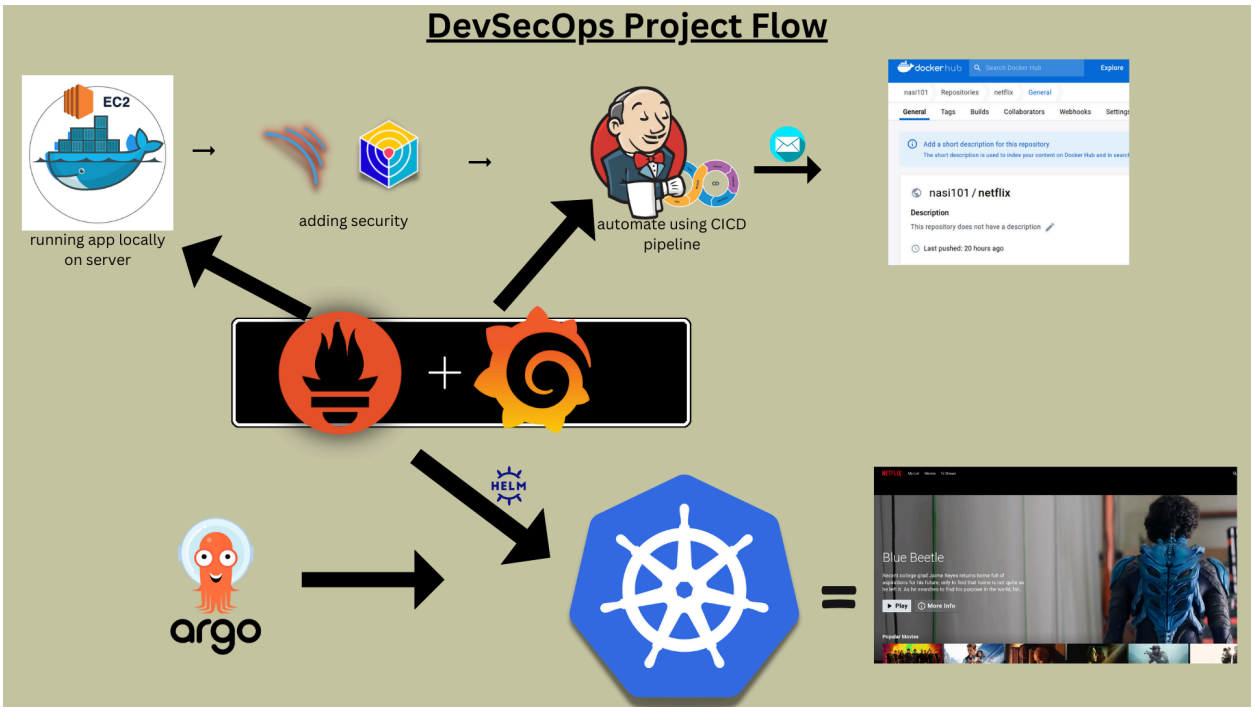
DevSecOps Pipeline Project: Deploy Netflix Clone using Jenkins on Kubernetes

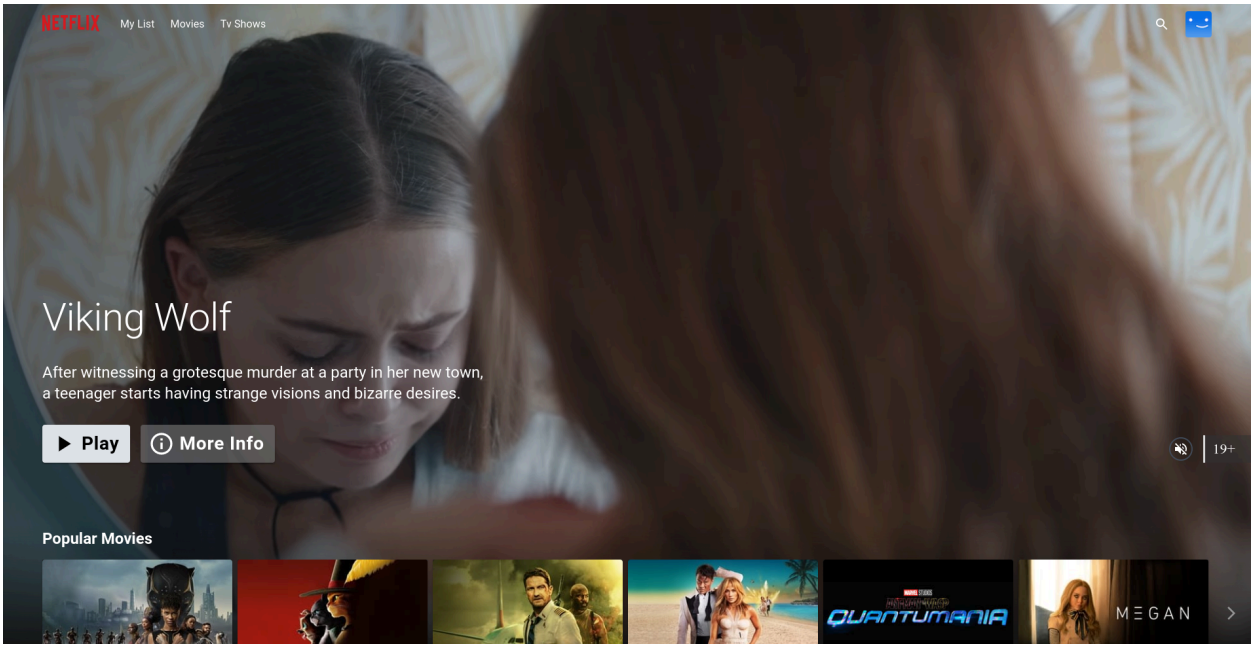
Project Description

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





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-Netflix-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
```

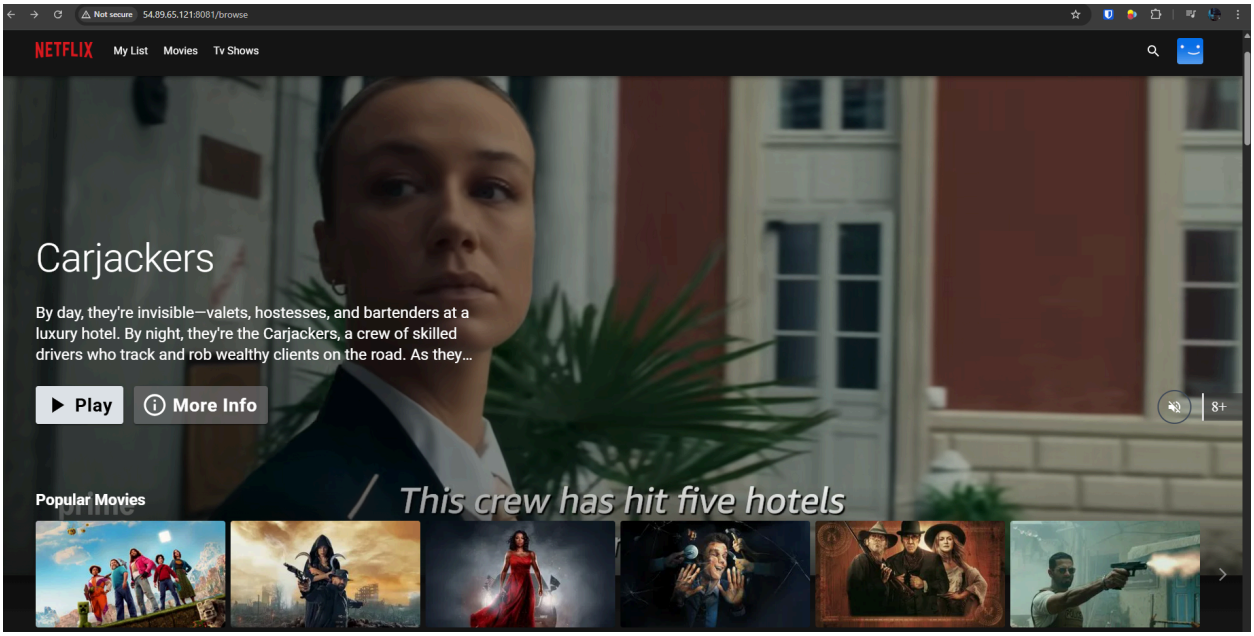
```
# Give full permissions to docker.sock (its communication with Docker on your machine)
sudo chmod 777 /var/run/docker.sock
```

- 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:lts-community
docker ps
```

Example output:

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

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
```

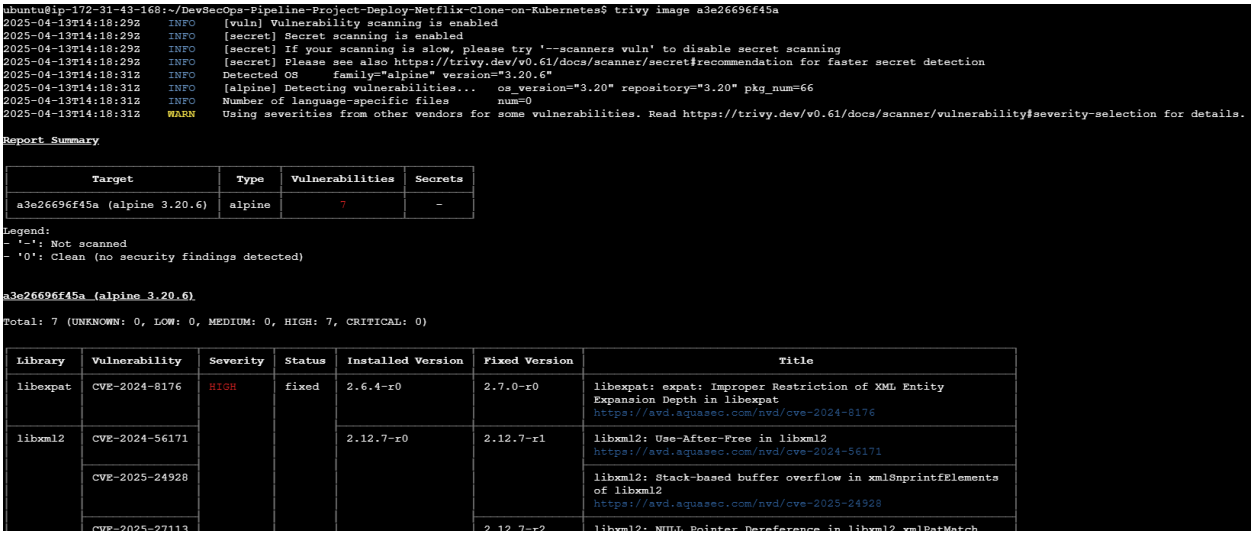
Verify installation:

```
trivy version
```

Scan image:

```
trivy image <imageid>
```

Example output:



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, sh
aring)
#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

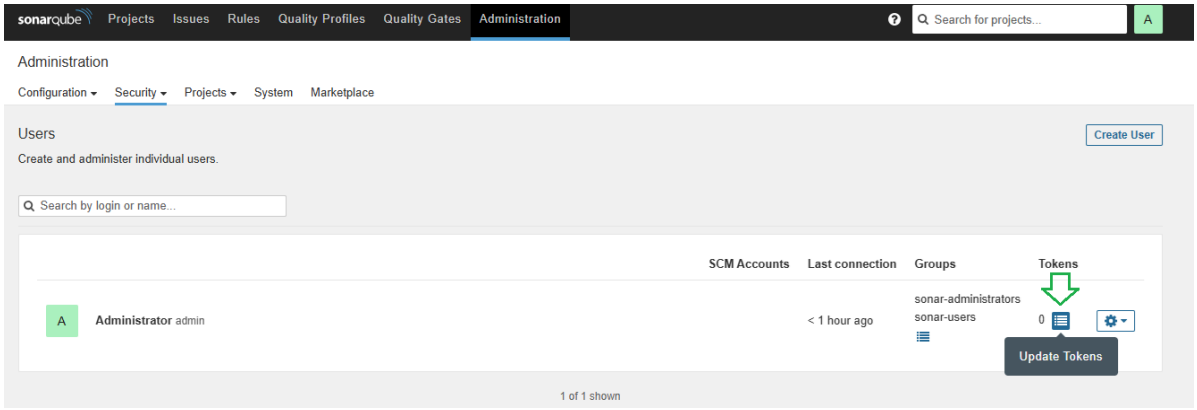
- OWASP Dependency-Check
- Docker
- Docker Commons
- Docker Pipeline
- Docker API
- docker-build-step

3. Click "Install without restart”

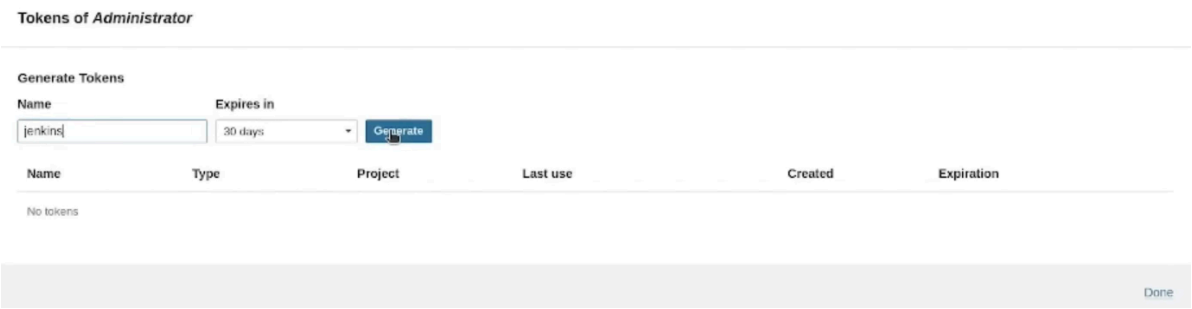
Step 3: Add Credentials

SonarQube Token

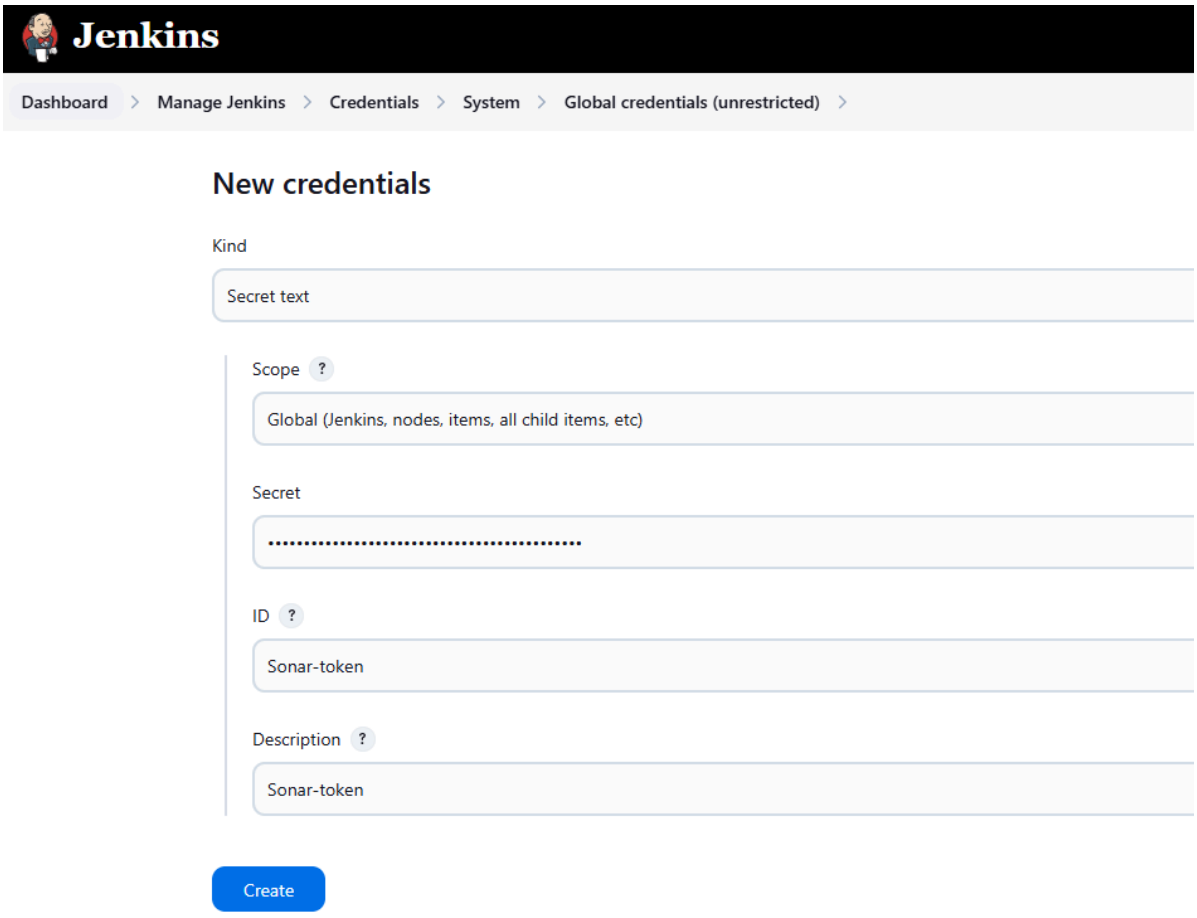
1. Access SonarQube via: `http://<netflix-jenkins-elastic IP>:9000`
2. In SonarQube: **Administration → Security → Users → Update Tokens**



2. Generate and copy a new token

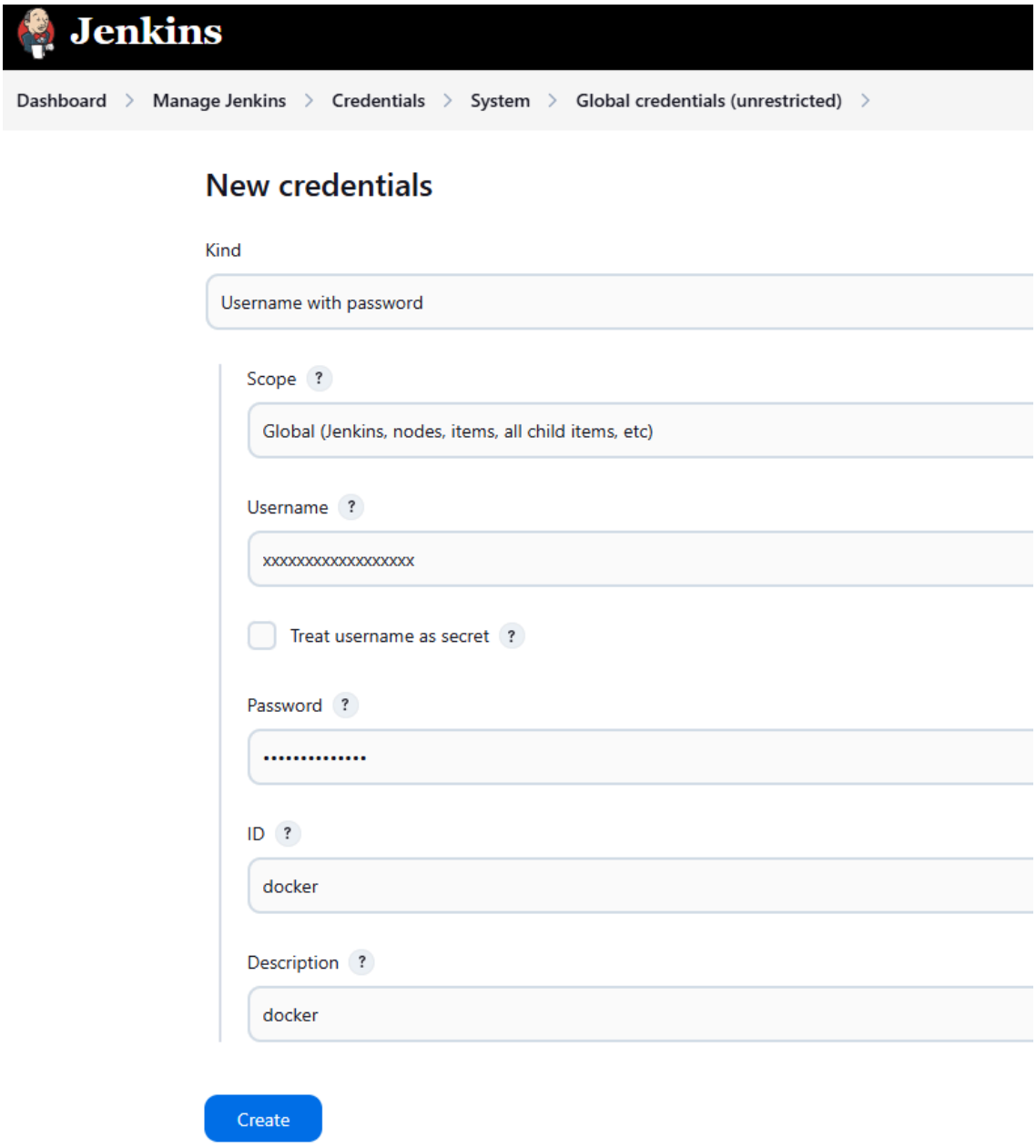


3. In Jenkins go to: **Manage Jenkins → Credentials → System → Global credentials (unrestricted)**
    - Type: Secret Text
    - Secret: (paste token)
    - ID: `sonar-token`



DockerHub

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



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:

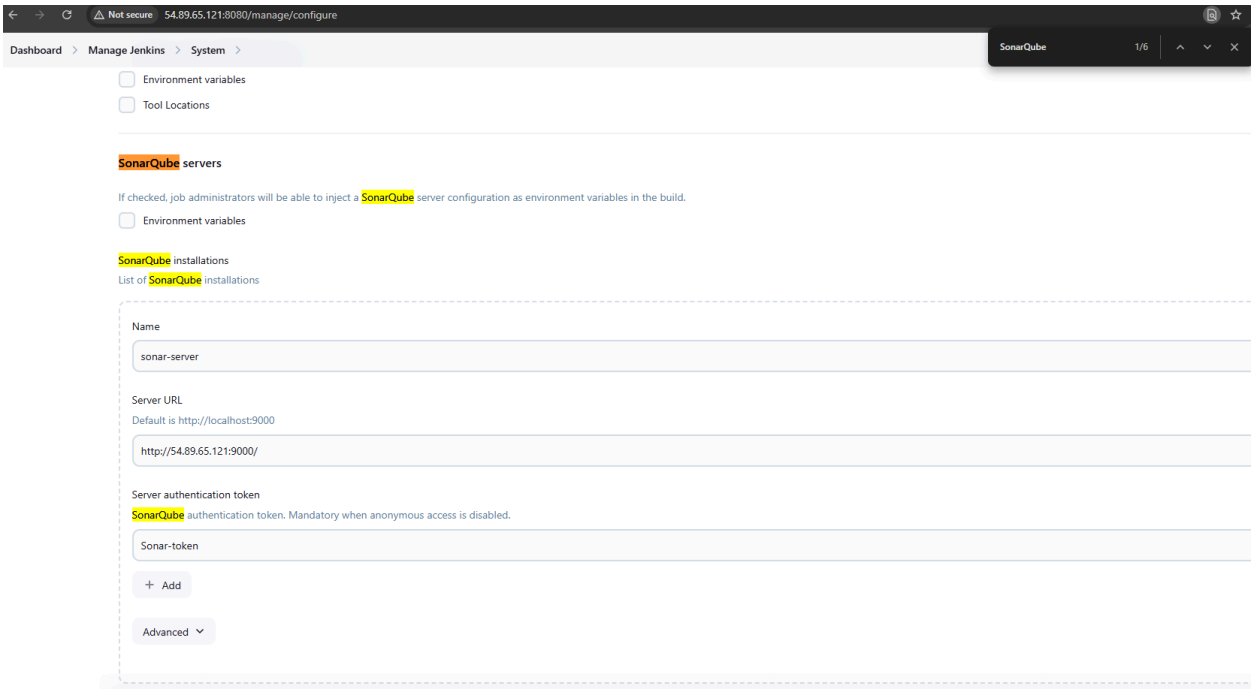
```
stage("Docker Build & Push") {
    steps {
        script {
            withCredentials([string(credentialsId: 'TMDB_V3_API_KEY', variable: 'T
                withDockerRegistry(credentialsId: 'docker', toolName: 'docker') {
                    sh "docker build --build-arg TMDB_V3_API_KEY=$TMDB_API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

Step 4: Configure Jenkins System

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

SonarQube

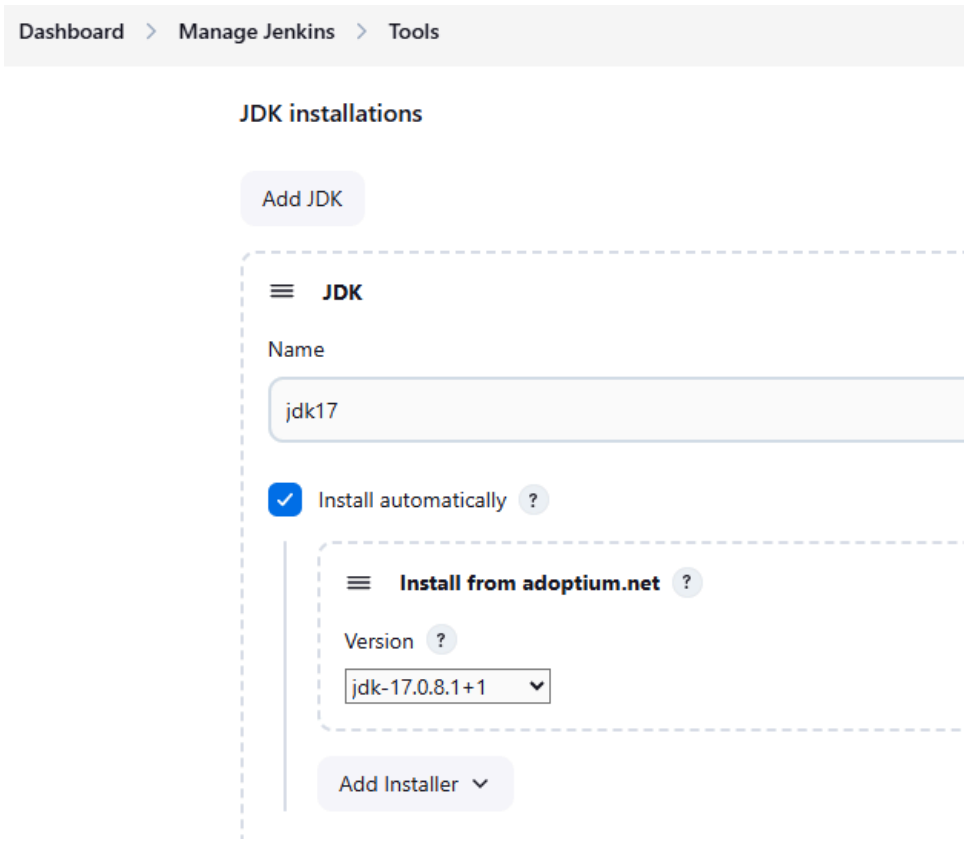
1. Scroll to **SonarQube servers**
2. Click **Add SonarQube**
    - Name: `sonar-server`
    - SonarQube URL: `http://<netflix-jenkins-elastic IP>:9000`
    - Token: Select `sonar-token`



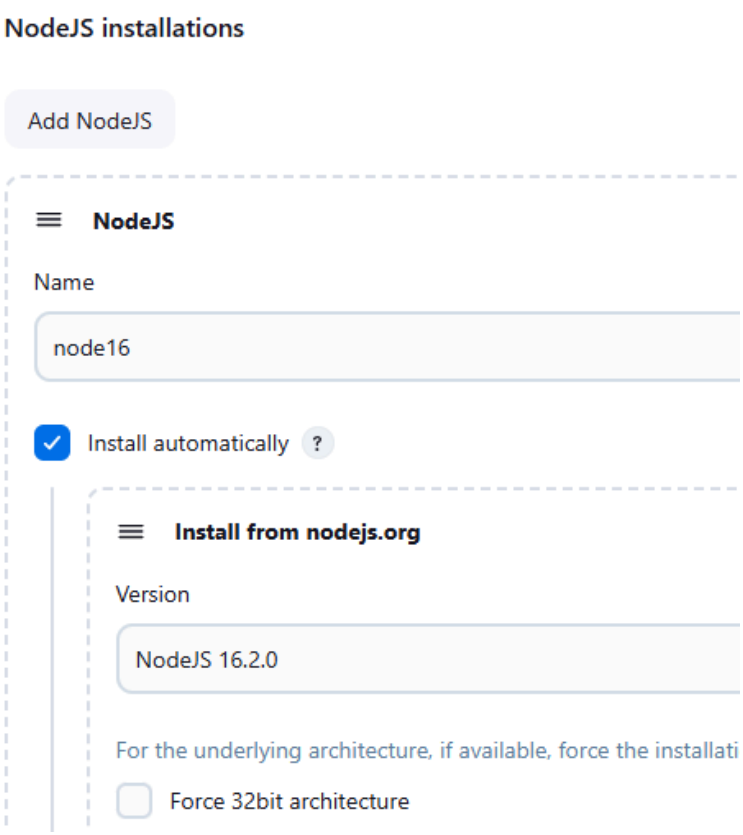
Step 5: Configure Global Tools

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

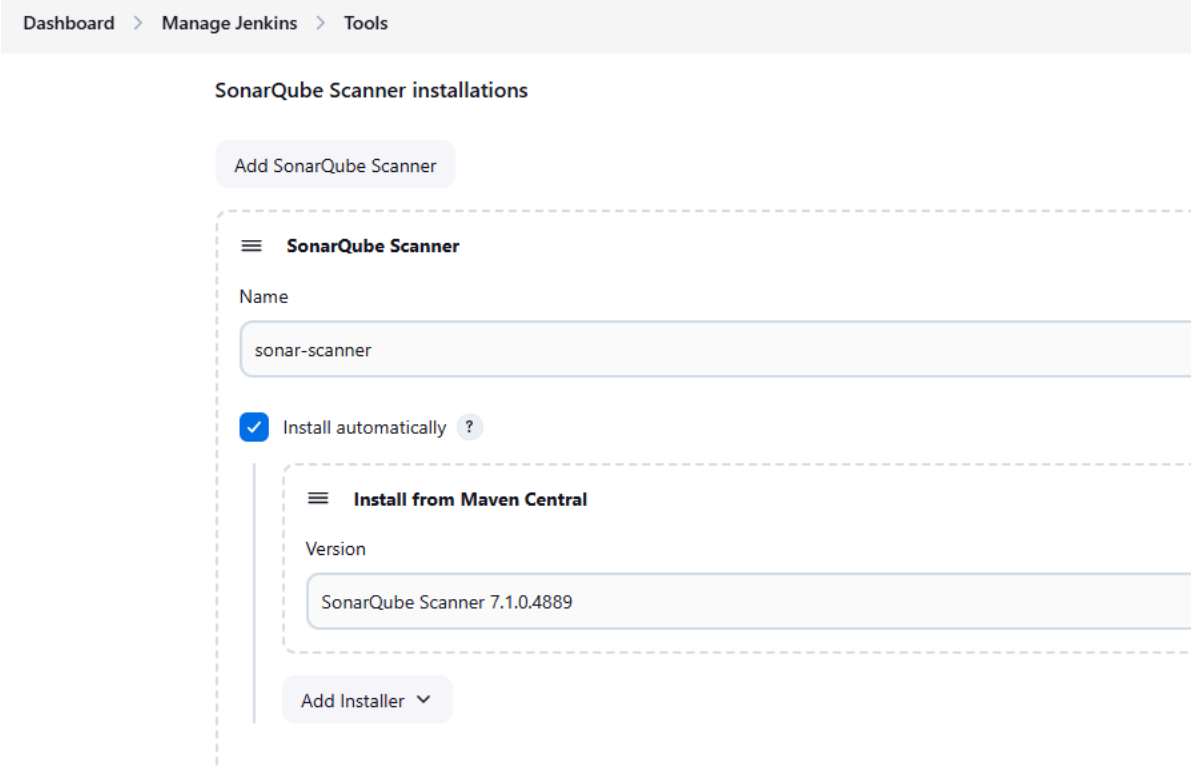
1. **Java:** Add JDK (version 17)



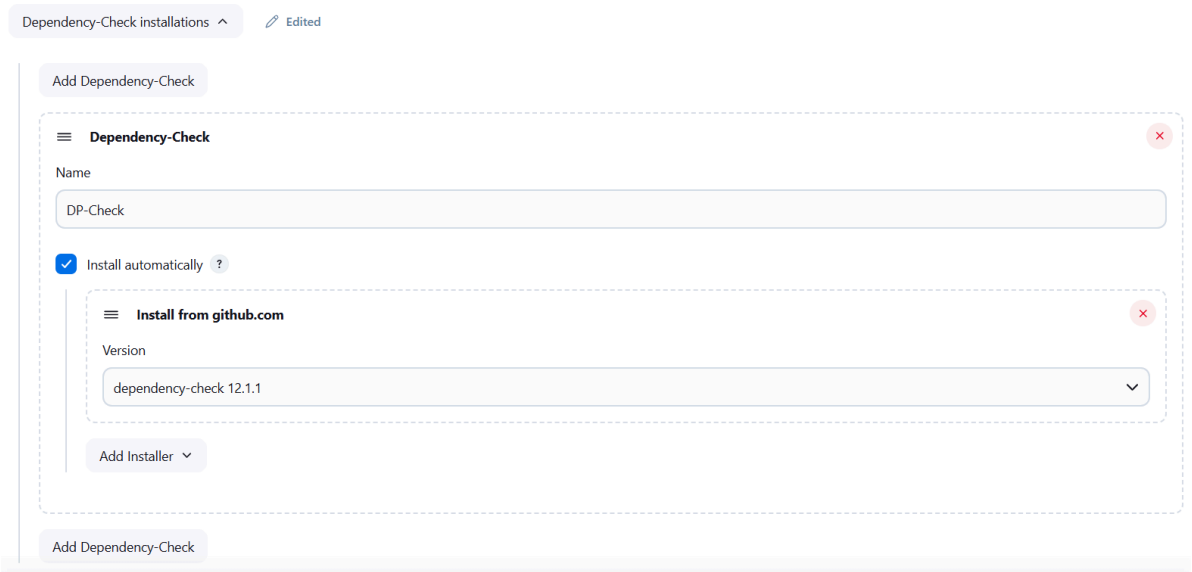
2. **NodeJS:** Install NodeJS (version 16)



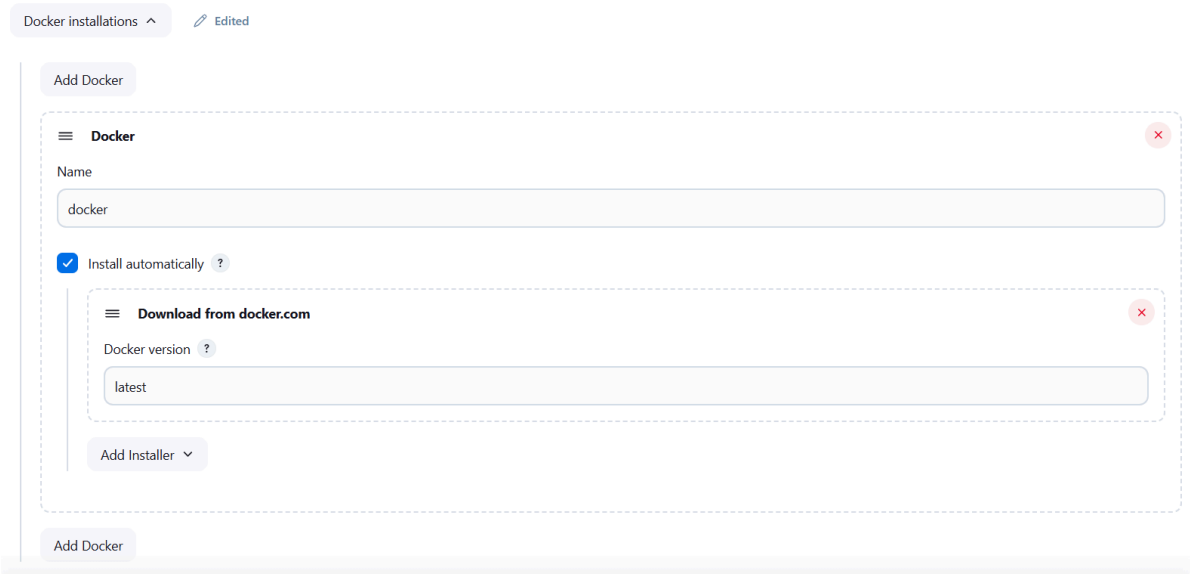
3. **SonarQube Scanner:** Add name `sonar-scanner` and install automatically



4. **Dependency-Check:** Add installation



5. **Docker:** Add Docker installation path

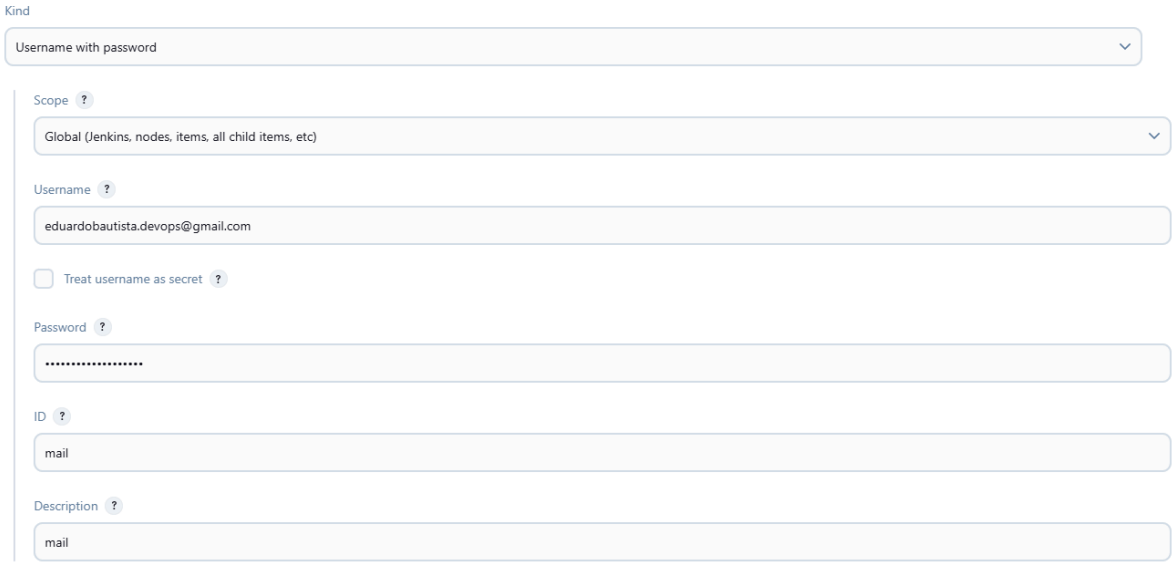


Click **Apply and Save**

Step 6: Configure E-mail Notification

Gmail SMTP (Optional - Email Notification)

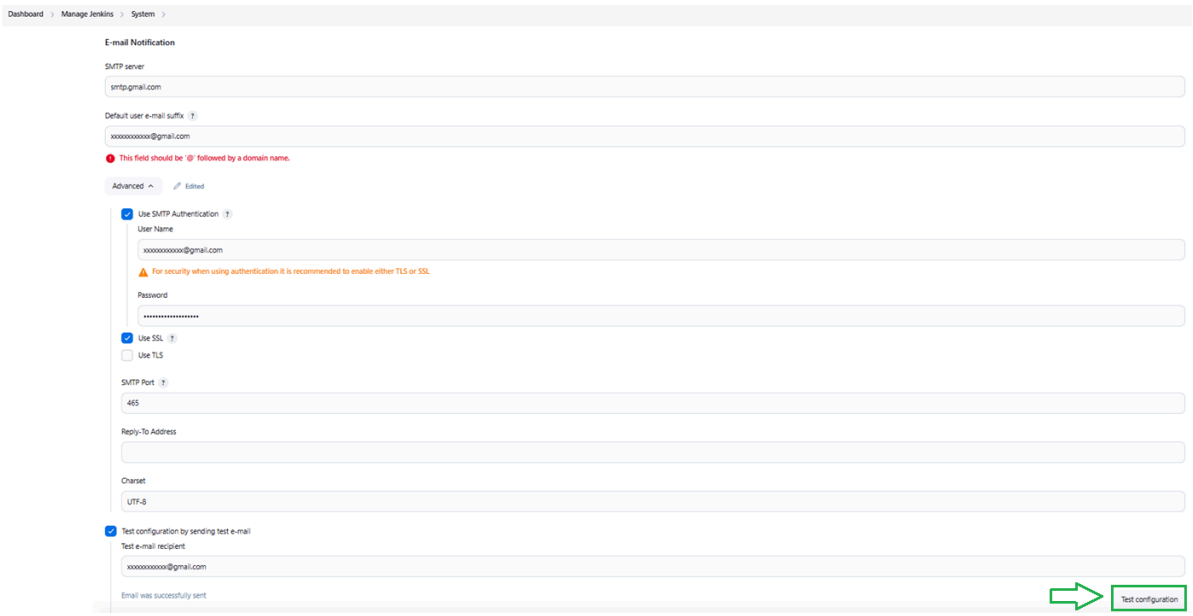
1. Enable 2FA in Gmail
    - Click on profile picture on the right → “Manage your google account”
    - Navigate to **Security**
    - Under **"Signing in to Google"**, enable **2-Step Verification**
2. Create “App password” for **Third-party apps**
    - Go to this URL: https://myaccount.google.com/apppasswords
    - 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



E-mail Notification

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

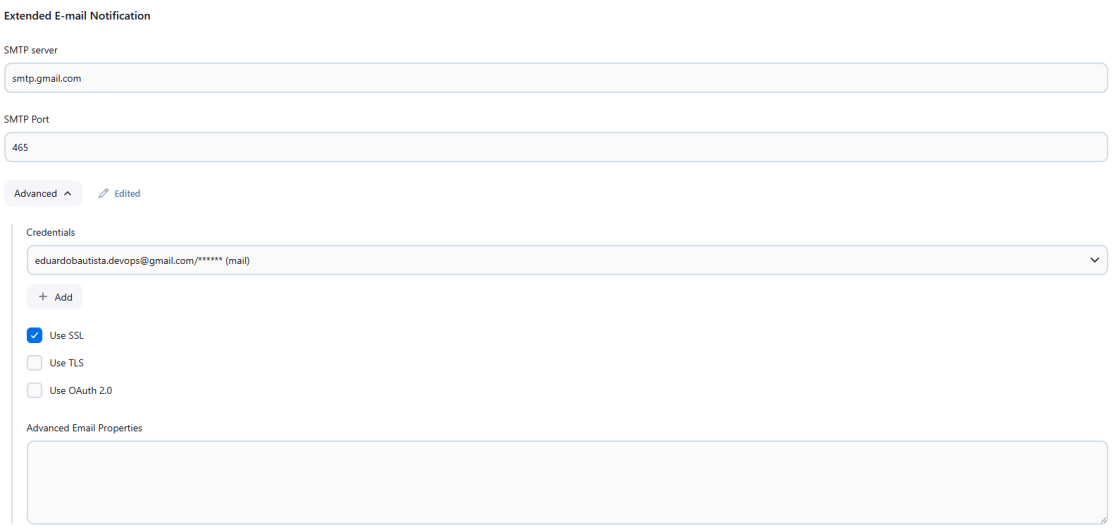
1. Scroll to **E-mail Notification**
2. SMTP server: `smtp.gmail.com`
3. Port: `465`, SSL enabled
4. User: your Gmail address
5. Password: Gmail App Password
6. Test the configuration



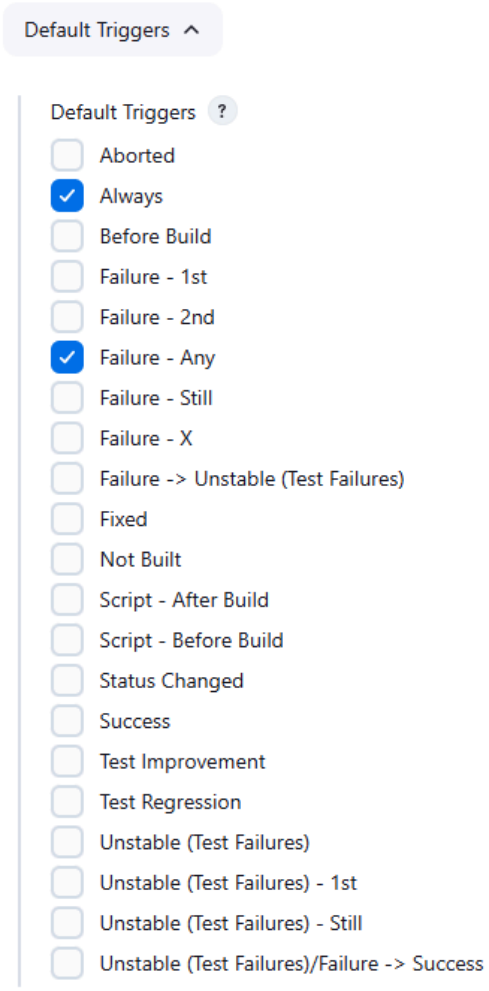
7. A test email from Jenkins should arrive in the email inbox.

Extended Email Notification

- Same SMTP config as above



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



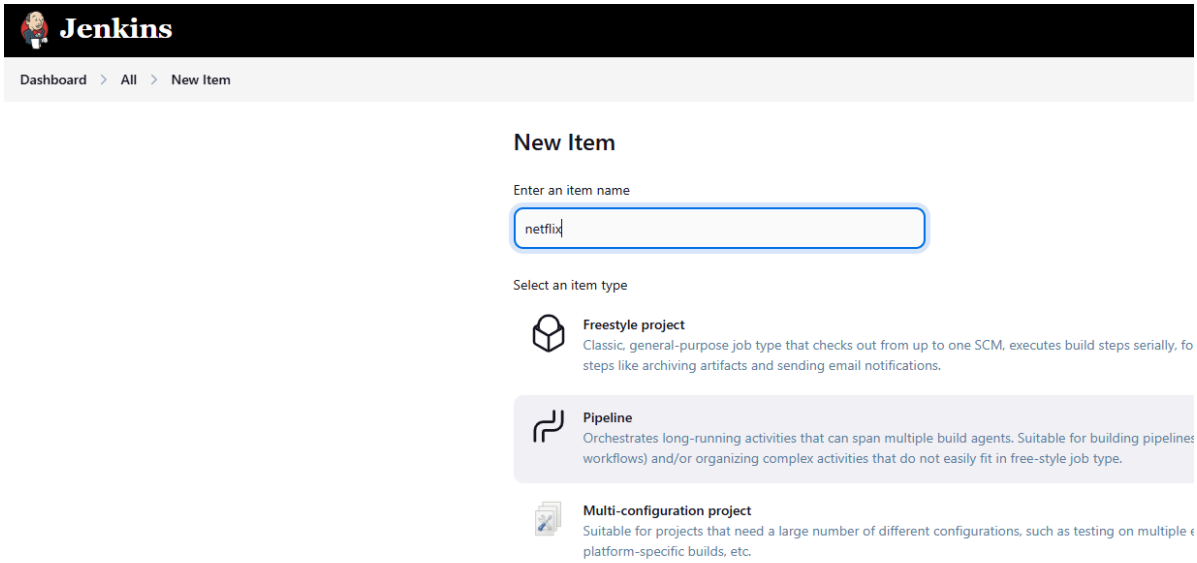
Click **Apply and Save**

Step 7: Create project “Netflix” for Sonarqube Analysis

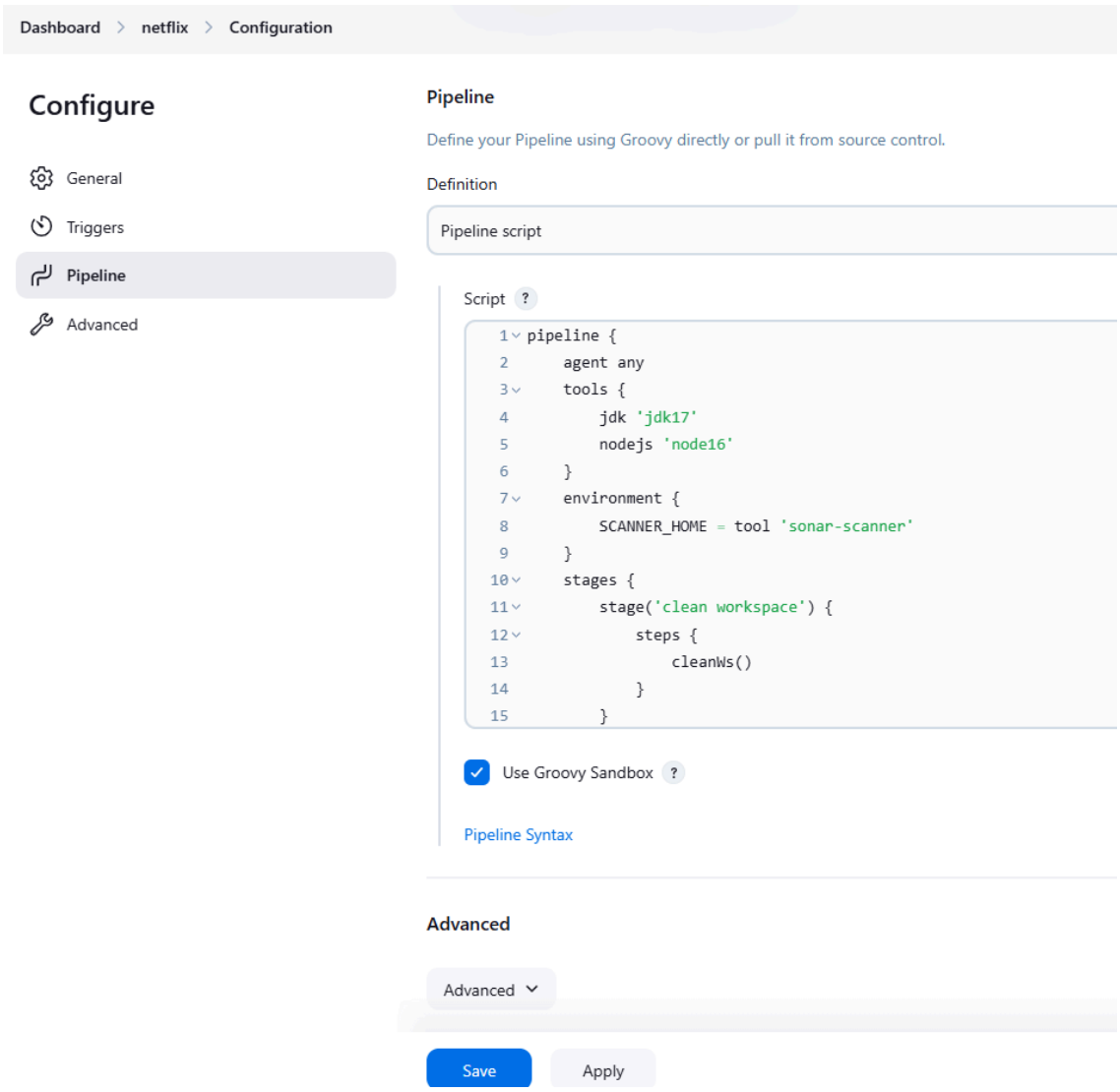
1. Access SonarQube via: `http://<netflix-jenkins-elastic IP>:9000`
2. In SonarQube: **Projects → Manually**
3. Project display name: Netflix
4. Click on: Set up
5. In SonarQube: **Projects → Locally → Generate**

Step 8: Create Pipeline Job

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



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



Complete Jenkins script:

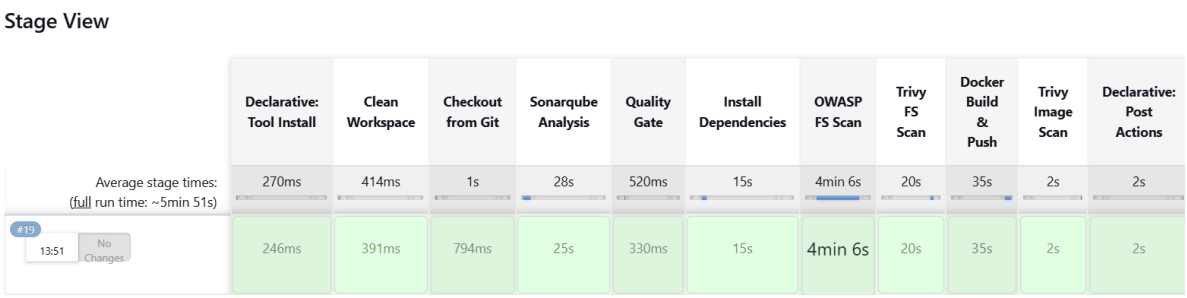
```
pipeline {
    agent any
    tools {
        jdk 'jdk17'
        nodejs 'node16'
    }
    environment {
        SCANNER_HOME = tool 'sonar-scanner'
    }
```

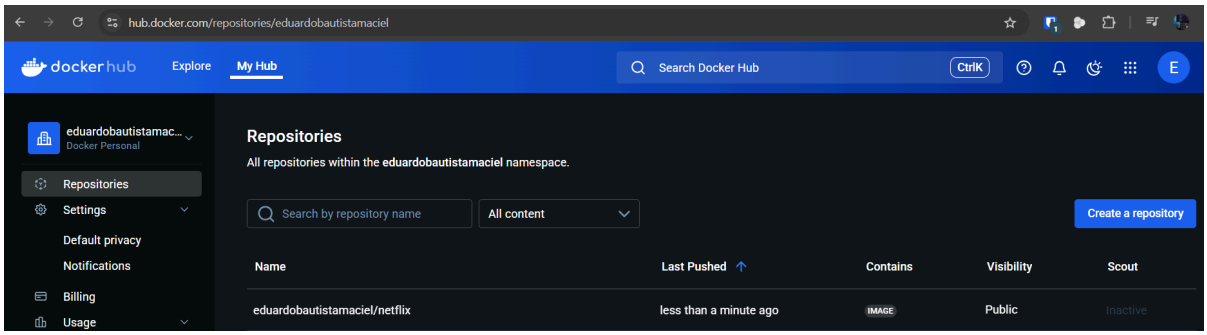
```
    stages {
        stage('Clean Workspace') {
            steps {
                cleanWs()
            }
        }
        stage('Checkout from Git') {
            steps {
                git branch: 'manual_project', url: 'https://github.com/eduardobautist
            }
        }
        stage("Sonarqube Analysis") {
            steps {
                withSonarQubeEnv('sonar-server') {
                    sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName
                    -Dsonar.projectKey=Netflix '''
                }
            }
        }
        stage("Quality Gate") {
            steps {
                script {
                    waitForQualityGate abortPipeline: false, credentialsId: 'Sonar-toke
                }
            }
        }
        stage('Install Dependencies') {
            steps {
                sh "npm install"
            }
        }
        stage('OWASP FS Scan') {
            steps {
                dependencyCheck additionalArguments: '--scan ./ --disableYarnAu
                dependencyCheckPublisher pattern: '**/dependency-check-report.
            }
```

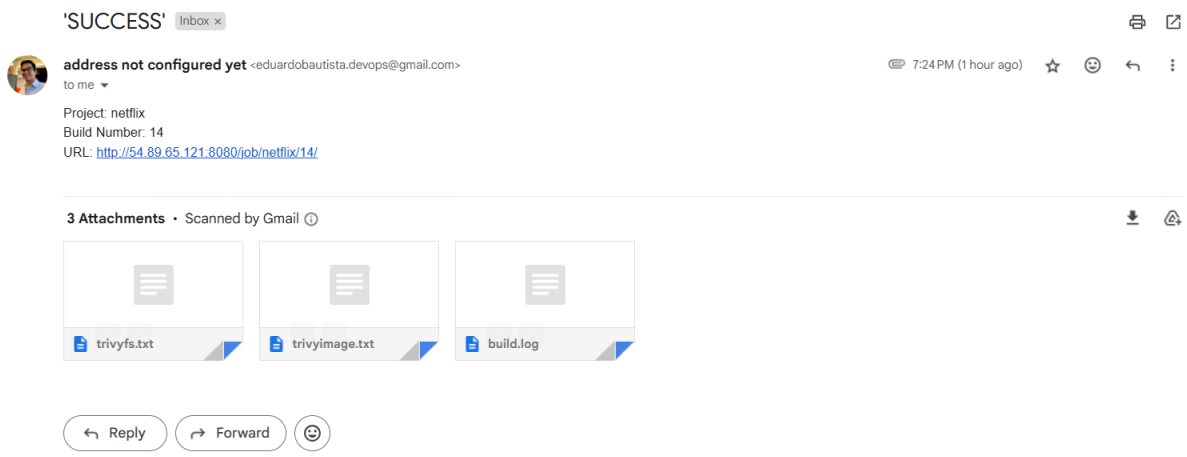
```
        }
        stage('Trivy FS Scan') {
            steps {
                sh "trivy fs . > trivyfs.txt"
            }
        }
        stage("Docker Build & Push") {
            steps {
                script {
                    withCredentials([string(credentialsId: 'TMDB_V3_API_KEY', variab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







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
```

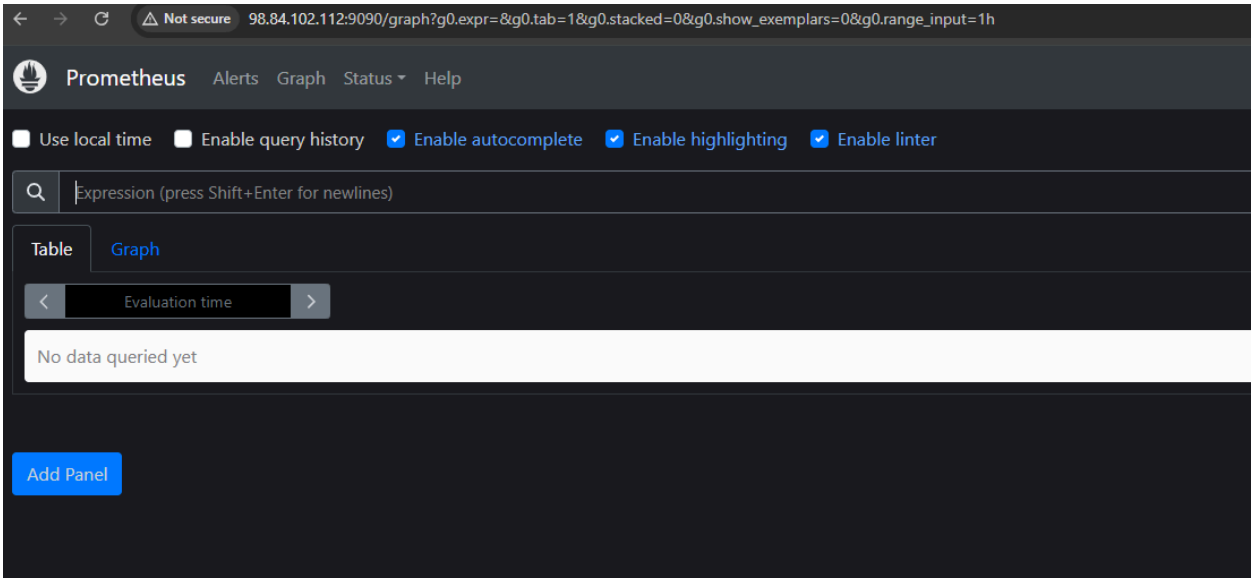
Verify Prometheus's status:

```
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
```

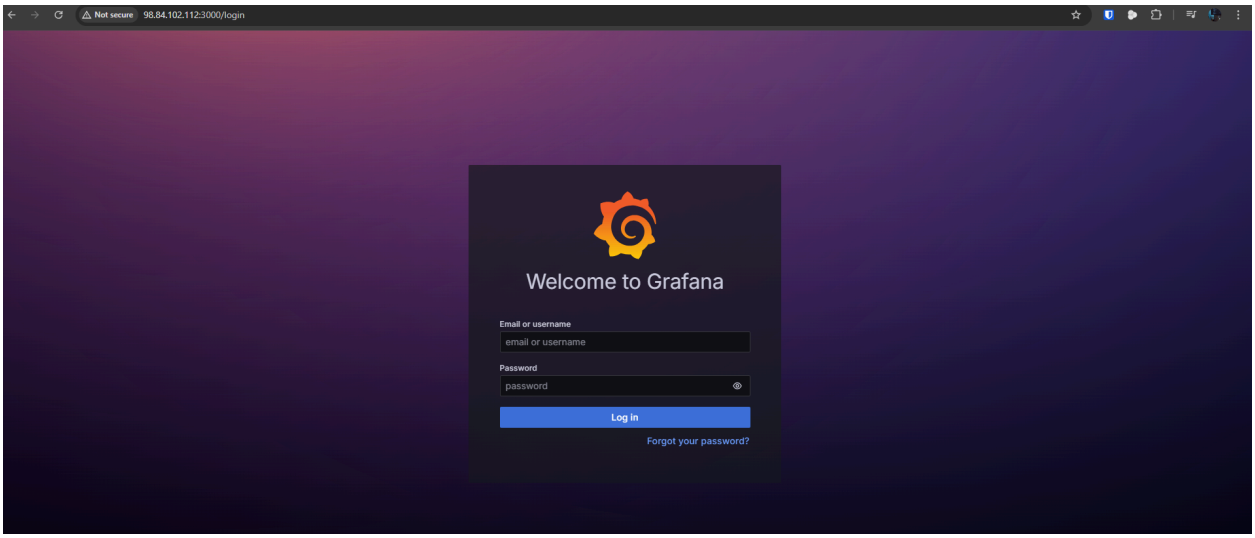
6. Check Grafana Status:

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

7. Access Grafana Web Interface:

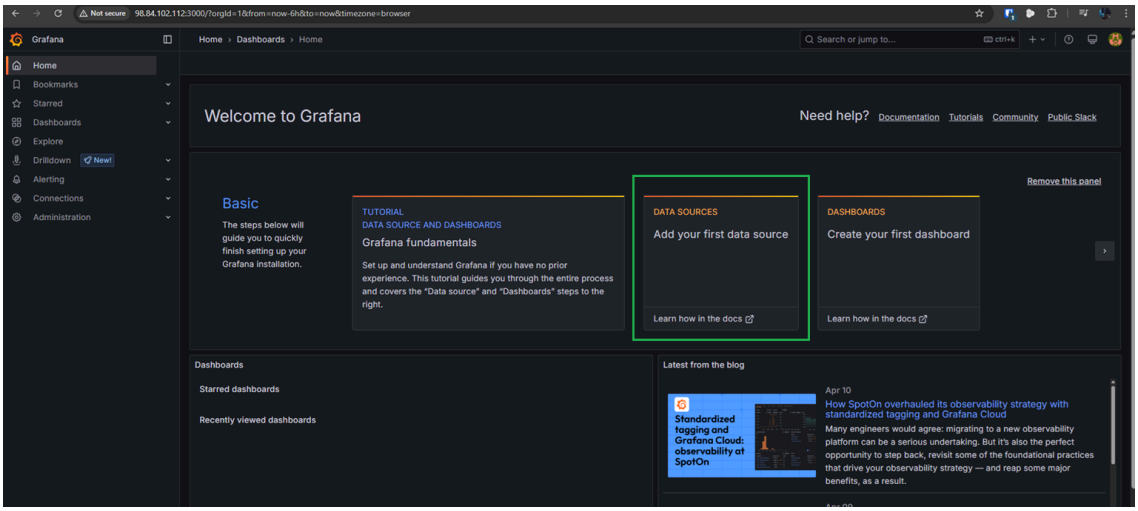
Open a web browser and navigate to Grafana using your server's IP address. The default port for Grafana is 3000.:

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



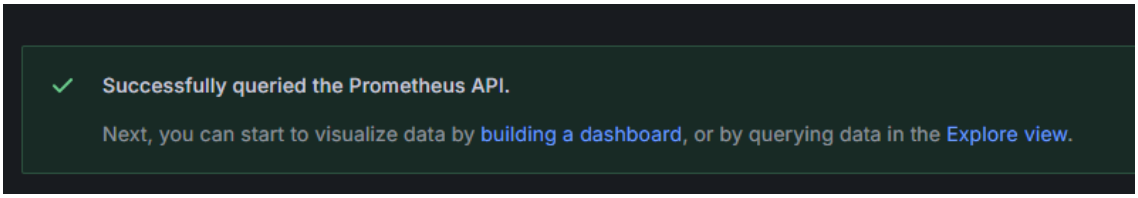
- Default credentials:
  - **Username**: `admin`
  - **Password**: `admin`
  - You'll be prompted to change the password on first login.

8. Add Prometheus as a Data Source in Grafana
   - Navigate to: **Gear Icon (⚙️) → Data Sources → Add data source**

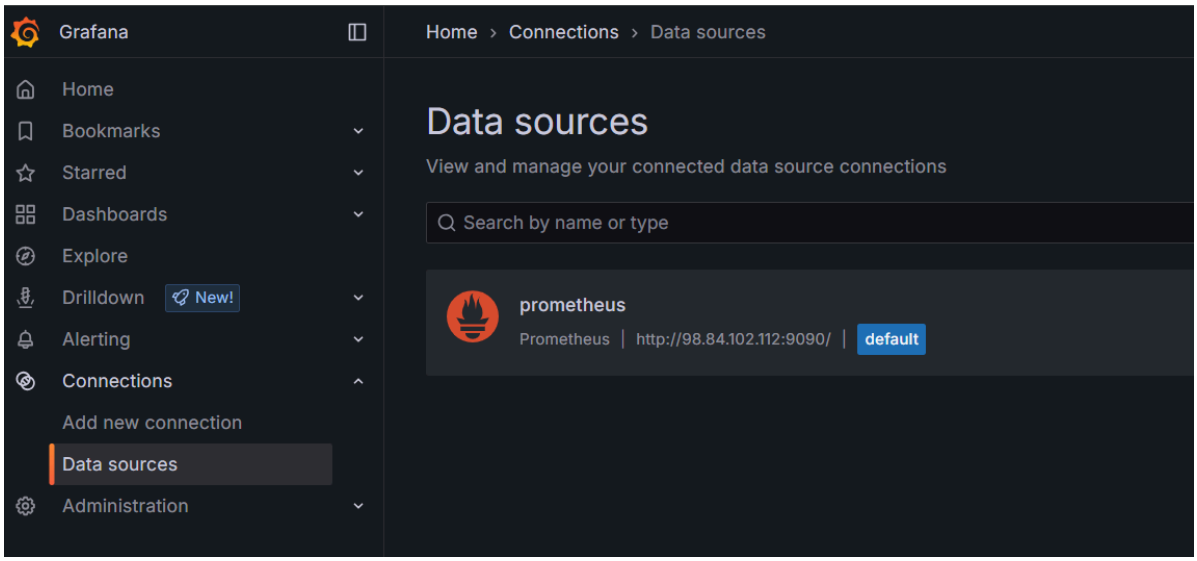


- Choose "Prometheus"
- In the "HTTP" section:
  - Set the "URL" to `http://<Monitoring-elastic IP>:9090`
- Click the "Save & Test" button to ensure the data source is working.

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



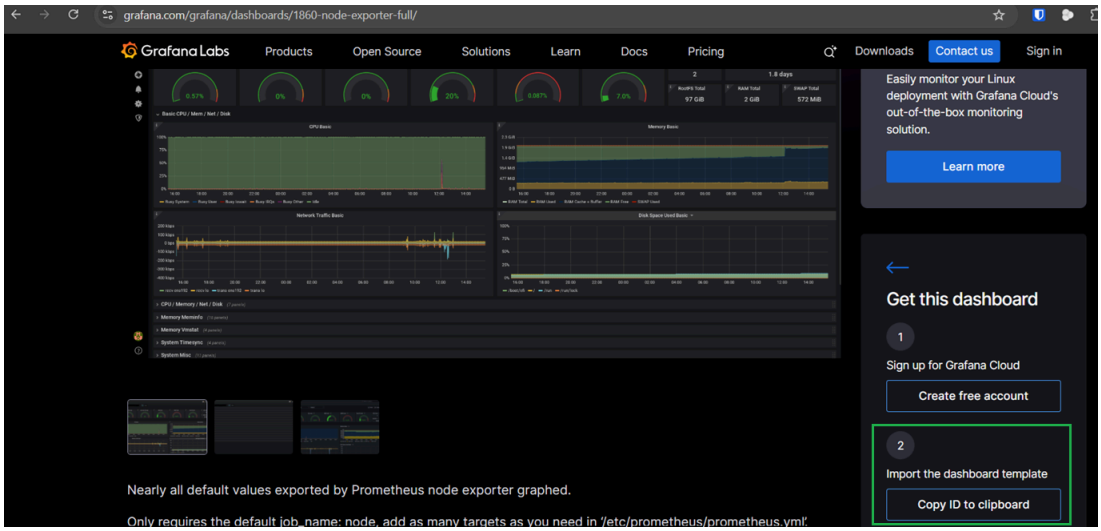
You can also confirm here:



9. Import Pre-built Dashboards:

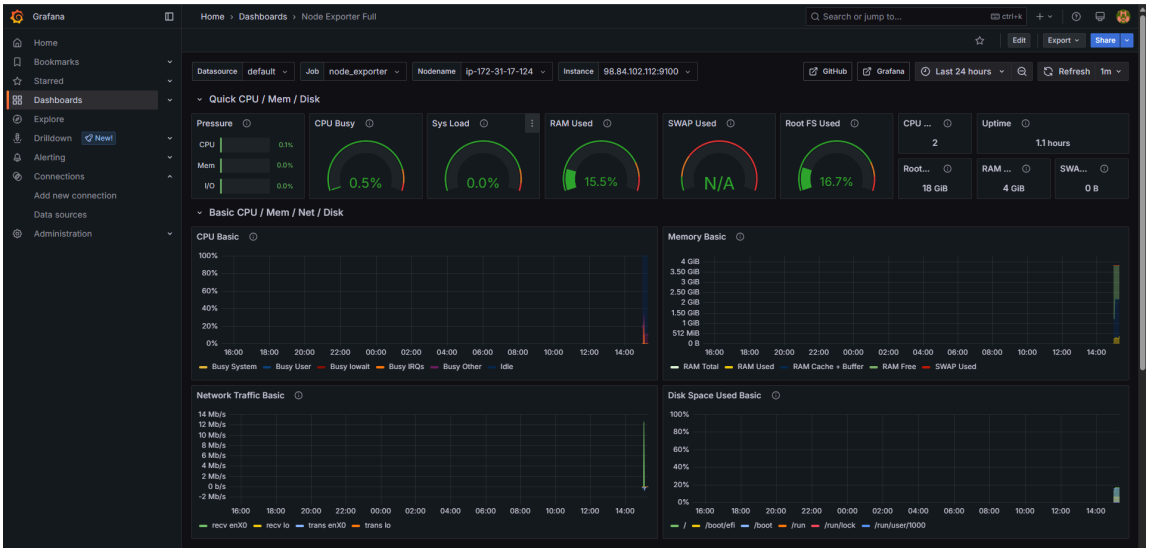
Set the monitoring server Node Exporter grafana dashboard:

- 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”



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

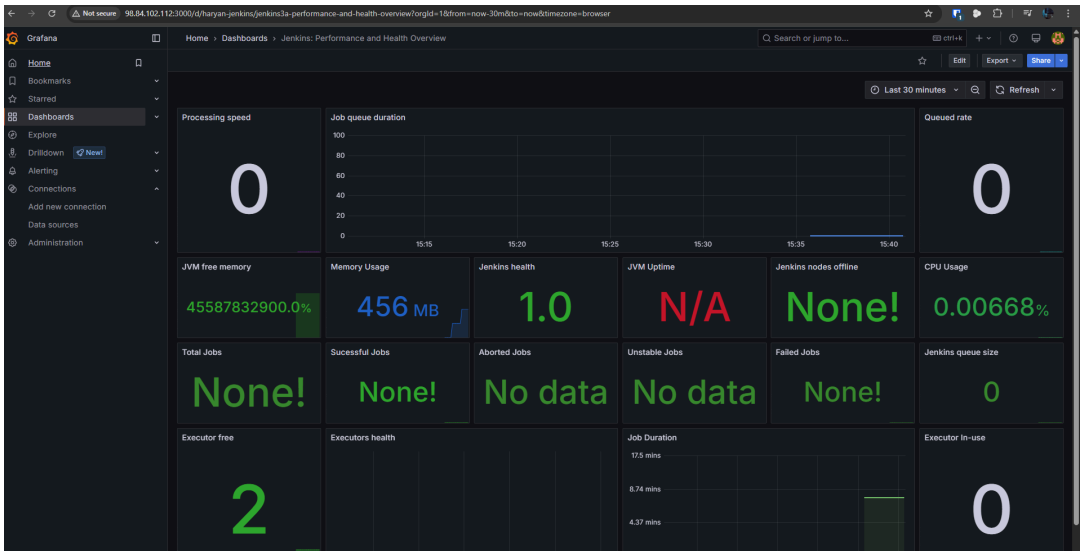
- Click **Import**.
- You should now have a Grafana dashboard set up to visualize metrics from Prometheus.
    - We can observer CPU, Memory, RAM from the monitoring server



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 observer 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 /h
```

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.github.io/helm-charts
   ```

2. Create a Kubernetes namespace for the Node Exporter:

   ```
   kubectl create namespace prometheus-node-exporter
   ```

3. Install the Node Exporter using Helm:

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

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']
```

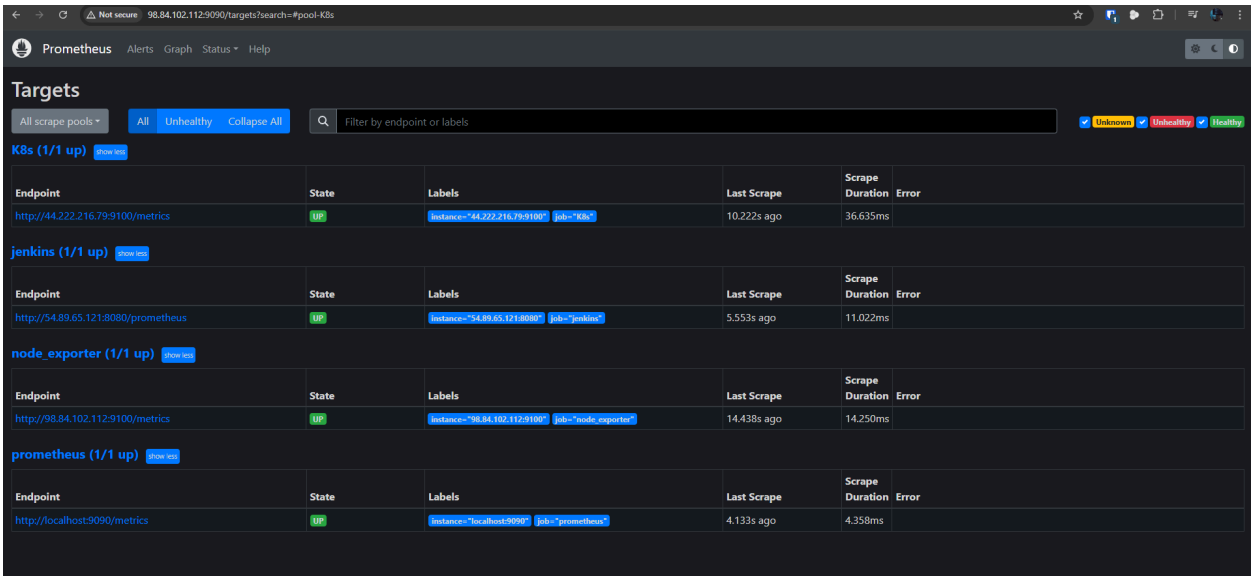
> Replace `<Your-Node-IP>` with your actual node IP.

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`



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             2m
pod/argocd-applicationset-controller-8477cf5d6d-bbb2w   1/1     Running   0
pod/argocd-dex-server-575bc85fc7-ffgw7          1/1     Running   2 (2m3
pod/argocd-notifications-controller-5fff6cd8b7-8whdp    1/1     Running   0
pod/argocd-redis-dd9464c7c-28krr                1/1     Running   0             2
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.amazona
ws.com
```

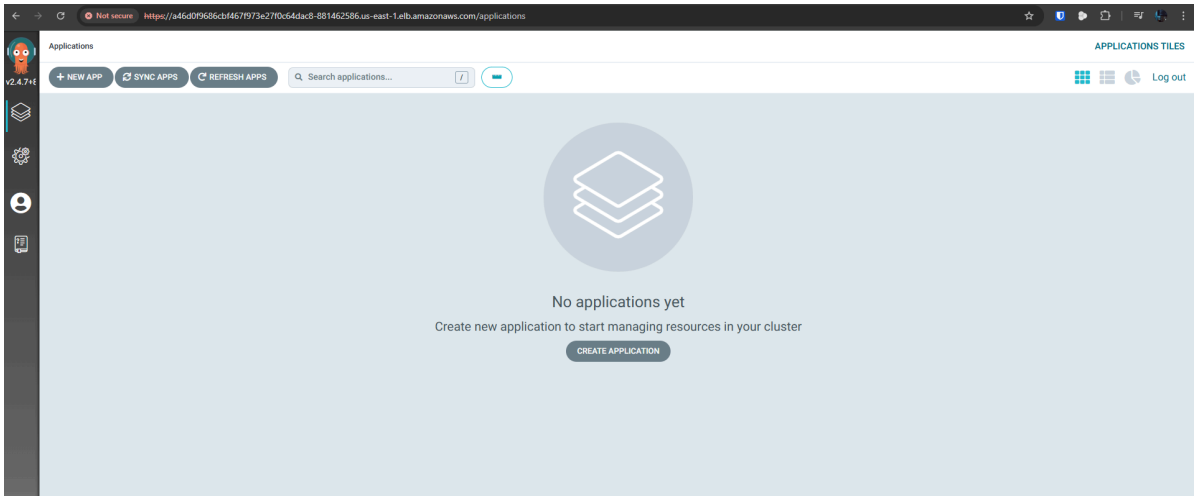
7. Get ArgoCD Admin Password:

```
export ARGO_PWD=`kubectl -n argocd get secret argocd-initial-admin-secre
echo $ARGO_PWD
```

8. Access ArgoCD:

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

   Expected Successful login:



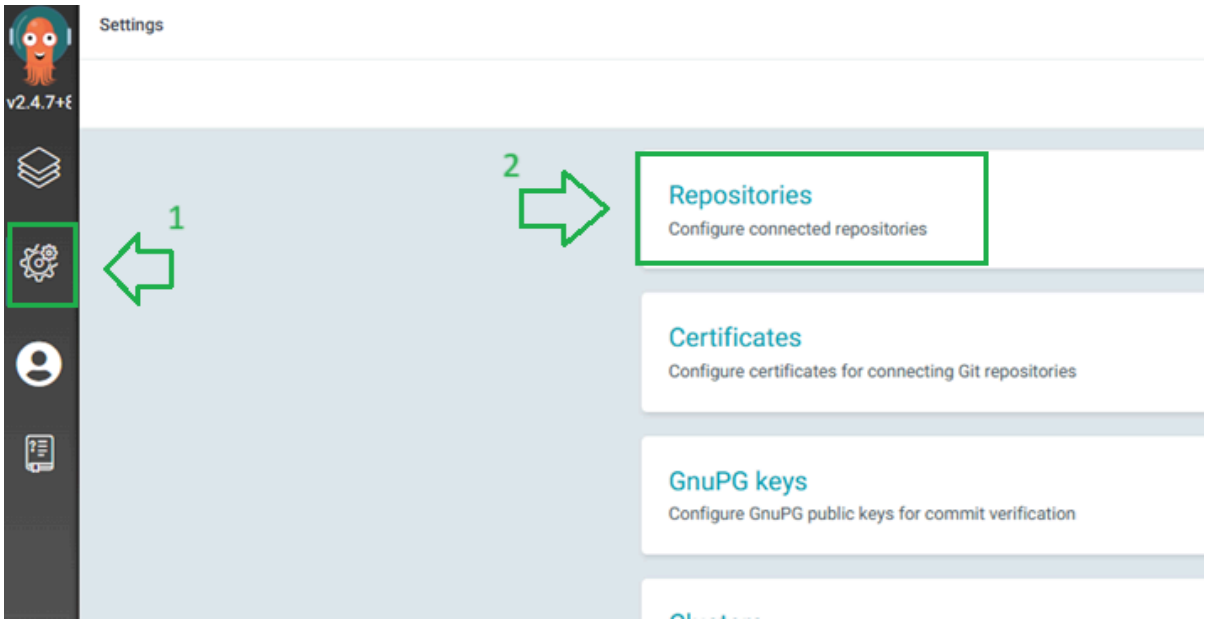
In the ArgoCD UI:

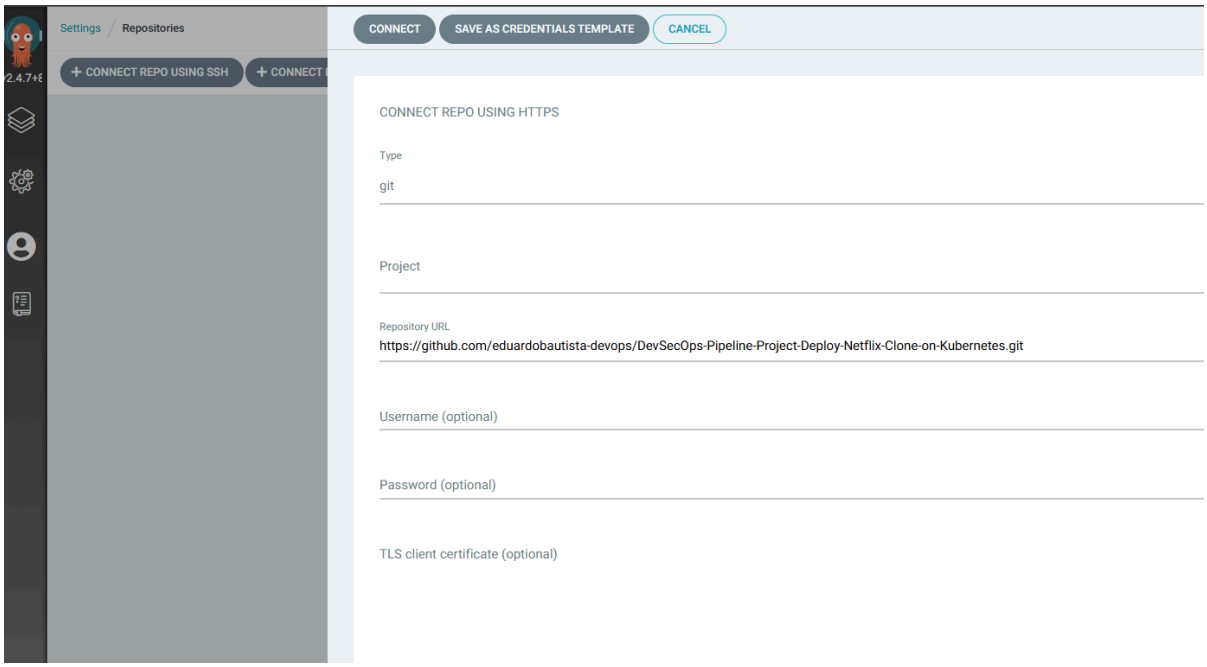
- Settings → Repositories → "CONNECT REPO USING HTTPS"

- Project: `default`
- URL:

  `https://github.com/eduardobautista-devops/DevSecOps-Pipeline-Project-Deploy-Netflix-Clone-on-Kubernetes.git`

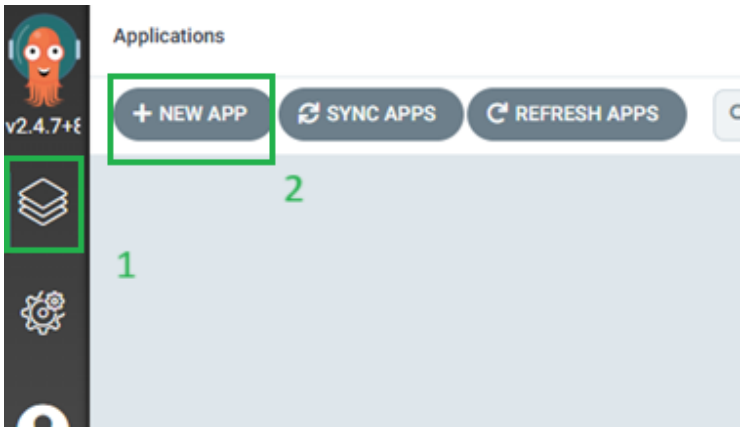
Step 2: Connect Your GitHub Repo





- Then click on CONNECT

Step 3: Create ArgoCD Application



1. **Application Name**: `netflix`
2. **Project**: `default`
3. **Source Repo**:

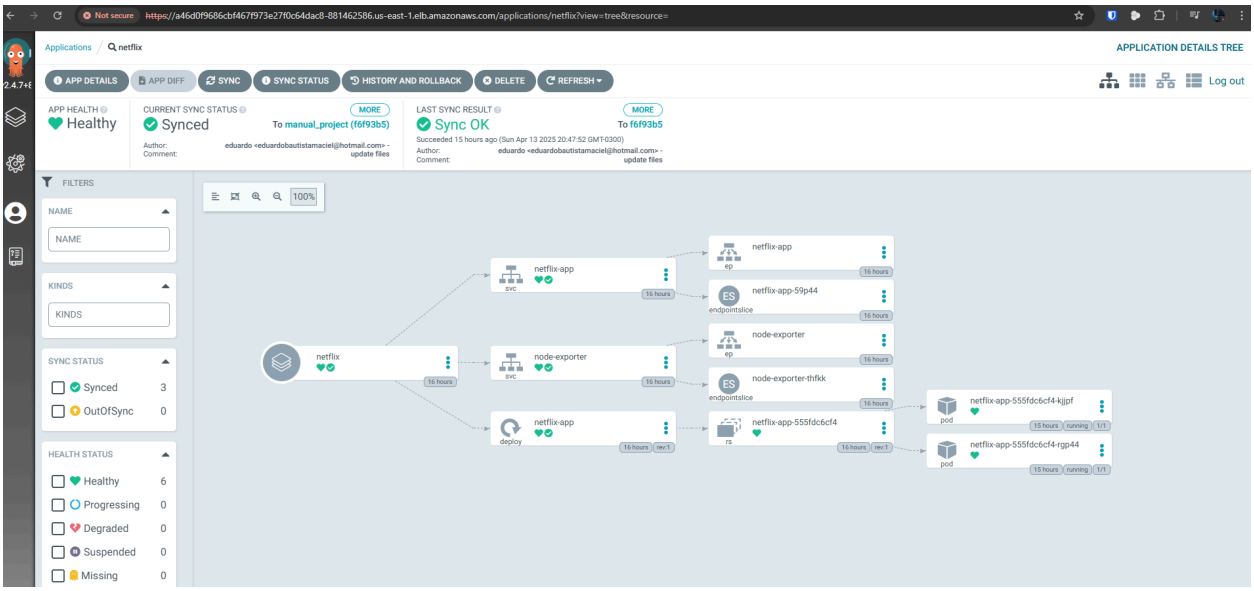
- URL: `https://github.com/eduardobautista-devops/DevSecOps-Pipeline-Project-Deploy-Netflix-Clone-on-Kubernetes.git`
- Revision: `HEAD` or branch name
- Path: `Kubernetes`

4. **Destination**:
   - Cluster URL: `https://kubernetes.default.svc`
   - Namespace: `default`

Click **Save**

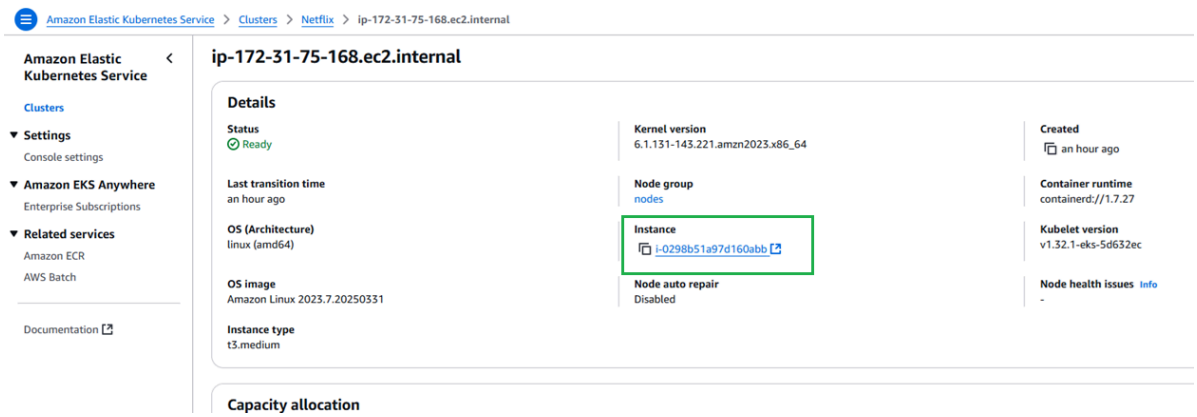
Step 4: Sync Application

Click **"SYNC" → FORCE → SYNCHRONIZE**

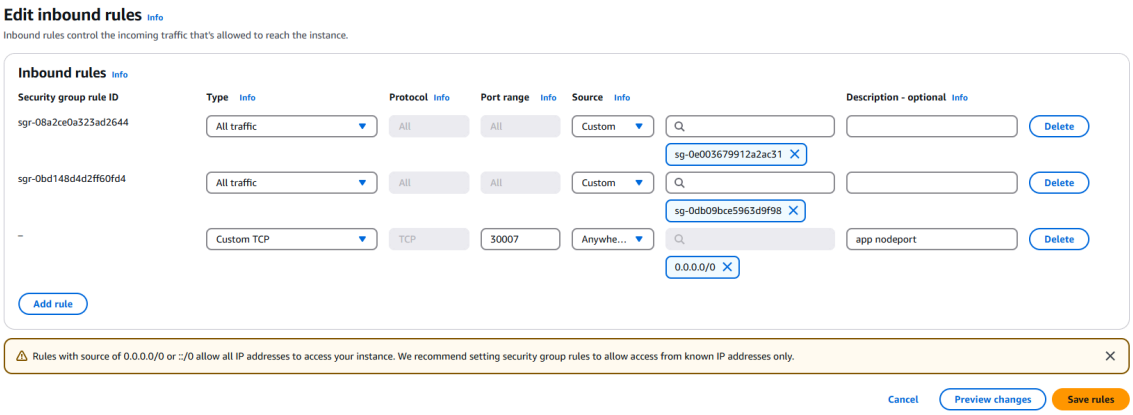


Step 5: Access Netflix App

1. Go to:
   - EKS → Compute → Nodes → Instance → Security Group

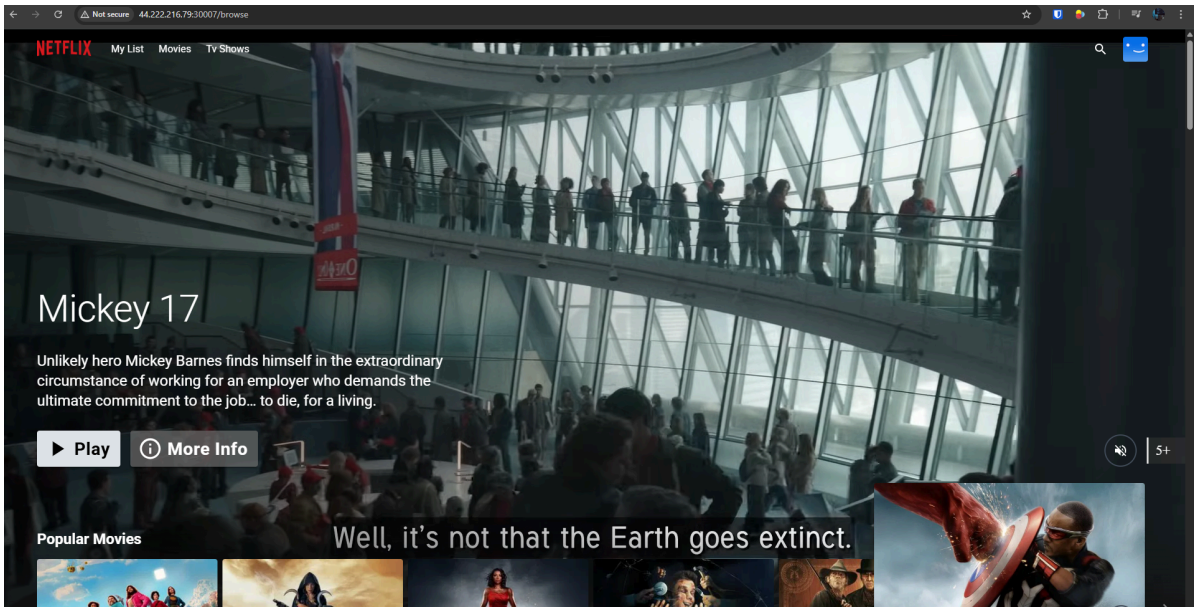


2. **Add Inbound Rule**: Port `30007`, Type: `Custom TCP`



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/d
Warning: failed to get default registry endpoint from daemon (Got permission der
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.
