

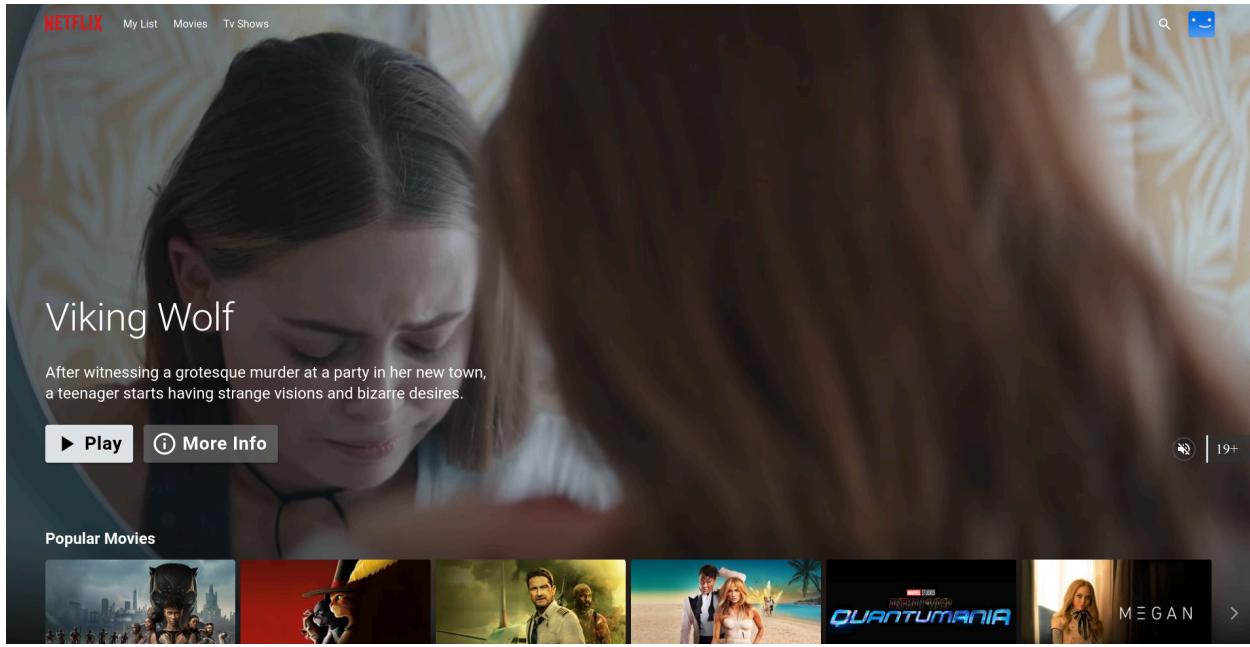
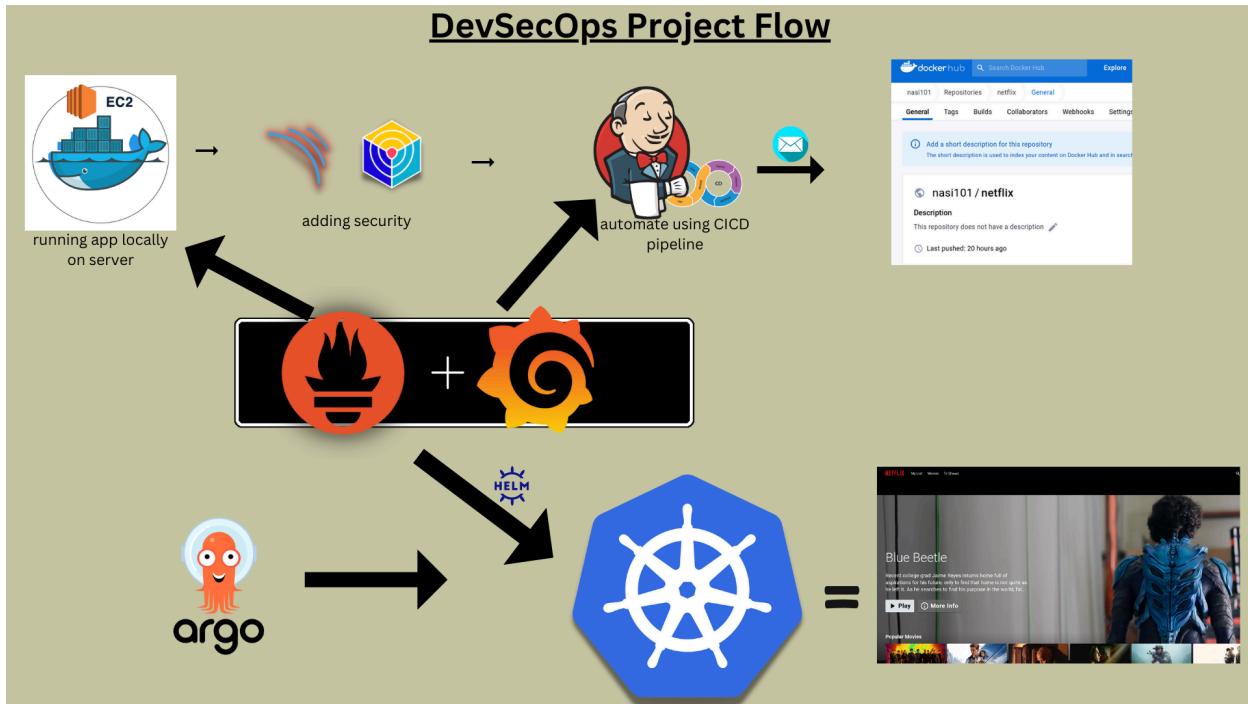
DevSecOps Pipeline Project: Deploy Netflix Clone using Jenkins on Kubernetes

Project Description

We will deploy a Netflix Clone application as a Docker container on a Kubernetes cluster through a secure CI/CD pipeline using Jenkins. This project incorporates several popular DevOps tools including Docker, SonarQube, Trivy, Prometheus, Grafana, ArgoCD, and Helm for Kubernetes deployments.

Pre-requisites

- **AWS account**
- **DockerHub account**
- **TMDB API Key** from The Movie Database (for application to fetch movie data).
 - Go to TMDB (The Movie Database) website <https://www.themoviedb.org/> .
 - Log in → Profile → Settings → API
 - Generate a v3 API Key by clicking "Create"
 - Provide the required basic details and click "Submit."
 - You will receive your TMDB API key. To be used to build the Docker image within this guide



Project Description

Pre-requisites

Phase 1: Initial Jenkins Setup and Deployment

Step 1: Launch EC2 (Ubuntu 22.04)

Step 2: Clone the Repository

Step 3: Install Docker

Step 4: Docker Build and run: with the TMDB API Key

Step 5: Stop and remove the Netflix container (since we want the build, push to Docker hub via Jenkins pipeline)

Phase 2: Security Tools Installation

Step 1: Install SonarQube (to analyze code for quality and security issues)

Step 2: Install Trivy (Container Vulnerability Scanner)

Phase 3: Jenkins CI/CD Setup

Step 1: Install Jenkins on Ubuntu

Step 2: Install Jenkins Plugins

Step 3: Add Credentials

Step 4: Configure Jenkins System

Step 5: Configure Global Tools

Step 6: Configure E-mail Notification

Step 7: Create project "Netflix" for Sonarqube Analysis

Step 8: Create Pipeline Job

Phase 4: Monitoring Setup (Prometheus & Grafana)

Step 1: Launch EC2 (Ubuntu 22.04)

Step 2: Allocate and Associate Elastic IP

Step 3: SSH into the Instance (to install Prometheus, Grafana, Node Explorer)

Step 4: Create user Prometheus & Install Prometheus

Step 5: Install Node Exporter

Step 6: Install Grafana on Ubuntu 22.04 and Set it up to Work with Prometheus

Phase 5: Create a Kubernetes Cluster with Node Groups (EKS)

Step 1: Create IAM Roles

Step 2: Create EKS Cluster

Step 3: Create Node Group

Step 4: Update your local kubeconfig file

Phase 6: Monitor Kubernetes with Prometheus

Step 1: Install Node Exporter with Helm

Step 2: Add Custom Scrape from K8s nodes in Prometheus

Phase 7: Deploy Application with ArgoCD

Step 1: Install ArgoCD

Step 2: Connect Your GitHub Repo

Step 3: Create ArgoCD Application

Step 4: Sync Application

Step 5: Access Netflix App

Phase 8: Cleanup

Troubleshoot

Solution:

Phase 1: Initial Jenkins Setup and Deployment

Step 1: Launch EC2 (Ubuntu 22.04)

1. Create a New EC2 Instance on AWS:

- **Name:** `netflix-jenkins`
- AMI: Latest Ubuntu 22.04 LTS
- Instance Type: t2.large (2 vCPUs, 8GB RAM)
- Key Pair: Your custom SSH key or can create a new one
- Storage: 25 GB gp2
- Network:
 - Default VPC & subnet (us-east-1a)
 - Auto-assign Public IP: Enabled
- Security Group:
 - SSH (22): 0.0.0.0/0
 - HTTP (80): 0.0.0.0/0
 - HTTPS (443): 0.0.0.0/0
 - Custom TCP (9000, 8081, etc.): Add as needed for app ports
 - All outbound traffic allowed



Security Tip: For production or team use, restrict SSH access to your IP.

Type	Protocol	Port range	Source	Description
HTTP	TCP	80	0.0.0.0/0	-
HTTPS	TCP	443	0.0.0.0/0	-
Custom TCP	TCP	9000	0.0.0.0/0	sonarqube
SSH	TCP	22	0.0.0.0/0	-
Custom TCP	TCP	8080	0.0.0.0/0	jenkins
Custom TCP	TCP	8081	0.0.0.0/0	app port

2. Allocate and Associate Elastic IP

- Allocate a new **Elastic IP** (Name it `netflix-eip`)
- Associate it with the `netflix-jenkins` EC2 instance

3. SSH into the instance:

```
ssh -i <your-key.pem> ubuntu@<elastic-ip>
sudo apt update -y
```

Step 2: Clone the Repository

Update packages, then clone the Netflix clone project:

```
git clone https://github.com/eduardobautista-devops/DevSecOps-Pipeline-Proje
cd DevSecOps-Pipeline-Project-Deploy-Network-Clone-on-Kubernetes
```

Step 3: Install Docker

- Install Docker and set permissions:

```
# Install Docker
sudo apt install docker.io -y

# Add current user to docker group (so you can run docker without sudo)
sudo usermod -aG docker $USER

# Activate group change above without logging out
newgrp docker
```

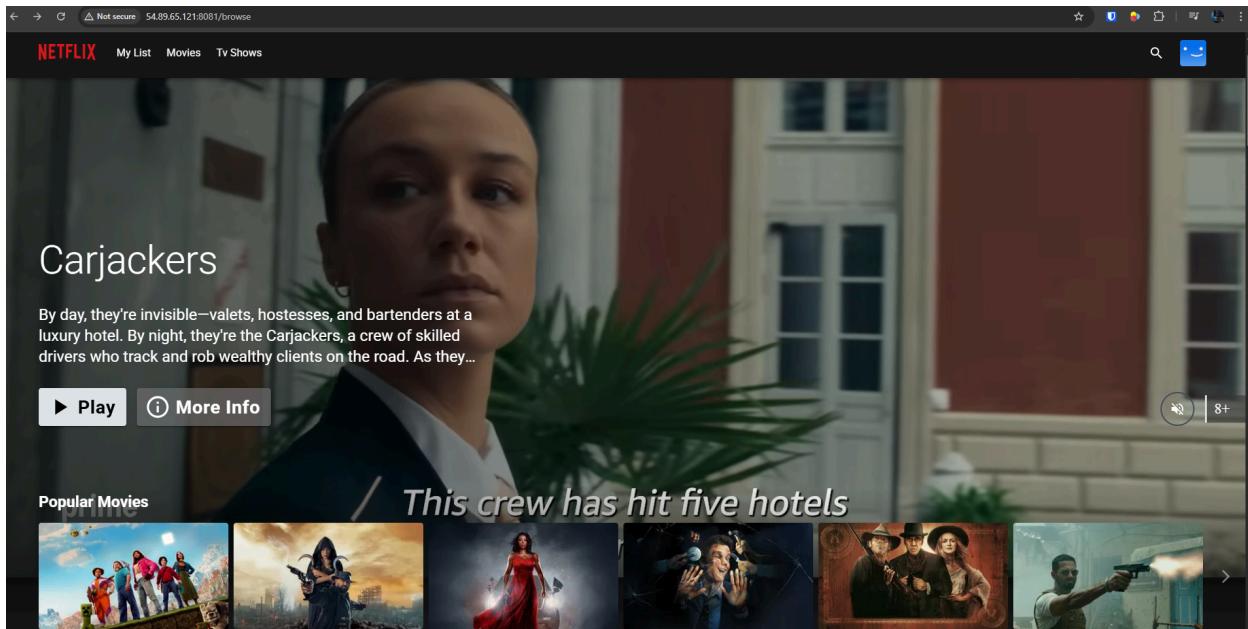
```
# Give full permissions to docker.sock (its communication with Docker on  
your machine)  
sudo chmod 777 /var/run/docker.sock
```

- Verify Docker installation: [docker version](#)

Step 4: Docker Build and run: with the TMDB API Key

```
docker build --build-arg TMDB_V3_API_KEY=<your-api-key> -t netflix .  
docker run -d --name netflix -p 8081:80 netflix:latest
```

- Open in browser: <http://<netflix-jenkins-elastic IP>:8081>



Step 5: Stop and remove the Netflix container (since we want the build, push to Docker hub via Jenkins pipeline)

```
docker ps  
docker stop <container_id>  
docker rm <container_id>
```

Phase 2: Security Tools Installation

Step 1: Install SonarQube (to analyze code for quality and security issues)

```
docker run -d --name sonar -p 9000:9000 sonarqube:its-community  
docker ps
```

Example output:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
32af19691b65	sonarqube:its-community	/opt/sonarqube/dock...	25 minutes	
a3e26696f45a	netflix:latest	"nginx -g 'daemon of..."	28 minutes ago	Up

Access: <http://<netflix-jenkins-elastic IP>:9000>

- Default credentials: [admin / admin](#)
- Change password on first login

Step 2: Install Trivy (Container Vulnerability Scanner)

```
sudo apt-get install -y 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) mai  
n | sudo tee -a /etc/apt/sources.list.d/trivy.list  
sudo apt-get update -y  
sudo apt-get install -y trivy
```

Verify installation:

```
trivy version
```

Scan image:

```
trivy image <imageid>
```

Example output:

```
ubuntu@ip-172-31-43-168:~/DevSecOps-Pipeline-Deploy-Network-Clone-on-Kubernetes$ trivy image a3e26696f45a
2025-04-13T14:18:29Z  INFO [vuln] Vulnerability scanning is enabled
2025-04-13T14:18:29Z  INFO [secret] Secret scanning is enabled
2025-04-13T14:18:29Z  INFO [secret] If your scanning is slow, please try '--scanners vuln' to disable secret scanning
2025-04-13T14:18:29Z  INFO [secret] Please see also https://trivy.dev/v0.61/docs/scanner/secret#recommendation for faster secret detection
2025-04-13T14:18:31Z  INFO Detected OS family="alpine" version="3.20.6"
2025-04-13T14:18:31Z  INFO [alpine] Detecting vulnerabilities... os_version="3.20" repository="3.20" pkg_num=66
2025-04-13T14:18:31Z  INFO Number of language-specific files num=0
2025-04-13T14:18:31Z  WARN Using severities from other vendors for some vulnerabilities. Read https://trivy.dev/v0.61/docs/scanner/vulnerability#severity-selection for details.

Report Summary



| Target                       | Type   | Vulnerabilities | Secrets |
|------------------------------|--------|-----------------|---------|
| a3e26696f45a (alpine 3.20.6) | alpine | 7               | -       |



Legend:
- -: Not scanned
- '0': Clean (no security findings detected)

a3e26696f45a (alpine 3.20.6)

Total: 7 (UNKNOWN: 0, LOW: 0, MEDIUM: 0, HIGH: 7, CRITICAL: 0)



| Library  | Vulnerability  | Severity | Status | Installed Version | Fixed Version | Title                                                                                                                        |
|----------|----------------|----------|--------|-------------------|---------------|------------------------------------------------------------------------------------------------------------------------------|
| libexpat | CVE-2024-8176  | HIGH     | fixed  | 2.6.4-r0          | 2.7.0-r0      | libexpat: expat: Improper Restriction of XML Entity Expansion Depth in libexpat<br>https://avd.aquasec.com/nvd/cve-2024-8176 |
| libxml2  | CVE-2024-56171 |          |        | 2.12.7-r0         | 2.12.7-r1     | libxml2: Use-After-Free in libxml2<br>https://avd.aquasec.com/nvd/cve-2024-56171                                             |
|          | CVE-2025-24928 |          |        |                   | 2.12.7-r2     | libxml2: Stack-based buffer overflow in xmlSprintfElements of libxml2<br>https://avd.aquasec.com/nvd/cve-2025-24928          |
|          | CVE-2025-27113 |          |        |                   |               | libxml2: NULL Pointer Dereference in libxml2_xmlPatMatch                                                                     |


```

Phase 3: Jenkins CI/CD Setup

Step 1: Install Jenkins on Ubuntu

1. Run the following installation:

```
# Update your system
sudo apt update -y
# Install Java
sudo apt install fontconfig openjdk-17-jre -y
java -version
# Add the Jenkins repository and key
openjdk version "17.0.8" 2023-07-18
OpenJDK Runtime Environment (build 17.0.8+7-Debian-1deb12u1)
OpenJDK 64-Bit Server VM (build 17.0.8+7-Debian-1deb12u1, mixed mode, sharing)
#jenkins
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
```

```
https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key  
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \  
https://pkg.jenkins.io/debian-stable binary/ | sudo tee \  
/etc/apt/sources.list.d/jenkins.list > /dev/null  
# Install Jenkins  
sudo apt-get update -y  
sudo apt-get install -y jenkins  
# Start and enable Jenkins  
sudo systemctl start jenkins  
sudo systemctl enable jenkins
```

2. Confirm jenkins was installed and running: `sudo service jenkins status`

3. Access Jenkins via: `http://<netflix-jenkins-elastic IP>:8080`

4. Retrieve initial admin password:

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

5. Follow setup wizard:

- Install suggested plugins
- Create admin user (or skip)
- Configure Jenkins URL (or keep default)
- Click "Start using Jenkins"

Step 2: Install Jenkins Plugins

1. Go to Manage Jenkins → Plugins → Available Plugins

2. Install the following plugins:

- Stage View
- Eclipse Temurin Installer
- SonarQube Scanner
- NodeJs

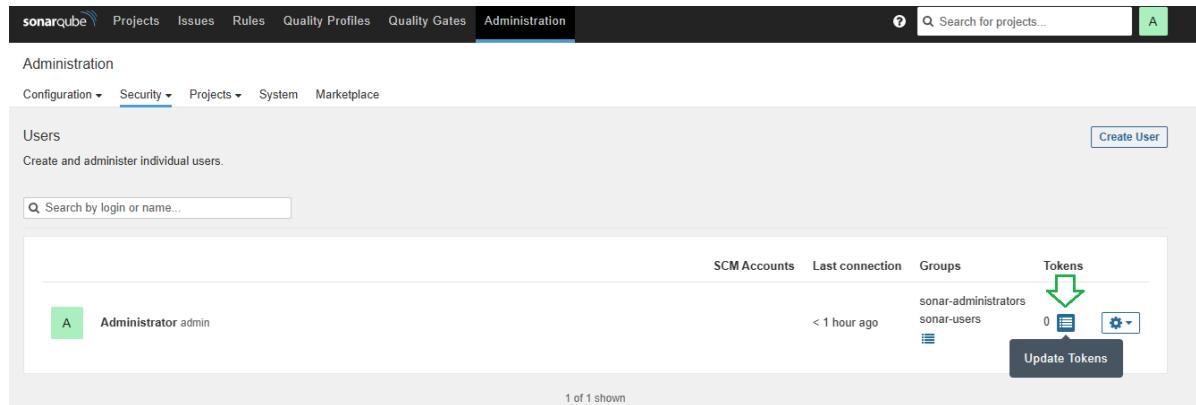
- OWASP Dependency-Check
- Docker
- Docker Commons
- Docker Pipeline
- Docker API
- docker-build-step

3. Click "Install without restart"

Step 3: Add Credentials

SonarQube Token

1. Access SonarQube via: <http://<netflix-jenkins-elastic IP>:9000>
2. In SonarQube: **Administration → Security → Users → Update Tokens**



The screenshot shows the SonarQube administration interface. The top navigation bar includes links for Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, and a search bar. The Administration menu is open, showing sub-links for Configuration, Security (which is selected), Projects, System, and Marketplace. The main content area is titled 'Users' and contains the sub-instruction 'Create and administer individual users.' Below this is a search bar labeled 'Search by login or name...'. A table lists a single user: 'Administrator admin'. The table columns include 'SCM Accounts', 'Last connection' (showing '< 1 hour ago'), 'Groups' (listing 'sonar-administrators' and 'sonar-users'), and 'Tokens' (showing '0'). A green arrow points to the 'Update Tokens' button, which is located at the bottom right of the table row.

2. Generate and copy a new token

Tokens of Administrator

The screenshot shows the Jenkins 'Tokens of Administrator' page. At the top, there's a 'Generate Tokens' section with a 'Name' input field containing 'jenkins', an 'Expires in' dropdown set to '30 days', and a 'Generate' button. Below this is a table header with columns: Name, Type, Project, Last use, Created, and Expiration. A note below the table says 'No tokens'. In the bottom right corner of the main area, there's a 'Done' button.

3. In Jenkins go to: **Manage Jenkins → Credentials → System → Global credentials (unrestricted)**

- Type: Secret Text
- Secret: (paste token)
- ID: `sonar-token`

The screenshot shows the Jenkins 'New credentials' creation form. The 'Kind' field is set to 'Secret text'. The 'Scope' field is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Secret' field contains a redacted password. The 'ID' field is set to 'Sonar-token'. The 'Description' field is also set to 'Sonar-token'. At the bottom is a 'Create' button.

DockerHub

- Add credentials:
 - Type: Username with Password
 - ID: docker
 - Fill DockerHub credentials

The screenshot shows the Jenkins 'New credentials' configuration page. The path in the top navigation bar is: Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >. The main title is 'New credentials'. The 'Kind' field is set to 'Username with password'. The 'Scope' is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Username' field contains 'xxxxxxxxxxxxxxxxxxxxxx' (redacted). The 'Treat username as secret' checkbox is unchecked. The 'Password' field contains '.....' (redacted). The 'ID' field is set to 'docker'. The 'Description' field is also set to 'docker'. At the bottom is a blue 'Create' button.

Kind

Username with password

Scope

Global (Jenkins, nodes, items, all child items, etc)

Username

xxxxxxxxxxxxxxxxxxxxxx

Treat username as secret

Password

.....

ID

docker

Description

docker

Create

TMDB_V3_API_KEY

Step 1: Add the Secret in Jenkins

1. Go to **Jenkins Dashboard** → **Manage Jenkins** → **Manage Credentials**.
2. Select your **credential domain** (e.g., **(global)**).
3. Click **Add Credentials**.
4. Set:
 - **Kind:** **Secret text**
 - **Secret:** **your_actual_api_key**
 - **ID:** **TMDB_V3_API_KEY**
 - (Optional) Description: **TMDB_V3_API_KEY**
5. Click **OK**.

Step 2: Use the Secret in the Pipeline

Update your **Docker Build & Push** stage like this:

```
stage("Docker Build & Push") {  
    steps {  
        script {  
            withCredentials([string(credentialsId: 'TMDB_V3_API_KEY', variable: 'T  
                withDockerRegistry(credentialsId: 'docker', toolName: 'docker') {  
                    sh "docker build --build-arg TMDB_V3_API_KEY=$TMDB_API -t ne  
                    sh "docker tag netflix eduardobautistamaci/netflix:latest"  
                    sh "docker push eduardobautistamaci/netflix:latest"  
                }  
            }  
        }  
    }  
}
```

Step 4: Configure Jenkins System

In Jenkins go to: **Manage Jenkins → System**

SonarQube

1. Scroll to **SonarQube servers**
2. Click **Add SonarQube**

- Name: `sonar-server`
- SonarQube URL: `http://<netflix-jenkins-elastic IP>:9000`
- Token: Select `sonar-token`

The screenshot shows the Jenkins configuration interface. The URL is `54.89.65.121:8080/manage/configure`. The navigation path is `Dashboard > Manage Jenkins > System >`. On the left, there are links for `Environment variables` and `Tool Locations`. The main content area is titled `SonarQube servers`. It contains a note: "If checked, job administrators will be able to inject a **SonarQube** server configuration as environment variables in the build." There is a checkbox for "Environment variables". Below it, under "SonarQube installations", there is a table with one row:

Name	Server URL	Server authentication token
sonar-server	<code>http://54.89.65.121:9000</code>	<code>sonar-token</code>

At the bottom of the table, there is a "Advanced" dropdown menu.

Step 5: Configure Global Tools

In Jenkins go to: **Manage Jenkins → Tools**

1. **Java:** Add JDK (version 17)

JDK installations

Add JDK

JDK

Name

jdk17

Install automatically ?

Install from adoptium.net ?

Version ?

jdk-17.0.8.1+1 ▾

Add Installer ▾

2. NodeJS: Install NodeJS (version 16)

NodeJS installations

Add NodeJS

☰ NodeJS

Name

node16

Install automatically ?

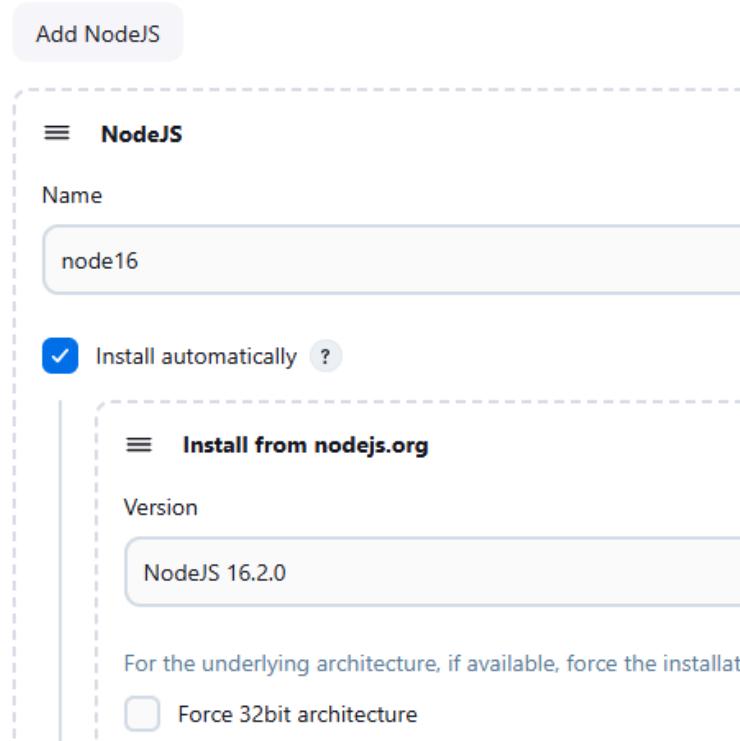
☰ Install from nodejs.org

Version

NodeJS 16.2.0

For the underlying architecture, if available, force the installati

Force 32bit architecture



3. **SonarQube Scanner:** Add name `sonar-scanner` and install automatically

SonarQube Scanner installations

Add SonarQube Scanner

SonarQube Scanner

Name

sonar-scanner

Install automatically ?

Install from Maven Central

Version

SonarQube Scanner 7.1.0.4889

Add Installer ▾

4. Dependency-Check: Add installation

Add Dependency-Check

Dependency-Check

Name

DP-Check

Install automatically ?

Install from github.com

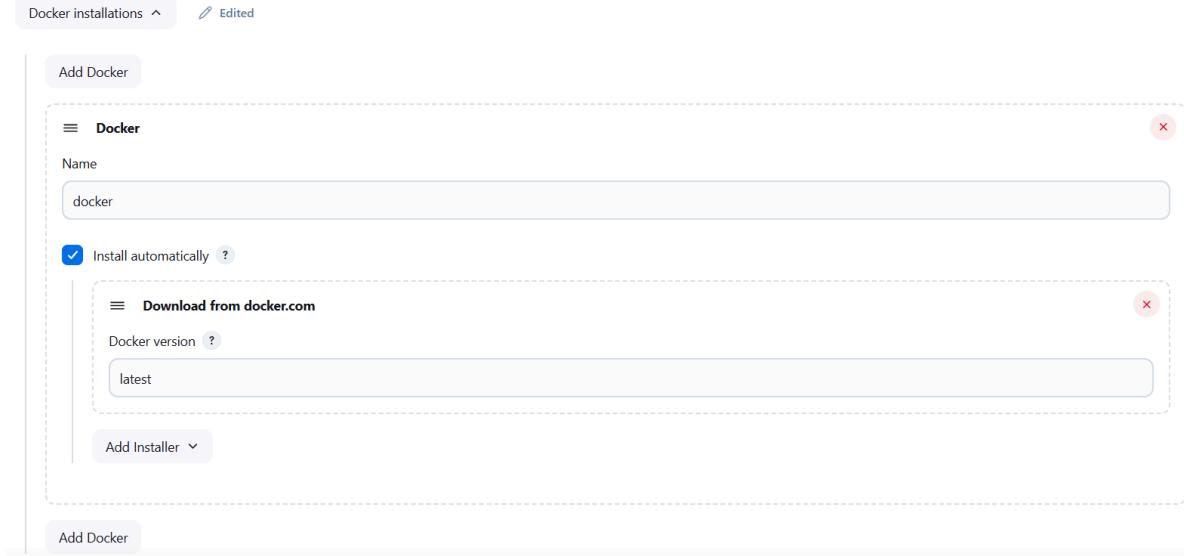
Version

dependency-check 12.1.1

Add Installer ▾

Add Dependency-Check

5. Docker: Add Docker installation path



Click Apply and Save

Step 6: Configure E-mail Notification

Gmail SMTP (Optional - Email Notification)

1. Enable 2FA in Gmail

- Click on profile picture on the right → "Manage your google account"
- Navigate to **Security**
- Under "**Signing in to Google**", enable **2-Step Verification**

2. Create "App password" for **Third-party apps**

- Go to this URL: <https://myaccount.google.com/apppasswords>
- App name: Netflix
- Click on Create
- Copy the password

2. In Jenkins go to: **Manage Jenkins** → **Credentials** → **System** → **Global credentials (unrestricted)** → Add App Password as Secret Text

Kind

Username with password

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

Username ?

eduardobautista.devops@gmail.com

Treat username as secret ?

Password ?

.....

ID ?

mail

Description ?

mail



E-mail Notification

In Jenkins go to: **Manage Jenkins → System**

1. Scroll to **E-mail Notification**
2. SMTP server: smtp.gmail.com
3. Port: [465](#), SSL enabled
4. User: your Gmail address
5. Password: Gmail App Password
6. Test the configuration

E-mail Notification

SMTP server
smtp.gmail.com

Default user e-mail suffix
xxxxxx@gmail.com

● This field should be ' '@ followed by a domain name.

Advanced ▾ Edited

Use SMTP Authentication ?
User Name
xxxxxxxxxx@gmail.com

⚠ For security when using authentication it is recommended to enable either TLS or SSL.

Password

Use SSL ?
 Use TLS

SMTP Port
465

Reply-To Address

Charset
UTF-8

Test configuration by sending test e-mail
Test e-mail recipient
xxxxxxxxxx@gmail.com

Email was successfully sent

➡ Test configuration

7. A test email from Jenkins should arrive in the email inbox.

Extended Email Notification

- Same SMTP config as above

Extended E-mail Notification

SMTP server
smtp.gmail.com

SMTP Port
465

Advanced ▾ Edited

Credentials
eduardobautista.devops@gmail.com/***** (mail)

+ Add

Use SSL
 Use TLS
 Use OAuth 2.0

Advanced Email Properties

- Content type: HTML (text/html)
- Triggers: Always, Failure - Any

Default Triggers ^

Default Triggers ?

- Aborted
- Always
- Before Build
- Failure - 1st
- Failure - 2nd
- Failure - Any
- Failure - Still
- Failure - X
- Failure -> Unstable (Test Failures)
- Fixed
- Not Built
- Script - After Build
- Script - Before Build
- Status Changed
- Success
- Test Improvement
- Test Regression
- Unstable (Test Failures)
- Unstable (Test Failures) - 1st
- Unstable (Test Failures) - Still
- Unstable (Test Failures)/Failure -> Success

Click Apply and Save

Step 7: Create project “Netflix” for Sonarqube Analysis

1. Access SonarQube via: <http://<netflix-jenkins-elastic IP>:9000>
 2. In SonarQube: **Projects → Manually**
 3. Project display name: Netflix
 4. Click on: Set up
 5. In SonarQube: **Projects → Locally → Generate**
-

Step 8: Create Pipeline Job

1. Go to Jenkins Dashboard → **New Item**
2. Enter name: `netflix`, choose **Pipeline**, click OK

The screenshot shows the Jenkins 'New Item' creation interface. At the top, there's a navigation bar with the Jenkins logo and the word 'Jenkins'. Below it, a breadcrumb navigation shows 'Dashboard > All > New Item'. The main area is titled 'New Item'. A search bar is present with the text 'Enter an item name' and the value 'netflix'. Below the search bar, there's a section titled 'Select an item type' with three options:

- Freestyle project**: Described as a classic, general-purpose job type.
- Pipeline**: Described as orchestrating long-running activities across multiple build agents.
- Multi-configuration project**: Described as suitable for projects with many configurations, like testing on multiple platforms.

The 'Pipeline' option is currently selected, indicated by a blue border around its card.

3. In the configuration:
 - Scroll to **Pipeline** section
 - Paste your `Jenkinsfile` script

Dashboard > netflix > Configuration

Configure

General

Triggers

Pipeline

Advanced

Pipeline

Define your Pipeline using Groovy directly or pull it from source control.

Definition

Pipeline script

```
Script ?  
1 v pipeline {  
2     agent any  
3 v     tools {  
4         jdk 'jdk17'  
5         nodejs 'node16'  
6     }  
7 v     environment {  
8         SCANNER_HOME = tool 'sonar-scanner'  
9     }  
10 v    stages {  
11 v        stage('clean workspace') {  
12 v            steps {  
13                 cleanWs()  
14             }  
15         }  
16     }
```

Use Groovy Sandbox ?

Pipeline Syntax

Advanced

Advanced ▾

Save Apply

Complete Jenkins script:

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

```

stages {
    stage('Clean Workspace') {
        steps {
            cleanWs()
        }
    }
    stage('Checkout from Git') {
        steps {
            git branch: 'manual_project', url: 'https://github.com/eduardobautista/Netflix-Clone'
        }
    }
    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'
            dependencyCheckPublisher pattern: '**/dependency-check-report.html'
        }
    }
}

```

```

    }
    stage('Trivy FS Scan') {
        steps {
            sh "trivy fs . > trivyfs.txt"
        }
    }
    stage("Docker Build & Push") {
        steps {
            script {
                withCredentials([string(credentialsId: 'TMDB_V3_API_KEY', variab
                    withDockerRegistry(credentialsId: 'docker', toolName: 'docker')
                    sh "docker build --build-arg TMDB_V3_API_KEY=$TMDB_API_
                    sh "docker tag netflix eduardobautistamaci/netflix:latest"
                    sh "docker push eduardobautistamaci/netflix:latest"
                }
            }
        }
    }
    stage("Trivy Image Scan") {
        steps {
            sh "trivy image eduardobautistamaci/netflix:latest > trivyimage.txt
        }
    }
/*
stage('Deploy to container') {
    steps {
        sh 'docker run -d -p 8081:80 eduardobautistamaci/netflix:latest'
    }
}
*/
}
post {
    always {
        emailext(
            attachLog: true,

```

```

        subject: "${currentBuild.result}",
        body: """\
Project: ${env.JOB_NAME}<br/>
Build Number: ${env.BUILD_NUMBER}<br/>
URL: ${env.BUILD_URL}<br/>""",
to: 'eduardobautista.devops@gmail.com',
attachmentsPattern: 'trivyfs.txt,trivyimage.txt'

    )
}
}
}

```

4. Click on Apply and Save

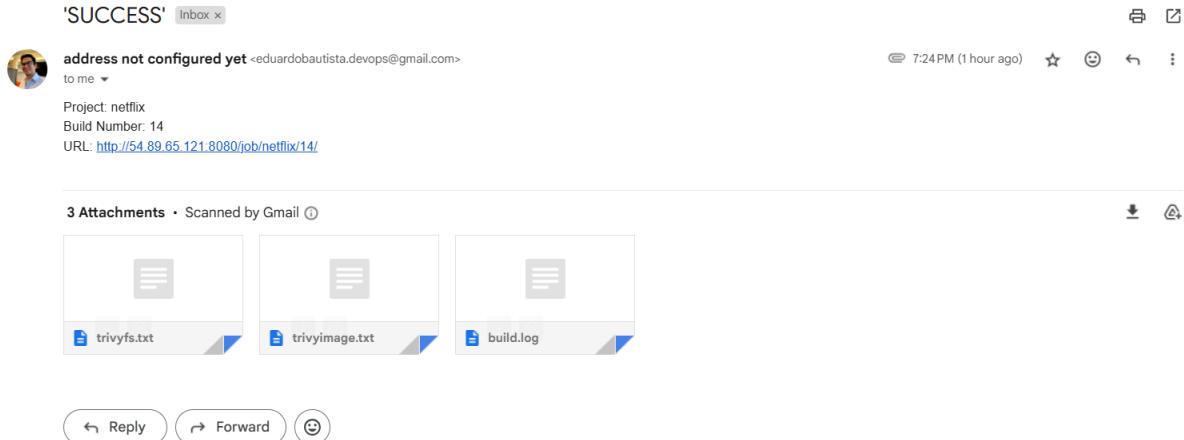
5. Click **Build Now** to run the pipeline

Stage View

	Declarative: Tool Install	Clean Workspace	Checkout from Git	Sonarqube Analysis	Quality Gate	Install Dependencies	OWASP FS Scan	Trivy FS Scan	Docker Build & Push	Trivy Image Scan	Declarative: Post Actions
Average stage times: (full run time: ~5min 51s)											
#19 13:51 No Changes	270ms	414ms	1s	28s	520ms	15s	4min 6s	20s	35s	2s	2s
	246ms	391ms	794ms	25s	330ms	15s	4min 6s	20s	35s	2s	2s

The screenshot shows the Docker Hub interface for the user 'eduardobautistamaciel'. The left sidebar shows the user's profile and navigation options like 'Repositories', 'Settings', 'Default privacy', 'Notifications', 'Billing', and 'Usage'. The main area is titled 'Repositories' and displays a single repository entry:

- Name: eduardobautistamaciel/netflix
- Last Pushed: less than a minute ago
- Type: IMAGE
- Visibility: Public
- Status: Inactive



Phase 4: Monitoring Setup (Prometheus & Grafana)

Step 1: Launch EC2 (Ubuntu 22.04)

1. Create a New EC2 Instance on AWS:

- **Name:** Monitoring
- AMI: Latest Ubuntu 22.04 LTS
- **Instance Type:** t2.medium (2 vCPUs, 4GB RAM)
 - Note: As per the server system requirements for Prometheus the minimum to install Prometheus are:
 - 2 CPU cores
 - 4GB of memory
 - 20 GB of free disk space
- **Key Pair:** Use the same as Jenkins server, else create a new one
- **Storage:** 20 GB gp2
- **Security Group Rules:**
 - Allow SSH (TCP 22) from Anywhere

- Allow HTTPS (TCP 443) from Anywhere
- Allow Grafana (TCP 3000) from Anywhere
- Allow Node Exporter (TCP 9100) from Anywhere
- Allow K8s Monitoring (TCP 8080) from Anywhere
- Allow Prometheus (TCP 9090) from Anywhere
- All outbound traffic allowed

⚠️ Security Tip: For production or team use, restrict SSH access to your IP.

Type	Protocol	Port range	Source	Description
Custom TCP	TCP	3000	0.0.0.0/0	grafana
HTTPS	TCP	443	0.0.0.0/0	-
SSH	TCP	22	0.0.0.0/0	-
Custom TCP	TCP	9100	0.0.0.0/0	nodeexporter
Custom TCP	TCP	8080	0.0.0.0/0	K8s monitoring
Custom TCP	TCP	9090	0.0.0.0/0	prometheus

Step 2: Allocate and Associate Elastic IP

- Create Elastic IP and name it `monitoring-eip`
- Attach it to the newly created instance

Step 3: SSH into the Instance (to install Prometheus, Grafana, Node Explorer)

```
ssh -i <key.pem> ubuntu@<Monitoring-elastic IP>
sudo apt update -y
```

Step 4: Create user Prometheus & Install Prometheus

```
sudo useradd --system --no-create-home --shell /bin/false prometheus
```

```
 wget https://github.com/prometheus/prometheus/releases/download/v2.47.1/pro
```

Make sure the .tar file is there: `ls`

Extract Prometheus files, create directory, and move the:

```
 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
```

Observer, “ubuntu” is the owner of the Prometheus files:

```
 ls -la /etc/prometheus/  
  
 drwxr-xr-x 4 root root 4096 Apr 13 17:15 .  
 drwxr-xr-x 109 root root 4096 Apr 13 17:15 ..  
 drwxr-xr-x 2 ubuntu ubuntu 4096 Oct 4 2023 console_libraries  
 drwxr-xr-x 2 ubuntu ubuntu 4096 Oct 4 2023 consoles  
 -rw-r--r-- 1 ubuntu ubuntu 934 Oct 4 2023 prometheus.yml
```

Set ownership for directories:

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

Now the owner is “prometheus”:

```
 ls -la /etc/prometheus/  
  
 drwxr-xr-x 4 prometheus prometheus 4096 Apr 13 17:15 .  
 drwxr-xr-x 109 root root 4096 Apr 13 17:15 ..  
 drwxr-xr-x 2 prometheus prometheus 4096 Oct 4 2023 console_libraries
```

```
drwxr-xr-x  2 prometheus prometheus 4096 Oct  4  2023 consoles  
-rw-r--r--  1 prometheus prometheus   934 Oct  4  2023 prometheus.yml
```

Create Prometheus Systemd Service:

```
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
```

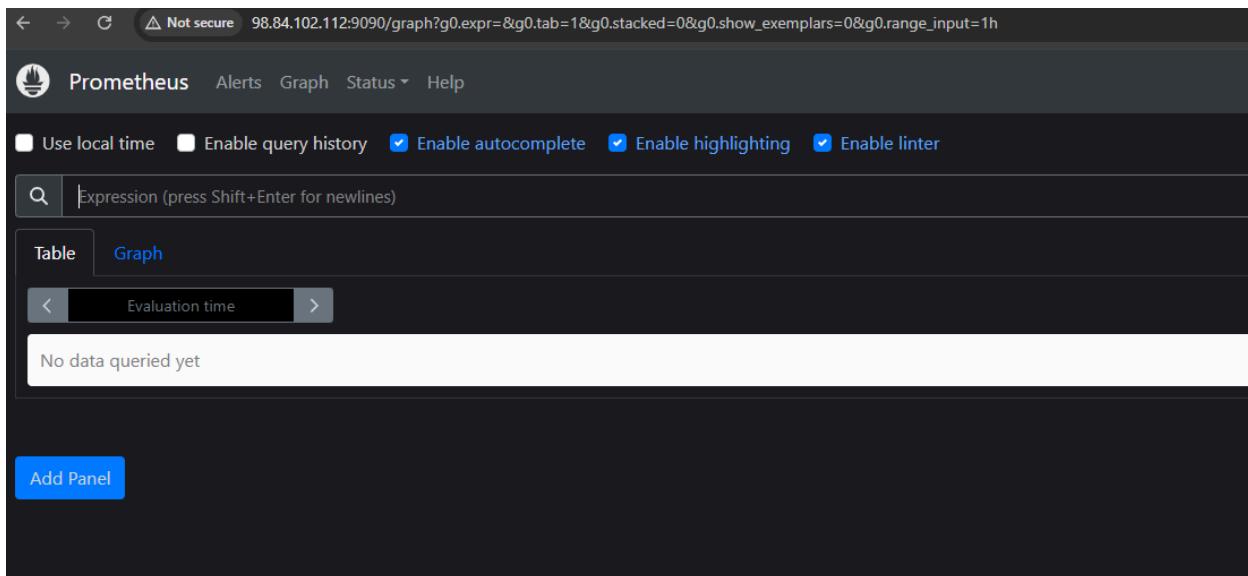
Verify Prometheus's status:

```
sudo systemctl status prometheus
```

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

<http://<Monitoring-elastic IP>:9090>

Example output:



Step 5: Install 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 Node Exporter Systemd Service:

```
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
```

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
```

Configure Prometheus to Scrape Jenkins, and Node Exporter:

```
cd /etc/prometheus/  
sudo nano prometheus.yml
```

Add:

```
global:  
  scrape_interval: 15s  
  
scrape_configs:  
  - job_name: 'node_exporter'  
    static_configs:  
      - targets: ['<Monitoring-elastic IP>:9100']  
  
  - job_name: 'jenkins'  
    metrics_path: '/prometheus'  
    static_configs:  
      - targets: ['<netflix-jenkins-elastic IP>:8080']
```

Validate and Reload (without restarting) Prometheus Configuration:

```
promtool check config /etc/prometheus/prometheus.yml  
curl -X POST http://localhost:9090/-/reload
```

You can access Prometheus targets at:

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

Step 6: Install Grafana on Ubuntu 22.04 and Set it up to Work with Prometheus

1. Install Dependencies:

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

2. Add the Grafana GPG Key:

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

3. Add Grafana Repository:

```
echo "deb https://packages.grafana.com/oss/deb stable main" | sudo tee -a /etc
```

4. Update and Install Grafana:

Update the package list and install Grafana:

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

5. Enable and Start Grafana Service:

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

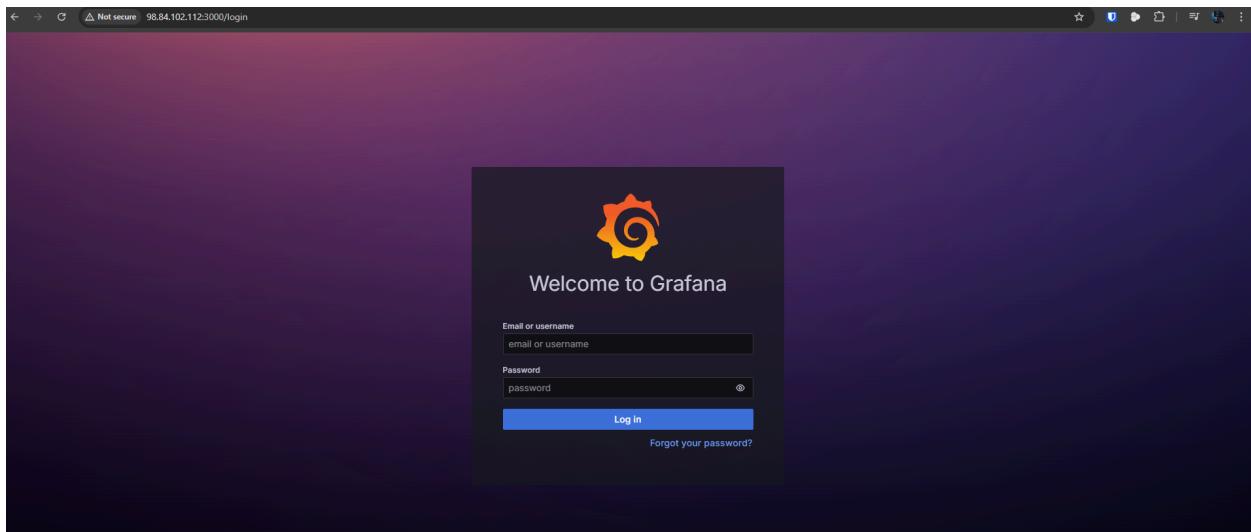
6. Check Grafana Status:

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

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.:

```
http://<Monitoring-elastic IP>:3000
```

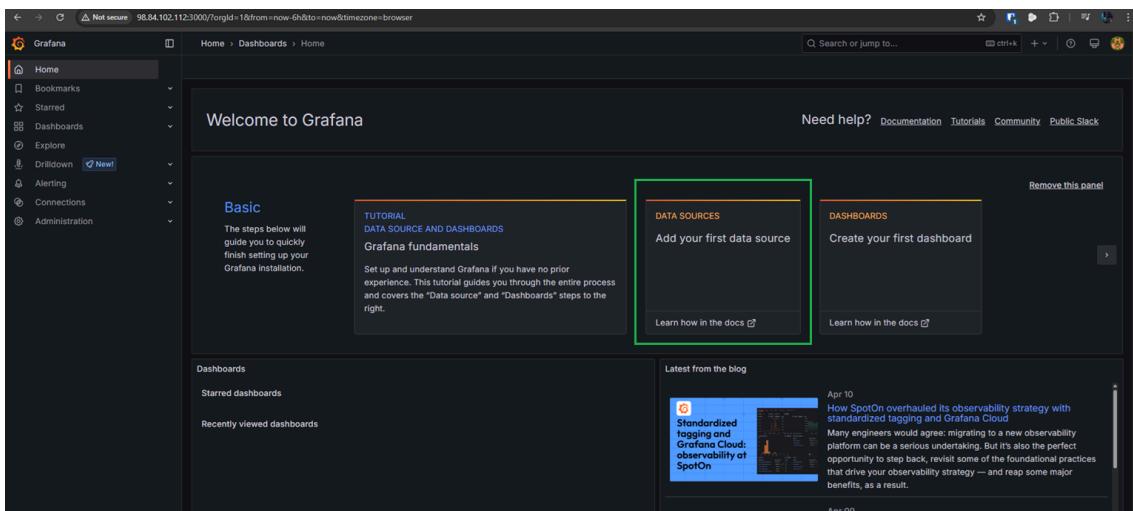


- Default credentials:

- **Username:** admin
 - **Password:** admin
 - You'll be prompted to change the password on first login.

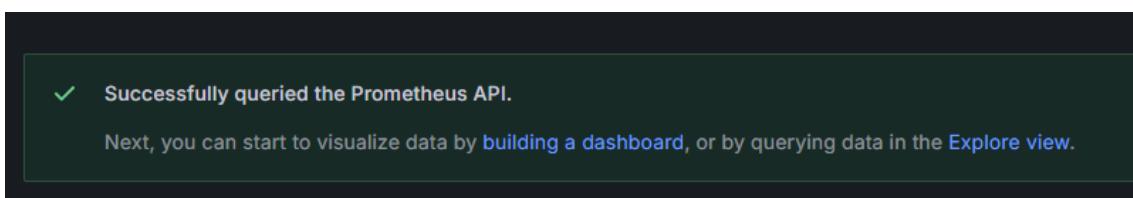
8. Add Prometheus as a Data Source in Grafana

- Navigate to: **Gear Icon (⚙)** → **Data Sources** → **Add data source**

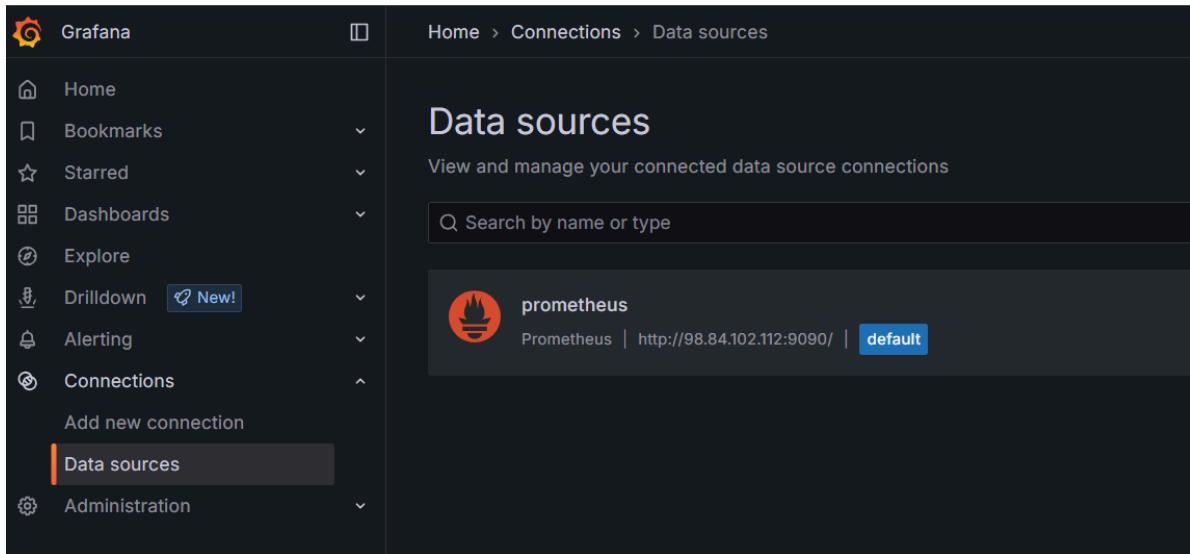


- Choose "Prometheus"
- In the "HTTP" section:
 - Set the "URL" to <http://<Monitoring-elastic IP>:9090>
- Click the "Save & Test" button to ensure the data source is working.

Expected output (a green "Data source is working" message):



You can also confirm here:



9. Import Pre-built Dashboards:

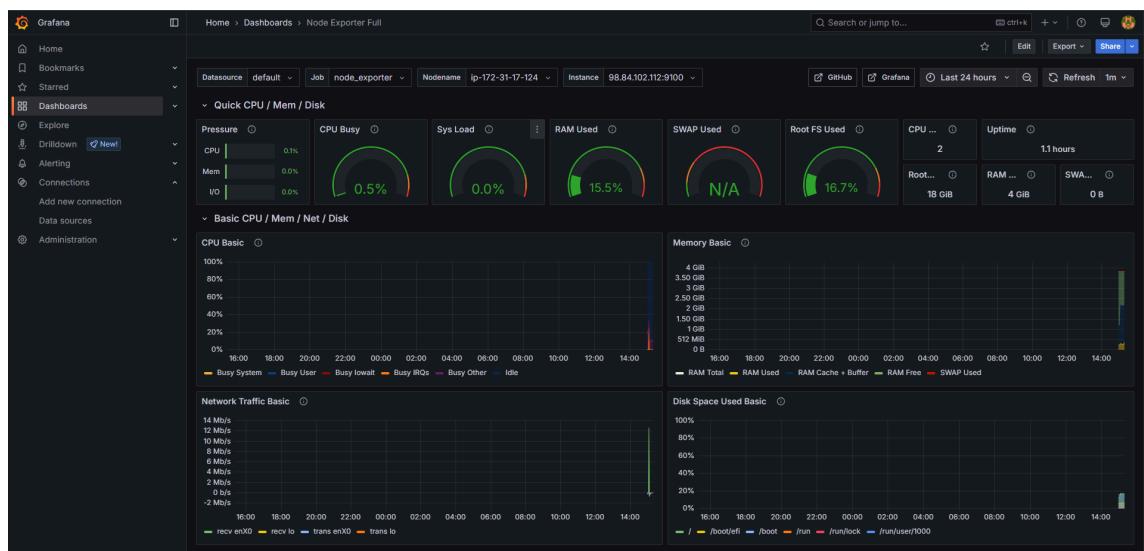
Set the monitoring server Node Exporter grafana dashboard:

- On the left sidebar to open select "Dashboard."
- Go to **+ → Import.**
- Enter a dashboard ID from [Grafana.com Dashboards](#), such as: **1860**
- On the right mid section click on "Copy ID to clipboard"

Nearly all default values exported by Prometheus node exporter graphed.
Only requires the default job_name: node, add as many targets as you need in /etc/prometheus/prometheus.yml!

- Click the "Load" button.
- Select Prometheus as the data source.

- Click **Import**.
- You should now have a Grafana dashboard set up to visualize metrics from Prometheus.
 - We can observe CPU, Memory, RAM from the monitoring server



Set the jenkins grafana dashboard:

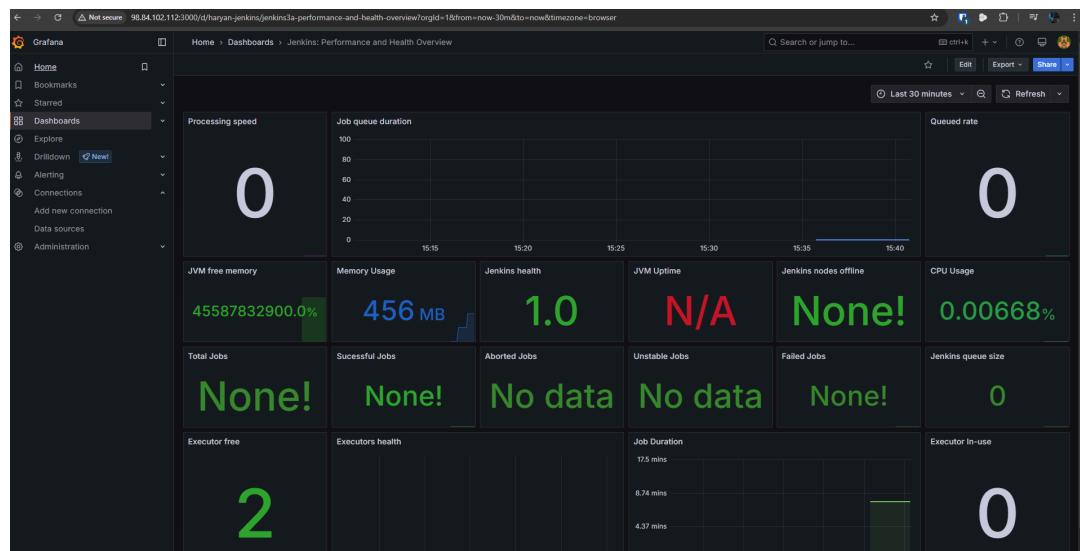
1. Configure Prometheus Plugin Integration in Jenkins:

- Integrate Jenkins with Prometheus to monitor the CI/CD pipeline.
 - Go to: [Manage Jenkins](#) → [Manage Plugins](#) → [Available](#) → Search: [prometheus](#)
 - This plugin exposes an endpoint at </prometheus> on your Jenkins server.
 - Install the **Prometheus metrics** plugin → Then RESTART Jenkins
 - Note: you need to have your Password ready since you will need to log back in
 - This exposes metrics at: <http://<your-jenkins-ip>:8080/prometheus/>

2. In Grafana

- On the left sidebar to open select "Dashboard."
- Go to **+** → **Import**.

- Enter a dashboard ID from [Grafana.com Dashboards](#), such as: [9964](#)
- On the right mid section click on "Copy ID to clipboard"
- Click the "Load" button.
- Select Prometheus as the data source.
- Click **Import**.
- You should now have a Grafana dashboard set up to visualize metrics from Prometheus.
 - We can observe CPU, Memory, RAM from the Jenkins netflix server



Phase 5: Create a Kubernetes Cluster with Node Groups (EKS)

You'll use **Amazon EKS** to create a managed Kubernetes cluster. This enables easy scaling and management of containerized applications.

Step 1: Create IAM Roles

1. Go to the **AWS Console → IAM → Roles**

2. Create Role

- **Trusted entity:** Select *EKS*
- **Use case:** EKS - Cluster
- Attach **AmazonEKSClusterPolicy**
- Trust relationships:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "eks.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

3. Repeat to create a second role:

- **Trusted entity:** EKS
- **Use case:** EKS - Nodegroup
- Attach:
 - [AmazonEKSWorkerNodePolicy](#)
 - [AmazonEKS_CNI_Policy](#)
 - [AmazonEC2ContainerRegistryReadOnly](#)
- Name it: [myAmazonEKSNodeRole](#)

Step 2: Create EKS Cluster

1. Go to **EKS** → **Clusters** → **Create Cluster**

2. Cluster name: `Netflix`
3. Kubernetes version: *default (latest)*
4. Cluster Service Role: `myAmazonEKSClusterRole`

Click **Next**

5. Networking:

- VPC: Choose **default**
- Subnets: **Only select public subnets**
- Security Group: `my-sg` (or create a new one with port 443, 80, 30007 open)

Click **Next** through the remaining steps and click **Create**.

Step 3: Create Node Group

1. After cluster creation → Go to **Compute** → **Add Node Group**
2. Node Group Name: `nodes`
3. Node IAM Role: `myAmazonEKSNodeRole`

Click **Next**

1. Instance Type: `t3.large`
2. Disk size: `20 GiB`
3. Desired, Min, Max size: `1`

Click **Next** until **Create**.

Step 4: Update your local kubeconfig file

(This is usually at `~/.kube/config`) so you can connect `kubectl` to your Amazon EKS (Elastic Kubernetes Service) cluster

Update kubeconfig:

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

Output:

```
Added new context arn:aws:eks:us-east-1:038462748802:cluster/Netflix to /t
```

Phase 6: Monitor Kubernetes with Prometheus

You'll install Prometheus Node Exporter using Helm to collect metrics from your cluster nodes.

Step 1: Install Node Exporter with Helm

Monitor Kubernetes with Prometheus

1. Add the Prometheus Community Helm repository:

```
helm repo add prometheus-community https://prometheus-community.git  
hub.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
```

4. Verify Installation:

```
kubectl ns
```

Example output with "prometheus-node-exporter":

```
default
external-dns
kube-node-lease
kube-public
kube-system
prometheus-node-exporter
```

- Verify pod: `kubectl get pods -n prometheus-node-exporter`

Expected output:

NAME	READY	STATUS	RESTARTS	AGE
prometheus-node-exporter-lsghr	1/1	Running	0	3m57

Step 2: Add Custom Scrape from K8s nodes in Prometheus

Add a Job to Scrape Metrics on `nodeip:9100/metrics` in `prometheus.yml`:

Update your Prometheus configuration (`prometheus.yml`) to add a new job for scraping metrics from `nodeip:9100/metrics`. You can do this by adding the following configuration to your `prometheus.yml` file:

```
cd /etc/prometheus/
sudo nano prometheus.yml
```

```
- job_name: 'K8s'
  metrics_path: '/metrics'
  static_configs:
    - targets: ['<nodes-ip>:9100']
```

Replace `<Your-Node-IP>` with your actual node IP.

Validate and Reload (without restarting) Prometheus Configuration:

```
promtool check config /etc/prometheus/prometheus.yml  
curl -X POST http://localhost:9090/-/reload
```

You can access Prometheus targets at:

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

The screenshot shows the Prometheus Targets page with four sections: K8s (1/1 up), jenkins (1/1 up), node_exporter (1/1 up), and prometheus (1/1 up). Each section contains a table with columns: Endpoint, State, Labels, Last Scrape, Scrape Duration, and Error. The 'K8s' section has one entry: 'http://44.222.216.79:9100/metrics' with labels 'instance="44.222.216.79:9100"' and 'job="k8s"'. The 'jenkins' section has one entry: 'http://54.89.65.121:8080/prometheus' with labels 'instance="54.89.65.121:8080"' and 'job="jenkins"'. The 'node_exporter' section has one entry: 'http://98.84.102.112:9100/metrics' with labels 'instance="98.84.102.112:9100"' and 'job="node_exporter"'. The 'prometheus' section has one entry: 'http://localhost:9090/metrics' with labels 'instance="localhost:9090"' and 'job="prometheus"'. All targets are marked as 'UP'.

Phase 7: Deploy Application with ArgoCD

Fetches the manifests files from the Kubernetes folder from the GitHub Repo

Step 1: Install ArgoCD

1. **Install ArgoCD:** (You can install ArgoCD on your Kubernetes cluster by following the instructions provided in the [EKS Workshop](#) documentation.)

```
kubectl create namespace argocd  
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-
```

3. Verify Namespace "argocd" :

```
kubectl get ns
```

Expected output:

NAME	STATUS	AGE
argocd	Active	112s
default	Active	37m
external-dns	Active	35m
kube-node-lease	Active	37m
kube-public	Active	37m
kube-system	Active	37m

4. Check all pods for argocd:

```
kubectl get all -n argocd
```

Expected output:

NAME	READY	STATUS	RESTARTS	AGE
pod/argocd-application-controller-0	1/1	Running	0	2n
pod/argocd-applicationset-controller-8477cf5d6d-bbb2w	1/1	Running	0	
pod/argocd-dex-server-575bc85fc7-ffgw7	1/1	Running	2 (2m3s)	
pod/argocd-notifications-controller-5fff6cd8b7-8whdp	1/1	Running	0	
pod/argocd-redis-dd9464c7c-28krr	1/1	Running	0	
pod/argocd-repo-server-58d7f66d97-f66pl	1/1	Running	0	
pod/argocd-server-c9596d75c-cpt6j	1/1	Running	0	

5. Expose argocd-server:

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

- Wait about 2 minutes for the LoadBalancer creation

6. Get LoadBalancer DNS:

```
export ARGOCD_SERVER=`kubectl get svc argocd-server -n argocd -o json | echo $ARGOCD_SERVER
```

Example output:

```
a46d0f9686cbf467f973e27f0c64dac8-881462586.us-east-1.elb.amazonaws.com
```

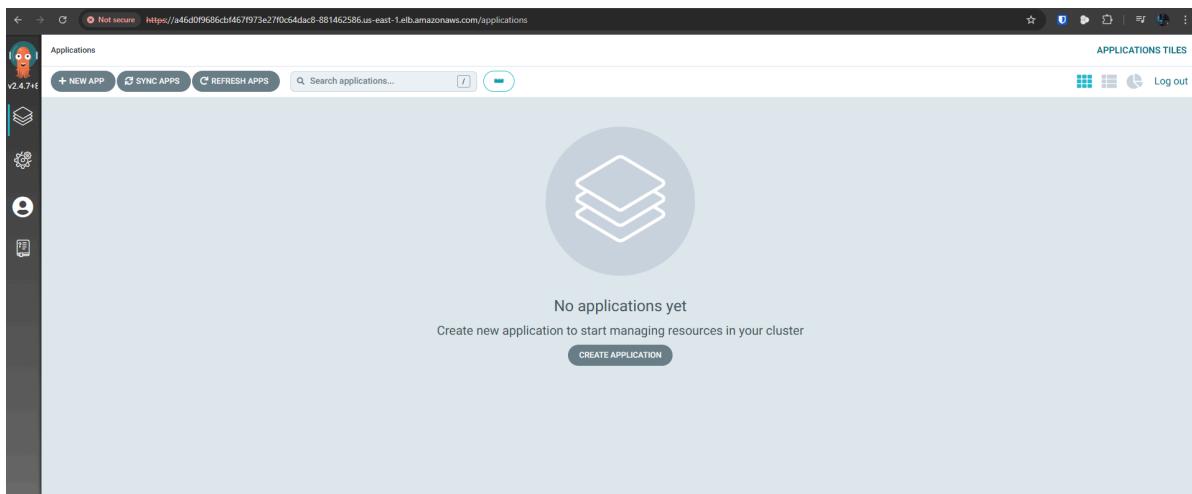
7. Get ArgoCD Admin Password:

```
export ARGO_PWD=`kubectl -n argocd get secret argocd-initial-admin-secret -o yaml | grep 'stringData' | awk '{print $2}' | base64 -d`  
echo $ARGO_PWD
```

8. Access ArgoCD:

- Visit [http://\\$ARGOCD_SERVER](http://$ARGOCD_SERVER)
- Username: `admin`
- Password: (output of `$ARGO_PWD`)

Expected Successful login:

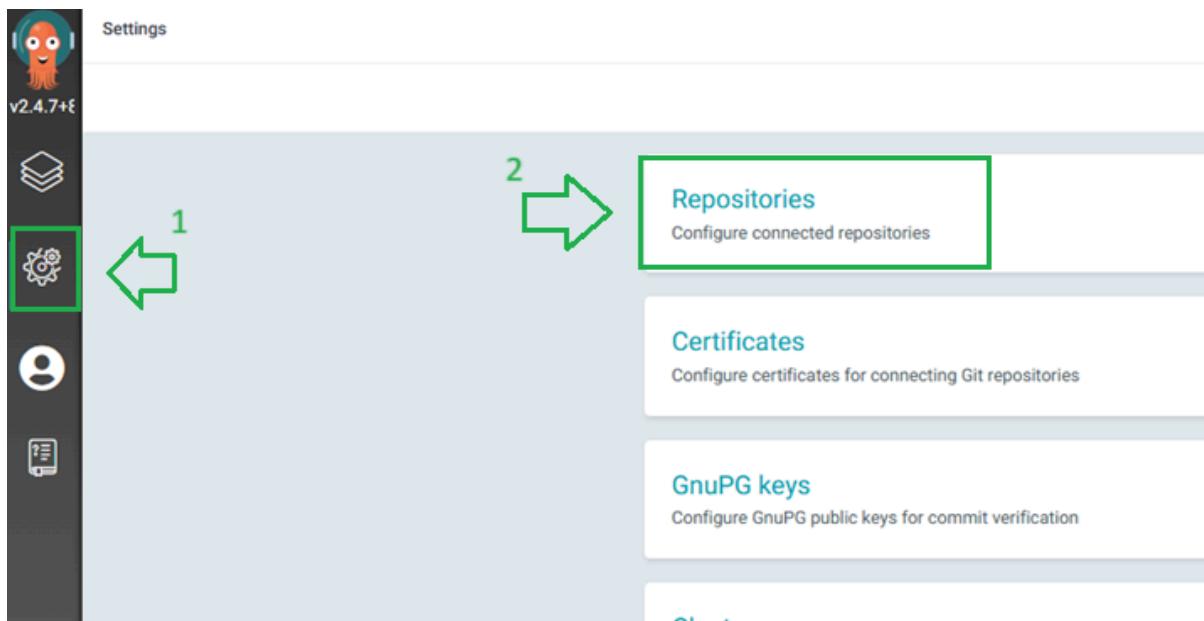


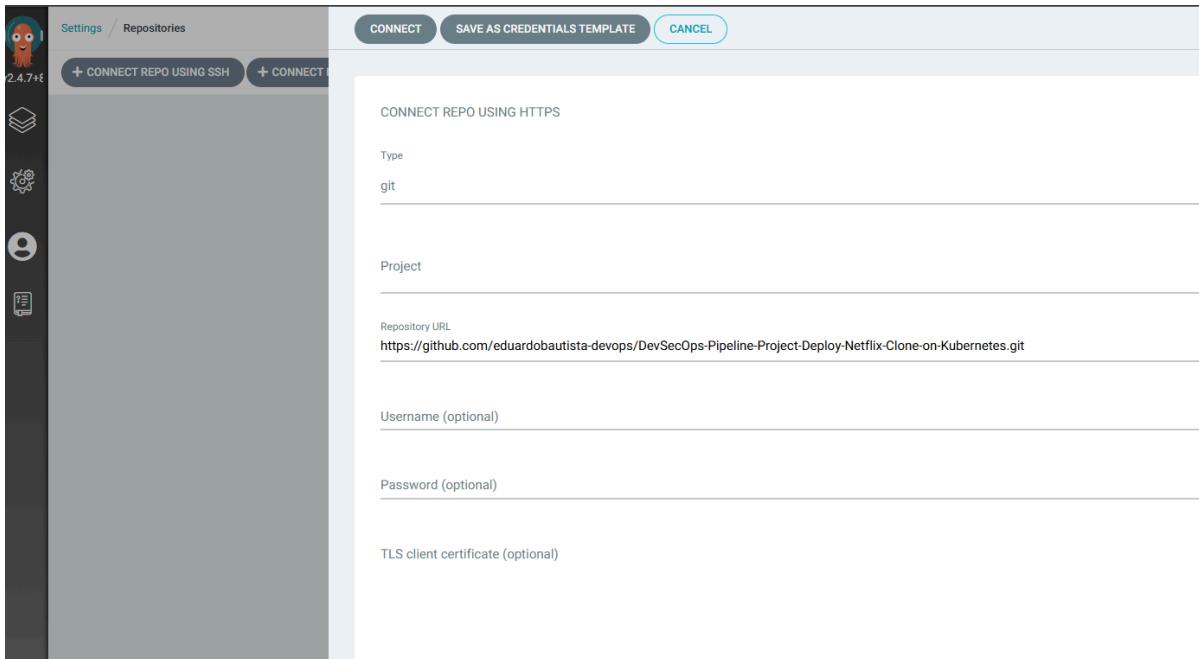
In the ArgoCD UI:

- Settings → Repositories → "CONNECT REPO USING HTTPS"

- Project: default
- URL:
<https://github.com/eduardobautista-devops/DevSecOps-Pipeline-Project-Deploy-Network-Clone-on-Kubernetes.git>

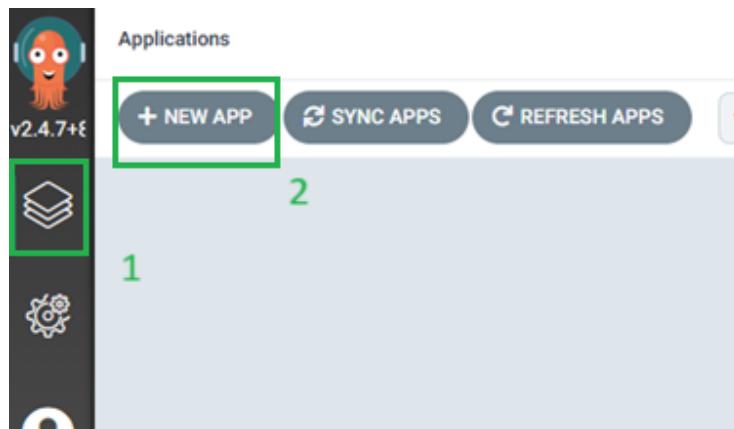
Step 2: Connect Your GitHub Repo





- Then click on CONNECT

Step 3: Create ArgoCD Application



1. **Application Name:** `netflix`

2. **Project:** `default`

3. **Source Repo:**

- URL: <https://github.com/eduardobautista-devops/DevSecOps-Pipeline-Project-Deploy-Network-Clone-on-Kubernetes.git>
- Revision: `HEAD` or branch name
- Path: `Kubernetes`

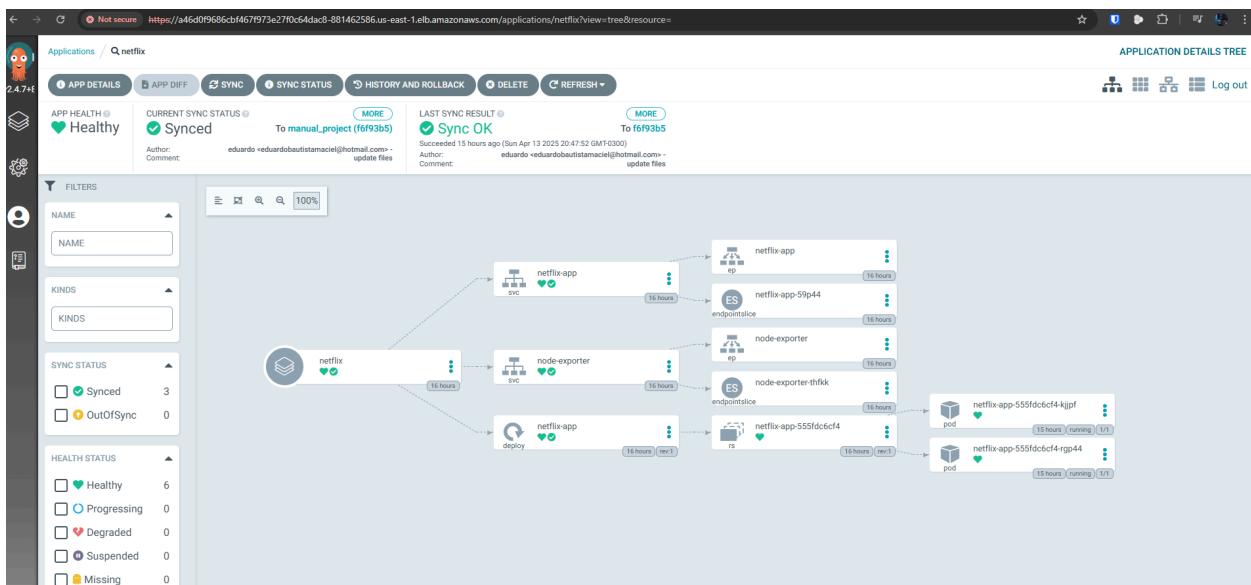
4. Destination:

- Cluster URL: <https://kubernetes.default.svc>
- Namespace: `default`

Click **Save**

Step 4: Sync Application

Click "SYNC" → FORCE → SYNCHRONIZE



Step 5: Access Netflix App

1. Go to:

- EKS → Compute → Nodes → Instance → Security Group

2. Add Inbound Rule: Port **30007**, Type: **Custom TCP**

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-08a2ce0a323ad2644	All traffic	All	All	Custom	sg-e003679912a2ac31
sgr-0bd148d4d2ff60fd4	All traffic	All	All	Custom	sg-0db09bce5963d9f98
-	Custom TCP	TCP	30007	Anywhere	app nodeport 0.0.0.0/0

Add rule

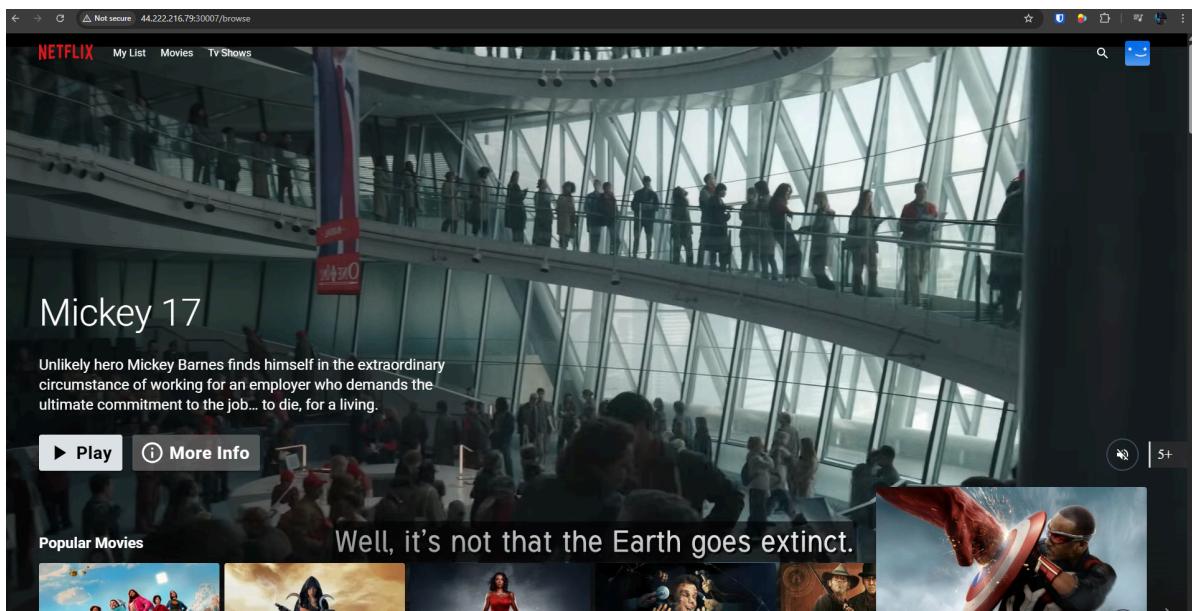
⚠ Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Cancel **Preview changes** **Save rules**

3. Get Public IP of your Node

4. Access app at:

http://<Node-Public-IP>:30007



Phase 8: Cleanup

1. Terminate the node group from the EKS cluster → compute → Node group
2. Go to EC2 → Load balancers, delete the AgroCD LB
3. Cleanup AWS EC2 Instances:
 - Terminate AWS EC2 instances that are no longer needed, the Jenkins and Monitoring
 - Delete elastic IP
 - Select the IP not using → click on Actions → Dissociate Elastic IP address
 - After all EPs were dissociated, then you can Release Elastic IP
4. Delete EKS cluster

Troubleshoot

If you get docker login failed error, for example:

```
Downloading Docker client latest
Unpacking https://get.docker.com/builds/Linux/x86_64/docker-latest.tgz to /var/
$ /var/lib/jenkins/tools/org.jenkinsci.plugins.docker.commons.tools.DockerTool/c
Warning: failed to get default registry endpoint from daemon (Got permission de
Got permission denied while trying to connect to the Docker daemon socket at u
```

Solution:

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

- This Append the user to the `docker` group, which has permission to access the Docker socket.