Deploy Netflix Clone on Kubernetes

Project link:
https://medium.com/@aakibkhan1/project-4-deploy-netflix-clone-on-kubernetes-9ae6091b88bd

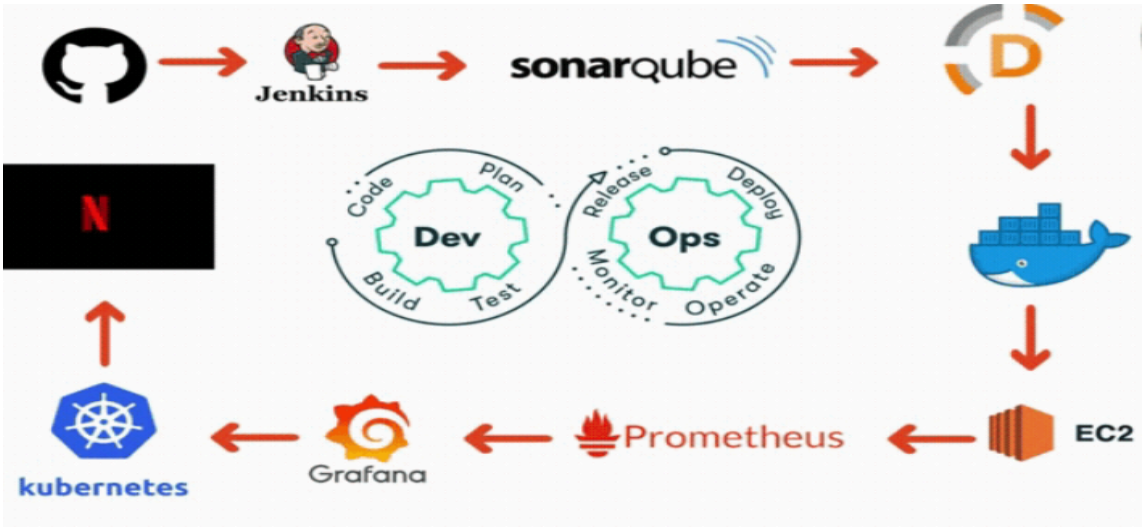
Phase 1 →deploy netflix locally on ec2 (t2.large)

Phase 2 → Implementation of security with sonarqube and trivy

Phase 3 →Now we automate the whole deployment using by jenkins pipeline

Phase 4 →Monitoring via Prometheus and grafana

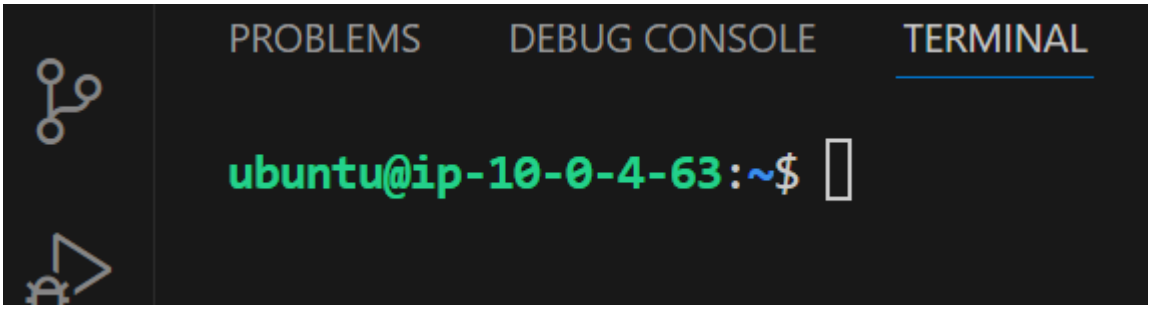
Phase 5 → Kubernetes →



Phase 1 → deploy netflix locally on ec2 (t2.large)

1. go to aws console and launch ubuntu 22.04 with t2.large and 25 gb of storage allocated with it don't forget to enable public ip in vpc settings

Step 1 →setup ec2



3. run the following commands

a. sudo su

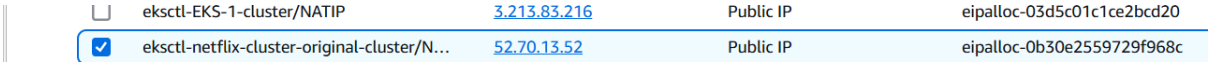
b. apt update

clone the github repo by

c. git clone

https://github.com/Aakibgithuber/Deploy-Netflix-Clone-on-Kubernetes

Make sure to create an elastic ip address to associate with your instance



Step 2 → setup docker and build images

run the following commands to install docker→

a. **apt-get install docker.io**

b. **usermod -aG docker $USER** # Replace with your username e.g 'ubuntu'

c. **newgrp docker**

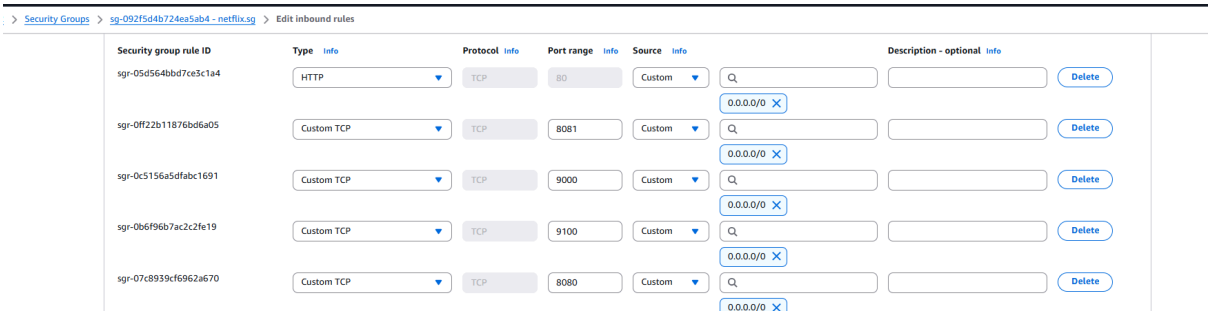
d. **sudo chmod 777 /var/run/docker.sock** → #is used to grant full read, write, and execute permissions to all users for the Docker socket file, which allows unrestricted access to the Docker daemon and is a significant security risk.

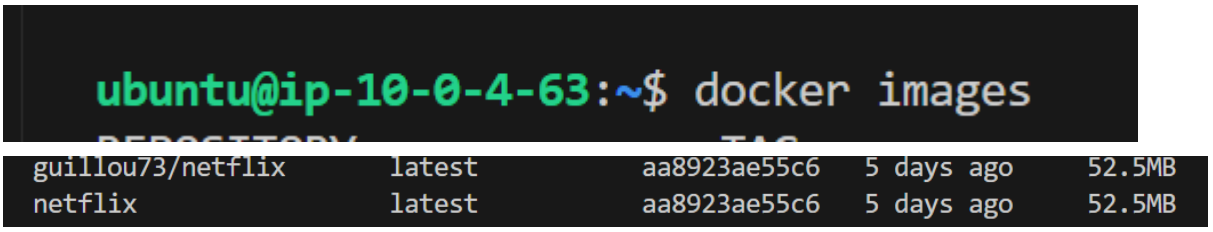
run the following commands to build and run docker container →

a. **docker build -t netflix .**

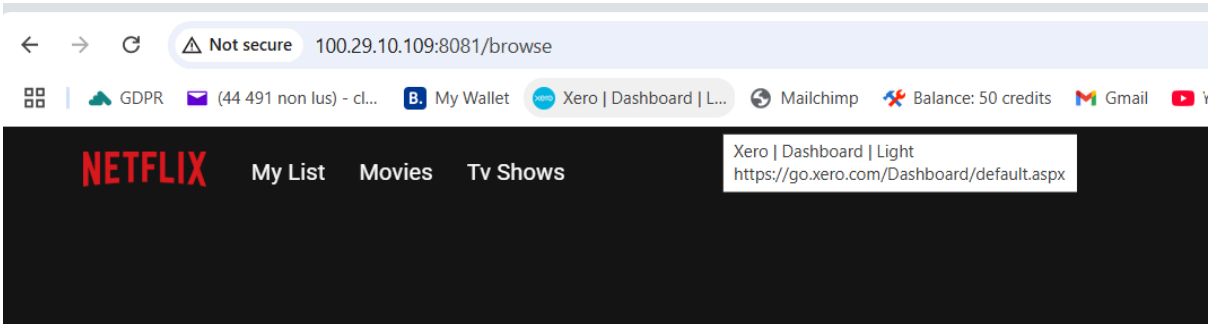
b. **docker run -d — name netflix -p 8081:80 netflix:latest** → this maps the container port to your ec2 port

c. **go to your ec2 →security groups →open the port 8081** by adding rule custom tcp = 8081





your aplication is running go to your ec2 copy public ip and browse http://your_public_ip:8081 and it's open like this



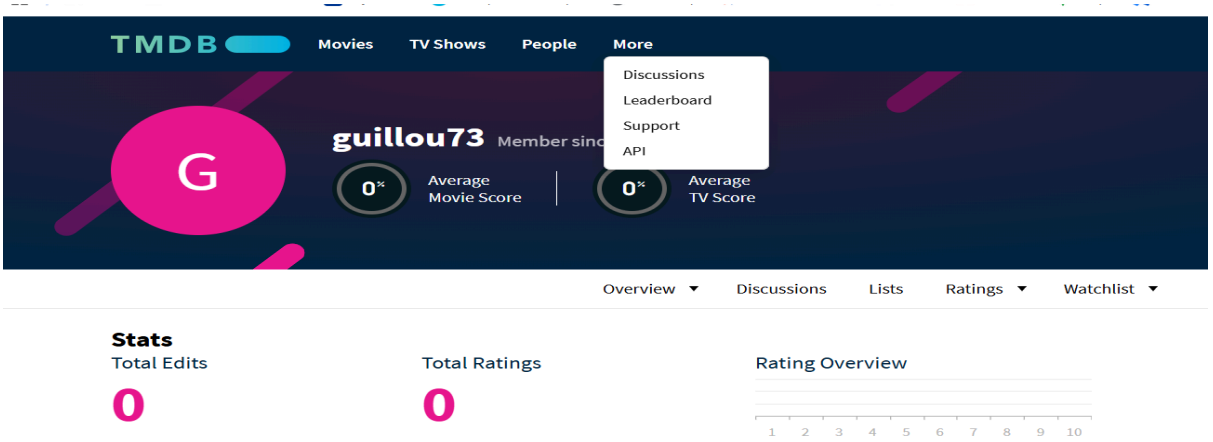
Its shows a blank page of netflix because you don't have a api that's communicate with netflix database let's solve this

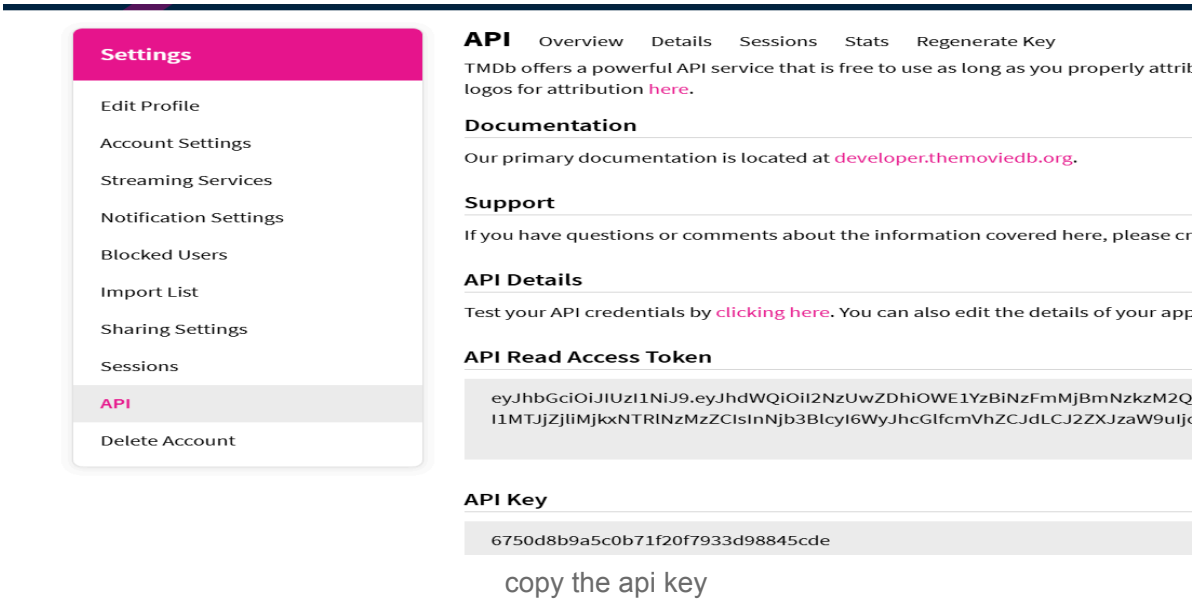
what is an API→

An API (Application Programming Interface) is like a menu for a restaurant that lets you order food (data or functions) from a software system or application, allowing different programs to talk to each other and share information in a structured way.

Step 3 → setup netflix API

- Open a web browser and navigate to TMDB (The Movie Database) website.
- Click on sign up
- Now enter your username and pass for sign in then go to your profile and select "Settings."
- Click on "API" from the left-side panel.
- Create a new API key by clicking "generate new api key " and accepting the terms and conditions.
- Provide the required basic details such as name and website url of netflix and click "Submit."
- You will receive your TMDB API key





Now delete the existing image by

a. **docker stop <containerid>**

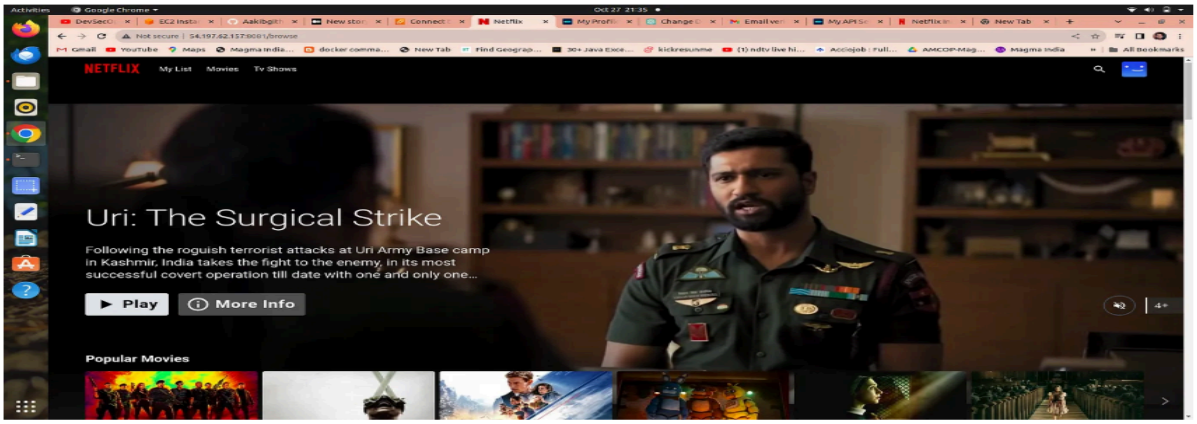
b. **docker rmi -f netflix**

Run the new container by following command

1. docker build -t netflix:latest — build-arg TMDB_V3_API_KEY=your_api_key .
2. docker run -d -p 8081:80 netflix

your container is created

now again browse the same url you will see the whole netflix database is connected to your netflix clone app



congrats your aplication is running

Phase 2 → Implementation of security with sonarqube and trivy

what is sonarqube →

SonarQube is like a “code quality detective” tool for software developers that scans your code to find and report issues, bugs, security vulnerabilities, and code smells to help you write better and more reliable software. It provides insights and recommendations to improve the overall quality and maintainability of your codebase

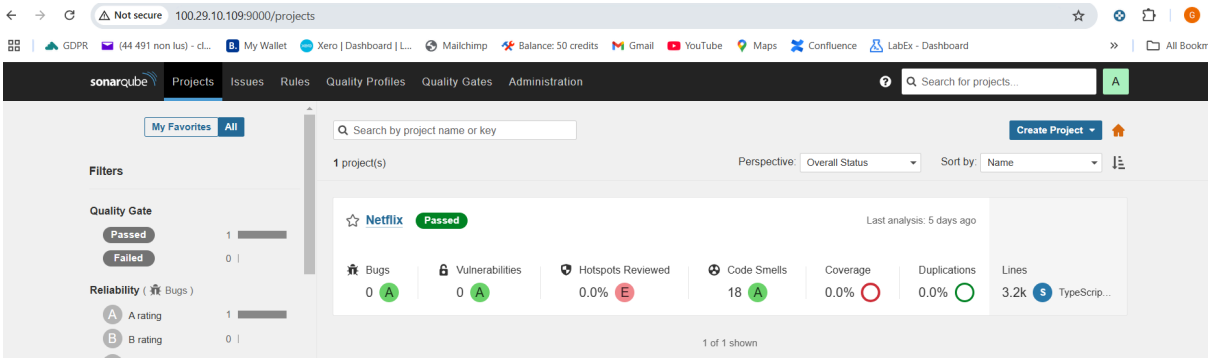
what is trivy →

Trivy is like a “security guard” for your software that checks for vulnerabilities in the components (like libraries) your application uses, helping you find and fix potential security problems to keep your software safe from attacks and threats. It’s a tool that scans your software for known security issues and provides information on how to address them.

step 1 → setup sonarqube on your ec2

run the following commands to install and run the container of sonarqube on port no. 9000

a. **docker run -d — name sonar -p 9000:9000 sonarqube:lts-community**



Step 2 → setup Trivy

run the following commands

a. sudo apt-get install wget apt-transport-https gnupg lsb-release

b. wget -qO — https://aquasecurity.github.io/trivy-repo/deb/public.key | sudo apt-key add -

c. echo deb https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc) main | sudo tee -a /etc/apt/sources.list.d/trivy.list

d. sudo apt-get update

e. sudo apt-get install trivy

now your trivy is ready to check and scan your image for any vulnerabilities

to check run the following commands

a. trivy image <image id>

```
ubuntu@ip-10-0-4-63:~$ trivy image aa8923ae55c6
2025-01-18T10:15:08Z    INFO    [vulndb] Need to update DB
2025-01-18T10:15:08Z    INFO    [vulndb] Downloading vulnerability DB...
2025-01-18T10:15:08Z    INFO    [vulndb] Downloading artifact...        repo="mirror.gcr.io/aquasec/trivy-db:2"
58.63 MiB / 58.63 MiB [-------------------------------------------------------------------------------------] 100.00% 26.04 MiB p/s 2.5s
2025-01-18T10:15:12Z    INFO    [vulndb] Artifact successfully downloaded       repo="mirror.gcr.io/aquasec/trivy-db:2"
2025-01-18T10:15:12Z    INFO    [vuln] Vulnerability scanning is enabled
2025-01-18T10:15:12Z    INFO    [secret] Secret scanning is enabled
2025-01-18T10:15:12Z    INFO    [secret] If your scanning is slow, please try '--scanners vuln' to disable secret scanning
2025-01-18T10:15:12Z    INFO    [secret] Please see also https://aquasecurity.github.io/trivy/v0.58/docs/scanner/secret#recommendation f
or faster secret detection
2025-01-18T10:15:13Z    INFO    Detected OS     family="alpine" version="3.20.5"
2025-01-18T10:15:13Z    INFO    [alpine] Detecting vulnerabilities...   os_version="3.20" repository="3.20" pkg_num=66
2025-01-18T10:15:13Z    INFO    Number of language-specific files       num=0

aa8923ae55c6 (alpine 3.20.5)

Total: 0 (UNKNOWN: 0, LOW: 0, MEDIUM: 0, HIGH: 0, CRITICAL: 0)
```

here's the scan of your image of netflix

Phase 3 →Now we automate the whole deployment using by jenkins pipeline

Step 1 →install and configure jenkins via terminal …jenkins also required java for run itself so we install java and then jenkins

commands to run on the terminal ⟶

setup java

1. sudo apt update
2. sudo apt install fontconfig openjdk-17-jre
3. java -version

   output →openjdk version "17.0.8" 2023–07–18

   OpenJDK Runtime Environment (build 17.0.8+7-Debian-1deb12u1)

   OpenJDK 64-Bit Server VM (build 17.0.8+7-Debian-1deb12u1, mixed mode, sharing)

Setup jenkins

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

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

2. 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

3. sudo apt-get update
4. sudo apt-get install jenkins
5. sudo systemctl start jenkins
6. sudo systemctl enable jenkins
7. Access Jenkins in a web browser using the public IP of your EC2 instance

   http://publicIp:8080

Install some suggested plugins for pipeline to run without errors

1 Eclipse Temurin Installer (Install without restart)

2 SonarQube Scanner (Install without restart)

3 NodeJs Plugin (Install Without restart)

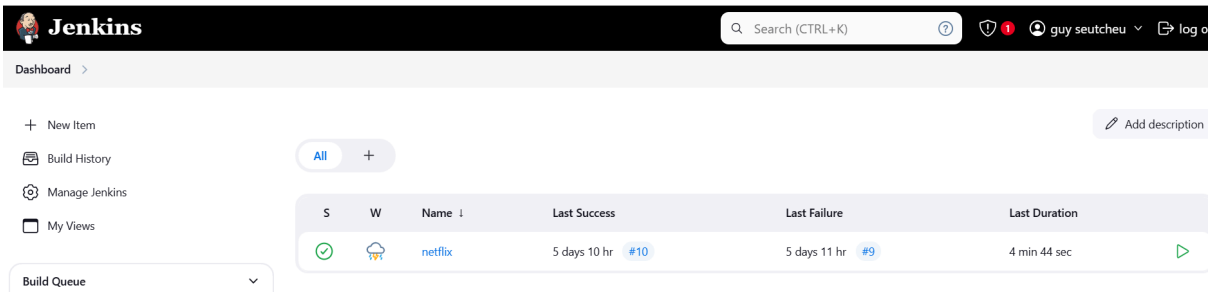
4 Email Extension Plugin

6. **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.

7. Prometheus metrics →to monitor jenkins on grafana dashboard

8. Download all the docker related plugins

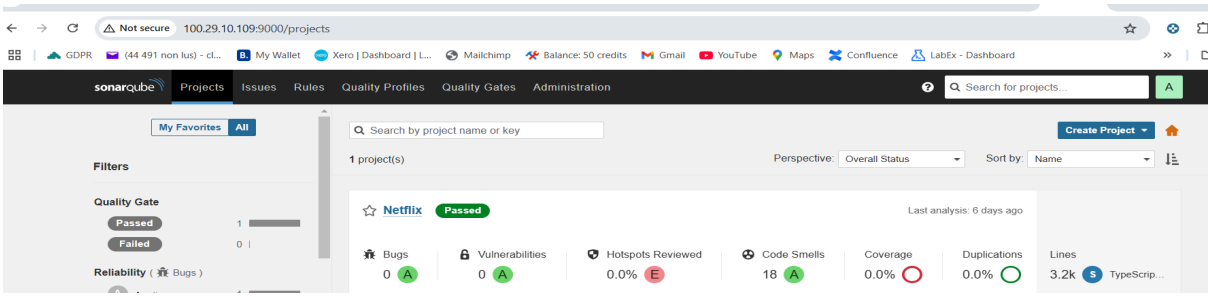


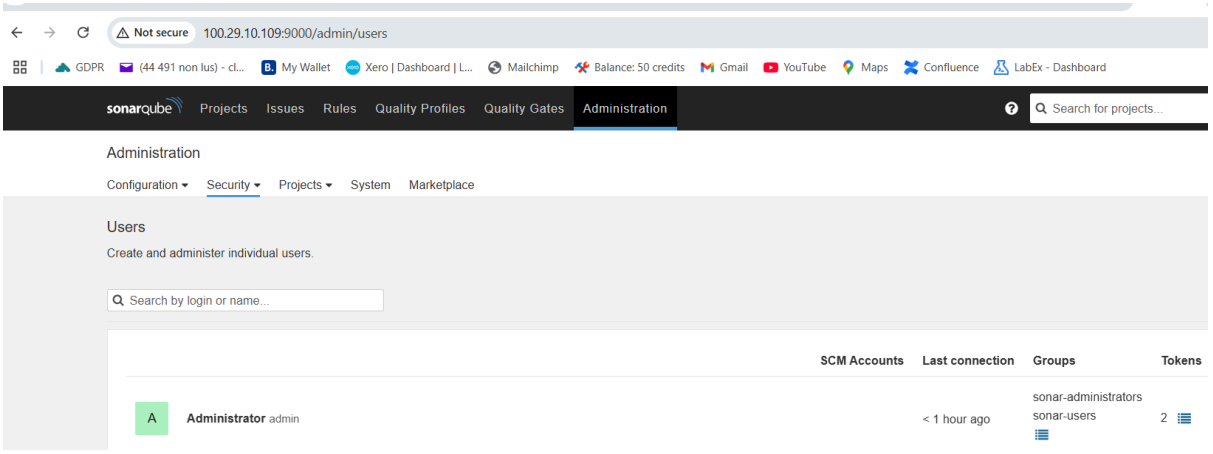
Step 2 → add credentials of Sonarqube and Docker

1st we generate a token for sonarqube to use in jenkins credentials as secret text

setup sonarqube credentials

1. go to http://publicip:9000
2. now enter your username and password
3. click on security →users →token →generate token





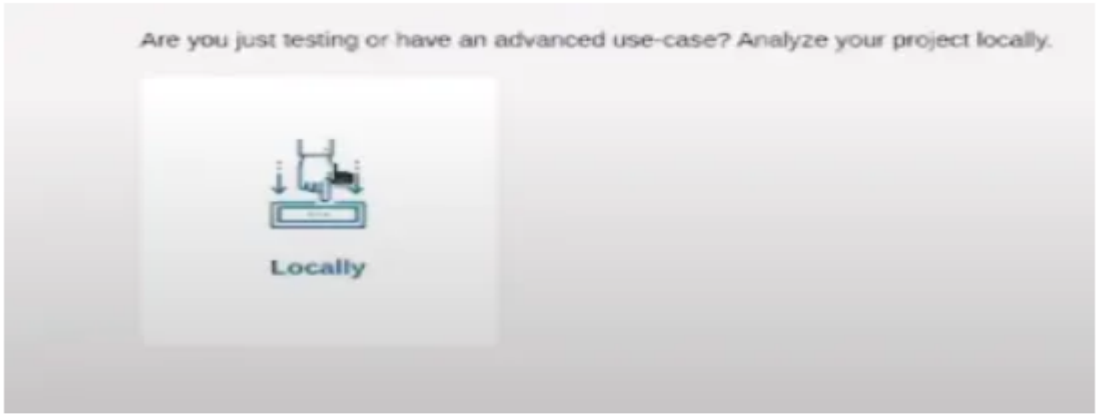
4. copy the token and go to your jenkins →manage jenkins →credentials →global →add credentials

5. select secret text from dropdown

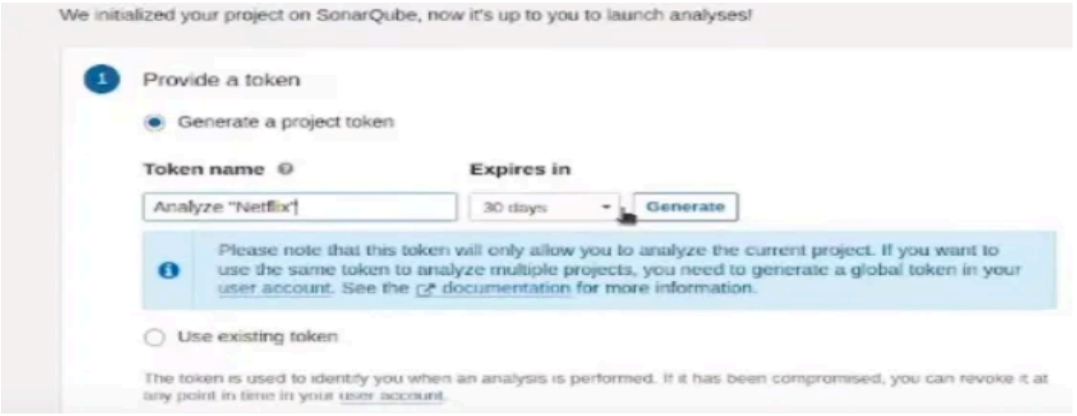
6. secret text ==your token , id =sonar-token →click on create

setup projects in sonarqube for jenkins

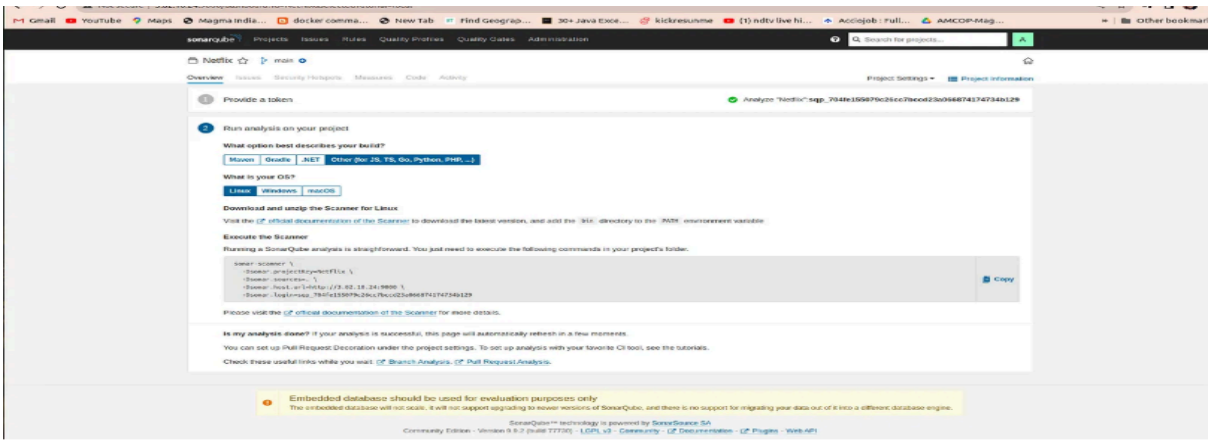
1. go to your sonarqube server
2. click on projects
3. in the name field type Netflix
4. click on set up



click on this option



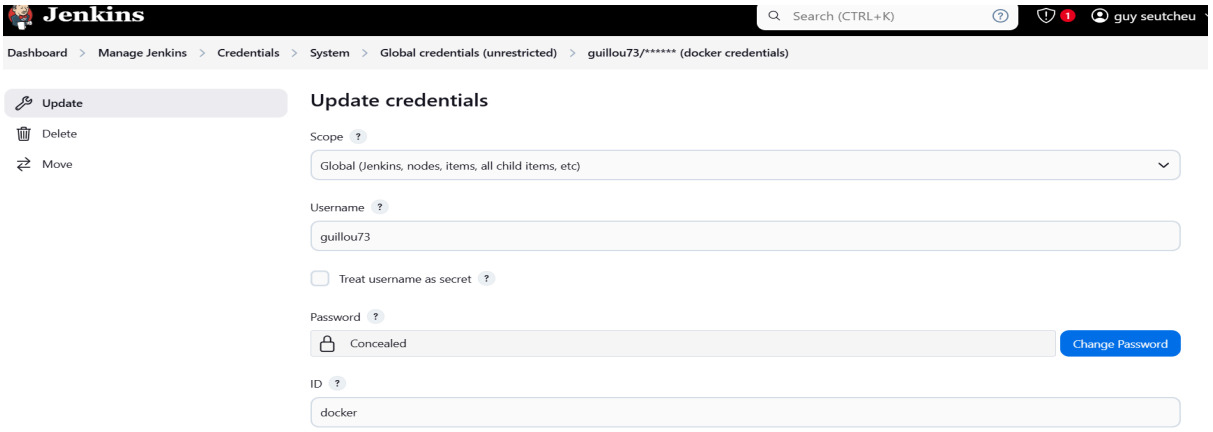
click on generate



select the os and the following commands used in jenkins pipeline code that set sonarqube to watch over jenkins

Setup docker credentials

1. go to your jenkins →manage jenkins →credentials →global →add credentials
2. provide your username and password of your dockerhub
3. id==docker



Step 3→Now we are going to setup tools for jenkins

go to manage 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 jdk17

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

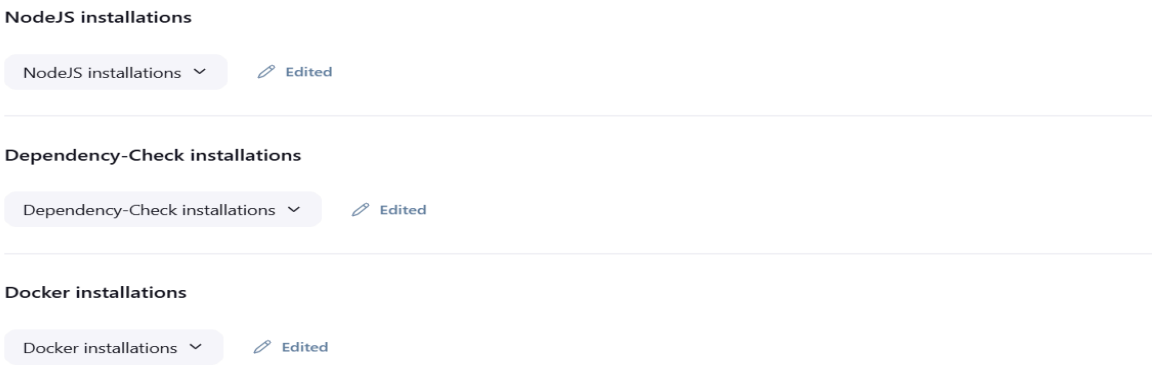
d. add sonarqube →

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

e. add owasp dependency check →

Adding the Dependency-Check plugin in the “Tools” section of Jenkins allows you to perform automated security checks on the dependencies used by your application

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



Step 4 →Configure global setting for sonarube

1. go to manage jenkins →Configure global setting →add sonarqube servers
2. **name ==sonar-server**
3. **server_url==http://public_ip:9000**
4. **server authentication token == sonar-token**

Step 5 →let's run the Pipeline →

1. **go to new item →select pipeline →in the name section type netflix**
2. **scroll down to the pipeline script and copy paste the following code**

```
pipeline{

    agent any

    tools{

        jdk 'jdk17'

        nodejs 'node16'

    }

    environment {

        SCANNER_HOME=tool 'sonar-scanner'

    }

    stages {

        stage('clean workspace'){

            steps{
```

```
                cleanWs()

            }

        }

        stage('Checkout from Git'){

            steps{

                git branch: 'main', url:
'https://github.com/Aakibgithuber/Deploy-Netflix-Clone-on-Kube
rnetes.git'

            }

        }

        stage("Sonarqube Analysis "){

            steps{

                withSonarQubeEnv('sonar-server') {

                    sh ''' $SCANNER_HOME/bin/sonar-scanner
-Dsonar.projectName=Netflix \
```

```
                    -Dsonar.projectKey=Netflix '''

                }

            }

        }

        stage("quality gate"){

            steps {

                script {

                    waitForQualityGate abortPipeline: false,
credentialsId: 'Sonar-token'

                }

            }

        }

        stage('Install Dependencies') {
```

```
            steps {

                sh "npm install"

            }

        }

        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.txt"

            }

        }

        stage("Docker Build & Push"){

            steps{

                script{

                    withDockerRegistry(credentialsId: 'docker',
toolName: 'docker'){

                        sh "docker build --build-arg
TMDB_V3_API_KEY=<yourapikey> -t netflix ."

                        sh "docker tag netflix
aakibkhan1212/netflix:latest "

                        sh "docker push
aakibkhan1212/netflix:latest "

                    }
```

```
                }

            }

        }

        stage("TRIVY"){

            steps{

                sh "trivy image nasi101/netflix:latest > trivyimage.txt"

            }

        }

        stage('Deploy to container'){

            steps{

                sh 'docker run -d --name netflix -p 8081:80 nasi101/netflix:latest'

            }
```

```
        }

    }

}
```

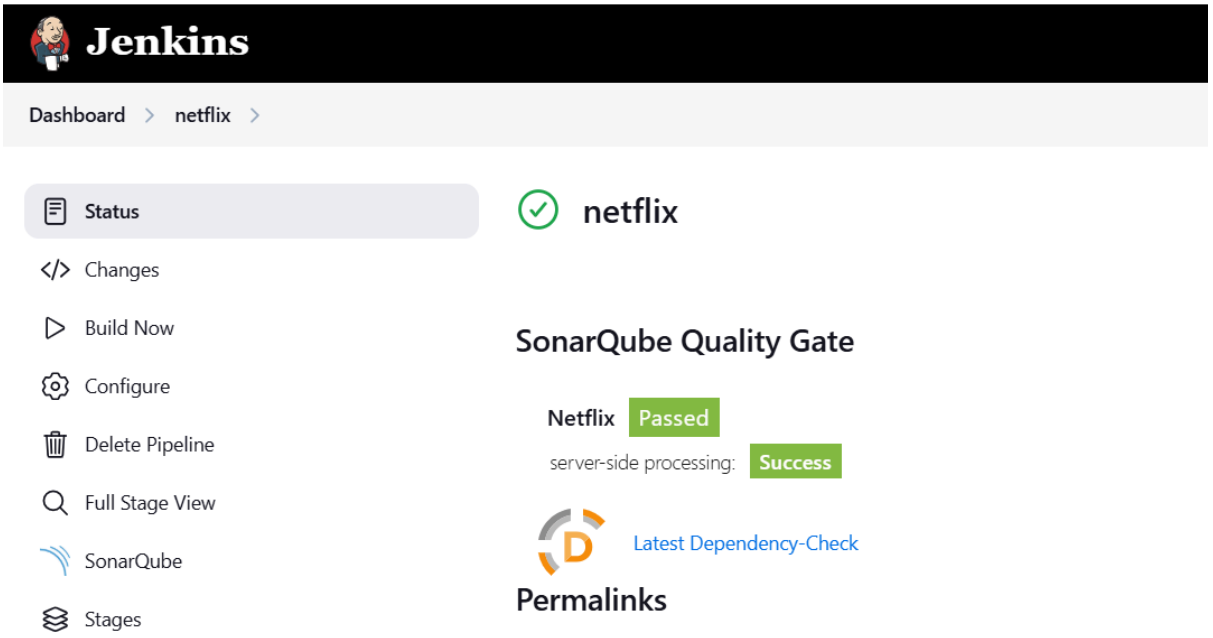
```
If you get docker login failed errorr
```

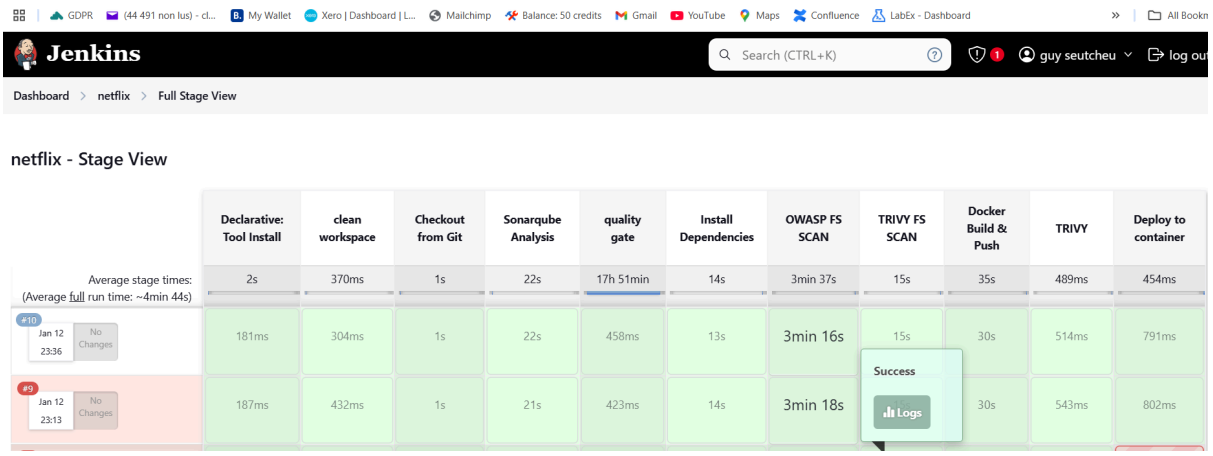
```
sudo su
```

```
sudo usermod -aG docker jenkins
```

```
sudo systemctl restart jenkins
```

Your Pipeline is started to build the following docker image and send it to dockerhub with all security checks





Phase 4 →Monitoring via Prometheus and grafana

- **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 1 →Setup another server or EC2 for monitoring

1. go to ec2 console and launch an instance having a base image of ubuntu and with t2.medium specs because **Minimum Requirements to Install Prometheus :**

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

Step 2 →Installing Prometheus:

1. **First, create a dedicated Linux user for Prometheus and download Prometheus:**

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

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

2. Extract Prometheus files, move them, and create directories:

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

b. cd prometheus-2.47.1.linux-amd64/

c. sudo mkdir -p /data /etc/prometheus

d. sudo mv prometheus promtool /usr/local/bin/

e. sudo mv consoles/ console_libraries/ /etc/prometheus/

f. sudo mv prometheus.yml /etc/prometheus/prometheus.yml

3. Set ownership for directories:

a. sudo chown -R prometheus:prometheus /etc/prometheus/ /data/

4. Create a systemd unit configuration file for Prometheus:

a. 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
```

5. Enable and start Prometheus:

a. sudo systemctl enable prometheus

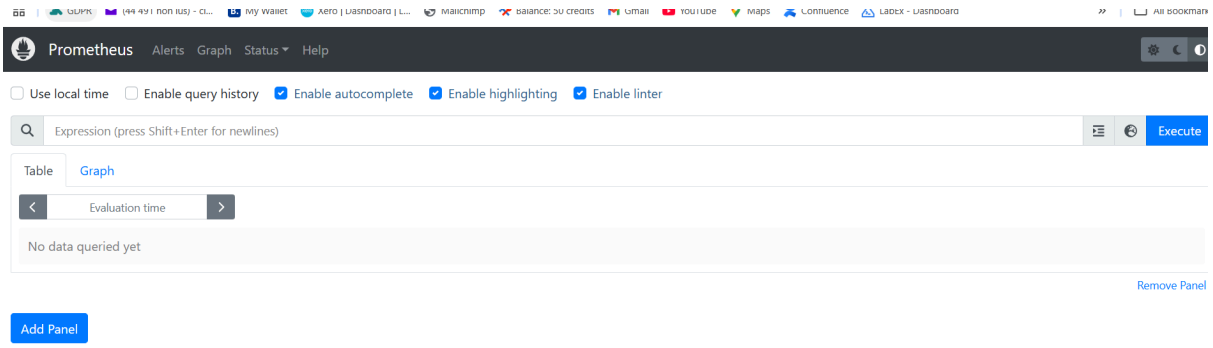
b. sudo systemctl start prometheus

c. sudo systemctl status prometheus

```
ubuntu@ip-10-0-4-18:~$ systemctl status prometheus
● prometheus.service - Prometheus
     Loaded: loaded (/etc/systemd/system/prometheus.service; enabled; vendor preset: enabled)
     Active: active (running) since Fri 2025-01-17 02:04:07 UTC; 2 days ago
   Main PID: 29094 (prometheus)
      Tasks: 8 (limit: 4676)
     Memory: 97.2M
        CPU: 2min 57.076s
     CGroup: /system.slice/prometheus.service
             └─29094 /usr/local/bin/prometheus --config.file=/etc/prometheus/prometheus.yml --storage.tsdb.path=/data --

Jan 19 11:00:00 ip-10-0-4-18 prometheus[29094]: ts=2025-01-19T11:00:00.199Z caller=compact.go:523 level=info component=
Jan 19 11:00:00 ip-10-0-4-18 prometheus[29094]: ts=2025-01-19T11:00:00.202Z caller=head.go:1298 level=info component=tsd
```

Now go to your security group of your ec2 to enable port 9090 in which prometheus will run

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



Step 3 → Installing Node Exporter:

Node exporter is like a “reporter” tool for Prometheus, which helps collect and provide information about a computer (node) so Prometheus can monitor it. It gathers data about things like CPU usage, memory, disk space, and network activity on that computer.

A Node Port Exporter is a specific kind of Node Exporter that is used to collect information about network ports on a computer. It tells Prometheus which network ports are open and what kind of data is going in and out of

those ports. This information is useful for monitoring network-related activities and can help you ensure that your applications and services are running smoothly and securely.

Run the following commands for installation

1. **Create a system user for Node Exporter and download Node Exporter:**

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

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

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

a. tar -xvf node_exporter-1.6.1.linux-amd64.tar.gz

b. sudo mv node_exporter-1.6.1.linux-amd64/node_exporter /usr/local/bin/

c. rm -rf node_exporter*

3. Create a systemd unit configuration file for Node Exporter:

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

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

provide more detailed information about what might be going wrong. For example:

```
[Unit]

Description=Node Exporter

Wants=network-online.target

After=network-online.target

StartLimitIntervalSec=500

StartLimitBurst=5

[Service]

User=node_exporter

Group=node_exporter

Type=simple

Restart=on-failure

RestartSec=5s

ExecStart=/usr/local/bin/node_exporter --collector.logind

[Install]
```

```
WantedBy=multi-user.targetb. press → ctrl+o then ctrl+x
```

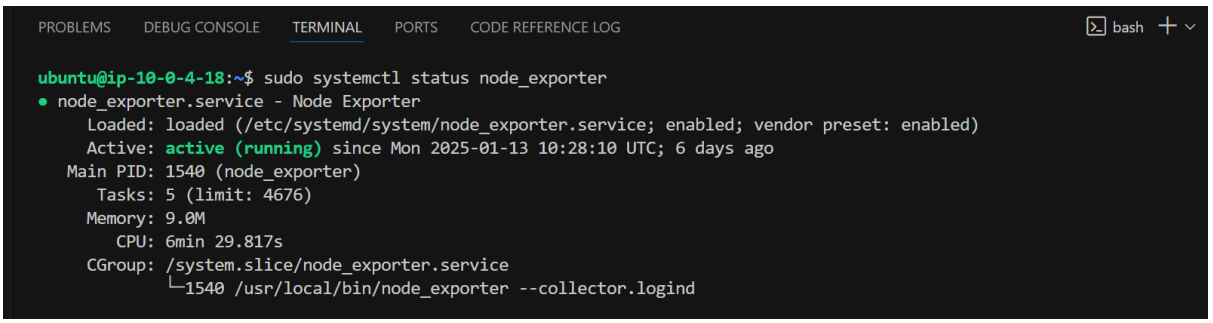
Enable and start Node Exporter:

a. sudo systemctl enable node_exporter

b. sudo systemctl start node_exporter

c. sudo systemctl status node_exporter

node exporter service is now running



You can access Node Exporter metrics in Prometheus.

Step 4→Configure Prometheus Plugin Integration:

1. go to your EC2 and run →**cd /etc/prometheus**

2. you have to edit the prometheus.yml file to monitor anything

```
global:

 scrape_interval: 15s




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>']
```

add the above code with proper indentation like this →

a. Check the validity of the configuration file →

promtool check config /etc/prometheus/prometheus.yml

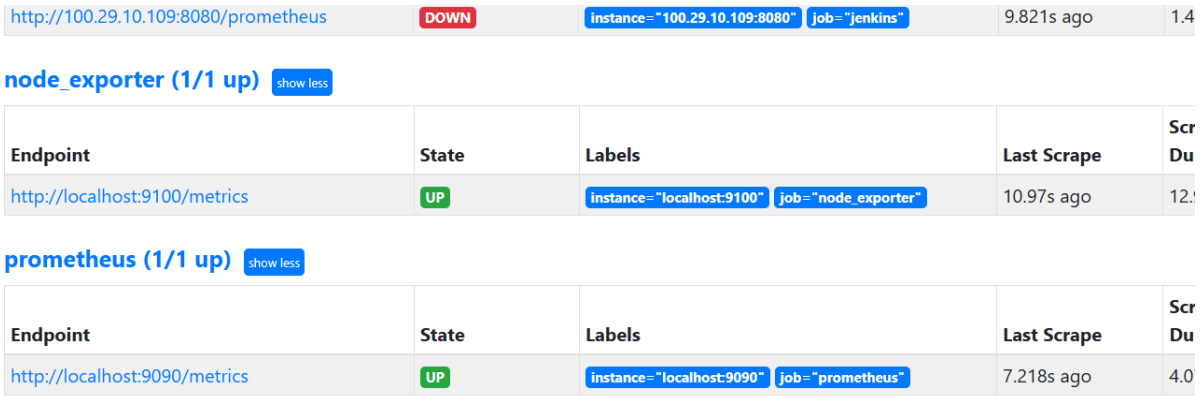
o/p →success

b. Reload the Prometheus configuration without restarting →

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

go to your prometheus tab again and click on status and select targets you will there is three targets present as we enter in yaml file for monitoring

```
a  console_libraries  consoles  prometheus.yml  –
ubuntu@ip-10-0-4-18:/etc/prometheus$ vi prometheus.yml
ubuntu@ip-10-0-4-18:/etc/prometheus$ promtool check config /etc/prometheus/prometheus.yml
Checking /etc/prometheus/prometheus.yml
 SUCCESS: /etc/prometheus/prometheus.yml is valid prometheus config file syntax
```



Step 5 →Setup Grafana →

Install Dependencies:

a. sudo apt-get update

b. sudo apt-get install -y apt-transport-https software-properties-common

Add the GPG Key for Grafana:

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

Add the repository for Grafana stable releases:

a. 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:

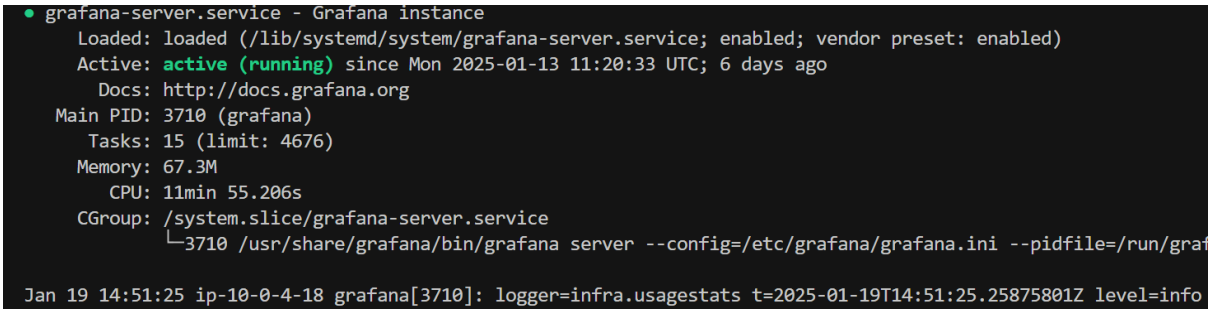
a. sudo apt-get update

b. sudo apt-get -y install grafana

c. sudo systemctl enable grafana-server

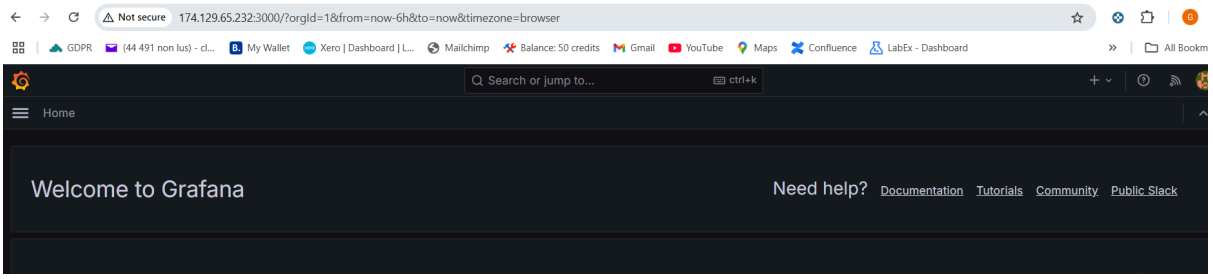
d. sudo systemctl start grafana-server

e. sudo systemctl status grafana-server



Now go to your ec2 security group and open port no. 3000 in which grafana runs

Go and browse http://public_ip:3000 to access your grafana web interface



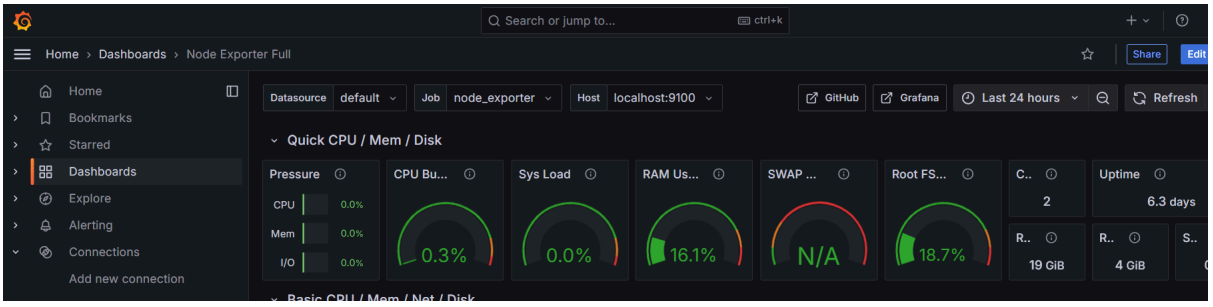
username = admin, password =admin

Step 6 →Add Prometheus Data Source:

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

Follow these steps:

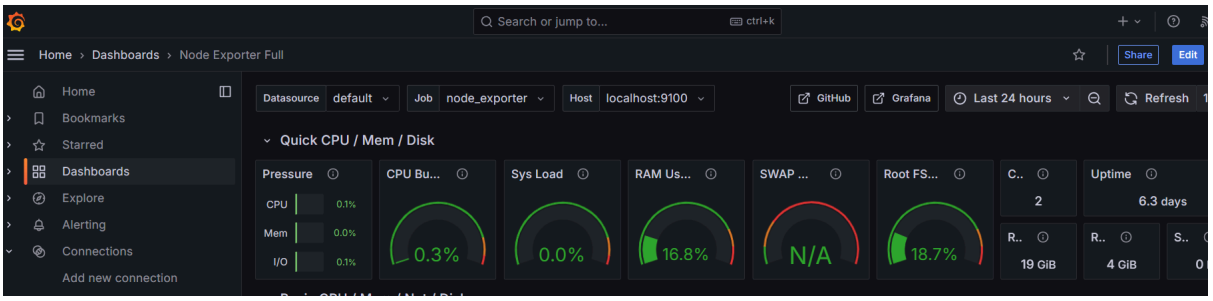
- Click on the gear icon (⚙️) in the left sidebar to open the “Configuration” menu.
- Select “Data Sources.”
- Click on the “Add data source” button.



Step 7 → Import a Dashboard

Importing a dashboard in Grafana is like using a ready-made template to quickly create a set of charts and graphs for monitoring your data, without having to build them from scratch.

- **Click on the "+" (plus) icon in the left sidebar to open the "Create" menu.**
- **Select the data source you added (Prometheus) from the dropdown.**
- **Click on the "Import" button.**

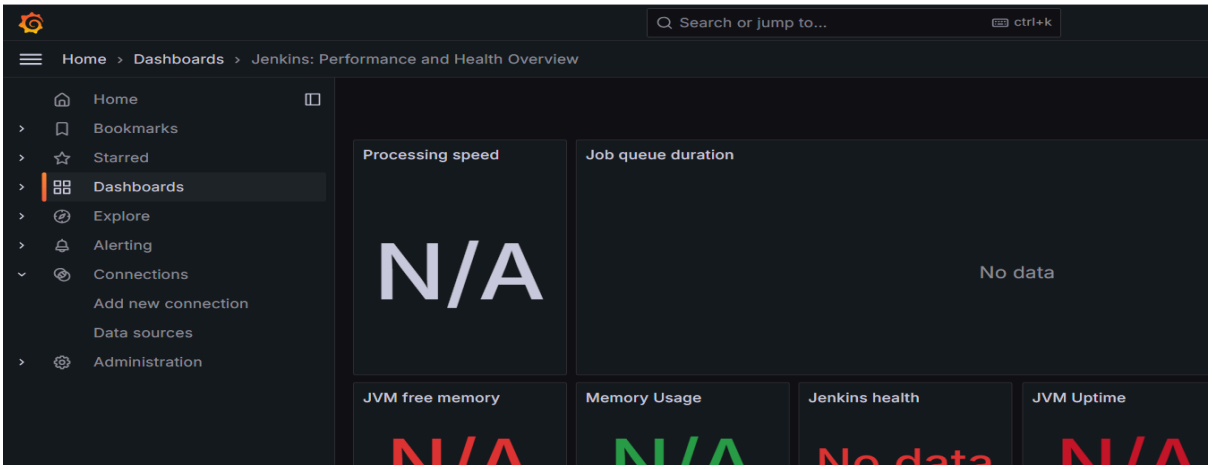


Step 7 →Configure global setting for promotheus

go to manage jenkins →system →search for promotheus — apply →save

Step 8→ import a dashboard for jenkins

- **Click on the "+" (plus) icon in the left sidebar to open the "Create" menu.**
- **Select "Dashboard."**
- **Click on the "Import" dashboard option.**
- **Enter the dashboard code you want to import (e.g., code 9964).**
- **Click the "Load" button.**
- **Select the data source you added (Prometheus) from the dropdown.**



Phase 5: Kubernetes →

Step 1 →Setup EKS(Elastic Kubernetes Service) on aws

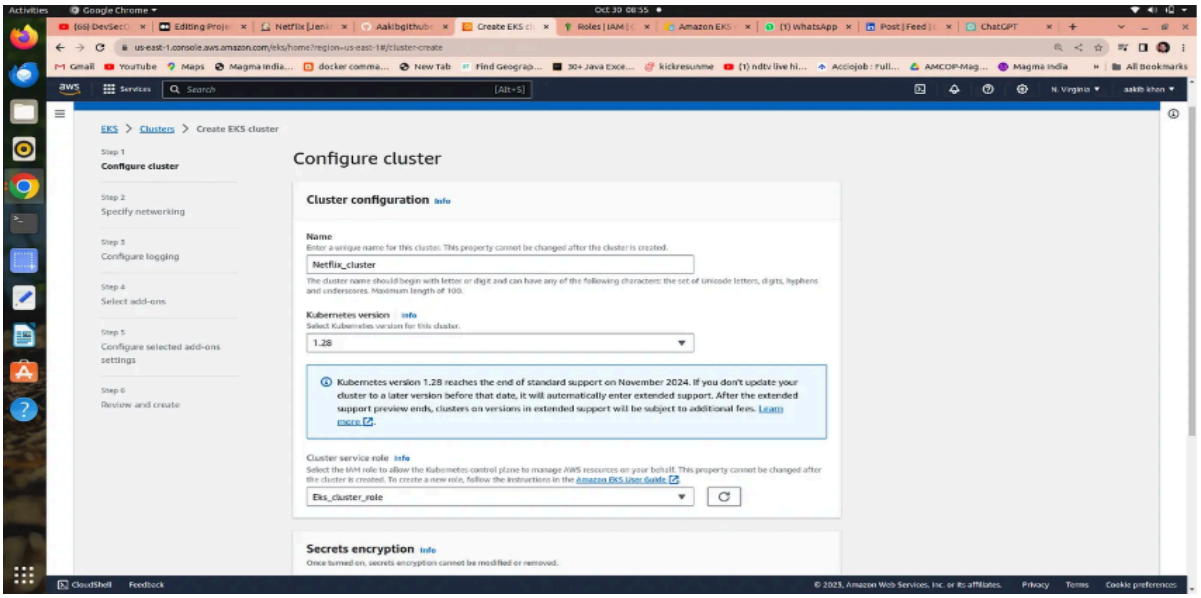
1. go to console and search for EKS
2. click on add a cluster

click on create

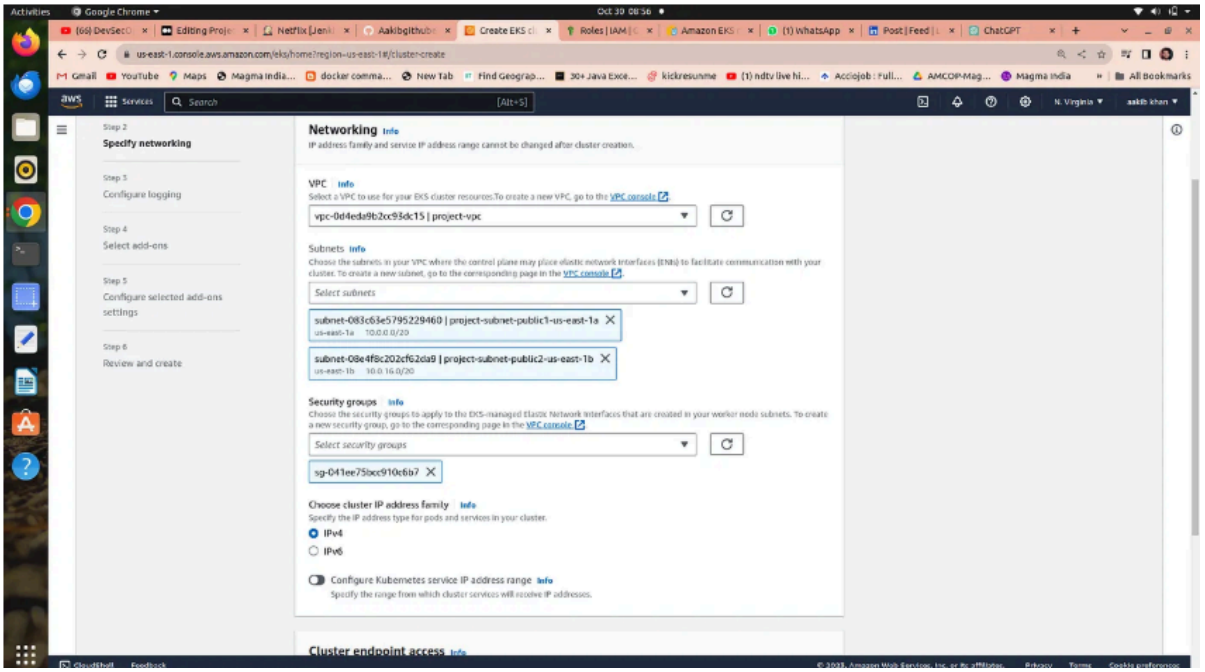
3. add a name to your cluster and choose a service role if you don't have then follow below documentation to create it

Amazon EKS cluster IAM role

The Amazon EKS cluster IAM role is required for each cluster. Kubernetes clusters managed by Amazon EKS use this role…

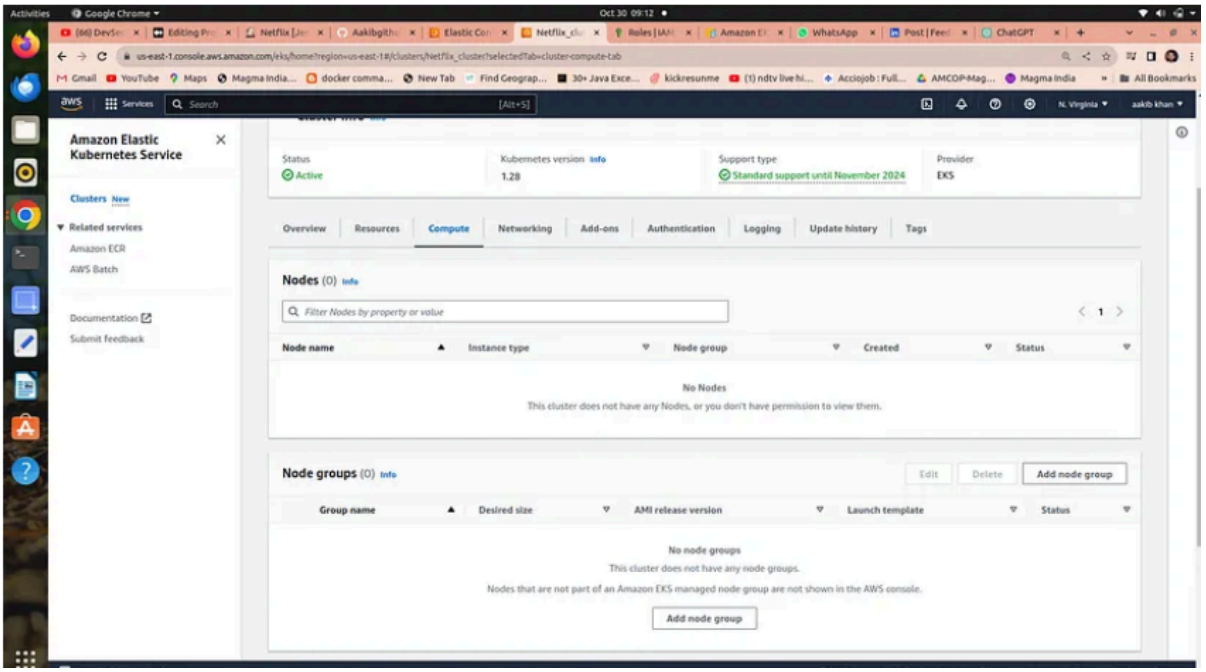


4. click on next and choose the default vpc and in subnet option make sure to remove all the private subnet and remain everything as it is

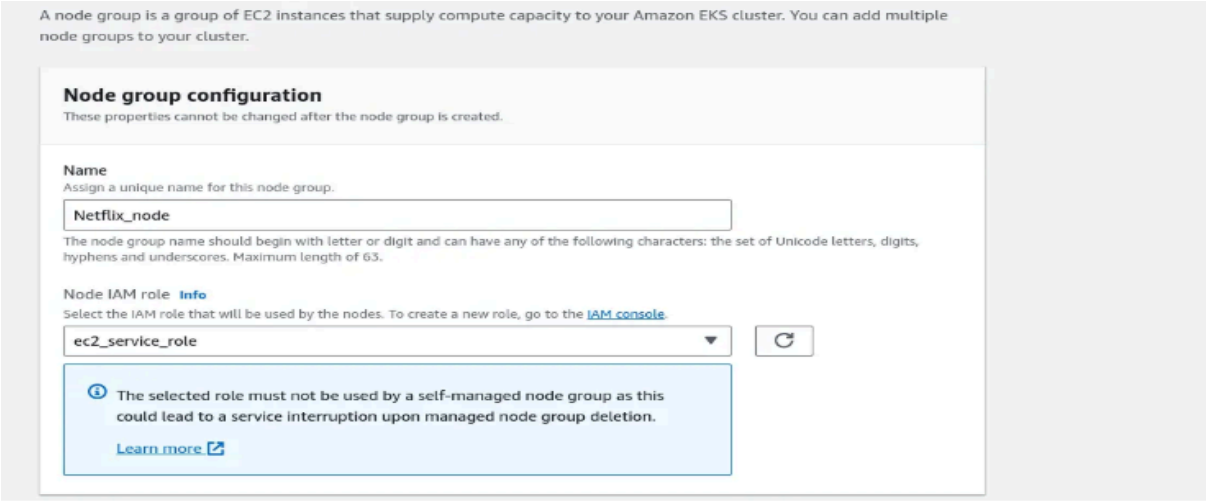


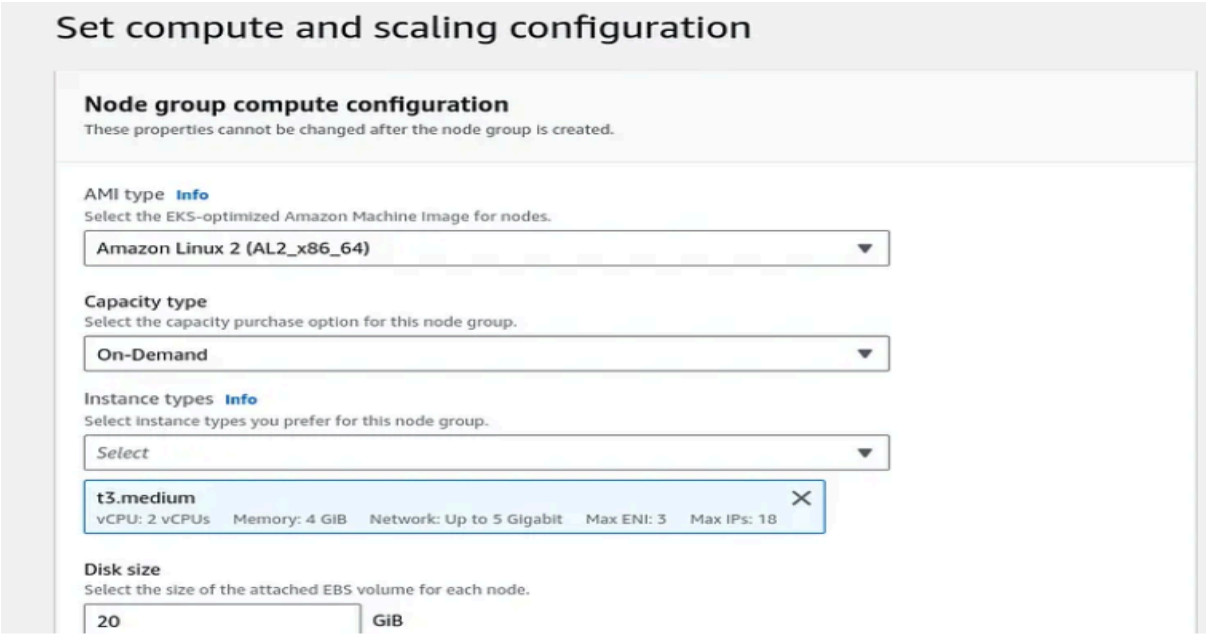
5. click on next →next →create

6. when you cluster is ready then go to compute option and add a node group

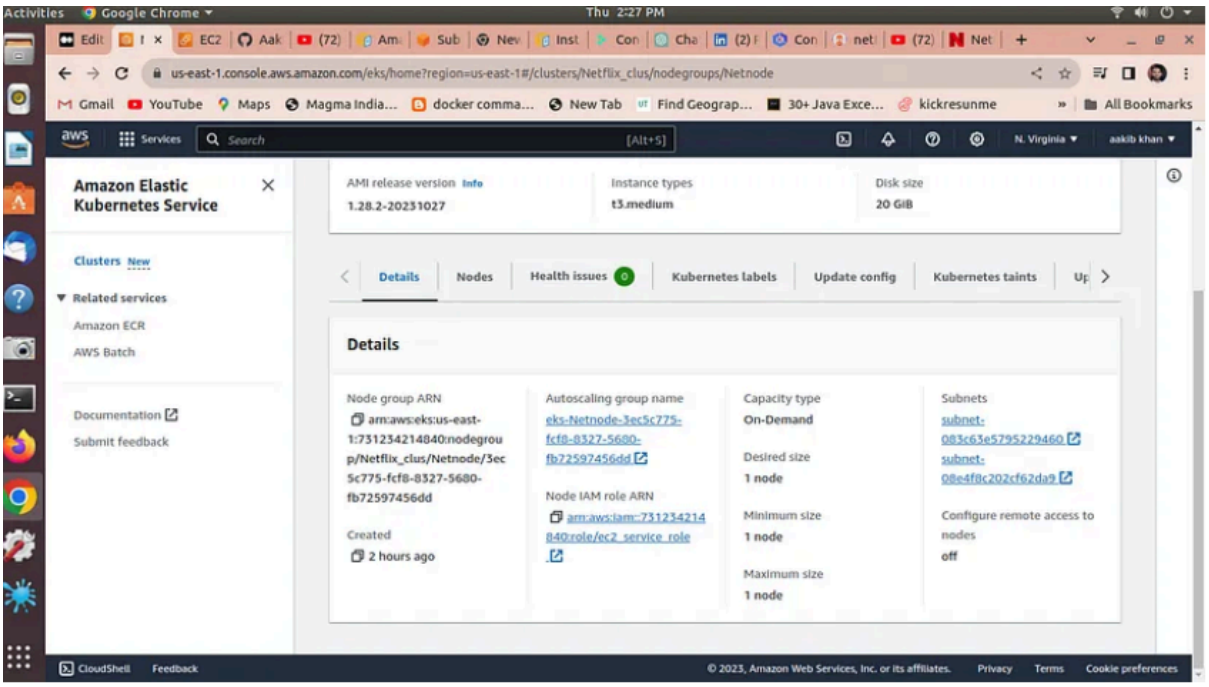


7. choose the below options





Remain everything as it is →click on create

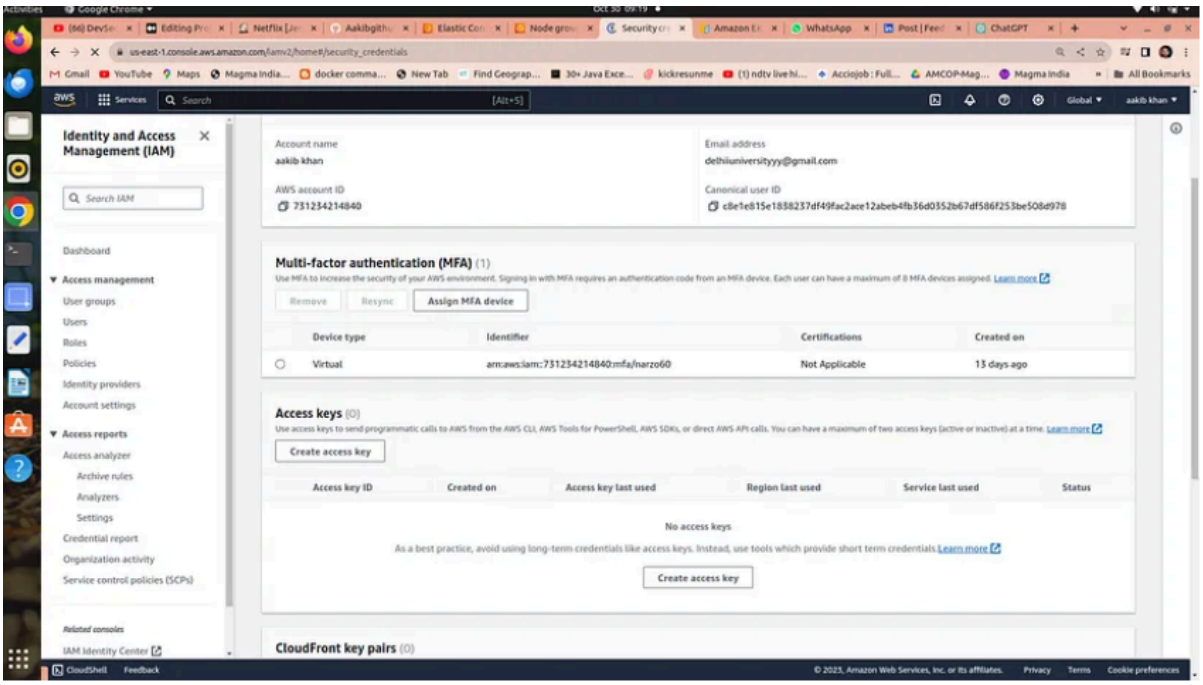


Step 2 →Installing aws -cli to control the cluster from our terminal

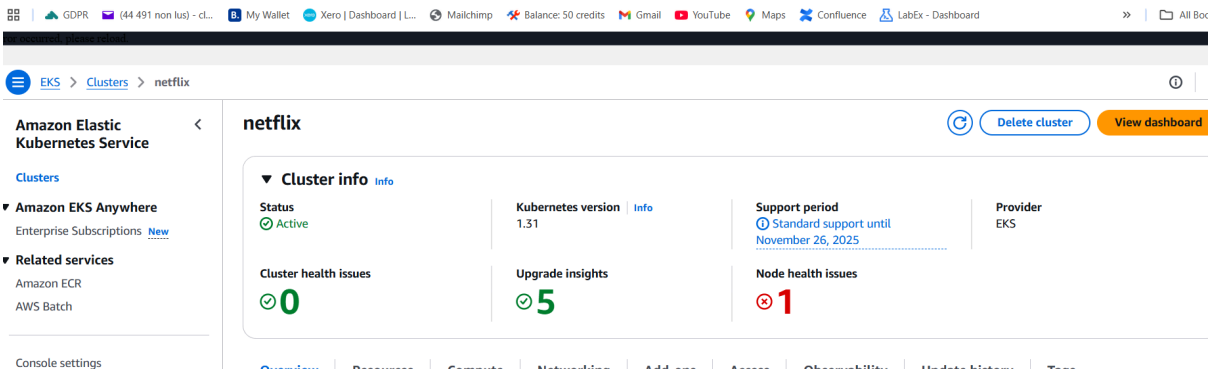
Now go to your terminal and run

1. **pip install awscli**
2. **aws configure**
3. **enter a access key and then secret access key**

2. scroll down to access keys and click on create



copy and paste it to terminal



Run the following command

1. **aws eks update-kubeconfig — name YourClusterName — region us-east-1**

This command is used to update the Kubernetes configuration (kubeconfig) file for an Amazon Elastic Kubernetes Service (EKS) cluster named "Netflix_cluster" in the "us-east-" region, allowing you to interact with the cluster using `kubectl`.

Step 3→Install helm →

1. curl https://baltocdn.com/helm/signing.asc | gpg — dearmor | sudo tee /usr/share/keyrings/helm.gpg > /dev/null
2. sudo apt-get install apt-transport-https — yes
3. echo “deb [arch=$(dpkg — print-architecture) signed-by=/usr/share/keyrings/helm.gpg] https://baltocdn.com/helm/stable/debian/ all main” | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list
4. sudo apt-get install helm

Step →4 Install Node Exporter using Helm

To begin monitoring your Kubernetes cluster, you’ll install the Prometheus Node Exporter by Using Helm to install Node Exporter in Kubernetes makes it easy to set up and manage the tool for monitoring your servers by providing a pre-packaged, customizable, and consistent way to do it.

1. Add the Prometheus Community Helm repository:

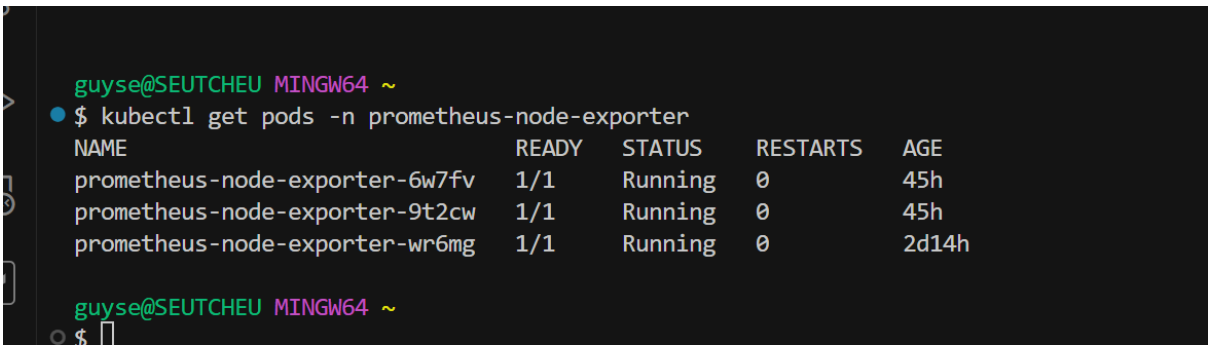
- helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

1. Create a Kubernetes namespace for the Node Exporter:

- kubectl create namespace prometheus-node-exporter

1. Install the Node Exporter using Helm:

- helm install prometheus-node-exporter prometheus-community/prometheus-node-exporter — namespace prometheus-node-exporter



Add a Job to Scrape Metrics on nodeip:9001/metrics in prometheus.yml:

Update your Prometheus configuration (prometheus.yml) to add a new job for scraping metrics from nodeip:9001/metrics. You can do this by adding the following configuration to your prometheus.yml file:

```
- job_name: 'Netflix'

  metrics_path: '/metrics'

  static_configs:

    - targets: ['node1Ip:9100']
```

Replace 'your-job-name' with a descriptive name for your job. The static_configs section specifies the targets to scrape metrics from, and in this case, it's set to nodeip:9001.

Don't forget to reload or restart Prometheus to apply these changes to your configuration Deploy Application with ArgoCDon.

Step 5→Install Argo cd

Argo CD is a tool that helps software developers and teams manage and deploy their applications to Kubernetes clusters more easily. It simplifies the process of keeping your applications up to date and in sync with your desired configuration by automatically syncing your code with what's running in your Kubernetes environment. It's like a traffic cop for your chmod 700 get_helm.shapplications on Kubernetes, ensuring they are always in the right state without you having to manually make changes.

1. **Add the Argo CD Helm repository:**

- `helm repo add argo-cd https://argoproj.github.io/argo-helm`

2. Update your Helm repositories:

- `helm repo update`

3. Create a namespace for Argo CD (optional but recommended):

- `kubectl create namespace argocd`

4. Install Argo CD using Helm:

- `helm install argocd argo-cd/argo-cd -n argocd`

5. kubectl get all -n argocd

```
NAME                                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/argocd-applicationset-controller    1/1     1            1           2d14h
deployment.apps/argocd-dex-server                   1/1     1            1           2d14h
deployment.apps/argocd-notifications-controller     1/1     1            1           2d14h
deployment.apps/argocd-redis                        1/1     1            1           2d14h
deployment.apps/argocd-repo-server                  1/1     1            1           2d14h
deployment.apps/argocd-server                       1/1     1            1           2d14h

NAME                                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/argocd-applicationset-controller-68dd598849    1         1         1       2d14h
replicaset.apps/argocd-dex-server-86877d5f88                   1         1         1       2d14h
replicaset.apps/argocd-notifications-controller-74584f69d8     1         1         1       2d14h
replicaset.apps/argocd-redis-95497c66                          1         1         1       2d14h
replicaset.apps/argocd-repo-server-689dd9d6d4                  1         1         1       2d14h
replicaset.apps/argocd-server-6997845b6                        1         1         1       2d14h

NAME                                             READY   AGE
statefulset.apps/argocd-application-controller   1/1     2d14h
```

Expose argocd-server

By default argocd-server is not publicaly exposed. For the purpose of this workshop, we will use a Load Balancer to make it usable:

1. **kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}**

export ARGOCD_SERVER=$(kubectl get svc argocd-server -n argocd -o json | jq --raw-output '.status.loadBalancer.ingress[0].hostname')

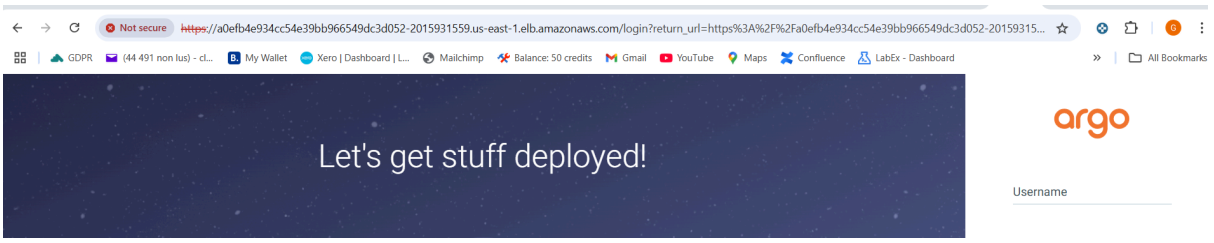
echo $ARGOCD_SERVER

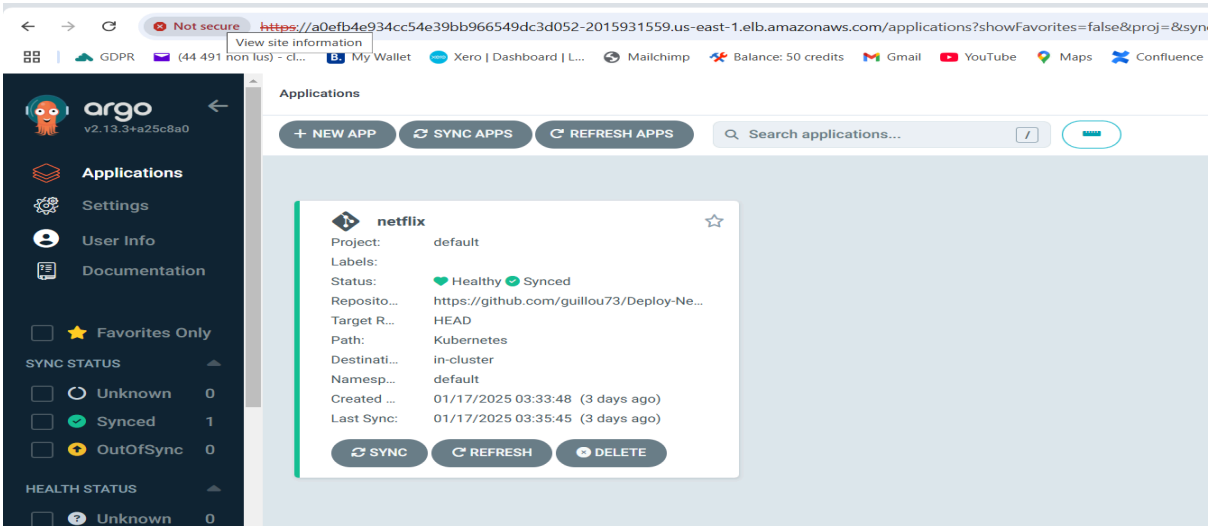
export ARGO_PWD=$(kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d)

echo $ARGO_PWD

```
guyse@SEUTCHEU MINGW64 ~
$ export ARGOCD_SERVER=$(kubectl get svc argocd-server -n argocd -o json | jq --raw-output '.status.loadBalancer.ingress[0].host
ame')

guyse@SEUTCHEU MINGW64 ~
$ echo $ARGOCD_SERVER
a0efb4e934cc54e39bb966549dc3d052-2015931559.us-east-1.elb.amazonaws.com

guyse@SEUTCHEU MINGW64 ~
$ export ARGO_PWD=$(kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d)

guyse@SEUTCHEU MINGW64 ~
$ echo $ARGO_PWD
kJ1k1USrJZU0cBKS
```

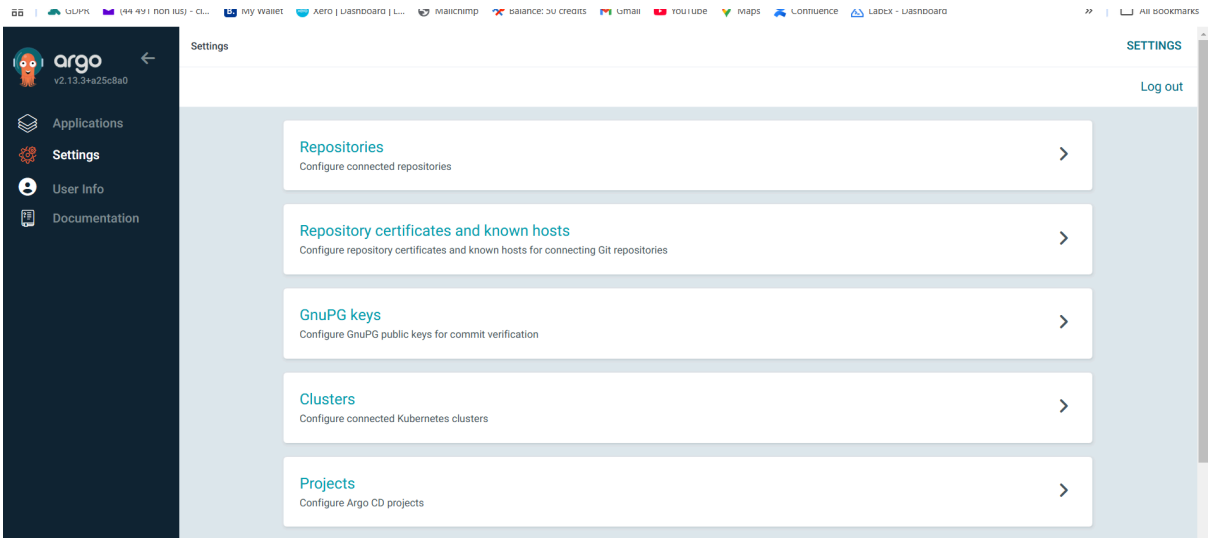
you have given a url copy and paste it on your chrome browser



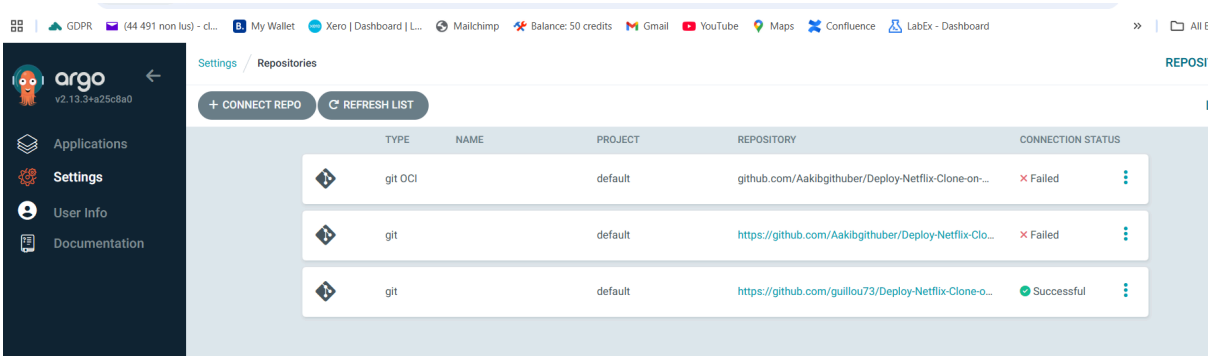


Step 5 →Deploy Application with ArgoCD

Sign into argo cd using above steps →

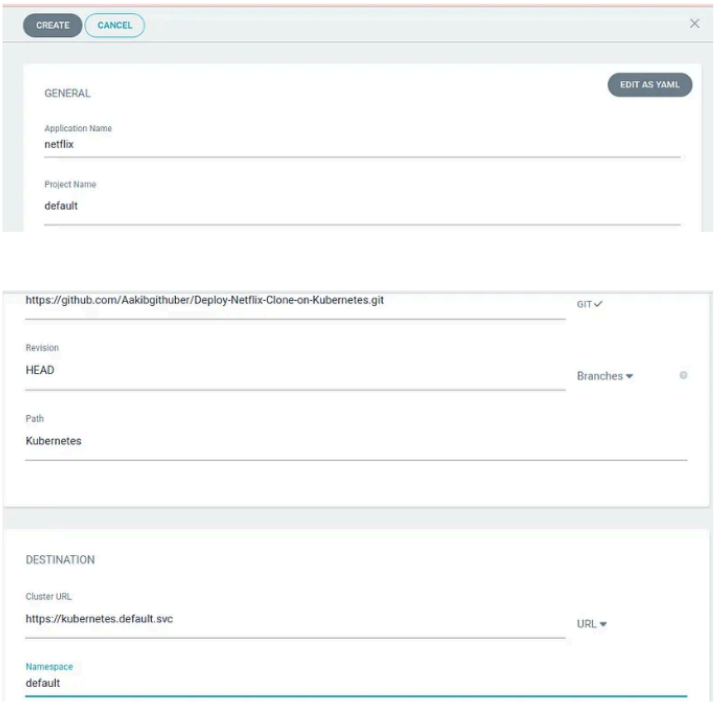


1. go to setting →repositories →connect repo



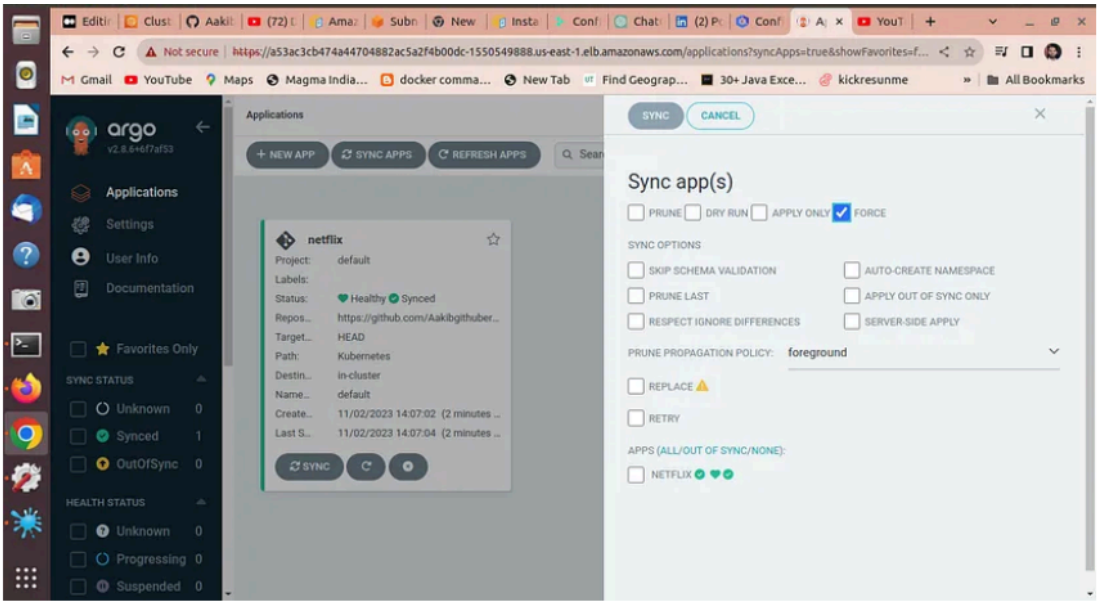
2. click on connect output →successful

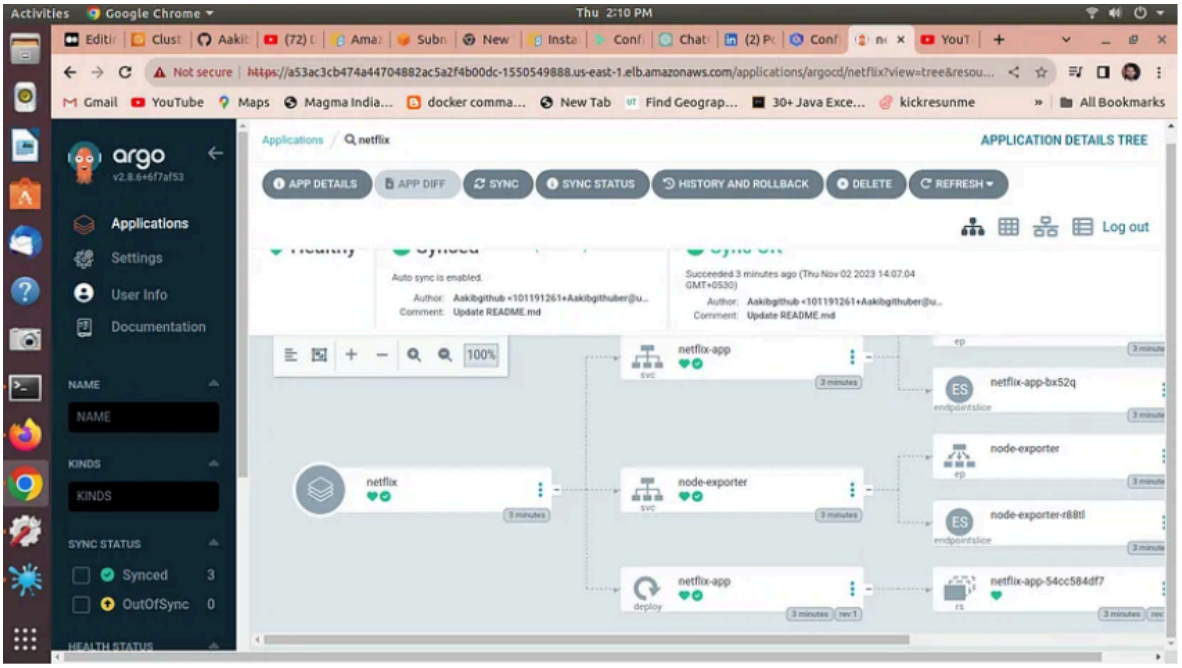
3. Now go to application →newapp



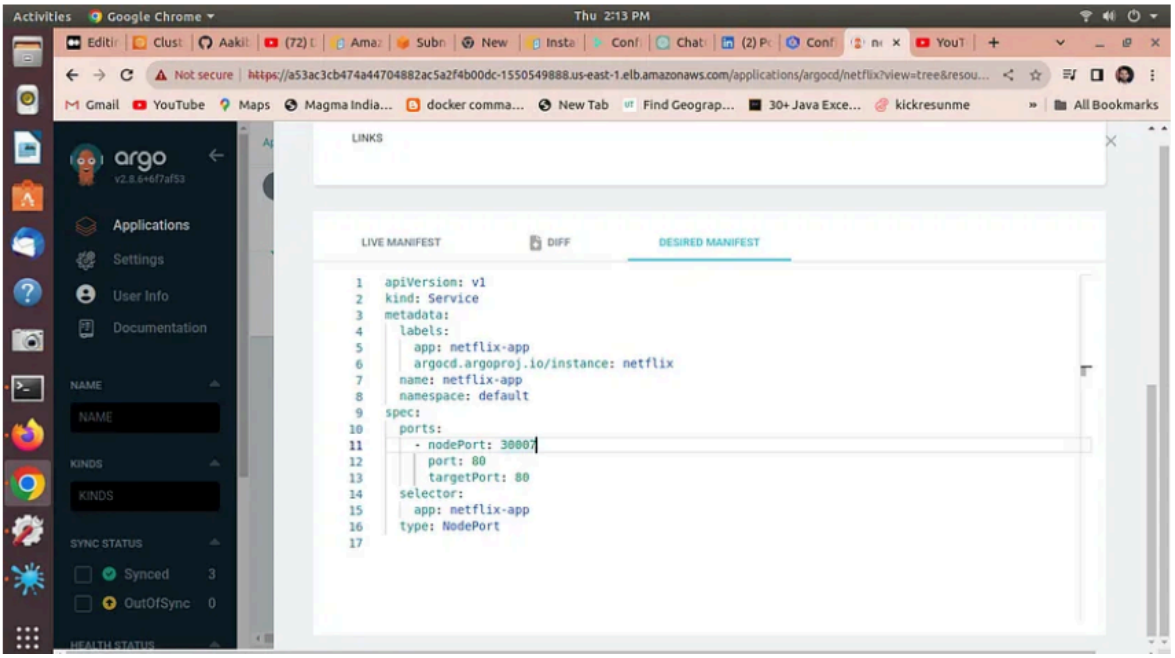
4. select the above options and click on create

5. Now click on sync →force →sync →ok

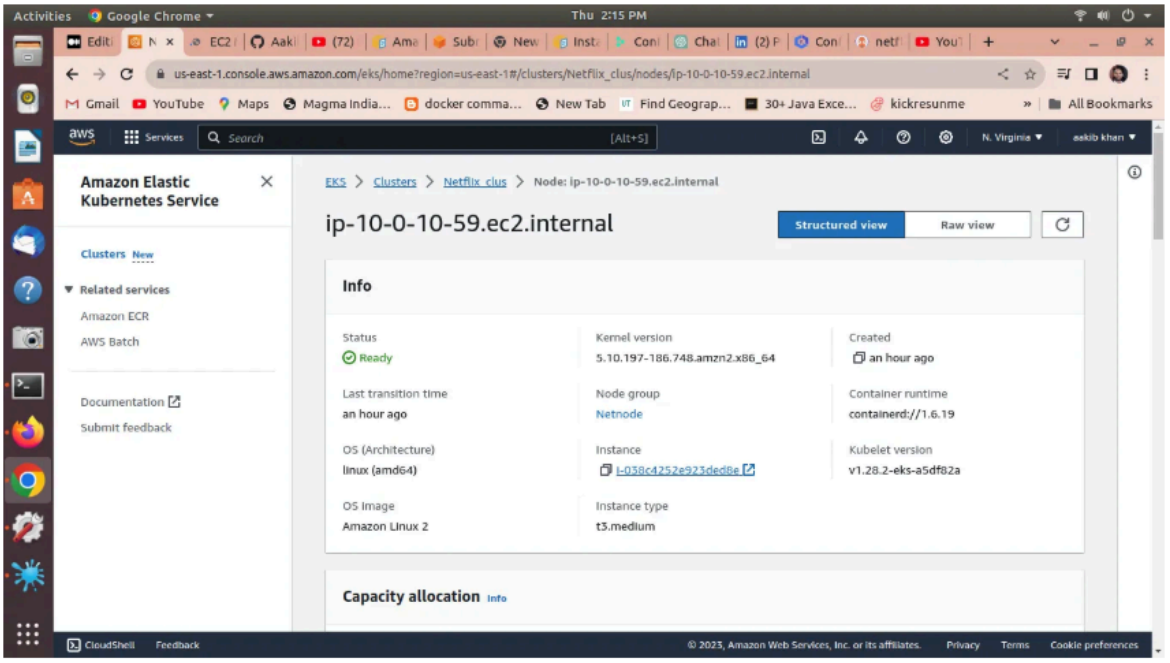




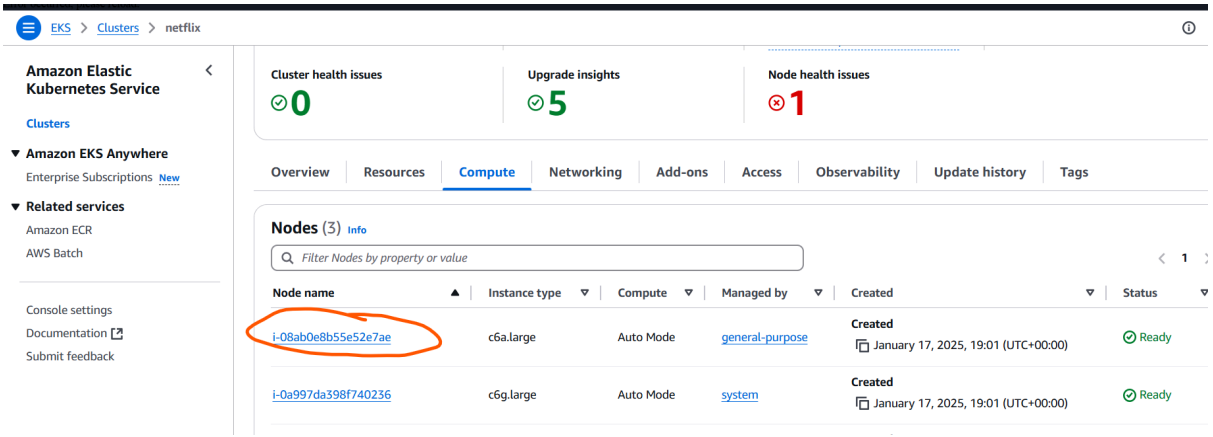
click on netflix app

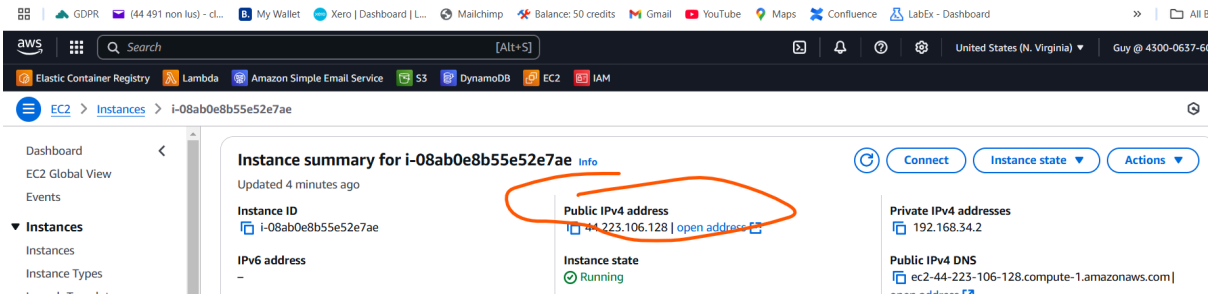


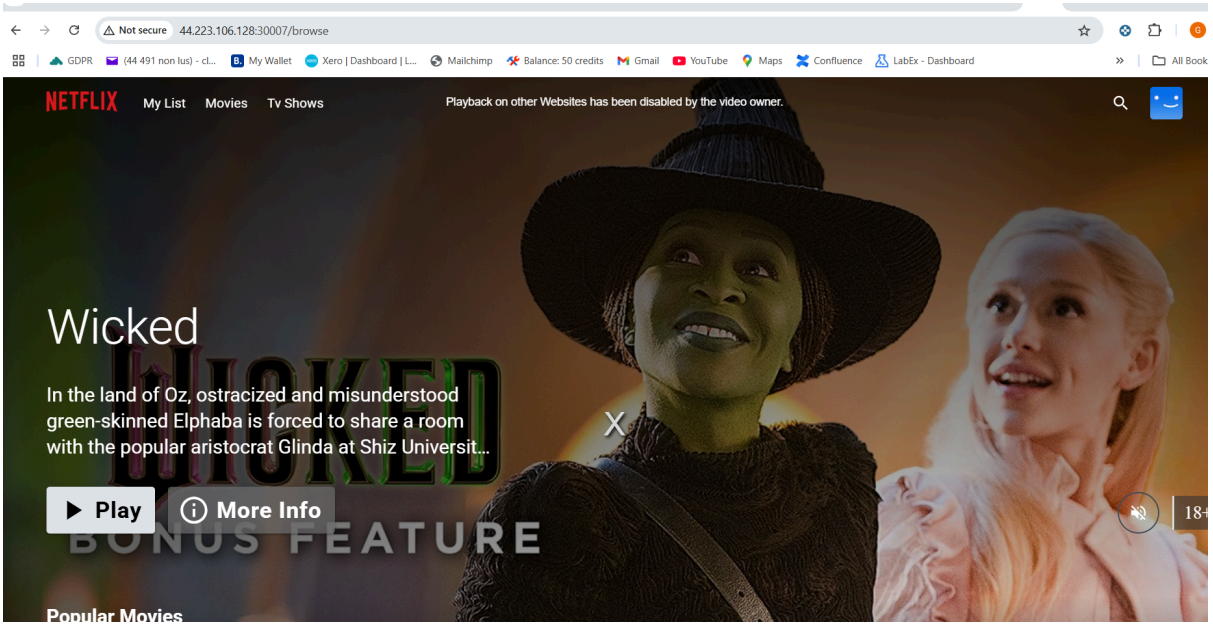
6.go to your cluster and select the node created by node group and expose port 30007



now browse the aplication by Publicip:30007






Not secure
44.223.106.128:30007/browse
GDPR
(44 491 non lus) - cl...
My Wallet
Xero | Dashboard | L...
Mailchimp
Balance: 50 credits
Gmail
YouTube
Maps
Confluence
LabEx - Dashboard
All Book
NETFLIX
My List
Movies
Tv Shows
Playback on other Websites has been disabled by the video owner.
Wicked
In the land of Oz, ostracized and misunderstood green-skinned Elphaba is forced to share a room with the popular aristocrat Glinda at Shiz Universit...
Play
More Info
BONUS FEATURE
Popular Movies