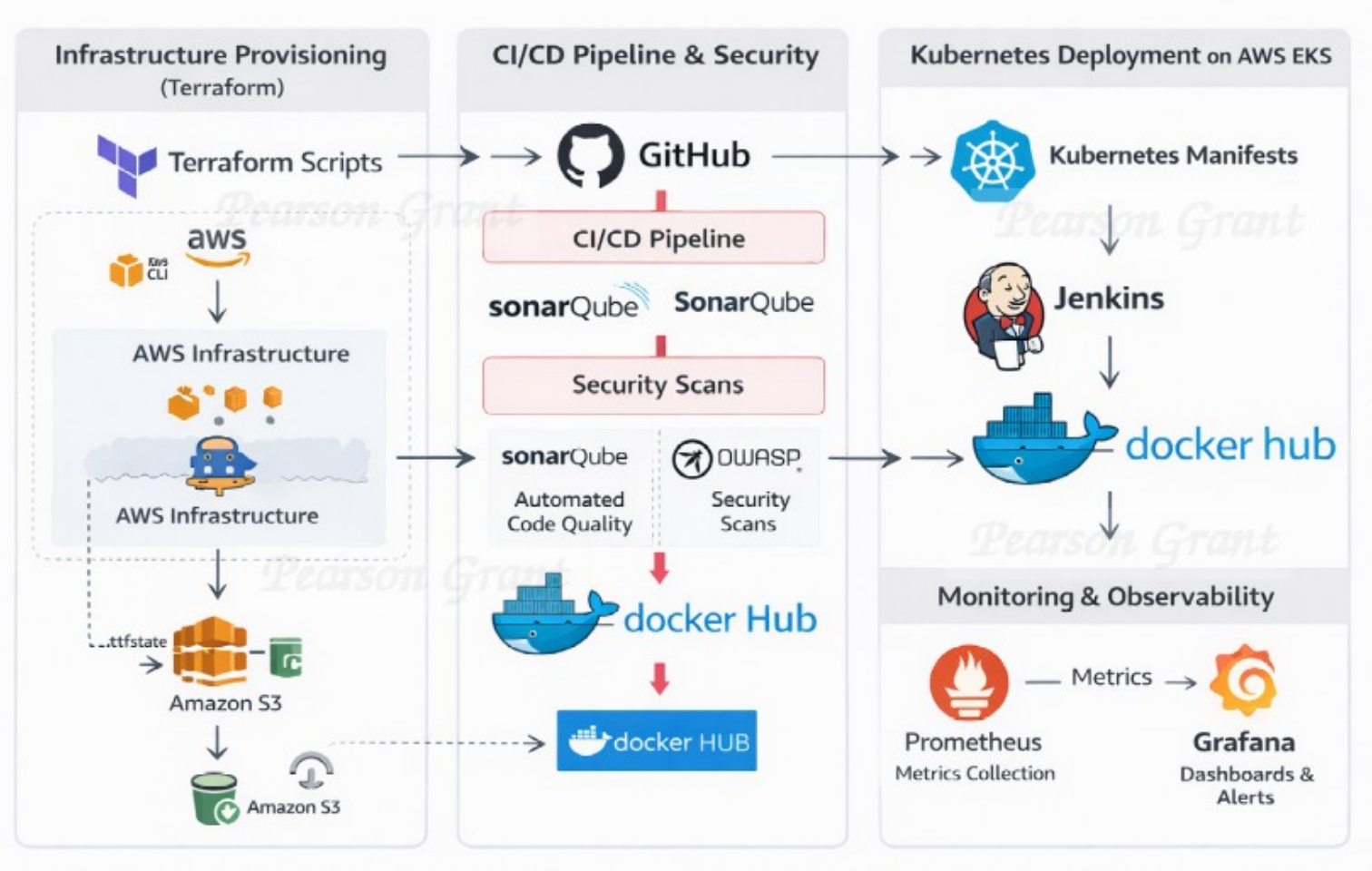
Deploying a ChatGPT Clone on AWS EKS Using Terraform, Jenkins, and Prometheus/Grafana



Introduction

In this project, I designed and implemented an end-to-end DevOps solution to deploy a ChatGPT-like application on AWS EKS. The implementation covers infrastructure provisioning with Terraform, CI/CD automation using Jenkins, containerization with Docker, Kubernetes-based deployment, security and quality enforcement, and full observability using Prometheus and Grafana.

This article documents a complete, step-by-step DevOps implementation, focusing not only on what was deployed, but on *how and why* each decision was made. The structure reflects real-world DevOps problem-solving workflows and is intended for technical readers, hiring managers, and recruiters evaluating practical cloud and DevOps experience.

Project Objectives

- Provision cloud infrastructure using Infrastructure as Code (Terraform)
- Implement a secure CI/CD pipeline with Jenkins
- Deploy the application on Kubernetes (EKS)
- Apply code quality and security scanning practices
- Implement monitoring and observability using Prometheus and Grafana
- Ensure proper resource cleanup to control cloud costs

Technology Stack

- **Cloud Provider:** AWS
- **IaC:** Terraform
- **CI/CD:** Jenkins
- **Containerization:** Docker
- **Orchestration:** Kubernetes (Amazon EKS)
- **Code Quality:** SonarQube
- **Security:** OWASP Dependency Check, Trivy
- **Monitoring:** Prometheus, Node Exporter, Grafana
- **Application:** Node.js-based ChatGPT UI

Prerequisites

- Active AWS account
- Basic knowledge of Terraform, Jenkins, Docker, and Kubernetes
- AWS CLI installed locally
- clone this repo: https://github.com/PearsonGrant24/Chat-gpt-clone-deployment.git

---

PHASE 1: Local Setup + AWS Foundations

Step 1.1 – AWS Account Basics (Very Important)

You need:

- An AWS account
- IAM user (not root)
- Programmatic access

IAM Permissions (Minimum Required)

Your IAM user should have:

- `AdministratorAccess` (for learning projects)

If you don't know how to create an IAM user:

A: Log in as ROOT (One Time Only)

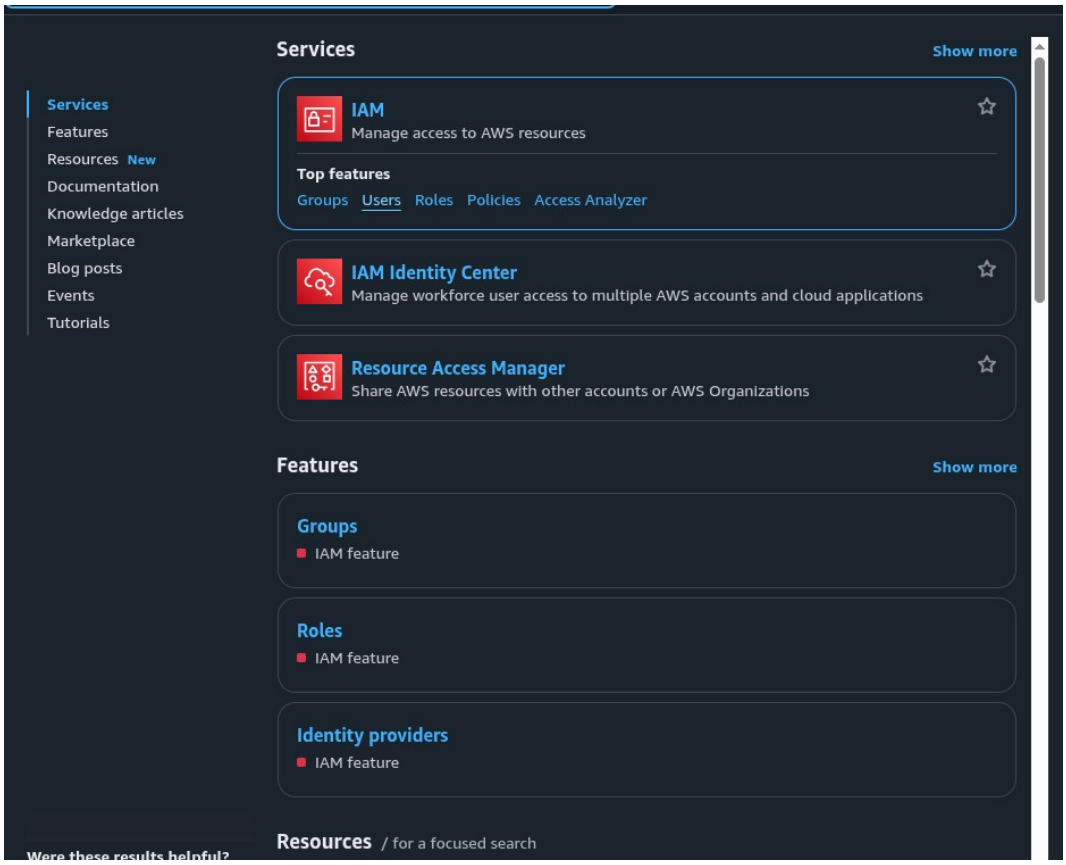
1. Go to **AWS Console**
2. Login using **root email + password**
3. Complete MFA if enabled

This is the **only time** we'll use root.

B: Create IAM User

Navigate

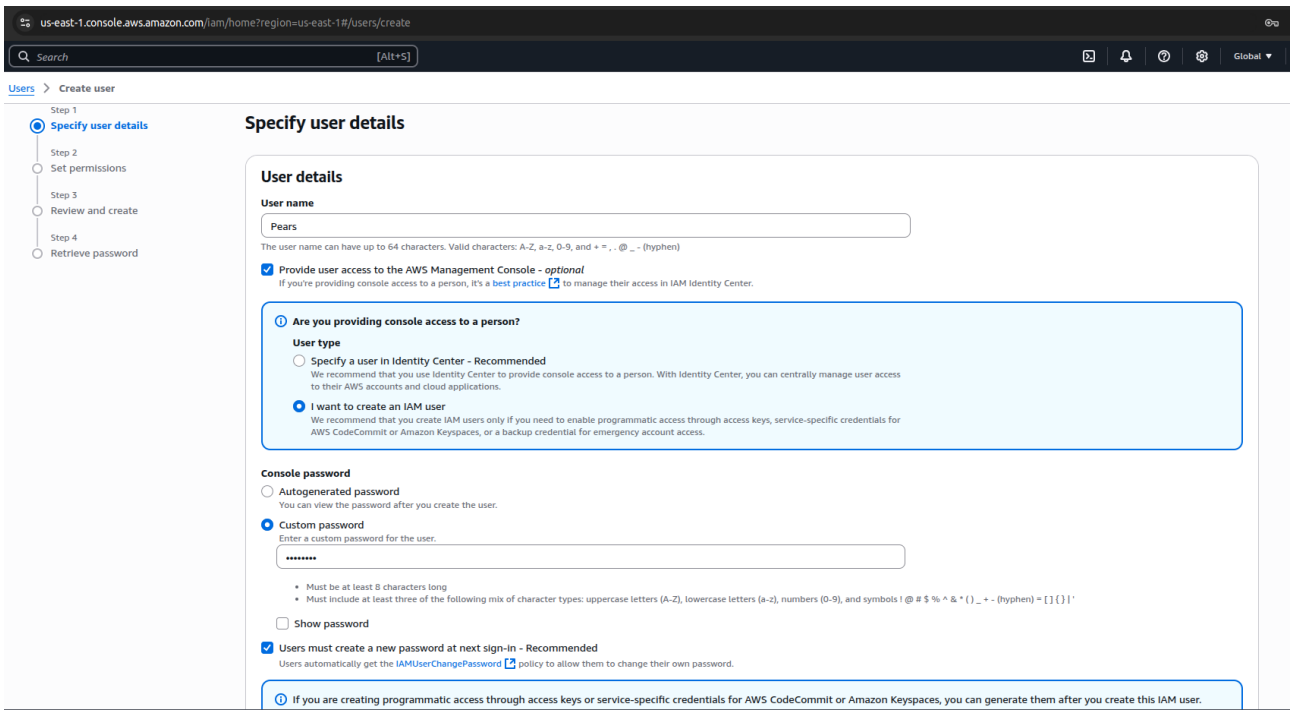
```
AWS Console → IAM → Users → Create IAM user
```



User Details

- **Enter user name:** `Pears;`
- Check: **Provide user access to the AWS Management Console**

Click **Next**



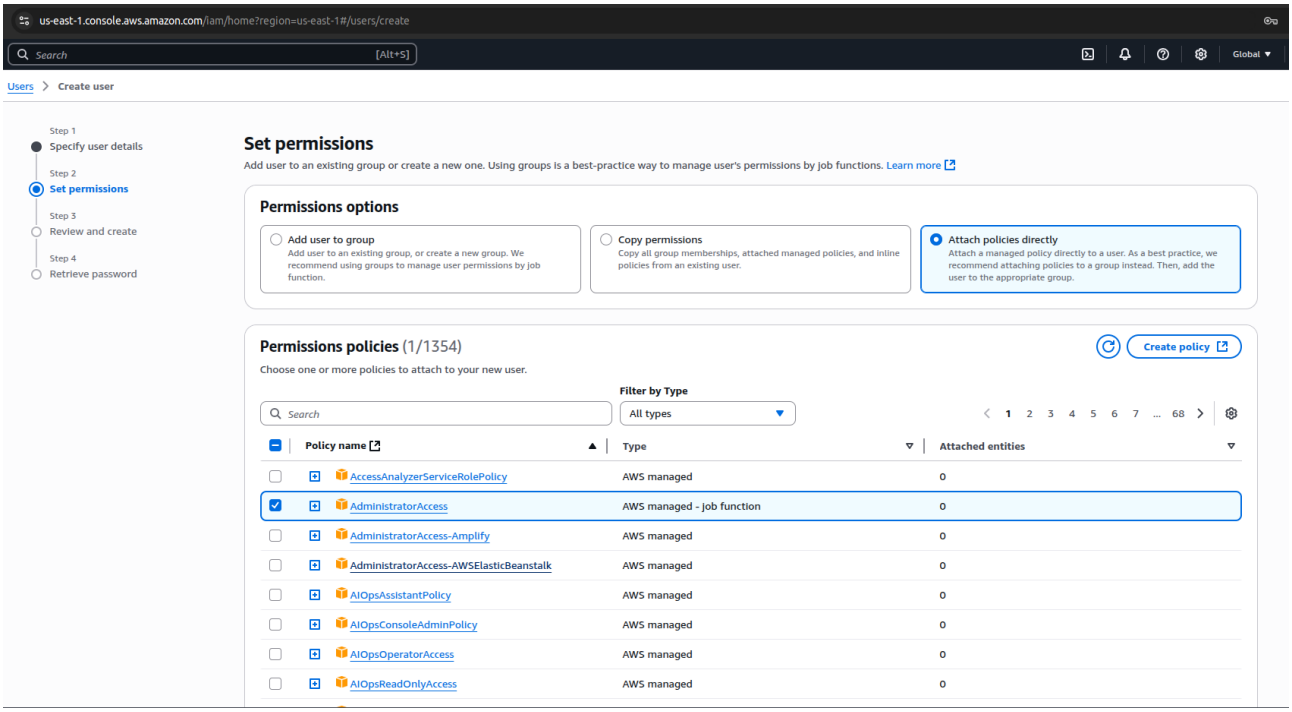
C: Permissions (IMPORTANT)

Choose:

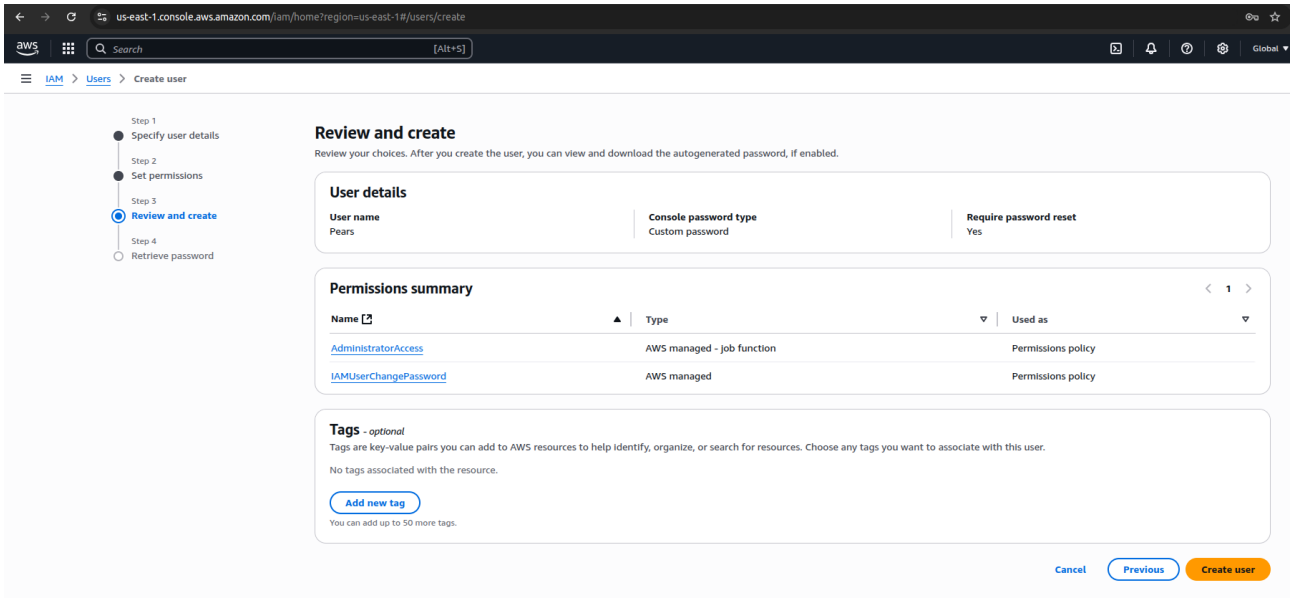
`Attach policies directly`

Select:

`AdministratorAccess`

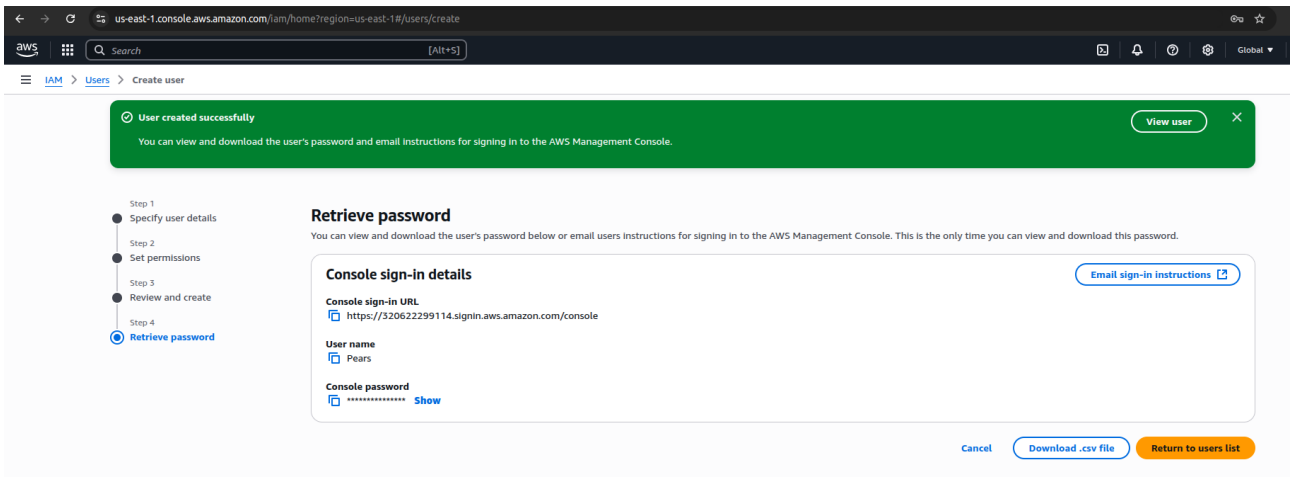


.



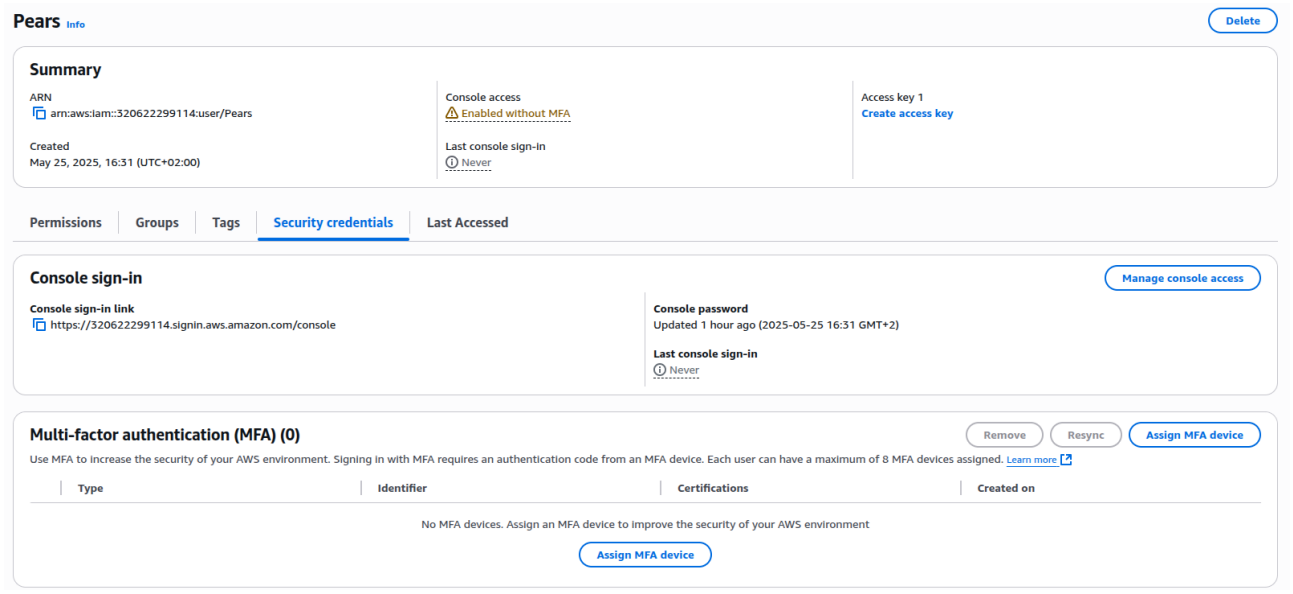
This is OK for learning projects
(Production would be more restricted)

Click **Next → Create user**

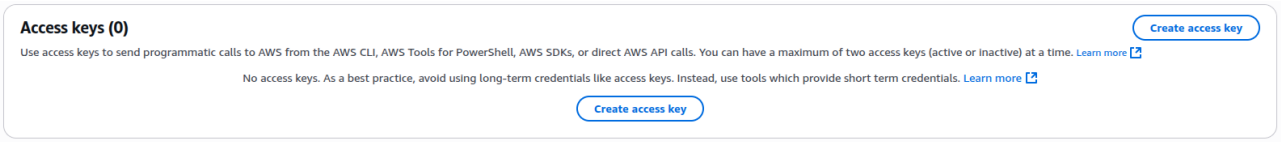


D: Create Access Keys (Programmatic Access)

1. Open `devops-user , we named Pears`
2. Go to **Security credentials**



3. Click **Create access key**



4. Choose:

   `Command Line Interface (CLI)`

.

5. Click **Create**

SAVE THESE IMMEDIATELY:

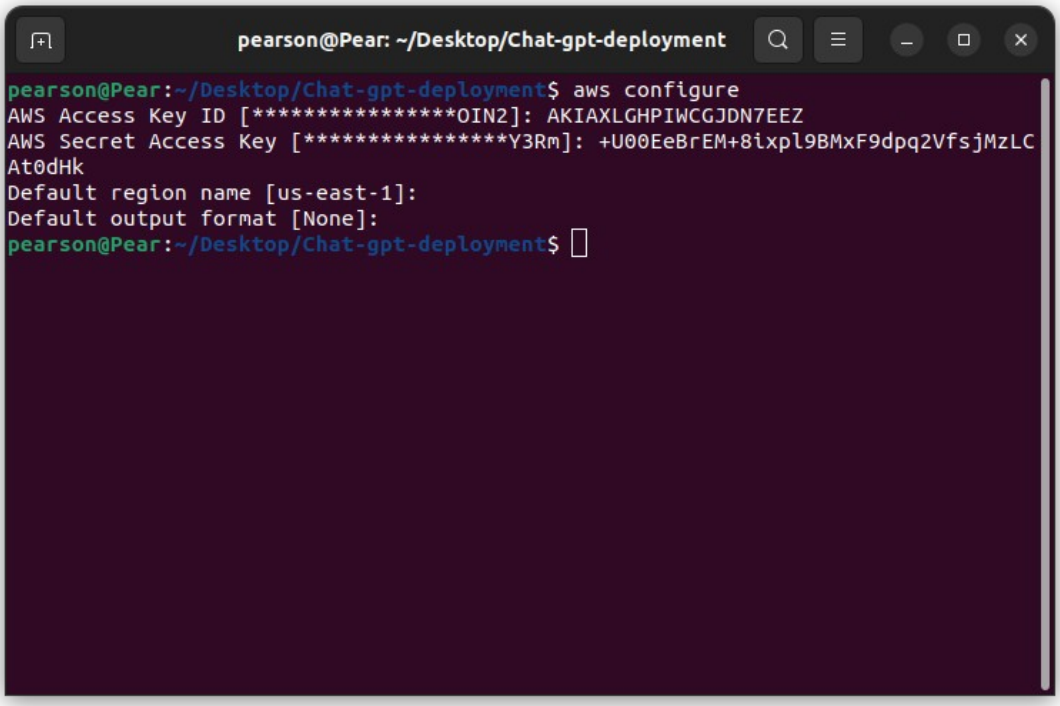
- Access Key ID
- Secret Access Key

You will NOT see the secret again.

E: Configure AWS CLI on Linux

On your Linux terminal:

```
aws configure
```



Enter:

```
AWS Access Key ID:     <your-access-key>
AWS Secret Access Key: <your-secret-key>
Default region name:  us-east-1
Default output format: json
```

F: Verify IAM Setup

Run:

```
aws sts get-caller-identity
```

Expected Output

```
{
  "Account": "123456789012",
  "Arn": "arn:aws:iam::123456789012:user/devops-user",
  "UserId": "AIDA..."
}
```

This confirms:

- IAM user works

- Credentials are correct
- AWS CLI is connected

Step 1.2 – Install Required Tools (Local Machine)

AWS CLI

Check:

```
aws --version
```

If not installed:

```
# Linux
sudo apt install awscli -y
```

Terraform

Check:

```
terraform version
```

If not installed:

```
# Linux
sudo apt install terraform -y
```

---

kubectl (We won’t use it yet, but install now)

```
kubectl version --client
```

Install if missing:

```
sudo apt install kubectl -y
```

Step 1.3 – Configure AWS Credentials

This connects **your machine → AWS**

Run:

```
aws configure
```

Enter:

```
AWS Access Key ID:     <your-key>
AWS Secret Access Key: <your-secret>
Default region name:  us-east-1
Default output format: json
```

Verify Connection

```
aws sts get-caller-identity
```

Expected output:

```
{
  "Account": "123456789",
  "Arn": "arn:aws:iam::123456789:user/your-user",
  "UserId": "AIDA..."
}
```

PHASE 2: Create Infrastructure as Code using Terraform

Step 2.1 – Terraform Code Review (CRITICAL)

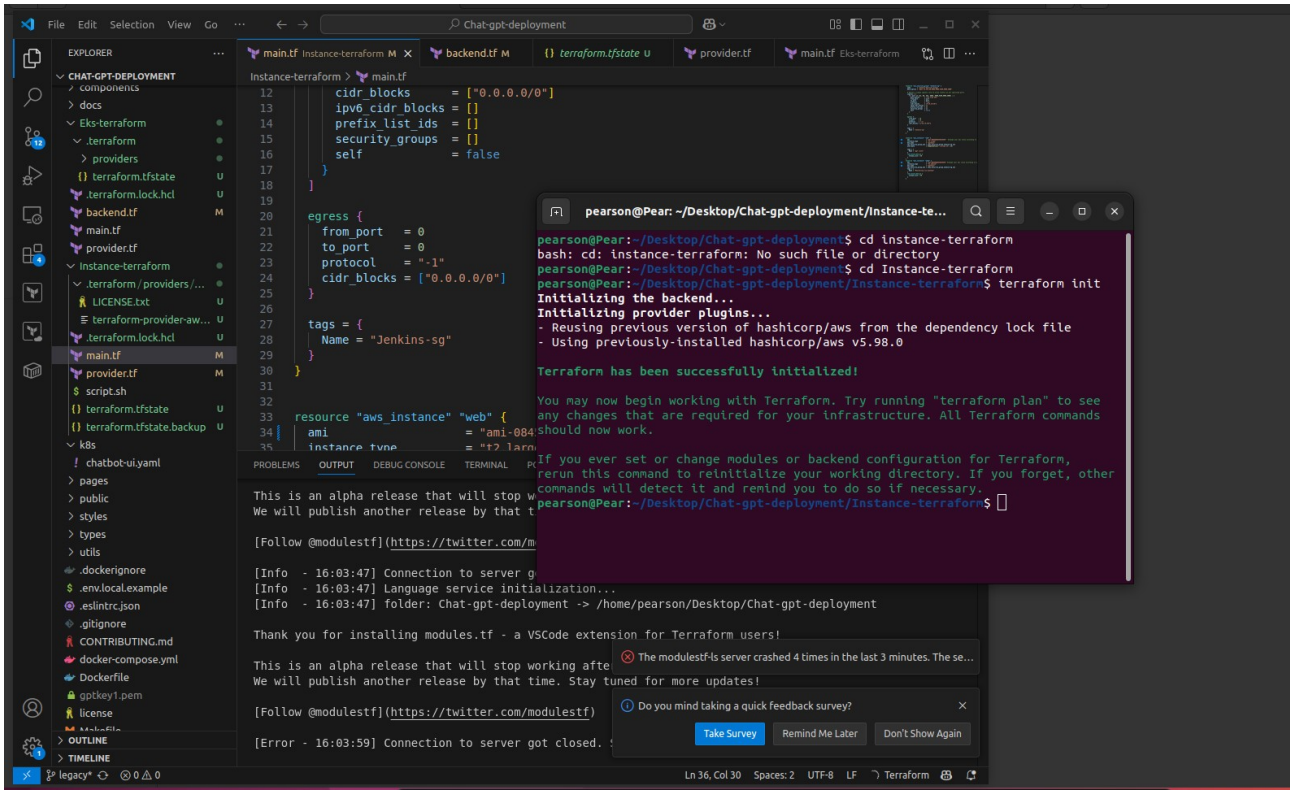
Now let’s make sure your Terraform code is safe.

Go to your terraform folder

```
cd terraform
ls
```

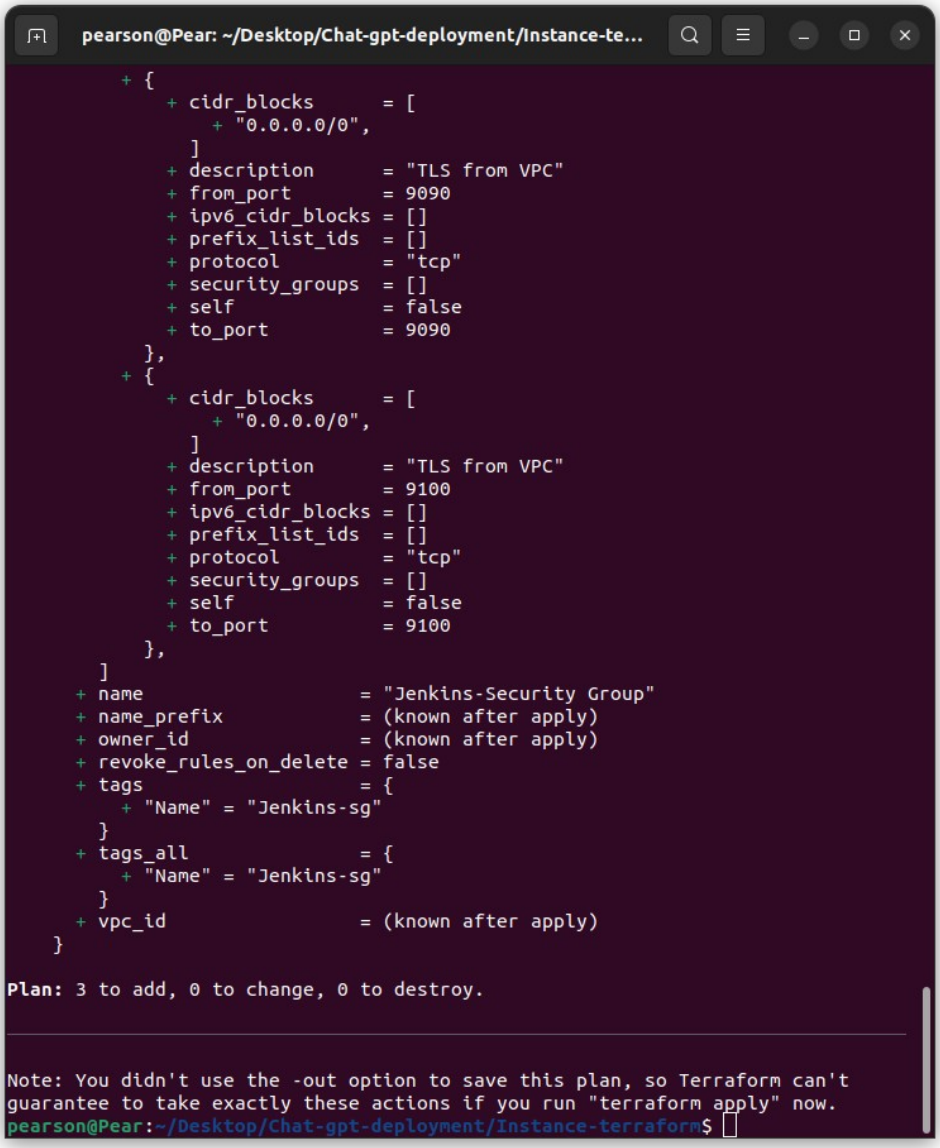
Run Terraform Init

```
terraform init
```



This downloads providers

Run Terraform Plan

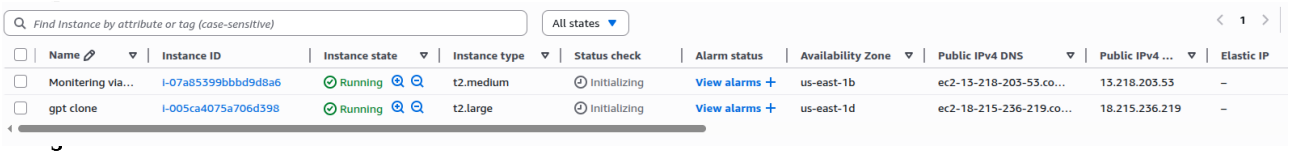


This shows:

- What AWS resources will be created
- How many EC2s, subnets, IAM roles

Terraform apply --auto-approve

gpt instance got created by terraform with the following configuration



s

e

n

t

e

r

o

r

c

l

i

c

k

-

t

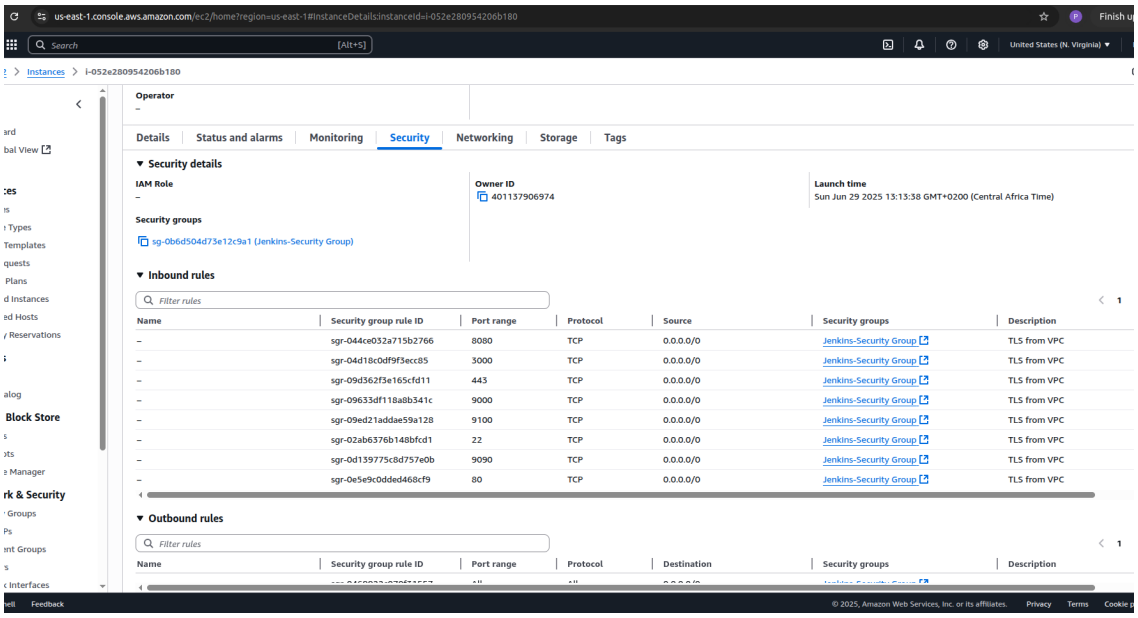
o

v

i

e

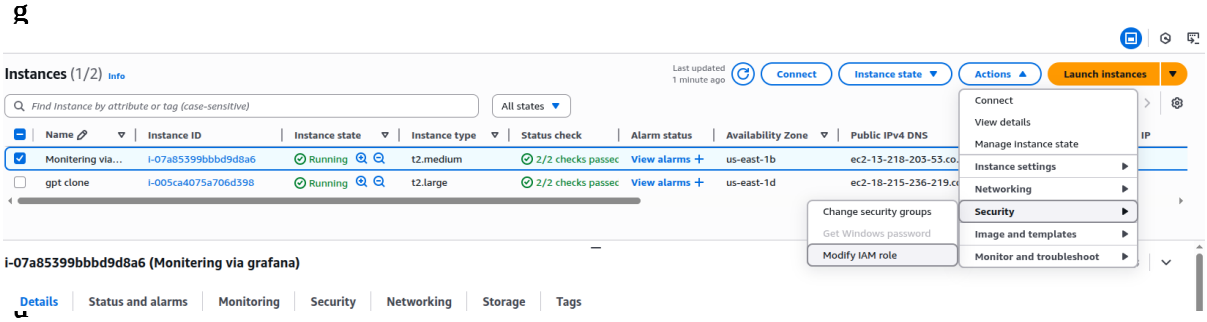
w



Attach IAM role with your EC2

1.go to EC2 section

2.click on actions → security → modify IAM role option



d

l

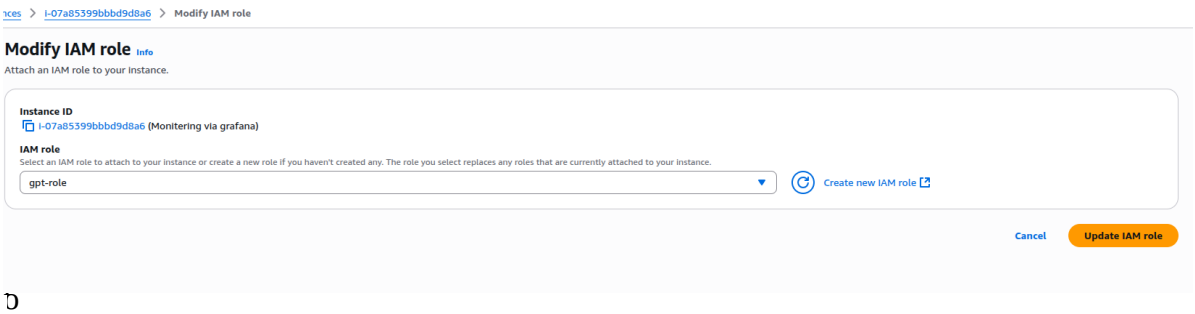
s

i

z

e

3. choose the role from dropdown and click on update IAM role



Phase 3 : Setup Sonarqube and Jenkins

STEP 3.1 Sonarqube

copy the public ip of your machine

Access sonarqube

`http://publicip:9000`

Username: admin

Password: admin

Update your password

Welcome to Sonarqube interface

STEP 3.2 — Setup Jenkins

Access Jenkins

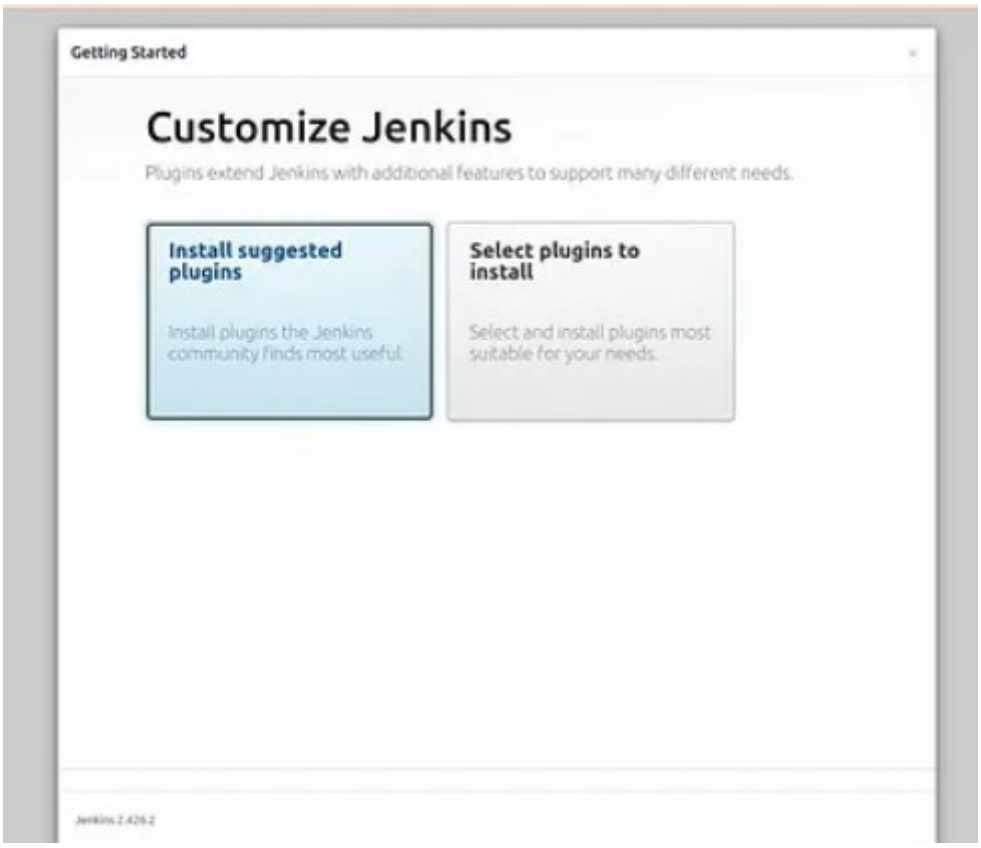
`http://<public_ip>:8080`

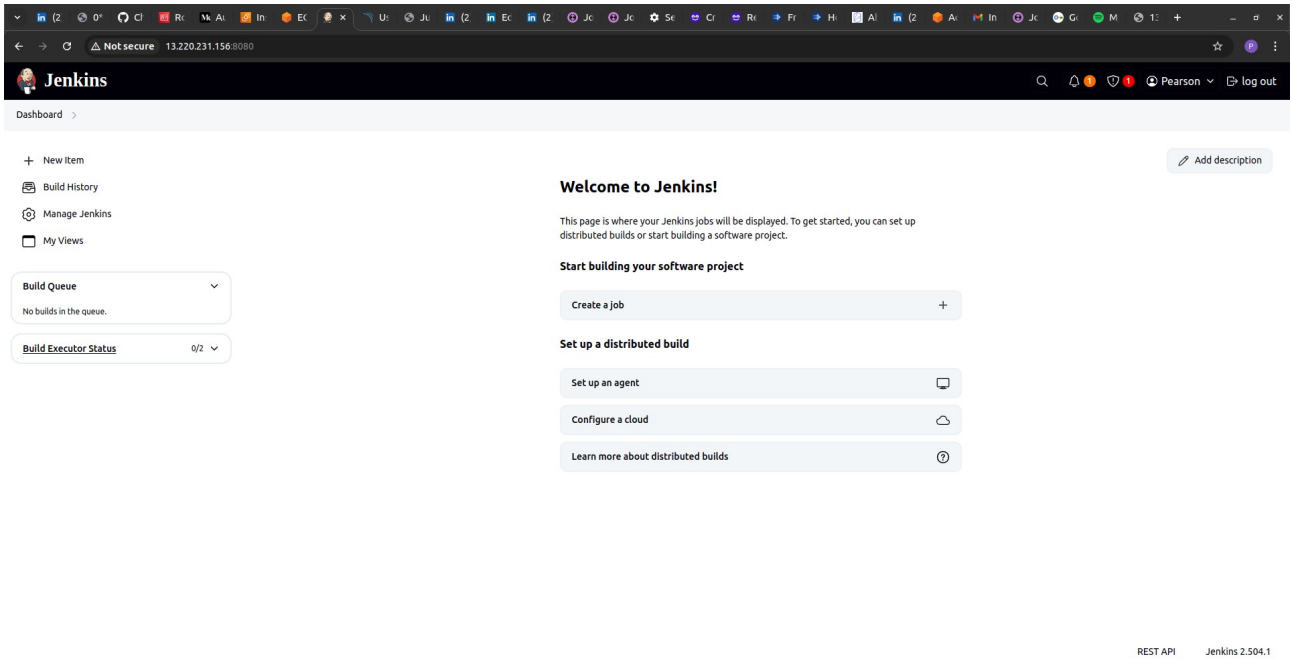
Unlock Jenkins:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

Paste your password on jenkins

Press enter or click to view





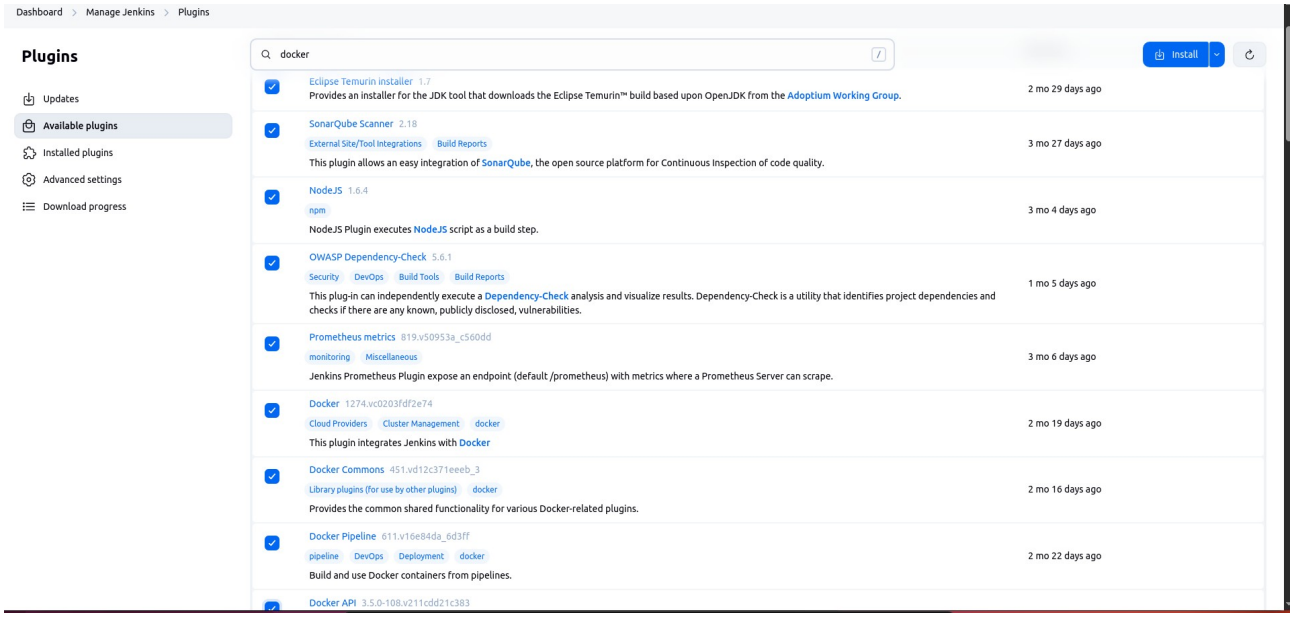
size

Welcome to Jenkins dashboard

Phase 4 → CI-CD pipeline

STEP 4.1 — Install Required Tools on Jenkins Server

All the Docker plugins



Terraform

Eclipse Temurin Installer (Install without restart)

SonarQube Scanner (Install without restart)

NodeJs Plugin (Install Without restart)

Owasp →The OWASP Plugin in Jenkins is like a “security assistant” that helps you find and fix security issues in your software. It uses the knowledge and guidelines from the Open Web Application Security Project (OWASP) to scan your web applications and provide suggestions on how to make them more secure. It’s a tool to ensure that your web applications are protected against common security threats and vulnerabilities.

Prometheus metrics →to moniter Jenkins on Grafana dashboard

Download all the Docker related plugins

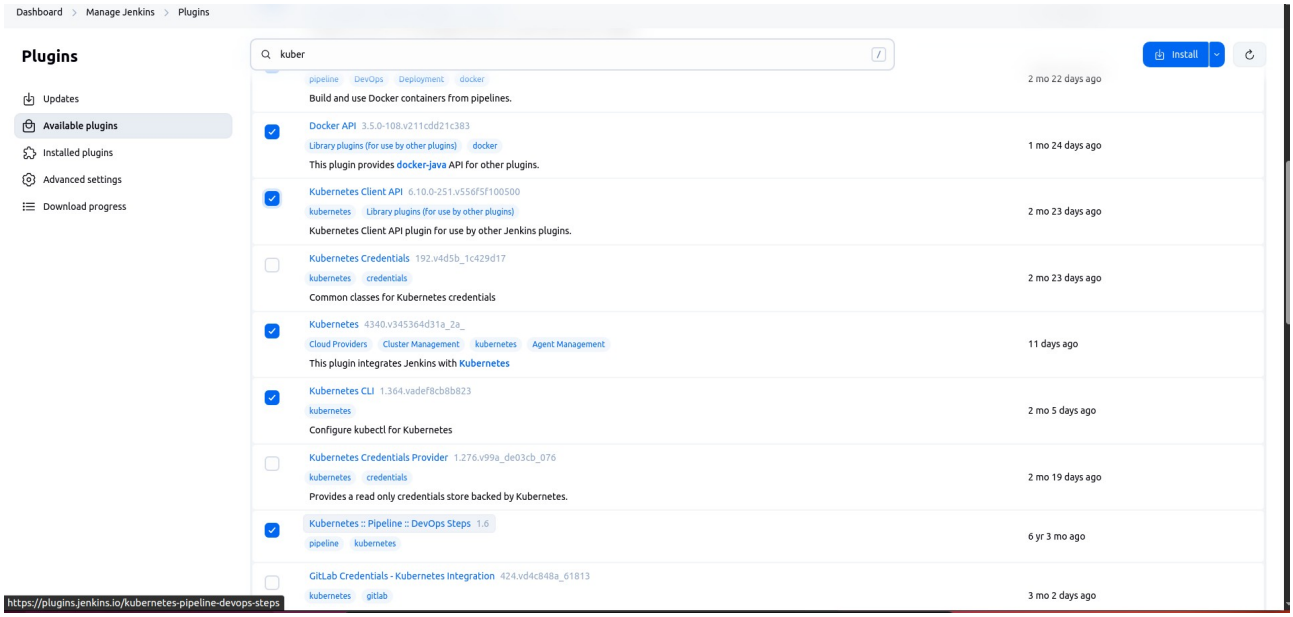
Kubernetes

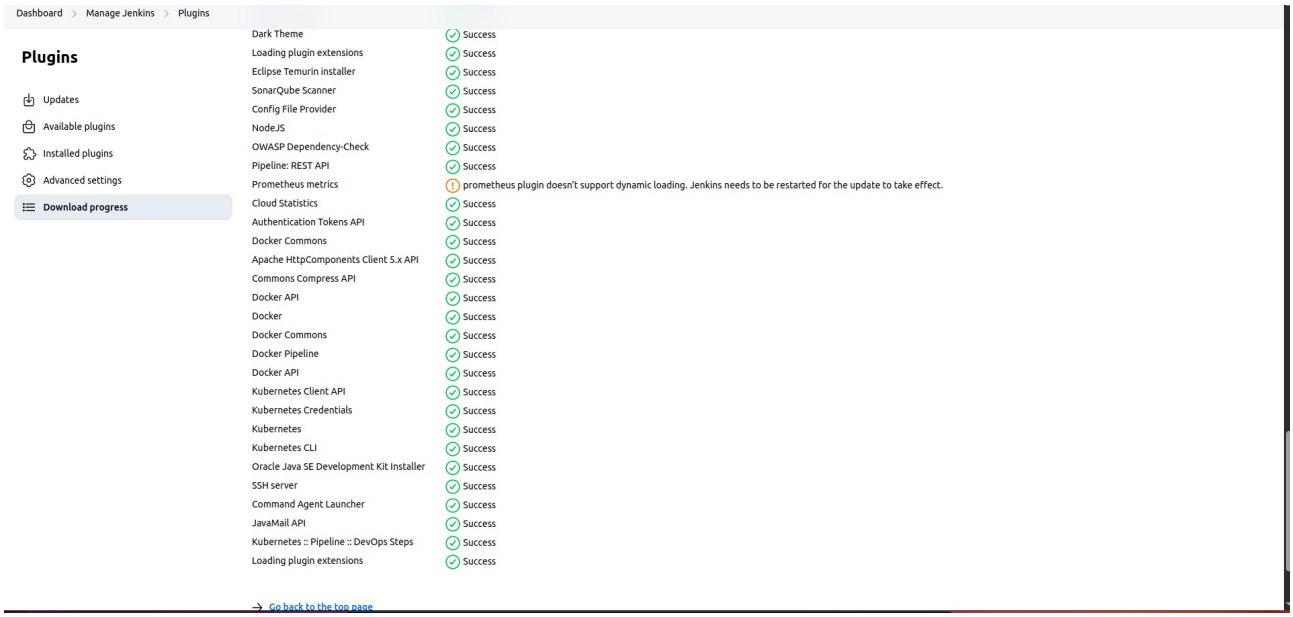
Kubernetes CLI

Kubernetes Client API

Kubernetes Pipeline DevOps steps

AWS CLI





STEP 4.2 — Add SonarQube credentials to Jenkins

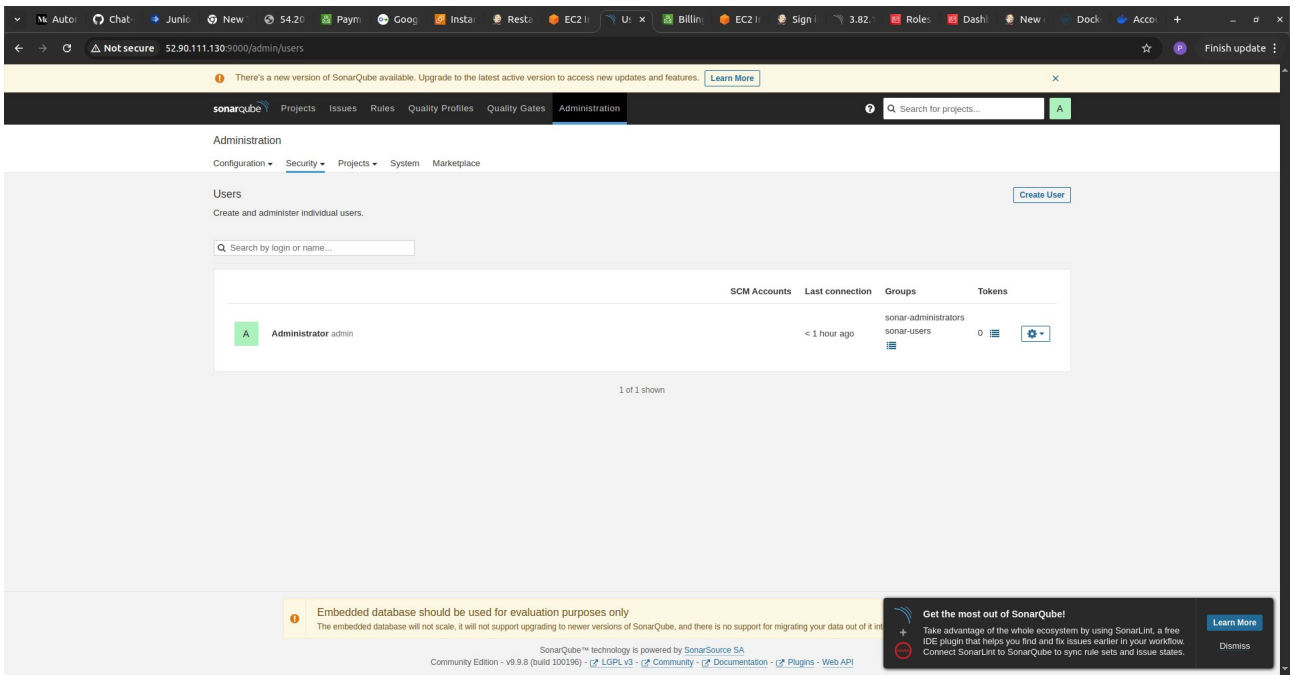
WE FIRST GENERATE A TOKEN FOR SONARQUBE TO USE IT IN JENKINS CREDENTIALS AS SECRETE TEXT

Go to **SonarQube Server**:

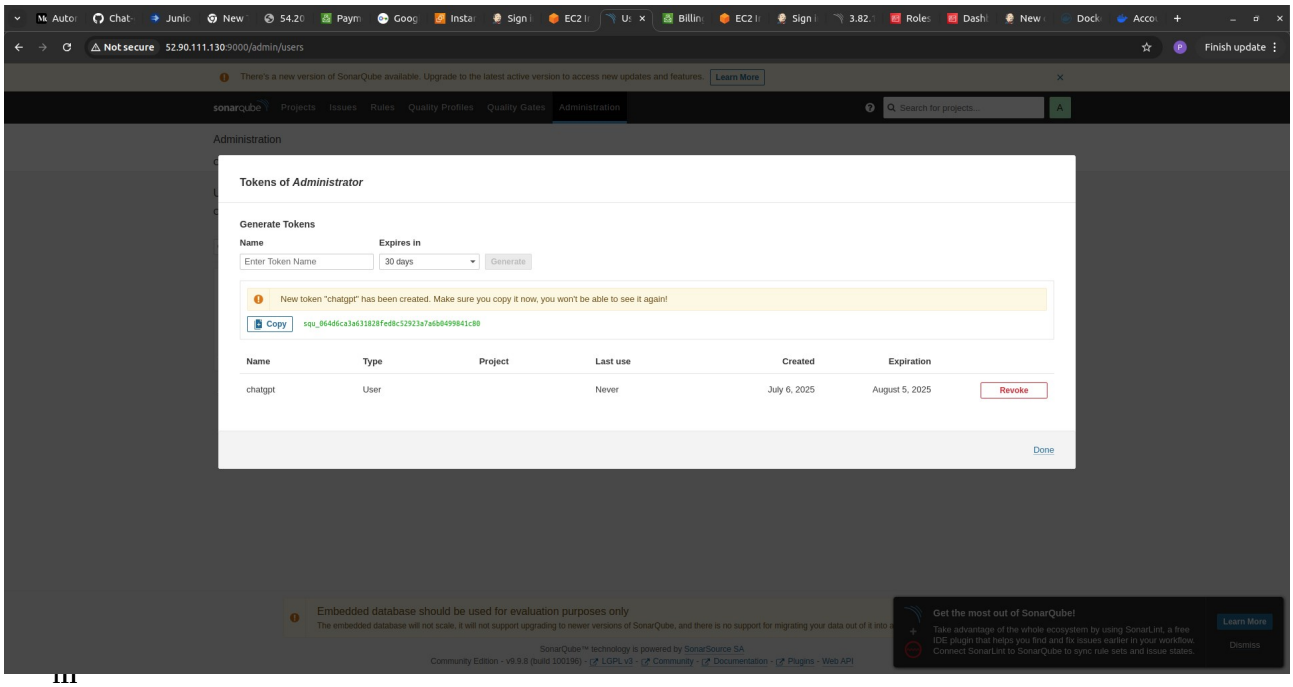
- URL: `http://publicip:9000`

security → users → token → generate token

- Token name: `jenkins`



r



in
a
g
e

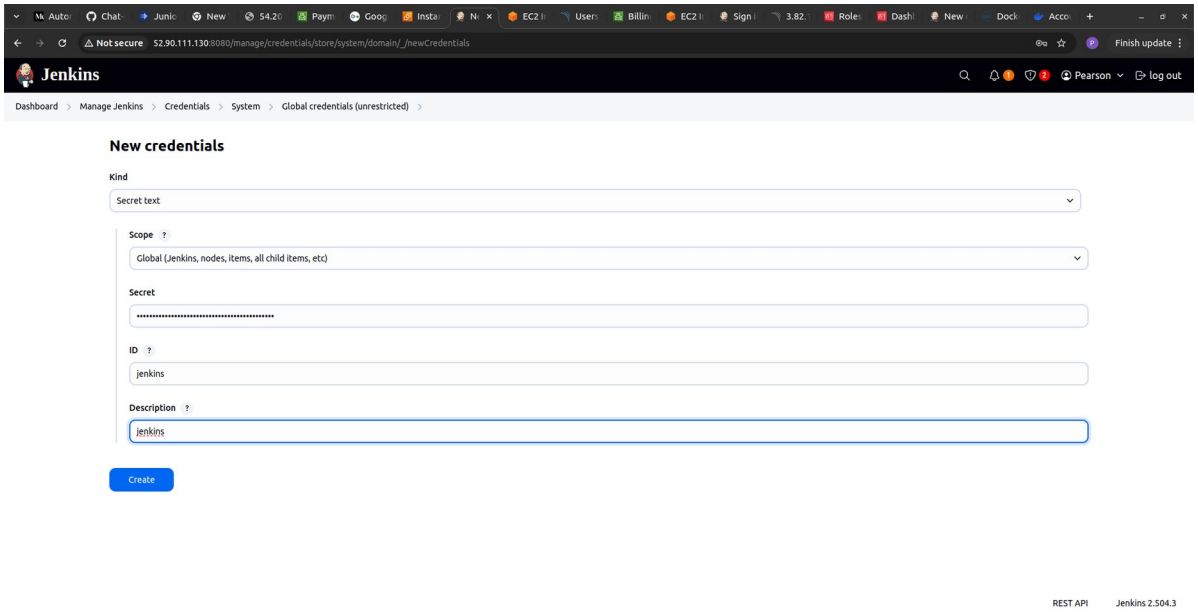
---

i• **Copy the token and go to your Jenkins → manage Jenkins**
n **→ credentials → global → add credentials**

f• Select secret text from dropdown
u
l• secret text: **your token**
l
• id: **jenkinsn**

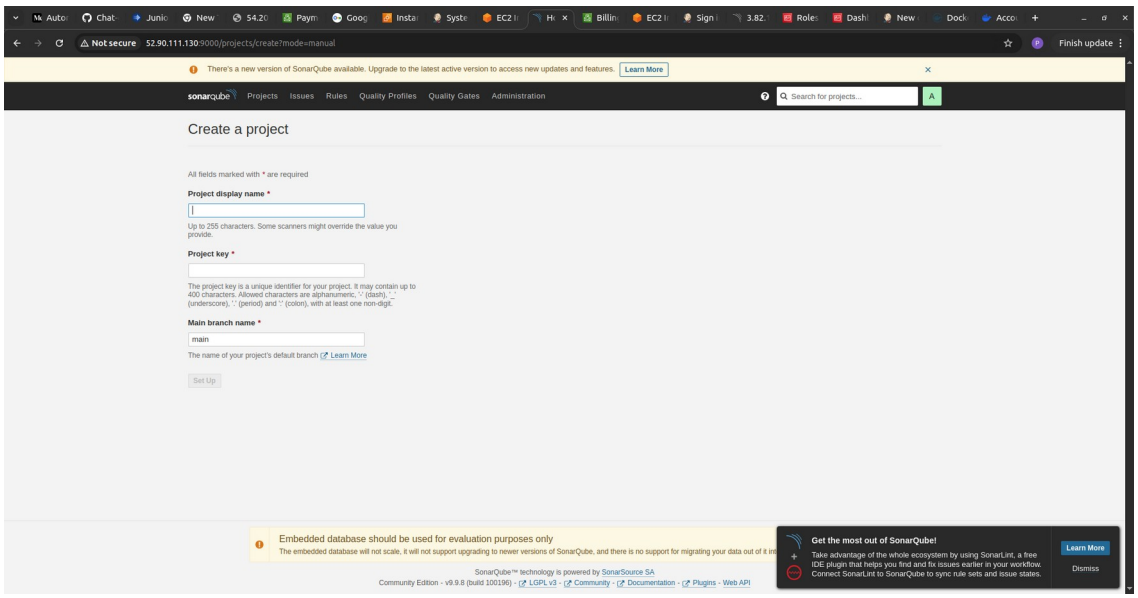
s
i
z
e

- Click: create

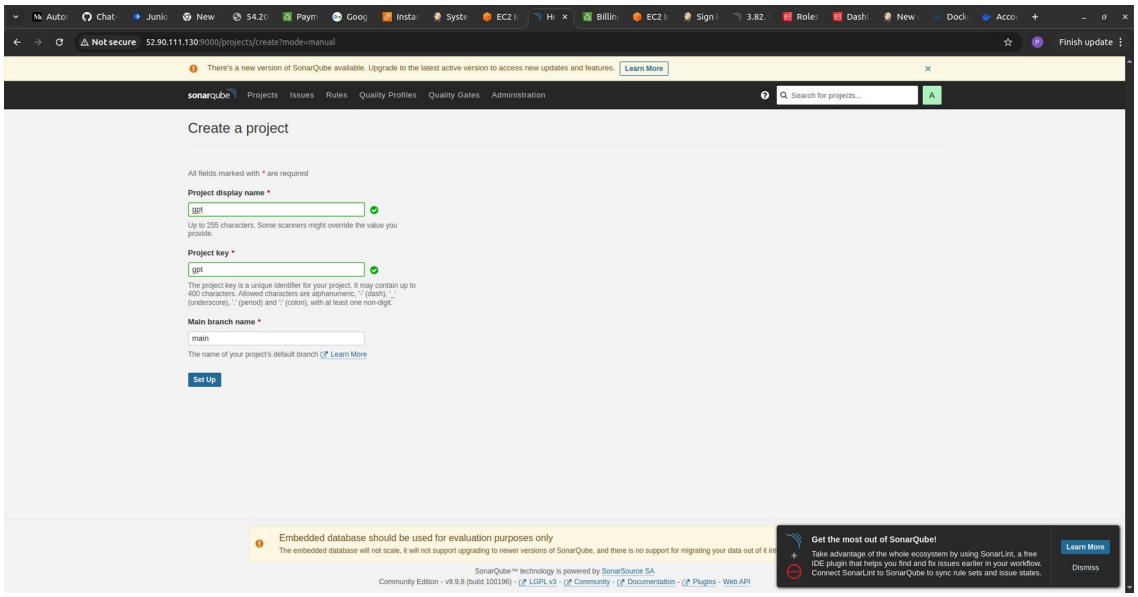


STEP 4.3 — **Setup projects in Sonarqube for Jenkins**

- Go to your Sonarqube server
- Click on projects
- In the name field type gpt
- Click on set up

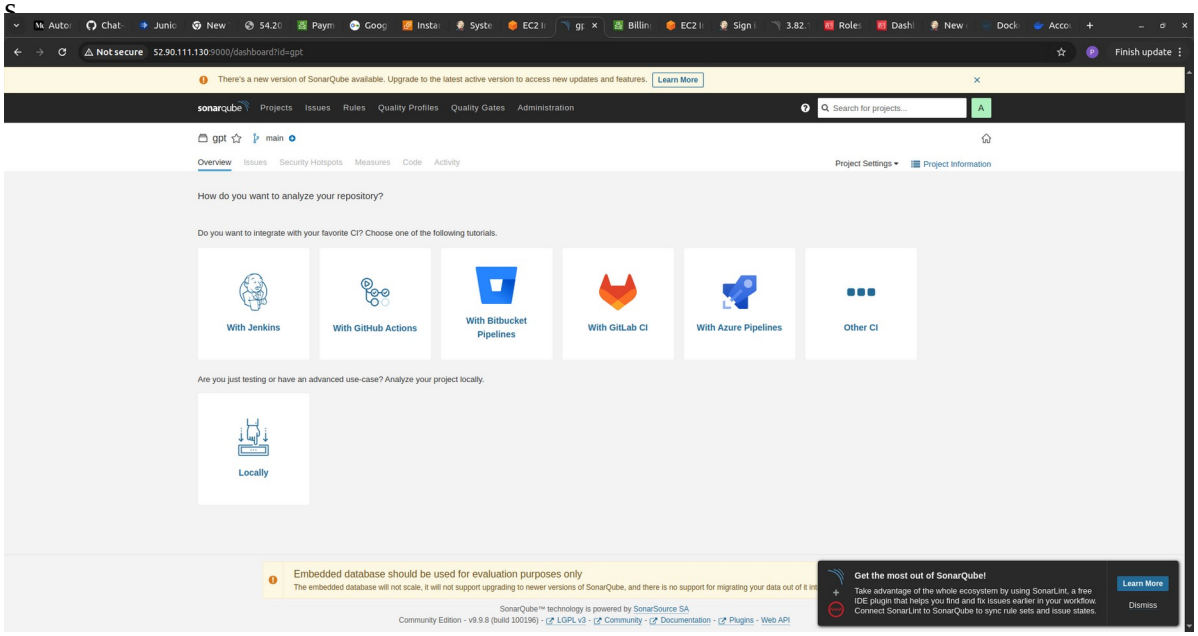


-



click on locally

Press enter or click to.



click on generate

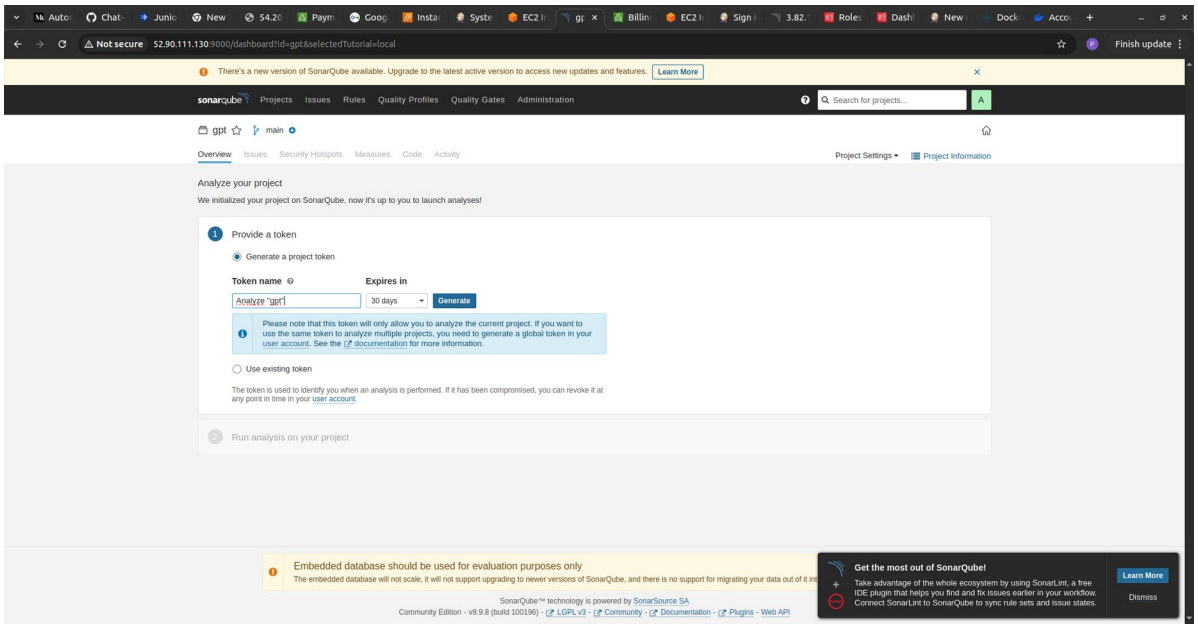
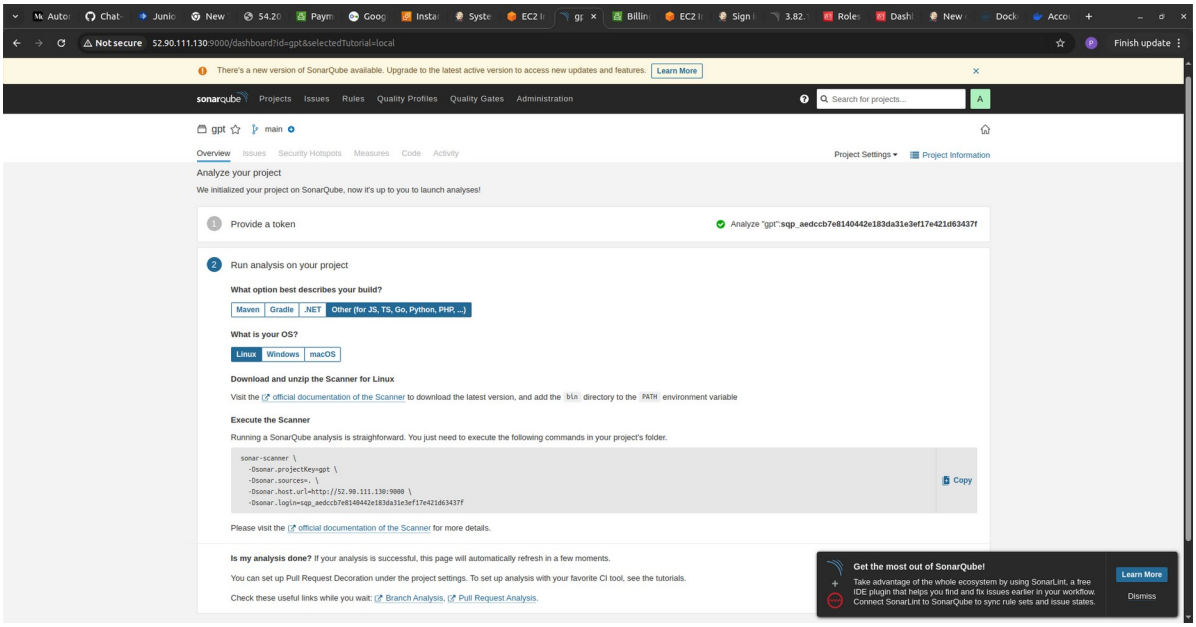


image in full size



click on continue ….

Sonarqube project for Jenkins is setup now

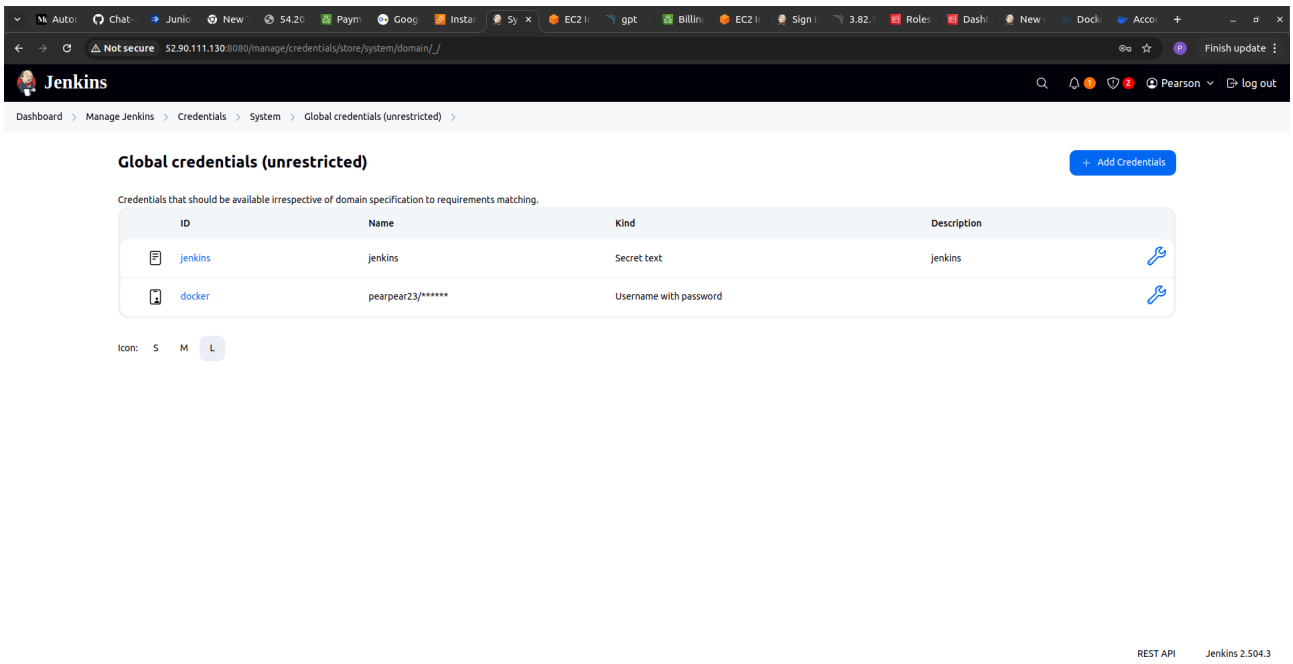
STEP 4.4 — `DockerHub Credentials (REQUIRED)`

Create DockerHub credentials in Jenkins

Jenkins → Manage Jenkins → Credentials → Global → Add Credentials

- Kind: `Username with password`
- ID: docker
- Username: `<your-dockerhub-username>`
- Password: `<dockerhub-password>`

Save.

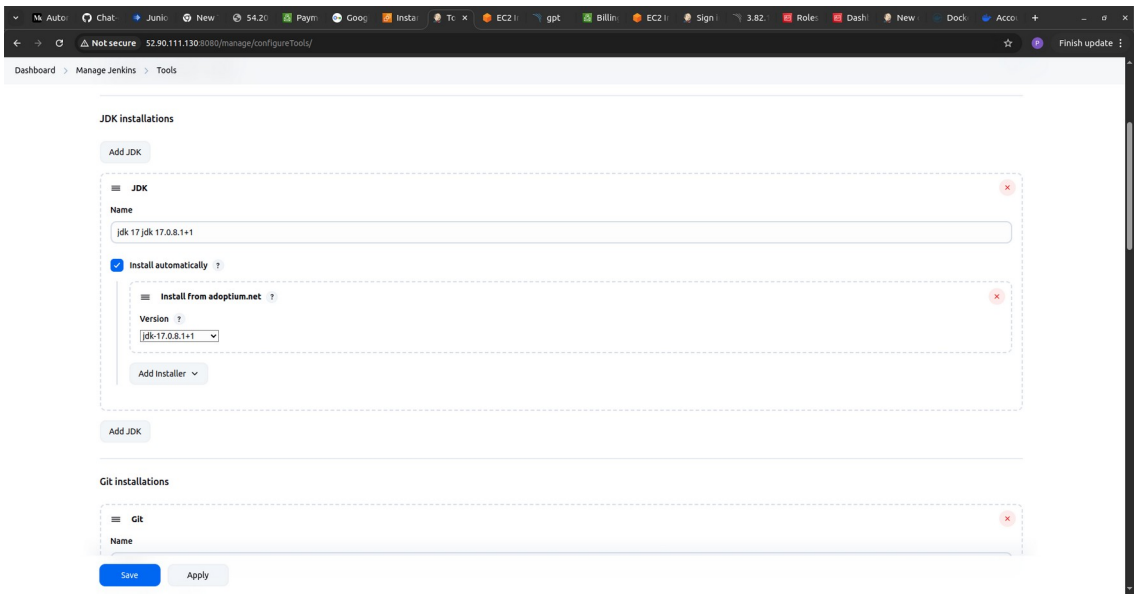


STEP 4.5 — **Setup tools for jenkins**

go to:
jenkins → tools

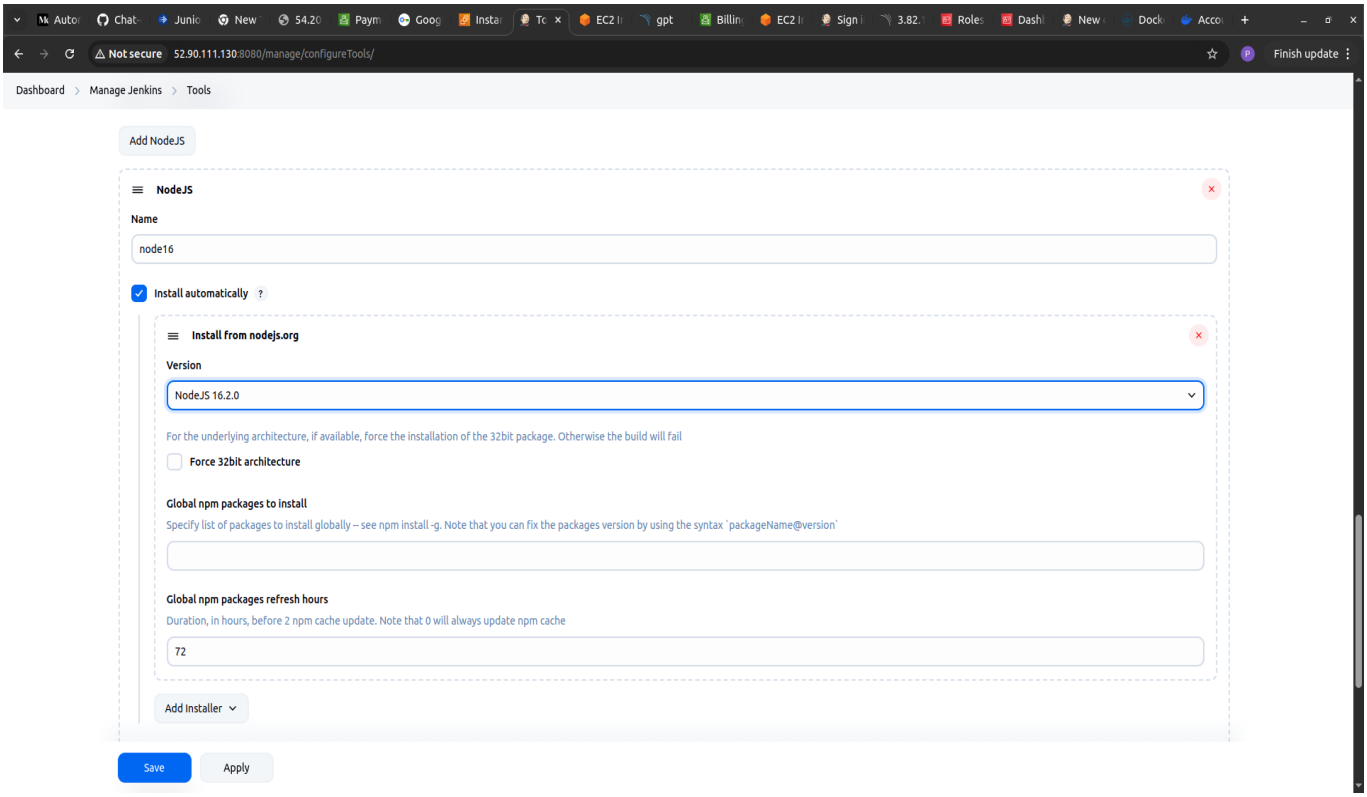
a. Add jdk

1. click on add jdk and select installer adoptium.net
2. choose jdk 17.0.8.1+1version and in name section enter jdk 17



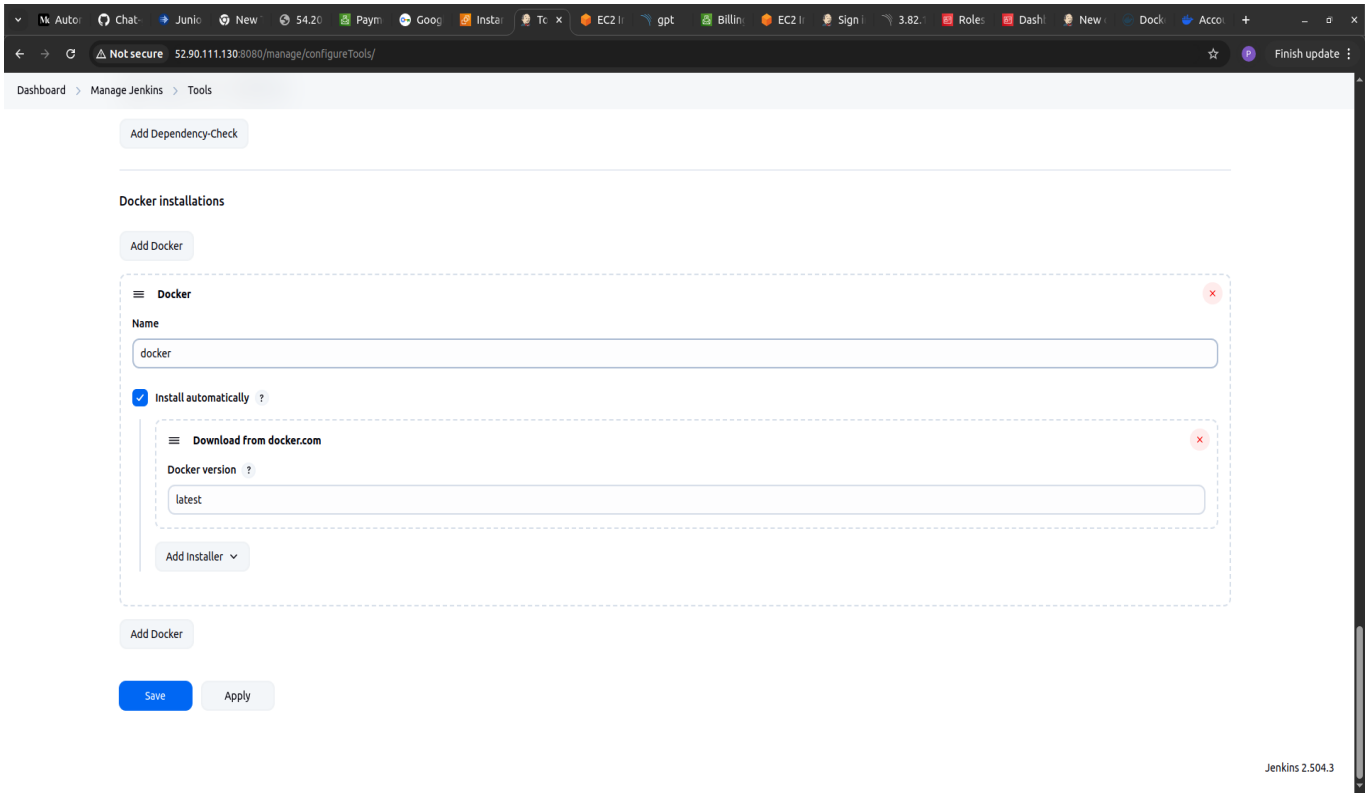
b. Add node js

1. click on add nodejs
2. enter node16 in name section
3. choose version nodejs 16.2.0



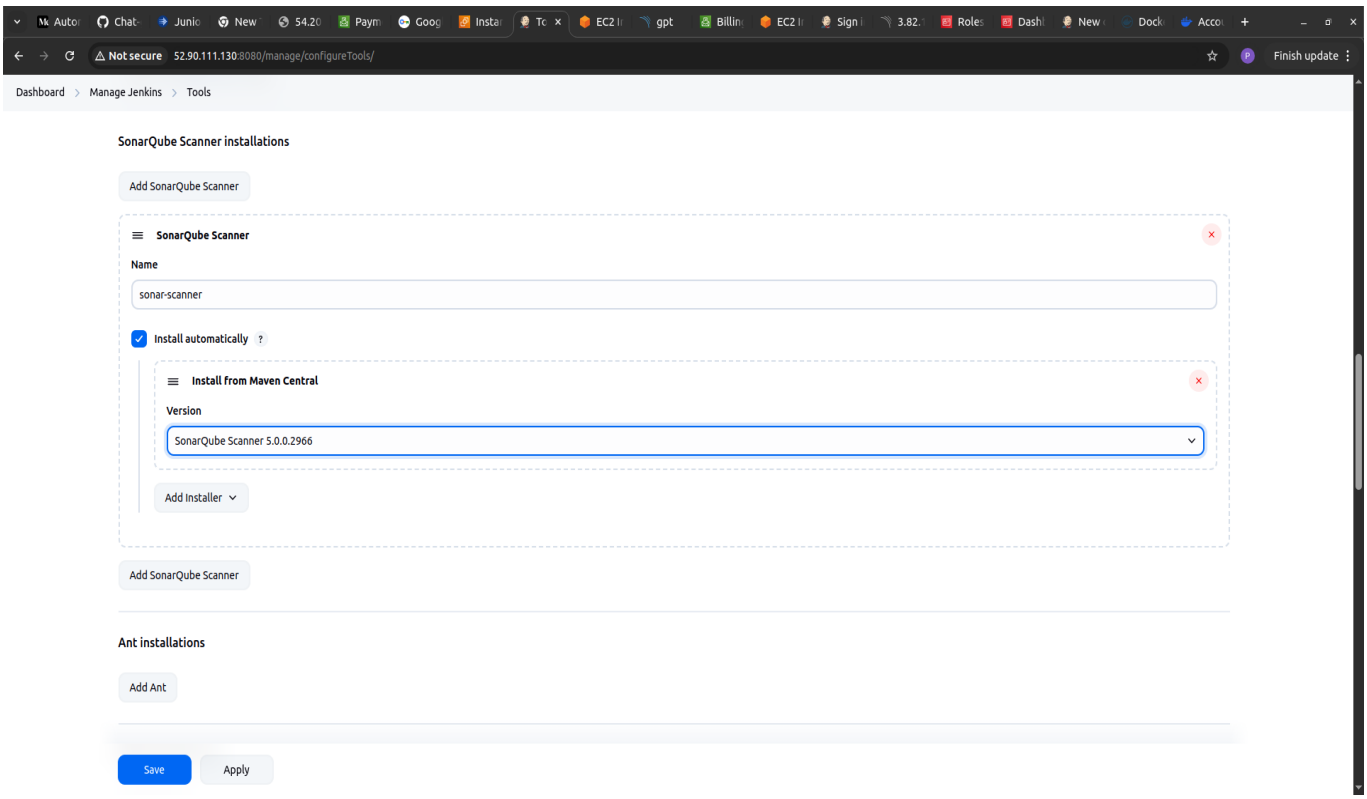
c. Add Docker

1. click on add docker
2. name==docker
3. add installer ==download from docker.com



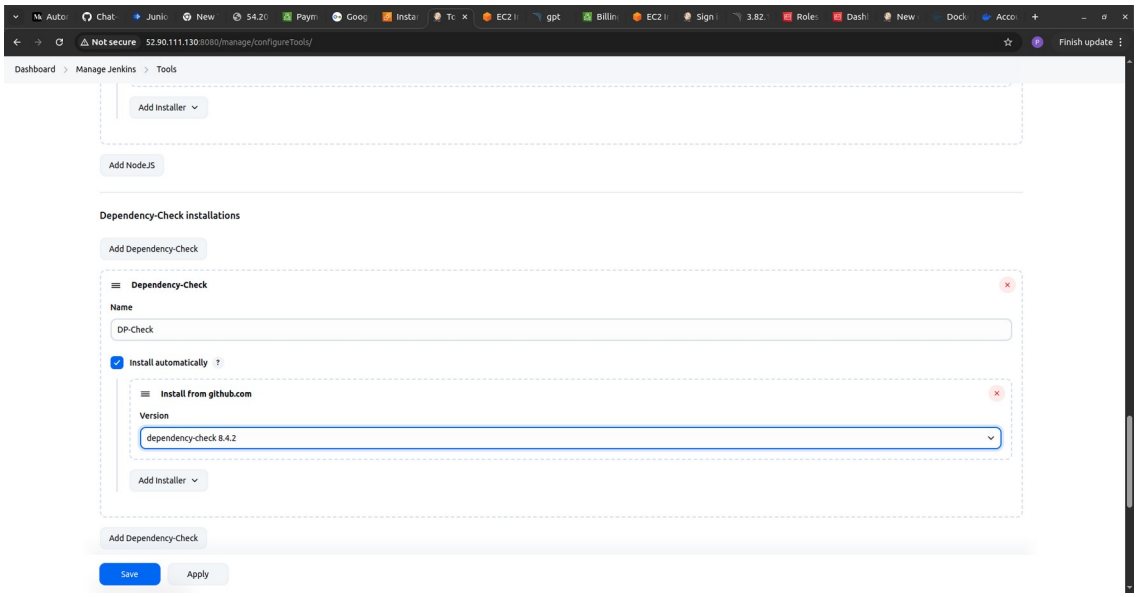
d. Add Sonarqube

1. add sonar scanner
2. name ==sonar-scanner



e. Add owasp dependency check

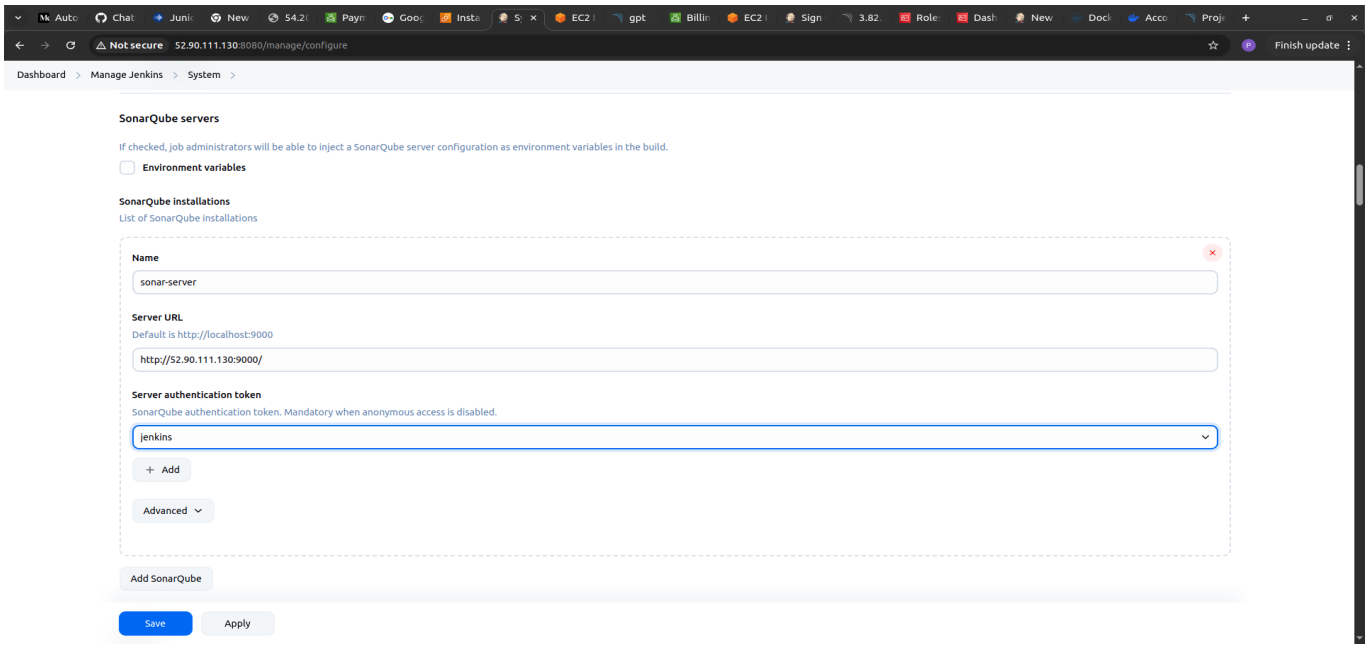
1. add dependency check
2. name == DP-Check
3. from add installer select install from github.com



STEP 4.6 – Configure global settings for sonarqube and webhooks

jenkins → configure global setting → Add Sonarqube servers

- Jenkins → New Item
- Name: sonar-server
- server_url: http://public_ip:9000
- server authentication token: jenkins (its created in sonarqube security configurations)



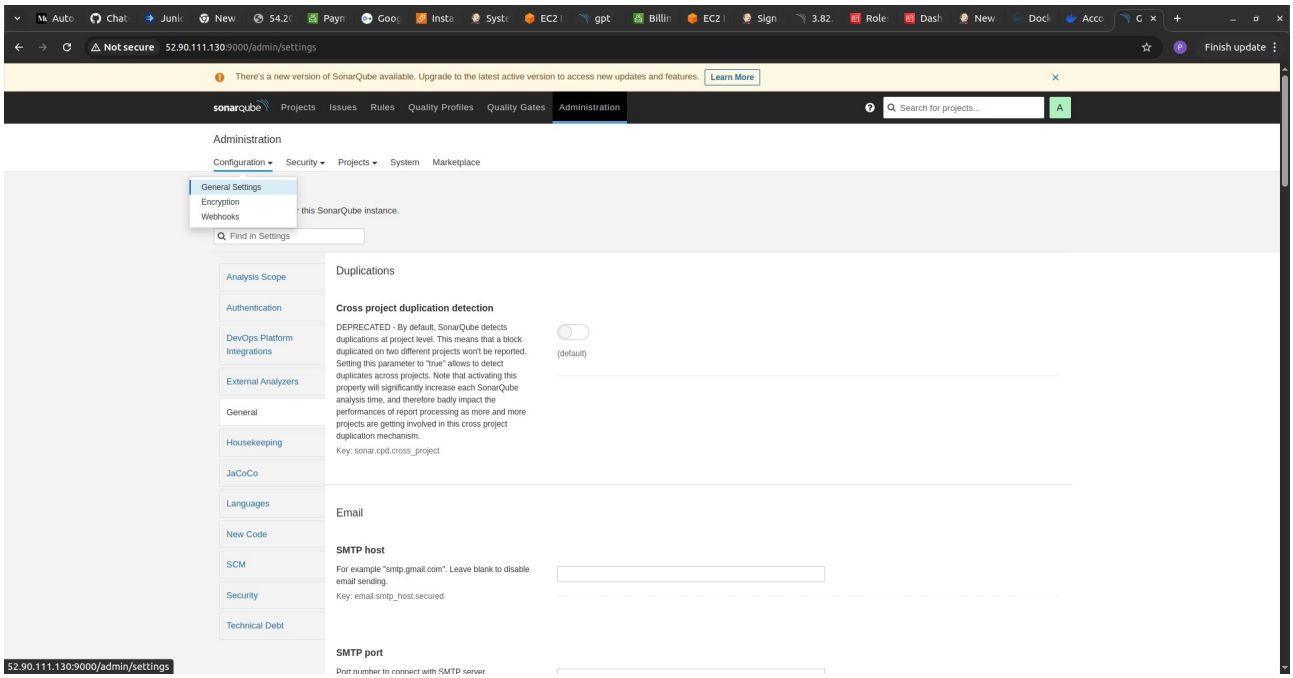
Setup webhooks

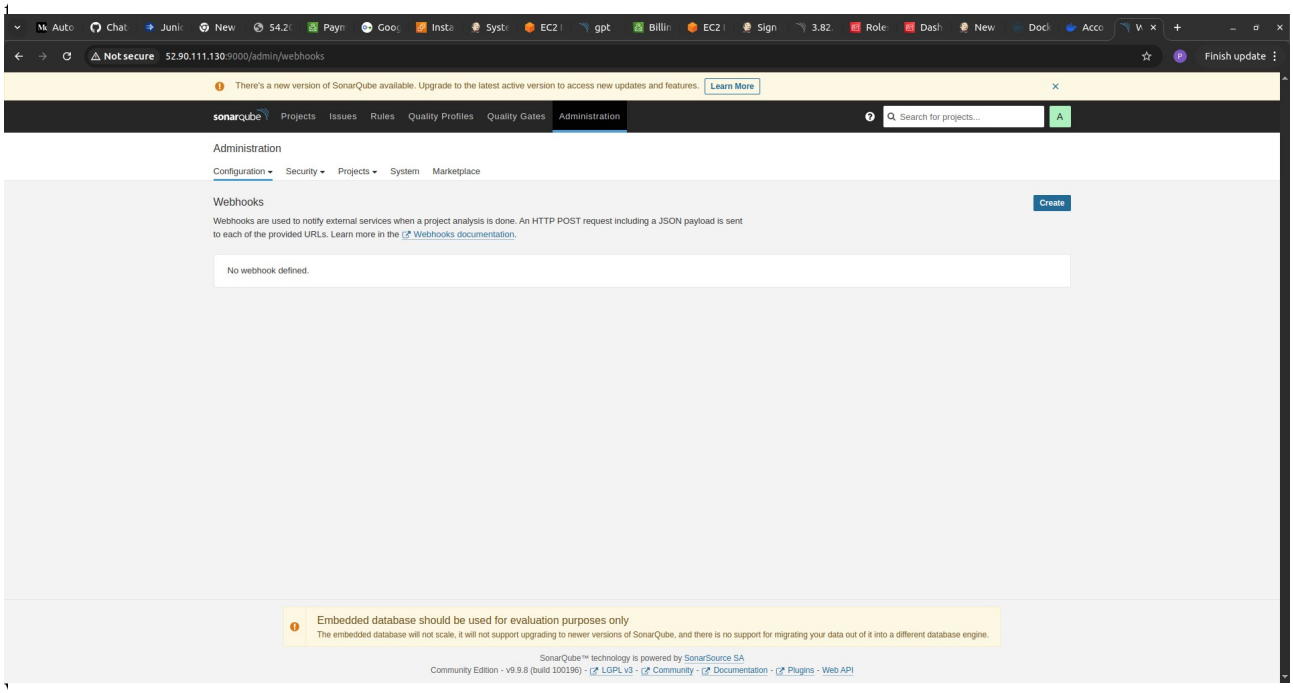
In sonarqube server:

administration → configuration → webhooks

1. create webhook
2. provide and in the url section enter →
3. http://jenkins-public-ip:8080/sonarqube-webhook/

4. click create





STEP 4.8 — Run the pipeline in Jenkins

In Jenkins server

new item → select pipeline → type: gpt-pipeline

scroll down to the pipeline script and paste the script below

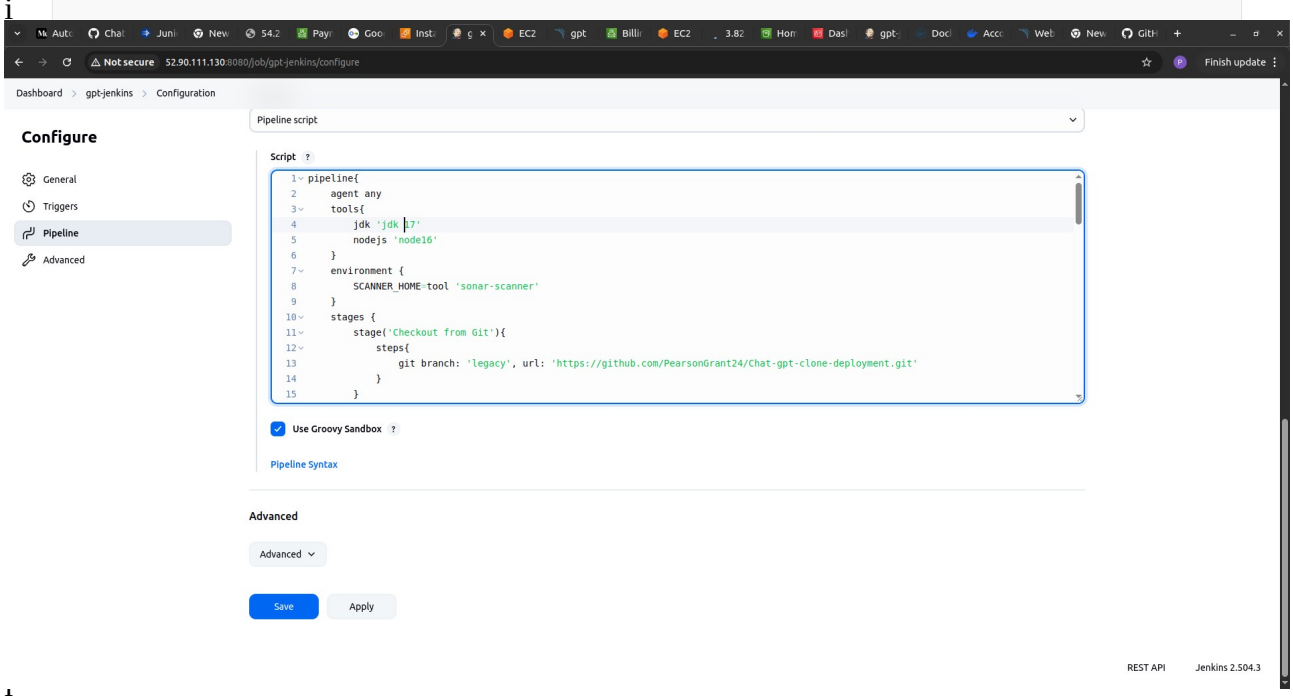
run the pipeline

```
pipeline{
    agent any
    tools{
        jdk 'jdk17'
        nodejs 'node16'
    }
    environment {
        SCANNER_HOME=tool 'sonar-scanner'
    }
    stages {
        stage('Checkout from Git'){
            steps{
                git branch: 'legacy', url: 'https://github.com/PearsonGrant24/Chat-gpt-deployment.git'
            }
        }
        stage('Install Dependencies') {
            steps {
                sh "npm install"
            }
        }
        stage("Sonarqube Analysis "){
            steps{
```

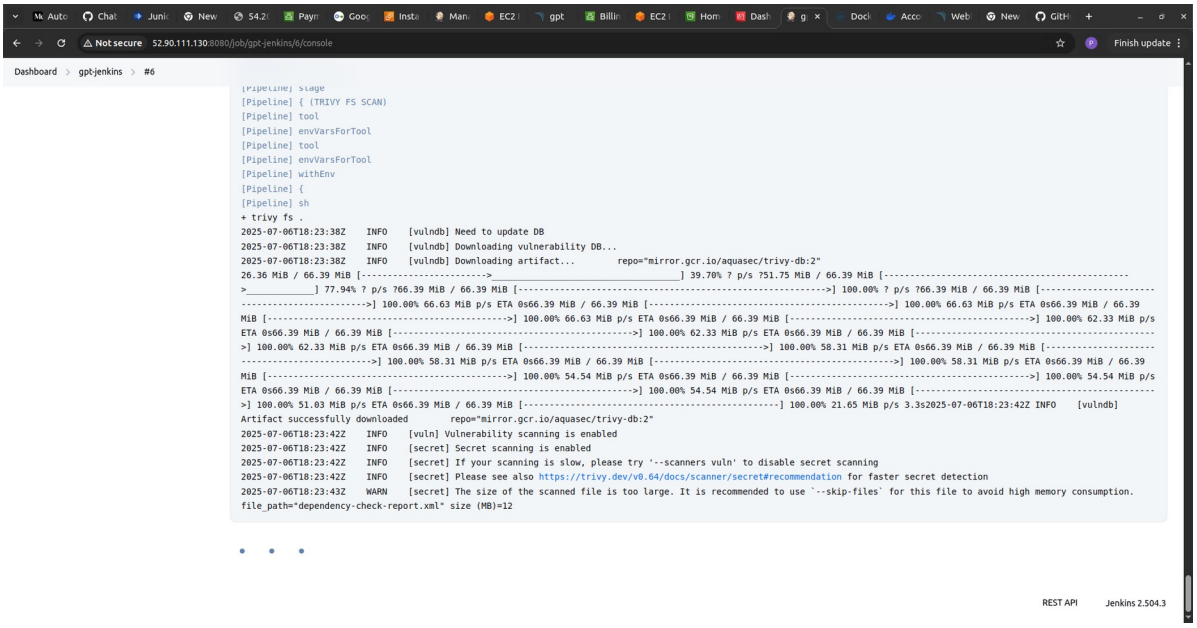
```
            withSonarQubeEnv('sonar-server') {
                sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=Chatbot \
                -Dsonar.projectKey=Chatbot '''
            }
        }
    }
    stage("quality gate"){
        steps {
            script {
                waitForQualityGate abortPipeline: false, credentialsId: 'Sonar-token'
            }
        }
    }
    stage('OWASP FS SCAN') {
        steps {
            dependencyCheck additionalArguments: '--scan ./ --disableYarnAudit --disableNodeAudit', odcInstallation: 'DP-Check'
            dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
        }
    }
    stage('TRIVY FS SCAN') {
        steps {
            sh "trivy fs . > trivyfs.json"
        }
    }
    stage("Docker Build & Push"){
        steps{
            script{
              withDockerRegistry(credentialsId: 'docker', toolName: 'docker'){
                sh "docker build -t chatbot ."
                sh "docker tag chatbot #your_dockerhub_id/chatbot:latest "
                sh "docker push #your_dockerhub_id/chatbot:latest "
              }
            }
```

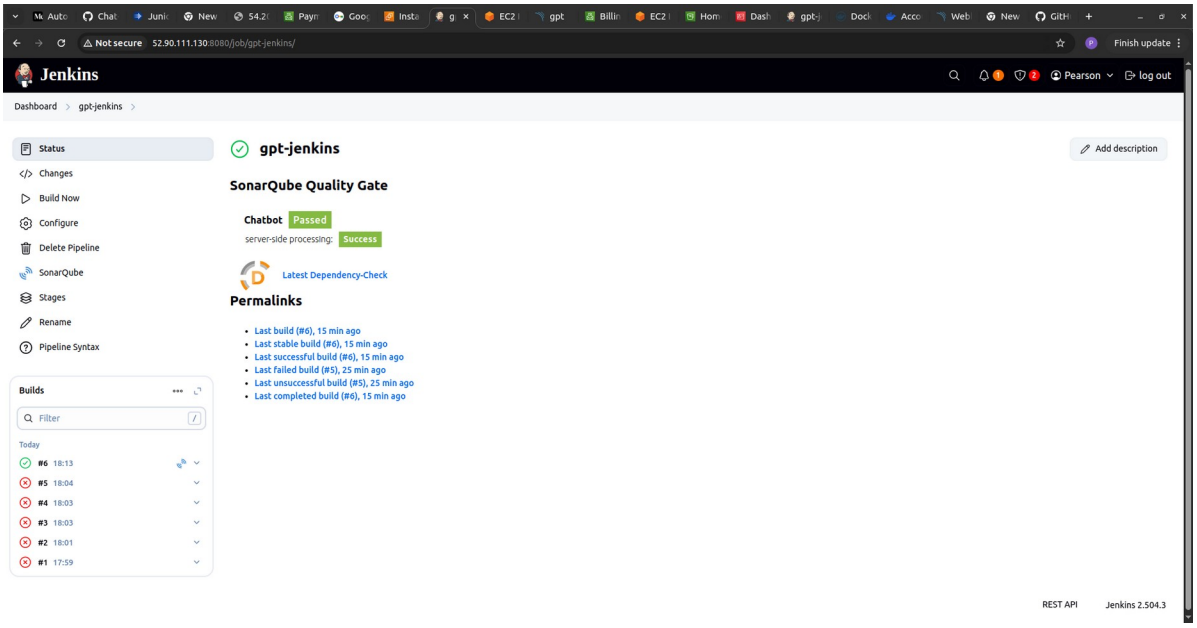
Press enter or click to view image size

```
            }
        }
        stage("TRIVY"){
            steps{
                sh "trivy image #your_dockerhub_id/chatbot:latest > trivy.json"
            }
        }
        stage ("Remove container") {
            steps{
                sh "docker stop chatbot | true"
                sh "docker rm chatbot | true"
             }
        }
        stage('Deploy to container'){
            steps{
                sh 'docker run -d --name chatbot -p 3000:3000
#your_dockerhub_id/chatbot:latest'
            }
        }
    }
}
```

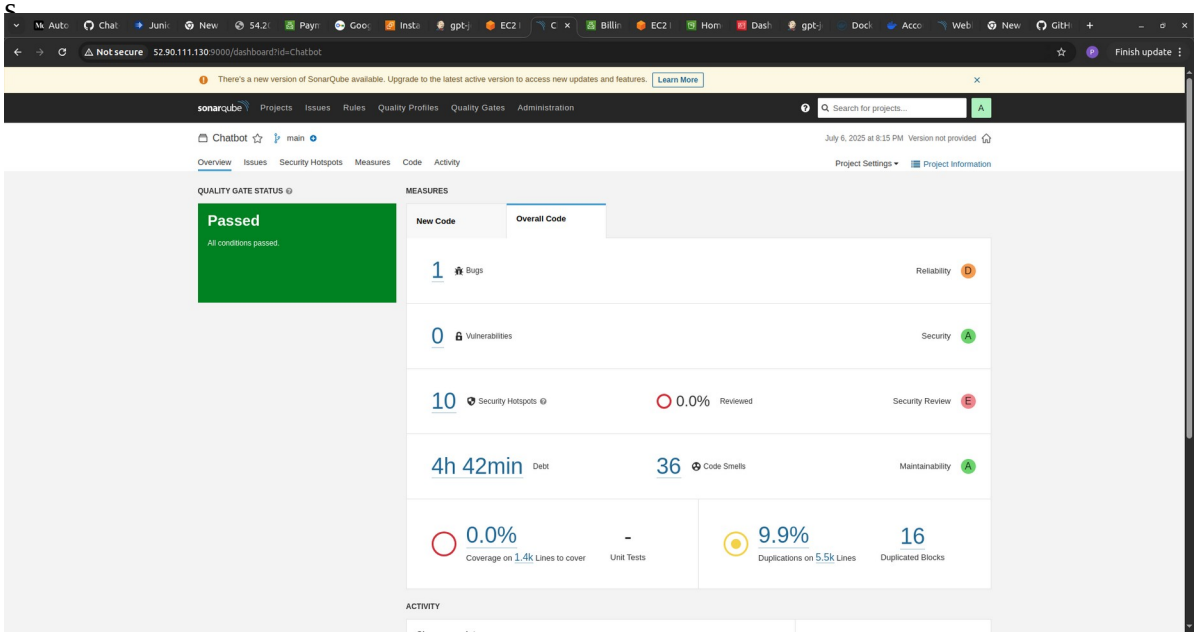


Press enter or click to view image in full size





Press

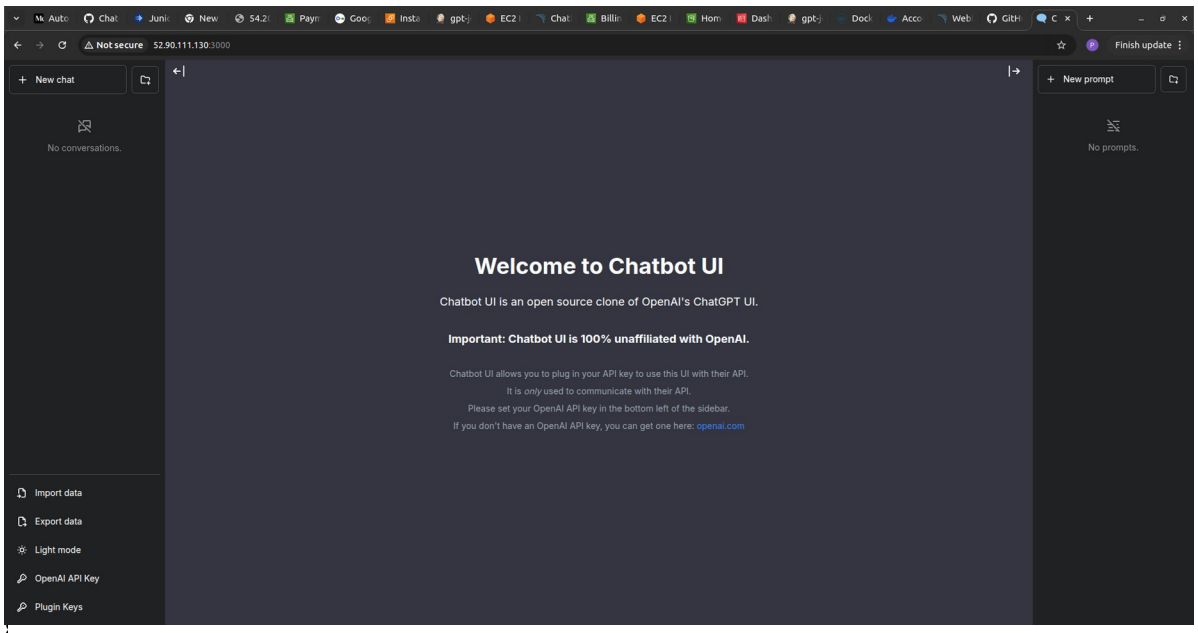


check to

Application is successfully deployed

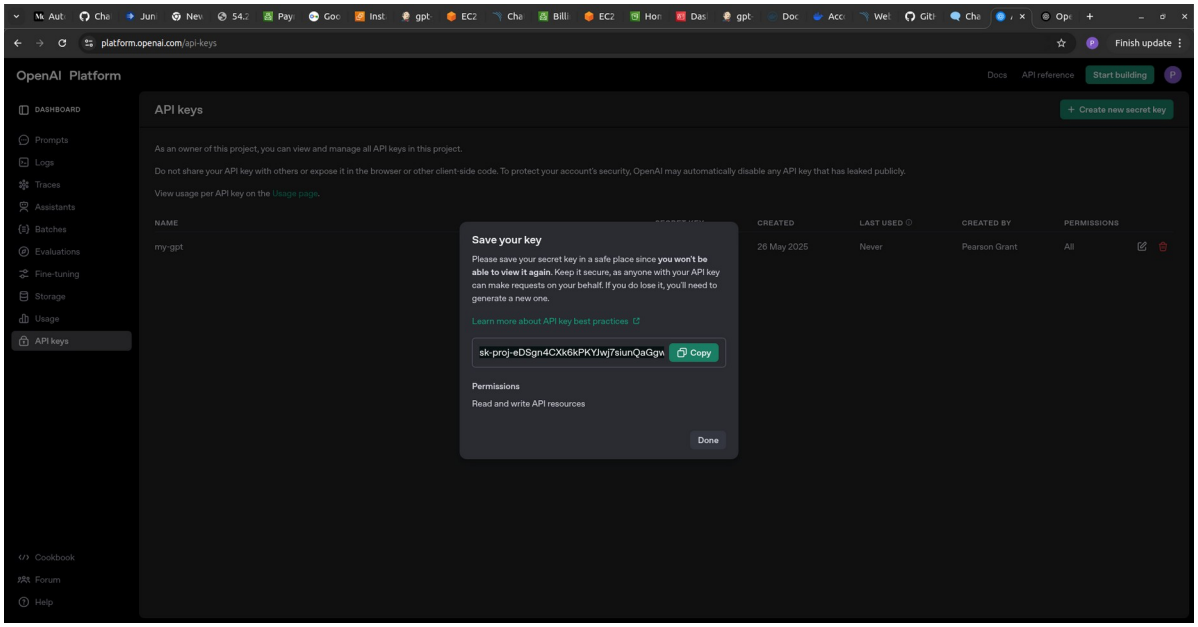
Checkout your dockerhub and to access your application

Go to your browser and type

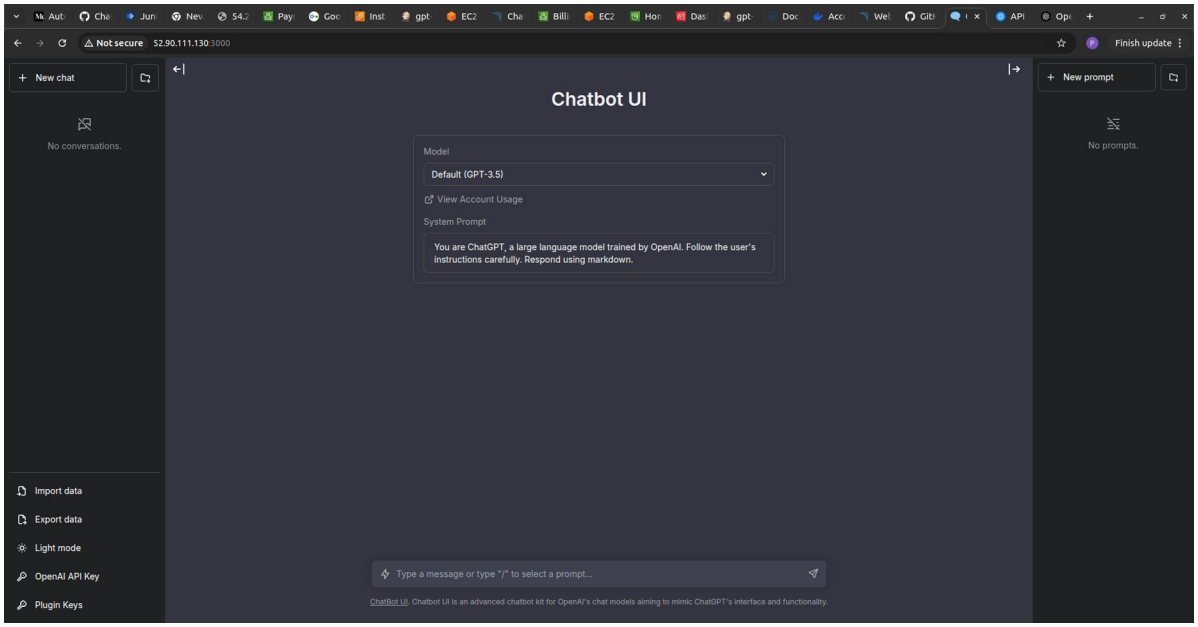


STEP 4.8 – Get API to connect your app to Openai

- Click or search: openai.com
- Click: create a new secret access key
- Provide: name and click on create new secret key
- Copy the secret access key and comeback to your application
- Click: on openai api key present on bottom left corner and paste your api key and now your application is ready to be used



•



Press enter or click to view image in full size

PHASE 5 — Kubernetes cluster

Step 5.1 — What is Kubernetes (Simple Explanation)

Think of Kubernetes as:

- **Manager** → schedules apps
- **Pods** → smallest runnable unit
- **Nodes** → EC2 machines
- **Deployment** → desired state
- **Service** → network access

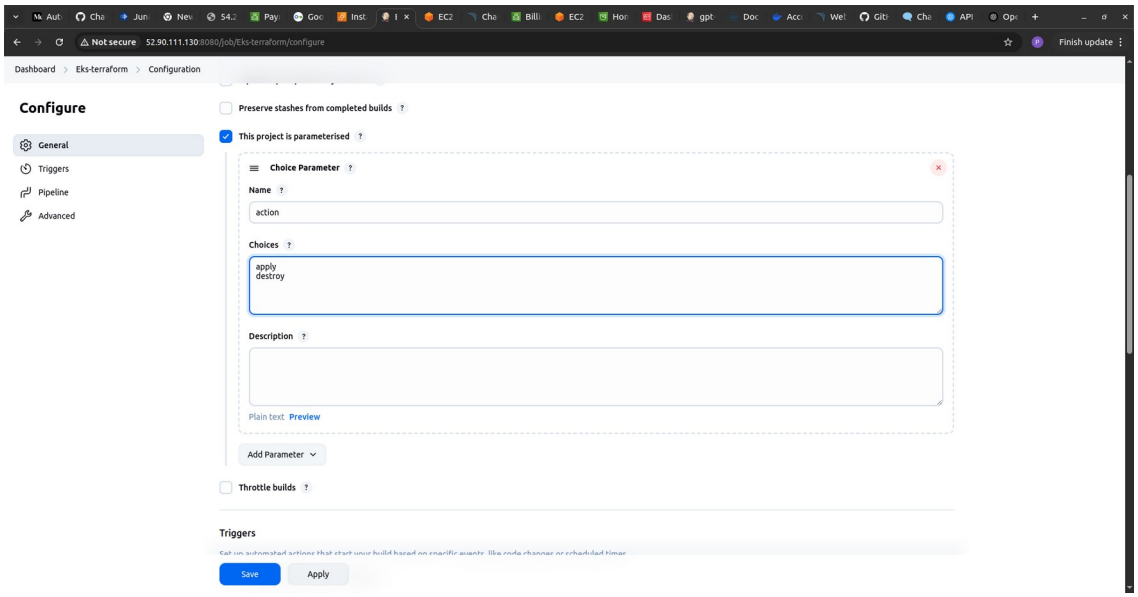
You already have:

- EKS control plane (AWS-managed)

STEP 5.2 — Creating kubernetes cluster

- create a new pipeline named as eks-terraform
- scroll down and select this project as parameterized
- choice: apply and destroy

Press enter or click to view image in full size



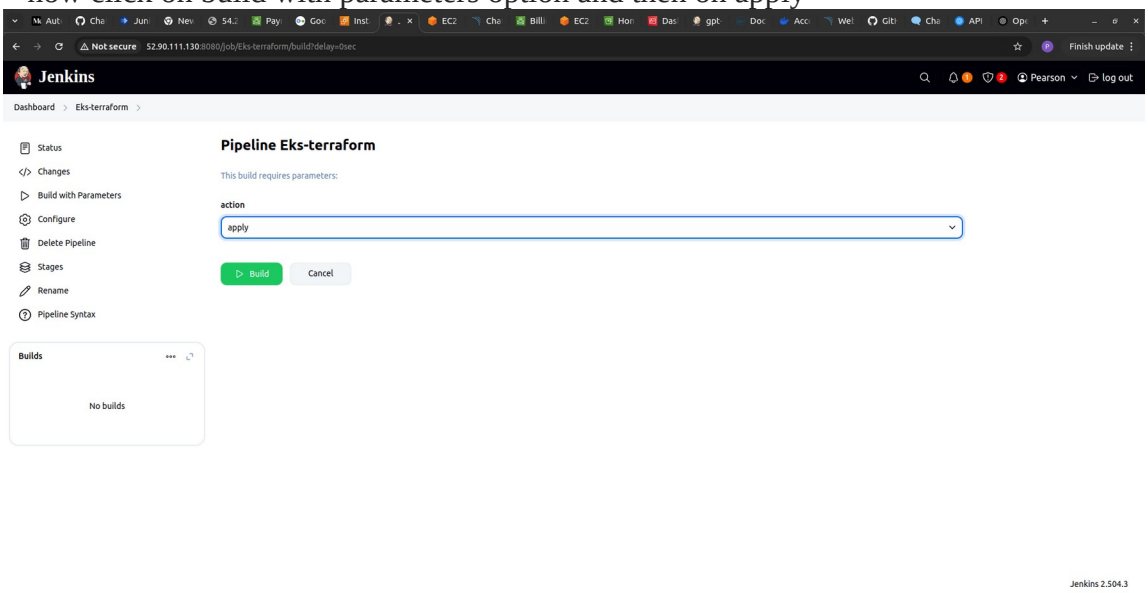
3. scroll down to the pipeline script and paste the below pipeline script:

```
pipeline{
    agent any
    stages {
        stage('Checkout from Git'){
            steps{
                git branch: 'legacy', url: 'https://github.com/PearsonGrant24/Chat-gpt-deployment.git'
            }
        }
        stage('Terraform version'){
            steps{
                sh 'terraform --version'
            }
        }
        stage('Terraform init'){
            steps{
                dir('Eks-terraform') {
                    sh 'terraform init'
                }
            }
        }
        stage('Terraform validate'){
```

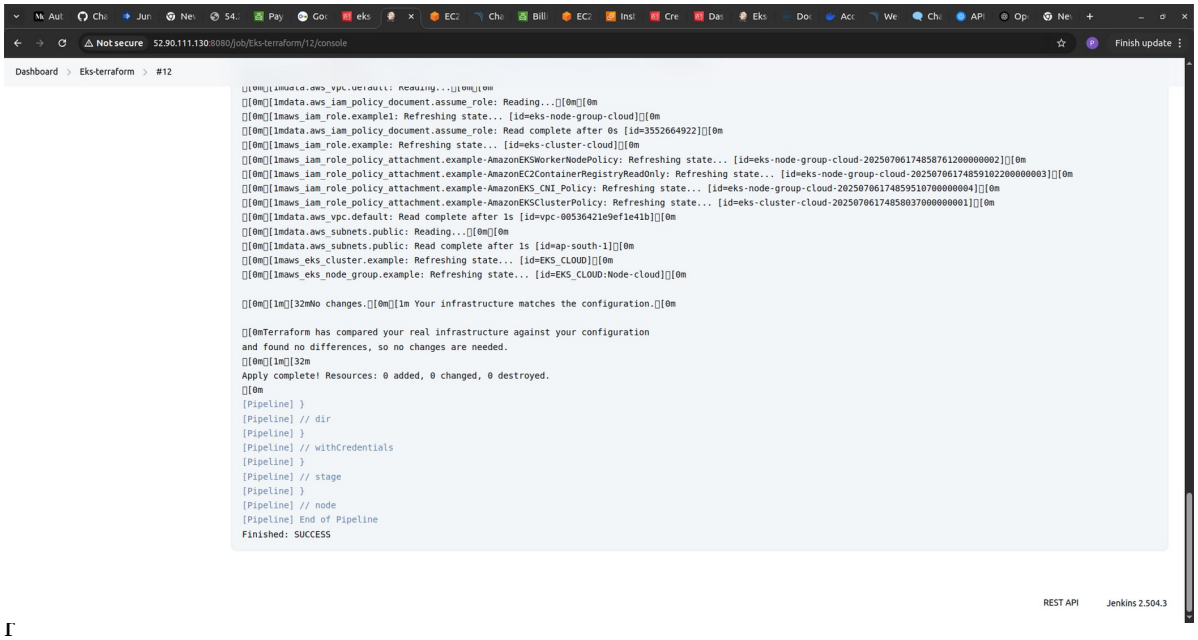
Press enter or click to view image in full size

```
                steps{
                    dir('Eks-terraform') {
                        sh 'terraform validate'
                    }
                }
            }
            stage('Terraform plan'){
                steps{
                    dir('Eks-terraform') {
                        sh 'terraform plan'
                    }
                }
            }
            stage('Terraform apply/destroy'){
                steps{
                    dir('Eks-terraform') {
                        sh 'terraform ${action} --auto-approve'
                    }
                }
            }
        }
    }
```

- click on apply and then save
- now click on build with parameters option and then on apply

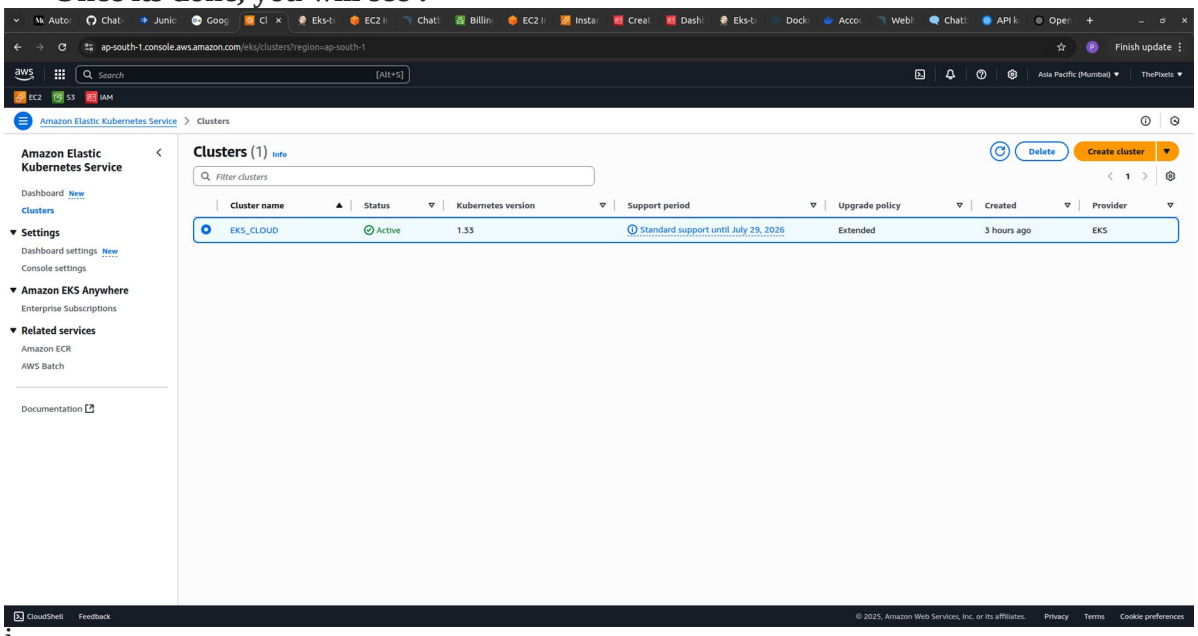


Press enter or click to view image in full size



Its gonna take some time (like 5 – 10 minutes):

Once its done, you will see :



8. check for the node groups →go to your ec2 instances

Press enter or click view image in full size

STEP 5.3 — Deploy on kubernetes

- go to your gpt instance ,
- Run this (replace cluster name if needed):

```
aws eks update-kubeconfig --region <ap-south-1> \
  --name  <clustername>
```

This creates:

`~/.kube/config`

Verify Connection

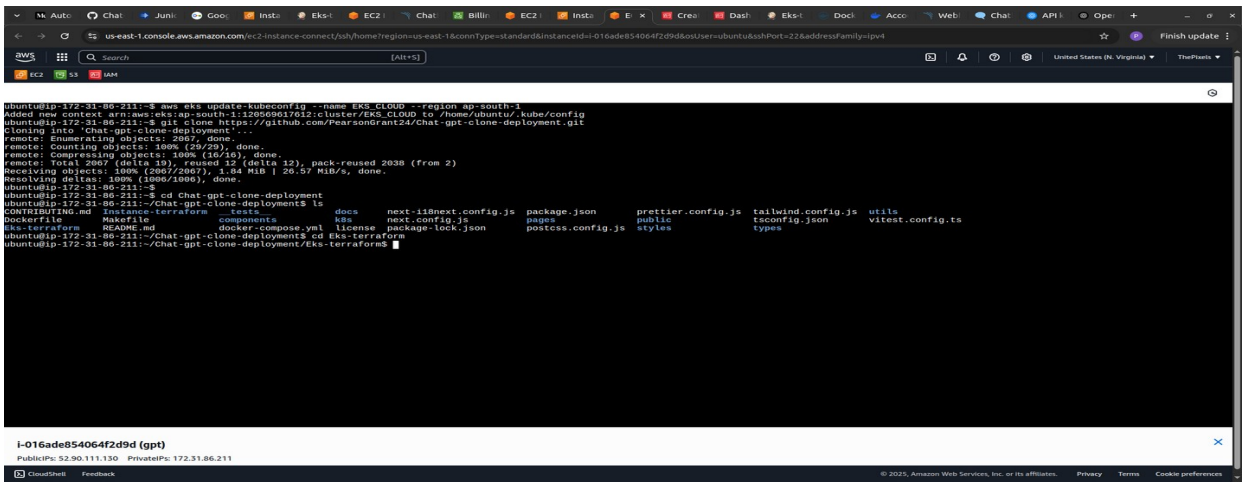
```
kubectl get nodes
```

Expected:

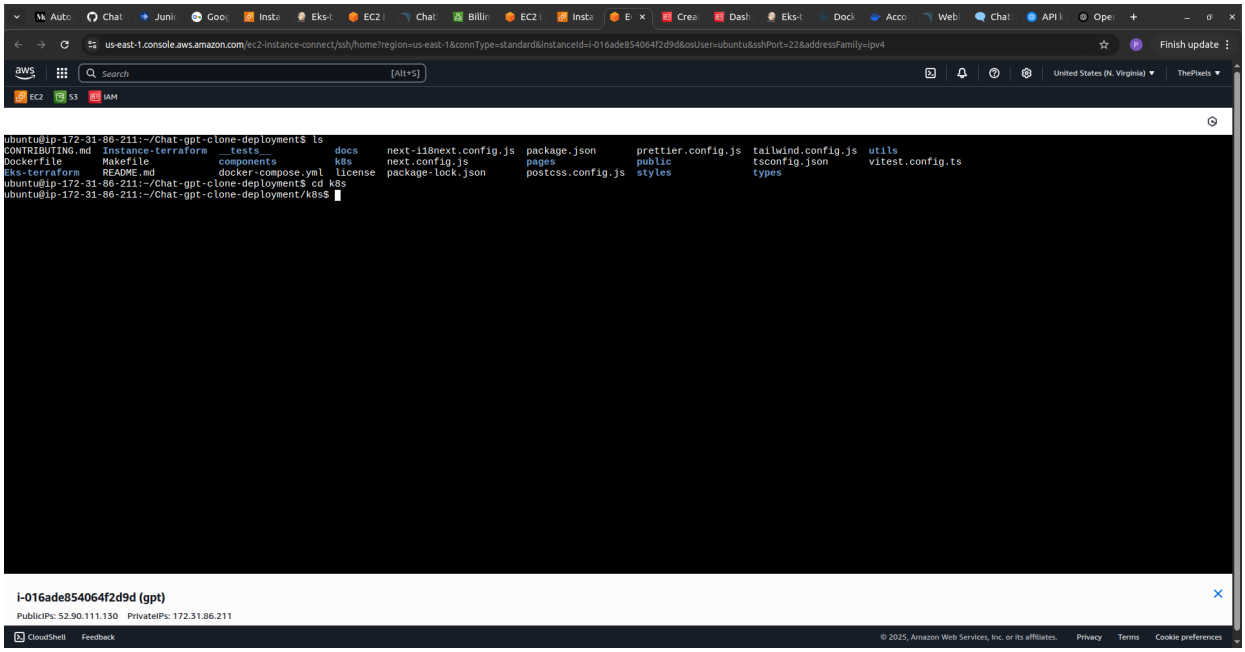
```
NAME                          STATUS   ROLES    AGE   VERSION
ip-192-168-x-x.ec2.internal Ready    <none>   5m    v1.xx
```

Clone the repository on your ec2

Go to k8s folder



-

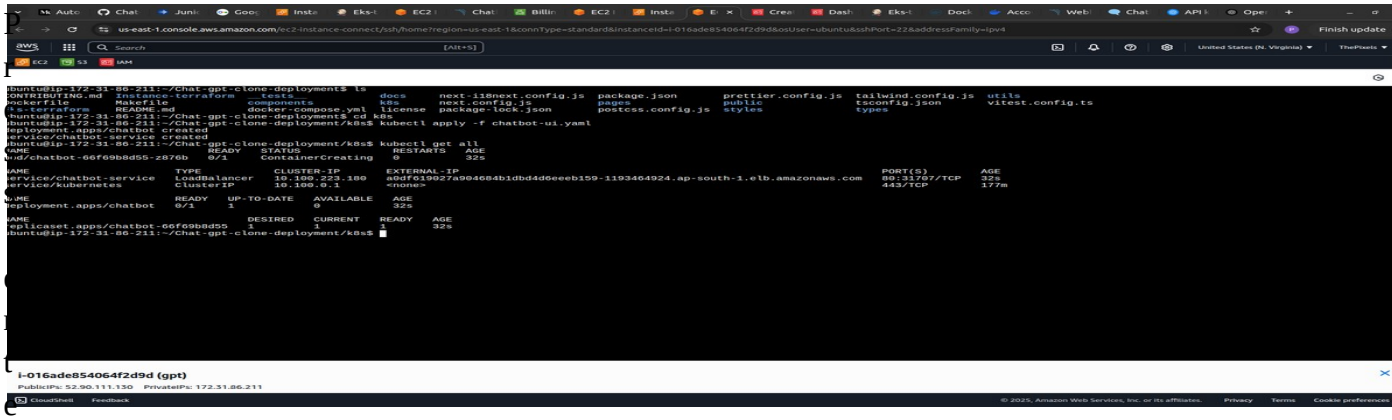


Expected:

You will find the deployment file for Kubernetes chatbot run the following command to deploy application

```
kubectl apply -f chatbot-ui.yaml

kubectl get all
```

Press enter or click to view image in full size

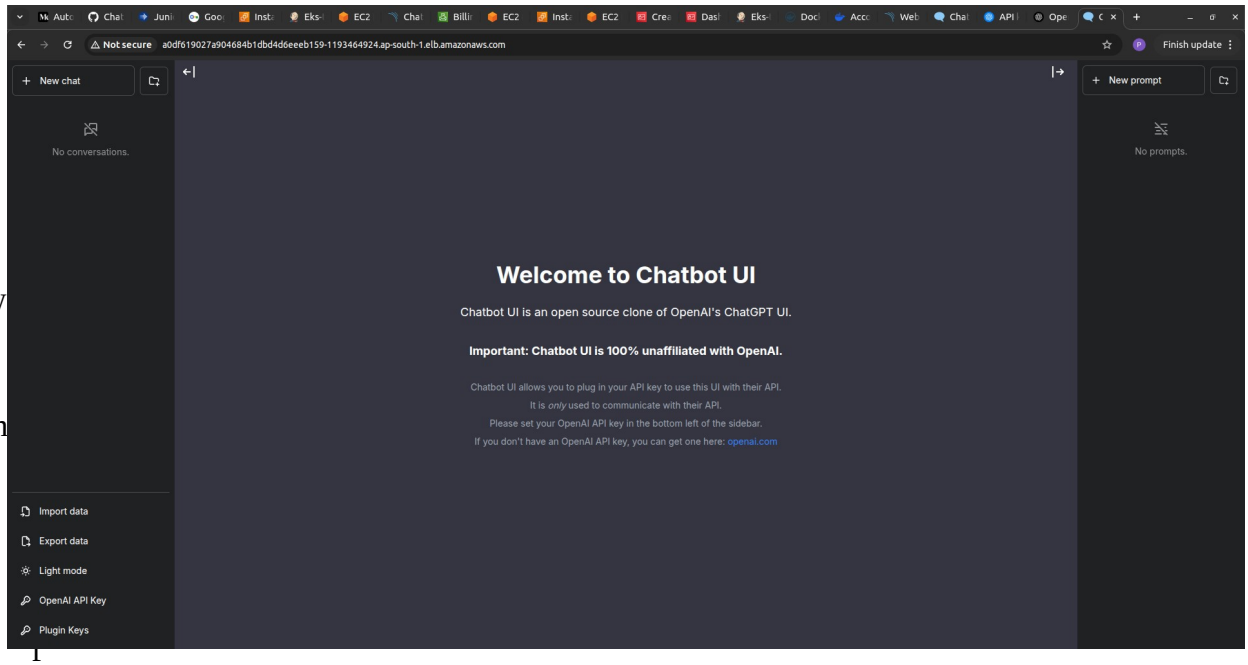


STEP 3.5 — Expose the App

Copy the load balancer external ip

Open browser: paste external ip

YOUR CHATGPT APP IS LIVE ON EKS



Load Balancer Ingress →

It is a mechanism that helps distribute incoming internet traffic among multiple servers or services, ensuring efficient and reliable delivery of requests.
It’s like having a receptionist at a busy office building entrance who guides visitors to different floors or departments, preventing overcrowding at any one location. In the digital world, a Load Balancer Ingress helps maintain a smooth user experience, improves application performance, and ensures that no single server becomes overwhelmed with too much traffic.

Service.yaml

`service.yaml`file is like a set of rules that helps computers find and talk to each other within a software application. It's like a directory that says, "Hey, this is how you can reach different parts of our application." It specifies how different parts of your application communicate and how other services or users can connect to them.

PHASE 6 — **Prometheus And Grafana on EKS**

This is the **final technical phase**.

- **Prometheus** is like a detective that constantly watches your software and gathers data about how it’s performing. It’s good at collecting metrics, like how fast your software is running or how many users are visiting your website.

- **Grafana**, on the other hand, is like a dashboard designer. It takes all the data collected by Prometheus and turns it into easy-to-read charts and graphs. This helps you see how well your software is doing at a glance and spot any problems quickly.

In other words, Prometheus collects the information, and Grafana makes it look pretty and understandable so you can make decisions about your software. They’re often used together to monitor and manage applications and infrastructure.

STEP 6.1 — Use EC2 Server for monitoring

- Go to ec2 your console and launch an instance having a base image of Ubuntu and with t2.medium specs because **Minimum Requirements to Install Prometheus :**

Expected:

- 2 CPU cores.
- 4 GB of memory.
- 20 GB of free disk space.

STEP 6.2 — Install Prometheus

- Paste the following script to your ec2 instance

```
sudo useradd — system — no-create-home — shell /bin/false prometheus

wget https://github.com/prometheus/prometheus/releases/download/v2.47.1/prometheus-2.47.1.linux-amd64.tar.gz
```

- Extract Prometheus files, move them, and create directories:

```
tar -xvf prometheus-2.47.1.linux-amd64.tar.gz

cd prometheus-2.47.1.linux-amd64/

sudo mkdir -p /data /etc/prometheus

sudo mv prometheus promtool /usr/local/bin/

sudo mv consoles/ console_libraries/ /etc/prometheus/

sudo mv prometheus.yml /etc/prometheus/prometheus.yml
```

- Set ownership for directories:

```
useradd prometheus

sudo chown -R prometheus:prometheus /etc/prometheus/ /data/
```

- Create a systemd unit configuration file for Prometheus:

```
sudo nano /etc/systemd/system/prometheus.service
```

- Add the following code to the `prometheus.service`file:

```
[Unit]

Description=Prometheus

Wants=network-online.target

After=network-online.target
StartLimitIntervalSec=500
StartLimitBurst=5[Service]
User=prometheus
Group=prometheus
Type=simple
Restart=on-failure
RestartSec=5s
ExecStart=/usr/local/bin/prometheus \
    --config.file=/etc/prometheus/prometheus.yml \
    --storage.tsdb.path=/data \
    --web.console.templates=/etc/prometheus/consoles \
    --web.console.libraries=/etc/prometheus/console_libraries \
```

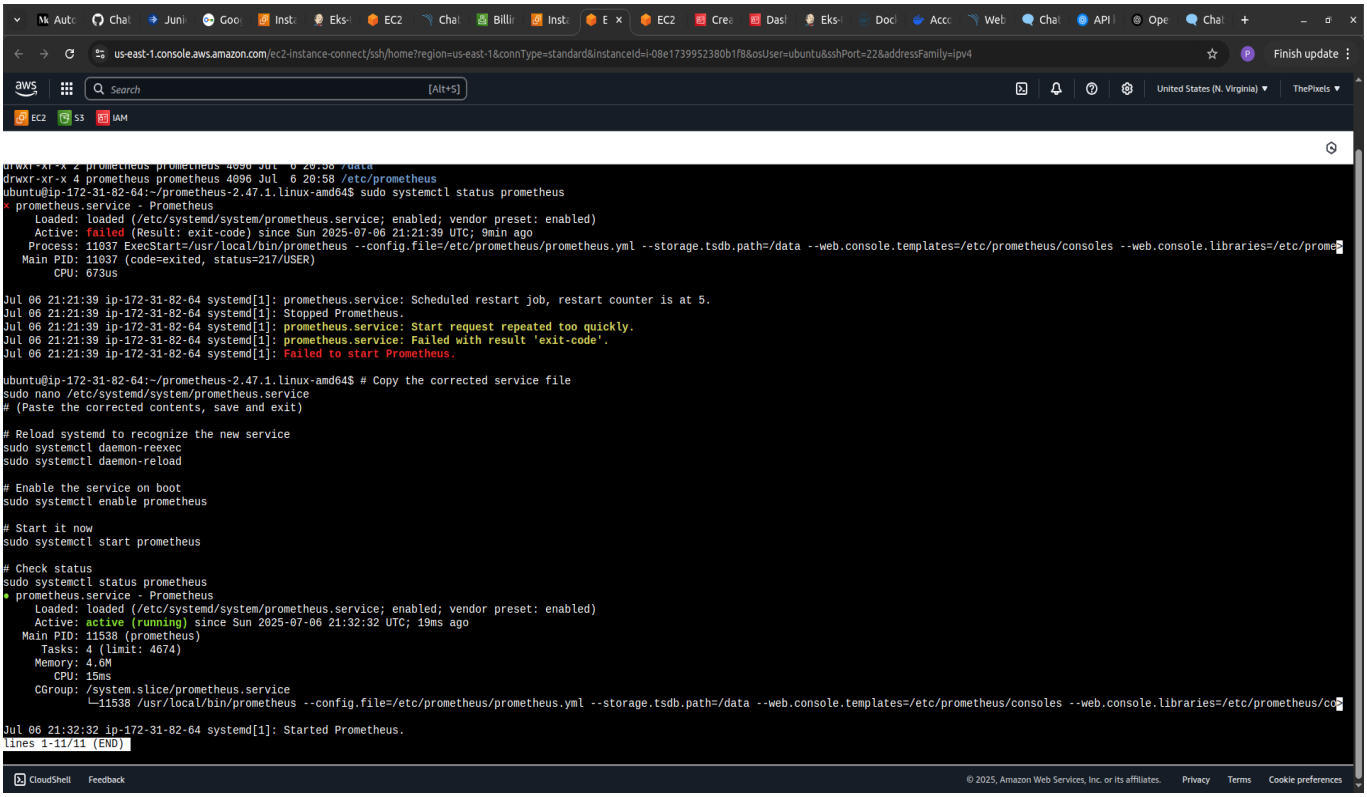
```
    --web.listen-address=0.0.0.0:9090 \
    --web.enable-lifecycle[Install]
WantedBy=multi-user.target
```

press →**ctrl+o** #for save and then **ctrl+x** #for exit from the file

---

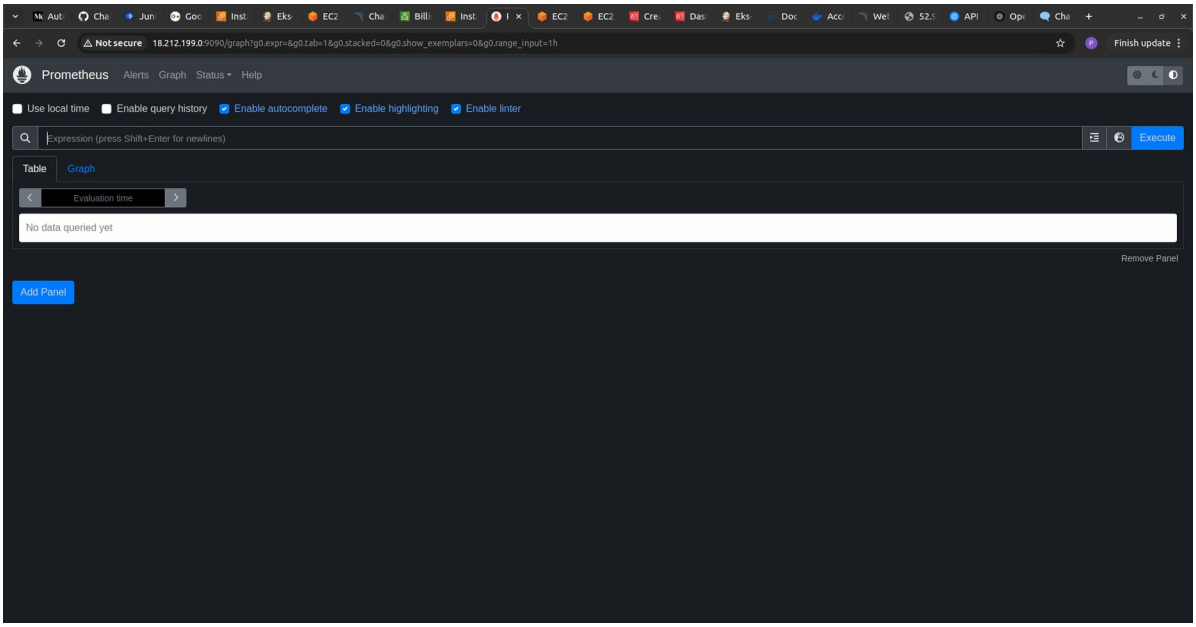
STEP 6.3 — Enable and start Prometheus

```
sudo systemctl enable prometheus
sudo systemctl start prometheus
sudo systemctl status prometheus
```



Copy public ip of the EC2 your prometheus is running at

go to → http://public_ip:9090 to see the webpage of Prometheus



STEP 6.1 — Install Node Exporter

What Node Exporter Does

- Runs on **every node**
- Collects:
    - CPU
    - Memory
    - Disk
    - Network

How Node Exporter is Deployed

Node Exporter is usually:

- A **DaemonSet**
- One pod per node

Create a system user for Node Exporter and download Node Exporter:

```
sudo useradd — system — no-create-home — shell /bin/false
node_exporter

wget
https://github.com/prometheus/node_exporter/releases/download/v1.6.1/
node_exporter-1.6.1.linux-amd64.tar.gz
```

Extract Node Exporter files, move the binary, and clean up:

```
tar -xvf node_exporter-1.6.1.linux-amd64.tar.gz
sudo mv node_exporter-1.6.1.linux-amd64/node_exporter /usr/local/bin/
rm -rf node_exporter*
```

Create a systemd unit configuration file for Node Exporter:

```
sudo nano /etc/systemd/system/node_exporter.service
```

Add the following code to the `node_exporter.service` file:

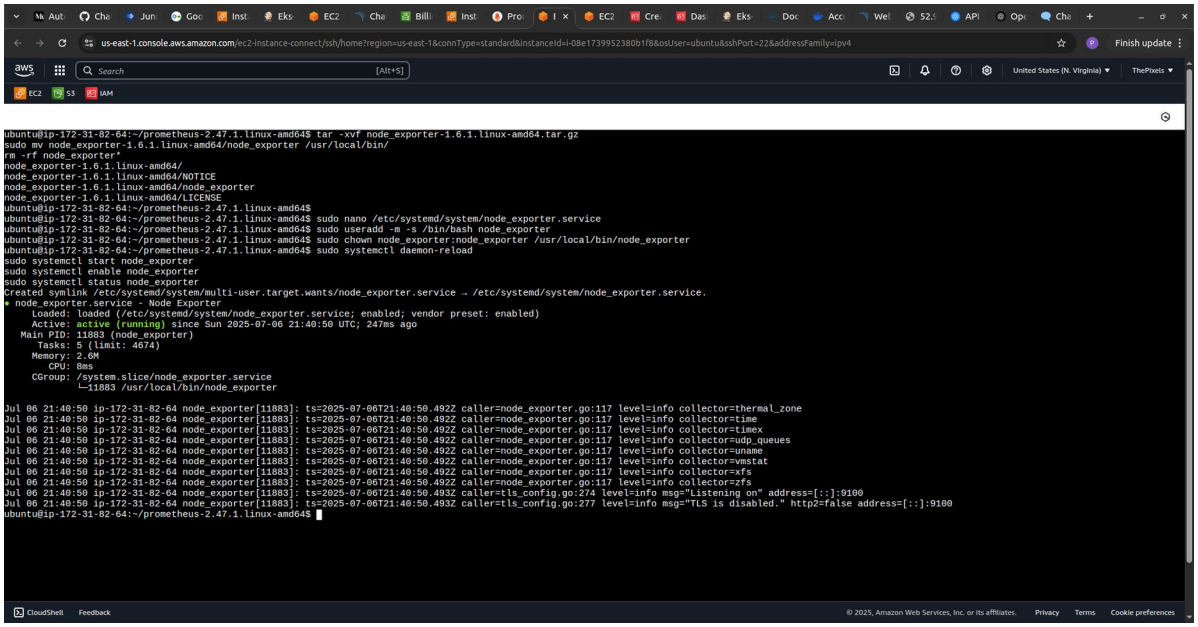
```
[Unit]

Description=Node Exporter

After=network.target
[Service]

User=node_exporter

Group=node_exporter

Type=simple

ExecStart=/usr/local/bin/node_exporter

[Install]

WantedBy=default.target
```

Enable and start Node Exporter

```
sudo useradd -m -s /bin/bash node_exporter

sudo chown node_exporter:node_exporter /usr/local/bin/node_exporter
```

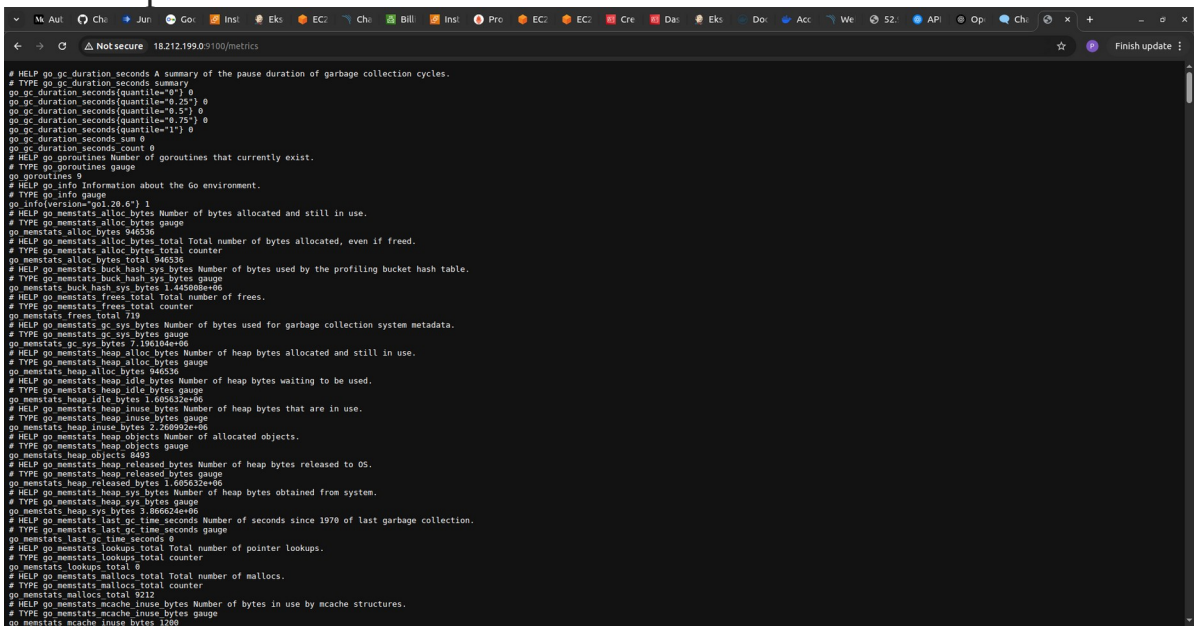
```
sudo systemctl daemon-reload

sudo systemctl start node_exporter

sudo systemctl enable node_exporter

sudo systemctl status node_exporter
```

Node Exporter service is now running



Check metrics

Node Exporter metrics is now accessible in Prometheus.



STEP 6.3 — **Configure Prometheus Plugin Integration:**

- go to your EC2 and run →

```
cd /etc/prometheus
```

- you have to edit the prometheus.yml file to monitor anything
- add the code with proper indentation like this:

```
scrape_configs:

 - job_name: 'node_exporter'

   static_configs:

    - targets: ['localhost:9100']

 - job_name: 'jenkins'

   metrics_path: '/prometheus'

   static_configs:

    - targets: ['<your-jenkins-ip>:<your-jenkins-port>']
```

press esc+:wq to save and exit

Check the validity of the configuration file

```
promtool check config /etc/prometheus/prometheus.yml
```

o/p : success

Reload the Prometheus configuration without restarting

curl -X POST http://localhost:9090/-/reload

Prometheus -> status - targets

you will find three targets present as we entered in the yaml file for monitoring

Prometheus targets dashboard

STEP 6.3 — **Setup Grafana**

Install Dependencies:

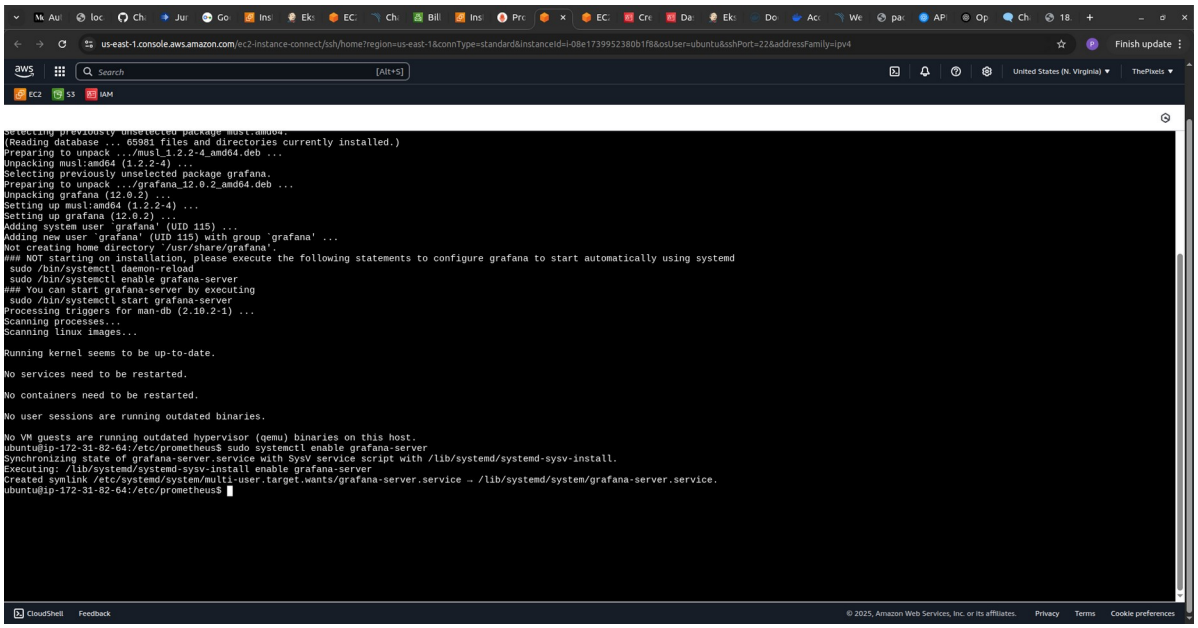
```
sudo apt-get update

sudo apt-get install -y apt-transport-https software-properties-common
```

Add the GPG Key for Grafana:

```
wget -q -O —  https://packages.grafana.com/gpg.key | sudo apt-key add -
```

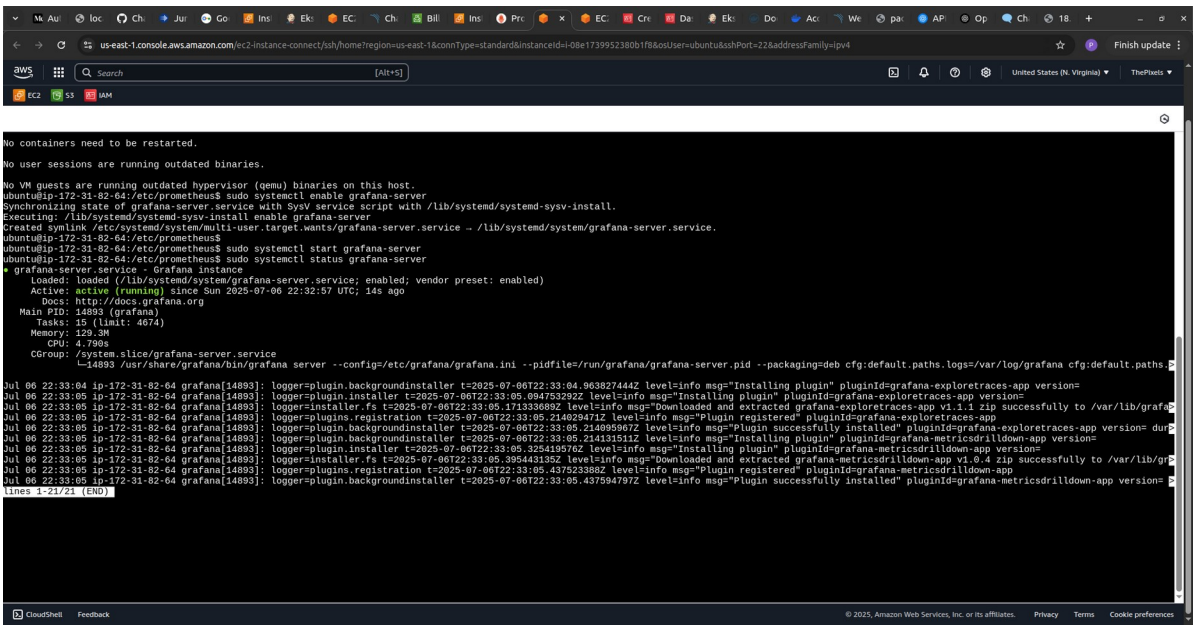
Add the repository for Grafana stable releases:

```
echo “deb https://packages.grafana.com/oss/deb stable main” | sudo tee — a /etc/apt/sources.list.d/grafana.list
```



Update the package list, install and start Grafana:

```
sudo apt-get update

sudo apt-get -y install grafana

sudo systemctl enable grafana-server

sudo systemctl start grafana-server

sudo systemctl status grafana-server
```

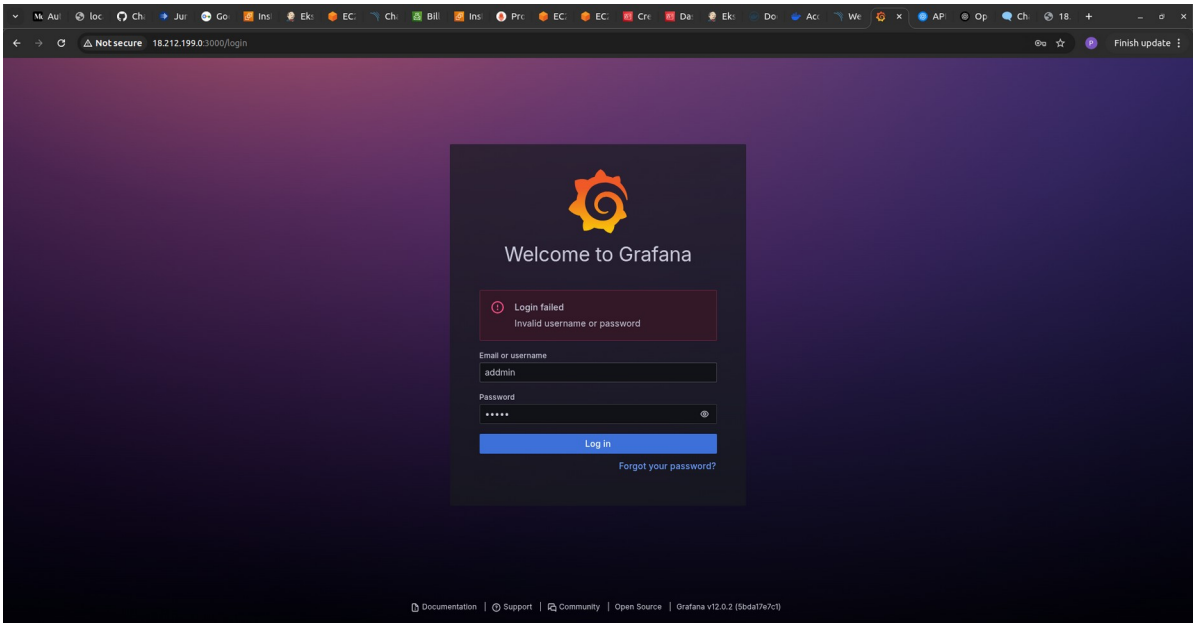


Open Grafana on port 3000

http://<public_ip>:3000

username: admin

password: admin

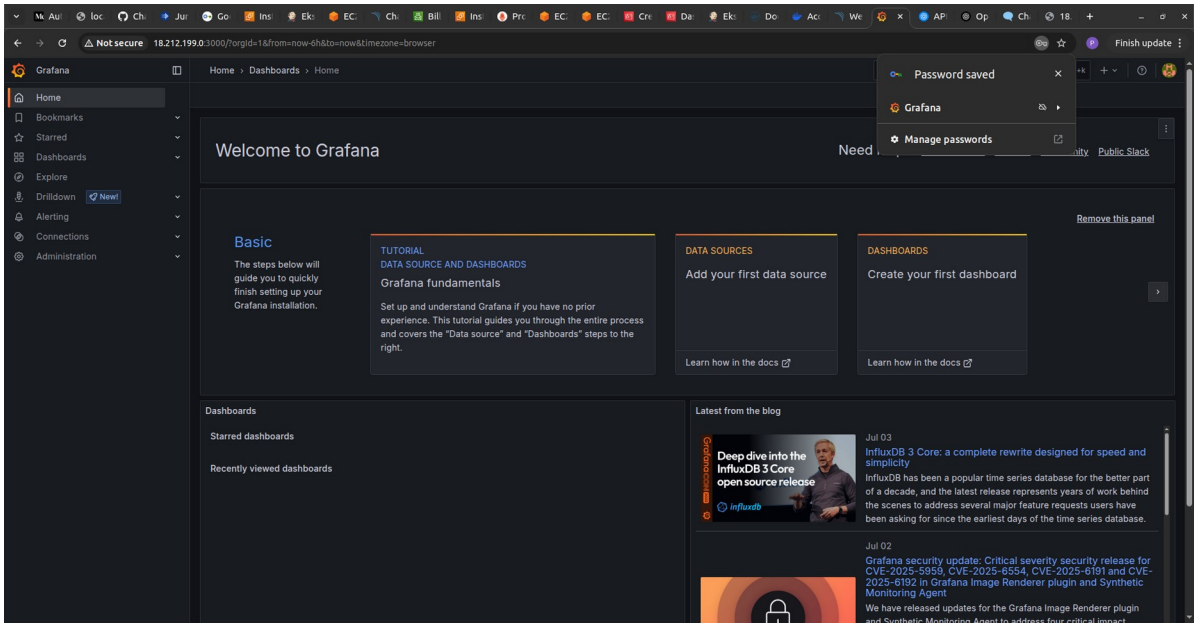


STEP 6.3 — **Add Prometheus as Data Source**

To visualize metrics, you need to add a data source.

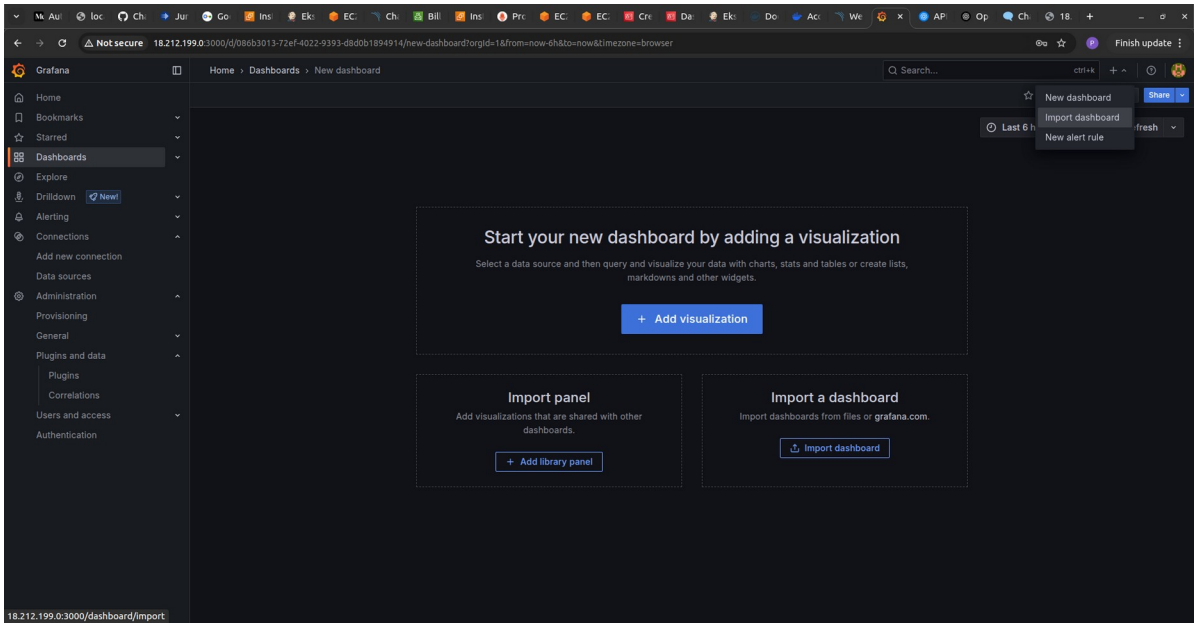
Follow these steps:

- Click on the gear icon ( ) in the left sidebar to open the “Configuration” menu.
- Select “Data Sources.”
- Click on the “Add data source” button.



- Type Prometheus
- URL:
- http://localhost:9090(assuming Prometheus is running on the same server).
- Save & Test

STEP 6.5 — Import Node Exporter Dashboard



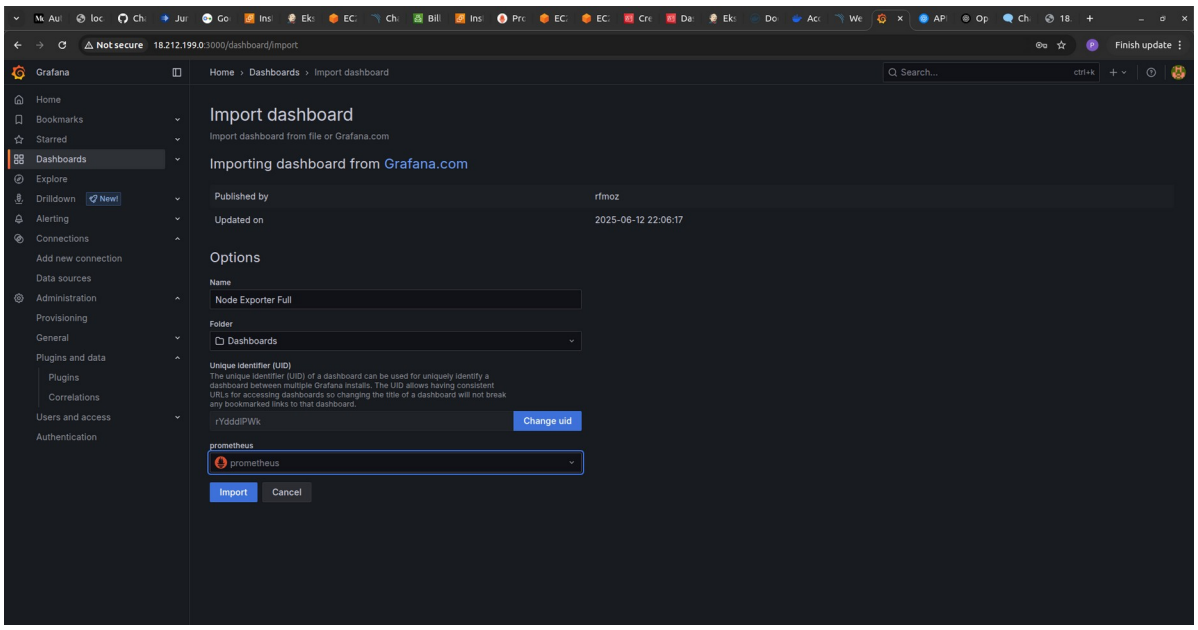
Grafana → Dashboards → Import

Use Dashboard ID:

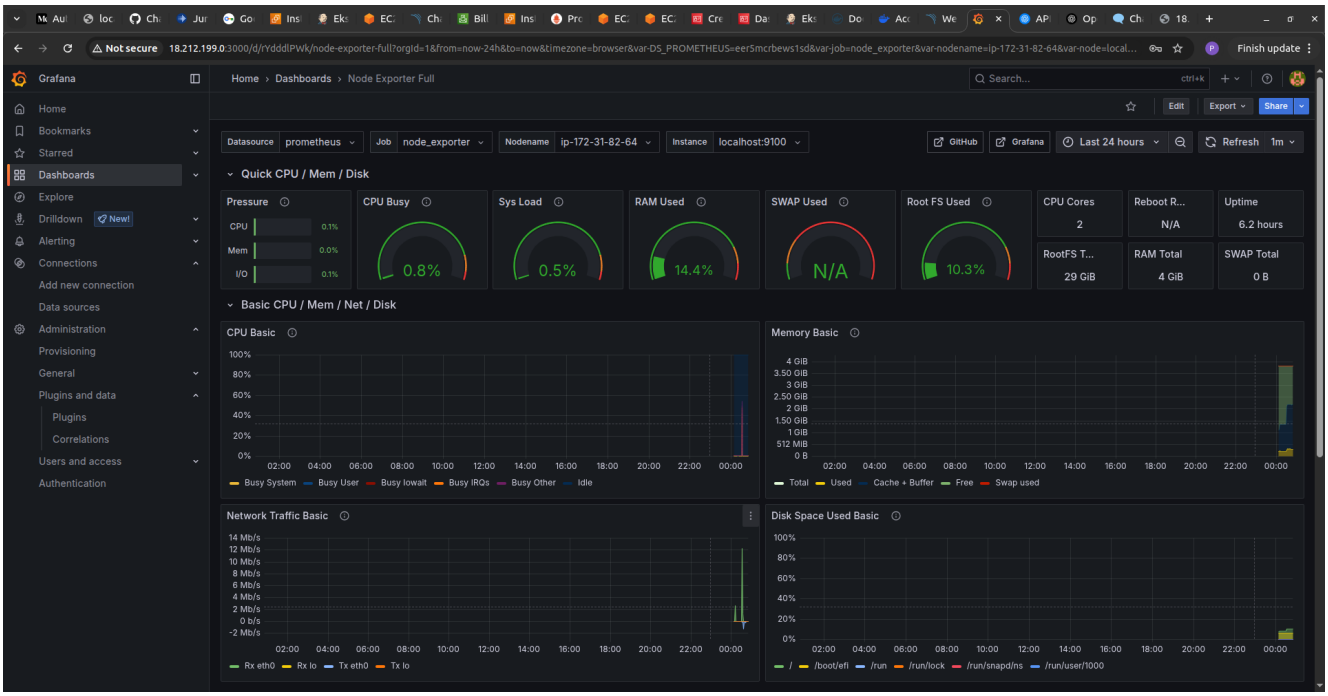
`1860`

You now see:

- CPU
- Memory
- Disk
- Network per node



- Click: Import
- Done



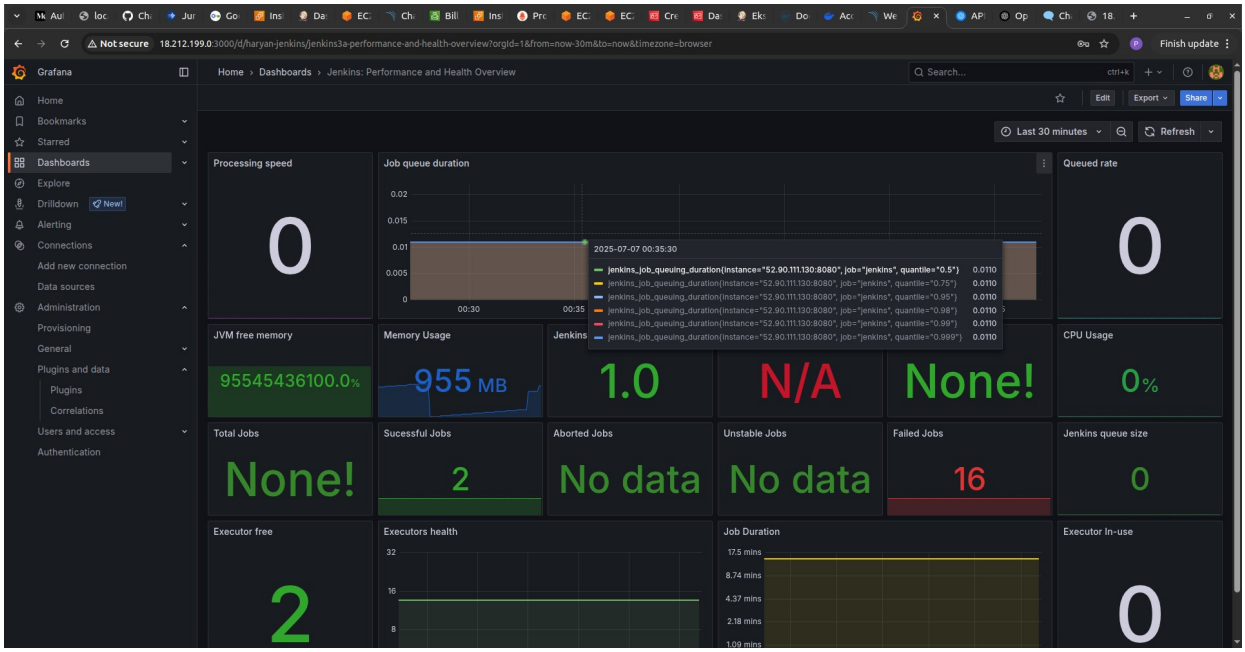
STEP 6.5 — Jenkins monitoring dashboard (ID: 9964)

manage Jenkins →system →search for Prometheus → apply →save

import a dashboard for Jenkins

+ (plus) icon → Dashboard → Import (dashboard option) → Enter Dashboard ID → Load Button

Select the data source you added (Prometheus) from the dropdown.



▪

FINAL Phase 7→ Infrastructure Cleanup

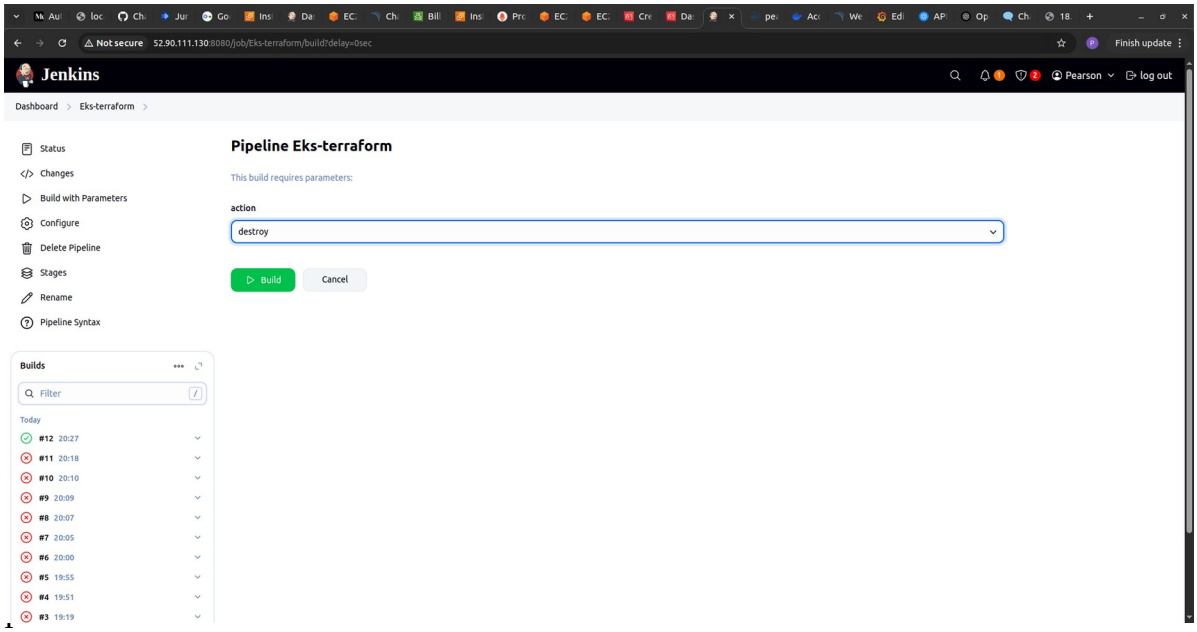
Destroy Infrastructure by:

```
kubectl delete deployment chatbot
```

Go to your Jenkins and locate the eks-terraform pipeline Click on build with parameter and select: destroy and then click on build
It will destroy your eks cluster and node group

3. After EKS cluster deletion let’s delete the base instance → “gpt-instance and monitoring instance

Destroy : gpt-instance, monitoring instance together with all their supporting infrastructure



Press enter or click to view image in full size

go to your local machine, in the folder where instance terraform is located and then run

```
terraform destroy --auto-approve
```

Conclusion

This project showcases a **realistic DevOps implementation** combining Terraform, Jenkins, Kubernetes, and observability tooling. It highlights not just deployment success, but disciplined engineering practices around automation, security, monitoring, and cost control—core expectations for modern DevOps engineers.