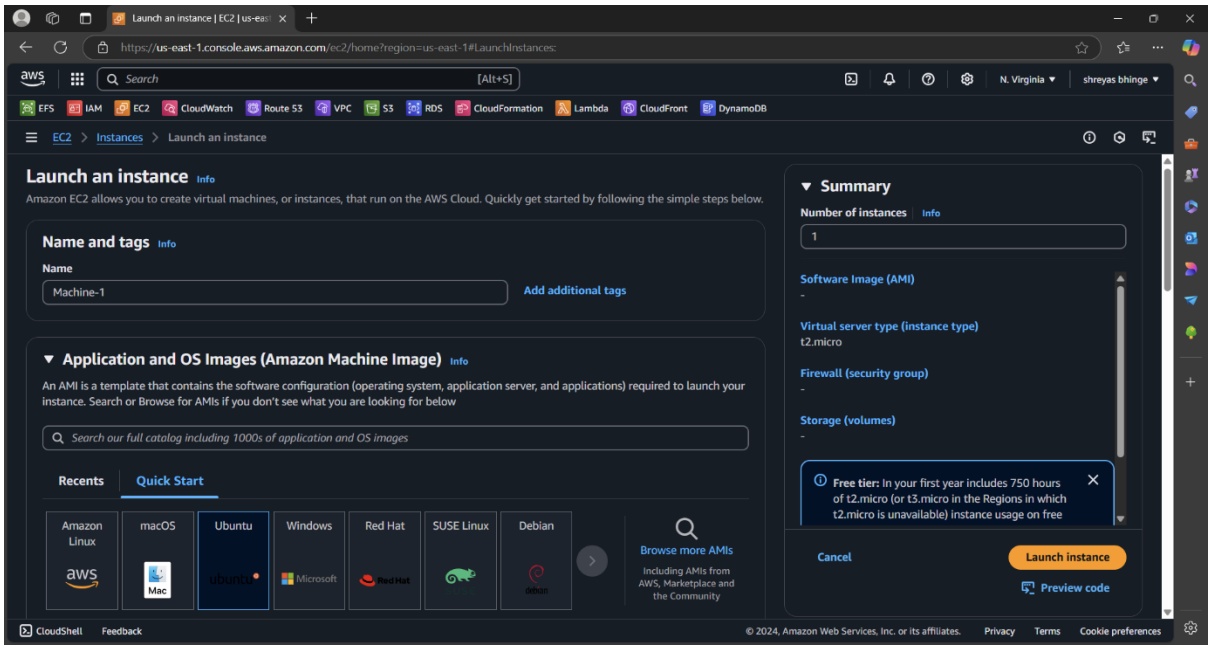
Worker1: Jenkins, Java

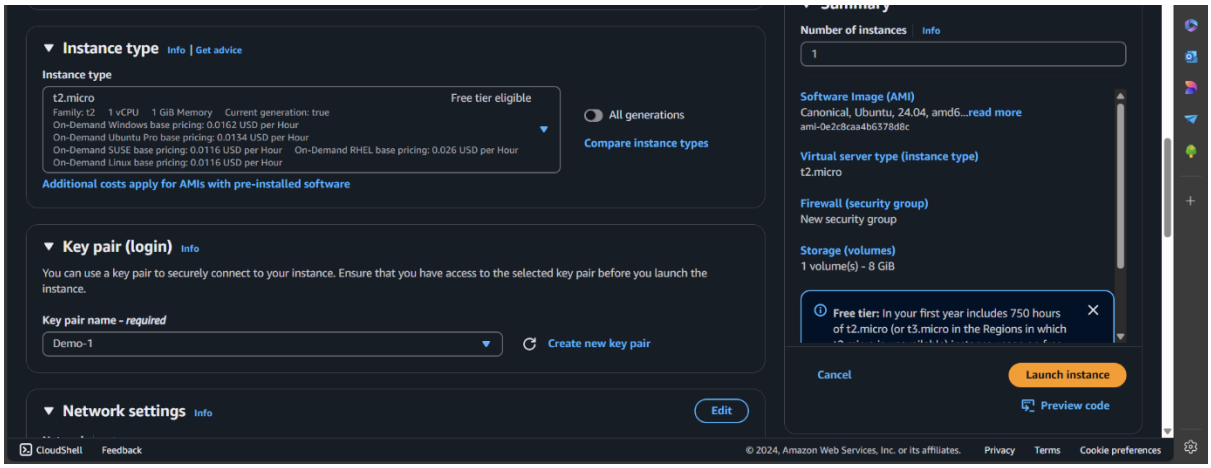
Worker2: Java, Docker, Kubernetes

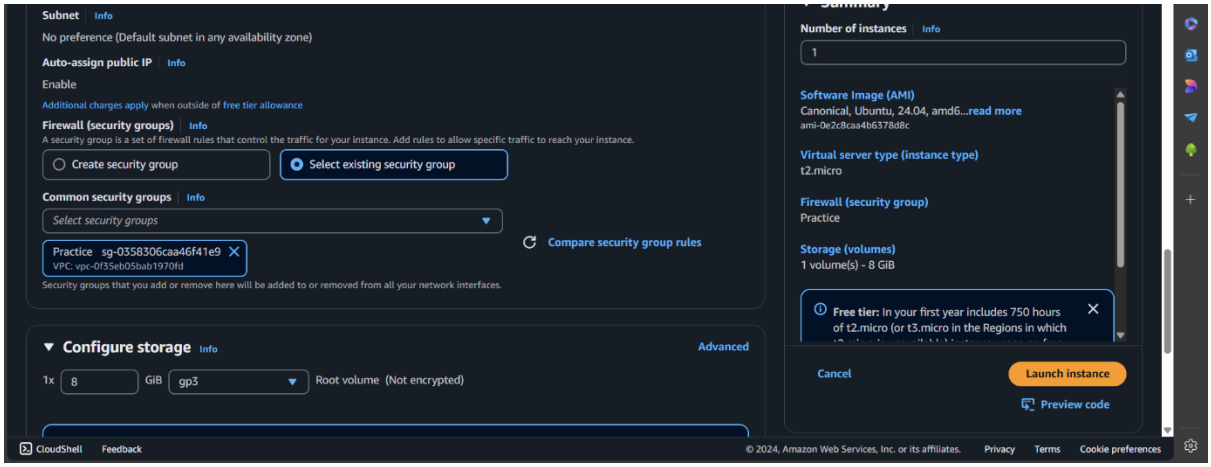
Worker3: Docker, Kubernetes

Worker4: Docker, Kubernetes

1. Create an Instance Manually on AWS







2. Install Terraform on Instance [Machine-1 (Main)]

#Enter this Command

wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

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 install terraform

terraform –version

Installing the CLI

```
ubuntu@Machine-1:~$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
```

```
ubuntu@Machine-1:~$ unzip awscliv2.zip
```

```
ubuntu@Machine-1:~$ sudo ./aws/install
```

```
ubuntu@Machine-1:~$ /usr/local/bin/aws --version
```

```
ubuntu@Machine-1:~$ aws configure
```

Enter Access Key and secrete key

3. Run “Terraform Script” to Create Other Three Instances

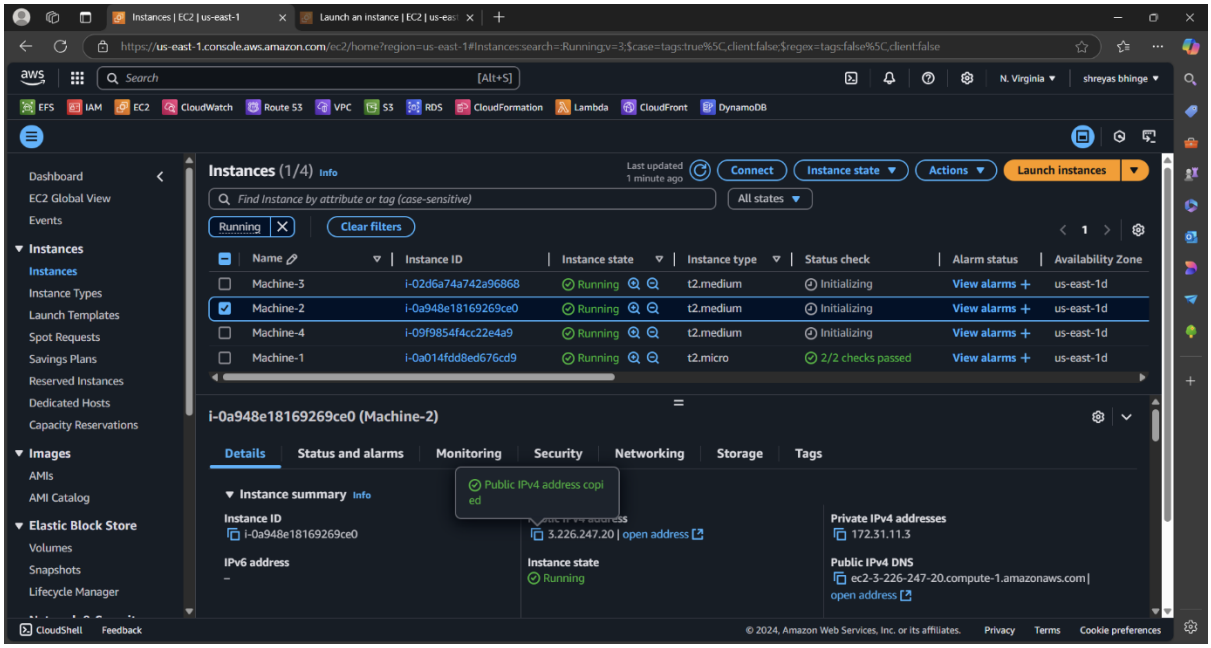
```
ubuntu@Machine-1:~$ vi main.tf
```

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "instance-1" {
  ami           = "ami-005fc0f236362e99f"
  key_name      = "Demo-1"
  instance_type = "t2.medium"
  tags = {
    Name = "Machine-2"
 }
}

resource "aws_instance" "instance-2" {
  ami           = "ami-005fc0f236362e99f"
  key_name      = "Demo-1"
  instance_type = "t2.medium"
  tags = {
    Name = "Machine-3"
 }
}

resource "aws_instance" "instance-3" {
  ami           = "ami-005fc0f236362e99f"
  key_name      = "Demo-1"
  instance_type = "t2.medium"
  tags = {
    Name = "Machine-4"
 }
}
~
~
~
~
"main.tf" 30L, 576B                                                    5,35          All
```



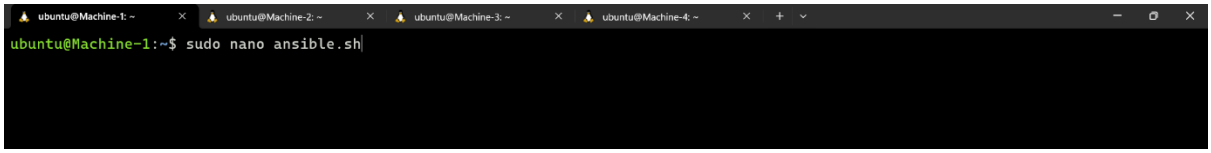
Commands

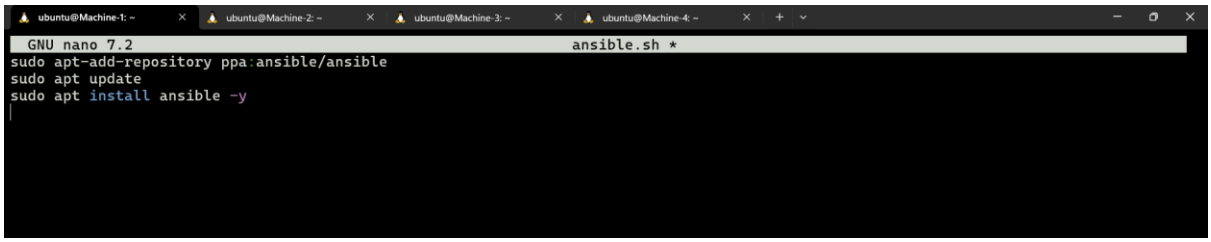
terraform init

terraform plan

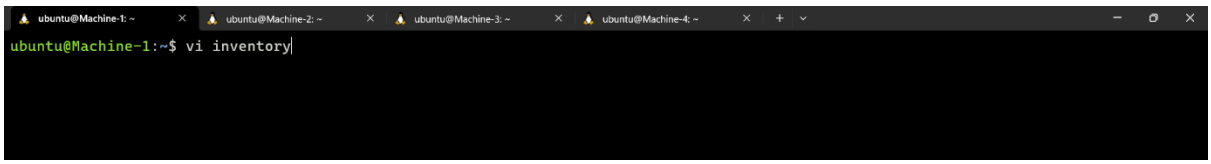
terraform apply

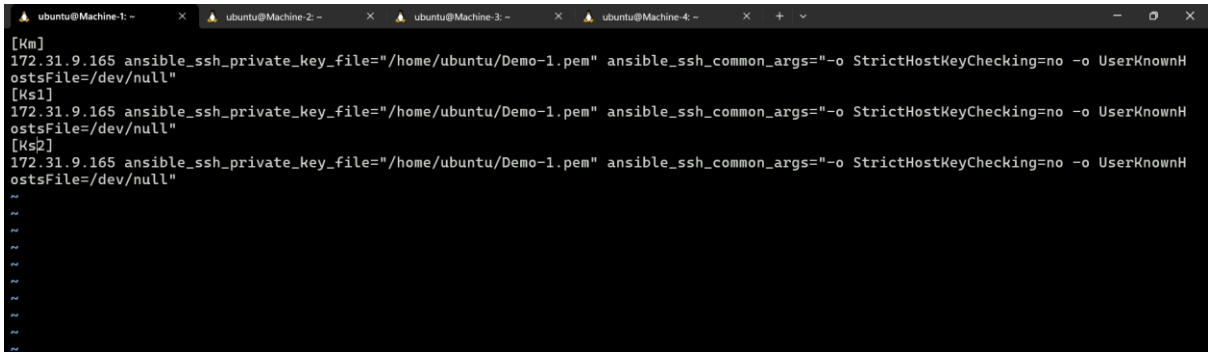
4. Install “Ansible” on Machine 1 (main)



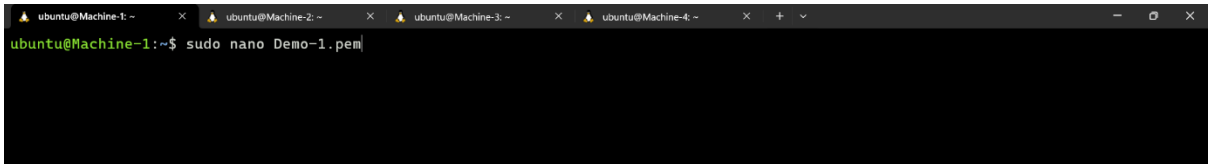


Paste the slaves & master Public IP Addresses here:

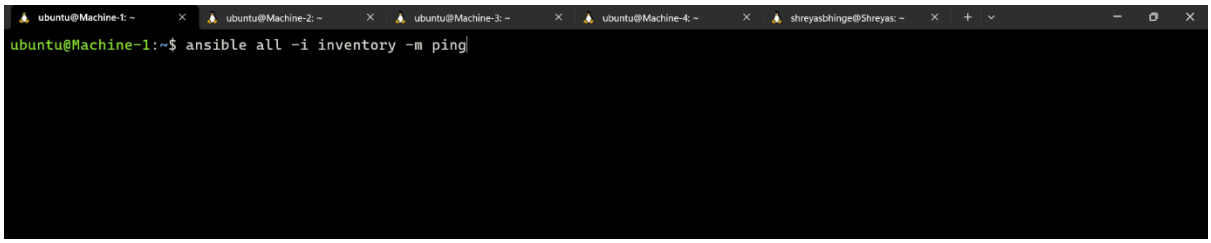


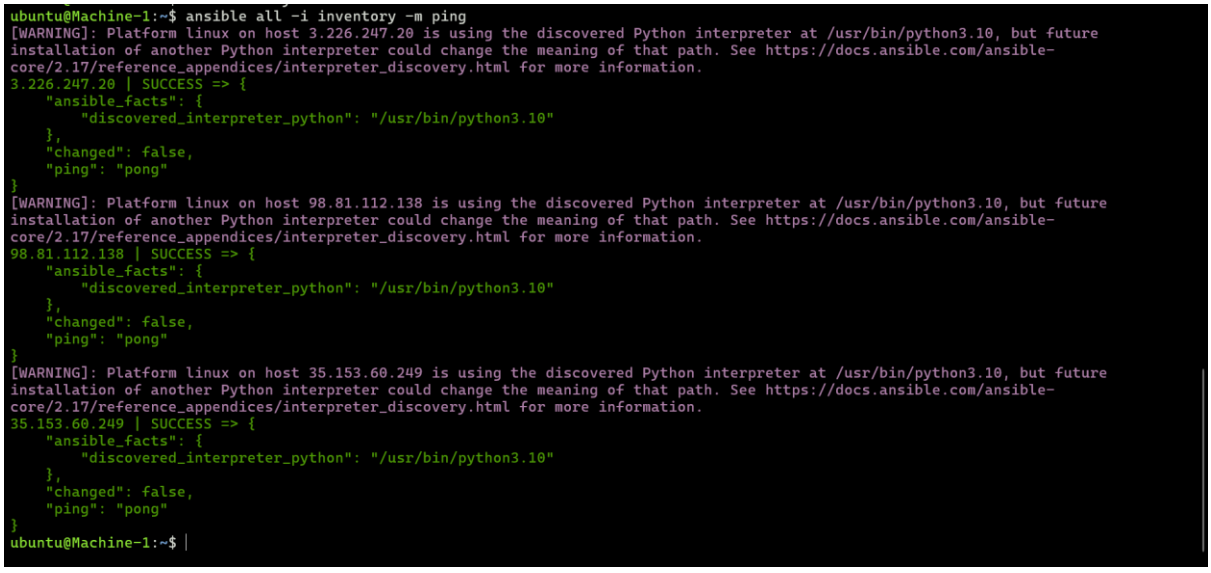


Add private key to home directory

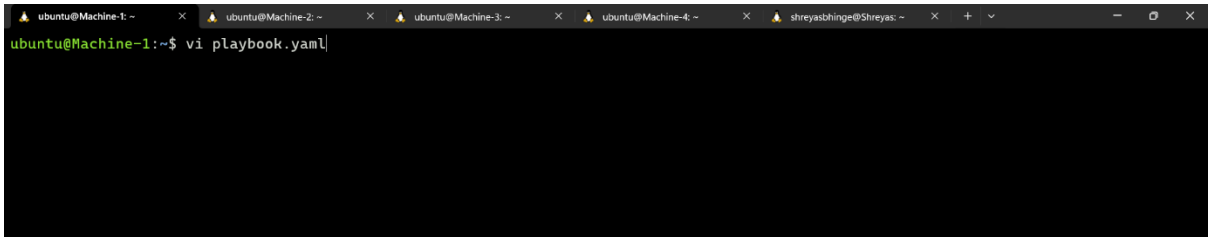


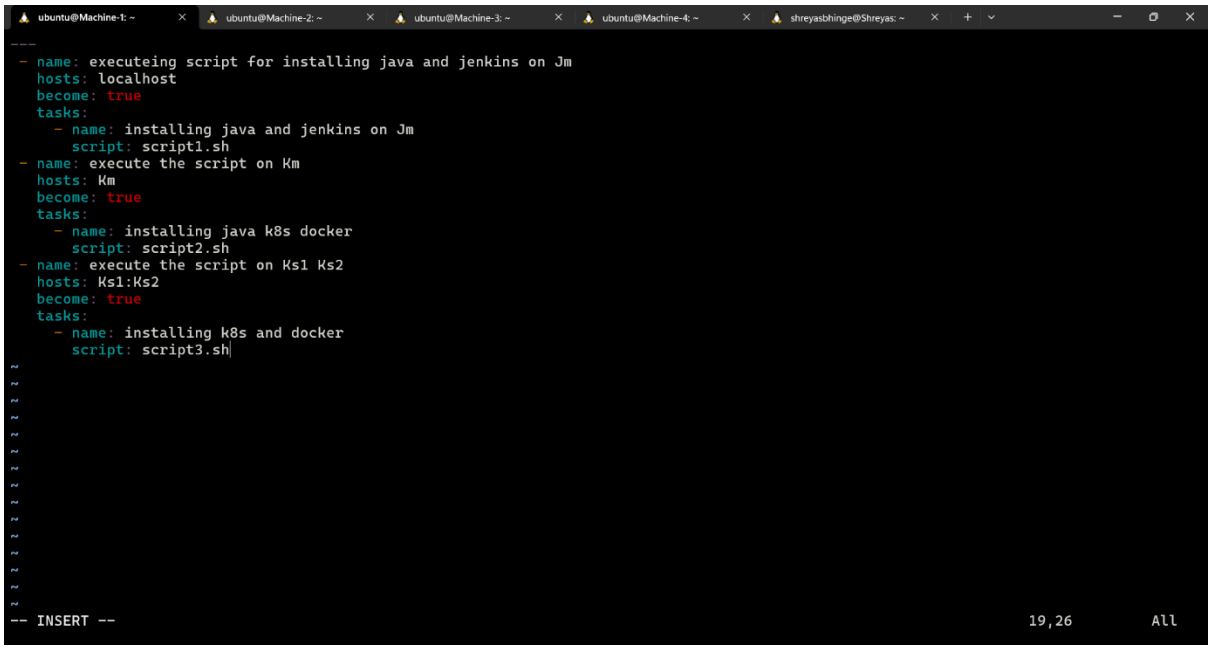
Now, we will **ping all machines using** the **below-given command:**





Create the Playbooks to Run these Scripts to Install the Much Needed Tools





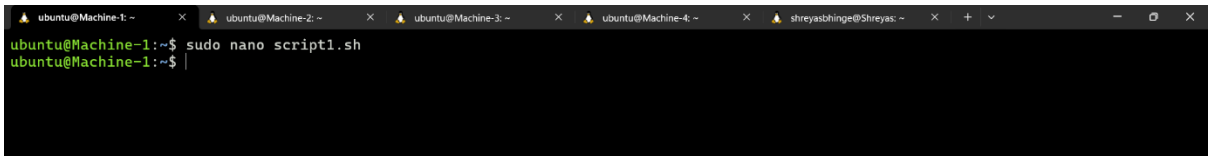
Create Three Scripts for Installing Required Tools on Machines

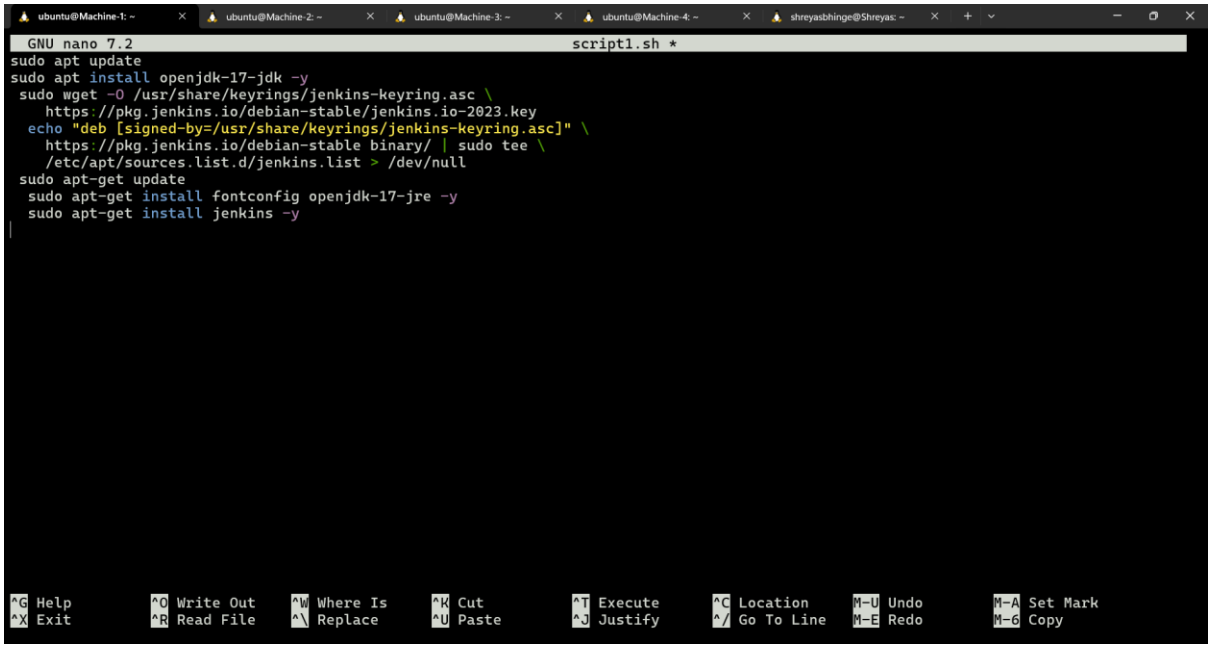
Worker1: Jenkins, Java

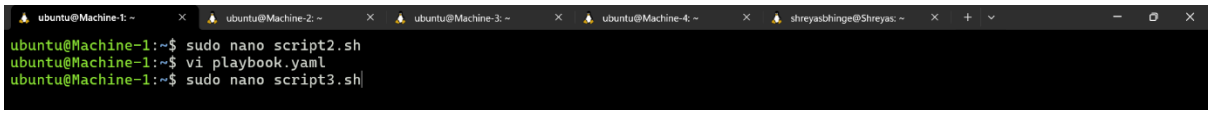
Worker2: Java, Docker, Kubernetes

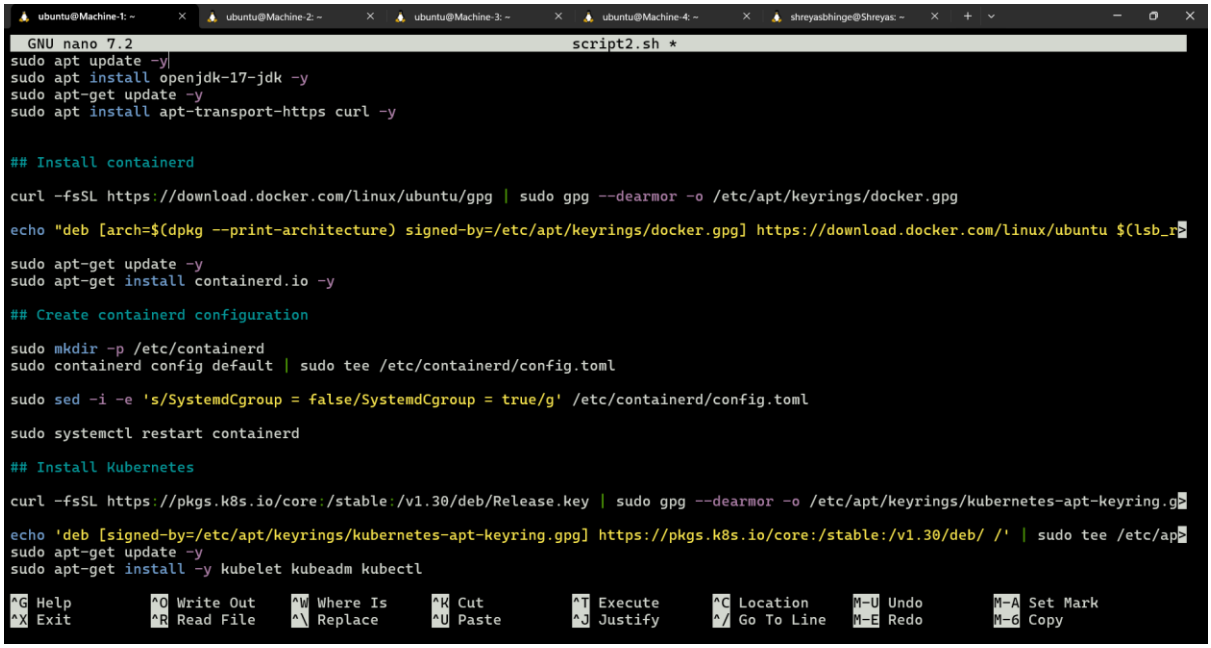
Worker3: Docker, Kubernetes

Worker4: Docker, Kubernetes




ubuntu@Machine-1: ~
ubuntu@Machine-2: ~
ubuntu@Machine-3: ~
ubuntu@Machine-4: ~
shreyasbhinge@Shreyas: ~
GNU nano 7.2 script1.sh *
sudo apt update
sudo apt install openjdk-17-jdk -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-get update
sudo apt-get install fontconfig openjdk-17-jre -y
sudo apt-get install jenkins -y
^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set Mark
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line M-E Redo M-6 Copy


ubuntu@Machine-1: ~
ubuntu@Machine-2: ~
ubuntu@Machine-3: ~
ubuntu@Machine-4: ~
shreyasbhinge@Shreyas: ~
ubuntu@Machine-1:~$ sudo nano script2.sh
ubuntu@Machine-1:~$ vi playbook.yaml
ubuntu@Machine-1:~$ sudo nano script3.sh


ubuntu@Machine-1: ~
ubuntu@Machine-2: ~
ubuntu@Machine-3: ~
ubuntu@Machine-4: ~
shreyasbhinge@Shreyas: ~
GNU nano 7.2 script2.sh *
sudo apt update -y
sudo apt install openjdk-17-jdk -y
sudo apt-get update -y
sudo apt install apt-transport-https curl -y
Install containerd
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(lsb_r
sudo apt-get update -y
sudo apt-get install containerd.io -y
Create containerd configuration
sudo mkdir -p /etc/containerd
sudo containerd config default | sudo tee /etc/containerd/config.toml
sudo sed -i -e 's/SystemdCgroup = false/SystemdCgroup = true/g' /etc/containerd/config.toml
sudo systemctl restart containerd
Install Kubernetes
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.g
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /' | sudo tee /etc/ap
sudo apt-get update -y
sudo apt-get install -y kubelet kubeadm kubectl
^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set Mark
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line M-E Redo M-6 Copy


ubuntu@Machine-1: ~
ubuntu@Machine-2: ~
ubuntu@Machine-3: ~
ubuntu@Machine-4: ~
shreyasbhinge@Shreyas: ~
ubuntu@Machine-1:~$ sudo nano script2.sh
ubuntu@Machine-1:~$ vi playbook.yaml
ubuntu@Machine-1:~$ sudo nano script3.sh

```
 GNU nano 7.2                                                script3.sh *
sudo apt update -y
sudo apt install apt-transport-https curl -y


## Install containerd

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu $(lsb_r>

sudo apt-get update -y
sudo apt-get install containerd.io -y

## Create containerd configuration

sudo mkdir -p /etc/containerd
sudo containerd config default | sudo tee /etc/containerd/config.toml

sudo sed -i -e 's/SystemdCgroup = false/SystemdCgroup = true/g' /etc/containerd/config.toml

sudo systemctl restart containerd

## Install Kubernetes

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.g>

echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /' | sudo tee /etc/ap>
sudo apt-get update -y
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
sudo systemctl enable --now kubelet

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

Change permissions of all scripts

```
ubuntu@Machine-1:~$ sudo chmod +r /home/ubuntu/script1.sh
ubuntu@Machine-1:~$ sudo chown ubuntu:ubuntu /home/ubuntu/script1.sh
ubuntu@Machine-1:~$
```

```
ubuntu@Machine-1:~$ sudo chmod +r /home/ubuntu/script2.sh
ubuntu@Machine-1:~$ sudo chown ubuntu:ubuntu /home/ubuntu/script2.sh
ubuntu@Machine-1:~$
```

```
ubuntu@Machine-1:~$ sudo chmod +r /home/ubuntu/script3.sh
ubuntu@Machine-1:~$ sudo chown ubuntu:ubuntu /home/ubuntu/script3.sh
ubuntu@Machine-1:~$
```

Check the **syntax using** the **below-given command:**

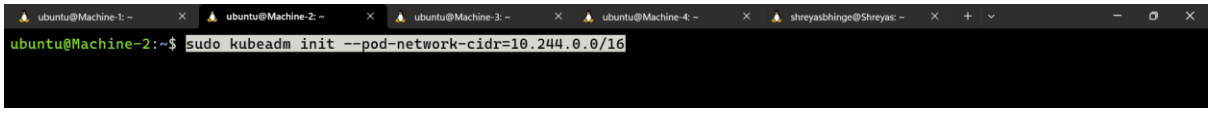
```
ubuntu@Machine-1:~$ ansible-playbook -i inventory playbook.yaml --syntax-check

playbook: playbook.yaml
ubuntu@Machine-1:~$
```

Now, we will run the **“play.yaml” using** the **below-given command**

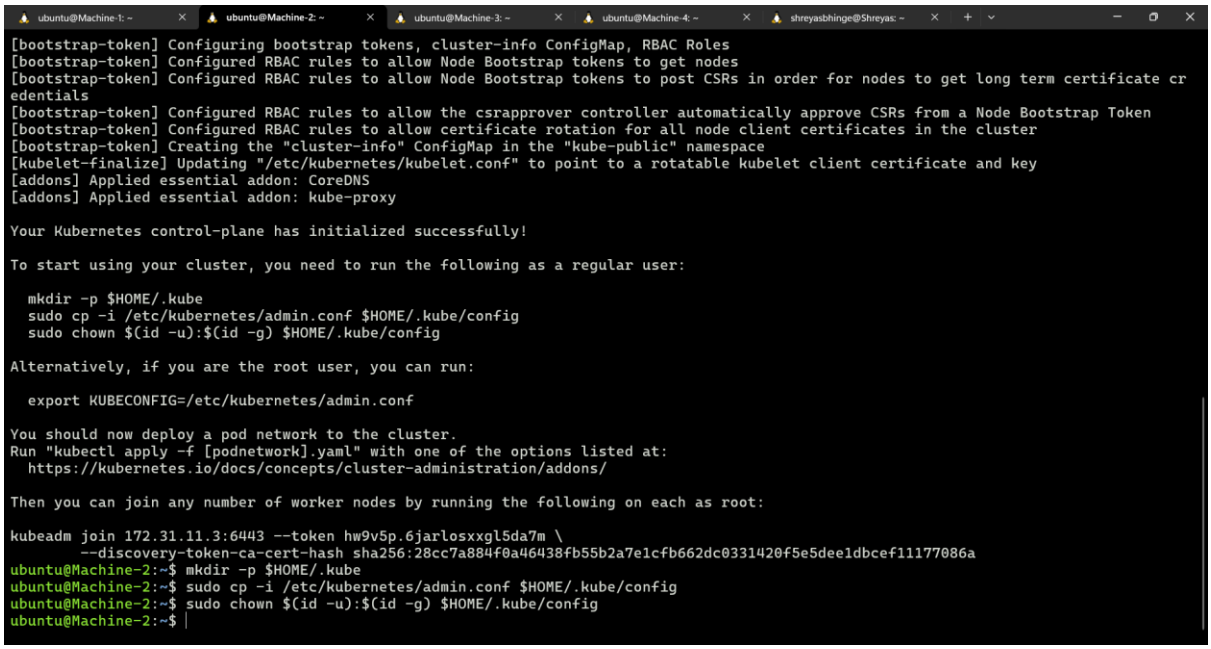
```
ubuntu@Machine-1:~$ ansible-playbook -i inventory playbook.yaml
```

Configure Kubernetes on Master (Machine-2)



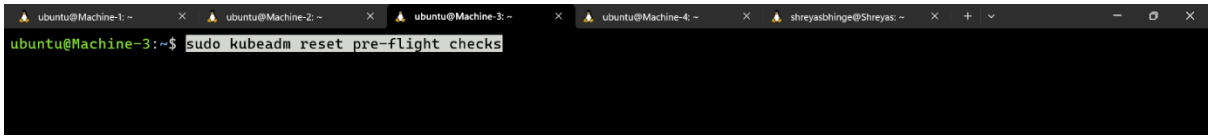
Paste the below-given commands in the “Master” node:

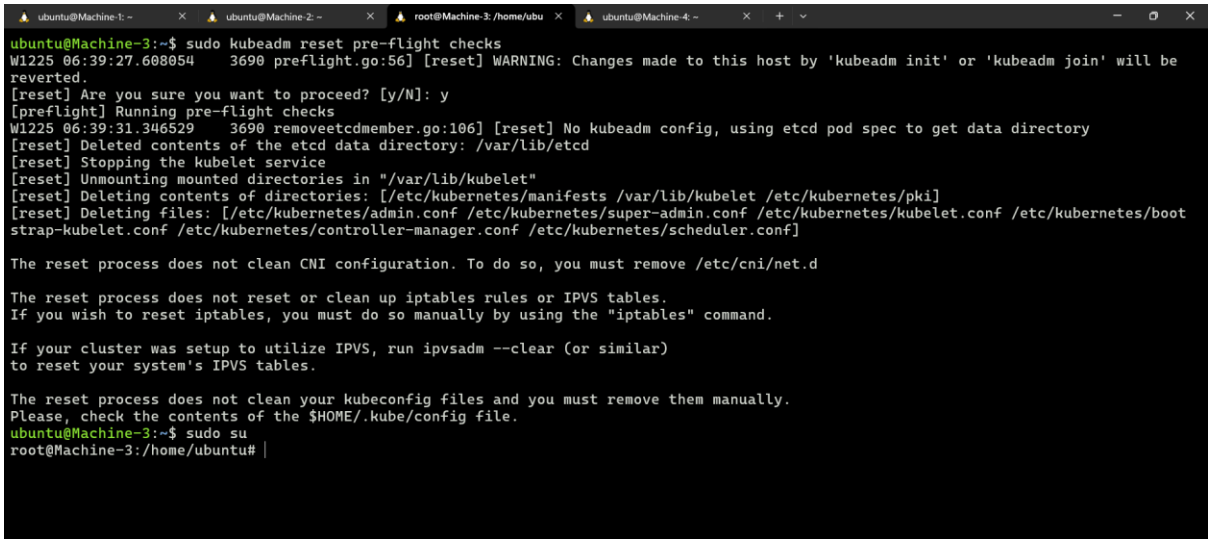
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config



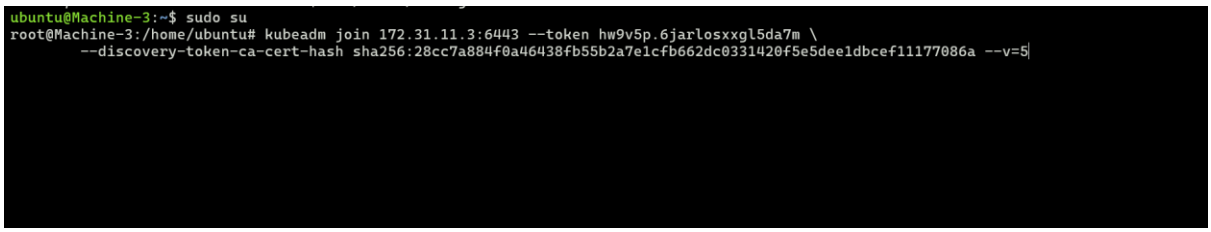
In K8s Slaves (Machine-3 and Machine-4}

Machine-3





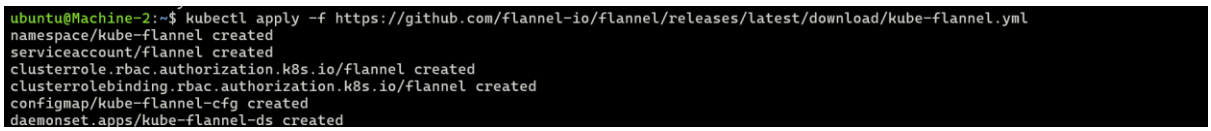
Copy Token from Machine-2 and paste above –v=5 in machine3



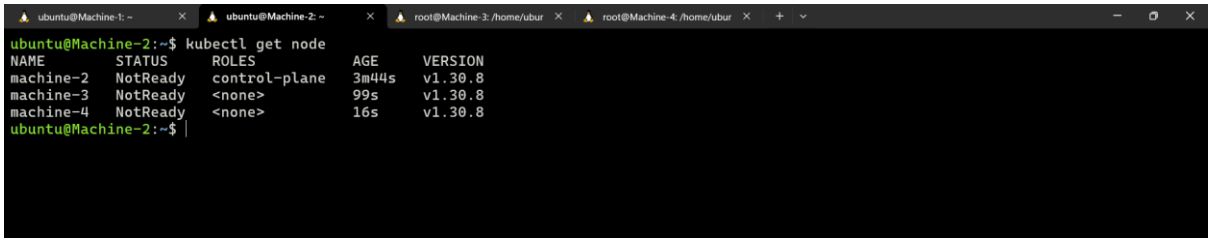
Same steps to follow with Machine-4

nstall the “Calico Network” to run the cluster using the below-given command in Machine-2 (Km):

kubectl apply -f https://raw.githubusercontent.com/flannel-io/flannel/master/Documentation/kube-flannel.yml



All the deployments have been successfully created. Now, run the below-given command to get all the nodes:



(In Machine-2) i.e Master

The error indicates that containerd.io conflicts with another package (containerd) on your system, or that there are held packages causing dependency resolution issues. Here's how to resolve it step by step:

1. Check for Held Packages

Held packages can prevent updates or installations. Check for held packages using:

bash

Copy code

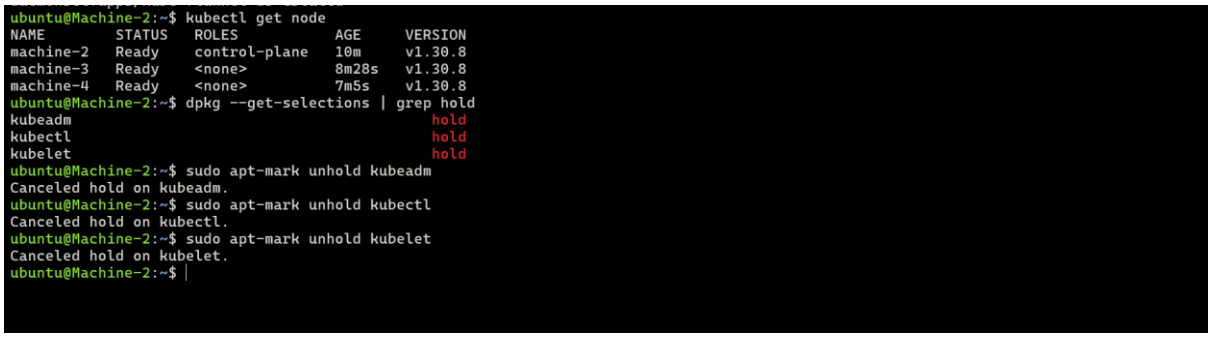
dpkg --get-selections | grep hold

If any packages are held, unhold them:

bash

Copy code

sudo apt-mark unhold <package-name>



2. Remove Conflicting Packages

Since containerd.io conflicts with containerd, you may need to remove containerd:

bash

Copy code

sudo apt-get remove --purge containerd

3. Clean Up and Update

Clean up and refresh the package manager

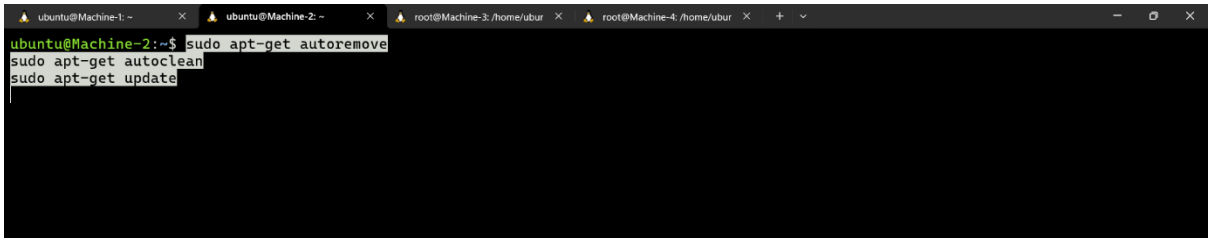
bash

Copy code

sudo apt-get autoremove

sudo apt-get autoclean

sudo apt-get update



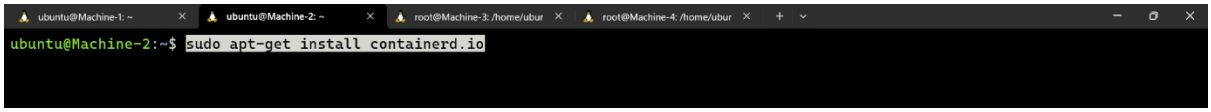
4. Install containerd.io

Try installing containerd.io again:

bash

Copy code

sudo apt-get install containerd.io



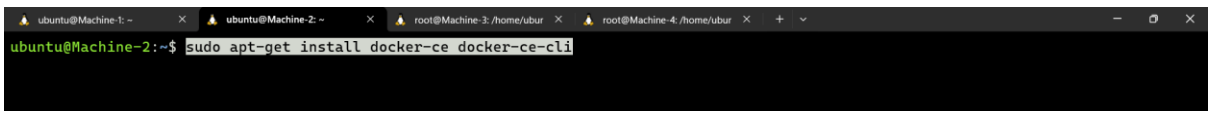
Check Docker and Containerd Versions

If you're using Docker, ensure the versions of Docker and containerd.io are compatible. Update Docker if necessary:

bash

Copy code

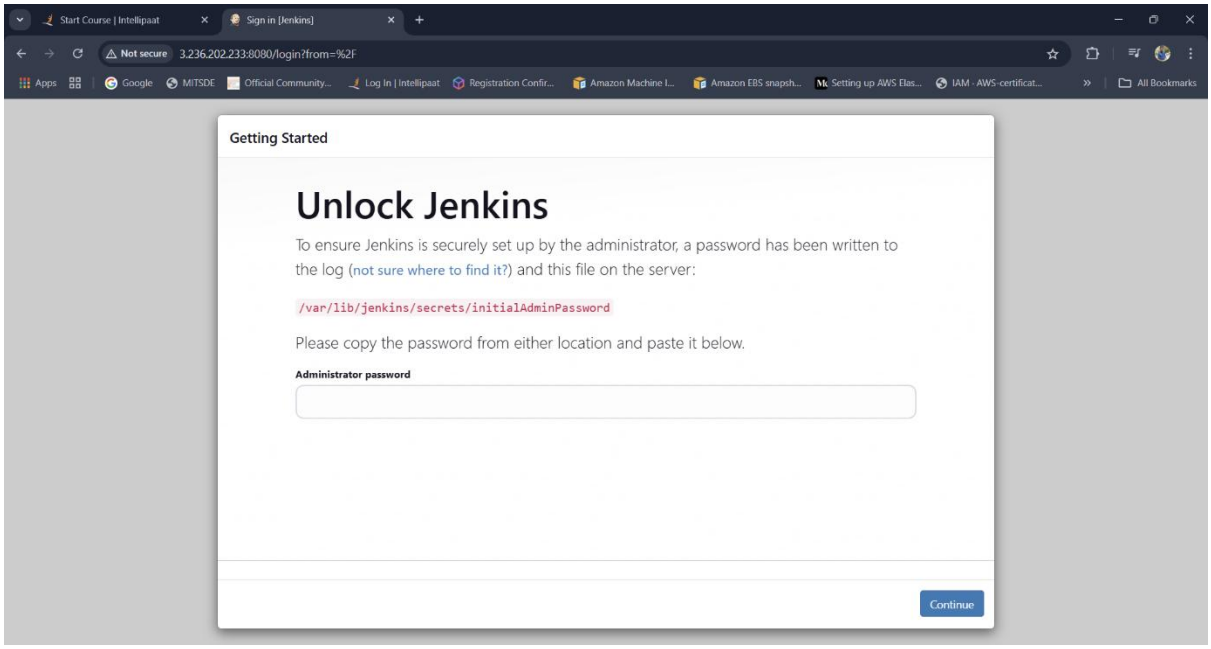
sudo apt-get install docker-ce docker-ce-cli



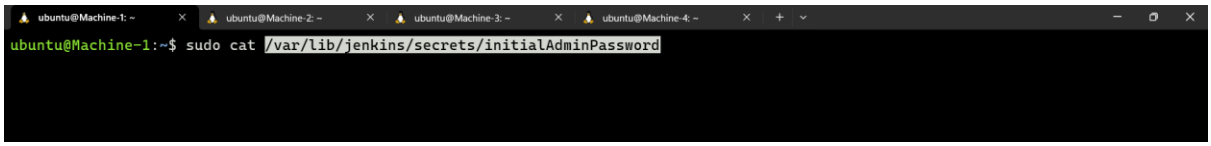
Copy same steps to Machine-3 and Machine-4

Configure Jenkins Setup Properly Here on Machine-1

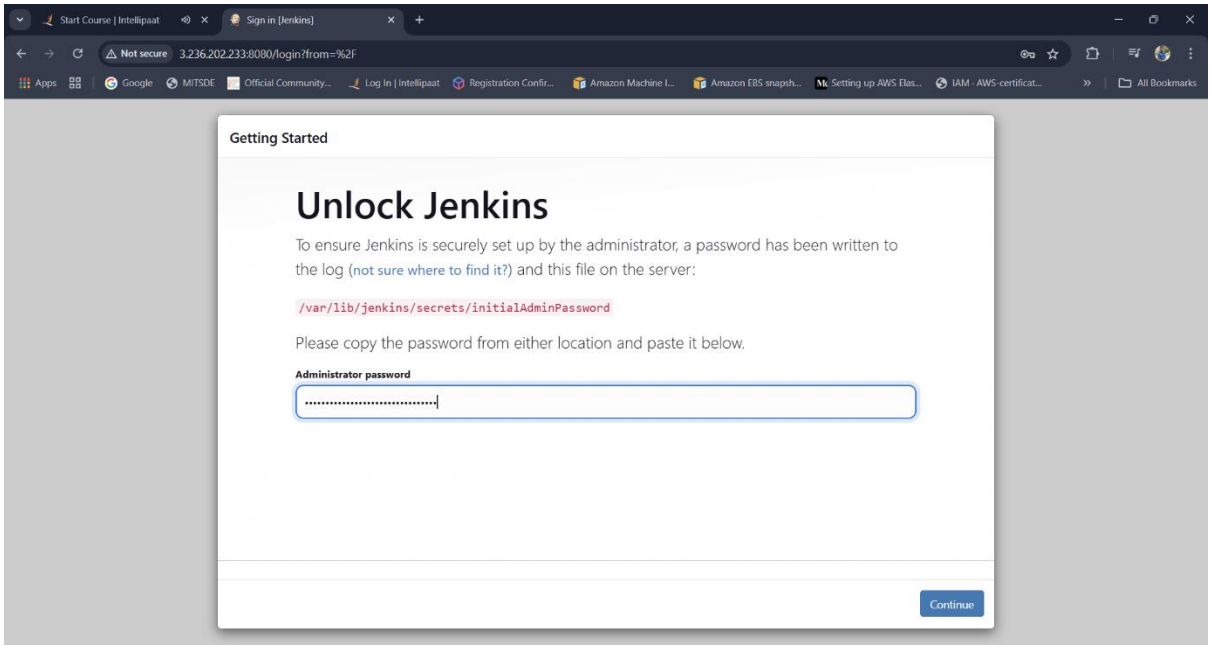
Copy Public Ip of Machine-1



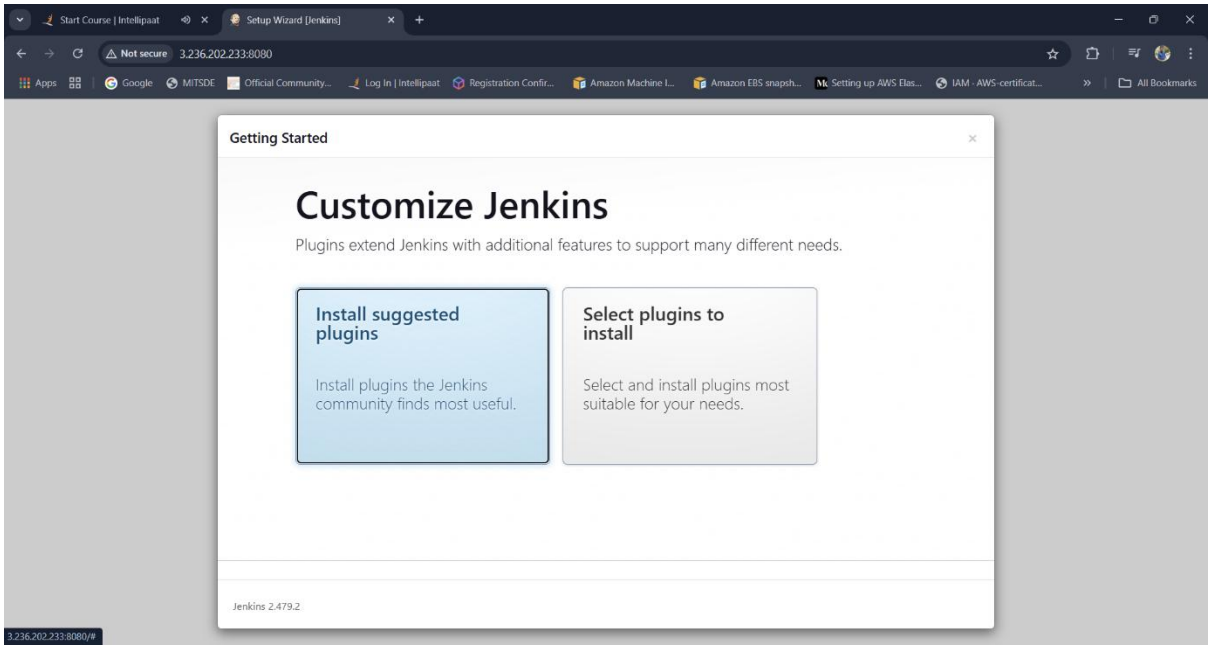
Go to the “Machine-1 (Main)” & paste the below-given command with “sudo cat”



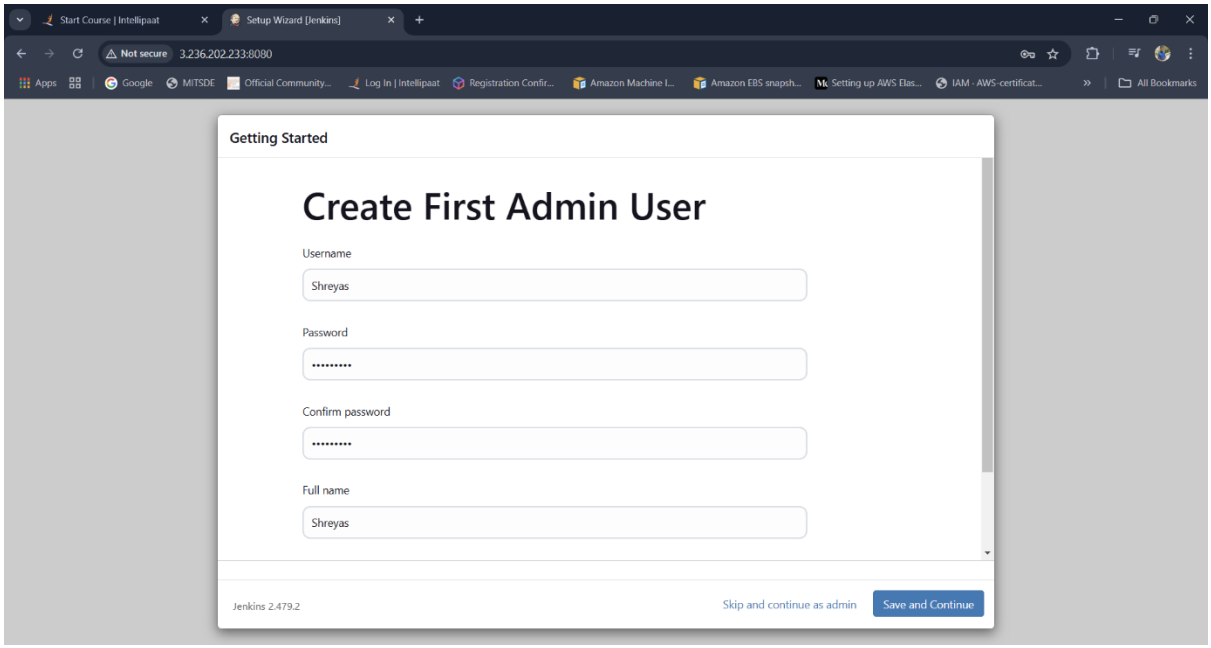
Paste the token in the “Administrator Password” section & click on “Continue”.



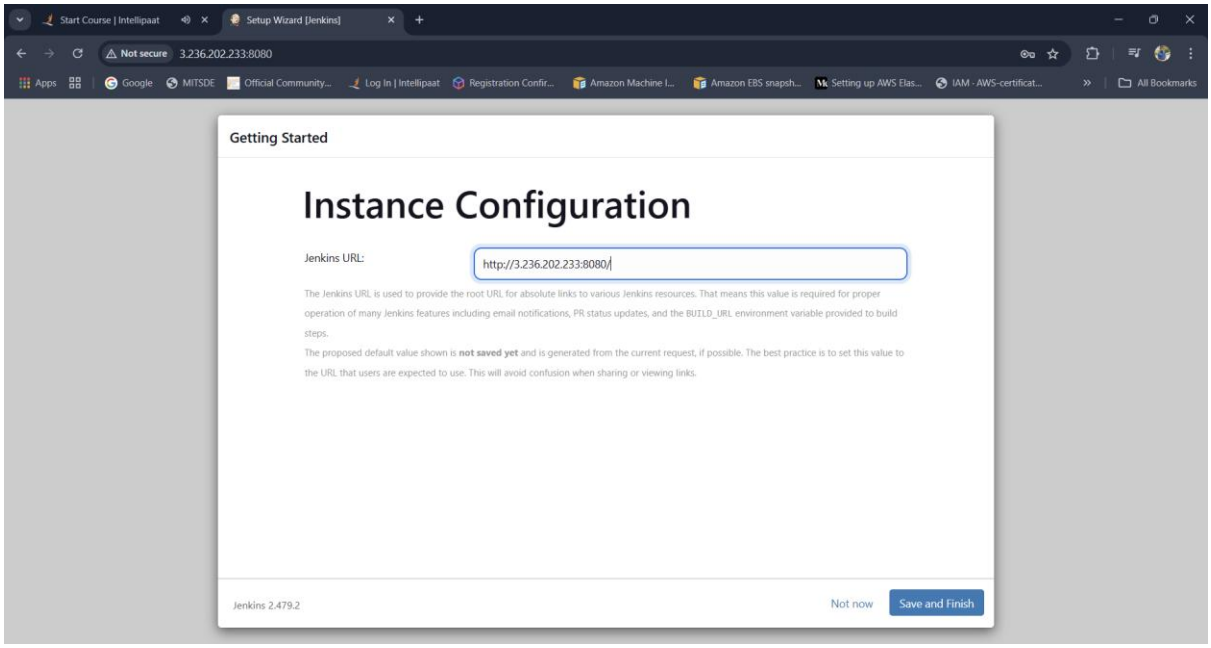
Click on “Install Suggested Plugins”.



Create the user by filling the below-given details:



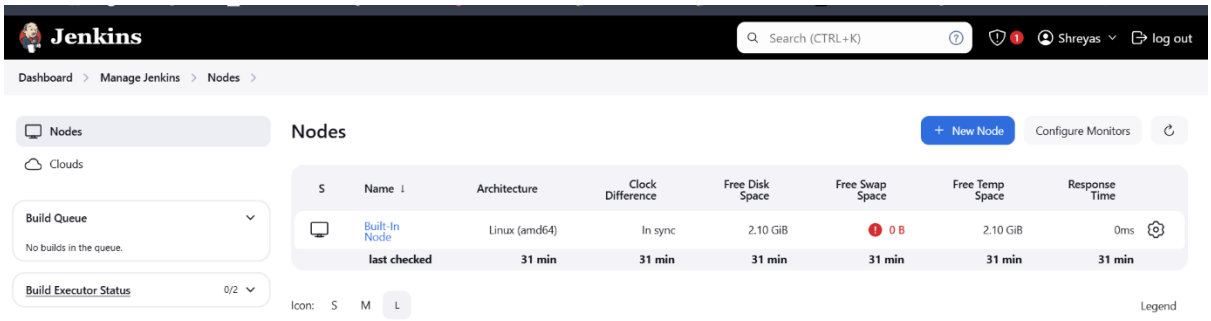
Click on “Save and Finish”



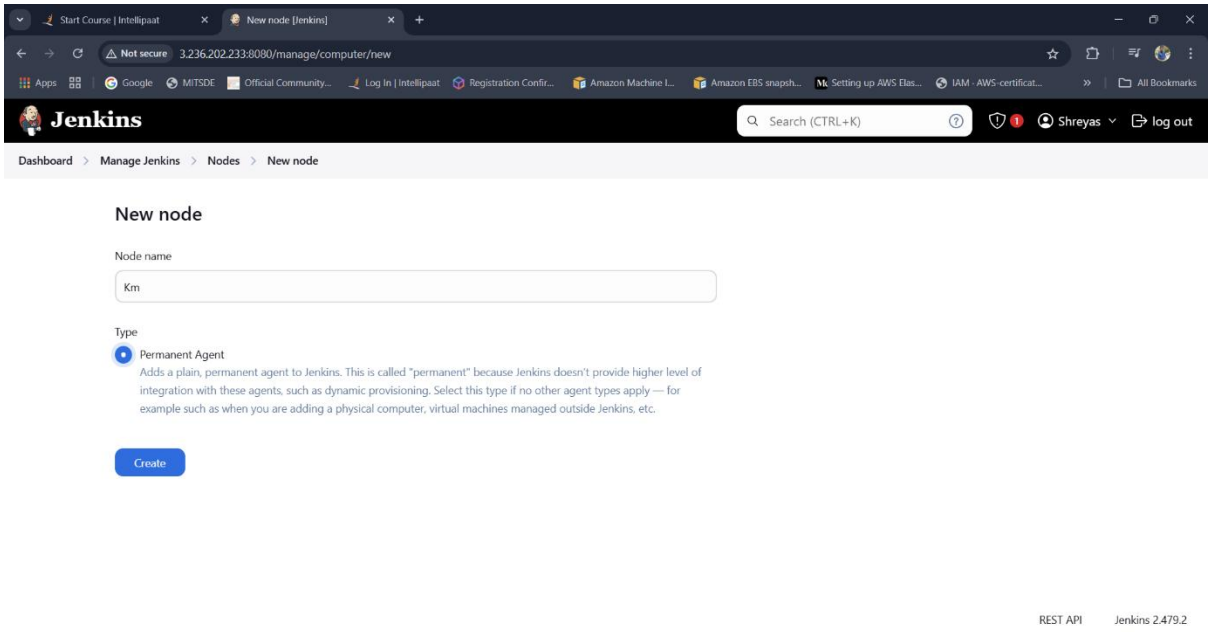
Add “Kubernetes Master (Machine-2)” as a Node Here

Click on “Set up an agent”.

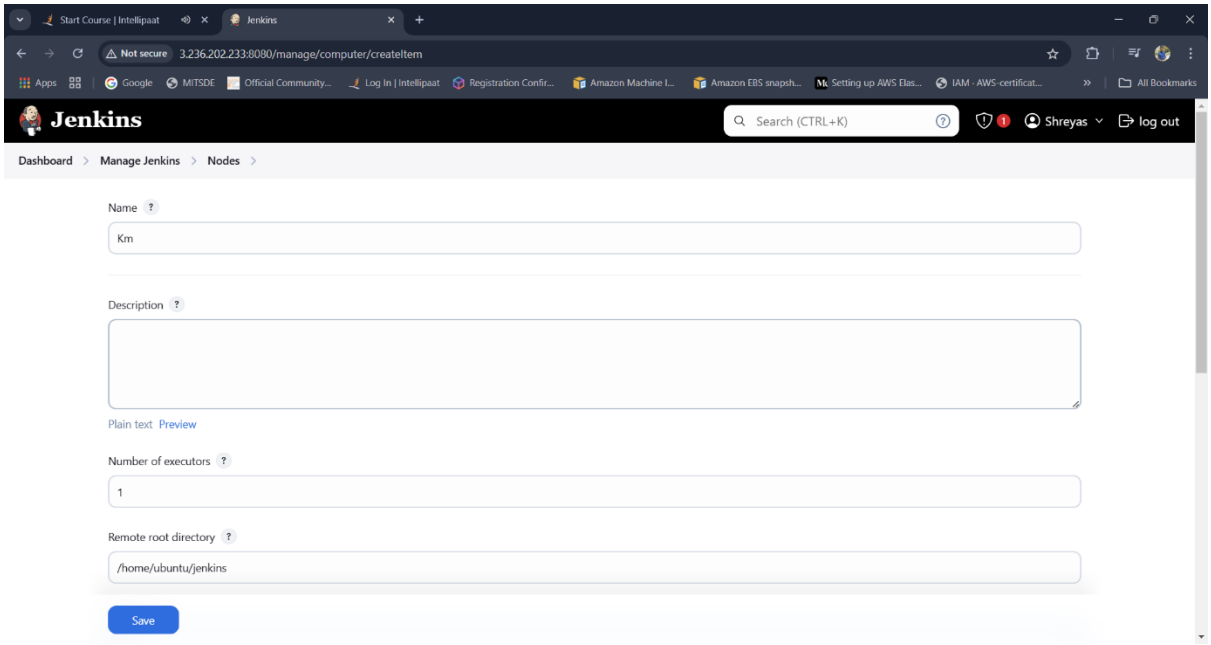
Add new Node

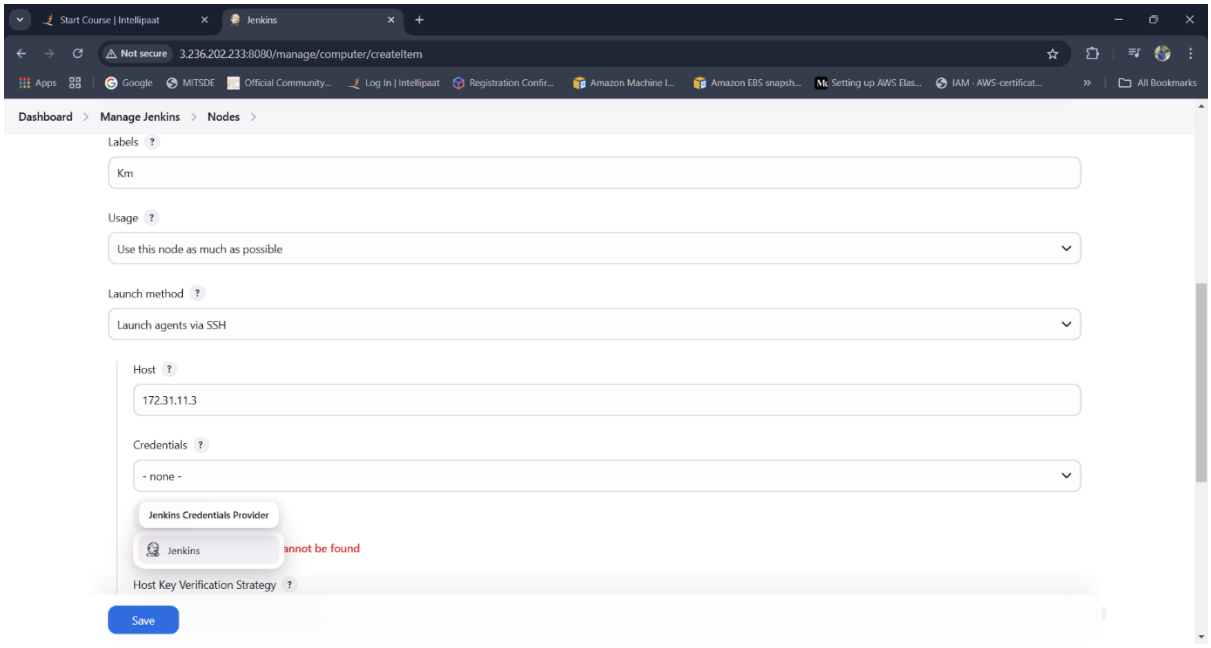


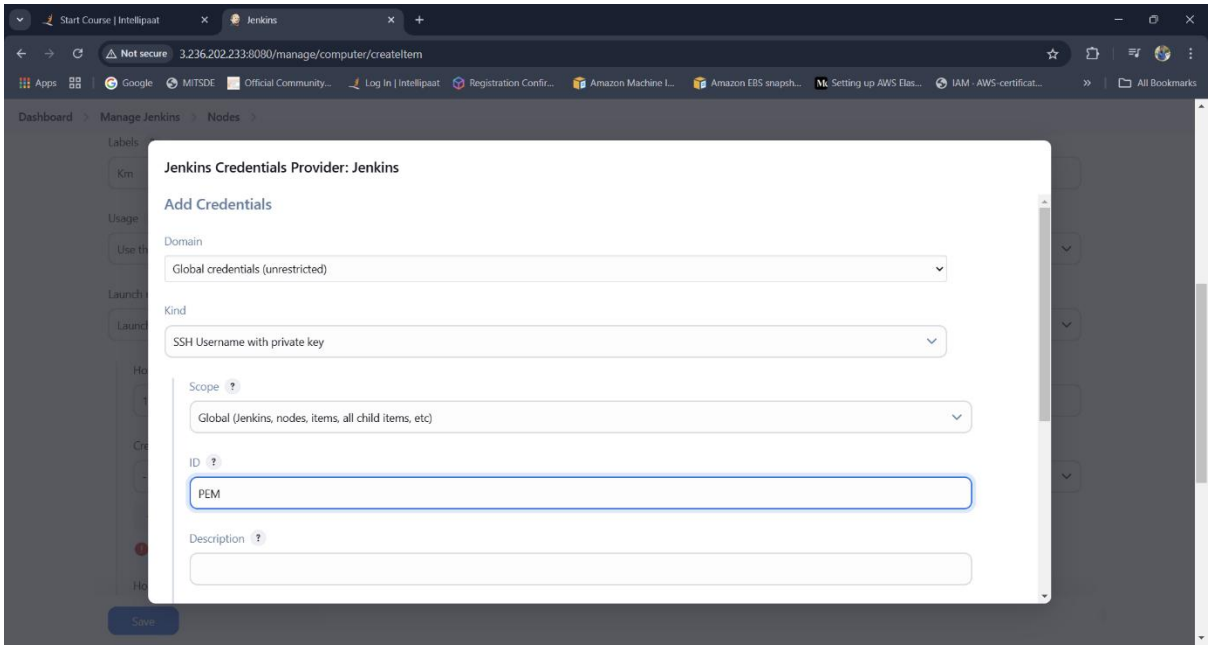
hoose “Node name” as “Km” & “Type” as “Permanent Agent”.

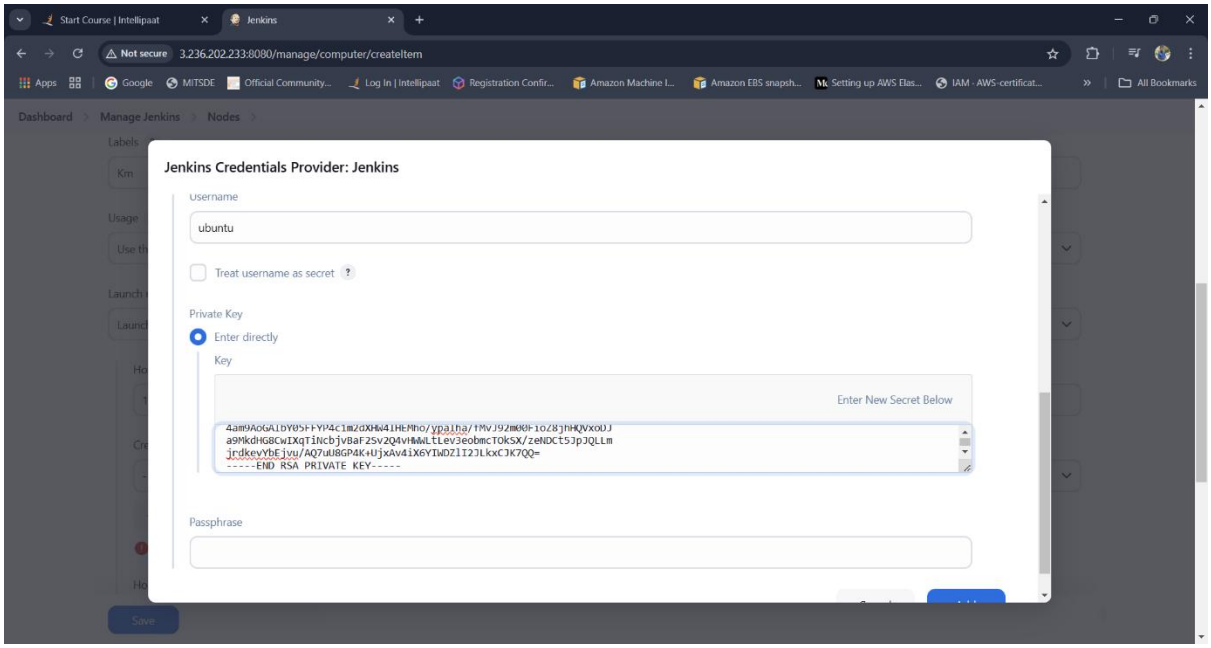


Choose the following options here



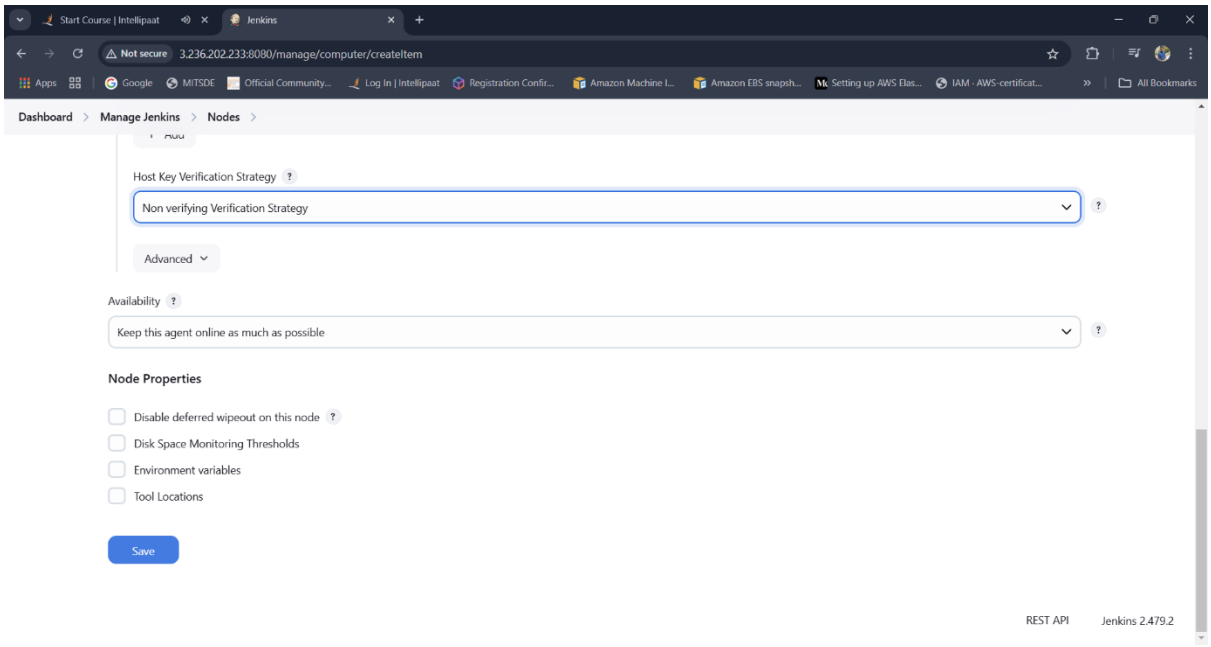

Start Course | Intellipaat
Jenkins
Not secure
3.236.202.233:8080/manage/computer/createItem
Dashboard
Manage Jenkins
Nodes
Labels
Km
Usage
Use this node as much as possible
Launch method
Launch agents via SSH
Host
172.31.11.3
Credentials
- none -
Jenkins Credentials Provider
Jenkins
annot be found
Host Key Verification Strategy
Save


Start Course | Intellipaat
Jenkins
Not secure
3.236.202.233:8080/manage/computer/createItem
Dashboard
Manage Jenkins
Nodes
Jenkins Credentials Provider: Jenkins
Add Credentials
Domain
Global credentials (unrestricted)
Kind
SSH Username with private key
Scope
Global (Jenkins, nodes, items, all child items, etc)
ID
PEM
Description

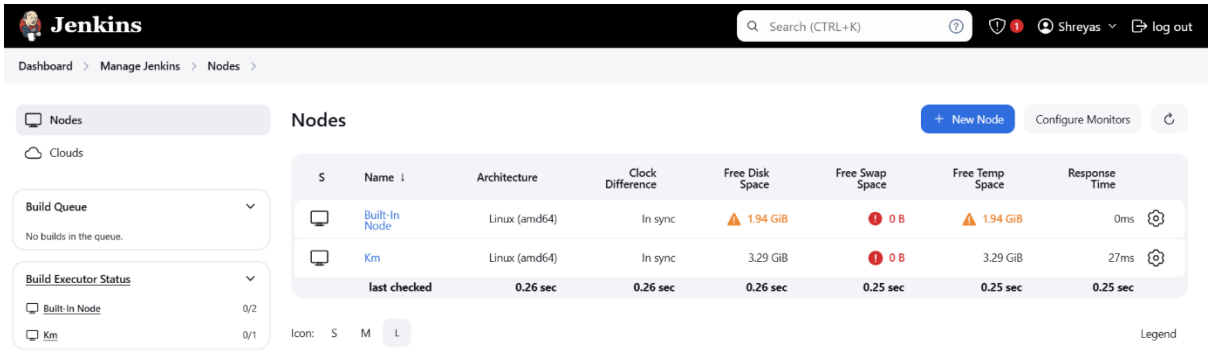


Choose “Host Key Verification Strategy” as “Non verifying Verification Strategy”.

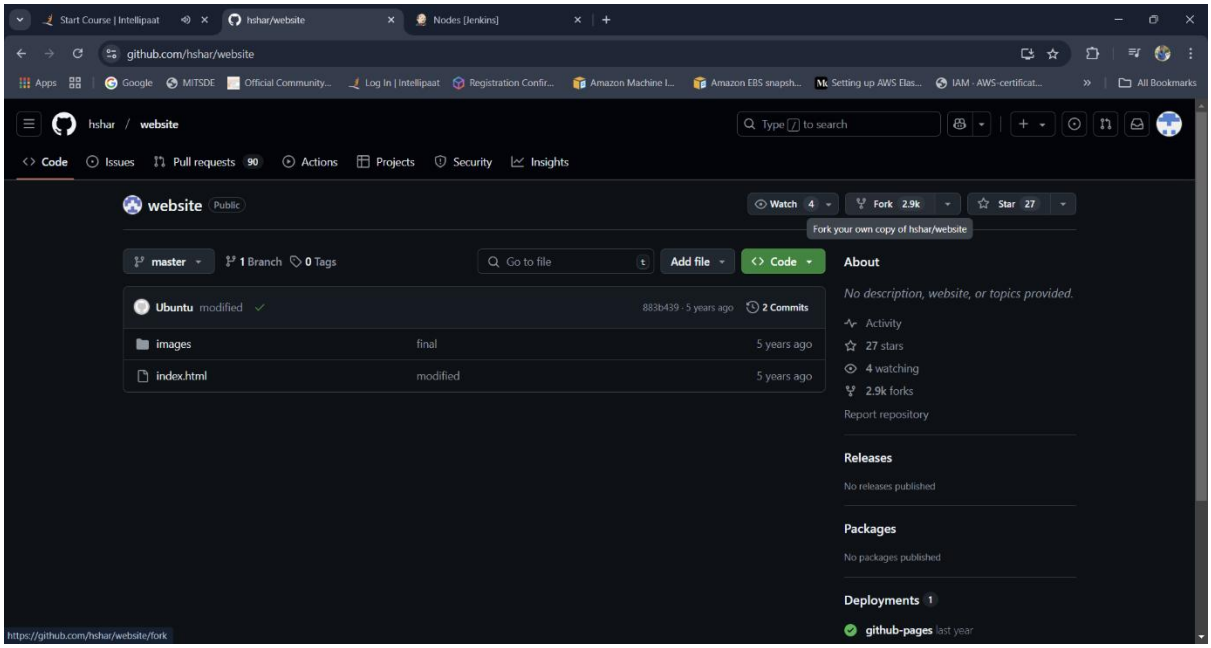
Click on “Save”.

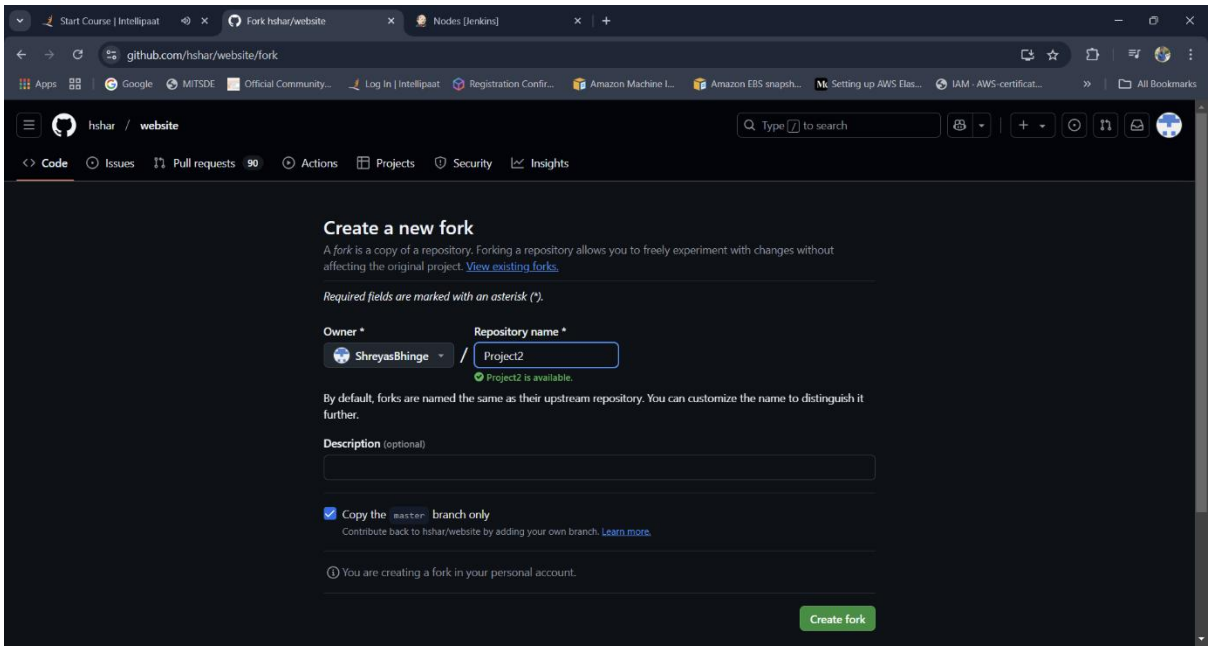


“Km” has been successfully added as a “Node”.

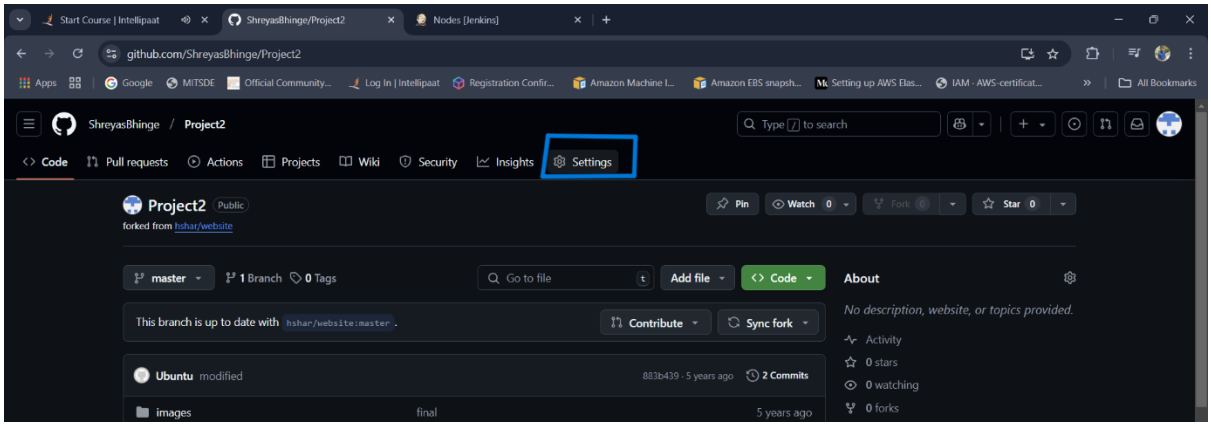


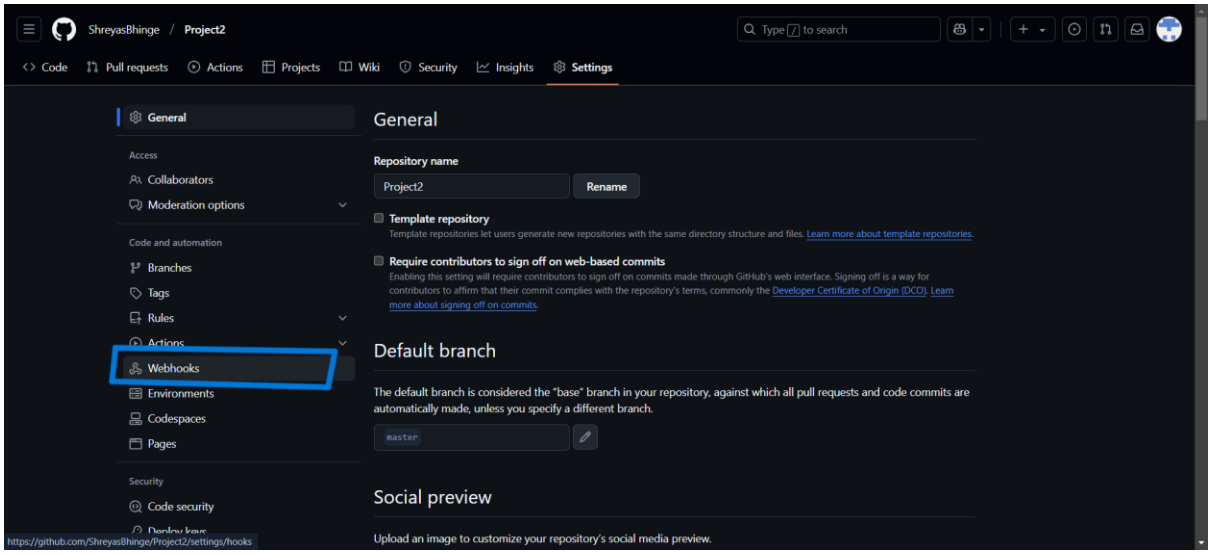
Fork the Repository in the GitHub Account

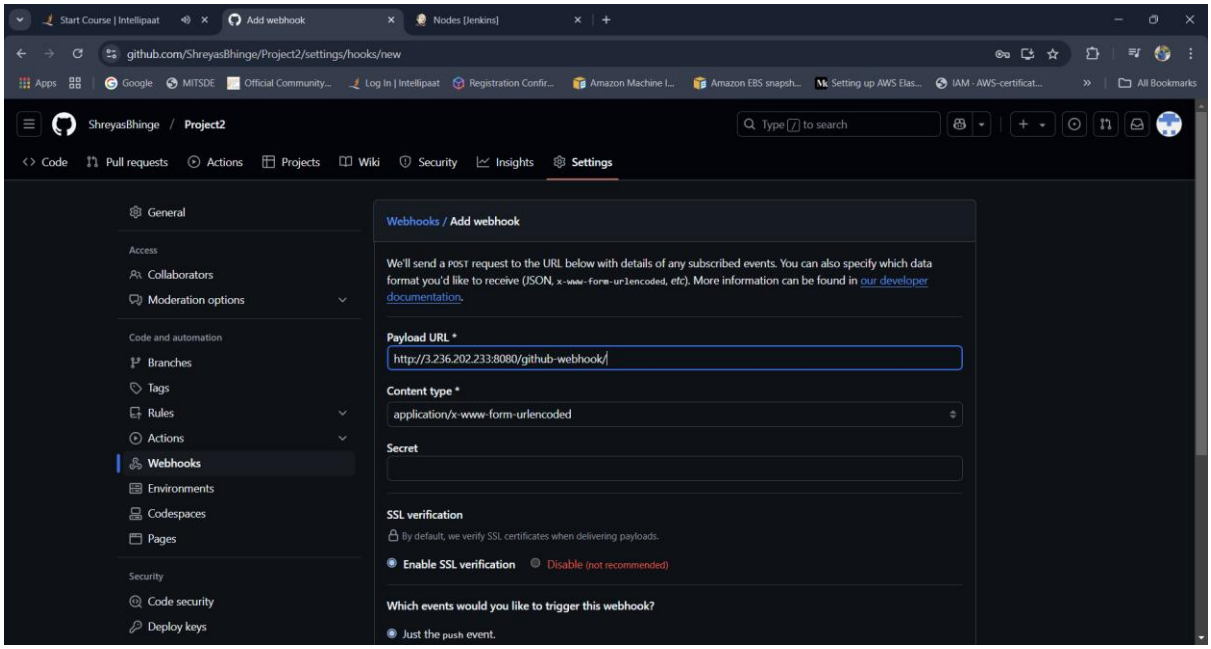


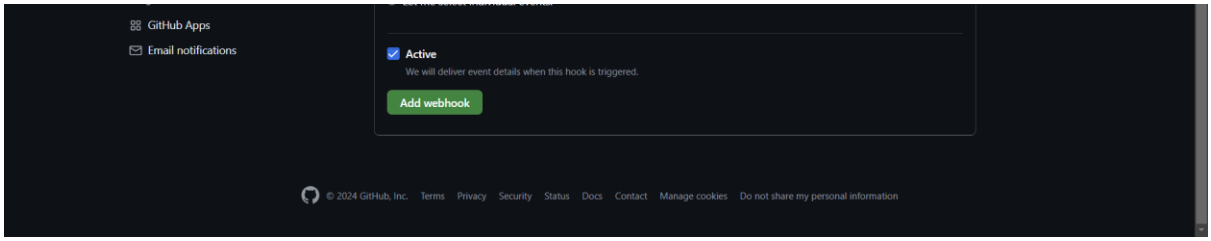


Create webhook

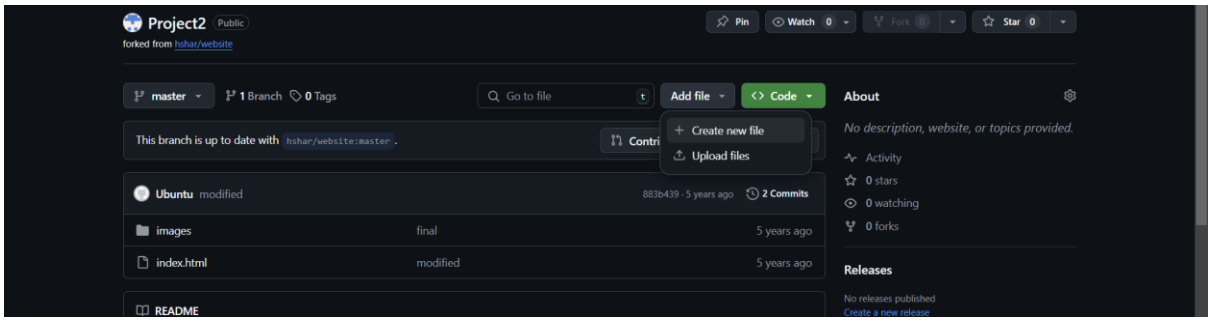

ShreyasBhinge / Project2
Code
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Project2
Public
forked from hshar/website
Pin
Watch 0
Fork 0
Star 0
master
1 Branch
0 Tags
Go to file
Add file
Code
About
No description, website, or topics provided.
This branch is up to date with hshar/website:master.
Contribute
Sync fork
Ubuntu modified
883b439 · 5 years ago
2 Commits
images
final
5 years ago
Activity
0 stars
0 watching
0 forks

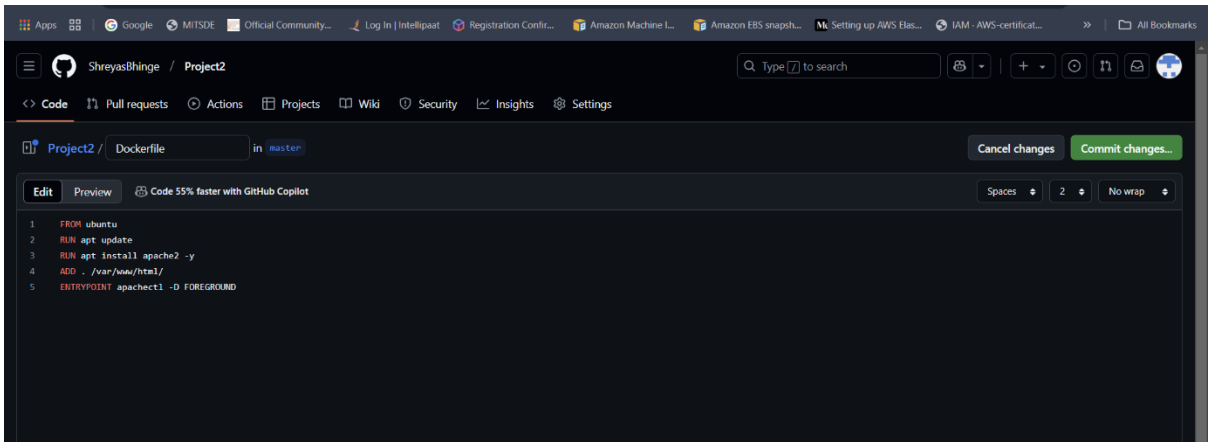

ShreyasBhinge / Project2
Code
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
General
Access
Collaborators
Moderation options
Code and automation
Branches
Tags
Rules
Actions
Webhooks
Environments
Codespaces
Pages
Security
Code security
General
Repository name
Project2
Rename
Template repository
Template repositories let users generate new repositories with the same directory structure and files. Learn more about template repositories.
Require contributors to sign off on web-based commits
Enabling this setting will require contributors to sign off on commits made through GitHub's web interface. Signing off is a way for contributors to affirm that their commit complies with the repository's terms, commonly the Developer Certificate of Origin (DCO). Learn more about signing off on commits.
Default branch
The default branch is considered the "base" branch in your repository, against which all pull requests and code commits are automatically made, unless you specify a different branch.
master
Social preview
Upload an image to customize your repository's social media preview.
https://github.com/ShreyasBhinge/Project2/settings/hooks

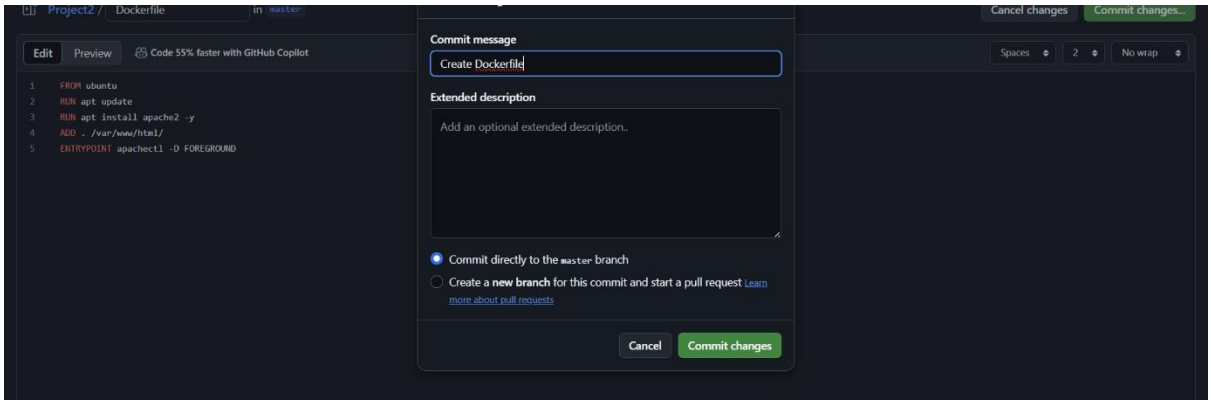

Add webhook
github.com/ShreyasBhinge/Project2/settings/hooks/new
ShreyasBhinge / Project2
Code
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
General
Access
Collaborators
Moderation options
Code and automation
Branches
Tags
Rules
Actions
Webhooks
Environments
Codespaces
Pages
Security
Code security
Deploy keys
Webhooks / Add webhook
We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in our developer documentation.
Payload URL *
http://3.236.202.233:8080/github-webhook/
Content type *
application/x-www-form-urlencoded
Secret
SSL verification
By default, we verify SSL certificates when delivering payloads.
Enable SSL verification
Disable (not recommended)
Which events would you like to trigger this webhook?
Just the push event.



Create a Docker file in Given GitHub Repository

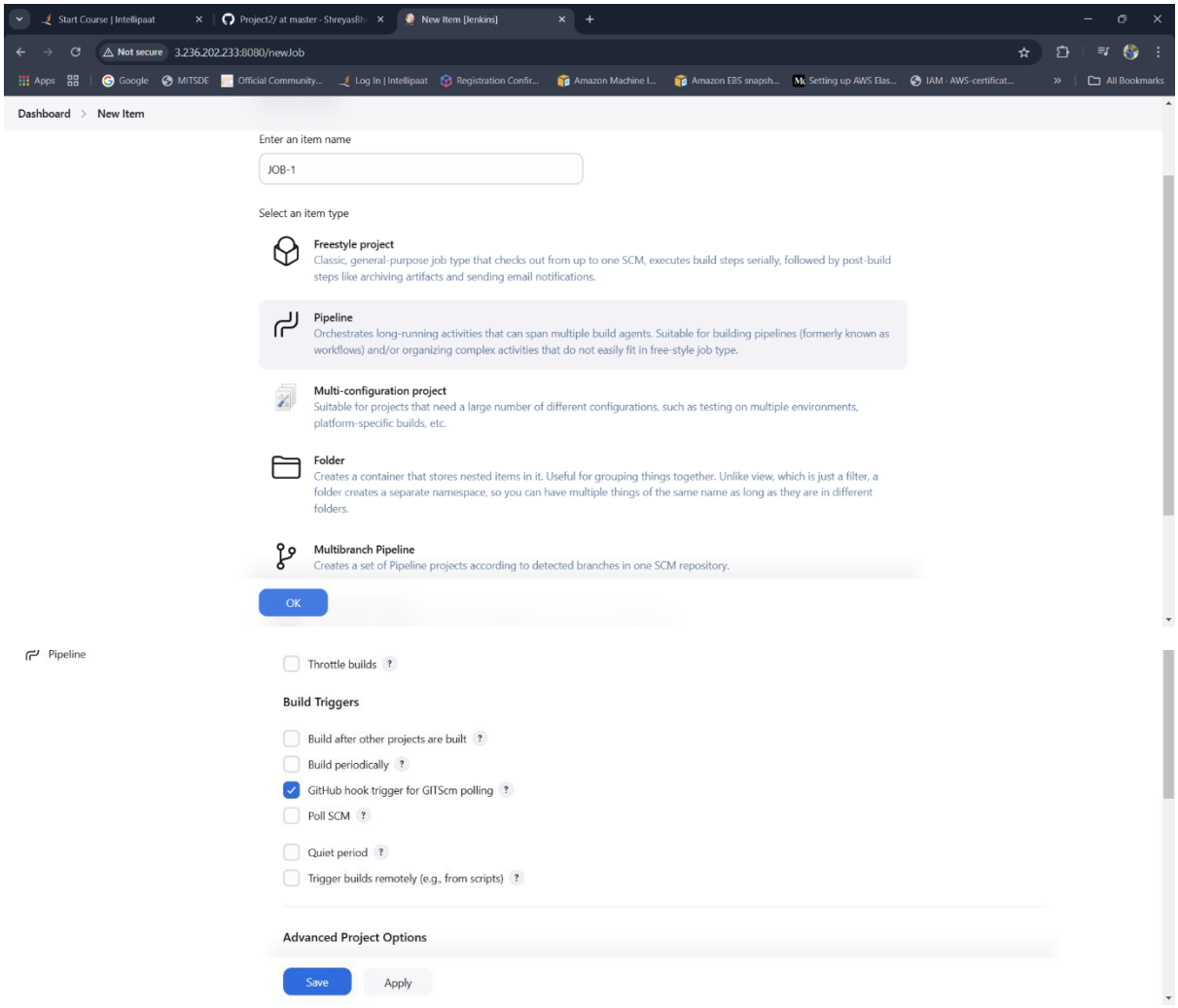




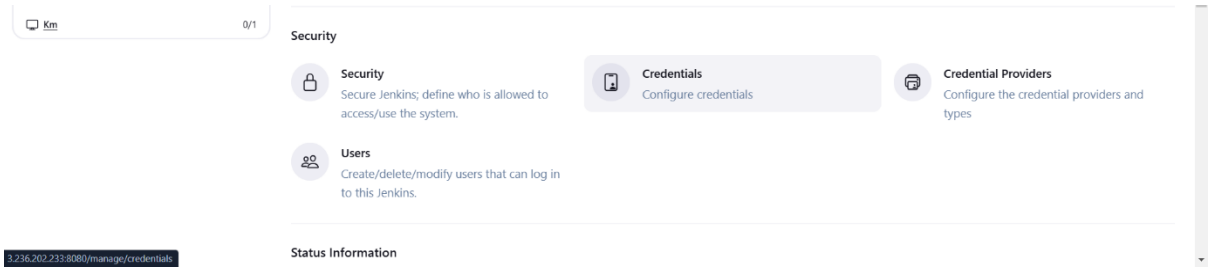


Create a Pipeline to Automate the Tasks

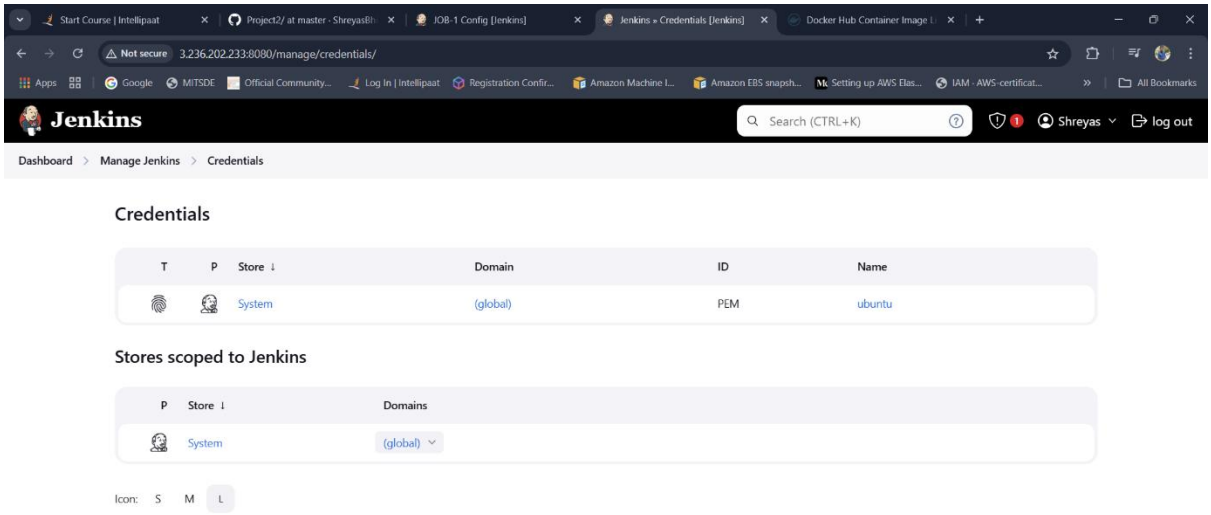
Click on “Create a job”.



Add DockerHub credential to Credentials

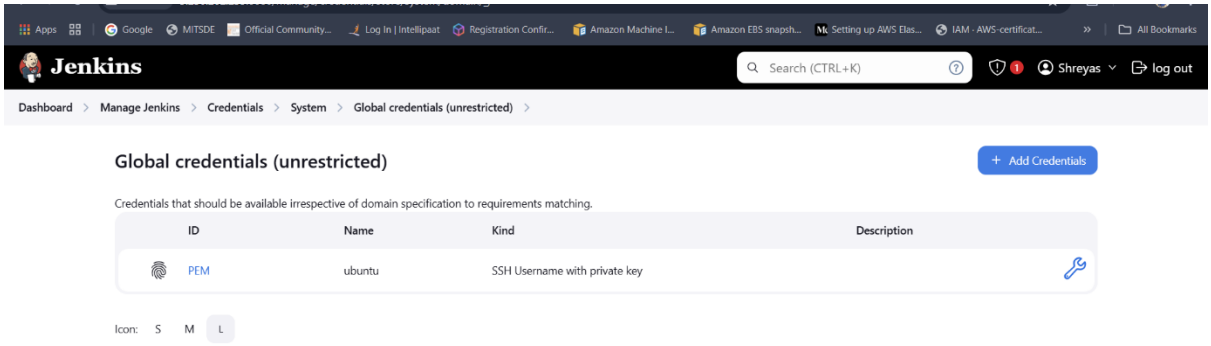


Click on global

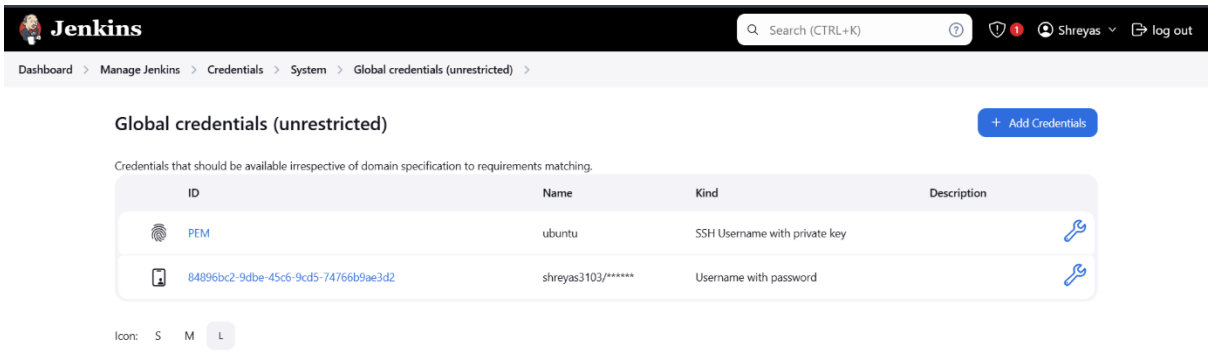


3.236.202.233:8080/manage/credentials/store/system/domain/_/

Click on Add credential

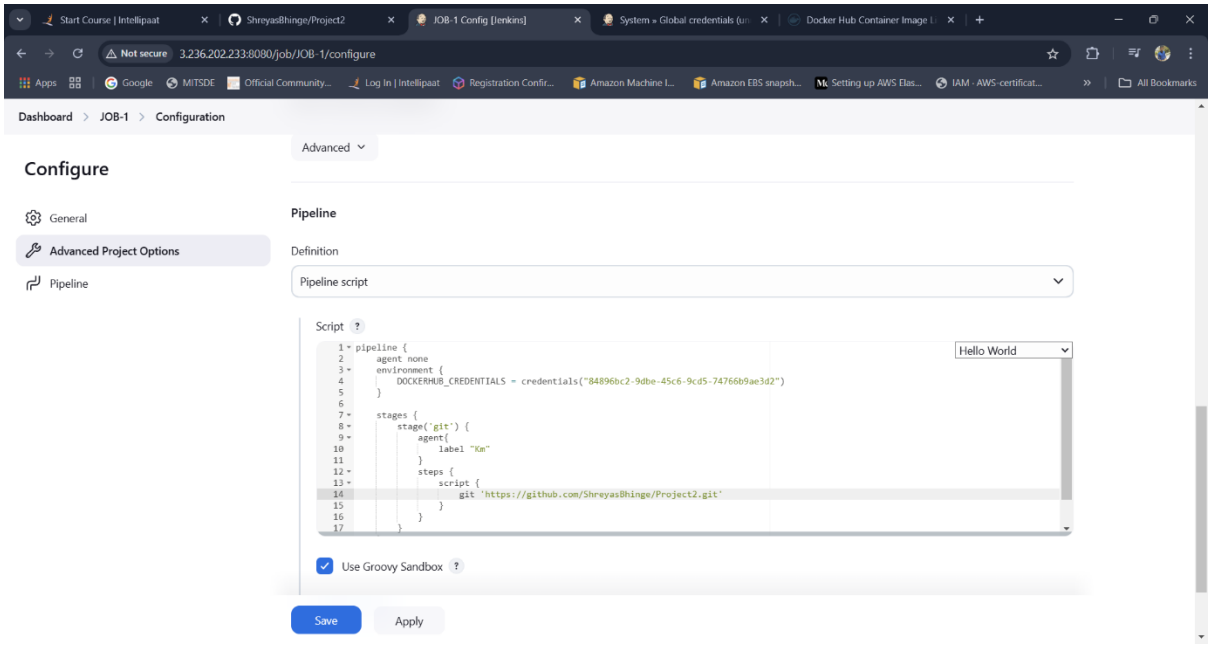


Enter your dockerhub user name and password

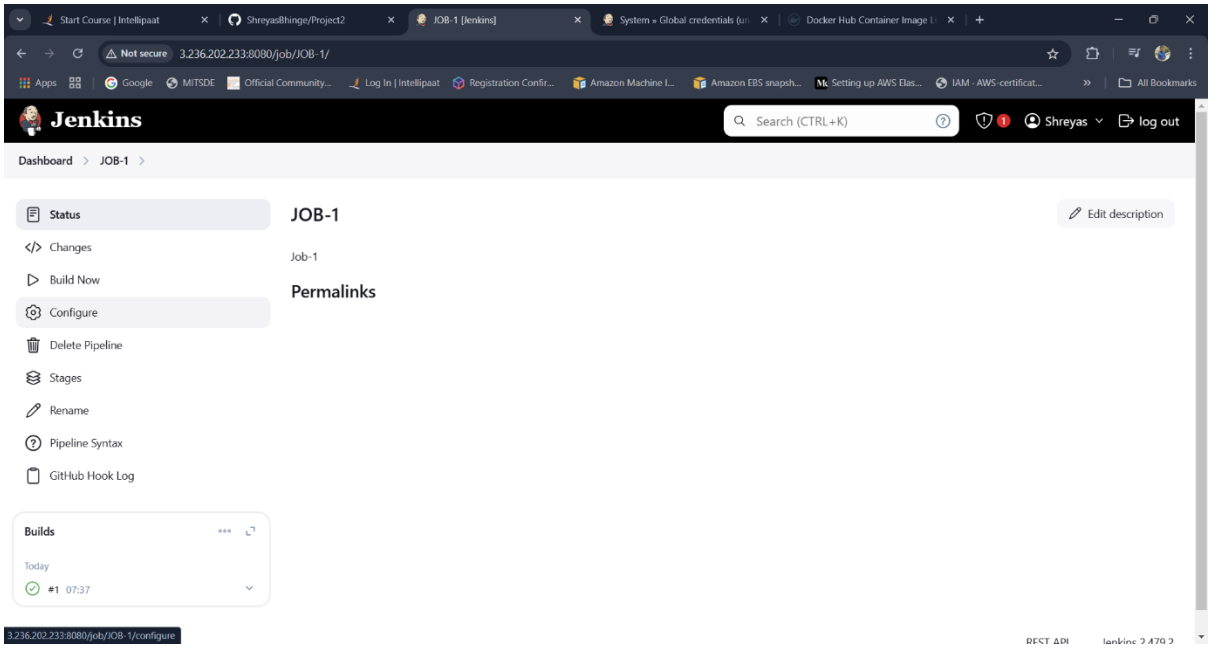


Go to the “Pipeline” section & choose the “Hello World” script here.

Edit the script add you github url and docker credential



Apply the script

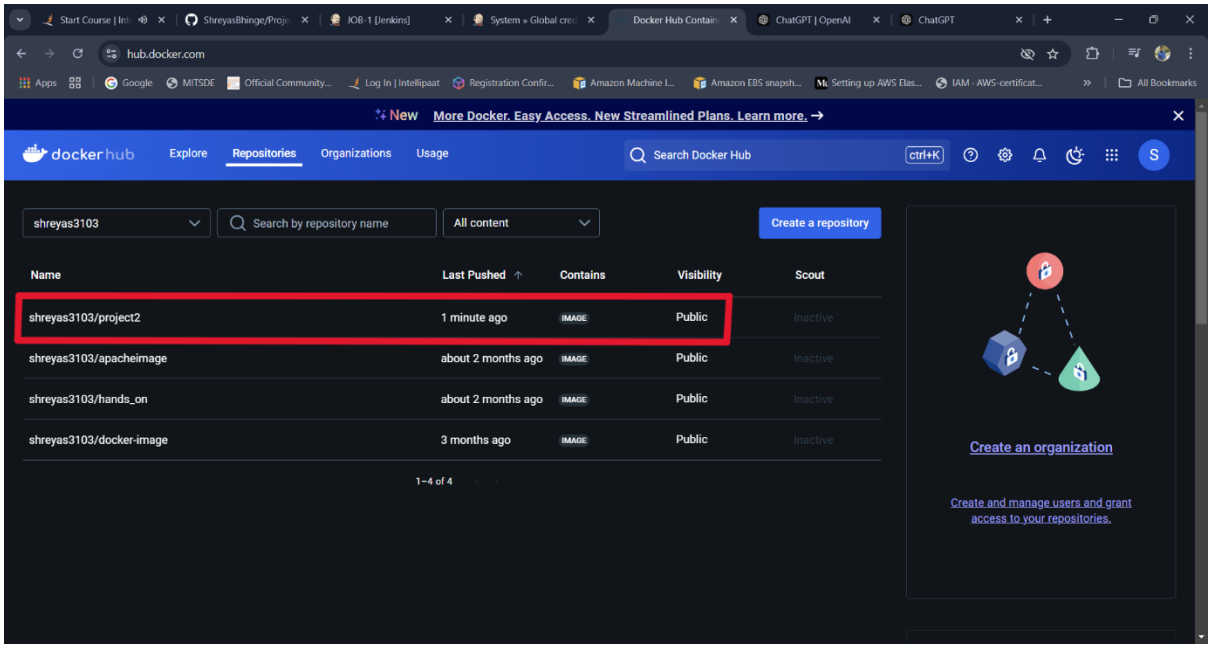


Confirm it

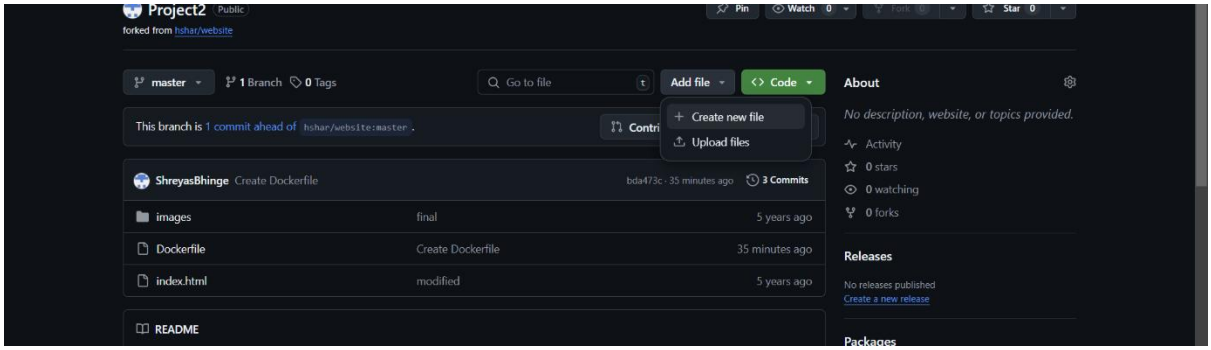
Again Configure

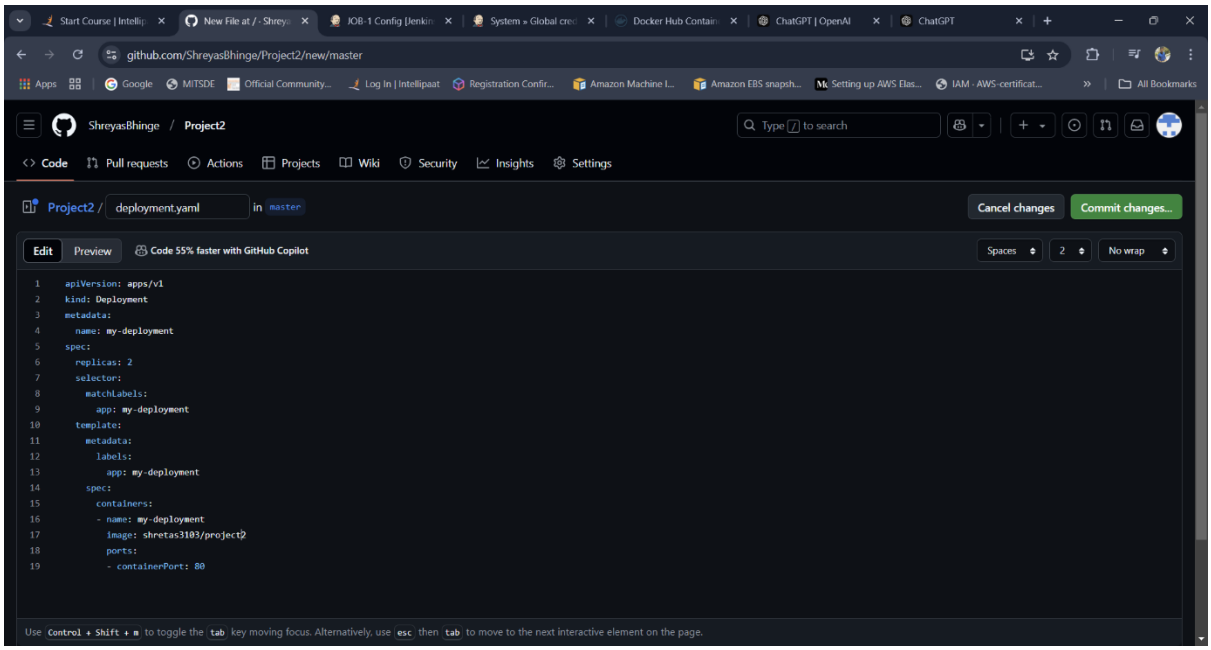
Now, we will push the “Docker Hub Image” using the pipeline code.

Apply it

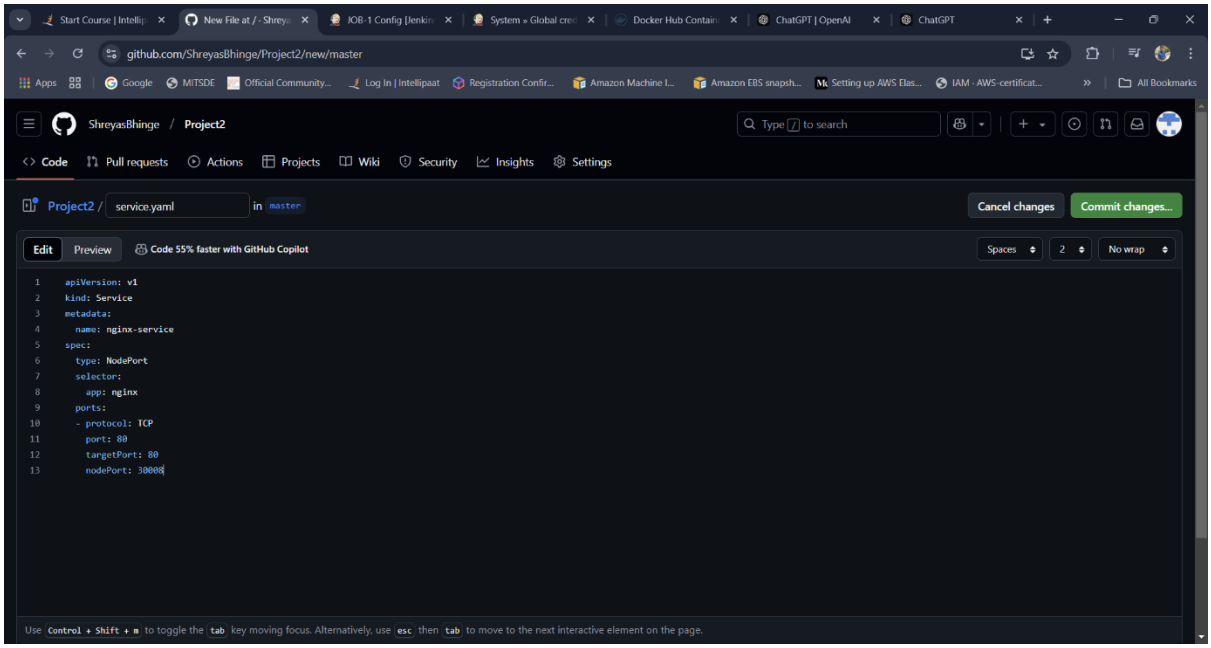


Now, we will create the “deployment.yaml” & “service.yaml” file to deploy the website using the “Kubernetes” tool.





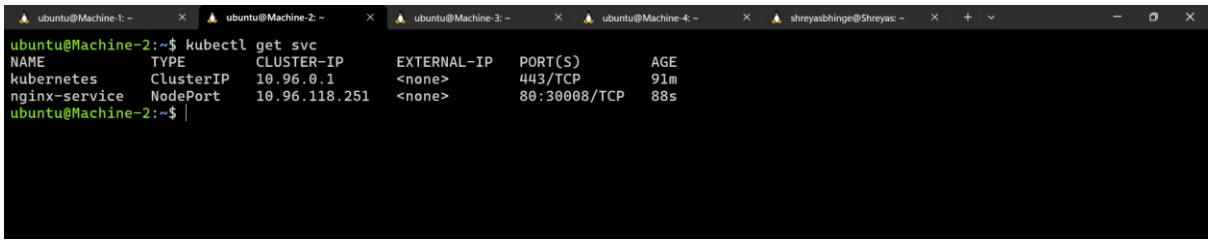
Service.yaml

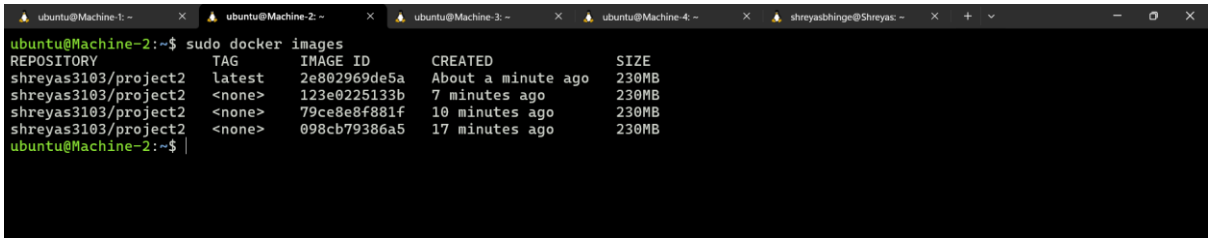


Add Kubernetes stage in pipline below docker

```
    stage('K8s') {
        agent {
            label 'Kubernetes-Master'
        }
        steps {
            sh 'kubectl apply -f deployment.yaml'
            sh 'kubectl apply -f service.yaml'
        }
    }
  }
}
```

Now apply





Pipeline script

```
pipeline {
    agent none
    environment {
        DOCKERHUB_CREDENTIALS = credentials("cb26ac19-f954-40ca-a12e-d790594bcca7")
    }
    stages {
        stage('git') {
            agent {
                label "Km"
            }
            steps {
                script {
```

```
            git'https://github.com/Sameer-8080/website.git'
        }
      }
    }
    stage('docker') {
      agent {
        label "K8-Master"
      }
      steps {
        script {
          sh 'sudo docker build shreyas3103/project2'
          sh 'sudo docker login -u ${DOCKERHUB_CREDENTIALS_USR} -p
${DOCKERHUB_CREDENTIALS_PSW}'
          sh 'sudo docker push shreyas3103/project2'
        }
      }
    }
    stage('kubernetes') {
      agent {
        label "K8-Master"
      }
      steps {
        script {
          sh 'kubectl delete deploy nginx-deployment'
          sh 'kubectl apply -f deployment.yaml'
          sh 'kubectl delete service my-service'
          sh 'kubectl apply -f service.yaml'
        }
      }
    }
```

```
    }
}
```