Health Care MEDICURE

Submission by : Pallavi
Submitted to : Vikul
Date of submission: 17-12-24

Healthcare Domain-MEDICURE

The project involves creating a fully automated pipeline with Terraform, Jenkins, Docker, Kubernetes, Ansible, Prometheus, and Grafana for monitoring.

✓ *Git - For version control for tracking changes in the code files*
✓ *Jenkins - For continuous integration and continuous deployment*
✓ *Docker - For containerizing applications*
✓ *Ansible - Configuration management tools*
✓ *Terraform - For creation of infrastructure.*
✓ *Kubernetes – for running containerized application in managed cluster*

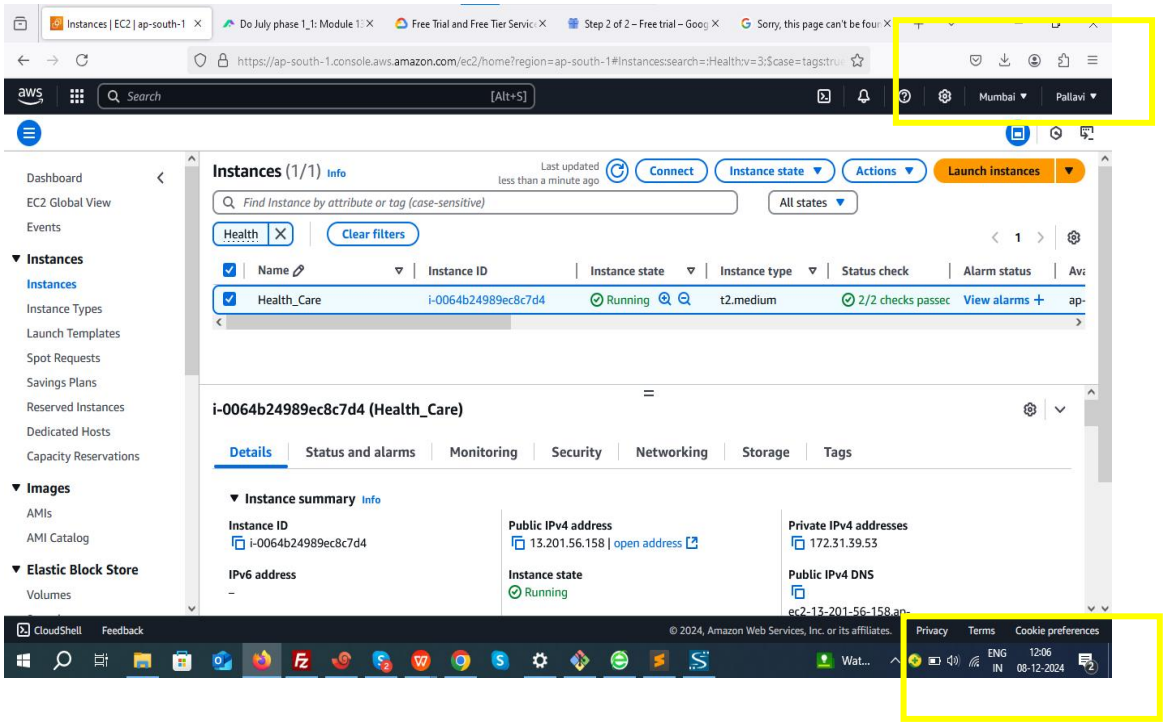
Medicure project code is attached below

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

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)**.

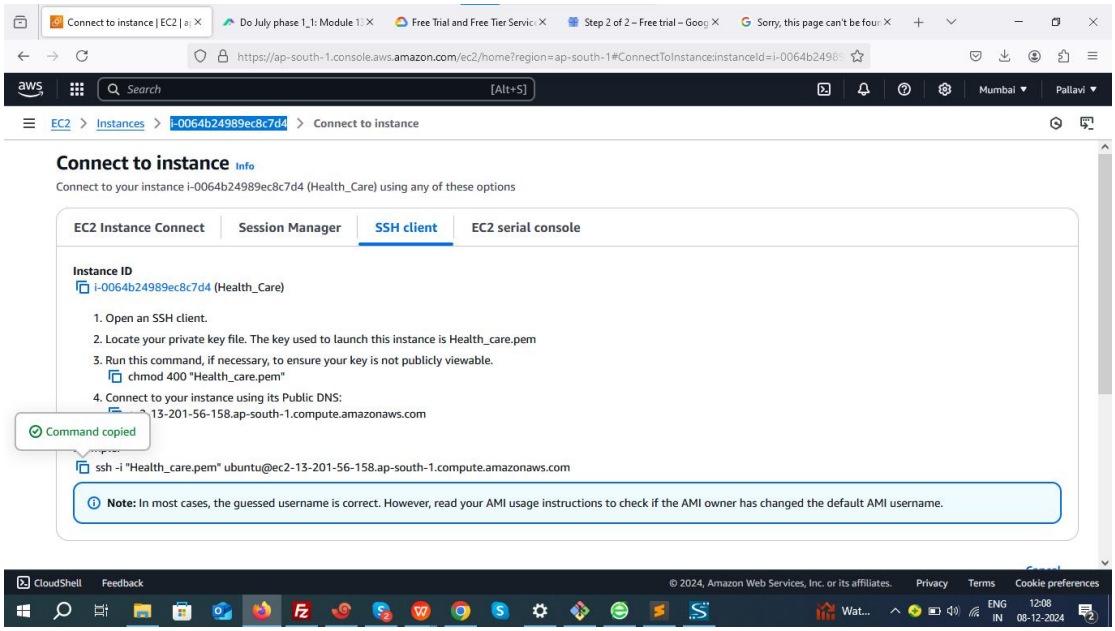


2. Connect to the VM:

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



3. Install Terraform:

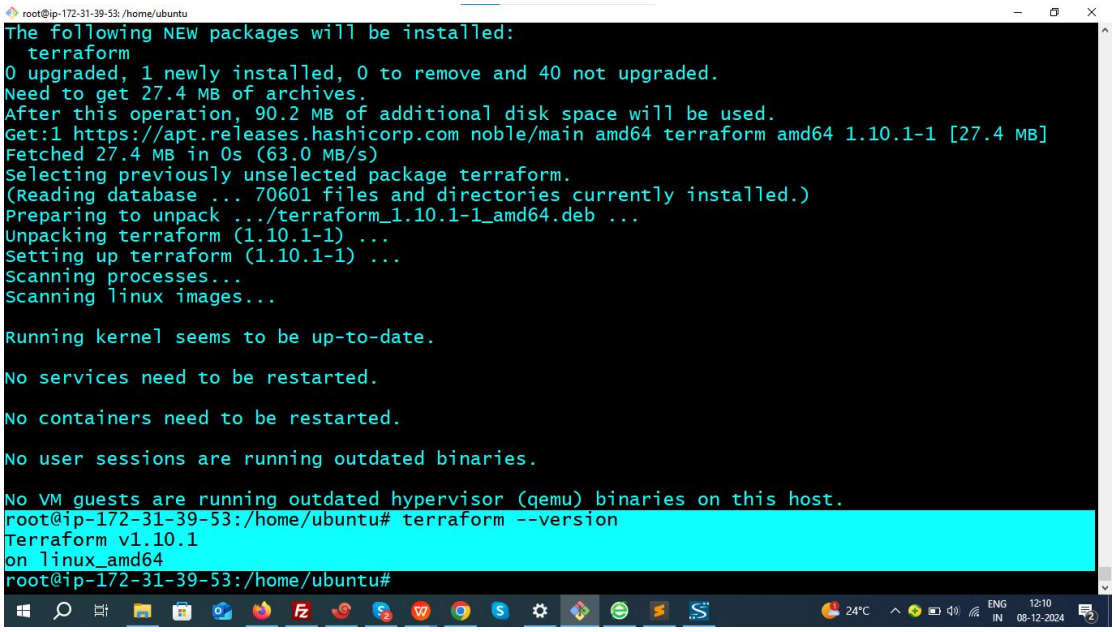
wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

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



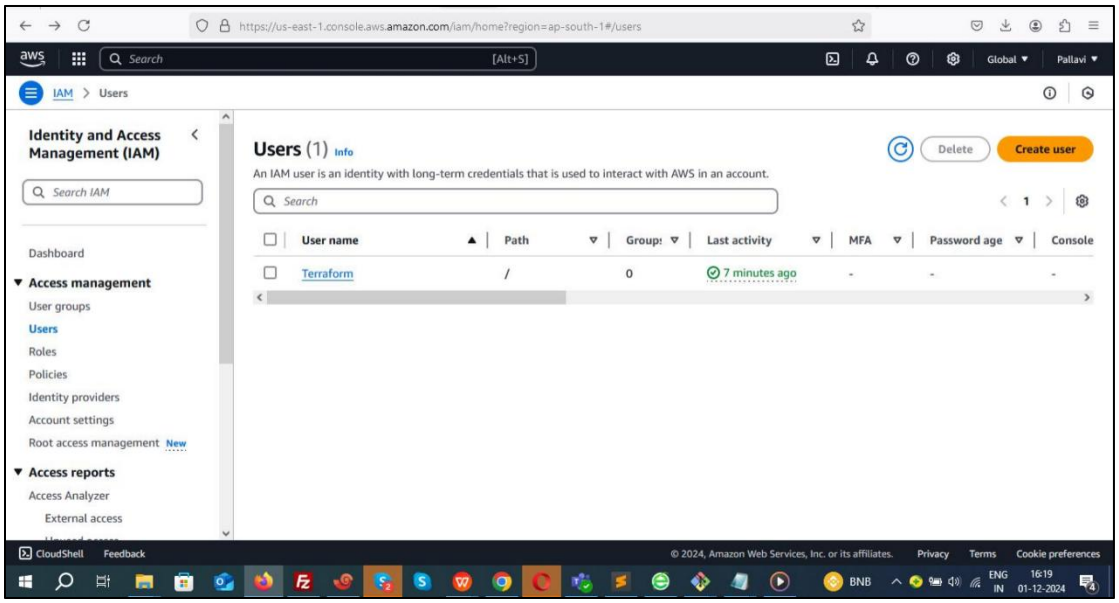
4.Establishing a connection between Terraform and AWS.

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.

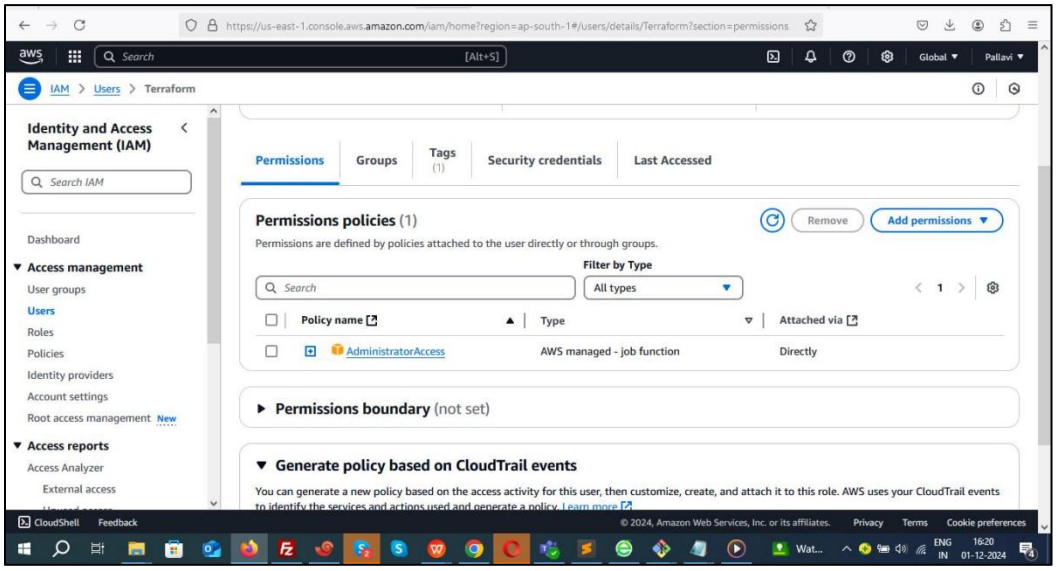
1. **Create an AWS IAM User with Administrator Access**

Login to AWS Management Console.
Navigate to IAM > Users.
Create a New User: Enable **Programmatic Access**.



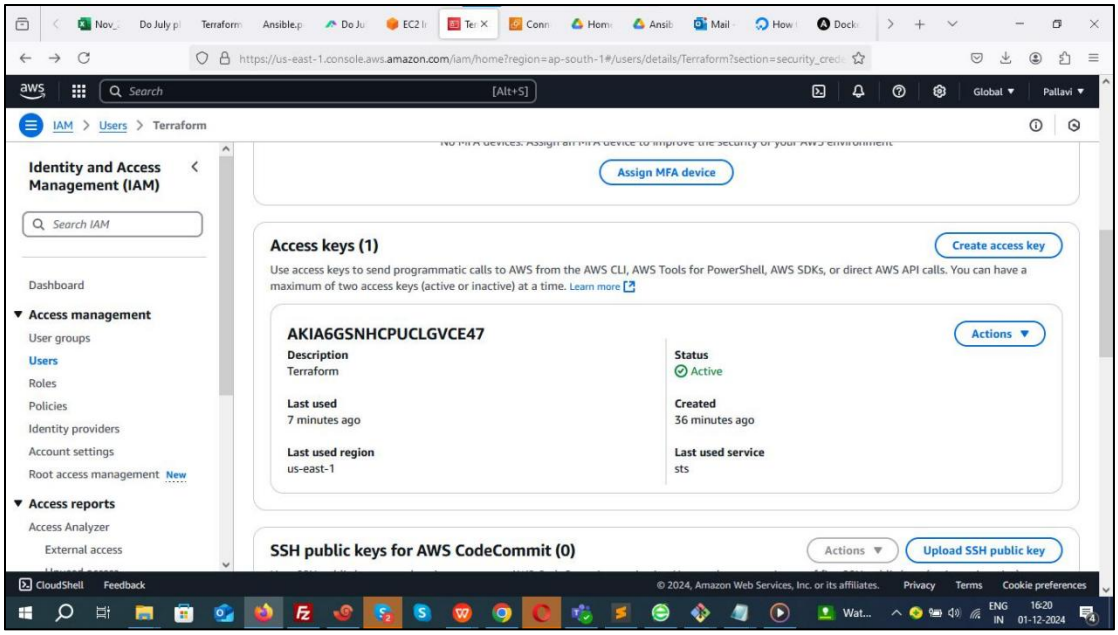
Attach a Policy:

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



Generate Access Keys:

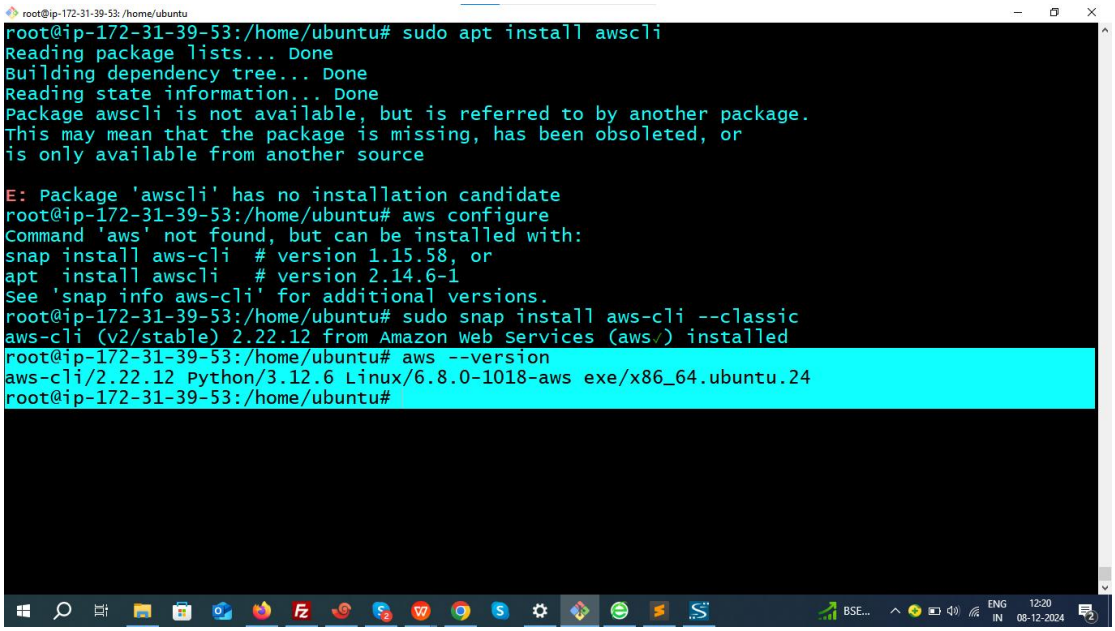
Save the **Access Key ID** and **Secret Access Key** securely (you will need these).



1. **Configure AWS CLI**

Install AWS CLI (if not installed):

sudo apt install awscli

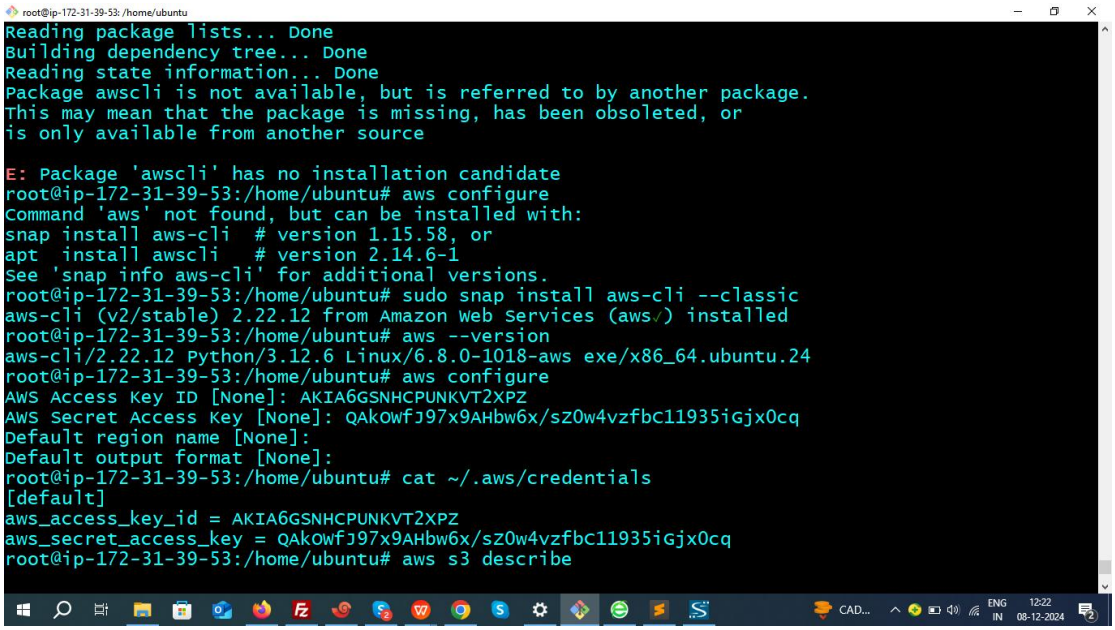


aws configure

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)
```

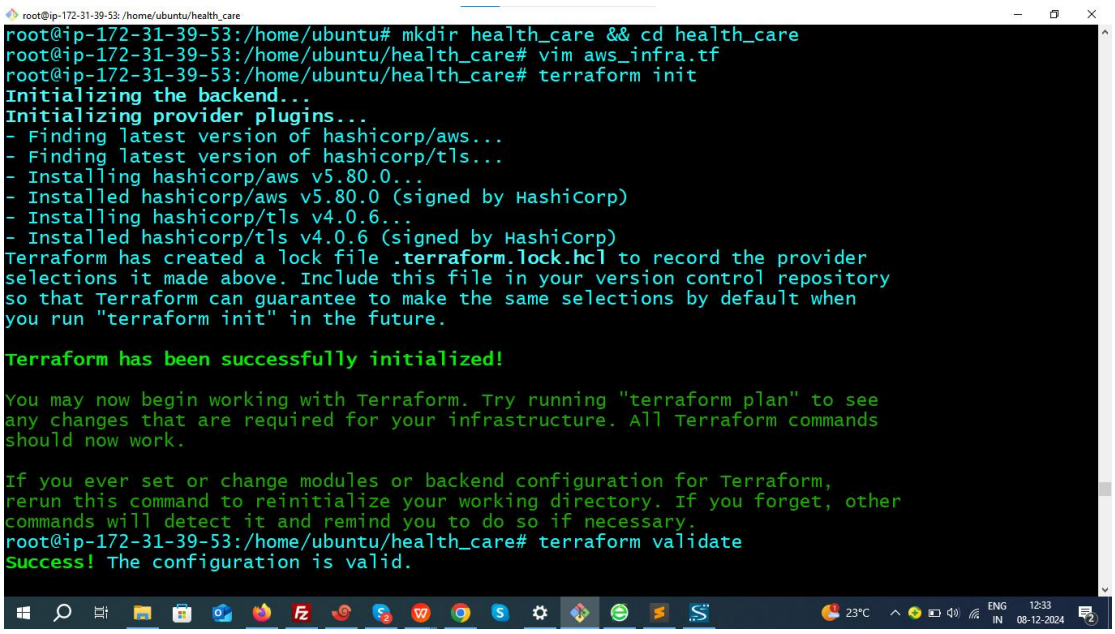
```
  default = [8080, 80, 22, 443]
}

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"
          ]
        }
      }
```

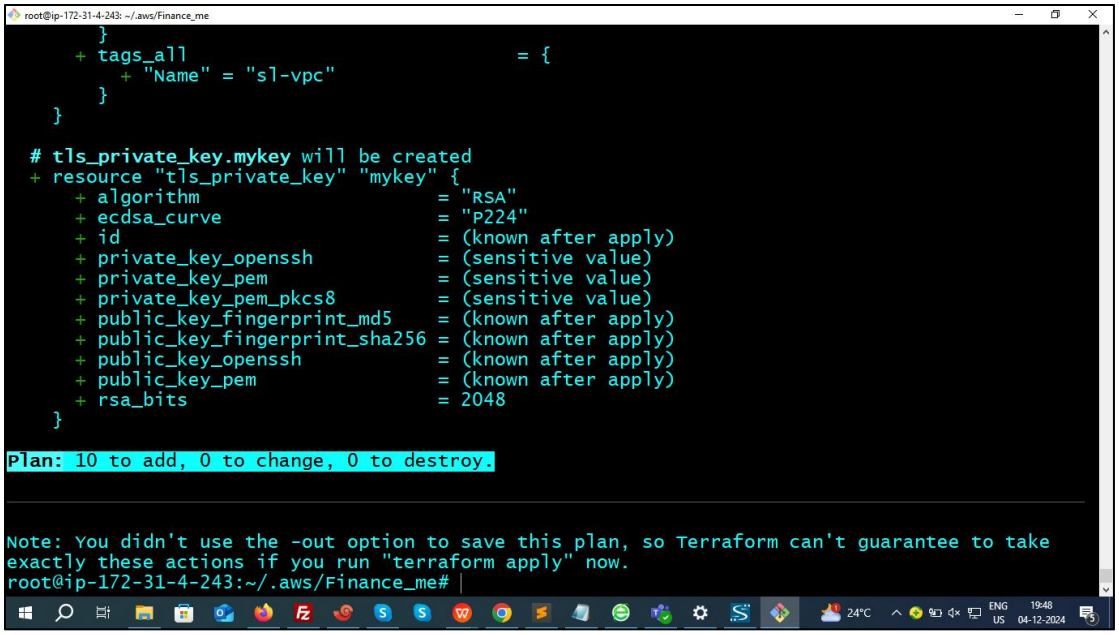
Initialize and Apply Terraform:

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

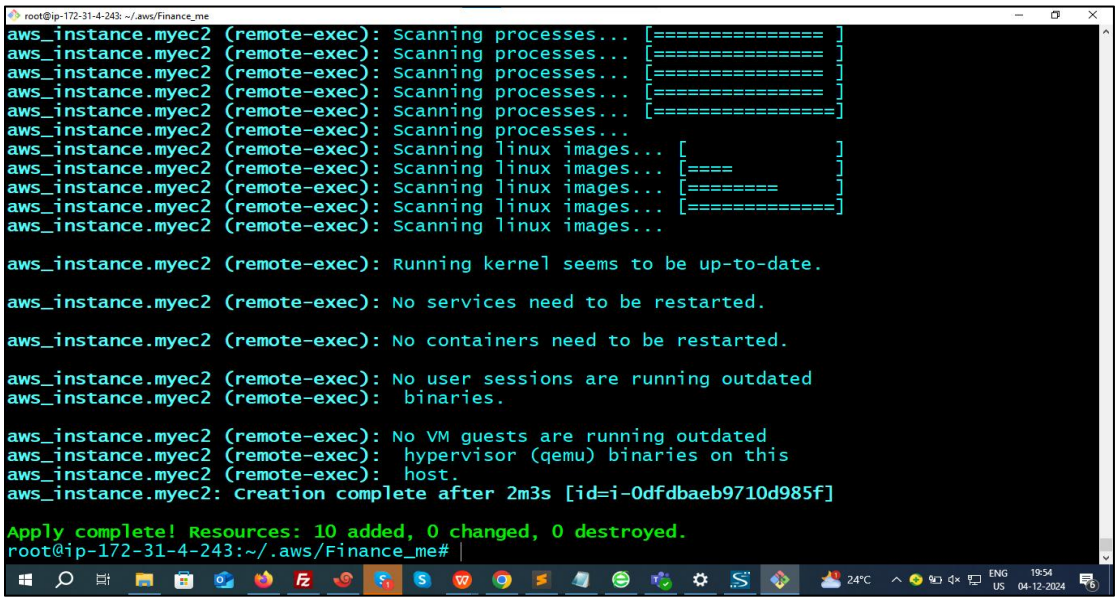


terraform validate - Checks for syntax errors and configuration validity.

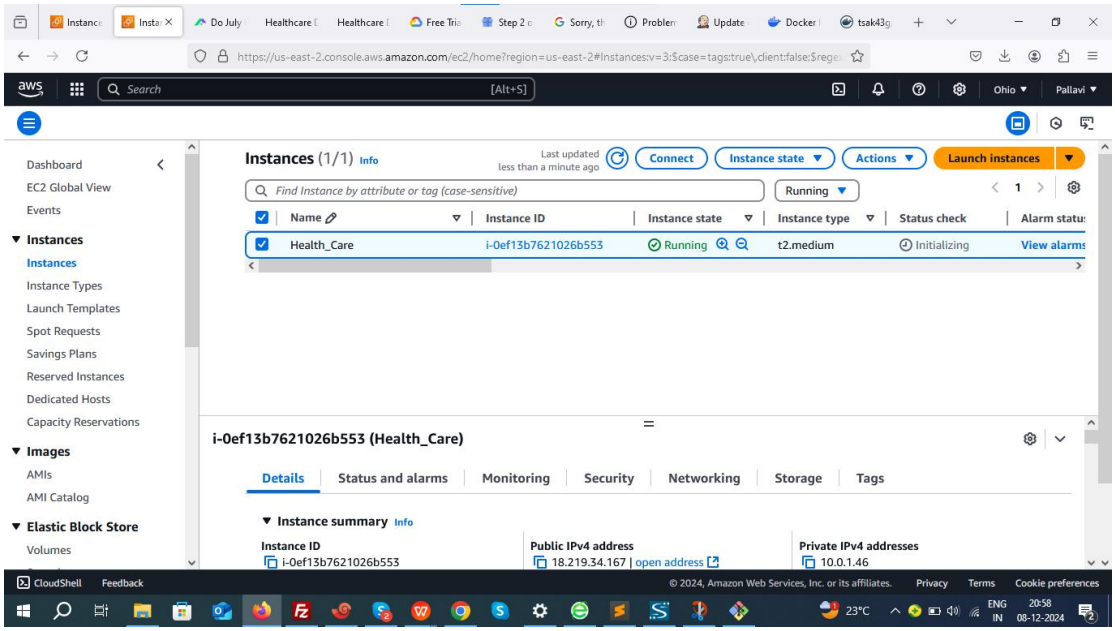
terraform plan - Generates and displays an execution plan without making changes.



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



Instance has been created in us-east-2



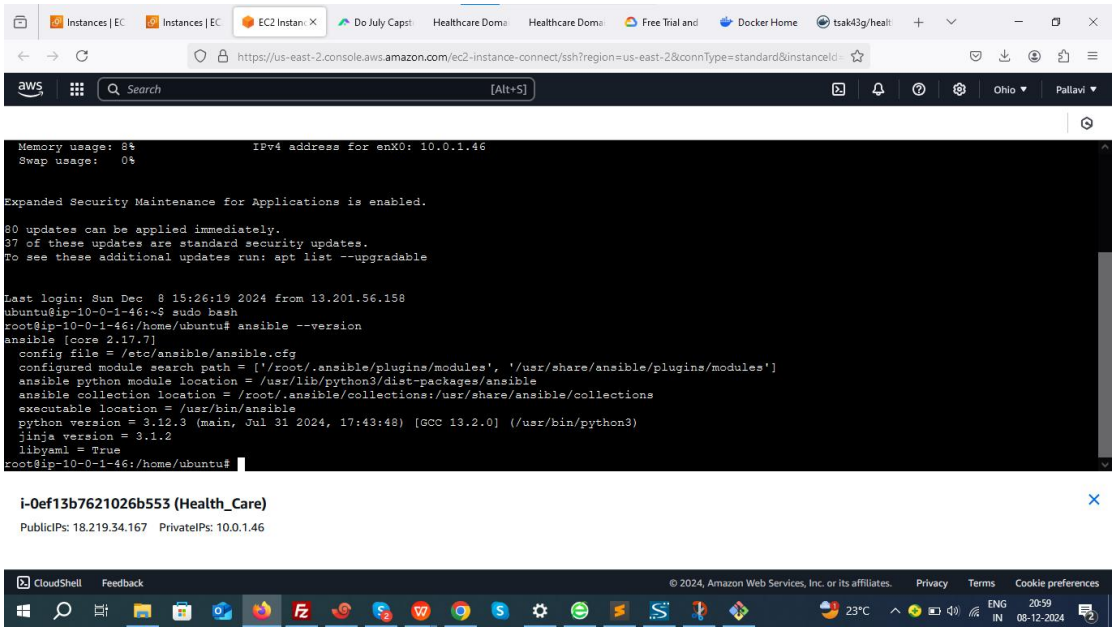
Step 2: Configure Ansible on the Created EC2 Instance

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

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

2. Check Ansible Installation

ansible --version



3. **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

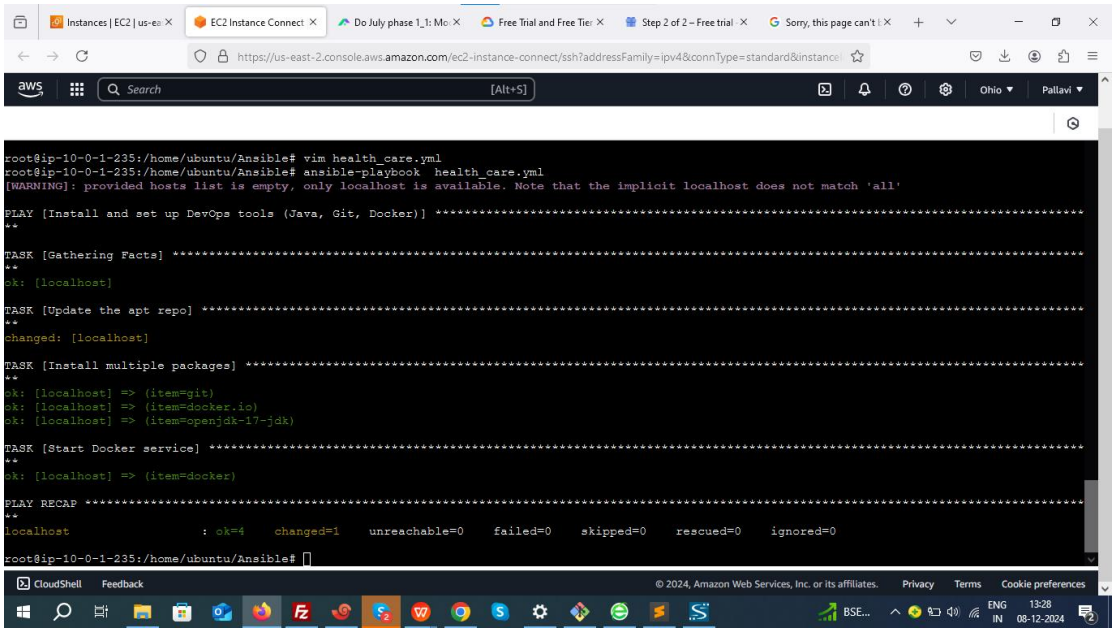
Add the following:

```
- 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
```

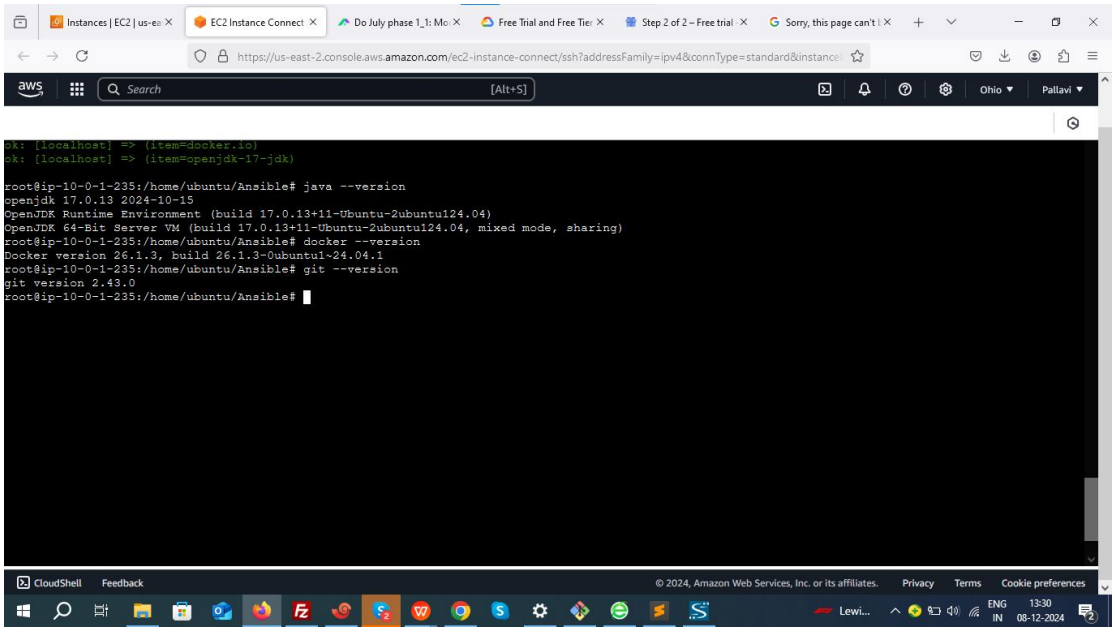
```
- name: Start Docker service
  service:
    name: "{{ item }}"
    state: started
  loop:
    - docker
```

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

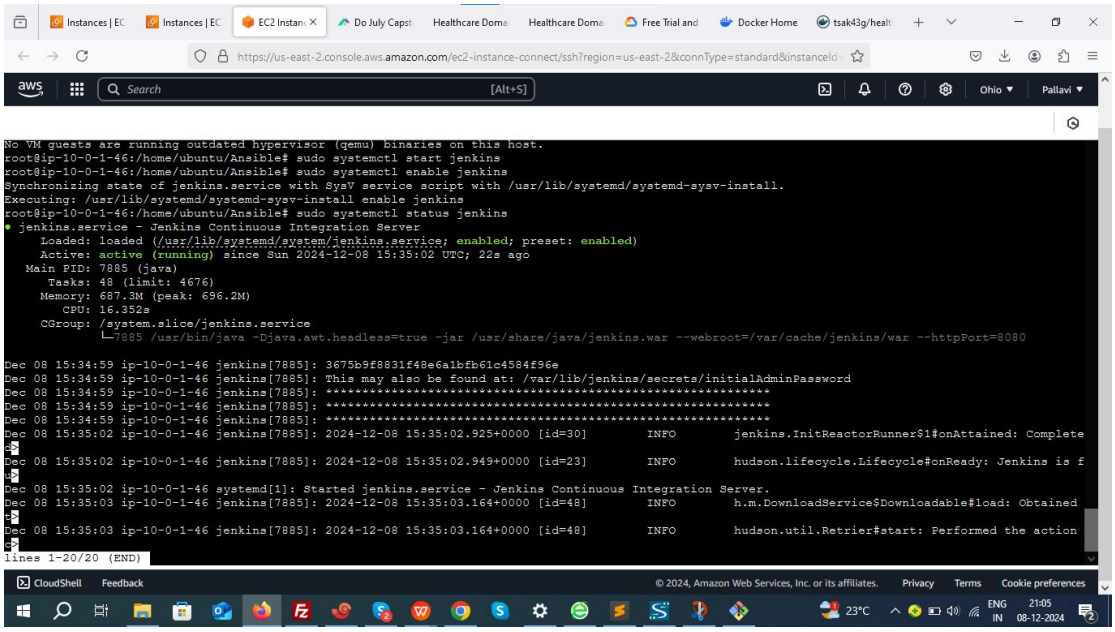
```
sudo apt-get install jenkins -y
```

Start and Enable Jenkins:

```
sudo systemctl start jenkins
sudo systemctl enable jenkins
```

Verify:

```
sudo systemctl status jenkins
```



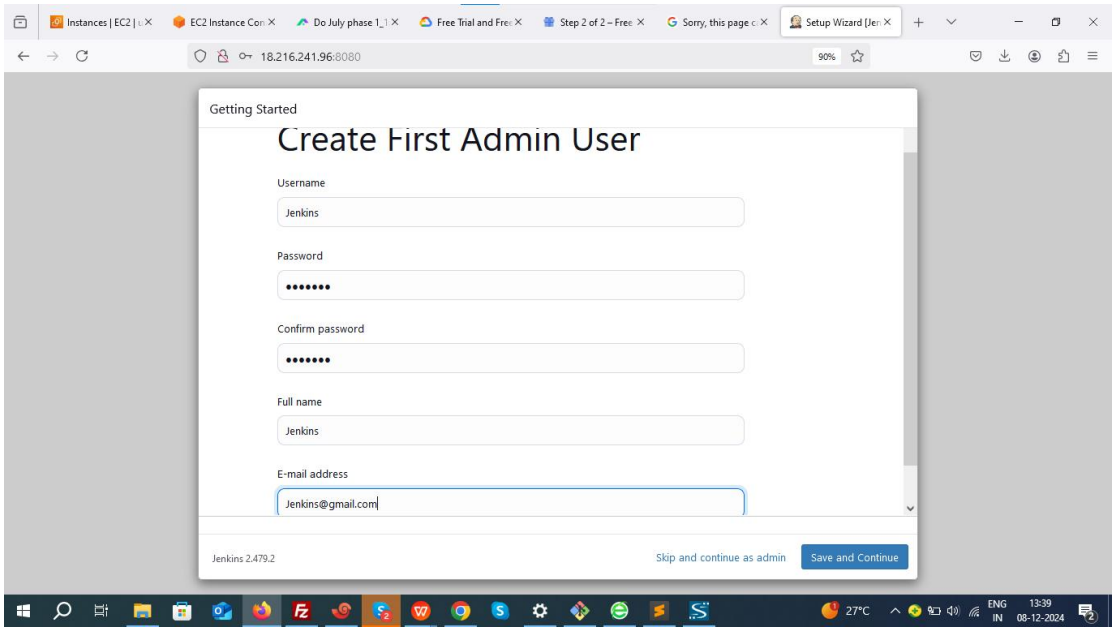
1. Now access to the jenkins dashboard using below url:

http://18.219.34.167:8080

Default port: 8080.

2. Set up plugins and admin user.

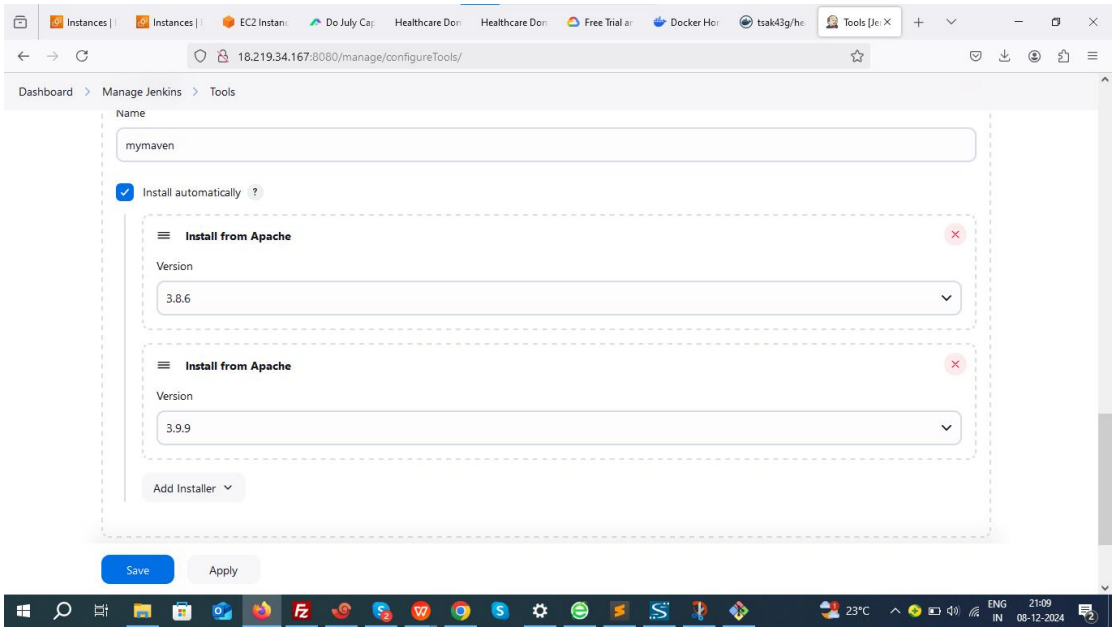
cat /var/lib/jenkins/secrets/initialAdminPassword



3. **Configure Maven**:

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

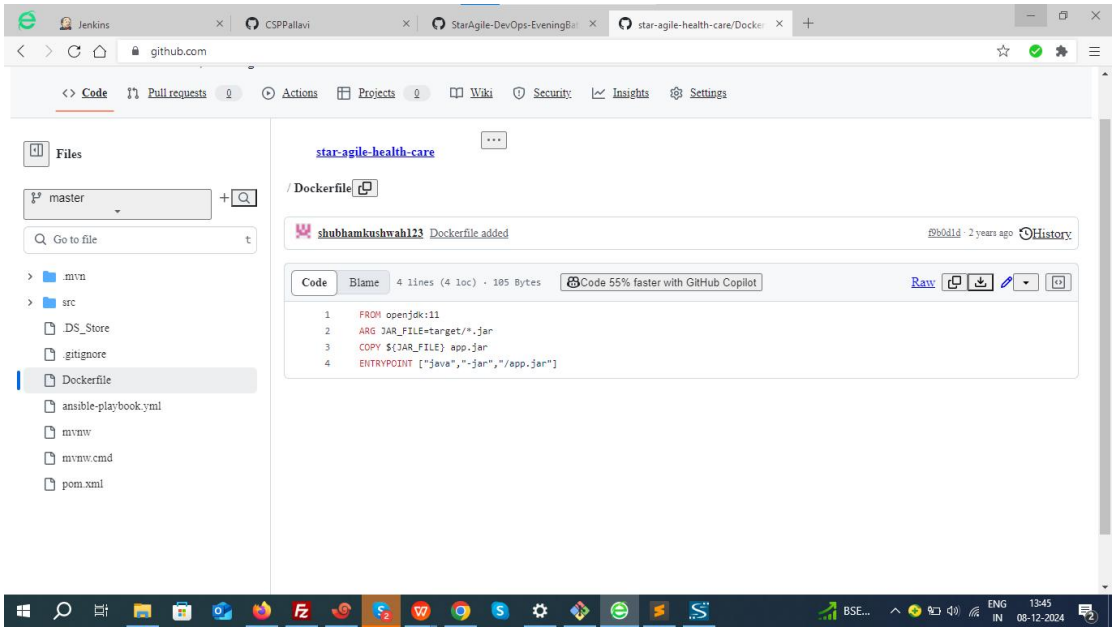
Add Maven and set it up.



Make sure to give same name for maven in pipeline.

4. 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"]
```

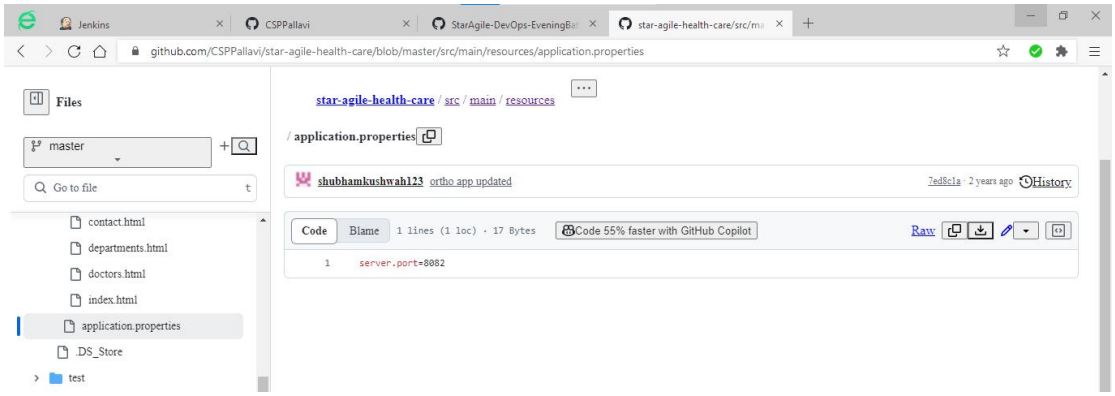


5. **Create Pipeline**:

Points to be noted befor running pipeline:

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

1. Check server port



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

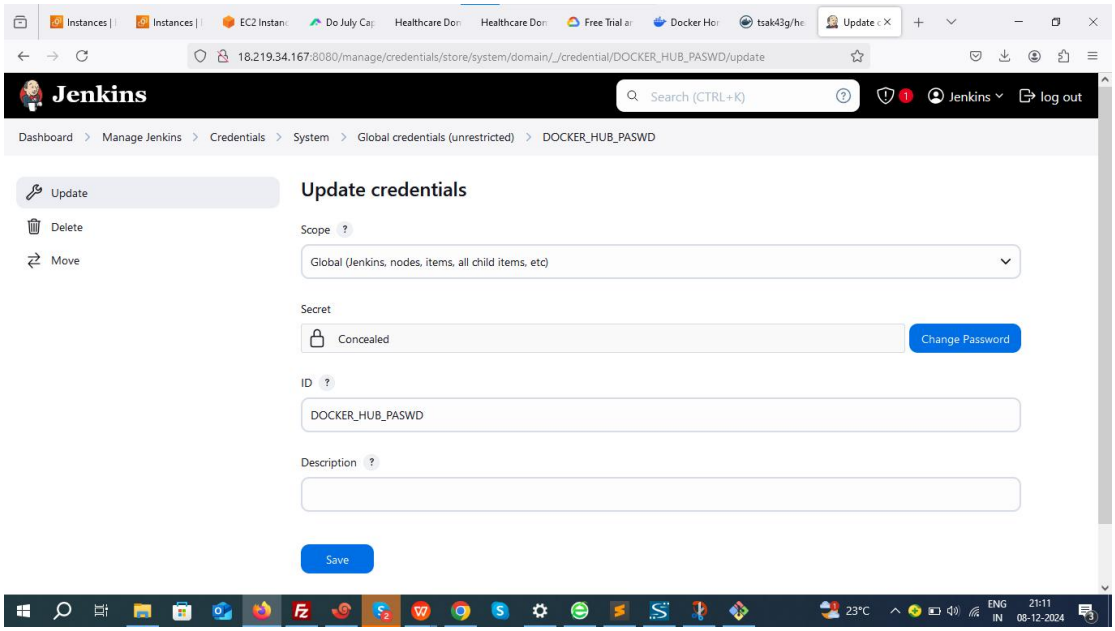
docker run -d -p 8082:8082 myproject1:$BUILD_NUMBER

3. Enable 8082 in your ec2 security groups.
4. Also, same port number should be exposed in docker file.

5. 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**.

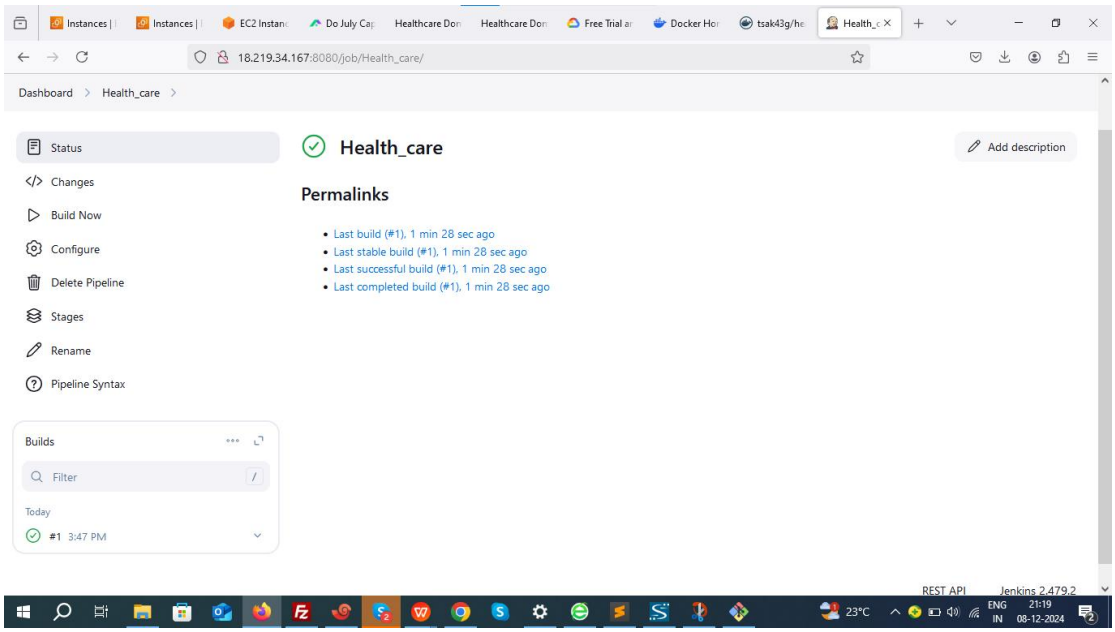
```
pipeline{

    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.

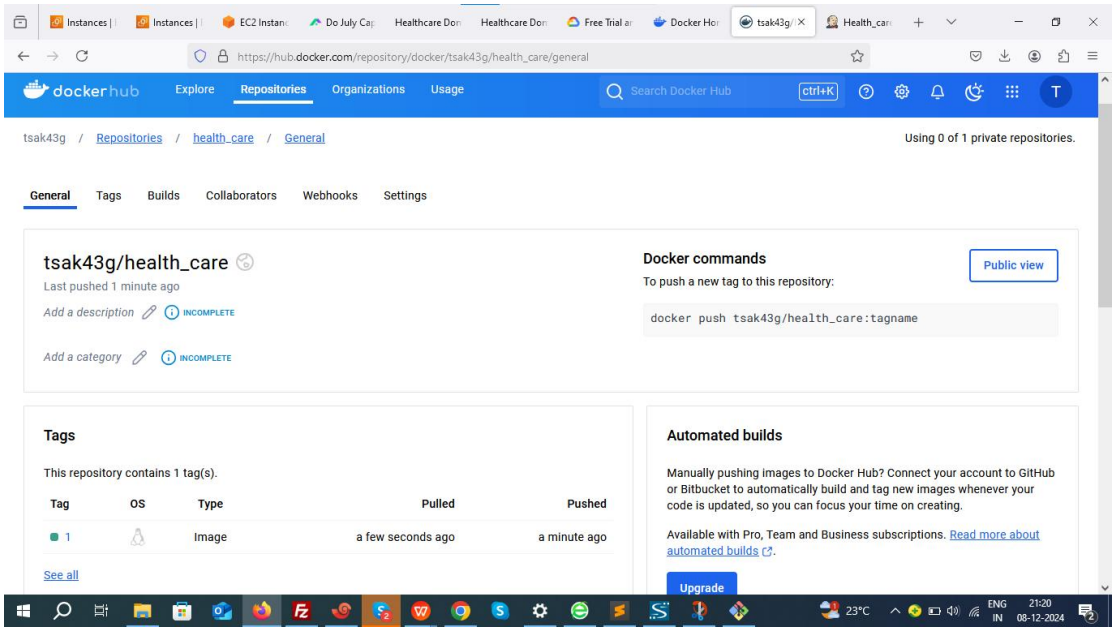
Build the job successfully.



Verify docker image.

```
root@ip-10-0-1-235:/home/ubuntu/Ansible# docker images
REPOSITORY            TAG       IMAGE ID       CREATED          SIZE
health_care           3         199db6f675ae   21 minutes ago   695MB
health_care           4         199db6f675ae   21 minutes ago   695MB
health_care           5         199db6f675ae   21 minutes ago   695MB
health_care           6         199db6f675ae   21 minutes ago   695MB
tsak43g/health_care   6         199db6f675ae   21 minutes ago   695MB
openjdk               11        47a932d998b7   2 years ago      654MB
root@ip-10-0-1-235:/home/ubuntu/Ansible#
```

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

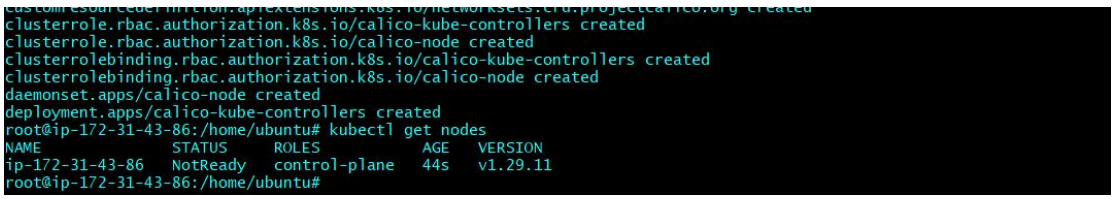
sudo kubeadm init --ignore-preflight-errors=all

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

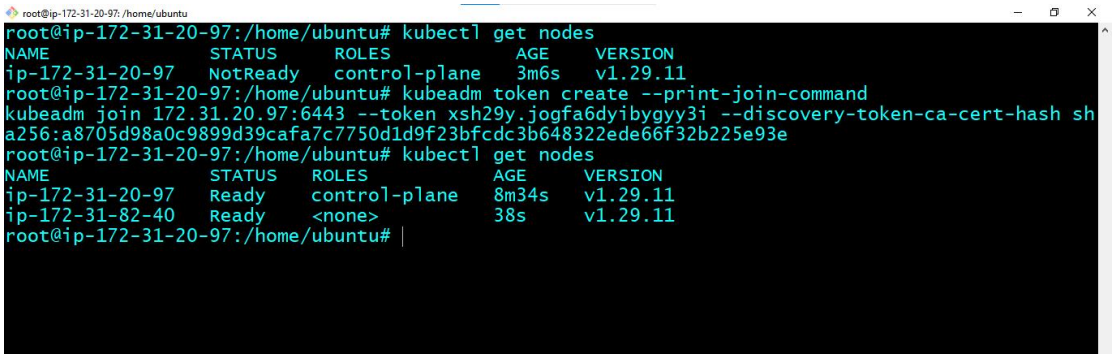
```
kubeadm token create --print-join-command
```

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
```

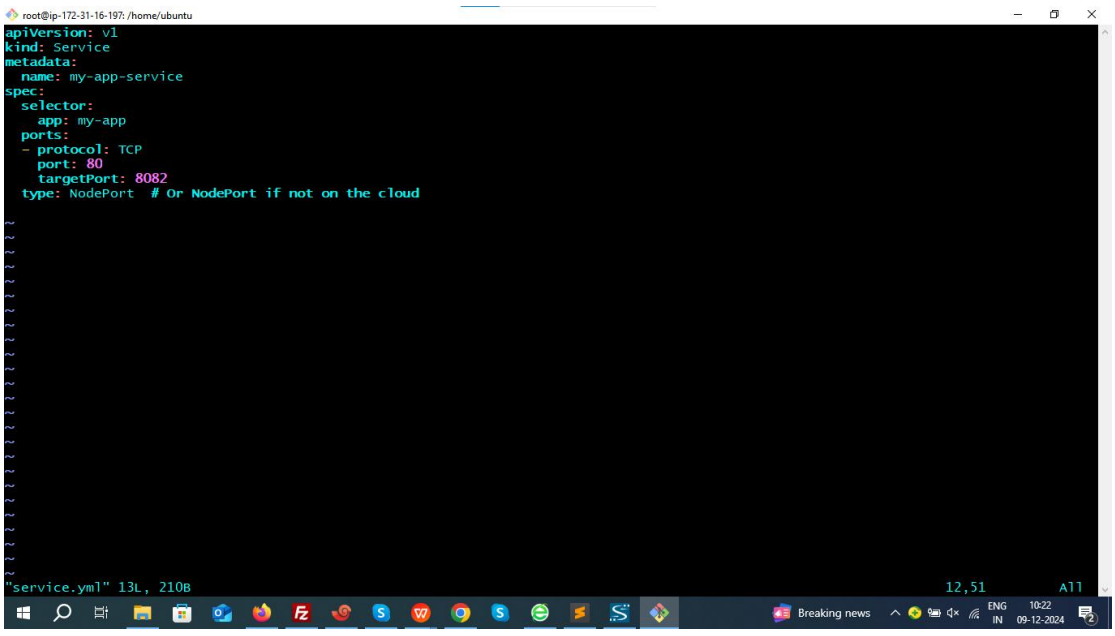
2. Apply the Deployment

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
```



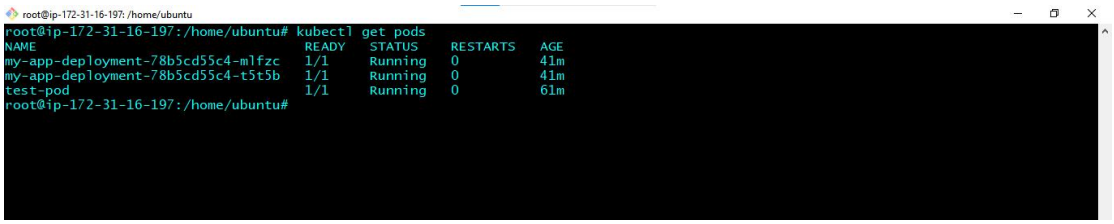
Apply the service:

kubectl apply -f service.yaml

5.**Verify the Deployment and Service**

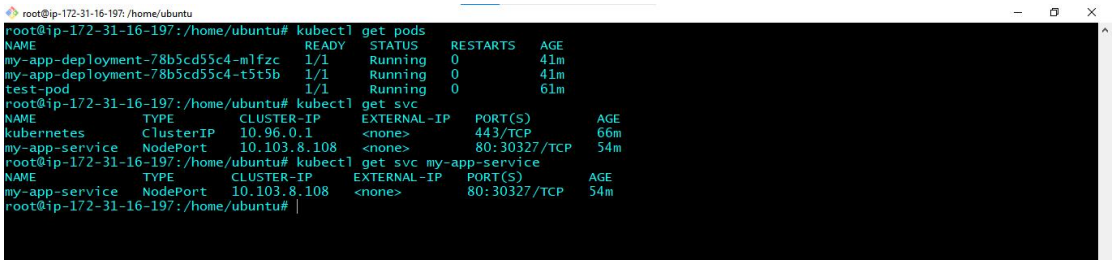
Check the Pods:

kubectl get pods



Check the Service:

kubectl get service my-app-service

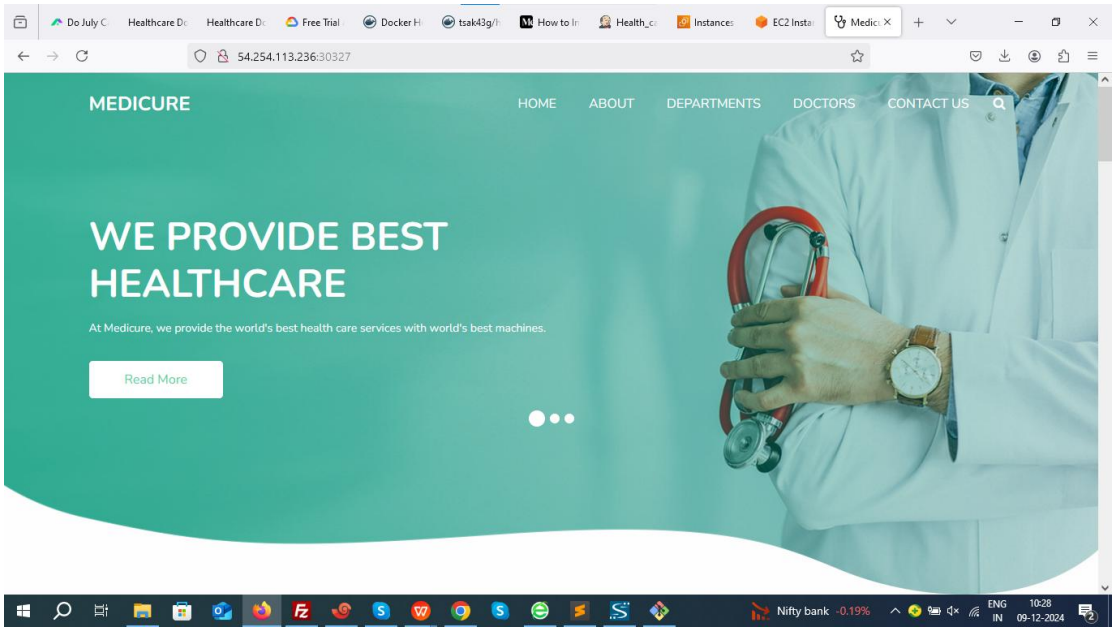


Verify that the service has a auto assigned NodePort (30996)

Access the container.

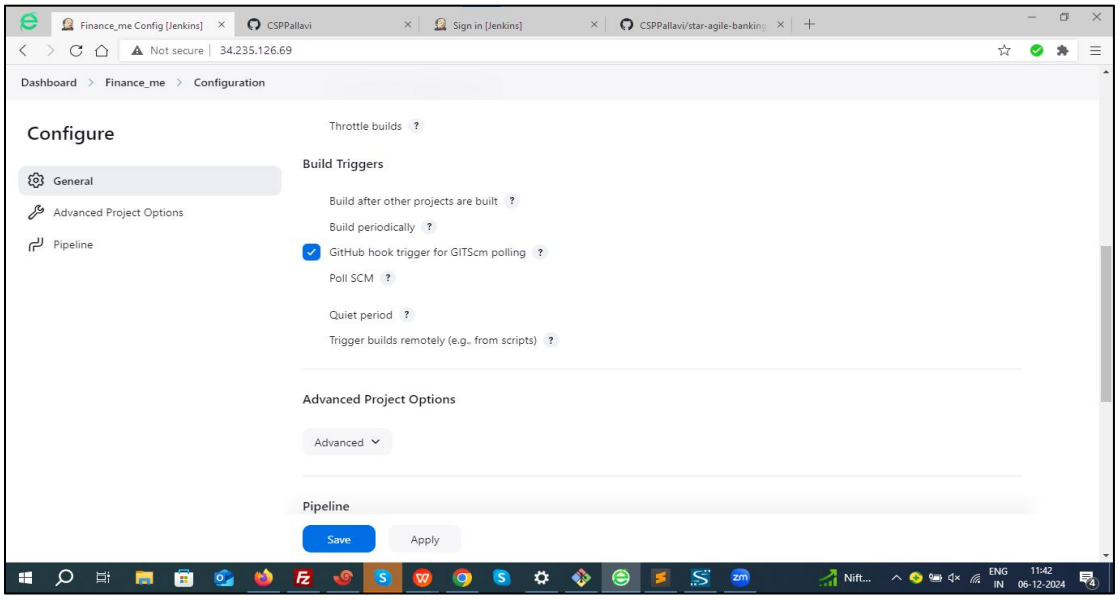
http://< public ip of worker node>:30996

http://13.126.231.60:30996/



STEP 4: WEBHOOK CONFIGURATION:

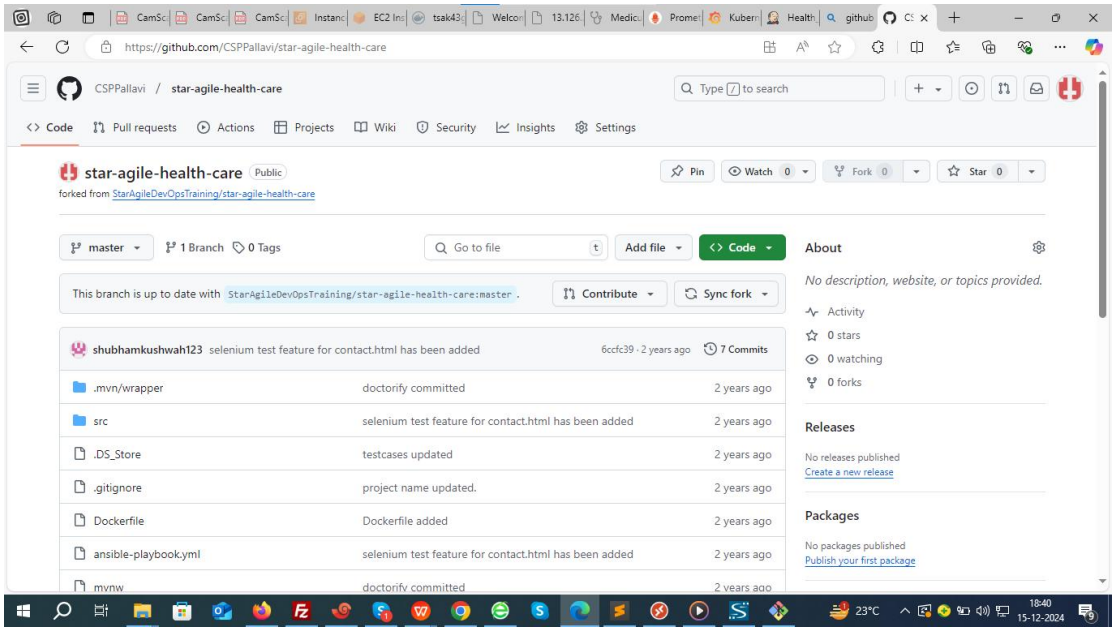
Go to your Jenkins dashboard and add build trigger- **GitHub hook trigger for GITScm polling**



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.

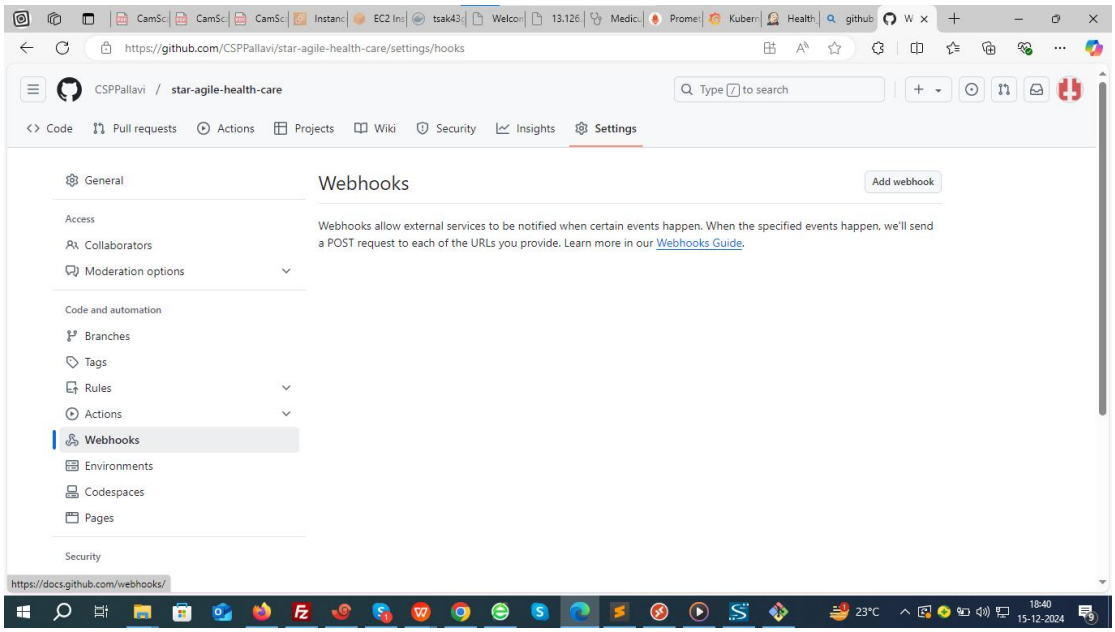
Step 1: Open Your Repository Settings

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



Step 2: Find the Webhooks Section

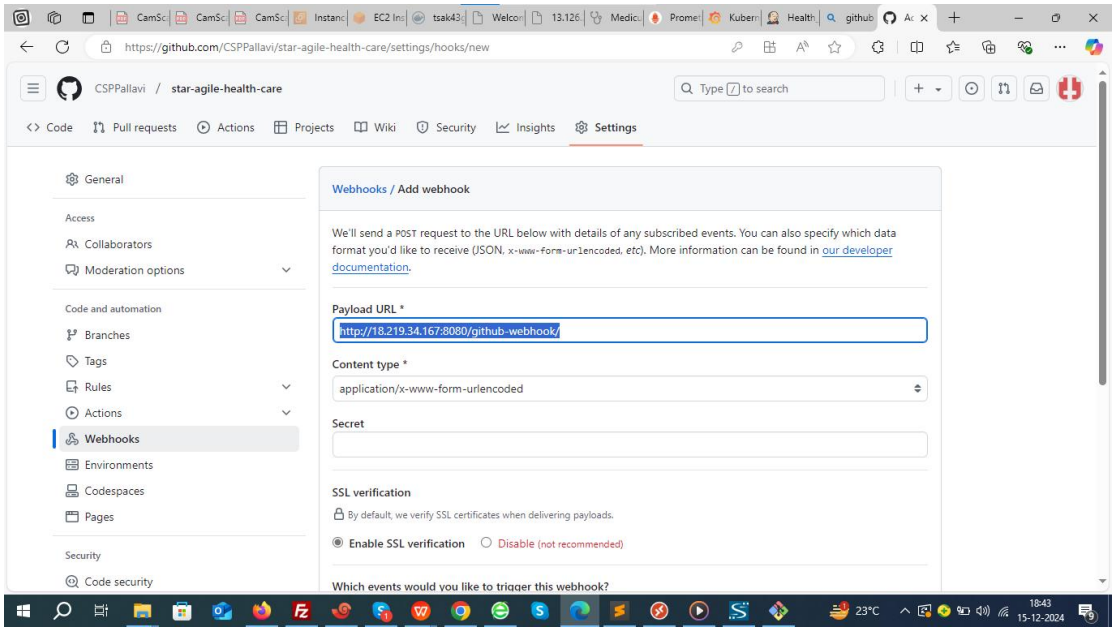
Go to **Repository Settings** > **Webhooks**.



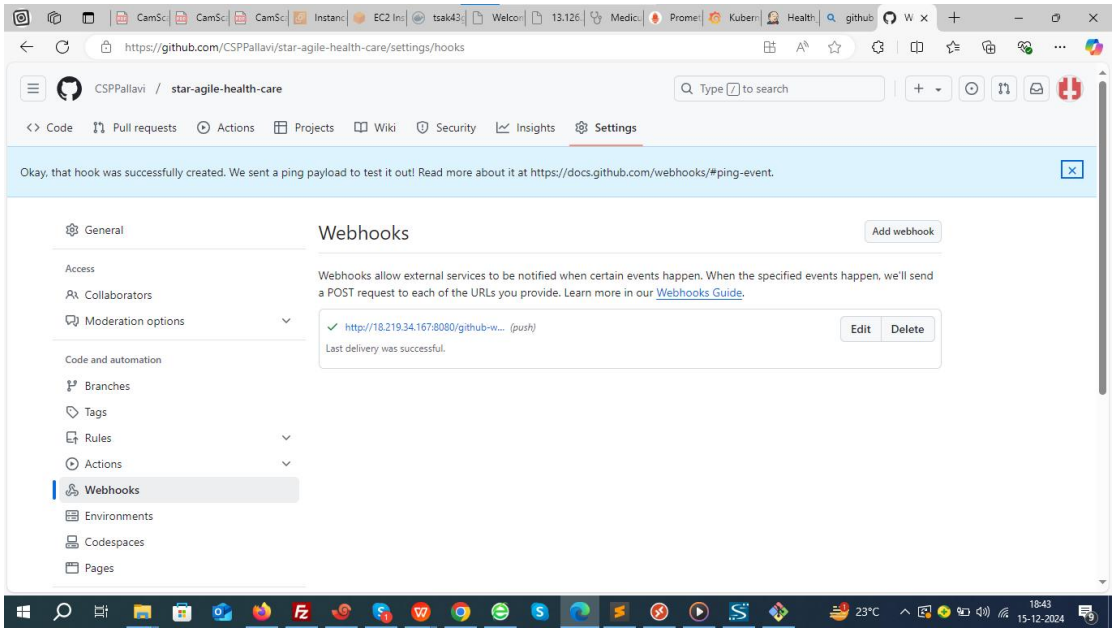
Step 3: Add a New Webhook

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.

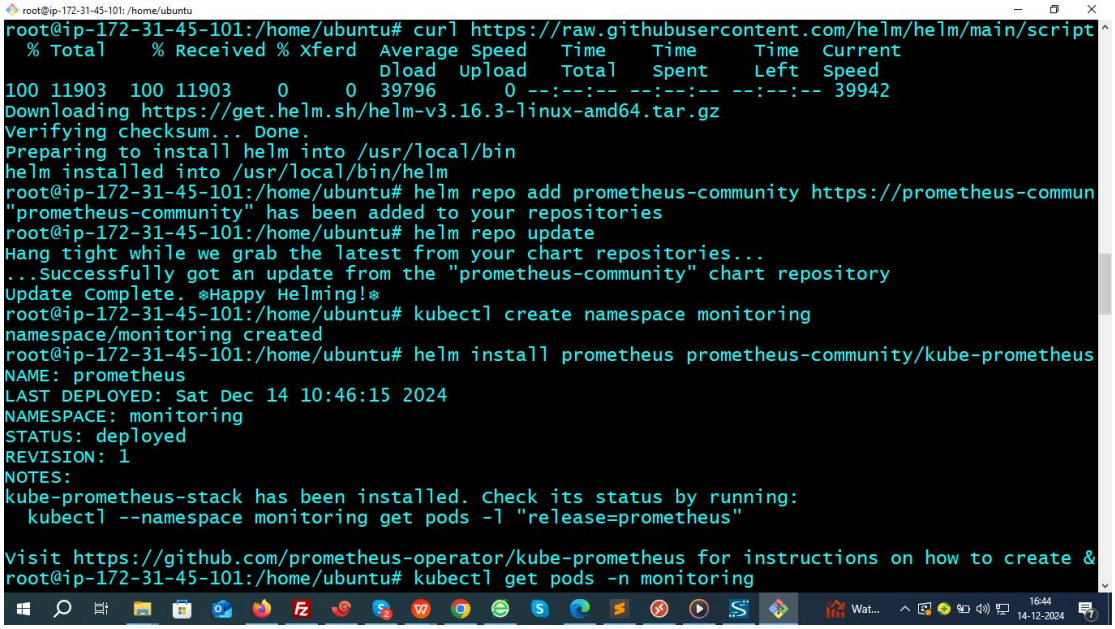
Install Helm (if not already installed):

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

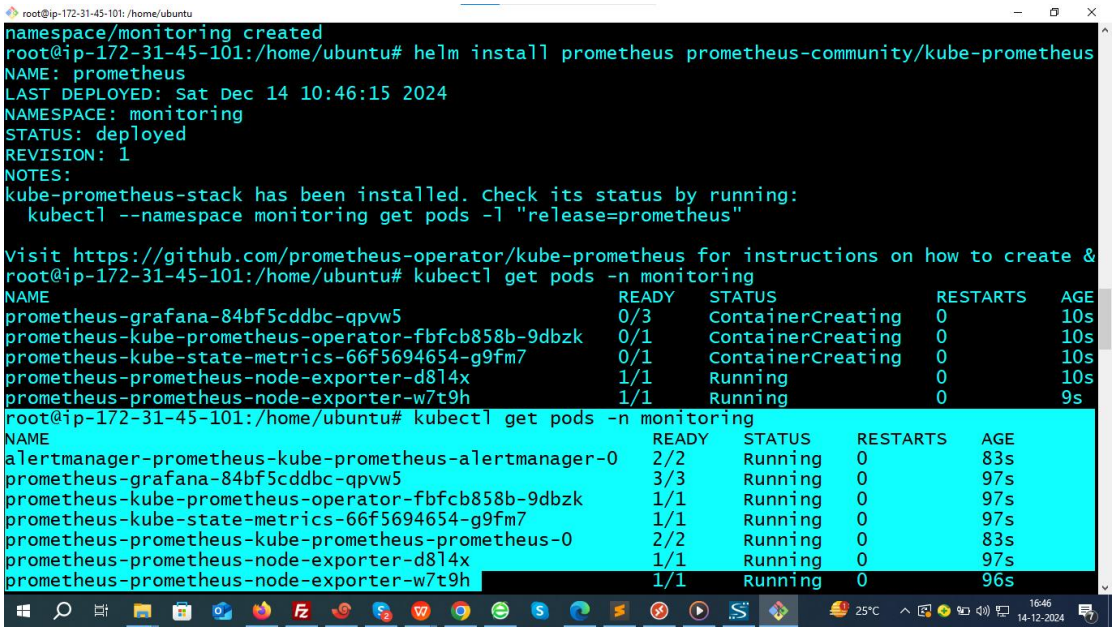
helm install prometheus prometheus-community/kube-prometheus-stack --namespace monitoring

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):

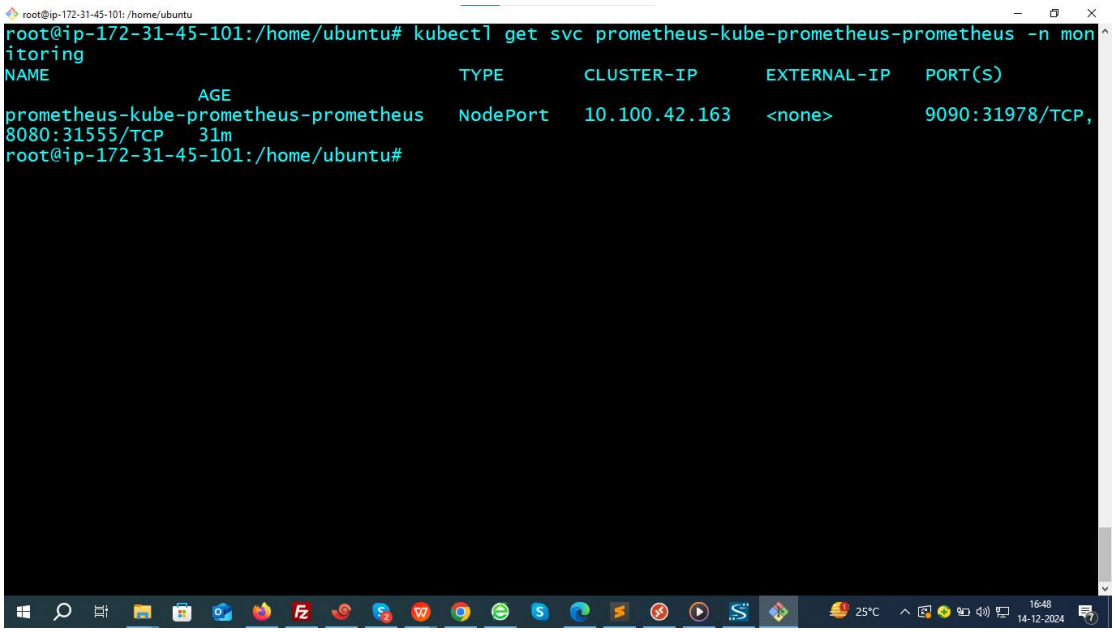
kubectl port-forward svc/prometheus-kube-prometheus-prometheus -n monitoring 9090:9090

This command creates service type cluster IP

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

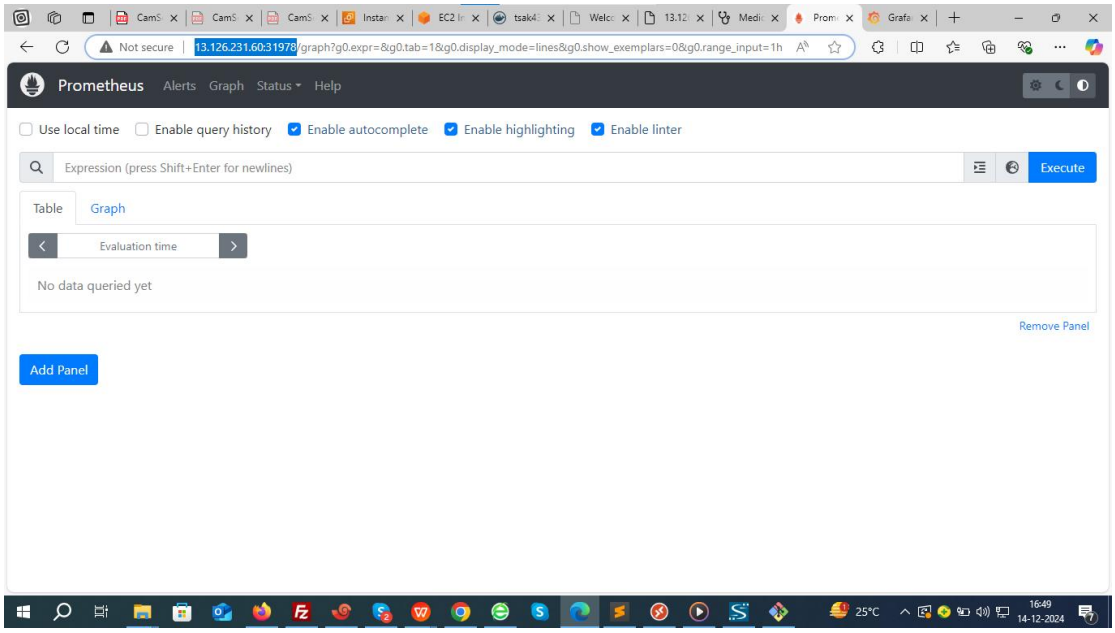
kubectl patch svc prometheus-kube-prometheus-prometheus -n monitoring -p '{"spec": {"type": "NodePort"}}'

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

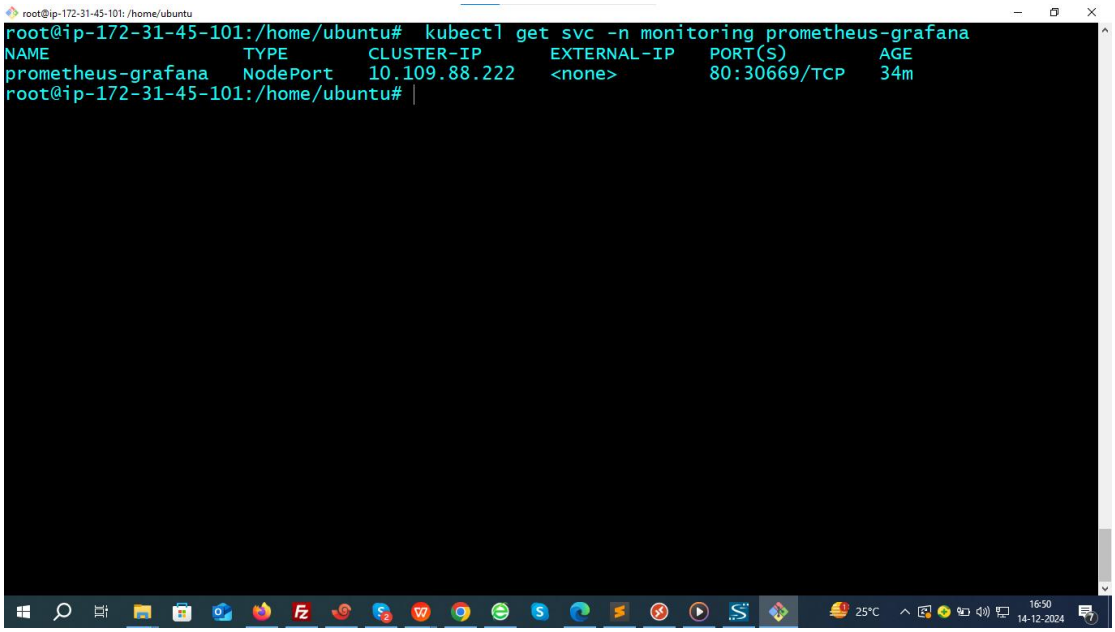
This command again creates service type cluster IP

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:

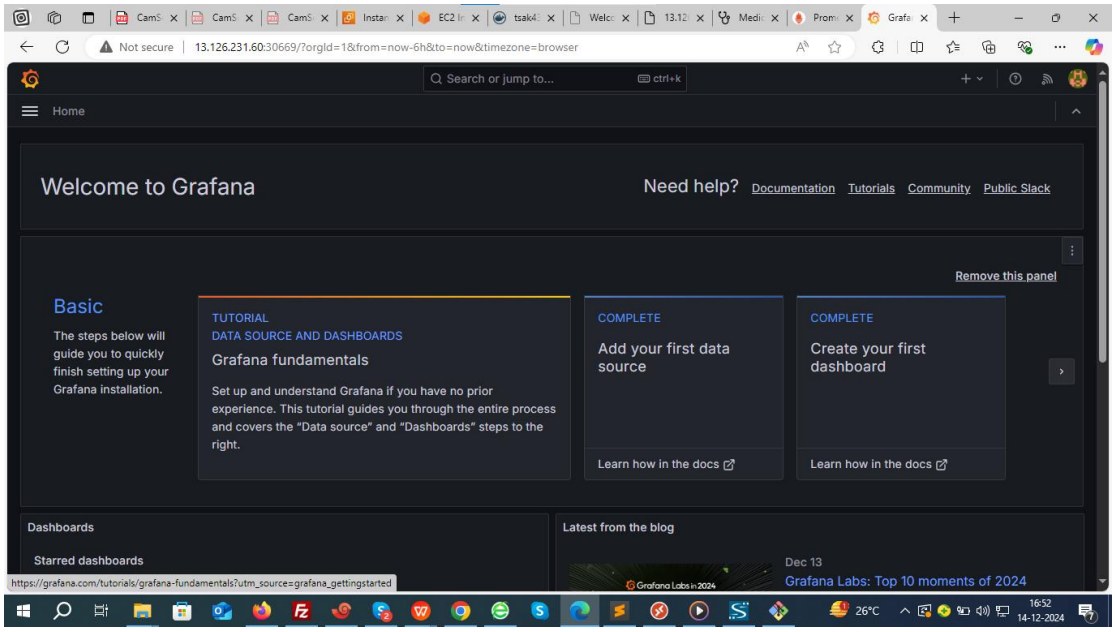
kubectl get svc -n monitoring prometheus-grafana



http://13.126.231.60:30669/

The default login credentials are:

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



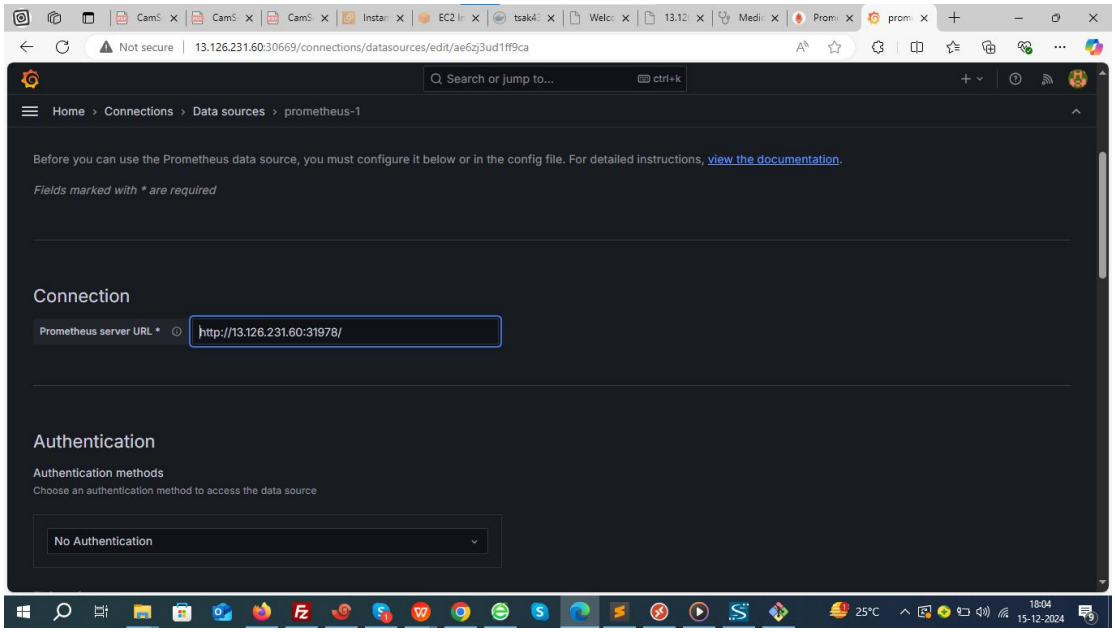
5. Configure Grafana with Prometheus Data Source

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/

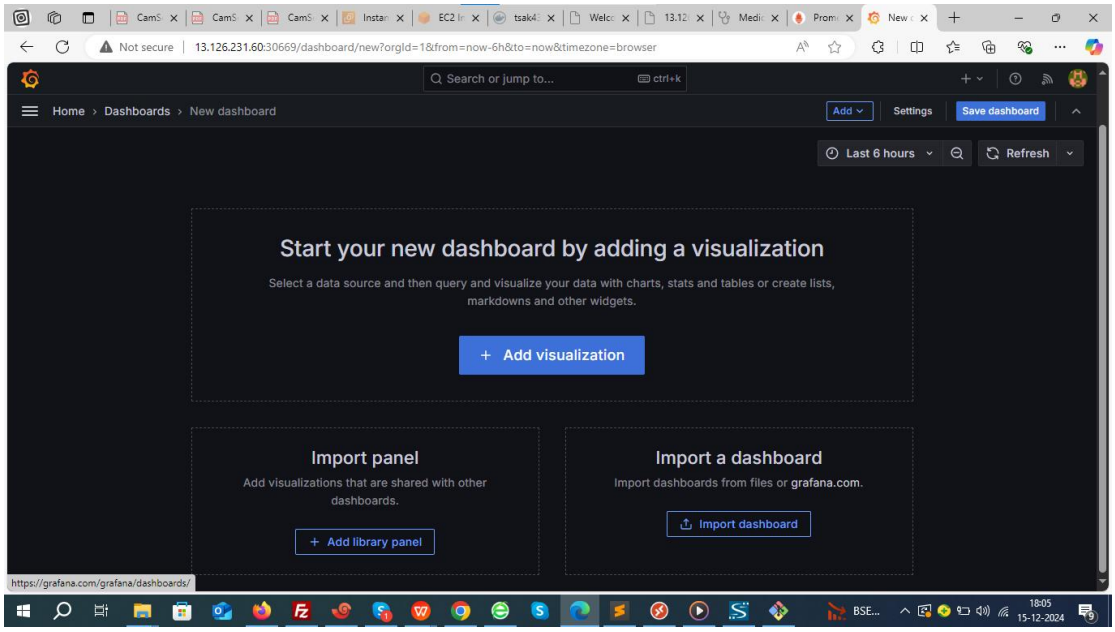
Click **Save & Test** to ensure the connection is successful.



6. Import Grafana Dashboards

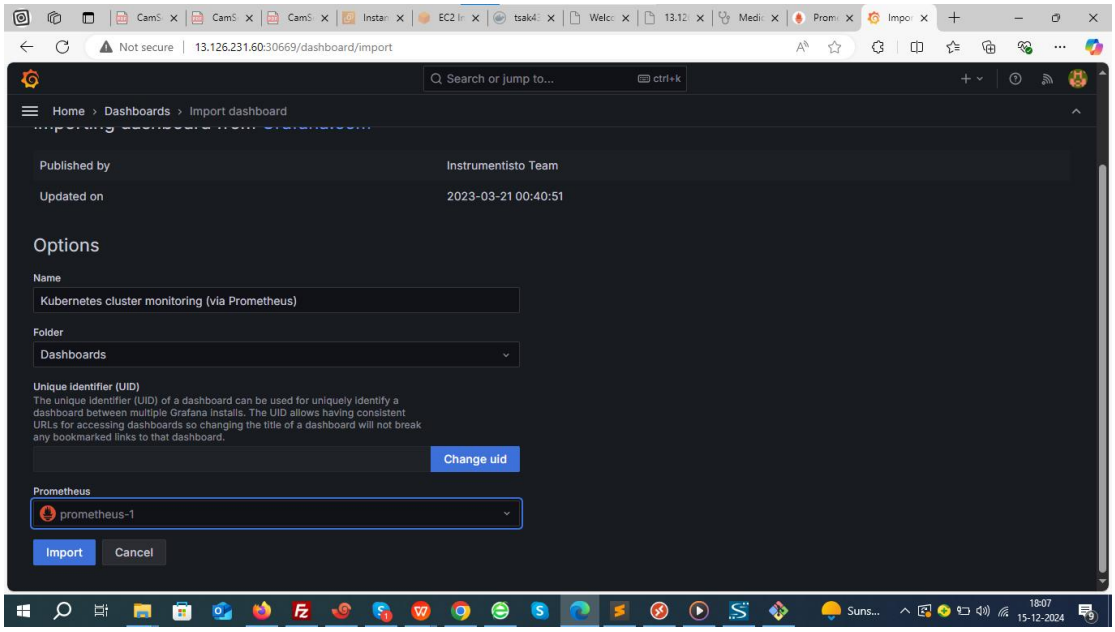
Grafana has pre-built dashboards for Kubernetes monitoring that you can import.

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

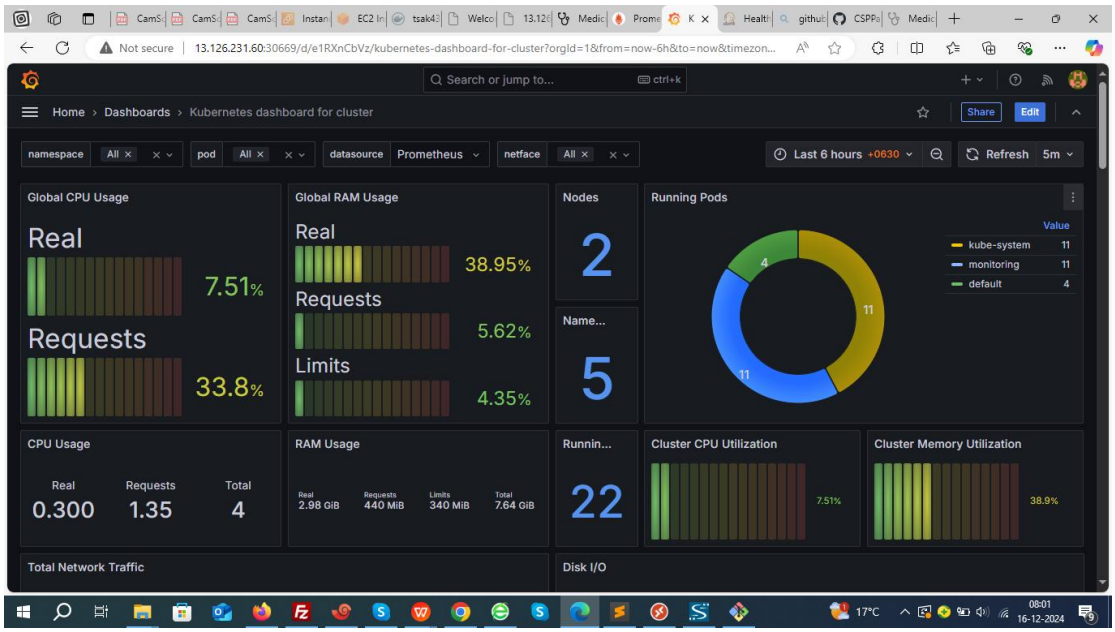


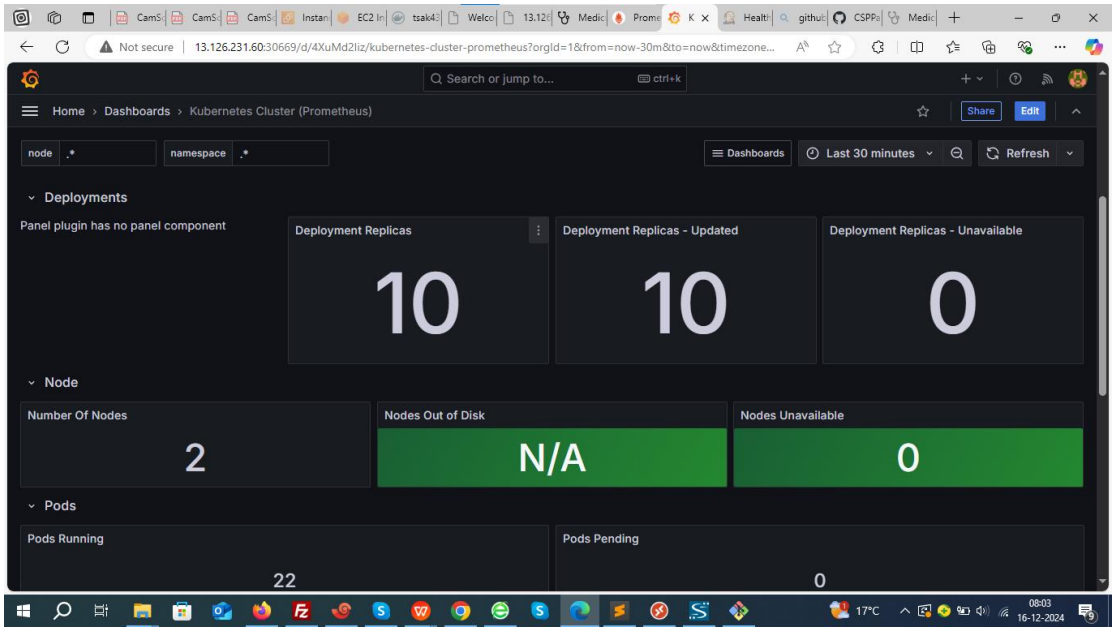
- Enter **kubernetes monitoring dashboards** or use specific dashboard IDs like 315 for Kubernetes or 6417 for a more specific one from the Grafana Dashboard repository.

- 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.




Not secure
13.126.231.60:30669/d/4XuMd2liz/kubernetes-cluster-prometheus?orgId=1&from=now-30m&to=now&timezone...
Search or jump to...
ctrl+k
Home › Dashboards › Kubernetes Cluster (Prometheus)
Share
Edit
node .*
namespace .*
Dashboards
Last 30 minutes
Refresh
Deployments
Panel plugin has no panel component
Deployment Replicas
10
Deployment Replicas - Updated
10
Deployment Replicas - Unavailable
0
Node
Number Of Nodes
2
Nodes Out of Disk
N/A
Nodes Unavailable
0
Pods
Pods Running
22
Pods Pending
0
17°C
08:03
16-12-2024