# 🚀 Project Title:

## Deploy Netflix Clone on Cloud using Jenkins and kubernetes {DevSecOps Project}

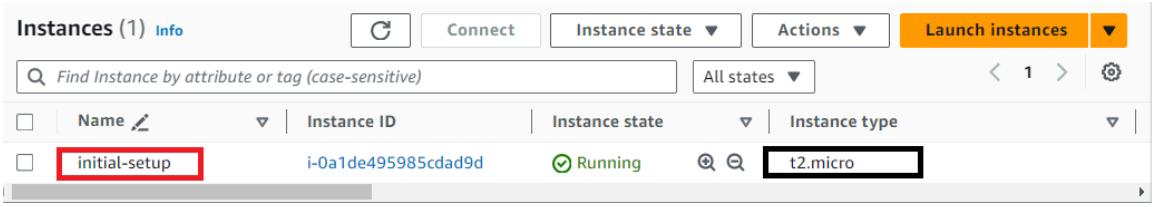
## 🔍 Project Overview:

This project demonstrates a comprehensive DevSecOps pipeline for deploying a Netflix clone application on AWS, leveraging Jenkins, Docker, SonarQube, Trivy, Prometheus, Grafana, and Kubernetes.

## Key Phases and Steps:

### Phase 1: Initial Setup and Deployment

Step-1) Launch EC2 Instance - Provision an Ubuntu 22.04 instance on AWS.




Step-2) Clone Code Repository - Clone the Netflix clone code from GitHub.

```
https://github.com/mayur4279/DevSecOps-Project-Netflixapp.git
```

Step-3) Install Docker - Set up Docker and build the application container.

```
sudo apt-get update

sudo apt-get install –y docker.io

sudo usermod -aG docker $USER

newgrp docker

sudo chmod 777 /var/run/docker.sock
```

Step-4) Build and run your application using Docker containers

```
docker build -t netflix .

docker run -d --name netflix -p 8081:80 netflix:latest

#After successfully running the application stop the application because we need to run our application using api-key in next step.
```

Step-5) API Key Integration - Retrieve and integrate TMDB API key into the application.

- Open a web browser and navigate to TMDB (The Movie Database) website.
- Click on "Login" and create an account.
- Once logged in, go to your profile and select "Settings."
- Click on "API" from the left-side panel.
- Create a new API key by clicking "Create" and accepting the terms and conditions.
- Provide the required basic details and click "Submit."
- You will receive your TMDB API key.

Step-6) Now recreate the Docker image with your api key:

```
docker build  --build-arg  TMDB_V3_API_KEY= 42ea3b97b***  -t  netflix  .
```

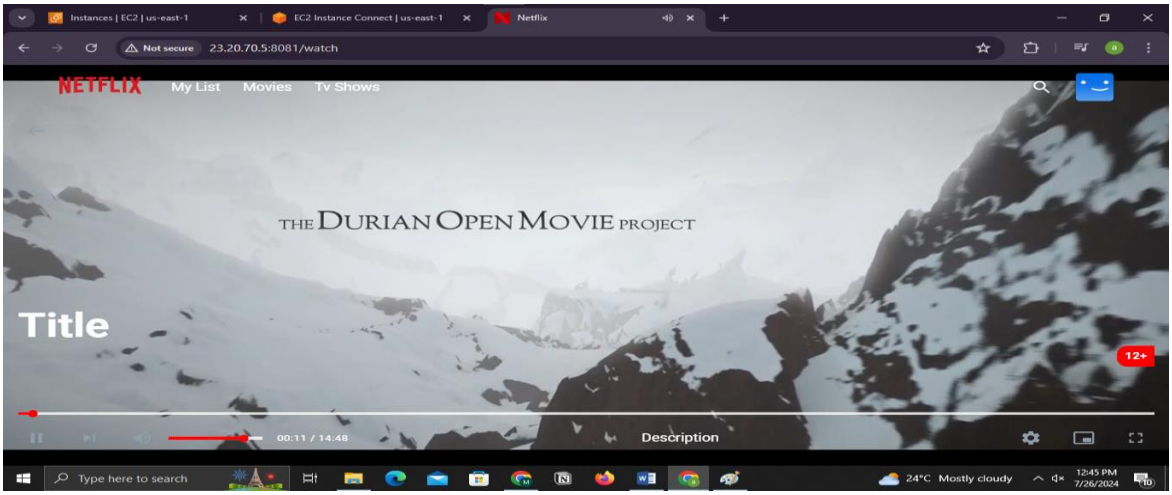
```
ubuntu@ip-172-31-35-121:~/DevSecOps-Project-Netflixapp$ docker images
REPOSITORY   TAG              IMAGE ID       CREATED          SIZE
netflix      latest           230463905c34   22 seconds ago   52.5MB
<none>       <none>           6aee0cc8ec9c   23 seconds ago   844MB
<none>       <none>           95ef740b4b68   7 minutes ago    52.5MB
<none>       <none>           62825200e0f2   8 minutes ago    844MB
nginx        stable-alpine    b32ed582bddb   5 weeks ago      43.2MB
node         16.17.0-alpine   5dcd1f6157bd   23 months ago    115MB
ubuntu@ip-172-31-35-121:~/DevSecOps-Project-Netflixapp$
```

Step-7) Run the docker image Using following command.

```
docker run -d --name netflix -p 8081:80 netflix:latest
```

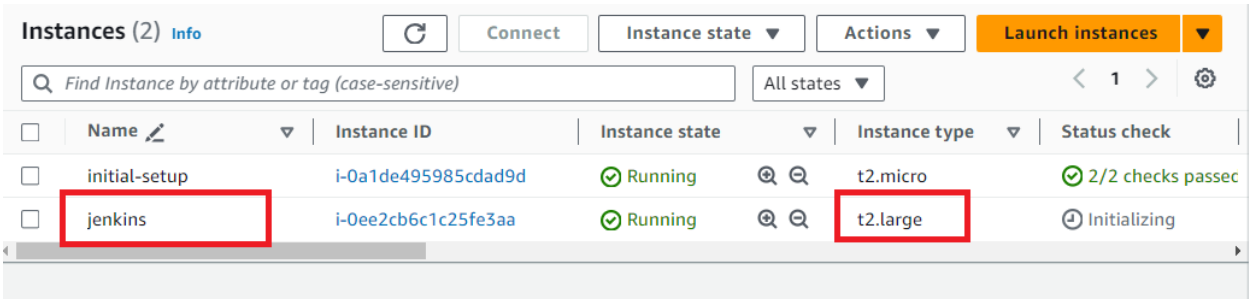
*Make sure to add port 8081 in security group*


Successfully able to run our Netflix clone application using Docker container.



## Phase 2: CI/CD Setup using Jenkins along with security tools like (trivy & SonarQube)

Step-1) Launch EC2 Instance - Provision an Ubuntu 22.04 instance on AWS.
*Minimum requirements:- t2.large instance*

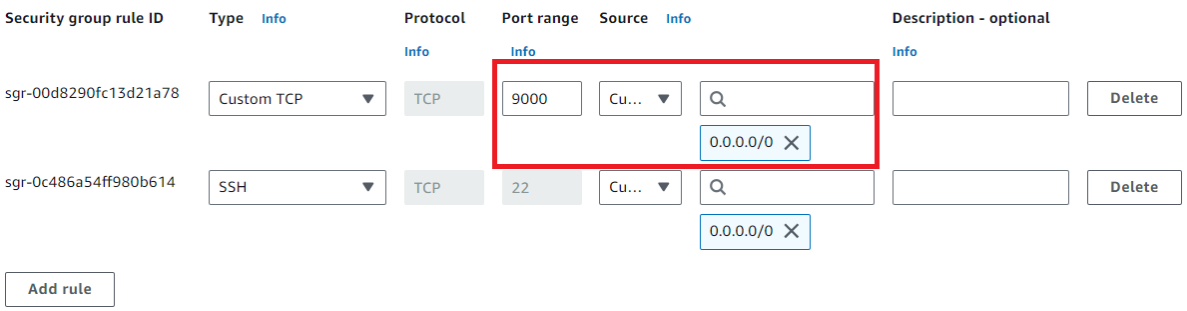



Step-2) Install SonarQube and Trivy for security:

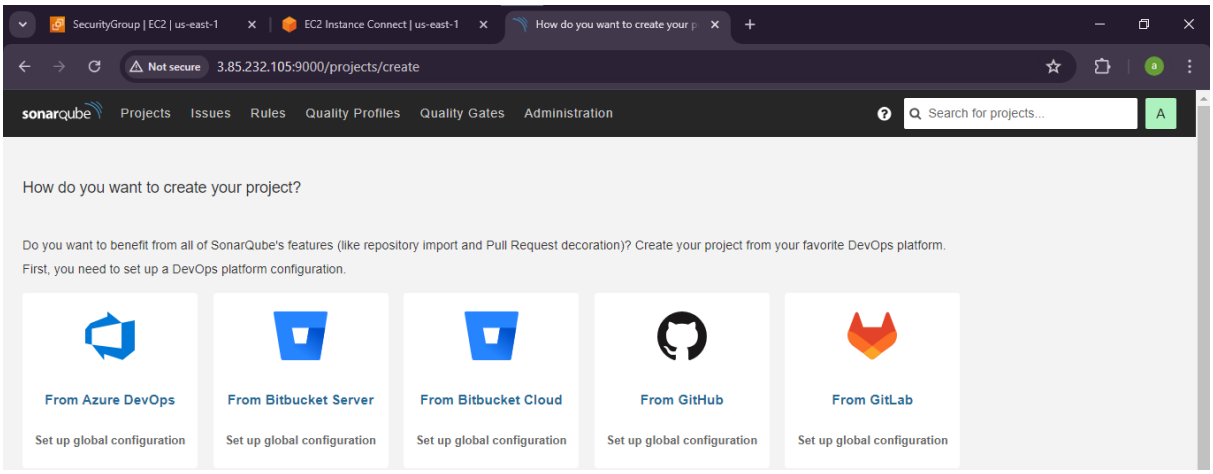
SonarQube:

```
sudo apt-get update –y

sudo apt install docker.io –y

sudo systemctl  start docker

sudo docker run -d --name sonar -p 9000:9000 sonarqube:lts-community
```

*For Access add port  9000 in security group..*



Successfully Able to access SonarQube… {Default username and password is admin, admin }




Trivy:

```
sudo apt-get install wget apt-transport-https gnupg lsb-release

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

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

sudo apt-get update

sudo apt-get install –y trivy
```

To scan docker image using trivy use following command.

```
trivy  image  <docker_images_id_or_name>
```

Step-3) Install Jenkins using following commands.

```
sudo apt update

sudo apt install fontconfig openjdk-17-jre

java -version

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)


#jenkins

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

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

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

sudo apt-get update

sudo apt-get install jenkins

sudo systemctl start jenkins

sudo systemctl enable jenkins
```

Step-4) Access Jenkins in a web browser using the public IP of your EC2 instance.

**<PublicIp>:8080**

Copy Jenkins password from the following location

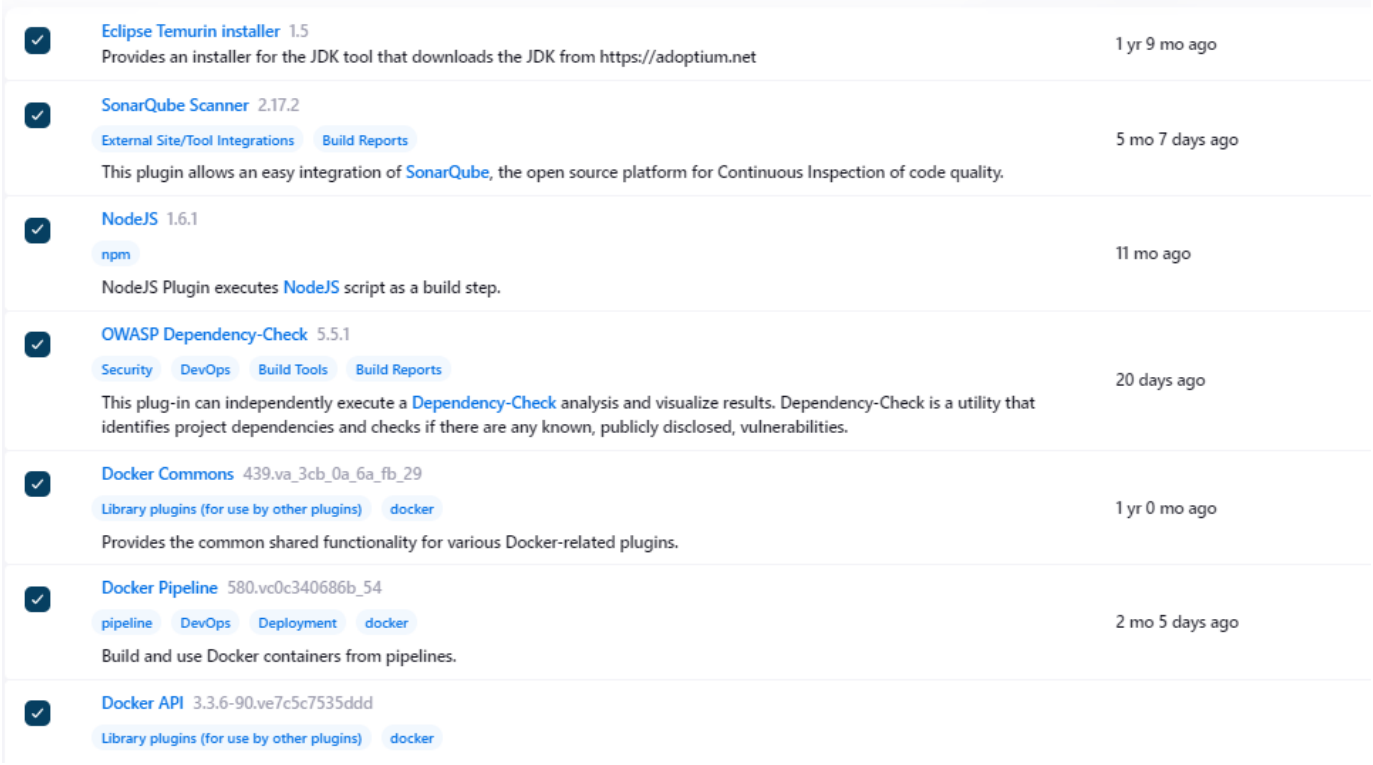
`cat /var/lib/jenkins/secrets/initialAdminPassword`

Step-5) Install Necessary Plugins in Jenkins:

Goto Manage Jenkins →Plugins → Available Plugins →

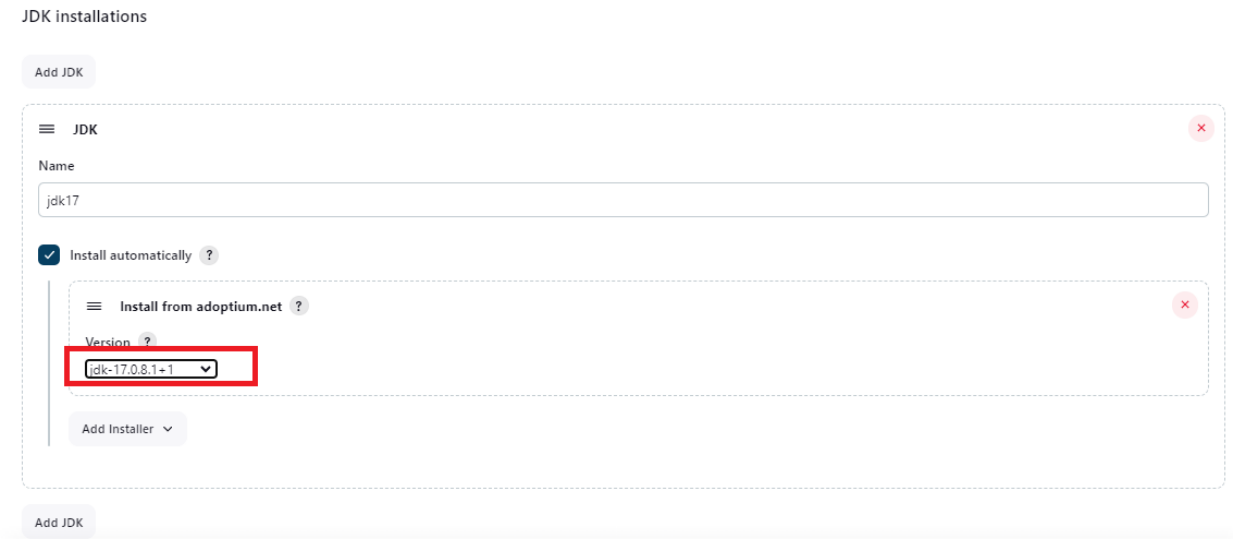
Install below plugins:

- Eclipse Temurin Installer (Install without restart)
- SonarQube Scanner (Install without restart)
- NodeJs Plugin (Install Without restart)
- OWASP Dependency-Check.
- Docker
- Docker Commons
- Docker Pipeline
- Docker API
- docker-build-step

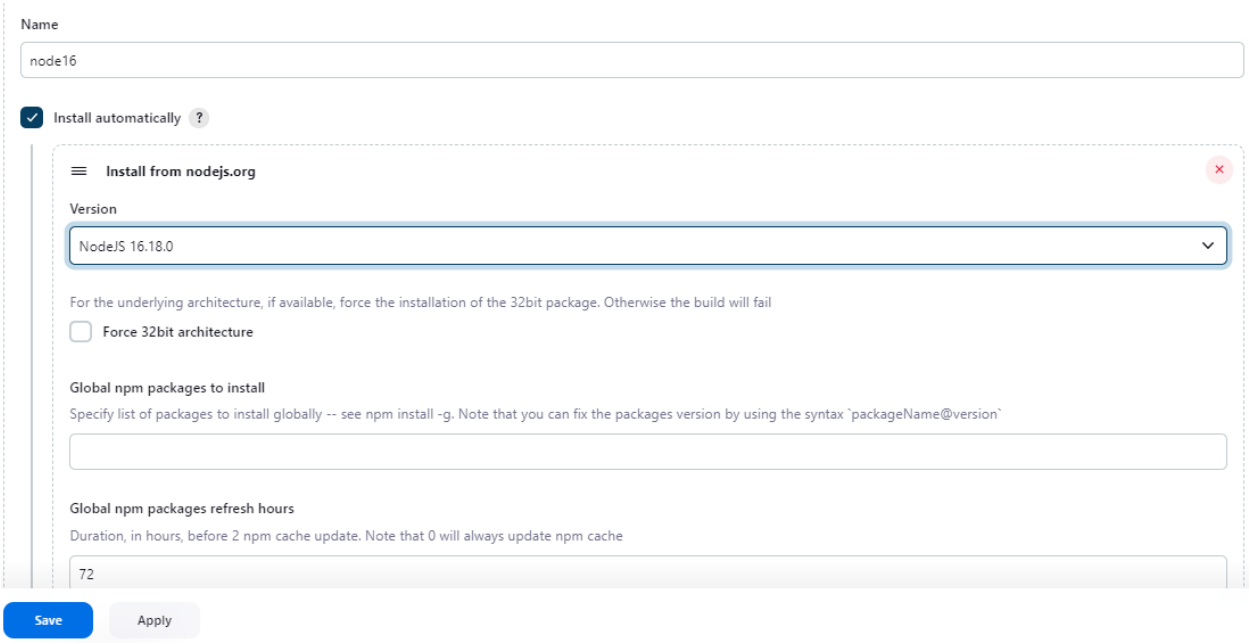


## Step-6) Configure Java and Nodejs in Global Tool Configuration

### Go to Manage Jenkins → Tools → Install JDK(17)

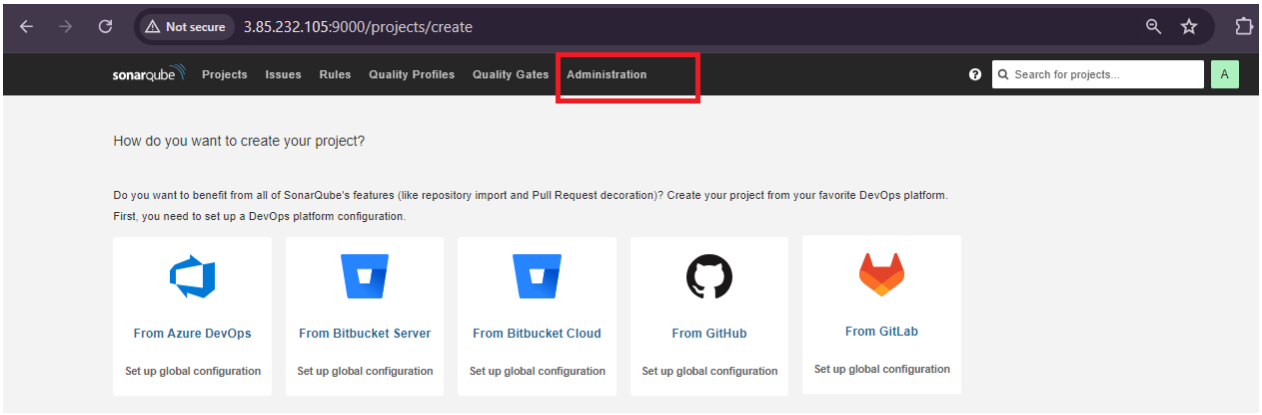



### Go to Manage Jenkins → Tools → Install node(16) → Click on Apply and Save

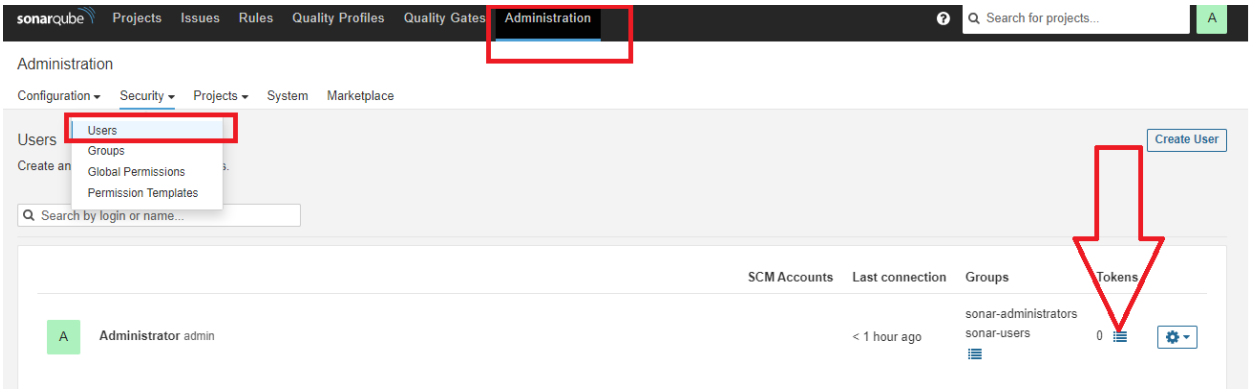


Step-7) Integrate SonarQube  with jenkins

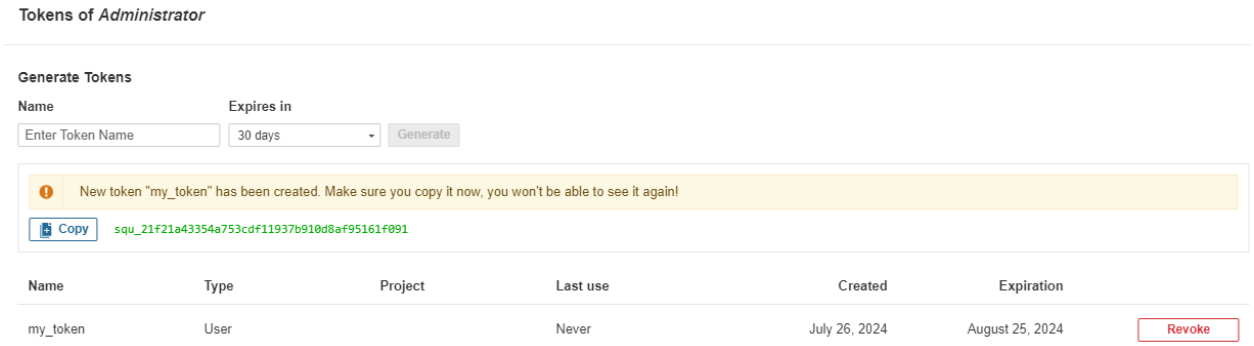
- Create Token:
  Select the Administration option in SonarQube




Click on users → Select token option  → && create your token

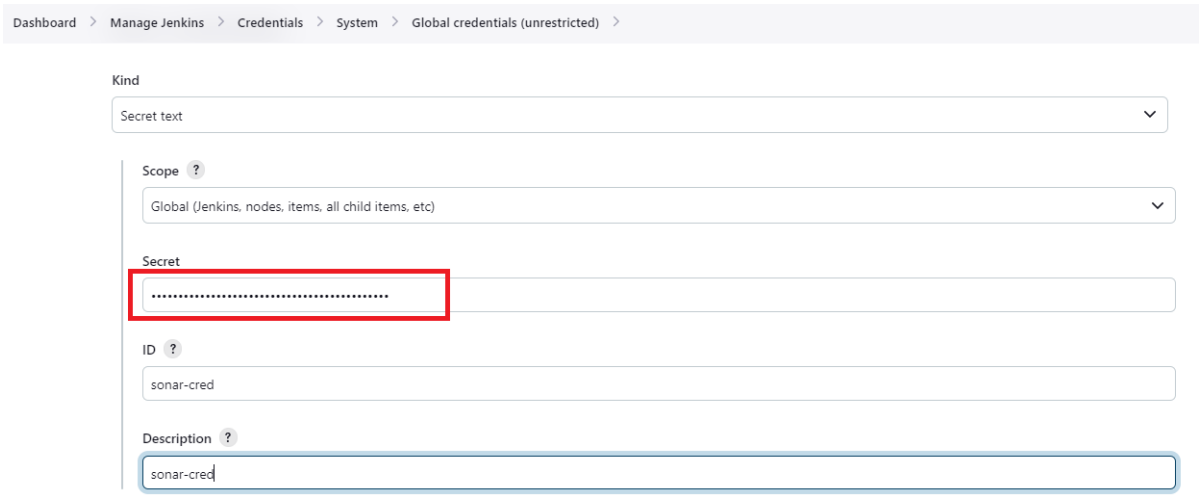



Give name as per your choice and generate It.

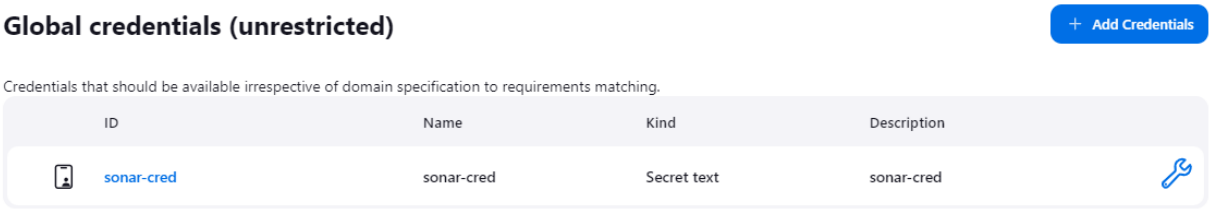


- Add token in Jenkins credentials

**Go to Jenkins Dashboard → Manage Jenkins → Credentials → Add Secret Text.**

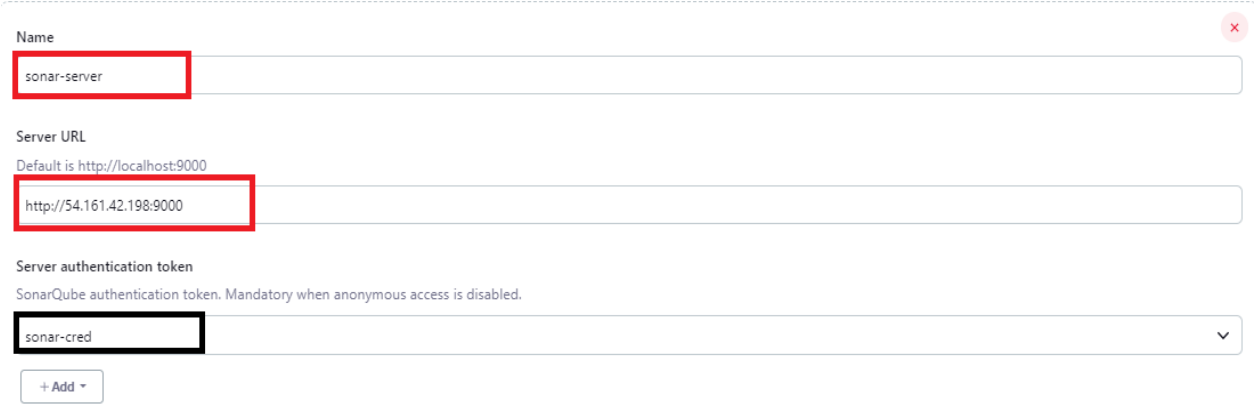



It should be look like this.



- Configure SonarQube Server.

**Go to Jenkins Dashboard → Manage Jenkins → SonarQube installations → Give name, Server url , Credentials → Click on Apply & Save**



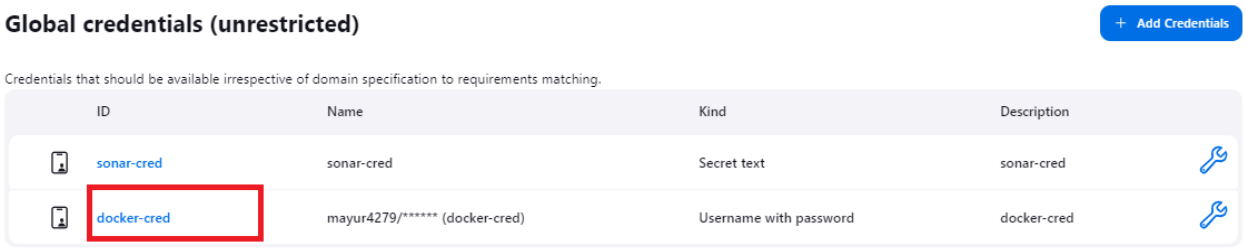

**Go to Jenkins Dashboard → Manage Jenkins → Tools → SonarQube Scanner installations → Add SonarQube scanner**

Step-8) **Integrate docker with Jenkins.**

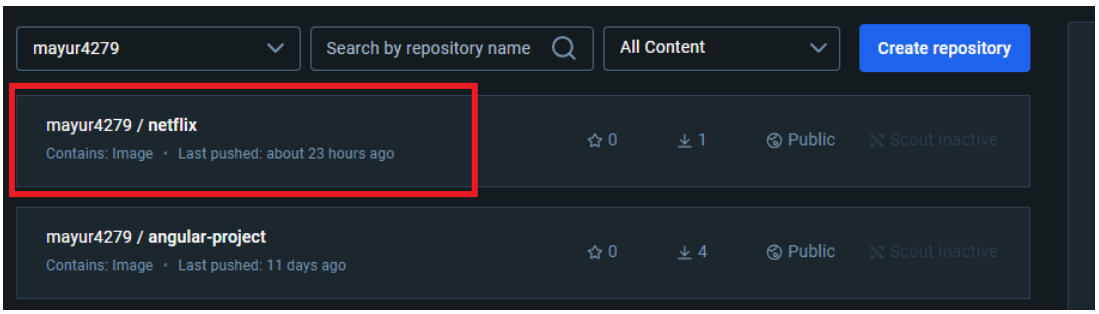
1. To securely handle DockerHub credentials in your Jenkins pipeline, follow these steps:

❖ Go to "Dashboard" → "Manage Jenkins" → "Manage Credentials."
❖ Click on "System" and then "Global credentials (unrestricted)."
❖ Click on "Add Credentials" on the left side.
❖ Choose "Secret text" as the kind of credentials.
❖ Enter your DockerHub credentials (Username and Password) and give the credentials an ID (e.g., "docker").
❖ Click "OK" to save your DockerHub credentials.

2. It Should look like this.




3. Make sure to create netflix  Repo in your dockrhub.

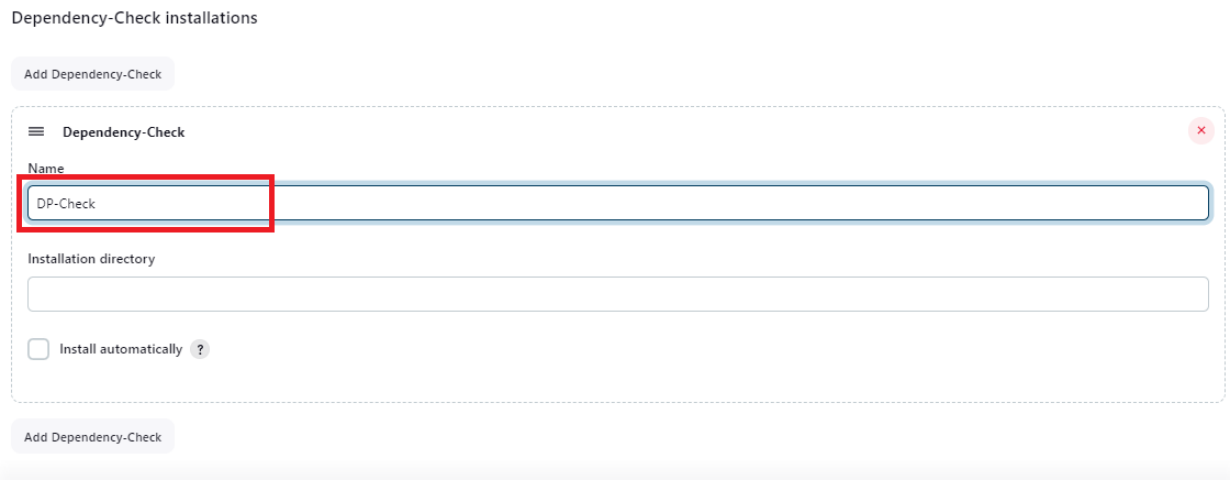



4. To avoid permission denied error add jenkins in docker group using following command

```
sudo usermod -aG docker jenkins

sudo systemctl restart jenkins
```

Step-9) Configure Dependency-Check Tool:

- Go to "Dashboard" → "Manage Jenkins" → "Global Tool Configuration."
- Find the section for "OWASP Dependency-Check."
- Add the tool's name, e.g., "DP-Check."
- Save your settings.




Step-10) Create CI/CD Pipeline - Develop a Jenkins pipeline for automated deployment.

**Use below Pipeline script.**

```
pipeline{
  agent any
  tools{
    jdk 'jdk17'
    nodejs 'node16'
  }
```

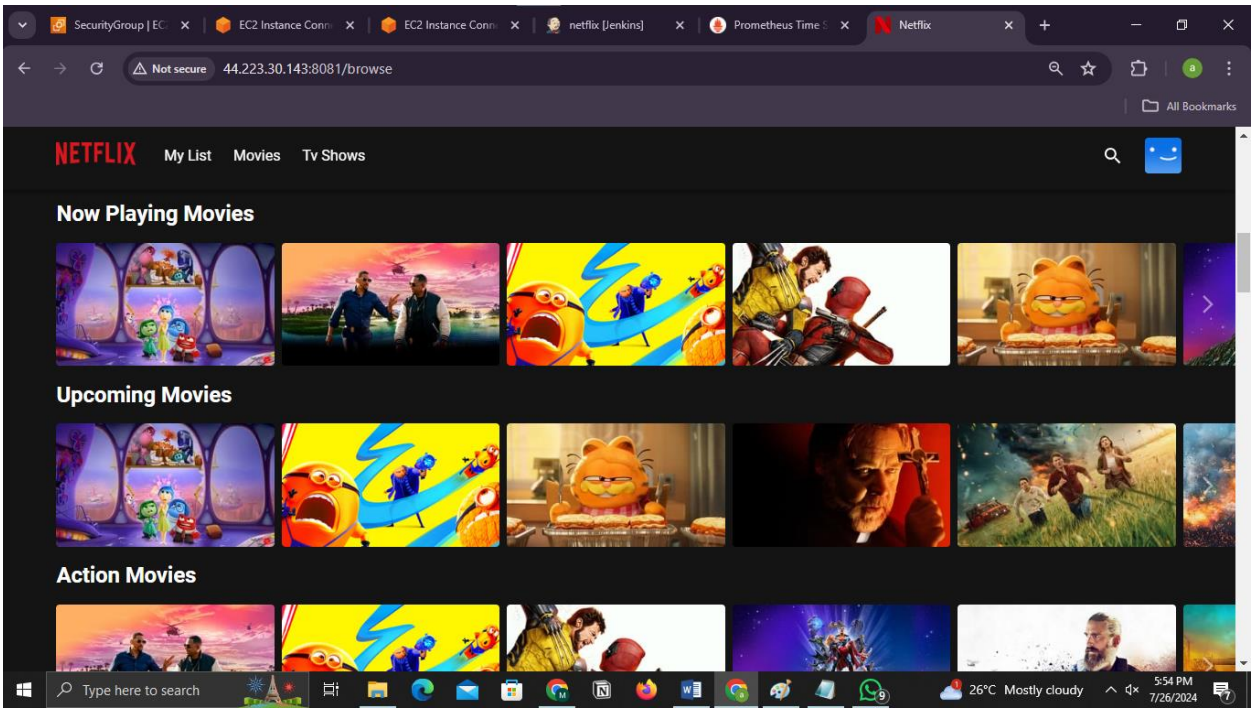
```
environment {
   SCANNER_HOME=tool 'sonar-scanner'
}
stages {
   stage('clean workspace'){
      steps{
         cleanWs()
      }
   }
   stage('Checkout from Git'){
      steps{
         git branch: 'main', url: 'https://github.com/mayur4279/DevSecOps-Project-Netflixapp.git'
      }
   }
   stage("Sonarqube 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-cred'
              echo "done"
          }
      }
    }
    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'
          echo "done"
      }
    }
    stage('TRIVY FS SCAN') {
      steps {
          sh "trivy fs . > trivyfs.txt"
      }
    }
```

```
        stage("Docker Build & Push"){
            steps{
                script{
                    withDockerRegistry(credentialsId: 'docker-cred', toolName: 'docker'){
                        sh "docker build --build-arg
TMDB_V3_API_KEY=42ea3b97bdbfd435337cce5c5e4d8be3 -t netflix ."
                        sh "docker tag netflix mayur4279/netflix:latest "
                        sh "docker push mayur4279/netflix:latest "
                    }
                }
            }
        }
        stage("TRIVY"){
            steps{
                sh "trivy image mayur4279/netflix:latest > trivyimage.txt"
            }
        }
        stage('Deploy to container'){
            steps{
                sh 'docker rm  -f netflix '
                sh 'docker run -d --name netflix -p 8081:80 mayur4279/netflix:latest'
            }
        }
    }
}
```

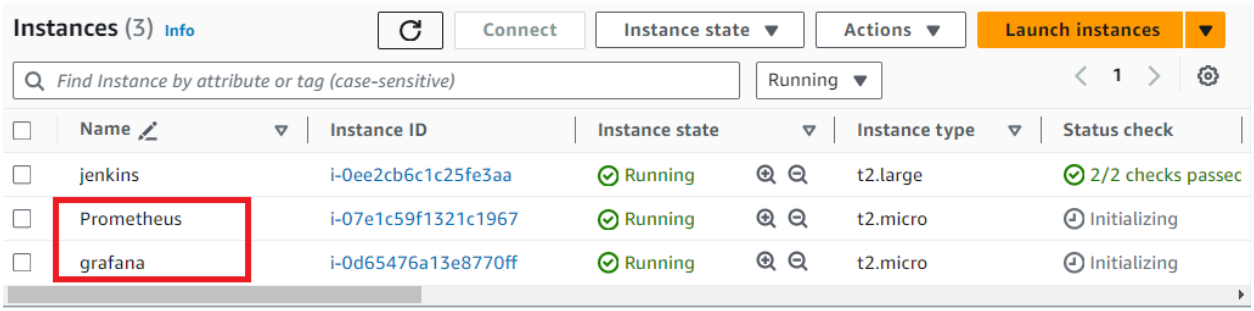
Successfully run our pipeline.


Successfully Able to access website using CICD pipeline



## Phase 4: Monitoring with (Prometheus and Grafana)

First, create a dedicated Linux server for Prometheus & Grafana




## Installing Prometheus (In Prometheus instance):

Download Prometheus:

```
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:

```
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 content 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
```

Here's a brief explanation of the key parts in this prometheus.service file:

- **User and Group** specify the Linux user and group under which Prometheus will run.
- **ExecStart** is where you specify the Prometheus binary path, the location of the configuration file (prometheus.yml), the storage directory, and other settings.
- **web.listen-address** configures Prometheus to listen on all network interfaces on port 9090.
- **web.enable-lifecycle** allows for management of Prometheus through API calls.

Enable and start Prometheus:

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

You can access Prometheus in a web browser using your server's IP and port 9090:

*Make sure to add port 9090 in security group*

```
http://<your-server-ip>:9090
```

## Installing Node Exporter in Prometheus:-

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_exp
orter-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 content to the node_exporter.service file:

```
[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.target
```

**Replace --collector.logind with any additional flags as needed.**

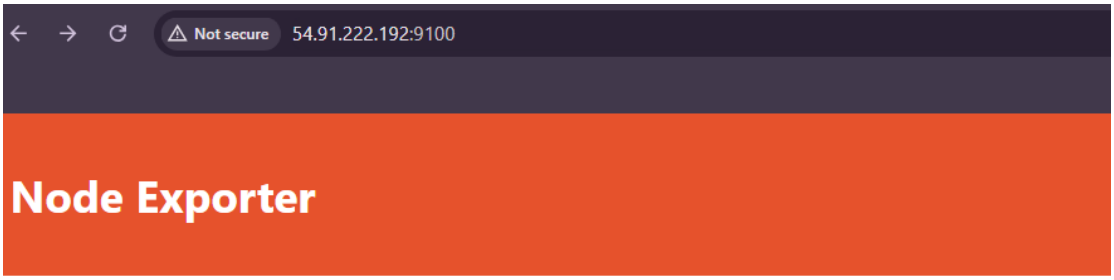
Enable and start Node Exporter:

```
sudo systemctl enable node_exporter
sudo systemctl start node_exporter
```

Verify the Node Exporter's status:

```
sudo systemctl status node_exporter
```

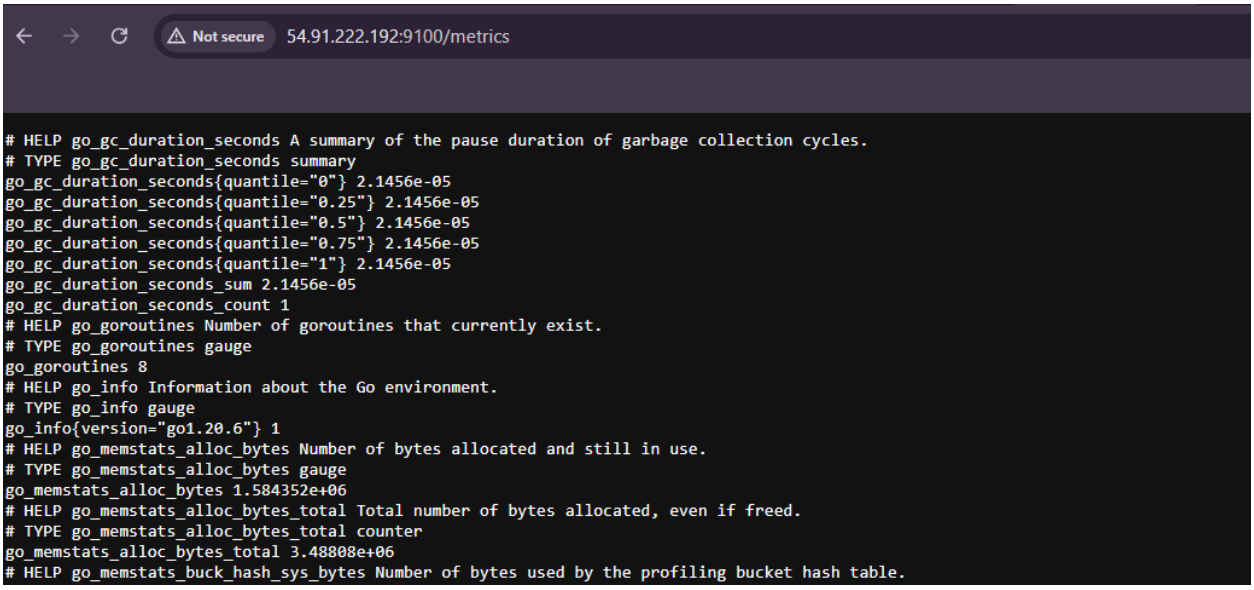
Now You can access Node Exporter metrics in Prometheus.




## Prometheus Node Exporter

Version: (version=1.6.1, branch=HEAD, revision=4a1b77600c1873a8233f3ffb55afcedbb63b8d84)

- Metrics



## Configure Prometheus Plugin Integration:

Integrate Jenkins with Prometheus to monitor the CI/CD pipeline.

Prometheus Configuration:

To configure Prometheus to scrape metrics from Node Exporter and Jenkins, you need to modify the **prometheus.yml** file. Here is an

Example prometheus.yml configuration for your setup:

```
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>']
```

Make sure to replace ***<your-jenkins-ip>*** *and* ***<your-jenkins-port>*** *with* the appropriate values for your Jenkins setup.

```
- job_name: "prometheus"

  # metrics_path defaults to '/metrics'
  # scheme defaults to 'http'.

  static_configs:
    - targets: ["localhost:9090"]


- job_name: 'node_exporter'
  static_configs:
    - targets: ['localhost:9100']

- job_name: 'jenkins'
  metrics_path: '/prometheus'
  static_configs:
    - targets: ['54.91.222.192:8080']
```

Check the validity of the configuration file use following command:

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

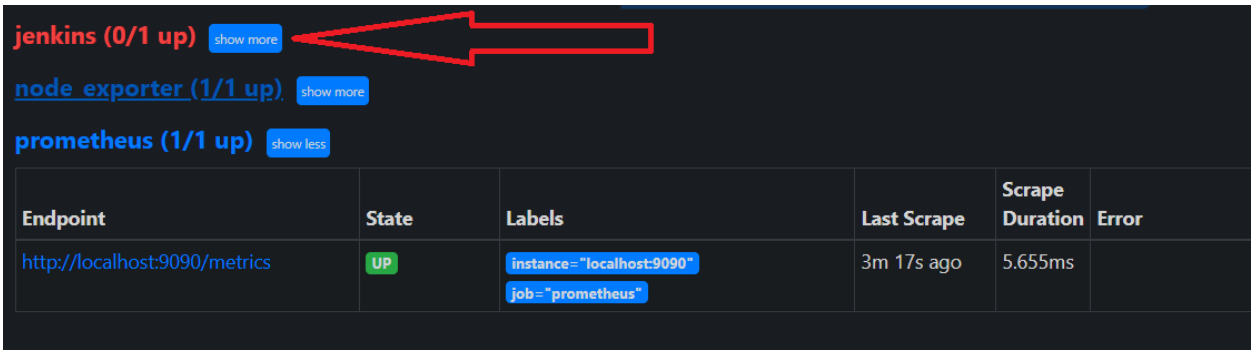
Reload the Prometheus configuration without restarting:

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

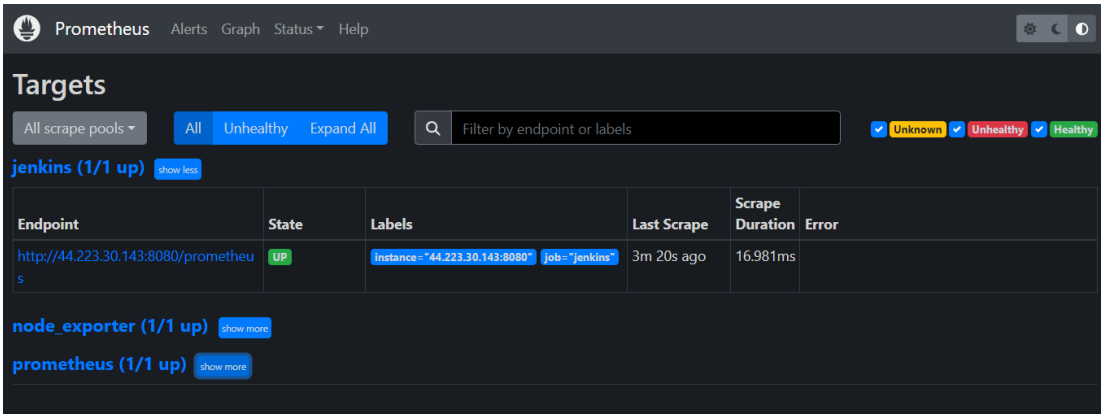
You can access Prometheus targets at:

```
http://<your-prometheus-ip>:9090/targets
```

This error is showing because we are not added **Prometheus metrics** plugin in Jenkins so let's add it...




After adding plugin our Premetheus server started capturing the matrises from jenkins..



## Installing Grafana (In grafana instance):

Step-1): Install Dependencies:

```
sudo apt-get update

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

Step-2: Add the GPG Key:

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

Step-3): 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
```

Step-4): Update and Install Grafana:

```
sudo apt-get update

sudo apt-get -y install grafana
```

Step-5): Enable and Start Grafana Service:

```
sudo systemctl enable grafana-server

#Then, start Grafana:

sudo systemctl start grafana-server
```

Step-6): Access Grafana Web Interface:

*Open a web browser and navigate to Grafana using your server's IP address. The default port for Grafana is 3000.*

For example:

http://<your-server-ip>:3000

You'll be prompted to log in to Grafana. The default username is "admin," and the default password is also "admin."

Step-7): 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.
- Choose "Prometheus" as the data source type.
- In the "HTTP" section:
    - Set the "URL" to http://localhost:9090 (assuming Prometheus is running on the same server).
    - Click the "Save & Test" button to ensure the data source is working.

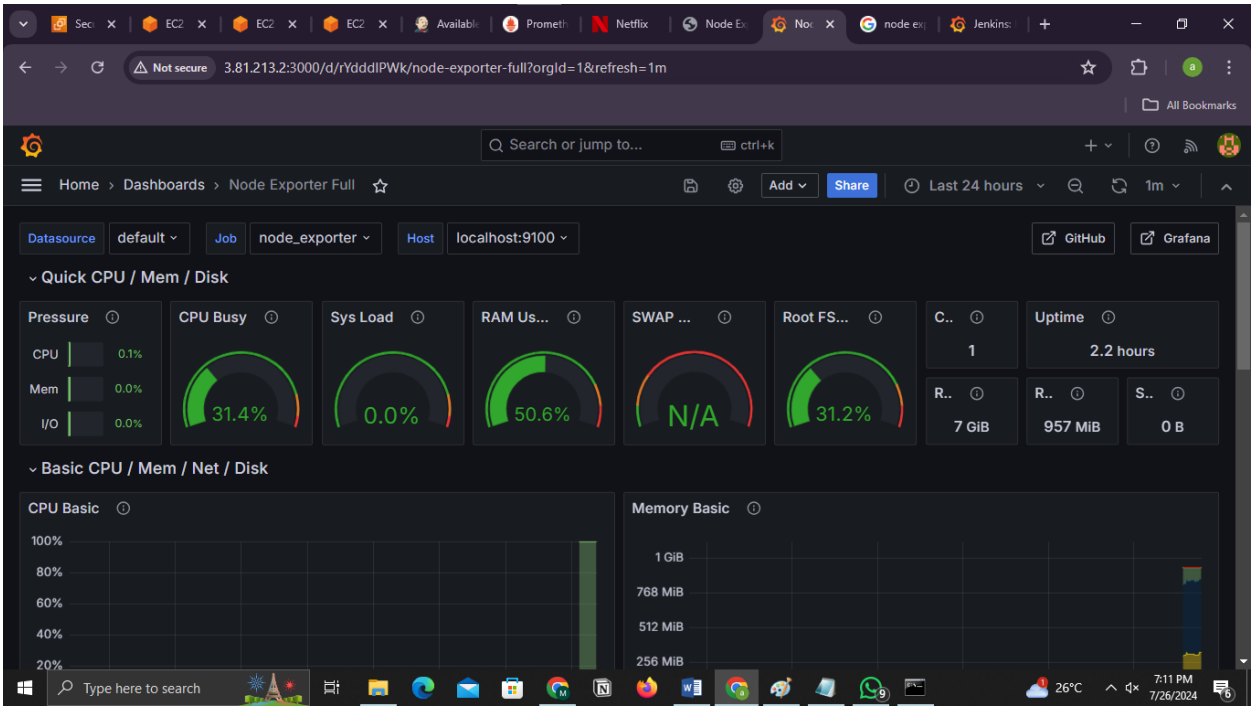
Step-8): Import a Dashboard:

To make it easier to view metrics, you can import a pre-configured dashboard. Follow these steps:

- 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 1860).
- Click the "Load" button.
- Select the data source you added (Prometheus) from the dropdown.
- Click on the "Import" button.

**You should now have a Grafana dashboard set up to visualize metrics from Prometheus.**

**Grafana is a powerful tool for creating visualizations and dashboards, and you can further customize it to suit your specific monitoring needs.**




That's it! successfully installed and set up Grafana to work with Prometheus for monitoring and visualization….

## Phase 5: Kubernetes:

### Create Kubernetes cluster with Nodegroup:

- Create role for cluster and nodegroup
  For cluster → use policy → AmazonEKSClusterPolicy
  For nodegroup → use policies →{ AmazonEC2ContainerRegistryReadOnly, AmazonEKS_CNI_Policy, AmazonEKSWorkerNodePolicy }

- Configure the eks cluster using following commands (in cloudshell)

```
aws eks update-kubeconfig --name my_cluster --region us-east-1
```

### Monitor Kubernetes with Prometheus

Prometheus is a powerful monitoring and alerting toolkit, and you'll use it to monitor your Kubernetes cluster. Additionally, you'll install the node exporter using Helm to collect metrics from your cluster nodes.

#### Install Node Exporter using Helm:

Node Exporter component allows you to collect system-level metrics from your cluster nodes.

Steps:-

Step-1) Install Helm binary using following commands in cloudshell

```
curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 > get_helm.sh

chmod 700 get_helm.sh

./get_helm.sh
```

Step-2) Add the Prometheus Community Helm repository:

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

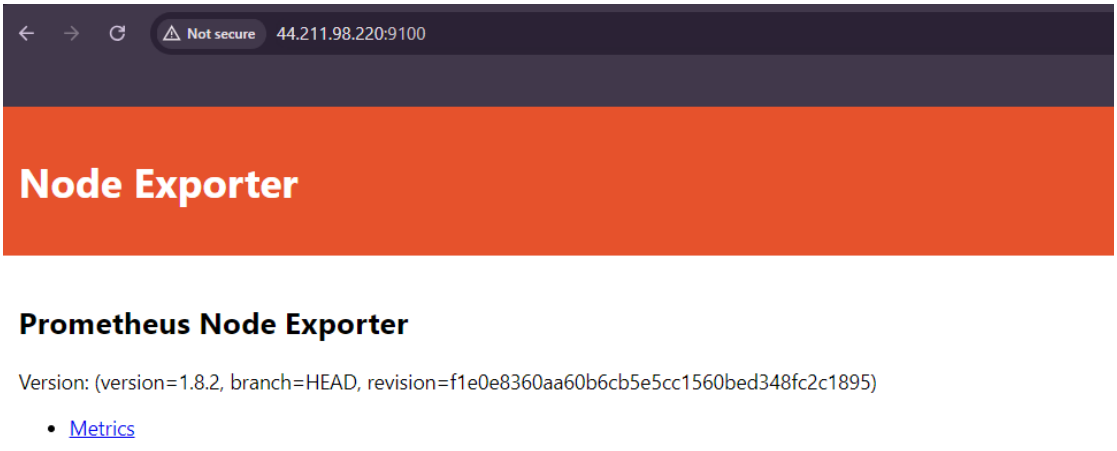
Step-3) Create a Kubernetes namespace for the Node Exporter:

```
kubectl create namespace prometheus-node-exporter
```

Step-4) Install the Node Exporter using Helm:

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

Step-5) We are able access our cluster matrix.  (nodeip:9100)

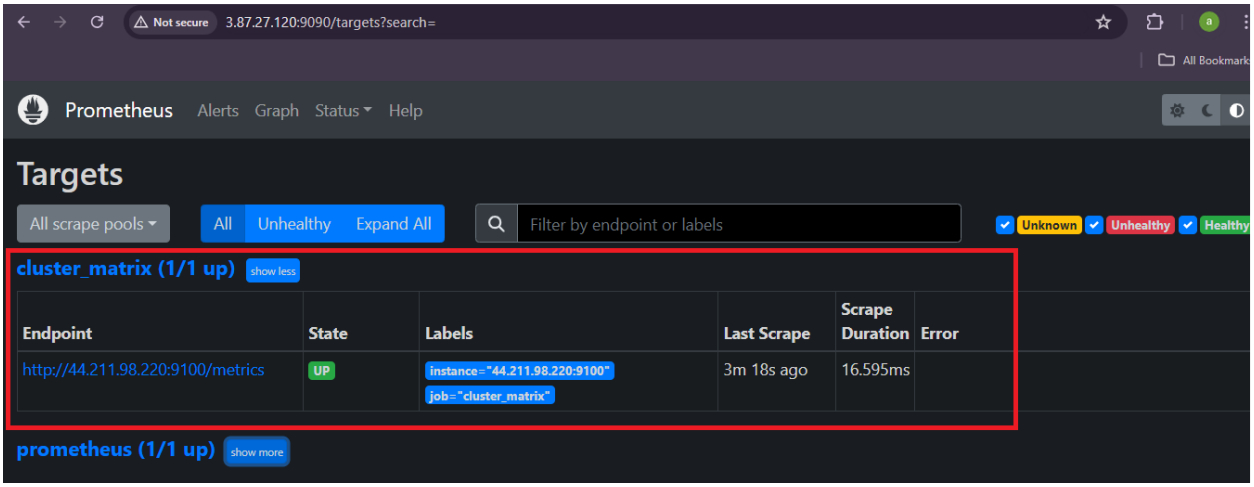


Step-4) Edit your **Prometheus.yaml** configuration file in your Prometheus server.

```
- job_name: "prometheus"

  # metrics_path defaults to '/metrics'
  # scheme defaults to 'http'.

  static_configs:
    - targets: ["localhost:9090"]


- job_name: 'cluster_matrix'
  metrics_path: '/metrics'
  static_configs:
    - targets: ['44.211.98.220:9100']
```

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.

Successfully able to capture matrix data from cluster.



## Deploy Application with ArgoCD

Step-1) Install ArgoCd in kubernetes

```
kubectl create namespace argocd

kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/v2.4.7/manifests/install.yaml
```

Step-2) Expose ArgoCd-server in kubernetes over **LoadBalancer.**

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

*Wait about 2 minutes for the LoadBalancer creation*

Step-3) Use below commands for view DNS endpoint.

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

echo $ARGOCD_SERVER
```

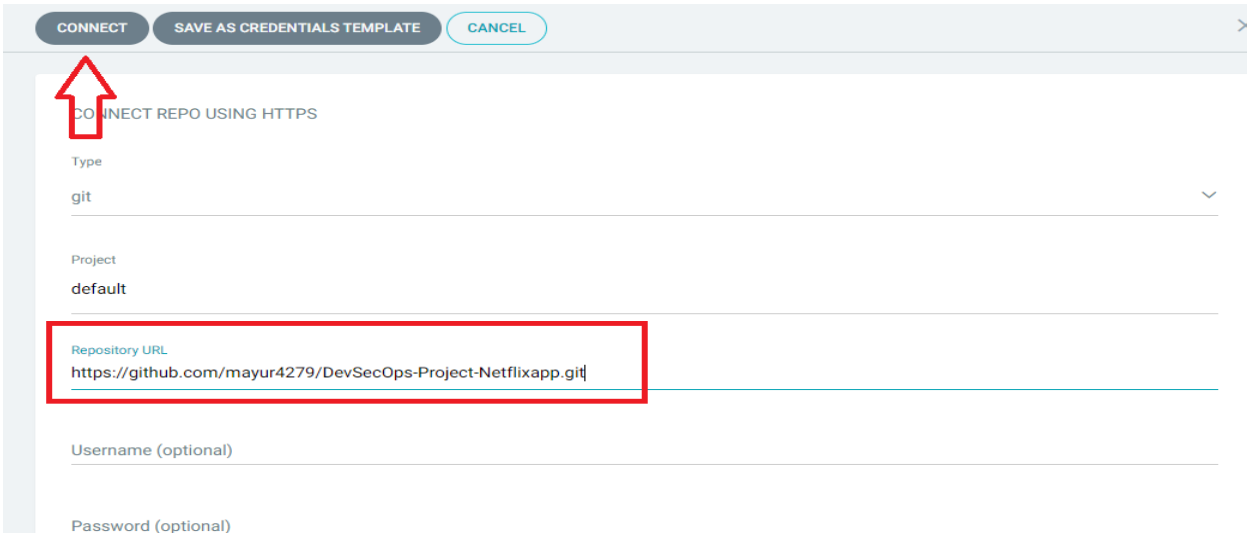
Step-3) use below command for grabbing the auto-generated password

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

echo $ARGO_PWD
```

Use, Username ➔ admin

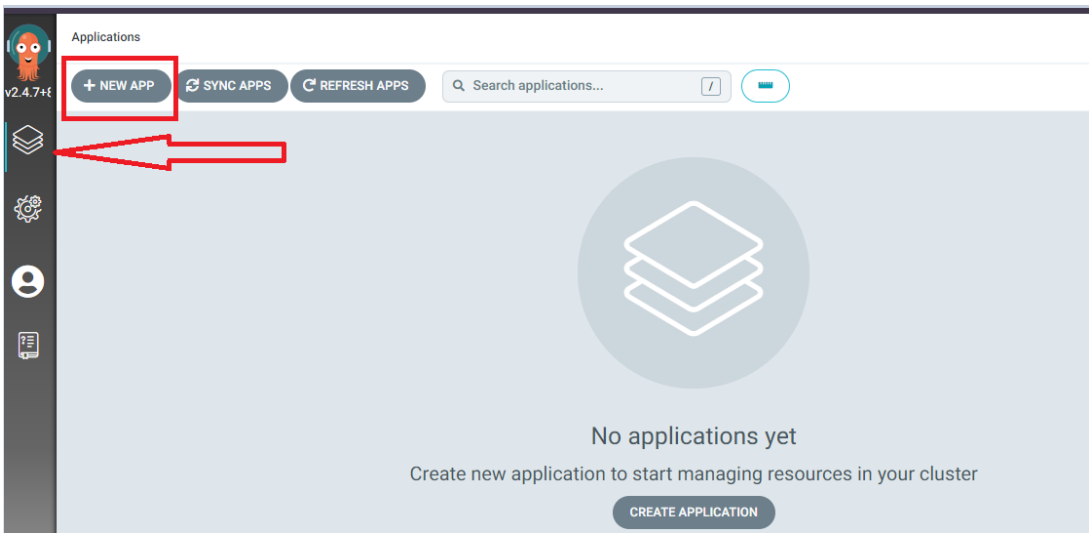
Password ➔ <your secret password>

Step-4) Set Your GitHub Repository as a Source:

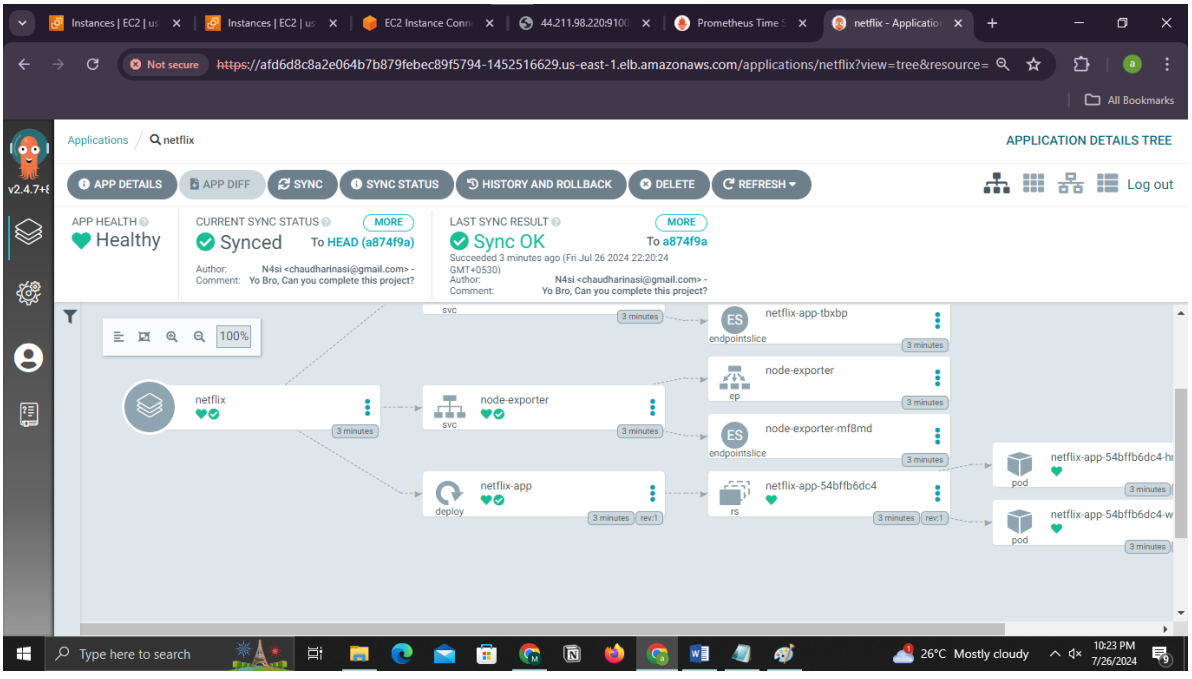




Step-5) Create an ArgoCD Application:

- name: Set the name for your application.
- destination: Define the destination where your application should be deployed.
- project: Specify the project the application belongs to.
- source: Set the source of your application, including the GitHub repository URL, revision, and the path to the application within the repository.
- syncPolicy: Configure the sync policy, including automatic syncing, pruning, and self-healing.

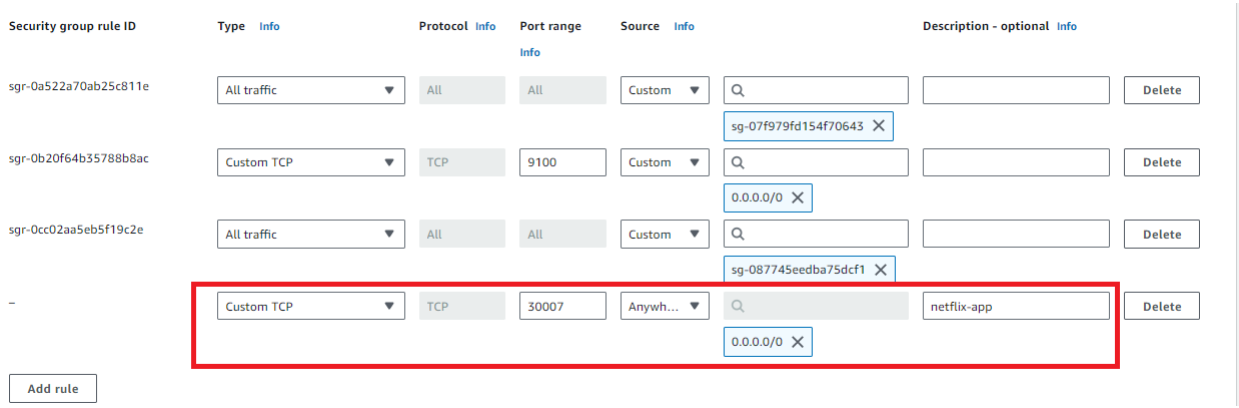



We successfully able to sync application from github



Step-6) Access your application

To Access the app make sure port 30007 is open in your security group and then open a new tab paste your **NodeIP:30007**, your app should be running.




Boom!! Now You can access you Netflix clone app  using NodeIp