DevSecOps Pipeline Project: Deploy Netflix Clone on Kubernetes

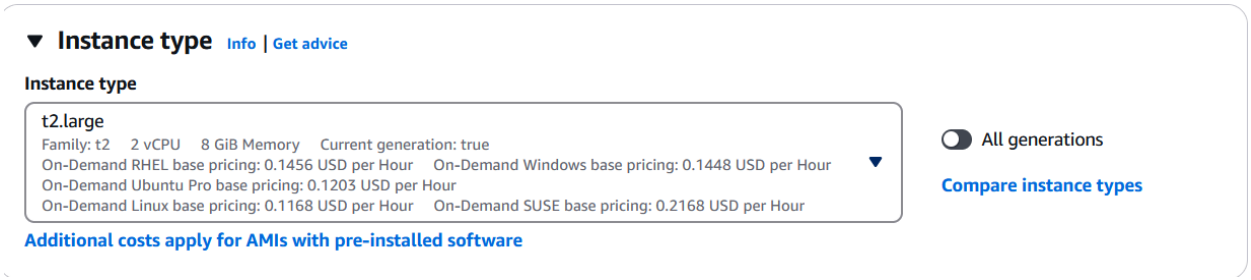
For this project, I will deploy a Netflix clone application on AWS using DevSecOps practices, integrating security from the start with tools like SonarQube, Trivy, and OWASP. The deployment pipeline will use CI/CD with Jenkins, monitoring with Prometheus and Grafana, and Kubernetes orchestration using ArgoCD and Helm.

DevSecOps Phase 1: DEV

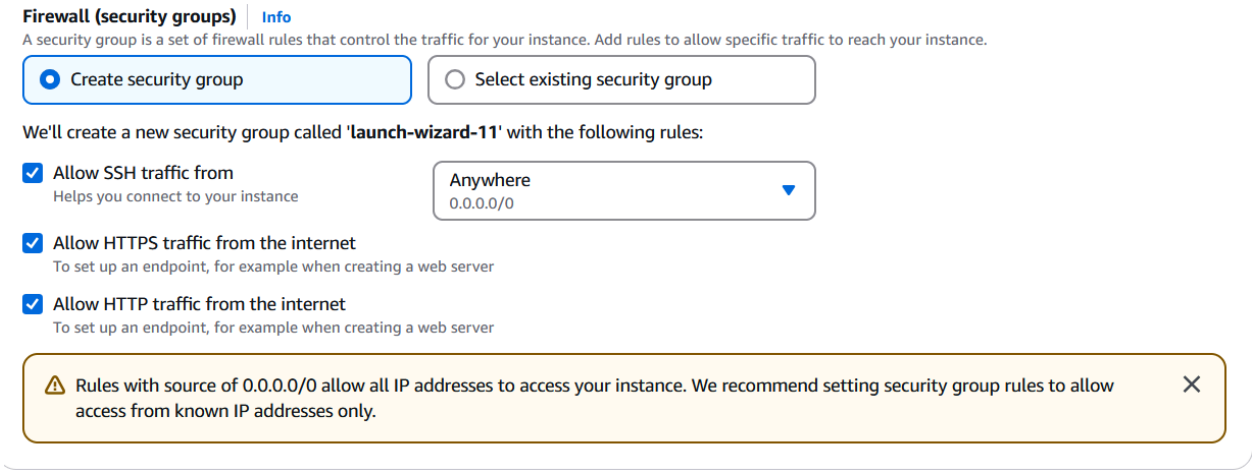
The DEV part of DevSecOps deals with ensuring the application runs locally.
Steps

Initial Set-up and Deployment: Create an EC2 Server

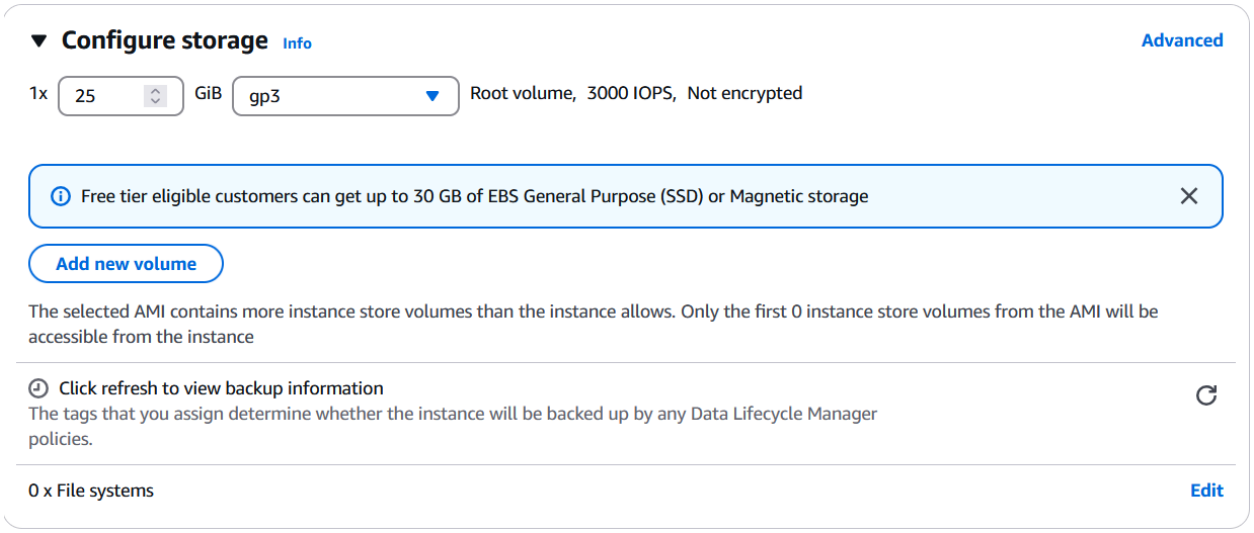
To deploy this application, I used a **t2.large** EC2 instance for Jenkins, SonarQube, and Trivy. This instance type is necessary because multiple plugins will be installed for Jenkins, and I need to ensure the server can handle the resource requirements.



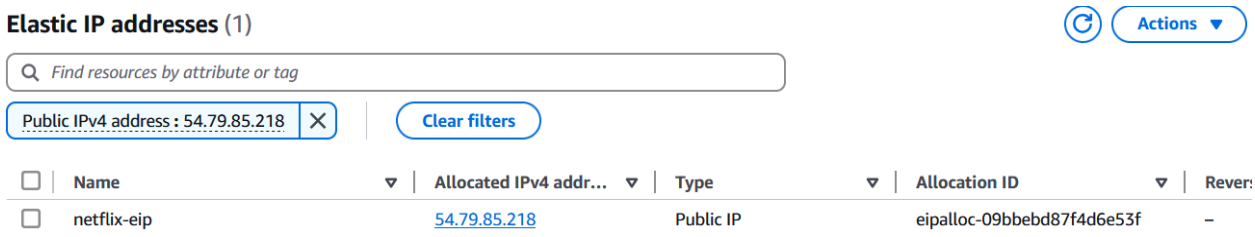
For Firewall settings, I enabled HTTP and HTTPS traffic options since Jenkins will be needing to connect to Dockerhub via HTTPS.



Storage was also increased to accommodate the anticipated requirements of the project.



After the instance has been successfully created, I attached an ElasticIP to the instance. This will help if the project will be done in more than one or two sessions, so the same IP instance will still be used even if the instance is stopped.



Connect to EC2 Instance

Next, I connected to my EC2 instance via EC2 Instance Connect, which supports Ubuntu (the AMI used to create the instance) and ran *sudo apt update -y* to update all relevant packages.

```
Get:28 http://ap-southeast-2.ec2.archive.ubuntu.com/ubuntu noble-backports/main amd64 c-n-f Metadata [112 B]
Get:29 http://ap-southeast-2.ec2.archive.ubuntu.com/ubuntu noble-backports/universe amd64 Packages [14.2 kB]
Get:30 http://ap-southeast-2.ec2.archive.ubuntu.com/ubuntu noble-backports/universe Translation-en [12.1 kB]
Get:31 http://ap-southeast-2.ec2.archive.ubuntu.com/ubuntu noble-backports/universe amd64 Components [19.8 kB]
Get:32 http://ap-southeast-2.ec2.archive.ubuntu.com/ubuntu noble-backports/universe amd64 c-n-f Metadata [1104 B]
Get:33 http://ap-southeast-2.ec2.archive.ubuntu.com/ubuntu noble-backports/restricted amd64 Components [216 B]
Get:34 http://ap-southeast-2.ec2.archive.ubuntu.com/ubuntu noble-backports/restricted amd64 c-n-f Metadata [116 B]
Get:35 http://ap-southeast-2.ec2.archive.ubuntu.com/ubuntu noble-backports/multiverse amd64 Components [212 B]
Get:36 http://ap-southeast-2.ec2.archive.ubuntu.com/ubuntu noble-backports/multiverse amd64 c-n-f Metadata [116 B]
Get:37 http://security.ubuntu.com/ubuntu noble-security/main amd64 Packages [634 kB]
Get:38 http://security.ubuntu.com/ubuntu noble-security/main Translation-en [121 kB]
Get:39 http://security.ubuntu.com/ubuntu noble-security/main amd64 Components [9000 B]
Get:40 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Packages [815 kB]
Get:41 http://security.ubuntu.com/ubuntu noble-security/universe Translation-en [174 kB]
Get:42 http://security.ubuntu.com/ubuntu noble-security/universe amd64 Components [51.9 kB]
Get:43 http://security.ubuntu.com/ubuntu noble-security/universe amd64 c-n-f Metadata [13.5 kB]
Get:44 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Packages [650 kB]
Get:45 http://security.ubuntu.com/ubuntu noble-security/restricted Translation-en [128 kB]
Get:46 http://security.ubuntu.com/ubuntu noble-security/restricted amd64 Components [212 B]
Get:47 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Packages [19.4 kB]
Get:48 http://security.ubuntu.com/ubuntu noble-security/multiverse Translation-en [4308 B]
Get:49 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 Components [212 B]
Get:50 http://security.ubuntu.com/ubuntu noble-security/multiverse amd64 c-n-f Metadata [356 B]
Fetched 32.4 MB in 18s (1761 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
120 packages can be upgraded. Run 'apt list --upgradable' to see them.
ubuntu@ip-172-31-1-131:~$
```

Cloning The Application

I cloned the Netflix application source code from a GitHub repo: https://github.com/N4si/DevSecOps-Project onto my EC2 instance by running the command *git clone <repo link>.* Running the *ls* command, we can see that the code has been successfully cloned into a *DevSecOps-Project* repository in the EC2 instance.

```
ubuntu@ip-172-31-1-131:~$ ls
DevSecOps-Project
```

```
ubuntu@ip-172-31-1-131:~/DevSecOps-Project$ ls
Dockerfile  Kubernetes  README.md  index.html  package.json  pipeline.txt  public  src  tsconfig.json  tsconfig.node.json  vercel.json  vite.config.ts  yarn.lock
```

Then I installed Docker, which is needed to create the Docker image.

```
sudo apt-get update
sudo apt-get install docker.io -y
sudo usermod -aG docker $USER  # Rep
newgrp docker
sudo chmod 777 /var/run/docker.sock
```

```
ubuntu@ip-172-31-1-131:~/DevSecOps-Project$ docker version
Client:
 Version:           26.1.3
 API version:       1.45
 Go version:        go1.22.2
 Git commit:        26.1.3-0ubuntu1~24.04.1
 Built:             Mon Oct 14 14:29:26 2024
 OS/Arch:           linux/amd64
 Context:           default

Server:
 Engine:
  Version:          26.1.3
  API version:      1.45 (minimum version 1.24)
  Go version:       go1.22.2
  Git commit:       26.1.3-0ubuntu1~24.04.1
  Built:            Mon Oct 14 14:29:26 2024
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.7.24
  GitCommit:
 runc:
  Version:          1.1.12-0ubuntu3.1
  GitCommit:
 docker-init:
```

Once docker has successfully been installed, the next step is to build and run the application using docker containers. Docker build was initiated using the command *docker build -t netflix .*

```
Successfully built 061aa0ad9b76
Successfully tagged netflix:latest
ubuntu@ip-172-31-1-131:~/DevSecOps-Project$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED         SIZE
netflix      latest    061aa0ad9b76   2 minutes ago   56.5MB
```

Next, I ran *docker run -d -p 8081:80 <image ID>* to start the Docker container from the Netflix image.

-d (detached mode)

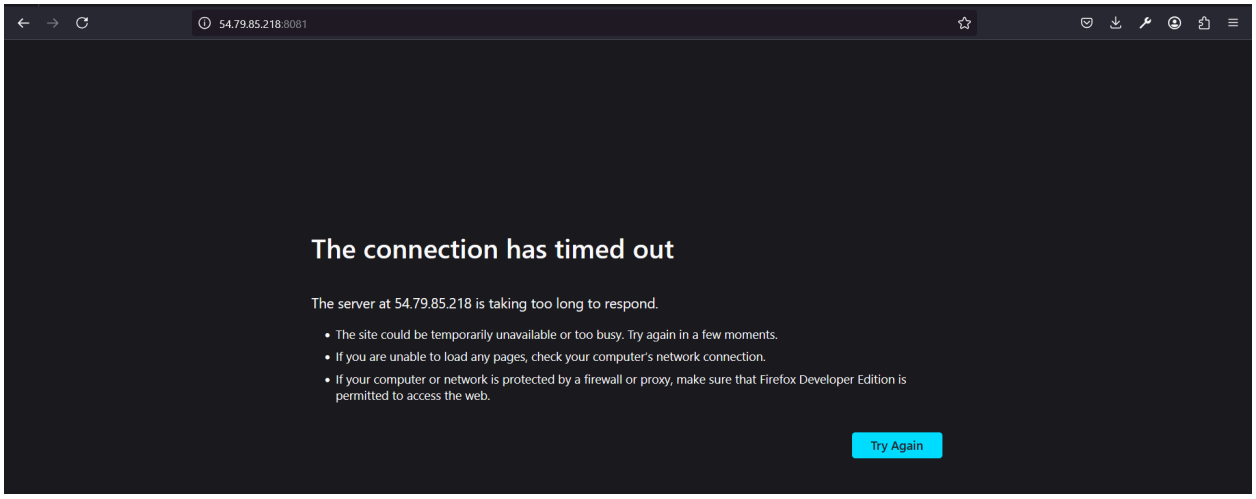
- Runs the container in the background (detached mode).
- Without -d, the container runs in the foreground, with the logs visible in the terminal.

-p 8081:80 (port mapping)

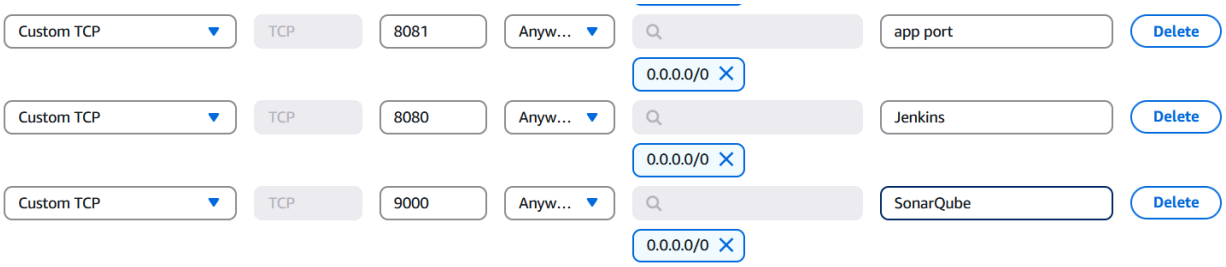
- Maps **port 8081** on the local machine (host) to **port 80** inside the container.

This will start the *netflix* image inside a container. The *netflix* server listens on port 80 inside the container. It can be accessed through http://localhost:8081

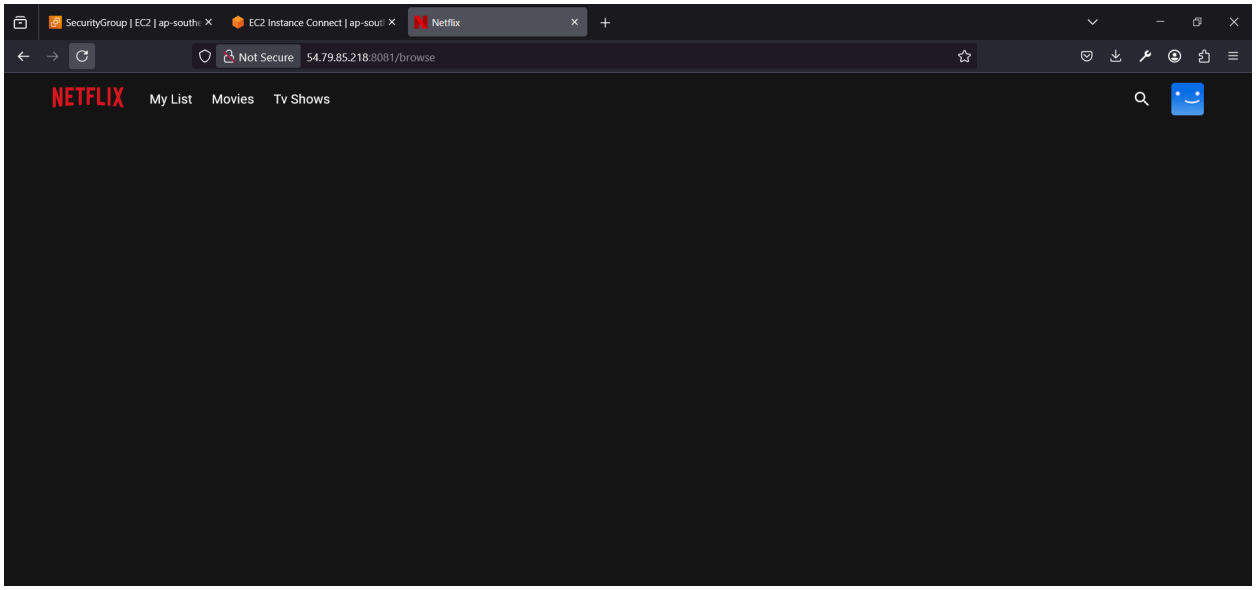
Accessing the localhost of the instance with port 8081:



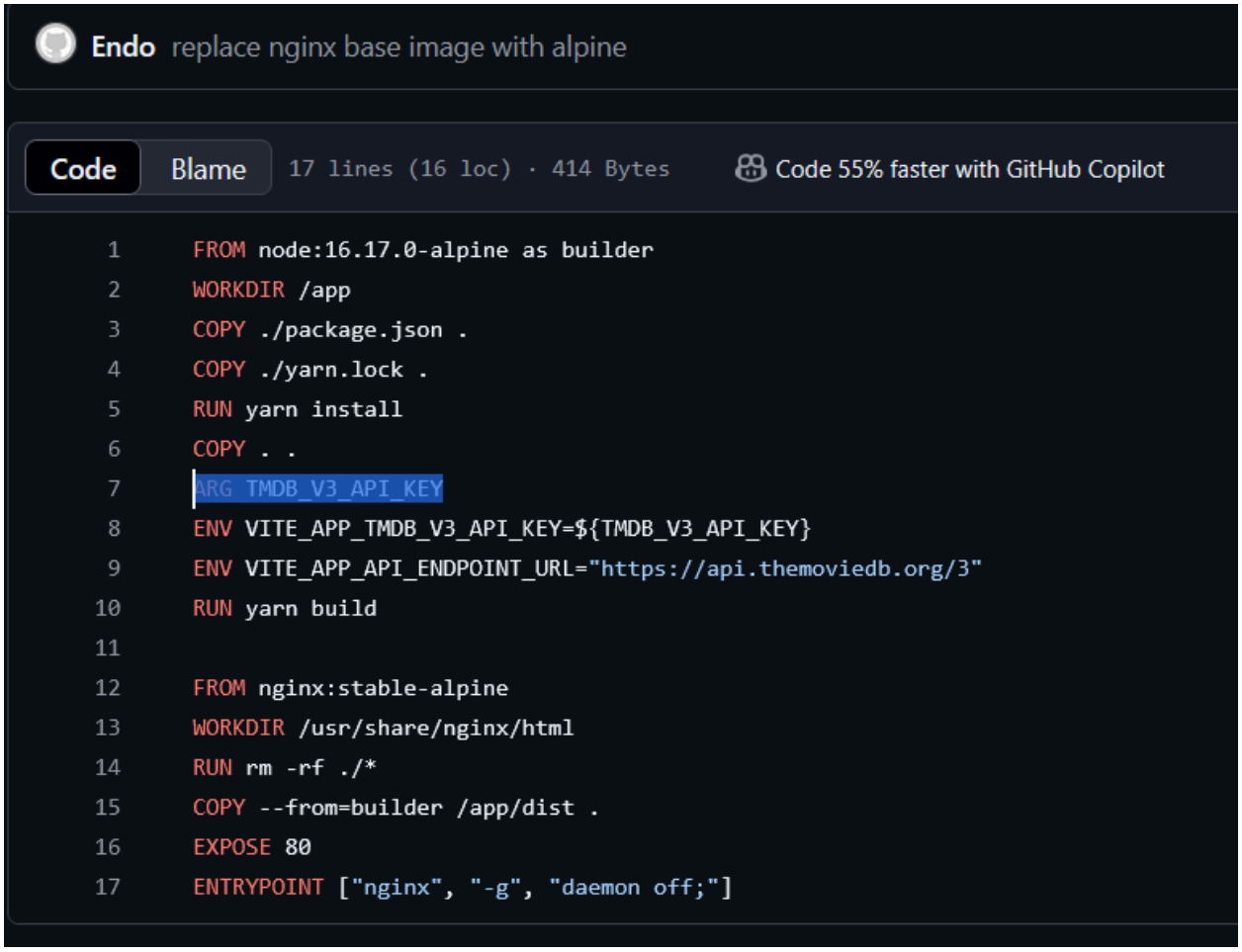
This happens because port 8081 is not open in the instance Security Group. Ports 8081 for the application, 8080 for Jenkins, and 9000 for SonarQube were added to the inbound rules for the Security Group.



Now that we've allowed the ports on the Security Group, we can now access the Application on port 8081.



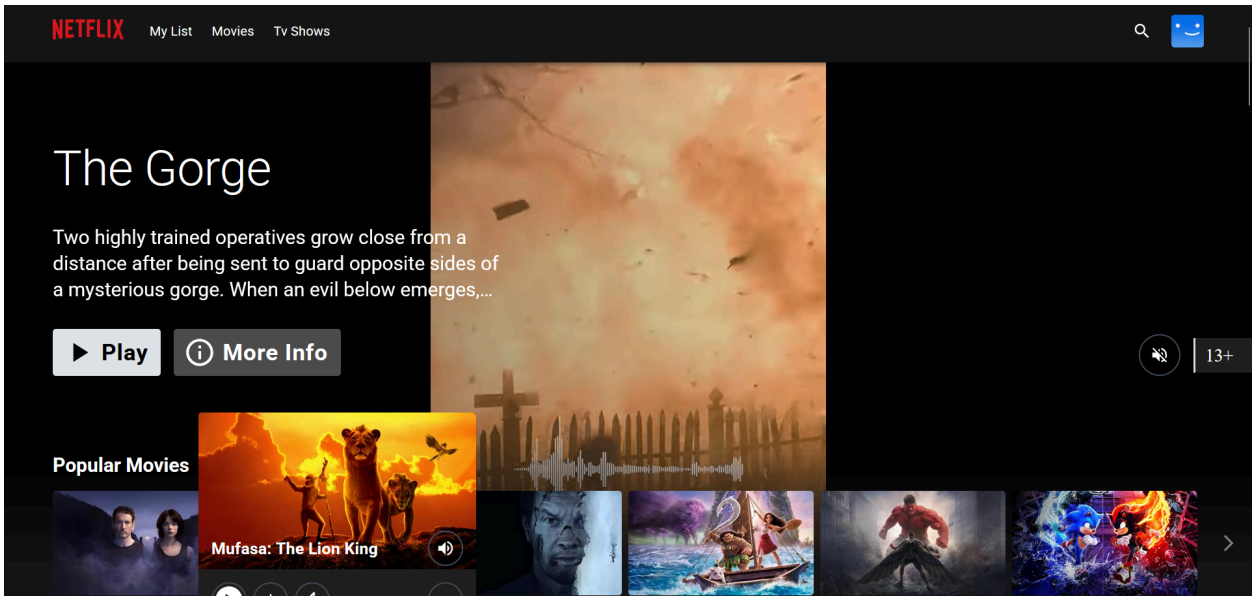
The application works; however, there is no movie selection or catalog of shows to choose from, which are expected features of the Netflix website. This issue is caused by the ARG statement in the Dockerfile as seen on the source code.



The dockerfile expects an argument, and I haven't provided its requirements yet, an API key from an application called TMDB for the movie database. The API key is necessary so that the netflix application can fetch the data into the application.

I stopped and removed the container before starting it again with the needed API key with this command:

docker build --build-arg TMDB_V3_API_KEY=<api-key> -t netflix .



The site is now accessible from the local machine on port 8081 with the catalogue of films.

DevSecOps Phase 2: Security

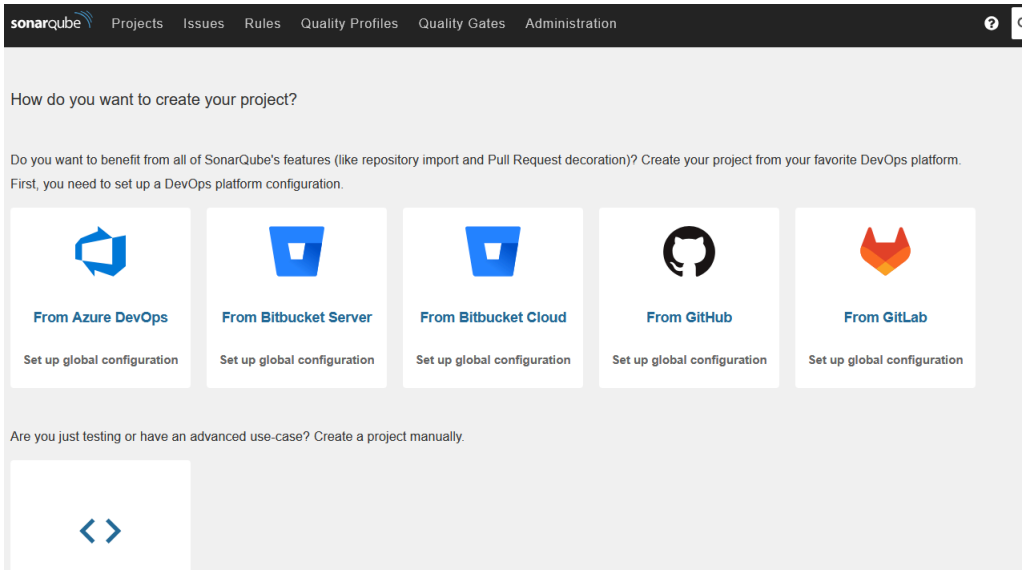
The Security Phase deals with ensuring that the code, and all its dependencies and images are correct, and properly set-up. For security in this project, two tools will be utilized: SonarQube (analyzes code and gives reports on potential bugs and issues with code) and Trivy (scans file systems and gives reports on file system issues).

Install SonarQube and Trivy

To install SonarQube I ran *docker run -d --name sonar -p 9000:9000 sonarqube:lts-community*

```
CONTAINER ID   IMAGE                     COMMAND                  CREATED              STATUS              PORTS                                       NAMES
5855a1aa9fe0   sonarqube:lts-community   "/opt/sonarqube/dock…"   About a minute ago   Up About a minute   0.0.0.0:9000->9000/tcp, :::9000->9000/tcp   sonar
424ba7b155e6   netflix                   "nginx -g 'daemon of…"   12 minutes ago       Up 12 minutes       0.0.0.0:8081->80/tcp, :::8081->80/tcp       tender_faraday
```

Accessing SonarQube at port 9000



Next I installed Trivy by Running the command:

```
sudo apt-get install wget apt-transport-https gnupg lsb-release
wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | sudo apt-key add -
echo deb https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc) main | sudo tee -a /
sudo apt-get update
sudo apt-get install trivy
```

And scanned the netflix image

```
ubuntu@ip-172-31-1-131:~/DevSecOps-Project$ trivy image f6ca600c8b76
2025-02-27T08:43:22Z    INFO    [vuln] Vulnerability scanning is enabled
2025-02-27T08:43:22Z    INFO    [secret] Secret scanning is enabled
2025-02-27T08:43:22Z    INFO    [secret] If your scanning is slow, please try '--scanners vuln' to disable secret scanning
2025-02-27T08:43:22Z    INFO    [secret] Please see also https://aquasecurity.github.io/trivy/v0.59/docs/scanner/secret#recommendation for faster secret detection
2025-02-27T08:43:22Z    INFO    Detected OS     family="alpine" version="3.20.6"
2025-02-27T08:43:22Z    INFO    [alpine] Detecting vulnerabilities...   os_version="3.20" repository="3.20" pkg_num=66
2025-02-27T08:43:22Z    INFO    Number of language-specific files       num=0
```

DevSecOps Phase 3: Operations

The operations segment includes Integrating a CI/CD Pipeline into the application using Jenkins.

Install Jenkins for Automation

Java installation is required for Jenkins to work. Using the following commands, Java and Jenkins were installed.

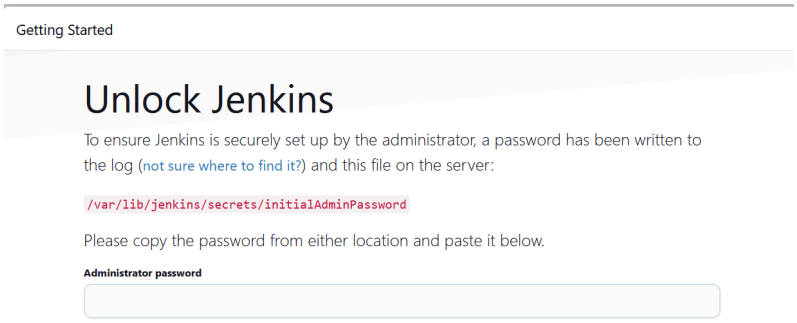
```
sudo apt update
sudo apt install fontconfig openjdk-17-jre
java -version
openjdk version "17.0.8" 2023-07-18
OpenJDK Runtime Environment (build 17.0.8+7-Debian-1deb12u1)
OpenJDK 64-Bit Server VM (build 17.0.8+7-Debian-1deb12u1, mixed mode, sharing)

#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
sudo apt-get update
sudo apt-get install jenkins
sudo systemctl start jenkins
sudo systemctl enable jenkins
```

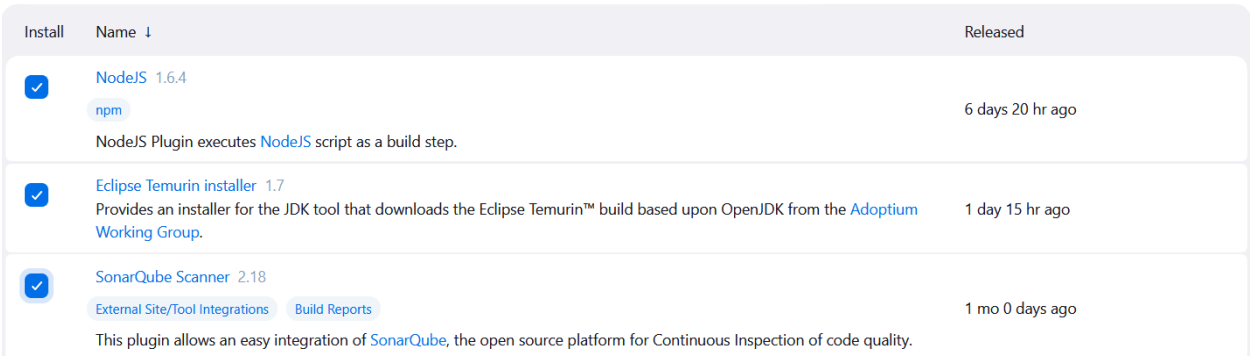
```
No VM guests are running outdated hypervisor (qemu) binaries on this host.
Synchronizing state of jenkins.service with SysV service script with /usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable jenkins
ubuntu@ip-172-31-1-131:~/DevSecOps-Project$ sudo service jenkins status
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/usr/lib/systemd/system/jenkins.service; enabled; preset: enabled)
     Active: active (running) since Thu 2025-02-27 08:53:49 UTC; 45s ago
   Main PID: 15154 (java)
      Tasks: 50 (limit: 9507)
     Memory: 815.3M (peak: 833.0M)
        CPU: 15.598s
     CGroup: /system.slice/jenkins.service
             └─15154 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Feb 27 08:53:44 ip-172-31-1-131 jenkins[15154]: 98dace7ceaf148ecbc06b9bcf7793dec
Feb 27 08:53:44 ip-172-31-1-131 jenkins[15154]: This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
Feb 27 08:53:44 ip-172-31-1-131 jenkins[15154]: ************************************************************
Feb 27 08:53:44 ip-172-31-1-131 jenkins[15154]: ************************************************************
Feb 27 08:53:44 ip-172-31-1-131 jenkins[15154]: ************************************************************
Feb 27 08:53:49 ip-172-31-1-131 jenkins[15154]: 2025-02-27 08:53:49.410+0000 [id=30]        INFO        jenkins.InitReactorRunner$1#onAttained: Completed initialization
Feb 27 08:53:49 ip-172-31-1-131 jenkins[15154]: 2025-02-27 08:53:49.426+0000 [id=23]        INFO        hudson.lifecycle.Lifecycle#onReady: Jenkins is fully up and runn
Feb 27 08:53:49 ip-172-31-1-131 systemd[1]: Started jenkins.service - Jenkins Continuous Integration Server.
Feb 27 08:53:50 ip-172-31-1-131 jenkins[15154]: 2025-02-27 08:53:50.676+0000 [id=48]        INFO        h.m.DownloadService$Downloadable#load: Obtained the updated data
Feb 27 08:53:50 ip-172-31-1-131 jenkins[15154]: 2025-02-27 08:53:50.677+0000 [id=48]        INFO        hudson.util.Retrier#start: Performed the action check updates se
```

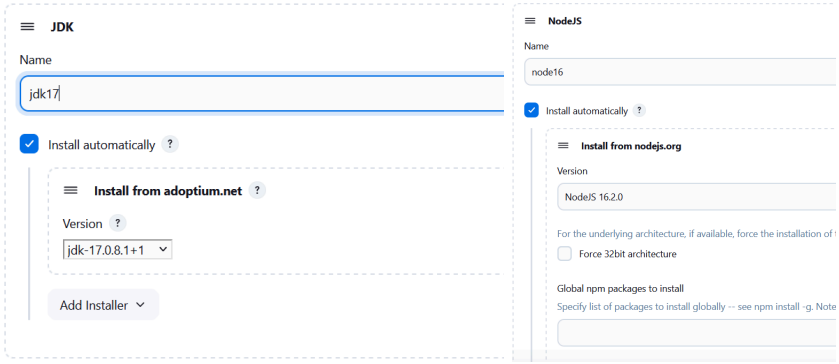
Accessing Jenkins on port 8080 of the EC2 Instance, I landed on a sign-in page.



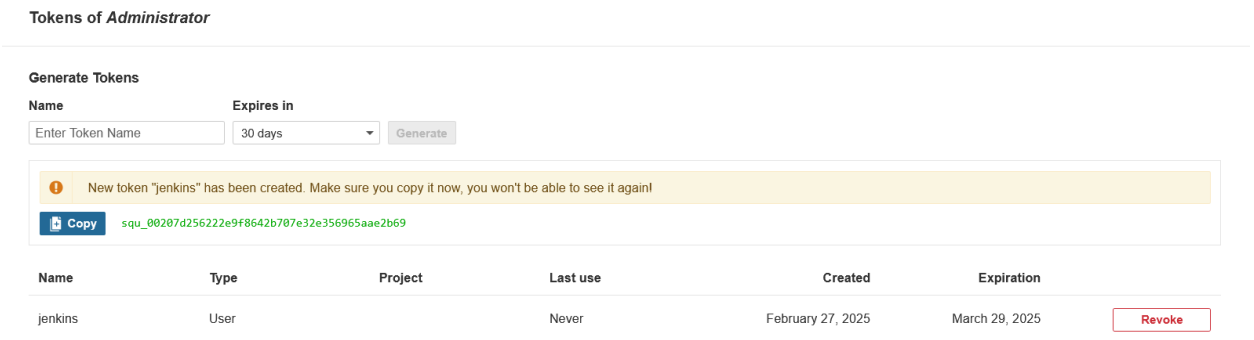
I ran the command *sudo cat /var/lib/jenkins/secrets/initialAdminPassword* which outputs the administrator password, and allowed me to log in and start suggested downloads. After Jenkins has successfully completed its download into the machine, I downloaded some of the plugins needed such as NodeJS, Eclipse Temurin Installer, and SonarQube.



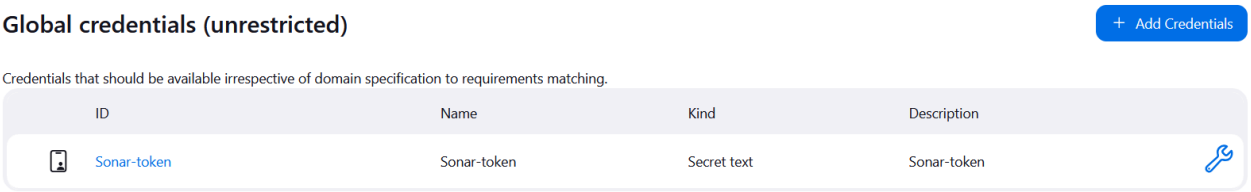
Once these were downloaded, these plugins were set in the Tools section for NodeJS and JDK.

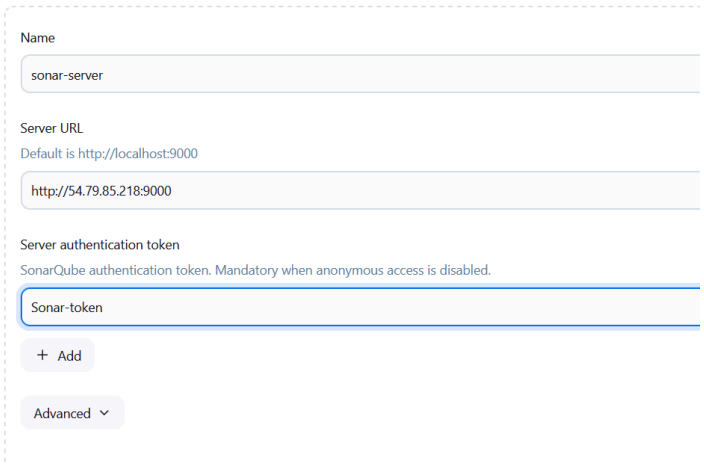


For SonarQube, it has to be first connected to Jenkins, and then added to the system, and lastly to the tools like the other plugins. On SonarQube (Port 9000) I generated a token

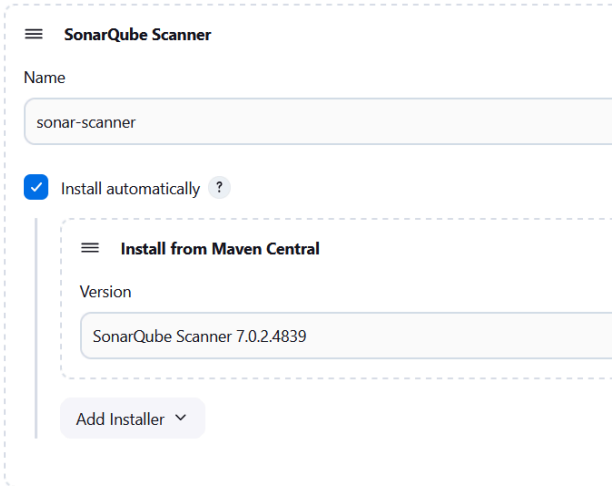


Back in Jenkins under ‘credentials’ I added credentials for the SonarQube token I just generated. These newly created credentials are important because it will be used for the SonarQube Tool that I need to create.





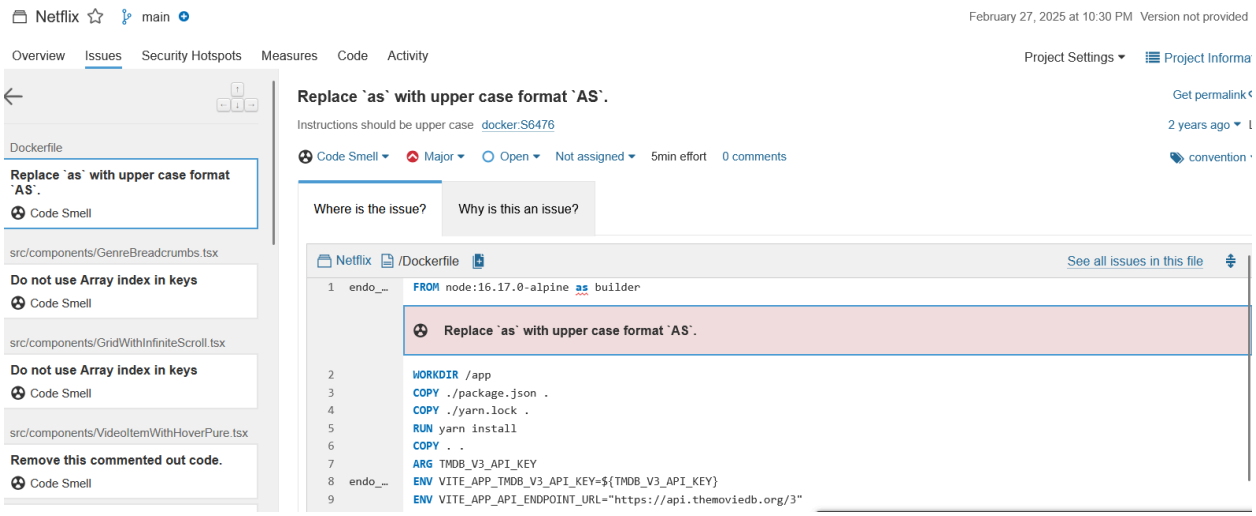
Next, I created a ‘sonar-scanner’ tool since it will be used on the pipeline to be made.



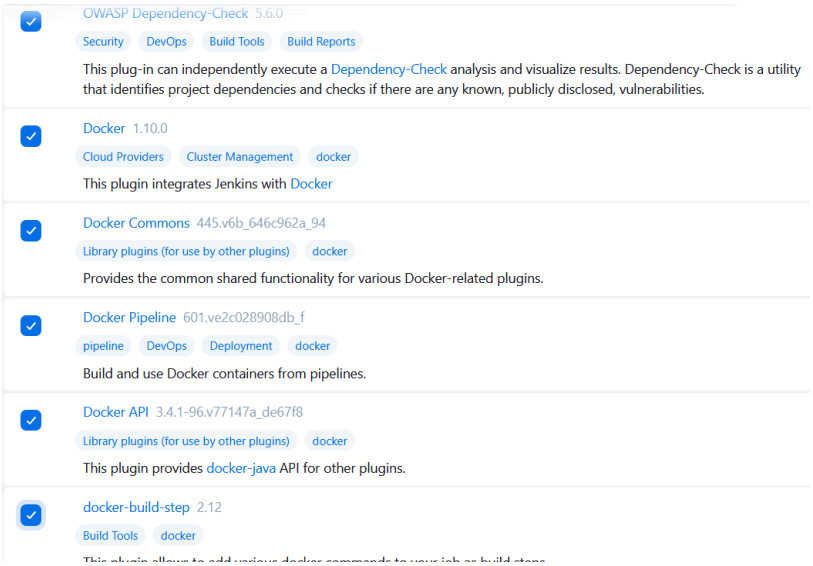
Now I was able to deploy and build an application on the pipeline using this script:

```
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: 'main', url: 'https://github.com/N4si/DevSecOps-Project.git'
            }
        }
        stage("Sonarqube Analysis") {
            steps {
                withSonarQubeEnv('sonar-server') {
                    sh '''$SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=Netflix \
                    -Dsonar.projectKey=Netflix'''
                }
            }
        }
        stage("quality gate") {
            steps {
                script {
                    waitForQualityGate abortPipeline: false, credentialsId: 'Sonar-token'
                }
```

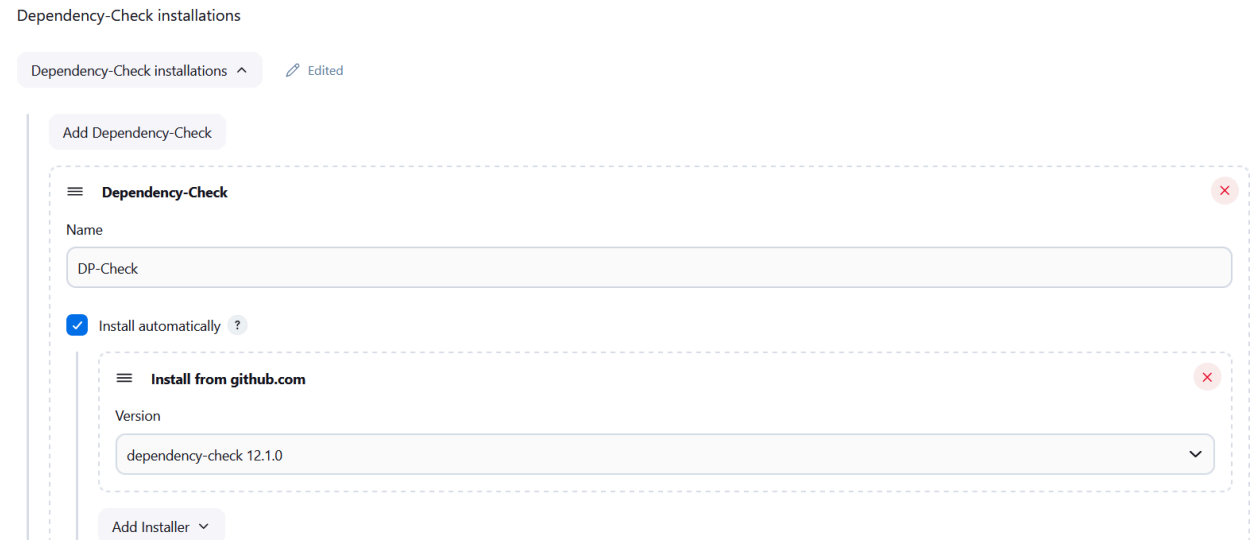
Back at the SonarQube, it started analyzing the code for any issues, vulnerabilities and security recommendations

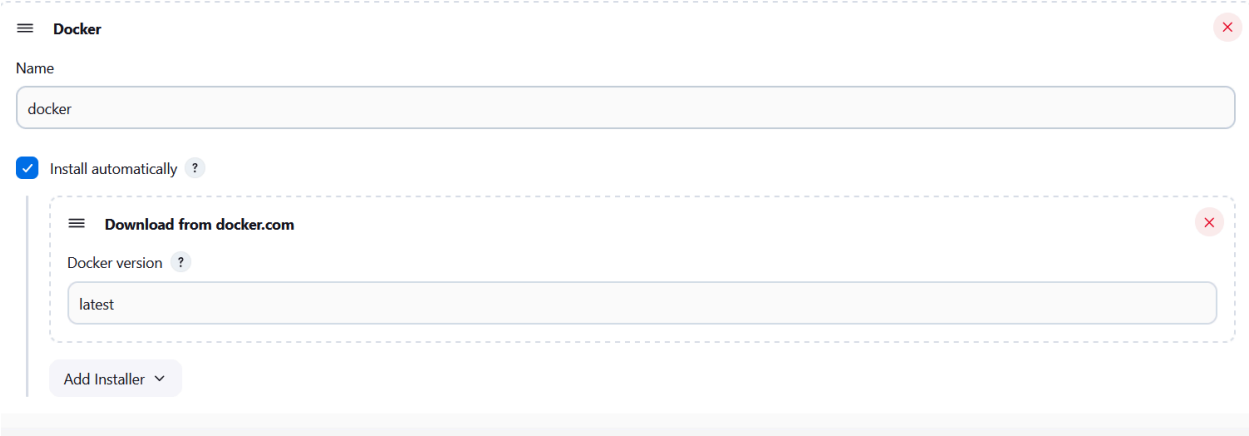


While it was building, I downloaded other plugins required: OWASP Dependency Check and Docker, and added credentials for Docker in Jenkins so that I can login using the provided DockerHub username and password.



Then, I added OWASP and Docker (recently added plugins) into the Tools section





Then I created a pipeline that will be scanning images through Trivy, check dependencies through OWASP, and create the image using commands, integrating all the new features downloaded, and the needed API key.

```
pipeline{
    agent any
    tools{
        jdk 'jdk17'
        nodejs 'node16'
    }
    environment {
        SCANNER_HOME=tool 'sonar-scanner'
    }
    stages {
        stage('clean workspace'){
            steps{
                cleanWs()
            }
        }
        stage('Checkout from Git'){
            steps{
                git branch: 'main', url: 'https://github.com/N4si/DevSecOps-Project.git'
            }
        }
        stage("Sonarqube Analysis "){
            steps{
                withSonarQubeEnv('sonar-server') {
                    sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=Netflix \
                    -Dsonar.projectKey=Netflix '''
                }
            }
        }
        stage("quality gate"){
           steps {
                script {
                    waitForQualityGate abortPipeline: false, credentialsId: 'Sonar-token'
```

The pipeline:

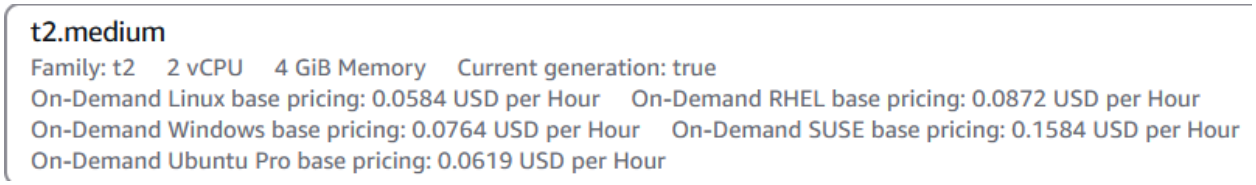
Cleanup workspace, get the code, sonarqube, quality gate check, install dependencies, run dependency scanner, scan file system, run command to build tag image and push image, once image created, scan thru trivy, and then deploy image as container on server.

Phase 4: Monitoring

Install Prometheus and Grafana:

Set up Prometheus and Grafana to monitor the application. Prometheus is used for monitoring alerting about the system's health and performance, optimizing troubleshooting issues. Grafana is then used to visualize the monitoring data that comes in.

To have monitoring enabled, I need to create a new t.2.medium instance due to Prometheus' system requirements (2 CPU Cores, 4GB RAM).



Once it was created, I attached an elasticIP to it, and then connected to the instance to install Grafana, Prometheus, Node Exporter, and everything else necessary for monitoring.

Installing Prometheus

```
sudo useradd --system --no-create-home --shell /bin/false prometheus
wget https://github.com/prometheus/prometheus/releases/download/v2.47.1/prometheus-2.47.1.linux-amc
```

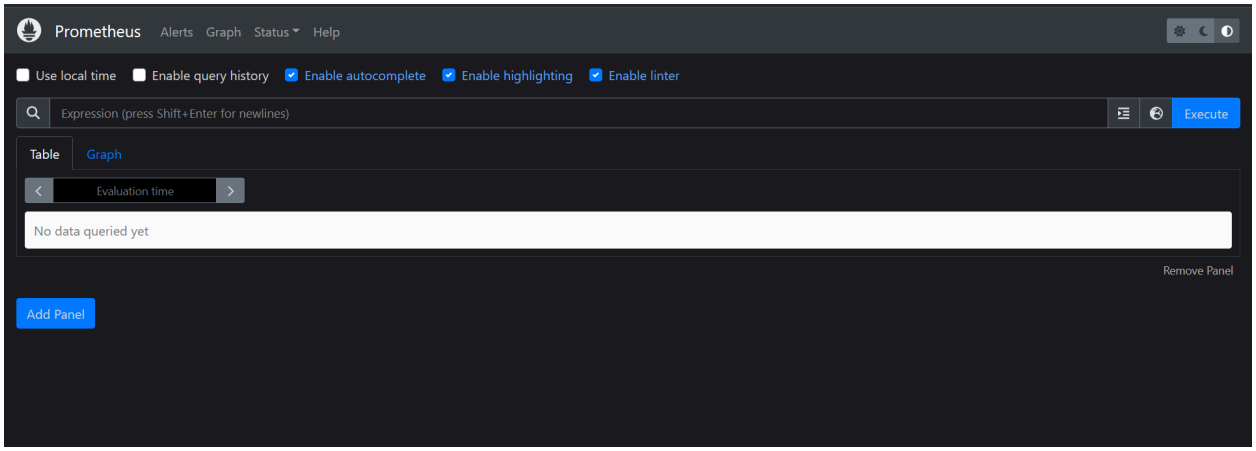
```
ubuntu@ip-172-31-12-9:~$ ls
prometheus-2.47.1.linux-amd64.tar.gz
```

Extract Prometheus files, move them, and create directories:

```
tar -xvf prometheus-2.47.1.linux-amd64.tar.gz
cd prometheus-2.47.1.linux-amd64/
sudo mkdir -p /data /etc/prometheus
sudo mv prometheus promtool /usr/local/bin/
sudo mv consoles/ console_libraries/ /etc/prometheus/
sudo mv prometheus.yml /etc/prometheus/prometheus.yml
```

Promtool is a query tool that uses queries to get data about monitoring

Prometheus.yml is where I can set the servers I want to monitor. To start monitoring I need a Node Exporter. Node Exporter is a lightweight agent developed by the Prometheus project that gathers and exposes system-level metrics from Linux systems, enabling Prometheus to monitor them. Node Exporter will extract the metric, and these metrics will be used by Prometheus to start monitoring.



Installing 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
```

Back at the Pipeline build: Build Failed, Docker Failed to log in

```
[Pipeline] End of Pipeline
ERROR: docker login failed
Finished: FAILURE
```

I ran commands:

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

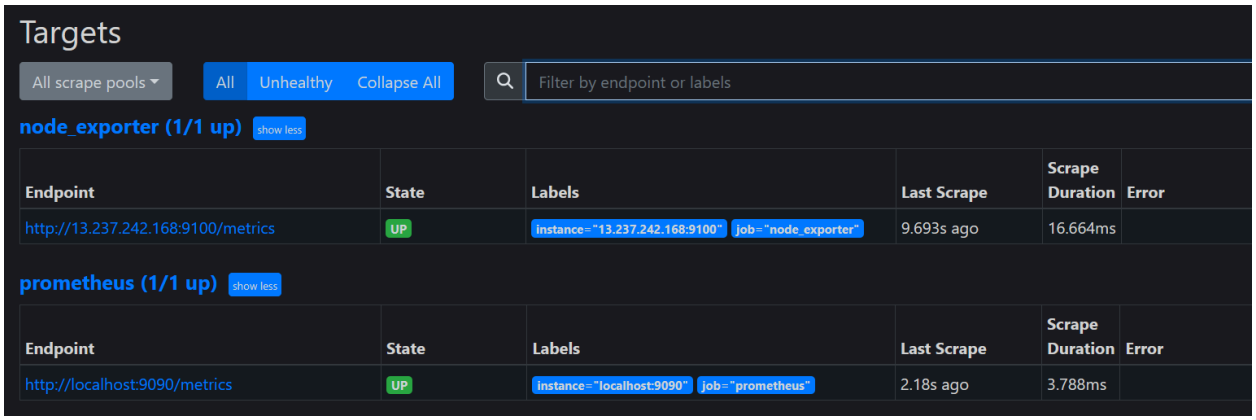
To add to the Docker Group

Installing Node Exporter:

Create a system user for Node Exporter and download Node Exporter:

```
sudo useradd --system --no-create-home --shell /bin/false node_exporter
wget https://github.com/prometheus/node_exporter/releases/download/v1.6.1/node_exporter-1.6.1.linux
```

```
ubuntu@ip-172-31-12-9:~$ ls
node_exporter-1.6.1.linux-amd64.tar.gz  prometheus-2.47.1.linux-amd64  prometheus-2.47.1.linux-amd64.tar.gz
```

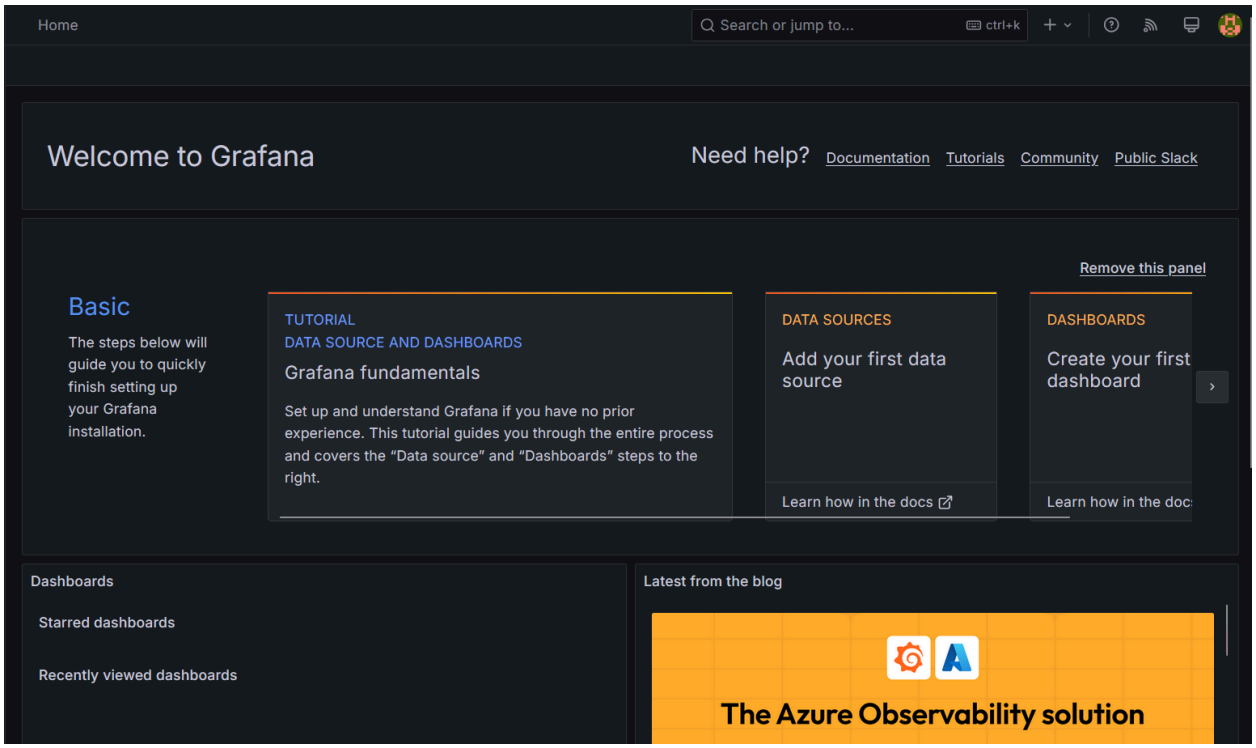


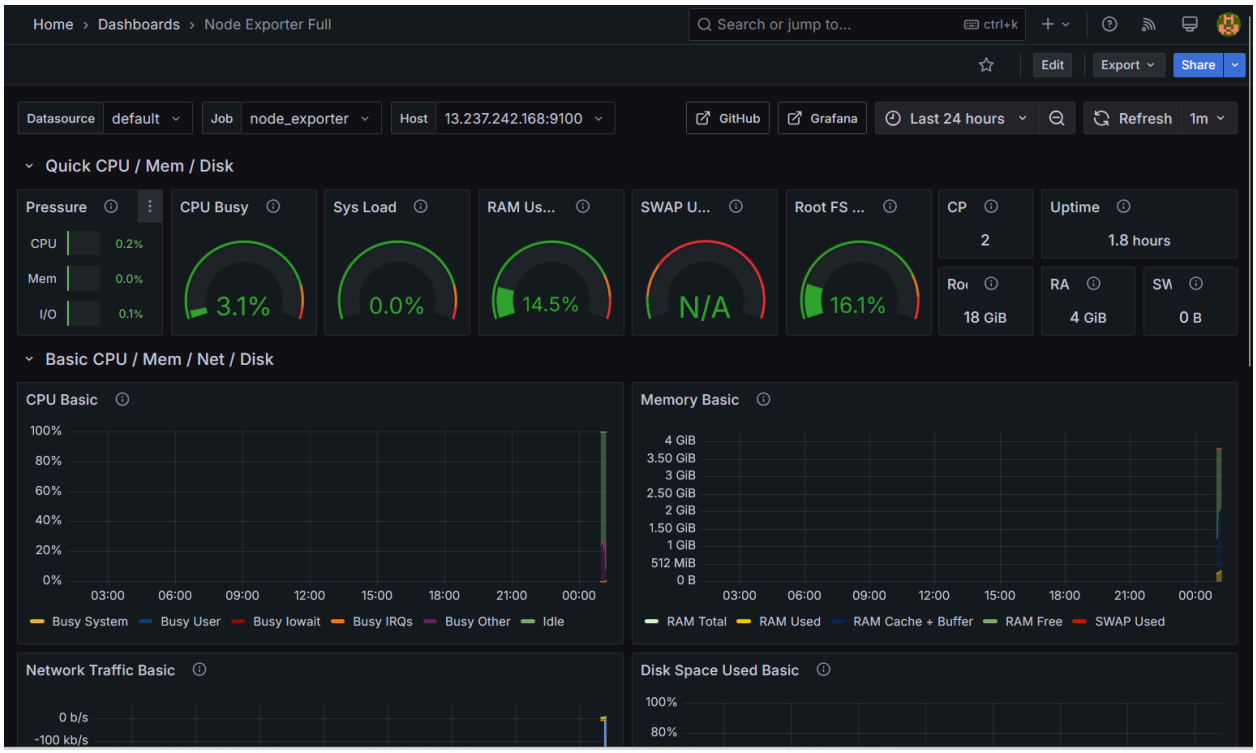
Install Grafana on Ubuntu 22.04 and Set it up to Work with Prometheus

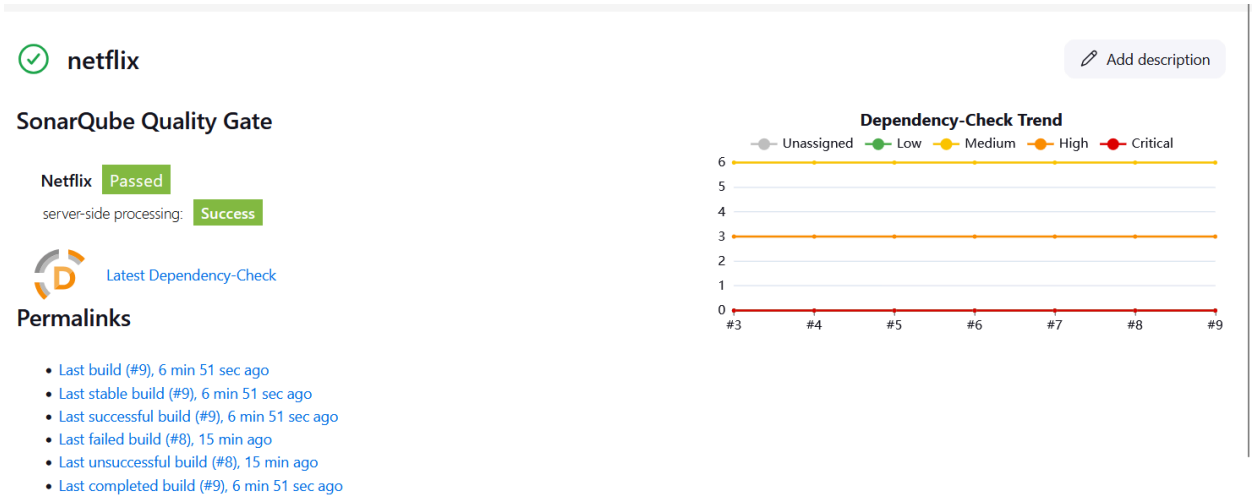
Grafana is another tool that works with Prometheus that creates graphs out of provided metrics. I installed Grafana using the following commands.

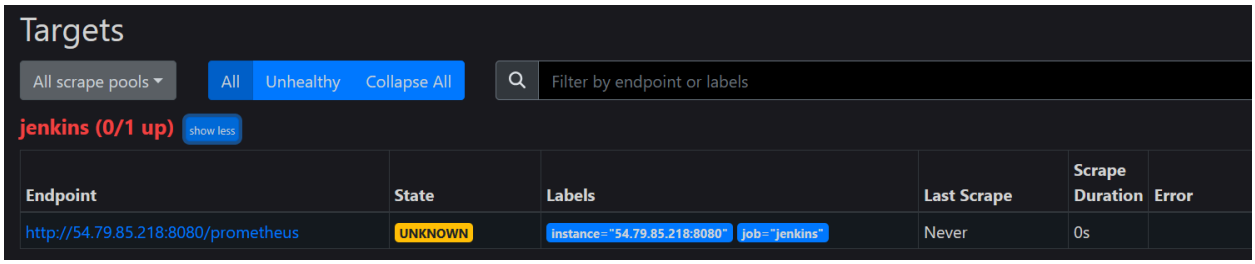
```
sudo apt-get update
sudo apt-get install -y apt-transport-https software-properties-common
```

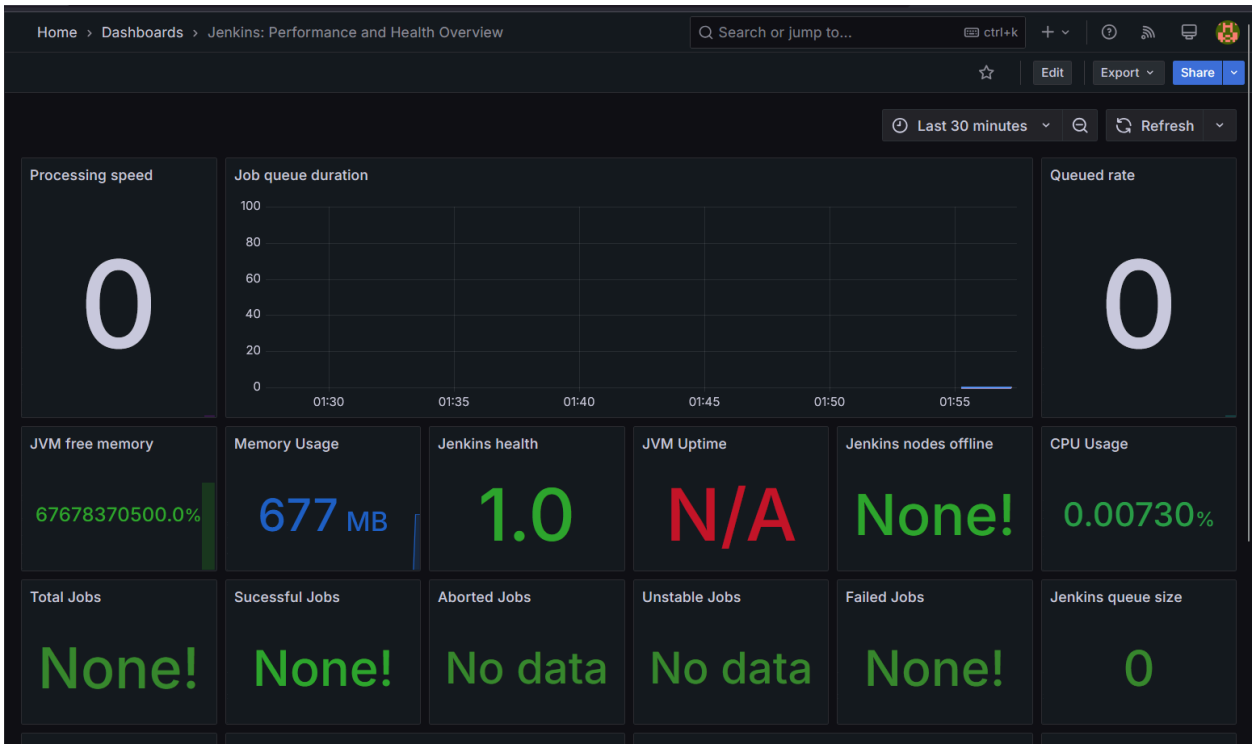
☐	–	sgr-01e8c09af5dcaa2cd	IPv4	Custom TCP	TCP	3000





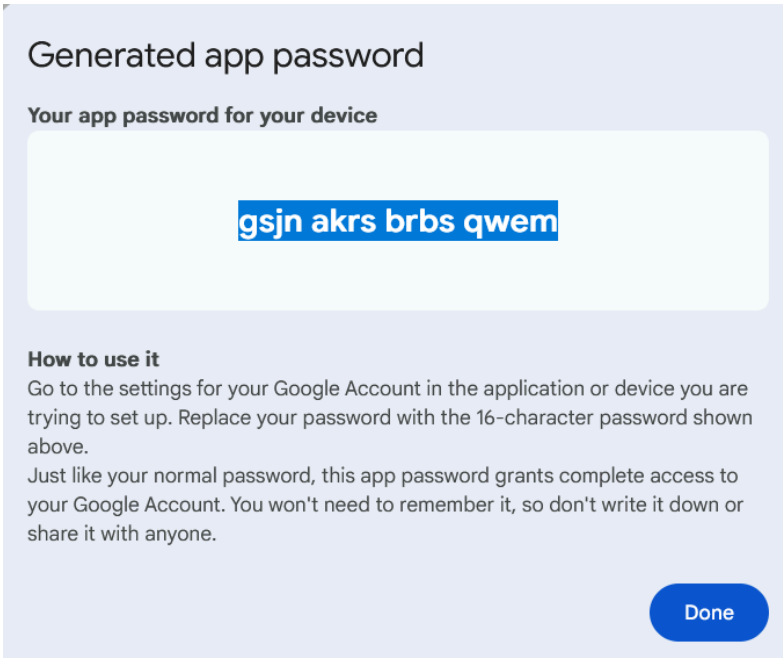

netflix
Add description
SonarQube Quality Gate
Netflix Passed
server-side processing: Success
Latest Dependency-Check
Permalinks
Last build (#9), 6 min 51 sec ago
Last stable build (#9), 6 min 51 sec ago
Last successful build (#9), 6 min 51 sec ago
Last failed build (#8), 15 min ago
Last unsuccessful build (#8), 15 min ago
Last completed build (#9), 6 min 51 sec ago
Dependency-Check Trend
Unassigned
Low
Medium
High
Critical
6
5
4
3
2
1
0
#3
#4
#5
#6
#7
#8
#9

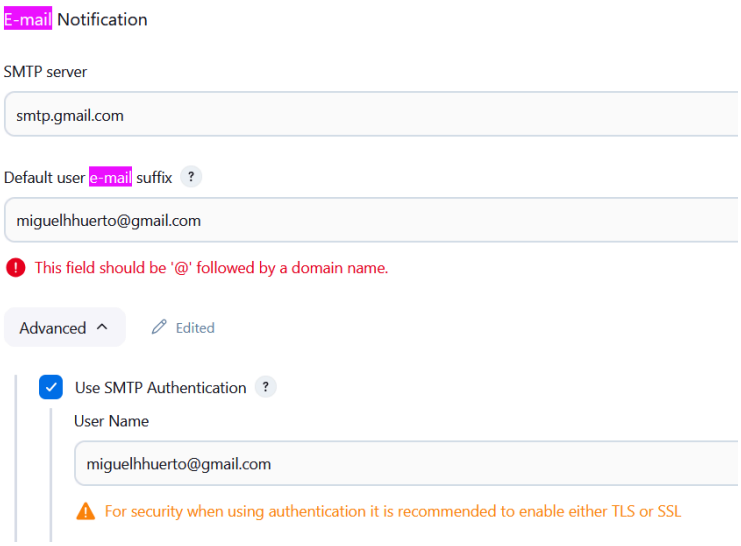

Targets
All scrape pools
All
Unhealthy
Collapse All
Filter by endpoint or labels
jenkins (0/1 up)
show less
Endpoint
State
Labels
Last Scrape
Scrape Duration
Error
http://54.79.85.218:8080/prometheus
UNKNOWN
instance="54.79.85.218:8080"
job="jenkins"
Never
0s


Home
Dashboards
Jenkins: Performance and Health Overview
Search or jump to...
ctrl+k
Edit
Export
Share
Last 30 minutes
Refresh
Processing speed
0
Job queue duration
100
80
60
40
20
0
01:30
01:35
01:40
01:45
01:50
01:55
Queued rate
0
JVM free memory
67678370500.0%
Memory Usage
677 MB
Jenkins health
1.0
JVM Uptime
N/A
Jenkins nodes offline
None!
CPU Usage
0.00730%
Total Jobs
None!
Sucessful Jobs
None!
Aborted Jobs
No data
Unstable Jobs
No data
Failed Jobs
None!
Jenkins queue size
0

Implement Notification Services:

Email notifications were setup for Jenkins

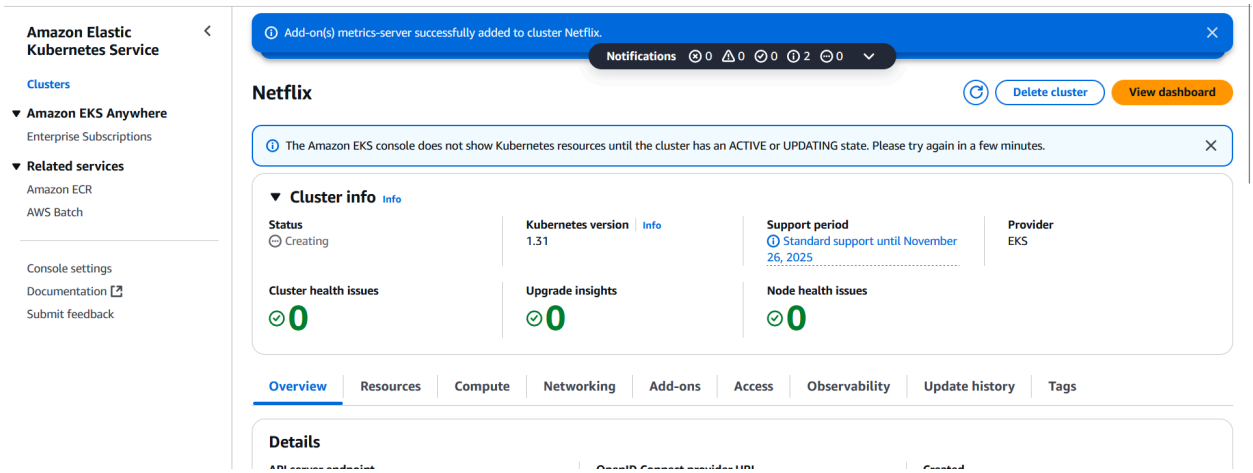




Script ?

```
        }
    }
}
post {
success {
    emailext to: "miguelhhuerto@gmail.com",
             subject: "Build Successful: ${env.JOB_NAME} #${env.BUILD_NUMBER}",
             body: "Good news! Your build for ${env.JOB_NAME} was successful.\nCheck it here: ${env.BUILD_URL}"
}
failure {
    emailext to: "miguelhhuerto@gmail.com",
             subject: "Build Failed: ${env.JOB_NAME} #${env.BUILD_NUMBER}",
             body: "Oh no! Your build for ${env.JOB_NAME} failed.\nCheck the logs: ${env.BUILD_URL}"
}
}
}
```

Create Kubernetes Cluster with Nodegroups



Monitor Kubernetes with Prometheus

Prometheus is then used to monitor the Kubernetes cluster. Additionally, node exporter was installed using Helm to collect metrics from the cluster nodes. Helm simplifies Kubernetes application deployment and management by acting as a package manager.

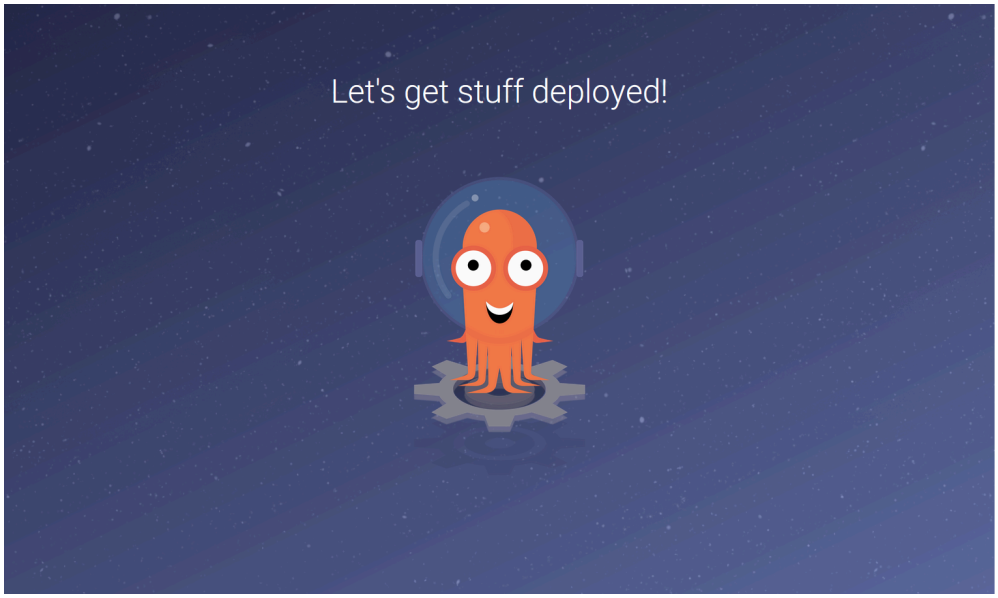
A Job was then added to prometheus.yml to Scrape Metrics on nodeip:9001/metrics i

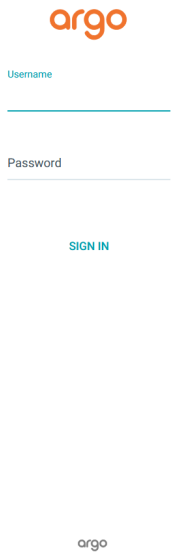
```
- job_name: 'Netflix'
  metrics_path: '/metrics'
  static_configs:
    - targets: ['node1Ip:9100']
```

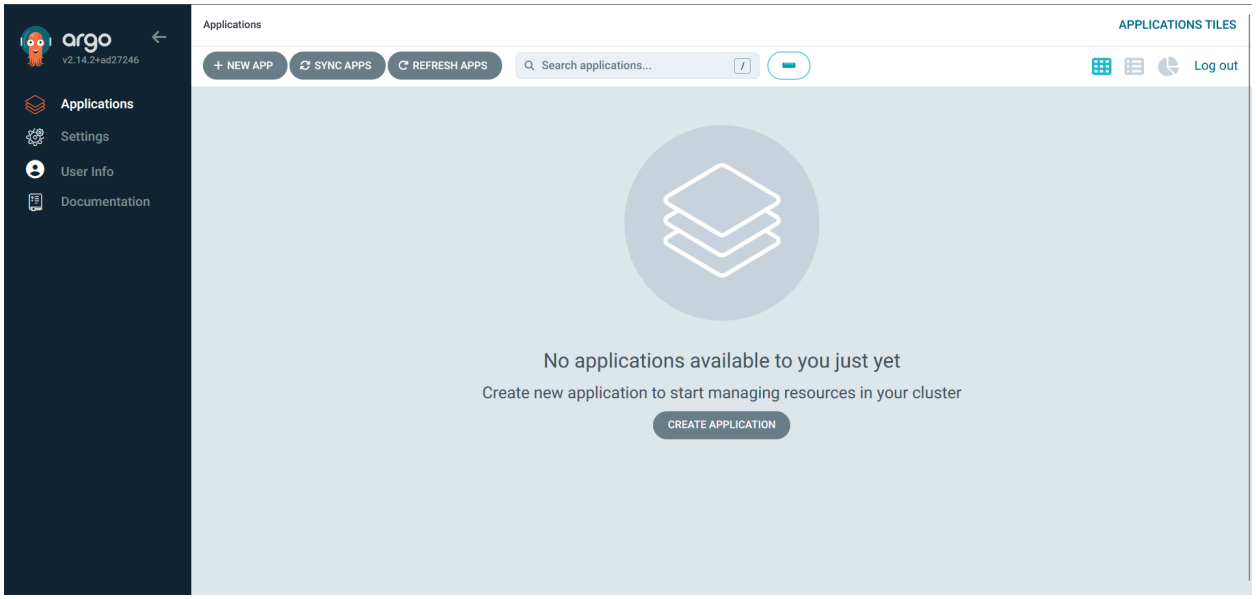
Deploy Application with ArgoCD

Argo CD is a declarative continuous delivery (CD) tool for Kubernetes that automates the deployment of applications by comparing the desired state defined in a Git repository with the current state of the cluster. Installing ArgoCD on the Kubernetes cluster:

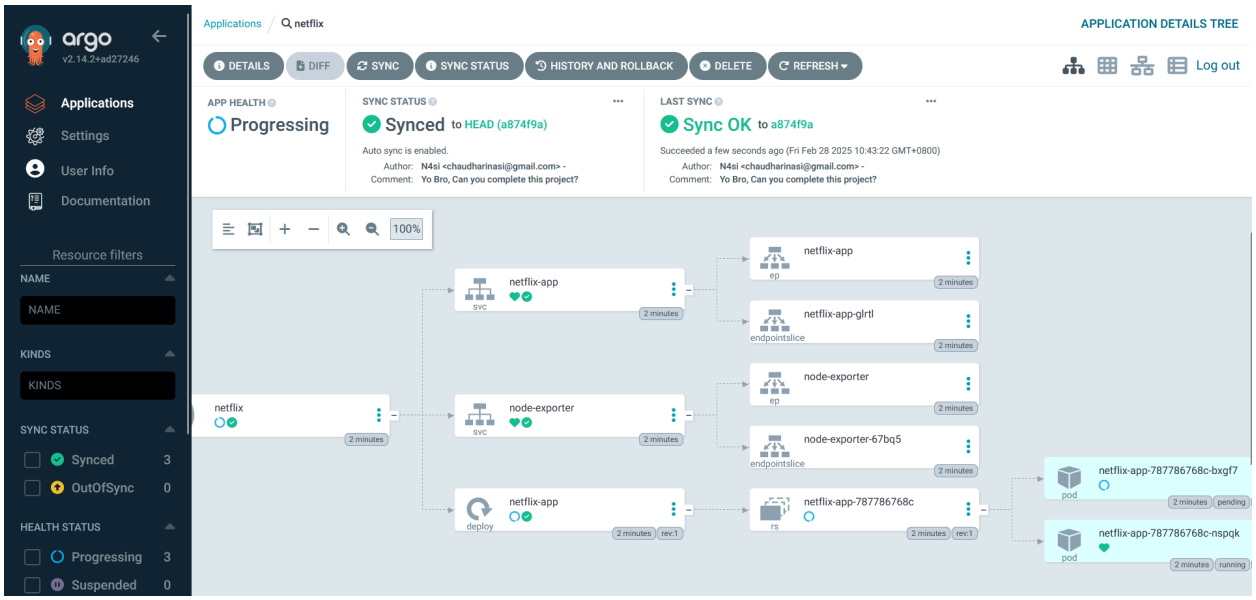
```
kube-system       Active   12m
PS C:\Users\Miguel Huerto> kubectl create namespace argocd
namespace/argocd created
PS C:\Users\Miguel Huerto> kubectl apply -n argocd -f https://raw.githubusercon
s/install.yaml
customresourcedefinition.apiextensions.k8s.io/applications.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/applicationsets.argoproj.io creat
customresourcedefinition.apiextensions.k8s.io/appprojects.argoproj.io created
serviceaccount/argocd-application-controller created
serviceaccount/argocd-applicationset-controller created
serviceaccount/argocd-dex-server created
serviceaccount/argocd-notifications-controller created
serviceaccount/argocd-redis created
serviceaccount/argocd-repo-server created
serviceaccount/argocd-server created
role.rbac.authorization.k8s.io/argocd-application-controller created
role.rbac.authorization.k8s.io/argocd-applicationset-controller created
role.rbac.authorization.k8s.io/argocd-dex-server created
role.rbac.authorization.k8s.io/argocd-notifications-controller created
role.rbac.authorization.k8s.io/argocd-redis created
role.rbac.authorization.k8s.io/argocd-server created
clusterrole.rbac.authorization.k8s.io/argocd-application-controller created
clusterrole.rbac.authorization.k8s.io/argocd-applicationset-controller created
clusterrole.rbac.authorization.k8s.io/argocd-server created
rolebinding.rbac.authorization.k8s.io/argocd-application-controller created
rolebinding.rbac.authorization.k8s.io/argocd-applicationset-controller created
rolebinding.rbac.authorization.k8s.io/argocd-dex-server created
rolebinding.rbac.authorization.k8s.io/argocd-notifications-controller created
rolebinding.rbac.authorization.k8s.io/argocd-redis created
```


Let's get stuff deployed!


argo
Username
Password
SIGN IN
argo


argo
v2.14.2+ad27246
Applications
Settings
User Info
Documentation
Applications
APPLICATIONS TILES
+ NEW APP
SYNC APPS
REFRESH APPS
Search applications...
Log out
No applications available to you just yet
Create new application to start managing resources in your cluster
CREATE APPLICATION

Create the ArgoCD Application



Accessing the working application on port 3007