

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.

Instances (1/2) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
<input checked="" type="checkbox"/> Netflix-jenkins	i-0c8025c60cb72077b	Running	t2.large		2/2 checks passed	View alarms +	ap-south-1a	ec2-13-233-21-75.ap... 13.233.21.75	13.233.21.75 15.233.2
<input type="checkbox"/> Monitoring	i-0eb3d3ab0ba8707f	Running	t2.medium		Initializing	View alarms +	ap-south-1a	ec2-13-203-138-66.ap... 13.203.138.66	13.203.138.66 15.203.1

i-0c8025c60cb72077b (Netflix-jenkins)

Details Status and alarms | Monitoring | Security | Networking | Storage | Tags

Instance summary

Instance ID	i-0c8025c60cb72077b	Public IPv4 address	13.233.21.75 Open address
IPv6 address	-	Instance state	Running
Hostname type	IP name: ip-172-31-45-15.ap-south-1.compute.internal	Private IP DNS name (IPv4 only)	ip-172-31-45-15.ap-south-1.compute.internal
Another private resource DNS name	IPv4 (A)	Instance type	t2.large

Private IPv4 addresses

- 172.31.45.15

Public IPv4 DNS

- ec2-13-233-21-75.ap-south-1.compute.amazonaws.com | [Open address](#)

Elastic IP addresses

- 13.233.21.75 (netflix-esp) [Public IP]

Ec2 instance Launching

sg-07e1fd5bd5c15f114 - netflix-jenkins

Details

Security group name
[netflix-jenkins](#)

Security group ID
[sg-07e1fd5bd5c15f114](#)

Description
[launch-wizard-1 created 2024-12-29T14:14:44.733Z](#)

VPC ID
[vpc-0b4c5dd12d63f0419](#)

Owner
[288761748975](#)

Inbound rules count
6 Permission entries

Outbound rules count
1 Permission entry

[Inbound rules](#)

[Outbound rules](#)

Sharing - new

VPC associations - new

Tags

Inbound rules (6)

[Search](#)

[Edit inbound rules](#)

	Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
<input type="checkbox"/>	-	sgr-0f8acdf5f1bfaf90009	IPv4	HTTP	TCP	80	0.0.0.0/0	-
<input type="checkbox"/>	-	sgr-0c1c6fc1bde648fffb	IPv4	HTTPS	TCP	443	0.0.0.0/0	-
<input type="checkbox"/>	-	sgr-0405acf1bb0fe770b	IPv4	Custom TCP	TCP	8081	0.0.0.0/0	application
<input type="checkbox"/>	-	sgr-08a0d967b6eeb59c	IPv4	Custom TCP	TCP	9000	0.0.0.0/0	sonar port
<input type="checkbox"/>	-	sgr-045fc0d45ec169ffa	IPv4	Custom TCP	TCP	8080	0.0.0.0/0	jenkins port
<input type="checkbox"/>	-	sgr-0102abfeaf5bffe3	IPv4	SSH	TCP	22	0.0.0.0/0	-

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>
```

```
krupakar@krupakar:~/Downloads$ ssh -i NEW-Key.pem ubuntu@13.233.21.75
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-1021-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Sat Jan  4 16:57:06 UTC 2025

System load:  0.0          Processes:           129
Usage of /:   45.0% of 23.17GB  Users logged in:     0
Memory usage: 46%          IPv4 address for enX0: 172.31.43.15
Swap usage:   0%

* Ubuntu Pro delivers the most comprehensive open source security and
  compliance features.

  https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

32 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

1 additional security update can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm
```

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.

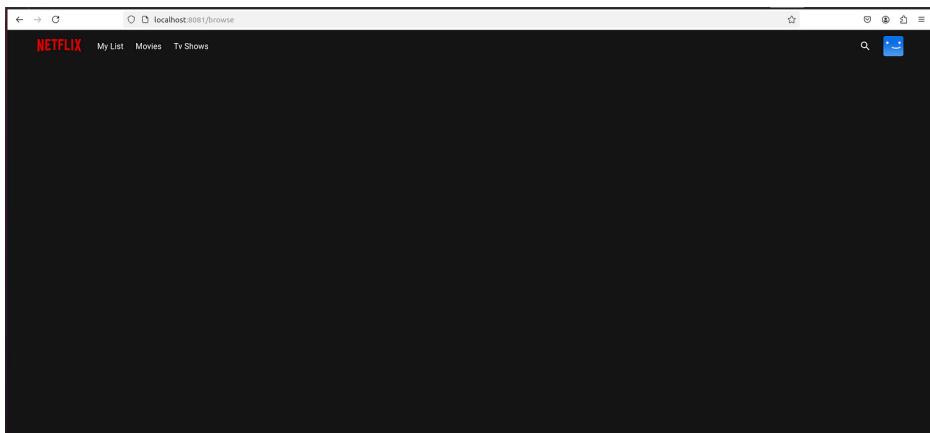
Run Docker Container:

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 netflix
```

```
ubuntu@ip-172-31-43-151:/DevSecOps-Project$ docker run -d --name netflix -p 8081:80 netflix:latest  
e8ee0451bdd3f548501f28badff84639c0dd88fbfd4dd3cbf388afef15f1c1  
ubuntu@ip-172-31-43-151:/DevSecOps-Project$ docker ps  
CONTAINER ID        IMAGE               CREATED             STATUS              PORTS               NAMES  
e8ee0451bdd3        netflix:latest      "nginx -g 'daemon off;'"   6 seconds ago       Up 5 seconds          0.0.0.0:8081->80/tcp   netflix  
ubuntu@ip-172-31-43-151:/DevSecOps-Project$
```

Docker container



Accessing the app without TMDB API Key

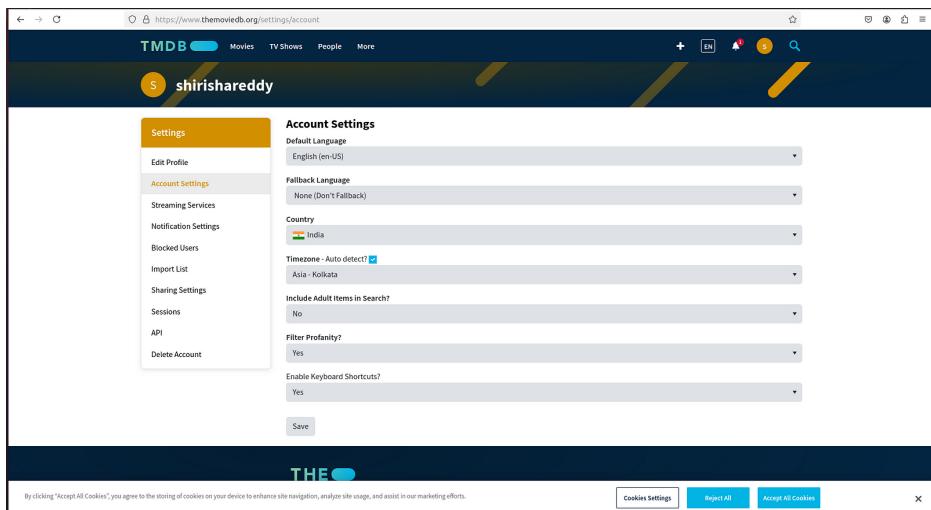
Rebuild Docker Image with TMDB API Key

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:

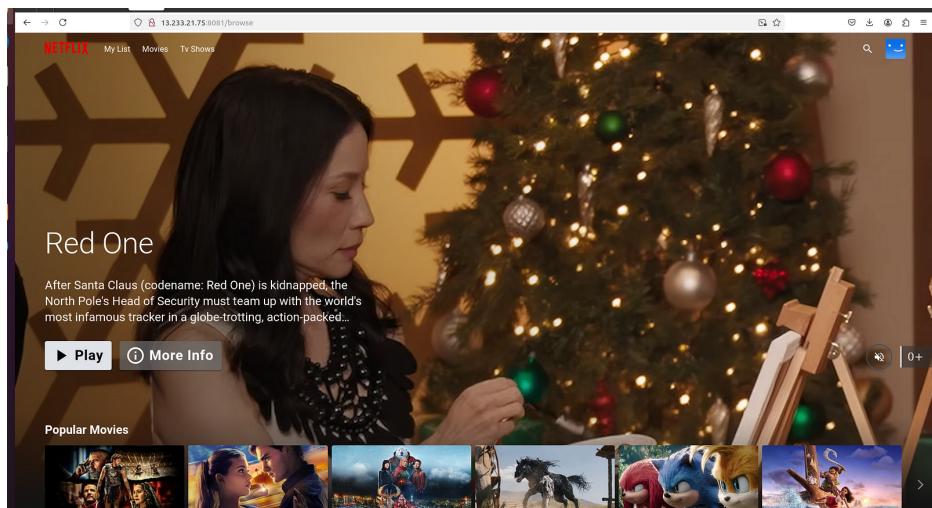
- With API Key:** Full access to detailed movie information (e.g., ratings, cast, genres).
- Without API Key:** Limited or no access to data, and the API will return an error.

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 :

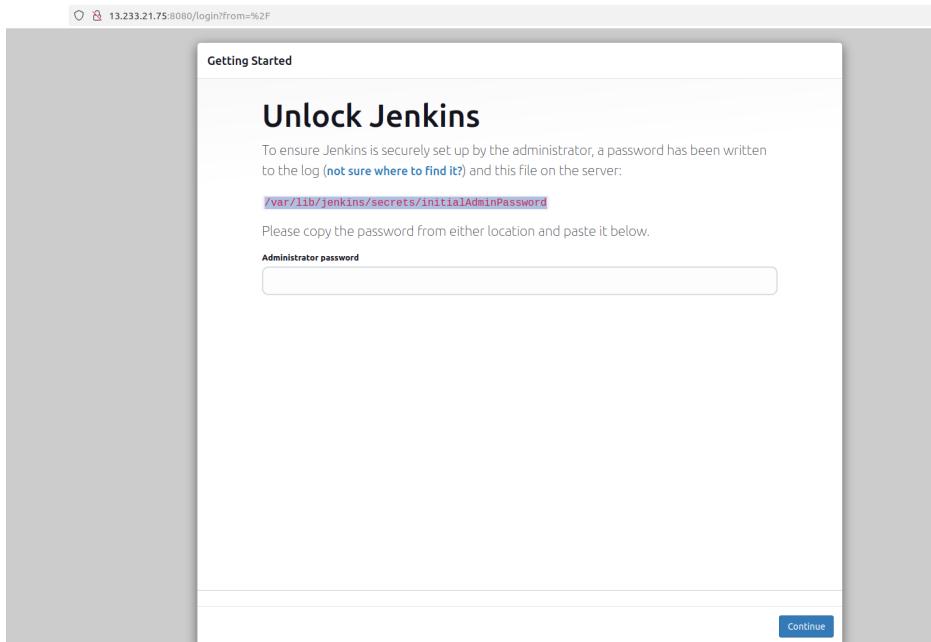
Verify Java Installation & Install Jenkins

```
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-Debian

# Jenkins
sudo wget -O /usr/share/keyrings/jenkins-keyring.gpg https://jenkins-ci.org/debian/jenkins-ci.org.key
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.gpg] http://pkg.jenkins.io/debian-stable/ jenkins main
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)

* Configure 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 workspace
            }
        }
        stage('Checkout from Git') {
            steps {
                git branch: 'main', url: '<URL>'
            }
        }
        stage('Dependency-Check Scan') {
            steps {
                dependencyCheck()
            }
        }
    }
}
```

```
        }
    stage('Install Dependencies') {
        steps {
            sh "npm install" // Install Node.js dependencies
        }
    }
    stage('OWASP Dependency-Check Scan') {
        steps {
            // Run Dependency-Check scan or
            dependencyCheck additionalArguments
                // Publish the Dependency-Check report
            dependencyCheckPublisher pattern
        }
    }
}
}
```

* **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 SonarQube instance
    }
    stages {
        stage('Build') {
            steps {
                script {
                    // Your build steps here
                }
            }
        }
        stage('SonarQube Analysis') {
```

```
steps {
    script {
        // Run SonarQube Scanner for Java
        withSonarQubeEnv('SonarQube Java') {
            sh 'mvn clean install sonar:sonar'
        }
    }
}
```

The screenshot shows the SonarQube dashboard for the Netflix project. At the top, there's a navigation bar with links for Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, and a search bar. Below the navigation, it shows the project name "Netflix" and the branch "main". The main area has tabs for Overview, Issues, Security Hotspots, Measures, Code, and Activity. The Overview tab is selected. On the left, there's a green box indicating "Passed" with the message "All conditions passed". The right side displays various quality gate status cards and measures. One card shows "Overall Code" with "New Code" (0 new bugs), "Reliability" (A), and "Security" (A). Another card shows "New Vulnerabilities" (0 new vulnerabilities) and "Security Review" (A). A third card shows "New Code Smells" (0 added debt) and "Maintainability" (A). At the bottom, there are cards for "Coverage" (0 new lines to cover) and "Duplications" (0 new lines).

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.

```

ubuntu@ip-172-31-41-151:/DevSecOps-Project5$ trivy image 94f1800e4bcc
[...]
2025-01-05T07:15:02Z [INFO] [vulnlib] Downloading vulnerability DB...
2025-01-05T07:15:02Z [INFO] [vulnlib] Downloading artifact... repo="mirror.gcr.io/aquasec/trivy-db:2"
2025-01-05T07:15:02Z [INFO] [vulnlib] Artifect successfully downloaded repo="mirror.gcr.io/aquasec/trivy-db:2"
2025-01-05T07:15:02Z [INFO] [vulnlib] Artifect successfully indexed
2025-01-05T07:15:02Z [INFO] [secret] Secret scanning is enabled
2025-01-05T07:15:02Z [INFO] [secret] To enable secret scanning, run 'trivy --secret' or 'trivy --secret=on'. You can also run 'trivy --secret-help' to disable secret scanning
2025-01-05T07:15:02Z [INFO] [secret] Please see also https://aquasecurity.github.io/trivy/v0.58/docs/scanner/secret-scanning
2025-01-05T07:15:02Z [INFO] [secret] Detected OS: alpine" version="3.20.3"
2025-01-05T07:15:02Z [INFO] [secret] Detected language: C
2025-01-05T07:15:02Z [INFO] Number of language-specific files: 0
2025-01-05T07:15:02Z [INFO] Using severities from other vendors for some vulnerabilities. Read https://aquasecurity.github.io/trivy/v0.58/docs/scanner/vulnerability#severity-selection for details.
24c1800e4bcc ( Alpine 3.20.3)

Total: 2 (UNKNOWN: 0, LOW: 0, MEDIUM: 2, HIGH: 0, CRITICAL: 0)

```

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
curl	CVE-2024-11053	MEDIUM	fixed	8.11.0-r2	8.11.1-r0	curl: curl netrc password leak https://curl.aquasec.dev/cve/CVE-2024-11053
libcurl						

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.

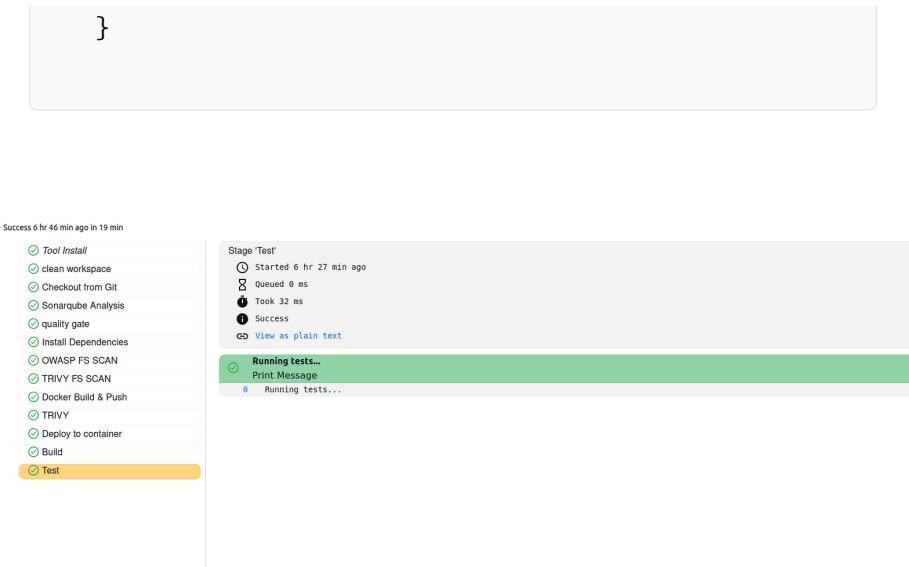
```

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-scanner
                        ...
                    '''
                }
            }
        }
        stage('Quality Gate') {
            steps {
                script {
                    waitForQualityGate abortPipeline
                }
            }
        }
    }
    stage('Install Dependencies') {
        steps {
            sh "npm install"
        }
    }
}
```

```
stage('OWASP FS Scan') {
    steps {
        dependencyCheck additionalArguments
        dependencyCheckPublisher pattern
    }
}
stage('TRIVY FS Scan') {
    steps {
        sh "trivy fs . > trivyfs.txt"
    }
}
stage('Docker Build & Push') {
    steps {
        script {
            withDockerRegistry(credentialsId: 'sirishassss-docker-registry')
                sh "docker build --build-arg=IMAGE_NAME=sirishassss/netflix:latest ."
                sh "docker tag netflix:latest sirishassss/netflix:latest"
                sh "docker push sirishassss/netflix:latest"
        }
    }
}
stage('TRIVY Image Scan') {
    steps {
        sh "trivy image sirishassss/netflix:latest"
    }
}
stage('Deploy to container') {
    steps {
        sh 'docker run -d -p 8081:80 sirishassss/netflix'
    }
}
```



Pipeline

sirinhassss / [Repositories](#) / [netflix](#) / General

General Tags Builds Collaborators Webhooks Settings

sirinhassss/netflix [Edit](#)

Last pushed less than a minute ago

Add a description [Incomplete](#)

Add a category [Incomplete](#)

Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	a few seconds ago	a few seconds ago

[See all](#)

Docker commands

To push a new tag to this repository:

```
docker push sirinhassss/netflix:tagname
```

[Public view](#)

Automated builds

Manually pushing images to Docker Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions. [Read more about automated builds](#)

[Upgrade](#)

Repository overview [Incomplete](#)

An overview describes what your image does and how to run it. It displays in [the public view of your repository](#) once you have pushed some content.

[Add overview](#)

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**.

The screenshot shows the AWS CloudWatch Instances console. At the top, there is a search bar and a filter dropdown set to 'All states'. Below the header, there are buttons for 'Connect', 'Instance state', 'Actions', and 'Launch instances'. The main table lists two instances:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
Netflix-jenkins	i-0c025c60cb72077b	Running	t2.large	2/2 checks passed	View alarms +	ap-south-1a	ec2-15-233-21-75.ap-s...	13.233.21.75	13.233.2
Monitoring	i-0eb83d9ab5ba8707f	Running	t2.medium	Initializing	View alarms +	ap-south-1a	ec2-15-203-138-86.ap...	13.203.138.86	13.203.1

Below the table, a specific instance is selected: 'i-0eb83d9ab5ba8707f (Monitoring)'. The 'Details' tab is active, showing the following details:

- Instance summary**: Info
- Instance ID**: i-0eb83d9ab5ba8707f
- IPv6 address**: -
- Hostname type**: IP name: ip-172-31-37-78.ap-south-1.compute.internal
- Answer private resource DNS name**: IPv4 (A)
- Public IPv4 address**: 13.203.138.86 | open address
- Instance state**: Running
- Private IP DNS name (IPv4 only)**: ip-172-31-37-78.ap-south-1.compute.internal
- Instance type**: t2.medium
- Private IPv4 addresses**: 172.31.57.78
- Public IPv4 DNS**: ec2-15-203-138-86.ap-south-1.compute.amazonaws.com | open address
- Elastic IP addresses**: 13.203.138.86 (monitor) [Public IP]

Instance to setup Monitoring tools

The screenshot shows the AWS Security Groups interface for a security group named 'sg-0960033dcff2b44e4'. The 'Details' tab is selected, displaying basic information like the security group name, owner, and VPC ID. The 'Inbound rules' tab is active, showing five entries. These rules allow traffic from specific IP ranges or security groups (e.g., Jenkins instances) on various ports (SSH, TCP, Custom TCP, HTTP) to port 80 on this security group.

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
-	sgr-04185702491c04ec8	IPv4	SSH	TCP	22	0.0.0.0/0	-
-	sgr-0b008891d6019657	IPv4	Custom TCP	TCP	5000	0.0.0.0/0	Grafana
-	sgr-0250d893b4ec19753	IPv4	Custom TCP	TCP	9090	0.0.0.0/0	promethe
-	sgr-0b1f0d0708ecf5d7	IPv4	Custom TCP	TCP	9100	0.0.0.0/0	node-exp
-	sgr-0cbddc2d157cae530	IPv4	HTTP	TCP	80	0.0.0.0/0	-

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/releases
```

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
```

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/console_libraries \
--web.console.libraries=/etc/prometheus/console_libraries
```

```
--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>

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://0.110.62.164:9100/metrics	UP	instance="0.110.62.164:9100"; job="K8s"	2.801s ago	14.810ms	

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://13.233.21.75:8080/prometheus	UP	instance="13.233.21.75:8080"; job="jenkins"	12.226s ago	9.281ms	

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://13.203.138.86:9100/metrics	UP	instance="13.203.138.86:9100"; job="node_exporter"	4.770s ago	16.140ms	

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	instance="localhost:9090"; job="prometheus"	3.624s ago	5.580ms	

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\_exporter
```

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.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 --collector

[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-jenki
```

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.yaml
```

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 so1
```

Step 2: Add the GPG Key:

Add the GPG key for Grafana:

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

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.

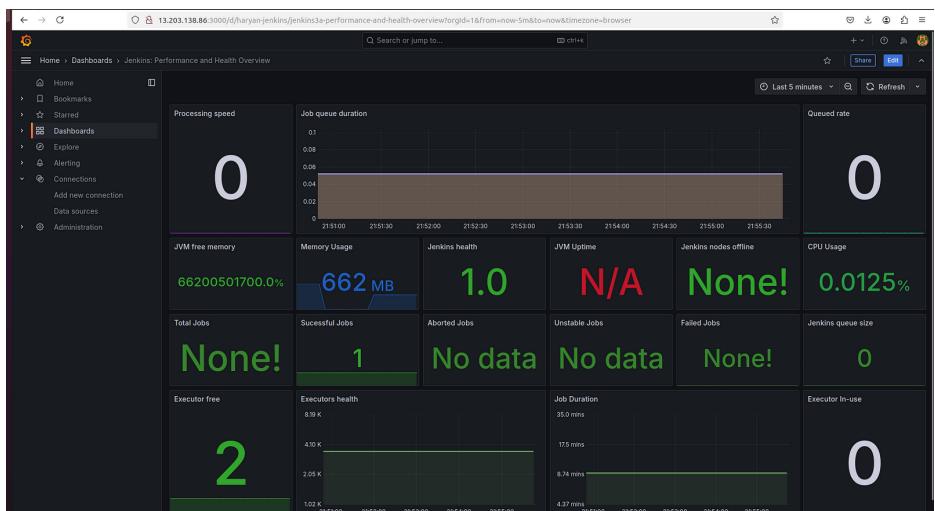
Grafana is a powerful tool for creating visualizations and dashboards,

Configure Prometheus Plugin Integration:

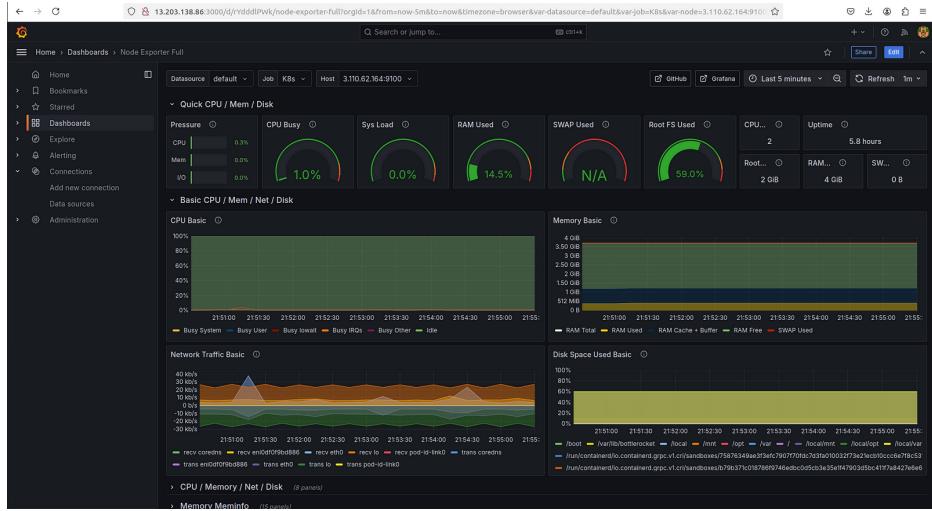
Integrate Jenkins with Prometheus to monitor the CI/CD pipeline.

The screenshot shows the Grafana interface with the URL `13.203.138.86:3000/dashboards`. On the left, a sidebar menu includes Home, Bookmarks, Starred, Dashboards (selected), Explore, Alerting, Connections, Data sources, and Administration. The main area is titled "Dashboards" with the sub-instruction "Create and manage dashboards to visualize your data". A search bar at the top says "Search for dashboards and folders". Below it, a filter section has "Filter by tag" and "Starred" checkboxes, followed by a table with columns "Name" and "Tags". Two items are listed: "Jenkins: Performance and Health Overview" and "Node Exporter Full", both with a "Tags" column containing "Issue". A "New" button is in the top right.

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.

The screenshot shows the AWS EKS Cluster list page. On the left, there's a navigation sidebar for 'Amazon Elastic Kubernetes Service' with sections for Clusters, Amazon EKS Anywhere, and Related services (Amazon ECR, AWS Batch). The main area displays a table titled 'Clusters (1) Info' with one entry for 'Netflix'. The table columns include Cluster name, Status, Kubernetes version, Support period, Upgrade policy, Created, and Provider (EKS). A 'Create cluster' button is at the top right.

EKS Cluster

This screenshot shows the detailed view of the 'Netflix' cluster. It includes sections for Cluster info (Status: Active, Kubernetes version: 1.31, Support period: Standard support until November 26, 2025, Provider: EKS), Cluster health issues (0), Upgrade insights (5), Node health issues (0), and various tabs like Overview, Resources, Compute, Networking, Add-ons, Access, Observability, Update history, and Tags. The Overview tab is selected. There are also sections for API server endpoint, Certificate authority, IAM roles, and Kubernetes version settings.

Cluster

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
1	i-0f9cd85ff6e11357	Running	c6a.large	2/3 checks passed	View alarms +	ap-south-1c	ec2-13-200-138-162.ap...	13.200.138.162	-
Netflix-jenkins	i-0c8025d0cb72077b	Running	t2.large	2/2 checks passed	View alarms +	ap-south-1a	ec2-13-253-21-75.ap-s...	13.253.21.75	13.253.2
Monitoring	i-0eb83d9ab5ba707f	Running	t2.medium	2/2 checks passed	View alarms +	ap-south-1a	ec2-13-203-138-86.ap...	13.203.138.86	13.203.1
cluster	i-07906110493989e09	Running	c6g.large	3/3 checks passed	View alarms +	ap-south-1a	ec2-3-110-62-164.ap-s...	3.110.62.164	-

c6alarge instances are nodes for the cluster

The screenshot shows the AWS CloudFormation console with a security group named 'sg-09c9f7398d74e64c3' for the 'eks-cluster-sg-Network-787338382' stack. The 'Details' tab is selected, displaying information such as the security group name, owner, and VPC ID. The 'Inbound rules' tab is active, showing four rules with details like source IP ranges and protocols. The 'Outbound rules' tab shows one rule. The 'Sharing - new' and 'VPC associations - new' tabs are also present.

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
sgr-079df8562f5649fc1	-	All traffic	All	All	0.0.0.0/0	sg-09c9f7398d74e64c3...	-
sgr-0ab85d059208635b3	IPv4	Custom TCP	TCP	30007	0.0.0.0/0	app-node-port	-
sgr-001c50fa6e2fe26bf	-	All traffic	All	All	0.0.0.0/0	sg-078b36fb3345f922f...	-
sgr-019729bf61d24ab30	IPv4	Custom TCP	TCP	9100	0.0.0.0/0	node-exporter	-

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.

```
krupakar@krupakar:~/workspace/kubernetes$ aws eks update-kubeconfig --name "netflix" --region "ap-south-1"
Added new context arn:aws:eks:ap-south-1:288761748973:cluster/netflix to /home/krupakar/.kube/config
krupakar@krupakar:~/workspace/kubernetes$
```

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 -c "gpg --dearmor" - > /usr/share/keyrings/baltocdn.gpg  
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://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 prometheu
```

```
[root@krukaart ~]# kubectl create namespace prometheus-node-exporter
namespace/prometheus-node-exporter created
[krukaart@krukaart:~/workspace/kubernetes]$ helm install prometheus-node-exporter prometheus-community/prometheus-node-exporter --namespace prometheus-node-exporter
LAST DEPLOYED: Sun Jan 5 16:04:24 2025
NAMESPACE: prometheus-node-exporter
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
1. Get the application URL by running these commands:
export POD_NAME=$(kubectl get pods --namespace prometheus-node-exporter -l app.kubernetes.io/name=prometheus-node-exporter,app.kubernetes.io/instance=prometheus-node-exporter" -o jsonpath=".items[0].status.podIP")
echo "Visit http://$POD_NAME:9100 to use your application"
kubectl port-forward --namespace prometheus-node-exporter $POD_NAME 9100
```

prometheus-node-exporter installation through 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.

```
krupakar@krupakar:~/workspace/kubernetes$ kubectl create namespace argocd
namespace/argocd created
krupakar@krupakar:~/workspace/kubernetes$ kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/v2.4.7/manifests/install.yaml
customresourcedefinition.aplxextensions.k8s.io/applications.argoproj.io created
customresourcedefinition.aplxextensions.k8s.io/applicationsets.argoproj.io created
customresourcedefinition.aplxextensions.k8s.io/applicationsets.argoproj.io created
serviceaccount/argocd-application-controller created
serviceaccount/argocd-applicationset-controller created
serviceaccount/argocd-dex-server created
serviceaccount/argocd-notifications-controller created
serviceaccount/argocd-redis created
serviceaccount/argocd-repo-server created
serviceaccount/argocd-server created
role.rbac.authorization.k8s.io/argocd-application-controller created
role.rbac.authorization.k8s.io/argocd-applicationset-controller created
role.rbac.authorization.k8s.io/argocd-dex-server created
role.rbac.authorization.k8s.io/argocd-notifications-controller created
role.rbac.authorization.k8s.io/argocd-server created
serviceaccount/argocd-authentication-controller created
```

Argocd installation

Load balancers (1/1)

Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

<input checked="" type="checkbox"/>	Name	DNS name	State	VPC ID	Availability Zones	Type	Date created
<input checked="" type="checkbox"/>	a542cf5c97ed84154b98e9678fa23856	a542cf5c97ed84154b98e9...	-	vpc-0b4c5dd12d03f0419	3 Availability Zones	classic	January 4, 2025, 21:40 (UTC+05:30)

Load balancer: a542cf5c97ed84154b98e9678fa23856

[Details](#) | [Listeners](#) | [Network mapping](#) | [Security](#) | [Health checks](#) | [Target instances](#) | [Monitoring](#) | [Attributes](#) | [Tags](#)

Details

Load balancer type Classic	Status 2 of 3 instances in service	VPC vpc-0b4c5dd12d03f0419	Date created January 4, 2025, 21:40 (UTC+05:30)
Scheme Internet-facing	Hosted zone ZP97RAFLXTNZK	Availability Zones subnet-0a9b812dbd69f4809 subnet-0ad29a5dbf0eeecd4 subnet-0e67a0801ba406df7	ap-south-1a (aps1-az1) ap-south-1b (aps1-az3) ap-south-1c (aps1-az2)

LB

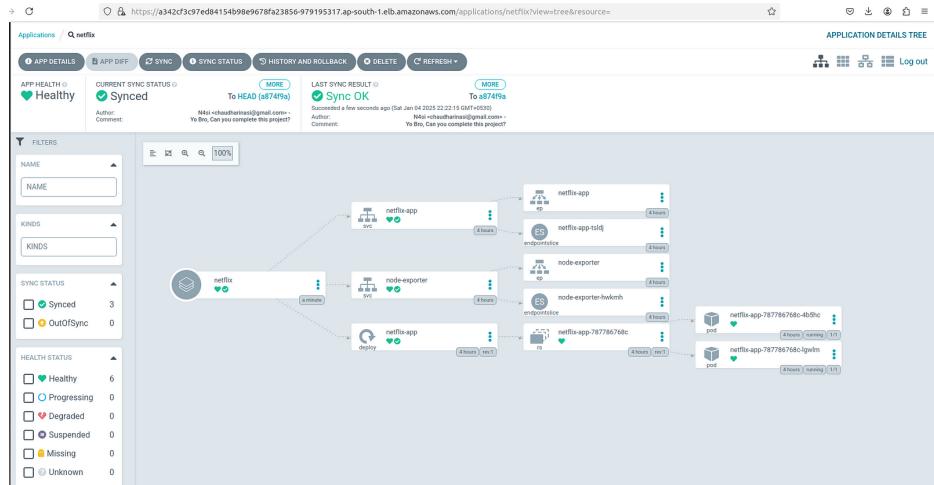
- Set the GitHub Repository as a Source:**
- After installing ArgoCD, we need to set up the GitHub repository as a source for our application deployment. This typically involves configuring the connection to our repository and defining the source for our ArgoCD application. The specific steps will depend 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

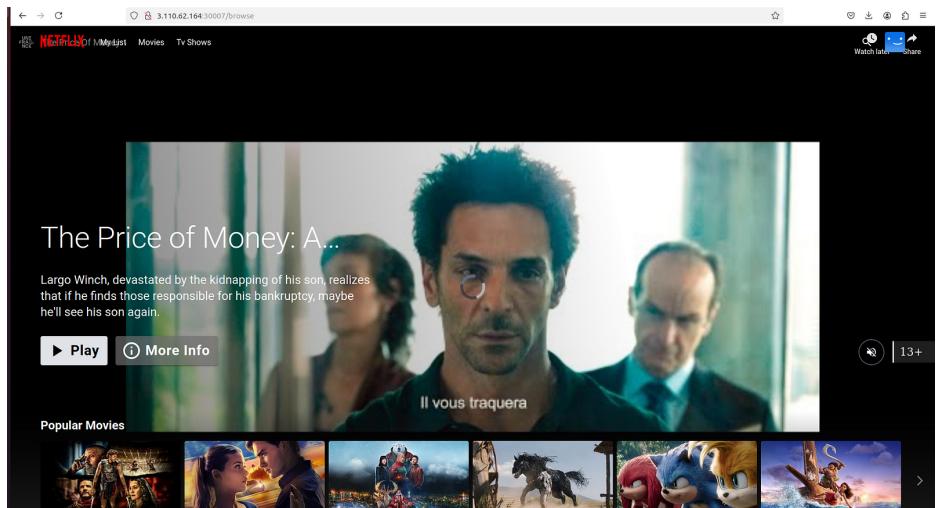
```
[krupakar@krupakar:~/workspace/kubernetes]$ kubectl get svc argocd-server -n argocd
NAME           TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
argocd-server  LoadBalancer 10.100.97.136  alads385bb2e94505937d55193343  8082/5492,443:32744/TCP  21m
[krupakar@krupakar:~/workspace/kubernetes]$ argocd login $ARGOCD_SERVER --username admin --password $ARGO_PWD --insecure
admin:login' logged in successfully
Context 'aiad5385bb2e9450599327d55193343-882925492.ap-south-1.elb.amazonaws.com' updated
[krupakar@krupakar:~/workspace/kubernetes]$ echo $ARGOCD_SERVER
aiad5385bb2e9450599327d55193343-882925492.ap-south-1.elb.amazonaws.com
[krupakar@krupakar:~/workspace/kubernetes]$ echo $ARGO_PWD
tsgpNIO99ZG3Gr
[krupakar@krupakar:~/workspace/kubernetes]$ kubectl get po
NAME           READY   STATUS    RESTARTS   AGE
httpbin-app-78778676bc-h0jpx   1/1     Running   0          59s
httpbin-app-78778676bc-rqnp   1/1     Running   0          59s
[krupakar@krupakar:~/workspace/kubernetes]$ kubectl get nodes
NAME           STATUS   ROLES   AGE
k8s-worker-node-8fhng   Ready   <none>  49m
tsgpNIO99ZG3Gr-7771774   Ready   <none>  77m
V1.31.1-eks-ib8e056
[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   10.100.0.121  <none>       80:38007/TCP  92s
[krupakar@krupakar:~/workspace/kubernetes]$ kubectl get ns
NAME           STATUS   AGE
argocd          Active   28m
default         Active   50s
kube-node-lease Active   50m
kube-public     Active   50m
kube-system     Active   50m
prometheus-node-exporter  30m
[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.