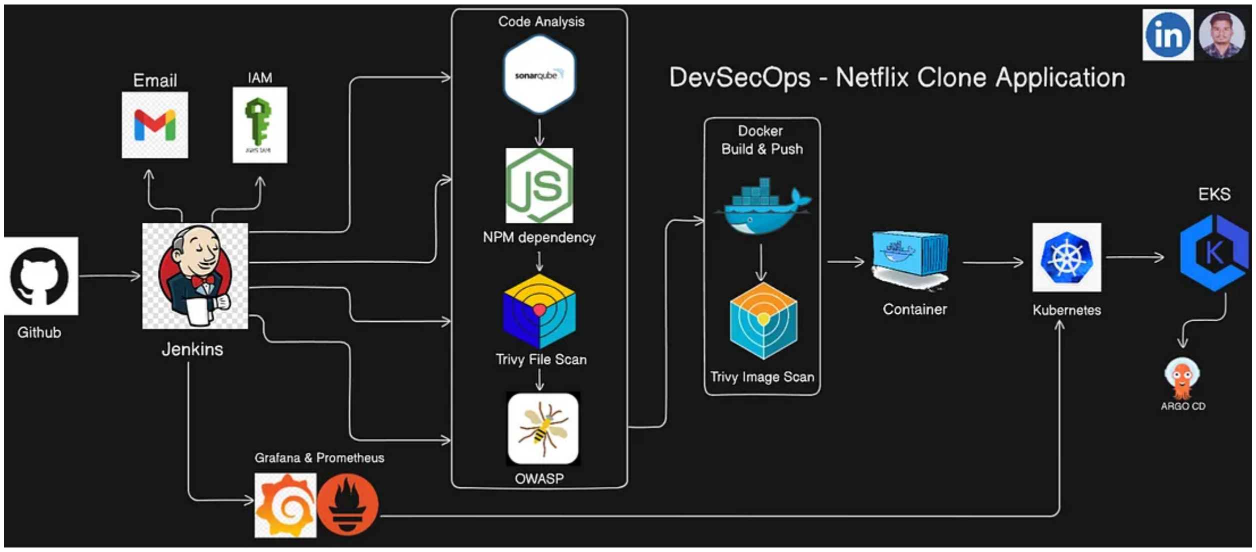
DevSecOps Project



Prerequisites:

· Repository: https://github.com/N4si/DevSecOps-Project

· A DockerHub account is necessary for handling Docker images.

· For email alerts, a Gmail account is required.

Overview:

This project sets up a smooth DevOps environment for easy software development, integration, and deployment. Key tools include Jenkins for continuous integration, SonarQube for code quality, Trivy and OWASP Dependency-Check for security, and Prometheus with Grafana for monitoring.

Key Features:

1. **Infrastructure Setup:**

- Servers for Jenkins, Monitoring, and Kubernetes are set up.

1. **Tool Integration:**

- Jenkins is the central hub, integrating SonarQube, Trivy, Prometheus, Grafana, and OWASP Dependency-Check.

1. **CI/CD Pipelines:**

- Automated workflows are created using Jenkins pipelines, covering code analysis, Docker image creation, and Kubernetes deployment.

1. **Security Checks:**

- Trivy and OWASP Dependency-Check are implemented for continuous security scans in the DevOps pipeline.

1. **Monitoring and Visualization:**

- Prometheus is configured for real-time metrics, and Grafana for visualizing comprehensive data.

1. **Email Alerts:**

- Jenkins is configured to send email alerts based on pipeline results.

We need four servers for our today's Project.

For this setup GCP is used, you can use any other cloud provider as per your choice.

For better performance CPU: "n1-standard-2" and RAM: 4GB chosen.

Jenkins Server- On this Server, Jenkins will be installed with some other tools such as sonarqube(docker container), trivy, and kubectl.

Monitoring Server- This Server will be used for Monitoring where we will use Prometheus, Node Exporter, and Grafana.

Kubernetes Master Server- This Server will be used as the Kubernetes Master Cluster Node which will deploy the applications on worker nodes.

Kubernetes Worker Server- This Server will be used as the Kubernetes Worker Node on which the application will be deployed by the master node.

SETUP

Jenkins Server:

· Name: Provide a name for your Jenkins instance.

· Image: Select Ubuntu 22.04 LTS.

· Machine Type: Choose n1-standard-2 or higher, ensuring it has at least 4GB RAM.

· Boot Disk: Increase capacity from 15 to 35GB.

· Networking: Keep the default settings.

· Firewall: Allow HTTP (80) and HTTPS (443) traffic.

Monitoring Server:

· Name: Provide a name for your Monitoring instance.

· Image: Select Ubuntu 22.04 LTS.

· Machine Type: Choose n1-standard-2 or higher, ensuring it has at least 4GB RAM.

· Boot Disk: Increase capacity from 15 to 35GB.

· Networking: Keep the default settings.

· Firewall: Allow HTTP (80) and HTTPS (443) traffic.

Kubernetes Master:

· Name: Provide a name for your Kubernetes Master instance.

· Image: Select Ubuntu 22.04 LTS.

· Machine Type: Choose n1-standard-2 or higher, ensuring it has at least 4GB RAM.

· Boot Disk: Increase capacity from 15 to 35GB.

· Networking: Keep the default settings.

· Firewall: Allow HTTP (80) and HTTPS (443) traffic.

Worker Nodes:

· Name: Provide a name for your Kubernetes Worker instance.

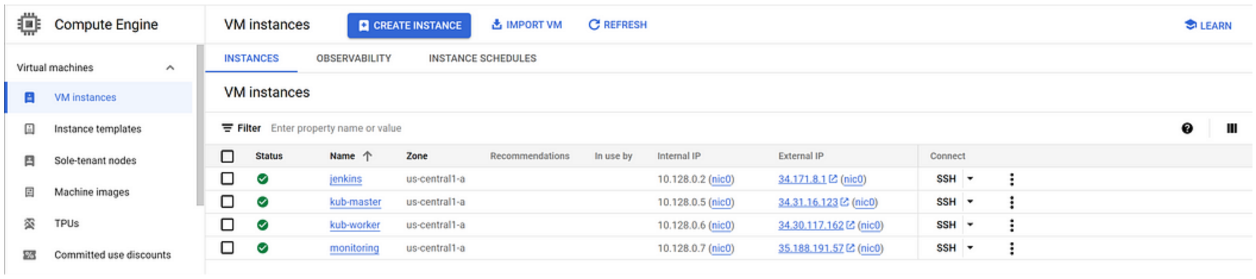
· Image: Select Ubuntu 22.04 LTS.

· Machine Type: Choose n1-standard-2 or higher, ensuring it has at least 4GB RAM.

· Boot Disk: Increase capacity from 15 to 35GB.

· Networking: Keep the default settings.

· Firewall: Allow HTTP (80) and HTTPS (443) traffic.



Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
✓	jenkins	us-central1-a			10.128.0.2 (nic0)	34.171.8.1 (nic0)	SSH
✓	kub-master	us-central1-a			10.128.0.5 (nic0)	34.31.16.123 (nic0)	SSH
✓	kub-worker	us-central1-a			10.128.0.6 (nic0)	34.30.117.162 (nic0)	SSH
✓	monitoring	us-central1-a			10.128.0.7 (nic0)	35.188.191.57 (nic0)	SSH

Log in to Jenkins Server:

Connect to Jenkins Server:

- Use SSH to connect to your Jenkins Server.

Install OpenJDK and Jenkins:

```
sudo apt update -y
sudo apt install openjdk-11-jre -y
java –version
```

Install Jenkins:

```
curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.key
| sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/nul
l
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]
https://pkg.jenkins.io/debian binary/ | sudo tee /etc/apt/sou
rces.list.d/jenkins.list > /dev/null
sudo apt-get update -y
sudo apt-get install jenkins -y
```

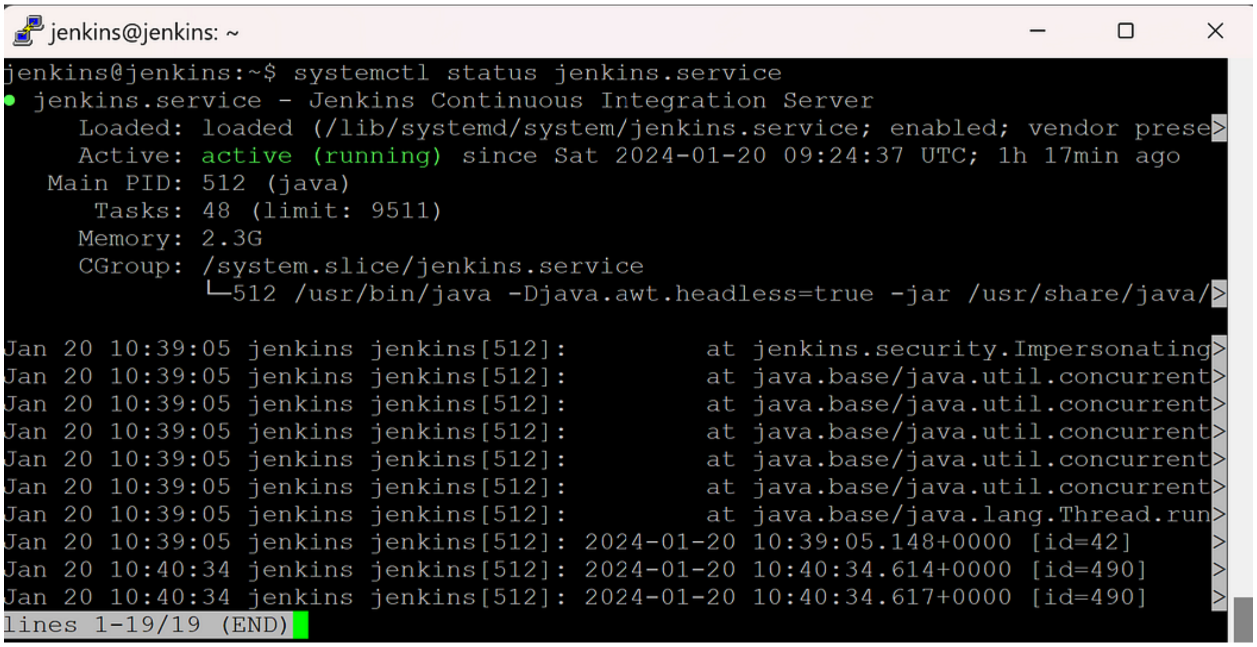
To Jenkins Start:

```
sudo systemctl enable Jenkins.service
sudo systemctl start Jenkins.service
```

Check Jenkins Server Status:

- After installation, ensure that Jenkins is running properly.

```
sudo systemctl status Jenkins.service
```



```
jenkins@jenkins: ~
jenkins@jenkins:~$ systemctl status jenkins.service
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor prese>
     Active: active (running) since Sat 2024-01-20 09:24:37 UTC; 1h 17min ago
   Main PID: 512 (java)
      Tasks: 48 (limit: 9511)
     Memory: 2.3G
     CGroup: /system.slice/jenkins.service
             └─512 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/>

Jan 20 10:39:05 jenkins jenkins[512]:         at jenkins.security.Impersonating>
Jan 20 10:39:05 jenkins jenkins[512]:         at java.base/java.util.concurrent>
Jan 20 10:39:05 jenkins jenkins[512]:         at java.base/java.util.concurrent>
Jan 20 10:39:05 jenkins jenkins[512]:         at java.base/java.util.concurrent>
Jan 20 10:39:05 jenkins jenkins[512]:         at java.base/java.util.concurrent>
Jan 20 10:39:05 jenkins jenkins[512]:         at java.base/java.util.concurrent>
Jan 20 10:39:05 jenkins jenkins[512]:         at java.base/java.lang.Thread.run>
Jan 20 10:39:05 jenkins jenkins[512]: 2024-01-20 10:39:05.148+0000 [id=42]     >
Jan 20 10:40:34 jenkins jenkins[512]: 2024-01-20 10:40:34.614+0000 [id=490]    >
Jan 20 10:40:34 jenkins jenkins[512]: 2024-01-20 10:40:34.617+0000 [id=490]    >
lines 1-19/19 (END)
```

Access Jenkins Web Interface:

- Copy Jenkins Server Public IP and paste it into your browser with port number 8080.

Obtain Jenkins Initial Admin Password:

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

Continue Setup:

- Paste the obtained password into the Jenkins web interface and click on Continue.

Jenkins Configuration:

Install Suggested Plugins:

- Click on “Install suggested plugins.”

You will get “Create First Admin User” or You can “Skip and Continue as Admin” and setup later

Save and Finish:

- Click on “Save and Finish.”

Installing Docker, configuring it on the Jenkins Server, and installing SonarQube using a Docker container.

Install Docker and Configure on Jenkins Server:

Update and Install Docker:

```
sudo apt update
sudo apt install docker.io -y
```

Add Jenkins User to Docker Group:

```
sudo usermod -aG docker jenkins
Add Ubuntu User to Docker Group (if needed):
sudo usermod -aG docker ubuntu
```

Restart Docker Service:

```
sudo systemctl restart docker
```

Adjust Docker Socket Permissions:

```
sudo chmod 777 /var/run/docker.sock
```

Install SonarQube on Jenkins Server:

Run SonarQube Docker Container:

```
docker run -d - name sonar -p 9000:9000 sonarqube:lts-communi
ty
```

Access SonarQube Web Interface:

- Copy your Jenkins Server Public IP and add port 9000 in your browser.

Login to SonarQube:

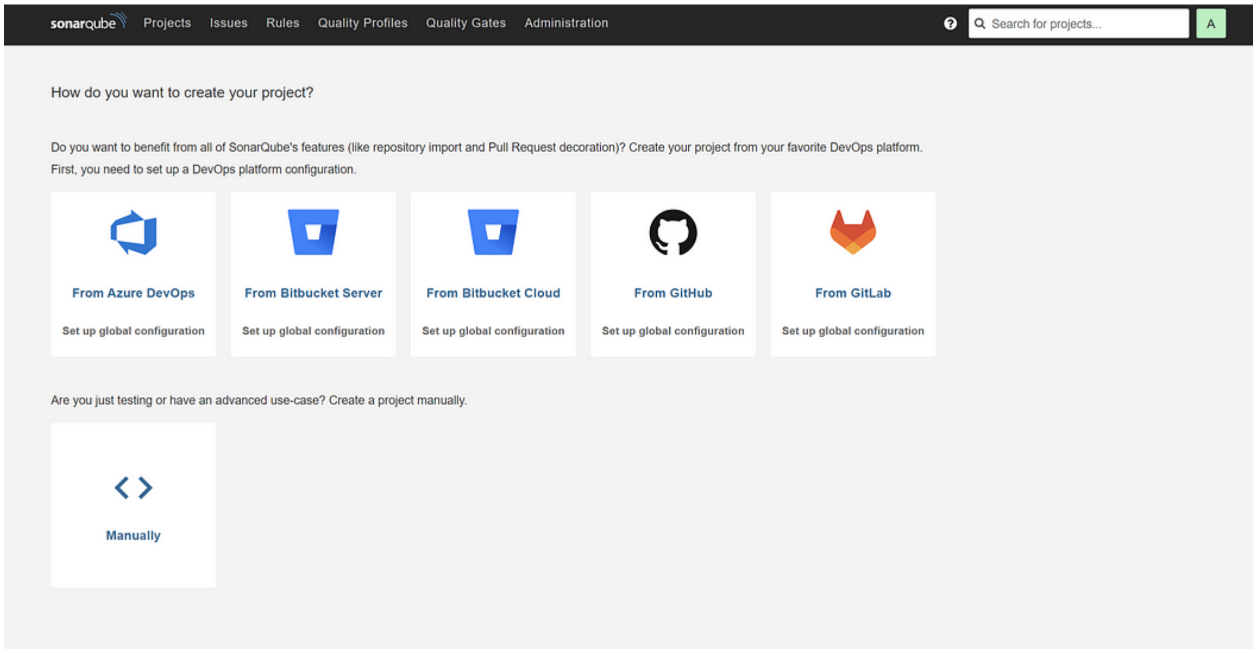
- Username: admin
- Password: admin

Reset Password and Update:

- Reset the password and click on Update.

Verify SonarQube Server:

- Confirm that your SonarQube Server is accessible.



How do you want to create your project?

Do you want to benefit from all of SonarQube's features (like repository import and Pull Request decoration)? Create your project from your favorite DevOps platform.
First, you need to set up a DevOps platform configuration.

From Azure DevOps
Set up global configuration
From Bitbucket Server
Set up global configuration
From Bitbucket Cloud
Set up global configuration
From GitHub
Set up global configuration
From GitLab
Set up global configuration

Are you just testing or have an advanced use-case? Create a project manually.

Installing the Trivy tool on the Jenkins Server.

Install Required Packages:

```
sudo apt-get install wget apt-transport-https gnupg lsb-relea
se
```

Add Trivy GPG Key:

```
wget -qO - https://aquasecurity.github.io/trivy-repo/deb/publ
ic.key | sudo apt-key add -
```

Add Trivy Repository:

```
echo deb https://aquasecurity.github.io/trivy-repo/deb $(lsb_
release -sc) main | sudo tee -a /etc/apt/sources.list.d/triv
y.list
```

Install Trivy:

```
sudo apt-get update
sudo apt-get install trivy
```

Install and Configure the Prometheus, Node Exporter, and Grafana on the Monitoring Server

Login to the Monitoring Server

Install and Configure Prometheus:

Create Prometheus User:

```
sudo useradd -r -s /bin/false prometheus
```

Download and Extract Prometheus

```
wget https://github.com/prometheus/prometheus/releases/downlo
ad/v2.49.0-rc.1/prometheus-2.49.0-rc.1.linux-amd64.tar.gz
tar -xvf prometheus-2.49.0-rc.1.linux-amd64.tar.gz
```

Move Prometheus Binaries and Configuration:

```
sudo mkdir -p /data /etc/prometheus
```

Now, enter into the prometheus package file that you have untar in the earlier step.

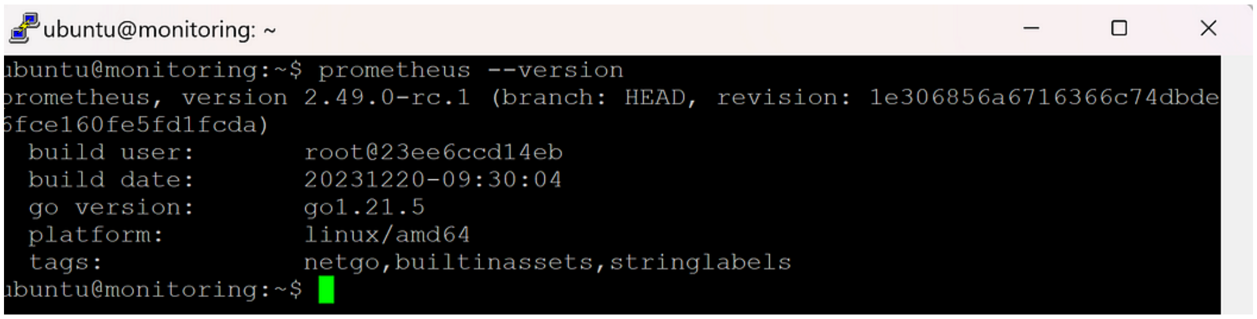
```
cd prometheus-2.49.0-rc.1.linux-amd64/
sudo mv prometheus promtool /usr/local/bin/
sudo mv consoles console_libraries/ prometheus.yml /etc/prome
theus/
```

Set Permissions:

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

Check and validate the Prometheus

```
prometheus –version
```


ubuntu@monitoring: ~

```
ubuntu@monitoring:~$ prometheus --version
rometheus, version 2.49.0-rc.1 (branch: HEAD, revision: 1e306856a6716366c74dbde
fce160fe5fd1fcda)
  build user:       root@23ee6ccd14eb
  build date:       20231220-09:30:04
  go version:       go1.21.5
  platform:         linux/amd64
  tags:             netgo,builtinassets,stringlabels
ubuntu@monitoring:~$
```

Create systemd Configuration for Prometheus:

- Edit `/etc/systemd/system/prometheus.service` and paste the provided configuration.

```
[Unit]
Description=Prometheus
Wants=network-online.target
```

```
After=network-online.target
[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
```

Enable and Start Prometheus Service:

```
sudo systemctl enable prometheus.service
sudo systemctl start prometheus.service
systemctl status prometheus.service
```

Access Prometheus Web Interface:

- Open Monitoring Server Public IP with port 9090 in your browser

Install and Configure Node Exporter:

Create Node Exporter User:

```
sudo useradd -r -s /bin/false node_exporter
```

Download and Extract Node Exporter:

```
wget https://github.com/prometheus/node_exporter/releases/dow
nload/v1.7.0/node_exporter-1.7.0.linux-amd64.tar.gz
```

```
tar -xvf node_exporter-1.7.0.linux-amd64.tar.gz
sudo mv node_exporter-1.7.0.linux-amd64/node_exporter /usr/lo
cal/bin/
```

Create systemd Configuration for Node Exporter:

- Edit `/etc/systemd/system/node_exporter.service` and paste the provided configuration.

```
[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 Service:

```
sudo systemctl enable node_exporter
sudo systemctl start node_exporter
systemctl status node_exporter.service
```

Update Prometheus Configuration:

Edit Prometheus Configuration:**

- Edit `/etc/prometheus/prometheus.yml` and add a new target for Node Exporter.

```
 - job_name: "node_exporter"
    static_configs:
      - targets: ["localhost:9100"]
```

Validate and Reload Prometheus Configuration:

promtool check config /etc/prometheus/prometheus.yml

```
curl -X POST http://localhost:9090/-/reload
```

Verify Node Exporter in Prometheus:

Now, go to your Prometheus server and this time, you will see one more target section as node_exporter which should be up and running.

Now, install the Grafana tool to visualize all the data that is coming with the help of Prometheus.

Install and Configure Grafana:

Install Dependencies:

```
sudo apt-get install -y apt-transport-https software-properti
es-common wget
```

Add Grafana GPG Key and Repository:

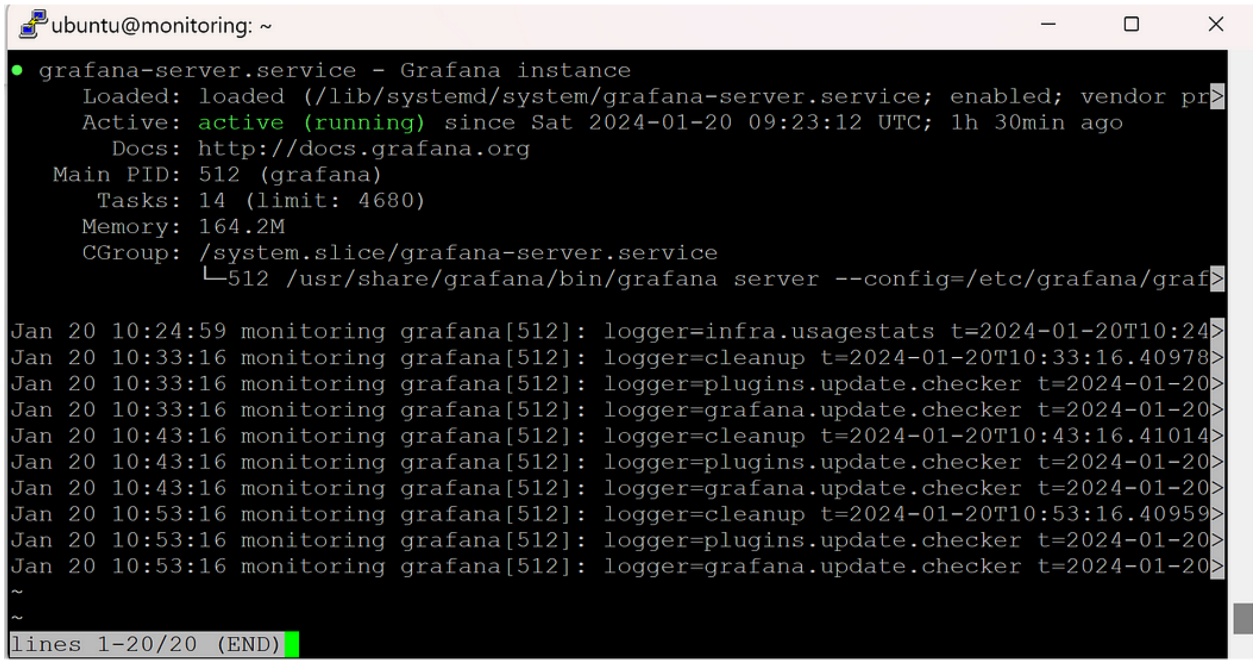
```
sudo mkdir -p /etc/apt/keyrings/
wget -q -O - https://apt.grafana.com/gpg.key | gpg - dearmor
| sudo tee /etc/apt/keyrings/grafana.gpg > /dev/null
echo "deb [signed-by=/etc/apt/keyrings/grafana.gpg] https://a
pt.grafana.com stable main" | sudo tee -a /etc/apt/sources.li
st.d/grafana.list
```

Install Grafana:

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

Enable and Start Grafana Service:

```
sudo systemctl enable grafana-server.service
sudo systemctl start grafana-server.service
systemctl status grafana-server.service
```



ubuntu@monitoring: ~

```
● grafana-server.service - Grafana instance
     Loaded: loaded (/lib/systemd/system/grafana-server.service; enabled; vendor pr>
     Active: active (running) since Sat 2024-01-20 09:23:12 UTC; 1h 30min ago
       Docs: http://docs.grafana.org
   Main PID: 512 (grafana)
      Tasks: 14 (limit: 4680)
     Memory: 164.2M
     CGroup: /system.slice/grafana-server.service
             └─512 /usr/share/grafana/bin/grafana server --config=/etc/grafana/graf>

Jan 20 10:24:59 monitoring grafana[512]: logger=infra.usagestats t=2024-01-20T10:24>
Jan 20 10:33:16 monitoring grafana[512]: logger=cleanup t=2024-01-20T10:33:16.40978>
Jan 20 10:33:16 monitoring grafana[512]: logger=plugins.update.checker t=2024-01-20>
Jan 20 10:33:16 monitoring grafana[512]: logger=grafana.update.checker t=2024-01-20>
Jan 20 10:43:16 monitoring grafana[512]: logger=cleanup t=2024-01-20T10:43:16.41014>
Jan 20 10:43:16 monitoring grafana[512]: logger=plugins.update.checker t=2024-01-20>
Jan 20 10:43:16 monitoring grafana[512]: logger=grafana.update.checker t=2024-01-20>
Jan 20 10:53:16 monitoring grafana[512]: logger=cleanup t=2024-01-20T10:53:16.40959>
Jan 20 10:53:16 monitoring grafana[512]: logger=plugins.update.checker t=2024-01-20>
Jan 20 10:53:16 monitoring grafana[512]: logger=grafana.update.checker t=2024-01-20>
~
~
lines 1-20/20 (END)
```

Access Grafana Web Interface:

Copy Monitoring Server Public IP and open it in your browser with port 3000.

Username: admin

Password: admin

Reset Password:

Change the password for security.

Configure Prometheus Data Source:

Go to "**Data sources**" in Grafana.

Select Prometheus.

Provide Monitoring Server Public IP with port 9090.

Click on Save and test.

Import Node Exporter Dashboard:

Go to “**Dashboard**” → “Import.”

Add **1860** for the Node Exporter dashboard.

Select Prometheus as the data source.

Click on Import.

Monitor Jenkins Server:

Install Prometheus metric plugin on Jenkins.

Edit /etc/prometheus/prometheus.yml and add a job for Jenkins.

```
- job_name: "jenkins"
    static_configs:
      - targets: ["<jenkins-server-public-ip>:8080"]
    metrics_path: "/prometheus"
```

Validate the Prometheus configuration.

```
promtool check config /etc/prometheus/prometheus.yml
```

Reload Prometheus configuration:

```
curl -X POST http://localhost:9090/-/reload
```

Access Prometheus targets in the browser.

Import Jenkins Dashboard to Grafana:

Go to “**Dashboard**” → “Import.”

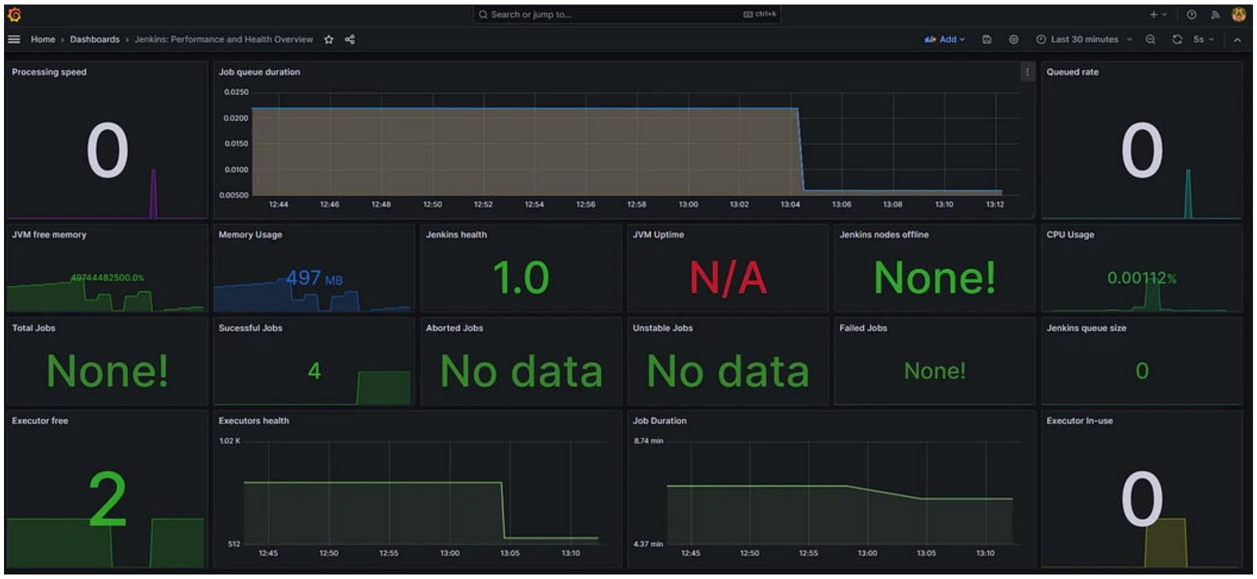
Add **9964** for the Jenkins Monitoring dashboard.

Select Prometheus as the data source.

Click on Import.

View Dashboards:

You will see your Node Exporter and Jenkins Monitoring dashboards in Grafana.



Setting Up Jenkins Email Alerts:

Install Email Extension Template Plugin:

Navigate to “Manage Jenkins” → “Plugins.”

Install the “Email Extension Template” plugin.

Create App Password for Gmail:

Visit your Gmail account and go to “Manage account.”

Under the Security section, find “App passwords” and click on it.

Gmail will ask for your password; provide it.

Specify your app’s name for integrating the email service.

Save the generated password securely.

Add Gmail Credentials in Jenkins:

Go to “Manage Jenkins” → “Credentials.”

Click on “(global)” and then “Add credentials.”

Choose “Username with password” as the kind.

Provide your email ID and the generated app password.

Set the ID as “mail” to easily reference both credentials.

Configure Email Notifications:

Navigate to “Jenkins” → “Manage Jenkins” → “System.”

Search for “Extend E-mail Notification.”

Provide “smtp.gmail.com” as the SMTP server and “465” as the SMTP port.

Advanced>Credentials select created credentials

Choose option “Use SSL”

Under **E-mail Notification**

Select “Use SMTP Authentication” and enter your Gmail ID and the generated app password in the Username and Password fields.

Choose option Test configuration by sending test e-mail

Verify the email configuration by sending a test email.

Setting Up Jenkins Pipeline:

Install Required Plugins:

Download and install the following plugins:

Eclipse Temurin installer

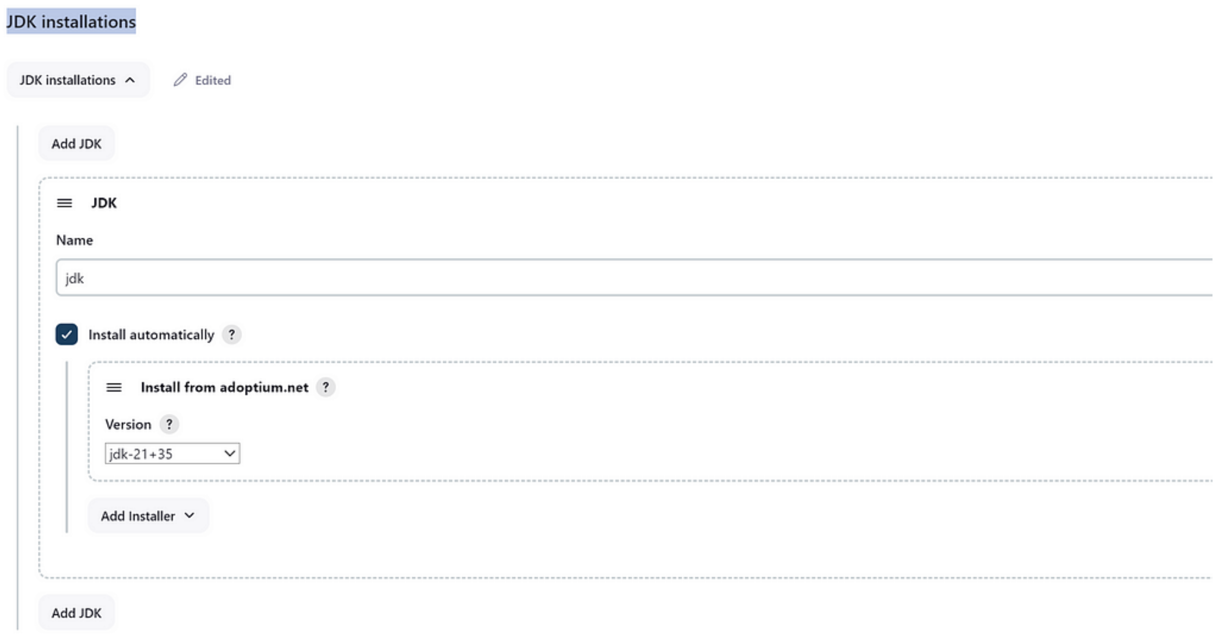
SonarQube Scanner

NodeJS

Configure Plugins:

Go to “Manage Jenkins” → “Tools.”

Add JDK by providing necessary details.



JDK installations

JDK installations

Add JDK

JDK

Name: jdk

[✓] Install automatically

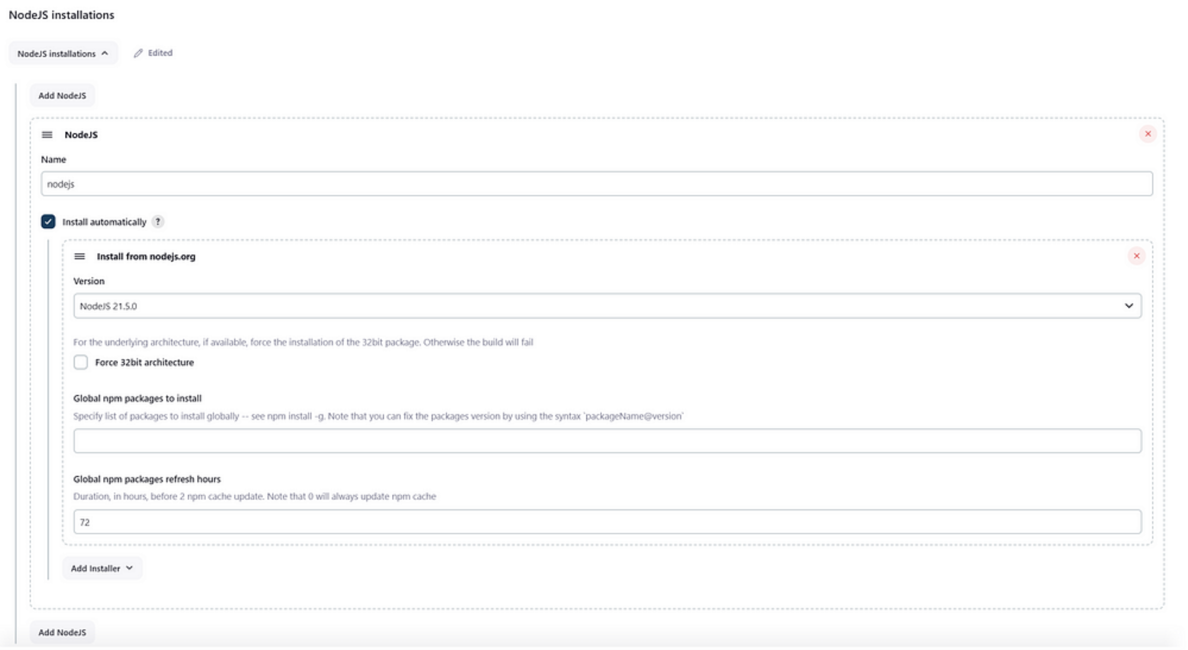
Install from adoptium.net

Version: jdk-21+35

Add Installer

Add JDK

Add NodeJS by providing the required information.



NodeJS installations

NodeJS installations

Add NodeJS

NodeJS

Name: nodejs

[✓] Install automatically

Install from nodejs.org

Version: NodeJS 21.5.0

For the underlying architecture, if available, force the installation of the 32bit package. Otherwise the build will fail

[ ] Force 32bit architecture

Global npm packages to install

Specify list of packages to install globally -- see npm install -g. Note that you can fix the packages version by using the syntax `packageName@version`

Global npm packages refresh hours

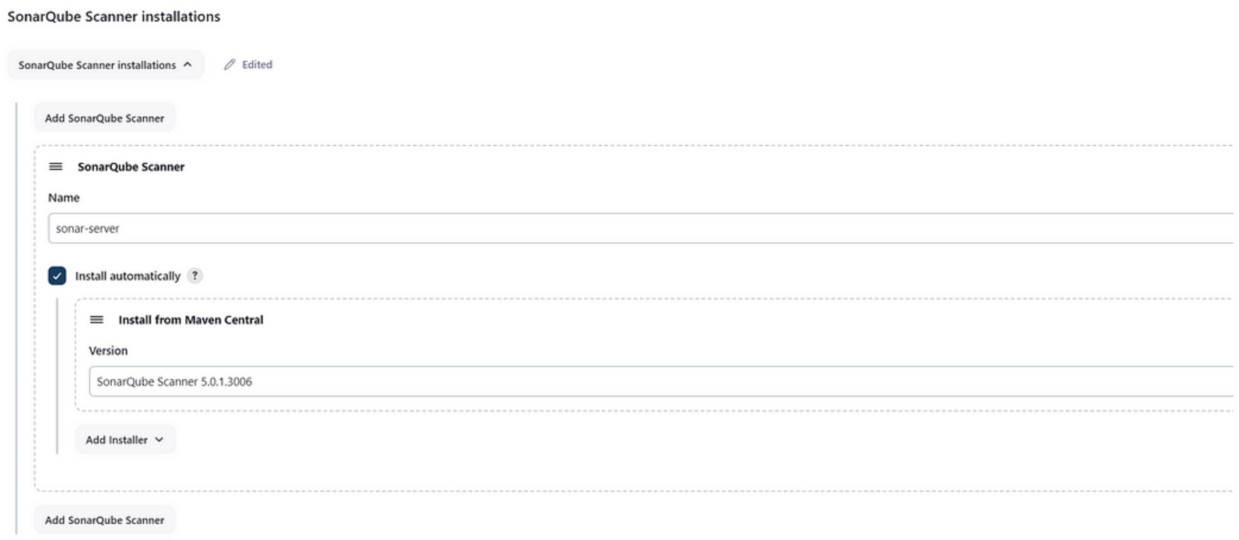
Duration, in hours, before 2 npm cache update. Note that 0 will always update npm cache

72

Add Installer

Add NodeJS

Configuring SonarQube:


SonarQube Scanner installations

SonarQube Scanner installations

Add SonarQube Scanner

SonarQube Scanner

Name: sonar-server

Install automatically ✓

Install from Maven Central

Version: SonarQube Scanner 5.0.1.3006

Add Installer

Add SonarQube Scanner

Follow the steps for installing and configuring the SonarQube plugin. Ensure it is integrated into your Jenkins setup.

Configuring SonarQube:

Access SonarQube:

Copy the public IP of your Jenkins Server along with port 9000. Navigate to SonarQube, click on “Adminstration”>”Security,” and then on “Users.”

Generate Token:

Click the highlighted blue box on the right to generate a token.

Provide a name for your token and click “Generate.”

Copy the generated token and store it securely.

Add Token to Jenkins Credentials:

Go to “Manage Jenkins” → “Credentials.”

Select “Secret text” in Kind.

Provide your SonarQube token, and set the ID as “sonar” for future reference.

Configure SonarQube Server in Jenkins:

Go to “Manage Jenkins” → “System.”

Click on “Add Sonarqube.”

Provide the name “sonar-server,” set the Server URL, and choose the credentials you added.

Create Webhook in SonarQube:

Navigate to SonarQube, click on “Configuration,” and select “Webhooks.”

Click “Create.”

Provide a name, Jenkins URL, and click “Create.”

> Ex: http://34.207.155.151:8080/sonarqube-webhook

Create SonarQube Project:

Click “Manually” to create a project.

Provide your project name and click “Set up.”

Select the existing token and continue.

Choose “Other” as your build tool and “Linux” as the operating system.

Create Jenkins Pipeline:

Open Jenkins Web

Create a New Pipeline:

On the Jenkins dashboard, click on “New Item” or “Create a job.”

Enter a name for your pipeline (e.g., Netflix-Clone) and choose “Pipeline” as the type.

Configure Pipeline Source:

Under “Pipeline,” select “Pipeline script from SCM.”

Select SCM (Source Code Management):

Choose “Git” as the SCM.

Specify Repository URL:

Enter the GitHub repository URL

Credentials:

Click on “Add” under “Credentials.”

Enter your GitHub username and provide the corresponding password or access token.

Branches to Build:

Specify the branch to build, such as */main.

Repository Browser:

Choose “(Auto)” for the repository browser.

Script Path:

Enter “Jenkinsfile” as the script path. This assumes that your pipeline script is named “Jenkinsfile” and located at the root of the GitHub repository.

Leave everything default.

Set up the pipeline for SonarQube analysis, quality gate check, and dependency installation.

In the post-build, add email alerts for pipeline success or failure.

```
pipeline{
    agent any
    tools{
        jdk 'jdk'
        nodejs 'nodejs'
    }
    environment {
        SCANNER_HOME=tool 'sonar-server'
    }
    stages {
        stage('Workspace Cleaning'){
            steps{
                cleanWs()
            }
        }
        stage('Checkout from Git'){
            steps{
                git branch: 'main', credentialsId: 'github', 
url: 'https://github.com/N4si/DevSecOps-Project'
```

```
             }
         }
         stage("Sonarqube Analysis"){
             steps{
                 withSonarQubeEnv('sonar-sever') {
                     sh ''' $SCANNER_HOME/bin/sonar-scanner -D
sonar.projectName=Netflix \
                     -Dsonar.projectKey=Netflix \
                     '''
                 }
             }
         }
         stage("Quality Gate"){
            steps {
                 script {
                     waitForQualityGate abortPipeline: false,
credentialsId: 'sonar'
                 }
             }
         }
         stage('Install Dependencies') {
             steps {
                 sh "npm install"
             }
         }
post {
     always {
        emailext attachLog: true,
            subject: "'${currentBuild.result}'",
            body: "Project: ${env.JOB_NAME}<br/>" +
                "Build Number: ${env.BUILD_NUMBER}<br/>" +
                "URL: ${env.BUILD_URL}<br/>",
            to: 'example@gmail.com',
            attachmentsPattern: 'trivyfs.txt,trivyimage.txt'
        }
        }
    }
```

Click on build pipeline and after getting the success of the pipeline.

You will see the Sonarqube code quality analysis which will look like the below snippet.

Install OWASP Dependency-Check Plugin:

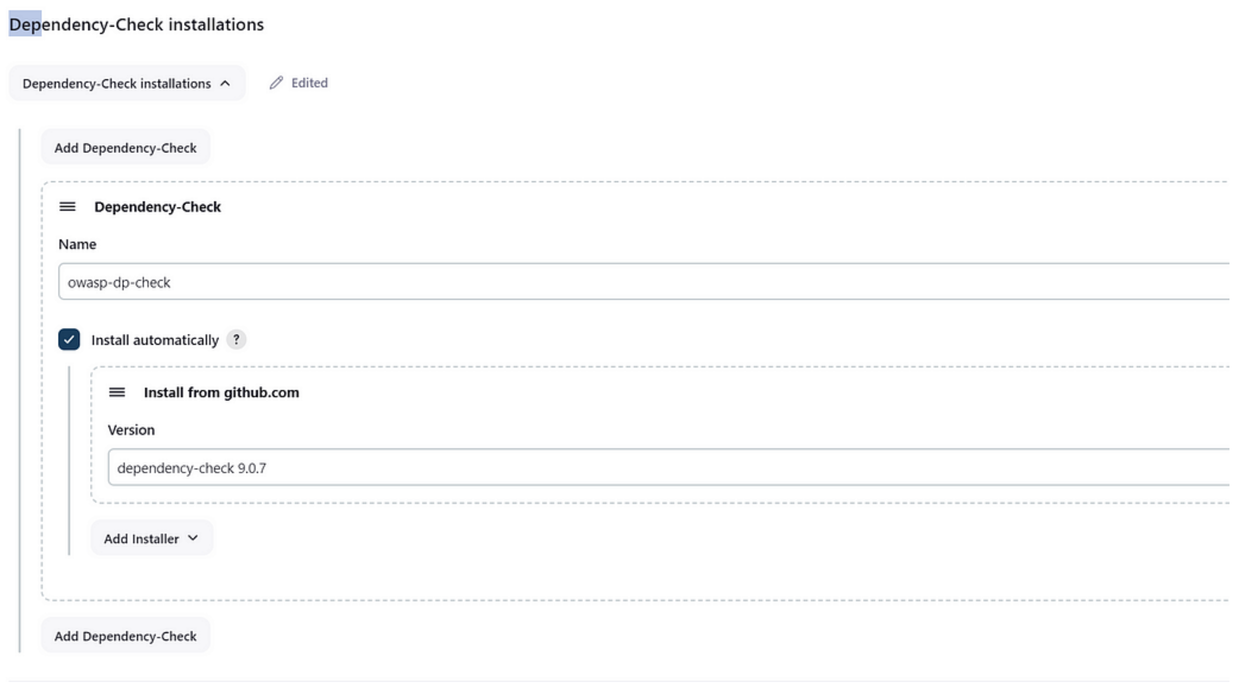
Go to “Manage Jenkins” → “Plugins.”

Search for “OWASP Dependency-Check” and install it.

Configure OWASP Dependency-Check:

After installing, go to “Manage Jenkins” → “Tools”

Provide a name, select the latest version of OWASP, and click “Save.”


Dependency-Check installations

Name: owasp-dp-check

✓ Install automatically

Install from github.com

Version: dependency-check 9.0.7

Add OWASP Dependency-Check and Trivy filesystem scan Stage to Jenkins Pipeline:

```
stage('OWASP DP SCAN') {
            steps {
                dependencyCheck additionalArguments: '--scan
./ --disableYarnAudit --disableNodeAudit', odcInstallation:
'owasp-dp-check'
                dependencyCheckPublisher pattern: '**/depende
ncy-check-report.xml'
            }
        }
        stage('TRIVY FS SCAN') {
            steps {
                sh "trivy fs . > trivyfs.txt"
            }
        }
```

Save your Jenkinsfile.

Click on “Build Now” to run the pipeline.

View Dependency Check Results:

Once the pipeline is successful, scroll down to view the OWASP Dependency-Check results in the Jenkins build page.

Click on the Dependency Check link to see detailed results.

Configure Docker Credentials:

Go to “Manage Jenkins” → “Credentials.”

Add DockerHub credentials:

Click on “Add Credentials.”

Choose “Secret text” as the kind.

Provide your DockerHub username and password.

Click on “Create.”

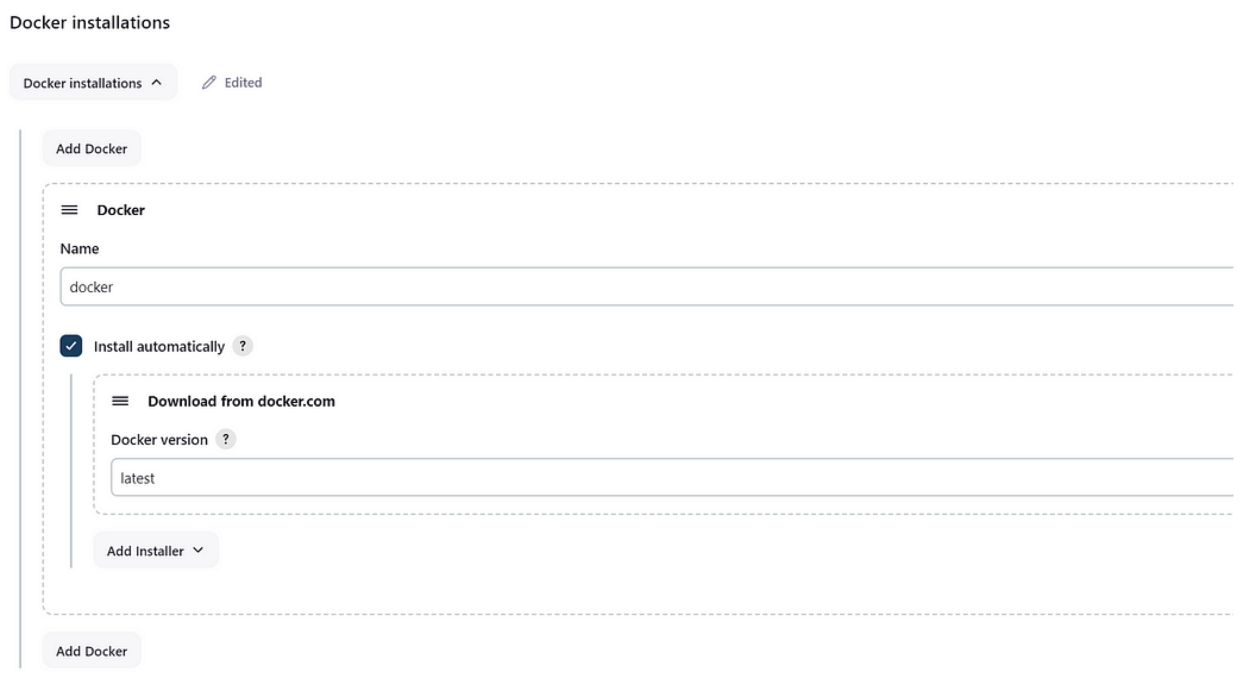
Install the following Docker plugins on your Jenkins

> Docker
>
> *Docker Commons*

> *Docker Pipeline*
>
> *Docker API*
>
> *docker-build-step*

Configure the tool in Jenkins

Go to **Manage Jenkins** → Tools and provide the below details.


Docker installations

Docker installations ^ Edited

Add Docker

Docker

Name: docker

[✓] Install automatically ?

Download from docker.com

Docker version ?: latest

Add Installer

Add Docker

Our application is Netflix Clone. So we need some movie databases on our application.

For that, we have one application that will provide the API. So, we can use the API to get the movies on our application.

Obtain TMDB API Key:

- Go to TMDB website.
- Click on "Join TMDB" and sign up by providing the necessary details.
- Confirm your account by clicking the confirmation link sent to your email.
- Log in to your TMDB account, go to settings, and navigate to the API section.

- Click on “Create” to generate a new API key.
- Select “Developer” and accept the Terms & Conditions.
- Provide the basic details and click on “Submit.”
- Copy the generated API key and store it securely.

Configure Docker Images:

Update your Dockerfile with the TMDB API key. For example:

```
FROM node:16.17.0-alpine as builder
WORKDIR /app
COPY ./package.json .
COPY ./yarn.lock .
RUN yarn install
COPY . .
ARG TMDB_V3_API_KEY
ENV VITE_APP_TMDB_V3_API_KEY=${TMDB_V3_API_KEY}
ENV VITE_APP_API_ENDPOINT_URL="https://api.themoviedb.org/3"
RUN yarn build

FROM nginx:stable-alpine
WORKDIR /usr/share/nginx/html
RUN rm -rf ./*
COPY --from=builder /app/dist .
EXPOSE 80
ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

Configure Docker Image Build in Jenkins:

Update your Jenkinsfile to include Docker image build and push stages.

```
stage("Docker Image Build"){
            steps{
                script{
                   withDockerRegistry(credentialsId: 'docke
r', toolName: 'docker'){
```

```
                       sh "docker system prune -f"
                       sh "docker container prune -f"
                       sh "docker build --build-arg TMDB_V3_A
PI_KEY=your-tmdb-api-key-t netflix ."
                    }
                }
            }
        }
        stage("Docker Image Pushing"){
            steps{
                script{
                   withDockerRegistry(credentialsId: 'docke
r', toolName: 'docker'){
                       sh "docker tag netflix nasi101/netfli
x:latest "
                       sh "docker push nasi101/netflix:latest
"
                    }
                }
            }
        }
        stage("TRIVY Image Scan"){
            steps{
                sh "trivy image nasi101/netflix:latest > triv
yimage.txt"
            }
        }
```

Make sure to replace placeholders like your-dockerhub-username and your-image-name with your actual DockerHub username and image name and your-tmdb-api-key.

Run the Pipeline Again:

Save your Jenkinsfile.

Click on “Build Now” to run the updated pipeline.

View Docker Build Results:

After a successful build, check the Docker build and push stages in the Jenkins build page.

Now, your Jenkins pipeline includes OWASP Dependency-Check, Trivy filesystem scan (optional), Docker build, and Docker push stages.

Now, we have to deploy our application using Kubernetes.

To do that, we need to install kubectl on the Jenkins server.

```
sudo apt update
sudo apt install curl
curl -LO https://dl.k8s.io/release/$(curl -L -s https://dl.k8
s.io/release/stable.txt)/bin/linux/amd64/kubectl
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/k
ubectl
kubectl version --client
```

Instructions to set up Kubernetes on both the Master and Worker Nodes

Add the hostname to your Kubernetes master node

```
sudo hostnamectl set-hostname K8s-Master
```

Add the hostname to your Kubernetes worker nod

```
sudo hostnamectl set-hostname K8s-Worker
```

Run the below commands on the both Master and worker Nodes.

```
sudo su
swapoff -a; sed -i '/swap/d' /etc/fstab
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF
sudo modprobe overlay
sudo modprobe br_netfilter
# sysctl params required by setup, params persist across rebo
ots
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward = 1
EOF
# Apply sysctl params without reboot
sudo sysctl --system
apt update
sudo apt-get install -y apt-transport-https ca-certificates c
url
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg
| gpg --dearmor -o /usr/share/keyrings/kubernetes-archive-key
ring.gpg
echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-k
eyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial mai
n" | sudo tee /etc/apt/sources.list.d/kubernetes.list >/dev/n
ull
apt update
apt-get install -y kubelet kubeadm kubectl kubernetes-cni
apt install docker.io -y
sudo mkdir /etc/containerd
sudo sh -c "containerd config default > /etc/containerd/confi
g.toml"
sudo sed -i 's/ SystemdCgroup = false/ SystemdCgroup = true/'
/etc/containerd/config.toml
systemctl restart containerd.service
systemctl restart kubelet.service
systemctl enable kubelet.service
```

Now, run the following commands **Only on the Master Node,** and then you will get the command that is highlighted in the below snippet

```
kubeadm config images pull
kubeadm init
```

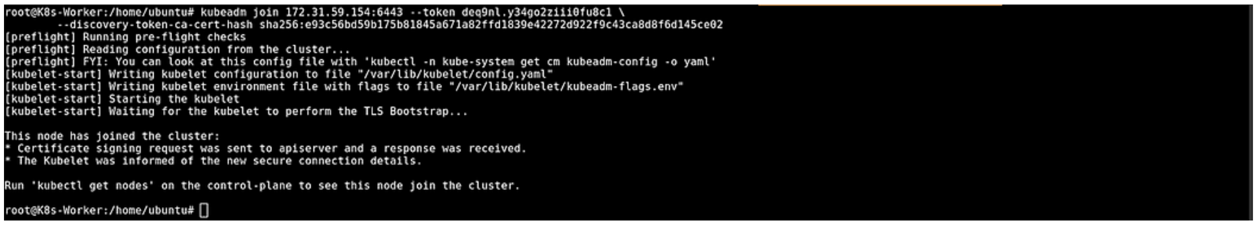
Exit from the root user and run the below commands

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

After running these commands, the Kubernetes Master Node will be initialized, and you'll have a kubeconfig file set up for accessing the cluster.

The output of the `kubeadm init` command will include a `kubeadm join` command that you'll need to run on the Worker Node to join it to the cluster. Look for a line in the output that looks similar to the following:

```
kubeadm join 192.168.0.100:6443 --token <your-token> --discov
ery-token-ca-cert-hash sha256:<your-cert-hash>
```



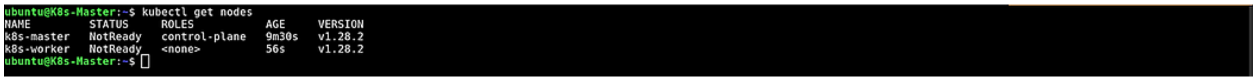
```
root@K8s-Worker:/home/ubuntu# kubeadm join 172.31.59.154:6443 --token deq9nl.y34go2ziii0fu8cl \
        --discovery-token-ca-cert-hash sha256:e93c56bd59b175b81845a671a82ffd1839e42272d922f9c43ca8d8f6d145ce02
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

root@K8s-Worker:/home/ubuntu# 
```

Both nodes are not ready because the network plugin is not installed on the master node



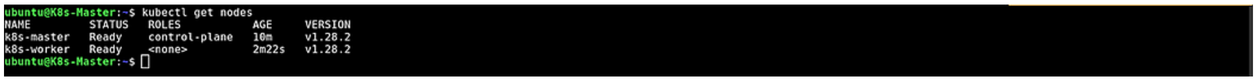
```
ubuntu@K8s-Master:~$ kubectl get nodes
NAME         STATUS     ROLES           AGE     VERSION
k8s-master   NotReady   control-plane   9m30s   v1.28.2
k8s-worker   NotReady   <none>          56s     v1.28.2
ubuntu@K8s-Master:~$ 
```

Only on the Master Node

Run the below command to install the network plugin on the Master node

```
kubectl apply -f https://raw.githubusercontent.com/projectcal
ico/calico/v3.25.0/manifests/calico.yaml
```

Both nodes are ready.



```
ubuntu@K8s-Master:~$ kubectl get nodes
NAME         STATUS   ROLES           AGE     VERSION
k8s-master   Ready    control-plane   10m     v1.28.2
k8s-worker   Ready    <none>          2m22s   v1.28.2
ubuntu@K8s-Master:~$
```

Install the following Kubernetes Plugins on your Jenkins.

```
Kubernetes
Kubernetes Credentials
Kubernetes Client API
Kubernetes CLI
Kubernetes Credential Provider
```

Let's set up Kubernetes monitoring for both the Master and Worker Nodes using Prometheus and Node Exporter. Follow the steps below:

Create Prometheus User:

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

Download and Install Node Exporter:

```
wget https://github.com/prometheus/node_exporter/releases/dow
nload/v1.7.0/node_exporter-1.7.0.linux-amd64.tar.gz
tar -xvf node_exporter-1.7.0.linux-amd64.tar.gz
sudo mv node_exporter-1.7.0.linux-amd64/node_exporter /usr/lo
cal/bin/
```

Create Node Exporter systemd Configuration:

sudo vim /etc/systemd/system/node_exporter.service

```
[Unit]
Description=Node Exporter
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/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 Node Exporter Status:

```
systemctl status node_exporter.service
```

Edit Prometheus Configuration:

```
sudo vim /etc/prometheus/prometheus.yml
```

Add job configurations for both Master and Worker nodes:

```
- job_name: 'k8s-master'
  static_configs:
    - targets: ['k8s-master-ip:9100']
- job_name: 'k8s-worker'
  static_configs:
    - targets: ['k8s-worker-ip:9100']
```

Replace ‘k8s-master-ip’ and ‘k8s-worker-ip’ with the actual IPs of your Kubernetes Master and Worker Nodes.

Validate and Reload Configuration:

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

configure Jenkins to deploy an application to a Kubernetes cluster:

```
cat .kube/config
```

Save the contents to a text file (e.g., kube-config.txt)

On Jenkins:

Add Kubernetes Configuration as a Jenkins Secret:

Go to Jenkins Dashboard.

Click on “Credentials” in the left menu.

Click on “(global)” → “Add Credentials.”

Select “Secret file” as the kind.

Provide the path to the kube-config.txt file.

Enter an ID (e.g., k8s) and click on “Create.”

Update Jenkins Pipeline:

Open your Jenkinsfile

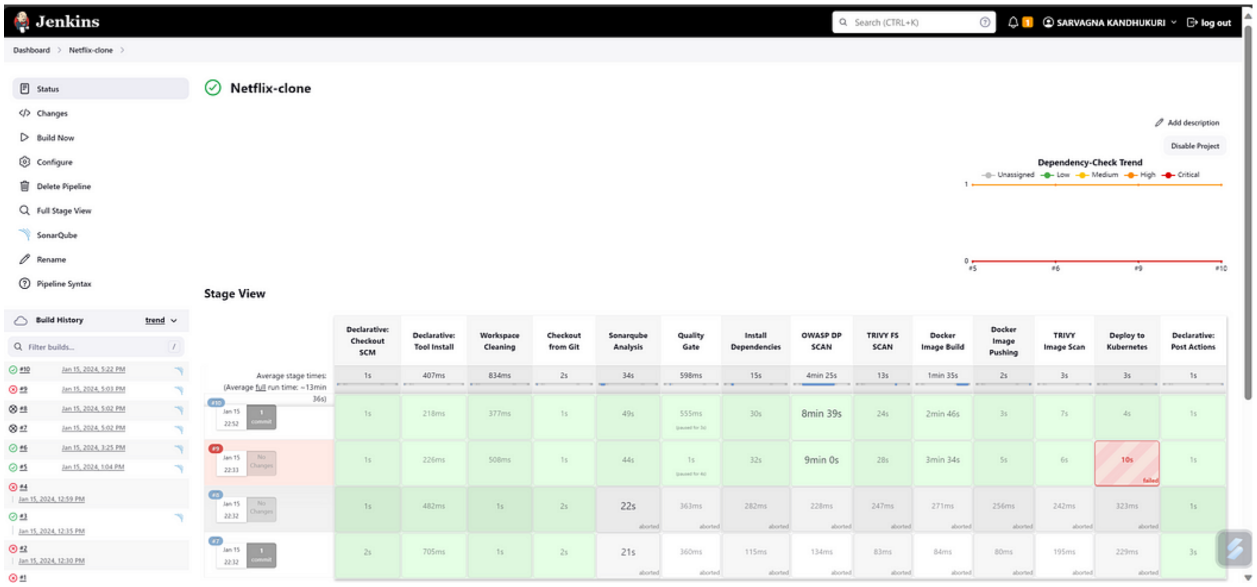
Add the deploy to Kubernetes stage:

```
stage('Deploy to Kubernetes'){
            steps{
                script{
                    dir('Kubernetes') {
                        withKubeConfig(caCertificate: '', clusterName: '', contextName: '', credentialsId: 'k8s', namespace: '', restrictKubeConfigAccess: false, serverUrl: '') {
                                sh 'kubectl apply -f deployment.yml'
                                sh 'kubectl apply -f service.yml'
                                sh 'kubectl get svc'
                                sh 'kubectl get all'
                        }
                    }
                }
            }
        }
    }
```

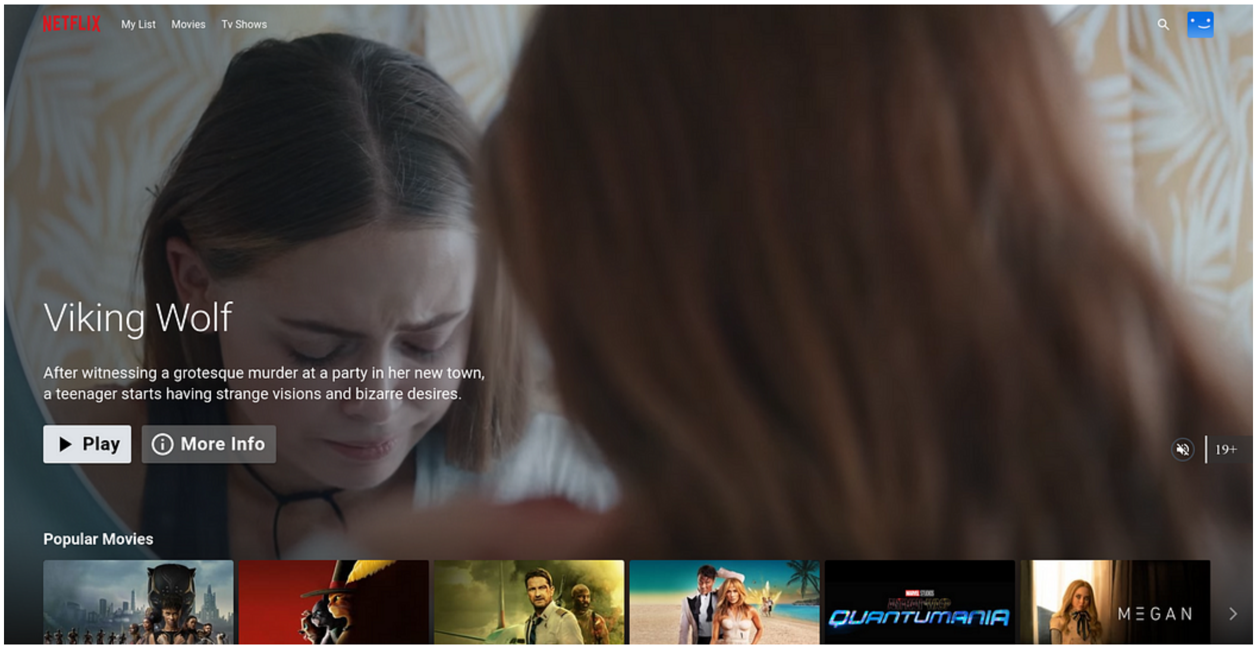
Click on **Build Now**

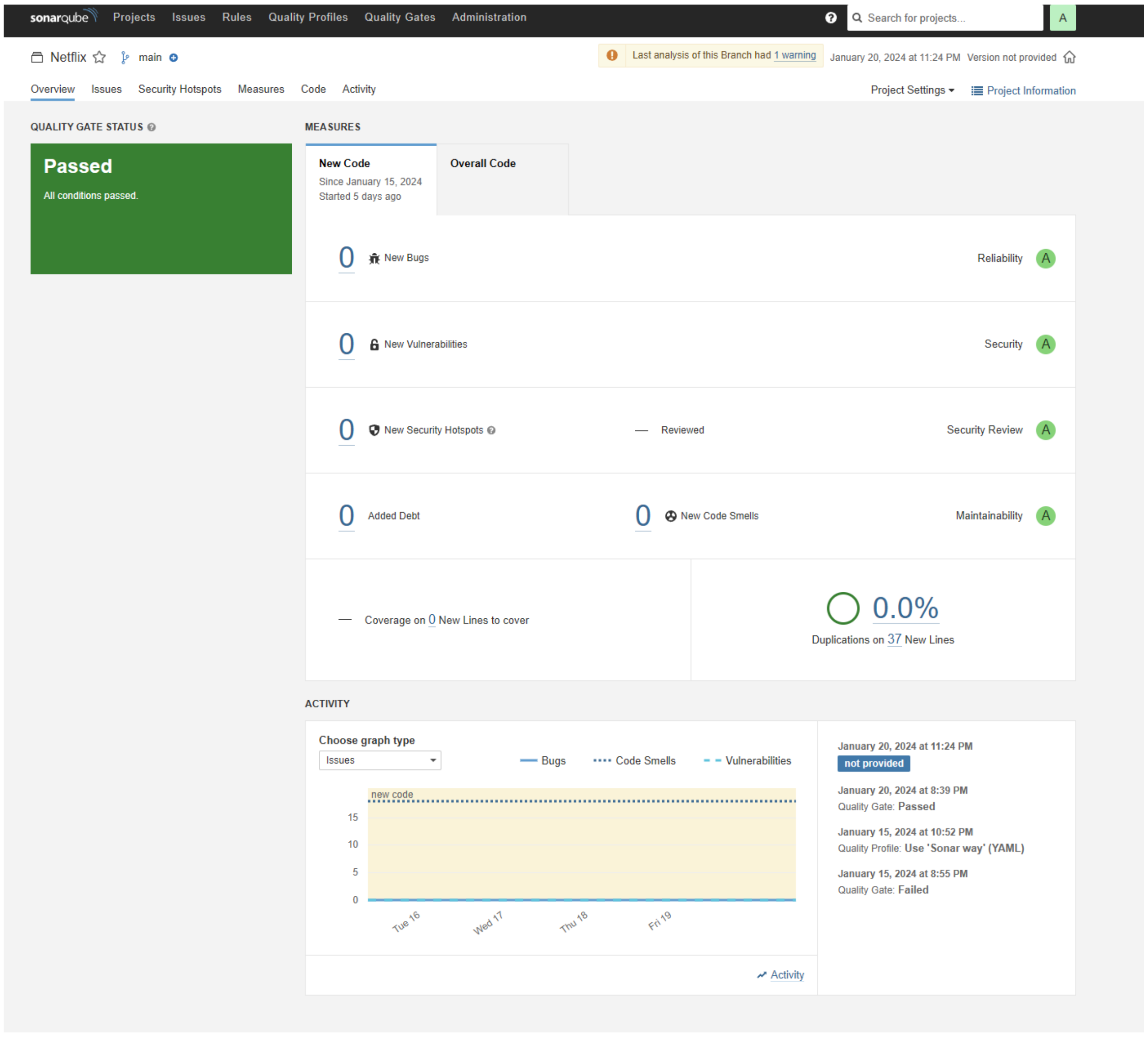
You will see that our application has been deployed successfully on Kubernetes.

In your browser open you Kubernetes worker External IP address with port 32000



	Declarative: Checkout SCM	Declarative: Tool Install	Workspace Cleaning	Checkout from Git	Sonarqube Analysis	Quality Gate	Install Dependencies	OWASP DP SCAN	TRIVY FS SCAN	Docker Image Build	Docker Image Pushing	TRIVY Image Scan	Deploy to Kubernetes	Declarative: Post Actions
Average stage times: (Average full run time: ~13min 36s)	1s	407ms	834ms	2s	34s	598ms	15s	4min 25s	13s	1min 35s	2s	3s	3s	1s
#10 Jan 15 22:52 1 commit	1s	218ms	377ms	1s	49s	555ms	30s	8min 39s	24s	2min 46s	3s	7s	4s	1s
#9 Jan 15 22:33 No Changes	1s	226ms	508ms	1s	44s	1s	32s	9min 0s	28s	3min 34s	5s	6s	10s failed	1s
#8 Jan 15 22:32 No Changes	1s	482ms	1s	2s	22s aborted	363ms aborted	282ms aborted	228ms aborted	247ms aborted	271ms aborted	256ms aborted	242ms aborted	323ms aborted	1s
#7 Jan 15 22:32 1 commit	2s	705ms	1s	2s	21s aborted	360ms aborted	115ms aborted	134ms aborted	83ms aborted	84ms aborted	80ms aborted	195ms aborted	229ms aborted	3s


NETFLIX
My List
Movies
Tv Shows
Viking Wolf
After witnessing a grotesque murder at a party in her new town, a teenager starts having strange visions and bizarre desires.
Play
More Info
19+
Popular Movies
QUANTUMANIA
M3GAN



sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration

Search for projects...

Netflix main

Last analysis of this Branch had 1 warning January 20, 2024 at 11:24 PM Version not provided

Overview Issues Security Hotspots Measures Code Activity Project Settings Project Information

QUALITY GATE STATUS

Passed

All conditions passed.

MEASURES

New Code
Since January 15, 2024
Started 5 days ago

Overall Code

0 New Bugs — Reliability A

0 New Vulnerabilities — Security A

0 New Security Hotspots — Reviewed — Security Review A

0 Added Debt — 0 New Code Smells — Maintainability A

— Coverage on 0 New Lines to cover

0.0% Duplications on 37 New Lines

ACTIVITY

Choose graph type: Issues

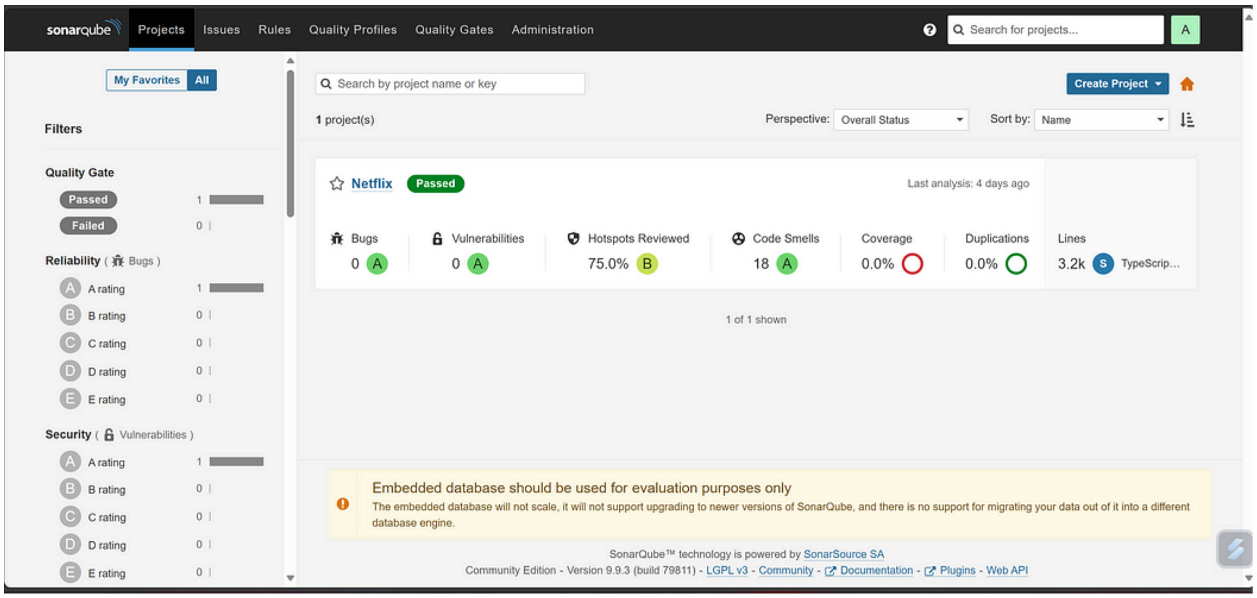
Activity

January 20, 2024 at 11:24 PM
not provided

January 20, 2024 at 8:39 PM
Quality Gate: Passed

January 15, 2024 at 10:52 PM
Quality Profile: Use 'Sonar way' (YAML)

January 15, 2024 at 8:55 PM
Quality Gate: Failed



Search by project name or key

1 project(s)   Perspective: Overall Status   Sort by: Name

Filters

Quality Gate
Passed 1
Failed 0

Reliability (Bugs)
A rating 1
B rating 0
C rating 0
D rating 0
E rating 0

Security (Vulnerabilities)
A rating 1
B rating 0
C rating 0
D rating 0
E rating 0

Netflix Passed   Last analysis: 4 days ago

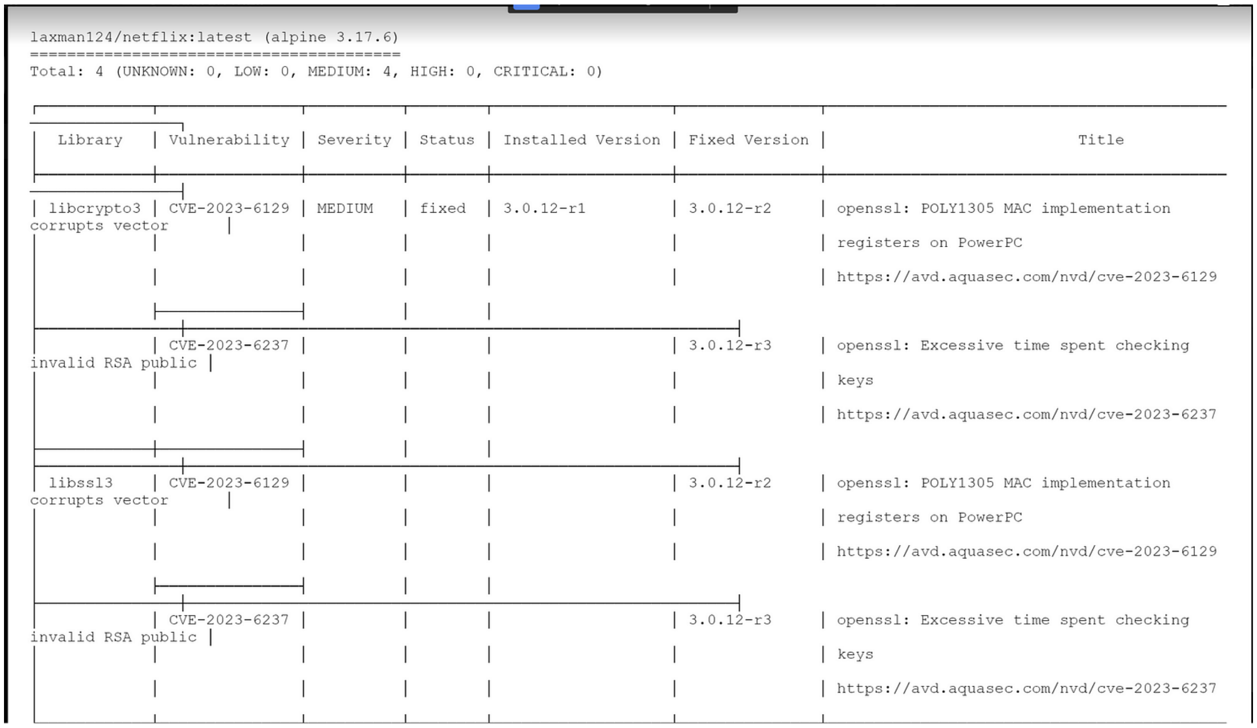
Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications	Lines
0 A	0 A	75.0% B	18 A	0.0%	0.0%	3.2k S TypeScrip...

1 of 1 shown

Embedded database should be used for evaluation purposes only

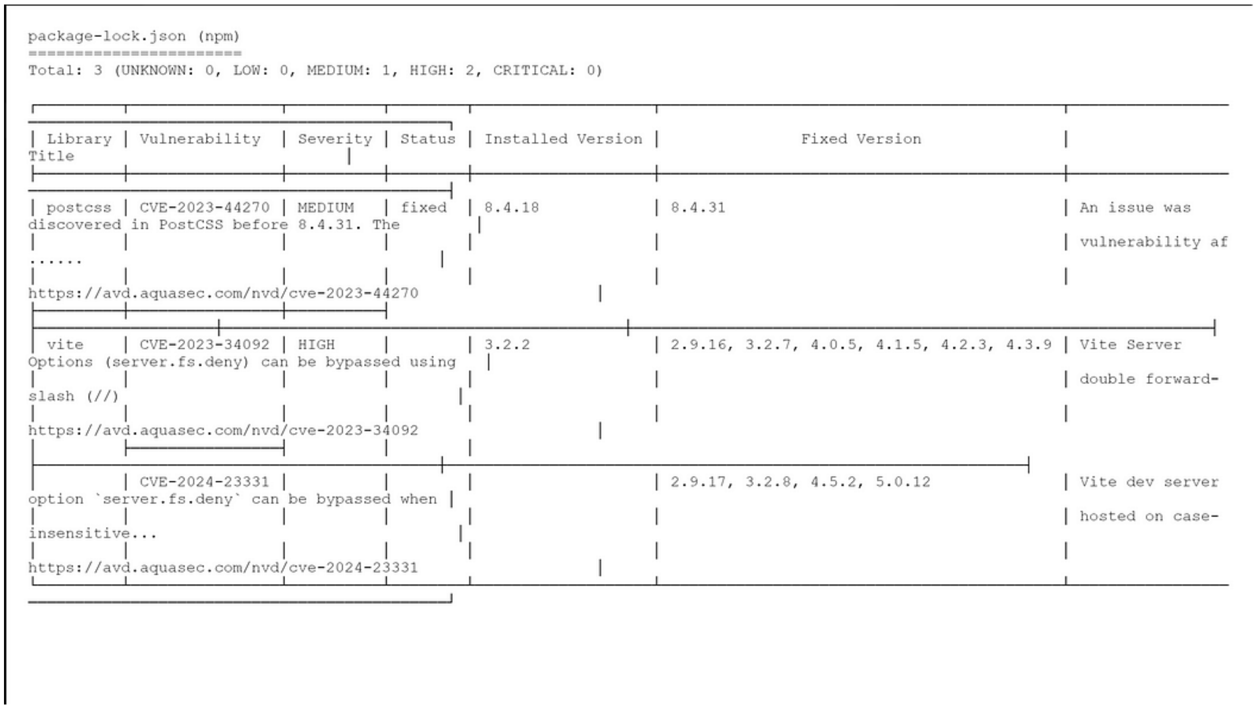
The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.

SonarQube™ technology is powered by SonarSource SA
Community Edition - Version 9.9.3 (build 79811) - LGPL v3 - Community - Documentation - Plugins - Web API



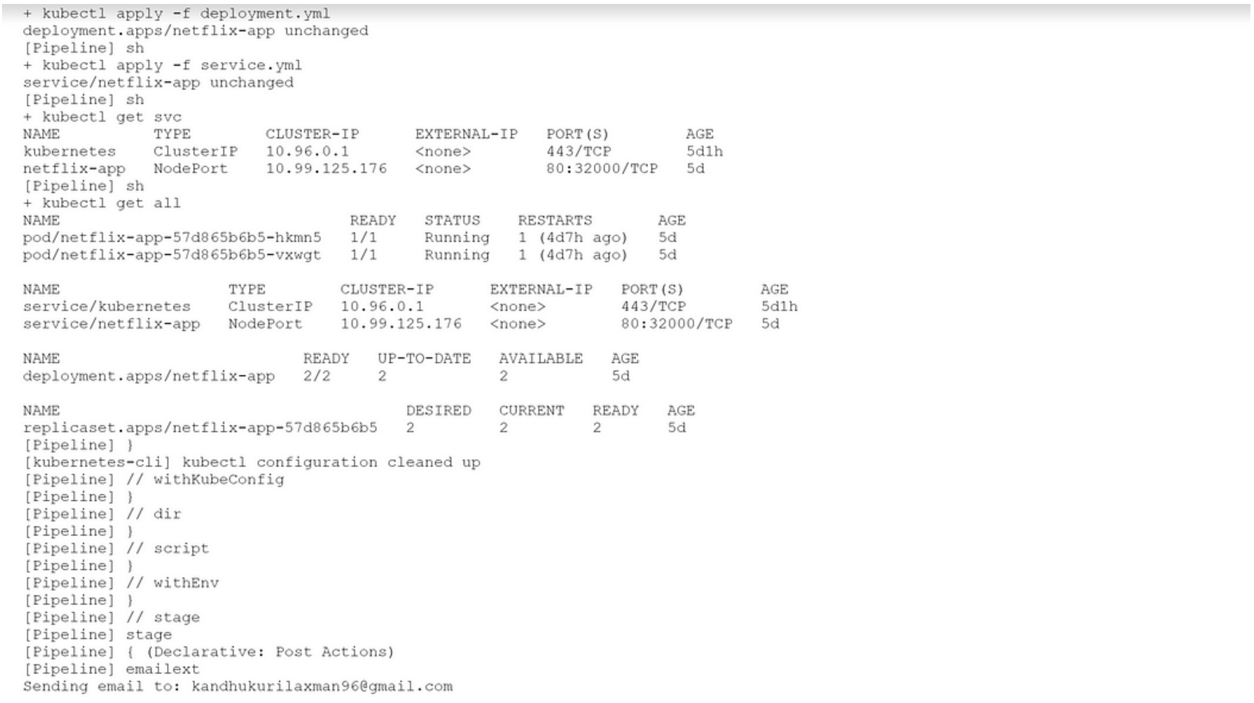
```
laxman124/netflix:latest (alpine 3.17.6)
========================================
Total: 4 (UNKNOWN: 0, LOW: 0, MEDIUM: 4, HIGH: 0, CRITICAL: 0)

┌────────────┬────────────────┬──────────┬────────┬───────────────────┬───────────────┬──────────────────────────────────────────────┐
│  Library   │ Vulnerability  │ Severity │ Status │ Installed Version │ Fixed Version │                    Title                     │
├────────────┼────────────────┼──────────┼────────┼───────────────────┼───────────────┼──────────────────────────────────────────────┤
│ libcrypto3 │ CVE-2023-6129  │ MEDIUM   │ fixed  │ 3.0.12-r1         │ 3.0.12-r2     │ openssl: POLY1305 MAC implementation corrupts vector │
│            │                │          │        │                   │               │ registers on PowerPC                         │
│            │                │          │        │                   │               │ https://avd.aquasec.com/nvd/cve-2023-6129    │
│            ├────────────────┤          │        │                   ├───────────────┼──────────────────────────────────────────────┤
│            │ CVE-2023-6237  │          │        │                   │ 3.0.12-r3     │ openssl: Excessive time spent checking invalid RSA public │
│            │                │          │        │                   │               │ keys                                         │
│            │                │          │        │                   │               │ https://avd.aquasec.com/nvd/cve-2023-6237    │
├────────────┼────────────────┤          │        │                   ├───────────────┼──────────────────────────────────────────────┤
│ libssl3    │ CVE-2023-6129  │          │        │                   │ 3.0.12-r2     │ openssl: POLY1305 MAC implementation corrupts vector │
│            │                │          │        │                   │               │ registers on PowerPC                         │
│            │                │          │        │                   │               │ https://avd.aquasec.com/nvd/cve-2023-6129    │
│            ├────────────────┤          │        │                   ├───────────────┼──────────────────────────────────────────────┤
│            │ CVE-2023-6237  │          │        │                   │ 3.0.12-r3     │ openssl: Excessive time spent checking invalid RSA public │
│            │                │          │        │                   │               │ keys                                         │
│            │                │          │        │                   │               │ https://avd.aquasec.com/nvd/cve-2023-6237    │
└────────────┴────────────────┴──────────┴────────┴───────────────────┴───────────────┴──────────────────────────────────────────────┘
```



```
package-lock.json (npm)
=======================
Total: 3 (UNKNOWN: 0, LOW: 0, MEDIUM: 1, HIGH: 2, CRITICAL: 0)

| Library | Vulnerability  | Severity | Status | Installed Version |              Fixed Version                |
Title              |
| postcss | CVE-2023-44270 | MEDIUM   | fixed  | 8.4.18            | 8.4.31                                    | An issue was
discovered in PostCSS before 8.4.31. The   |
|         |                |          |        |                   |                                           | vulnerability af
......                                   |
|         |                |          |        |                   |                                           |
https://avd.aquasec.com/nvd/cve-2023-44270        |
| vite    | CVE-2023-34092 | HIGH     |        | 3.2.2             | 2.9.16, 3.2.7, 4.0.5, 4.1.5, 4.2.3, 4.3.9 | Vite Server
Options (server.fs.deny) can be bypassed using   |
|         |                |          |        |                   |                                           | double forward-
slash (//)                                |
|         |                |          |        |                   |                                           |
https://avd.aquasec.com/nvd/cve-2023-34092        |
|         | CVE-2024-23331 |          |        |                   | 2.9.17, 3.2.8, 4.5.2, 5.0.12              | Vite dev server
option `server.fs.deny` can be bypassed when |
|         |                |          |        |                   |                                           | hosted on case-
insensitive...                            |
|         |                |          |        |                   |                                           |
https://avd.aquasec.com/nvd/cve-2024-23331        |
```



```
+ kubectl apply -f deployment.yml
deployment.apps/netflix-app unchanged
[Pipeline] sh
+ kubectl apply -f service.yml
service/netflix-app unchanged
[Pipeline] sh
+ kubectl get svc
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes    ClusterIP   10.96.0.1       <none>        443/TCP        5d1h
netflix-app   NodePort    10.99.125.176   <none>        80:32000/TCP   5d
[Pipeline] sh
+ kubectl get all
NAME                               READY   STATUS    RESTARTS      AGE
pod/netflix-app-57d865b6b5-hkmn5   1/1     Running   1 (4d7h ago)   5d
pod/netflix-app-57d865b6b5-vxwgt   1/1     Running   1 (4d7h ago)   5d

NAME                  TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
service/kubernetes    ClusterIP   10.96.0.1       <none>        443/TCP        5d1h
service/netflix-app   NodePort    10.99.125.176   <none>        80:32000/TCP   5d

NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/netflix-app   2/2     2            2           5d

NAME                                     DESIRED   CURRENT   READY   AGE
replicaset.apps/netflix-app-57d865b6b5   2         2         2       5d
[Pipeline] }
[kubernetes-cli] kubectl configuration cleaned up
[Pipeline] // withKubeConfig
[Pipeline] }
[Pipeline] // dir
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] emailext
Sending email to: kandhukurilaxman96@gmail.com
```