DevSecOps Project with EC2, Docker, Jenkins, Prometheus, Grafana, and Kubernetes

This project involves deploying a Netflix application using AWS EC2, Docker, Jenkins, SonarQube, Trivy, Prometheus, Grafana, and Kubernetes. It covers phases of setup, security scanning, CI/CD automation, monitoring, and more.

Phase 1: Launching EC2 Instance

Setting Up EC2

1. **EC2 Instance Launch:**

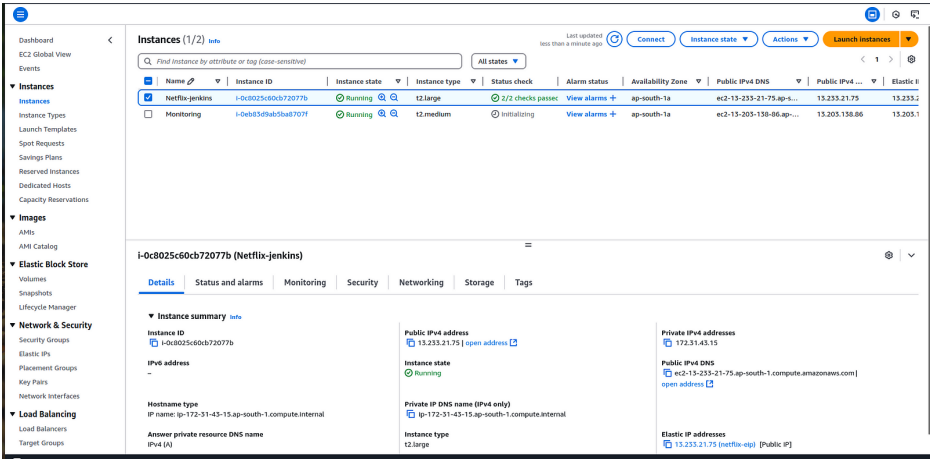
- Launched an EC2 instance with Ubuntu OS for the project.

2. Instance Type Selection:

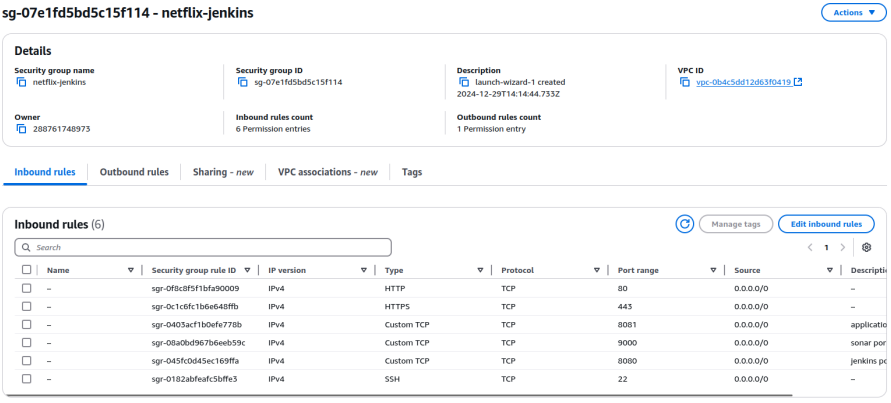
- Choose t2.large instance (2 vCPUs, 8 GB RAM) to efficiently handle the workload.

3. Elastic IP Association:

- Associated an Elastic IP to ensure a static public IP address for consistent access.



Ec2 instance Launching



This ports should be open in this server

Cloning the Application Code

Update and Clone: First, update the instance and clone the application code from GitHub.

```
git clone <URL>
```

Connecting to the EC2 Server

Phase 2: Security Setup

Install Docker: Set up Docker on the EC2 instance to containerize the application. By using the following commands.

```
sudo apt-get update
sudo apt-get install docker.io -y
```

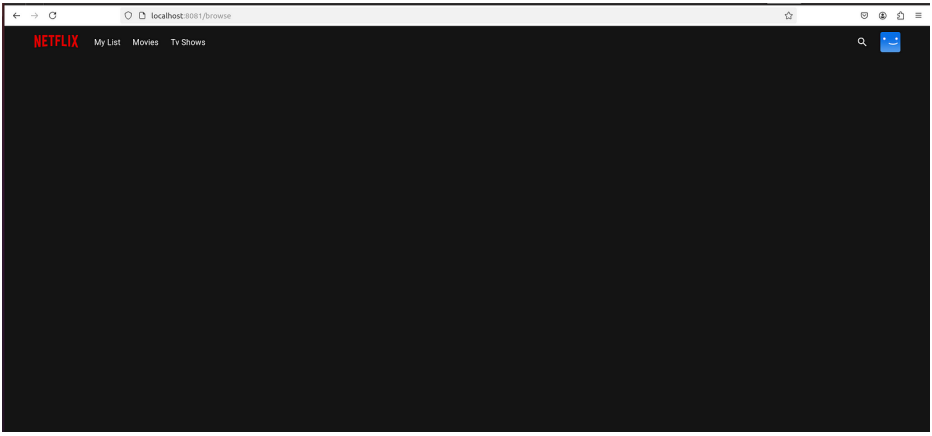
```
sudo usermod -aG docker $USER
newgrp docker
```

The command `sudo usermod -aG docker $USER` adds the current user to the docker group. It allows the user to run Docker commands without needing sudo privileges every time.

Run Docker Container:
Built and ran the Docker container to deploy the application securely and efficiently.

```
docker build -t netflix .
docker run -d --name netflix -p 8081:80 netfli›
```

Docker container



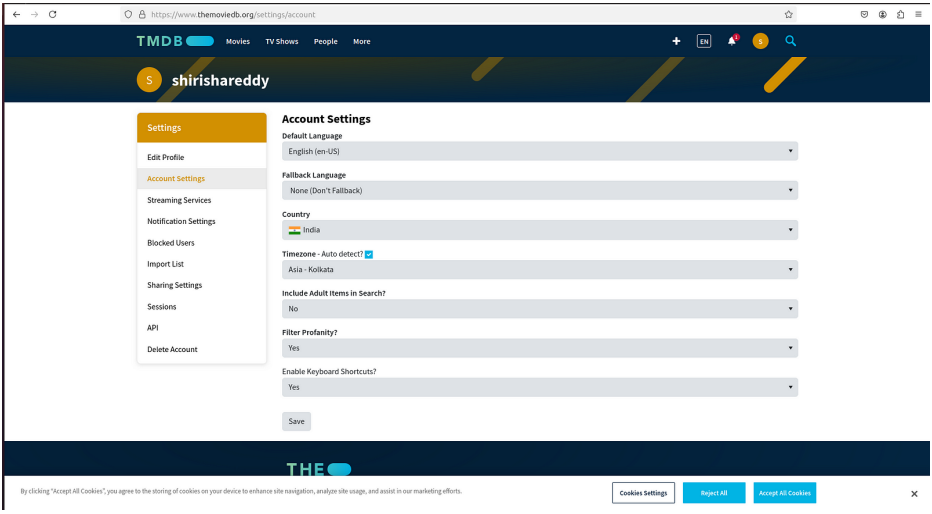
Accessing the app without TMDB API Key

Rebuild Docker Image with TMDB API Key

1. **TMDB?**
   TMDB (The Movie Database) is an online database that provides movie and TV show data through an API. We can use it to fetch information like movie titles, ratings, cast, etc.

2. **How to Get TMDB API Key:**

- Go to TMDB.
- Create an account or log in.
- Navigate to Account Settings > API.
- Generate a new API key.



TMDB

3. Why Use TMDB API Key?

- The API key is required to authenticate and get access to TMDB data.
- Without it, our requests will be rejected.

4. Difference With and Without API Key:

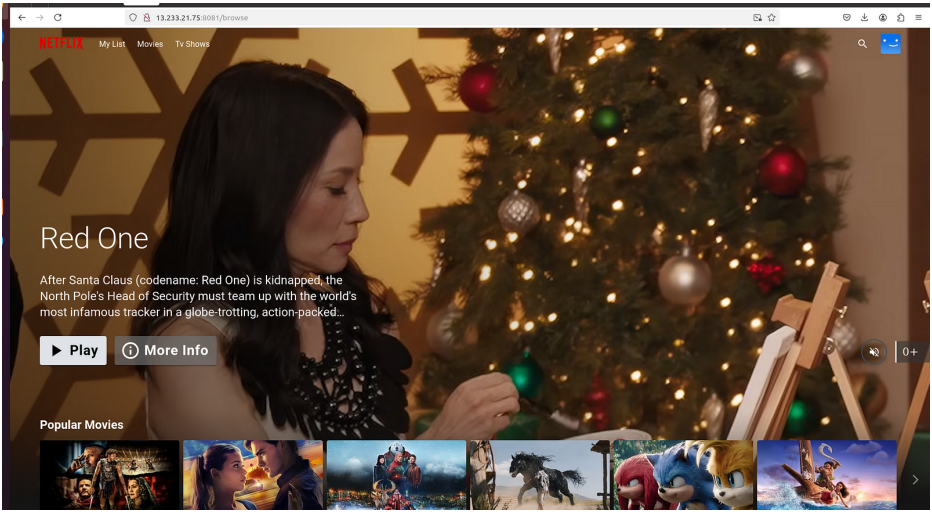
- **With API Key:** Full access to detailed movie information (e.g., ratings, cast, genres).
- **Without API Key:** Limited or no access to data, and the API will return an error.

5. Rebuild Docker Image with API Key:

Once we have the API key, include it in Docker container to enable access to TMDB data when the app runs.

Rebuild the Docker image after configuring the API key in application.

```
docker build --build-arg TMDB_V3_API_KEY=<your-
```



Accessing the app with TMDB API Key

Phase 3: CI/CD Pipeline with Jenkins

In this phase, we will install and configure Jenkins on an EC2 instance, set up essential plugins, and define a CI/CD pipeline. The pipeline will incorporate steps for building, analyzing code quality (SonarQube), and security scanning (Trivy). This will help us to automate the development process, ensuring secure and high-quality applications.

Step 1: Install Jenkins on EC2

Before starting, ensure that we have a Java Development Kit (JDK) installed, as Jenkins is built on Java and requires it to run.

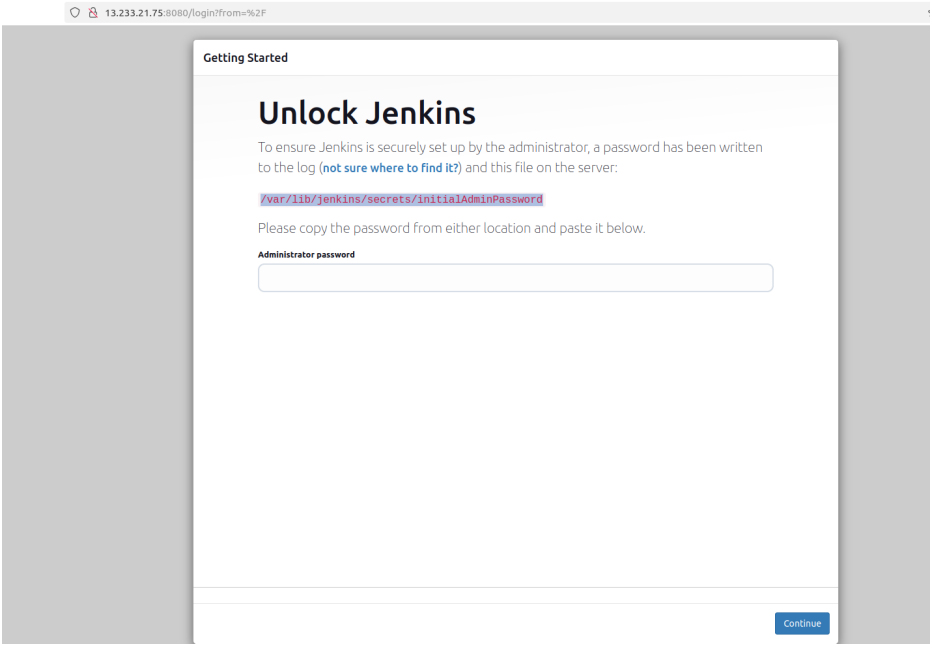
Install OpenJDK :
Verify Java Installation & Install Jenkins

```
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-D
# OpenJDK 64-Bit Server VM (build 17.0.8+7-Debi

# Jenkins
sudo wget -O /usr/share/keyrings/jenkins-keyrir
echo deb [signed-by=/usr/share/keyrings/jenkins
sudo apt-get update
sudo apt-get install jenkins
```

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

Access Jenkins Web Interface: Open browser and navigate to `http://<EC2-public-IP>:8080`



Jenkins UI

To unlock Jenkins for the first time, get the initial password:

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

Use this password to complete the setup. Once done, we will be able
to install the suggested plugins and set up an admin user.

Step 2: Configure Jenkins Plugins

To integrate SonarQube, Docker, and NodeJS into Jenkins, we need to install the necessary plugins.

Install Necessary Jenkins Plugins

Go to Manage Jenkins > Manage Plugins.
In the Available tab, search for and install the following plugins:

- SonarQube Scanner (for SonarQube integration)
- Docker (to integrate Docker builds)
- NodeJS (if needed for Node.js builds)
- Eclipse Temurin Installer (Install)

* Congufire Java and Nodejs in Global Tool

Configuration

Goto Manage Jenkins → Tools → Install JDK(17) and NodeJs(16)→ Click
on Apply and Save.

* OWASP Dependency-Check Configuration in Jenkins

OWASP Dependency-Check is a tool that identifies project dependencies and checks them for known, publicly disclosed vulnerabilities. It scans project's dependencies, such as libraries, frameworks, and tools, to identify if any have known security vulnerabilities.

In a **DevSecOps pipeline**, integrating OWASP Dependency-Check helps automate the security scanning of project's dependencies during the CI/CD process.

Steps to Install and Configure OWASP Dependency-Check in Jenkins

1. Install the OWASP Dependency-Check Plugin

Go to "Dashboard" in Jenkins web interface.

Navigate to "Manage Jenkins" → "Manage Plugins."

Click on the "Available" tab and search for "OWASP Dependency-Check Plugin." Click on **Install**.

This plugin will allow Jenkins to scan the dependencies of the project and report vulnerabilities.

2. Configure the OWASP Dependency-Check Tool in Jenkins

Once the plugin is installed, we need to configure the **OWASP Dependency-Check** tool in Jenkins:

Go to Jenkins Dashboard → **Manage Jenkins** → **Global Tool Configuration**.

Scroll down to the **OWASP Dependency-Check** section.

Click on **Add Dependency-Check** and provide the following details:

Name: Give a name to the tool, e.g., DP-Check (this name will be referenced in the pipeline).

Install automatically: Check this option to allow Jenkins to download and install the tool automatically.

Directory: Optionally, we can specify a custom directory to install the tool if needed.

Save configuration.

This configuration ensures that Jenkins has access to the **Dependency-Check** tool for security scans.

3. Configure OWASP Dependency-Check in the Jenkins Pipeline

To use the **Dependency-Check** tool within Jenkins pipeline, we will integrate it into the stages of our pipeline configuration.
Here's an example pipeline where **OWASP Dependency-Check** is used to scan project dependencies for vulnerabilities:

```
pipeline {
    agent any

    tools {
        jdk 'jdk17'
        nodejs 'node16'
    }

    environment {
        SCANNER_HOME = tool 'sonar-scanner'
    }

    stages {
        stage('Clean Workspace') {
            steps {
                cleanWs() // Cleans the workspa
            }
        }
        stage('Checkout from Git') {
            steps {
                git branch: 'main', url: '<URL>
            }
```

```
            }
            stage('Install Dependencies') {
                steps {
                    sh "npm install" // Install Noc
                }
            }
            stage('OWASP Dependency-Check Scan') {
                steps {
                    // Run Dependency-Check scan or
                    dependencyCheck additionalArgum

                    // Publish the Dependency-Check
                    dependencyCheckPublisher patter
                }
            }
        }
    }
```

* Install Docker Tools and Docker Plugins:

Go to “Dashboard” in Jenkins web interface.

Navigate to “**Manage Jenkins**” → “**Manage Plugins**.”

Click on the “**Available**” tab and search for “**Docker**.”

Check the following Docker-related plugins:

Docker

Docker Commons

Docker Pipeline

Docker API

docker-build-step

Click on the "**Install without restart**" button to install these plugins.

Add DockerHub Credentials:

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 DockerHub credentials (Username and Password) and give the credentials an ID (e.g., "docker").

Click "**OK**" to save DockerHub credentials.

Now, we have installed the **Dependency-Check** plugin, configured the tool, and added **Docker-related plugins** along with **DockerHub credentials** in Jenkins. We can now proceed with configuring Jenkins pipeline to include these tools and credentials in the CI/CD process.

Pipeline

Security Scanning with SonarQube & Trivy

In this phase, we'll be configuring **SonarQube** and **Trivy** for security scanning in the **DevSecOps pipeline**. This involves integrating SonarQube with Jenkins for continuous code analysis and using Trivy to scan Docker images for vulnerabilities.

Install and Configure SonarQube with Jenkins

SonarQube is a tool used for static code analysis to find bugs, code smells, and security vulnerabilities. Integrating SonarQube with Jenkins allows us to run security and quality scans automatically as part of our CI/CD pipeline.

Here's how to install and configure SonarQube with Jenkins:

Install and Run SonarQube in a Docker Container

First, run SonarQube in a Docker container on our EC2 instance:

```
docker run -d --name sonar -p 9000:9000 sonarqu
```

This command will pull the official **SonarQube** Docker image and run it in detached mode (`-d`), mapping the container's port 9000 to the EC2 instance's port 9000.

We can access **SonarQube** at `http://<EC2-public-IP>:9000` (default credentials are `admin/admin`).

Install SonarQube Plugin in Jenkins:

Open Jenkins in a web browser at `http://<EC2-public-IP>:8080` .

Go to **Manage Jenkins > Manage Plugins**.

In the **Available** tab, search for **SonarQube Scanner**.

Check the box next to **SonarQube Scanner** and click **Install without restart** or **with restart**.

Configure SonarQube Server in Jenkins:

Once the plugin is installed, navigate to **Manage Jenkins > Configure System**.

Scroll down to **SonarQube Servers** and click **Add SonarQube**.

Fill in the following details:

Name: Enter a name for the SonarQube instance (e.g., **SonarQube**).

SonarQube Server URL: `http://<EC2-public-IP>:9000` .

Authentication Token: Generate an authentication token from SonarQube:

Log in as **admin** and navigate to **My Account > Security > Generate Tokens**.

Copy the token and paste it into Jenkins under **Authentication Token**.

Click **Save**.

Configure SonarQube Scanner in Jenkins

Now, configure the **SonarQube Scanner** in Jenkins to analyze code during the CI/CD pipeline.

Go to **Global Tool Configuration**:

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

Scroll down to **SonarQube Scanner** and click **Add SonarQube Scanner**.

Give it a name (e.g., **SonarQube Scanner**).

Select **Install automatically** to have Jenkins download and

install the scanner.

We can also choose **Install from specific location** if we have a custom setup.

Click **Save**.

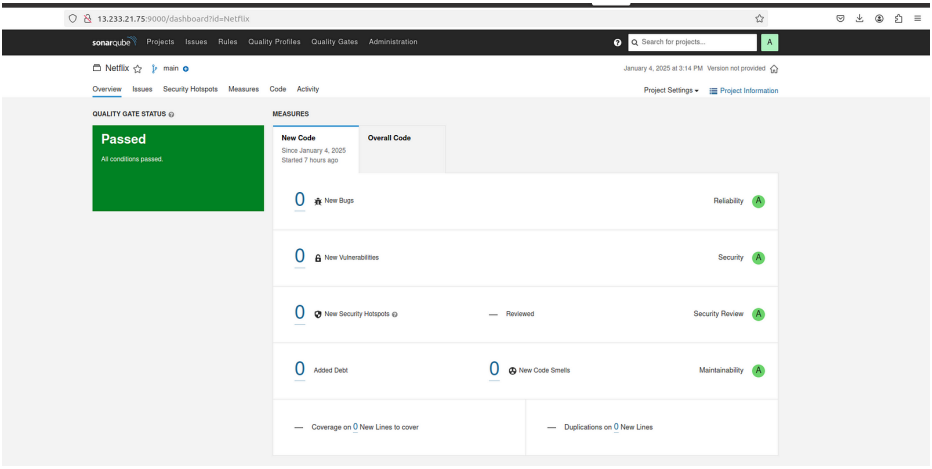
Configure Jenkins Pipeline for SonarQube Analysis

We can now configure a Jenkins pipeline to integrate **SonarQube** scanning. Depending on whether we use a **Freestyle Project** or **Pipeline**.

For **Pipeline Job (Jenkinsfile)**: If we use a Jenkinsfile for pipeline as code, here's a basic pipeline setup:

```
pipeline {
    agent any
    environment {
        SONARQUBE = 'SonarQube' // Name of Sona
    }
    stages {
        stage('Build') {
            steps {
                script {
                    // Your build steps here
                }
            }
        }
        stage('SonarQube Analysis') {
```

```
                steps {
                    script {
                        // Run SonarQube Scanner fo
                        withSonarQubeEnv('SonarQube
                            sh 'mvn clean install s
                        }
                    }
                }
            }
        }
    }
```



SonarQube Analysis

Install and Run Trivy for Docker Image Scanning

Trivy is a simple yet powerful security scanner for Docker images that detects vulnerabilities and misconfigurations.

1. Install Trivy on EC2

Run the following command to install Trivy on EC2 instance:

```
sudo apt-get install trivy
```

2. Scan Docker Image with Trivy

To scan Docker image for vulnerabilities, run the following command:

```
trivy image <imageid>
```

Replace `<image_id>` with the ID or name of the Docker image you want to scan (e.g., `netflix:latest`).

The output will show the vulnerabilities detected in the image, with details like severity and affected packages.

Img scanning through trivy

Integrating SonarQube and Trivy into Jenkins Pipeline

After setting up **SonarQube** and **Trivy**, we can integrate both tools into Jenkins pipeline to scan the code and Docker images automatically during the CI/CD process.

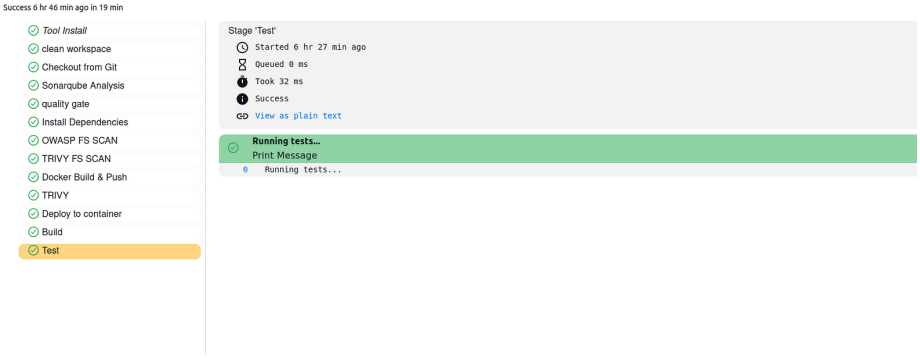
- **SonarQube**: Scans the code for quality issues, bugs, code smells, and security vulnerabilities.
- **Trivy**: Scans Docker images for known vulnerabilities.

```
pipeline {
    agent any
    tools {
        jdk 'jdk17'
        nodejs 'node16'
    }
    environment {
        SCANNER_HOME = tool 'sonar-scanner'
```

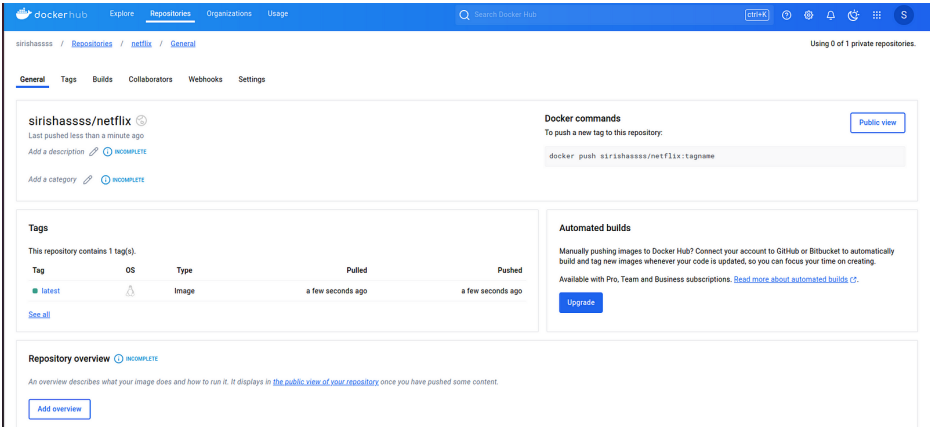
```
    }
    stages {
        stage('clean workspace') {
            steps {
                cleanWs()
            }
        }
        stage('Checkout from Git') {
            steps {
                git branch: 'main', url: '<URL>
            }
        }
        stage('SonarQube Analysis') {
            steps {
                withSonarQubeEnv('sonar-server'
                    sh '''
                    $SCANNER_HOME/bin/sonar-sca
                    '''
                }
            }
        }
        stage('Quality Gate') {
            steps {
                script {
                    waitForQualityGate abortPip
                }
            }
        }
        stage('Install Dependencies') {
            steps {
                sh "npm install"
            }
        }
```

```
        stage('OWASP FS Scan') {
            steps {
                dependencyCheck additionalArgum
                dependencyCheckPublisher patter
            }
        }
        stage('TRIVY FS Scan') {
            steps {
                sh "trivy fs . > trivyfs.txt"
            }
        }
        stage('Docker Build & Push') {
            steps {
                script {
                    withDockerRegistry(credenti
                        sh "docker build --buil
                        sh "docker tag netflix
                        sh "docker push sirisha
                    }
                }
            }
        }
        stage('TRIVY Image Scan') {
            steps {
                sh "trivy image sirishassss/net
            }
        }
        stage('Deploy to container') {
            steps {
                sh 'docker run -d -p 8081:80 si
            }
        }
    }
```

```
    }
```



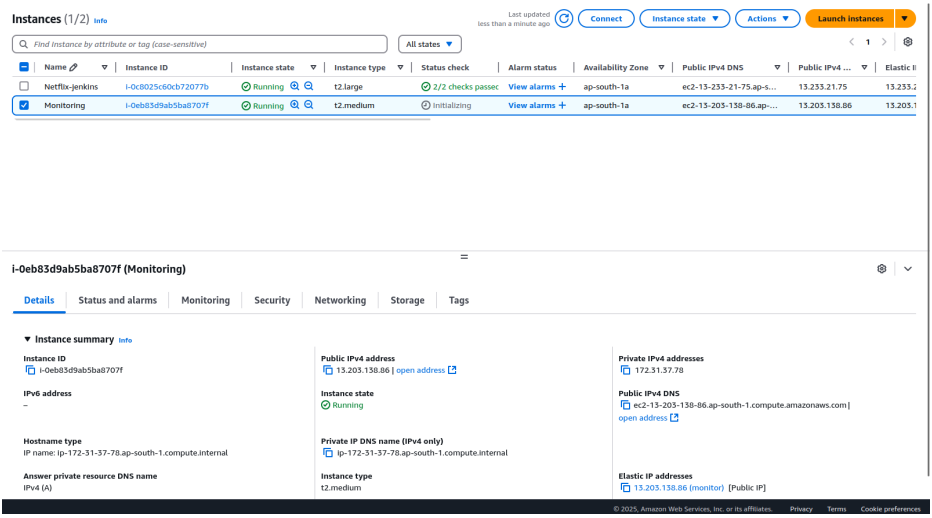
Pipeline



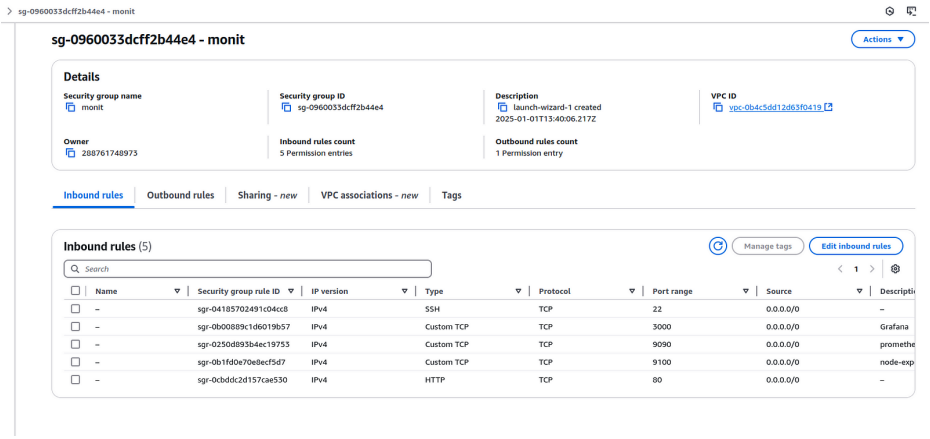
Img pushed to DockerHub

Phase 4: Monitoring

We can set up **Prometheus** and **Grafana** on the same EC2 instance as Jenkins server. However, if the instance starts running slowly because it's handling both Jenkins and the monitoring tools, it's better to **separate them.**



Instance to setup Monitoring tools



This ports should be open in this server

- By moving **Prometheus** and **Grafana** to a different EC2 instance, we
  can make sure that monitoring doesn't slow down Jenkins pipeline. This way, we can keep both Jenkins and the monitoring tools running smoothly without affecting each other.

Install Prometheus and Grafana:

- Set up Prometheus and Grafana to monitor application.

Prometheus is an open-source monitoring tool that collects and stores metrics as time-series data. It helps monitor applications, infrastructure, and services, providing powerful querying and alerting capabilities.

Installing Prometheus:

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

```
sudo useradd --system --no-create-home --shell
wget https://github.com/prometheus/prometheus/r
```

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

Set ownership for directories:

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

Create a systemd unit configuration file for Prometheus:

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

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/consc
  --web.console.libraries=/etc/prometheus/consc
```

```
  --web.listen-address=0.0.0.0:9090 \
  --web.enable-lifecycle

[Install]
WantedBy=multi-user.target
```

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 we 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.

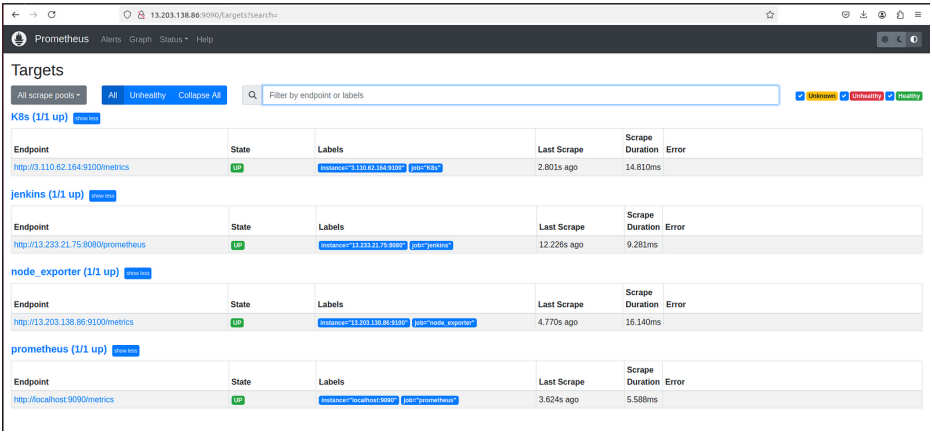
Enable and start Prometheus:

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

Verify Prometheus's status:

```
sudo systemctl status prometheus
```

We can access Prometheus in a web browser using server's IP and port 9090: `http://<your-server-ip>:9090`



I have added the Targets in the prometheus

Installing Node Exporter:

Node Exporter is an open-source Prometheus exporter that exposes hardware and OS-level metrics from Linux and Unix-based

systems. It collects data like CPU usage, memory, disk, and network statistics, allowing Prometheus to monitor system performance.

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

```
sudo useradd --system --no-create-home --shell
wget https://github.com/prometheus/node_exporte
```

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_ex
rm -rf node_exporter*
```

Create a systemd unit configuration file for Node Exporter:

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

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 --collec
[Install]
WantedBy=multi-user.target
```

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
```

We can access Node Exporter metrics in Prometheus.

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 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-jenk
```

replace `<your-jenkins-ip>` and `<your-jenkins-port>` with the appropriate values for your Jenkins setup.

Check the validity of the configuration file:

```
promtool check config /etc/prometheus/prometheu
```

Reload the Prometheus configuration without restarting:

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

We can access Prometheus targets at: `http://<your-prometheus-ip>:9090/targets`

Install Grafana

Grafana is an open-source data visualization and monitoring platform that allows users to create interactive dashboards. It integrates with various data sources, including Prometheus, to visualize time-series data, metrics, and logs for monitoring applications, infrastructure, and services.

Install Grafana on Ubuntu and Set it up to Work with Prometheus

Step 1: Install Dependencies:

First, ensure that all necessary dependencies are installed:

```
sudo apt-get update
sudo apt-get install -y apt-transport-https so
```

Step 2: Add the GPG Key:

Add the GPG key for Grafana:

```
wget -q -O - https://packages.grafana.com/gpg.k
```

Step 3: Add Grafana Repository:

Add the repository for Grafana stable releases:

```
echo "deb https://packages.grafana.com/oss/deb
```

Step 4:Update the package list and install Grafana:

```
sudo apt-get update
sudo apt-get -y install grafana
```

Step 5: Enable and Start Grafana Service:

To automatically start Grafana after a reboot, enable the service:

```
sudo systemctl enable grafana-server
```

Then, start Grafana:

```
sudo systemctl start grafana-server
```

Step 6: Check Grafana Status:

Verify the status of the Grafana service to ensure it's running correctly:

```
sudo systemctl status grafana-server
```

Step 7: Access Grafana Web Interface:

Open a web browser and navigate to Grafana using server's IP address. The default port for Grafana is 3000. `http://<your-server-ip>:3000`

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

Step 8: Change the Default Password:

When we log in for the first time, Grafana will prompt us to change the default password for security reasons. Follow the prompts to set a new password.

Step 9: Add Prometheus Data Source:

To visualize metrics, we 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 10: Import a Dashboard:

To make it easier to view metrics, we 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 we want to import (e.g., code 1860).

Click the “Load” button.

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

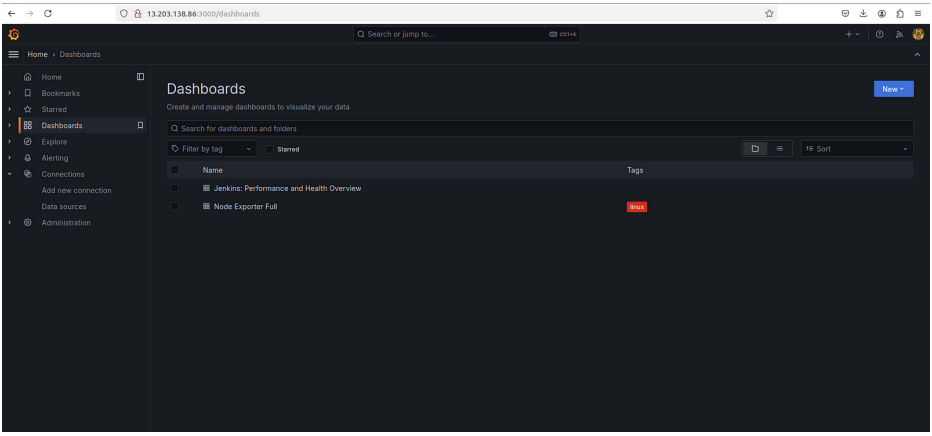
Click on the “Import” button.

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

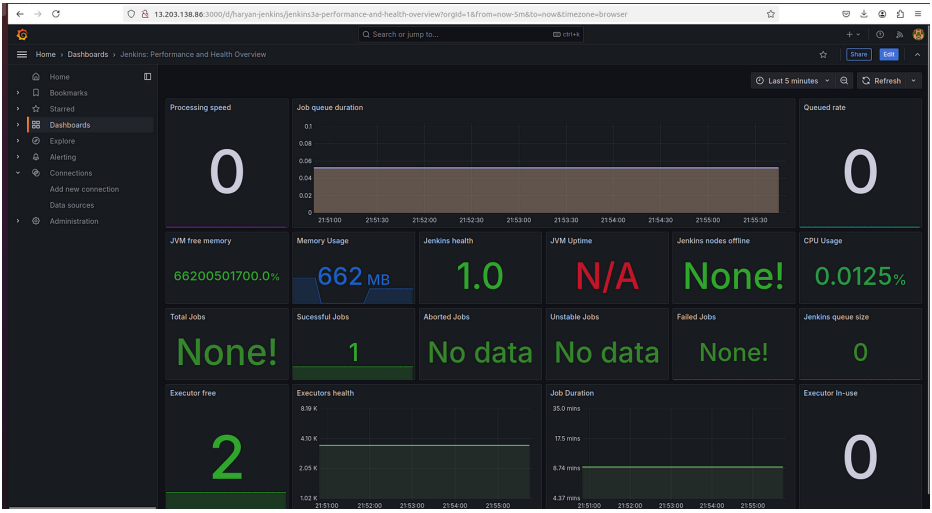
Grafana is a powerful tool for creating visualizations and dashboards,

Configure Prometheus Plugin Integration:

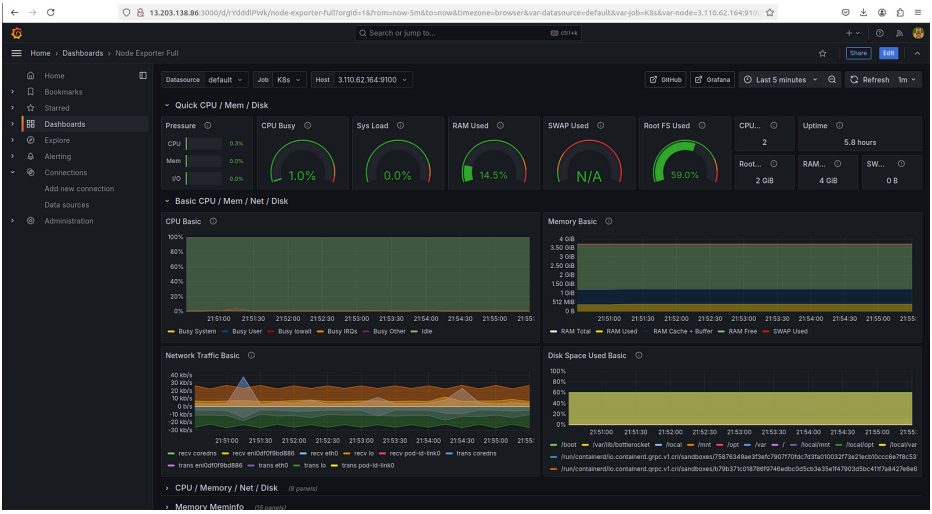
Integrate Jenkins with Prometheus to monitor the CI/CD pipeline.



Grafana dashboard after configuring Prometheus.



Jenkins Performance and overview

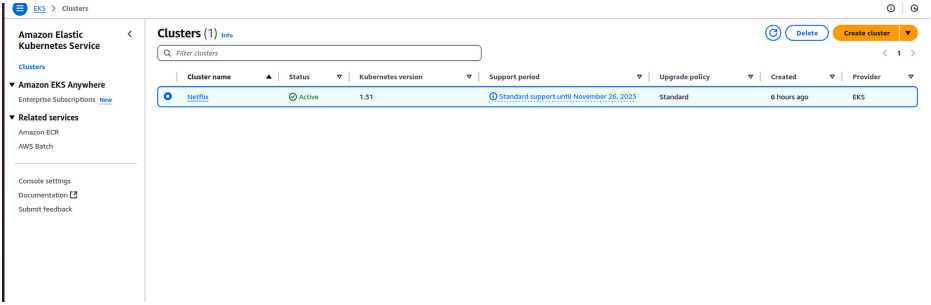


Node Exporter(Metrics Overview)

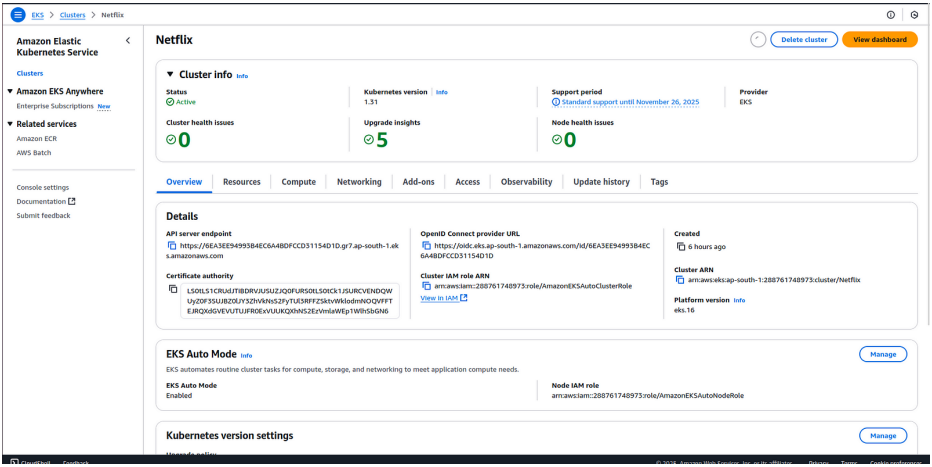
Phase 6: Kubernetes Cluster and Monitoring

EKS Cluster Setup

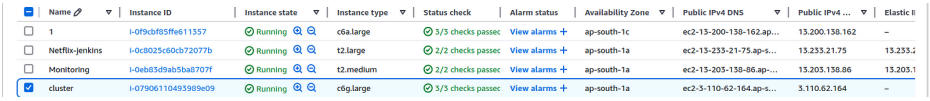
In this phase, we'll set up a Kubernetes cluster with node groups in console. This will provide a scalable environment to deploy and manage applications.



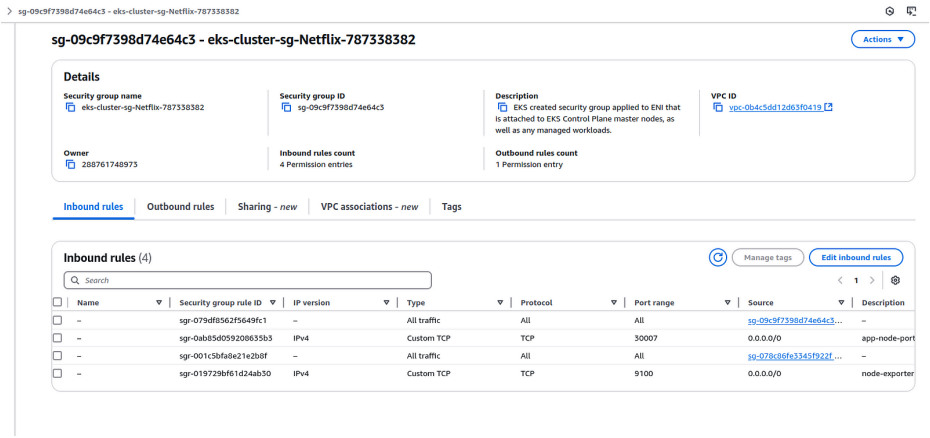
EKS Cluster



Cluster



c6alarge instances are nodes for the cluster



Ports

Setting Up AWS CLI and Configuring Credentials

Before accessing the **EKS cluster**, we need to install and configure the **AWS CLI**. Follow these steps:

Install AWS CLI:

First, we'll need to install the AWS CLI on our machine. We can do this by running the following commands:

```
sudo apt-get update
sudo apt-get install awscli
aws --version
```

Configure AWS CLI with Credentials:

Next, we need to configure AWS credentials (which will allow us to access our AWS account). Run the following command:

```
aws configure
```

It will prompt us to enter the following information:

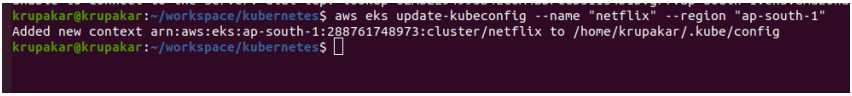
> **AWS Access Key ID**: We can find this in our AWS account (under IAM > Users > [your user] > Security Credentials > Access keys).
>
> **AWS Secret Access Key**: This is also available under IAM user's security credentials.
>
> **Default region name**: Enter the region where EKS cluster is located, such as `ap-south-1` or `us-east-1` .
>
> **Default output format**: We can choose between `json` , `text` , or `table` (typically `json` is fine).

Access the EKS Cluster:

Now the AWS CLI is set up, we can use it to access **EKS cluster** by updating kubeconfig file. Run the following command:

```
aws eks update-kubeconfig --name "Cluster-Name'
```

This will configure `kubectl` to interact with our **EKS cluster.**



Accessing **EKS cluster** by updating kubeconfig file

Monitor Kubernetes with Prometheus

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

Installing Helm

Helm is a package manager for Kubernetes that simplifies the deployment and management of applications and services. It allows us to easily install and configure applications using "charts" (pre-configured Kubernetes resources). To install Prometheus Node Exporter (and other services) on Kubernetes cluster, we'll need to have Helm install

Install Helm on Local Machine or Server:

```
sudo apt-get update
sudo apt-get install -y apt-transport-https
curl https://baltocdn.com/helm/signing.asc | su
sudo apt-get install -y helm
```

Install Node Exporter using Helm

To begin monitoring Kubernetes cluster, we'll install the Prometheus Node Exporter. This component allows us to collect system-level metrics from cluster nodes. Here are the steps to install the Node Exporter using Helm:

Add the Prometheus Community Helm repository:

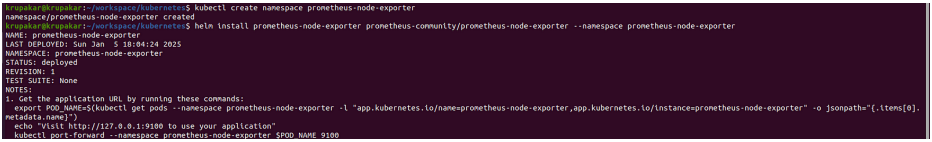
```
helm repo add prometheus-community https://prom
```

2. Create a Kubernetes namespace for the Node Exporter:

```
kubectl create namespace prometheus-node-export
```

3. Install the Node Exporter using Helm:

```
helm install prometheus-node-exporter prometheu
```



prometheus-node-exporter installation throuh helm

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

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

```
- job_name: 'Netflix(job name)'
    metrics_path: '/metrics'
    static_configs:
      - targets: ['node1Ip:9100']
```

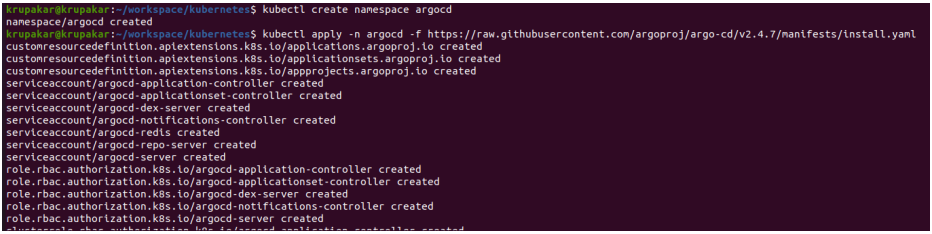
Reload or restart Prometheus to apply these changes to configuration.

To deploy an application with ArgoCD, we can follow these steps:

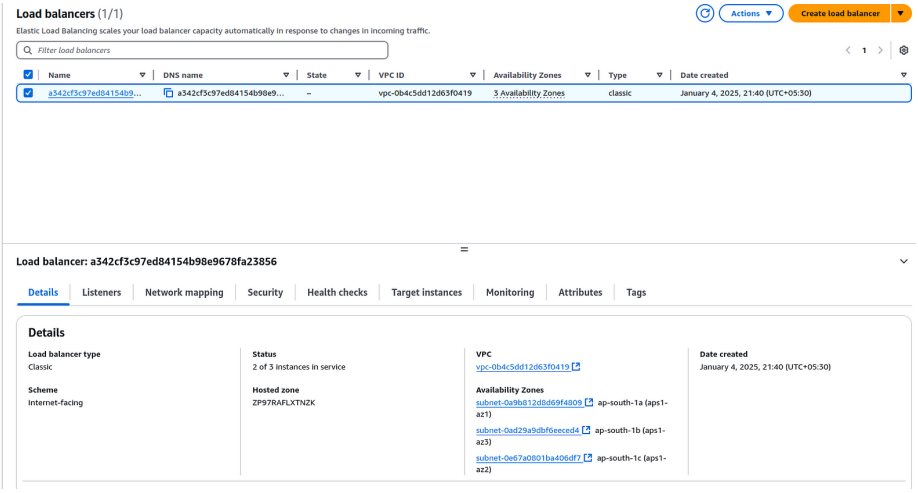
Deploy Application with ArgoCD

Install ArgoCD:

We can install ArgoCD on Kubernetes cluster by following the instructions provided in the EKS Workshop documentation.



Argocd installation



LB

1. **Set the GitHub Repository as a Source:**

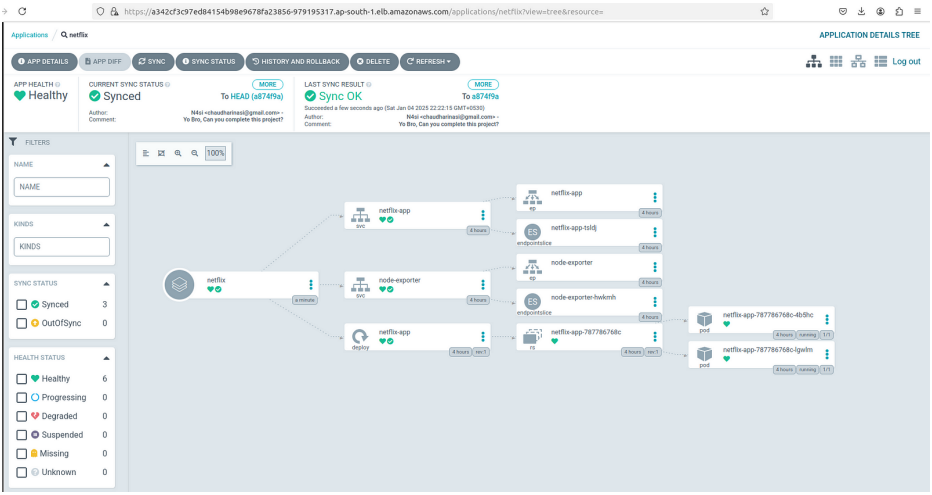
2. After installing ArgoCD, we need to set up the GitHub repository as a source for our application deployment. This typically involves configuring the connection to our repository and defining the source for our ArgoCD application. The specific steps will depend on our setup and requirements.

3. **Create an ArgoCD Application:**

- - `name` : Set the name for application.
  - `destination` : Define the destination where application should be deployed.
  - `project` : Specify the project the application belongs to.

`source` : Set the source of 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.

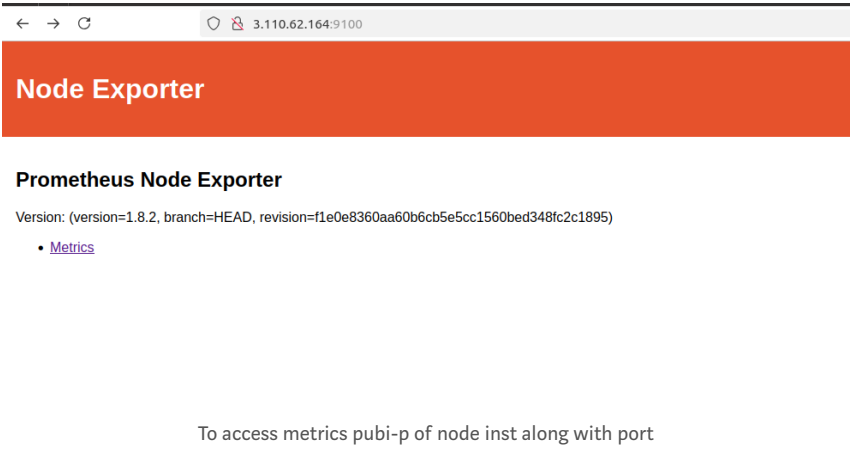


Argocd

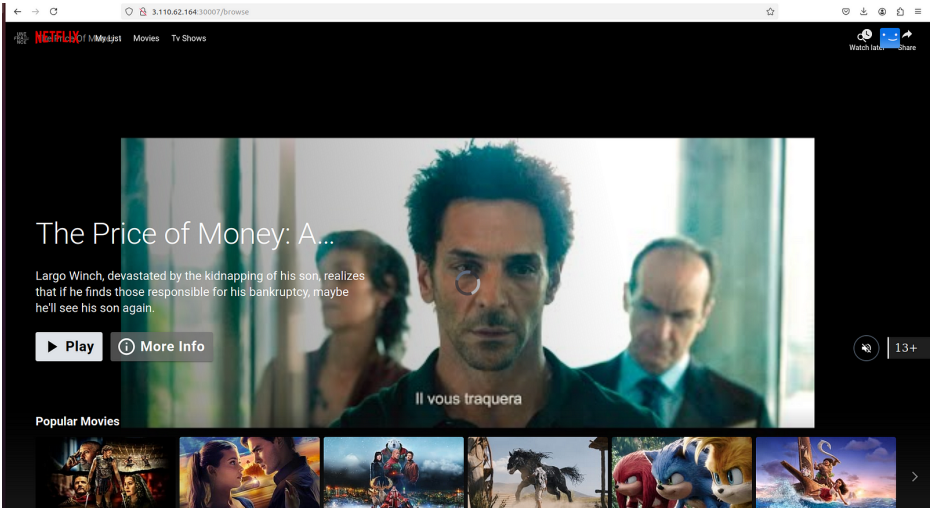
```
krupakar@krupakar:~/workspace/kubernetes$ kubectl get svc argocd-server -n argocd
NAME            TYPE           CLUSTER-IP       EXTERNAL-IP                                                               PORT(S)                      AGE
argocd-server   LoadBalancer   10.100.97.136    a1ad5385bb2e9450599327d553193343-882925492.ap-south-1.elb.amazonaws.com   80:30952/TCP,443:32744/TCP   21m
krupakar@krupakar:~/workspace/kubernetes$ argocd login $ARGOCD_SERVER --username admin --password $ARGO_PWD --insecure
'admin:login' logged in successfully
Context 'a1ad5385bb2e9450599327d553193343-882925492.ap-south-1.elb.amazonaws.com' updated
krupakar@krupakar:~/workspace/kubernetes$ echo $ARGOCD_SERVER
a1ad5385bb2e9450599327d553193343-882925492.ap-south-1.elb.amazonaws.com
krupakar@krupakar:~/workspace/kubernetes$ echo $ARGO_PWD
t6gKNIQo9ZG3SrQC
krupakar@krupakar:~/workspace/kubernetes$ kubectl get po
NAME                            READY   STATUS    RESTARTS   AGE
netflix-app-787786768c-h9jsw    1/1     Running   0          59s
netflix-app-787786768c-rqrnp    1/1     Running   0          59s
krupakar@krupakar:~/workspace/kubernetes$ kubectl get nodes
NAME                  STATUS   ROLES    AGE   VERSION
i-01aefabd0e56c8fb6   Ready    <none>   47m   v1.31.1-eks-1b3e656
i-0bfb0dc1127718ef4   Ready    <none>   47m   v1.31.1-eks-1b3e656
krupakar@krupakar:~/workspace/kubernetes$ kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)       AGE
kubernetes   ClusterIP   10.100.0.1       <none>        443/TCP       58m
netflix-app  NodePort    10.100.215.212   <none>        80:30007/TCP  92s
krupakar@krupakar:~/workspace/kubernetes$ kubectl get ns
NAME                   STATUS   AGE
argocd                 Active   28m
default                Active   58m
kube-node-lease        Active   58m
kube-public            Active   58m
kube-system            Active   58m
prometheus-node-exporter   Active   31m
krupakar@krupakar:~/workspace/kubernetes$ kubectl get deployment
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
netflix-app   2/2     2            2           119s
krupakar@krupakar:~/workspace/kubernetes$
```

4. Access the Application

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



To access metrics pubi-p of node inst along with port



Accessing application

Cleanup

Cleanup AWS EC2 Instances:

- Terminate AWS EC2 instances that are no longer needed.