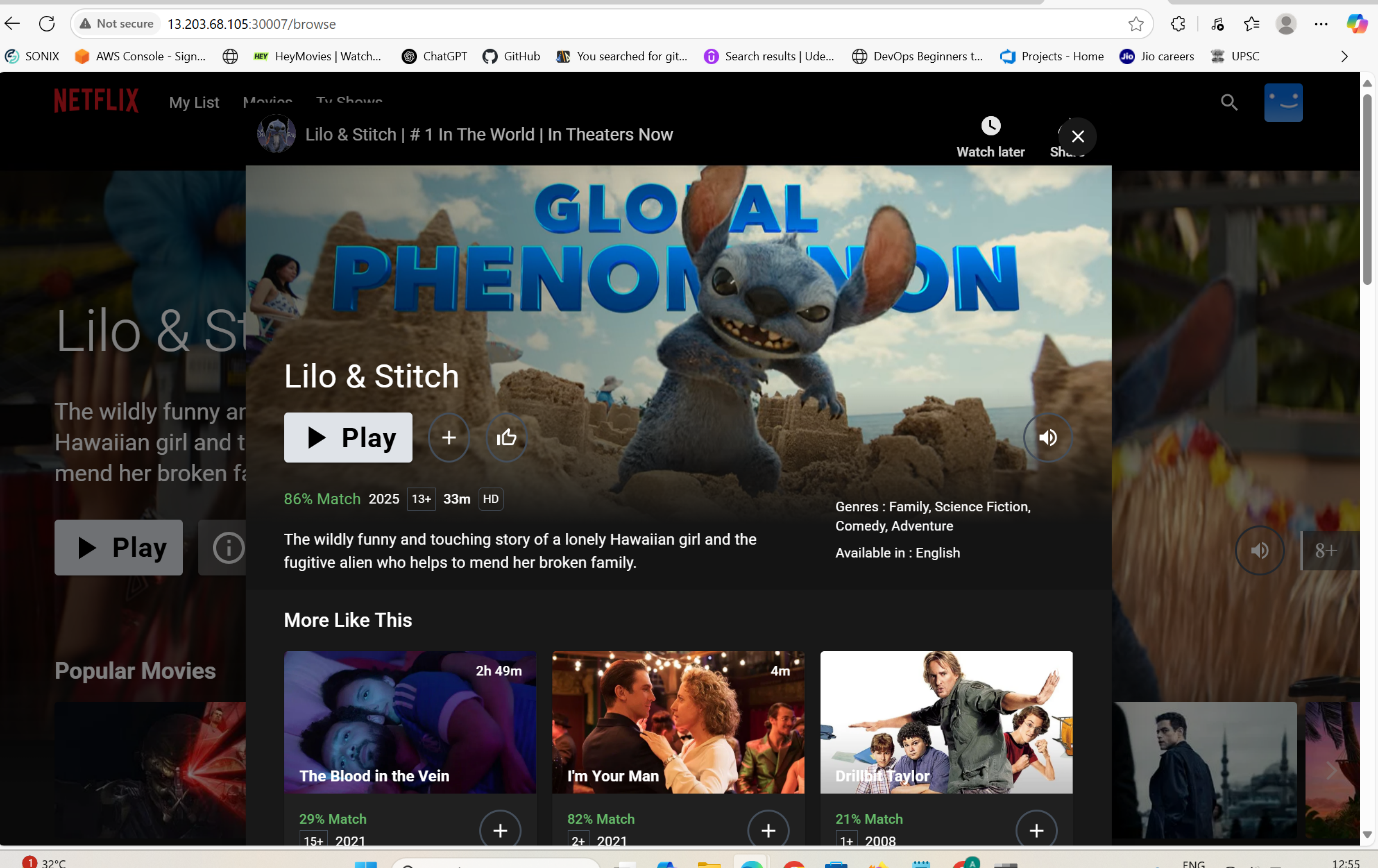
**DevSecOps Project Netflix clone application**

**Deploy Netflix Clone on Cloud using Jenkins:**

****

**Phase 1: Initial Setup and Deployment**

**Step1**: Launch EC2 (Ubuntu 22.04)

* Provision an EC2 instance on AWS with Ubuntu 22.04.
* Choose the instance type as t2.large.
* Create a new key pair or use an existing one.
* Ensure the following checkboxes are selected:
* Allow SSH traffic from the internet
* Allow HTTPS traffic from the internet
* Allow HTTP traffic from the internet
* Configure the instance with 30 GB of storage.
* Connect to the instance using SSH.

Ensure the following security ports added into ec2 instance security group

* 8080 PORTS FOR JENKINS
* 9000 PORTS FOR SONARQUBE
* 8081 PORTS FOR DOCKER IMAGE

A screenshot of a computer

AI-generated content may be incorrect.

**Step 2**: Clone the code:

* Update all the packages and then clone the code.
* Clone your application's code repository onto the EC2 instance:

git clone https://github.com/prudviA/netflix-clone-project.git

* After clone the repository onto the ec2 instance and enter the Devops-project directory:

cd Devops-project

**step 3**: Install docker and run the app using a container:

* Setup docker on EC2 instance:

sudo apt-get update

sudo apt-get install docker.io -y

sudo usermod -aG docker ubuntu

newgrp docker

sudo chmod 777 /var/run/docker.sock

* Build and run your application on ec2 instance using docker containers: verification purpose application is running or not?

docker build -t Netflix .

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

* And with public Ip just copy the Ip paste it on browser verify application is working or not?
* Later, delete docker image and container if it's works successfully.
* Again, go with TDMB (THE MOVIE DATABASE) API

**Step 4**: get the api key:

* 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.

Now recreate the docker image with your api key:

docker build --build-arg TMDB\_V3\_API\_KEY=<YOUR-API-KEY> -t netflix .

docker run -d -p 8081:8 Netflix:latest

After here also delete images and container we are just verify the Docker file with API key.

**Phase 2:** security

1. Install SonarQube and trviy:

Install sonarqube and trivy on the ec2 instance to scan for vulnerabilities.

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

To access:

publici:9000(by default username and password is admin)

**To install 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 trivy

trivy version

trivy image <imageid> # to scan the image we will find vulnerabilities.

2. Integrate SonarQube and configure:

Integrated SonarQube with your CI/CD pipeline.

configure SonarQube to analyse code for quality and security issues.

**Phase 3**: CI/CD SETUP

1. Install Jenkins for automation:

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

Here do one more thing :

sudo su

sudo usermod -aG docker jenkis

sudo systemctl restart Jenkins

Access Jenkins in a web browser using the public Ip of your ec2 instance public ip:8080

2. install necessary plugins in Jenkins:

Go to manage Jenkins --- plugins ----- available plugins---- install below plugins

1.Eclipse temurin installer (install without restart)

2. SonarQube scanner (install without restart)

3. NodeJS plugin (install without restart)

4. email extension plugin

Configure Java and Nodejs in Global Tool Configuration

Goto Manage Jenkins → Tools → Install JDK(17)

Name it as jdk17 enable add installer choose version jdk-17.0.01+12 and same as NodeJS (16) → name it node16 version NodeJS 16.2.0 Click on Apply and Save

**SonarQube:**

Create the token how?

* Open SonarQube go to- administration option--security---click on users----right click on 3lines ----create new token named as a Jenkins and generate token copy it later paste it on sonar-token in Jenkins credentials
* Goto Jenkins Dashboard → Manage Jenkins → Credentials → Add Secret Text. It should look like this, After adding Sonar-token, Click on Apply and Save The Configure System option is used in Jenkins to configure different server.
* Global Tool Configuration is used to configure different tools that we install using Plugin
* We will install a sonar scanner in the tools, configure CI/CD pipeline in Jenkins:
* click on new item named as a Netflix.
* choose pipeline click save it, then go to pipeline section there you have to paste our Jenkins groovy script then save it and click on build now.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

**Jenkins groovy script pipeline:**

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/N4si/DevSecOps-Project.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"

}

}

}

}

Install Dependency-Check and Docker Tools in Jenkins

**Install Dependency-Check Plugin:**

* Go to "Dashboard" in your Jenkins web interface.
* Navigate to "Manage Jenkins" → "Manage Plugins."
* Click on the "Available" tab and search for "OWASP Dependency-Check."
* Check the checkbox for "OWASP Dependency-Check" and click on the "Install without restart" button.

Configure Dependency-Check Tool:

* After installing the Dependency-Check plugin, you need to configure the 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.

Install Docker Tools and Docker Plugins:

* Go to "Dashboard" in your Jenkins web interface.
* Navigate to "Manage Jenkins" → "Manage Plugins."
* Click on the "Available" tab and search for "Docker."
* Check the following Docker-related plugins:

1. Docker
2. Docker Commons
3. Docker Pipeline
4. Docker API
5. docker-build-step

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

**Add DockerHub Credentials:**

* 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.

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

Again Jenkins groovy script pipeline

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/N4si/DevSecOps-Project.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 nasi101/netflix:latest "

sh "docker push nasi101/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'

}

}

}

}

#### replace your tmdb api key and docker hub name

**In mean time pipeline is creating we will create Prometheus and node exporter, Grafana on monitor server:**

**Phase 4**: Monitoring

* Launch EC2 (Ubuntu 24.04)
* Provision an EC2 instance on AWS with Ubuntu 24.04
* Choose the instance type as t2.medium
* Create a new key pair or use an existing one.
* Ensure the following checkboxes are selected:
* Allow SSH traffic from the internet
* Allow HTTPS traffic from the internet
* Allow HTTP traffic from the internet
* Configure the instance with 20 GB of storage.
* Connect to the instance using SSH.

1. Instal Prometheus and Grafana:

Set up Prometheus and Grafana to monitor your application.

Installing Prometheus:

First, create a dedicated Linux user for Prometheus and 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

Verify Prometheus's status:

sudo systemctl status Prometheus

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

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

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

sudo useradd --system --no-create-home --shell /bin/false node\_exporter

wget https://github.com/prometheus/node\_exporter/releases/download/v1.6.1/node\_exporter-1.6.1.linux-amd64.tar.gz

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

tar -xvf node\_exporter-1.6.1.linux-amd64.tar.gz

sudo mv node\_exporter-1.6.1.linux-amd64/node\_exporter /usr/local/bin/

rm -rf node\_exporter\*

Create a systemd unit configuration file for Node Exporter:

sudo nano /etc/systemd/system/node\_exporter.service

Add the following 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

[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

You can access Node Exporter metrics in Prometheus.

2.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:

cd ~

cd /etc/Prometheus

sudo nano Prometheus.yml

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.
* Check the validity of the configuration file:

promtool check config /etc/prometheus/prometheus.yml

A computer screen shot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

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
* ####Grafana
* Install Grafana on Ubuntu 22.04 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 software-properties-common

Step 2: Add the GPG Key:

* Add the GPG key for Grafana:

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

Step 3: Add Grafana Repository:

* 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:

* 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 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 8: Change the Default Password:

* When you log in for the first time, Grafana will prompt you 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, 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 10: 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! You've successfully installed and set up Grafana to work with Prometheus for monitoring and visualization.

Configure Prometheus Plugin Integration:

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

Phase 5: Notification

* Implement Notification Services:
* Set up email notifications in Jenkins or other notification mechanisms.

-----------------------------------------------------------------

Phase 6: Kubernetes

**Create Kubernetes Cluster with Nodegroups**

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

**Monitor Kubernetes with Prometheus**

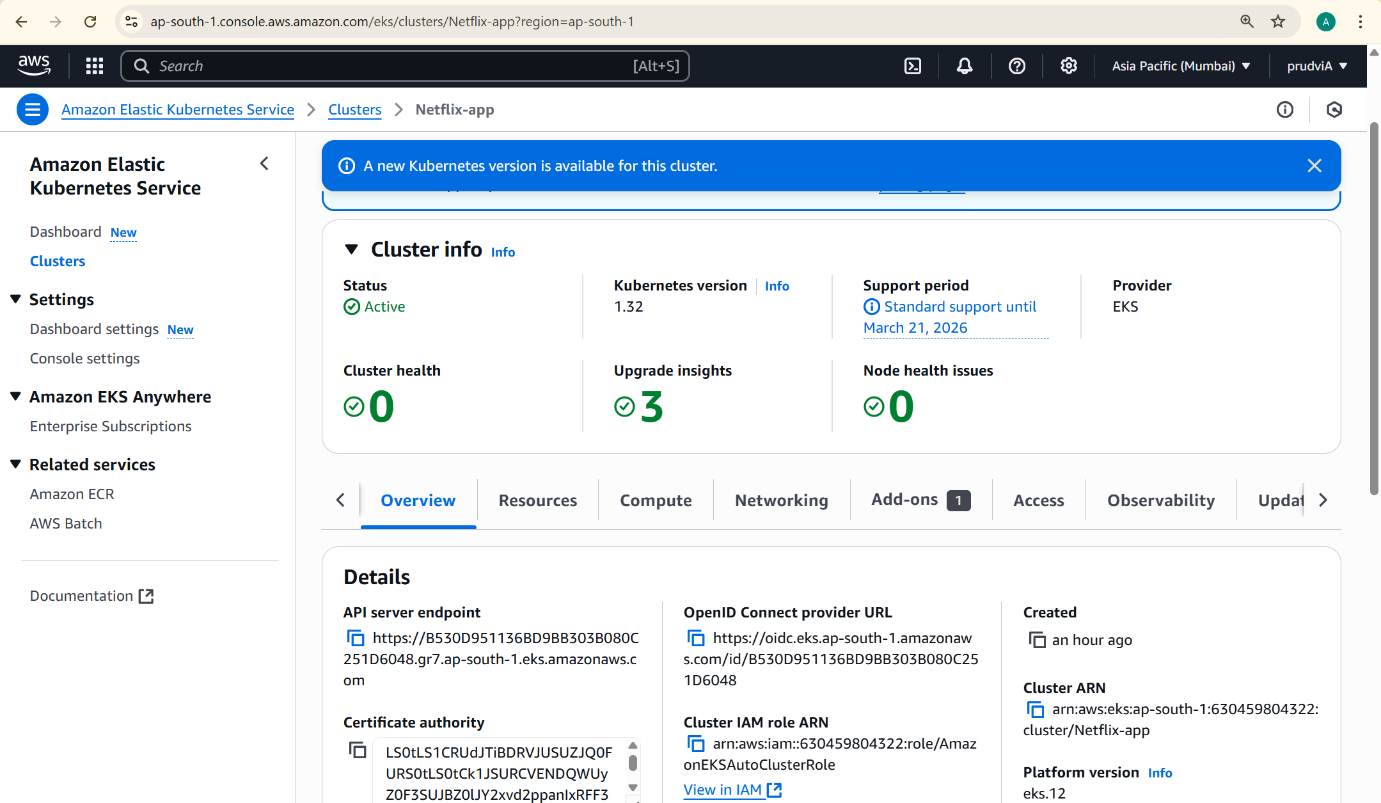
* Prometheus is a powerful monitoring and alerting toolkit, and you'll use it to monitor your Kubernetes cluster. Additionally, you'll install the node exporter using Helm to collect metrics from your cluster nodes.

**Install Node Exporter using Helm**

* To begin monitoring your Kubernetes cluster, you'll install the Prometheus Node Exporter. This component allows you to collect system-level metrics from your cluster nodes. Here are the steps to install the Node Exporter using Helm
* open your ubuntu terminal on local machine did not require any ec2 instance
* so let's open your ubuntu terminal

A screenshot of a computer

AI-generated content may be incorrect.A computer screen shot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

before we have to create eks cluster on aws free account, open eks (elastic Kubernetes service)

* Click on create cluster
* Name: Netflix
* Add cluster role and
* Add node role,
* Remaining leave it as default settings
* Security group with default one
* Add on: Kube proxy, metrics, cli policy.
* Then create cluster
* After created successfully then **create node group** also when both will created successfully then go to the remaining process.

In the meantime in our ubuntu machine install :

aws cli, kubectl, argocd, helm

Installation of aws-cli:

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo apt update

sudo apt install unzip -y

sudo ./aws/install

aws --version

rm -rf awscliv2.zip aws/

aws configure list

verify the cluster is there or not?

aws eks update-kubeconfig --name Netflix --region ap-south-1

kubectl get ns

kubectl get pods

Ensure the following are installed and working:

kubectl

aws-cli (with aws eks update-kubeconfig done)

EKS cluster is running

step-1: Create argocd Namespace

kubectl create namespace argocd

Step 2: Install Argo CD Core Components

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

This installs:

Argo CD API server

Repo server

Controller

Redis

kubectl get ns

kubectl get all -n argocd

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

Step 3: Verify Argo CD Pods

kubectl get pods -n argocd

Step 4: Expose Argo CD API Server

kubectl port-forward svc/argocd-server -n argocd 8080:443

To install Helm (the Kubernetes package manager) on Ubuntu, follow the official and reliable method below.

1. Update your package list

sudo apt update

2. Install prerequisites

sudo apt install apt-transport-https curl gnupg -y

3. Add the Helm GPG key

curl https://baltocdn.com/helm/signing.asc | sudo gpg --dearmor -o /usr/share/keyrings/helm.gpg

4. Add the Helm APT repository

echo "deb [signed-by=/usr/share/keyrings/helm.gpg] https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable.list > /dev/null

5. Update package list again

sudo apt update

6. Install Helm

sudo apt install helm -y

7. Verify Installation

helm version

Add the Prometheus Community Helm repository:

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

Create a Kubernetes namespace for the Node Exporter.

kubectl create namespace prometheus-node-exporter

Install the Node Exporter using Helm:

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

after verifying the prometheus-node-exporter is active or not?

for that:

kubectl get ns

there you find below like this:

pen@Prudhviachuth:~$ kubectl get ns

NAME STATUS AGE

argocd Active 10m

default Active 32m

kube-node-lease Active 32m

kube-public Active 32m

kube-system Active 32m

prometheus-node-exporter Active 2m35s

kubectl get pods -n prometheus-node-exporter

export ARGOCD\_SERVER=$(kubectl get svc argocd-server -n argocd -o json | jq -r '.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

* You will get argocd page opened:
* click on manage your repo
* click on repos
* click on connect using https
* copy your repo
* and paste it url section, project name is default
* click on connect then go to argo cd home page click on add new app application name:

Netflix

project: default

source code: paste your repo url

revision: HEAD

sync policy : automatic

path: Kubernetes

destination: cluster url

namespace: kubectl get ns

default

save and click on sync , synchronize click on force

ok

* When application is on up then you will copy the node public ip before that add your security port in your node the copy the public ip:30007 along with port there you will get Netflix page, 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:

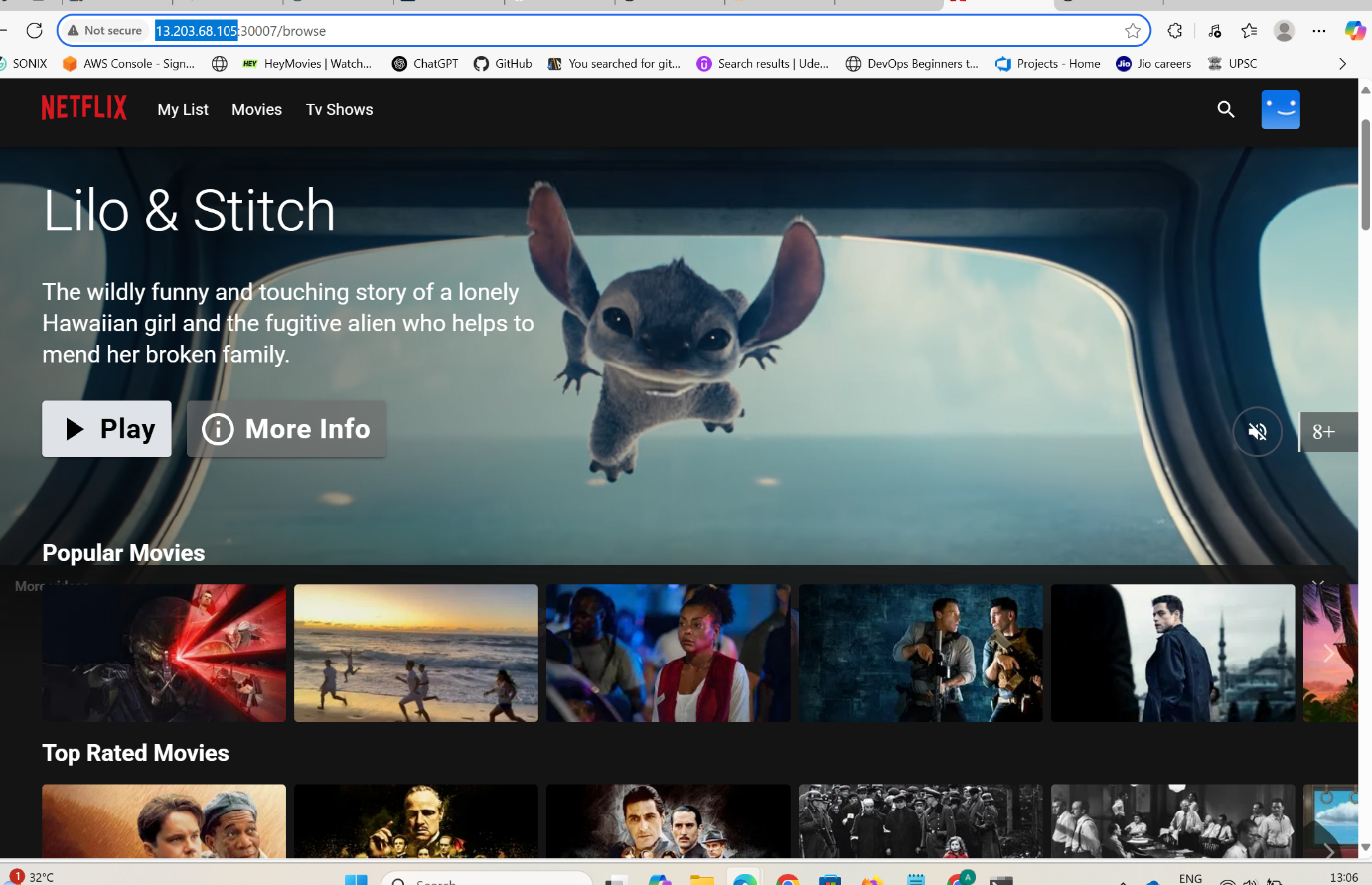
- job\_name: 'Netflix'

metrics\_path: '/metrics'

static\_configs:

- targets: ['node1Ip:9100']

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

* 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.
* To deploy an application with ArgoCD, you can follow these steps, which I'll outline in Markdown format:

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.

**Phase 7: Cleanup:**

Cleanup AWS EC2 Instances:

Terminate AWS EC2 instances that are no longer neede