

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

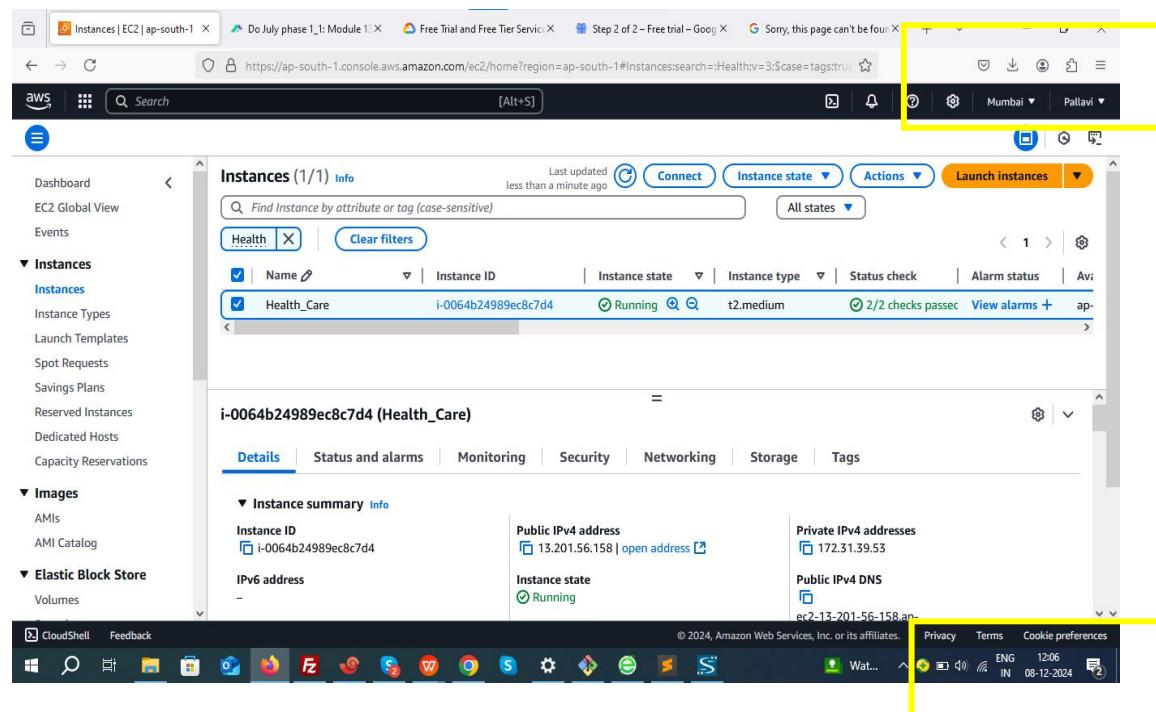
Medicure project code is attached below

[/https://github.com/CSPPallavi/star-agile-health-care.git](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).

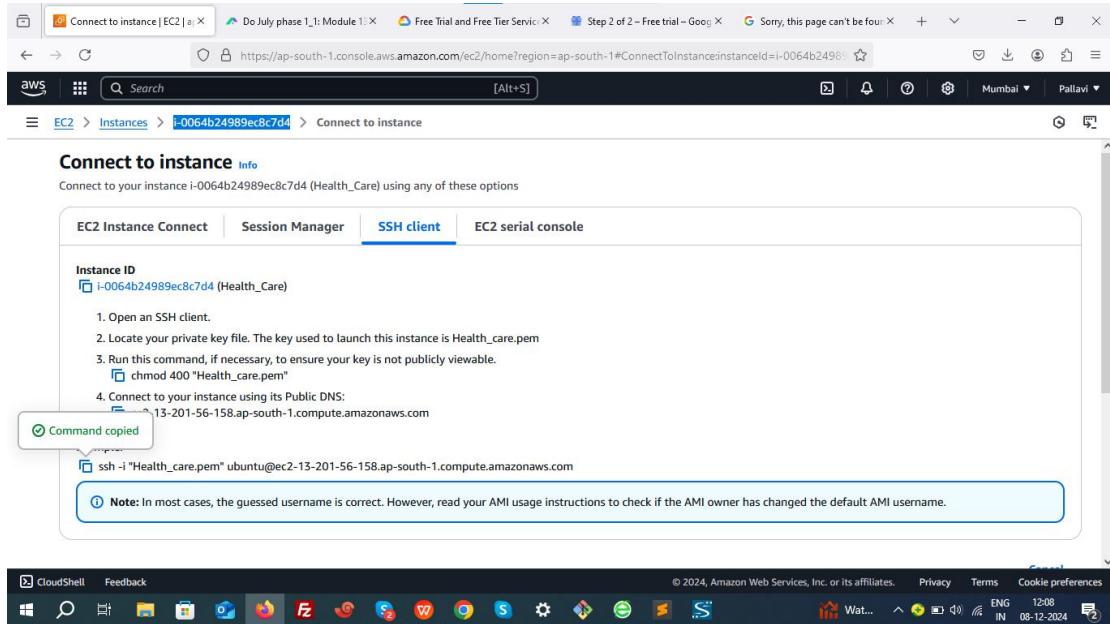


2. Connect to the VM:

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](https://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
```

```

root@ip-172-31-39-53: /home/ubuntu
The following NEW packages will be installed:
  terraform
0 upgraded, 1 newly installed, 0 to remove and 40 not upgraded.
Need to get 27.4 MB of archives.
After this operation, 90.2 MB of additional disk space will be used.
Get:1 https://apt.releases.hashicorp.com noble/main amd64 terraform amd64 1.10.1-1 [27.4 MB]
Fetched 27.4 MB in 0s (63.0 MB/s)
Selecting previously unselected package terraform.
(Reading database ... 70601 files and directories currently installed.)
Preparing to unpack .../terraform_1.10.1-1_amd64.deb ...
Unpacking terraform (1.10.1-1) ...
Setting up terraform (1.10.1-1) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
root@ip-172-31-39-53:/home/ubuntu# terraform --version
Terraform v1.10.1
on linux_amd64
root@ip-172-31-39-53:/home/ubuntu#

```

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.

User name	Path	Group	Last activity	MFA	Password age	Console
Terraform	/	0	7 minutes ago	-	-	-

Attach a Policy:

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

The screenshot shows the AWS IAM console with the URL <https://us-east-1.console.aws.amazon.com/iam/home?region=ap-south-1#users/details/Terraform?section=permissions>. The left sidebar shows 'Access management' with 'Users' selected. The main area is titled 'Permissions policies (1)'. A table lists one policy: 'AdministratorAccess' (AWS managed - job function, Attached via Directly). There are tabs for 'Groups', 'Tags', 'Security credentials', and 'Last Accessed'.

Generate Access Keys:

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

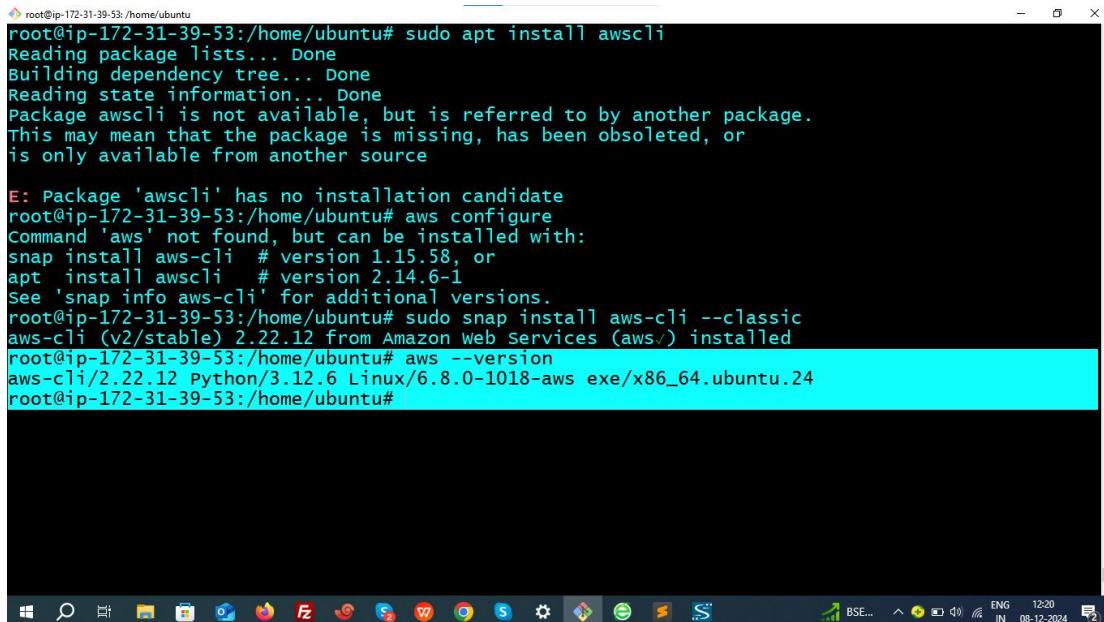
The screenshot shows the AWS IAM console with the URL https://us-east-1.console.aws.amazon.com/iam/home?region=ap-south-1#users/details/Terraform?section=security_creds. The left sidebar shows 'Access management' with 'Users' selected. The main area is titled 'Access keys (1)'. It shows one key: 'AKIA6GSNHCPUCLGVCE47'. The key has a status of 'Active', was last used 7 minutes ago, created 36 minutes ago, and was last used in the 'us-east-1' region for the 'sts' service. There is a 'Create access key' button and an 'Actions' dropdown.

1. Configure AWS CLI

Install AWS CLI (if not installed):



`sudo apt install awscli`



```
root@ip-172-31-39-53:/home/ubuntu# sudo apt install awscli
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Package awscli is not available, but is referred to by another package.
This may mean that the package is missing, has been obsoleted, or
is only available from another source

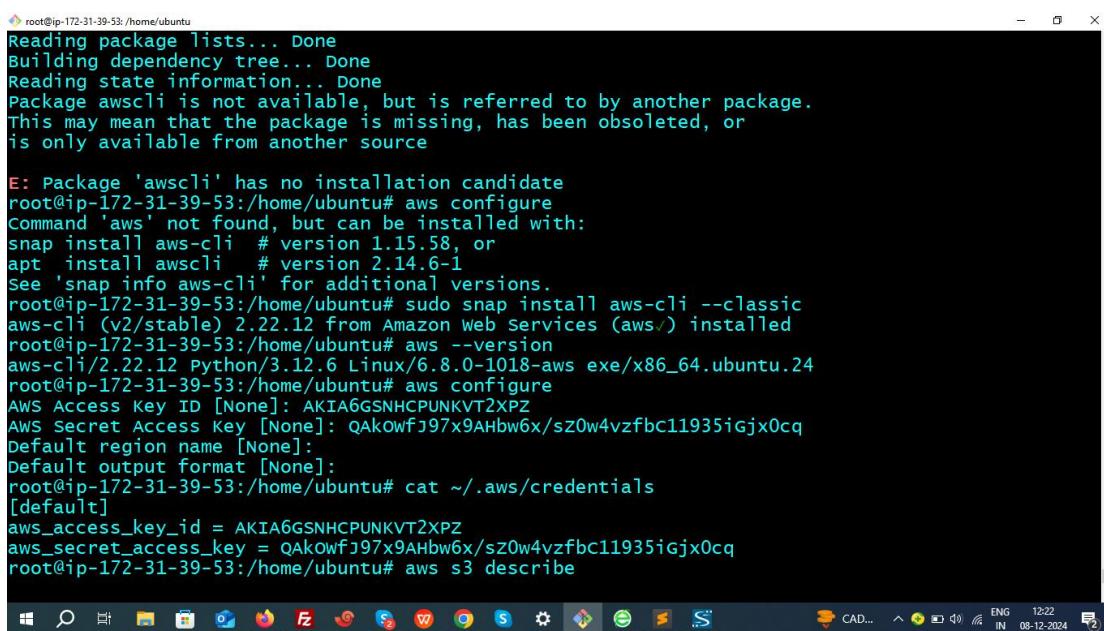
E: Package 'awscli' has no installation candidate
root@ip-172-31-39-53:/home/ubuntu# aws configure
Command 'aws' not found, but can be installed with:
snap install aws-cli # version 1.15.58, or
apt install awscli # version 2.14.6-1
See 'snap info aws-cli' for additional versions.
root@ip-172-31-39-53:/home/ubuntu# sudo snap install aws-cli --classic
aws-cli (v2/stable) 2.22.12 from Amazon web services (aws/) installed
root@ip-172-31-39-53:/home/ubuntu# aws --version
aws-cli/2.22.12 Python/3.12.6 Linux/6.8.0-1018-aws exe/x86_64.ubuntu.24
root@ip-172-31-39-53:/home/ubuntu#
```

`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`



```
root@ip-172-31-39-53:/home/ubuntu#
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Package awscli is not available, but is referred to by another package.
This may mean that the package is missing, has been obsoleted, or
is only available from another source

E: Package 'awscli' has no installation candidate
root@ip-172-31-39-53:/home/ubuntu# aws configure
Command 'aws' not found, but can be installed with:
snap install aws-cli # version 1.15.58, or
apt install awscli # version 2.14.6-1
See 'snap info aws-cli' for additional versions.
root@ip-172-31-39-53:/home/ubuntu# sudo snap install aws-cli --classic
aws-cli (v2/stable) 2.22.12 from Amazon web services (aws/) installed
root@ip-172-31-39-53:/home/ubuntu# aws --version
aws-cli/2.22.12 Python/3.12.6 Linux/6.8.0-1018-aws exe/x86_64.ubuntu.24
root@ip-172-31-39-53:/home/ubuntu# aws configure
AWS Access Key ID [None]: AKIA6GSNHCUNKVT2XPZ
AWS Secret Access Key [None]: QAkowfJ97x9AHbw6x/sz0w4vzfbcl11935igjx0cq
Default region name [None]:
Default output format [None]:
root@ip-172-31-39-53:/home/ubuntu# cat ~/.aws/credentials
[default]
aws_access_key_id = AKIA6GSNHCUNKVT2XPZ
aws_secret_access_key = QAkowfJ97x9AHbw6x/sz0w4vzfbcl11935igjx0cq
root@ip-172-31-39-53:/home/ubuntu# aws s3 describe
```

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"
]

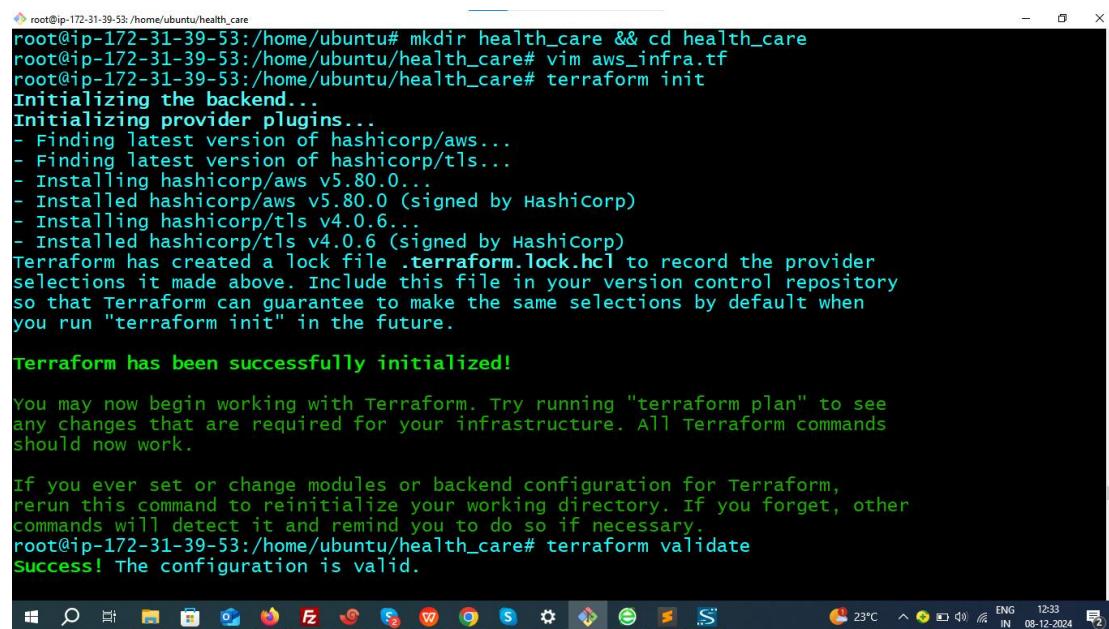
}

}

```

Initialize and Apply Terraform:

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



```

root@ip-172-31-39-53:/home/ubuntu/health_care#
root@ip-172-31-39-53:/home/ubuntu/health_care# mkdir health_care && cd health_care
root@ip-172-31-39-53:/home/ubuntu/health_care# vim aws_infra.tf
root@ip-172-31-39-53:/home/ubuntu/health_care# terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Finding latest version of hashicorp/tls...
- Installing hashicorp/aws v5.80.0...
- Installed hashicorp/aws v5.80.0 (signed by Hashicorp)
- Installing hashicorp/tls v4.0.6...
- Installed hashicorp/tls v4.0.6 (signed by Hashicorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
root@ip-172-31-39-53:/home/ubuntu/health_care# terraform validate
Success! The configuration is valid.

```

terraform validate - Checks for syntax errors and configuration validity.

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

```
root@ip-172-31-4-243:~/aws/Finance_me
    }
+ tags_all
+ "Name" = "s1-vpc"
}
}

# tls_private_key.mykey will be created
+ resource "tls_private_key" "mykey" {
+ algorithm           = "RSA"
+ ecdsa_curve         = "P224"
+ id                  = (known after apply)
+ private_key_openssh = (sensitive value)
+ private_key_pem     = (sensitive value)
+ private_key_pem_pkcs8 = (sensitive value)
+ public_key_fingerprint_md5 = (known after apply)
+ public_key_fingerprint_sha256 = (known after apply)
+ public_key_openssh   = (known after apply)
+ public_key_pem       = (known after apply)
+ rsa_bits             = 2048
}

Plan: 10 to add, 0 to change, 0 to destroy.

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take
exactly these actions if you run "terraform apply" now.
root@ip-172-31-4-243:~/aws/Finance_me# |
```

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

```
root@ip-172-31-4-243:~/aws/Finance_me
aws_instance.myec2 (remote-exec): Scanning processes... [=====]
aws_instance.myec2 (remote-exec): Scanning processes...
aws_instance.myec2 (remote-exec): Scanning linux images... [====]
aws_instance.myec2 (remote-exec): Scanning linux images... [====]
aws_instance.myec2 (remote-exec): Scanning linux images... [=====]
aws_instance.myec2 (remote-exec): Scanning linux images... [=====]
aws_instance.myec2 (remote-exec): scanning linux images...

aws_instance.myec2 (remote-exec): Running kernel seems to be up-to-date.
aws_instance.myec2 (remote-exec): No services need to be restarted.
aws_instance.myec2 (remote-exec): No containers need to be restarted.
aws_instance.myec2 (remote-exec): No user sessions are running outdated
aws_instance.myec2 (remote-exec): binaries.

aws_instance.myec2 (remote-exec): No VM guests are running outdated
aws_instance.myec2 (remote-exec): hypervisor (qemu) binaries on this
aws_instance.myec2 (remote-exec): host.
aws_instance.myec2: Creation complete after 2m3s [id=i-0dfdbaeb9710d985f]

Apply complete! Resources: 10 added, 0 changed, 0 destroyed.
root@ip-172-31-4-243:~/aws/Finance_me# |
```

Instance has been created in us-east-2

The screenshot shows the AWS CloudShell interface. On the left, a sidebar navigation includes 'Dashboard', 'EC2 Global View', 'Events', 'Instances' (selected), 'Instance Types', 'Launch Templates', 'Spot Requests', 'Savings Plans', 'Reserved Instances', 'Dedicated Hosts', 'Capacity Reservations', 'Images' (AMIs, AMI Catalog), and 'Elastic Block Store' (Volumes). The main content area displays 'Instances (1/1) Info' with a table showing one instance: 'Health_Care' (i-0ef13b7621026b553), which is 'Running' and 't2.medium'. Below this, a detailed view for 'i-0ef13b7621026b553 (Health_Care)' shows the 'Details' tab selected, along with tabs for 'Status and alarms', 'Monitoring', 'Security', 'Networking', 'Storage', and 'Tags'. Under 'Instance summary', it shows the 'Public IPv4 address' as 18.219.34.167 and the 'Private IPv4 addresses' as 10.0.1.46. The bottom of the screen shows the AWS navigation bar and a footer with copyright information.

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`

The screenshot shows the AWS CloudShell interface. The terminal window displays the output of the 'ansible --version' command. The output includes system memory usage (8% memory, 0% swap), the IPv4 address for enX0 (10.0.1.46), and expanded security maintenance information. It also shows the last login time (Sun Dec 8 15:26:19 2024) and the user's sudo bash session. The command output continues with the Ansible configuration file path (/etc/ansible/ansible.cfg), module search paths, and various Python and Jinja versions. The bottom of the screen shows the AWS navigation bar and a footer with copyright information.

3. Install Java, git and docker Using Ansible Playbook:

Create a dedicated directory for 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 }}"
```

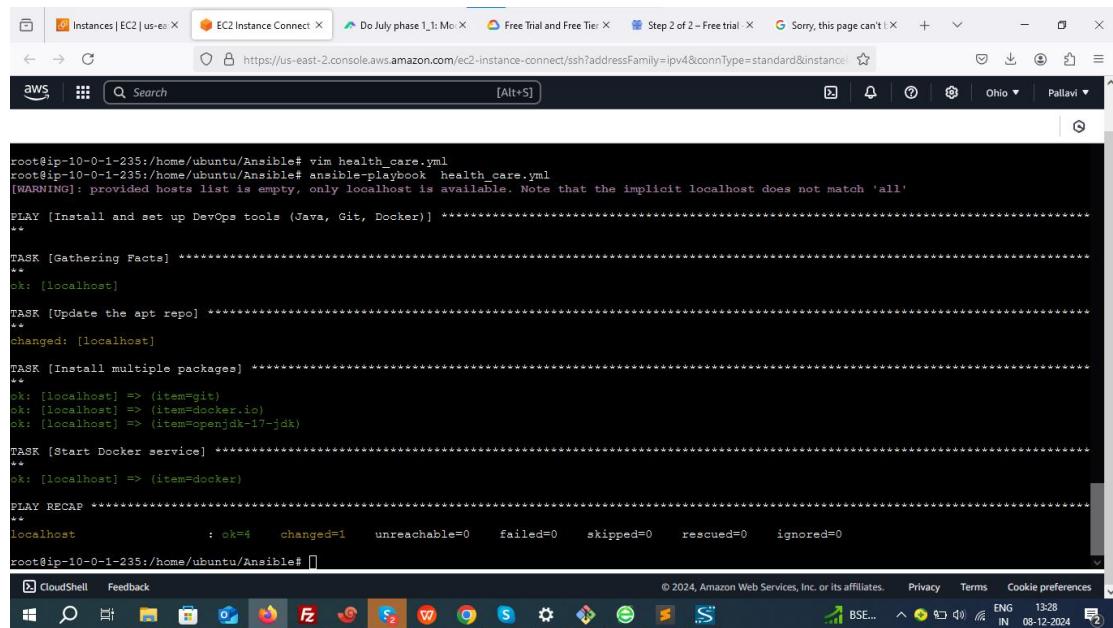
```
state: started
```

```
loop:
```

```
- docker
```

Run the playbook:

ansible-playbook health_care.yml



```
root@ip-10-0-1-235:/home/ubuntu/Ansible# vim health_care.yml
root@ip-10-0-1-235:/home/ubuntu/Ansible# ansible-playbook health_care.yml
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'

PLAY [Install and set up DevOps tools (Java, Git, Docker)] ****
**

TASK [Gathering Facts] ****
**
ok: [localhost]

TASK [Update the apt repo] ****
**
changed: [localhost]

TASK [Install multiple packages] ****
**
ok: [localhost] => (item=git)
ok: [localhost] => (item=docker.io)
ok: [localhost] => (item=openjdk-17-jdk)

TASK [Start Docker service] ****
**
ok: [localhost] => (item=docker)

PLAY RECAP ****
**
localhost : ok=4    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

root@ip-10-0-1-235:/home/ubuntu/Ansible#
```

Verify installations.

```

xt: [localhost] => (item=docker_1)
xt: [localhost] => (item=openjdk-17-jdk)

root@ip-10-0-1-235:/home/ubuntu/Ansible# java --version
openjdk 17.0.13 2024-10-15
OpenJDK Runtime Environment (build 17.0.13+11-Ubuntu-2ubuntu124.04)
OpenJDK 64-Bit Server VM (build 17.0.13+11-Ubuntu-2ubuntu124.04, mixed mode, sharing)
root@ip-10-0-1-235:/home/ubuntu/Ansible# docker --version
Docker version 26.1.3, build 26.1.3-0ubuntu1~24.04.1
root@ip-10-0-1-235:/home/ubuntu/Ansible# git --version
git version 2.43.0
root@ip-10-0-1-235:/home/ubuntu/Ansible#

```

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences ENG IN 13:30 Lewi... 08-12-2024

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
```

```

No VM guests are running outdated hypervisor (qemu) binaries on this host.
root@ip-10-0-1-46:/home/ubuntu/.Ansible# sudo systemctl start jenkins
root@ip-10-0-1-46:/home/ubuntu/.Ansible# sudo systemctl enable jenkins
Synchronizing state of jenkins.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable jenkins
root@ip-10-0-1-46:/home/ubuntu/.Ansible# sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
    Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
      Active: active (running) since Sun 2024-12-08 15:35:02 UTC; 22s ago
        Main PID: 7885 (java)
          Tasks: 48 (limit: 4676)
            Memory: 687.3M (peak: 696.2M)
              CPU: 16.352s
            CGroup: /system.slice/jenkins.service
                    └─7885 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Dec 08 15:34:59 ip-10-0-1-46 jenkins[7885]: 3675b9f8831f48e6a1bfb61c4584f96e
Dec 08 15:34:59 ip-10-0-1-46 jenkins[7885]: This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
Dec 08 15:34:59 ip-10-0-1-46 jenkins[7885]: ****
Dec 08 15:34:59 ip-10-0-1-46 jenkins[7885]: ****
Dec 08 15:34:59 ip-10-0-1-46 jenkins[7885]: ****
Dec 08 15:35:02 ip-10-0-1-46 jenkins[7885]: 2024-12-08 15:35:02.925+0000 [id=30]      INFO      jenkins.InitReactorRunner$1@onAttained: Complete
Dec 08 15:35:02 ip-10-0-1-46 jenkins[7885]: 2024-12-08 15:35:02.949+0000 [id=23]      INFO      hudson.lifecycle.Lifecycle#onReady: Jenkins is f
Dec 08 15:35:02 ip-10-0-1-46 systemd[1]: Started Jenkins Continuous Integration Server.
Dec 08 15:35:03 ip-10-0-1-46 jenkins[7885]: 2024-12-08 15:35:03.164+0000 [id=48]      INFO      h.m.DownloadService$Downloadable#load: Obtained
Dec 08 15:35:03 ip-10-0-1-46 jenkins[7885]: 2024-12-08 15:35:03.164+0000 [id=48]      INFO      hudson.util.Retrier#start: Performed the action

```

lines 1-20/20 (END)

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences 23°C ENG 21:05 IN 08-12-2024

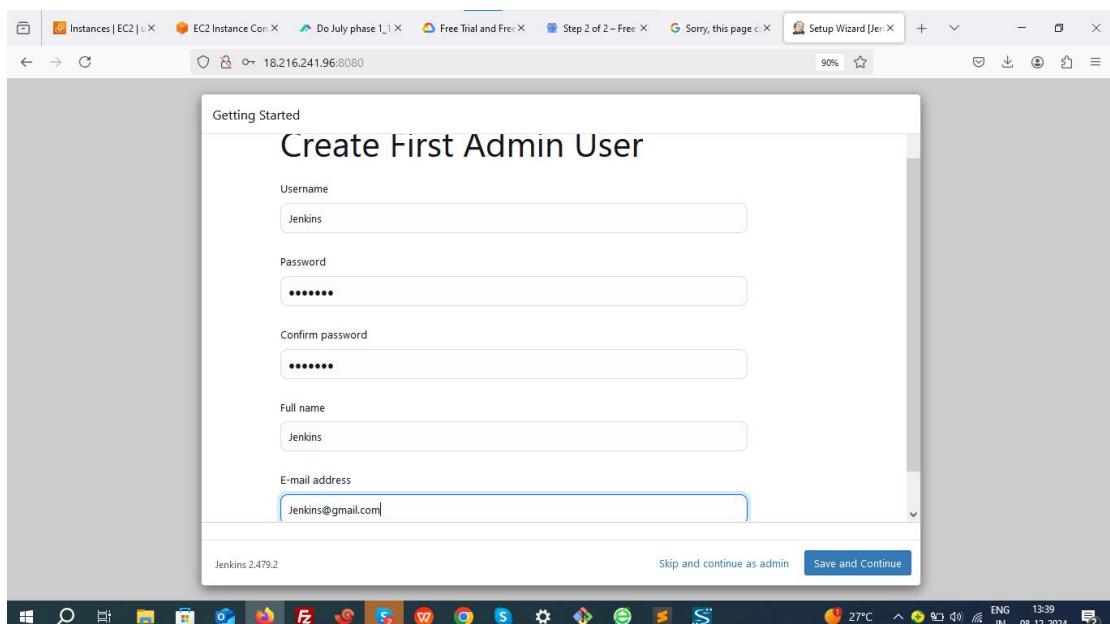
1. Now access to the jenkins dashboard using below url:

<http://18.219.34.167:8080>

Default port: 8080.

2. Set up plugins and admin user.

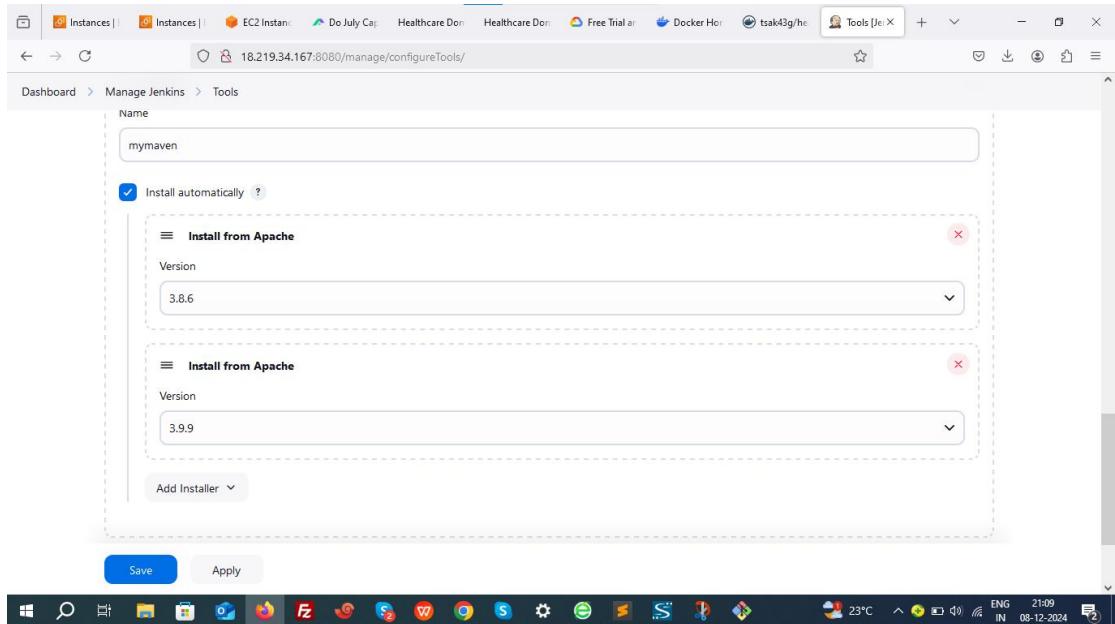
cat /var/lib/jenkins/secrets/initialAdminPassword



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.

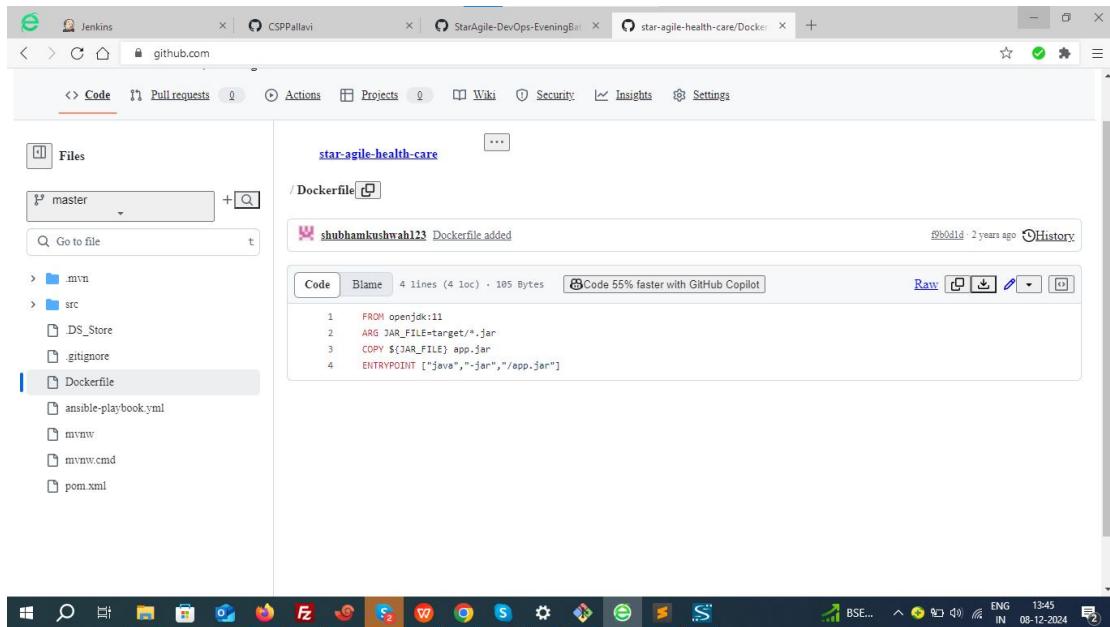
```
FROM openjdk:11
```

```
ARG JAR_FILE=target/*.jar
```

```
COPY ${JAR_FILE} app.jar
```

```
EXPOSE 8082
```

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

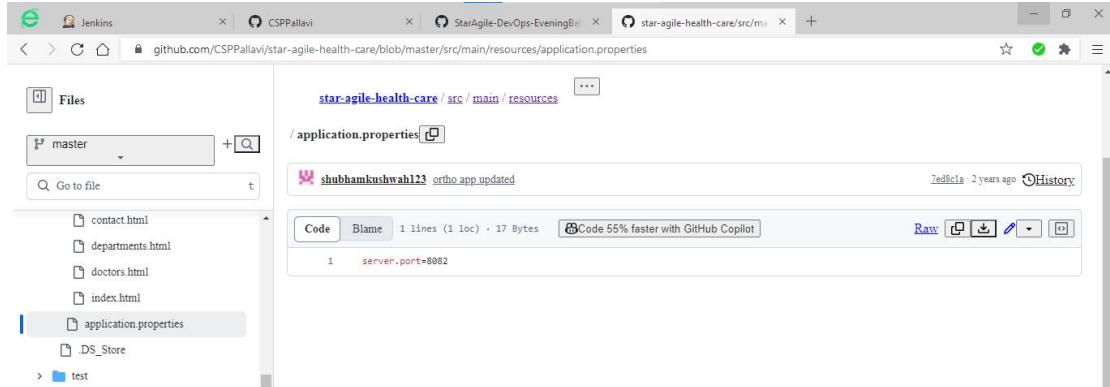


5. Create Pipeline:

Points to be noted before 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

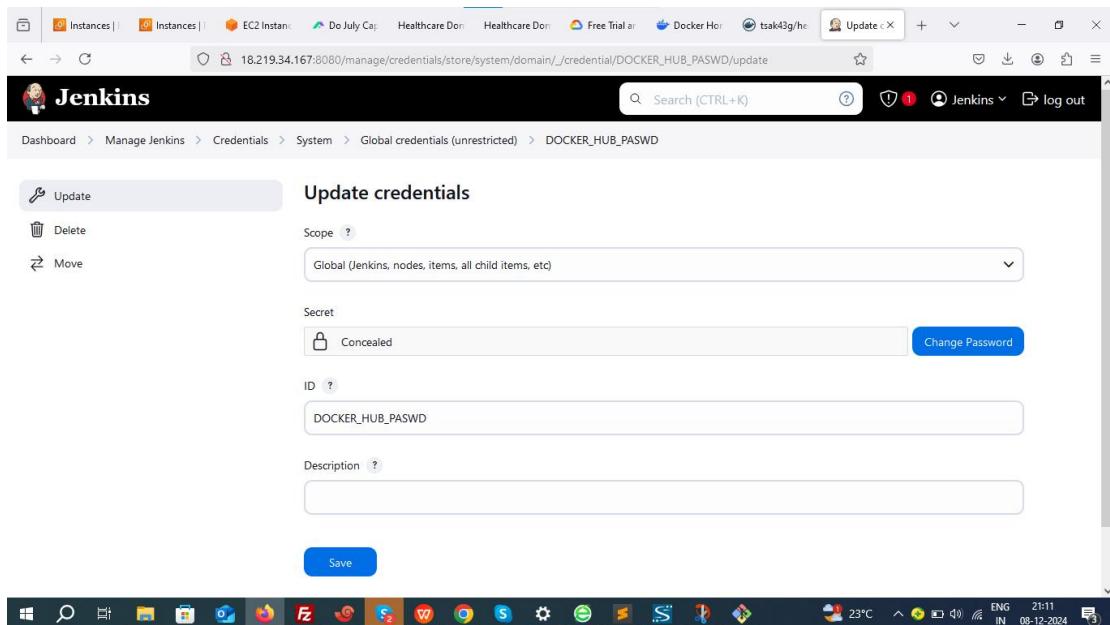
docker run -d -p 8082:8082 myproject1:\$BUILD_NUMBER

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. Before 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'

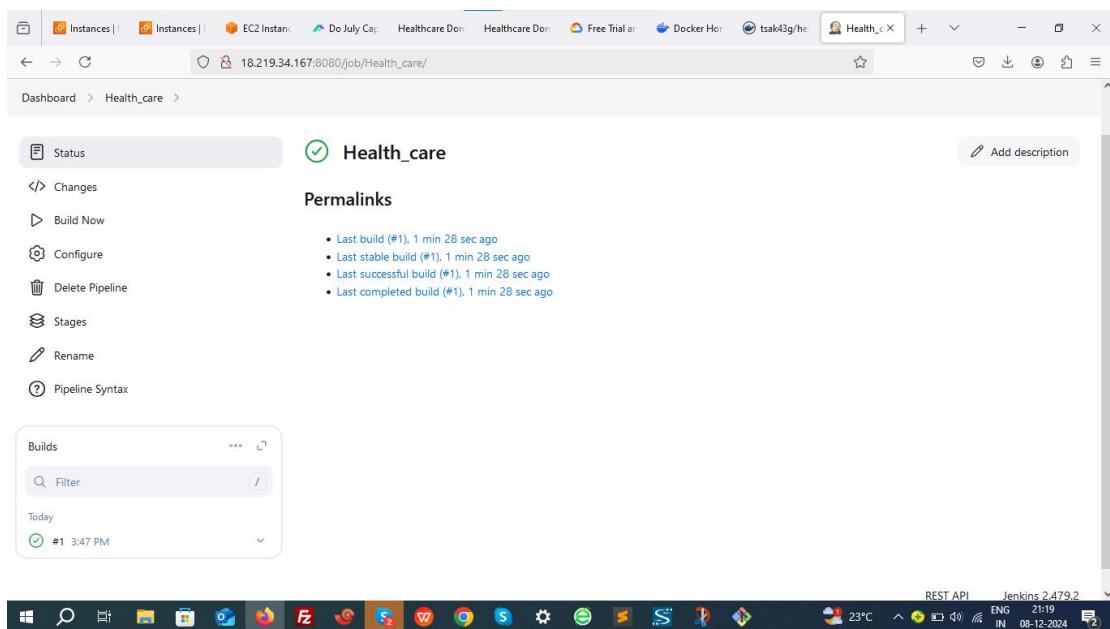
        sh 'docker push
tsak43g/health_care:$BUILD_NUMBER'

    }
}

}
```

- **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.

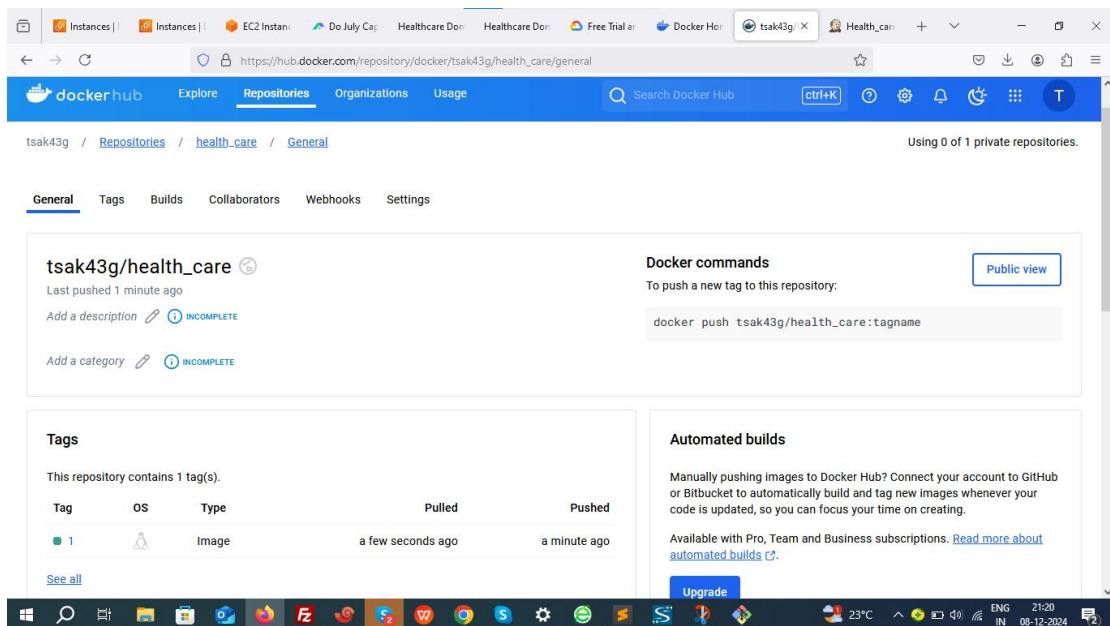
Build the job successfully.



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
```

```
customresourcedefinitions.apirextensions.k8s.io/networksets.crd.projectcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node created
deployment.apps/calico-kube-controllers created
root@ip-172-31-43-86:/home/ubuntu# kubectl get nodes
NAME           STATUS    ROLES   AGE     VERSION
ip-172-31-43-86  NotReady control-plane  44s   v1.29.11
root@ip-172-31-43-86:/home/ubuntu#
```

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

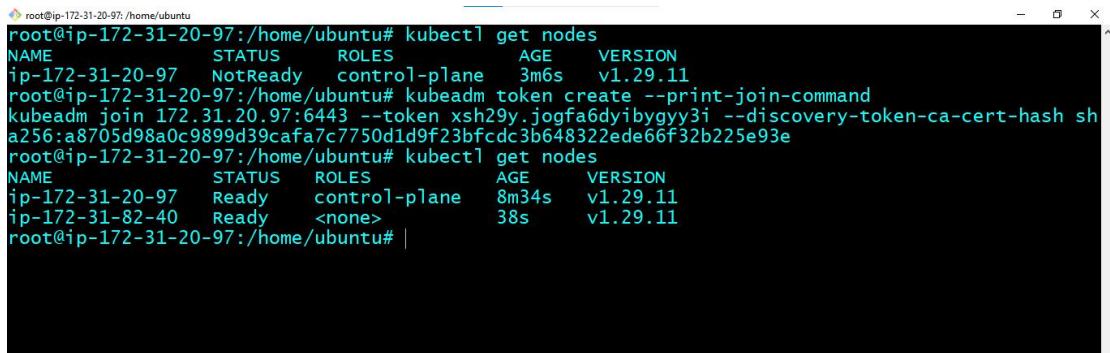
```
kubeadm token create --print-join-command
```

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
```



```

root@ip-172-31-20-97:/home/ubuntu# kubectl get nodes
NAME           STATUS    ROLES      AGE   VERSION
ip-172-31-20-97  NotReady  control-plane  3m6s  v1.29.11
root@ip-172-31-20-97:/home/ubuntu# kubeadm token create --print-join-command
kubeadm join 172.31.20.97:6443 --token xsh29y.jogfa6dyibgyy3i --discovery-token-ca-cert-hash sha256:a8705d98a0c9899d39cafa7c7750d1d9f23bfc3b648322ede66f32b225e93e
root@ip-172-31-20-97:/home/ubuntu# kubectl get nodes
NAME           STATUS    ROLES      AGE   VERSION
ip-172-31-20-97  Ready     control-plane  8m34s  v1.29.11
ip-172-31-82-40  Ready     <none>     38s   v1.29.11
root@ip-172-31-20-97:/home/ubuntu# |

```

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

```

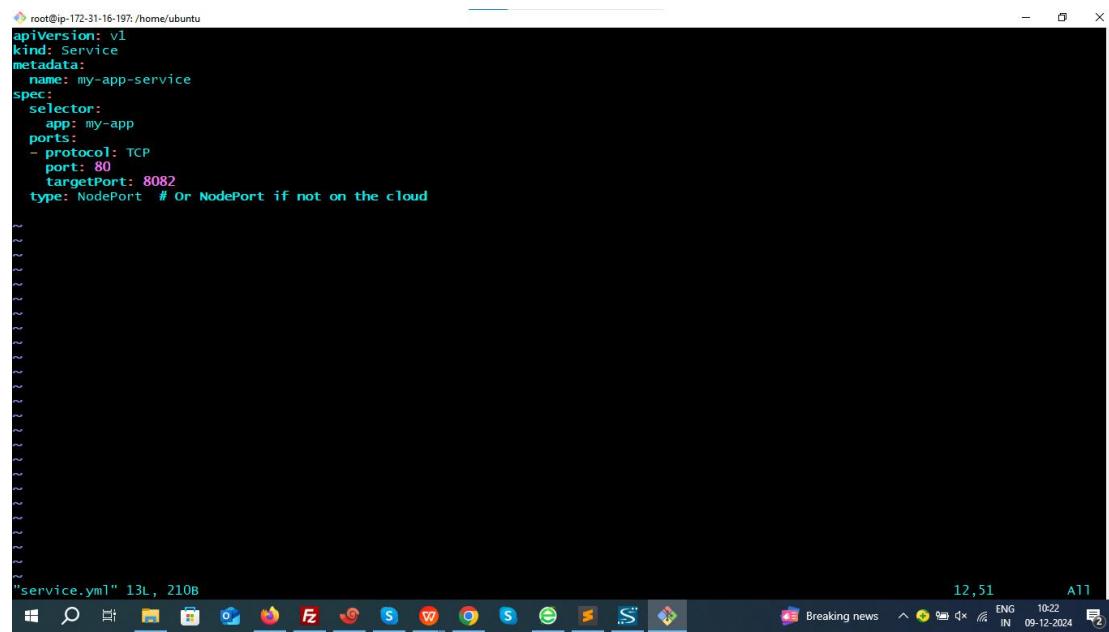
2. Apply the Deployment

```
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
```



The screenshot shows a terminal window on a Linux system (Ubuntu) with a black background and white text. The command `root@ip-172-31-16-197: /home/ubuntu` is at the top. Below it is the YAML configuration for a service. The terminal window has a title bar and a status bar at the bottom showing the date and time.

```
root@ip-172-31-16-197: /home/ubuntu
apiVersion: v1
kind: Service
metadata:
  name: my-app-service
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8082
  type: NodePort # Or NodePort if not on the cloud

"service.yaml" 13L, 210B
```

Apply the service:

```
kubectl apply -f service.yaml
```

5.Verify the Deployment and Service

Check the Pods:



kubectl get pods

```
root@ip-172-31-16-197:/home/ubuntu# kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
my-app-deployment-78b5cd5c4-mlfzc 1/1     Running   0          41m
my-app-deployment-78b5cd5c4-t5t5b 1/1     Running   0          41m
test-pod      1/1     Running   0          61m
root@ip-172-31-16-197:/home/ubuntu#
```

Check the Service:

kubectl get service my-app-service

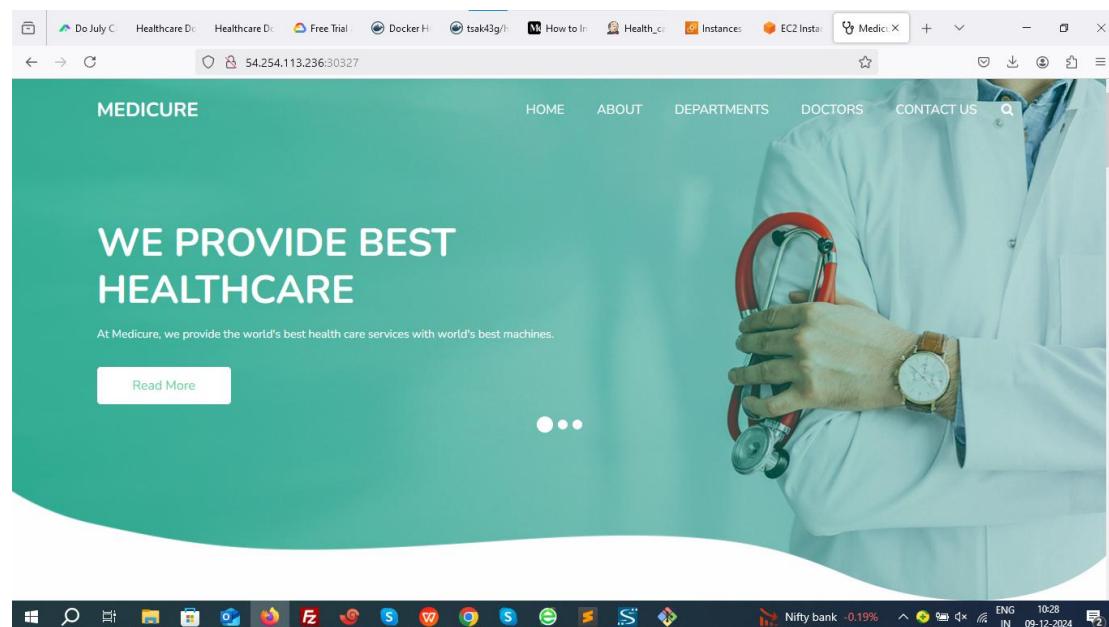
```
root@ip-172-31-16-197:/home/ubuntu# kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
my-app-deployment-78b5cd5c4-mlfzc 1/1     Running   0          41m
my-app-deployment-78b5cd5c4-t5t5b 1/1     Running   0          41m
test-pod      1/1     Running   0          61m
root@ip-172-31-16-197:/home/ubuntu# kubectl get svc
NAME           TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
kubernetes     ClusterIP  10.96.0.1    <none>        443/TCP         66m
my-app-service NodePort    10.103.8.108 <none>        80:30327/TCP   54m
root@ip-172-31-16-197:/home/ubuntu# kubectl get svc my-app-service
NAME           TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
my-app-service NodePort    10.103.8.108 <none>        80:30327/TCP   54m
root@ip-172-31-16-197:/home/ubuntu#
```

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

Access the container.

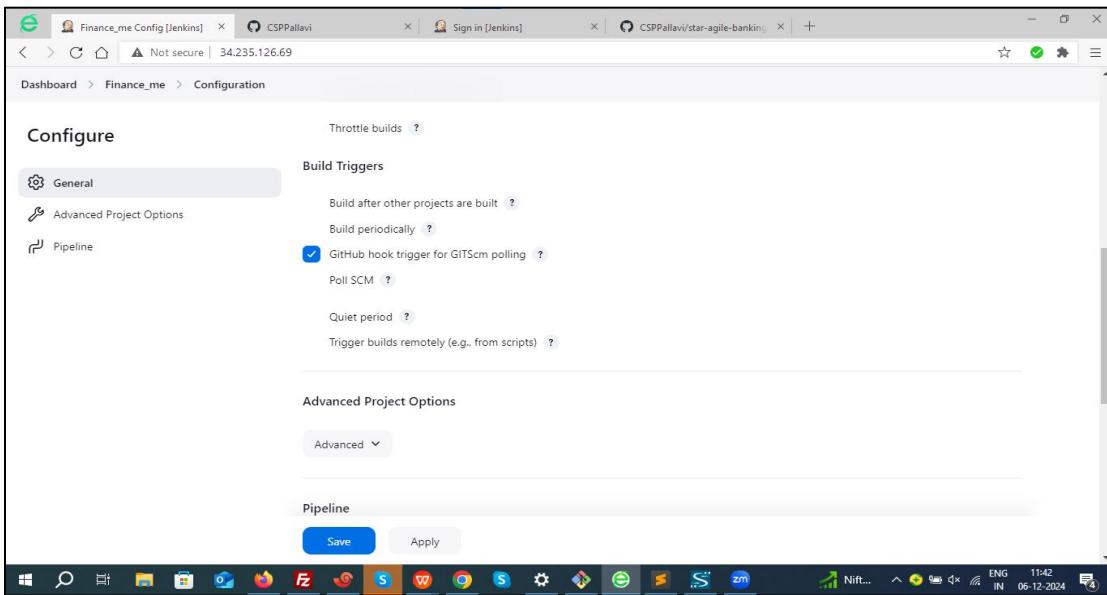
<http://< public ip of worker node>:30996>

<http://13.126.231.60:30996/>



STEP 4: WEBHOOK CONFIGURATION:

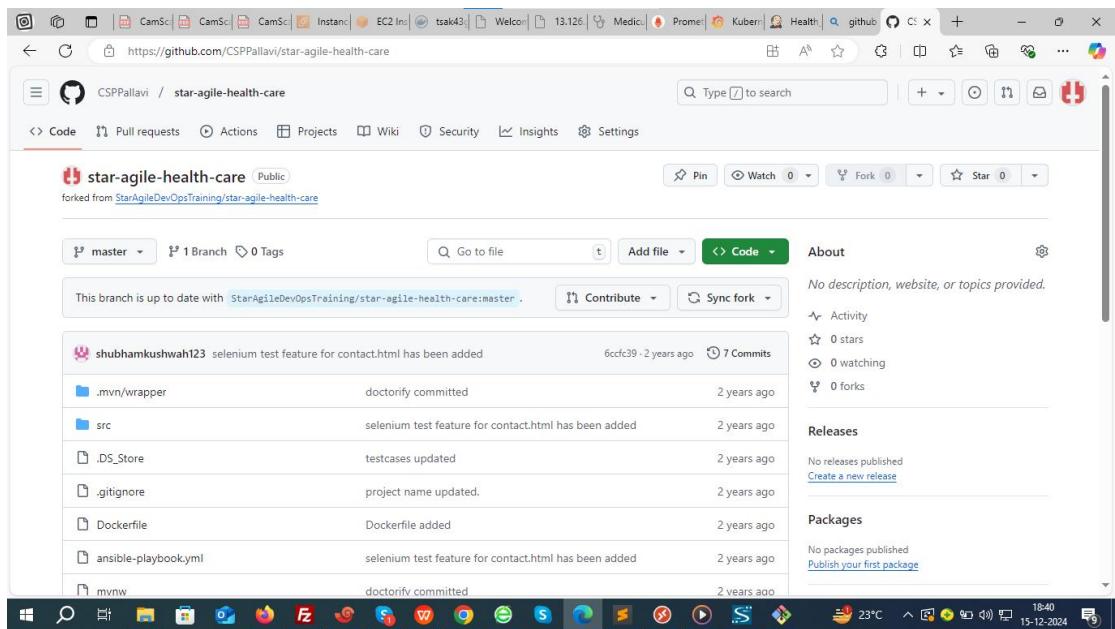
Go to your Jenkins dashboard and add build trigger- **GitHub hook trigger for GITScm polling**



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.

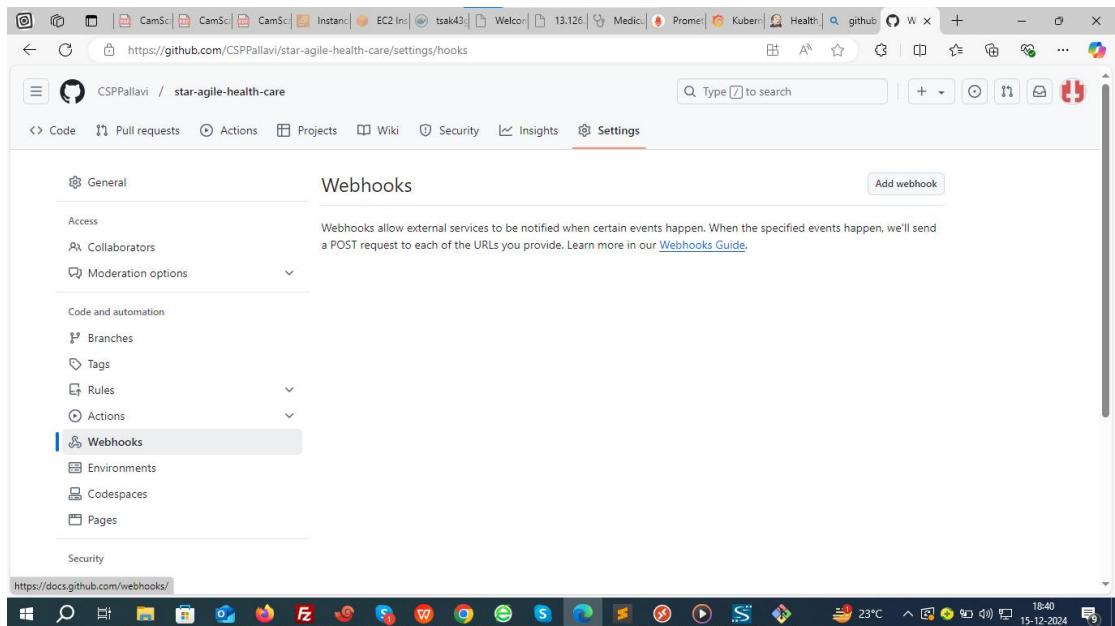
Step 1: Open Your Repository Settings

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

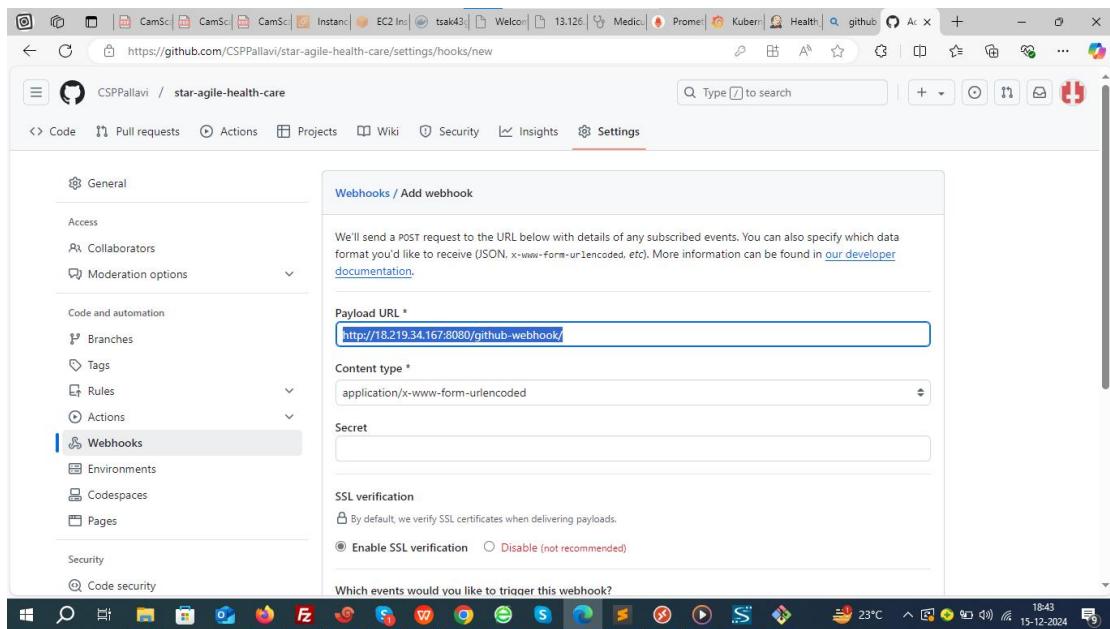
Fill in the webhook details:

Payload URL:

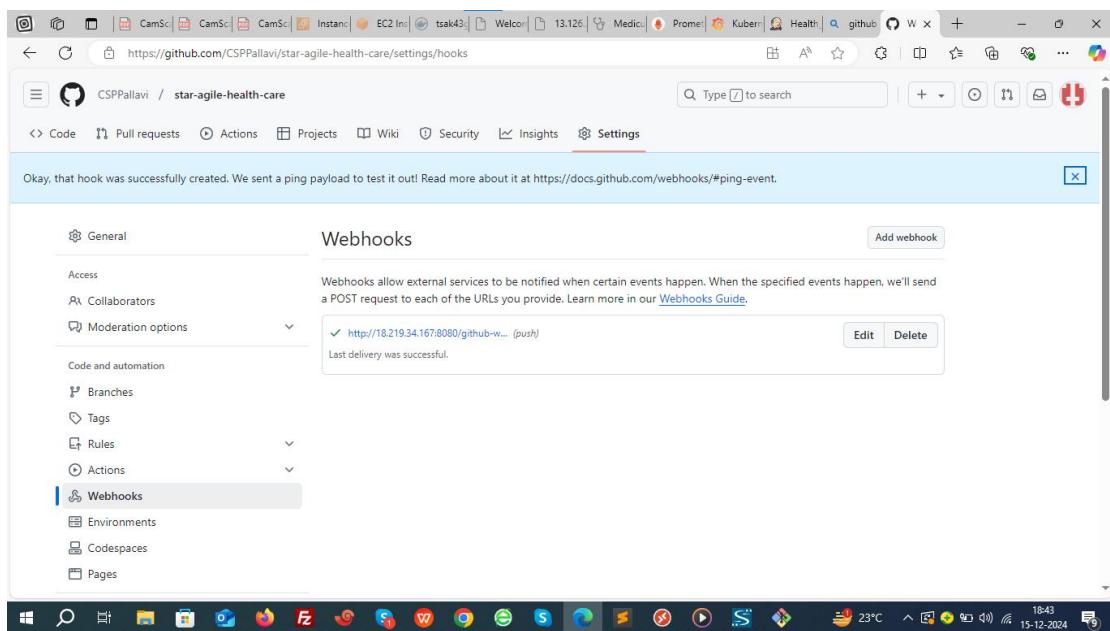
- This is the endpoint that will receive the webhook payload.
- <http://<Jenkins-URL>/github-webhook/>

<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):

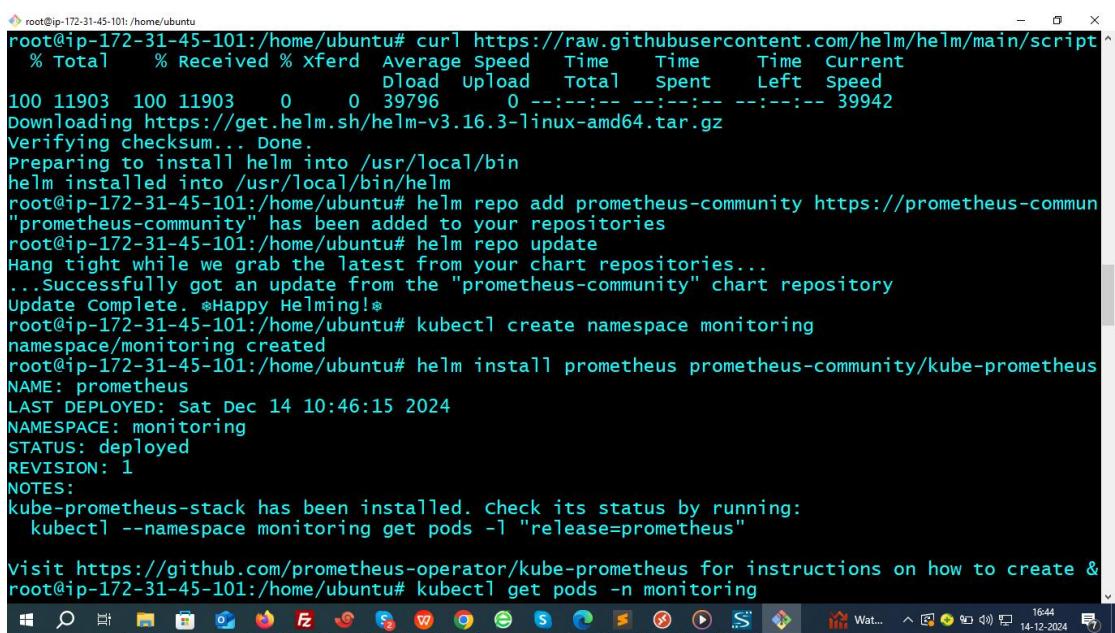
```
curl
```

```
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
```



```
root@ip-172-31-45-101:/home/ubuntu# curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
% Total    % Received % Xferd  Average Speed   Time     Time      Current
          Dload  Upload   Total   Spent    Left  Speed
100 11903  100 11903    0     0  39796      0 --:--:-- --:--:-- --:--:-- 39942
Downloaded https://get.helm.sh/helm-v3.16.3-linux-amd64.tar.gz
Verifying checksum... Done.
Preparing to install helm into /usr/local/bin
helm installed into /usr/local/bin/helm
root@ip-172-31-45-101:/home/ubuntu# helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
"prometheus-community" has been added to your repositories
root@ip-172-31-45-101:/home/ubuntu# helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "prometheus-community" chart repository
Update Complete. *Happy Helm-ing!*
root@ip-172-31-45-101:/home/ubuntu# kubectl create namespace monitoring
namespace/monitoring created
root@ip-172-31-45-101:/home/ubuntu# helm install prometheus prometheus-community/kube-prometheus
NAME: prometheus
LAST DEPLOYED: Sat Dec 14 10:46:15 2024
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace monitoring get pods -l "release=prometheus"

Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create &
root@ip-172-31-45-101:/home/ubuntu# kubectl get pods -n monitoring
```

Install Prometheus and grafana using Helm:

```
kubectl create namespace monitoring
```

```
# Create monitoring namespace
```

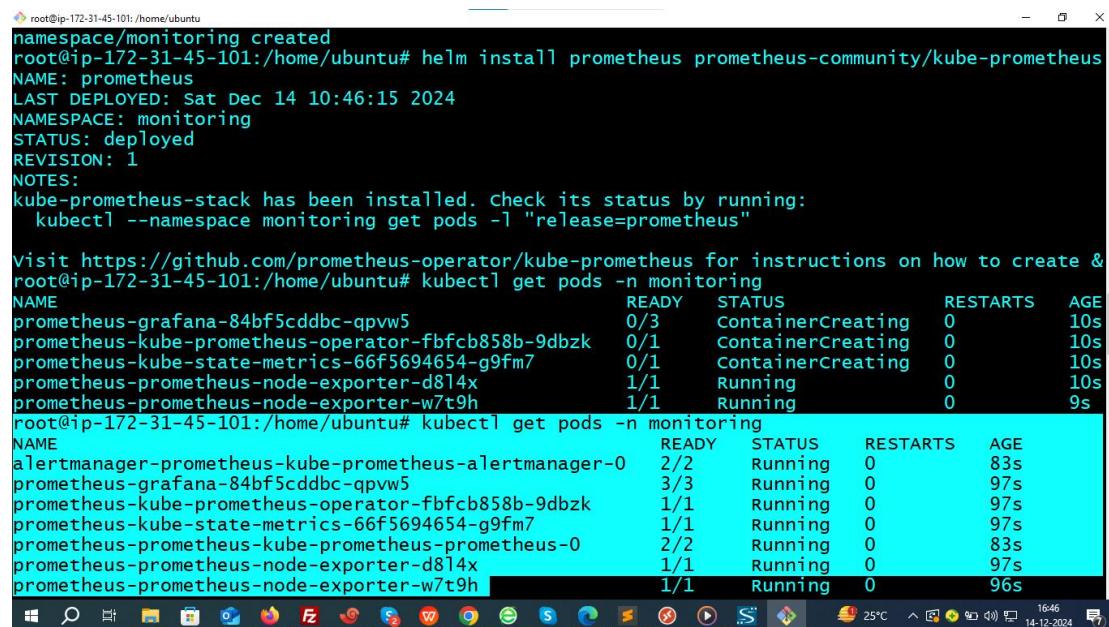
```
helm install prometheus prometheus-community/kube-prometheus-stack --namespace monitoring
```

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
```



```
root@ip-172-31-45-101:/home/ubuntu
namespace/monitoring created
root@ip-172-31-45-101:/home/ubuntu# helm install prometheus prometheus-community/kube-prometheus
NAME: prometheus
LAST DEPLOYED: Sat Dec 14 10:46:15 2024
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace monitoring get pods -l "release=prometheus"

Visit https://github.com/prometheus-operator/kube-prometheus for instructions on how to create &
root@ip-172-31-45-101:/home/ubuntu# kubectl get pods -n monitoring
NAME                               READY   STATUS        RESTARTS   AGE
prometheus-grafana-84bf5cddb-qpvw5   0/3    ContainerCreating   0          10s
prometheus-kube-prometheus-operator-fbfcb858b-9dbzk   0/1    ContainerCreating   0          10s
prometheus-kube-state-metrics-66f5694654-g9fm7   0/1    ContainerCreating   0          10s
prometheus-prometheus-node-exporter-d814x      1/1    Running       0          10s
prometheus-prometheus-node-exporter-w7t9h      1/1    Running       0          9s
root@ip-172-31-45-101:/home/ubuntu# kubectl get pods -n monitoring
NAME                               READY   STATUS        RESTARTS   AGE
alertmanager-prometheus-kube-prometheus-alertmanager-0  2/2    Running       0          83s
grafana-grafana-84bf5cddb-qpvw5      3/3    Running       0          97s
prometheus-kube-prometheus-operator-fbfcb858b-9dbzk   1/1    Running       0          97s
prometheus-kube-state-metrics-66f5694654-g9fm7   1/1    Running       0          97s
prometheus-prometheus-kube-prometheus-prometheus-0  2/2    Running       0          83s
prometheus-prometheus-node-exporter-d814x      1/1    Running       0          97s
prometheus-prometheus-node-exporter-w7t9h      1/1    Running       0          96s
```

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):

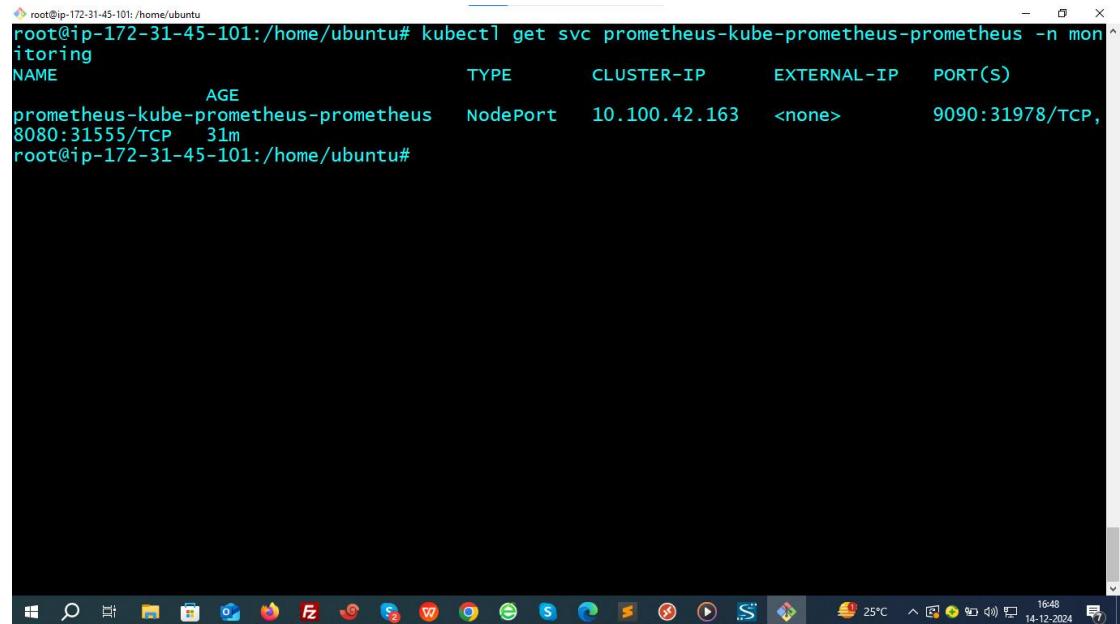
```
kubectl port-forward svc/prometheus-kube-prometheus-prometheus -n monitoring 9090:9090
```

This command creates service type cluster IP

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

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

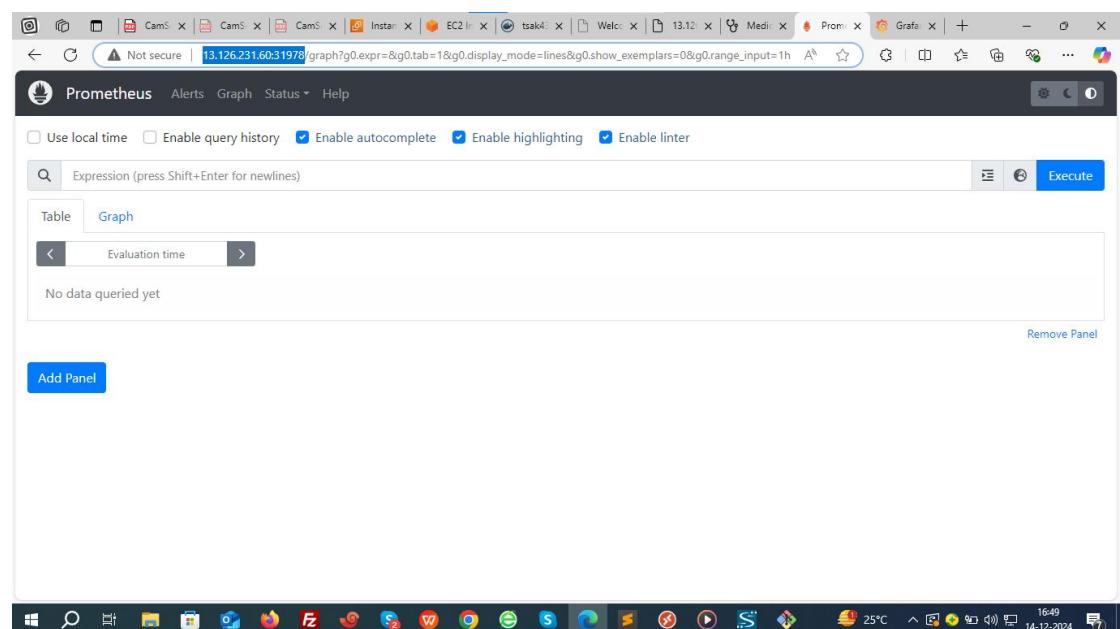
Check NodePort:



```
root@ip-172-31-45-101:/home/ubuntu# kubectl get svc prometheus-kube-prometheus-prometheus -n monitoring
NAME                           TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)
prometheus-kube-prometheus-prometheus   NodePort  10.100.42.163  <none>        9090:31978/TCP,
8080:31555/TCP   31m
root@ip-172-31-45-101:/home/ubuntu#
```

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
```

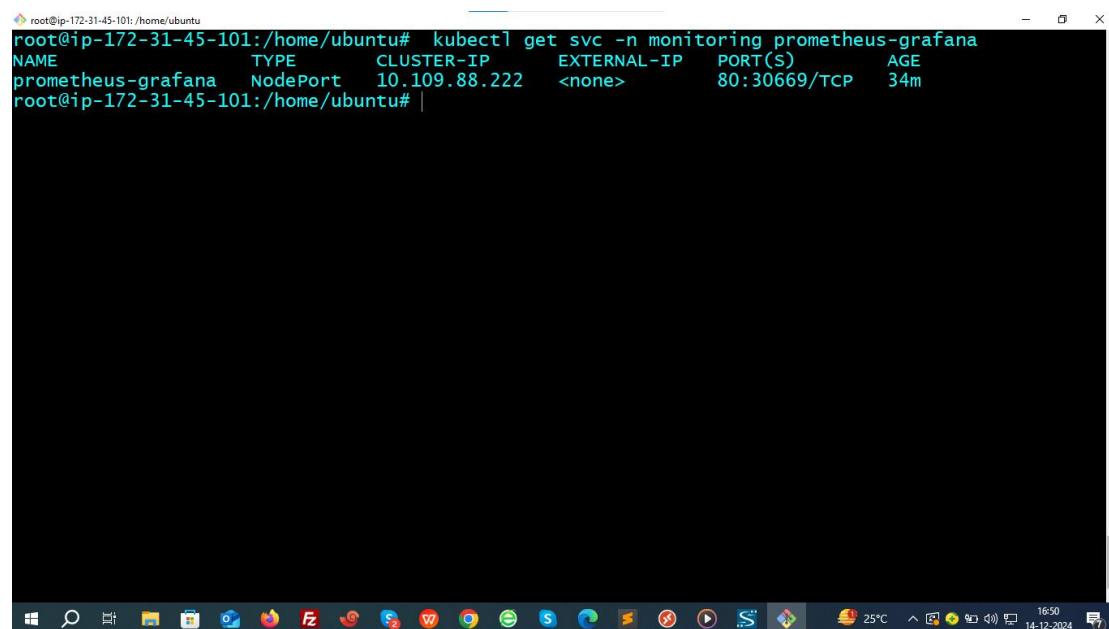
This command again creates service type cluster IP

Below is the command to switch 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
```

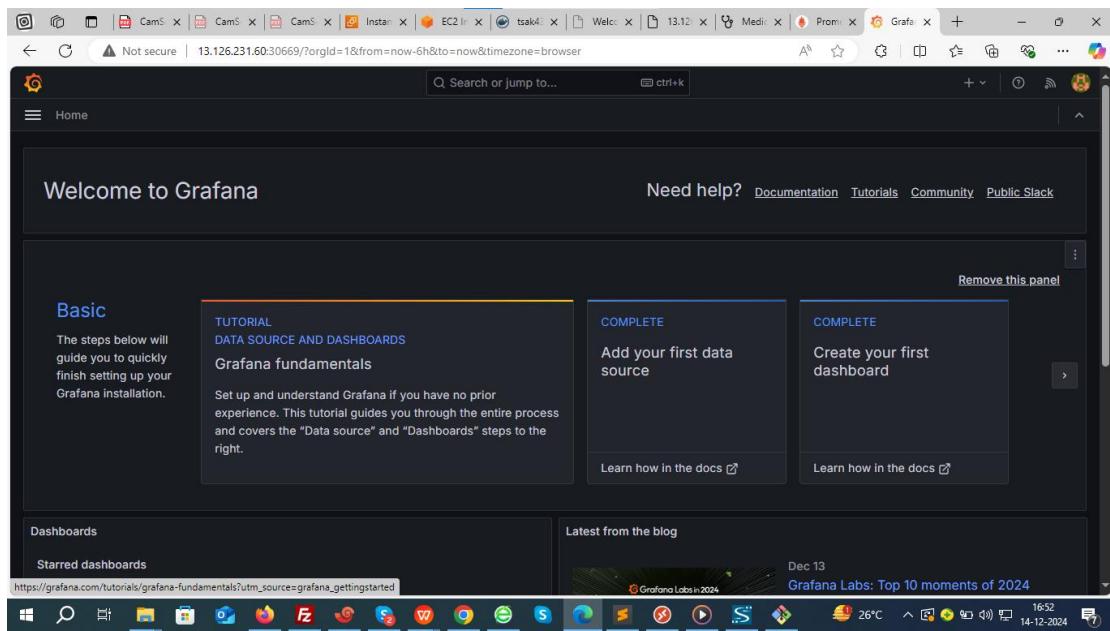


NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
prometheus-grafana	NodePort	10.109.88.222	<none>	80:30669/TCP	34m

<http://13.126.231.60:30669/>

The default login credentials are:

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



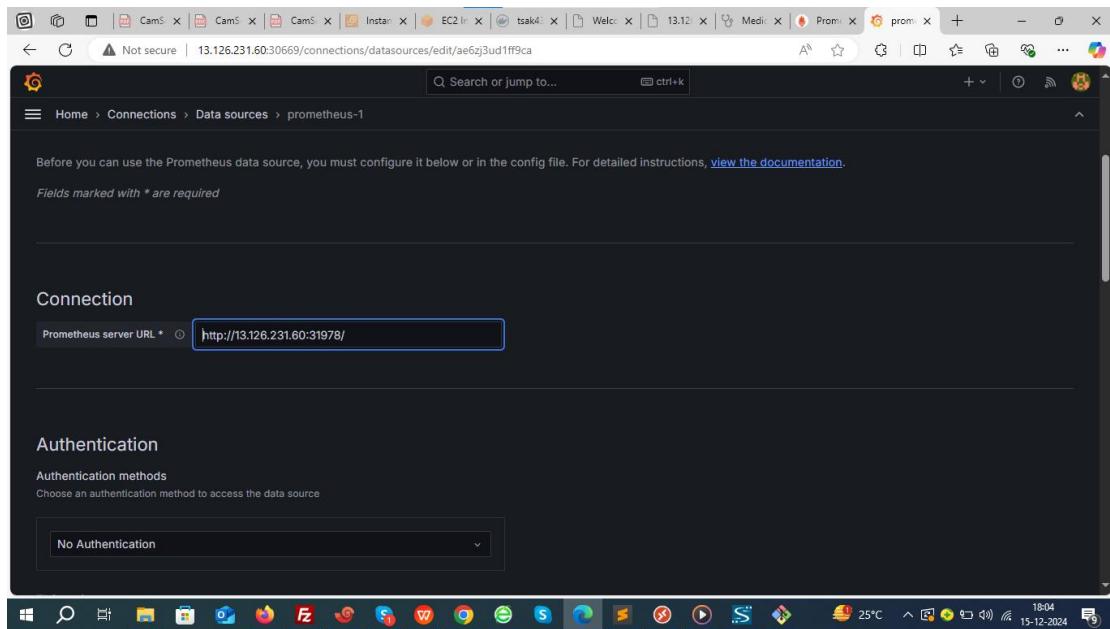
5. Configure Grafana with Prometheus Data Source

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/>

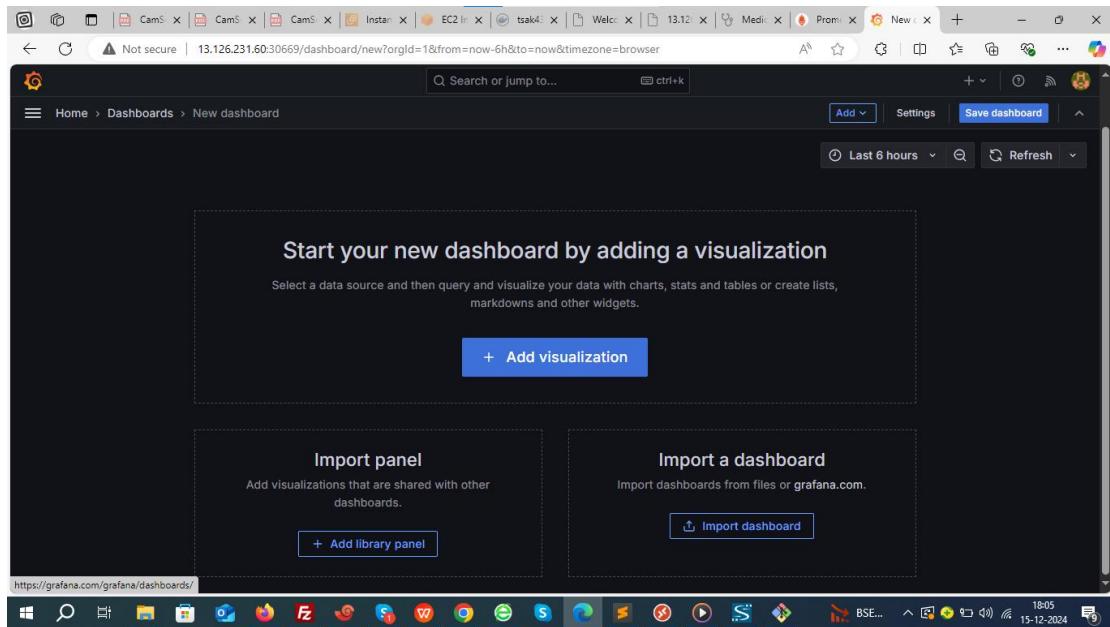
Click **Save & Test** to ensure the connection is successful.



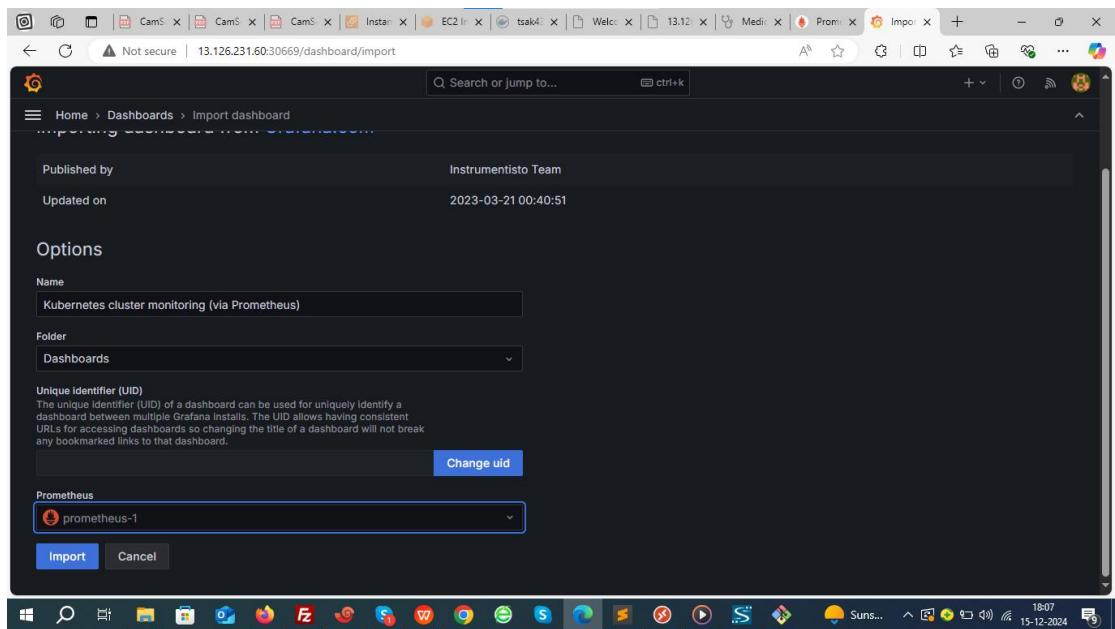
6. Import Grafana Dashboards

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.

