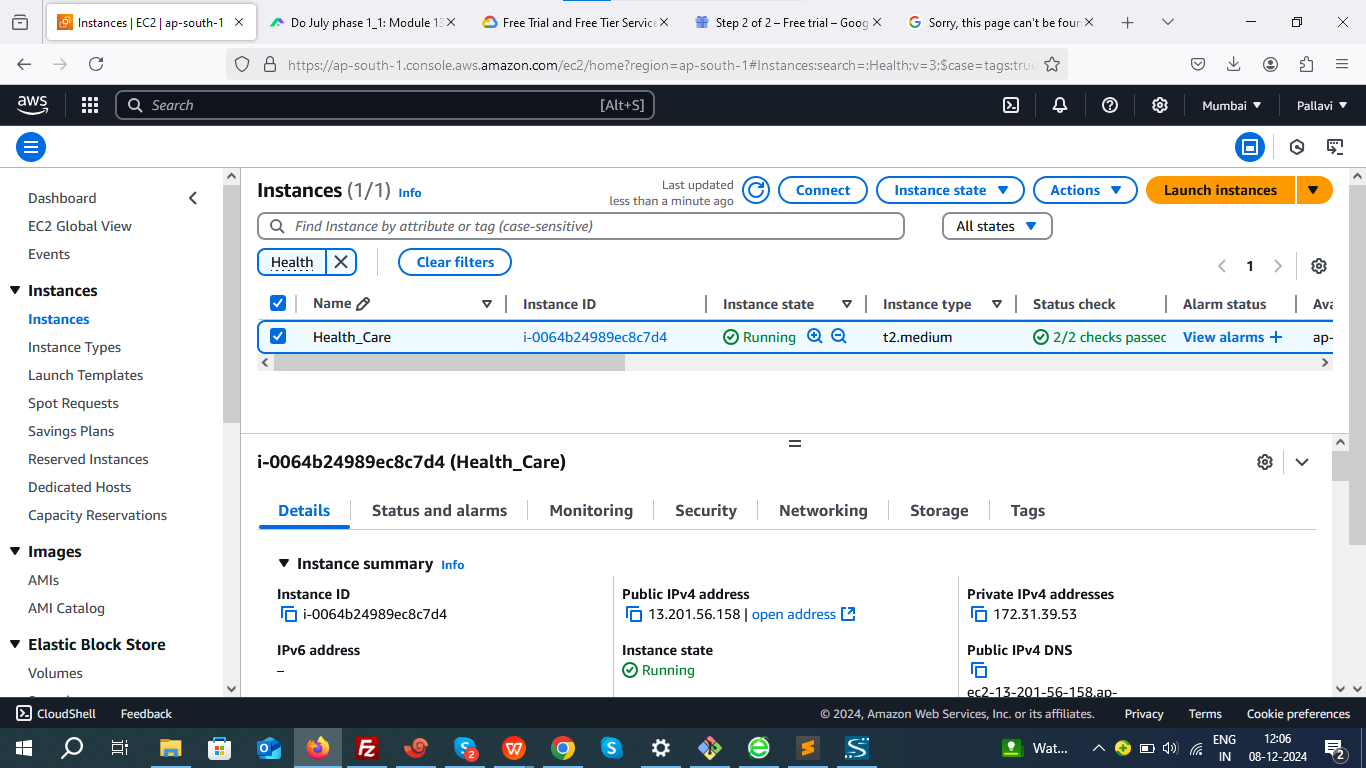
****Step 1: Create a VM and Install Terraform****

**1.Launch a VM in AWS**:

1. Instance Type: t2.medium
2. Security Group: Allow **All traffic (Anywhere)**.

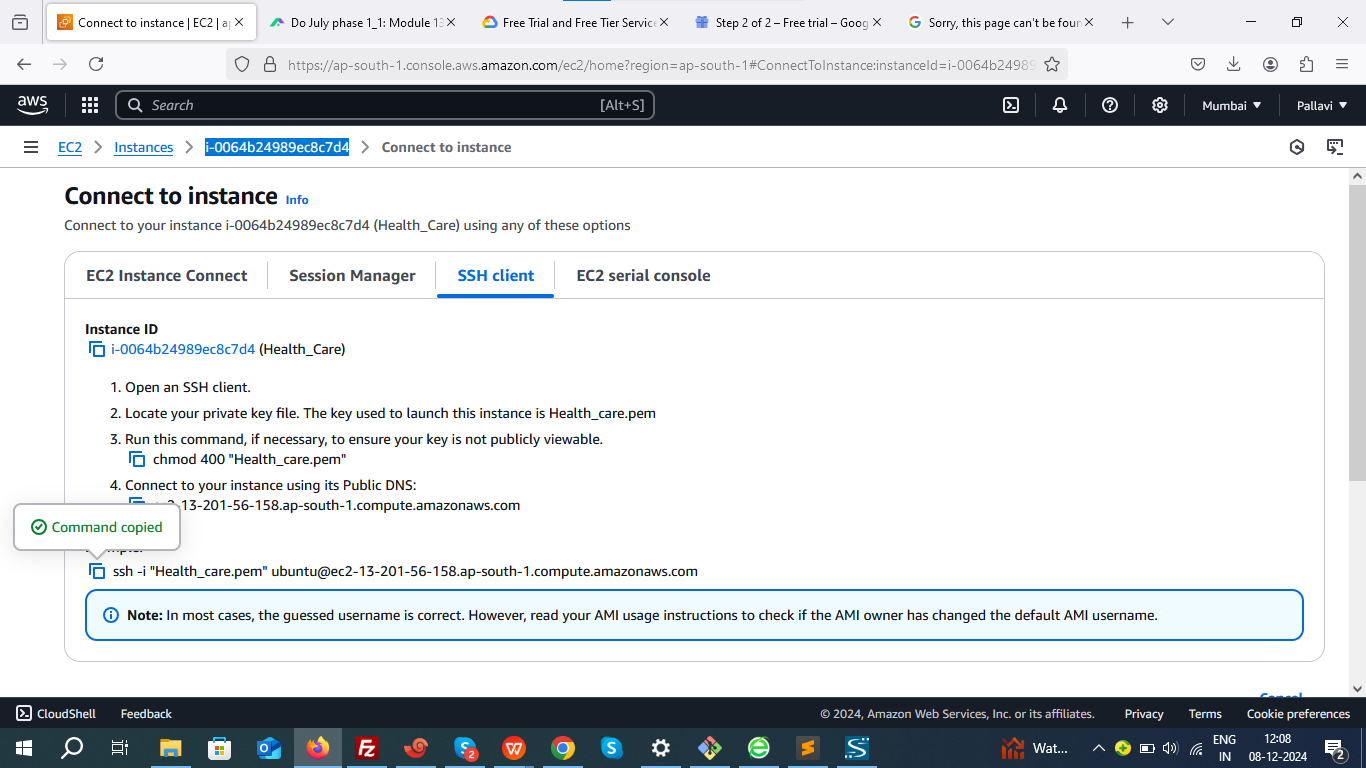


**2. Connect to the VM**:

Use SSH to connect.

EC2>>Instances>>i-0064b24989ec8c7d4>>Connect\_to\_instance

ssh -i "Health\_care.pem" [ubuntu@ec2-13-201-56-158.ap-south-1.compute.amazonaws.com](mailto:ubuntu@ec2-13-201-56-158.ap-south-1.compute.amazonaws.com)



**3. Install Terraform**:

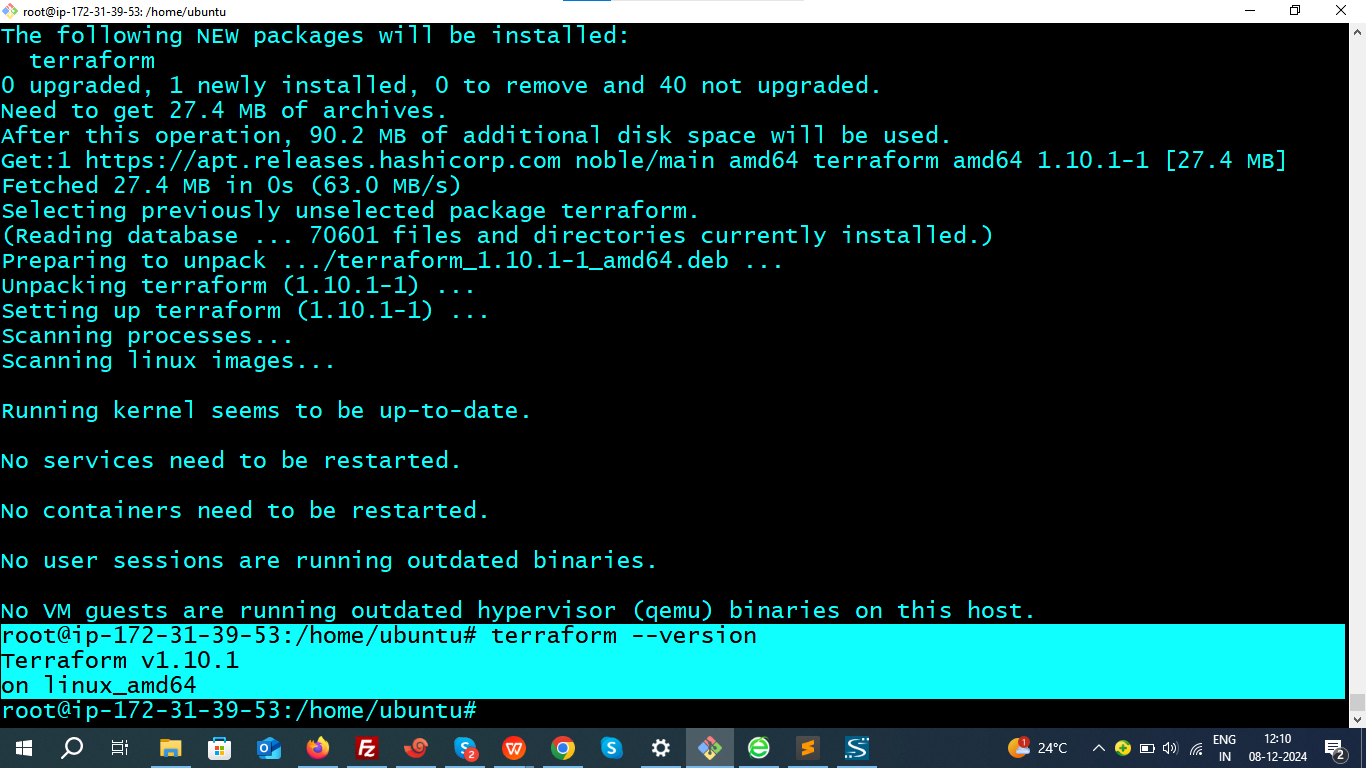
wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

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

Verify installation.

terraform --version



**4.Establishing a connection between Terraform and AWS**.

Create AWS Access Key and Secret Key.

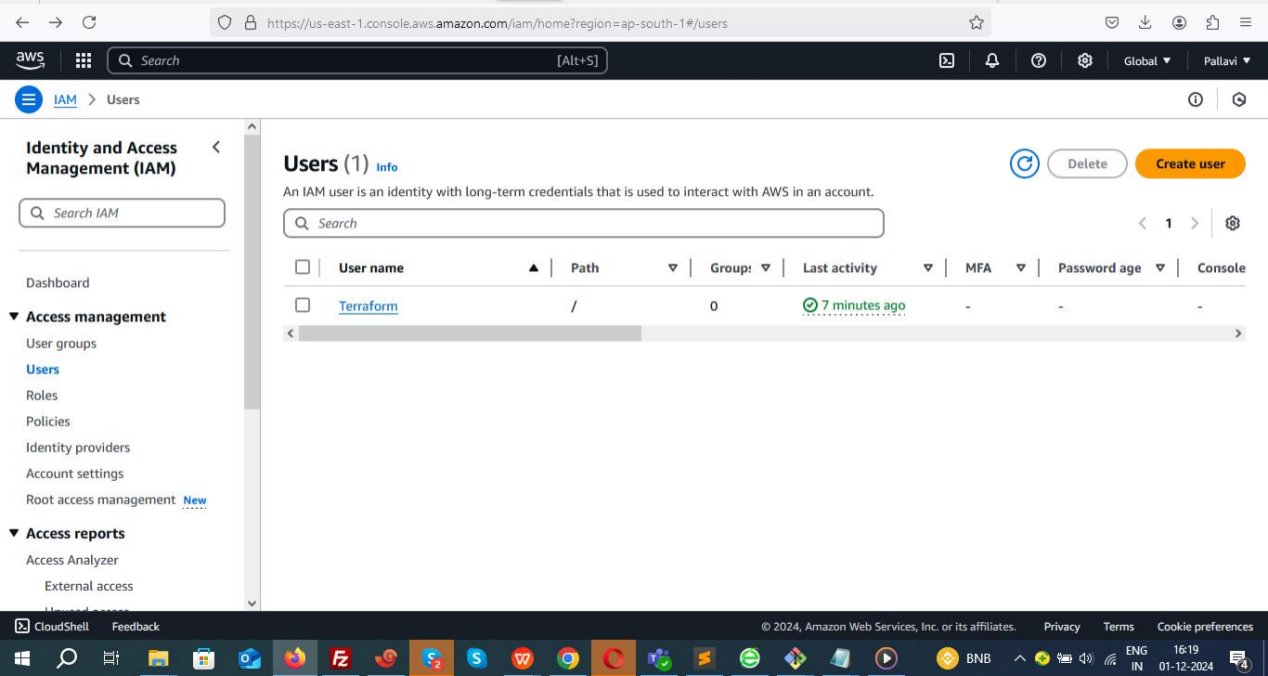
Creating an Access Key and Secret Key in AWS is a crucial step to allow Terraform to authenticate and interact with AWS resources.

1. **Create an AWS IAM User with Administrator Access**

**Login to AWS Management Console.**

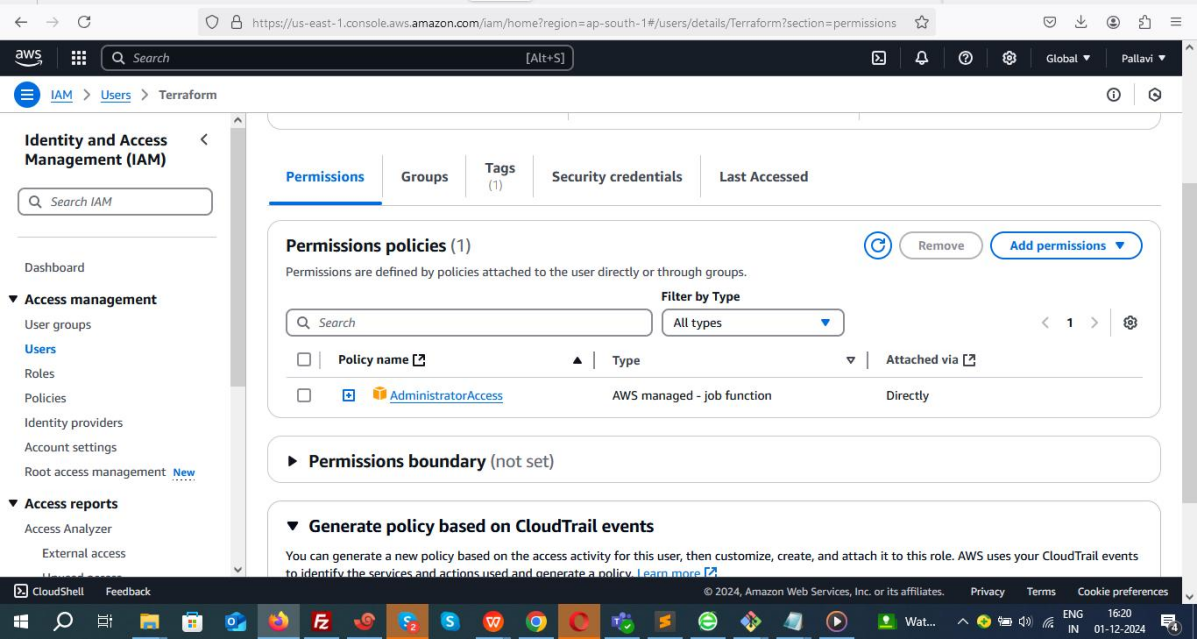
**Navigate to IAM > Users.**

**Create a New User:** Enable **Programmatic Access**.



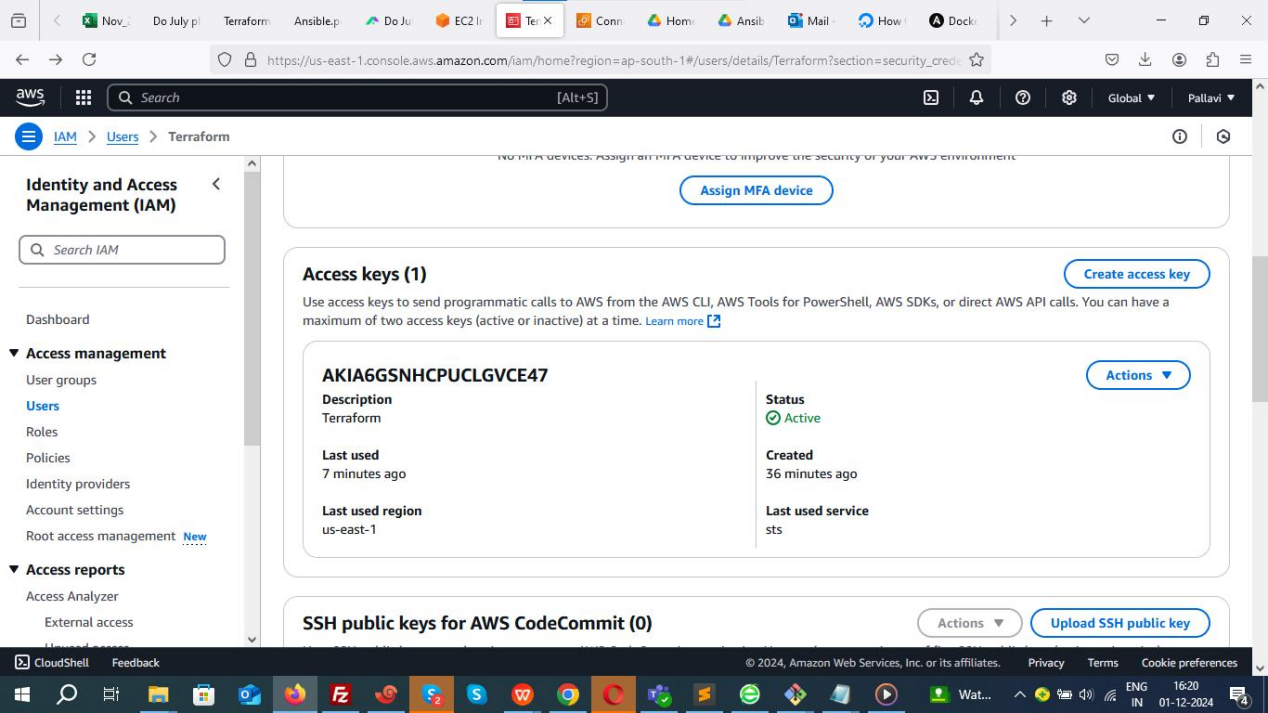
**Attach a Policy:**

Attach the **AdministratorAccess** policy to the user.



**Generate Access Keys:**

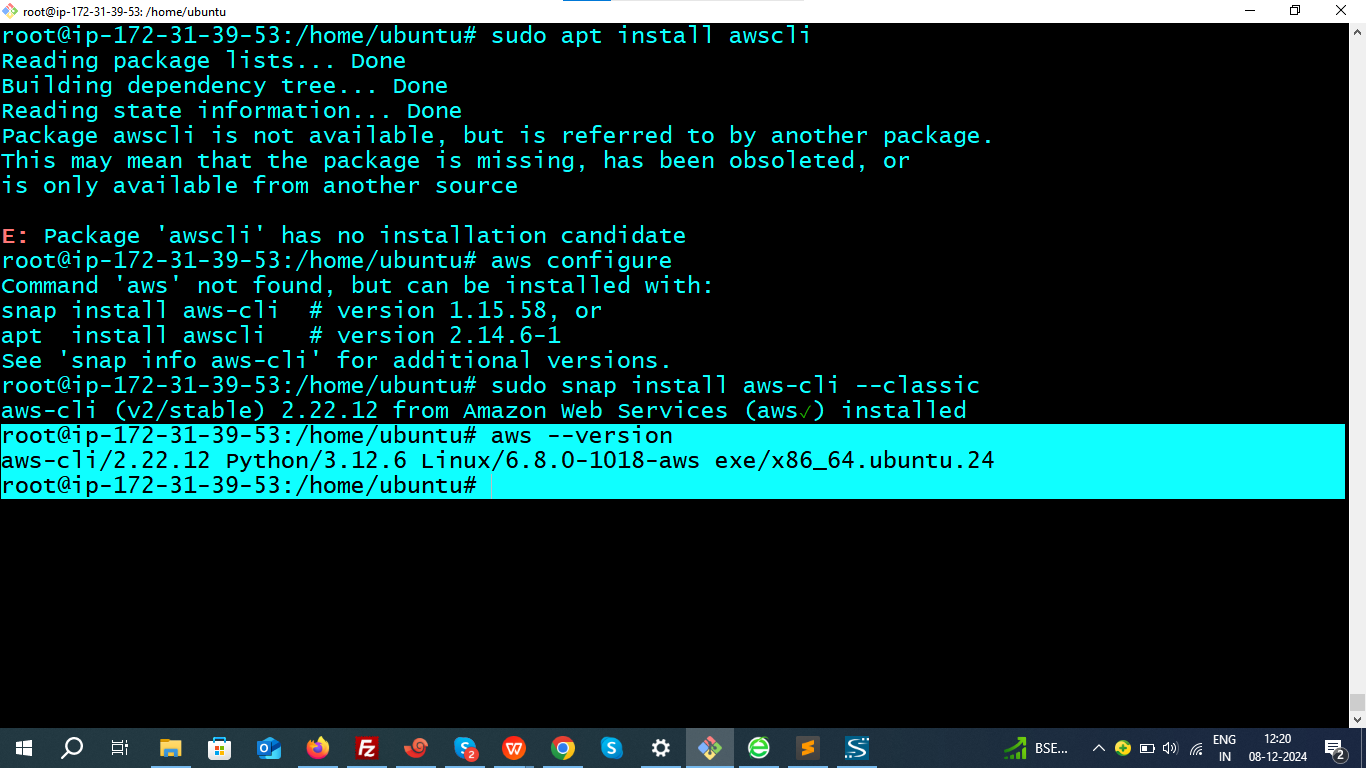
Save the **Access Key ID** and **Secret Access Key** securely (you will need these).



1. **Configure AWS CLI**

**Install AWS CLI (if not installed):**

sudo apt install awscli



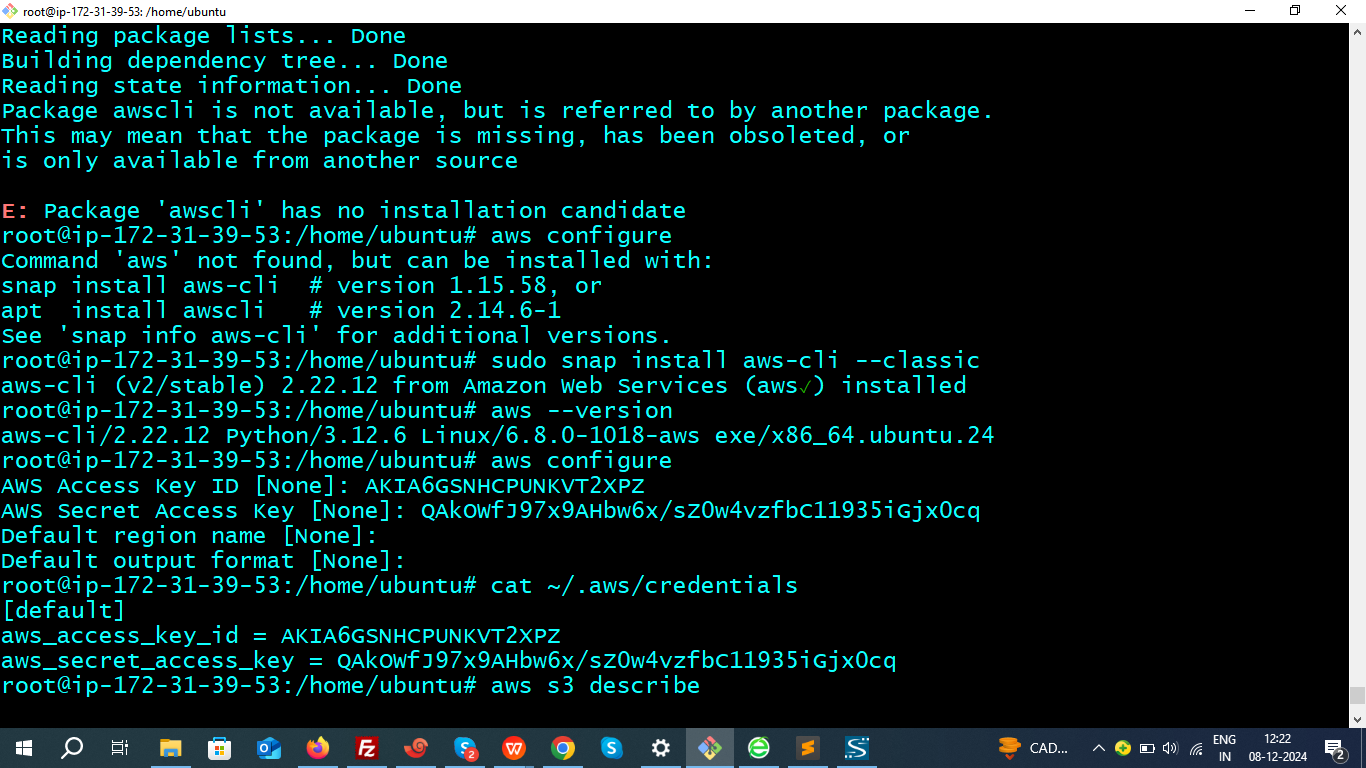
aws configure

Enter the **Access Key ID**, **Secret Access Key**, region (e.g., us-east-1), and output

format (json).

**Verify AWS Configuration:**

cat ~/.aws/credentials



**5.Write Terraform Configuration File**:

**Create dedicated directory and navigate.**

mkdir health\_care && cd health\_care

vim aws\_infra.tf

Paste the provided Terraform configuration code to create an EC2 instance, VPC, Subnet, Security Group, and install Ansible.

provider "aws" {

region = "us-east-2"

shared\_credentials\_files = ["~/.aws/credentials"]

}

resource "tls\_private\_key" "mykey" {

algorithm = "RSA"

}

resource "aws\_key\_pair" "aws\_key" {

key\_name = "web-key"

public\_key = tls\_private\_key.mykey.public\_key\_openssh

provisioner "local-exec" {

command = "echo '${tls\_private\_key.mykey.private\_key\_openssh}' > ./web-key.pem"

}

}

resource "aws\_vpc" "sl-vpc" {

cidr\_block = "10.0.0.0/16"

tags = {

Name = "sl-vpc"

}

}

resource "aws\_subnet" "subnet-1" {

vpc\_id = aws\_vpc.sl-vpc.id

cidr\_block = "10.0.1.0/24"

depends\_on = [aws\_vpc.sl-vpc]

map\_public\_ip\_on\_launch = true

tags = {

Name = "sl-subnet"

}

}

resource "aws\_route\_table" "sl-route-table" {

vpc\_id = aws\_vpc.sl-vpc.id

tags = {

Name = "sl-route-table"

}

}

resource "aws\_route\_table\_association" "a" {

subnet\_id = aws\_subnet.subnet-1.id

route\_table\_id = aws\_route\_table.sl-route-table.id

}

resource "aws\_internet\_gateway" "gw" {

vpc\_id = aws\_vpc.sl-vpc.id

depends\_on = [aws\_vpc.sl-vpc]

tags = {

Name = "sl-gw"

}

}

resource "aws\_route" "sl-route" {

route\_table\_id = aws\_route\_table.sl-route-table.id

destination\_cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.gw.id

}

variable "sg\_ports" {

type = list(number)

default = [8080, 80, 22, 443]

}

resource "aws\_security\_group" "sl-sg" {

name = "sg\_rule"

vpc\_id = aws\_vpc.sl-vpc.id

dynamic "ingress" {

for\_each = var.sg\_ports

iterator = port

content {

from\_port = port.value

to\_port = port.value

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

}

resource "aws\_instance" "myec2" {

ami = "ami-005fc0f236362e99f"

instance\_type = "t2.medium"

key\_name = "web-key"

subnet\_id = aws\_subnet.subnet-1.id

security\_groups = [

aws\_security\_group.sl-sg.id

]

tags = {

Name = "Health\_Care"

}

provisioner "remote-exec" {

connection {

type = "ssh"

user = "ubuntu"

private\_key = tls\_private\_key.mykey.private\_key\_pem

host = self.public\_ip

}

inline = [

"sudo apt update",

"sudo apt install software-properties-common",

"sudo add-apt-repository --yes --update ppa:ansible/ansible",

"sudo apt install ansible -y"

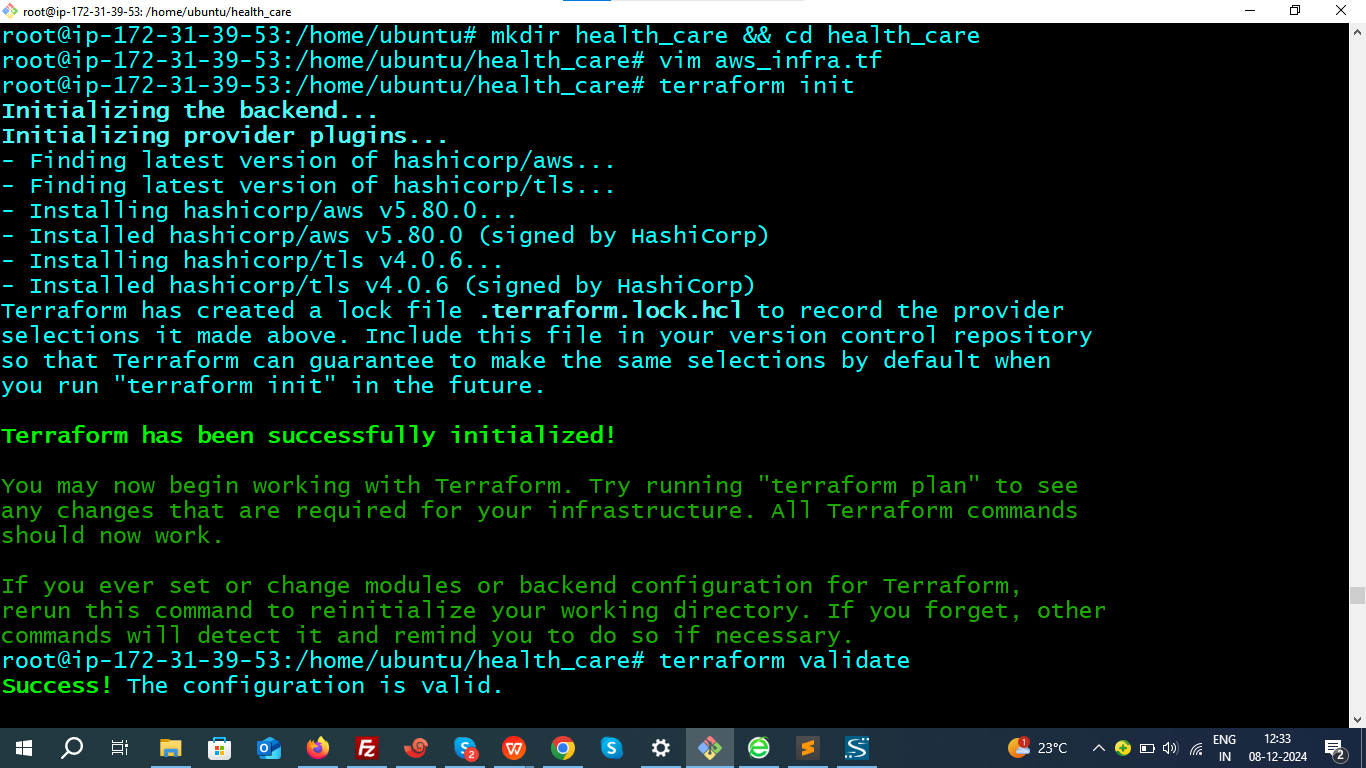
]

}

}

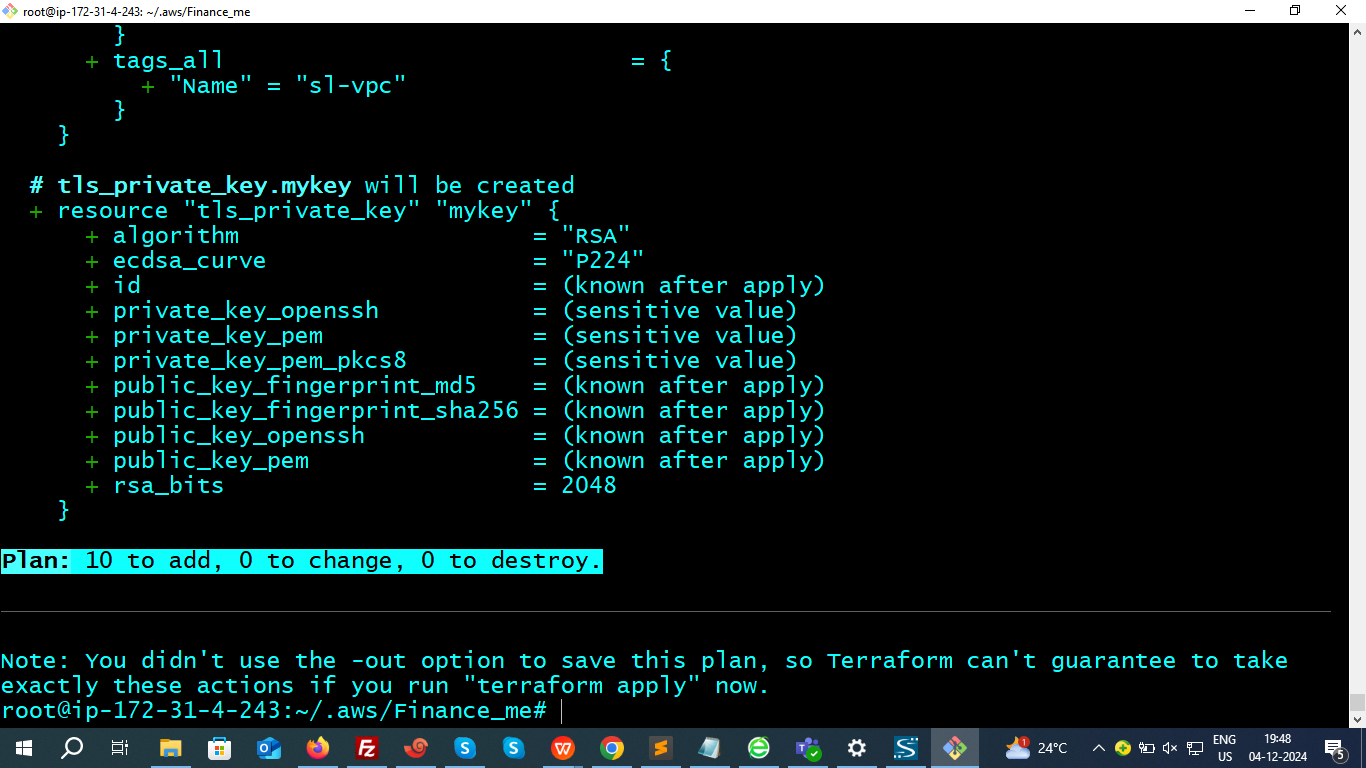
**Initialize and Apply Terraform**:

terraform init - Prepares the working directory for Terraform by downloading plugins and setting up the backend.

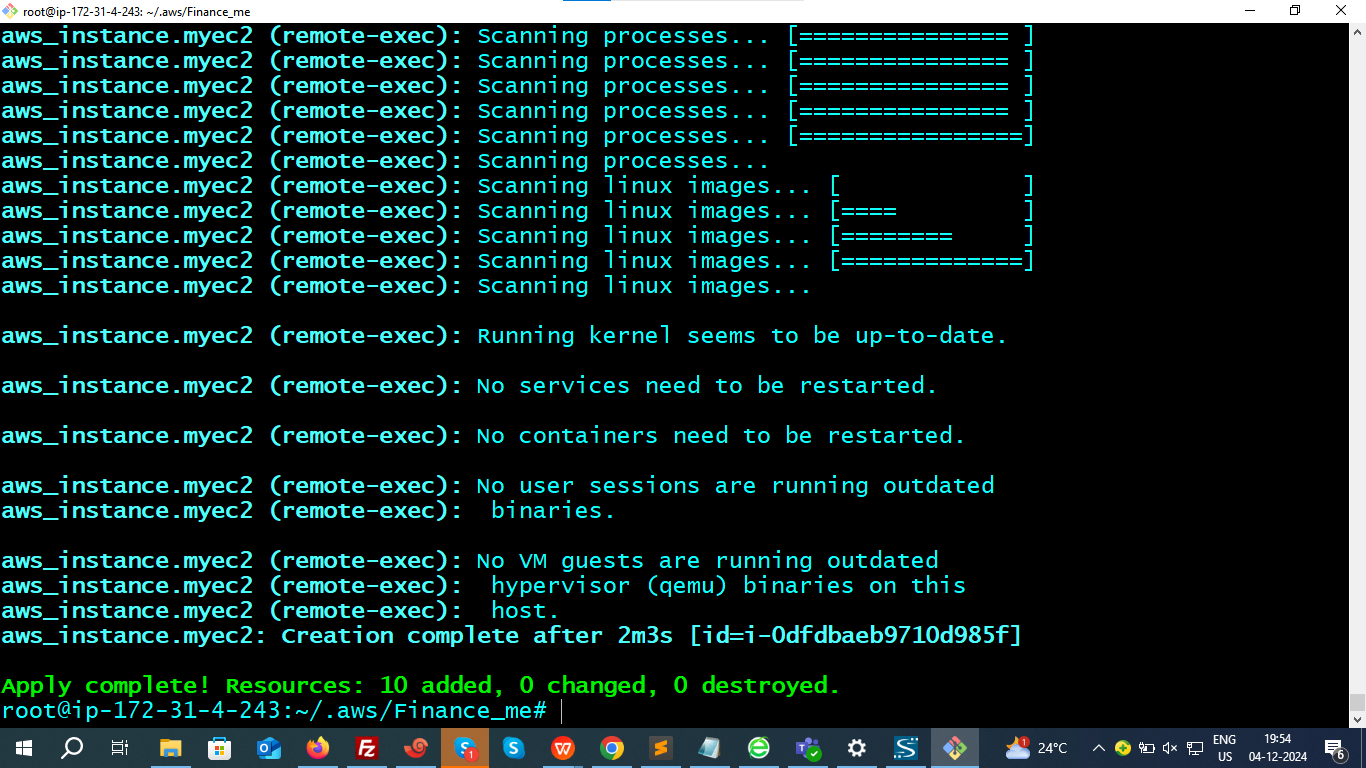


terraform validate - Checks for syntax errors and configuration validity.

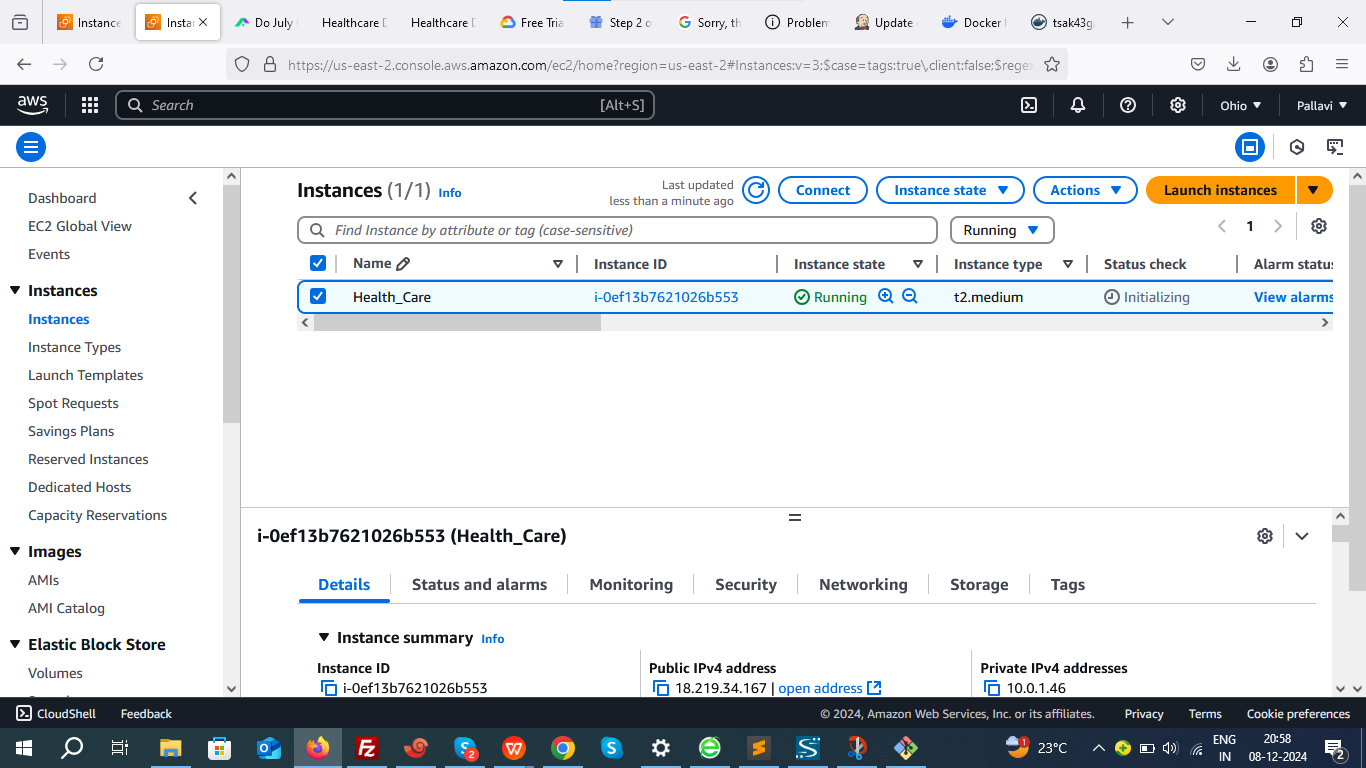
terraform plan - Generates and displays an execution plan without making changes.



terraform apply --auto-approve - Executes the plan and creates or updates infrastructure without requiring manual confirmation.



Instance has been created in us-east-2



### ****Step 2: Configure Ansible on the Created EC2 Instance****

1. **Connect to the Newly Created EC2 Instance**:

Use SSH with the generated private key (web-key.pem).

1. **Check Ansible Installation**

ansible --version



1. **Install Java, git and docker Using Ansible Playbook**:

Create a dedictaed directory fro ansible and create ansible playbook in it.

Create an Ansible playbook:

vim health\_care.yml

Add the following:

- name: Install and set up DevOps tools (Java, Git, Docker)

hosts: localhost

become: true

tasks:

- name: Update the apt repo

apt:

update\_cache: yes

- name: Install multiple packages

apt:

name: "{{ item }}"

state: present

loop:

- git

- docker.io

- openjdk-17-jdk

- name: Start Docker service

service:

name: "{{ item }}"

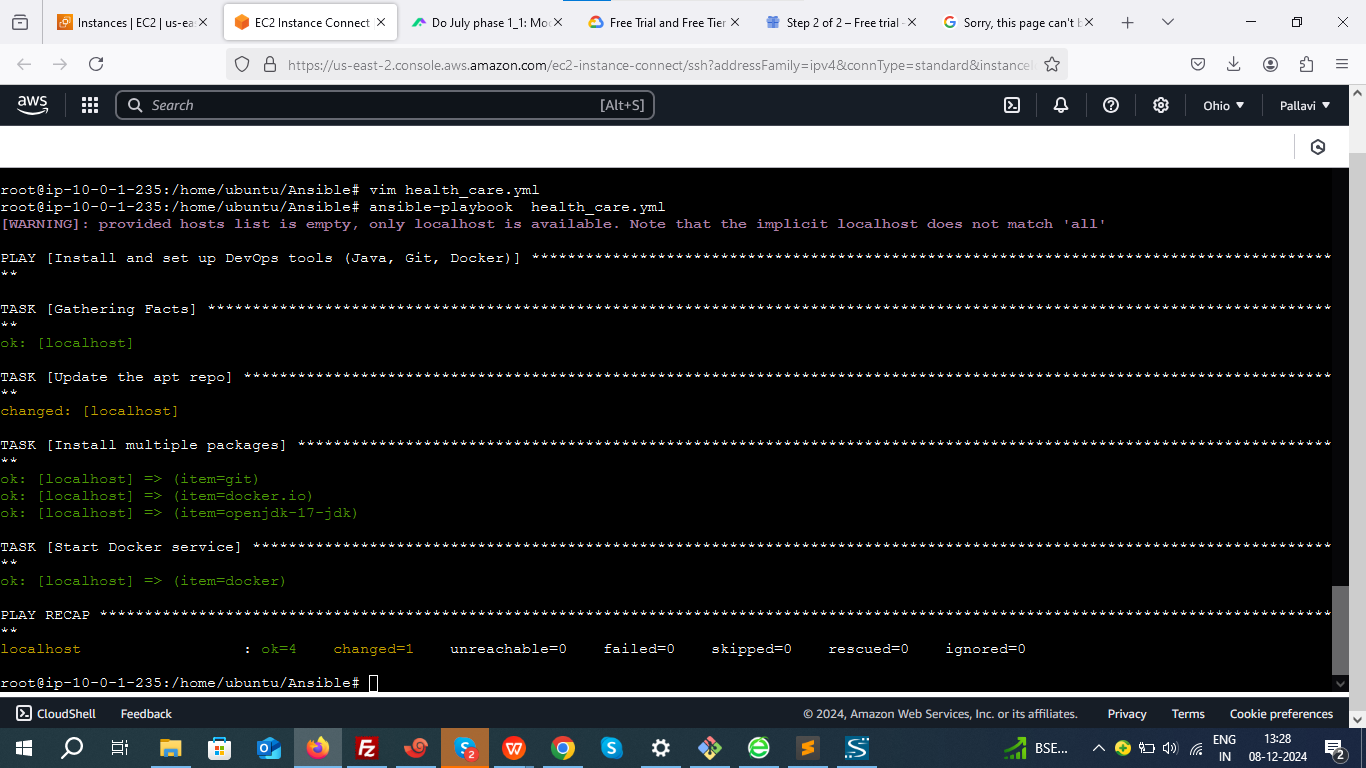
state: started

loop:

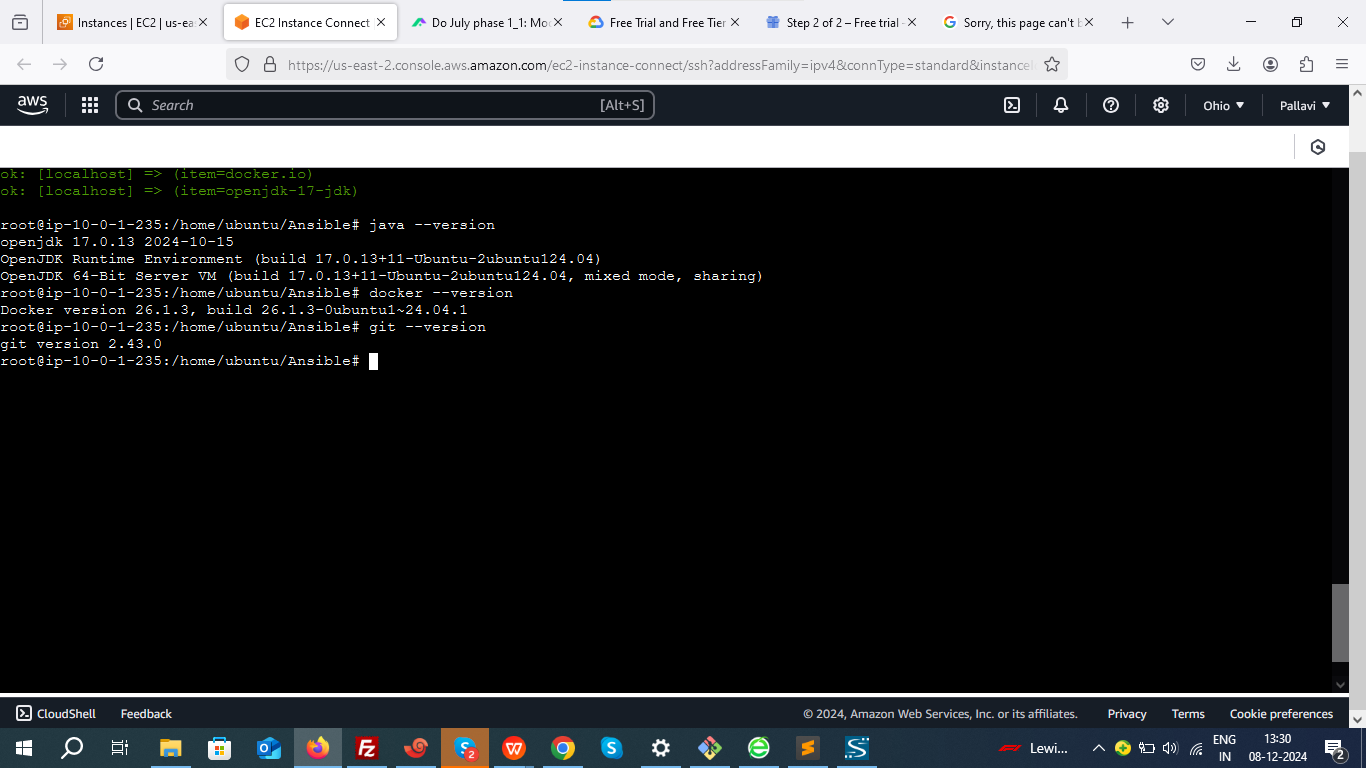
- docker

Run the playbook:

ansible-playbook health\_care.yml



Verify installations.



### ****Step 3: Installing and setup Jenkins CI/CD Pipeline****

**1. Install Jenkins.**

**ADD GPG KEY:**

sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \

https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

**ADD Jenkins repository:**

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

**UPDATE:**

sudo apt-get update

**INSTALL**

sudo apt-get install jenkins -y

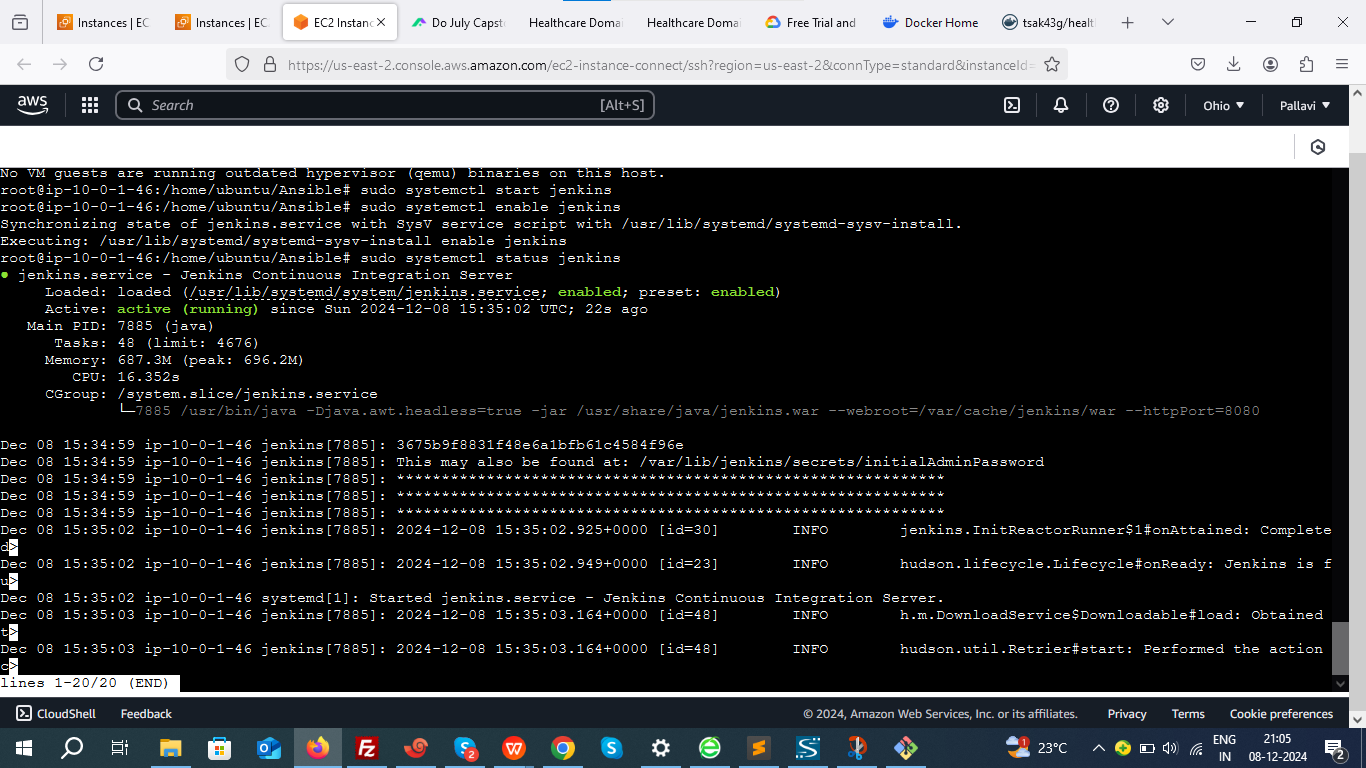
**Start and Enable Jenkins:**

sudo systemctl start jenkins

sudo systemctl enable jenkins

**Verify:**

sudo systemctl status jenkins



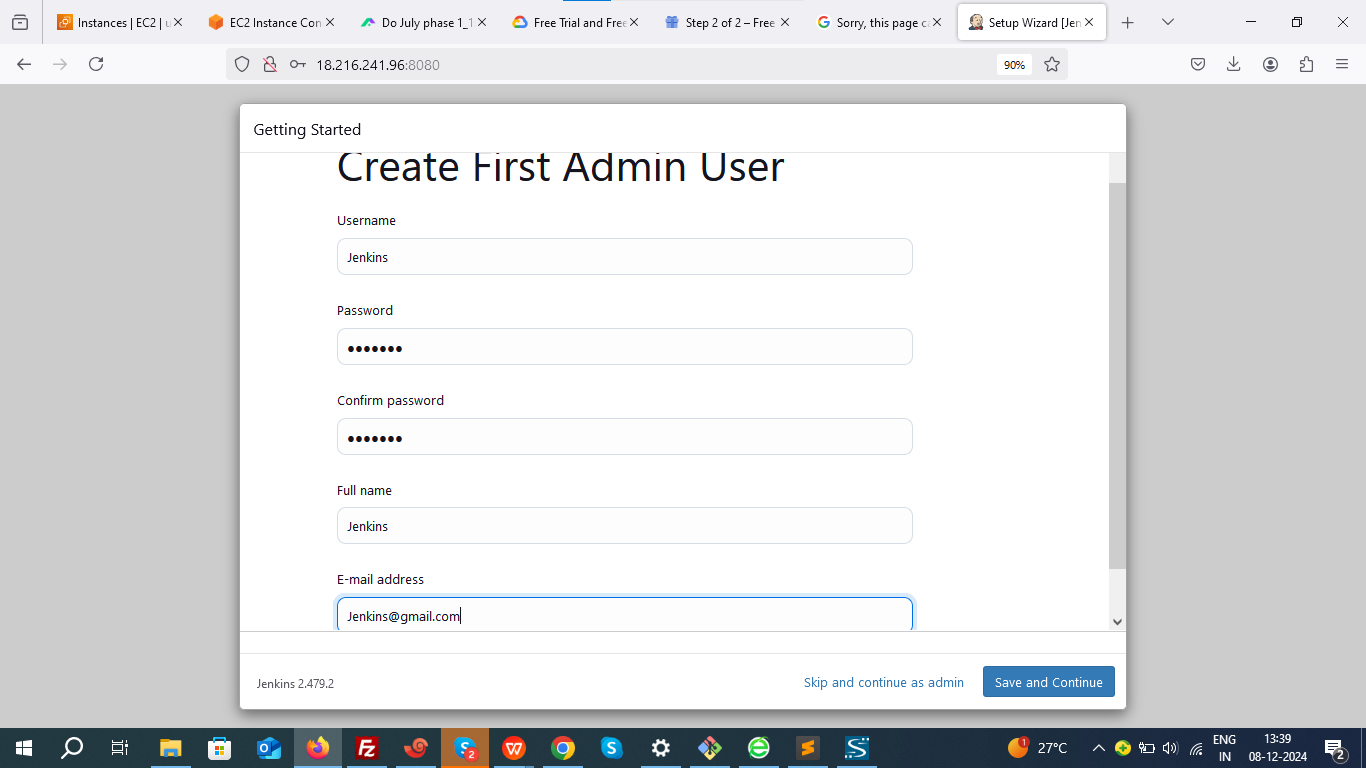
1. **Now access to the jenkins dashboard using below url:**

http://18.219.34.167:8080

Default port: 8080.

1. **Set up plugins and admin user.**

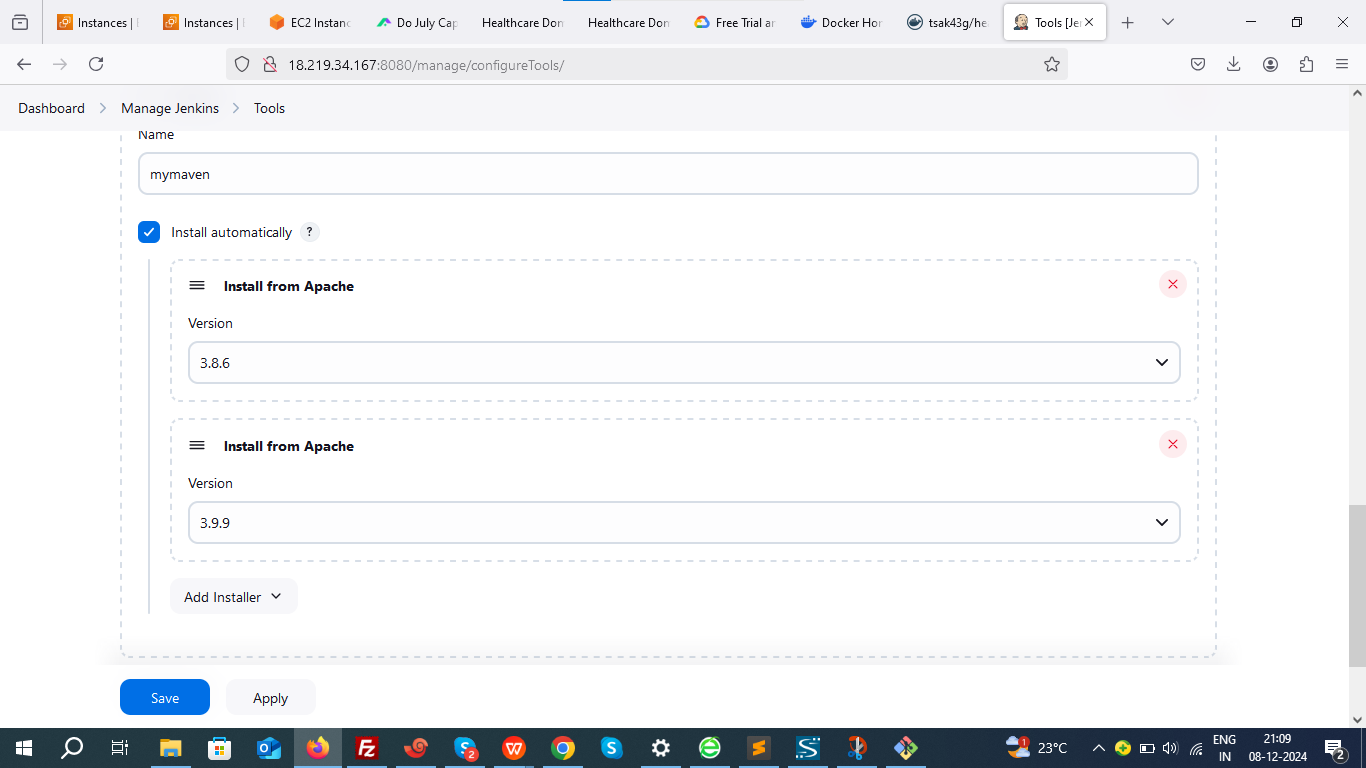
cat /var/lib/jenkins/secrets/initialAdminPassword



1. **Configure Maven**:

Go to **Manage Jenkins > Global Tool Configuration**.

Add Maven and set it up.



Make sure to give same name for maven in pipeline.

1. Write docker file and add it in your github repository.

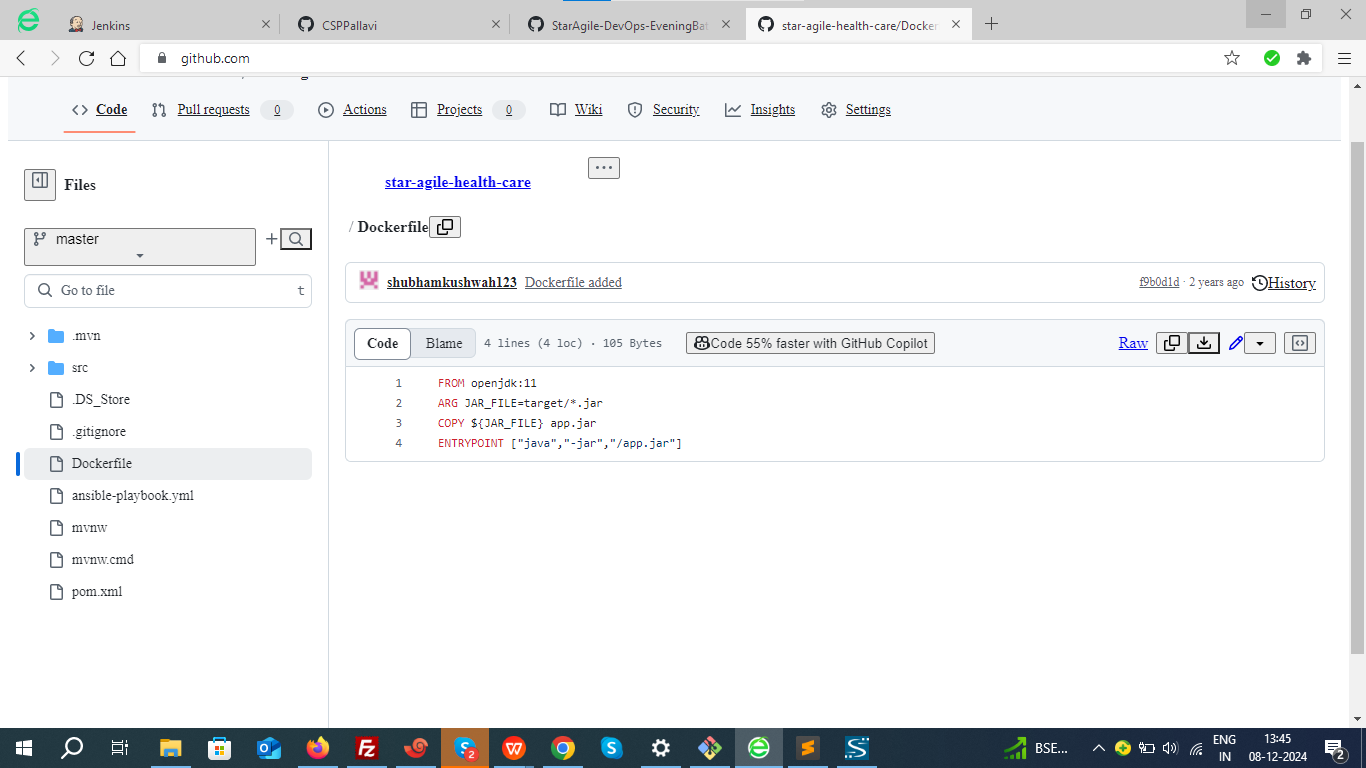
FROM openjdk:11

ARG JAR\_FILE=target/\*.jar

COPY ${JAR\_FILE} app.jar

EXPOSE 8082

ENTRYPOINT ["java","-jar","/app.jar"]

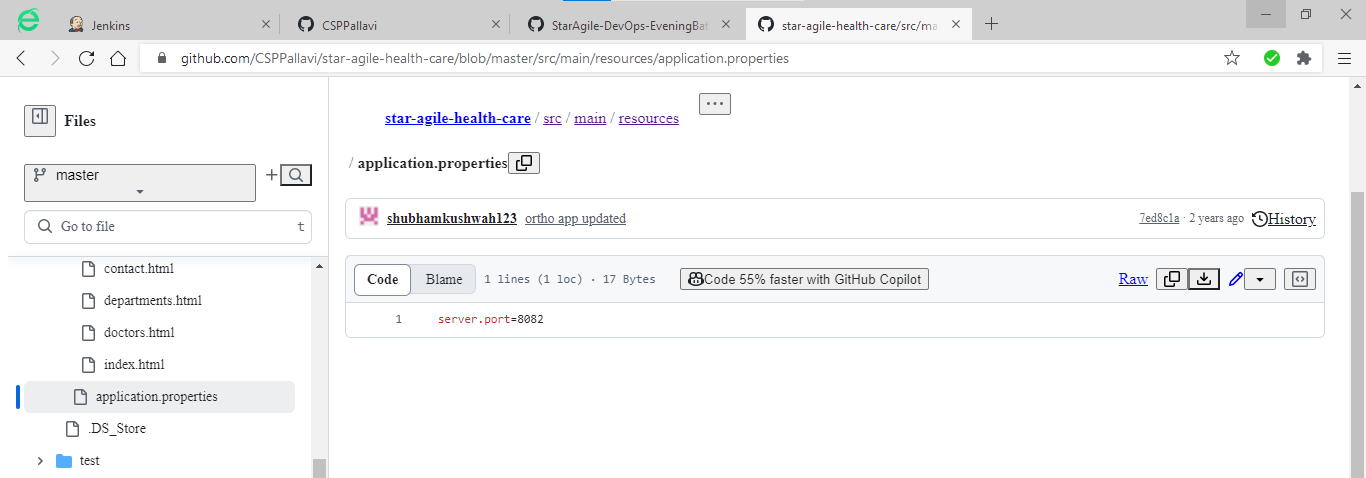


1. **Create Pipeline**:

Points to be noted befor running pipeline:

In source code go to src/main/resources/application.properties

1. **Check server port**



1. **Map the internal container port (8082) to an external port on your host machine**

docker run -d -p 8082:8082 myproject1:$BUILD\_NUMBER

1. **Enable 8082 in your ec2 security groups.**
2. **Also, same port number should be exposed in docker file.**

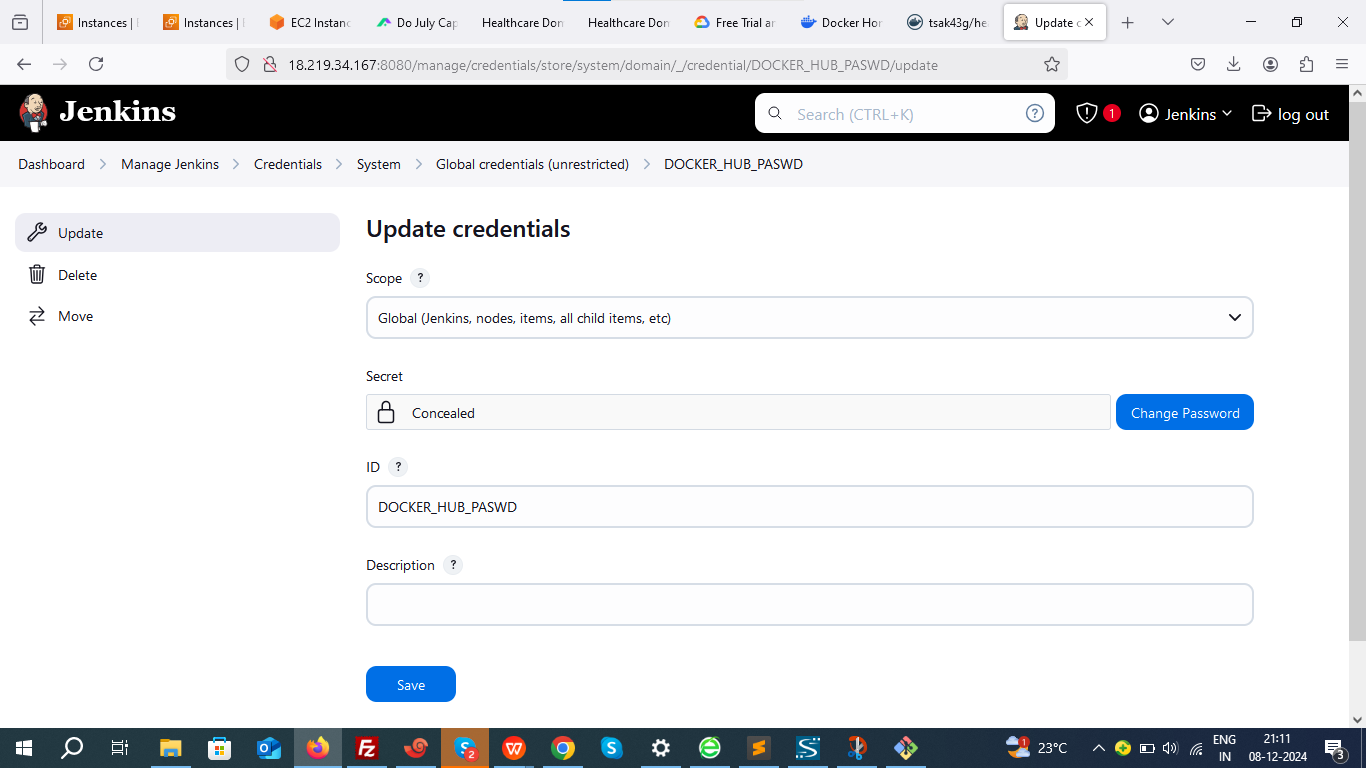
#### ****Check and Configure Docker Hub Credentials in Jenkins****

· Navigate to your Jenkins dashboard.

· Go to **Manage Jenkins** > **Manage Credentials**.

· Under the appropriate credentials scope (e.g., global or specific folder/job scope):

1. Click **Add Credentials**.
2. Choose **Username with password**.
3. Fill in the fields:
   1. **Username**: Your Docker Hub username.
   2. **Password**: Your Docker Hub password or access token.
   3. **ID**: Set it as DOCKER\_HUB\_PASWD (this must match the credentialsId in your pipeline script).
4. Save the credentials.



**6. Befor building the job allow Jenkins to Run Docker Commands**:

chmod -R 777 /var/run/docker.sock

7.Navigate to **New Item > Pipeline**.

pipeline{

agent any

tools{

maven 'mymaven'

}

stages{

stage('Clone Repo')

{

steps{

git 'https://github.com/CSPPallavi/star-agile-health-care.git'

}

}

stage('Test Code')

{

steps{

sh 'mvn test'

}

}

stage('Build Code')

{

steps{

sh 'mvn package'

}

}

stage('Build Image')

{

steps{

sh 'docker build -t health\_care:$BUILD\_NUMBER .'

}

}

stage('Push the Image to dockerhub')

{

steps{

withCredentials([string(credentialsId: 'DOCKER\_HUB\_PASWD', variable: 'DOCKER\_HUB\_PASWD')])

{

sh 'docker login -u tsak43g -p ${DOCKER\_HUB\_PASWD}

}

sh 'docker tag health\_care:$BUILD\_NUMBER tsak43g/health\_care:$BUILD\_NUMBER '

sh 'docker push tsak43g/health\_care:$BUILD\_NUMBER'

}

}

}

}

· **Clone Repo:** Clones the GitHub repository containing the project.

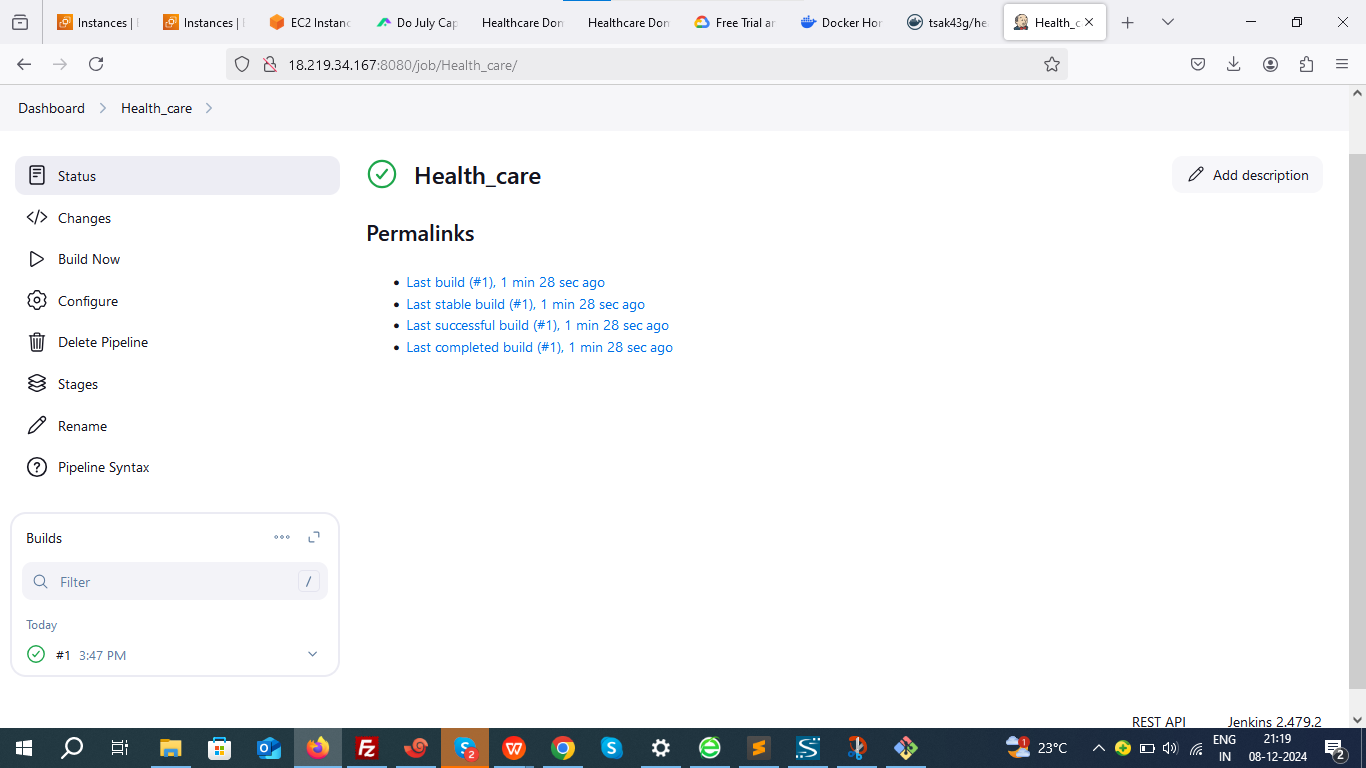
**· Test Code:** Runs tests on the project using Maven (mvn test).

**· Build Code:** Packages the project into a JAR file using Maven (mvn package).

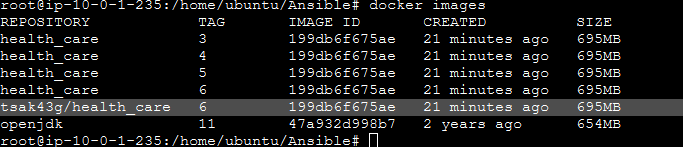
**· Build Image**: Builds a Docker image with the JAR file (docker build).

**· Push the Image to Dockerhub:** Logs into Docker Hub and pushes the newly built image to a Docker repository.

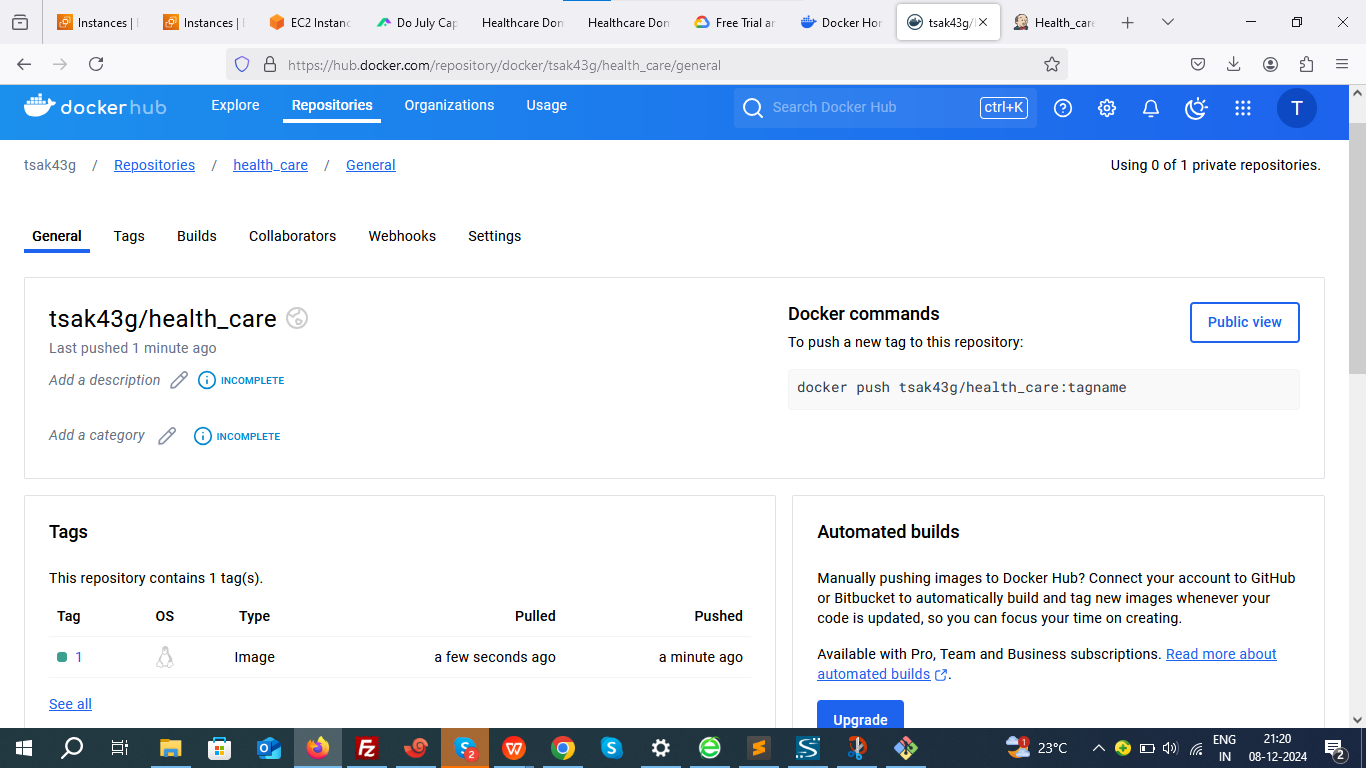
**Build the job successfully.**



**Verify docker image.**



Verify whether docker image is successfully pushed to your docker hub.



### ****Step 4: Installing and setup Kubernetes controller and worker.****

### ****Prerequisites:****

### ****Install docker and container-d****

### ****Install kubernetes component****

### ****Initiate kubernetes****

### ****Install Container network interface****

**SetUp kubernetes Master Node:**

Using AWS we will create Ubuntu master server of 4 GB RAM 2 CPU core → t2.medium

Install Containerd

sudo wget https://raw.githubusercontent.com/lerndevops/labs/master/scripts/installContainerd.sh -P /tmp

sudo bash /tmp/installContainerd.sh

sudo systemctl restart containerd.service

Install kubeadm,kubelet,kubectl

You will install these packages on all of your machines:

kubeadm: the command to bootstrap the cluster.

kubelet: the component that runs on all of the machines in your cluster and does things like starting pods and containers.

kubectl: the command line util to talk to your cluster.

sudo wget https://raw.githubusercontent.com/lerndevops/labs/master/scripts/installK8S.sh -P /tmp

sudo bash /tmp/installK8S.sh

Initialize kubernetes Master Node

sudo kubeadm init --ignore-preflight-errors=all

Execute the below commands to setup kubectl and apiserver communication

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

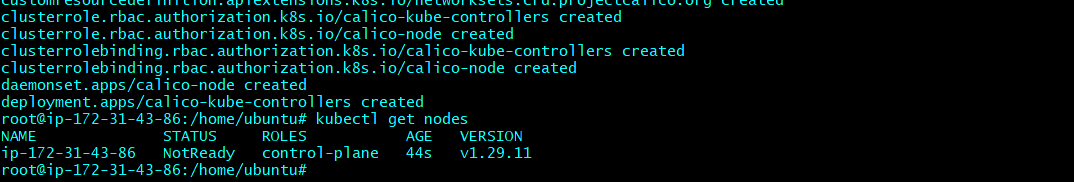
sudo chown $(id -u):$(id -g) $HOME/.kube/config

install networking driver -- Weave/flannel/canal/calico etc...

below installs calico networking driver

kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.24.1/manifests/calico.yaml

# Validate: kubectl get nodes



The node status is **NotReady which means** control plane is not yet ready.

**SetUp kubernetes worker Node:**

Create an Ubuntu server: instance type t2.medium(AWS), e2 medium(GCP)

OS : ubuntu 22

Install Containerd

sudo su -

sudo wget https://raw.githubusercontent.com/lerndevops/labs/master/scripts/installContainerd.sh -P /tmp

sudo bash /tmp/installContainerd.sh

sudo systemctl restart containerd.service

Install kubeadm,kubelet,kubectl

sudo wget https://raw.githubusercontent.com/lerndevops/labs/master/scripts/installK8S.sh -P /tmp

sudo bash /tmp/installK8S.sh

**Run Below on Master Node to get join token**

kubeadm token create --print-join-command

copy the kubeadm join token from master & run it on all worker nodes

### **We will use a tool called kubeadm to automatically set up kubernetes components for us.**

**Now recheck the node. Control plane should be ready.**

kubectl get nodes

### 

### ****Allow all the ports in security group on your master and worker nodes.****

Steps to Deploy a Docker Image in Kubernetes:

#### 1. ****Create a Deployment YAML File****

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-app-deployment

labels:

app: my-app

spec:

replicas: 2 # Number of pods to run

selector:

matchLabels:

app: my-app

template:

metadata:

labels:

app: my-app

spec:

containers:

- name: health-care

image: tsak43g/health\_care:1 # Replace with your image name and tag

ports:

- containerPort: 8082 # Replace with the port your app listens on

#### 2. ****Apply the Deployment****

kubectl apply -f deployment.yaml

#### ****3. Create the NodePort Service****

Create another YAML file named service.yaml with the following content:

apiVersion: v1

kind: Service

metadata:

name: my-app-service

spec:

selector:

app: my-app

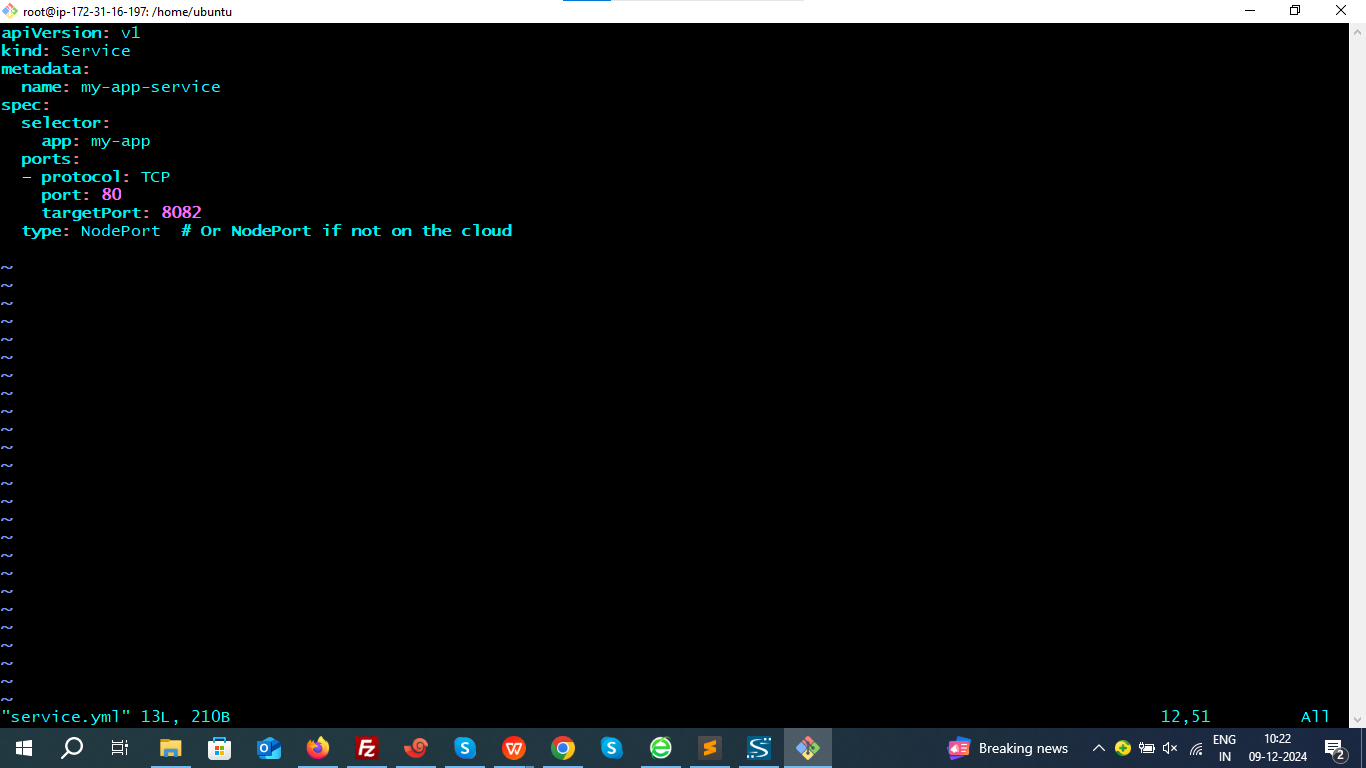
ports:

- protocol: TCP

port: 80

targetPort: 8082

type: NodePort



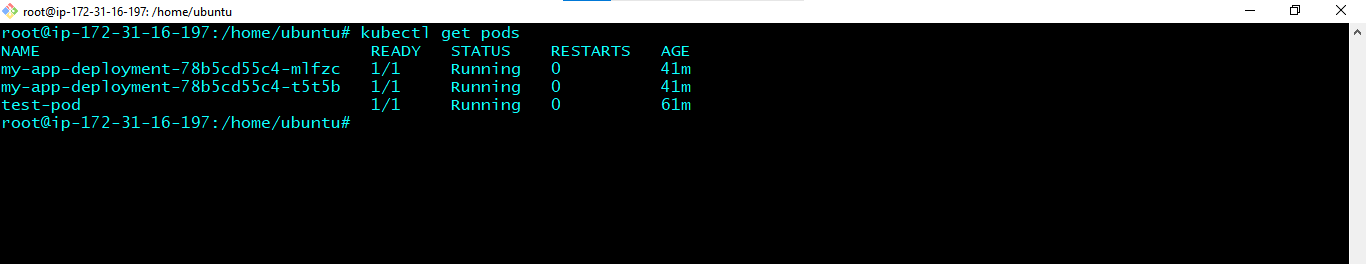
**Apply the service:**

kubectl apply -f service.yaml

5.****Verify the Deployment and Service****

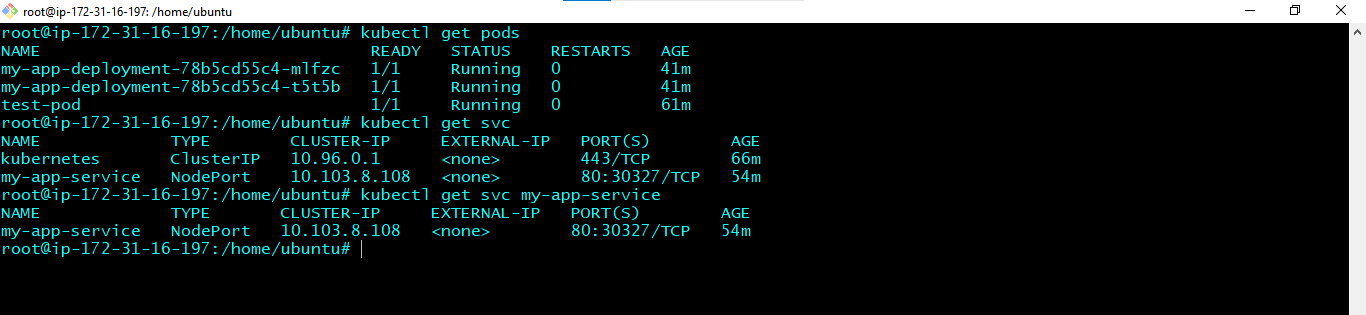
**Check the Pods:**

kubectl get pods



**Check the Service:**

kubectl get service my-app-service

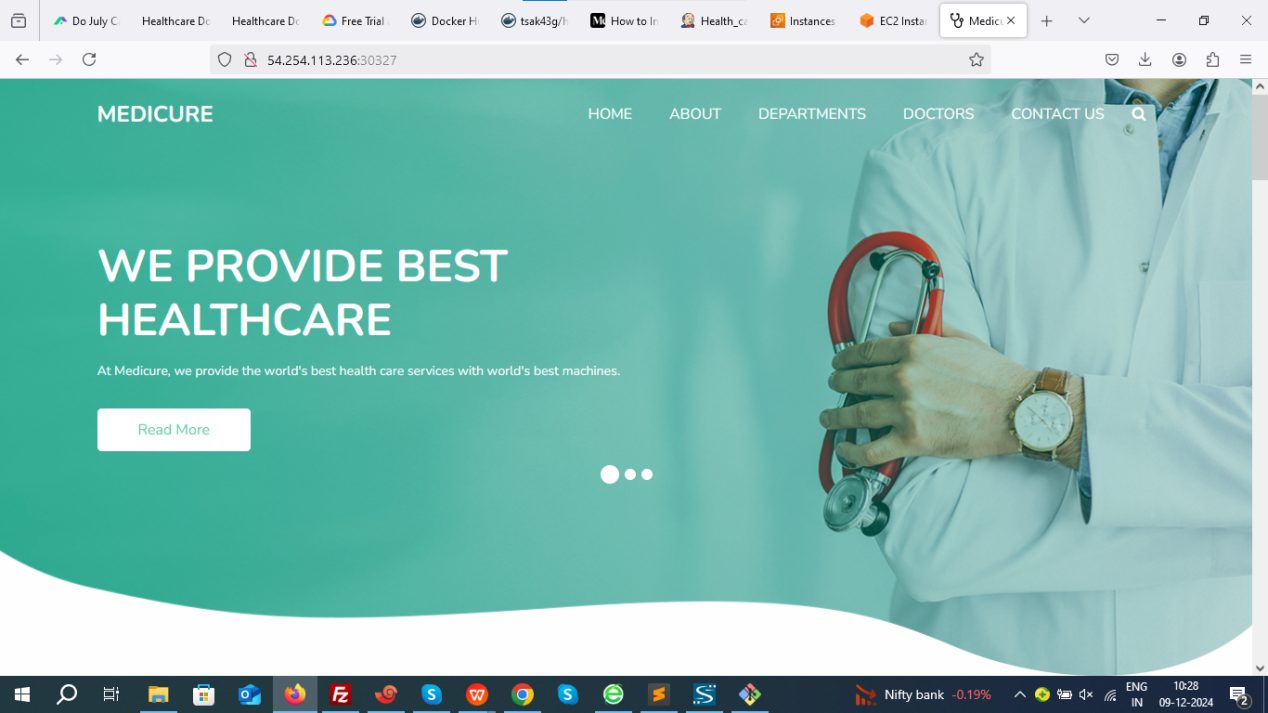


Verify that the service has a auto assigned NodePort (30996)

**Access the container.**

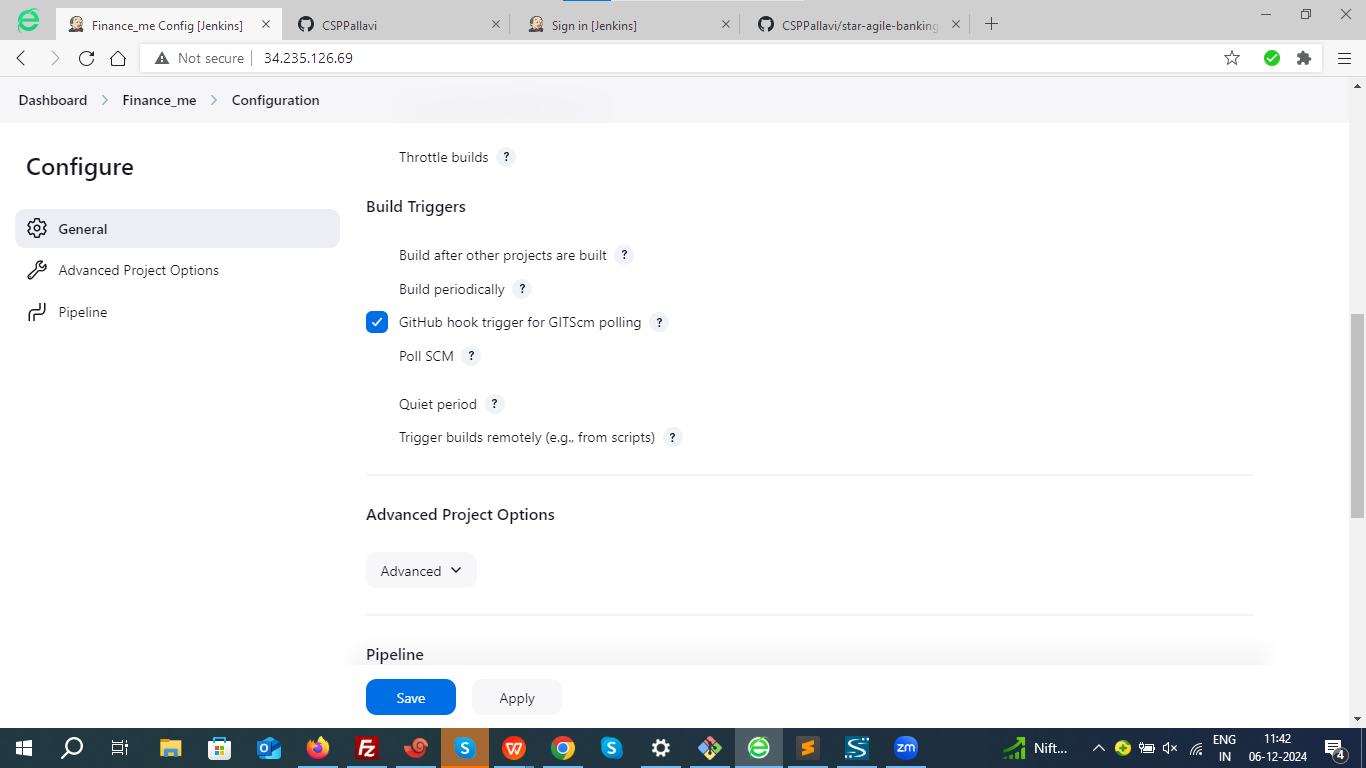
**[http://< public](http://<public) ip of worker node>:30996**

**<http://13.126.231.60:30996/>**



**STEP 4: WEBHOOK CONFIGURATION:**

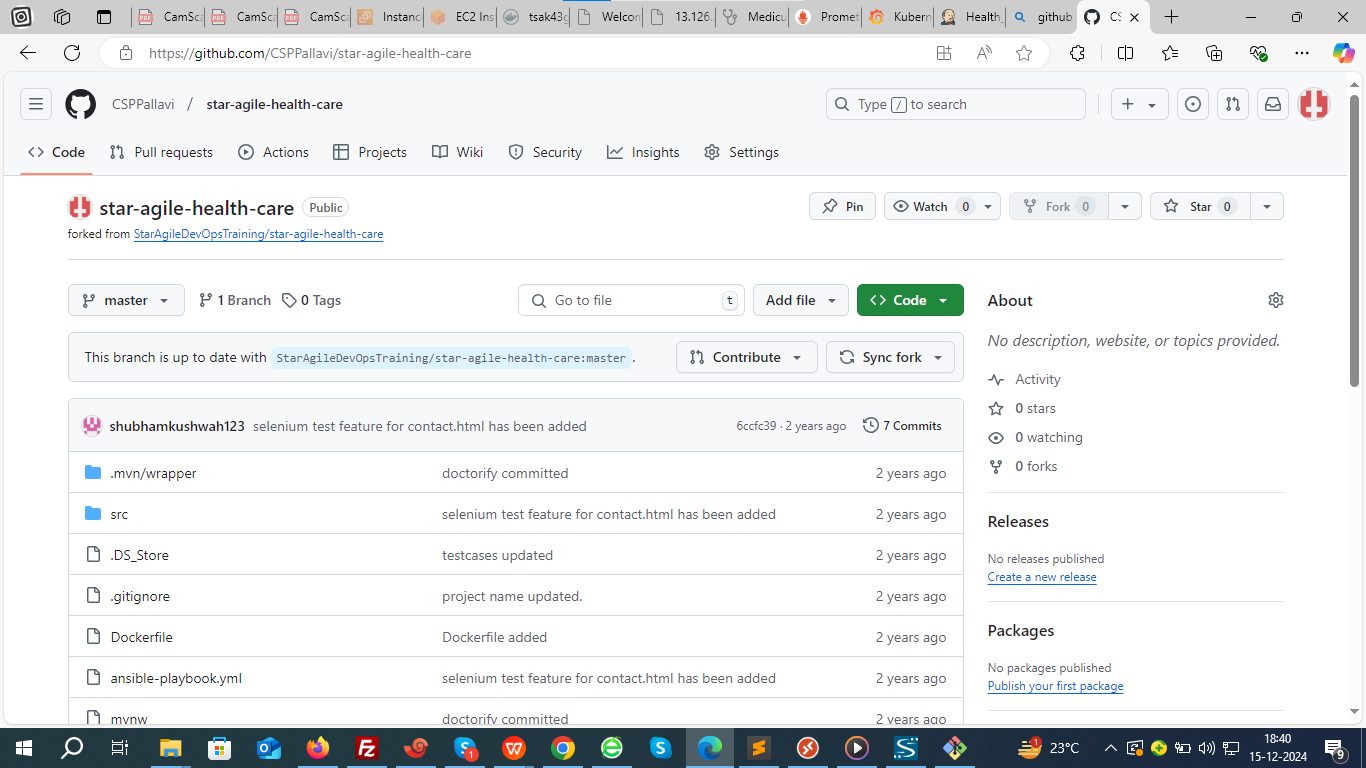
Go to your Jenkins dashboard and add build trigger- **GitHub hook trigger for GITScm polling**



Configuring a webhook in Git is a crucial step to enable automated actions, such as triggering a CI/CD pipeline in Jenkins or notifying an application about repository events.

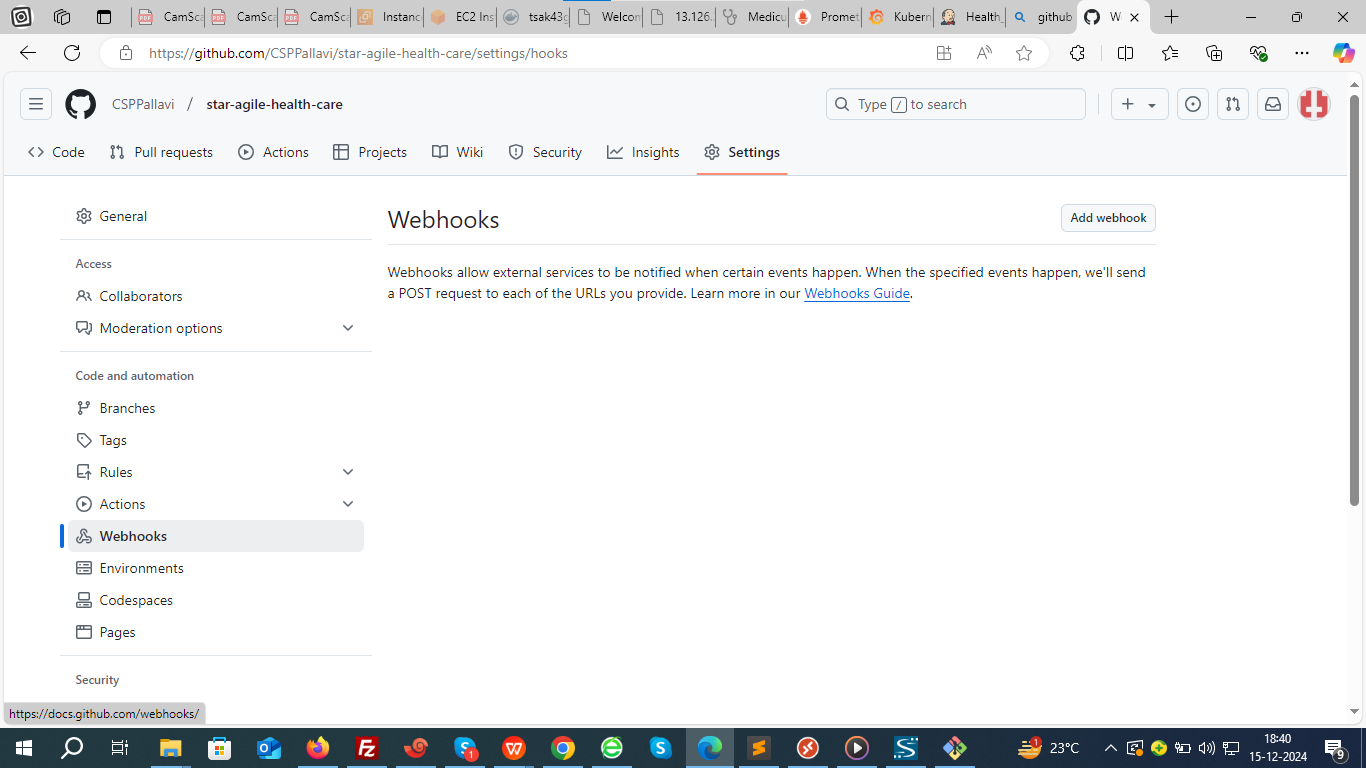
### ****Step 1: Open Your Repository Settings****

1. Navigate to the repository where you want to configure the webhook.



### ****Step 2: Find the Webhooks Section****

Go to **Repository Settings** > **Webhooks**.



### ****Step 3: Add a New Webhook****

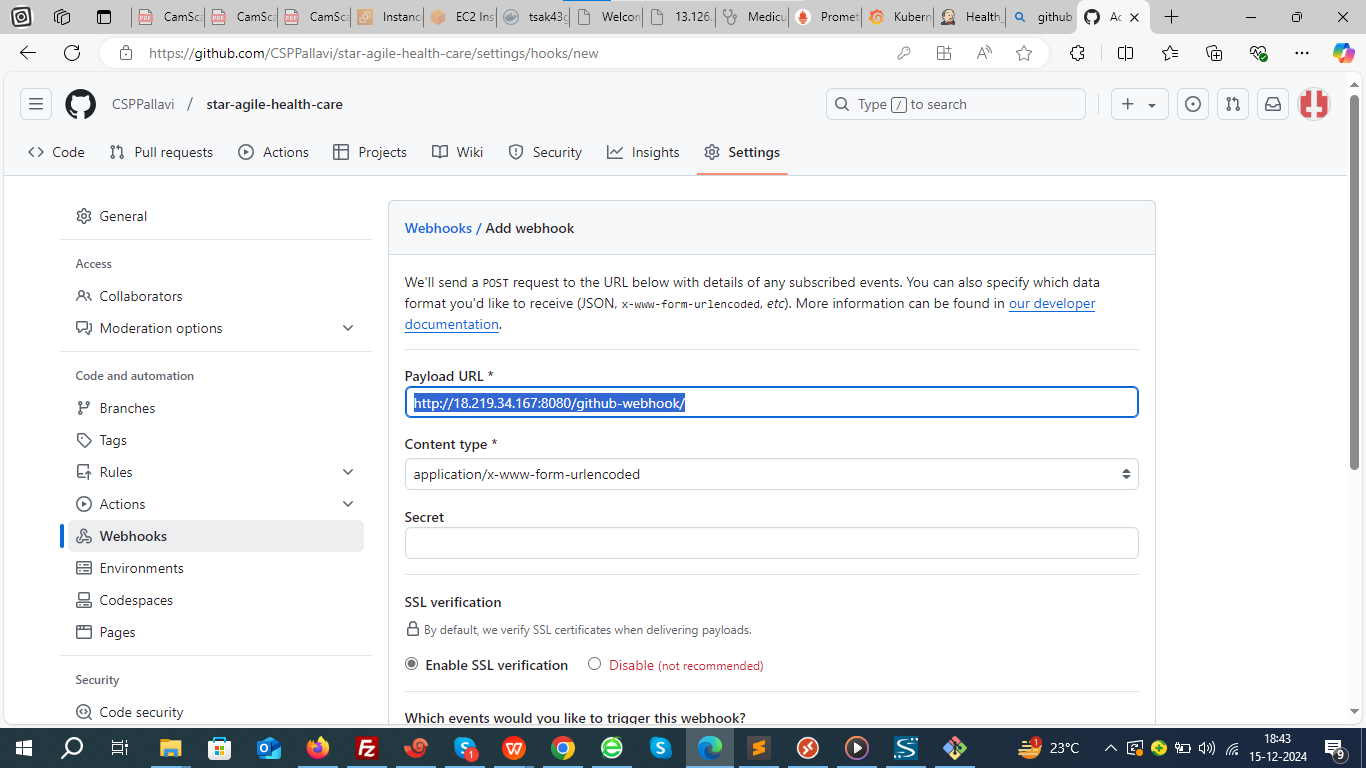
Fill in the webhook details:

**Payload URL**:

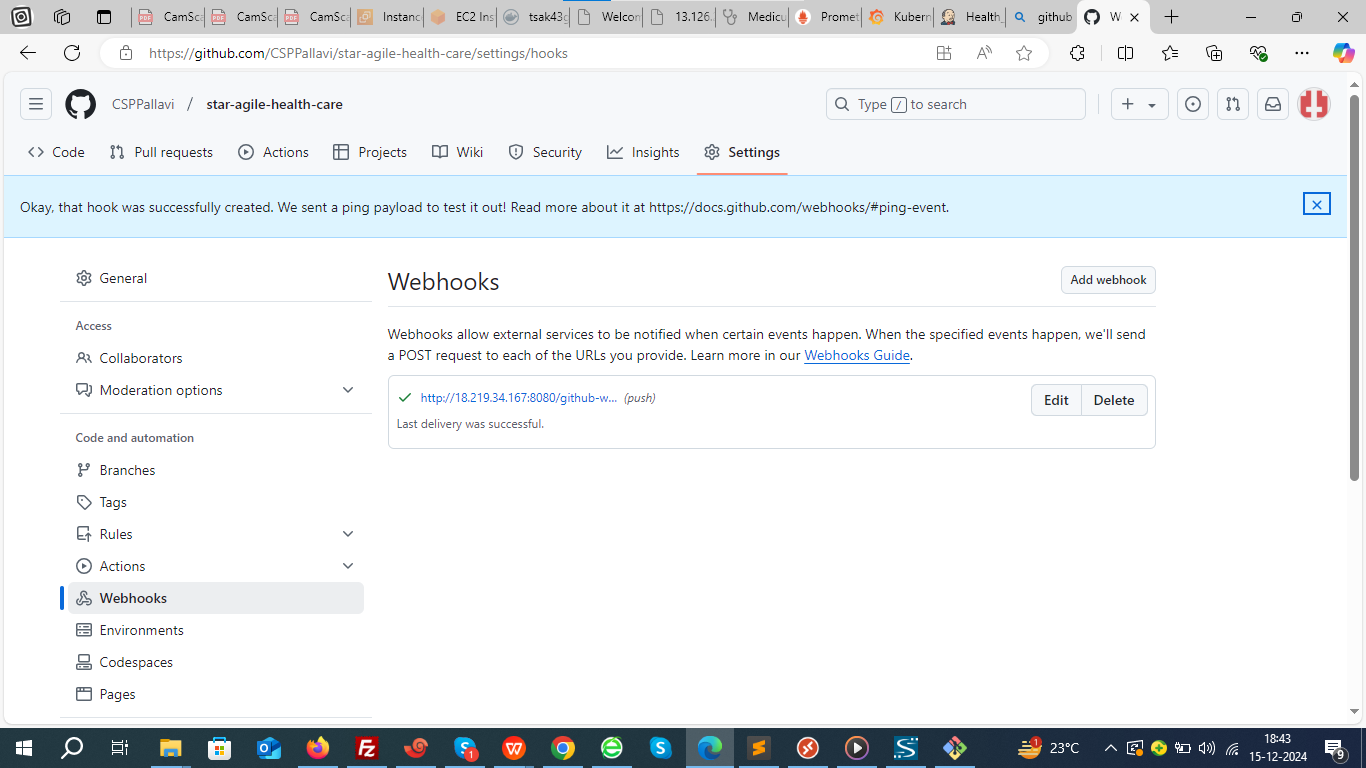
· This is the endpoint that will receive the webhook payload.

· <http://<Jenkins-URL>/github-webhook/>

<http://18.219.34.167:8080/github-webhook/>



### ****Step 4 : Commit changes and save.****



### ****Step 5: Setup Monitoring with Prometheus and Grafana in K8S controller****

Monitor your Kubernetes (K8s) cluster with **Prometheus** and **Grafana**

### Steps to set up ****Prometheus**** and ****Grafana****:

#### 1. ****Install Prometheus****

Prometheus is an open-source monitoring and alerting toolkit used to collect and store metrics in Kubernetes.

You can install Prometheus using **Helm** (a package manager for Kubernetes) for easy installation.

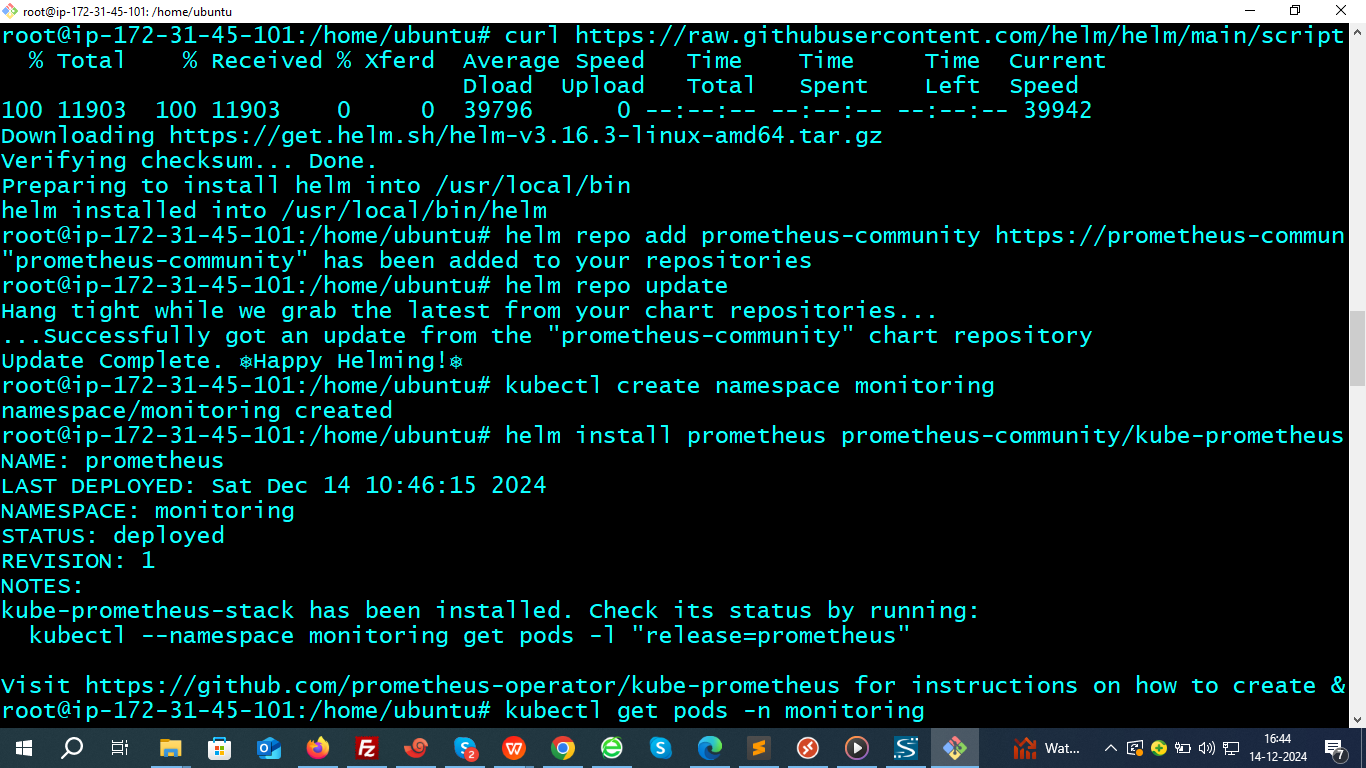
##### Install Helm (if not already installed):

curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

##### Add Prometheus Helm chart repository:

helm repo add prometheus-community <https://prometheus-community.github.io/helm-charts>

helm repo update



##### Install Prometheus and grafana using Helm:

kubectl create namespace monitoring

# Create monitoring namespace

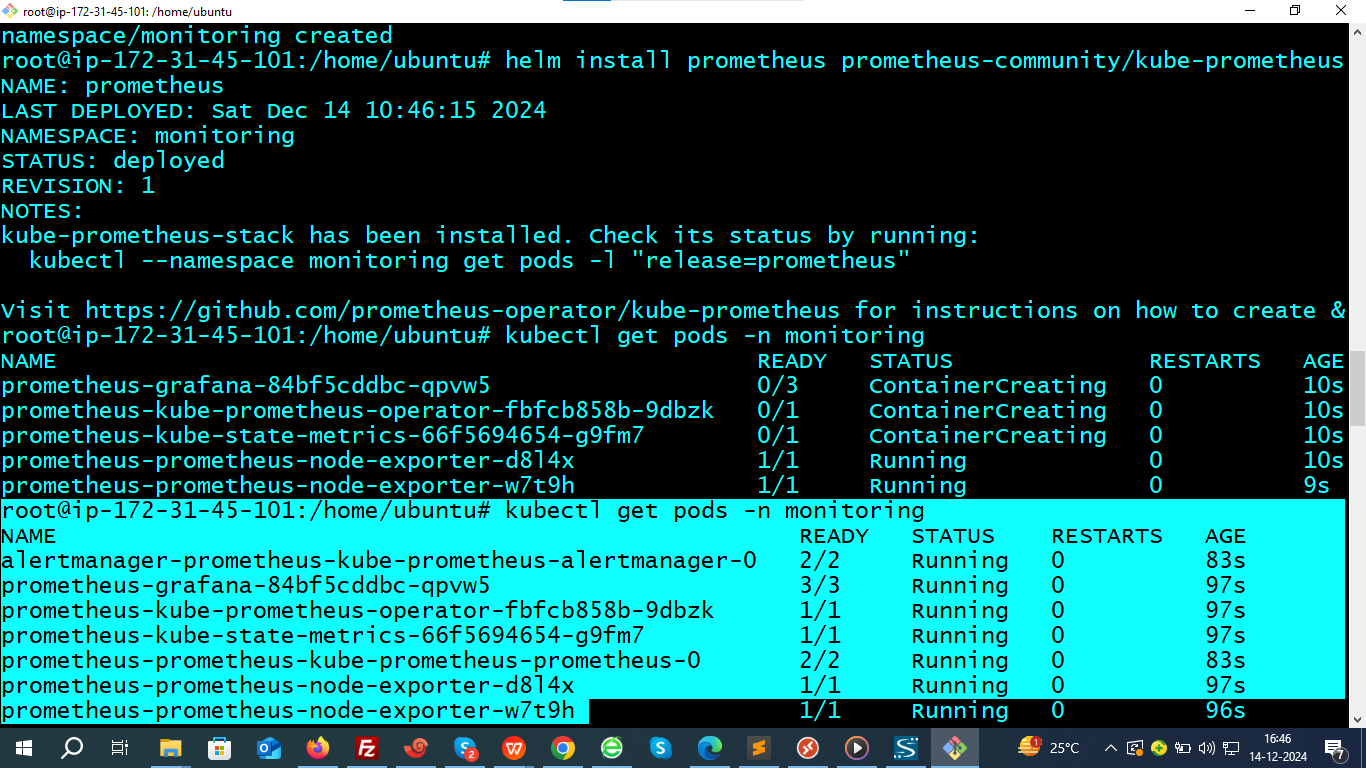
helm install prometheus prometheus-community/kube-prometheus-stack --namespace monitoring

This command installs both **Prometheus** and **Grafana** along with other monitoring components like **node-exporter**, **kube-state-metrics**, and **Alertmanager**.

#### 2. ****Verify Prometheus Installation****

After the installation completes, you can check if the Prometheus components are running by checking the pods in the **monitoring** namespace:

kubectl get pods -n monitoring



You should see pods like prometheus-xxxx, grafana-xxxx, and other components related to monitoring.

#### 3. ****Access Prometheus Dashboard****

You can access Prometheus through a port-forwarding command (for testing):

kubectl port-forward svc/prometheus-kube-prometheus-prometheus -n monitoring 9090:9090

This command creates service type cluster IP

Below is the command to swich it to node port type.

kubectl patch svc prometheus-kube-prometheus-prometheus -n monitoring -p '{"spec": {"type": "NodePort"}}'

Check NodePort:



Access prometheus dashboard.

<http://13.126.231.60:31978/>



#### 4. ****Access Grafana Dashboard****

Grafana will be installed alongside Prometheus. To access it, you can port-forward it similarly:

kubectl port-forward svc/prometheus-grafana -n monitoring 3000:80

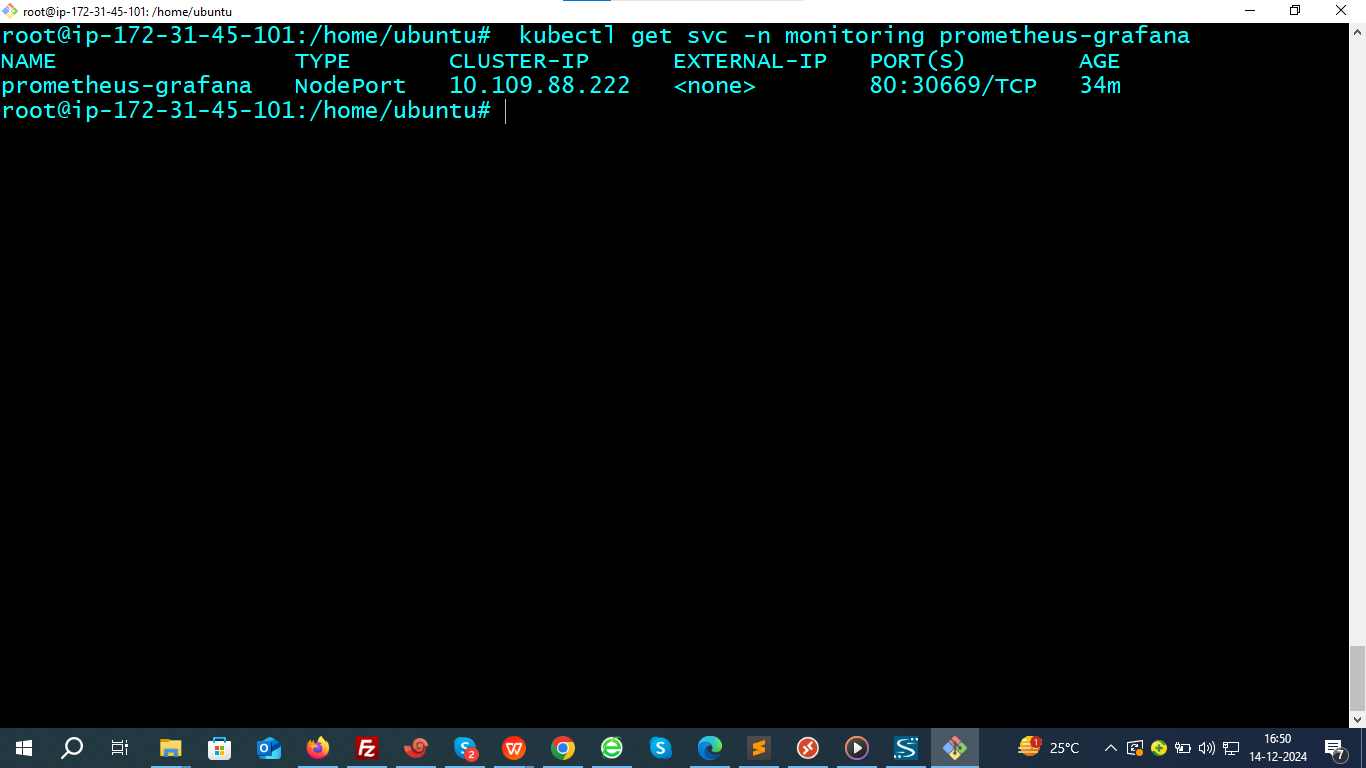
This command again creates service type cluster IP

Below is the command to swich it to node port type.

kubectl patch svc prometheus-grafana -n monitoring -p '{"spec": {"type": "NodePort"}}'}'

Check node port:

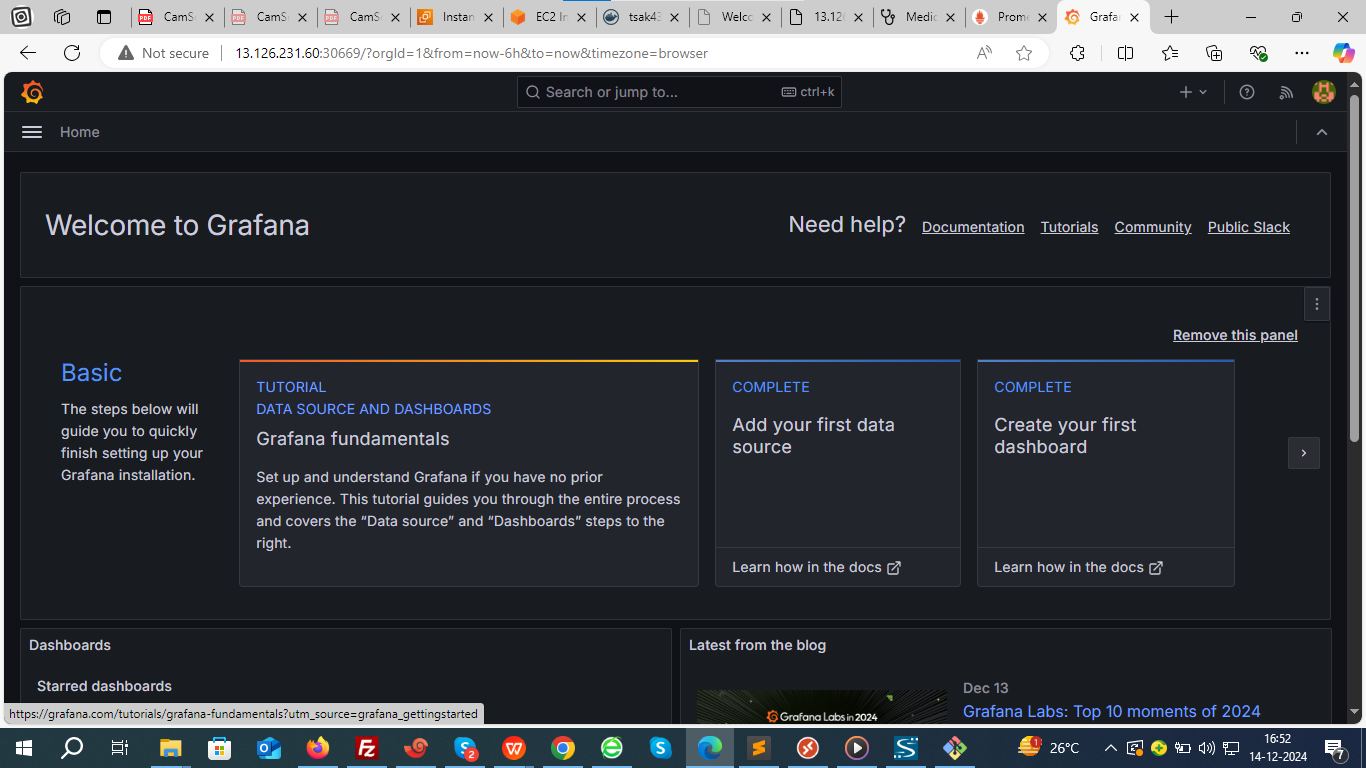
kubectl get svc -n monitoring prometheus-grafana



<http://13.126.231.60:30669/>

The default login credentials are:

* **Username**: admin
* **Password**: prom-operator



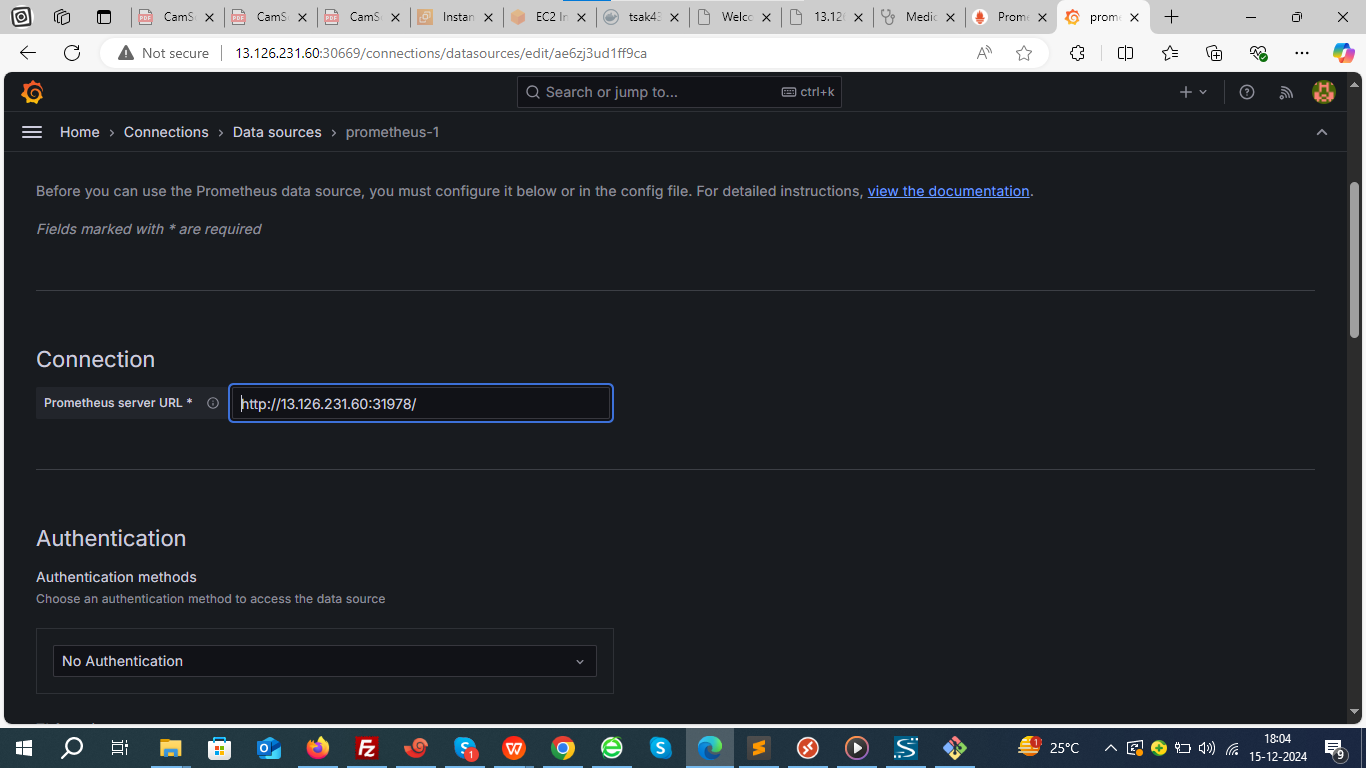
#### 5. ****Configure Grafana with Prometheus Data Source****

Once you're inside Grafana, follow these steps to connect it to Prometheus as a data source:

* Go to **Configuration** (gear icon) > **Data Sources**.

Select **Prometheus** and set the URL to <http://13.126.231.60:31978/>

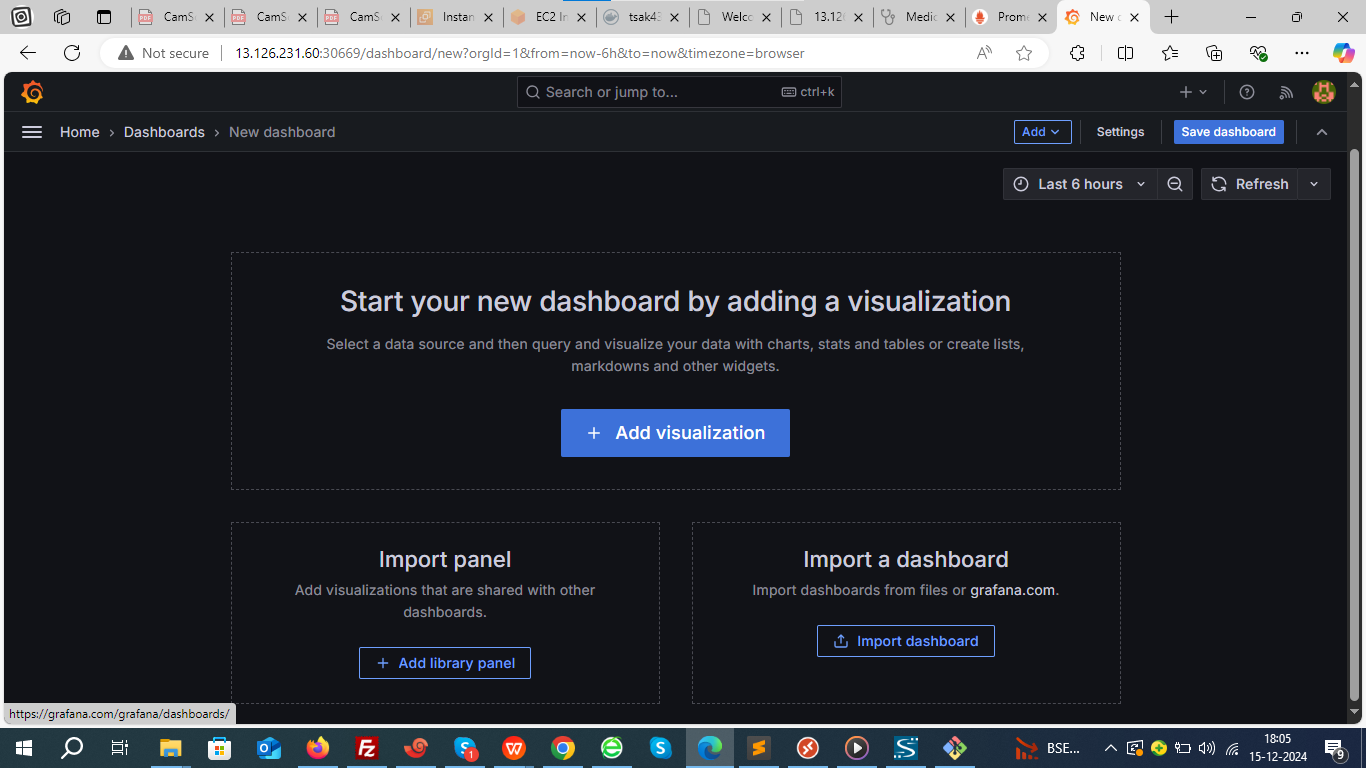
Click **Save & Test** to ensure the connection is successful.



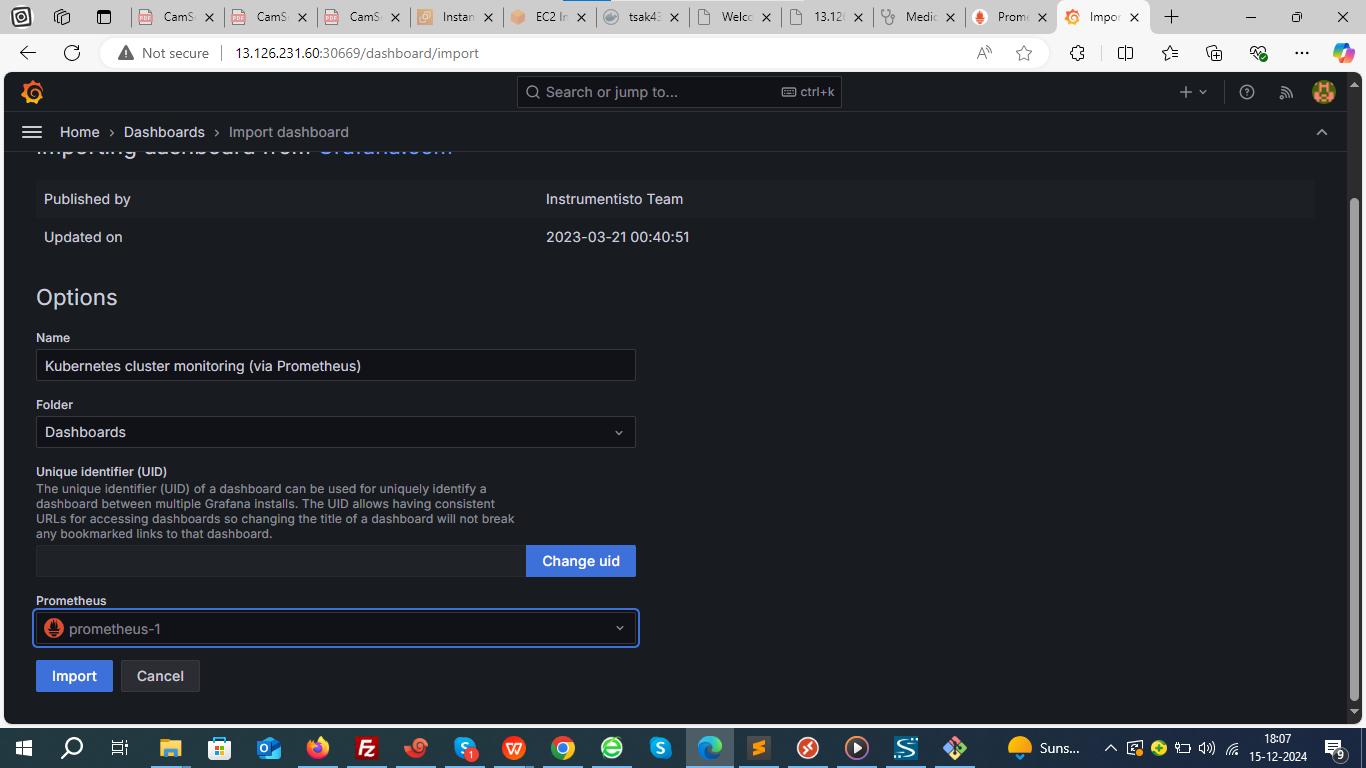
#### 6. ****Import Grafana Dashboards****

Grafana has pre-built dashboards for Kubernetes monitoring that you can import.

* Go to **Create** (plus icon) > **Import**.



* Enter **kubernetes monitoring dashboards** or use specific dashboard IDs like 315 for Kubernetes or 6417 for a more specific one from the Grafana Dashboard repository.
* Select **Load** and then select database as promrtheus and import



Now, you'll have an interactive dashboard displaying Kubernetes metrics, including pod status, node health, CPU usage, memory usage, and more.

