DEPLOY NETFLIX CLONE ON CLOUD USING JENKINS - DEVSECOPS PROJECT

PHASE 1: INITIAL SETUP AND DEPLOYMENT

Step 1: Launch EC2 (Ubuntu 22.04):

- Provision an EC2 instance on AWS with Ubuntu 22.04.
- Connect to the instance using SSH.

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/N4si/DevSecOps-Project.git

Step 3: Install Docker and Run the App Using a Container:

- Set up Docker on the EC2 instance:
- sudo apt-get update
- sudo apt-get install docker.io -y
- sudo usermod -aG docker $USER # Replace with your system's username, e.g., 'ubuntu'
- newgrp docker

sudo chmod 777 /var/run/docker.sock

- Build and run your application using Docker containers:
- docker build -t netflix .
- docker run -d --name netflix -p 8081:80 netflix:latest
-
- #to delete
- docker stop <containerid>

docker rmi -f netflix

It will show an error cause you need API key

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 .

PHASE 2: SECURITY

1. **Install SonarQube and Trivy:**
    - Install SonarQube and Trivy on the EC2 instance to scan for vulnerabilities.

sonarqube

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

To access:

publicIP:9000 (by default username & 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

to scan image using trivy

trivy image <imageid>

2. **Integrate SonarQube and Configure:**
    - Integrate SonarQube with your CI/CD pipeline.
    - Configure SonarQube to analyze code for quality and security issues.

PHASE 3: CI/CD SETUP

1. **Install Jenkins for Automation:**
    - Install Jenkins on the EC2 instance to automate deployment: Install Java
2. sudo apt update
3. sudo apt install fontconfig openjdk-17-jre
4. java -version
5. openjdk version "17.0.8" 2023-07-18
6. OpenJDK Runtime Environment (build 17.0.8+7-Debian-1deb12u1)
7. OpenJDK 64-Bit Server VM (build 17.0.8+7-Debian-1deb12u1, mixed mode, sharing)
8. #jenkins
9. sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
10. https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
11. echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
12. https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
13. /etc/apt/sources.list.d/jenkins.list > /dev/null
14. sudo apt-get update
15. sudo apt-get install jenkins
16. sudo systemctl start jenkins

sudo systemctl enable jenkins

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

publicIp:8080

17. **Install Necessary Plugins in Jenkins:**

Goto 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) and NodeJs(16)→ Click on Apply and Save

SonarQube

Create the token

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 Plugins

We will install a sonar scanner in the tools.

Create a Jenkins webhook

1. **Configure CI/CD Pipeline in Jenkins:**

- Create a CI/CD pipeline in Jenkins to automate your application deployment.

```
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"
        }
    }
  }
}
```

Certainly, here are the instructions without step numbers:

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:
    - Docker
    - Docker Commons
    - Docker Pipeline
    - Docker API
    - 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.

```
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'
            }
        }
    }
}
```

If you get docker login failed errorr

```
sudo su
sudo usermod -aG docker jenkins
sudo systemctl restart Jenkins
```

PHASE 4: MONITORING

1. **Install 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

Installing Node Exporter:

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 --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

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:

```
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

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.

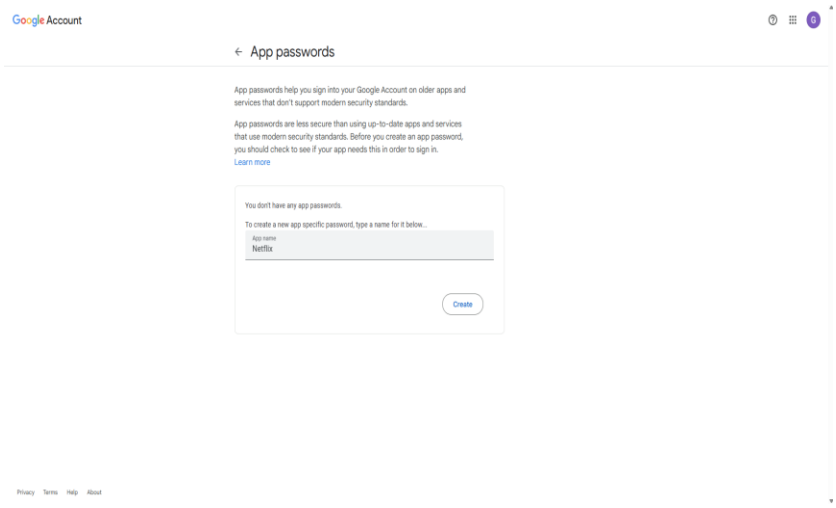
Grafana is a powerful tool for creating visualizations and dashboards, and you can further customize it to suit your specific monitoring needs.

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

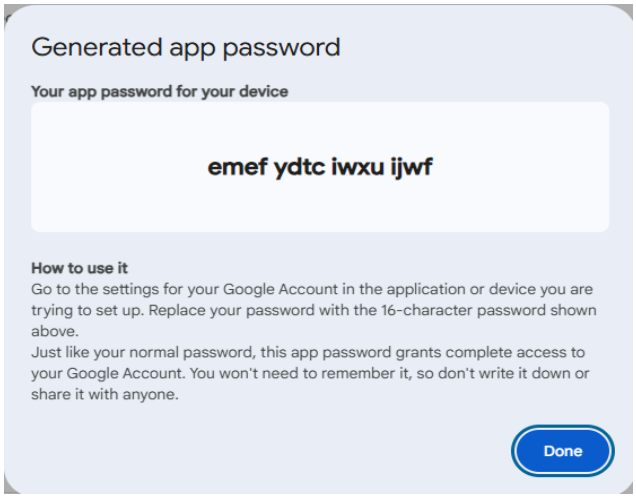
2. **Configure Prometheus Plugin Integration:**
    - Integrate Jenkins with Prometheus to monitor the CI/CD pipeline.

PHASE 5: NOTIFICATION

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



2.
3.



4.

PHASE 6: KUBERNETES

To install Amazon EKS (Elastic Kubernetes Service) tooling in WSL (Windows Subsystem for Linux), you'll mainly need to set up **AWS CLI**, **kubectl**, and **eksctl**. Here's a step-by-step guide tailored for your environment:

✅ **Prerequisites**

- WSL2 installed and running (Ubuntu is recommended)
- AWS account with appropriate IAM permissions
- Docker Desktop with WSL integration (optional but useful for local testing)

🔧 **Step 1: Install AWS CLI**

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

aws --version

✅ **Use -L to follow redirects**

Try this instead:

curl -Ls https://dl.k8s.io/release/stable.txt

The -L flag tells curl to follow the redirect, and -s keeps it quiet except for the output.

If that works, you should see something like:

v1.30.1

Then you can proceed with:

KUBECTL_VERSION=$(curl -Ls https://dl.k8s.io/release/stable.txt)

curl -LO https://dl.k8s.io/release/${KUBECTL_VERSION}/bin/linux/amd64/kubectl

chmod +x kubectl

sudo mv kubectl /usr/local/bin/

kubectl version --client

Let me know what output you get from the first command—if it still misbehaves, we'll try a different mirror or fetch it manually.

🔧 Step 3: Install eksctl

curl --silent --location "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp

sudo mv /tmp/eksctl /usr/local/bin

eksctl version

🔧 Step 4: Configure AWS CLI

aws configure

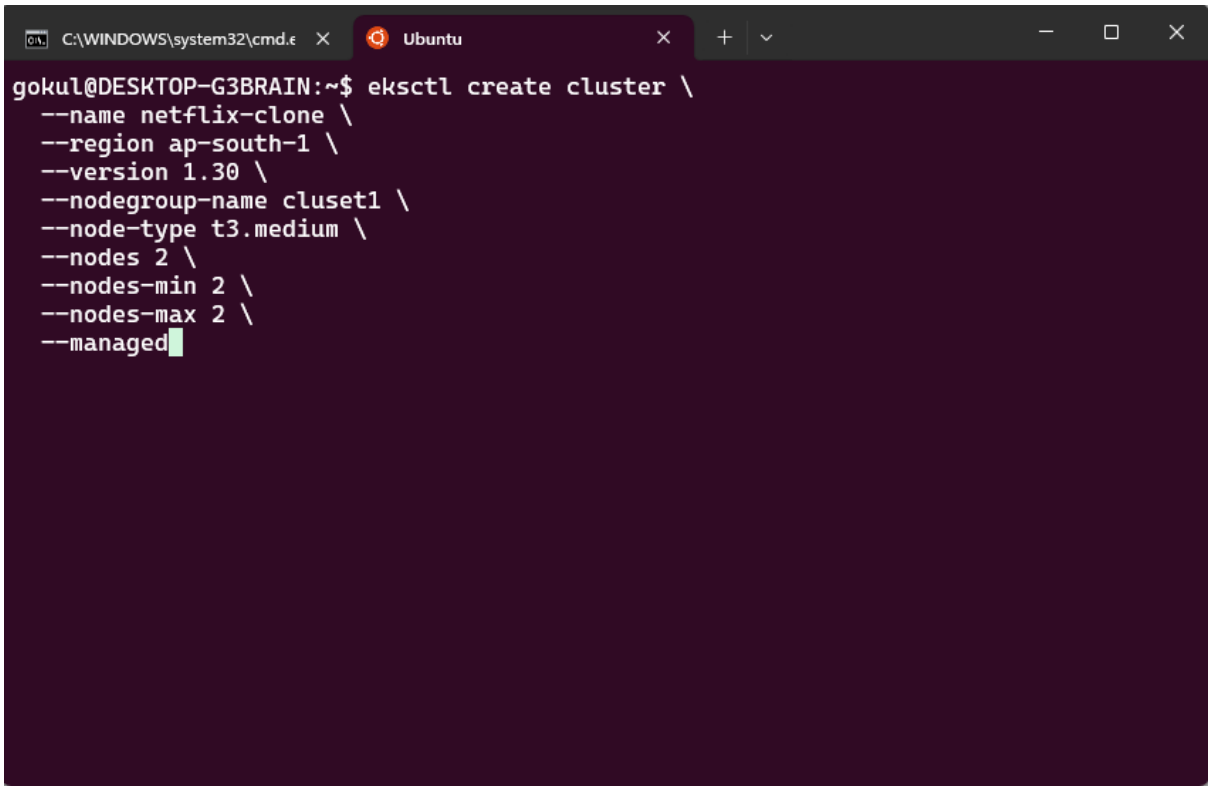
Enter your AWS Access Key, Secret Key, region (e.g., ap-south-1), and output format (json is fine).

🚀 Step 5: Create an EKS Cluster (Optional)

eksctl create cluster --name demo-cluster --region ap-south-1 --nodes 2

This will spin up a basic EKS cluster. You can tweak the config or use a YAML file for more control.

Let me know if you want help with setting up IAM roles, using eksctl with a config file, or integrating this with Jenkins or SonarQube—your specialties!



These are the commands to create cluster in AWS CLI:

```
eksctl create cluster \
  --name netflix-clone \
  --region ap-south-1 \
  --version 1.30 \
  --nodegroup-name cluset1 \
  --node-type t3.medium \
  --nodes 2 \
  --nodes-min 2 \
  --nodes-max 2 \
  --managed
```

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:

1. Add the Prometheus Community Helm repository:

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

2. Create a Kubernetes namespace for the Node Exporter:

kubectl create namespace prometheus-node-exporter

3. Install the Node Exporter using Helm:

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

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

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

```
 - job_name: 'Netflix'

   metrics_path: '/metrics'

   static_configs:

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

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

Don't forget to reload or restart Prometheus to apply these changes to your configuration.

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

Deploy Application with ArgoCD

1. **Install ArgoCD:**

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

2. **Set Your GitHub Repository as a Source:**

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

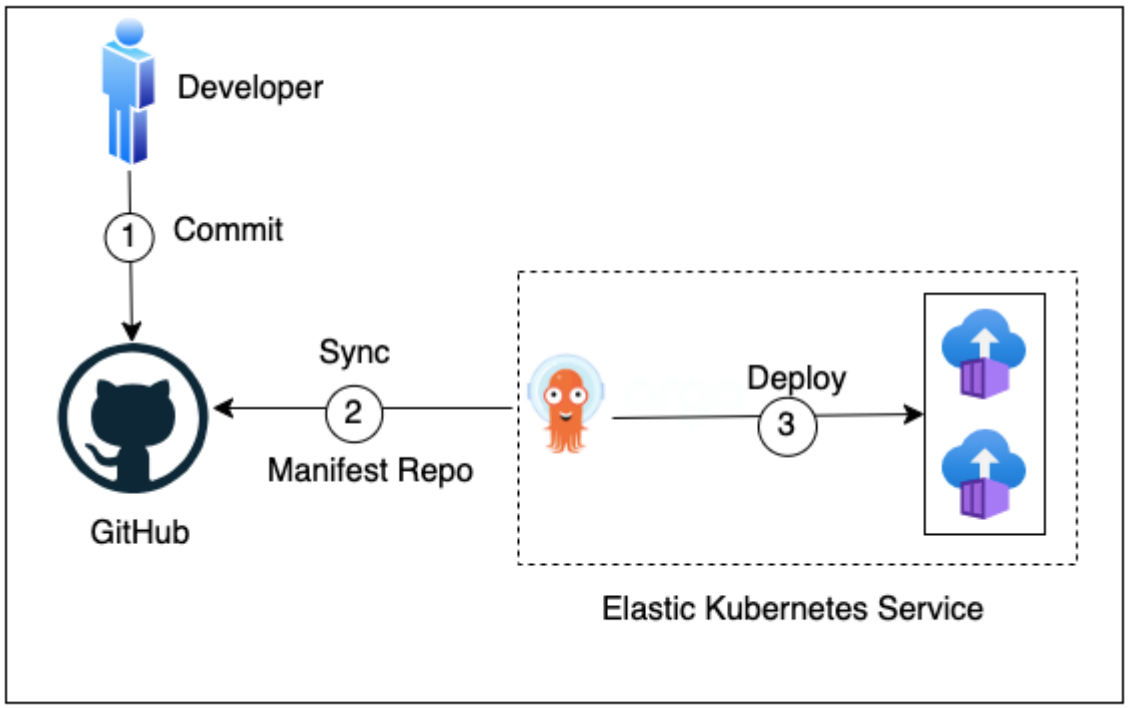
3. **Create an ArgoCD Application:**
    - name: Set the name for your application.
    - destination: Define the destination where your application should be deployed.
    - project: Specify the project the application belongs to.
    - source: Set the source of your application, including the GitHub repository URL, revision, and the path to the application within the repository.
    - syncPolicy: Configure the sync policy, including automatic syncing, pruning, and self-healing.
4. **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.

Phase 7: Cleanup

1. **Cleanup AWS EC2 Instances:**
    - Terminate AWS EC2 instances that are no longer needed.

No releases published

INSTALL ARGOCD ON THE EKS CLUSTER AND CONFIGURE SYNC WITH THE GITHUB MANIFEST REPOSITORY.



This image shows how ArgoCD syncs with GitHub to deploy code.

📝 Introduction

Hello everyone! Introducing myself as Himani Chauhan, I am a passionate DevOps and Cloud enthusiast with a relentless drive to explore the latest advancements in cloud technology and their seamless integrations. Join me as I explore the captivating world of DevOps and the Cloud through my blog posts.

✅ Prerequisites

- Amazon EKS Cluster
- AWS CLI
- Blog 3 *How to set pipeline using CircleCI, update GitHub Kubernetes manifest repo, and push image on Docker Hub*

✍ Note

1. Blog 1: *Deploying Dockerized App on AWS EKS Cluster using ArgoCD and GitOps methodology with CircleCI (Introduction to EKS and GitOps)*.
2. Blog 2: *How to provision Amazon EKS Cluster using Terraform.*
3. Blog 3: *How to set pipeline using CircleCI, update GitHub Kubernetes manifest repo, and push image on Docker Hub*.

4. Blog 4: *Install ArgoCD on the EKS cluster and configure sync with the GitHub manifest repository.* Currently, you are looking at blog 4.

✍ Narrative

This blog post will explore the synchronization process between the Kubernetes manifest repository and ArgoCD. By setting up this synchronization, ArgoCD can automatically detect changes in the Kubernetes manifest file. Upon detection, it pulls the updated files from the repository and applies them to the cluster, ensuring that the desired state aligns with the actual state of the cluster.

Let's see how we can accomplish this!

⏩ Step-by-Step guide

Step 1: Log in to CloudShell or your terminal window

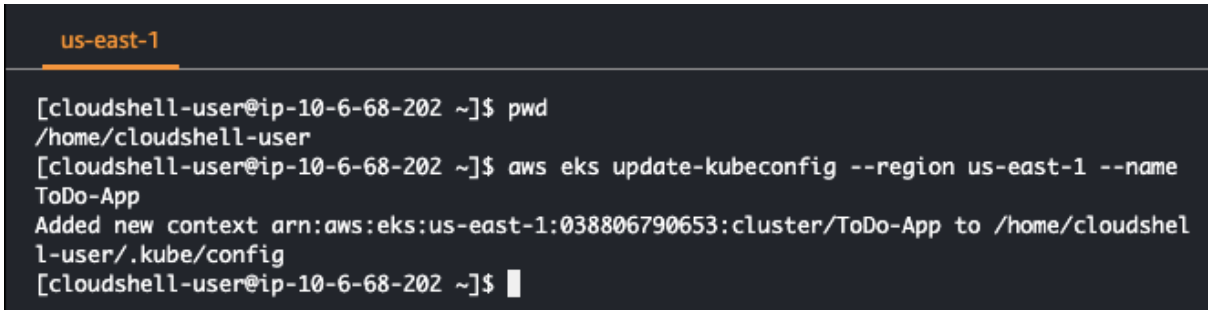
Open your AWS console and look for the CloudShell icon at the top right bar next to the notification bell icon, as shown below:

This image shows how to open the cloud shell from the AWS console

Step 2: Create kubeconfig

Create or update kubeconfig using the below command:

aws eks update-kubeconfig --region $REGION --name $CLUSTER_NAME



This image shows how to update kubeconfig with your created cluster name and chosen region

Step 3: Install ArgoCD

Download the latest Argo CD version from https://github.com/argoproj/argo-cd/releases/latest.

If you face any issues with the installation, try using the curl command below:

curl -sSL -o /usr/local/bin/argocd https://github.com/argoproj/argo-cd/releases/latest/download/argocd-linux-amd64
chmod +x /usr/local/bin/argocd

More detailed installation instructions can be found via the CLI installation documentation.

The namespace will be created as part of the installation in the above step. If you didn't, please refer to the below command to create a namespace:

kubectl create namespace argocd

```
[cloudshell-user@ip-10-6-2-129 ~]$ kubectl create namespace argocd
namespace/argocd created
[cloudshell-user@ip-10-6-2-129 ~]$
```

This image shows how to create a namespace

Step 4: Create ArgoCD Objects

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

Ensure that pods are created by ArgoCD using the below command:

kubectl get pods -n argocd

```
[cloudshell-user@ip-10-6-2-129 ~]$ kubectl get pods -n argocd
NAME                                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0                     1/1     Running   0          44s
argocd-applicationset-controller-64c9969bf9-d47kn   1/1     Running   0          44s
argocd-redis-b5d6bf5f5-m26xg                        1/1     Running   0          44s
argocd-repo-server-956bbd468-4rtnl                  1/1     Running   0          44s
```

This image shows the status of pods running in the ArgoCD namespace

Step 5: Publicly accessible ArgoCD-Server

By default, Argocd-server is not publicly accessible, but we can achieve external accessibility by utilizing a load balancer. While alternative methods like port forwarding can also provide access, this blog will focus on using a load balancer.

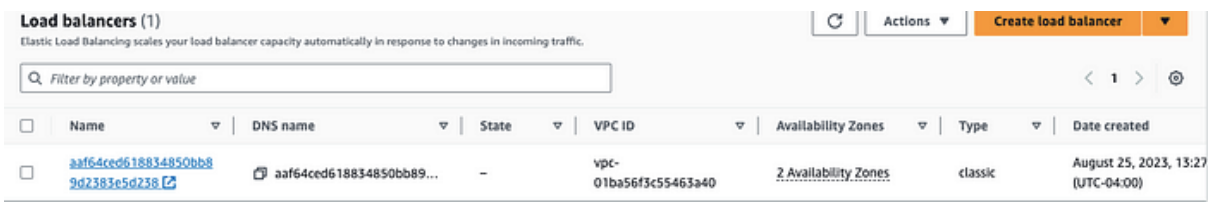
kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'

```
[cloudshell-user@ip-10-6-2-129 ~]$ kubectl patch svc argocd-server -n argocd -p '{"spec"
: {"type": "LoadBalancer"}}'
service/argocd-server patched
[cloudshell-user@ip-10-6-2-129 ~]$
```

This image shows how to make the ArgoCD server publically accessible using a Load Balancer

LoadBalancer will take a couple of minutes to be up and running. You can check your LoadBalancer status in the AWS console as below:

Login to AWS Console -> Search for '*EC2*' in the search bar -> Click on '*LoadBalancers*' in the left panel.



This image shows the created LoadBalancer in the AWS console

Use the command below to store your DNS name in a variable ARGOCD_SERVER.

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

Step 6: Login to ArgoCD from CLI

We will address the requirements for accessing a system, which include a username and password. However, it is essential to note that the initial password is automatically generated. By default, the username is set to '*admin*'.

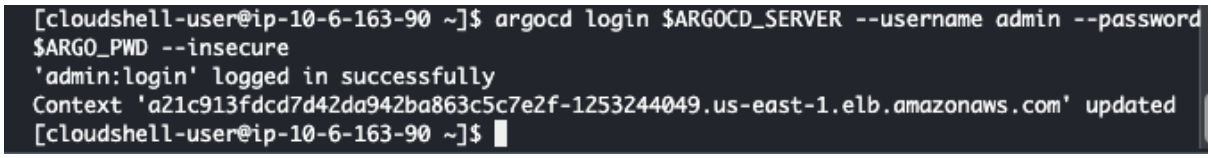
Take load balancer DNS from the AWS console or the terminal by running the command "echo $ARGOCD_SERVER" and paste it into the browser.

Store password in variable 'ARGO_PWD' by using the below command:

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

Now, log into ArgoCD using the below command:

argocd login $ARGOCD_SERVER --username admin --password $ARGO_PWD --insecure



This image shows how to log in to ArgoCD using the CLI

Step 7: Sync the Github Manifest repo with ArgoCD

In this section, I will guide you through setting up synchronization between Argocd and the GitHub manifest repository. To initiate the process, kindly copy the declarative file shared below and save it to your local computer with any preferred name.

Please change the repo URL *in the below file with your corresponding repo at your GitHub.*

```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: todo-app-argo
  namespace: argocd
spec:
  project: default

  source:
    repoURL: https://github.com/chauhan-himani/kube_manifest
    targetRevision: HEAD
    path: manifest
  destination:
    server: https://kubernetes.default.svc
    namespace: myapp  syncPolicy:
    syncOptions:
    - CreateNamespace=true    automated:
```

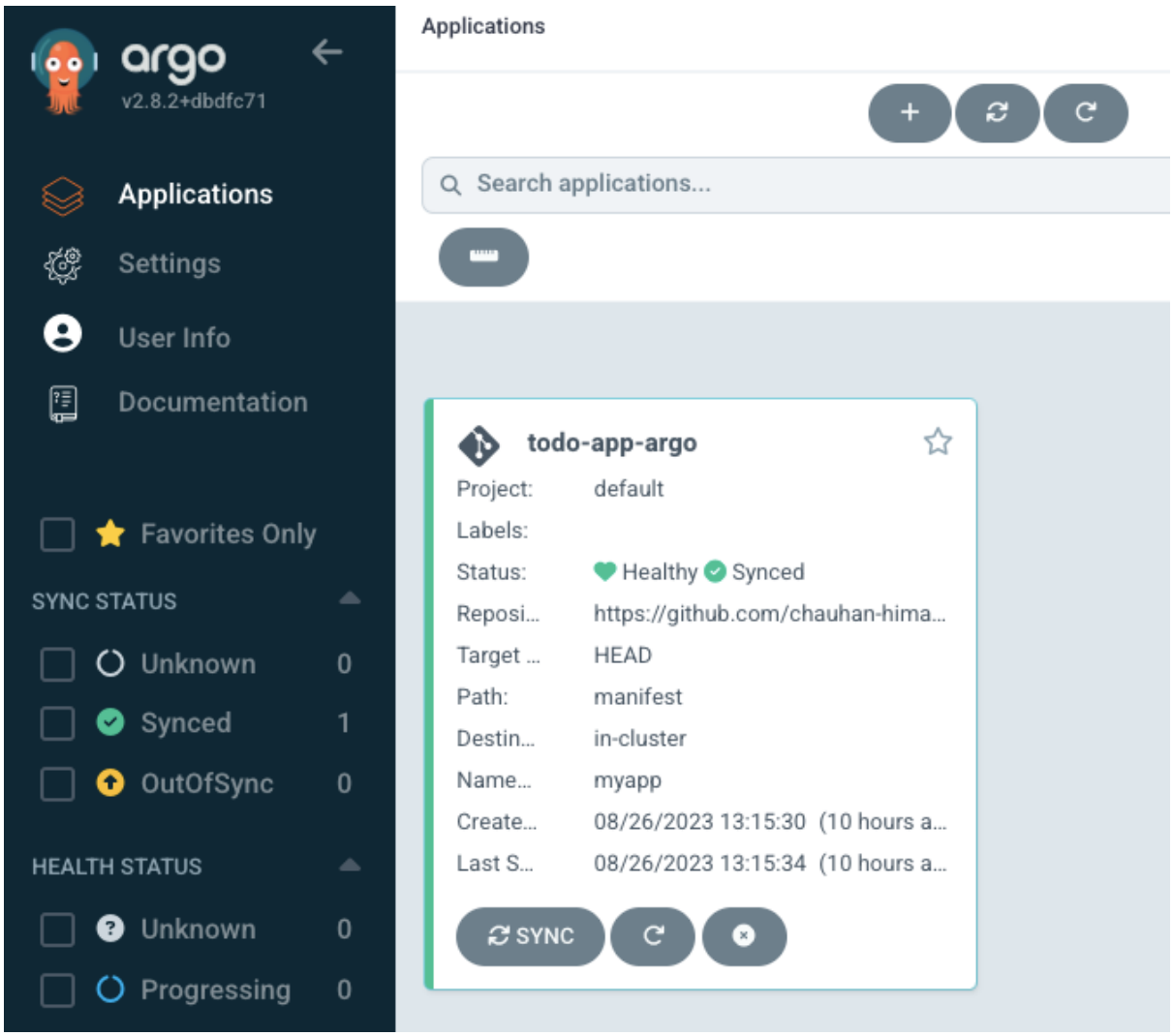
selfHeal: true
prune: true

Let’s apply the above file to start the sync process of the Manifest repo with ArgoCD using the below command:

kubectl apply -f manifest-argo-sync.yaml

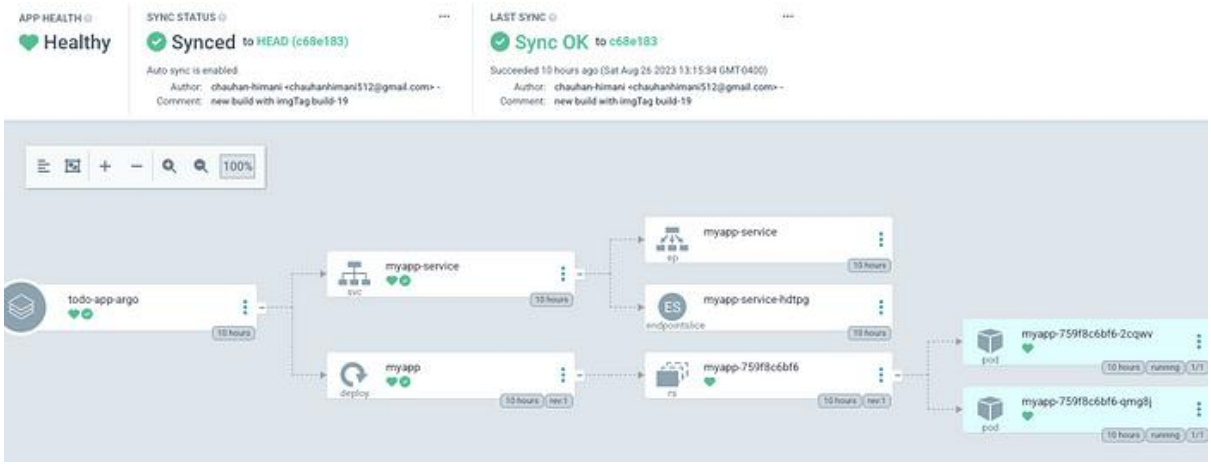
Step 8: ArgoCD UI after successful deployment

Let's look at what has been done in the background to understand the next step better.



This image shows the ArgoCD UI

ArgoCD has applied the manifest file. We have configured this file to be created on ALB to access the dockerized application.

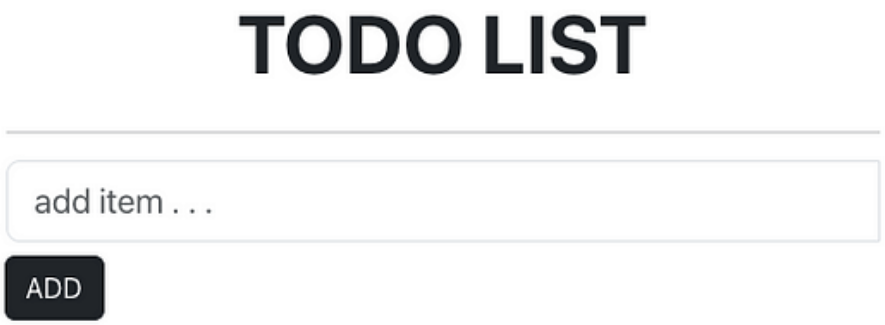


This image shows the flow of our deployment in the ArgoCD

Step 9: Test

We have completed all three parts of this blog, and it is time for testing 😳 .

Open the LoadBalancer DNS link extracted from the AWS console pointing to our application inside the cluster ToDo-App.



This image shows the web interface for our application

To test our application, we will utilize the GitOps workflow. Let's modify and commit the application code to the associated code repository.

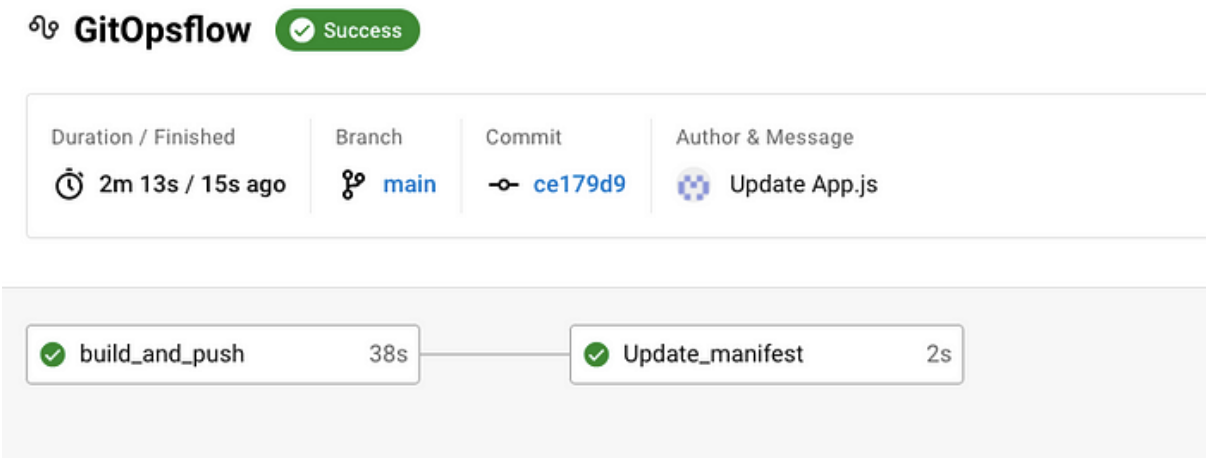
This test will help us verify the functionality of our pipeline. We will observe whether ArgoCD can pull the updated image and successfully apply it to the cluster.

I have updated *the App.js file under the src/ code of my Application code and added my name* to showcase the change i.e., *TODO LIST* to *Himani -> TODO LIST*

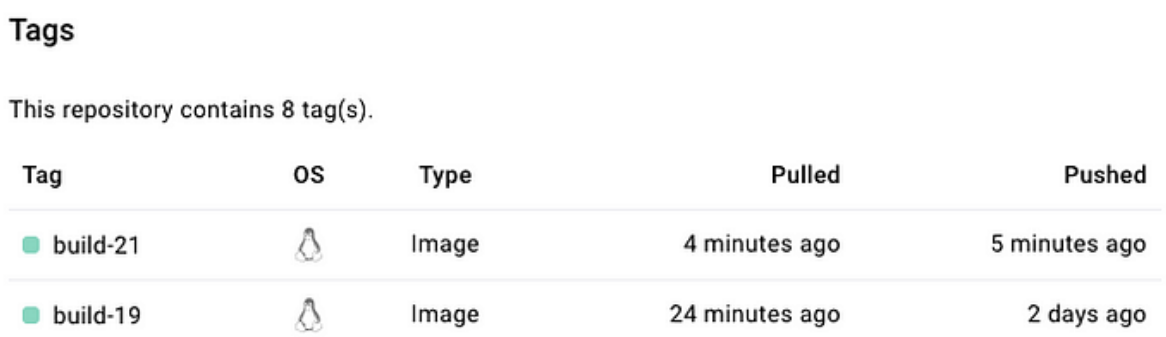
```
85                      alignItems: "center",
86                      fontSize: "3rem",
87                      fontWeight: "bolder",
88                  }}
89              >
90                  Himani - TODO LIST
91              </Row>
92
93              <hr />
94              <Row>
95                  <Col md={{ span: 5, offset: 4 }}>
96                      <InputGroup className="mb-3">
97                          <FormControl
98                              placeholder="add item . . . "
99                              size="lg"
```

This image shows the change that has been made to the src code

As soon as you commit these changes, you will observe the pipeline gets triggered in CircleCI, the new image build will be pushed to Docker Hub, and the manifest file will be updated with the latest build tag.

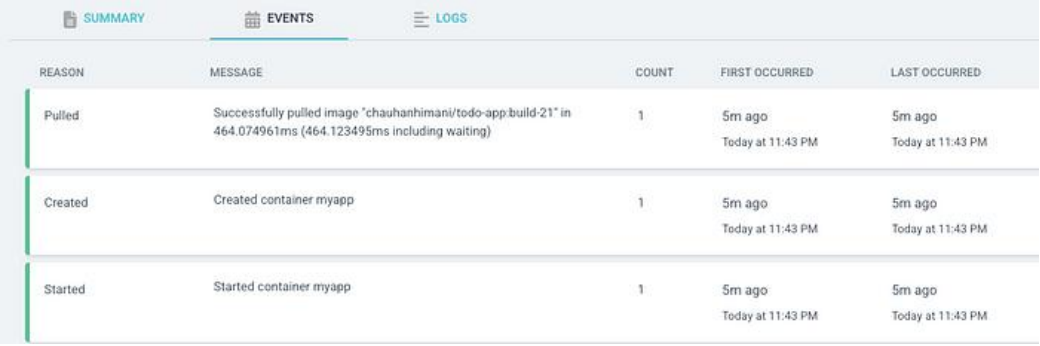


This image shows jobs triggered in CircleCI after the code change



This image shows that the latest Image has been pushed to DockerHub

The image has been pulled from DockerHub. ArgoCD updated the manifest file with a new build tag, and changes have been implemented successfully. ArgoCD takes a couple of minutes to apply changes,



This image shows that the latest image has been pulled from the docker hub successfully

Once the above deployment is successful, please paste the LoadBalancer DNS URL into the browser and verify if changes are visible.

This image shows the web interface after the code change has been applied

Conclusion

This blog has detailed steps to deploy a dockerized application on an AWS EKS cluster utilizing ArgoCD and GitOps methodology using CircleCI. We began using Terraform to set up and provision an Amazon EKS cluster for our deployment. We then configured our pipeline with GitHub and CircleCI, enabling a smooth integration between the Kubernetes manifests and our project's CI/CD workflow. This approach guarantees a streamlined and efficient deployment process for our dockerized application on the AWS EKS cluster.

Resources

My GitHub profile: https://github.com/chauhan-himani

GitHub Manifest Repository: https://github.com/chauhan-himani/kube_manifest

ArgoCD Repository: https://github.com/argoproj/argo-cd/releases/latest

ArgoCD CLI installation: https://argo-cd.readthedocs.io/en/stable/cli_installation/

CircleCI https://circleci.com/docs/create-project/

Blog https://www.showwcase.com/show/35858/install-argocd-on-the-eks-cluster-and-configure-sync-with-github-manifest-repository

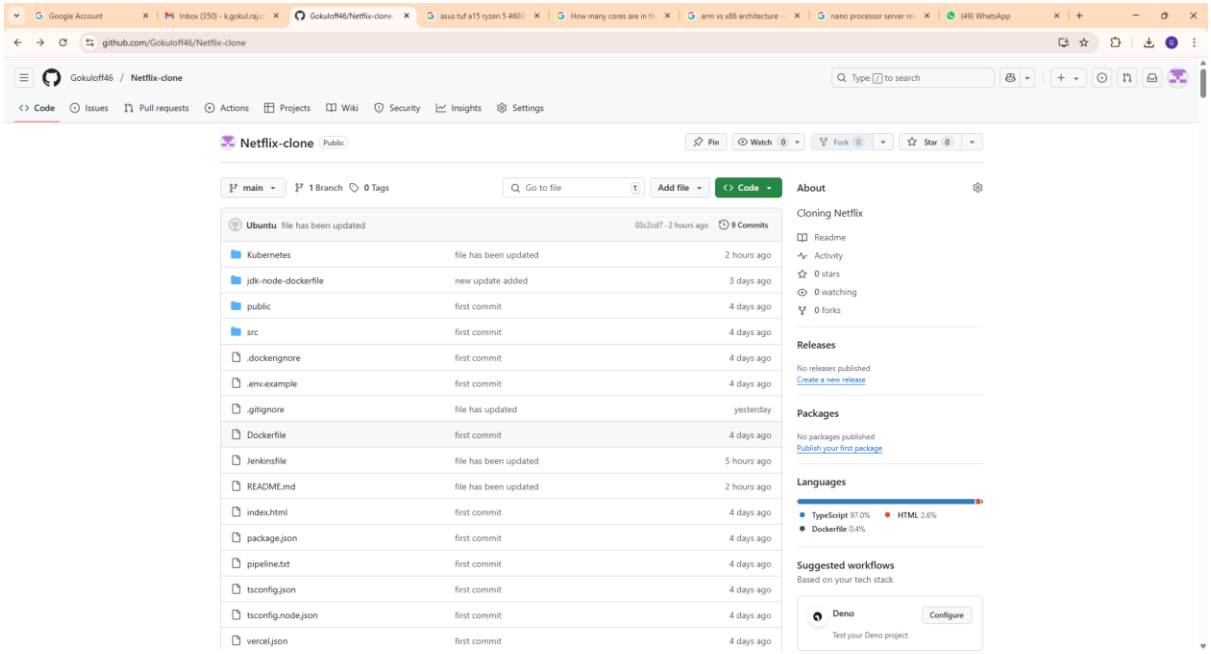
References

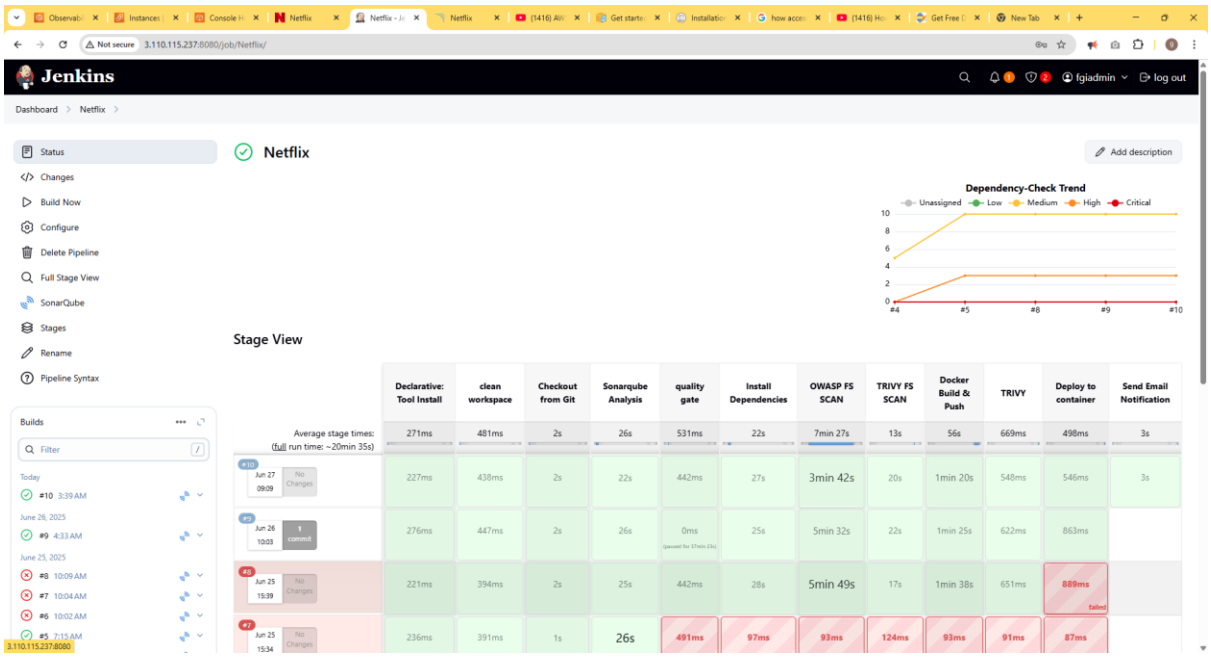
Linkedln https://www.linkedin.com/in/himani-chauhan/

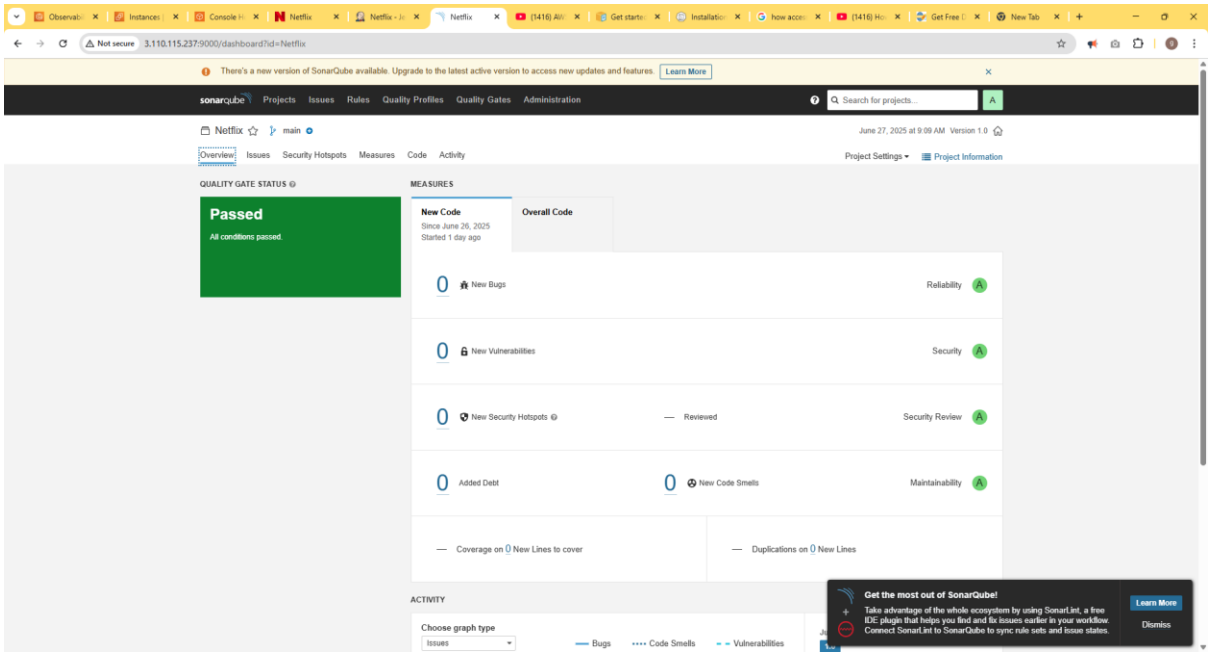
Github profile: https://github.com/chauhan-himani/

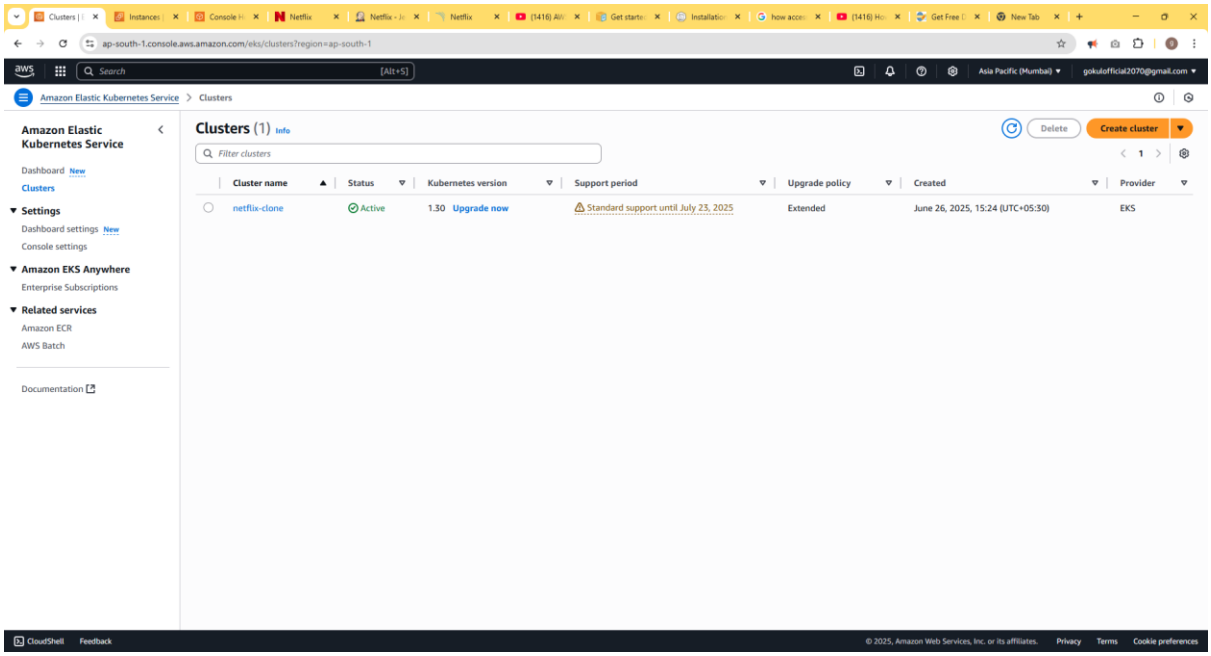
🙏 Thank you for reading this blog.

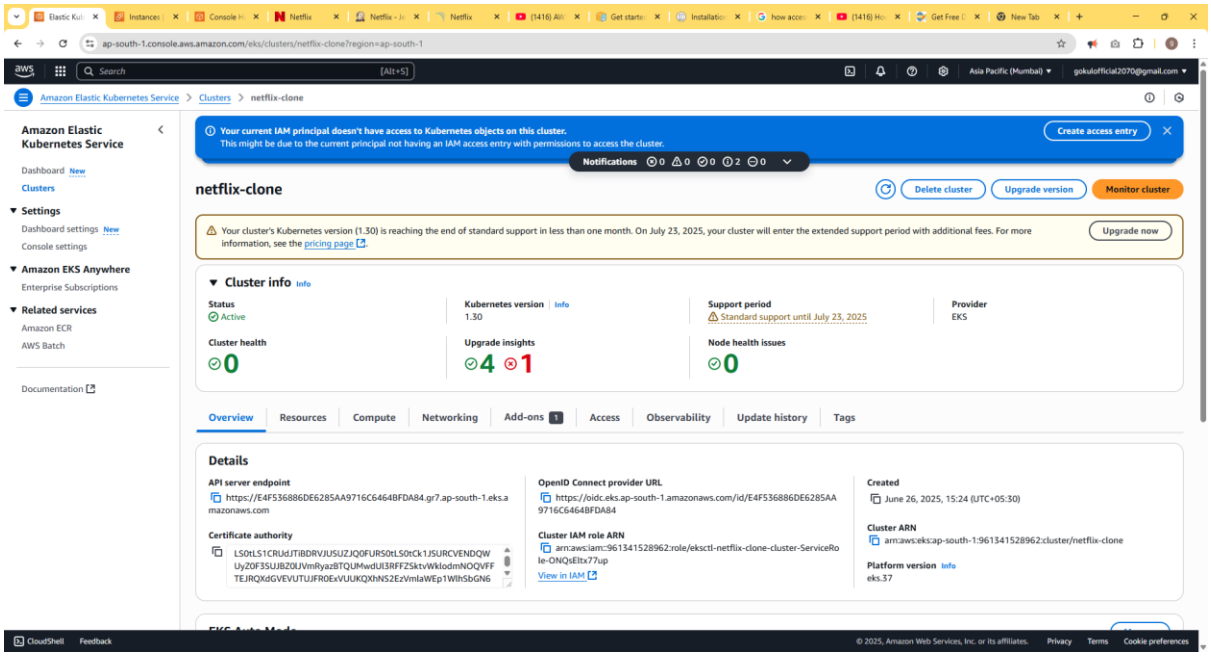
OVERALL SCREEN SHOOTS

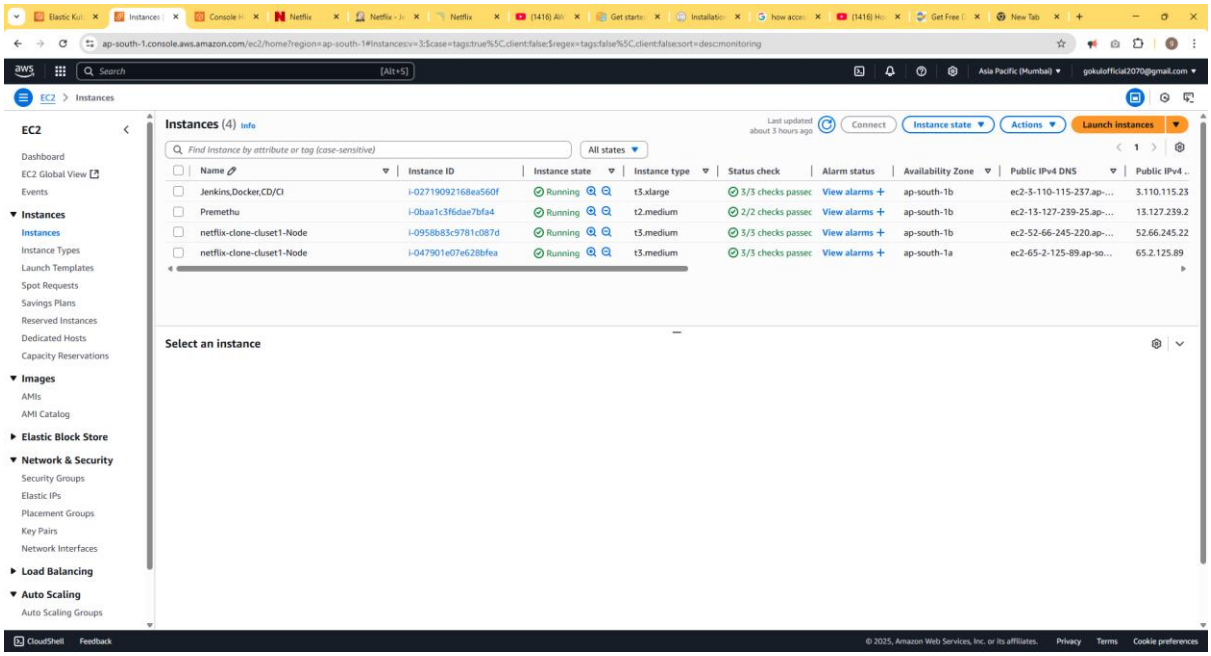


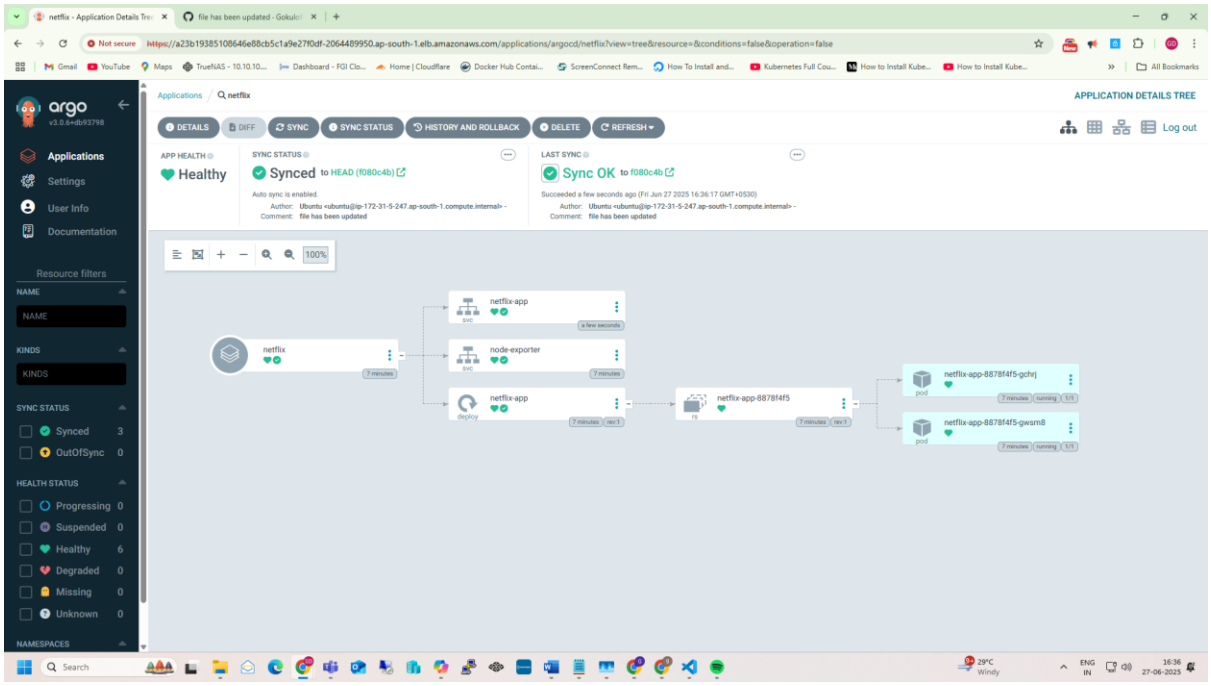


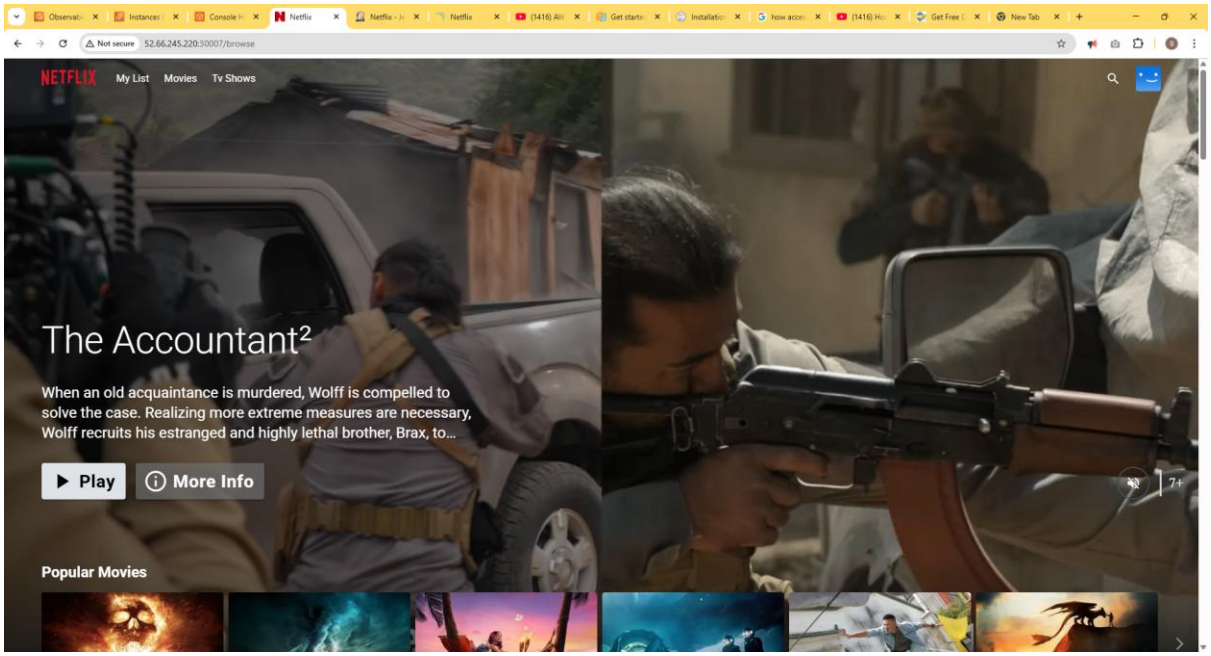

Not secure 3.110.115.237:9000/dashboard?id=Netflix
There's a new version of SonarQube available. Upgrade to the latest active version to access new updates and features. Learn More
sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration
Search for projects...
Netflix main
June 27, 2025 at 9:09 AM Version 1.0
Overview Issues Security Hotspots Measures Code Activity
Project Settings Project Information
QUALITY GATE STATUS
Passed
All conditions passed.
MEASURES
New Code
Since June 26, 2025
Started 1 day ago
Overall Code
0 New Bugs
Reliability A
0 New Vulnerabilities
Security A
0 New Security Hotspots
Reviewed
Security Review A
0 Added Debt
0 New Code Smells
Maintainability A
Coverage on 0 New Lines to cover
Duplications on 0 New Lines
ACTIVITY
Choose graph type
Issues
Bugs
Code Smells
Vulnerabilities
Get the most out of SonarQube!
Take advantage of the whole ecosystem by using SonarLint, a free IDE plugin that helps you find and fix issues earlier in your workflow. Connect SonarLint to SonarQube to sync rule sets and issue states.
Learn More
Dismiss

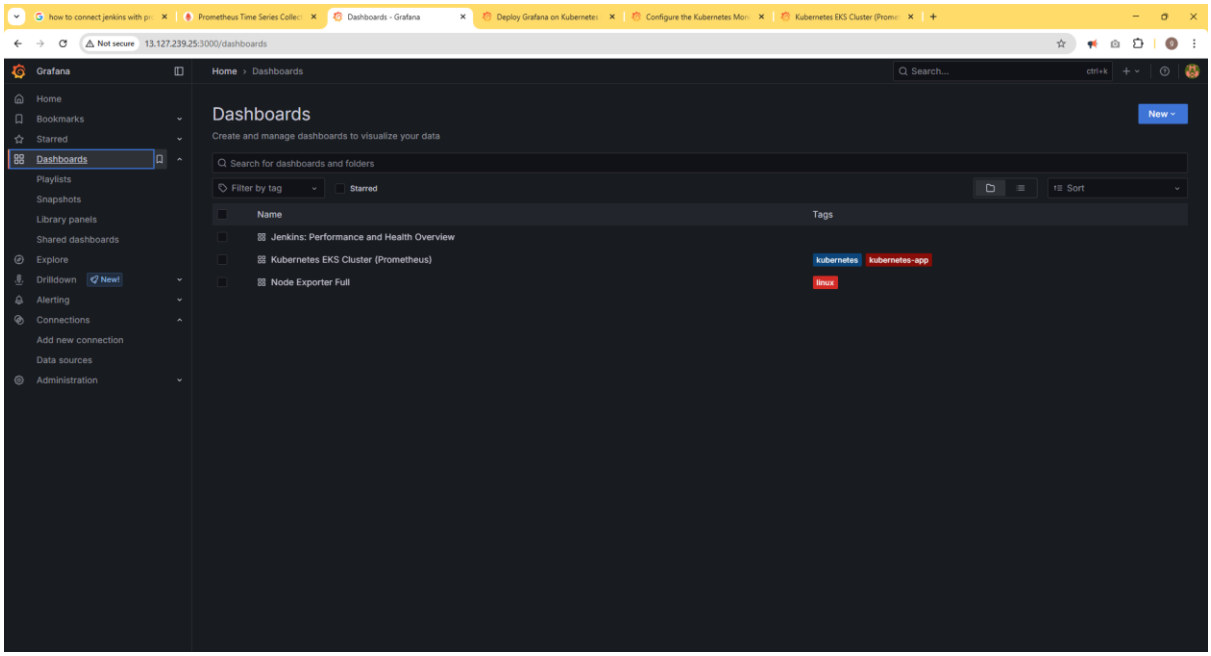

ap-south-1.console.aws.amazon.com/eks/clusters?region=ap-south-1
Search [Alt+S]
Asia Pacific (Mumbai)
gokulofficial2070@gmail.com
Amazon Elastic Kubernetes Service > Clusters
Amazon Elastic Kubernetes Service
Dashboard New
Clusters
Settings
Dashboard settings New
Console settings
Amazon EKS Anywhere
Enterprise Subscriptions
Related services
Amazon ECR
AWS Batch
Documentation
Clusters (1) Info
Filter clusters
Delete
Create cluster
Cluster name
Status
Kubernetes version
Support period
Upgrade policy
Created
Provider
netflix-clone
Active
1.30 Upgrade now
Standard support until July 23, 2025
Extended
June 26, 2025, 15:24 (UTC+05:30)
EKS
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

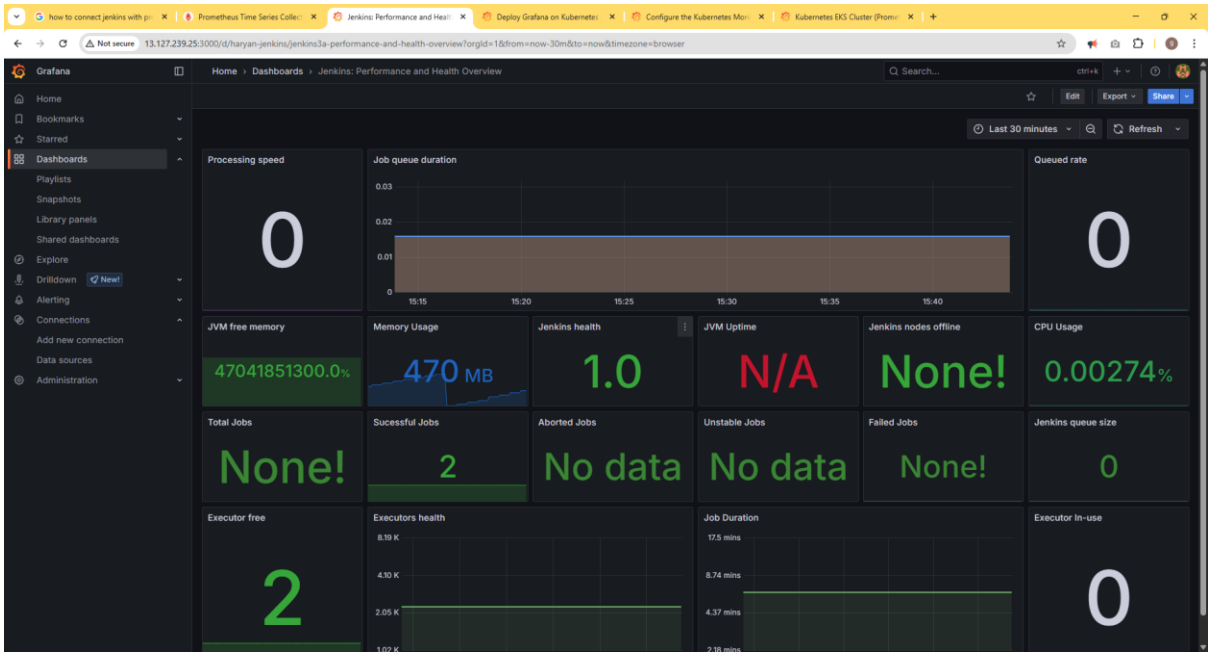

Amazon Elastic Kubernetes Service
Clusters
netflix-clone
Your current IAM principal doesn't have access to Kubernetes objects on this cluster.
This might be due to the current principal not having an IAM access entry with permissions to access the cluster.
Create access entry
Delete cluster
Upgrade version
Monitor cluster
Your cluster's Kubernetes version (1.30) is reaching the end of standard support in less than one month. On July 23, 2025, your cluster will enter the extended support period with additional fees. For more information, see the pricing page.
Upgrade now
Cluster info
Status
Active
Kubernetes version
1.30
Support period
Standard support until July 23, 2025
Provider
EKS
Cluster health
0
Upgrade insights
4 1
Node health issues
0
Overview
Resources
Compute
Networking
Add-ons
Access
Observability
Update history
Tags
Details
API server endpoint
https://E4F536886DE6285AA9716C6464BFDA84.gr7.ap-south-1.eks.amazonaws.com
OpenID Connect provider URL
https://oidc.eks.ap-south-1.amazonaws.com/id/E4F536886DE6285AA9716C6464BFDA84
Created
June 26, 2025, 15:24 (UTC+05:30)
Certificate authority
Cluster IAM role ARN
arn:aws:iam::961341528962:role/eksctl-netflix-clone-cluster-ServiceRole-ONQsEltx77up
View in IAM
Cluster ARN
arn:aws:eks:ap-south-1:961341528962:cluster/netflix-clone
Platform version
eks.37

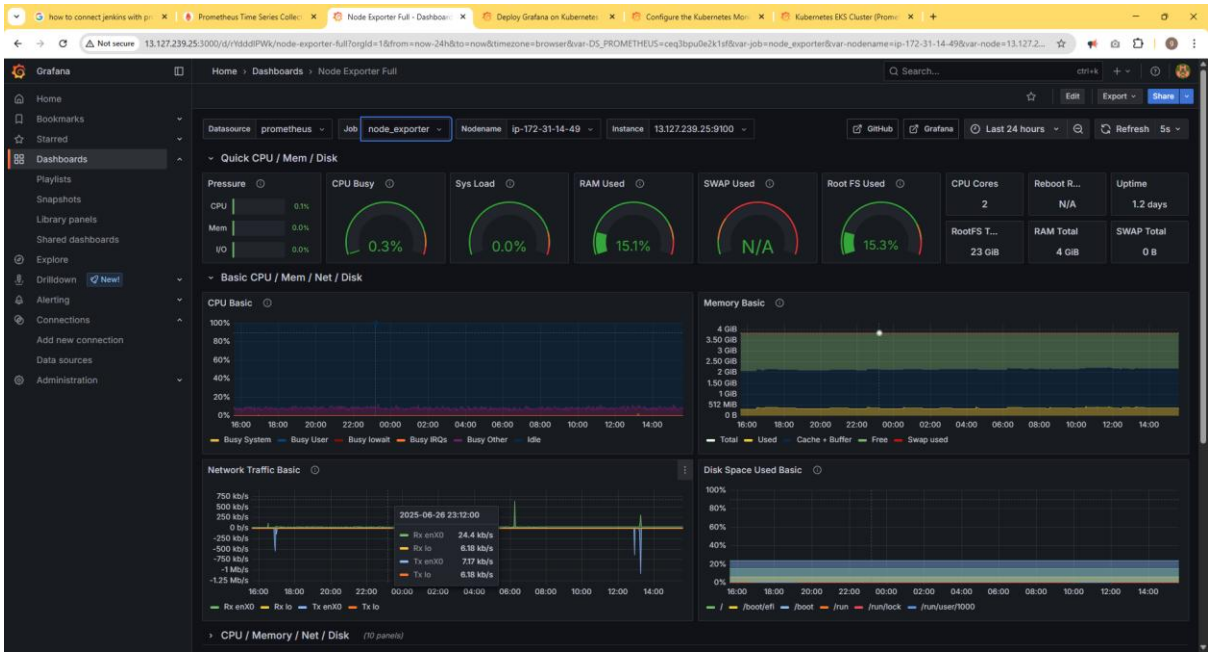

EC2
Instances
Instances (4)
Launch instances
Jenkins,Docker,CD/CI
i-02719092168ea560f
Premethu
i-0baa1c3f6dae7bfa4
netflix-clone-cluset1-Node
i-0958b83c9781c087d
netflix-clone-cluset1-Node
i-047901e07e628bfea
Select an instance

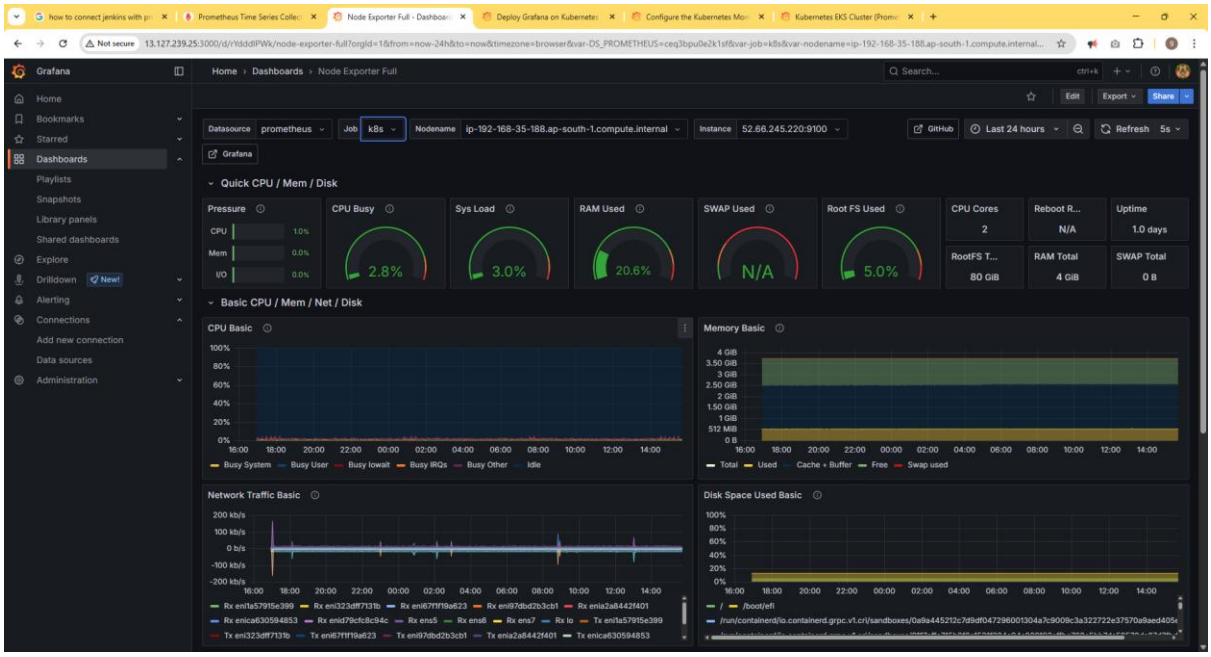

argo
Applications
Settings
User Info
Documentation
Resource filters
Applications
netflix
APPLICATION DETAILS TREE
DETAILS
DIFF
SYNC
SYNC STATUS
HISTORY AND ROLLBACK
DELETE
REFRESH
Log out
APP HEALTH
Healthy
SYNC STATUS
Synced to HEAD (f080c4b)
LAST SYNC
Sync OK to f080c4b
netflix
netflix-app
node-exporter
netflix-app
netflix-app-8878f4f5
netflix-app-8878f4f5-gchrj
netflix-app-8878f4f5-gwsm8
SYNC STATUS
Synced 3
OutOfSync 0
HEALTH STATUS
Progressing 0
Suspended 0
Healthy 6
Degraded 0
Missing 0
Unknown 0
NAMESPACES


NETFLIX
My List
Movies
Tv Shows
The Accountant²
When an old acquaintance is murdered, Wolff is compelled to solve the case. Realizing more extreme measures are necessary, Wolff recruits his estranged and highly lethal brother, Brax, to...
Play
More Info
7+
Popular Movies


Grafana
Home › Dashboards
Search...
Home
Bookmarks
Starred
Dashboards
Playlists
Snapshots
Library panels
Shared dashboards
Explore
Drilldown
New!
Alerting
Connections
Add new connection
Data sources
Administration
Dashboards
Create and manage dashboards to visualize your data
New
Search for dashboards and folders
Filter by tag
Starred
Sort
Name
Tags
Jenkins: Performance and Health Overview
Kubernetes EKS Cluster (Prometheus)
kubernetes
kubernetes-app
Node Exporter Full
linux


Grafana
Home › Dashboards › Jenkins: Performance and Health Overview
Search...
Edit
Export
Share
Last 30 minutes
Refresh
Processing speed
0
Job queue duration
0.03
0.02
0.01
0
15:15
15:20
15:25
15:30
15:35
15:40
Queued rate
0
JVM free memory
47041851300.0%
Memory Usage
470 MB
Jenkins health
1.0
JVM Uptime
N/A
Jenkins nodes offline
None!
CPU Usage
0.00274%
Total Jobs
None!
Sucessful Jobs
2
Aborted Jobs
No data
Unstable Jobs
No data
Failed Jobs
None!
Jenkins queue size
0
Executor free
2
Executors health
8.19 K
4.10 K
2.05 K
Job Duration
17.5 mins
8.74 mins
4.37 mins
Executor In-use
0


Grafana
Home › Dashboards › Node Exporter Full
Datasource prometheus
Job node_exporter
Nodename ip-172-31-14-49
Instance 13.127.239.25:9100
Quick CPU / Mem / Disk
Pressure
CPU Busy 0.3%
Sys Load 0.0%
RAM Used 15.1%
SWAP Used N/A
Root FS Used 15.3%
CPU Cores 2
Reboot R... N/A
Uptime 1.2 days
RootFS T... 23 GiB
RAM Total 4 GiB
SWAP Total 0 B
Basic CPU / Mem / Net / Disk
CPU Basic
Memory Basic
Network Traffic Basic
Disk Space Used Basic
CPU / Memory / Net / Disk (10 panels)


Grafana
Home › Dashboards › Node Exporter Full
Datasource prometheus
Job k8s
Nodename ip-192-168-35-188.ap-south-1.compute.internal
Instance 52.66.245.220:9100
Quick CPU / Mem / Disk
Pressure
CPU Busy 2.8%
Sys Load 3.0%
RAM Used 20.6%
SWAP Used N/A
Root FS Used 5.0%
CPU Cores 2
Reboot R... N/A
Uptime 1.0 days
RootFS T... 80 GiB
RAM Total 4 GiB
SWAP Total 0 B
Basic CPU / Mem / Net / Disk
CPU Basic
Memory Basic
Network Traffic Basic
Disk Space Used Basic